VIT-AP UNIVERSITY, ANDHRA PRADESH

CSE4005 – Data ware house and data mining - Lab Sheet :7

Academic year: 2022-2023 **Branch/ Class:** B.Tech
Semester: Fall **Date:** 16-10-22
Faculty Name: Dr Aravapalli rama sathish **School:** SCOPE
Student name: Majjiga Jaswanth **Reg. no.:** 20BCD7171

---

1. Develop User Defined functions to calculate dissimilarity matrices for Nominal attributes, Binary attributes, Numeric attributes, Ordinal attributes, Mixed attributes. Apply these functions on the following data:

Obj Id	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
1	4	D3	0	24	T	Y	P	poor	A	14
2	5	D1	1	36	F	Y	N	average	B	16
3	6	D4	1	43	T	N	P	good	C	18
4	7	D6	0	13	T	N	N	average	E	12
5	8	D1	1	22	F	N	P	poor	B	21

Hint:

- All Binary attributes are symmetric.
- All Numeric attributes are in Euclidean space
- Use the following ordinal relationships in the following attributes:
  A8: poor < average < good **A9:** A < B < C < D < E **A2:** D6 < D5 < D4 < D3 < D2 < D1

```
import pandas as pd
import numpy as np
data={
    'Obj id':[1,2,3,4,5],
    'A1':[4,5,6,7,8], 'A2':['D3','D1','D4','D6','D1'],
    'A3':[0,1,1,0,1],
    'A4':[24,36,43,13,22], 'A5':['T','F','T','T','F'],
    'A6':['Y','Y','N','N','N'],
    'A7':['P','N','P','N','P'],
    'A8':['poor','average','good','average','poor'],
    'A9':['A','B','C','E','B'],
    'A10':[14,16,18,12,21]
  }
df=pd.DataFrame(data)
df
```

Output

	Obj id	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
0	1	4	D3	0	24	T	Y	P	poor	A	14
1	2	5	D1	1	36	F	Y	N	average	B	16
2	3	6	D4	1	43	T	N	P	good	C	18
3	4	7	D6	0	13	T	N	N	average	E	12
4	5	8	D1	1	22	F	N	P	poor	B	21

```
def nominal(x):
    DATA=dict()
    for i in x:
        for j in x:
            if i!=j:
                DATA[i]=1
            else:
                DATA[i]=0
    return DATA
print(nominal(data['A2']))
```

Code

```
{'D3': 1, 'D1': 0, 'D4': 1, 'D6': 1}
```

2. Calculate Term Frequency Vector (Document Vectors) for the following documents and calculate Cosine Similarity between every pair of documents to identify which documents are more similar.

Data 1:

doc1 = "I want to start learning to charge something in life"
doc2 = "reading something about life no one else knows"
doc3 = "Never stop learning"
doc4 = "life learning"

Data 2:

doc1 = "Mr. Imran Khan win the president seat after winning the National election 2020-2021. Though he lost the support of some republican friends, Imran Khan is friends with President Nawaz Sharif"

doc2 = "President Imran Khan says Nawaz Sharif had no political interference is the election outcome. He claimed President Nawaz Sharif is a friend who had nothing to do with the election"

```
import numpy as np
from nltk.tokenize import word_tokenize

import nltk
nltk.download('punkt')

text = ["I want to start learning to charge something in life",
        "reading something about life no one else knows",
        "Never stop learning",
        "life learning"]

sentences = []
word_set = []

for sent in text:
    x = [i.lower() for i in word_tokenize(sent) if i.isalpha()]
    sentences.append(x)
    for word in x:
      if word not in word_set:
        word_set.append(word)

word_set = set(word_set)
total_documents = len(sentences)

index_dict = {}
i = 0
for word in word_set:
    index_dict[word] = i
    i += 1

def count_dict(sentences):
    word_count = {}
```

```
    word_count = {}
    for word in word_set:
        word_count[word] = 0
        for sent in sentences:
            if word in sent:
                word_count[word] += 1
    return word_count

word_count = count_dict(sentences)

def termfreq(document, word):
    N = len(document)
    occurance = len([token for token in document if token == word])
    return occurance/N

def inverse_doc_freq(word):
    try:
        word_occurance = word_count[word] + 1
    except:
        word_occurance = 1
    return np.log(total_documents/word_occurance)

def tf_idf(sentence):
    tf_idf_vec = np.zeros((len(word_set),))
    for word in sentence:
        tf = termfreq(sentence,word)
        idf = inverse_doc_freq(word)

        value = tf*idf
        tf_idf_vec[index_dict[word]] = value
    return tf_idf_vec
```

```
vectors = []
for sent in sentences:
    vec = tf_idf(sent)
    vectors.append(vec)

print(vectors[0])
```

