DataTransformation-KalebFlores

February 22, 2024

1 Data Transformation

Data in different scales.

Values in a dataset might have a variety of different magnitudes, ranges, or scales. Algorithms that use distance as a parameter may not weigh all these in the same way. There are various data transformation techniques that are used to transform the features of our data so that they use the same scale, magnitude, or range. This ensures that each feature has an appropriate effect on a model's predictions. Some features in our data might have high-magnitude values (for example, annual salary), while others might have relatively low values (for example, the number of years worked at a company). Just because some data has smaller values does not mean it is less significant.

Reference: Data Science with Python By Rohan Chopra, Aaron England, Mohamed Noordeen Alaudeen July 2019

https://subscription.packtpub.com/book/data/9781838552862/1/ch01lvl1sec08/data-in-different-scales

2 Implementing Scaling Using the Standard Scaler Method

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

[ ]:
```
df = pd.read_csv('Wholesale customers data.csv')
```

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

[ ]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

[ ]:
```
dtypes = df.dtypes
dtypes
```

[ ]:
```
Channel             int64
Region              int64
Fresh               int64
```

```
Milk                int64
Grocery             int64
Frozen              int64
Detergents_Paper    int64
Delicassen          int64
dtype: object
```

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 440 entries, 0 to 439
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Channel           440 non-null    int64
 1   Region            440 non-null    int64
 2   Fresh             440 non-null    int64
 3   Milk              440 non-null    int64
 4   Grocery           440 non-null    int64
 5   Frozen            440 non-null    int64
 6   Detergents_Paper  440 non-null    int64
 7   Delicassen        440 non-null    int64
dtypes: int64(8)
memory usage: 27.6 KB
```

Perform standard scaling and print the first five rows of the new dataset. To do so, use the StandardScaler() class from sklearn.preprocessing and implement the fit_transorm() method. Using the StandardScaler method, we will scale the data into a uniform unit over all the columns. The values of all the features will be converted into a uniform range of the same scale. Because of this, it becomes easier for the model to make predictions.

[ ]:
```
from sklearn import preprocessing
std_scale = preprocessing.StandardScaler().fit_transform(df)
scaled_frame = pd.DataFrame(std_scale, columns=df.columns)

scaled_frame.head()
```

[ ]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	1.448652	0.590668	0.052933	0.523568	-0.041115	-0.589367	-0.043569	-0.066339
1	1.448652	0.590668	-0.391302	0.544458	0.170318	-0.270136	0.086407	0.089151
2	1.448652	0.590668	-0.447029	0.408538	-0.028157	-0.137536	0.133232	2.243293
3	-0.690297	0.590668	0.100111	-0.624020	-0.392977	0.687144		
4	1.448652	0.590668	0.840239	-0.052396	-0.079356	0.173859		

```
3         -0.498588    0.093411
4         -0.231918    1.299347
```

3 Implementing Scaling Using the MinMax Scaler Method

Perform MinMax scaling and print the initial five values of the new dataset. To do so, use the MinMaxScaler() class from sklearn.preprocessing and implement the fit_transorm() method. Add the following code to implement this: Using the MinMaxScaler method, we will scale the data into a uniform unit over all the columns

[ ]:
```
from sklearn import preprocessing
minmax_scale = preprocessing.MinMaxScaler().fit_transform(df)
scaled_frame = pd.DataFrame(minmax_scale,columns=df.columns)
scaled_frame.head()
```

[ ]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	\
0	1.0	1.0	0.112940	0.130727	0.081464	0.003106	0.065427	
1	1.0	1.0	0.062899	0.132824	0.103097	0.028548	0.080590	
2	1.0	1.0	0.056622	0.119181	0.082790	0.039116	0.086052	
3	0.0	1.0	0.118254	0.015536	0.045464	0.104842	0.012346	
4	1.0	1.0	0.201626	0.072914	0.077552	0.063934	0.043455	

	Delicassen
0	0.027847
1	0.036984
2	0.163559
3	0.037234
4	0.108093

4 Implementación de escalamiento decimal

[ ]:
```
def Decimal_scale(df):
    decimal_sca=pd.DataFrame()
    for x in df:
        p = df[x].max()
        q = len(str(abs(p)))
        decimal_sca[x] = df[x]/10**q
    return decimal_sca
```

[ ]:
```
decimal_scale = Decimal_scale(df)
decimal_scale.head()
```

[ ]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	\
0	0.2	0.3	0.012669	0.09656	0.07561	0.00214	0.02674	
1	0.2	0.3	0.007057	0.09810	0.09568	0.01762	0.03293	
2	0.2	0.3	0.006353	0.08808	0.07684	0.02405	0.03516	

```
3      0.1     0.3  0.013265  0.01196  0.04221  0.06404           0.00507
4      0.2     0.3  0.022615  0.05410  0.07198  0.03915           0.01777

   Delicassen
0     0.01338
1     0.01776
2     0.07844
3     0.01788
4     0.05185
```

5 Escalado Min-Max con límite entre [0,1] y [-1,1]

[ ]:
```
minmax_scale01 = preprocessing.MinMaxScaler(feature_range=(0,1)).
 ↪fit_transform(df)
minmax01_scaled_frame = pd.DataFrame(minmax_scale01,columns=df.columns)

minmax_scale11 = preprocessing.MinMaxScaler(feature_range=(-1,1)).
 ↪fit_transform(df)
minmax11_scaled_frame = pd.DataFrame(minmax_scale11,columns=df.columns)
```

[ ]:
```
minmax01_scaled_frame.head()
```

[ ]:
```
   Channel  Region     Fresh      Milk   Grocery    Frozen  Detergents_Paper  \
0      1.0     1.0  0.112940  0.130727  0.081464  0.003106          0.065427
1      1.0     1.0  0.062899  0.132824  0.103097  0.028548          0.080590
2      1.0     1.0  0.056622  0.119181  0.082790  0.039116          0.086052
3      0.0     1.0  0.118254  0.015536  0.045464  0.104842          0.012346
4      1.0     1.0  0.201626  0.072914  0.077552  0.063934          0.043455

   Delicassen
0    0.027847
1    0.036984
2    0.163559
3    0.037234
4    0.108093
```

[ ]:
```
minmax11_scaled_frame.head()
```

[ ]:
```
   Channel  Region     Fresh      Milk   Grocery    Frozen  Detergents_Paper  \
0      1.0     1.0 -0.774120 -0.738546 -0.837072 -0.993787         -0.869146
1      1.0     1.0 -0.874202 -0.734352 -0.793807 -0.942903         -0.838820
2      1.0     1.0 -0.886757 -0.761638 -0.834420 -0.921767         -0.827895
3     -1.0     1.0 -0.763491 -0.968928 -0.909072 -0.790316         -0.975309
4      1.0     1.0 -0.596747 -0.854173 -0.844897 -0.872132         -0.913090

   Delicassen
```

```
0   -0.944305
1   -0.926033
2   -0.672883
3   -0.925532
4   -0.783813
```

6 Construcción de nuevos atributos y agregación de datos

[ ]:
```
n_by__region = df.groupby('Region').size()
n_by__region
```

[ ]:
```
Region
1     77
2     47
3    316
dtype: int64
```

[ ]:
```
df["Channel"].value_counts()
```

[ ]:
```
1    298
2    142
Name: Channel, dtype: int64
```

[ ]:
```
spending_df = df.copy()
spending_df["Spending"] = df["Fresh"] + df["Milk"] + df["Grocery"] +␣
 ↪df["Frozen"] + df["Detergents_Paper"] + df["Delicassen"]
spending_df.head(10)
```

[ ]:
```
   Channel  Region  Fresh   Milk  Grocery  Frozen  Detergents_Paper  \
0        2       3  12669   9656     7561     214              2674
1        2       3   7057   9810     9568    1762              3293
2        2       3   6353   8808     7684    2405              3516
3        1       3  13265   1196     4221    6404               507
4        2       3  22615   5410     7198    3915              1777
5        2       3   9413   8259     5126     666              1795
6        2       3  12126   3199     6975     480              3140
7        2       3   7579   4956     9426    1669              3321
8        1       3   5963   3648     6192     425              1716
9        2       3   6006  11093    18881    1159              7425

   Delicassen  Spending
0        1338     34112
1        1776     33266
2        7844     36610
3        1788     27381
4        5185     46100
5        1451     26710
```

```
6          545      26465
7         2566      29517
8          750      18694
9         2098      46662
```

```
[ ]: regiondf = spending_df.groupby('Region')["Spending"].sum()
     print(regiondf)
     print()
     channeldf = spending_df.groupby('Channel')["Spending"].sum()
     print(channeldf)
```

```
Region
1     2386813
2     1555088
3    10677599
Name: Spending, dtype: int64

Channel
1    7999569
2    6619931
Name: Spending, dtype: int64
```

7 Gasto de leche por región

```
[ ]: regionmilk = df.groupby('Region')["Milk"].sum()
     regionmilk
```

```
[ ]: Region
     1     422454
     2     239144
     3    1888759
     Name: Milk, dtype: int64
```

8 Gasto de abarrotes por canal

```
[ ]: channelcost = df.groupby('Channel')[["Frozen","Grocery","Detergents_Paper"]].
       ↪sum()
     channelcost["Frozen"] + channelcost["Grocery"] + channelcost["Detergents_Paper"]
```

```
[ ]: Channel
     1    2533283
     2    3584786
     dtype: int64
```

En esta actividad aprendimos como transformar los datos desde un dataframe, de manera que podamos normalizar los datos o escalarlos para que las inteligencias artificiales (IA) sean capaces de procesarlos de manera más eficiente. De igual manera aprendimos como crear nuevos atributos

permiten generalizar más la información, de esta manera no tiene que manejar tanas variables una IA y estos datos pueden ser más significativos.