Northeastern University, Khoury College of Computer Science

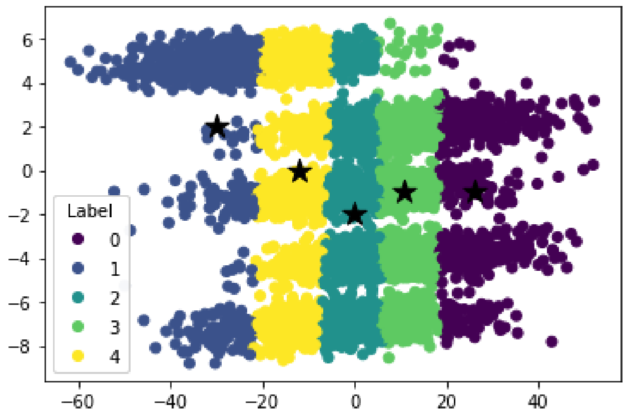
CS 6220 Data Mining | Assignment 4

Yuanxun Qin
timothyq
qin.yuan@northeastern.edu

Question 1
1a. Implementation

```
class MyKMeans:
    def __init__(self, n_clusters=2, max_iter=100):
        self.n_clusters = n_clusters
        self.max_iter = max_iter

    def fit(self, X, init_centroids=None):
        if init_centroids is None:
            # randomly initialize cluster centroids
            self.centroids = X[np.random.choice(X.shape[0],
self.n_clusters, replace=False)]
        else:
            # use specified initial centroids
            self.centroids = init_centroids

        for i in range(self.max_iter):
            # assign each data point to the nearest cluster
            distances = np.sqrt(((X - self.centroids[:,
np.newaxis])**2).sum(axis=2))
            # print((X - self.centroids[:, np.newaxis]).shape)
            # print(X.shape)
            labels = np.argmin(distances, axis=0)

            # update cluster centroids to the mean of the assigned data
points
            for j in range(self.n_clusters):
                self.centroids[j] = X[labels == j].mean(axis=0)

    def predict(self, X):
        # assign each data point to the nearest cluster
        distances = np.sqrt(((X - self.centroids[:,
np.newaxis])**2).sum(axis=2))
        labels = np.argmin(distances, axis=0)
        return labels
```

1b. Scatter the results(codes see the .ipynb)

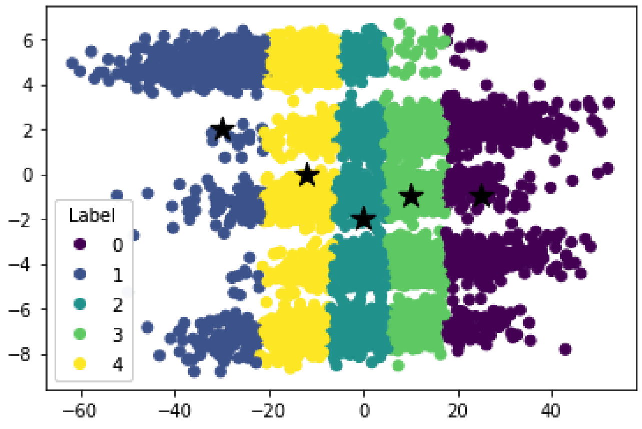


1c. You will notice that in the above, there are only five initialization clusters. Why is k = 5 a logical choice for this dataset? After plotting your resulting clusters, what do you notice? Did it cluster very well? Is there an initialization that would make it cluster well?

Answer: There are several methods for selecting k, such as the elbow method or silhouette method, which rely on evaluating the clustering performance for different values of k.

In this case, as the data is only two dimentional, we can easily observe the pattern and choose k = 5 by intuition. The cluster is not as excepted, or not very well.

