Assignment 2 - CS 484 Introduction to Machine Learning

CWID : A20522183

```
#***********************************Question 1*******************************
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

def calculate_metrics(association_rules_result):
    confidence = association_rules_result['confidence'].values[0]
    support_soda = association_rules_result['antecedent support'].values[0]
    lift = association_rules_result['lift'].values[0]
    leverage = association_rules_result['leverage'].values[0]
    zhang = (confidence - support_soda) / max(support_soda, 1 - support_soda)

    return confidence, lift, leverage, zhang

# Friend items
friend_items = {
    'Andrew': ['Cheese', 'Cracker', 'Soda', 'Wings'],
    'Betty': ['Cheese', 'Soda', 'Tortilla', 'Wings'],
    'Carl': ['Cheese', 'Ice Cream', 'Wings'],
    'Danny': ['Cheese', 'Ice Cream', 'Salsa', 'Soda', 'Tortilla'],
    'Emily': ['Salsa', 'Soda', 'Tortilla', 'Wings'],
    'Frank': ['Cheese', 'Cracker', 'Ice Cream', 'Wings']
}

# Convert friend_items to list of lists for transaction encoding
transactions = list(friend_items.values())

# Transaction encoding
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_ary, columns=te.columns_)

# Apply Apriori algorithm
frequent_itemsets = apriori(df, min_support=0.3, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)

# Filter rules for {Cheese, Wings} ==> {Soda}
filtered_rules = rules[(rules['antecedents'] == {'Cheese', 'Wings'}) & (rules['consequents'] == {'Soda'})]

# Calculate metrics for {Cheese, Wings} ==> {Soda}
confidence, lift, leverage, zhang = calculate_metrics(filtered_rules)

print('Metrics for {Cheese, Wings} ==> {Soda}:')
print('Confidence:', confidence)
print('Lift:', lift)
print('Leverage:', leverage)
print('Zhang\'s metric:', zhang)
```

```
Metrics for {Cheese, Wings} ==> {Soda}:
Confidence: 0.5
Lift: 0.75
Leverage: -0.1111111111111111
Zhang's metric: -0.24999999999999994
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
#***********************************Question 2.a*******************************
import numpy as np
import pandas as pd
import random
import sys

import matplotlib.pyplot as plt
from google.colab import files
```

```
uploaded = files.upload()
import io

def submissionDetails():
    print("""\nName : Sai Ram Oduri \n Course: CS 484 Introduction to Machine Learning\n\n""")

trainData = pd.read_csv(io.BytesIO(uploaded['Chinese_Bakery.csv']))
trainData.head()
#The table has two columns Customer & Item
#Customer i
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable

```
Saving Chinese_Bakery.csv to Chinese_Bakery.csv
```

	Customer	Item
0	1	Egg Custard w/ Sweet Top Bun
1	1	Plain Dinner Rolls
2	1	Pineapple Sweet Top Bun
3	1	Bean Paste Bun
4	1	Ham & Egg Bun

```
#**********************************Question 2.a********************************
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

print('\nFit-Transform using Transaction Encoder\n')

train_data = data.groupby('Customer')['Item'].apply(set).tolist()
te = TransactionEncoder()
te_array = te.fit(trainData).transform(trainData)
df = pd.DataFrame(te_array, columns=te.columns_)

print('\nEncoded Data:\n')
print(df.head())
```

```
Fit-Transform using Transaction Encoder


Encoded Data:

   BBQ Pork Bun  Bean Paste Bun  Coconut Cocktail Bun  Coconut Sweet Top Bun  \
0          True            True                 False                  False
1          True            True                  True                   True
2         False            True                  True                  False
3          True            True                  True                  False
4          True            True                 False                  False

   Coconut Tart  Coconut Twist Bun  Coffee  Egg Custard Tart  \
0         False               True    True              True
1          True               True   False             False
2         False              False   False              True
3         False              False   False              True
4         False               True    True             False

   Egg Custard w/ Sweet Top Bun  Ham & Egg Bun  Milk Tea (Hot)  \
0                          True           True           False
1                          True           True            True
2                          True          False           False
3                         False          False           False
4                          True           True            True

   Pineapple Sweet Top Bun  Plain Dinner Rolls  \
0                     True                True
1                    False               False
2                     True                True
3                    False                True
4                     True                True

   Portuguese-Style Milk Egg Tart  Sponge Cake  \
0                           False         True
```

```
1                           False         True
2                            True        False
3                            True        False
4                            True        False

   Steamed Rice Cake (Brown or White Sugar)
0                                      True
1                                     False
2                                     False
3                                     False
4                                     False
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
#**********************************Question 2.b********************************
# Calculate the number of unique items in the Universal Set
universal_items = set(item for items in train_data  for item in items)
unique_items = len(universal_items)

# Calculate the maximum number of itemsets (2^n - 1)
max_num_itemsets = 2**unique_items - 1

# Calculate the maximum number of association rules (3^n - 2^n - n)
maximum_assoc_rules = 3**unique_items - 2**unique_items - unique_items

print('Number of items in the Universal Set:', unique_items)
print('Maximum number of itemsets in theory:', max_num_itemsets)
print('Maximum number of association rules in theory:', maximum_assoc_rules)
```

```
Number of items in the Universal Set: 16
Maximum number of itemsets in theory: 65535
Maximum number of association rules in theory: 42981169
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

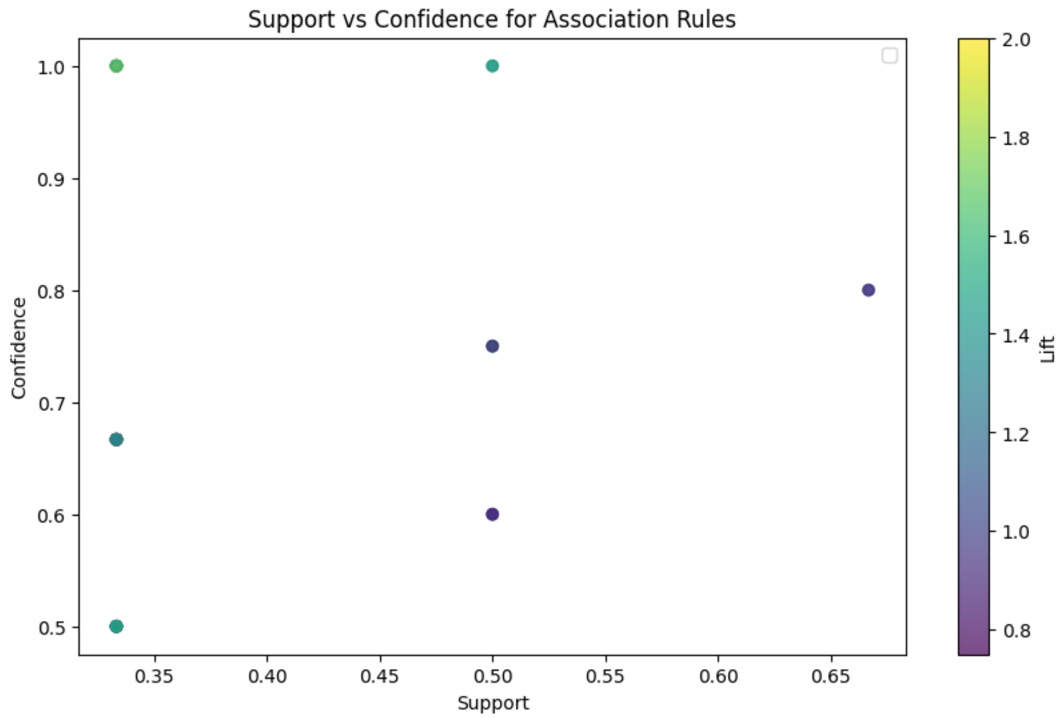
```
#**********************************Question 2.c********************************
min_support_threshold = 100 / len(train_data)
freq_its_fil = frequent_itemsets[frequent_itemsets['support'] >= min_support_threshold]
largest_item_set = freq_its_fil['itemsets'].apply(lambda x: len(x)).max()

print('Number of itemsets with atleast 100 customers:',len(freq_its_fil))
print('Largest number of items (k) among these itemsets:', largest_item_set)
```

```
Number of itemsets with atleast 100 customers: 25
Largest number of items (k) among these itemsets: 3
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
#**********************************Question 2.d********************************
import matplotlib.pyplot as plt

min_confidence_threshold = 0.01


filtered_rules_confidence = rules[rules['confidence'] >= min_confidence_threshold]

# Plot Support vs Confidence with a color gradient based on Lift
plt.figure(figsize=(10, 6))
scatter = plt.scatter(
    filtered_rules_confidence['support'],
    filtered_rules_confidence['confidence'],
    c=filtered_rules_confidence['lift'],
    cmap=plt.cm.get_cmap('viridis'),
    alpha=0.7
)

plt.xlabel('Support')
plt.ylabel('Confidence')
plt.title('Support vs Confidence for Association Rules')
plt.colorbar(scatter, label='Lift')
plt.legend()
```

```
plt.show()

# Number of association rules meeting the confidence threshold
num_association_rules = len(filtered_rules_confidence)
print('Number of association rules with at least 1% confidence:', num_association_rules)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
<ipython-input-17-e23d3796abf4>:14: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be rem
  cmap=plt.cm.get_cmap('viridis'),
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ign
```




```
Number of association rules with at least 1% confidence: 47
```

```
# Corrected column names based on the rules DataFrame
table_columns = ['antecedents', 'consequents', 'support', 'confidence', 'conviction', 'lift']

# Display the relevant metrics in a table
display_table = sorted_rules[table_columns]

# Display the table
print(display_table)
```

```
            antecedents       consequents   support  confidence  conviction  \
13              (Salsa)        (Tortilla)  0.333333         1.0         inf
38       (Soda, Salsa)        (Tortilla)  0.333333         1.0         inf
41              (Salsa)  (Tortilla, Soda)  0.333333         1.0         inf
11              (Salsa)            (Soda)  0.333333         1.0         inf
14           (Tortilla)            (Soda)  0.500000         1.0         inf
22            (Cracker)   (Cheese, Wings)  0.333333         1.0         inf
28  (Tortilla, Cheese)            (Soda)  0.333333         1.0         inf
37   (Tortilla, Salsa)            (Soda)  0.333333         1.0         inf
43   (Tortilla, Wings)            (Soda)  0.333333         1.0         inf
0             (Cracker)          (Cheese)  0.333333         1.0         inf
1           (Ice Cream)          (Cheese)  0.500000         1.0         inf
8             (Cracker)           (Wings)  0.333333         1.0         inf
19   (Cracker, Cheese)           (Wings)  0.333333         1.0         inf
20    (Cracker, Wings)          (Cheese)  0.333333         1.0         inf
24  (Ice Cream, Wings)          (Cheese)  0.333333         1.0         inf

    lift
13   2.0
38   2.0
41   2.0
11   1.5
14   1.5
22   1.5
28   1.5
```

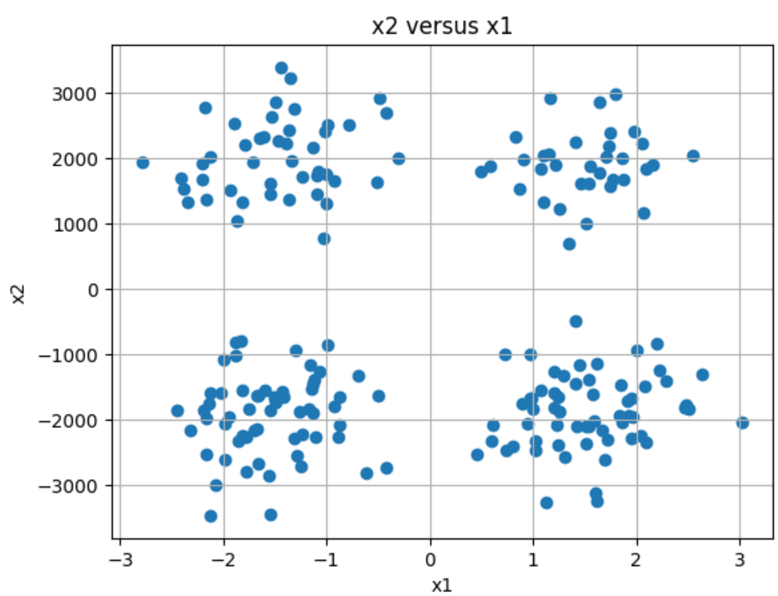
```
37    1.5
43    1.5
0     1.2
1     1.2
8     1.2
19    1.2
20    1.2
24    1.2
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
  and should_run_async(code)
```

```
#***********************************Question 3**********************

#***********************************Question 3.a*******************************

import numpy as np
import pandas as pd
import random
import sys

import matplotlib.pyplot as plt
from google.colab import files
uploaded = files.upload()
import io
# Load the data
data = pd.read_csv(io.BytesIO(uploaded['TwoFeatures.csv']))
x1 = data['x1']
x2 = data['x2']

# Plot x2 versus x1
plt.scatter(x1, x2)
plt.xlabel('x1')
plt.ylabel('x2')
plt.grid(True)
plt.title('x2 versus x1')
plt.show()
```

Choose Files TwoFeatures.csv

- **TwoFeatures.csv**(text/csv) - 2751 bytes, last modified: 9/13/2023 - 100% done

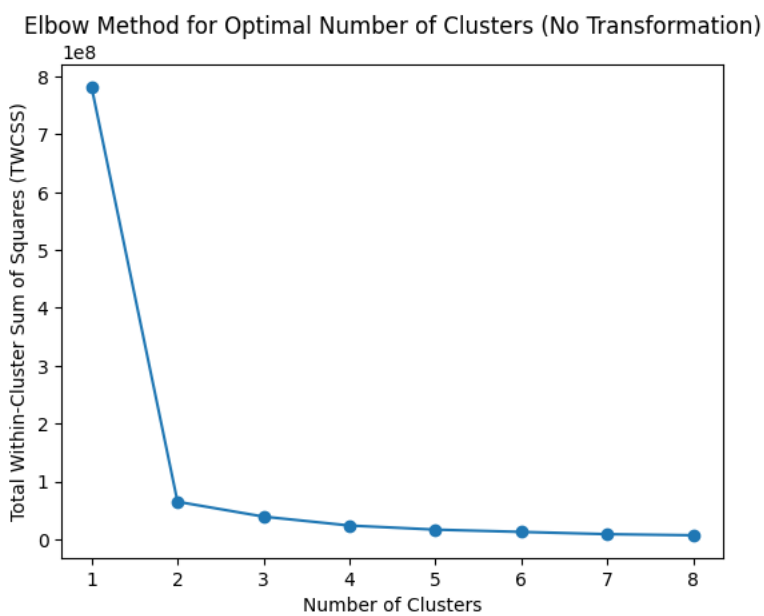
```
Saving TwoFeatures.csv to TwoFeatures.csv
```




```
#***********************************Question 3.b*******************************

from sklearn.cluster import KMeans

X = data[['x1', 'x2']].values

# Find optimal number of clusters without any transformation
def find_optimal_clusters(X, max_clusters):
    distortions = []
```

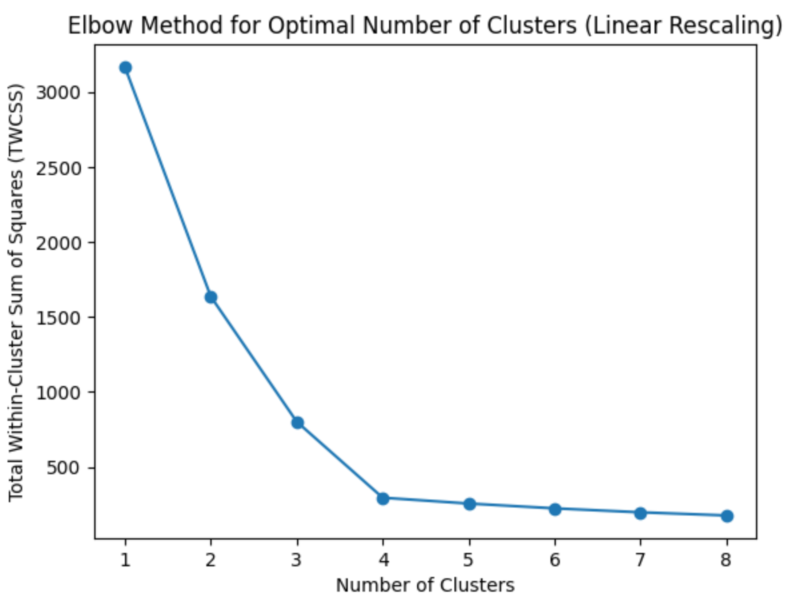
```
        for i in range(1, max_clusters + 1):
            kmeans = KMeans(n_clusters=i, random_state=0).fit(X)
            distortion = kmeans.inertia_
            distortions.append(distortion)
        return distortions, kmeans.cluster_centers_

    # Find optimal clusters without any transformation
    max_clusters = 8
    distortions, centroids = find_optimal_clusters(X, max_clusters)

    # Plot Elbow Values vs. Number of Clusters
    plt.plot(range(1, max_clusters + 1), distortions, marker='o')
    plt.xlabel('Number of Clusters')
    plt.ylabel('Total Within-Cluster Sum of Squares (TWCSS)')
    plt.title('Elbow Method for Optimal Number of Clusters (No Transformation)')
    plt.show()

    # Display results in a table
    table_no_transform = pd.DataFrame({
        'Number of Clusters': range(1, max_clusters + 1),
        'TWCSS': distortions
    })
    print('Results without any transformation:')
    print(table_no_transform)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
```




```
Results without any transformation:
   Number of Clusters         TWCSS
0                   1  7.817878e+08
1                   2  6.482690e+07
2                   3  3.913061e+07
3                   4  2.373439e+07
4                   5  1.661674e+07
5                   6  1.267157e+07
6                   7  8.781406e+06
7                   8  6.874608e+06
```

```
#**********************************Question 3.c********************************
# Linear rescale x1 and x2 to have a minimum of zero and a maximum of ten
x1_rescaled = 10 * (x1 - x1.min()) / (x1.max() - x1.min())
x2_rescaled = 10 * (x2 - x2.min()) / (x2.max() - x2.min())
X_rescaled = np.column_stack((x1_rescaled, x2_rescaled))

# Find optimal number of clusters with linear rescaling
distortions_rescaled, centroids_rescaled = find_optimal_clusters(X_rescaled, max_clusters)

# Plot Elbow Values vs. Number of Clusters after rescaling
plt.plot(range(1, max_clusters + 1), distortions_rescaled, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Total Within-Cluster Sum of Squares (TWCSS)')
plt.title('Elbow Method for Optimal Number of Clusters (Linear Rescaling)')
plt.show()

# Display results in a table after rescaling
table_rescaled = pd.DataFrame({
    'Number of Clusters': range(1, max_clusters + 1),
    'TWCSS': distortions_rescaled
})
print('\nResults with linear rescaling:')
print(table_rescaled)

# Inverse transform the centroids to the original scale
centroids_original_scale = centroids_rescaled * [(x1.max() - x1.min()) / 10, (x2.max() - x2.min()) / 10]

# Display centroids in the original scale
print('\nCentroids in the original scale:')
print(pd.DataFrame(centroids_original_scale, columns=['x1', 'x2']))
```




```
Results with linear rescaling:
   Number of Clusters        TWCSS
0                   1  3165.876012
1                   2  1635.898475
2                   3   802.191722
3                   4   296.004591
4                   5   257.053608
5                   6   225.020274
6                   7   198.526713
7                   8   177.766000

Centroids in the original scale:
         x1           x2
0  0.926571  1461.842857
1  4.955500  1750.480000
2  4.289118  5368.223529
3  1.667143  5639.432143
4  4.016818  1052.572727
5  0.735882  5228.047059
6  1.737600  1636.584000
7  4.001053  2034.115789
```

3D Ans. Due to the differing sizes of the features (x1 and x2), two distinct optimum cluster solutions are produced. The clustering in the first solution, which has no alteration, is based on the original scales of x1 and x2. Clustering is conducted on the rescaled features in the second solution using linear rescaling.

Clusters are produced without alteration based on the original distribution of x1 and x2. Linear rescaling alters the feature range and distribution, thereby accentuating distinct patterns in the data. As a result, the ideal number of clusters and their centroids differ between the two methods, indicating how feature scaling affects clustering outcomes.