CAB330 Case Study 1: Students

Class: CAB330

Students:

- Christopher Ayling | 9713581 | christopher.ayling@connect.qut.edu.au
- Benjamin Saljooghi | 9448233 | benjamin.saljooghi@connect.qut.edu.au
- Jordi Smit | 10294139 | jordi.smit@connect.qut.edu.au

Due Date: 9th September 2018

Project Demo: Week 8 Wednesday Lab

Weighting: 25%

Setup

In [1]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2
```

In [2]:
```
# Manipulating Data
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, precision_score, recall_score, f
1_score, accuracy_score, auc, roc_curve, roc_auc_score
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
from collections import defaultdict
import math

# Visualisations
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import SVG, Image
import graphviz
import pydot
from io import StringIO
from sklearn.tree import export_graphviz

# Algorithms
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_grap
hviz
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.neural_network import MLPClassifier, MLPRegressor

from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import RFE, RFECV
from sklearn.model_selection import StratifiedKFold

from sklearn.preprocessing import StandardScaler
```

In [3]:
```
randomSeed = 330
np.random.seed(randomSeed)
```

Data Loading

In [4]:
```
students = pd.read_csv("./STUDENT.csv")
students.head()

rows, columns = students.shape; rows, columns
```

Out[4]: (1044, 35)

Task 1. Data Selection and Distribution. (4 marks)

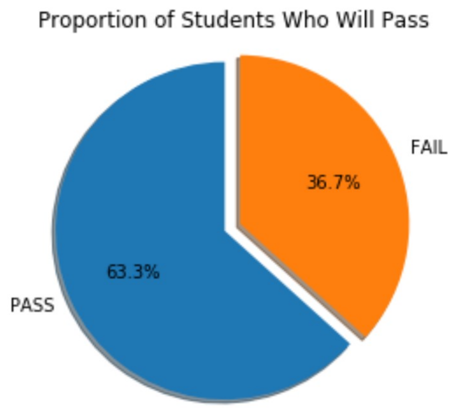
Variable Descriptions

The following information would assist you in assigning the variables roles.

- There are three target variables namely, G1, G2 and G3, with different types. Choose the target that suits best according to the given task.
- Identify if the variable is an input variable or a supplementary variable.
- Data transformation is required for a few input variables to get improved accuracy

1.1 Proportion of Students Who Will Pass

In [5]:
```
G3_counts = students['G3'].value_counts()
plt.pie(G3_counts, labels=G3_counts.index, startangle=90, shadow=True, explode=(0,
0.1), autopct='%1.1f%%')
plt.title("Proportion of Students Who Will Pass")
plt.axis('equal')
plt.show()
```



1.2 Data Cleaning

In [6]:
```
cleaned = students.copy()

# impute age NAs with mean
cleaned['age'].fillna(cleaned['age'].mean(), inplace=True)

# impute reason NAs with unknown value because else it will be read as float instea
d of a string, which cause trouble with the hot one transformation.
cleaned['reason'].fillna("-", inplace=True)

# impute school NAs with unknown value because else it will be read as float instea
d of a string, which cause trouble with the hot one transformation..
cleaned['school'].fillna("-", inplace=True)

# drop unused columns
cleaned.drop(columns=["id", "InitialName"], inplace=True)

#TODO maby better to remove missing target variable?
# impute G1 NAs with mean
cleaned['G1'].fillna(cleaned['G1'].mean(), inplace=True)

# impute G2 NAs with mean
cleaned['G2'].fillna(cleaned['G2'].mean(), inplace=True)
```

Missing value

The following features had missing values:

- Age We replaced the missing values with the mean of all the known values for age;
- Reason We replaced the missing values with a '-' to indicated it is value is unknown;
- School We replaced the missing values with a '-' to indicated it is value is unknown;
- G1 We replaced the missing values with the mean of all the known values for G1;
- G2 We replaced the missing values with the mean of all the known values for G2;

We used the mean values for missing numerical values because this introduces as little diviation as possible. For the missing catagorical values we created a new catagorical value which indicates a missing value. We have choosen this approach over the majority vote because we thought it would be better to let the algorithms sort these features out instead of introducing false data that could result into deviations.

Removed value

We also removed the following features:

- id
- InitialName

These features where removed because they do not provide us with relevant information. They will most likely only introduce noise.

Outliers value

We didn't spot any collumns with outliers so we didn't preform any outlier transformation.

1.3 Level of Measurement

In [7]:
```
descriptions = {
    "id": ["student's id", False, False, 'nominal', False, "Integer", "None"],
    "InitialName": ["student's initial", False, False, 'nominal', False,  "String",
"None"],
    "school": ["student's school name", True, True, 'nominal', False, "String", "In
put"],
    "sex": ["student's sex", True, True, 'nominal', False, "Char", "Input"],
    "age": ["student's age", True, True, 'numerical', False, "Integer", "Input"],
    "address": ["student's home address type", True, True, 'nominal', False, "Char"
, "Input"],
    "famsize": ["family size (≤ 3 or > 3)", True, True, 'ordinal', False, "String",
"Input"],
    "Pstatus": ["parent's cohabitation status (living together or apart)", True, Tr
ue, 'nominal', False, "Char", "Input"],
    "Medu": ["mother’s education(0 - none, 1 - primary education (4th grade), 2 - 5
th to 9th grade, 3 - secondary education or 4 - higher education)", True, True, 'or
dinal',False, "Integer", "Input"],
    "Fedu": ["father’s education(0 - none, 1 - primary education (4th grade), 2 - 5
th to 9th grade, 3 - secondary education or 4 - higher education)", True, True, 'or
dinal', False, "Integer", "Input"],
    "Mjob": ["mother's job", True, True, 'nominal',False, "String", "Input"],
    "Fjob": ["father's job", True, True, 'nominal',False, "String", "Input"],
    "reason": ["reason to choose this school", True, True, 'nominal',False, "String
", "Input"],
    "guardian": ["student's guardian", True, True, 'nominal',False, "String", "Inpu
t"],
    "traveltime": ["home to school travel time (1 - < 15 min., 2 - 15 to 30 min., 3
- 30 min. to 1 hour or 4 - > 1 hour)", True, True, 'ordinal',False, "Integer", "Inp
ut"],
    "studytime": ["weekly study time (1 - < 2 hours, 2 - 2 to 5 hours, 3 - 5 to 10
hours or 4 - > 10 hours)", True, True, 'ordinal',False, "Integer", "Input"],
    "failures": ["number of past class failures(n if 1 ≤ n < 3, else 4)", True, Tru
e, 'ordinal', False, "Integer", "Input"],
    "schoolsup": ["extra educational school support (yes or no)", True, True, 'nomi
nal', False, "String", "Input"],
    "famsup": ["family educational support (yes or no)", True, True, 'nominal', Fal
se, "String", "Input"],
    "paid": ["extra paid classes (yes or no)", True, True, 'nominal', False, "Strin
g", "Input"],
    "activities": ["extra-curricular activities (yes or no)", True, True, 'nominal'
, False, "String", "Input"],
    "nursery": ["attended nursery school (yes or no)", True, True, 'nominal', False
, "String", "Input"],
    "higher": ["wants to take higher education (yes or no)", True, True, 'nominal',
False, "String", "Input"],
    "internet": ["Internet access at home (yes or no)", True, True, 'nominal', Fals
e, "String", "Input"],
    "romantic": ["with a romantic relationship (yes or no)", True, True, 'nominal',
False, "String", "Input"],
    "famrel": ["quality of family relationships (1 - very bad to 5 - excellent)", T
rue, True, 'ordinal', False, "Integer", "Input"],
    "freetime": ["free time after school (1 - very low to 5 - very high)", True, Tr
ue, 'ordinal', False, "Integer", "Input"],
    "goout": ["going out with friends (1 - very low to 5 - very high)", True, True,
'ordinal', False, "Integer", "Input"],
    "Dalc": ["workday alcohol consumption (1 - very low to 5 - very high)", True, T
rue, 'ordinal', False, "Integer", "Input"],
    "Walc": ["weekend alcohol consumption (1 - very low to 5 - very high)", True, T
rue, 'ordinal', False, "Integer", "Input"],
    "health": ["current health status (1 - very bad to 5 - very good)", True, True,
'ordinal',False, "Integer", "Input"],
    "absences": ["number of school absences (0 to 75)", True, True, 'numerical', Fa
lse, "Integer", "Input"],
    "G1": ["first period grade (0 to 20)", True, True, 'numerical', False, "Float",
```

	Description	Target	Variable Type	For Classification	For Regression	Dtype	Variable usage
id	student's id	False	nominal	False	False	Integer	None
InitialName	student's initial	False	nominal	False	False	String	None
school	student's school name	False	nominal	True	True	String	Input
sex	student's sex	False	nominal	True	True	Char	Input
age	student's age	False	numerical	True	True	Integer	Input
address	student's home address type	False	nominal	True	True	Char	Input
famsize	family size (≤ 3 or > 3)	False	ordinal	True	True	String	Input
Pstatus	parent's cohabitation status (living together or apart)	False	nominal	True	True	Char	Input
Medu	mother's education(0 – none, 1 – primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education)	False	ordinal	True	True	Integer	Input
Fedu	father's education(0 – none, 1 – primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education)	False	ordinal	True	True	Integer	Input
Mjob	mother's job	False	nominal	True	True	String	Input
Fjob	father's job	False	nominal	True	True	String	Input
reason	reason to choose this school	False	nominal	True	True	String	Input
guardian	student's guardian	False	nominal	True	True	String	Input
traveltime	home to school travel time (1 – < 15 min., 2 – 15 to 30 min., 3 – 30 min. to 1 hour or 4 – > 1 hour)	False	ordinal	True	True	Integer	Input
studytime	weekly study time (1 – < 2 hours, 2 – 2 to 5 hours, 3 – 5 to 10 hours or 4 – > 10 hours)	False	ordinal	True	True	Integer	Input
failures	number of past class failures(n if 1 ≤ n < 3, else 4)	False	ordinal	True	True	Integer	Input
schoolsup	extra educational school support (yes or no)	False	nominal	True	True	String	Input
famsup	family educational	False	nominal	True	True	String	Input

As can be seen in the table above we decided to use all the features except the *id* and *InitialName*. As mentioned in 1.2 these values do not provided any information about the final target variable *G3*. These features will most likely only introduce noise.

All other variables were used as input variables, sinces none of these variables could be used for visualization. We might have been able to use the address for visualization. However we are unable to do this because all the addresses have been made anonymous.

1.4 Distribution Scheme

Distribution scheme

We read the data as a CSV schema and transformed it into a Panda dataframe. Inside this frame the data is still stored in record format as a Long-Narrow table. For each model we apply separate transformation based on the preferences of the model. For example for the decision tree, we kept the data in a Long-Narrow format. We only encoded the nominal and ordinal values into numerical values. While we transformed the nominal and ordinal data for the data linear regression and neural network models into a Short-Wide format using Hot-One-Encoding.

Data partitioning allocation

We have split the data randomly into two partisions: a training set (80%) and a test set (20%). We have chosen a fixed training/test set size over K-fold validation because the grid search method of Sklearn already applies K-fold validation. This way the final accuracy mesurement will be as representative as possible while the training set is still used as optimal as possible.

We have chosen a 80/20 partitions because the algorithm requires more training data to train effectively. While we only need a small amount of data to test effectively.

In [8]:
```
target_variables = ['G3']
features = cleaned.loc[:, cleaned.columns.difference(target_variables)]
targets = cleaned.loc[:, target_variables]

