Data Mining & Machine Learning – Assignment 1
(20% of total grade)

* If two homework submissions are found to be similar to each other, both submissions will receive 0.
* Homework solutions must be submitted through Canvas. If you have multiple files, please include all files as one zip file.
* For coding assignments, recommended to use **Jupytor notebook** and submit your **.ipynb** file.
* If you find any **typo/error** in assignment, let me know.

1. [3 pts] Both classification and regression look for a function for prediction. Explain the difference in meaning of these functions using the example in slide p. 18 (Intro-DM-ML chapter)

Regression: - The goal is to predict a quantitative outcome, such as a continuous value. In the chart on the left, the model predicts how long patients survive based on a feature (Gene 1). Each blue point in the chart represents a patient, and the line represents the model's prediction. The output (years survived) is a continuous variable.

Classification: - The objective here is to predict a qualitative outcome, a definite value. In the chart on the right, the model attempts to classify patients into two categories: "disease" (red points) and "healthy" (blue points). The black line in the chart separates the data into these categories. This function divides the data into discrete classes rather than predicting a continuous number.

2. [3pts/ea] Suppose we have a dataset with many features, and it contains every possible combination of feature values.

1) explain why there is no difference in performance (supervised or unsupervised) between different algorithms

When a dataset contains **every possible combination** of feature values, the dataset is **complete**, and each possible state or condition of the data is represented.

Supervised Learning: In a complete dataset, every input-output mapping (label) is already known. So, regardless of the algorithm (e.g., decision trees, k-NN, neural networks), the algorithms will all perform well because they can find an exact match for any new input based on the training set. There's no need for any generalization because every possible feature combination has already been seen during training.

Unsupervised Learning: In unsupervised learning (e.g., clustering or dimensionality reduction), if every feature combination is represented, the data points will already perfectly reflect the structure of the dataset. Any algorithm applied would not provide additional insights, as every data pattern or cluster would already be evident due to the completeness of the data.

In both cases, there is no significant difference in performance between algorithms because the entire

distribution of the data is captured. The algorithms do not need to generalize or discover unseen patterns since all possible patterns are already present in the dataset.

2) With enough number of features, explain why this type of dataset is impossible to happen.

As the number of features in a dataset increases, the number of possible combinations of feature values grows exponentially. For example, if a feature can take on two values (binary feature), and there are n features, then the total number of possible combinations is 2^n. For datasets with more complex feature types (e.g., categorical with multiple levels, continuous variables), the number of combinations becomes even larger.

For a small number of features, it might be possible to have all combinations of feature values represented in the dataset. However, as the number of features increases, the total number of possible combinations becomes so vast that it is computationally and practically impossible to have a dataset that contains every single possible combination. This is due to the curse of dimensionality, where the volume of data required to represent all combinations grows exponentially with the number of features.

3. [3pts/ea] With the following dataset,

A	B	C	Class
10	c0	100	1
7	c2	160	0
20	c1	40	1
4	c1	80	1
6	c0	20	1
12	c1	30	1
16	c2	110	0

1) what kind of preprocessing tasks are necessary in this dataset.

Preprocessing Tasks:

Handling Categorical Data: The attribute B is categorical with values like c0, c1, and c2. We need to convert it into a numerical format, typically using one-hot encoding or target encoding.

Normalization or Scaling: The numerical features like A and C vary significantly in scale, with A ranging from 4 to 20 and C ranging from 20 to 160. To ensure that all features contribute equally during model training, normalization techniques such as min-max scaling or standardization should be applied.

Class Imbalance Check: Since this is a classification task, we need to ensure that the target variable Class (binary: 0 or 1) is balanced. If there is a class imbalance, techniques like SMOTE (Synthetic Minority Over-sampling Technique) or class weighting may be applied.

2) change A=10, 7, 20 to min-max scaling, respectively.

* Data mining and machine learning assignment.

Q-3

2)

min-max scaling transforms a value in a range $[x_{min}, x_{max}]$ to a value between 0 and 1 using the formula.

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} (max - min) + min$$

For column A, the minimum value is 4, and the maximum value is 20.