No initialization can help as the it will alway converge to similiar result.
Use sets of initial centroids which scatter across the Y coordinate, and limit the iteration as only 10. We can still see the similiar result:

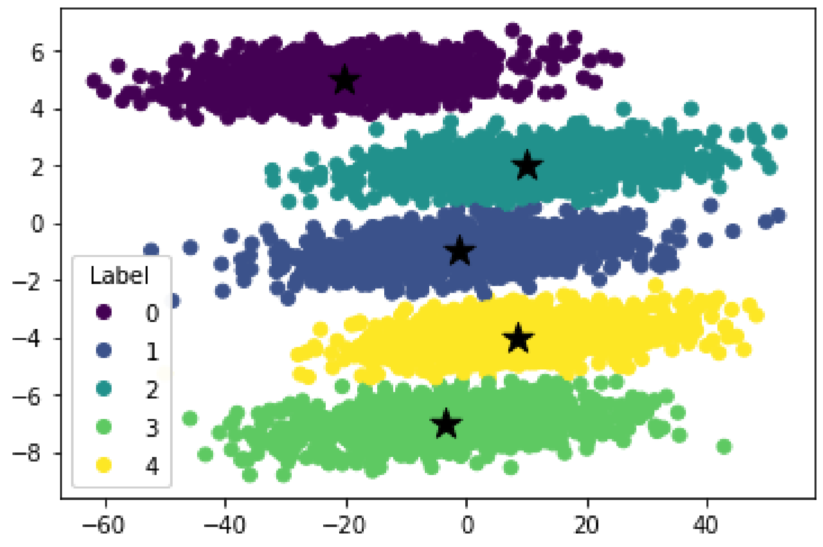


2. Mahalanobis Distance

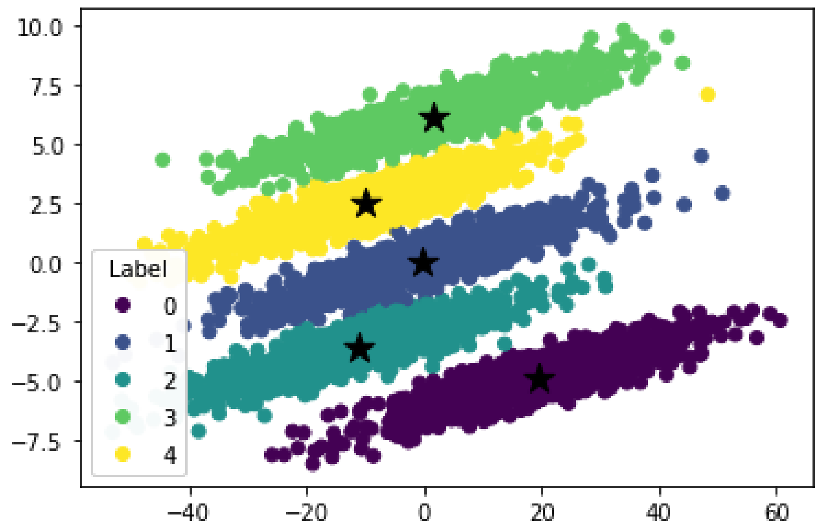
2a. Using the same data as Question 1 and the same initialization instances { x1 , x2 , x3 , x4 , x5 } implement a specialized k-means with the above Mahalanobis Distance. Scatter the results with the different clusters as different colors.
What do you notice? You may want to pre-compute $P-1$ so that you aren't calculating an inverse every single loop of the the k-Means algorithm.

Answer: The result is much better than the 1. K-Mean solution

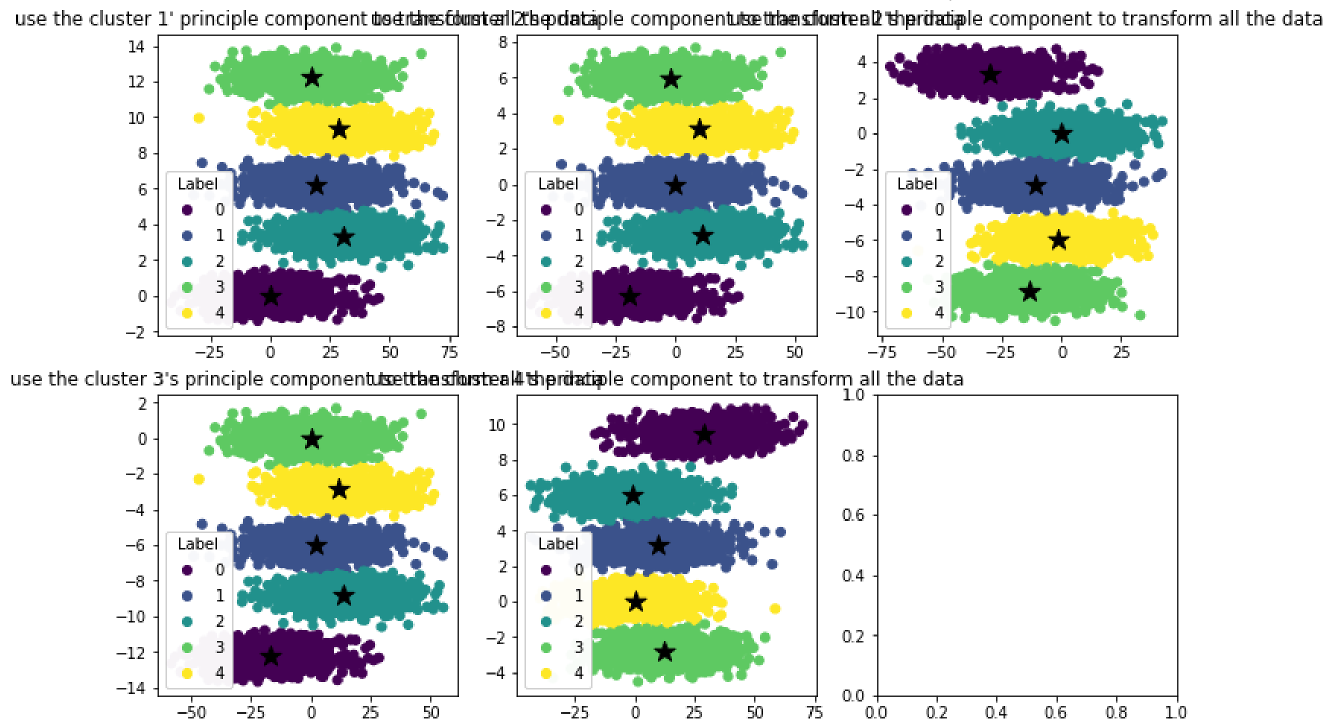
```
P = np.array([[10, 0.5], [-10,0.25]])
R = np.linalg.inv(P.T @ P)
R
output:
array([[ 0.00555556, -0.04444444], [-0.04444444, 3.55555556]])
```



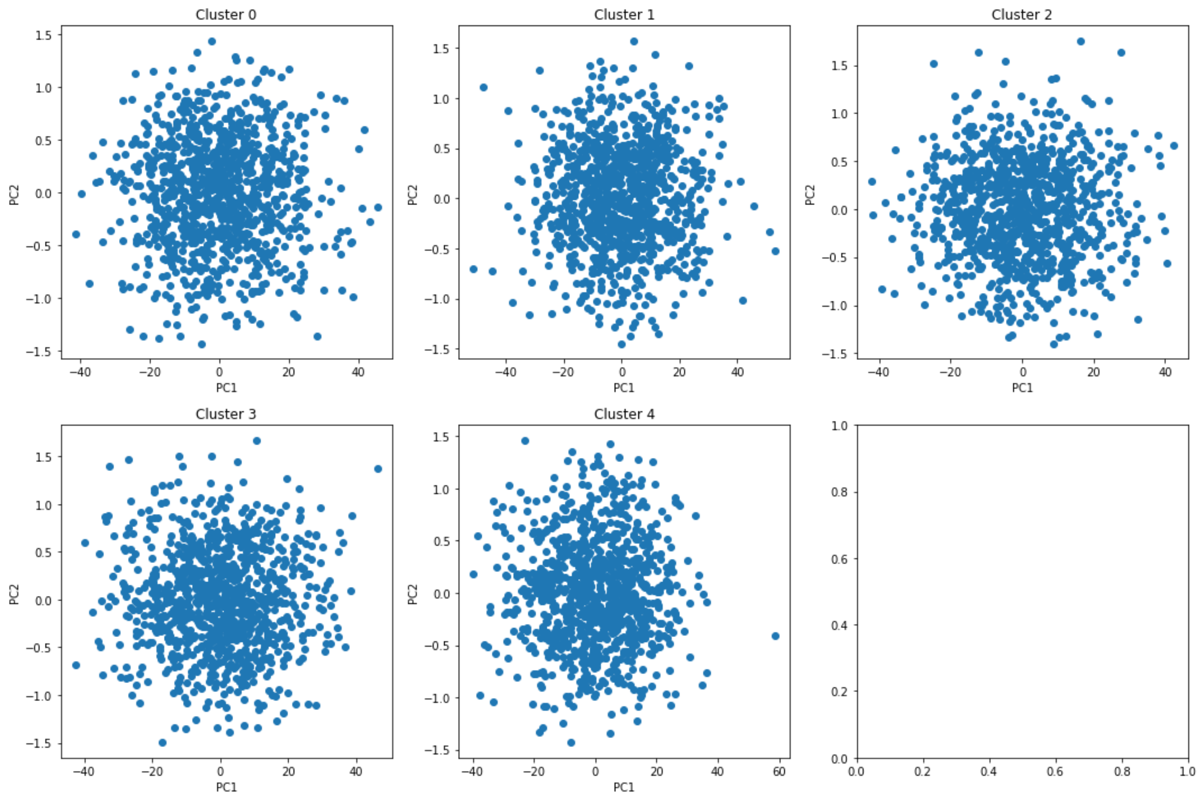
2b. Calculate and print out the principle components of the aggregate data.



Use individual’s PCA transformation matrix to transform all the data will yield:



Split each cluster and perform PCA on them and plot them respectively:



2d.)

```
P = np.array([[10, 0.5], [-10, 0.25]])
eigenvalues, eigenvectors = np.linalg.eig(P)
D = np.diag(eigenvalues)
V = eigenvectors
P_n = V @ D
P_n
```

output:

```
array([[ 6.4054879 , -0.04300577], [-6.95724558, 0.79190224]])
```

Given a covariance matrix C and its eigenvector matrix V, the product λ * v, where λ is an eigenvalue and v is its corresponding eigenvector, represents the direction and magnitude of the maximum variance of the data along that eigenvector.

new_data_transformed = (new_data - means) @ C

When we multiply each principal component by its corresponding eigenvalue λ, we obtain the amount of variance that is captured by that principal component.

original_data = transformed_data @ C.T + means

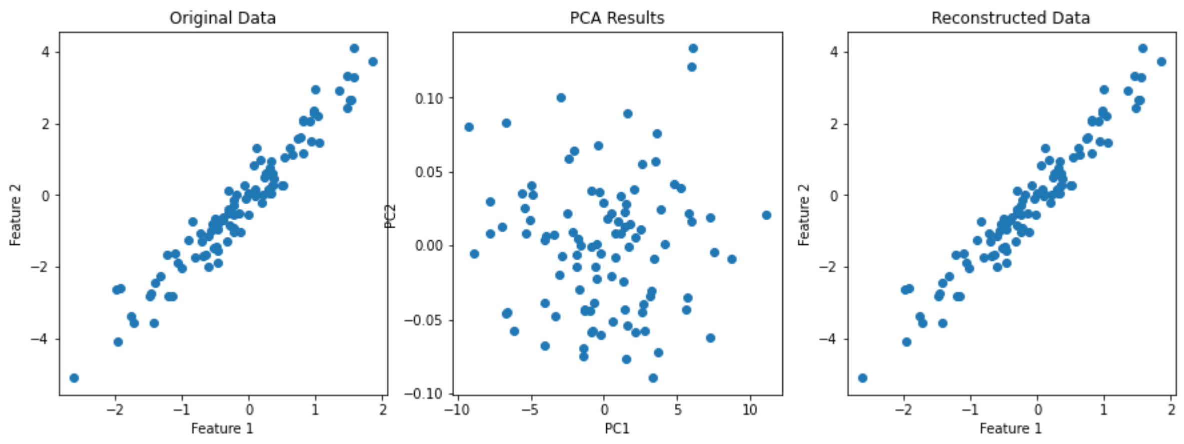
Therefore, the relationship between λ * v and the original data is that λ * v represents the direction and amount of variance captured by the corresponding principal component of the original data.

