Tecnológico de Monterrey

Sistemas Inteligentes

Decision Tree

Martín Noé Márquez García - A01351211

Abstract

The present paper is focused in the development and implementation of a binary decision tree model, based on a data set that will help me to predict if a student could reach an acceptable average score of the 3 tests based on different features like: gender, parental level of education, lunch, test preparation course.
The dataset used was dowloaded from kaggle: https://www.kaggle.com/spscientist/students-performance-in-exams#StudentsPerformance.csv

Theoretical Framework

Decision Tree

It's a type of diagram that helps us to defines potential outcomes for a collection of related choices, it can handle both numerical and categorical variables.
The paths from the root to lead represent classification rules, so we can say that it consists in a structure in which each internal node represent a test on an attribute, each branch represents the outcome of the test and each node represent a class label.

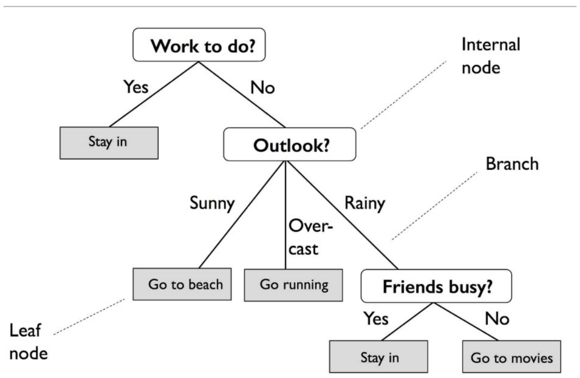

Figure 1: *"Decision Tree Example"*

The types of decision tree depends on the target variable that we are working on, the most popular are:

- Classification Trees: Has a categorical target, so it is useful to predict values of a categorical depends variable from one or more categorical/continuous categorical predictor variables.
- Regression Trees: Are those where we attempt to predict values of continuous variable from others continuous/categorical predictor variables.

One of the biggest advantages is that we can understand and interpret the outputs of the three in a very easy way, even for people with non-analytical background.
It is also one of the fastest way to identify most significant variables and the relation between and with the results we can trying to get a hypothesis of the model results more easy.
Its important to remind that we are working with a huge data base so if we want better results it's recommended to clean our data set.

Random Forest

Is a method that combines a large number of independent decision trees sets with equal distribution. Here each tree is evaluated and the forest prediction will be the average of all the trees that we create.
Each of the tree has been designed to fit in a different scenario, with this skill random forest has the ability to properly adjust to net unknown scenarios.

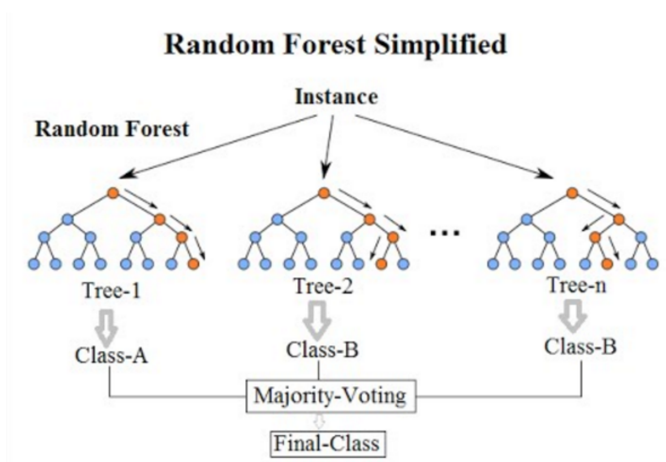

Figure 2: *"Random Forest Example"*

Overfitting

As we know decision trees learns can create complex solutions that do not generalize our data well, and is one of the most practical difficulty for a decision tree model, this is known as overfitting and can be solved by setting constraints on the model parameters an pruning.

Before Start Programming

For the method that will be presented in this paper we need to consider to install some stuff and libraries in python:

Library	Python Enviroment	Conda Enviroment
import pandas as pd	pip install pandas	conda install pandas
from sklearn.tree import DecisionTreeClassifier	pip install scikit-learn	conda install scikit-learn
from sklearn.tree import export_graphviz	pip install graphviz	conda install Graphviz
import matplotlib.pyplot as plt	pip install matplotlib	conda install matplotlib
import pydotplus	pip install pydotplus	conda install pydotplus
from sklearn.model_selection import train_test_split	Same as sklearn.tree	Same as sklearn.tree
import numpy as np	pip install numpy	conda install bumpy
from sklearn.ensemble import RandomForestClassifier	Same as pip install scikit-learn	Same as conda install scikit-learn

Development/Code:

In the first lines of code you will find how I clean and prepare form python my data set. I cleaned in a binary way so that when I create my decision tree model it results more easy and efficient.
As we can see in the next code i'm reading my csv data set and cleaning and dropping all the bad lines existing in my file.

```
if create_dataset:
    df = pd.read_csv("students.csv", error_bad_lines=False)
#eliminated all the bad data
    df.dropna(axis=0, inplace=True)
```

