urop-new

November 28, 2023

0.1 Preprocessing

[43]:
```python
import pandas as pd

# Assuming your CSV file is named 'your_file.csv'
file_path = 'oasis_longitudinal_new.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(file_path)

# Drop rows with missing values
df = df.dropna()
cleaned_file_path = 'urop.csv'
df_cleaned.to_csv(cleaned_file_path, index=False)
```

[44]:
```python
df.columns
```

[44]:
```
Index(['Visit', 'OR Delay', 'M/F', 'Age', 'EDUC', 'SES', 'OOSE', 'CDR', 'eTIV',
       'nWBV', 'AS1', 'Group'],
      dtype='object')
```

[45]:
```python
df.info
```

[45]:
```
<bound method DataFrame.info of      Visit  OR Delay  M/F  Age  EDUC  SES  OOSE
CDR  eTIV   nWBV    AS1  \
0        1         0    0   87    14  2.0  27.0  0.0  1987  0.696  0.883
1        2       457    0   88    14  2.0  30.0  0.0  2004  0.681  0.876
5        1         0    1   88    18  3.0  28.0  0.0  1215  0.710  1.444
6        2       538    1   90    18  3.0  27.0  0.0  1200  0.718  1.462
7        1         0    0   80    12  4.0  28.0  0.0  1689  0.712  1.039
..     …         …  …  …    …   …  …    …    …     …      …
368      2       842    0   82    16  1.0  28.0  0.5  1693  0.694  1.037
369      3      2297    0   86    16  1.0  26.0  0.5  1688  0.675  1.040
370      1         0    1   61    13  2.0  30.0  0.0  1319  0.801  1.331
371      2       763    1   63    13  2.0  30.0  0.0  1327  0.796  1.323
372      3      1608    1   65    13  2.0  30.0  0.0  1333  0.801  1.317

     Group
```

```
0        0
1        0
5        0
6        0
7        0
..     …
368      1
369      1
370      0
371      0
372      0

[354 rows x 12 columns]>
```

```
[46]: df.head(50)
```

```
[46]:     Visit  OR Delay  M/F  Age  EDUC  SES  OOSE  CDR  eTIV   nWBV    AS1  Group
      0       1         0    0   87    14  2.0  27.0  0.0  1987  0.696  0.883      0
      1       2       457    0   88    14  2.0  30.0  0.0  2004  0.681  0.876      0
      5       1         0    1   88    18  3.0  28.0  0.0  1215  0.710  1.444      0
      6       2       538    1   90    18  3.0  27.0  0.0  1200  0.718  1.462      0
      7       1         0    0   80    12  4.0  28.0  0.0  1689  0.712  1.039      0
      8       2      1010    0   83    12  4.0  29.0  0.5  1701  0.711  1.032      0
      9       3      1603    0   85    12  4.0  30.0  0.0  1699  0.705  1.033      0
      13      1         0    1   93    14  2.0  30.0  0.0  1272  0.698  1.380      0
      14      2       742    1   95    14  2.0  29.0  0.0  1257  0.703  1.396      0
      15      1         0    0   68    12  2.0  27.0  0.5  1457  0.806  1.205      1
      16      2       576    0   69    12  2.0  24.0  0.5  1480  0.791  1.186      1
      17      1         0    1   66    12  3.0  30.0  0.5  1447  0.769  1.213      1
      18      2       854    1   68    12  3.0  29.0  0.5  1482  0.752  1.184      1
      19      1         0    1   78    16  2.0  29.0  0.0  1333  0.748  1.316      0