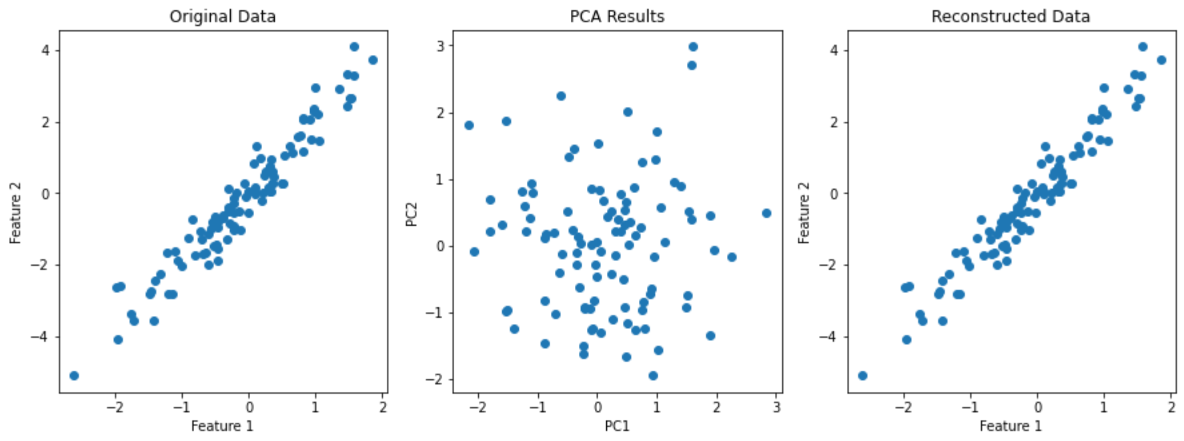
The transformation matrix represents a linear transformation that scales the data along two directions, defined by its eigenvectors, and the amount of scaling is determined by its eigenvalues. This transformation can be used to transform the data into a new coordinate

system where the axes correspond to the eigenvectors of the transformation matrix, and the variance of the data along each axis is given by the corresponding eigenvalue. This can be useful in data analysis, as it allows for easier visualization and interpretation of the data, as well as for dimensionality reduction and feature selection.
P_n(V@D) performs a change of basis to a coordinate system where the transformation performed by the matrix is diagonal. This means that in the new coordinate system, the matrix performs a scaling operation along each axis, where the scale factors are given by the diagonal elements of D.
Assuming that the data is arranged in a matrix X, where each row represents a sample and each column represents a feature, we can use P_n to perform a linear transformation of the data into a new coordinate system where the dimensions (i.e., features) are aligned with the eigenvectors of X.T @ X. Specifically, if we let Y = X @ P_n, then the columns of Y will represent the data transformed into the new coordinate system, where the first column corresponds to the direction of highest variance in the data, the second column corresponds to the direction of second-highest variance, and so on.
Result(codes in .ipynb):

```
Original matrix P:
 [[ 10.     0.5 ]
 [-10.     0.25]]
Matrix with orthonormal eigenvectors:
 [[ 1.00000000e+00 -8.15985884e-18]
 [-8.42756479e-17  1.00000000e+00]]
Diagonal matrix of eigenvalues:
 [[1. 0.]
 [0. 1.]]
Eigenvectors:
 [[ 0.29711317 -0.99959179]
 [ 0.95484227 -0.02857002]]
Transformed data:
 [[-3.701254   -4.40373262 -5.10621124]
 [ 0.84056218  1.76683443  2.69310668]]
Reconstructed data:
 [[1. 2. 3.]
 [4. 5. 6.]]
```

