Pattern Analysis and Machine Intelligence

# A Decision Tree Classification Model for University Admission System

Submitted by: Sumedha
M.Tech (S.P.D.D)
Second Semester
Roll No.: 2K19/SPD/17

# RESEARCH PAPER

(IJACSA) International Journal of Advanced Computer Science and Applications,
Vol. 3, No. 10, 2012

# A Decision Tree Classification Model for University Admission System

Abdul Fattah Mashat
Faculty of Computing and Information Technology
King Abdulaziz University
Jeddah, Saudi Arabia

Mohammed M. Fouad
Faculty of Informatics and Computer Science
The British University in Egypt (BUE)
Cairo, Egypt

Philip S. Yu
University of Illinois,
Chicago, IL, USA
King Abdulaziz University
Jeddah, Saudi Arabia

Tarek F. Gharib
Faculty of Computing and Information Technology
King Abdulaziz University
Jeddah, Saudi Arabia

***Abstract*— Data mining is the science and techniques used to analyze data to discover and extract previously unknown patterns. It is also considered a main part of the process of knowledge discovery in databases (KDD). In this paper, we introduce a supervised learning technique of building a decision tree for King Abdulaziz University (KAU) admission system. The main objective is to build an efficient classification model with high recall under moderate precision to improve the efficiency and effectiveness of the admission process. We used ID3 algorithm for decision tree construction and the final model is evaluated using the common evaluation methods. This model provides an analytical view of the university admission system.**

*Keywords- Data Mining; Supervised Learning; Decision Tree; University Admission System; Model Evaluation*

## I. Introduction

Data mining, the science and technology of exploring data in order to discover unknown patterns, is an essential part of the overall process of knowledge discovery in databases (KDD). In today's computer driven world, these databases contain massive quantities of information. The accessibility and abundance of this information make data mining a matter of considerable importance and necessity [1].

Data mining includes many methods and techniques, but mainly we can divide them into two main types; verification and discovery. In verification-oriented methods, the system verify the user's input hypothesis like goodness of fit, hypothesis testing and ANOVA test. On the other hand, discovery-oriented methods automatically find new rules and identify patterns in the data. Discovery-oriented methods include clustering, classification and regression techniques.

Supervised learning methods attempt to discover the relationship between input attributes and target attribute. Once the model is constructed, it can be used for predicting the value of the target attribute for a new input data. There are two main supervised models: classification models, which is our interest in this paper, and regression models. Classification models build a classifier that maps the input space (features) into one of the predefined classes. For example, classifiers can be used to classify objects in an outdoor scene image as person, vehicle, tree, or building. While, regression models map the input space into real-value domain. For example, a regression model can be built to predict house price based on its characteristics like size, no. of rooms, garden size and so on.

In data mining, a decision tree (it may be also called Classification Tree) is a predictive model that can be used to represent the classification model. Classification trees are useful as an exploratory technique and are commonly used in many fields such as finance, marketing, medicine and engineering [2, 3, 4, 5]. The use of decision trees is very popular in data mining due to its simplicity and transparency. Decision trees are usually represented graphically as a hierarchical structure that makes them easier to be interpreted than other techniques. This structure mainly contains a starting node (called root) and group of branches (conditions) that lead to other nodes until we reach leaf nodes that contain final decision of this route. The decision tree is a self explanatory model because its representation is very simple. Each internal node test an attribute while each branch corresponds to attribute value (or range of values). Finally each leaf assigns a classification.

Fig. 1 shows an example for a simple decision tree for "Play Tennis" classification. It simply decides whether to play tennis or not (i.e. classes are Yes or No) based on three weather attributes which are outlook, wind and humidity [6].


Figure 1. Decision Tree Example.

As shown in Fig. 1, if we have a new pattern with attributes outlook is "Rain" and wind is "Strong", we shall decide not to play tennis because the route starting from the root node will end up with a decision leaf with "NO" class.





In this paper, we introduce a supervised learning technique of building a decision tree model for King Abdulaziz University (KAU) admission system to provide a filtering tool to improve the efficiency and effectiveness of the admission process. KAU admission system contains a database of records that represent applicant student information and his/her status of being rejected or accepted to be enrolled in the university. Analysis of these records is required to define the relationship between applicant's data and the final enrollment status.

This paper is organized into five sections. In section 2, the decision tree model is presented. Section 3 provides brief details about commonly used methods for classification model evaluation. In section 4, experimental results are presented and analyzed with respect to model results and admission system perspective. Finally, the conclusion of this work are presented in Section 5.

