Nguyễn Văn Nam

20162844

**Machine Learning & Data Mining – Homework 1**

**I – Problem Formulation**

Looking at the final (y) column in the data set, we can see that each observation belongs to one of 6 classes named: 1, 2, 3, 4, 5 or 6. Moreover, the class labels do not provide much meaning about themselves or label interactions.

After examining further, we can clearly see that this is a classification problem, that is where we will build and train a machine learning model to predict which category or class each observation belongs to.

**II – Dealing With Missing Values**

Since we found that this should be a classification problem, one of the best way to deal with missing values is to use the class-specific mean, median or popular value, particularly mean value in our solution of an attribute.

#Replace invalid values with NaN

df[[0,1,2,3,4,5,6,7]] = df[[0,1,2,3,4,5,6,7]].replace('?', numpy.NaN)

#Replace ? in this particular value with 5

df[[0]] = df[[0]].replace('1.?277005796', 1.5277005796)

#Ensure all values are numeric

df = df.apply(pd.to\_numeric, *errors*='coerce')

#Replace all NaN values in every column with mean of a particular group to which a row belongs

#In other words, use the class-specific mean value of an attribute

df[0] = df[0].fillna(df.groupby(8)[0].transform('mean'))

df[1] = df[1].fillna(df.groupby(8)[1].transform('mean'))

df[2] = df[2].fillna(df.groupby(8)[2].transform('mean'))

df[3] = df[3].fillna(df.groupby(8)[3].transform('mean'))

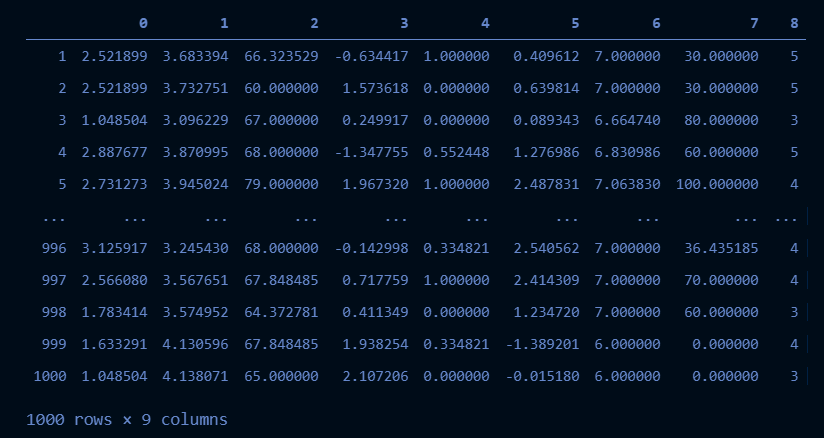
df[4] = df[4].fillna(df.groupby(8)[4].transform('mean'))

df[5] = df[5].fillna(df.groupby(8)[5].transform('mean'))

df[6] = df[6].fillna(df.groupby(8)[6].transform('mean'))

df[7] = df[7].fillna(df.groupby(8)[7].transform('mean'))

The result is that all missing values of each observation are replaced with mean value of a specific class that the observation belongs to.



**III – Model Selection & Assessment**

Since this is a classification problem, we need to select an appropriate model. A very popular classification technique where several conditions are put on the dataset is decision tree. That is the model that we will try to apply on our dataset.

First, we need to split the dataset into test set and train set with ratio test\_size/total = 0.2:

#Split data set into test set and train set

x = df.iloc[:,:-1].values

y = df.iloc[:,-1].values

from sklearn.model\_selection import train\_test\_split

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,*test\_size*=0.2)

Then, we simply use scikit learn library to train our dataset with decision tree:

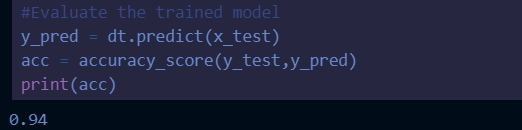
#Use decision tree to train a model

from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()

dt.fit(x\_train,y\_train)

Making predictions and assessing the model are also no-brainer tasks using scikit’s built-in functions predict() and accuracy\_score():



So our trained model has an accuracy score of 94%.

**IV – Making Predictions For 10 Testing Examples**

Using scikit’s built-in function predict(), we can easily make predictions for all examples in our testing file:

