Ex No. 1	Data Structuring Techniques in ML
Date:	

Aim

To structure data using various data structuring techniques such as Data Binning, Data Sampling, Data Aggregation and Data Dimensionality Reduction, in machine learning.

Definitions

Data Binning

Data binning, also called data discrete binning or data bucketing, is a data pre-processing technique used to reduce the effects of minor observation errors. The original data values which fall into a given small interval, a bin, are replaced by a value representative of that interval, often a central value.

Data Sampling

Sampling is the practice of analyzing a subset of all data in order to uncover the meaningful information in the larger data set.

Data Aggregation

Data aggregation is the process where raw data is gathered and expressed in a summary form for statistical analysis.

Data Dimensionality Reduction

Dimensionality reduction, or dimension reduction, is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension.

Procedure

Open PyCharm Community Edition.

Go to File menu → New Project → Specify the project name → Press "Create" button.

Right Click on Project name → New → Python File → Specify the file name → Press Enter.

Type the following codes. Right click on file name or coding window → Select "Run" to view the result.

Data Binning

Binning.py

```
import numpy as np
import math
from sklearn.datasets import load_iris
from sklearn import datasets, linear_model, metrics

# load iris data set
dataset = load_iris()
a = dataset.data
b = np.zeros(150)

# take 1st column among 4 column of data set
for i in range(150):
        b[i] = a[i, 1]

b = np.sort(b)    # sort the array

# create bins
bin1 = np.zeros((30, 5))
bin2 = np.zeros((30, 5))
bin3 = np.zeros((30, 5))

# Bin mean
for i in range(0, 150, 5):
        k = int(i / 5)
        mean = (b[i] + b[i + 1] + b[i + 2] + b[i + 3] + b[i + 4]) / 5
        for j in range(5):
                bin1[k, j] = mean
print("Bin Mean: \n", bin1)

# Bin boundaries
for i in range(0, 150, 5):
        k = int(i / 5)
        for j in range(5):
                if (b[i + j] - b[i]) < (b[i + 4] - b[i + j]):
```

```
                bin2[k, j] = b[i]
            else:
                bin2[k, j] = b[i + 4]
print("Bin Boundaries: \n", bin2)

# Bin median
for i in range(0, 150, 5):
    k = int(i / 5)
    for j in range(5):
        bin3[k, j] = b[i + 2]
print("Bin Median: \n", bin3)
```

Output

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit (AMD64)] on win32

runfile('C:/Users/2mca1/binning.py', wdir='C:/Users/2mca1')

Bin Mean:

 [[2.18 2.18 2.18 2.18 2.18]

 [2.34 2.34 2.34 2.34 2.34]

 [2.48 2.48 2.48 2.48 2.48]

 [2.52 2.52 2.52 2.52 2.52]

 [2.62 2.62 2.62 2.62 2.62]

 [2.7  2.7  2.7  2.7  2.7 ]

 [2.74 2.74 2.74 2.74 2.74]

 [2.8  2.8  2.8  2.8  2.8 ]

 [2.8  2.8  2.8  2.8  2.8 ]

 [2.86 2.86 2.86 2.86 2.86]

 [2.9  2.9  2.9  2.9  2.9 ]

 [2.96 2.96 2.96 2.96 2.96]

 [3.   3.   3.   3.   3.  ]

 [3.   3.   3.   3.   3.  ]

 [3.   3.   3.   3.   3.  ]
```

[3.   3.   3.   3.   3.]

[3.04 3.04 3.04 3.04 3.04]

[3.1 3.1 3.1 3.1 3.1]

[3.12 3.12 3.12 3.12 3.12]

[3.2 3.2 3.2 3.2 3.2]

[3.2 3.2 3.2 3.2 3.2]

[3.26 3.26 3.26 3.26 3.26]

[3.34 3.34 3.34 3.34 3.34]

[3.4 3.4 3.4 3.4 3.4]

[3.4 3.4 3.4 3.4 3.4]

[3.5 3.5 3.5 3.5 3.5]

[3.58 3.58 3.58 3.58 3.58]

[3.74 3.74 3.74 3.74 3.74]

[3.82 3.82 3.82 3.82 3.82]

[4.12 4.12 4.12 4.12 4.12]]

Bin Boundaries:

[[2. 2.3 2.3 2.3 2.3]

[2.3 2.3 2.3 2.4 2.4]

[2.4 2.5 2.5 2.5 2.5]

[2.5 2.5 2.5 2.5 2.6]

[2.6 2.6 2.6 2.6 2.7]

[2.7 2.7 2.7 2.7 2.7]

[2.7 2.7 2.7 2.8 2.8]

[2.8 2.8 2.8 2.8 2.8]

[2.8 2.8 2.8 2.8 2.8]

[2.8 2.8 2.9 2.9 2.9]

[2.9 2.9 2.9 2.9 2.9]

[2.9 2.9 3. 3. 3.]

[3. 3. 3. 3. 3.]

[3. 3. 3. 3. 3.]

[3. 3. 3. 3. 3.]

[3. 3. 3. 3. 3.]

[3. 3. 3. 3.1 3.1]

[3.1 3.1 3.1 3.1 3.1]

[3.1 3.1 3.1 3.1 3.2]

[3.2 3.2 3.2 3.2 3.2]

[3.2 3.2 3.2 3.2 3.2]

[3.2 3.2 3.3 3.3 3.3]

[3.3 3.3 3.3 3.4 3.4]

[3.4 3.4 3.4 3.4 3.4]

[3.4 3.4 3.4 3.4 3.4]

[3.5 3.5 3.5 3.5 3.5]

[3.5 3.6 3.6 3.6 3.6]

[3.7 3.7 3.7 3.8 3.8]

[3.8 3.8 3.8 3.8 3.9]

[3.9 3.9 3.9 4.4 4.4]]

Bin Median:

[[2.2 2.2 2.2 2.2 2.2]

[2.3 2.3 2.3 2.3 2.3]

[2.5 2.5 2.5 2.5 2.5]

[2.5 2.5 2.5 2.5 2.5]

[2.6 2.6 2.6 2.6 2.6]

[2.7 2.7 2.7 2.7 2.7]

[2.7 2.7 2.7 2.7 2.7]

```
 [2.8 2.8 2.8 2.8 2.8]
 [2.8 2.8 2.8 2.8 2.8]
 [2.9 2.9 2.9 2.9 2.9]
 [2.9 2.9 2.9 2.9 2.9]
 [3.  3.  3.  3.  3. ]
 [3.  3.  3.  3.  3. ]
 [3.  3.  3.  3.  3. ]
 [3.  3.  3.  3.  3. ]
 [3.  3.  3.  3.  3. ]
 [3.  3.  3.  3.  3. ]
 [3.1 3.1 3.1 3.1 3.1]
 [3.1 3.1 3.1 3.1 3.1]
 [3.2 3.2 3.2 3.2 3.2]
 [3.2 3.2 3.2 3.2 3.2]
 [3.3 3.3 3.3 3.3 3.3]
 [3.3 3.3 3.3 3.3 3.3]
 [3.4 3.4 3.4 3.4 3.4]
 [3.4 3.4 3.4 3.4 3.4]
 [3.5 3.5 3.5 3.5 3.5]
 [3.6 3.6 3.6 3.6 3.6]
 [3.7 3.7 3.7 3.7 3.7]
 [3.8 3.8 3.8 3.8 3.8]
 [4.1 4.1 4.1 4.1 4.1]]
```

Data Sampling

Random-sampling.py

```
import numpy as np
# generating population data following Normal Distribution
```

```
N = 10000
mu = 10
std = 2
population_df = np.random.normal(mu,std,N)

# function that creates random sample
def random_sampling(df, n):
    random_sample = np.random.choice(df,replace = False, size = n)
    return(random_sample)
randomSample = random_sampling(population_df, N)
print(randomSample)
```

Output

```
C:\Users\2mca2\PycharmProjects\sumaiya\venv\Scripts\python.exe
C:/Users/2mca2/PycharmProjects/sumaiya/randomsampling.py

[10.50911361 10.17222969 10.40282203 ...   8.91980209 12.00965177

 11.12126602]

Process finished with exit code 0
```

Systematic-sampling.py

```
import numpy as np
import pandas as pd
# generating population data following Normal Distribution
N = 10000
mu = 10
std = 2
population_df = np.random.normal(mu,std,N)

# function that creates random sample using Systematic Sampling
def systematic_sampling(df, step):
    id = pd.Series(np.arange(1,len(df),1))
    df = pd.Series(df)
    df_pd = pd.concat([id, df], axis = 1)
    df_pd.columns = ["id", "data"]
    # these indices will increase with the step amount not 1
    selected_index = np.arange(1,len(df),step)
    # using iloc for getting thee data with selected indices
    systematic_sampling = df_pd.iloc[selected_index]
    return(systematic_sampling)

n = 10
step = int(N/n)
sample = systematic_sampling(population_df, step)
print(sample)
```

Output

C:\Users\2mca2\PycharmProjects\sumaiya\venv\Scripts\python.exe C:/Users/2mca2/PycharmProjects/sumaiya/systematicsampling.py

	id	data
1	2.0	11.376670
1001	1002.0	7.902640
2001	2002.0	9.362893
3001	3002.0	13.432706
4001	4002.0	12.517116
5001	5002.0	7.566955
6001	6002.0	11.995210
7001	7002.0	11.061732
8001	8002.0	7.779656
9001	9002.0	12.856361

Process finished with exit code 0

Data Aggregation

Aggregation.py

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(10, 4),
        index = pd.date_range('1/1/2000', periods=10),
        columns = ['A', 'B', 'C', 'D'])

print(df)

r = df.rolling(window=3,min_periods=1)
print(r)
```

Output

C:\Users\2mca2\PycharmProjects\sumaiya\venv\Scripts\python.exe C:/Users/2mca2/PycharmProjects/sumaiya/dataaggregation.py

A B C D

```
2000-01-01    1.083986    0.590291 -0.702095 -2.022874

2000-01-02 -1.408233 -1.543042 -0.659563    0.578917

2000-01-03 -1.019651    1.454612 -0.742483    0.310860

2000-01-04 -1.837544 -2.381635    0.275290 -1.383948

2000-01-05 -0.183242    0.285801 -0.156981    0.400867

2000-01-06    1.397718    0.221107    0.910881    0.495881

2000-01-07 -1.013295    0.767176 -2.444918 -1.207323

2000-01-08    0.245632 -0.710832    0.006713    2.212504

2000-01-09    1.010055 -0.330848 -0.768370 -0.034331

2000-01-10 -1.172167 -1.429019    1.280776    0.227238

Rolling [window=3,min_periods=1,center=False,axis=0,method=single]

Process finished with exit code 0
```

Data Dimensionality Reduction

Dimen-reduction.py

```
# Import necessary libraries
from sklearn import datasets # to retrieve the iris Dataset
import pandas as pd # to load the dataframe
from sklearn.preprocessing import StandardScaler # to standardize the features
from sklearn.decomposition import PCA # to apply PCA
import seaborn as sns # to plot the heat maps
import matplotlib.pyplot as plt

#Load the Dataset
iris = datasets.load_iris()
#convert the dataset into a pandas data frame
df = pd.DataFrame(iris['data'], columns = iris['feature_names'])
#display the head (first 5 rows) of the dataset
print(df.head())

#Standardize the features
#Create an object of StandardScaler which is present in sklearn.preprocessing
scalar = StandardScaler()
scaled_data = pd.DataFrame(scalar.fit_transform(df)) #scaling the data
print(scaled_data)


#Check the Co-relation between features without PCA
sns.heatmap(scaled_data.corr())
```

```
plt.show()

#Applying PCA
#Taking no. of Principal Components as 3
pca = PCA(n_components = 3)
pca.fit(scaled_data)
data_pca = pca.transform(scaled_data)
data_pca = pd.DataFrame(data_pca,columns=['PC1','PC2','PC3'])
print(data_pca.head())


#Checking Co-relation between features after PCA
sns.heatmap(data_pca.corr())
plt.show()
```

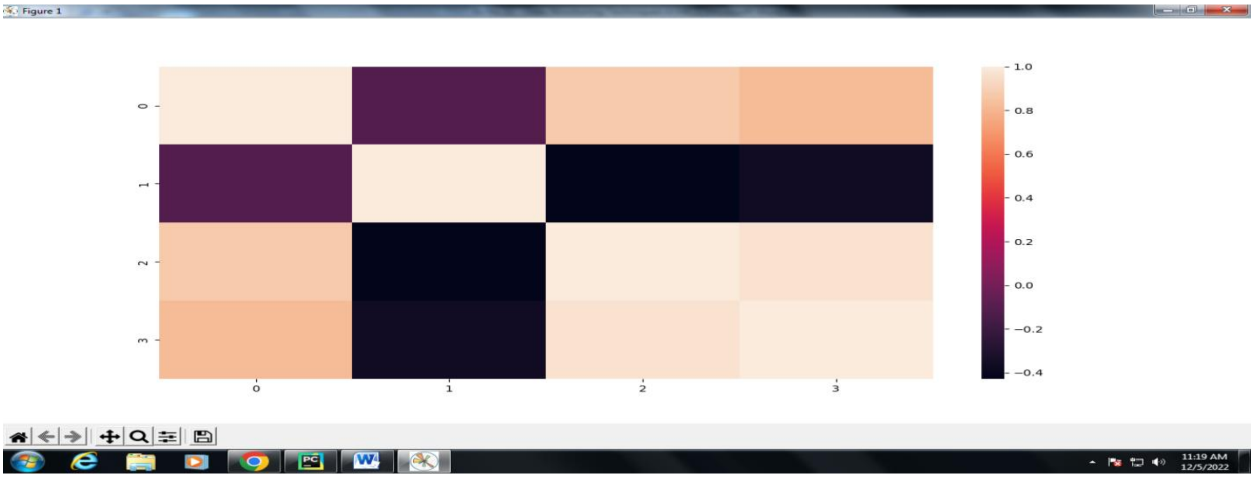
Output

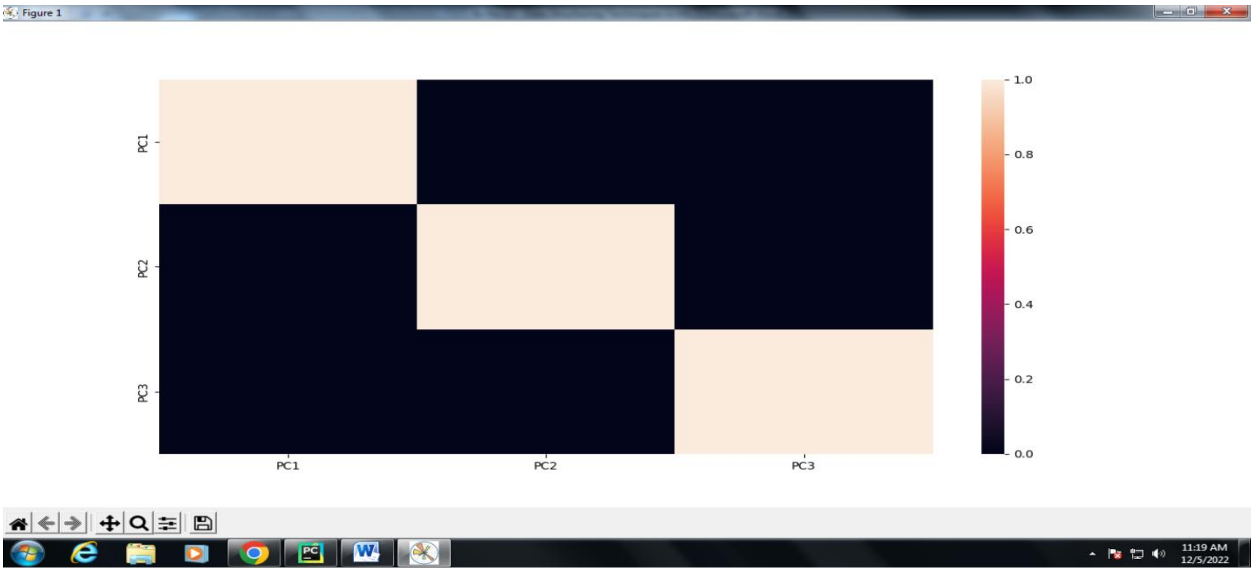
```
C:\Users\2mca2\PycharmProjects\sumaiya\venv\Scripts\python.exe
C:/Users/2mca2/PycharmProjects/sumaiya/dimensionality-reduction.py

     sepal length (cm)   sepal width (cm)   petal length (cm)   petal width (cm)

0    5.1                 3.5                1.4                 0.2

1    4.9                 3.0                1.4                 0.2

2    4.7                 3.2                1.3                 0.2

3    4.6                 3.1                1.5                 0.2

4    5.0                 3.6                1.4                 0.2

            0          1          2          3

0    -0.900681   1.019004 -1.340227 -1.315444

1    -1.143017 -0.131979 -1.340227 -1.315444

2    -1.385353   0.328414 -1.397064 -1.315444

3    -1.506521   0.098217 -1.283389 -1.315444

4    -1.021849   1.249201 -1.340227 -1.315444

..         ...        ...        ...        ...

145   1.038005 -0.131979   0.819596   1.448832

146   0.553333 -1.282963   0.705921   0.922303

147   0.795669 -0.131979   0.819596   1.053935
```

148   0.432165   0.788808   0.933271   1.448832

149   0.068662 -0.131979   0.762758   0.790671

[150 rows x 4 columns]





Thus various data structuring techniques have been successfully applied to structure the data in machine learning.

Result

Thus various data structuring techniques have been successfully applied to structure the data in machine learning.