By using iterates to split the content in the columns helps me to separate all my info in different columns to achieve the binary data, with the next code, I apply this fo each of my features.

```
gender = df["gender"]
all_gender = []  # see all the strings in the effects row
for index, row in df.iterrows(): # iterate
    string_ef = row['gender']  # choose the row effects
    for genders in string_ef.split(" "):  # split the strings
        if genders not in all_gender:
    all_gender.append(genders)
```

Then with a help of dictionaries I create a new column for each of the labels included in the column gender with the help of an if function.

```
raw_data = {'female': [], 'male':
[]} #Create a library with the
info of the columns of gender
for index, row in df.iterrows():
    gender_row =
row['gender'].split(' ') #split
the strings
    for gender in all_gender:
        if gender in gender_row:
#append a 1 or a 0 if the label
is in the row
raw_data[gender].append(1)
        else:
raw_data[gender].append(0)
for gender, values in
raw_data.items():
    df[gender] = values
```

Having this we can now drop the old columns that we don't need anymore.

```
df.drop(['gender', 'race/
ethnicity', 'parental level of
education', 'lunch', 'test
preparation course'], axis=1,
        inplace=True)  # drop the
columns
```

The data set has the 3 different average of the 3 test, so for this application I decide to create a new column with the average of the 3 results.

```
df['average'] = (df['math score']
+df['writing score']+df['reading
score'])/3
```

```
df['average'] =
df['average'].apply(lambda x:
round(x))
```

After doing this with the help of the cut function I create a new column where I collect my values and average and classify them between failed and accredited (0,70,100)

```
df['labels'] =
pd.cut(x=df['labels'], bins=[0,
70, 100], labels=['Failed',
'Accredited'])
```

Now we can drop those features

```
df.drop(['math score', 'reading
score', 'writing score',
'average'], axis=1, inplace=True)
```

We can see how my data set change in the next pictures.

gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
female	group B	bachelor's degree	standard	none	72	72	74
female	group C	some college	standard	completed	69	90	88
female	group B	master's degree	standard	none	90	95	93
male	group A	associate's degree	free/reduced	none	47	57	44
male	group C	some college	standard	none	76	78	75
female	group B	associate's degree	standard	none	71	83	78
female	group B	some college	standard	completed	88	95	92

Figure 3: *"Orginal CSV"*

	female	male	bachelors degree	some college	masters degree	associates degree	high school	some high school	standard	free/reduced	none	completed	labels
0	1	0	1	0	0	0	0	0	1	0	1	0	Accredited
1	1	0	0	1	0	0	0	0	1	0	0	1	Accredited
2	1	0	0	0	1	0	0	0	1	0	1	0	Accredited
3	0	1	0	0	0	1	0	0	0	1	1	0	Failed
4	0	1	0	1	0	0	0	0	1	0	1	0	Accredited
5	1	0	0	0	0	1	0	0	1	0	1	0	Accredited
6	1	0	0	1	0	0	0	0	1	0	0	1	Accredited
7	0	1	0	1	0	0	0	0	0	1	1	0	Failed

Figure 4: *"Clean CSV"*

Having my clean csv I start creating my binary decision tree, so first I have ti declare my label variables and features variable.

```
X = df.drop(['labels'], axis=1)
#create my X varibles "Features"
for the tree
y = df['labels'] #create my y
variables "label" for the tree
```

Then I use the the decision tree classifier function to tell my python program that I will be working with a classification tree.

```
clf = DecisionTreeClassifier()
x_train, x_val, y_train, y_val =
train_test_split(X, y,
test_size=0.25, random_state=213)
```

As we can see we use 2 parameters of the train test split function to separate my training data and my test data. In test size you choose how many percentage of you want to use of your data set to train your model and the rest go for the test.
And the random state is used to choose the same data every time you run your program, so you results don't have changes.

To choose the value of 195 on random state I first run the program with a for function and having a range of values, I play with the "random state" variable. With an array I safe all this values and make a graph to see which value gives me the best accuracy.

```
training_accuracy2 = []
test_accuracy2 = []
random_state = range(1, 500)
for n_samples in random_state:
    x_train, x_val, y_train,
y_val = train_test_split(X, y,
test_size=0.25,
random_state=n_samples)
    tree =
DecisionTreeClassifier(max_depth=
12, random_state=n_samples)
    tree.fit(x_train, y_train)

training_accuracy2.append(clf.sco
re(x_train, y_train))

test_accuracy2.append(clf.score(x
_val, y_val))
    print('Accuracy on the
training subset 2:', n_samples,
format(tree.score(x_train,
y_train)))
    print('Accuracy on the test
subset 2:', n_samples,
format(tree.score(x_val, y_val)))
plt.plot(random_state,
training_accuracy2,
label='Accuracy of the Training
set')
plt.plot(random_state,
test_accuracy2, label='Accuracy
of the Test set')
plt.ylabel('% Accuracy')
plt.xlabel('Random state Value')
plt.legend()
plt.show()
```

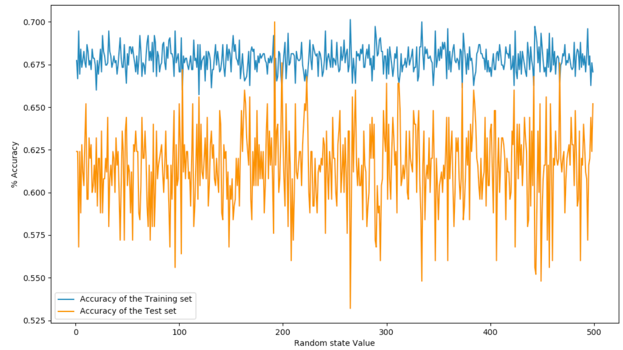

Figure 5: *"Accuracy Graph"*

After doing this i found that my best value was 195 so now I can create my classification decision tree with this parameters.

```
x_train, x_val, y_train, y_val =
train_test_split(X, y,
test_size=0.25, random_state=195)
tree =
DecisionTreeClassifier(max_depth=
7, random_state=195)
tree.fit(x_train, y_train)
print('Accuracy on the training
subset 2:',
format(tree.score(x_train,
y_train)))
print('Accuracy on the test
subset 2:',
format(tree.score(x_val, y_val)))
clf = tree
```

As output I get an accuracy:

Accuracy on the training subset: 0.668
Accuracy on the test subset: 0.64

Having this I create my graphic tree with the next code:

```
dot_data = export_graphviz(clf,
out_file=None,
class_names=['Reprobado',
'Logrado'],
```

```
feature_names=X.columns,
filled=True, rounded=True,
special_characters=True)
graph =
pydotplus.graph_from_dot_data(dot
_data)
graph.write_pdf("tree-vis")
```

Testing the tree manually

To test my decision tree with values external to those of the csv use the following code.

```
Xnew = ([[1, 0, 0, 1, 0, 0, 0,
0, 0, 1, 1, 0]])
y_predict =
forest.predict(Xnew)
print('Case selected is:',
Xnew[0], y_predict)
```

In this first case we can see that we are looking the results at the test of a female, parents have some college, with a standard lunch and she doesn't assist to the preparation course.

Row selected is: [1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0] ['Accredited']

In the second case we can see that we are looking the results at the test of a male, parents have associate's degree, with a free/reduced lunch and he doesn't assist to the preparation course.

Row selected is: [0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0] ['Failed']

Forest implementation

I create a random forest just to compare the results against my decision tree classifier, I got almost the same values in the train and test accuracy but I found something ver interesting when I graphed their feature importance. We can se that in the decision tree we have male as the most important feature this means that this feature has the greatest Gini in our tree, but in our Random forest we can appreciate that the lunch free/reduced was the one with the highest Gini.

Comparing Features

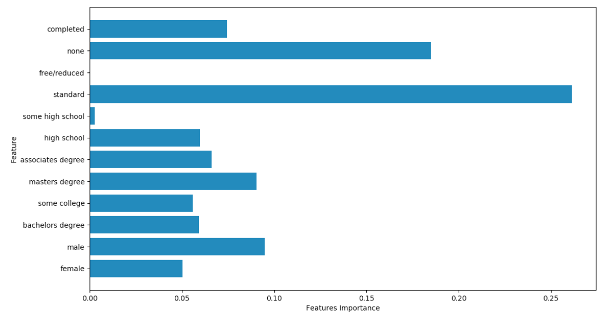

***Figure 6:** "Features Importance Decision Tree"*

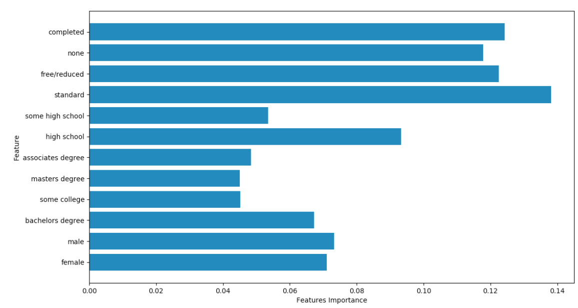

***Figure 7:** "Features Importance Random Forest"*

Conclusion/Improvements

When I start this project I get fewer values of accuracy so I decide to change to change my labels to only two options and do a binary tree and after testing the model tree we can see that it results increased in a value very close to 70% of accuracy.

I also drop a column of my data set because I think that column does not have much relevance for this test, maybe dropping some other column can help to upgrade the accuracy, but that depends on the study being done or what you are looking for.

Maybe applying the random forest method could gave us better results, I will try it later.

Performing this project was a great challenge for me, since it is the first time I work with python, but at the end I got the results that I was looking for.