TD : Crédit Scoring

Elaboré par : CHAFIKI Fatima elzahra

In [117…

```
import pandas as pd
from numpy import  *
```

ÉNONCÉ DU PROBLÈME

Un prêt sur valeur domiciliaire ou Home Equity Loan est un prêt à terme garanti qui permet d'emprunter sur la valeur acquise de la maison . donc notre problématique est rectifier et prédire que le demandeur qui demande les prêts sur valeur domiciliaire paiera le prêt ou sera un délinquant sans payer le prêt .

L'analyse du dataset

Notre data set contient des observations pour 5 960 demandeurs d'hypothèque. La variable nommée BAD est la variable dépendante qui indique si un demandeurs de prêt a payé le prêt ou s'il est en défaut càd ne l'a pas payé .

In [118…

```
ds = pd.read_csv("./td7/CreditScoring.csv")
ds
```

Out[118…

	BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ	CLAGE	NINQ	CLNO	DEBTINC
0	1	1100	25860.0	39025.0	HomeImp	Other	10.5	0.0	0.0	94.366667	1.0	9.0	NaN
1	1	1300	70053.0	68400.0	HomeImp	Other	7.0	0.0	2.0	121.833333	0.0	14.0	NaN
2	1	1500	13500.0	16700.0	HomeImp	Other	4.0	0.0	0.0	149.466667	1.0	10.0	NaN
3	1	1500	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	0	1700	97800.0	112000.0	HomeImp	Office	3.0	0.0	0.0	93.333333	0.0	14.0	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
5955	0	88900	57264.0	90185.0	DebtCon	Other	16.0	0.0	0.0	221.808718	0.0	16.0	36.112347
5956	0	89000	54576.0	92937.0	DebtCon	Other	16.0	0.0	0.0	208.692070	0.0	15.0	35.859971
5957	0	89200	54045.0	92924.0	DebtCon	Other	15.0	0.0	0.0	212.279697	0.0	15.0	35.556590
5958	0	89800	50370.0	91861.0	DebtCon	Other	14.0	0.0	0.0	213.892709	0.0	16.0	34.340882
5959	0	89900	48811.0	88934.0	DebtCon	Other	15.0	0.0	0.0	219.601002	0.0	16.0	34.571519

5960 rows × 13 columns

On a 13 variables dans le dataset : BAD : 0 si le client a rembourse sa dette et 1 sinon CLAG : l'age de credit le plus ancien par mois CLNO : le nombre de credits DEBTINC : taux de credit à revenu DELINQ : nombre de credits non rembourses DEROG : nombre detats derogatoires principaux JOB : categorie professionnelle du client LOAN : montant du credit YOJ : Anciennete du travail le plus recent VALUE : la valeur de la propreite MORTDUE : montant du sur lhypothe`que existante NINQ : nombre denquetes recentes de degre de solvabilite REASON : Debtcon 1 HomeImp 2 1. credit de consolidation2. credit immobilier

Traitement du data set

le problème de ce dataset est que les lignes contiennent plusieurs données manquantes . Les données manquantes constituent un problème majeur, puisque l'information à disposition est incomplète et donc moins fiable

In [119…

```
df = ds[ds['VALUE'].isnull()]
df = df[df['JOB'].isnull()]
df = df[df['REASON'].isnull()]
df = df[df['DEROG'].isnull()]
df = df[df['DELINQ'].isnull()]
df = df.reset_index()
df = df['index']
df
```

Out[119…

```
0       3
1      10
2      17
3    1405
4    1507
5    1897
6    1959
7    1960
8    2303
9    2449
Name: index, dtype: int64
```

In [120…

```
ds = ds.drop(df)
ds = ds.reset_index()
ds = ds.drop(['index'], axis = 1)
```

In [121…

```
def Entropie(S):
    S = array(S)
    classes_distinctes = set(S[:,0])
#     print(classes_distinctes)
    classes = list(S[:,0])
#     print(classes)
    s= 0
    for c in classes_distinctes :
        p = classes.count(c)/len(classes)
        s+= p*log2(p)
    return -1.0*s



def GINI(S):
    S = np.array(S)
    clonne_des_classes = list(S[:, 0])
    classes = list(set(S[:, 0]))
    k = len(classes)
    s = 0
    for i in range(k):
        pi = clonne_des_classes.count(classes[i]/len(S[:, 0]))
        s += pi**2
    return 1-s


def Mesure_desorde(S, numero_colon_attribut):
    S = S.to_numpy()
    Bonne_mesure_desorde = + inf
    Bonne_valeur_de_repartition = 0
    classes = list(set(S[:, 0]))
    valeurs_attribut = list(set(S[:, numero_colon_attribut]))
    valeurs_attribut.sort()
    Gauche = []
    Droite = []

    for i in range(len(valeurs_attribut)-1):
        v = valeurs_attribut[i]
        Gauche.clear()
        Droite.clear()
        for e in S:
            val_attribut_pour_e = e[numero_colon_attribut]
            if val_attribut_pour_e <= v:
                Gauche.append(e)
            else:
                Droite.append(e)
        pGau = len(Gauche)/len(S)
        pDr = len(Droite)/len(S)
        mesure = (pGau)*GINI(Gauche)+(pDr)*GINI(Droite)
        if mesure < Bonne_mesure_desordre:
            Bonne_mesure_desordre = mesure
            Bonne_valeur_de_repartition = v
    return(Bonne_mesure_desordre, Bonne_valeur_de_repartition)


def meilleur_attribut_meilleure_valeur(S):
    meilleur_attribut = 0
    meilleure_val = 0
    meilleure_mesure = + inf
    S = S.to_numpy()
    for num_attribut in range(len(S[0])-1):
        mesure, valeur = Mesure_desordre(S, num_attribut)
        if mesure < meilleure_mesure:
            meilleure_val = valeur
            meilleure_mesure = mesure
            meilleur_attribut = num_attribut
    return meilleur_attribut, meilleure_val
```