      20      2       730    1   80    16  2.0  29.0  0.0  1323  0.738  1.326      0
      21      3      1598    1   83    16  2.0  29.0  0.0  1323  0.718  1.327      0
      22      1         0    1   81    12  4.0  30.0  0.0  1230  0.715  1.427      0
      23      2       643    1   82    12  4.0  30.0  0.0  1212  0.720  1.448      0
      24      3      1456    1   85    12  4.0  29.0  0.0  1225  0.710  1.433      0
      25      1         0    0   76    16  3.0  21.0  0.5  1602  0.697  1.096      1
      26      2       504    0   77    16  3.0  16.0  1.0  1590  0.696  1.104      1
      27      1         0    0   88     8  4.0  25.0  0.5  1651  0.660  1.063      1
      28      2       707    0   90     8  4.0  23.0  0.5  1668  0.646  1.052      1
      29      1         0    0   80    12  3.0  29.0  0.0  1783  0.752  0.985      0
      30      3       617    0   81    12  3.0  27.0  0.5  1814  0.759  0.968      0
      31      4      1861    0   85    12  3.0  30.0  0.0  1820  0.755  0.964      0
      32      5      2400    0   86    12  3.0  27.0  0.0  1813  0.761  0.968      0
      33      1         0    1   87    14  1.0  30.0  0.0  1406  0.715  1.248      2
      34      3       489    1   88    14  1.0  29.0  0.0  1398  0.713  1.255      2
      35      4      1933    1   92    14  1.0  27.0  0.5  1423  0.696  1.234      2
```

```
36      1       0   0   80     20  1.0   29.0  0.0  1587  0.693  1.106      2
37      2     756   0   82     20  1.0   28.0  0.5  1606  0.677  1.093      2
38      3    1563   0   84     20  1.0   26.0  0.5  1597  0.666  1.099      2
39      1       0   0   72     20  1.0   26.0  0.5  1911  0.719  0.919      1
40      2    1164   0   76     20  1.0   25.0  0.5  1926  0.736  0.911      1
41      1       0   1   61     16  3.0   30.0  0.0  1313  0.805  1.337      0
42      2     828   1   64     16  3.0   29.0  0.0  1316  0.796  1.333      0
43      1       0   1   86     12  4.0   21.0  0.5  1247  0.662  1.407      1
44      2     578   1   87     12  4.0   21.0  0.5  1250  0.652  1.405      1
45      1       0   0   82     12  3.0   27.0  0.5  1420  0.713  1.236      1
46      2     673   0   84     12  3.0   27.0  0.5  1445  0.695  1.214      1
47      1       0   1   69     12  3.0   29.0  0.0  1365  0.783  1.286      0
48      2     609   1   71     12  3.0   30.0  0.0  1360  0.782  1.291      0
49      3    1234   1   73     12  3.0   30.0  0.0  1358  0.775  1.293      0
50      4    1779   1   74     12  3.0   30.0  0.0  1353  0.772  1.297      0
51      1       0   0   64     18  2.0   22.0  0.5  1547  0.737  1.134      1
52      2     610   0   66     18  2.0   21.0  1.0  1562  0.717  1.124      1
53      1       0   1   77     12  4.0   29.0  0.0  1377  0.734  1.275      0
54      2    1099   1   80     12  4.0   30.0  0.0  1390  0.735  1.263      0
55      1       0   1   60     18  1.0   30.0  0.0  1402  0.822  1.252      0
```

0.2 ifsm_final_mod.py CODE

[47]:

```python
import csv
import statistics
import time

# Load data from CSV
data = []
with open("urop.csv", "r") as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        data.append([float(val) for val in row])

totalObjCount = len(data)
print('Total no. of objects=', totalObjCount)
totalLen = len(data[0])
print('Total no. of attributes=', totalLen)

# Define functions
def attrBroadcast(i):
    attr = [col[i] for col in data]
    return attr

def sNorm(a, b):
    return a + b
```

```python
def truncate(n, decimals=0):
    multiplier = 10 ** decimals
    return int(n * multiplier) / multiplier

def findSim(x, y, std):
    valA = x - y + std
    valA = valA / std

    valB = y - x + std
    valB = valB / std

    s = min(valA, valB)
    sRes = max(s, 0)
    sRes = truncate(sRes, 2)
    return sRes

def similarityMatrix_ID(descId, snorm, R):
    listA = []
    decValue = decisionVariable
    attributeValue = attrBroadCastVariable
    for attrId in range(0, len(attributeValue)):
        if decValue[descId] != decValue[attrId] and descId < attrId:
            listA.append((descId, attrId, snorm, R))
    return listA

def similarityMatrix(x1, x2, Reduct, std):
    attrVal = attrBC_Var
    dSimVal = 1 - findSim(attrVal[x1], attrVal[x2], std)
    norm = sNorm(dSimVal, Reduct)
    norm = truncate(norm, 2)
    return (x1, x2, norm, Reduct)

def similarityBackward(x1, x2, Reduct, std):
    attrVal = attrBC_Var2
    dSimVal = 1 - findSim(attrVal[x1], attrVal[x2], std)
    norm = Reduct - dSimVal
    norm = truncate(norm, 2)
    return (x1, x2, norm, Reduct)

# Main
st = time.time()

decisionVariable = [row[0] for row in data]
reductOutput = []
d = {}

posReg, currPosReg = 0.0, 0.0
```

```
snorm, R = 0, 0

arrBroadCast = attrBroadcast(1)
attrBroadCastVariable = arrBroadCast

resultFirst = []
for descId in range(totalObjCount):
    resultFirst.extend(similarityMatrix_ID(descId, snorm, R))

for i in range(1, totalLen - 1):
    BC_List = attrBroadcast(i)
    attrBC_Var = BC_List
    std_dev = statistics.stdev(BC_List)
    if std_dev == 0:
        continue
    d[i] = std_dev

    resultRdd = []
    for x in resultFirst:
        resultRdd.append(similarityMatrix(x[0], x[1], x[3], std_dev))

    combinedRdd = []
    for x in resultRdd:
        combinedRdd.extend([(x[0], x[2]), (x[1], x[2])])

    storeKeyVal = {}
    for col in combinedRdd:
        key = col[0]
        val = min(1, col[1])
        if key not in storeKeyVal:
            storeKeyVal[key] = val
        else:
            storeKeyVal[key] = min(storeKeyVal[key], val)

    values1 = list(storeKeyVal.values())
    currPosReg1 = sum(values1)

    if posReg < currPosReg1:
        reductOutput.append(i)
        posReg = currPosReg1
        finalRes = []
        for x in resultRdd:
            finalRes.append((x[0], x[1], 0.0, x[2]))
        resultFirst = finalRes

    if posReg == totalObjCount:
        break
```

```
print("reductOutput after forward process :: ", reductOutput)

for i in list(reductOutput):
    BC_List = attrBroadcast(i)
    attrBC_Var2 = BC_List
    std_dev2 = d.get(i)
    if std_dev2 == 0:
        continue

    resultRdd = []
    for x in resultFirst:
        resultRdd.append(similarityBackward(x[0], x[1], x[3], std_dev2))

    combinedRdd = []
    for x in resultRdd:
        combinedRdd.extend([(x[0], x[2]), (x[1], x[2])])

    storeKeyVal = {}
    for col in combinedRdd:
        key = col[0]
        val = min(1, col[1])
        if key not in storeKeyVal:
            storeKeyVal[key] = val
        else:
            storeKeyVal[key] = min(storeKeyVal[key], val)

    values1 = list(storeKeyVal.values())
    currPosReg1 = sum(values1)

    if posReg == currPosReg1:
        reductOutput.remove(i)
        finalRes = []
        for x in resultRdd:
            finalRes.append((x[0], x[1], 0.0, x[2]))
        resultFirst = finalRes

print("reductOutput after backward elimination :: ", reductOutput)
et = time.time()

elapsed_time = et - st
print('Execution time:', elapsed_time, 'seconds')


# In[ ]:
```

```



# In[ ]:
```

```
Total no. of objects= 354
Total no. of attributes= 12
reductOutput after forward process ::  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
reductOutput after backward elimination ::  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Execution time: 5.458693742752075 seconds
```

0.3 SVM and KNN classification

[48]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load your CSV file
file_path = 'urop.csv'
df = pd.read_csv(file_path)

# Assuming the last column is the target variable and the rest are features
selected_columns = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
X = df.iloc[:, selected_columns]
y = df.iloc[:, -1]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Standardize the features (important for SVM)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# K-Nearest Neighbors (KNN)
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f'KNN Accuracy: {accuracy_knn:.2f}')

# Support Vector Machine (SVM)
svm_model = SVC(kernel='linear')
```

```
svm_model.fit(X_train_scaled, y_train)
y_pred_svm = svm_model.predict(X_test_scaled)
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f'SVM Accuracy: {accuracy_svm:.2f}')
```

```
KNN Accuracy: 0.54
SVM Accuracy: 0.90
```

[ ]:

```
```