Statistical Learning- Luca Pennella Project

Summary

1.1 Data description 1
1.2 Modules used and Data Import 2
1.3 Data Cleaning 3
1.4 Data Exploration 3
1.5 Encoding Categorical Features 7
1.6 SMOTE for Imbalanced Classification 8
1.7 Split in Train and Test Set 8
1.8 Pruning in Decision Trees 8
1.9 Viewing code and its output 11

1.1 Data description

This data was extracted from the 1994 Census bureau database by Ronny Kohavi and Barry Becker (Data Mining and Visualization , Silicon Graphics). The prediction task is to determine whether a person makes over $ 50K a year .
The dataset is made up of 48,842 rows and 15 columns, the details of the columns are shown below:

Categorical Attributes

- **workclass :** Private, Self- emp - not-inc , Self- emp - inc , Federal- gov , Local- gov , State- gov , Without-pay , Never-worked .
- **education :** Bachelors, Some-college, 11th, HS- grad , Prof-school, Assoc-acdm , Assoc-voc , 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate , 5th-6th, Preschool.
- **marital -status:** Married -civ- spouse , Divorced , Never-married , Separated , Widowed , Married-spouse-absent , Married -AF- spouse .
- **occupation :** Tech-support, Craft-repair , Other -service, Sales, Exec-managerial , Prof- specialty , Handlers-cleaners , Machine-op- inspct , Adm-clerical , Farming-fishing, Transport-moving , Priv-house- serv , Protective-serv , Armed-Forces .
- **relationship :** Wife , Own-child , Husband , Not-in-family, Other -relative, Unmarried .
- **race:** White, Asian-Pac- Islander , Amer - Indian -Eskimo, Other , Black.
- **sex:** Female , Male.
- **native-country:** United-States, Cambodia, England , Puerto-Rico, Canada, Germany, Outlying -US (Guam-USVI- etc), India, Japan, Greece , South, China, Cuba, Iran, Honduras, Philippines, Italy , Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland , France, Dominican -Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary , Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia , El-Salvador, Trinadad & Tobago , Peru, Hong , Holand -Netherlands.

Continuos Attributes

- **age:** Age of an individual .
- **fnlwgt :** The weights on the CPS files are controlled to independent estimates of the civilian noninstitutional population of the US.

- **capital-gain:** continuous .
- **capital- loss :** continuous .
- **hours-per-week** : Individual's working hour per week.

1.2 Modules used and Data Import

The modules used are shown in the following image. The characters followed by the # symbol show a first brief comment on the modules used. Among the non-commented modules we find *numpy* for performing calculations and managing matrices, *seaborn* for the creation and visualization of the graphs and *imblearn* to balance the dataset.
The use of specific modules allows you to work more easily on problems related to data analysis and allows you to use Machine Learning algorithms

```
import pandas as pd #to load and manipulate data
import numpy as np
import matplotlib.pyplot as plt #to draw graphs
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier #to build a classificarion Tree
from sklearn.tree import plot_tree #to draw a classification Tree
from sklearn.model_selection import train_test_split #to split data into training and testing sets
from sklearn.model_selection import cross_val_score # for cross-validation
from sklearn.metrics import confusion_matrix #to create a confusion matrix
from sklearn.metrics import plot_confusion_matrix  #to draw a confusion matrix
from sklearn.metrics import accuracy_score,classification_report #to plot score information
from imblearn.over_sampling import SMOTE
```

The data was imported through the pd command. read_csv and saved in the variable *df* , once the data was imported, I checked by displaying the first ten lines using the df command. head (10):

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K
4	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Female	0	0	30	United-States	<=50K
5	34	Private	198693	10th	6	Never-married	Other-service	Not-in-family	White	Male	0	0	30	United-States	<=50K
6	29	?	227026	HS-grad	9	Never-married	?	Unmarried	Black	Male	0	0	40	United-States	<=50K
7	63	Self-emp-not-inc	104626	Prof-school	15	Married-civ-spouse	Prof-specialty	Husband	White	Male	3103	0	32	United-States	>50K
8	24	Private	369667	Some-college	10	Never-married	Other-service	Unmarried	White	Female	0	0	40	United-States	<=50K
9	55	Private	104996	7th-8th	4	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	10	United-States	<=50K

As can be seen from the upper image, the dataset has missed values indicated with the character *?* , we must then proceed to clean up the dataset.

