h1ybyyvxx

March 30, 2023

1.How does similarity is calculated if data is categorical in nature.

Categorical data (also known as nominal data) has been studied for a long time in various contexts. However computing similarity between categorical data instances is not straightforward owing to the fact that there is no explicit notion of ordering between categorical values.To overcome this problem, several data-driven similarity measures have been proposed for categorical data. The behavior of such measures directly depends on the data.

Let us try to get to know some of the measures. To define these measures lets begin with a few notations.

Definitions :

Consider a categorical data set D containing N data points (rows), defined over a set of d categorical attributes let A represent the kth attribute where K=1,2,3...d. Let A take n unique values in the data set. Let f (x) be the frequency of the attribute A taking the value x which is one of the n values. let p (x) be the sample probability of the attribute A taking the value x. There are many similarity measures in the literature here we are only going to define a few of them and implement them in python.

The relationship between similarity measure and distance measure :

sim = $^1/$ where sim = similarity and len= distance. Similarity measures assigns value between two data instances X and Y belonging to the data set D as follows:

S(X,Y)= w S (X ,Y) Here w is weight assigned to each attribute and S (X ,Y) is a similarity score function for kth attribute in the given two instances of the data set i.e, X and Y. With different definitions for w and S (X ,Y) we get different similarity measures having different properties and use cases. Here all the measures are defined using w and S (X ,Y)

Overlap Definition : S (X ,Y)=1 if X = Y and equal to 0 otherwise and weight w is 1/d here k =1,2,....d

Eskin Definition: S (X ,Y)=1 if X = Y and equal to n ^{2}/(n ^{2}+2) otherwise and the weight w is 1/d here k =1,2,....d

IOF-Inverse Occurrence Frequency Definition: S (X ,Y)=1 if X = Y and equal to

1/(1+log(f (X))*log(f (Y))) otherwise and the weight w is 1/d here k =1,2,....d

OF Definition: S (X ,Y)=1 if X = Y and equal to

1/(1+log(N/f (X))*log(N/f (Y)) otherwise and the weight w is 1/d here k =1,2,....d

Lin Definition: S (X ,Y)=2log(p (X)) if X = Y and equal to 2log(p (X)+p (Y)) otherwise and the weight w is 1/(log(p (X))+log(p (Y))) here k =1,2,....d

2.Implement K-means for "Car dataset" and come up with the business insights

[1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```python
data = pd.read_csv('Cars_mileage.csv')
```

[3]:
```python
data.head()
```

[3]:
```
   HP        MPG  VOL          SP         WT
0  49  53.700681   89  104.185353  28.762059
1  55  50.013401   92  105.461264  30.466833
2  55  50.013401   92  105.461264  30.193597
3  70  45.696322   92  113.461264  30.632114
4  53  50.504232   92  104.461264  29.889149
```

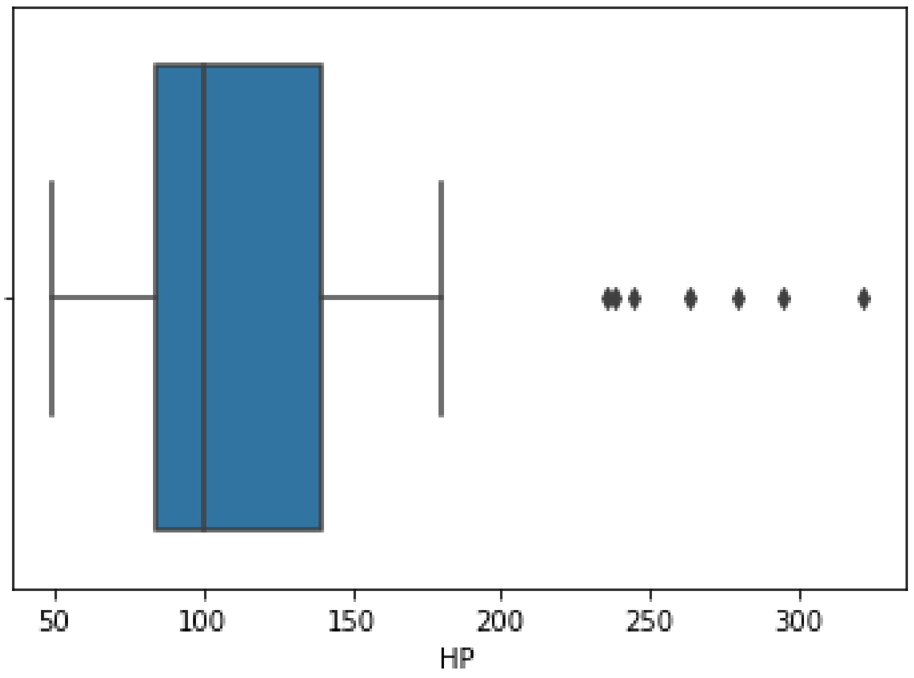
[4]:
```python
data.describe()
```

[4]:
```
               HP        MPG         VOL          SP         WT
count   81.000000  81.000000   81.000000   81.000000  81.000000
mean   117.469136  34.422076   98.765432  121.540272  32.412577
std     57.113502   9.131445   22.301497   14.181432   7.492813
min     49.000000  12.101263   50.000000   99.564907  15.712859
25%     84.000000  27.856252   89.000000  113.829145  29.591768
50%    100.000000  35.152727  101.000000  118.208698  32.734518
75%    140.000000  39.531633  113.000000  126.404312  37.392524
max    322.000000  53.700681  160.000000  169.598513  52.997752
```

[5]:
```python
sns.boxplot(data['HP'])
```

```
C:\Users\87548\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[5]:
```
<AxesSubplot:xlabel='HP'>
```



[6]:
```
data.columns
```

[6]:
```
Index(['HP', 'MPG', 'VOL', 'SP', 'WT'], dtype='object')
```

[7]:
```
# Removing Outliers
from scipy import stats

z = np.abs(stats.zscore(data[['HP', 'MPG', 'VOL', 'SP', 'WT']]))

print(z)

print('**********************************************************************')

print(np.where(z > 3))
```

```
[[1.20629511 2.12438703 0.44061061 1.23140253 0.49023816]
 [1.1005866  1.7180708  0.30525236 1.14087145 0.26129934]
 [1.1005866  1.7180708  0.30525236 1.14087145 0.29799301]
 [0.83631531 1.24235454 0.30525236 0.57323872 0.2391033 ]
 [1.13582277 1.77215741 0.30525236 1.21182554 0.33887816]
 [0.83631531 1.24235454 0.44061061 0.59281571 0.37881419]
 [1.1005866  1.7180708  0.30525236 1.14087145 0.28256505]
 [0.97726    1.354778   2.20026792 1.34399532 2.22453536]
 [0.97726    1.354778   2.20026792 1.34399532 2.15581433]
 [0.66013445 0.86799855 0.21501352 0.4182792  0.20042158]
```

```
[0.78346105 1.12736839 0.44061061 0.73472389 0.40949027]
[0.44871742 0.54347892 2.20026792 0.27968395 2.23718888]
[0.44871742 0.54347892 0.01058357 0.04007367 0.05385348]
[0.78346105 1.12736839 0.44061061 0.73472389 0.40746314]
[0.90678765 1.24660479 0.44061061 0.94758617 0.41164734]
[0.78346105 1.12736839 0.44061061 0.73472389 0.37710085]
[0.69537062 0.92208516 0.35037178 0.50881029 0.38633253]
[0.44871742 0.54347892 2.20026792 0.27968395 2.17802125]
[0.69537062 0.92208516 0.35037178 0.50881029 0.33347355]
[0.48395359 0.49364256 0.19106124 0.21764002 0.14831256]
[0.44871742 0.43955596 0.01058357 0.1727886  0.01180799]
[0.76584297 0.9263354  0.37153891 0.75917009 0.33514564]
[0.39586316 0.42849277 0.1008224  0.08878318 0.03535268]
[0.64251637 0.66696556 0.12477469 0.54713605 0.07727938]
[0.39586316 0.42849277 0.44061061 0.16709116 0.48759676]
[0.44871742 0.43955596 2.20026792 0.49254623 2.19829229]
[0.44871742 0.43955596 0.82273309 0.05532661 0.75878723]
[0.44871742 0.43955596 0.01058357 0.1727886  0.05673768]
[1.15344086 0.99696475 0.23618066 1.55924209 0.27807071]
[0.25491848 0.10822339 0.37153891 0.02132491 0.42124726]
[0.58966211 0.55197941 0.68737484 0.57158225 0.62174504]
[0.58966211 0.55197941 0.1008224  0.65641591 0.11035971]
[0.27253656 0.20533341 0.07965527 0.11488584 0.13866213]
[0.27253656 0.20533341 0.64225542 0.01047519 0.69304639]
[0.64251637 0.5630426  0.1008224  0.72737    0.03881962]
[0.48395359 0.3897196  0.03453585 0.46313063 0.06732815]
[0.48395359 0.3897196  0.48573003 0.52838729 0.49132041]
[0.27253656 0.038723   0.57596887 0.32857635 0.60870364]
[0.27253656 0.038723   0.57596887 0.32857635 0.50790524]
[0.22076984 0.37553513 0.30525236 0.49107265 0.30848967]
[0.39586316 0.0805134  0.64225542 0.36524564 0.66877093]
[0.39586316 0.0805134  0.32641949 0.4109253  0.35112169]
[0.27253656 0.038723   0.30525236 0.28942235 0.253162  ]
[0.39586316 0.0805134  0.48573003 0.52838729 0.54638368]
[0.43109934 0.1346     0.14594182 0.50798206 0.08944633]
[0.30777273 0.01536361 0.01058357 0.31469678 0.02811271]
[0.30777273 0.01536361 0.55201658 0.23638879 0.54872568]
[0.34300891 0.06945021 0.19106124 0.35954821 0.20110061]
[0.22076984 0.37553513 0.57596887 0.45191866 0.58309656]
[0.04350145 0.52806514 0.1008224  0.23069136 0.14045608]
[0.04350145 0.52806514 0.1008224  0.23069136 0.10761914]
[0.04350145 0.52806514 0.1008224  0.23069136 0.13753382]
[0.04350145 0.52806514 1.13856902 0.08060104 1.07240653]
[1.10167413 1.09474551 0.64225542 1.55051482 0.69941256]
[0.74931242 0.83414636 0.64225542 0.98288209 0.65022074]
[0.22076984 0.72351451 1.13856902 0.3451235  1.09804367]
[0.37824508 0.36457601 0.30525236 0.78610099 0.30417763]
[0.04350145 0.52806514 0.1008224  0.23069136 0.04323438]
```

```
 [0.30777273 0.47274922 0.21501352 0.63114147 0.24136353]
 [0.30777273 0.61288266 0.73249426 0.4231484  0.70507688]
 [0.48504113 0.77883043 0.55201658 0.6150603  0.60103751]
 [0.04458898 1.08132775 0.77761368 0.27471455 0.73160477]
 [0.3969507  1.20179346 1.45440495 0.31984907 1.47432127]
 [0.3969507  1.20179346 1.0934496  0.26764375 1.11600704]
 [0.57313156 1.1919596  1.00321077 0.46745469 1.0403533 ]
 [0.83740285 0.62016359 2.20026792 0.35890287 2.24265139]
 [0.83740285 1.24727552 0.68737484 0.77654549 0.74765875]
 [0.83740285 1.24727552 1.27392727 0.86137914 1.23030947]
 [0.83740285 1.24727552 1.0934496  0.83527648 1.08232792]
 [2.24684972 1.44887149 0.597136   2.60830053 0.63509149]
 [2.86348272 1.62465312 2.20026792 3.05515833 2.22785207]
 [0.78454859 1.23621234 1.63488262 0.84263038 1.55787152]
 [0.78454859 1.23621234 1.49952437 0.82305338 1.4692362 ]
 [0.3969507  1.68990629 2.76286807 0.22527702 2.76444017]
 [0.3969507  1.68990629 1.36416611 0.02298138 1.3706083 ]
 [1.01358371 1.72555448 1.36416611 0.80347638 1.39203071]
 [3.60344233 0.27305243 2.20026792 3.40992879 2.18623649]
 [2.12352311 1.67761447 0.73249426 2.06024479 0.74002517]
 [2.56397526 0.04651022 2.20026792 2.13275515 2.23502801]
 [3.12775401 1.60754809 0.91297193 3.292567   0.94146241]
 [2.08828694 2.45962006 0.37153891 1.29849856 0.34057153]]
**********************************************************************
(array([70, 76, 76, 79, 79], dtype=int64), array([3, 0, 3, 0, 3], dtype=int64))
```

[8]:
```
data_outlier_removed = data[(z<3).all(axis=1)]
```

[9]:
```
data_outlier_removed.head(10)
```

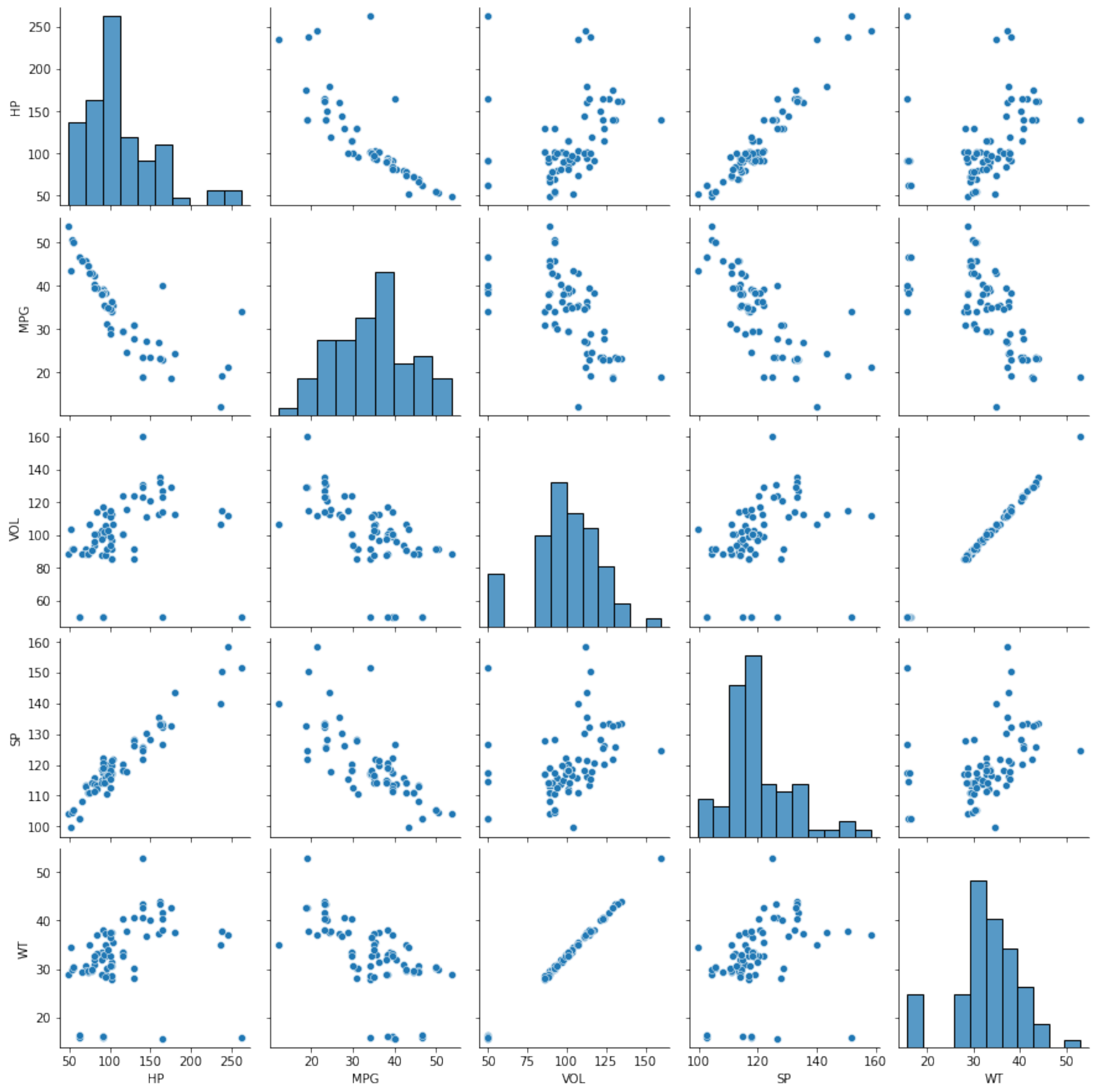
[9]:
```
   HP        MPG  VOL          SP         WT
0  49  53.700681   89  104.185353  28.762059
1  55  50.013401   92  105.461264  30.466833
2  55  50.013401   92  105.461264  30.193597
3  70  45.696322   92  113.461264  30.632114
4  53  50.504232   92  104.461264  29.889149
5  70  45.696322   89  113.185353  29.591768
6  55  50.013401   92  105.461264  30.308480
7  62  46.716554   50  102.598513  15.847758
8  62  46.716554   50  102.598513  16.359484
9  80  42.299078   94  115.645204  30.920154
```