However, as the relationship between P and data is not a speccify, and the P = np.array([[10, 0.5], [-10, 0.25]]) doesn't have the propoties that could make the reconstruction simplified(eg. normalized matrix such that its eigenvecs are orthogonal then V^-1 == V.T). It doesn't make too much sense for the D @ V.
However, if we use the coverience matrix of the data as P, then we can perform PCA on the data by multiple V * sqrt(D) then we can get the normalized PCA.





3. Market Basket Analysis and Algorithms

3a.)

To obtain candidate 4-itemsets using the F k−1 × F 1 merging strategy, we need to merge each frequent k-1 itemset with a frequent 1-itemset that is not already in the k-1 itemset. In this case, since we are looking for candidate 4-itemsets, k-1 = 3. Therefore, we need to merge each frequent 3-itemset in F3 with a frequent 1-itemset that is not already in the 3-itemset.

The frequent 1-itemsets are {1}, {2}, {3}, {4}, and {5}. We can merge each of these with each of the frequent 3-itemsets in F3 to obtain the following candidate 4-itemsets: {1, 2, 3, 4} {1, 2, 3, 5} {1, 2, 4, 5} {1, 3, 4, 5} {2, 3, 4, 5} Therefore, the candidate 4-itemsets obtained using the F k−1 × F 1 merging strategy are: {1, 2, 3, 4}, {1, 2, 3, 5}, {1, 2, 4, 5}, {1, 3, 4, 5}, {2, 3, 4, 5}

3b.)

To obtain candidate 4-itemsets using the F k−1 × F k−1 merging strategy in Apriori, we need to merge each frequent k-1 itemset with another frequent k-1 itemset that is not already in the k-1 itemset.

In this case, since we are looking for candidate 4-itemsets, k-1 = 3. Therefore, we need to merge each frequent 3-itemset in F3 with another frequent 3-itemset in F3 that is not already in the 3-itemset.
In Apriori Algorithm, when merging Fk−1 × Fk−1 into Fk, only pairs that share the first k-2 items are considered. This is because the Apriori property states that any subset of a frequent itemset must also be frequent. Therefore, when merging Fk−1 × Fk−1 into Fk, only those pairs that share the first k-2 items can potentially form a frequent itemset of size k. This is because if a pair of itemsets does not share the first k-2 items, then at least one of them will contain an item that is not frequent in the dataset, and therefore cannot be a part of a frequent itemset of size k. By considering only pairs that share the first k-2 items, the Apriori algorithm avoids generating and counting many candidate itemsets that are guaranteed to be infrequent. This helps to reduce the search space and improve the efficiency of the algorithm.

The frequent 3-itemsets in F3 are {1, 2, 3}, {1, 2, 4}, {1, 2, 5}, {1, 3, 4}, {2, 3, 4}, {2, 3, 5}, and {3, 4, 5}.
We can merge each of these with each of the other frequent 3-itemsets in F3 that are not already in the 3-itemset to obtain the following candidate itemsets:
{1,2,3}x{1,2,4} : {1,2,3,4}
{1,2,3}x{1,2,5} : {1,2,3,5}
{1,2,4}x{1,2,5} : {1,2,4,5}
{2,3,4}x{2,3,5} : {2,3,4,5}
Therefore, the candidate 4-itemsets obtained using the F k−1 × F k−1 merging strategy in Apriori are:
{1, 2, 3, 4}, {1, 2, 3, 5}, {1, 2, 4, 5}, {2, 3, 4, 5}

3c.)
For each candidate 4-itemset, we need to check whether all of its subsets of size 3 are frequent. If a candidate 4-itemset has at least one subset of size 3 that is not frequent, then it cannot be a frequent itemset and is pruned.
The infrequent 3-itemsets are:
{1, 3, 5}, {2, 4, 5}

The candidate 4-itemsets are:
{1, 2, 3, 4} Subsets of size 3: {1, 2, 3}, {1, 2, 4}, {1, 3, 4}, {2, 3, 4}. All of its subsets of size 3 are frequent, so it survives.