1.3 Data Cleaning

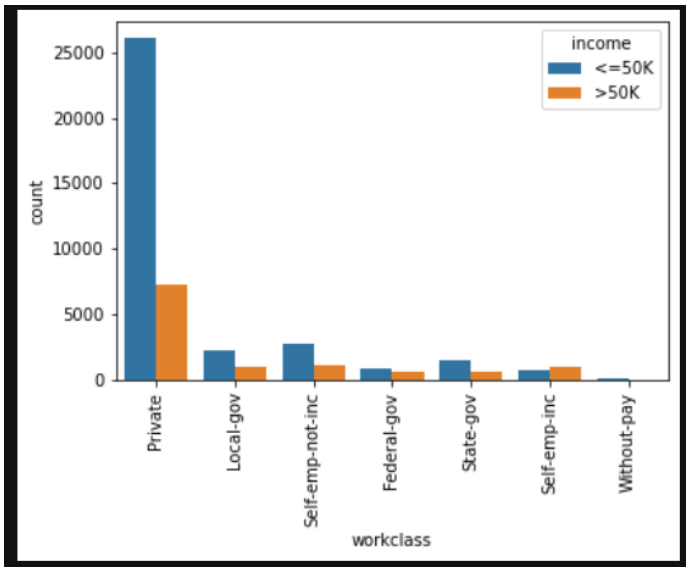
The second step was to check the missed columns values through the df [' workclass '] command. unique () where workclass is a specific column and unique is a command to display unique values within the column. Through the command len (df. Loc [df [' workclass '] == '?']) It is possible to have the missed count values .

The last step is the definition of a new daset without the missing values since they concern just under 6% of the data. To carry out this action I used the script shown below. Using the command *loc* and *! =* it was possible to select only the lines that do not have missed values . In this case I have saved the new daset in the variable *df_no_missing* which now has 45,222 lines.

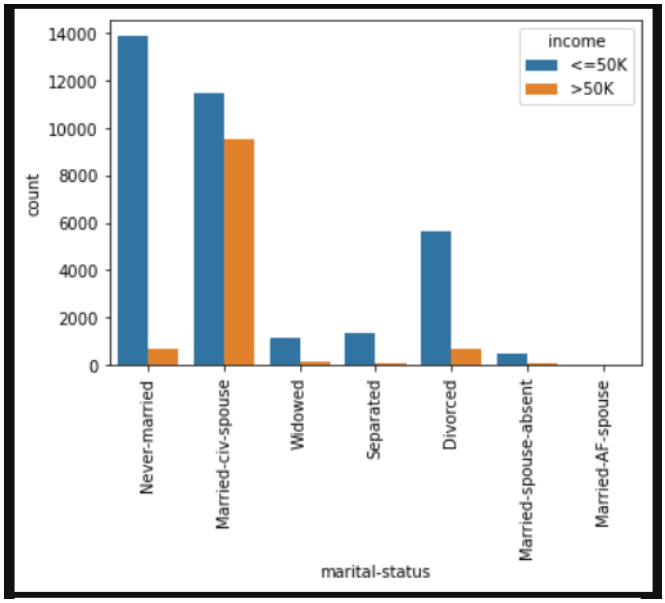
```
df_no_missing = df.loc[(df['workclass'] != '?')
                       &
                       (df['education'] != '?')
                       &
                       (df['marital-status'] != '?')
                       &
                       (df['occupation'] != '?')
                       &
                       (df['relationship'] != '?')
                       &
                       (df['race'] != '?')
                       &
                       (df['gender'] != '?')
                       &
                       (df['native-country'] != '?')
                      ]
```

1.4 Data Exploration

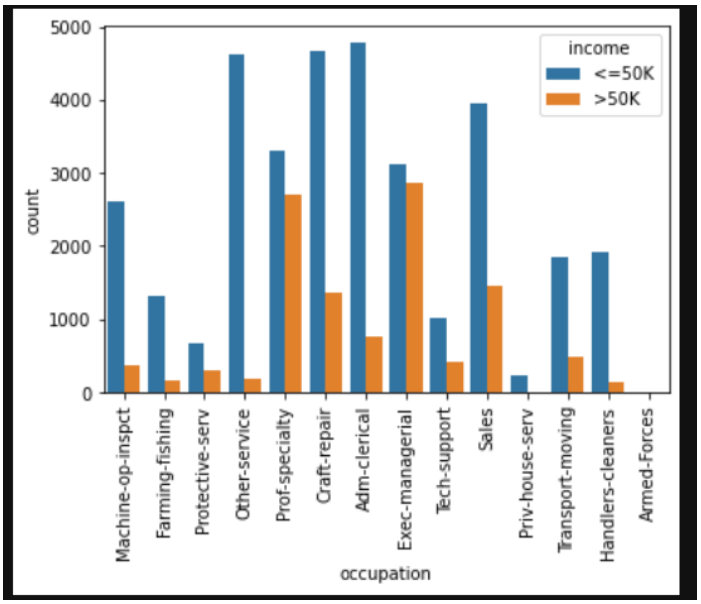
The data exploration phase allows you to have a first view of the distribution of the population within the variables and an initial evaluation of the relationship between the target variable and the features . Through two for loops , in order to keep the code clean and synthetic, I visualized the categorical and numeric variables separately.



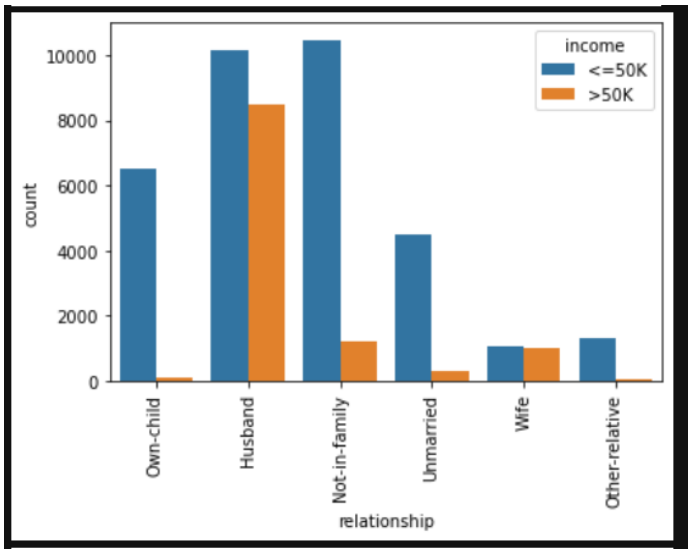
- In private workclass most of the people earn <= 50k.
- self- emp - inc workclass is only where more people earn > 50k.
- In Federal- gov workclass nearly more than half of the people earn > 50k.



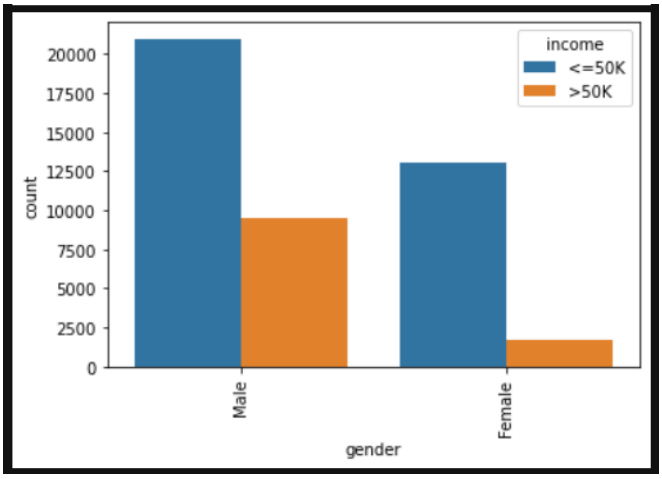
- Married -civ- spouse has the highest percentage of falling under the income group> 50k.
- Despite the fact that we have major observation in the marital -status attribute but only 1.5% of the people of " Never-married " earn more than 50k.