## II. Decision Tree Model

A decision tree is a classifier expressed as a recursive partition of the input space based on the values of the attributes. As stated earlier, each internal node splits the instance space into two or more sub-spaces according to certain function of the input attribute values. Each leaf is assigned to one class that represents the most appropriate or frequent target value.

Instances are classified by traversing the tree from the root node down to a leaf according to the outcome of the test nodes along this path. Each path can be transformed then into a rule by joining the tests along this path. For example, one of the paths in Fig. 1 can be transformed into the rule: "If Outlook is Sunny and Humidity is Normal then we can play tennis". The resulting rules are used to explain or understand the system well.

There are many algorithms proposed for learning decision tree from a given data set, but we will use ID3 algorithm due to its simplicity for implementation. In this section we will discuss ID3 algorithm for decision tree construction and some of the frequently used functions used for splitting the input space.

### A. ID3 Algorithm

ID3 is a simple decision tree learning algorithm developed by Quinlan [7]. It simply uses top-down, greedy search over the set of input attributes to be tested at every tree node. The attribute that has the best split, according to the splitting criteria function discussed later, is used to create the current node. This process is repeated at every node until one of the following conditions is met:

- Every attribute is included along this path.
- Current training examples in this node have the same target value.

Fig 2 shows the pseudo code for ID3 algorithm to construct a decision tree over a training set ($S$), input feature set ($F$), target feature ($c$) and some split criterion ($SC$).

### B. Splitting Criterion

ID3 algorithm uses some splitting criterion function to select the best attribute to split with. In order to define this criterion, we need first to define entropy index that measures the degree of impurity of the certain labeled dataset.

For a given labeled dataset S with some examples that have n (target values) classes {c1, c2, ..., cn}, we define entropy index (E) as in (1):

$$E(S) = \sum_{i=1}^{n} p_i * \log(p_i), \quad p_i = \frac{|S_i|}{S} \qquad (1)$$

Where $S_i$ the subset of the examples that have a target value that equals to $c_i$. Entropy (E) has its maximum value if all the classes have equal probability.

```
ID3(S, F, c, SC)
Output: Decision Tree T
  Create a new tree T with a single root node
  IF no more split (S) THEN
    Mark T as a leaf with the most common value of c in label.
  ELSE
    ∀f_i ∈ F find f that has best SC(f_i, S)
    Label t with f
    FOR each value v_i of f
      Set Subtree_i = ID3(S_{f=v_i}, F − {f}, c, SC)
      Connect node t to Subtree_i with edge labeled v_i
    END
  END
  Return T
```

Figure 2. ID3 algorithm

#### 1) Information Gain

To select the best attribute for splitting of certain node, we can use information gain measure. Gain (S, A) of an attribute A, for set of examples S. Information gain is defined as in (2):

$$Gain(S, A) = E(S) - \sum_{v \in V(A)} \frac{|S_{A=v}|}{|S|} E(S_{A=v}) \qquad (2)$$

Where E(S) is the entropy index for dataset S, V(A) is the set of all values for attribute A.

#### 2) Gain Ratio

Another measure can be used as a splitting criterion which is gain ratio. It is simply the ratio between information gain value Gain(S, A) and another value which is split information SInfo(S, A) that is defined as in (3):

$$SInfo(S, A) = -\sum_{v \in V(A)} \frac{|S_{A=v}|}{|S|} * \log \frac{|S_{A=v}|}{|S|} \qquad (3)$$

#### 3) Relief Algorithm

Kira and Rendell proposed the original Relief algorithm to estimate the quality of attributes according to how well their values distinguish between examples that are near to each

# DATASET USED

## Dataset Used in Research Paper:

Admission Dataset from King Abdulaziz University (KAU)

In this paper, we are provided by sample datasets from KAU system database that represent applicant student information and his/her status of being rejected or accepted to be enrolled in the university in three consecutive years (2010, 2011 and 2012). The dataset contains about 80262 records, while each record represents an instance with 4 attributes and the class attribute with two values: Rejected and Accepted. The classes are distributed as 53% of the total records for "Rejected" and 47% for "Accepted" class. Table 2 shows detailed information about datasets attributes.