Output

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[0.06931472 0.         0.06931472 0.         0.         0.13862944
 0.02876821 0.         0.         0.06931472 0.         0.06931472
 0.         0.         0.         0.         0.06931472]
```

```python
import math
import re
from collections import Counter

WORD = re.compile(r"\w+")

def get_cosine(vec1,vec2):
    intersection = set(vec1.keys()) & set(vec2.keys())
    numerator = sum([vec1[x] * vec2[x] for x in intersection])

    sum1 = sum([vec1[x] ** 2 for x in list(vec1.keys())])
    sum2 = sum([vec2[x] ** 2 for x in list(vec2.keys())])
    denominator = math.sqrt(sum1) * math.sqrt(sum2)

    if not denominator:
       return 0.0
    else:
       return float(numerator) / denominator

def text_to_vector(text):
    words = WORD.findall(text)
    return Counter(words)

doc1 = "I want to start learning to charge something in life"
doc2 = "reading something about life no one else knows"
doc3 = "Never stop learning"
doc4 = "life learning"
```

```python
v1 = text_to_vector(doc1)
v2 = text_to_vector(doc2)
v3 = text_to_vector(doc3)
v4 = text_to_vector(doc4)

c12 = get_cosine(v1, v2)
c13 = get_cosine(v1, v3)
c14 = get_cosine(v1, v4)
c23 = get_cosine(v2, v3)
c24 = get_cosine(v2, v4)
c34 = get_cosine(v3, v4)

print("Cosine of  document  1 and 2 is  :", c12)
print("Cosine of  document  1 and 3 is  :", c13)
print("Cosine of  document  1 and 4 is  :", c14)
print("Cosine of  document  2 and 3 is  :", c23)
print("Cosine of  document  2 and 4 is  :", c24)
print("Cosine of  document  3 and 4 is  :", c34)
```

Output

```
Cosine  of  document  1  and  2  is  : 0.20412414523193148
Cosine  of  document  1  and  3  is  : 0.16666666666666669
Cosine  of  document  1  and  4  is  : 0.40824829046386296
Cosine  of  document  2  and  3  is  : 0.0
Cosine  of  document  2  and  4  is  : 0.24999999999999994
Cosine  of  document  3  and  4  is  : 0.40824829046386296
```

```
import numpy as np
from nltk.tokenize import word_tokenize

import nltk
nltk.download('punkt')


text = ["Mr. Imran Khan win the president seat after winning the National election 2020- 2021. Though he lost the support of some republican friends, Imra
        "President Imran Khan says Nawaz Sharif had no political interfe rence is the election outcome.He claimed President Nawaz Sharif is a friend who h
        "Post elections, Vladimir Nawaz Sharif win the president seat of Russia. President Nawaz Sharif had served as the Prime Minister earlier in his po

sentences = []
word_set = []

for sent in text:
    x = [i.lower() for i in word_tokenize(sent) if i.isalpha()]
    sentences.append(x)
    for word in x:
        if word not in word_set:
            word_set.append(word)

word_set = set(word_set)

total_documents = len(sentences)
index_dict = {}
i = 0
for word in word_set:
    index_dict[word] = i
    i += 1
```

```
def count_dict(sentences):
    word_count = {}
    for word in word_set:
        word_count[word] = 0
        for sent in sentences:
            if word in sent:
                word_count[word] += 1
    return word_count

word_count = count_dict(sentences)

def termfreq(document, word):
    N = len(document)
    occurance = len([token for token in document if token == word])
    return occurance/N

def inverse_doc_freq(word):
    try:
        word_occurance = word_count[word] + 1
    except:
        word_occurance = 1
    return np.log(total_documents/word_occurance)

def tf_idf(sentence):
    tf_idf_vec = np.zeros((len(word_set),))
    for word in sentence:
        tf = termfreq(sentence,word)
        idf = inverse_doc_freq(word)
```