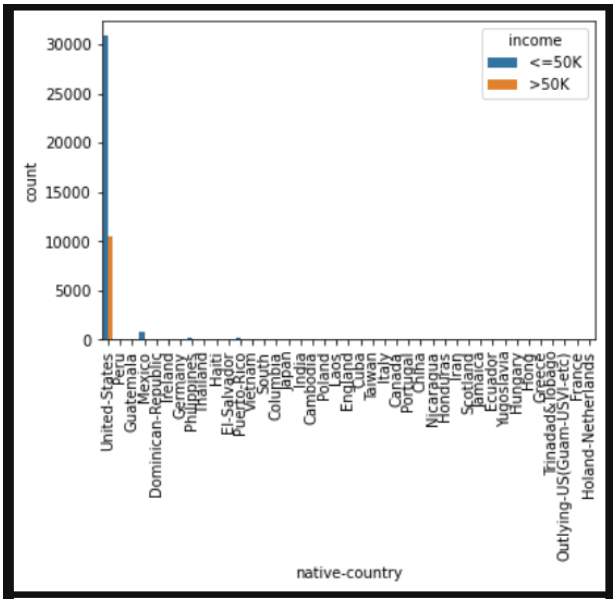
- In every occupation , people who earn less than 50k is greater than people who earn > 50k.
- Prof- specialty and Exec-managerial have maximum percentage that fall in income group 1.



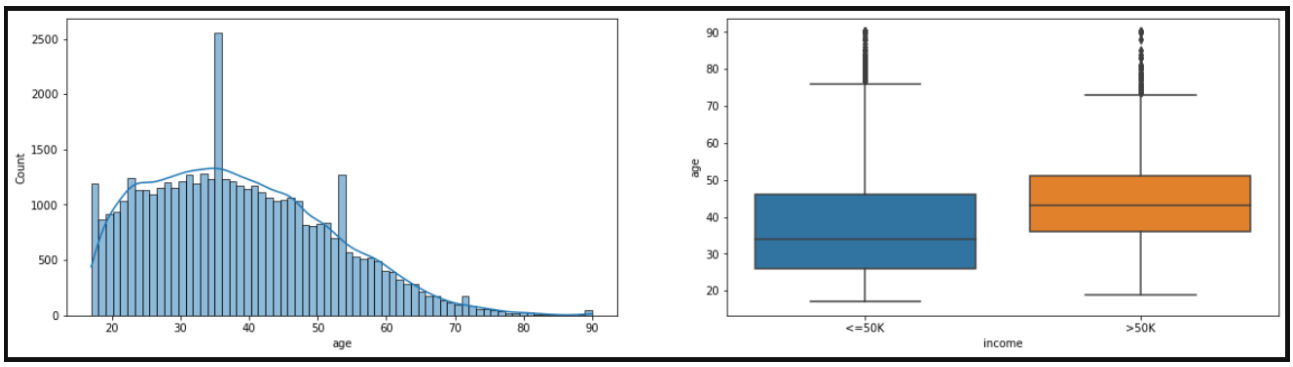
- husbands has the highest percentage of earning more than 50k in all the other categories .
- One thing to notice is that " not -in-family" has highest percentage to earn less than 50k.



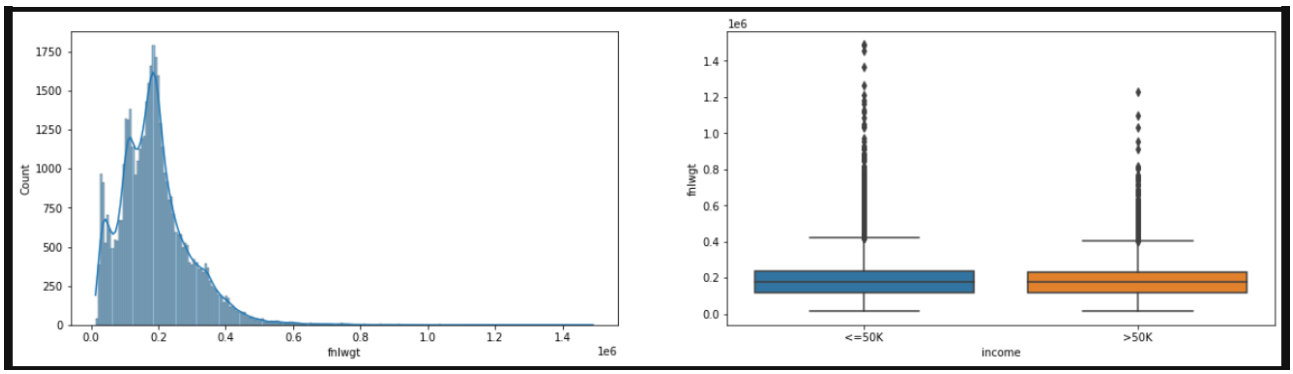
- For " female " earning more than 50k is rare with only 3.62% of all observations .



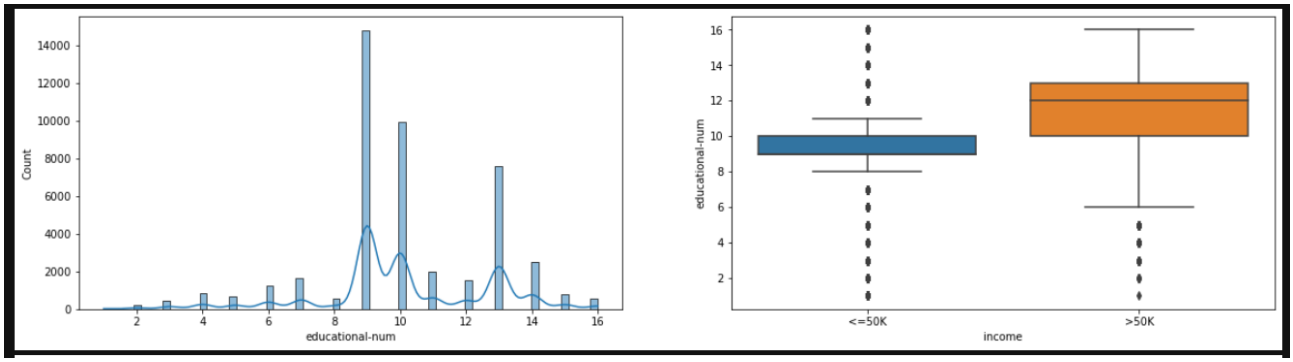
- Over 90% of the population is native to America.



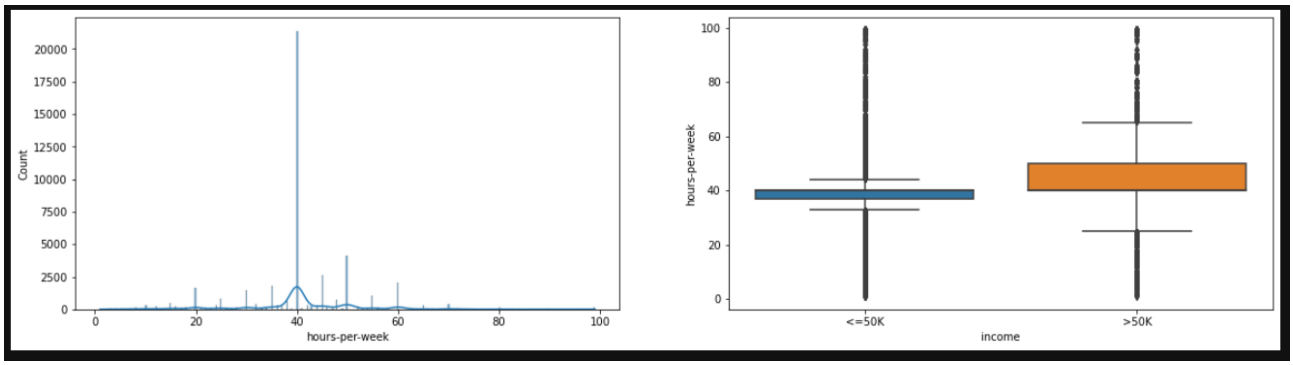
- *age* attribute is not symmetric .
- Minimum and Maximum age of the people is 17 and 90 respectively .
- Looking at the boxplots , it is possible to see that in proportion as age increases , earnings increase .



- The distribution of finalweight seems to be rightly skewed since mean (189,664) is greater than median (178.144).



- A higher level of education provides higher income on average



- Most people work 30-40 hours per week, they are roughly 27,000 people.
- There are also few people who works 80-100 hours per week and some less than 20 which is unusual .

Once you have explored the data to get an idea of the distributions of the variables, the next step is to prepare the data in a format suitable for the algorithm you plan to use. In our case we will use a Decision Tree Classifier via the Scikit framework Learn which requires only numeric data as input, for this it is necessary to proceed with the encoding of the categorical features.
One-hot encoding is a method of identifying whether a unique categorical value from a categorical feature is present or not . For example, in the case of Gender, two distinct columns gender_male and gender_female will be created which take the values 1 or 0 based on the gender of the individual.

1.5 Encoding Categorical Features

Encoding is very simple in Python, in fact only one command is needed in pandas , pd. get_ dummies ( ) within which to specify the dataset and columns to modify. The output dataset will have a new column for each category within the variable. Below is the image of a part of the table with the one -hot-encoding, it is

possible to observe how the number of columns has grown.

	age	fnlwgt	educational-num	capital-gain	capital-loss	hours-per-week	workclass_Federal-gov	workclass_Local-gov	workclass_Private	workclass_Self-emp-inc	...	native-country_Portugal
0	25	226802	7	0	0	40	0	0	1	0	...	0
1	38	89814	9	0	0	50	0	0	1	0	...	0
2	28	336951	12	0	0	40	0	1	0	0	...	0
3	44	160323	10	7688	0	40	0	0	1	0	...	0
5	34	198693	6	0	0	30	0	0	1	0	...	0

5 rows × 88 columns

In this case the encoding was carried out for convenience also on the target variable *y* in such a way as to have in a single column with 1 the individuals with an income higher than 50k instead with 0 the individuals with lower income.

1.6 SMOTE for Imbalanced Classification

In our Dataset, individuals with an income greater than 50k are about 25%, this imbalance causes a worse classification for this category also following the split in train in set, in these cases we speak of an unbalanced problem.
A problem with imbalanced classification is that there are too few examples of the minority class for a model to effectively learn the decision boundary .
SMOTE works by selecting examples that are close in the feature space , drawing a line between the examples in the feature space and drawing a new sample at a point along that line.
To use the SMOTE technique you need the imblearn module. over_sampling and through the following line of code *: X_balanced , y_balanced = smote. fit_resample (* X_encoded *, y_final)* it is possible to balance the set so that we have the same number of individuals for both income classes.

1.7 Split in Train and Test Set

Here we will discuss how to split a dataset into Train and Test sets in Python. The train -test split is used to estimate the performance of machine learning algorithms that are applicable for prediction-based Algorithms / Applications. This method is a fast and easy procedure to perform such that we can compare our own machine learning model results to machine results . By default, the Test set is split into 30% of actual data and the training set is split into 70% of the actual data.
We need to split a dataset into train and test sets to evaluate how well our machine learning model performs . The train set is used to fit the model, and the statistics of the train set are known . The second set is called the test data set, this set is solely used for predictions . The scikit-learn library provides us with the model_selection module in which we have the splitter function train_test_ split ( ):

```
# split the data into training e testing sets
X_train, X_test, y_train, y_test = train_test_split(X_balanced, y_balanced, random_state = 42)
```

1.8 Pruning in Decision Trees

Pruning is one of the techniques that is used to overcome our problem of Overfitting . Pruning , in its literal sense , is a practice which the selective _ removal of certain parts of a tree (or plant), such as branches , buds , or roots, to improve the tree's structure , and promote healthy growth . This is exactly what Pruning does to our Decision Trees as well . It makes it versatile so that it can adapt if we feed any new kind of data to it , thereby fixing the problem of overfitting .
It reduces the size of a Decision Tree which might slightly increase your training error but drastically decrease your testing error , hence making it more adaptable .

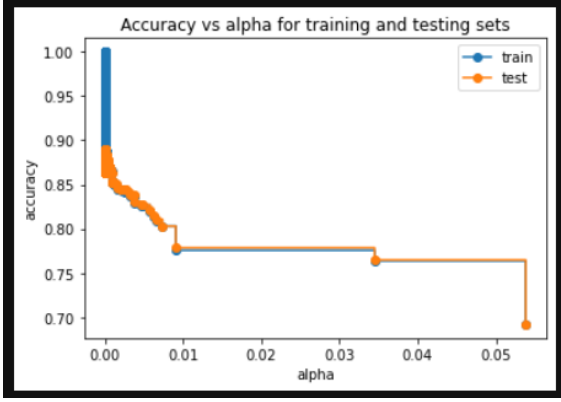
Minimal Cost- Complexity Pruning is one of the types of Pruning of Decision Trees .
This algorithm is parameterized by α (≥0) known as the complexity parameter.
The complexity parameter is used to define the cost- complexity measure , Rα (T) of a given tree T: Rα (T) = R (T) + α | T | where | T | is the number of terminal nodes in T and R (T) is traditionally defined as the total misclassification rate of the terminal nodes .
To apply this algorithm we use *cost_complexity_pruning_path* which provides the possible values of α (in our case 2,335 values), once the values of α have been obtained, one proceeds through a for loop to identify the value of α which provides the best result on the test set. The script on Python with its output is the following:

```
for ccp_alpha in ccp_alphas:
    clf_dt = DecisionTreeClassifier(random_state = 42, ccp_alpha=ccp_alpha)
    clf_dt.fit(X_train, y_train)
    clf_dts.append(clf_dt)
```

```
train_scores = [clf_dt.score(X_train, y_train) for clf_dt in clf_dts]
test_scores = [clf_dt.score(X_test, y_test) for clf_dt in clf_dts]

fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker = 'o', label ="train", drawstyle = "steps-post")
ax.plot(ccp_alphas, test_scores, marker = 'o', label ="test", drawstyle = "steps-post")
ax.legend()
plt.show
```

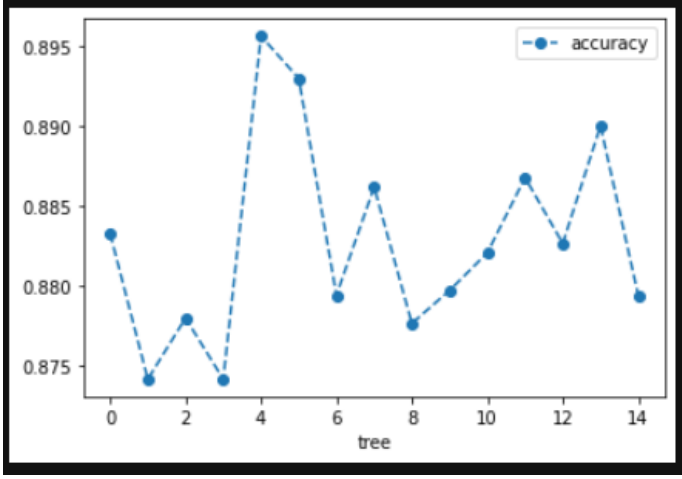
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



