Submit response?

Your username (**domyglad14@gmail.com**) and responses will be recorded when you submit this form.

SWITCH ACCOUNT

Makeup Quiz

domyglad14@gmail.com Switch account

The name, email, and photo associated with your Google account will be recorded when you upload files and submit this form

* Indicates required question

Email *

- [x] Record **domyglad14@gmail.com** as the email to be included with my response

Student ID *

Mit/ur124191/09

1.
In the lecture notes it is discussed that K-means is sensitive to initialization. Provide an illustrative example to support this claim. [5 pts] *

Certainly! K-means is sensitive to initialization because the algorithm's outcome can vary depending on the initial placement of the cluster centroids. To illustrate this, let's consider a simple example with two well-separated clusters. We'll use a two-dimensional space for simplicity.

Suppose we have the following data points:

Cluster 1: (2, 3), (2, 4), (3, 3), (3, 4)
Cluster 2: (8, 7), (8, 8), (9, 7), (9, 8)

Scenario 1: Good Initialization
Let's say we initialize the centroids close to the true cluster centers:

Initial Centroid 1: (3, 3)
Initial Centroid 2: (8, 8)
In this case, K-means is likely to converge to the correct clusters:

Iteration 1: Centroids move towards the cluster centers.
Iteration 2: Convergence.
Scenario 2: Bad Initialization
Now, let's consider a bad initialization:

Initial Centroid 1: (6, 6)
Initial Centroid 2: (4, 4)
In this case, K-means may converge to a suboptimal solution:

Iteration 1: Centroids move towards the center between the true clusters.
Iteration 2: Convergence, but the centroids may not reach the true centers.
As a result, K-means might assign some points from Cluster 1 to Cluster 2 and vice versa, leading to a less accurate clustering.

This sensitivity to initialization is one reason why K-means is often run multiple times with different initializations, and the solution with the lowest overall error is chosen. Techniques like K-means++ initialization can also help mitigate this sensitivity by providing a smarter way to initialize centroids.

2. *

Discuss how you would use the Silhouette coefficient for determining the value of parameter k for the k-means algorithm. [5 pts]

The Silhouette coefficient is a metric used to calculate the goodness of a clustering technique, such as the K-means algorithm. It measures how well-separated the clusters are. The Silhouette coefficient for a data point quantifies how similar it is to its own cluster (cohesion) compared to other clusters (separation). The coefficient ranges from -1 to 1, where a high value indicates well-defined clusters.

To determine the optimal value of the parameter k (the number of clusters) for the K-means algorithm using the Silhouette coefficient, you can follow these steps:

1. Choose a range of values for k:
Start by defining a range of possible values for the number of clusters (k). For example, you might consider values from 2 to a certain maximum number, depending on your problem domain.
2. Apply K-means for each value of k:
Run the K-means algorithm for each value of k in the chosen range. For each run, compute the Silhouette coefficient for the resulting clustering.
3. Calculate Silhouette coefficients:
For each value of k, calculate the average Silhouette coefficient over all data points. The formula for the Silhouette coefficient for a single data point i is given by:
$S(i)= b(i)-a(i)/\max\{a(i),b(i)\}$
where
$a(i)$ is the average distance from the
i-th point to the other points in the same cluster, and
$b(i)$ is the smallest average distance from the
i-th point to points in a different cluster. The Silhouette coefficient for a clustering is the average of $S(i)$ for all data points.
4. Choose the value of k with the highest Silhouette coefficient:
Select the value of k that maximizes the average Silhouette coefficient. A higher Silhouette coefficient indicates a better-defined clustering structure.

Here's a Python-like pseudocode example using scikit-learn:

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# X is your data

# Define a range of k values
k_values = range(2, max_k + 1)

# Initialize a list to store silhouette scores
silhouette_scores = []

# Iterate over k values
for k in k_values:
    # Fit K-means model
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)

    # Get cluster labels
```

```
    # Get cluster labels
    labels = kmeans.labels_

    # Calculate silhouette score
    silhouette_avg = silhouette_score(X, labels)

    # Append the score to the list
    silhouette_scores.append(silhouette_avg)

# Find the optimal k
optimal_k = k_values[silhouette_scores.index(max(silhouette_scores))]

print("Optimal number of clusters (k):", optimal_k)
```

This pseudocode uses the silhouette_score function from scikit-learn to compute the average silhouette coefficient for each value of k. The optimal number of clusters is the one that corresponds to the highest silhouette score.

*

Download Mammographic Mass Data set from Machine Learning Repository. Missing values are marked as "?" in the data.

http://archive.ics.uci.edu/ml/machine-learning-databases/mammographic-masses/mammographic_masses.data

From questions 3 up to 6 submit your Python code in a single file.

3. Find the dimension of the data (number of rows and columns). [2 pts]

4. Write a function that counts number of missing values in each column. [2 pts]

5. Create a new data set from the previous data that has no missing values. [3 pts]

6. Write a function to compute Mean of columns in the dataset and test this function on new data set created in 5. [3 pts]

Mmammographi...

Submit Clear form