Using min-max scaling:

↳ For A = 10:

$$A' = \frac{10-4}{20-4} = \frac{6}{16} = 0.375$$

↳ For A = 7:

$$A' = \frac{7-4}{20-4} = \frac{3}{16} = 0.1875$$

↳ for A = 20:

$$A' = \frac{20-4}{20-4} = \frac{16}{16} = 1$$

3) show the result of one-hot-encoding for attribute B

The column B contains categorical values: c0, c1, and c2. One-hot encoding converts these categorical values into multiple binary columns, each representing one unique value.

3)

B	B-c_0	B-c_1	B-c_2
c_0	1	0	0
c_2	0	0	1
c_1	0	1	0
c_1	0	1	0
c_0	1	0	0
c_1	0	1	0
c_2	0	0	1

4) compute target encoder value with smoothing for B='c2'

4)

Encoding $= \alpha \cdot P(t=1 \mid x = c_i) + (1-\alpha) \cdot P(t=1)$

where,

$$\alpha = \frac{1}{1 + e^{-(\#(t=1)-1)}}$$

$\#(t=1)$ is frequency of t occuring 1

↳ For $B = C_2$,

(i) $P(t=1) = \frac{5}{7} = 0.714$

For B = 'c2', the count is 2, and the sum of class for B = 'c2' is 0

using smoothing parameter $\alpha = 1$

$$TE(B=c2') = \frac{0 + 1 \times 0.714}{2+1}$$

$$= \frac{0.714}{3}$$

$$= 0.238$$

Thus the smoothed target encoder value for B = 'c2' is 0.238

5) In slide p. 38 (Data preprocessing), explain why sigmoid function is used in target encoding smoothing? Can you suggest any other function for smoothing?

The sigmoid function is utilized in target encoding smoothing because it transforms values into a range between 0 and 1, which is beneficial when smoothing probabilities or target means. It gradually adjusts values based on confidence and ensures that extreme values are pushed closer to 0 or 1 without abrupt transitions. This helps to smooth out the target encoding by preventing overfitting to rare categories.

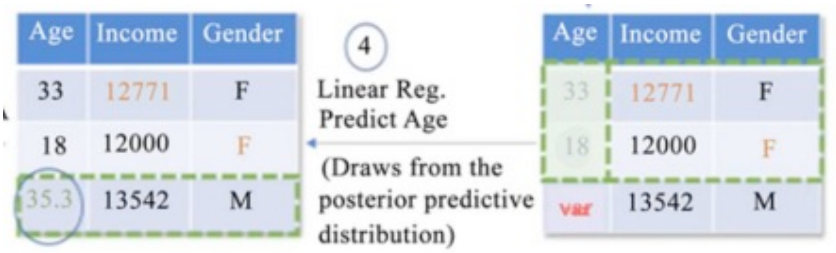
Sigmoid function formula:

$\sigma(x) = 1 / (1 + e^{(-x)})$

Alternative Smoothing Function:

Another commonly used function for smoothing is the logistic function, which is mathematically like the sigmoid but scales differently. Additionally, Laplace smoothing, or Beta distribution smoothing can be used, especially when handling sparse categorical data or in Bayesian modeling contexts. These techniques help handle categories with very few occurrences by adjusting the probabilities to avoid overfitting.

4. [3 pts] Explain the following process in p. 51 of slide (Data preprocessing)



In the image provided, we observe a process related to a regression model aimed at predicting "Age" using other features such as "Income" and "Gender." Let's break down the process step by step:

Initial Data:
The image displays a table with three columns:
Age: This is the target variable (dependent variable) we are attempting to predict.
Income: This is one of the independent variables (features) that will aid in predicting "Age."
Gender: Another independent variable (feature) that will contribute to predicting "Age."

Two entries have been provided, the third entry is partially filled where Age is missing.

Linear Regression Model:- The phrase "Linear Reg. Predict Age" indicates the use of a linear regression model to predict the missing value of "Age" based on the available data for "Income" and "Gender."