In [122…

```
def Gain_d_information(ds, numero_colon_attribut):

    S = ds.to_numpy()
    classes = list(set(S[:, 0]))
    valeurs_attribut = list(set(S[:, numero_colon_attribut]))
    Si = [[]for i in range(len(valeurs_attribut))]
    for e in S:
        e = list(e)
        val_attribut_pour_e = e[numero_colon_attribut]
        numero_sous_ensemble = valeurs_attribut.index(val_attribut_pour_e)
        Si[numero_sous_ensemble].append(e)
    Si = array(Si)
    som = 0
    for sous_ensemble in Si:
        som += (len(sous_ensemble)/len(S))*Entropie(sous_ensemble)
    return(Entropie(S)-som , ds.columns[numero_colon_attribut])
```

Calcul des gain d'information pour chaque attribut

In [123…

```
Gain_Information= [ Gain_d_information(ds,i) for i in range(1,13)]
Gain_Information
```

```
<ipython-input-122-da8ebe828683>:12: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequence
s (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If
you meant to do this, you must specify 'dtype=object' when creating the ndarray.
  Si = array(Si)
```

Out[123…

```
[(0.13247031605795323, 'LOAN'),
 (0.6599670262166042, 'MORTDUE'),
 (0.6490620261099311, 'VALUE'),
 (0.0010200707812508814, 'REASON'),
 (0.014172286113719235, 'JOB'),
 (0.09890515375629483, 'YOJ'),
 (0.11308343379194186, 'DEROG'),
 (0.12913853183220558, 'DELINQ'),
 (0.6689316114488737, 'CLAGE'),
 (0.07307049238021024, 'NINQ'),
 (0.05943267670920016, 'CLNO'),
 (0.7205815177959121, 'DEBTINC')]
```

D'aprés les résultat précedente , On constate que 4 variable ont un gain d'information élevé proche de 70% lesquels : MORTDUE,VALUE ,CLAGE ,DEBTINC et les autres variables ont un gain d'information bas Donc la variable dépendente BAD est affecté par MORTDUE,VALUE ,CLAGE ,DEBTINC .

In [124…

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
```

Traitement des données

Avant de passer le dataset à le modèle DecisionTreeClassifier , On doit convertir les colonnes de type string en format numérique comme REASON , JOB et remplacer les valeurs manquantes par la moyenne .

In [125…

```
T = LabelEncoder()

T.fit_transform(ds["REASON"])
replaced_reason = T.transform(ds["REASON"])
ds["REASON"] = replaced_reason

T.fit_transform(ds["JOB"])
replaced_job = T.transform(ds["JOB"])
ds["JOB"] = replaced_job

for i in ds.columns :
    ds[i] = ds[i].fillna(ds[i].mean())

ds
```

Out[125…

	BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ	CLAGE	NINQ	CLNO	DEBTINC
0	1	1100	25860.0	39025.0	1	2	10.5	0.0	0.0	94.366667	1.0	9.0	33.796792
1	1	1300	70053.0	68400.0	1	2	7.0	0.0	2.0	121.833333	0.0	14.0	33.796792
2	1	1500	13500.0	16700.0	1	2	4.0	0.0	0.0	149.466667	1.0	10.0	33.796792
3	0	1700	97800.0	112000.0	1	1	3.0	0.0	0.0	93.333333	0.0	14.0	33.796792
4	1	1700	30548.0	40320.0	1	2	9.0	0.0	0.0	101.466002	1.0	8.0	37.113614
...	...	...	...	...	...	...	...	...	...	...	...	...	...
5945	0	88900	57264.0	90185.0	0	2	16.0	0.0	0.0	221.808718	0.0	16.0	36.112347
5946	0	89000	54576.0	92937.0	0	2	16.0	0.0	0.0	208.692070	0.0	15.0	35.859971
5947	0	89200	54045.0	92924.0	0	2	15.0	0.0	0.0	212.279697	0.0	15.0	35.556590
5948	0	89800	50370.0	91861.0	0	2	14.0	0.0	0.0	213.892709	0.0	16.0	34.340882
5949	0	89900	48811.0	88934.0	0	2	15.0	0.0	0.0	219.601002	0.0	16.0	34.571519

5950 rows × 13 columns

In [127…

```
X = ds.iloc[:,1:13]
Y = ds.iloc[:,0]

x_train,x_test,y_train,y_test = train_test_split(X, Y, test_size=0.33, random_state=42)
clf_model = DecisionTreeClassifier(criterion="gini", random_state=42)
clf_model.fit(x_train,y_train)

clf_model.predict(x_test) , y_test
```

Out[127…

```
(array([0, 0, 0, ..., 1, 0, 0], dtype=int64),
 4038    0
 4246    0
 1966    0
 4763    1
 1374    0
        ..
 1907    1
 3692    1
 5924    1
 368     0
 1937    0
 Name: BAD, Length: 1964, dtype: int64)
```

In [ ]:

In [ ]: