Rohini Vishwanathan
Knowledge Discovery and Data Mining
IST 340

CE5 Report- Crime,Zoo, and Sequential Pattern

Problem 1: Association Rule Mining (AR) – Crime Dataset

1.a AR Experimentation

We performed association rule mining using one-hot encoded crime category data per state. A pivot table was created based on POSTCODE (state) and CRIME_CA combinations.

We used mlxtend's association_rules to generate rules and evaluated them based on metrics like **support**, **confidence**, **lift**, **conviction**, **reliability**, and **zhangs_metric**. Rules were further filtered based on the number of antecedents and confidence levels.

Pivoted 0/1 encoded dataset head-

CRIME_CA	ASSAULT - BOT 3RD	ASSAULT - MID 3RD	ASSAULT - TOP 3RD	AUTO - BOT 3RD	AUTO - MID 3RD	AUTO - TOP 3RD	BURGLARY - BOT 3RD	BURGLARY - MID 3RD	BURGLARY - TOP 3RD	LARCENY - BOT 3RD	LARCENY - MID 3RD	LARCENY - TOP 3RD	MURDER - BOT 3RD	MURDER - MID 3RD	MURDER - TOP 3RD	RAPE - BOT 3RD	RAPE - MID 3RD	RAPE - TOP 3RD	ROBBERY - BOT 3RD	ROBBERY - MID 3RD	ROBBERY - TOP 3RD
POSTCODE																					
AK	0	0	1	0	0	1	0	1	0	0	0	1	0	0	1	0	0	1	0	1	0
AL	0	0	1	0	1	0	0	1	0	1	0	0	0	0	1	0	1	0	0	1	0
AR	0	1	0	1	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	1	0
AZ	0	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	0	1	0	1	0
CA	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1

Top 5 rules-

consequent support	support	confidence	lift	representativity	leverage	conviction	zhangs_metric	jaccard	certainty	kulczynski	reliability	item_count	antecedent_count	consequent_count
0.32	0.24	0.750000	2.343750	1.0	0.1376	2.720000	0.843137	0.600000	0.632353	0.750000	0.430000	2	1	1
0.32	0.24	0.750000	2.343750	1.0	0.1376	2.720000	0.843137	0.600000	0.632353	0.750000	0.430000	2	1	1
0.36	0.24	0.705882	1.960784	1.0	0.1176	2.176000	0.742424	0.521739	0.540441	0.686275	0.345882	2	1	1
0.32	0.28	0.823529	2.573529	1.0	0.1712	3.853333	0.926407	0.736842	0.740484	0.849265	0.503529	2	1	1
0.34	0.28	0.875000	2.573529	1.0	0.1712	5.280000	0.899160	0.736842	0.810606	0.849265	0.535000	2	1	1

1.a Result Data

The top frequent rules discovered include associations like:

- **ASSAULT - BOT 3RD → RAPE - BOT 3RD** (Support = 0.24, Confidence = 0.75, Lift = 2.34)
- **RAPE - TOP 3RD → ASSAULT - TOP 3RD** (Support = 0.28, Confidence = 0.82, Lift = 2.57)

Filtered stats for Support, Confidence, and Reliability-

```
[10] # STEP 5.1: Min/Max for Support, Confidence, and Reliability
     # Filter for rules with one antecedent and lift > 2.0
     filtered_rules_q1 = rules[(rules['antecedent_count'] == 1) & (rules['lift'] > 2)]

     print("Support: Min =", filtered_rules_q1['support'].min(),
           "Max =", filtered_rules_q1['support'].max())

     print("Confidence: Min =", filtered_rules_q1['confidence'].min(),
           "Max =", filtered_rules_q1['confidence'].max())

     print("Reliability: Min =", filtered_rules_q1['reliability'].min(),
           "Max =", filtered_rules_q1['reliability'].max())
```

```
Support: Min = 0.24 Max = 0.28
Confidence: Min = 0.7058823529411764 Max = 0.8750000000000001
Reliability: Min = 0.36588235294117644 Max = 0.5350000000000001
```

Rules with ≥25% support and ≥60% confidence-

consequent support	support	confidence	lift	representativity	leverage	conviction	zhangs_metric	jaccard	certainty	kulczynski	reliability	item_count	antecedent_count	consequent_count
0.32	0.28	0.823529	2.573529	1.0	0.1712	3.853333	0.926407	0.736842	0.740484	0.849265	0.503529	2	1	1
0.34	0.28	0.875000	2.573529	1.0	0.1712	5.280000	0.899160	0.736842	0.810606	0.849265	0.535000	2	1	1
0.34	0.26	0.812500	2.389706	1.0	0.1512	3.520000	0.855204	0.650000	0.715909	0.788603	0.472500	2	1	1
0.32	0.26	0.764706	2.389706	1.0	0.1512	2.890000	0.881119	0.650000	0.653979	0.788603	0.444706	2	1	1

1.b Top Rules with 'ASSAULT' or 'ROBBERY'

We searched for rules whose **antecedents** included 'ASSAULT' or 'ROBBERY' with confidence ≥70% and support ≥25%.

Example:

- **ASSAULT - TOP 3RD → RAPE - TOP 3RD** (Support = 0.28, Confidence = 0.875, Lift = 2.57)

Code and matching filtered rules-

consequent support	support	confidence	lift	representativity	leverage	conviction	zhangs_metric	jaccard	certainty	kulczynski	reliability	item_count	antecedent_count	consequent_count
0.34	0.28	0.875000	2.573529	1.0	0.1712	5.28	0.899160	0.736842	0.810606	0.849265	0.535000	2	1	1
0.32	0.26	0.764706	2.389706	1.0	0.1512	2.89	0.881119	0.650000	0.653979	0.788603	0.444706	2	1	1

1.c High Complexity Rules (≥3 items, Confidence ≥75%, Lift ≥2.5)

We extracted and ranked rules with at least 3 items, support ≥15%, and confidence ≥75%.

Example:

- **(ASSAULT - BOT 3RD, AUTO - BOT 3RD) ➝ RAPE - BOT 3RD** (Support = 0.16, Confidence = 1.0, Lift = 3.12)

Screenshot of Q1.c output-

consequent support	support	confidence	lift	representativity	leverage	conviction	zhangs_metric	jaccard	certainty	kulczynski	reliability	item_count	antecedent_count	consequent_count
0.32	0.16	1.000000	3.125000	1.0	0.1088	inf	0.809524	0.500000	1.00000	0.750000	0.680000	3	2	1
0.34	0.16	1.000000	2.941176	1.0	0.1056	inf	0.785714	0.470588	1.00000	0.735294	0.660000	3	2	1
0.32	0.20	0.909091	2.840909	1.0	0.1296	7.48	0.830769	0.588235	0.86631	0.767045	0.589091	3	2	1
0.32	0.20	0.909091	2.840909	1.0	0.1296	7.48	0.830769	0.588235	0.86631	0.767045	0.589091	3	2	1
0.32	0.16	1.000000	3.125000	1.0	0.1088	inf	0.809524	0.500000	1.00000	0.750000	0.680000	3	2	1

1.d Observations

- Strong associations were found between lower-ranked crime categories.
- Multiple rules had perfect confidence but low support.
- Lift values greater than 2.5 indicated strong dependencies.
- Confidence filtering effectively prioritized stronger sequential patterns.

Problem 2: Sequential Pattern Mining – Assoc Dataset

2.1 Experimentation

We analyzed sequential patterns from purchase sequences grouped by CUSTOMER and ordered by TIME. Using the PrefixSpan algorithm, we mined frequent sequences with **support ≥ 20%** of customers.

Raw data load and grouped sequences-

```python
# STEP 1: Load Assoc Dataset
assoc_df = pd.read_csv('/content/Assoc.csv')

# Preview structure
assoc_df.head()
```

	CUSTOMER	TIME	PRODUCT
0	0	0	hering
1	0	1	corned_b
2	0	2	olives
3	0	3	ham
4	0	4	turkey

```python
# STEP 2: Build sequences grouped by CUSTOMER and ordered by TIME

# 1. Sort by CUSTOMER and TIME
assoc_df_sorted = assoc_df.sort_values(by=['CUSTOMER', 'TIME'])

# 2. Group by CUSTOMER and collect ordered PRODUCTS
from itertools import groupby

# Sequential pattern format: List of lists
sequences = assoc_df_sorted.groupby('CUSTOMER')['PRODUCT'].apply(list).tolist()

# Preview a few sequences
sequences[:5]
```

```
[['hering', 'corned_b', 'olives', 'ham', 'turkey', 'bourbon', 'ice_crea'],
 ['baguette', 'soda', 'hering', 'cracker', 'heineken', 'olives', 'corned_b'],
 ['avocado', 'cracker', 'artichok', 'heineken', 'ham', 'turkey', 'sardines'],
 ['olives', 'bourbon', 'coke', 'turkey', 'ice_crea', 'ham', 'peppers'],
 ['hering', 'corned_b', 'apples', 'olives', 'steak', 'avocado', 'turkey']]
```

2.2 Result Data

Examples of top frequent sequences:

- ['heineken'] – Support: 600
- ['cracker', 'heineken'] – Support: 337

- ['soda', 'heineken'] – Support: 218

We computed **confidence** for 2-item sequences, filtering only those with **confidence ≥ 50%**.

Example:

- ['coke', 'ice_crea']: Support = 217, Confidence = 0.73
- ['cracker', 'heineken']: Support = 337, Confidence = 0.69

Code + Output with confidence-based filtering-

```python
# STEP 3: Mine frequent sequences with support ≥ 20%
from prefixspan import PrefixSpan

# Support threshold: 20% of total sequences
min_support = int(0.2 * len(sequences))

# Initialize and mine sequences
ps = PrefixSpan(sequences)
frequent_sequences = ps.frequent(min_support)

# Sort by support descending
frequent_sequences = sorted(frequent_sequences, key=lambda x: -x[0])

# Show top 10
for support, seq in frequent_sequences[:10]:
    print(f"Support: {support}, Sequence: {seq}")
```

```
Support: 600, Sequence: ['heineken']
Support: 488, Sequence: ['cracker']
Support: 486, Sequence: ['hering']
Support: 473, Sequence: ['olives']
Support: 403, Sequence: ['bourbon']
Support: 392, Sequence: ['baguette']
Support: 391, Sequence: ['corned_b']
Support: 363, Sequence: ['avocado']
Support: 337, Sequence: ['cracker', 'heineken']
Support: 318, Sequence: ['soda']
```

```python
# STEP 4: Filter 2-item sequences with confidence ≥ 50%
confident_rules = []

# Create a dictionary for fast support lookup
support_dict = {tuple(seq): supp for supp, seq in frequent_sequences}

for supp, seq in frequent_sequences:
    if len(seq) == 2:
        prefix = tuple([seq[0]])
        if prefix in support_dict:
            confidence = supp / support_dict[prefix]
            if confidence >= 0.5:
                confident_rules.append((supp, seq, round(confidence, 2)))

# Display confident sequences
for supp, seq, conf in confident_rules:
    print(f"Support: {supp}, Confidence: {conf}, Sequence: {seq}")
```

```
Support: 337, Confidence: 0.69, Sequence: ['cracker', 'heineken']
Support: 225, Confidence: 0.57, Sequence: ['baguette', 'heineken']
Support: 220, Confidence: 0.56, Sequence: ['baguette', 'hering']
Support: 220, Confidence: 0.69, Sequence: ['soda', 'cracker']
Support: 218, Confidence: 0.69, Sequence: ['soda', 'heineken']
Support: 217, Confidence: 0.73, Sequence: ['coke', 'ice_crea']
Support: 213, Confidence: 0.53, Sequence: ['bourbon', 'cracker']
Support: 210, Confidence: 0.54, Sequence: ['corned_b', 'olives']
Support: 209, Confidence: 0.53, Sequence: ['baguette', 'avocado']
Support: 208, Confidence: 0.57, Sequence: ['avocado', 'heineken']
Support: 207, Confidence: 0.57, Sequence: ['avocado', 'artichok']
```

Problem 3: Zoo Dataset – Decision Tree Classification

3.1 Data Understanding

The Zoo dataset contains 101 animals described by 16 binary attributes and one target class (animal_type). We observed class imbalance, with majority class 1 (Mammals) having 41 records.

Class distribution-

```
[16] # STEP 3: Class Distribution of Animal Types
     zoo_df['animal_type'].value_counts().sort_index()
```

animal_type	count
1	41
2	20
3	5
4	13
5	4
6	8
7	10

dtype: int64

3.2 Data Preparation

We cleaned and prepared the dataset by removing animal_name and separating features and target.

Feature Shape: (101, 16)
Target Shape: (101,)

Features & Target Split-

```
[17] # STEP 4: Define Features and Target
     X = zoo_df.drop(columns=['animal_name', 'animal_type'])
     y = zoo_df['animal_type']

     X.shape, y.shape
```

```
((101, 16), (101,))
```

3.3 AR Experimentation (Classifier Training)

We used a **Decision Tree Classifier** and split the dataset (80% train / 20% test).

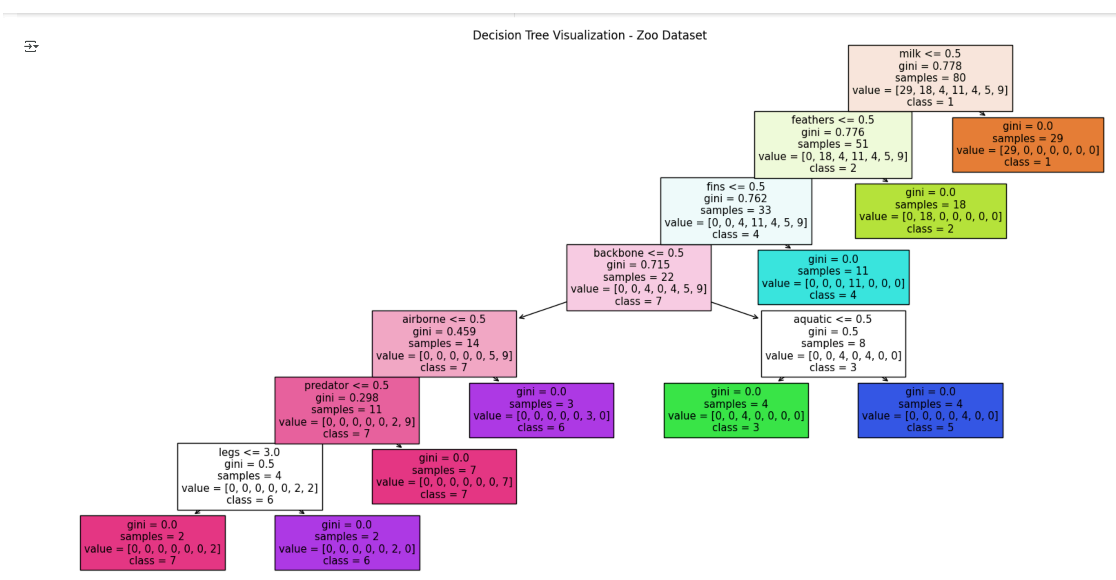
Accuracy Achieved: **95.24%**
Most classes were perfectly predicted except minority classes (e.g., classes 3 and 5) which had **0 recall** due to data imbalance.

Decision Tree Accuracy & Classification Report-

Decision Tree Structure-
To better understand the logic of the trained model, we visualized the Decision Tree using plot_tree from scikit-learn. The tree structure helps identify which features were most important in classifying animal types. For example, features like “hair,” “milk,” and “feathers” appear near the top of the tree, indicating their high influence in decision-making.

```
# STEP 5: Train Decision Tree Classifier

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# 1. Split 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)

# 2. Initialize and train the model
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# 3. Predict and evaluate
y_pred = clf.predict(X_test)

# 4. Show accuracy and classification report
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9523809523809523

Classification Report:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        12
           2       1.00      1.00      1.00         2
           3       0.00      0.00      0.00         1
           4       1.00      1.00      1.00         2
           5       0.00      0.00      0.00         0
           6       1.00      1.00      1.00         3
           7       1.00      1.00      1.00         1

    accuracy                           0.95        21
   macro avg       0.71      0.71      0.71        21
weighted avg       0.95      0.95      0.95        21
```



3.4 Result Data

The model performance was excellent overall. However, class imbalance remains a concern. Metrics like **macro avg (0.71)** reflect this disparity, while **weighted avg (0.95)** favors majority classes.

Google Colab Link-

https://colab.research.google.com/drive/1luj886UzExSol-X1bIgzqMK7CJo_wH2y?usp=sharing