Prediction (Posterior Predictive Distribution):- The third row has a missing value for "Age" (indicated by

the dotted green box). The linear regression model utilizes the known values of "Income" (13542) and "Gender" (M) to predict the "Age." The model derives this prediction from a posterior predictive distribution, meaning that the model was trained using Bayesian techniques. This implies that the prediction is based on a distribution of possible values rather than a single deterministic output, allowing for some uncertainty in the prediction. In this case, the predicted value for "Age" is 35.3.

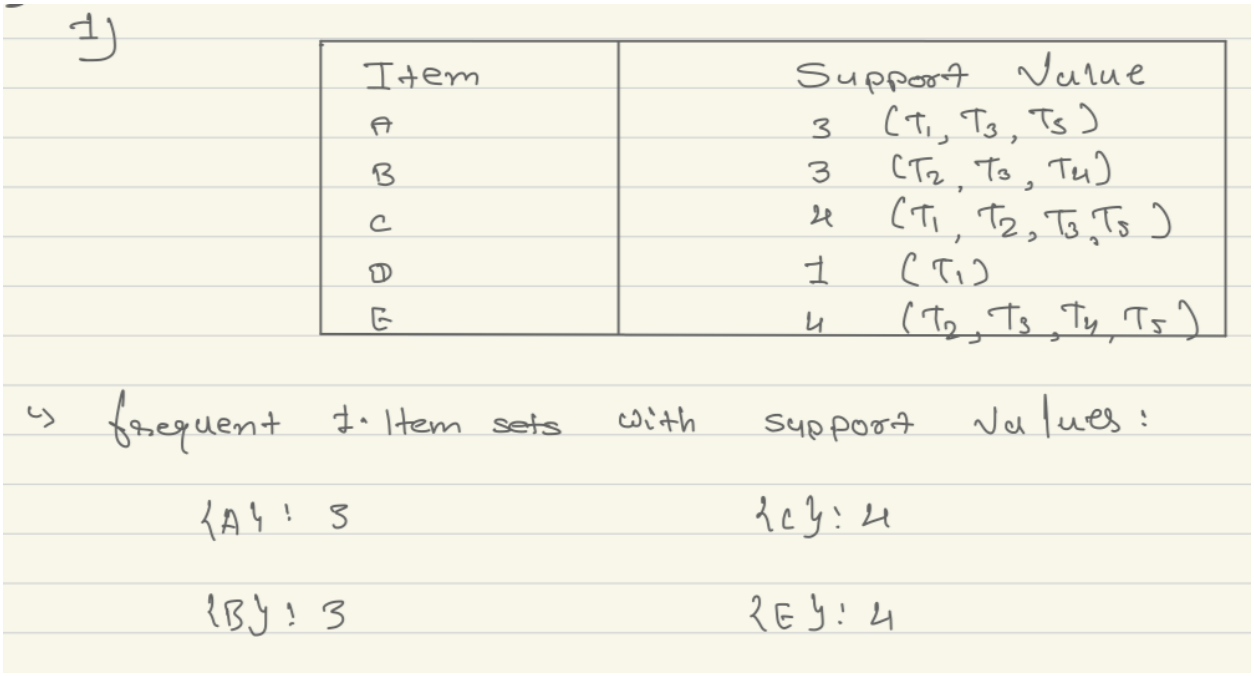
Final Table with Predictions- On the right side of the image, the missing "Age" value is replaced by the predicted value of 35.3, completing the dataset with no missing values for "Age."

5. [3 pts/ea] Using the following transaction database,

TID	Items
T1	A, C, D
T2	B, C, E
T3	A, B, C, E
T4	B, E
T5	A, C, E

(min support=3)

1) Show frequent 1-itemsets along with support values.



1)

Item	Support Value
A	3 (T_1, T_3, T_5)
B	3 (T_2, T_3, T_4)
C	4 (T_1, T_2, T_3, T_5)
D	1 (T_1)
E	4 (T_2, T_3, T_4, T_5)

↳ frequent 1-item sets with support values:

{A} : 3 {C} : 4

{B} : 3 {E} : 4

2) Show frequent 2-itemsets along with support values.

2)

Item set	Support Values
{A, B}	1 {T3}
{A, C}	3 {T1, T3, T5}
{A, E}	2 {T3, T5}
{B, C}	2 {T2, T3}
{B, E}	3 {T2, T3, T4}
{C, E}	3 {T2, T3, T5}

∴ frequent 2-itemsets with support values:

{A, C}: 3

{B, E}: 3

{C, E}: 3

3) Show frequent 3-itemsets along with support values.

3)

Item	Support Value
{A, B, C}	1 {T3}
{A, B, E}	1 {T3}
{A, C, E}	2 {T3, T5}
{B, C, E}	2 {T2, T3}

∴ No frequent 3 item sets meet the minimum support of 3

4) For an association rule {B,C} -> {E}, compute support, confidence, and lift values, respectively.

4)

Item	Support Value
{B, C, E}	2 {T2, T3}
{B, C}	2 {T2, T3}
{E}	4 {T2, T3, T4, T5}

↳ Support of {B, C} → {E}

$$\frac{\{B,C,E\}}{\{B,C\}} = \frac{2}{5} = 0.4$$

↳ confidence of {B, C} → {E}

$$\frac{\{B,C,E\}}{\{B,C\}} = \frac{2}{2} = 1$$

Lift of {B, C} → {E} = $\frac{\text{confidence}}{\text{support of } \{CE\}} = 1/(4/5) = 1.25$

5) With N items anf M transactions. What is the time complexity generating candidate itemsets (along with support values) using brute force method (without Apriori principle)

To generate itemsets using brute force, we need to consider all possible subsets of the N items. There are 2^N possible subsets, including the empty set. For each subset, we have to check its support by going through all M transactions. Therefore, the time complexity of generating candidate itemsets is O(M * 2^N).

6) [6 pts] Change transaction db to FP-Tree

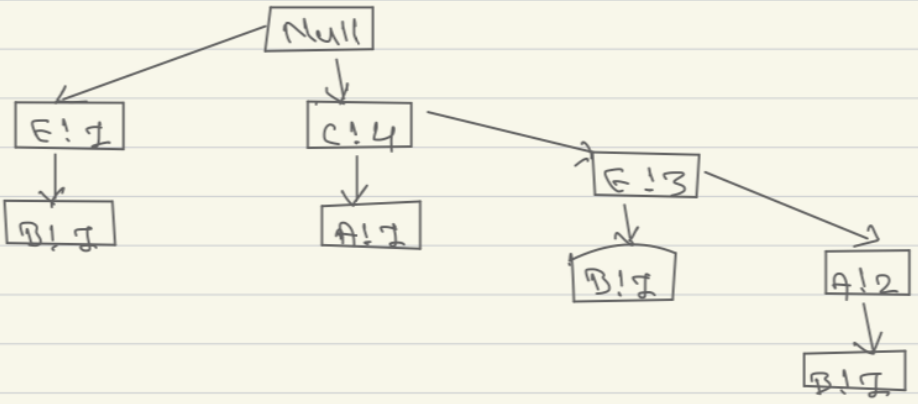
6)

Compute the frequency of each item

A:3
B:3
C:4
E:4
D:1 (since it's support is less than 3, we will ignore it)

Table is,

TId	Items	order Item
T_1	A, C, D	C, A
T_2	B, C, E	C, E, B
T_3	A, B, C, E	C, E, A, B
T_4	B, E	E, B
T_5	A, C, E	C, E, A



6. [3 pts/ea] (Refer to p. 28-29 in Association) We are going to apply association rules in classification learning.

1) To use association rules in classification learning, what form should the association rules take?

In order to use association rules for classification learning, the rules should be in the form of Antecedent → Consequent, where the antecedent (or left-hand side) comprises a combination of attributes (conditions) from the dataset, and the consequent (or right-hand side) is the class label.

For example, in your case, the rules would be something like:

(budget-resolution=n, MX-missile=n, el-salvador-aid=y) → republican

(crime=y, right-to-sue=y, physician-fee-freeze=y) → republican

In this context:
The antecedent (e.g., "budget-resolution=n, MX-missile=n") consists of a combination of features.
The consequent (e.g., "republican") represents the class label that we are predicting.

This rule structure allows the classification model to utilize the conditions from the antecedent to predict the corresponding class.

2) We want to assign different weights to each rule, and use weighted voting for classification. Provide your opinion on methods for determining the importance of rules.

Support and Confidence:
Support: Proportion of records satisfying both antecedent and consequent. Higher support means more frequent and important rule.
Confidence: Proportion of records satisfying antecedent and consequent. Higher confidence implies more reliable rule.

Lift:
Compares rule's confidence to expected confidence if antecedent and consequent were independent.
Lift > 1 suggests strong association.

Leverage:
Measures difference between observed and expected frequency of the rule. Higher leverage implies stronger relationships and higher rule importance.

Conviction:
Compares incorrect prediction frequency to expected incorrect prediction by chance. Higher conviction implies a stronger rule.

Rule Length:
Shorter rules are preferred for simplicity and less overfitting.

Coverage:
Proportion of records covered by the antecedent. Higher coverage indicates more important rules.

Weighted Voting:
Used for final prediction in classification. Each rule "votes" for its consequent, weighted by rule importance. Class with highest weighted votes is predicted.

7. [3 pts/ea] In p. 9 in Linear Regression, we use sum of squared error.
1) Explain why we use squared value.

In linear regression, we use the sum of squared errors (SSE) as the error function for the following reasons:

Mathematical convenience: Squaring the errors makes the error function differentiable and smooth. This allows us to use optimization techniques like gradient descent or analytical solutions (normal equations) to find the optimal parameters that minimize the error function efficiently.

Emphasis on larger errors: Squaring the residuals (errors) penalizes larger errors more heavily than smaller ones. This means that points that are farther from the regression line have a greater impact on the overall error, helping the model to adjust more precisely to reduce large deviations.

Uniqueness of the solution: Using squared errors results in a convex function concerning the model parameters, ensuring that there is a single global minimum.

2) If we use sum of absolute value, do we have any problem ?

When using the sum of absolute errors (SAE) instead of the sum of squared errors, several issues arise:

1. Non-differentiability: The absolute value function is not differentiable at zero. This makes optimization more challenging since methods like gradient descent require smooth, differentiable functions. While there are alternative optimization techniques (e.g., subgradient methods) that can handle non-differentiable points, they tend to be more computationally intensive and slower.

2. Less Sensitivity to Outliers: Using the absolute value does not penalize larger errors as strongly as the squared function. This could make the model less responsive to large deviations and less accurate in scenarios where large errors are critical to adjust.

3. Complex Optimization: The error function formed by absolute values is not a convex function, which means there might be multiple local minima. This makes finding the optimal solution more difficult compared to the squared error approach, which guarantees a single global minimum.

8. Given the error function of linear regression,

$$J(w) = \sum_i (y_i - \hat{y}_i)^2$$

1) [3 pts] If $\hat{y}_i = w_0 + w_1 X_1 + w_2 X_2 + \cdots + w_p X_p$, is it guaranteed to find global optimum? Explain the reason.

The error function J(w) = ∑i (yi - ŷi)^2, where ŷi = w0 + w1X1 + w2X2 + ... + wpXp, guarantees finding the global optimum in the case of linear regression. This is because the error function J(w) is a convex function with respect to the weights w0, w1, ..., wp, since it is a quadratic function in terms of the parameters. A quadratic function of this form is convex, meaning it has a single global minimum and no local minima or maxima. Therefore, optimization techniques such as gradient descent or analytical methods will always converge to the global minimum of the error function.

2) [3 pts] Suppose $\hat{y}_i = w_0 + w_1 X_1$. Show the derivative of $\frac{\partial J(w)}{\partial w_0}$ and $\frac{\partial J(w)}{\partial w_1}$, respectively

8)

2) To compute the derivatives of the cost function J(w) with respect to w_0 and w_1, we first need to define the cost function. Typically in linear regression, the cost function is the mean Squared error (MSE), define as!

Here, $\hat{y}_i = w_0 + w_1 x_1$, thus the error function is :

$$J(w) = \sum_i (y_i - (w_0 + w_1 x_1))^2$$

* Derivative with respect to w_0

$$\frac{\partial J(w)}{\partial w_0} = \frac{\partial}{\partial w_0} \sum_i (y_i - (w_0 + w_1 x_1))^2$$

using chain rule :

$$= \sum_i 2 (y_i - (w_0 + w_0 + w_1 x_1)) \cdot (-1)$$

$$\boxed{\frac{\partial J(w)}{\partial w_0} = -2 \sum_i (y_i - w_0 + w_1 x_1))}$$

* Derivative with respect to w_1

$$\frac{\partial J(w)}{\partial w_1} = \frac{\partial}{\partial w_1} \sum_i (y_i - (w_0 + w_1 x_1))^2$$

using chain rule:

$$= \sum_i 2(y_i - (w_0 + w_1 x_{i1}) \cdot (-x_{i2})$$

$$\boxed{\frac{\partial J(w)}{\partial w_1} = -2 \sum_i (y_i - (w_0 + w_1 x_{i1}) x_{i1}}$$

9. [5 pts] For data i, suppose y_i= target(label) and $\hat{y}_i$= prediction by linear regression. Suppose $\hat{y} = 1 + 3 * x_1$. Given a dataset D={(1,6), (3,8)} Compute residual error (e_i) for each data and total sum of squared errors.

9)

Calculate predictions $\hat{y}_i$ for each data point.

$\hat{y}_i = 1 + 3x_i$

$x_1 = 1$ and $y_1 = 6$

$\hat{y}_i = 1 + 3x_1 = 1 + 3(1) = 4$

∴ The residual error is:

$e_1 = y_1 - \hat{y}_1 = 6 - 4 = 2$

↳ for $x_1 = 3$ and $y_2 = 8$

$\hat{y}_2 = 1 + 3(3) = 1 + 9 = 10$

The residual error is

$e_2 = y_2 - \hat{y}_2 = 8 - 10 = -2$

The sum of squared error is the sum of squares of the residuals!

$SSE = e_1^2 + e_2^2$

$SSE = (2)^2 + (-2)^2 = 4 + 4 = 8$

10. (coding) Discretization

1) Implement equal width discretization

```
import numpy as np

#
# Suppose num_bins=3,
# if bins=[v1,v2,v3,v4],
# v1 < 1st interval(bin) <= v2
# v2 < 2nd interval(bin) <= v3
# v3 < 3rd interval(bin) <= v4

# Equal width discretization function
# hint: may use numpy's linspace
def equal_width(data, num_bins):
    '''
    YOUR WORK HERE [4 pts]
    '''
    return bins

# Continuous data
data = np.array([ your own numeric data ])

# Define the number of bins
num_bins = O

# compute intervals using equal width function
bins=equal_width(data, num_bins)

# Using bins, discretize the data by assigning it to bins
disc_data = np.digitize(data, bins, right=True)

# Display the results
# show the results of original data, bins, disc_data, etc
'''
YOUR WORK HERE [4 pts]
'''
```

2) Entropy-based discretization: The following program finds the best split point in a numeric data array.

```
# (input) y: array of target values
# (output) entropy: entropy value of y
# for simplicity. assume binary class only
# e.g., [0,0,1,0,0,1,1,1]
```

```
def calculate_entropy(y):
    '''
    YOUR WORK HERE [4 pts]
    '''
    return entropy

# (input) y: array of target values
#         y_left: left interval
#         y_right: right interval
# e.g., y=[1,2,3,4,5], y_left=[1,2], y_right=[3,4,5]
# (output) info_gain: information gain between entropy(y) and average entropy after split.
def information_gain(y, y_left, y_right):
    '''
    YOUR WORK HERE [4 pts]
    Implement the following
    # compute entropy of y
    # calculate entropy of y_left and y_right, respectively
    # compute information gain
    '''
    return info_gain


""" Find the best split point based on information gain. """
def best_split(X, y):
    '''
    YOUR WORK HERE [4 pts]
    Implement the following.
    #initialize best information gain value=-1 & best split point=None
    for value in np.unique(X):
        # y_left : interval with y values <=value
        # y_right: interval with y values > value
        # compute information gain using two intervals y_left & y_right
        # update best_info & best_split_point
    '''
    return best_split_point

'''
YOUR WORK HERE [4 pts]
data = np.array([ Your own data ])
show the results of running best_split, information gain, entropy values, etc
explain the correctness of best_split
'''
```

11. (coding) Apriori

```
trans_db=[ ['milk','bread','biscuit'], ['bread','milk','biscuit','cornflakes'],
    ['bread','tea','bournvita'], ['jam','maggi','bread','milk'], ['maggi','tea','biscuit'],
```

```
    ['bread','tea','bournvita'], ['maggi','tea','cornflakes'], ['maggi','bread','tea','biscuit'],
    ['jam','maggi','bread','tea'], ['bread','milk'], ['coffee','cock','biscuit','cornflakes'],
    ['coffee','cock','biscuit','cornflakes'], ['coffee','sugar','bournvita'], ['bread','coffee','cock'],
    ['bread','sugar','biscuit'], ['coffee','sugar','cornflakes'], ['bread','sugar','bournvita'],
    ['bread','coffee','sugar'], ['bread','coffee','sugar'], ['tea','milk','coffee','cornflakes'] ]

min_support=3

#
# infreq_list : the set of infrequent itemsets (itemsets whose support is less than min_support)
# 2D list structure. e.g., [ ['bread'], ['sugar'], ['coffee', bisciut'], ....]
infreq_itemsets=[]

# compute support value of an itemset
def compute_support(itemset):
    support=0
    for trans in trans_db:
        if set(itemset).intersection(set(trans)) == set(itemset):
            support += 1
    return support

#
# Generate k+1 frequent itemsets from k frequent itemsets
# (input) k_itemsets: k itemsets (itemset with length=k)
#             2D list (length k). e.g. (k=2), [ ['bread', 'sugar'], ['coffee', 'bisciut'], ....]
# (output) k_1_itemsets: (k+1) itemsets with length=k+1
#             2D list (length k+1). e.g. (length=3), [ ['bread', 'sugar', 'coffee'], ['coffee', 'bisciut', 'jam'], ....]
def generate_k_1_itemsets(k_itemsets):
    '''
    YOUR WORK HERE [12 pts]
    Implement the following

    # In k_itemsets, remove itemsets whose support < min_support
    # k_i_itemsets: frequent (k_+1) itemsets (support >= min_support)
    k_1_itemsets=[]
    for i in k_itemsets:
        for j in k_itemsets:
            # create 'new_itemset' by combining i and J
            # skip 'new_itemset' if
            # 1) i == j or
            # 2) 'new_itemset' is in in 'infreq_itemsets' or
            # 3) length is not (k+1) or
            # 4) support of 'new_itemset' < min_support
                  (in this case, add 'new_itemset' to infreq_itemsets)
     '''
    return k_1_itemsets
```

```
# original 1-items (wihout considering min_support values)
all_list_items = list(set(i for j in trans_db for i in j))

# frequent 1-itemsets
itemsets_1=[]

'''
YOUR WORK HERE
Create frequent 1-itemsets using all_list_items (length=1 & support >= min_support) [4 pts]
(within all_list_items, remove items whose support < min_support)
'''
'''
YOUR WORK HERE
Show frequent 2-itemsets and frequent 3-itemsets, respectively. [4 pts]
Verify whether the results are correct
'''
```

12. (extra 5 pts) Discretization using sklearn

```
from sklearn.datasets import load_iris
from sklearn.preprocessing import KBinsDiscretizer

dataset = load_iris()

response = dataset.target
feature_names = dataset.feature_names

'''
YOUR WORK HERE
Discretize features in Iris data using equal width and equal frequency discretization, respectively.
'''
```

13. (extra 5 pts) Apriori using library

```
'''
YOUR WORK HERE
Refer to p. 31-32 (Association).
Using the trans_db in Q. 11, generate all frequent itemsets and association rules.
'''
```

Due: 9/22(Sun) 23:59