[10]:
```
print('Shape of dataframe before outlier removal: ' + str(data.shape))
print('Shape of dataframe after outlier removal: ' + str(data_outlier_removed.
 ↪shape))
```

```
Shape of dataframe before outlier removal: (81, 5)
Shape of dataframe after outlier removal: (78, 5)
```

```
[11]: sns.pairplot(data_outlier_removed)
```

```
[11]: <seaborn.axisgrid.PairGrid at 0x293fe6f5f40>
```



```
[13]: x = data_outlier_removed
```

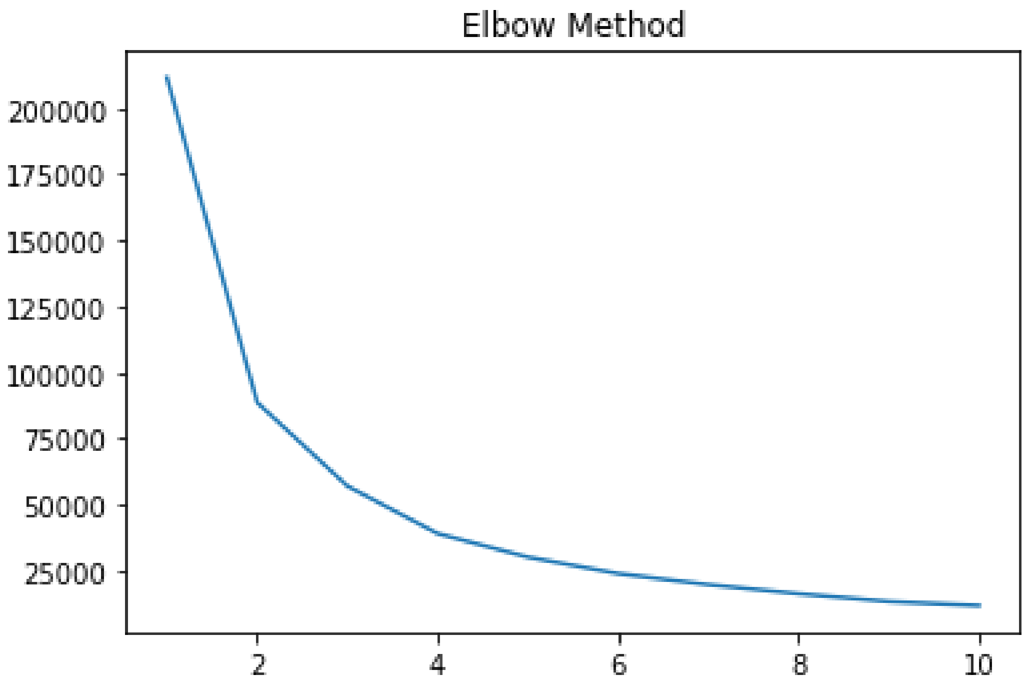
```
[14]: from sklearn.cluster import KMeans
      wcss = []

      for i in range(1,11):
          kmeans =␣
       ↪KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10,random_state=0)
          kmeans.fit(x)
```

```
    wcss.append(kmeans.inertia_)

plt.plot(range(1,11),wcss)
plt.title('Elbow Method')
plt.show()
```

```
C:\Users\87548\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



[15]:
```
kmeans =␣
↪KMeans(n_clusters=4,init='k-means++',max_iter=300,n_init=10,random_state=0)
y_hc = kmeans.fit_predict(x)
```

[16]:
```
y_hc
```

[16]:
```
array([3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 0, 0, 3,
       0, 3, 0, 3, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 2, 2])
```

[ ]: