PROJECT 1 - DATA SCIENCE

Authors :

Ariel Epshtein, ID: 316509504

Maor Mohav, ID: 316142363

Omer Ben David, ID: 316344449

Project structure:

---

1. The project contains a dictionary that will save the values that will run the model using functions we have written, (in PYCHARM these values will be received as input from the user in the GUI, and in JUPYTER, the values are default, values we have chosen and can be changed)
2. In each experiment, initial processing steps will be performed.
3. The project contains models and algorithms that use built-in libraries, but also models and algorithms with our implementation (functions with the extension at the end of the signature: _Implementation)
4. Before running the project file in PYCHARM, the user will need to read the Readme.text file
5. We add documentation for all the functions in the Pycharm project.

Libraries:

In [50]:
```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from scipy.stats import zscore
from sklearn.preprocessing import LabelEncoder
from pyitlib import discrete_random_variable as drv
from math import log
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
import csv
import math
import random
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.cluster import KMeans
import pickle
from pandas_profiling import ProfileReport
```

Uploading data

In [78]:
```python
dic = {'train_path': 'D:\\onedrive\\OneDrive - ac.sce.ac.il\\Desktop\\train.csv', 'test_path': 'D:\\onedrive\\OneDrive - ac.sce.ac.il\\Desktop\\test.csv', 'missing_data': 'By all data','normalization': 'Yes', 'discritization': 'D

train_file = pd.read_csv(dic['train_path'])
test_file = pd.read_csv(dic['test_path'])
```

EDA:

In [79]:
```python
profile = ProfileReport(train_file, title="Pandas Profiling Report")
```

In [80]:
```python
profile
```

Pandas Profiling Report | Overview | Variables | Interactions | Correlations | Missing values | Sample

Overview

Overview | Warnings 11 | Reproduction

Dataset statistics

Number of variables	16
Number of observations	42180
Missing cells	29
Missing cells (%)	< 0.1%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	5.1 MiB
Average record size in memory	128.0 B

Variable types

Numeric	6
Categorical	6
Boolean	4

Variables

age
Real number ($\mathbb{R}_{\geq 0}$)

Distinct	76	**Minimum**	18
Distinct (%)	0.2%	**Maximum**	500

Out[80]:

Our analysis:

---

1. Our data has 16 columns : 6 - numeric columns ,6 - categorical columns , 4- boolean columns.

1. Our data has 29 missing values.

2. Columns corrolations:
   contact is highly correlated with month
   education is highly correlated with job
   job is highly correlated with education
   month is highly correlated with contact and 2 other fields
   day is highly correlated with month
   month is highly correlated with housing
   housing is highly correlated with month

Delete classification missing data

In [81]:
```python
def Delete_Rows(data):
    data['class'].replace('', np.nan, inplace=True)
    data.dropna(subset=['class'], inplace=True)
```

In [82]:
```python
Delete_Rows(train_file)
Delete_Rows(test_file)
```

Complete Missing Data

In [83]:
```python
def numeric(j):
    for i in j:
        if type(i) == str and i != '':
            return False

    return True
```

In [84]:
```python
def D_Array(data):
    a = []
    no = data[data['class'] == 'no'].dropna(axis = 0)
    yes = data[data['class'] == 'yes'].dropna(axis = 0)

    for i in data.columns:
        if numeric(data[i]):
            a.append([yes[i].mean(),no[i].mean()])
        else:
            a.append([yes[i].value_counts().index[0],no[i].value_counts().index[0]])

    return a
```

In [85]:
```python
def Missing_Data(data):
    a = D_Array(train_file)
    c = data.columns.tolist()


    if dic['missing_data'] == 'By all data':
        for i in data.columns:
            if numeric(data[i]) == True:
                data[i].fillna(train_file[i].mean(), inplace = True)
            else:
                data[i].fillna(train_file[i].value_counts().index[0], inplace = True)

    else:
        for i,rows in data.iterrows():
            for b in data.columns:
                if pd.isnull(data[b][i]) and data['class'][i] == 'yes':
                    data.at[i,str(b)] = a[c.index(b)][0]
                elif pd.isnull(data[b][i]) and data['class'][i] == 'no':
                    data.at[i,str(b)] = a[c.index(b)][1]
```

In [86]:
```python
Missing_Data(train_file)
Missing_Data(test_file)
```

Split the data

In [87]:
```python
for i in train_file.columns:
    if numeric(train_file[i]):
        train_file[i] = pd.to_numeric(train_file[i], downcast = 'float')
        test_file[i] = pd.to_numeric(test_file[i], downcast = 'float')
```

Normalization

In [88]:
```python
def Normalization(train_file, test_file):

    if dic['normalization'] != 'No':
        for i in train_file.columns:
            if numeric(train_file[i]):
                avg = train_file[i].mean()
                std = train_file[i].std()

                for j in range(0, len(train_file[i])):
                    train_file.at[j, str(i)] = ((train_file[i][j] - avg) / std)

                for j in range(0, len(test_file[i])):
                    test_file.at[j, str(i)] = ((test_file[i][j] - avg) / std)
```

In [89]:
```python
Normalization(train_file, test_file)
```

Discritization

In [90]:
```python
def Equal_width_binning(data, bins, label):
        for i in data:
            if numeric(data[i]):
                data[i] = pd.cut(data[i], bins, labels = label)
```

In [91]:
```python
def Equal_frequency_binning(data, bins):
        for i in data:
            if numeric(data[i]):
                data[i] = pd.qcut(data[i],bins, duplicates = 'drop')
                temp_categories = {value: j for j, value in enumerate(data[i].sort_values().unique(), start=1)}
                data[i] = data[i].cat.rename_categories(temp_categories)
```

In [92]:
```python
#Equal-frequency discretization
def Equal_frequency_binning_implementation():

    def Equal_frequency(data, test, columns , m):

        a = []

        for i in data[columns]:
            a.append(i)

        a.sort()



        length = len(a)
        n = int(length / m)
        arri=[]
        for i in range(0, m):
            arr = []
            for j in range(i * n, (i + 1) * n):
                if j >= length:
                    break
                arr = arr + [a[j]]
            arri.append(arr)

        for i in range(0, len(arri)):
            arri[i] = list(set(arri[i]))


        for i in range(0, len(test[columns])):
            for j in range(0, len (arri)):
                if test[columns][i] in arri[j]:
                    test.at[i,str(columns)] = j

        for i in range(0, len(data[columns])):
            for j in range(0, len (arri)):
                if data[columns][i] in arri[j]:
                    data.at[i,str(columns)] = j




    for i in train_file.columns:
        if numeric(train_file[i]):
            Equal_frequency(train_file, test_file, i , dic['number_of_bins'])


#Equal-width discretization

def Equal_width_binning_implementation():

    def Equal_width(data, test, columns, m):

        a = []

        for i in data[columns]:
            a.append(i)

        a.sort()
        w = int((max(a) - min(a)) / m)+1
        min1 = min(a)
        arr = []
        for i in range(0, m + 1):
            arr = arr + [min1 + w * i]
        arri=[]

        for i in range(0, m):
            temp = []

            for j in a:
                if j >= arr[i] and j <= arr[i+1]:
                    temp += [j]

            arri += [temp]

        for i in range(0, len(arri)):
            arri[i] = list(set(arri[i]))


        for i in range(0, len(test[columns])):
            for j in range(0, len (arri)):
                if test[columns][i] in arri[j]:
                    test.at[i,str(columns)] = j

        for i in range(0, len(data[columns])):
            for j in range(0, len (arri)):
                if data[columns][i] in arri[j]:
                    data.at[i,str(columns)] = j

    for i in train_file.columns:
        if numeric(train_file[i]):
            Equal_width(train_file, test_file, i , dic['number_of_bins'])
```