TABLE II. SUMMARY OF DATASET ATTRIBUTES

| Attribute | Possible values |
|---|---|
| *Gender* | Student's gender<br>• Male<br>• Female |
| *HS_Type* | Type of high school study<br>• TS = Scientific Study<br>• TL = Literature Study<br>• TU = Unknown/Missing |
| *HS_Grade* | High school grade<br>• A = mark ≥ 85<br>• B = 75 ≥ mark > 85<br>• C = 65 ≥ mark > 75<br>• D = 50 ≥ mark > 65 |
| *Area* | Code for student's region city (116 distinct value) |

## Dataset Used while implementing Research Paper

UCLA institute of Digital Research and Education Graduate admission data

Source: Kaggle

Table 1

| admit | gre | gpa | rank |
|---|---|---|---|
| 0 | 380 | 3.61 | 3 |
| 1 | 660 | 3.67 | 3 |
| 1 | 800 | 4 | 1 |
| 1 | 640 | 3.19 | 4 |
| 0 | 520 | 2.93 | 4 |
| 1 | 760 | 3 | 2 |
| 1 | 560 | 2.98 | 1 |
| 0 | 400 | 3.08 | 2 |
| 1 | 540 | 3.39 | 3 |
| 0 | 700 | 3.92 | 2 |
| 0 | 800 | 4 | 4 |
| 0 | 440 | 3.22 | 1 |

# ALGORITHM

Step-1: Begin the tree with the root node, say S, which contains the complete dataset.

Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).

Attribute Selection Method:

Information Gain: It calculates how much information a feature provides us about a class.

It is the measurement of changes in entropy of a dataset based on an attribute.

Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature) ]

Entropy: It specifies randomness in data. Entropy can be calculated as:
Entropy(s)= -P(yes)*$\log_2$ P(yes)- P(no)*$\log_2$ P(no) Where, S= Total number of samples P(yes)= probability of yes P(no)= probability of no

# ALGORITHM

Step-3: Divide the S into subsets that contains possible values for the best attributes.

Step-4: Recursively make new decision trees using the subsets of the dataset created in step -2 and 3. Continue this process until a stage is reached where we cannot further classify the nodes and called the final node as a leaf node.

# CODE

```
In [1]: # Importing the libraries
        import pandas as pd
        import numpy as np
        from sklearn.model_selection import train_test_split
        import matplotlib.pyplot as plt
        %matplotlib inline
        import seaborn as sns
        from sklearn.datasets import load_iris
        from sklearn.externals.six import StringIO
        import pydotplus
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/externals/six.py:31: DeprecationWarning: The module is deprecated
in version 0.21 and will be removed in version 0.23 since we've dropped support for Python 2.7. Please rely on the of
ficial version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", DeprecationWarning)
```

```
In [2]: # Loading the data
        df = pd.read_csv("/Users/sumedha/Desktop/Decision Tree.csv")
        df.head()
```

Out[2]:

| | admit | gre | gpa | rank |
|---|---|---|---|---|
| 0 | 0 | 380 | 3.61 | 3 |
| 1 | 1 | 660 | 3.67 | 3 |
| 2 | 1 | 800 | 4.00 | 1 |
| 3 | 1 | 640 | 3.19 | 4 |
| 4 | 0 | 520 | 2.93 | 4 |

```
In [4]: #processing the data
        X = df.iloc[:, 1:].values
        y = df.iloc[:, 0].values
```

```
In [5]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.36, random_state=0)
```

# CODE

```
In [5]: #fitting decision tree
        from sklearn import tree
        clf = tree.DecisionTreeClassifier()
        clf = clf.fit(X_train, y_train)
```

```
In [20]: y_predict_train = clf.predict(X_train)
         y_predict_test = clf.predict(X_test)
         print('First five test set values are {}'.format(X_test[:6,:]))
         print('Predictions for first five test set values are {}'.format(y_predict_test[:6]))
```

```
First five test set values are [[580.     3.4    2.  ]
 [440.     2.98   3.  ]
 [560.     2.65   3.  ]
 [660.     3.07   3.  ]
 [680.     3.34   2.  ]
 [620.     3.18   2.  ]]
Predictions for first five test set values are [0 0 1 0 0 1]
```

```
In [7]: from sklearn.metrics import accuracy_score, f1_score
        acc_test = accuracy_score(y_test,y_predict_test)
        acc_train = accuracy_score(y_train,y_predict_train)
        f1_test = f1_score(y_test,y_predict_test)
        f1_train = f1_score(y_train,y_predict_train)
        print('The accuracy of the classifier on test data is {:.2f} out of 1'.format(acc_test))
        print('The accuracy of the classifier on training data is {:.2f} out of 1'.format(acc_train))
        print('The f1_score of the classifier on test data is {:.2f} out of 1'.format(f1_test))
        print('The f1_score of the classifier on training data is {:.2f} out of 1'.format(f1_train))
```

```
The accuracy of the classifier on test data is 0.61 out of 1
The accuracy of the classifier on training data is 0.99 out of 1
The f1_score of the classifier on test data is 0.40 out of 1
The f1_score of the classifier on training data is 0.99 out of 1
```

# CODE

```
In [9]: from sklearn.metrics import confusion_matrix
        cnf_matrix = confusion_matrix(y_test, y_predict_test)
        np.set_printoptions(precision=2)
        cnf_matrix

Out[9]: array([[70, 29],
               [28, 17]])
```

```
In [10]: True_pos = cnf_matrix[0][0]
         False_neg = cnf_matrix[0][1]
         False_pos = cnf_matrix[1][0]
         True_neg = cnf_matrix[1][1]
```

```
In [11]: Accuracy_Acc = (True_pos+True_neg)/(True_pos+False_pos+False_neg+True_neg)
         R_Accepted = (True_pos)/(True_pos+False_neg)
         R_Rejected = (True_neg)/(True_neg+False_pos)
         P_Accepted = (True_pos)/(True_pos+False_pos)
         P_Rejected = (True_neg)/(True_neg+False_neg)
         F1_Accepted = (2*R_Accepted*P_Accepted)/(R_Accepted+P_Accepted)
         F1_Rejected = (2*R_Rejected*P_Rejected)/(R_Rejected+P_Rejected)
         print('The accuracy of the classifier is {:.2f} out of 1'.format(Accuracy_Acc))
         print('The Recall_Accepted of the classifier is {:.2f} out of 1'.format(R_Accepted))
         print('The  Recall_score Rejected of the classifier is {:.2f} out of 1'.format(R_Rejected))
         print('The  Precision_score Accepted of the classifier is {:.2f} out of 1'.format(P_Accepted))
         print('The  Precision_score Rejected of the classifier is {:.2f} out of 1'.format(P_Rejected))
         print('The  F1_score Accepted of the classifier is {:.2f} out of 1'.format(F1_Accepted))
         print('The  F1_score Rejected of the classifier is {:.2f} out of 1'.format(F1_Rejected))

         The accuracy of the classifier is 0.60 out of 1
         The Recall_Accepted of the classifier is 0.71 out of 1
         The  Recall_score Rejected of the classifier is 0.38 out of 1
         The  Precision_score Accepted of the classifier is 0.71 out of 1
         The  Precision_score Rejected of the classifier is 0.37 out of 1
         The  F1_score Accepted of the classifier is 0.71 out of 1
         The  F1_score Rejected of the classifier is 0.37 out of 1
```

# OUTPUT COMPARISION

Output obtained while implementing Research Paper

```
The accuracy of the classifier is 0.68 out of 1
The Recall_Accepted of the classifier is 0.71 out of 1
The  Recall_score Rejected of the classifier is 0.38 out of 1
The  Precision_score Accepted of the classifier is 0.71 out of 1
The  Precision_score Rejected of the classifier is 0.37 out of 1
The  F1_score Accepted of the classifier is 0.71 out of 1
The  F1_score Rejected of the classifier is 0.37 out of 1
```

Output in Research Paper:

TABLE IV. MODEL EVALUATION MEASURES

| MeasureValue | |
|---|---|
| ***Accuracy*** | $Acc = \frac{12305 + 6729}{29056} = 0.655$ |
| ***Recall*** | $R_{Accepted} = \frac{12305}{13843} = 0.889$ <br> $R_{Rejected} = \frac{6729}{15213} = 0.442$ |
| ***Precision*** | $P_{Accepted} = \frac{12305}{20789} = 0.592$ <br> $P_{Rejected} = \frac{6729}{8267} = 0.834$ |
| ***$F_1$ Measure*** | $F1_{Accepted} = \frac{2 * 0.592 * 0.889}{0.592 + 0.889} = 0.711$ <br> $F1_{Rejected} = \frac{2 * 0.834 * 0.442}{0.834 + 0.442} = 0.578$ |

# IMPROVING EVALUATION MEASURES USING HYPER PARAMETERS

In [12]:
```
# Improving Accuracy using Hyperparameters
model = tree.DecisionTreeClassifier(max_depth = 5, min_samples_leaf=11,min_samples_split=35)
model.fit(X_train, y_train)
y_predict_train_m = model.predict(X_train)
y_predict_test_m = model.predict(X_test)
```

In [13]:
```
acc_test = accuracy_score(y_test,y_predict_test_m)
acc_train = accuracy_score(y_train,y_predict_train_m)
f1_test = f1_score(y_test,y_predict_test_m)
f1_train = f1_score(y_train,y_predict_train_m)
print('The accuracy of the classifier on test data is {:.2f} out of 1'.format(acc_test))
print('The accuracy of the classifier on training data is {:.2f} out of 1'.format(acc_train))
print('The f1_score of the classifier on test data is {:.2f} out of 1'.format(f1_test))
print('The f1_score of the classifier on training data is {:.2f} out of 1'.format(f1_train))
```

OUTPUT

```
The accuracy of the classifier on test data is 0.71 out of 1
The accuracy of the classifier on training data is 0.75 out of 1
The f1_score of the classifier on test data is 0.46 out of 1
The f1_score of the classifier on training data is 0.51 out of 1
```

OUTPUT WHEN HYPER-PARAMETERS ARE NOT USED

```
The accuracy of the classifier on test data is 0.62 out of 1
The accuracy of the classifier on training data is 0.99 out of 1
The f1_score of the classifier on test data is 0.38 out of 1
The f1_score of the classifier on training data is 0.99 out of 1
```

# IMPROVING EVALUATION MEASURES USING HYPER PARAMETERS

```
In [14]: cnf_matrix = confusion_matrix(y_test, y_predict_test_m)
         np.set_printoptions(precision=2)
         True_pos = cnf_matrix[0][0]
         False_neg = cnf_matrix[0][1]
         False_pos = cnf_matrix[1][0]
         True_neg = cnf_matrix[1][1]
```

```
In [15]: Accuracy_Acc = (True_pos+True_neg)/(True_pos+False_pos+False_neg+True_neg)
         R_Accepted = (True_pos)/(True_pos+False_neg)
         R_Rejected = (True_neg)/(True_neg+False_pos)
         P_Accepted = (True_pos)/(True_pos+False_pos)
         P_Rejected = (True_neg)/(True_neg+False_neg)
         F1_Accepted = (2*R_Accepted*P_Accepted)/(R_Accepted+P_Accepted)
         F1_Rejected = (2*R_Rejected*P_Rejected)/(R_Rejected+P_Rejected)
         print('The accuracy of the classifier is {:.2f} out of 1'.format(Accuracy_Acc))
         print('The Recall_Accepted of the classifier is {:.2f} out of 1'.format(R_Accepted))
         print('The  Recall_score Rejected of the classifier is {:.2f} out of 1'.format(R_Rejected))
         print('The  Precision_score Accepted of the classifier is {:.2f} out of 1'.format(P_Accepted))
         print('The  Precision_score Rejectted of the classifier is {:.2f} out of 1'.format(P_Rejected))
         print('The  F1_score Accepted of the classifier is {:.2f} out of 1'.format(F1_Accepted))
         print('The  F1_score Rejected of the classifier is {:.2f} out of 1'.format(F1_Rejected))
```

OUTPUT

```
The accuracy of the classifier is 0.77 out of 1
The Recall_Accepted of the classifier is 0.85 out of 1
The  Recall_score Rejected of the classifier is 0.40 out of 1
The  Precision_score Accepted of the classifier is 0.76 out of 1
The  Precision_score Rejectted of the classifier is 0.55 out of 1
The  F1_score Accepted of the classifier is 0.80 out of 1
The  F1_score Rejected of the classifier is 0.46 out of 1
```

OUTPUT WHEN HYPER-PARAMETERS ARE NOT USED

```
The accuracy of the classifier is 0.69 out of 1
The Recall_Accepted of the classifier is 0.73 out of 1
The  Recall_score Rejected of the classifier is 0.38 out of 1
The  Precision_score Accepted of the classifier is 0.72 out of 1
The  Precision_score Rejected of the classifier is 0.39 out of 1
The  F1_score Accepted of the classifier is 0.72 out of 1
The  F1_score Rejected of the classifier is 0.38 out of 1
```