```
            value = tf*idf
            tf_idf_vec[index_dict[word]] = value
        return tf_idf_vec

    vectors = []
    for sent in sentences:
        vec = tf_idf(sent)
        vectors.append(vec)

    print(vectors[0])
```

Output

```
[ 0.         -0.03082308  0.          0.          0.          0.0144809
  0.          0.02896179  0.          0.          0.          0.
  0.         -0.01027436  0.          0.          0.0144809   0.
  0.0144809   0.          0.          0.          0.          0.
  0.0144809   0.          0.0144809  -0.01027436  0.          0.0144809
  0.0144809   0.          0.          0.          0.          0.
 -0.02054872  0.          0.          0.          0.0144809   0.
  0.          0.          0.          0.0144809   0.        ]
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
doc1 = "Mr. Imran Khan win the president seat after winning the National election 2020-2021. Though he lost the support of some republican friends, Imran I
doc2 = "President Imran Khan says Nawaz Sharif had no political interference is the election outcome. He claimed President Nawaz Sharif is a f riend who h
doc3 = "Post elections, Vladimir Nawaz Sharif win the president seat of Russia. President Nawaz Sharif had served as the Prime Minister earlier in his pol

v1 = text_to_vector(doc1)
v2 = text_to_vector(doc2)
v3 = text_to_vector(doc3)

c12 = get_cosine(v1, v2)
c13 = get_cosine(v1, v3)
c23 = get_cosine(v2, v3)

print("Cosine of document 1 and 2 is :", c12)
print("Cosine of document 1 and 3 is :", c13)
print("Cosine of document 2 and 3 is :", c23)
```

Output

```
Cosine of document 1 and 2 is : 0.4773960376293314
Cosine of document 1 and 3 is : 0.41084327674858256
Cosine of document 2 and 3 is : 0.455157619711416
```

3.

a. Load ***crx.data*** into a data frame and do the following operations: (The data has no headers)

- Change the column names to A1 to A16
- Replace all '?' marks with np.nan
- Convert A2 and A14 attributes to float data type
- Convert '+' to 1 and "–" to 0 of A16 attribute
- Replace values of "A3, A8, A9, A10" attributes to np.nan in 50 random objects
- Save the file as `Transformed_crx.csv`

```
df = pd.read_csv("crx.csv")
df.columns = ['a1','a2','a3','a4','a5','a6','a7','a8','a9','a10','a11', 'a12','a13','a14','a15','a16']
df.replace('?', np.nan, inplace = True)

df['a2'] = df['a2'].astype(float)
df['a14'] = df['a14'].astype(float)

df['a16'].replace({'+':1, '-':0}, inplace = True)

length = len(df)
num = 50
idx_replace = np.random.randint(0, length-1, num)
df.loc[idx_replace, 'a3'] = np.nan
df.loc[idx_replace, 'a8'] = np.nan
df.loc[idx_replace, 'a9'] = np.nan
df.loc[idx_replace, 'a10'] = np.nan

df.to_csv("transformed_crx.csv")
```

b. Ignoring missing values:

- Load the Credit Approval Data Set `Transformed_crx.csv`
- Calculate the percentage of missing values for each variable and sort them in ascending order
- Remove the observations with missing data in any of the variables
- Print and compare the size of the original and complete case datasets

```
df = pd.read_csv("transformed_crx.csv")
percent_missing = df.isnull().sum()*100 / len(df)
percent_missing.sort_values()
```

Output

```
Unnamed: 0    0.000000
a13           0.000000
a12           0.000000
a11           0.000000
a15           0.000000
a16           0.000000
a4            0.870827
a5            0.870827
a6            1.306241
a7            1.306241
a2            1.741655
a1            1.741655
a14           1.886792
a9            6.676343
a10           6.676343
a3            6.676343
a8            6.676343
dtype: float64
```

```
df1 = df.dropna()

print("Size of the data after dropping null values: ", df1.shape)
print("Original size of the data: ", df.shape)
```

Output

```
Size of the data after dropping null values:  (610, 17)
Original size of the data:  (689, 17)
```

c. Performing mean and median imputation:

- Load the Credit Approval Data Set `Transformed_crx.csv`
- Replace the missing values with the median in five numerical variables 'A2', 'A3', 'A8', 'A11', 'A15' using pandas
- Replace the missing values with the mean in five numerical variables 'A2', 'A3', 'A8', 'A11', 'A15' using pandas
- Use SimpleImputer() of scikit-learn to fill the missing values with median and mean, separately.

```
df = pd.read_csv("transformed_crx.csv")

df['a2'].fillna(df['a2'].median())
df['a3'].fillna(df['a3'].median())
df['a8'].fillna(df['a8'].median())
df['a11'].fillna(df['a11'].median())
df['a15'].fillna(df['a15'].median())
df.head()
```

Output

	Unnamed: 0	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16
0	0	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	43.0	560	1
1	1	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	280.0	824	1
2	2	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	100.0	3	1
3	3	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	120.0	0	1
4	4	b	32.08	4.000	u	g	m	v	2.50	t	f	0	t	g	360.0	0	1

code

```
df['a2'].fillna(df['a2'].mean())
df['a3'].fillna(df['a3'].mean())
df['a8'].fillna(df['a8'].mean())
df['a11'].fillna(df['a11'].mean())
df['a15'].fillna(df['a15'].mean())
df.head()
```

Output

	Unnamed: 0	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16
0	0	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	43.0	560	1
1	1	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	280.0	824	1
2	2	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	100.0	3	1
3	3	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	120.0	0	1
4	4	b	32.08	4.000	u	g	m	v	2.50	t	f	0	t	g	360.0	0	1

code

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values = np.nan, strategy = 'median')
imputer = imputer.fit(df[['a2', 'a3', 'a8', 'a11', 'a15']])
data = imputer.transform(df[['a2', 'a3', 'a8', 'a11', 'a15']])
data
```

Output

```
array([[5.867e+01, 4.460e+00, 3.040e+00, 6.000e+00, 5.600e+02],
       [2.450e+01, 5.000e-01, 1.500e+00, 0.000e+00, 8.240e+02],
       [2.783e+01, 1.540e+00, 3.750e+00, 5.000e+00, 3.000e+00],
       ...,
       [2.525e+01, 2.750e+00, 1.000e+00, 1.000e+00, 1.000e+00],
       [1.792e+01, 2.050e-01, 4.000e-02, 0.000e+00, 7.500e+02],
       [3.500e+01, 3.375e+00, 8.290e+00, 0.000e+00, 0.000e+00]])
```

Code

```
imputer = SimpleImputer(missing_values = np.nan, strategy = 'mean')
imputer = imputer.fit(df[['a2', 'a3', 'a8', 'a11', 'a15']])
data = imputer.transform(df[['a2', 'a3', 'a8', 'a11', 'a15']])
data
```

Output

```
array([[5.86700000e+01, 4.46000000e+00, 3.04000000e+00, 6.00000000e+00,
        5.60000000e+02],
       [2.45000000e+01, 5.00000000e-01, 1.50000000e+00, 0.00000000e+00,
        8.24000000e+02],
       [2.78300000e+01, 1.54000000e+00, 3.75000000e+00, 5.00000000e+00,
        3.00000000e+00],
       ...,
       [2.52500000e+01, 4.79825039e+00, 2.16392691e+00, 1.00000000e+00,
        1.00000000e+00],
       [1.79200000e+01, 2.05000000e-01, 4.00000000e-02, 0.00000000e+00,
        7.50000000e+02],
       [3.50000000e+01, 3.37500000e+00, 8.29000000e+00, 0.00000000e+00,
        0.00000000e+00]])
```

d. Performing mode or frequent category imputation:

- Load the Credit Approval Data Set `Transformed_crx.csv`
- Replace the missing values with the mode in the attributes 'A4', 'A5', 'A6', 'A7' using pandas
- Use SimpleImputer() of scikit-learn to fill the missing values with mode.

Code

```
df['a4'].fillna(df['a4'].mode(), inplace = True)
df['a5'].fillna(df['a5'].mode(), inplace = True)
df['a6'].fillna(df['a6'].mode(), inplace = True)
df['a7'].fillna(df['a7'].mode(), inplace = True)
imputer = SimpleImputer(missing_values = np.nan, strategy = 'most_frequent')
imputer = imputer.fit(df[['a2', 'a3', 'a8', 'a11', 'a15']])
data = imputer.transform(df[['a2', 'a3', 'a8', 'a11', 'a15']])
data
```

Output

```
array([[5.867e+01, 4.460e+00, 3.040e+00, 6.000e+00, 5.600e+02],
       [2.450e+01, 5.000e-01, 1.500e+00, 0.000e+00, 8.240e+02],
       [2.783e+01, 1.540e+00, 3.750e+00, 5.000e+00, 3.000e+00],
       ...,
       [2.525e+01, 1.500e+00, 0.000e+00, 1.000e+00, 1.000e+00],
       [1.792e+01, 2.050e-01, 4.000e-02, 0.000e+00, 7.500e+02],
       [3.500e+01, 3.375e+00, 8.290e+00, 0.000e+00, 0.000e+00]])
```