In [93]:
```python
def Discritization(data):
    bins = dic['number_of_bins']

    label = []

    for i in list(range(bins)):
        label.append(i)

    if dic['discritization'] == 'Equal Width Binning':
        Equal_width_binning(data, bins, label)

    elif dic['discritization'] == 'Equal Frequency Binning':
        Equal_frequency_binning(data, bins)

    elif dic['discritization'] == 'Equal Width Binning-Implementation':
        Equal_width_binning_implementation()

    elif dic['discritization'] == 'Equal Frequency Binning-Implementation':
        Equal_frequency_binning_implementation()


    elif dic['discritization'] == 'Discritization based Entropy':
        for i in data.columns:
            if numeric(data[i]):
                val = np.array(data[i])
                new_val = val.astype(int)
                IG = drv.entropy(new_val)
                for j in range(0, len(data[i])):
                    if data[i][j] < IG:
                        data.at[j,str(i)] = 0
                    else:
                        data.at[j,str(i)] = 1
```

In [94]:
```python
Discritization(train_file)
Discritization(test_file)
```

Encoder

In [95]:
```python
def Encoder(data, dat):
    for i in data.columns:
        if not numeric(data[i]):
            le = LabelEncoder()
            data[i] = le.fit_transform(data[i])
            dat[i] = le.fit_transform(dat[i])
```

In [96]:
```python
Encoder(train_file, test_file)
```

Saving files

In [97]:
```python
train_file.to_csv('train_file_clean.csv')
test_file.to_csv('test_file_clean.csv')
```

Model:

In [98]:
```python
def Naive_bayes():
    X_train = train_file.iloc[:, : -1]
    Y_train = train_file.iloc[:, -1]
    X_test = test_file.iloc[:, : -1]
    Y_test = test_file.iloc[:, -1]

    gnb = GaussianNB()
    y_pred = gnb.fit(X_train, Y_train).predict(X_test)
    print("Number of mislabeled points out of a total %d points : %d"% (X_test.shape[0], (Y_test != y_pred).sum()))
    accuracy = accuracy_score(Y_test, y_pred)*100
    print("precentence: ", accuracy)
    print("Confusion matrix: ")
    print(confusion_matrix(Y_test, y_pred))
    print()
    print("Report: ")
    print()
    print(classification_report(Y_test, y_pred))

    pickle.dump(gnb, open('Naive_bayes.sav', 'wb'))
```

In [99]:
```python
def Decision_Tree():
    X_train = train_file.iloc[:, : -1]
    Y_train = train_file.iloc[:, -1]
    X_test = test_file.iloc[:, : -1]
    Y_test = test_file.iloc[:, -1]

    tree = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
    tree.fit(X_train, Y_train)
    pred = tree.predict(X_test)
    print("The prediction accuracy is: ", tree.score(X_test,Y_test)*100, "%")
    print("Confusion matrix: ")
    print(confusion_matrix(Y_test, pred))
    print()
    print("Report: ")
    print()
    print(classification_report(Y_test, pred))

    pickle.dump(tree, open('Id3.sav', 'wb'))
```

In [100…
```python
def Naive_bayes_implementation():
    py = 0
    pn = 0

    total = len(test_file['class'])

    correct = 0

    y = 0
    n = 0

    for i in train_file['class']:
        if i == 0:
            n = n + 1
        else:
            y = y + 1

    py = y / len(train_file['class'])
    pn = n / len(train_file['class'])

    for i,rows in test_file.iterrows():

        arryes = []
        arrno = []

        for j in test_file.columns:


            dyes = train_file[train_file[j] == test_file[j][i]]
            dyes = train_file[train_file['class'] == 1]

            arryes.append(len(dyes['class']) / y)

            dno = train_file[train_file[j] == test_file[j][i]]
            dno = train_file[train_file[j] == 0]

            arrno.append(len(dno['class']) / n)

        yes = 0
        no = 0

        for j in arryes:
            if yes == 0:
                yes = j
            else:
                yes = yes*j

        yes = yes * py

        for j in arrno:
            if no == 0:
                no = j
            else:
                no = no*j

        no = no * pn

        if yes > no and 1 == test_file['class'][i]:
            correct = correct + 1

        elif no > yes and 0 == test_file['class'][i]:
            correct = correct + 1
        print("in processing..................")
    print("Accuracy of the model: ",(correct / len(test_file['class']))*100, "%")
```

In [101…
```python
def Decision_Tree_implementation():

    dataset=train_file
    test_dataset = test_file

    def entropy(data_set):
        """ this function calculate data set entropy """
        Probability_set = []

        for i in data_set:                                                          # create a list of Probability
            counter = 0
            for j in data_set:
                if j == i:
                    counter += 1
            Probability_set.append(counter / len(data_set))

        return sum(list(map(lambda x: (-1) * (x * log(x,2)), Probability_set)))    # return entropy



    def InfoGain(data,split_attribute_name,target_name="class"):

        #Entropy of the total dataset
        total_entropy = entropy(data[target_name])

        ##Calculate the entropy of the dataset
        values,counts= np.unique(data[split_attribute_name],return_counts=True)

        #weighted entropy
        W_Entropy = np.sum([(counts[i]/np.sum(counts))*entropy(data.where(data[split_attribute_name]==values[i]).dropna()[target_name]) for i in range(len(values))])

        #Calculate the information gain
        Info_Gain = total_entropy - W_Entropy
        return Info_Gain


    def ID3(data,originaldata,features,target_attribute="class",Parent_Node = None):

        if len(np.unique(data[target_attribute])) <= 1:
            return np.unique(data[target_attribute])[0]


        elif len(data)==0:
            return np.unique(originaldata[target_attribute])[np.argmax(np.unique(originaldata[target_attribute],return_counts=True)[1])]


        elif len(features) ==0:
            return Parent_Node

        else:

            Parent_Node = np.unique(data[target_attribute])[np.argmax(np.unique(data[target_attribute],return_counts=True)[1])]


            item = [InfoGain(data,feature,target_attribute) for feature in features]
            index_OfBestFeature = np.argmax(item)
            Bestfeature = features[index_OfBestFeature]

            #Create the tree structure. The root gets the name of the feature (best_feature) with the maximum information
            #gain in the first run
            tree = {Bestfeature:{}}

            features = [i for i in features if i != Bestfeature]

            for value in np.unique(data[Bestfeature]):
                value = value

                sub_data = data.where(data[Bestfeature] == value).dropna()

                #Call the ID3 algorithm for each of those sub_datasets with the new parameters, Here the recursion comes in!
                subtree = ID3(sub_data,dataset,features,target_attribute,Parent_Node)

                tree[Bestfeature][value] = subtree

            return(tree)


    def predict(query,tree,default = 1):

        for key in list(query.keys()):
            if key in list(tree.keys()):

                try:
                    result = tree[key][query[key]]
                except:
                    return default

                result = tree[key][query[key]]

                if isinstance(result,dict):
                    return predict(query,result)

                else:
                    return result




    def testing(data,tree):

        queries = data.iloc[:,:-1].to_dict(orient = "records")

        predicted = pd.DataFrame(columns=["predicted"])

        for i in range(len(data)):
            predicted.loc[i,"predicted"] = predict(queries[i],tree,1.0)
        print('The prediction accuracy is: ',(np.sum(predicted["predicted"] == data["class"])/len(data))*100,'%')



    train_dataset=dataset.iloc[:100].reset_index(drop=True)

    """
    Train the tree, Print the tree and predict the accuracy
    """
    tree = ID3(train_dataset,train_dataset,train_dataset.columns[:-1])
    testing(test_dataset,tree)
```

In [102…
```python
def Model(train_file, test_file):

    if dic['model_type'] == 'Naive bayes':
        Naive_bayes()

    elif dic['model_type'] == 'Naive bayes-Implementation':
        Naive_bayes_implementation()

    elif dic['model_type'] == 'Decision tree':
        Decision_Tree()

    elif dic['model_type'] == 'Decision tree-Implementation':
        Decision_Tree_implementation()
```

In [103…
```python
Model(train_file,test_file)
```

```
Number of mislabeled points out of a total 3031 points : 1332
precentence:  56.05410755526229
Confusion matrix:
[[1279  188]
 [1144  420]]

Report:

              precision    recall  f1-score   support

           0       0.53      0.87      0.66      1467
           1       0.69      0.27      0.39      1564

    accuracy                           0.56      3031
   macro avg       0.61      0.57      0.52      3031
weighted avg       0.61      0.56      0.52      3031
```

KNN

In [104…
```python
def Knn():
    X_train = train_file.iloc[:, : -1]
    Y_train = train_file.iloc[:, -1]
    X_test = test_file.iloc[:, : -1]
    Y_test = test_file.iloc[:, -1]


    arr = []

    for i in range(2, 10):
        classifier = KNeighborsClassifier(n_neighbors = i, p = 2, metric = 'euclidean')
        classifier.fit(X_train, Y_train)
        y_pred = classifier.predict(X_test)
        arr.append(accuracy_score(Y_test, y_pred))

    index = 0
    mx = 0

    for i in range(len(arr)):
        if arr[i] >= mx:
            mx = arr[i]
            index = i

    best_n_neighbors = (index + 2)


    classifier = KNeighborsClassifier(n_neighbors = best_n_neighbors - 1, p = 2, metric = 'euclidean')
    classifier.fit(X_train, Y_train)
    y_pred = classifier.predict(X_test)

    print("The best n_neighbors is: " , best_n_neighbors)
    print("The prediction accuracy is: ", mx * 100)
    print("Confusion matrix: ")
    print(confusion_matrix(Y_test, y_pred))
    print()
    print("Report: ")
    print()
    print(classification_report(Y_test, y_pred))

    pickle.dump(classifier, open('Knn.sav', 'wb'))
```

In [105…
```python
Knn()
```

```
The best n_neighbors is:  3
The prediction accuracy is:  51.303200263939296
Confusion matrix:
[[1402   65]
 [1452  112]]

Report:

              precision    recall  f1-score   support

           0       0.49      0.96      0.65      1467
           1       0.63      0.07      0.13      1564

    accuracy                           0.50      3031
   macro avg       0.56      0.51      0.39      3031
weighted avg       0.56      0.50      0.38      3031
```

K- means

In [106…
```python
def K_means():
    X_train = train_file.iloc[:, : -1]
    Y_train = train_file.iloc[:, -1]
    X_test = test_file.iloc[:, : -1]
    Y_test = test_file.iloc[:, -1]
    arr = []

    for i in range(2, 10):
        classifier = KMeans(n_clusters = i)
        classifier.fit(X_train, Y_train)
        y_pred = classifier.predict(X_test)
        arr.append(accuracy_score(Y_test, y_pred))

    index = 0
    mx = 0

    for i in range(len(arr)):
        if arr[i] >= mx:
            mx = arr[i]
            index = i

    best_n_clusters = (index + 2)


    classifier = KMeans(n_clusters = best_n_clusters)
    classifier.fit(X_train, Y_train)
    y_pred = classifier.predict(X_test)

    print("The best n_clusters is: " , best_n_clusters)
    print("The prediction accuracy is: ", mx * 100)
    print("Confusion matrix: ")
    print(confusion_matrix(Y_test, y_pred))
    print()
    print("Report: ")
    print()
    print(classification_report(Y_test, y_pred))

    pickle.dump(classifier, open('K_means.sav', 'wb'))
```

In [107…
```python
K_means()
```

```
The best n_clusters is:  2
The prediction accuracy is:  52.29297261629825
Confusion matrix:
[[615 852]
 [593 971]]

Report:

              precision    recall  f1-score   support

           0       0.51      0.42      0.46      1467
           1       0.53      0.62      0.57      1564

    accuracy                           0.52      3031
   macro avg       0.52      0.52      0.52      3031
weighted avg       0.52      0.52      0.52      3031
```