test_size = 0.2
training_size = 1.0 - test_size
X_training, X_test, Y_training, Y_test = train_test_split(features, targets, test_s
ize=test_size, train_size=training_size, random_state=randomSeed, shuffle=False)
```

In [9]:
```
accuracy_overview = {
    "decision_tree": {},
    "regression": {},
    "neural_network": {},
}

def format_accuracy_overview(Y, predicted_Y, most_important_features):
    return {
        "precision": precision_score(Y, predicted_Y),
        "recall": recall_score(Y, predicted_Y),
        "f1": f1_score(Y, predicted_Y),
        "accuracy": accuracy_score(Y, predicted_Y),
        "ROC": roc_curve(Y, predicted_Y),
        "AUC": roc_auc_score(Y, predicted_Y),
        "most_important_features": most_important_features
    }
```

Task 2. Predictive Modeling Using Decision Trees

(4 marks)

2.1 Build a decision tree using default setting.

For the decision tree, we kept the data in a Long-Narrow format. We only encoded the nominal and ordinal values into numerical values. We did this because a decision tree can only compare numerical values in its rules. While the nominal and ordinal values are currently stored as strings.

In [10]:
```
#Create a lable encoder for each of the nominal and ordinal features.
DT_label_encoders = {}
for name in descriptions:
    if ("nominal" in descriptions[name] or "ordinal" in descriptions[name]) and nam
e in cleaned.columns.values.tolist():
        lb = preprocessing.LabelEncoder()
        #Check cleanded data for every possible class. If only done on training dat
a it might miss some.
        lb.fit(cleaned[name].tolist())
        DT_label_encoders[name] = lb

def transform_features(Data, encoders):
    """Transforms data based on the provided encoder"""
    Data_copy = Data.copy()
    for col_name in Data_copy.columns.values.tolist():
        if col_name in encoders:
            #Get encoder
            encoder = encoders[col_name]
            #Transform the data in this col
            col_values = Data_copy[col_name].tolist()
            Data_copy[col_name] = encoder.transform(col_values)

    return Data_copy

def transform_features_to_DT(Data):
    """Transforms the nominal and ordinal features into a format that can be compar
ed by the decision tree"""
    return transform_features(Data, DT_label_encoders)
```

In [11]:
```
#Transform the training data into a format with which the decision tree can work.
X_training_decision_tree_format = transform_features_to_DT(X_training)
Y_training_decision_tree_format = transform_features_to_DT(Y_training)


#Create a decision tree and train it on the formated training data.
dt = DecisionTreeClassifier(random_state=randomSeed)
dt.fit(X_training_decision_tree_format, Y_training_decision_tree_format)
```

Out[11]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=330,
            splitter='best')
```

a. What is the classification accuracy on training and test datasets?

In [12]:
```
#Transform test data
X_test_decision_tree_format = transform_features_to_DT(X_test)
Y_test_decision_tree_format = transform_features_to_DT(Y_test)

#Create a dataframe to display the values.
training_score = dt.score(X_training_decision_tree_format, Y_training_decision_tree
_format)
test_score = dt.score(X_test_decision_tree_format, Y_test_decision_tree_format)
dt_preformance = pd.DataFrame([training_score, test_score], columns=target_variable
s, index=['Train accuracy', 'Test accuracy'])
dt_preformance
```

Out[12]:

	G3
Train accuracy	1.000000
Test accuracy	0.827751

b. List the decision rules

In [13]:
```
dt_features = X_training_decision_tree_format.columns.values.tolist()
dotfile = StringIO()
export_graphviz(dt, out_file=dotfile, feature_names=dt_features)
graph = pydot.graph_from_dot_data(dotfile.getvalue())
graph[0].write_png("default_tree.png") # saved in the following file - will return
True if successful
```

The unique decision rules:

- G2 <= 10.45
- G2 <-=9.55
- G1 <= 9.75
- goout <=0.5
- failures <= 0.5
- sex <= 0.5
- G2 <= 9.35
- Medu <= 9.35
- goout <= 3.5
- Dalc < 3.5
- G1 <= 9.85
- age <= 17.5
- Fedu <= 1.5
- Fjob <= 0.5
- studytime <= 1.0
- age <= 16.868
- guardian <= 1.5
- absences <= 6.0
- absences <= 9.0
- G1 <= 11.225
- famrel <= 2.5
- walc <= 0.5
- guardian <= 0.5
- walc <= 2.5
- dalc <= 0.5
- Mjob <= 1.5
- Medu <= 1.5
- adress <= 0.5
- Fedu <= 2.5
- health <= 3.0
- school <= 1.5
- reason <= 2.5
- freetime <= 2.5
- G2 <= 11.35
- G1 <= 11.55
- traveltime <= 0.5
- Fjob <= 1.5
- reason <= 0.5
- Walc <= 3.5
- Walc <= 0.5
- health <= 0.5
- G1 <= 10.85
- reason <= 1.5
- paid <= 0.5
- activities <= 0.5
- nursery <= 0.5
- G1 <= 10.55
- famrel <= 3.5
- G2 <= 11.55
- G1 <= 12.85
- schoolsup <= 0.5
- absences <= 1.0
- goout <= 05

c. What are the 5 important variables in building the tree?

In [14]:
```
importances = dt.feature_importances_
importances_dt = pd.DataFrame(importances, index=dt_features, columns=["G3"])
importances_dt.nlargest(5, "G3")
```

Out[14]:

	G3
G2	0.732581
G1	0.053456
Walc	0.031882
Fedu	0.018104
health	0.014787

d. Report if you see any evidence of model overfitting.

There is model overfitting on the training set since the decision tree has a 100% accuracy at the training set while it only has a 82.8% accuracy on the test set. This can also be seen in the tree since it has become extremely complicated with very many paths. This a classic example of overfitting.

2.2 Build another decision tree tuned with GridSearchCV.

In [15]:
```
#The parameters to be searched
parameters = {
     "max_depth": [None, 1, 2, 3, 4, 5],
     "min_samples_split": [0.001, 0.005, 0.01, 0.05, 0.1],
     "min_samples_leaf": [1, 2, 4, 8, 16, 32],
     "criterion": ['gini', 'entropy'],
     "splitter" : ["best", "random"],
     "max_features": [None, "auto", "sqrt", "log2"],
     "max_leaf_nodes": [None, 2, 3, 4, 5, 6],
}
#creates and starts a grids search
gs_dt = GridSearchCV(DecisionTreeClassifier(random_state=randomSeed), parameters, n
_jobs=8)
gs_dt.fit(X_training_decision_tree_format, Y_training_decision_tree_format)
```

Out[15]:
```
GridSearchCV(cv=None, error_score='raise',
       estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max
_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=330,
            splitter='best'),
       fit_params=None, iid=True, n_jobs=8,
       param_grid={'max_depth': [None, 1, 2, 3, 4, 5], 'min_samples_split': [0.0
01, 0.005, 0.01, 0.05, 0.1], 'min_samples_leaf': [1, 2, 4, 8, 16, 32], 'criterio
n': ['gini', 'entropy'], 'splitter': ['best', 'random'], 'max_features': [None,
'auto', 'sqrt', 'log2'], 'max_leaf_nodes': [None, 2, 3, 4, 5, 6]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

a. What is the classification accuracy on training and test datasets?

In [16]:
```
#Calc scores
training_score = gs_dt.score(X_training_decision_tree_format, Y_training_decision_t
ree_format)
test_score = gs_dt.score(X_test_decision_tree_format, Y_test_decision_tree_format)

#Display training and test score
dt_grid_preformance = pd.DataFrame([training_score, test_score], columns=["G3 grid"
], index=['Train accuracy', 'Test accuracy'])
dt_grid_preformance
```

Out[16]:

	G3 grid
Train accuracy	0.932934
Test accuracy	0.866029

In [17]:
```
dt_y_predicted = gs_dt.best_estimator_.predict(X_test_decision_tree_format)
dt_score_overview = classification_report(Y_test_decision_tree_format, dt_y_predict
ed)
```

b. What are the parameters used? Explain your decision.

The following parameters are being considered:

- **max_depth**: Limits the maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- **min_samples_split**: The minimum number of samples required to split an internal node.
- **min_samples_leaf**: The minimum number of samples required to be at a leaf node.
- **criterion**: The function to measure the quality of a split. Examples are Gini impurity and entropy/information gain.
- **splitter**: The strategy used to choose the split at each node. Examples are best which choices the best criterion criteria or random which preforms a random split.
- **max_features**: Limits the number of features to consider when looking for the best split. For Example sqrt limits it to sqrt (total_number_of_features).
- **max_leaf_nodes**: Limits the number of leaf nodes. Nodes with relative reduction in impurity are added first to ensure the best possible tree with the constaint.

We decided to use the kitchen sink approach. We looked up the default values of the parameters and provided the grid search with a random range of values arround these default values. Then we the GridSearch run on multiple threads/jobs and we see which parameters return the optimal values.

c. What are the optimal parameters for this decision tree?

In [18]:
```
print(f"The following parameters result in the best decision tree:")
for name in parameters:
    print(f"   -{name}: {gs_dt.best_params_[name]}")
```

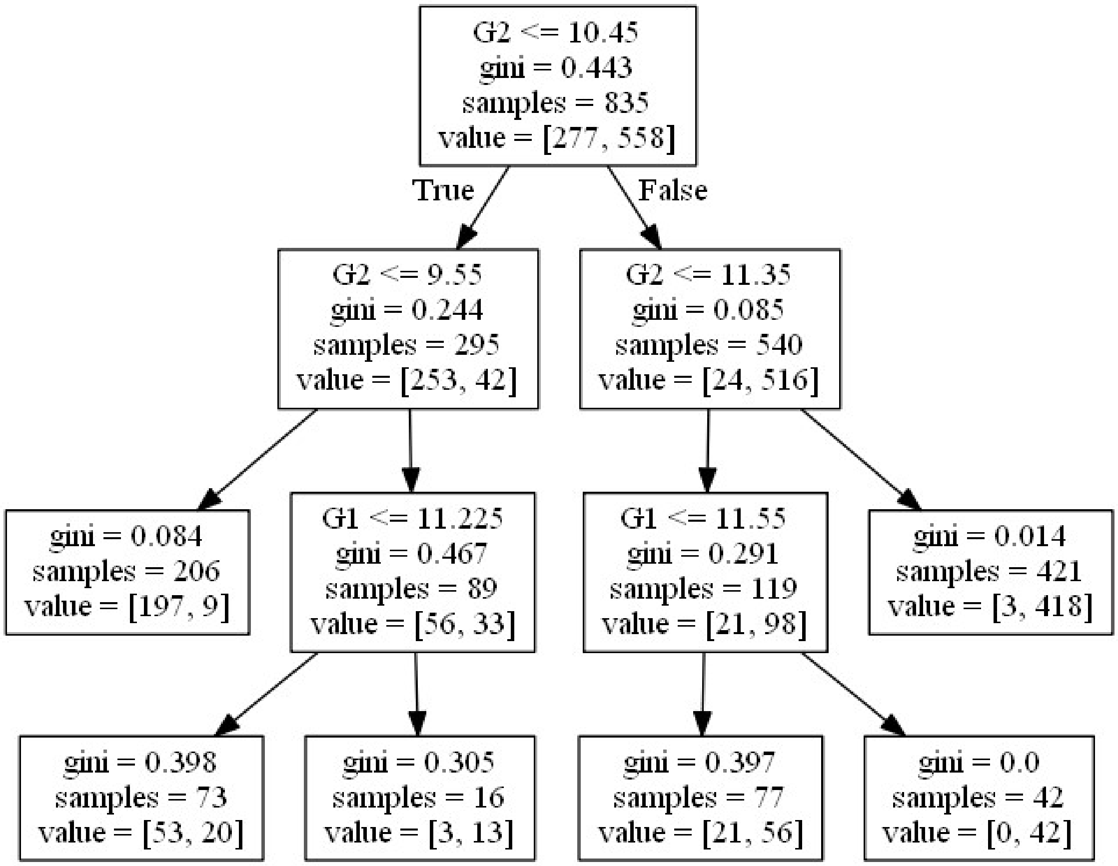
```
The following parameters result in the best decision tree:
   -max_depth: None
   -min_samples_split: 0.001
   -min_samples_leaf: 8
   -criterion: gini
   -splitter: best
   -max_features: None
   -max_leaf_nodes: 6
```

d. Which variable is used for the first split? What are the competing splits for this first split?

In [19]:
```
dt_features = X_training_decision_tree_format.columns.values.tolist()
dotfile = StringIO()
export_graphviz(gs_dt.best_estimator_, out_file=dotfile, feature_names=dt_features)
graph = pydot.graph_from_dot_data(dotfile.getvalue())
graph[0].write_png("grid_tree.png") # saved in the following file - will return Tru
e if successful
```

The first split is done based on the following rule: G2 <= 10.45. The split rules at the second level are G2 <= 9.55 & G2 <= 11.35. Which is the same as the tree with the default parameter. The only different is that this tree has far less leaf nodes.

The result:



e. What are the 5 important variables in building the tree?

The decision tree only uses the features G2 and G1. All other values have the same weight so the remaining features in the top 5 are selected based on the original sorting.

In [20]:
```
#Get the importances of each feature
importances = gs_dt.best_estimator_.feature_importances_

#display the 5 most important features
importances_dt_gs = pd.DataFrame(importances, index=dt_features, columns=["G3"])
importances_dt_gs
importances_dt_gs.nlargest(5, "G3")
```

Out[20]:

	G3
G2	0.958757
G1	0.041243
Dalc	0.000000
Fedu	0.000000
Fjob	0.000000

In [21]:
```
dt_5_most_imporant_features = importances_dt_gs.nlargest(5, "G3").index.values.toli
st()
accuracy_overview["decision_tree"] = format_accuracy_overview(Y_test_decision_tree_
format, dt_y_predicted, dt_5_most_imporant_features)
print(accuracy_overview["decision_tree"])
```

```
{'precision': 0.8640776699029126, 'recall': 0.8640776699029126, 'f1': 0.86407766
99029126, 'accuracy': 0.8660287081339713, 'ROC': (array([0.        , 0.13207547,
1.        ]), array([0.        , 0.86407767, 1.        ]), array([2, 1, 0], dtyp
e=int64)), 'AUC': 0.8660010991023998, 'most_important_features': ['G2', 'G1', 'D
alc', 'Fedu', 'Fjob']}
```

f. Report if you see an evidence of model overfitting.

There is no evidence of model overfitting, the accuracy on the train and test sets moved much closer to each other. The tree has also become much simpler compared to the original one as can be seen in the visualization image. This is also a good indicater that there is no overfitting.

2.3 What is the significant difference do you see between these two decision tree models? How do they compare performance-wise? Explain why those changes may have happened.

What is the significant difference do you see between these two decision tree models? When you compare the visualization of the trees, the first thing you notice is that the grid tree is much smaller and simpler than the original one. The grid tree also does not have duplicated rules. The original thee had some duplicated rules in different branches. The original tree did this to fit the training data perfectly which resulted into over-fitting.

How do they compare performance-wise The original tree fitted the training data perfectly but only achieved an 82% accuracy on the test data due to over-fitting. While the grid tree only achieve a 93% accuracy on the training data, it achieved a 86% accuracy on the test data. Thus the grid tree is much better at generalizing, which result into a higher accuracy on data it has never seen before. Making the grid tree the better one.

Explain why those changes may have happened These changes happen due to the grid search. Sklearning's grid search uses K-fold validation on each possible configuration. The K-fold validation prevents the training algorithm from over-fitting in combination with the configuration parameters. Eventually the search returns the configuration that has the highest K-fold score which result into a tree data is not over-fitted (assuming that the training data is representative). In our case this resulted into a much smaller tree. A much smaller tree means that there are fewer rules and that the decision rules are much more general and have to focus on the most important features. This means that the tree cannot perfectly fit the training data and over-fit, which makes it preform better on data it has never seen before.

In [22]:
```
comparision = pd.DataFrame([], columns=[])
print(comparision.append(dt_preformance["G3"]).append(dt_grid_preformance["G3 grid"
]).transpose())
```

```
                      G3   G3 grid
Test accuracy   0.827751  0.866029
Train accuracy  1.000000  0.932934
```

2.4 From the better model, can you identify which students to target for further consultation? Can you provide some descriptive summary of those students?

The following rules identify students that will most likely fail.

- `if G2 <= 9.55` There are 206 students in this group and 95.6% of them eventually failed. So this is the most important group to watch.
- `if 9.55 < G2 <= 10.45 & G1 <= 11.225` There are 73 students in this group and 72.6% of them eventually failed. So this is the second most important group to watch.

The following rules are far less interesting but still indicate students with a small changes of failing.

- `if 10.45 < G2 <= 11.35 & G1 <= 11.55` There are 77 students in this group and 27.3% of them eventually failed.
- `if 9.55 < G2 <= 10.45 & G1 > 11.225` There are 16 students in this group and 18.8% of them eventually failed.

As can be seen in the rules above only the grades provide a good indication about wheter the students will pass or not.

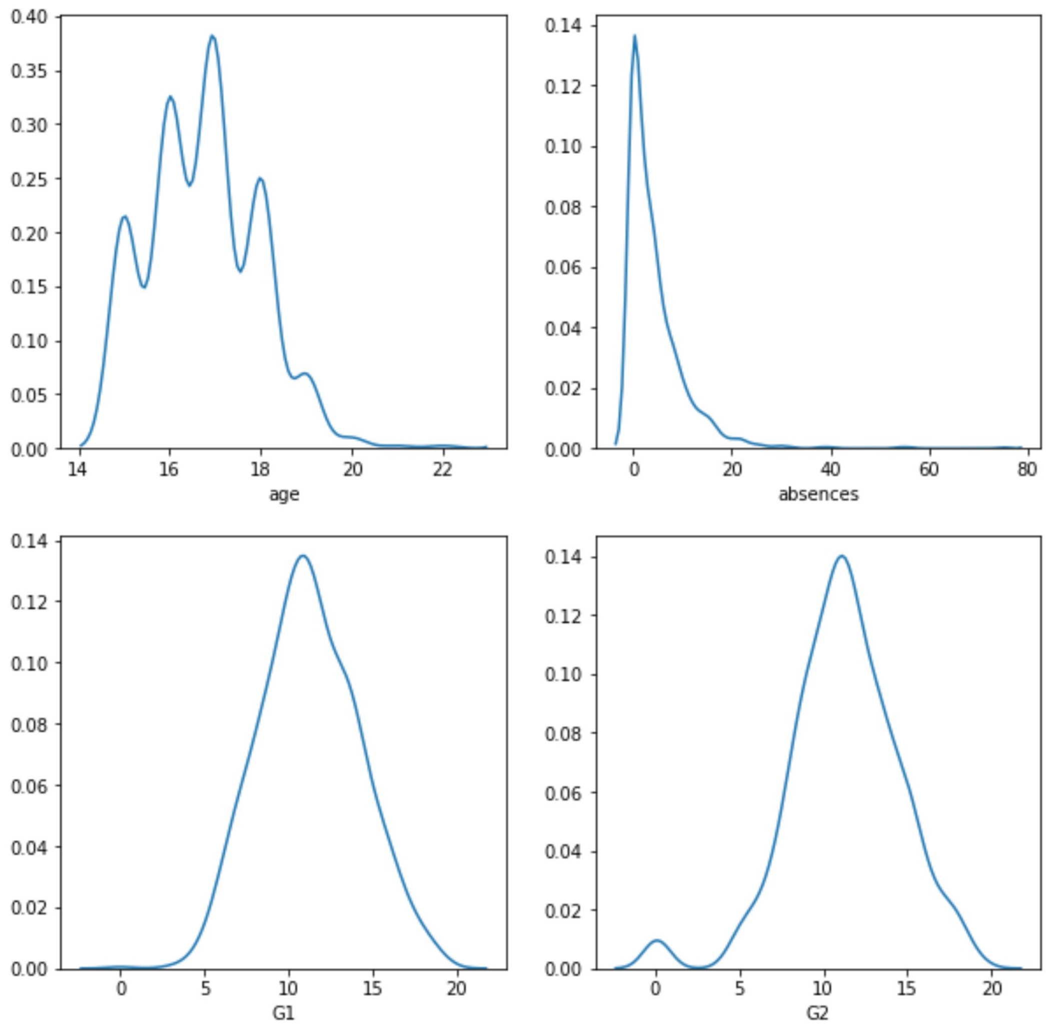
Task 3. Predictive Modeling Using Regression

(5.5 marks)

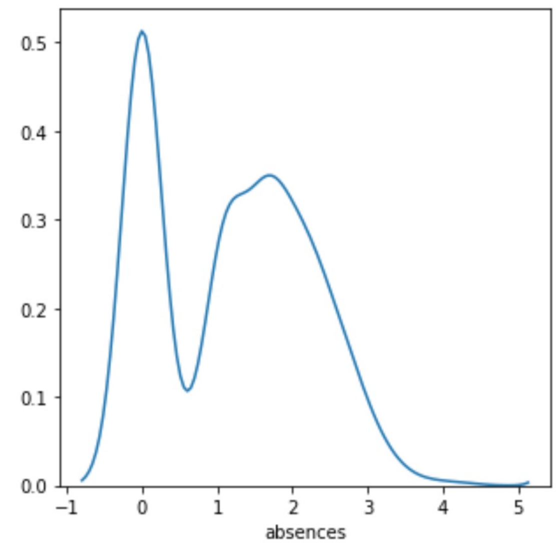
3.1 Apply transformation/scaling methods to variables.

In [23]:
```
# one-hot encode a column
def one_hot(data, column):
    print("One-hot encoding", column)
    dummies = pd.get_dummies(data[column], prefix=column)
    data = data.drop(column, axis=1)
    data = data.join(dummies)
    return data


# binary encode (with 0s and 1s) a binary column (e.g. Yes or No)
def binary(data, column, map):
    print("Binary encoding", column)
    new_col = data[column].map(map)
    data = data.drop(column, axis=1)
    data = data.join(new_col)
    return data


# get a copy of the data set
reg_data = cleaned.copy()


# apply one-hot encodings
print("\nApply one-hot encodings...")
for col in ['school', 'address', 'Mjob', 'Fjob', 'reason', 'guardian']:
    reg_data = one_hot(reg_data, col)


# prepare binary encodings
print("\nApply binary encodings...")
yes_no_map = {'no':0, 'yes':1}
binary_encodings = {
    'schoolsup': yes_no_map,
    'famsup': yes_no_map,
    'paid': yes_no_map,
    'activities': yes_no_map,
    'nursery': yes_no_map,
    'higher': yes_no_map,
    'internet': yes_no_map,
    'romantic': yes_no_map,
    'sex': {'F':0, 'M':1},
    'famsize': {'LE3':0, 'GT3':1},
    'Pstatus': {'A':0, 'T':1},
    'G3': {'FAIL':0, 'PASS':1},
}

# apply binary encodings
for col in binary_encodings:
    reg_data = binary(reg_data, col, binary_encodings[col])


# graph all numerical data
print("\nNumerical columns:")
f, axes = plt.subplots(2, 2, figsize=(10, 10), sharex=False)
sns.distplot(reg_data["age"], hist=False, ax=axes[0, 0])
sns.distplot(reg_data["absences"], hist=False, ax=axes[0, 1])
sns.distplot(reg_data["G1"], hist=False, ax=axes[1, 0])
sns.distplot(reg_data["G2"], hist=False, ax=axes[1, 1])
plt.show()


# apply a log transform to the absences column
print("\nAbsences is skewed so apply a log transform:")
reg_data["absences"] = reg_data["absences"].apply(lambda x: x+1)
```

```
Apply one-hot encodings...
One-hot encoding school
One-hot encoding address
One-hot encoding Mjob
One-hot encoding Fjob
One-hot encoding reason
One-hot encoding guardian

Apply binary encodings...
Binary encoding schoolsup
Binary encoding famsup
Binary encoding paid
Binary encoding activities
Binary encoding nursery
Binary encoding higher
Binary encoding internet
Binary encoding romantic
Binary encoding sex
Binary encoding famsize
Binary encoding Pstatus
Binary encoding G3

Numerical columns:
```




```
Absences is skewed so apply a log transform:
```




```
Visualise transformed data:
```

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	Walc	healt
0	18.000000	4	4	2	2	0	4	3	4	1	1	3
1	17.000000	1	1	1	2	0	5	3	3	1	1	3
...	...	...	...	...	...	...	...	...	...	...	...	...
1042	16.736354	3	1	2	1	0	2	4	5	3	4	2
1043	18.000000	3	2	3	1	0	4	4	1	3	4	5

1044 rows × 50 columns

In [24]:
```
# split transformed data into features and targets
reg_features = reg_data.loc[:, reg_data.columns.difference(target_variables)]
reg_targets = reg_data.loc[:, target_variables]

# train test split the transformed data
reg_X_train_frame, reg_X_test_frame, reg_Y_train_frame, reg_Y_test_frame = train_te
st_split(reg_features, reg_targets, test_size=test_size, train_size=training_size,
random_state=randomSeed, shuffle=False)

# convert to numpy matrices
reg_X_train = reg_X_train_frame.values
reg_X_test = reg_X_test_frame.values
reg_Y_train = reg_Y_train_frame.values
reg_Y_test = reg_Y_test_frame.values
reg_Y_train =  np.ravel(reg_Y_train) # make one-dimensional
reg_Y_test = np.ravel(reg_Y_test) # make one-dimensional


# scale data to have a mean of 0 and std dev of 1

# print the min, max, mean, and std dev of each feature
def print_data():
    # swap the axes of the numpy matrix to be column-wise
    column_wise = np.swapaxes(reg_X_train, 0, 1)
    # print each column's statistics
    for index, col in enumerate(reg_X_train_frame):
        values = column_wise[index]
        var_name = col.ljust(15)
        print("{} \t min {:.2f} max {:.2f} mean {:.2f} std dev {:.2f}".
        format(var_name, min(values), max(values), np.mean(values), np.std(values))
)

print("Before scaling:\n")
print_data()

# train a scaler on the training data then normalize the training data
scaler = StandardScaler()
reg_X_train = scaler.fit_transform(reg_X_train, reg_Y_train)
print("\nAfter scaling:\n")
print_data()

# apply the learned scaling transformation to the test data
reg_X_test = scaler.transform(reg_X_test)
```

```
Before scaling:

Dalc                 min 1.00 max 5.00 mean 1.47 std dev 0.89
Fedu                 min 0.00 max 4.00 mean 2.49 std dev 1.09
Fjob_at_home         min 0.00 max 1.00 mean 0.04 std dev 0.21
Fjob_health          min 0.00 max 1.00 mean 0.04 std dev 0.20
Fjob_other           min 0.00 max 1.00 mean 0.57 std dev 0.49
Fjob_services        min 0.00 max 1.00 mean 0.27 std dev 0.44
Fjob_teacher         min 0.00 max 1.00 mean 0.07 std dev 0.26
G1                   min 0.00 max 19.40 mean 11.48 std dev 2.89
G2                   min 0.00 max 19.40 mean 11.47 std dev 3.19
Medu                 min 0.00 max 4.00 mean 2.72 std dev 1.09
Mjob_at_home         min 0.00 max 1.00 mean 0.16 std dev 0.36
Mjob_health          min 0.00 max 1.00 mean 0.09 std dev 0.28
Mjob_other           min 0.00 max 1.00 mean 0.37 std dev 0.48
Mjob_services        min 0.00 max 1.00 mean 0.25 std dev 0.43
Mjob_teacher         min 0.00 max 1.00 mean 0.14 std dev 0.35
Pstatus              min 0.00 max 1.00 mean 0.88 std dev 0.32
Walc                 min 1.00 max 5.00 mean 2.27 std dev 1.29
absences             min 0.00 max 4.33 mean 1.26 std dev 1.03
activities           min 0.00 max 1.00 mean 0.51 std dev 0.50
address_R            min 0.00 max 1.00 mean 0.22 std dev 0.41
address_U            min 0.00 max 1.00 mean 0.78 std dev 0.41
age                  min 15.00 max 22.00 mean 16.67 std dev 1.21
failures             min 0.00 max 3.00 mean 0.25 std dev 0.65
famrel               min 1.00 max 5.00 mean 3.94 std dev 0.91
famsize              min 0.00 max 1.00 mean 0.71 std dev 0.45
famsup               min 0.00 max 1.00 mean 0.62 std dev 0.48
freetime             min 1.00 max 5.00 mean 3.20 std dev 1.00
goout                min 1.00 max 5.00 mean 3.13 std dev 1.13
guardian_father      min 0.00 max 1.00 mean 0.22 std dev 0.42
guardian_mother      min 0.00 max 1.00 mean 0.71 std dev 0.46
guardian_other       min 0.00 max 1.00 mean 0.07 std dev 0.26
health               min 1.00 max 5.00 mean 3.58 std dev 1.41
higher               min 0.00 max 1.00 mean 0.93 std dev 0.25
internet             min 0.00 max 1.00 mean 0.83 std dev 0.37
nursery              min 0.00 max 1.00 mean 0.80 std dev 0.40
paid                 min 0.00 max 1.00 mean 0.25 std dev 0.43
reason_-             min 0.00 max 1.00 mean 0.58 std dev 0.49
reason_course        min 0.00 max 1.00 mean 0.15 std dev 0.36
reason_home          min 0.00 max 1.00 mean 0.11 std dev 0.31
reason_other         min 0.00 max 1.00 mean 0.04 std dev 0.19
reason_reputation    min 0.00 max 1.00 mean 0.12 std dev 0.33
romantic             min 0.00 max 1.00 mean 0.34 std dev 0.47
school_-             min 0.00 max 1.00 mean 0.05 std dev 0.21
school_DCHS          min 0.00 max 1.00 mean 0.88 std dev 0.32
school_THS           min 0.00 max 1.00 mean 0.07 std dev 0.26
schoolsup            min 0.00 max 1.00 mean 0.13 std dev 0.34
sex                  min 0.00 max 1.00 mean 0.45 std dev 0.50
studytime            min 1.00 max 4.00 mean 2.02 std dev 0.84
traveltime           min 1.00 max 4.00 mean 1.44 std dev 0.70

After scaling:

Dalc                 min -0.53 max 3.97 mean 0.00 std dev 1.00
Fedu                 min -2.29 max 1.38 mean 0.00 std dev 1.00
Fjob_at_home         min -0.22 max 4.64 mean 0.00 std dev 1.00
Fjob_health          min -0.21 max 4.71 mean 0.00 std dev 1.00
Fjob_other           min -1.16 max 0.86 mean -0.00 std dev 1.00
Fjob_services        min -0.61 max 1.65 mean -0.00 std dev 1.00
Fjob_teacher         min -0.28 max 3.63 mean -0.00 std dev 1.00
G1                   min -3.97 max 2.73 mean 0.00 std dev 1.00
G2                   min -3.59 max 2.48 mean -0.00 std dev 1.00
Medu                 min -2.49 max 1.17 mean 0.00 std dev 1.00
```

3.2 Build regression models

In [25]:
```
models = {}

# evaluation
def report_overfitting(model_name, X_train, X_test):
    train_acc = models[model_name].score(X_train, reg_Y_train)
    test_acc = models[model_name].score(X_test, reg_Y_test)
    diff = train_acc - test_acc
    print("Train accuracy:", train_acc)
    print("Test accuracy:", test_acc)
    print("Difference (overfitting):", diff)

def report_accuracy(model_name, X_test):
    print(classification_report(reg_Y_test, models[model_name].predict(X_test)))

models["reg"] = LogisticRegression()
models["reg"].fit(reg_X_train, reg_Y_train)

params = {'C': [pow(10, x) for x in range(-6, 4)]}
models["reg_cv"] = GridSearchCV(LogisticRegression(), params, cv=10, n_jobs=-1)
models["reg_cv"].fit(reg_X_train, reg_Y_train)
```

Out[25]:
```
GridSearchCV(cv=10, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_in
tercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'C': [1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 100
0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

a. Report which variables are included in the regression model.

In [26]:
```
feature_names = reg_X_train_frame.columns
print(feature_names)
```

```
Index(['Dalc', 'Fedu', 'Fjob_at_home', 'Fjob_health', 'Fjob_other',
       'Fjob_services', 'Fjob_teacher', 'G1', 'G2', 'Medu', 'Mjob_at_home',
       'Mjob_health', 'Mjob_other', 'Mjob_services', 'Mjob_teacher', 'Pstatus',
       'Walc', 'absences', 'activities', 'address_R', 'address_U', 'age',
       'failures', 'famrel', 'famsize', 'famsup', 'freetime', 'goout',
       'guardian_father', 'guardian_mother', 'guardian_other', 'health',
       'higher', 'internet', 'nursery', 'paid', 'reason_-', 'reason_course',
       'reason_home', 'reason_other', 'reason_reputation', 'romantic',
       'school_-', 'school_DCHS', 'school_THS', 'schoolsup', 'sex',
       'studytime', 'traveltime'],
      dtype='object')
```

All variables are included.

b. Report the top-5 important variables.

In [27]:
```
coef = models["reg"].coef_[0]
indices = np.argsort(np.absolute(coef))
indices = np.flip(indices, axis=0)
top_5 = [(feature_names[i], coef[i]) for i in indices[:5]]
for feature, coefficient in top_5:
    print(feature, coefficient)
```

```
G2 4.483007274435303
G1 1.202094090380682
failures -0.4709443004349045
Fjob_health -0.3256520406787766
nursery -0.2828617525602498
```

G2 and G1 have much higher coefficients (4.5 and 1.2 respectively) compared to the other variables. The next three best variables are the number of past failures, the father's job, and whether or not the student attended nursery school.

c. Report any sign of overfitting.

In [28]:
```
print("Without GridSearchCV")
report_overfitting("reg", reg_X_train, reg_X_test)

print("\nWith GridSearchCV")
report_overfitting("reg_cv", reg_X_train, reg_X_test)
```

```
Without GridSearchCV
Train accuracy: 0.9341317365269461
Test accuracy: 0.84688995215311
Difference (overfitting): 0.08724178437383612

With GridSearchCV
Train accuracy: 0.932934131736527
Test accuracy: 0.8421052631578947
Difference (overfitting): 0.09082886857863226
```

There is a high degree of overfitting. Both the non-GridSearchCV and GridSearchCV models exhibit about a 10 percentage point decrease in accuracy on the test data.

d. What are the parameters used?

In [29]:
```
print("Parameters:")
print(models["reg"].get_params(), "\n")

print("Optimal parameters:")
print(models["reg_cv"].best_params_, "\n")
```

```
Parameters:
{'C': 1.0, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercep
t_scaling': 1, 'max_iter': 100, 'multi_class': 'ovr', 'n_jobs': 1, 'penalty': 'l
2', 'random_state': None, 'solver': 'liblinear', 'tol': 0.0001, 'verbose': 0, 'w
arm_start': False}

Optimal parameters:
{'C': 100}
```

The parameters are the default parameters for logistic regression:

{'C': 1.0, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_scaling': 1, 'max_iter': 100, 'multi_class': 'ovr', 'n_jobs': 1, 'penalty': 'l2', 'random_state': None, 'solver': 'liblinear', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}

The optimal parameters as determined by GridSearchCV are:

{'C': 100}

A logistic regression function is used because the target variable (G3) is binary (PASS/FAIL).

e. What is the classification accuracy on training and test datasets?

In [30]:
```
print("Without GridSearchCV:")
report_accuracy("reg", reg_X_test)

print("\nWith GridSearchCV:")
report_accuracy("reg_cv", reg_X_test)
```

```
Without GridSearchCV:
             precision    recall  f1-score   support

          0       0.80      0.92      0.86       106
          1       0.91      0.77      0.83       103

avg / total       0.85      0.85      0.85       209


With GridSearchCV:
             precision    recall  f1-score   support

          0       0.80      0.92      0.86       106
          1       0.91      0.76      0.83       103

avg / total       0.85      0.84      0.84       209
```

3.3 Build another regression model using the subset of inputs selected by RFE and selection by model methods

In [31]:
```
rfe = RFECV(estimator = LogisticRegression(), cv=10)
rfe.fit(reg_X_train, reg_Y_train)
reg_X_train_sel = rfe.transform(reg_X_train)
reg_X_test_sel = rfe.transform(reg_X_test)

models["reg_rfe"] = LogisticRegression()
models["reg_rfe"].fit(reg_X_train_sel, reg_Y_train)

params = {'C': [pow(10, x) for x in range(-6, 4)]}
models["reg_cv_rfe"] = GridSearchCV(LogisticRegression(), params, cv=10, n_jobs=-1)
models["reg_cv_rfe"].fit(reg_X_train_sel, reg_Y_train)
```

Out[31]:
```
GridSearchCV(cv=10, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_in
tercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'C': [1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 100
0]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

a. Report which variables are included in the regression model.

In [32]:
```
for boolean, feature in zip(rfe.support_, feature_names):
    if boolean:
        print(feature)
```

```
G1
G2
```

Recursive feature reduction (RFE) determined that only G1 and G2 should be included.

b. Report the top-5 important variables.

In [33]:
```
rfe_sorted_features = []
for i in range(len(rfe.ranking_)):
    for index in range(len(rfe.ranking_)):
        rank = rfe.ranking_[index]
        if (rank == i+1):
            rfe_sorted_features.append(feature_names[index])
print(rfe_sorted_features[:5])
```

```
['G1', 'G2', 'failures', 'school_THS', 'Fjob_health']
```

Although only G1 and G2 should be included, RFE determined that, similarly to the standard logistic regression model, the next three best variables are the number of past failures, the father's job, and which school the student attends.

c. Report any sign of overfitting.

In [34]:
```
print("Without GridSearchCV")
report_overfitting("reg_rfe", reg_X_train_sel, reg_X_test_sel)

print("\nWith GridSearchCV")
report_overfitting("reg_cv_rfe", reg_X_train_sel, reg_X_test_sel)
```

```
Without GridSearchCV
Train accuracy: 0.9173652694610779
Test accuracy: 0.8899521531100478
Difference (overfitting): 0.027413116351030054

With GridSearchCV
Train accuracy: 0.9233532934131736
Test accuracy: 0.8995215311004785
Difference (overfitting): 0.023831762312695126
```

There is minimal sign of overfitting compared to the previous two models. By eliminating most variables, the reduction in accuracy for the test data is now only about 2.5 percentage points.

d. What is the classification accuracy on training and test datasets?

In [35]:
```
print("Without GridSearchCV:")
report_accuracy("reg_rfe", reg_X_test_sel)

print("With GridSearchCV:")
report_accuracy("reg_cv_rfe", reg_X_test_sel)
```

```
Without GridSearchCV:
             precision    recall  f1-score   support

          0       0.89      0.90      0.89       106
          1       0.89      0.88      0.89       103

avg / total       0.89      0.89      0.89       209

With GridSearchCV:
             precision    recall  f1-score   support

          0       0.89      0.92      0.90       106
          1       0.91      0.88      0.90       103

avg / total       0.90      0.90      0.90       209
```

3.4 Using the comparison statistics, which of the regression models appears to be better? Is there any difference between two models (i.e one with selected variables and another with all variables)? Explain why those changes may have happened.

Logistic regression with RFE and GridSearchCV produces the best accuracy and f-score with the lowest degree of overfitting. This is likely because the elimination of unimportant variables has minimized the overall variation of dataset, leading to a more generalized model and thus less overfitting. Additionally, the reduction in features also removes many outliers, which regression is sensitive to.

3.5 From the better model, can you identify which students to target? Can you provide some descriptive summary of those students?

Students with a good G1 and G2 score and low number of previous failures are highly likely to pass. Therefore, students with poor grades so far should be targetted. Another somewhat important factor also seems to be the student's school of choice (favouring THS), and the father's job (favouring health). Perhaps students in non-THS schools should receive more attention.

In [36]:
```
# nominate RFE with GridSearchCV as the as best regression model
accuracy_overview["regression"] = format_accuracy_overview(reg_Y_test, models["reg_
cv_rfe"].predict(reg_X_test_sel), rfe_sorted_features[:5])
```

Task 4. Predictive Modeling Using Neural Networks

(5.5 marks)

In [37]:
```
Xtr_nn, ytr_nn = reg_X_train, reg_Y_train
Xte_nn, yte_nn = reg_X_test, reg_Y_test
```

1. Build a Neural Network model using the default setting.

In [38]:
```
mlp = MLPClassifier().fit(Xtr_nn, ytr_nn)
```

a. What is the network architecture of the model?

In [39]:
```
print(f"the default neural network has {len(mlp.hidden_layer_sizes)} hidden layers(
s) of size {mlp.hidden_layer_sizes} which uses {mlp.activation} activations")
```

```
the default neural network has 1 hidden layers(s) of size (100,) which uses relu
activations
```

b. How many iterations are needed to train this network?

In [40]:
```
print(f"the default neural network trained for {len(mlp.loss_curve_)} iterations")
```

```
the default neural network trained for 173 iterations
```

c. Do you see any sign of over-fitting?

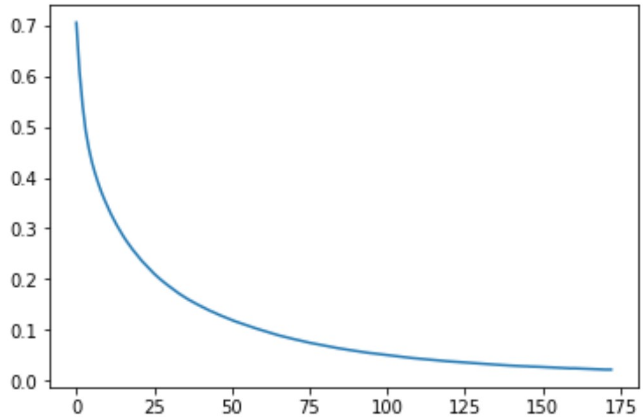
Over fitting is evident due to the almost perfect score on the training set and a score of approximately 87% on the test set.

In [41]:
```
mlp.score(Xtr_nn, ytr_nn), mlp.score(Xte_nn, yte_nn)
```

Out[41]:
```
(0.9988023952095808, 0.84688995215311)
```

In [42]:
```
plt.plot(mlp.loss_curve_)
```

Out[42]: [<matplotlib.lines.Line2D at 0x1f48e6ed048>]



d. Did the training process converge and result in the best model?

The training process converged and then proceeded to overfit, this is seen by the final loss of the model being higher than the best loss

In [43]:
```
mlp.best_loss_, mlp.loss_
```

Out[43]: (0.02220748804456104, 0.02220748804456104)

e. What is the classification accuracy on the training and test datasets?

In [44]:
```
print(f"The classification on the training set compared to the test set is {mlp.sco
re(Xtr_nn, ytr_nn)}/{mlp.score(Xte_nn, yte_nn)}")
```

```
The classification on the training set compared to the test set is 0.99880239520
95808/0.84688995215311
```

2. Refine this network by refining is with GridSearchCV.

In this section a neural network's hyperparameters are tuned using GridSearchCV.

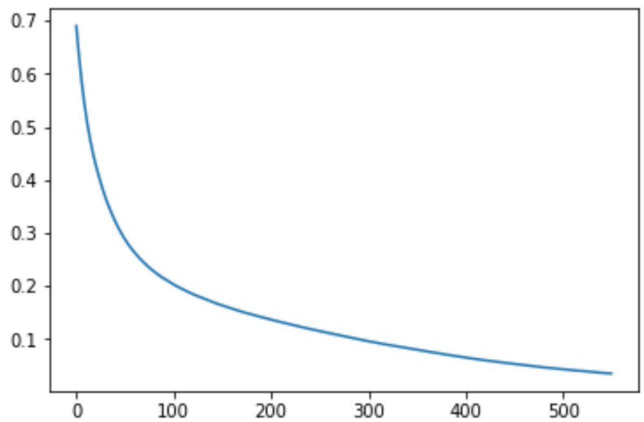
- `hidden_layer_size` was optimized to ensure that that a the complexity of the model was appropriate for the problem
- `activation` was optimized to ensure that the activation function of the model is capaable of representing the problm. Relu is max(0, x) and allows only relevant signals to pass through to the next layer. tanh is good when the sign of the signal is relevant but the magnitude needs to be limited. logistic is a mix of the two.
- `learning_rate` and `learning_rate_init` were used to ensure that the network had the ability to fine tune itself to find the best minima.

source: https://www.quora.com/How-should-I-choose-a-proper-activation-function-for-the-neural-network?share=1 (https://www.quora.com/How-should-I-choose-a-proper-activation-function-for-the-neural-network?share=1)

In [45]:
```
params = {
    "hidden_layer_sizes": [(50), (100), (100, 100)],
    "activation": ['logistic', 'tanh', 'relu'],
    "learning_rate_init": [0.0001, 0.001, 0.1, 0.5, 1],
    "learning_rate": ["constant", "invscaling", "adaptive"]
}

gs_mlp = GridSearchCV(MLPClassifier(max_iter=1000), params, n_jobs=-1).fit(Xtr_nn,
ytr_nn)
```

In [46]:
```
plt.plot(gs_mlp.best_estimator_.loss_curve_)
```

Out[46]:
```
[<matplotlib.lines.Line2D at 0x1f48c9dd240>]
```



The neural network using GridSearchCV showed no sign of improvement over the default parameters.

In [47]:
```
gs_mlp.score(Xtr_nn, ytr_nn), gs_mlp.score(Xte_nn, yte_nn)
```

Out[47]:
```
(0.9952095808383233, 0.8325358851674641)
```

In [48]:
```
print(f"the default neural network has hidden layers(s) of size {gs_mlp.best_estima
tor_.hidden_layer_sizes} which uses {gs_mlp.best_estimator_.activation} activations
")
```

```
the default neural network has hidden layers(s) of size (100, 100) which uses ta
nh activations
```

In [49]:
```
print(f"the neural network was trained in {len(gs_mlp.best_estimator_.loss_curve_)}
iterations")
```

```
the neural network was trained in 550 iterations
```

3. Build another Neural Network with inputs selected from RFE with regression.

(Use the best model generated in Task 3) and selection with decision tree (use the best model from Task 2).

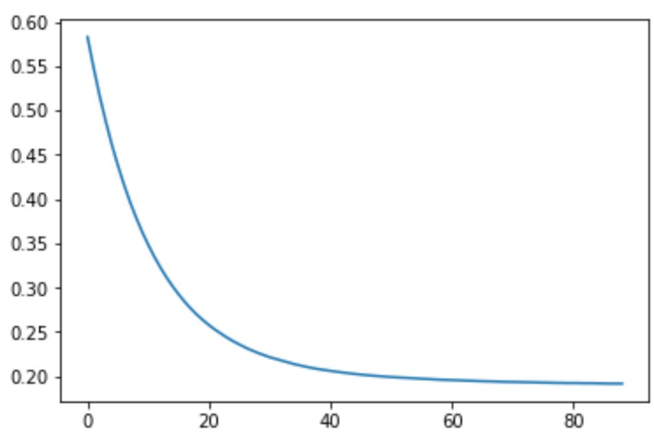
In [50]:
```
Xtr_nn_rfe, Xte_nn_rfe = rfe.transform(reg_X_train), rfe.transform(reg_X_test)
ytr_nn_rfe, yte_nn_rfe = ytr_nn, yte_nn
```

a. Did feature selection help here? Any changes in network architecture? What inputs are being used?

The inputs being used are G1 and G2, feature selection has increased performance and training time measurably, requiring only 100 epochs. The architecture used is is the default.

In [51]:
```
mlp_rfe = MLPClassifier(max_iter=1000).fit(Xtr_nn_rfe, ytr_nn_rfe)

plt.plot(mlp_rfe.loss_curve_)
display(
    f"train acc:{mlp_rfe.score(Xtr_nn_rfe, ytr_nn_rfe)} test acc: {mlp_rfe.score(Xt
e_nn_rfe, yte_nn_rfe)}",
    f"best loss: {mlp_rfe.best_loss_} loss:{mlp_rfe.loss_}",
    mlp_rfe
)
```

```
'train acc:0.9209580838323354 test acc: 0.8899521531100478'

'best loss: 0.19165807800923226 loss:0.19165807800923226'

MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=1000, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
```



b. What is the classification accuracy on the train and test datasets? Any improvements?

In [52]:
```
f"train acc:{mlp_rfe.score(Xtr_nn_rfe, ytr_nn_rfe)} test acc: {mlp_rfe.score(Xte_nn
_rfe, yte_nn_rfe)}"
```

Out[52]:
```
'train acc:0.9209580838323354 test acc: 0.8899521531100478'
```

c. How many iteration are needed to train this network?

In [53]:
```
print(f"the model was trained in only {len(mlp_rfe.loss_curve_)} iterations")
```

```
the model was trained in only 89 iterations
```

d. Do you see any sign of over-fitting?

No, the difference between the train and test performance for the GridSearch/RFE model only approximately 4%, this is insignificant and is much lower than the 16% difference found in the pure GridSearch model.

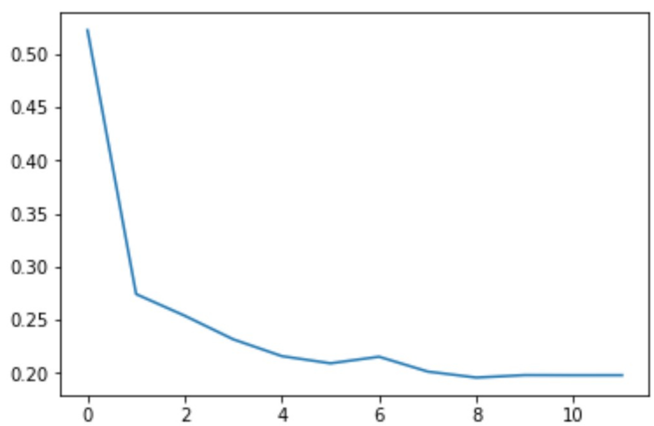
e. Did the training process converge and result in the best model?

Yes, the training process converged.

f. Use GridSearchCV to tune the network to see whether the change in network architecture can further improve the performance.

In [54]:
```
params = {
    "hidden_layer_sizes": [(50), (100), (100, 100)],
    "activation": ['logistic', 'tanh', 'relu'],
    "learning_rate_init": [0.0001, 0.001, 0.1, 0.5, 1],
    "learning_rate": ["constant", "invscaling", "adaptive"]
}

gs_mlp_rfe = GridSearchCV(MLPClassifier(max_iter=1000), params, n_jobs=-1).fit(Xtr_
nn_rfe, ytr_nn_rfe)

plt.plot(gs_mlp_rfe.best_estimator_.loss_curve_)
display(
    f"train acc:{gs_mlp_rfe.score(Xtr_nn_rfe, ytr_nn_rfe)} test acc: {gs_mlp_rfe.sc
ore(Xte_nn_rfe, yte_nn_rfe)}",
    f"best loss: {gs_mlp_rfe.best_estimator_.best_loss_} loss:{gs_mlp_rfe.best_esti
mator_.loss_}",
    gs_mlp_rfe
)
```

```
'train acc:0.9209580838323354 test acc: 0.8803827751196173'

'best loss: 0.19584926763772634 loss:0.1980439658646253'

GridSearchCV(cv=None, error_score='raise',
       estimator=MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto
', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=1000, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'hidden_layer_sizes': [50, 100, (100, 100)], 'activation': ['
logistic', 'tanh', 'relu'], 'learning_rate_init': [0.0001, 0.001, 0.1, 0.5, 1],
'learning_rate': ['constant', 'invscaling', 'adaptive']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```



In [55]:
```
print(f"The parameters used are the same as the those used for the earlier neural n
etwork gridsearch, for the same reasons. This model trained in {len(gs_mlp_rfe.best
_estimator_.loss_curve_)} epochs and did not reach a stable minima.")
```

```
The parameters used are the same as the those used for the earlier neural networ
k gridsearch, for the same reasons. This model trained in 12 epochs and did not
reach a stable minima.
```

In [56]:
```
accuracy_overview['neural_network'] = format_accuracy_overview(yte_nn_rfe, mlp_rfe.
predict(Xte_nn_rfe), ['G1', 'G2'])
```

4. Using the comparison methods, which of the models (i.e one with selected variables and another with all variables) appears to be better? From the better model, can you identify which students to target? Can you provide some descriptive summary of those students?

In [57]:
```
display(
    format_accuracy_overview(yte_nn, mlp.predict(Xte_nn), []),
    format_accuracy_overview(yte_nn, gs_mlp.predict(Xte_nn), []),
    format_accuracy_overview(yte_nn_rfe, mlp_rfe.predict(Xte_nn_rfe), []),
    format_accuracy_overview(yte_nn_rfe, gs_mlp_rfe.predict(Xte_nn_rfe), [])
)
```

```
{'precision': 0.9080459770114943,
 'recall': 0.7669902912621359,
 'f1': 0.8315789473684211,
 'accuracy': 0.84688995215311,
 'ROC': (array([0.        , 0.0754717, 1.        ]),
  array([0.        , 0.76699029, 1.        ]),
  array([2, 1, 0], dtype=int64)),
 'AUC': 0.8457592965744642,
 'most_important_features': []}

{'precision': 0.8863636363636364,
 'recall': 0.7572815533980582,
 'f1': 0.8167539267015707,
 'accuracy': 0.8325358851674641,
 'ROC': (array([0.        , 0.09433962, 1.        ]),
  array([0.        , 0.75728155, 1.        ]),
  array([2, 1, 0], dtype=int64)),
 'AUC': 0.8314709653782745,
 'most_important_features': []}

{'precision': 0.9,
 'recall': 0.8737864077669902,
 'f1': 0.8866995073891626,
 'accuracy': 0.8899521531100478,
 'ROC': (array([0.        , 0.09433962, 1.        ]),
  array([0.        , 0.87378641, 1.        ]),
  array([2, 1, 0], dtype=int64)),
 'AUC': 0.8897233925627405,
 'most_important_features': []}

{'precision': 0.875,
 'recall': 0.883495145631068,
 'f1': 0.8792270531400966,
 'accuracy': 0.8803827751196173,
 'ROC': (array([0.        , 0.12264151, 1.        ]),
  array([0.        , 0.88349515, 1.        ]),
  array([2, 1, 0], dtype=int64)),
 'AUC': 0.8804268180985528,
 'most_important_features': []}
```

The neural network model most likely to be chosen would be the neural network trained with RFE. This is due to the high f1 score. When selecting students to target the variables which are the most important are their G1 and G2 scores. These were the variables selected by RFE when used for regression and result in the highest performance when used in a neural network.

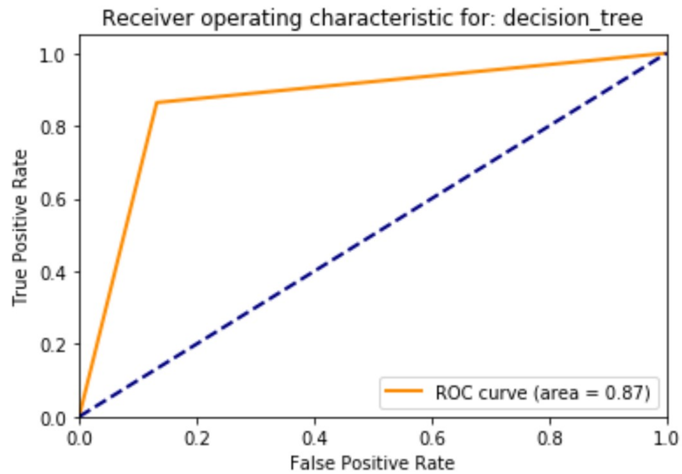
Task 5. Comparing Predictive Models

(4 marks)

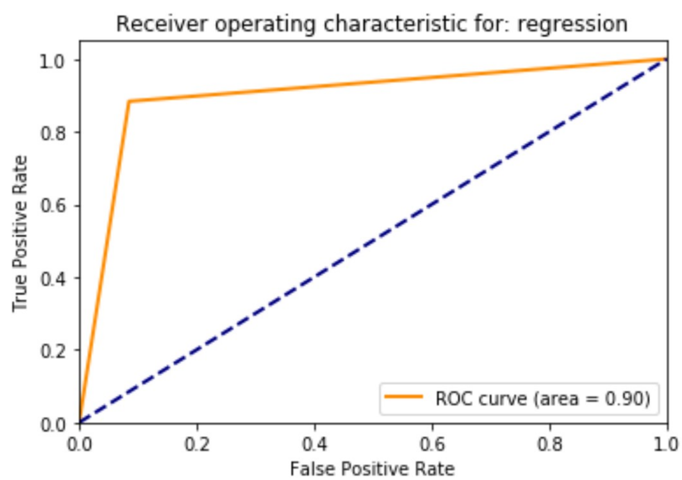
1. Using the comparison methods to compare the best decsision tree model, the best regression model and the best neural network model.

a. Discuss the findings led by (a) ROC Chart and Index; (b) Accuracy Score; (c) Classification Report.

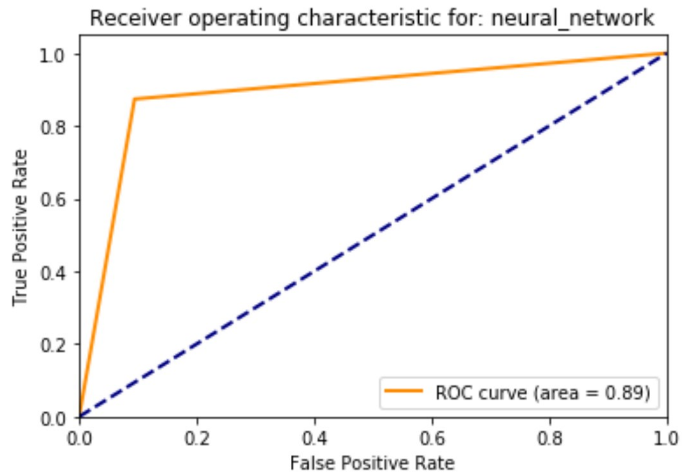
In [58]:
```
def print_roc(model_name):
    fpr, tpr, _ = accuracy_overview[model_name]["ROC"]
    roc_auc = auc(fpr, tpr)

    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic for: ' + model_name)
    plt.legend(loc="lower right")
    plt.show()

# Display an ROC graph and print metrics for each model
for model_name in accuracy_overview:
    print_roc(model_name)
    for metric in accuracy_overview[model_name]:
        print("{}: {}".format(metric.ljust(25), accuracy_overview[model_name][metri
c]))
    print("\n")
```




```
precision                 : 0.8640776699029126
recall                    : 0.8640776699029126
f1                        : 0.8640776699029126
accuracy                  : 0.8660287081339713
ROC                       : (array([0.        , 0.13207547, 1.        ]), array([
0.        , 0.86407767, 1.        ]), array([2, 1, 0], dtype=int64))
AUC                       : 0.8660010991023998
most_important_features   : ['G2', 'G1', 'Dalc', 'Fedu', 'Fjob']
```




```
precision                 : 0.91
recall                    : 0.883495145631068
f1                        : 0.896551724137931
accuracy                  : 0.8995215311004785
ROC                       : (array([0.        , 0.08490566, 1.        ]), array([
0.        , 0.88349515, 1.        ]), array([2, 1, 0], dtype=int64))
AUC                       : 0.8992947426268547
most_important_features   : ['G1', 'G2', 'failures', 'school_THS', 'Fjob_health']
```




```
precision                  : 0.9
recall                     : 0.8737864077669902
f1                         : 0.8866995073891626
accuracy                   : 0.8899521531100478
ROC                        : (array([0.        , 0.09433962, 1.        ]), array([
0.        , 0.87378641, 1.        ]), array([2, 1, 0], dtype=int64))
AUC                        : 0.8897233925627405
most_important_features    : ['G1', 'G2']
```

The regression model performs the best although all model types are very similar in their predictive accuracies. This means that the linear regression model is the best choice due to its accuracy and interpretability.

If a model had significantly higher precision or recall it may alter what model is chosen depending on the task. A higher false positive rate is preffered due to wanting to target students who need help, therefore a higher recall (or sensitivity) value would be a basis for choosing a model.

b. Do all the models agree on the customers' characteristics? How do they vary?

In [59]:
```
print(accuracy_overview["decision_tree"]["most_important_features"])
print(accuracy_overview["regression"]["most_important_features"])
print(accuracy_overview["neural_network"]["most_important_features"])
```

```
['G2', 'G1', 'Dalc', 'Fedu', 'Fjob']
['G1', 'G2', 'failures', 'school_THS', 'Fjob_health']
['G1', 'G2']
```

All models agree that G1 and G2 are most important with negligible variation.

2. Summarise your findings and present the results in a table.

In [60]:
```
pd.DataFrame(accuracy_overview).drop("ROC")
```

Out[60]:

	decision_tree	neural_network	regression
AUC	0.866001	0.889723	0.899295
accuracy	0.866029	0.889952	0.899522
f1	0.864078	0.8867	0.896552
most_important_features	[G2, G1, Dalc, Fedu, Fjob]	[G1, G2]	[G1, G2, failures, school_THS, Fjob_health]
precision	0.864078	0.9	0.91
recall	0.864078	0.873786	0.883495

3. Finally, based on all models and analysis, is there a particular model you will use in decision making? How the outcome of this study can be used by decision makers?

The regression model has the best accuracy metrics, so this model would be most suitable for decision making based on a formal prediction of students' G3 grades.

However, the decision tree, despite having a lower accuracy than regressions, offers a better degree of interpretability and could be used in informal contexts for a quick judgement of whether or not a student should be targetted. E.g. "Is G1 less than X and G2 less than Y? If so, the student should be targetted."

The neural network, due to its poorer performance compared to regression, as well as having a low degree of interpretability, should not be used for any decision making.

4. Can you summarise positives and negaitives of each modelling method based on this analysis?

Decision tree

Pros:

- Very easy to interpret. The model can be summarized into 2 simple if then rules. As can be seen in the image at 2.2.d.
- Is very robust. It can handle most data types. Only catagorical data has to be encoded.
- Relatively fast training time.

Cons:

- Cannot handel complicated relationship. This is the reason why the accuracy is not higher than 86% on the test data.
- Is very sensitive to overfitting. As can be seen in the default decision tree that had a 100% accuracy on the training set while it only had a 82% accuracy on the test set.

Regression

Pros:

- Good interpretability (visualization of correlations, model is a mathematical function).
- Very fast training time due to simple, parallelizable linear algebra operations.
- Higher accuracy with minimal overfitting compared to decision trees for this particular dataset.
- Rigorous scientific and mathematical foundation and acceptance.

Cons:

- Restricted to input that can only be expressed numerically (and requires additional transformation steps to encode non-numerical input numerically).
- Sensitive to outliers (requires additional scaling/transformation to compensate for this).
- Excessive amounts of scaling/transformation can introduce unrealistic biases and negatively affect interpretability.
- Does not accept missing values (requires imputation).
- Linear regressions are not appropriate if the phenomenon is non-linear. Likewise, polynomial and exponential regressions are not appropriate if the phenomenon is non-polynomial, non-exponential, etc.
- Relative to neural networks, regressions cannot model complicated phenomena.

Neural network

Pros:

- High predictive accuracy.
- Can work with data which isn't linearly seperable.
- Can scale complexity of model to suit problem.
- Training can be parallelized.

Cons:

- Long training time.
- Blackbox, hard to interpret results.
- Many hyper parameters.
- Can overfit easily.
- Requires appropriate preprocessing of data.