# VISUALIZING DECISION TREE

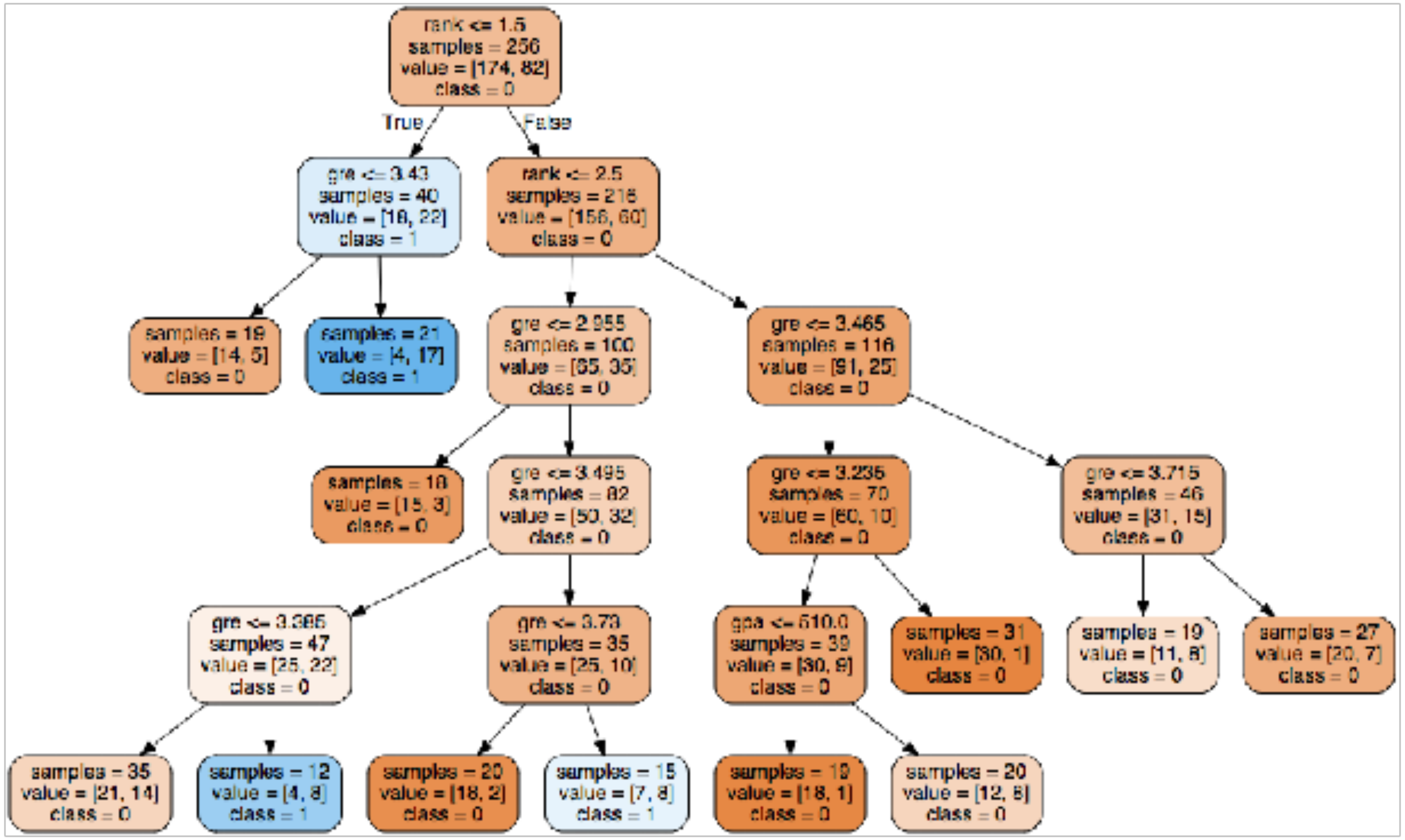
In [16]:
```python
fn=['gpa','gre','rank']
cn=['0', '1']
dot_data = StringIO()
tree.export_graphviz(model,
                     out_file = dot_data,
                     feature_names=fn,
                     class_names= cn,
                     filled=True, rounded=True,impurity = False)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
```

In [17]:
```python
#It creates a pdf and stores it in working folder
graph.write_pdf("Tree_Visualization.pdf")
```

Out[17]: True

---

## OUTPUT



## ADVANTAGES

- Simple to understand
- Less requirement of Data Cleaning

## DISADVANTAGES

- May have over-fitting issue
- As number of labels increase computational complexity also increases