{1, 2, 3, 5} Subsets of size 3: {1, 2, 3}, {1, 2, 5}, {1, 3, 5}, {2, 3, 5}. {1, 3, 5} is not frequent, so it is pruned.

{1, 2, 4, 5} Subsets of size 3: {1, 2, 4}, {1, 2, 5}, {1, 4, 5}, {2, 4, 5}. All of its subsets of size 3 are frequent, so it survives.

{2, 3, 4, 5} Subsets of size 3: {2, 3, 4}, {2, 3, 5}, {2, 4, 5}, {3, 4, 5}. {2, 4, 5} is not frequent, so it is pruned.

Therefore, the candidate 4-itemsets that survive the candidate pruning step of the Apriori algorithm are:

{1, 2, 3, 4}

4

4a.)

$$\sum_{k=1}^{r-1}\left[\binom{r}{k}\times\sum_{j=1}^{r-k}\binom{r-k}{j}\right]=3^r-2^{r+1}+1$$

the maximum number of association rules that can be extracted from the given data (including rules that have zero support) is:

3^7 - 2^{7+1} + 1 = 2187 - 256 + 1 = 1932.

Therefore, there are 1932 possible association rules that can be generated from the given data.

4b.)

confidence({ Milk, Diapers } ⇒ { Butter }) = support({ Milk, Diapers, Butter }) / support({ Milk, Diapers })

We need to find the support of both the antecedent and the consequent of the rule.

The support of { Milk, Diapers } is the number of transactions that contain both Milk and Diapers, which is 4 in this case (transactions 2, 3, 5, and 7).

The support of { Milk, Diapers, Butter } is the number of transactions that contain all three items, which is 2 (transactions 2 and 7).

Therefore, the confidence of the rule { Milk, Diapers } ⇒ { Butter } is:

confidence({ Milk, Diapers } ⇒ { Butter }) = support({ Milk, Diapers, Butter }) / support({ Milk, Diapers })

= 2 / 4

= 0.5

So the confidence of the rule is 0.5 or 50%. This means that out of all transactions that contain both Milk and Diapers, 50% also contain Butter.

4c.) What is the support for the rule { Milk, Diapers } ⇒ { Butter }?
the support for { Milk, Diapers, Butter } is 2 out of 10 transactions, or 0.2 (or 20%).

4d.)
True

4e.)
False.

Because even though { a,b,c } is a candidate itemset, it may not necessarily be a frequent itemset. It depends on the actual support count of the itemset in the dataset.

4f.)
False. If the {a,b} do not contain c and the {b, c} do not contain a, then the {b} will be at least 50.

4g.)
False.

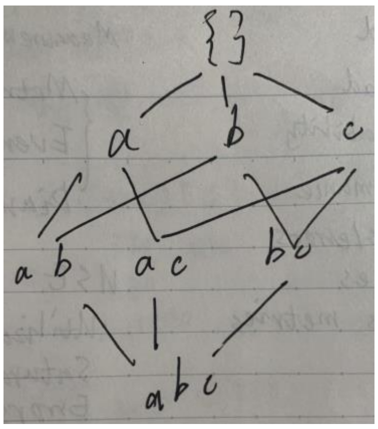
The maximum number of size-2 frequent itemsets that can be extracted from a dataset with 5 items is actually 10.

Combination formula:
C(n, r) = n! / r!(n-r)!

So, the number of size-2 frequent item sets is:

C(n, r) = 5C2 = 5! / 2!(5-2)! = 10

4h.)


{}
a
b
c
ab
ac
bc
abc