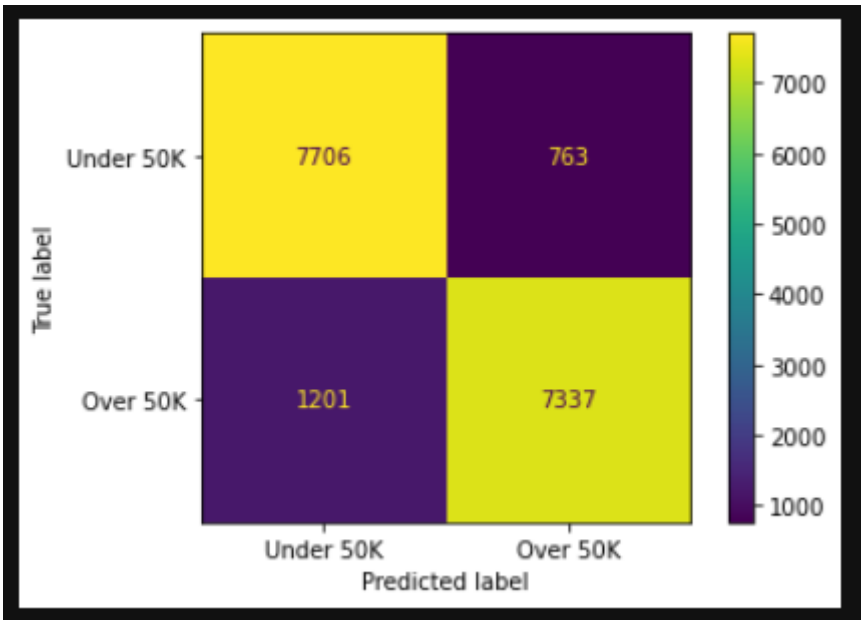
In this case it is possible to observe how the best values of α are between 0.0001 and 0.001. In my case I opted for a value of α equal to 0.001 but to be sure that it is a good value, in most cases, I use the Cross Validation technique , in this case with 15 fold , it is possible to observe the code with the relative result in the following image:

```
clf_dt = DecisionTreeClassifier(random_state = 42, ccp_alpha = 0.0001)
scores = cross_val_score(clf_dt, X_train, y_train, cv = 15)
df = pd.DataFrame(data= {'tree':range(15), 'accuracy':scores})
df.plot(x='tree', y='accuracy', marker = 'o', linestyle = '--')
```

```
<AxesSubplot:xlabel='tree'>
```



The value of α seems to perform well in all cases tested with an accuracy ranging from 87% up to 89.5%. At this point, having identified an excellent value of α, we can proceed to visualize the confusions matrix that allows you to visualize the results obtained graphically:



Beyond the confusion matrix that provides a first visualization of the results, it may be useful to calculate some precision indices, including accuracy , precision and others. Here is the code and the output obtained:

```
print(f'train accuracy: {accuracy_score(y_train,clf_dt_pruned.predict(X_train))}')
print(f'test accuracy:  {accuracy_score(y_test,clf_dt_pruned.predict(X_test))}')
print(classification_report(y_test,clf_dt_pruned.predict(X_test)))
```

```
train accuracy: 0.8924560475098489
test accuracy:  0.8845181395895808
              precision    recall  f1-score   support

           0       0.87      0.91      0.89      8469
           1       0.91      0.86      0.88      8538

    accuracy                           0.88     17007
   macro avg       0.89      0.88      0.88     17007
weighted avg       0.89      0.88      0.88     17007
```

A further useful information is the identification of the most important features from a classification point of view. To identify them I used *feature_importances* _ which allows the extraction and sorting of the features by importance.
This step can also be useful for eliminating the variables that are not important for the calculation of the Decision Tree allowing a possible clearer view of the tree.
In our case the 15 most important variables were:

```
marital-status_Never-married    0.314278
marital-status_Divorced         0.154663
educational-num                 0.125650
capital-gain                    0.073642
hours-per-week                  0.041065
age                             0.037019
marital-status_Separated        0.021144
workclass_Self-emp-not-inc      0.019156
occupation_Craft-repair         0.017905
gender_Male                     0.017760
marital-status_Widowed          0.017495
gender_Female                   0.013472
occupation_Other-service        0.013408
capital-loss                    0.013330
workclass_Private               0.012771
```

1. 9 Viewing code and its output

Statistical Learning_Luca Penn