Alex Friedrichsen

Homework 9 Data Mining 395

April 17, 2023

Problem 1

(Comparing Clustering Algorithms)

a) Compare and contrast the performance and suitability of three popular clustering algorithms: K-

Means Clustering, Hierarchical Clustering, and Density-Based Spatial Clustering of Applications with

Noise (DBSCAN). In your report, compare the three algorithms in terms of the following computational aspects:

- Time Complexity: Discuss the time complexity of each algorithm and how it might affect the scalability of the algorithm when dealing with large datasets.
- Robustness: Explain how each algorithm is robust to outliers or noise in the dataset and how it handles such data points.
- Cluster Shape and Size: Describe how each algorithm handles clusters of different shapes and sizes and discuss which algorithm might be more suitable for datasets with irregularly shaped clusters.

Answer Problem 1

Clustering is a an approach to analyzing a data set that involves grouping data points based on their characteristics. It is a type of unsupervised machine learning. It can help find patterns or groups in the data.

The time complexity of each algorithm is dependent on n, the number of data points.

- K-means time complexity: $O(nki)$, where k is the number of clusters, and I is the number of iterations. Worst case: exponential, and therefore risky for large data sets.
- Hierarchical clustering: $O(n^2)$. Slowest.

The robustness is the algorithm’s ability to handle noisy data.

- K-means is not very robust as it tends to assign outliers to the nearest cluster, despite the large gap.
- Hierarchical clustering is relatively robust due to its basis in distance metrics that can deal with noise. As the noise grows to great, though, it will eventually cause incorrect clustering.
- DBSCAN is highly robust to outliers because it does not cluster data points that shouldn’t belong to a specific cluster. Uses a density based approach to identify clusters. Requires minimum number of data points to form a cluster.

Cluster shape and cluster size refer to the distribution of data points within clusters.

- K-means assume spherical, equal size clusters. It is unsuitable for data sets with irregularly shaped clusters. It also requires the choice of centroids, a central data point for each cluster.
- Hierarchical clustering can make clusters of varying shapes and sizes due to the process of building its tree based structure. This allows it to capture more complex relationships.
- DBSCAN can handle varying shapes and sizes of clusters as well, due to the density-based approach.

Problem 2:

Compare and contrast the performance and suitability of three popular clustering algorithms: K-Means Clustering, Hierarchical Clustering, and Density-Based Spatial Clustering of Applications with Noise (DBSCAN). In your report, compare the three algorithms in terms of the following computational aspects: Time Complexity: Discuss the time complexity of each algorithm and how it might affect the scalability of the algorithm when dealing with large datasets. Robustness: Explain how each algorithm is robust to outliers or noise in the dataset and how it handles such data points. Cluster Shape and Size: Describe how each algorithm handles clusters of different shapes and sizes and discuss which algorithm might be more suitable for datasets with irregularly shaped clusters.

```
from sklearn.datasets import make_moons
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.metrics import silhouette_score, calinski_harabasz_score, davies_bouldin_sc
import time
import os
import psutil
import numpy as np
import pandas as pd
```

```
import pandas as pd


```

```
# Define a function to perform hierarchical clustering and save the evaluation scores ar
def hierarchical_clustering(X, filename):
    start_time = time.time()
    process = psutil.Process(os.getpid())
    hc = AgglomerativeClustering(n_clusters=2).fit(X)
    end_time = time.time()
    execution_time = end_time - start_time
    memory_usage = process.memory_info().rss / 1024 / 1024 # convert from bytes to MB
    silhouette = silhouette_score(X, hc.labels_)
    calinski = calinski_harabasz_score(X, hc.labels_)
    davies = davies_bouldin_score(X, hc.labels_)
    with open(filename, 'a') as f:
        f.write("Hierarchical Clustering Results:\n")
        f.write("Silhouette Score: {}\n".format(silhouette))
        f.write("Calinski-Harabasz Index: {}\n".format(calinski))
        f.write("Davies-Bouldin Index: {}\n".format(davies))
        f.write("Execution Time: {} seconds\n".format(execution_time))
        f.write("Memory Usage: {} MB\n".format(memory_usage))
```

```
# Define a function to perform k-means clustering and save the evaluation scores and exe
def kmeans_clustering(X, filename):
    start_time = time.time()
    process = psutil.Process(os.getpid())
    kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
    end_time = time.time()
    execution_time = end_time - start_time
    memory_usage = process.memory_info().rss / 1024 / 1024 # convert from bytes to MB
    silhouette = silhouette_score(X, kmeans.labels_)
    calinski = calinski_harabasz_score(X, kmeans.labels_)
    davies = davies_bouldin_score(X, kmeans.labels_)
    with open(filename, 'w') as f:
        f.write("K-Means Clustering Results:\n")
        f.write("Silhouette Score: {}\n".format(silhouette))
        f.write("Calinski-Harabasz Index: {}\n".format(calinski))
        f.write("Davies-Bouldin Index: {}\n".format(davies))
        f.write("Execution Time: {} seconds\n".format(execution_time))
        f.write("Memory Usage: {} MB\n".format(memory_usage))
```

```
def dbscan_clustering(X, filename):
    start_time = time.time()
    process = psutil.Process(os.getpid())
    n_samples = X.shape[0]
    valid_labels = False
    while not valid_labels:
        eps = np.random.uniform(0.1, 0.5) # adjust range as necessary
```

```
        min_samples = np.random.randint(2, int(n_samples / 2)) # adjust range as necessa
        try:
            dbscan = DBSCAN(eps=eps, min_samples=min_samples).fit(X)
            silhouette = silhouette_score(X, dbscan.labels_)
            calinski = calinski_harabasz_score(X, dbscan.labels_)
            davies = davies_bouldin_score(X, dbscan.labels_)
            valid_labels = len(set(dbscan.labels_)) > 1
        except:
            pass
    end_time = time.time()
    execution_time = end_time - start_time
    memory_usage = process.memory_info().rss / 1024 / 1024 # convert from bytes to MB
    with open(filename, 'a') as f:
        f.write("DBSCAN Clustering Results:\n")
        f.write("Eps: {}\n".format(eps))
        f.write("Min Samples: {}\n".format(min_samples))
        f.write("Silhouette Score: {}\n".format(silhouette))
        f.write("Calinski-Harabasz Index: {}\n".format(calinski))
        f.write("Davies-Bouldin Index: {}\n".format(davies))
        f.write("Execution Time: {} seconds\n".format(execution_time))
        f.write("Memory Usage: {} MB\n".format(memory_usage))

```

```
def clustering_pipeline(X, filename):
    # Perform K-Means clustering
    kmeans_clustering(X, filename)

    # Perform Hierarchical clustering
    hierarchical_clustering(X, filename)

    # Perform DBSCAN clustering
    dbscan_clustering(X, filename)

```

```

# # Load the make_moons data set
# X, y = make_moons(n_samples=1000, noise=0.05, random_state=0)


# clustering_pipeline(X, "make_moons_results.txt")

```

```
# # Load the smartphone data set
# # Load data from CSV file
# data = pd.read_csv("smartphone/train.csv")


# # Extract X and y data
# X = data.iloc[:, :-1].values # assuming last column is the target variable
```

```
# X = data.iloc[:, :-1].values # assuming last column is the target variable
# y = data.iloc[:, -1].values
# clustering_pipeline(X, "smartphone_results.txt")
```

```
def find_optimal_dbscan_params(X, eps_range, min_samples_range):
    best_silhouette = -1
    best_eps = 0
    best_min_samples = 0

    for eps in eps_range:
        for min_samples in min_samples_range:
            dbscan = DBSCAN(eps=eps, min_samples=min_samples)
            labels = dbscan.fit_predict(X)
            if len(set(labels)) > 1:
                silhouette = silhouette_score(X, labels)
                if silhouette > best_silhouette:
                    best_silhouette = silhouette
                    best_eps = eps
                    best_min_samples = min_samples

    return best_eps, best_min_samples, best_silhouette
```

```
# Load the dataset from file
data = pd.read_csv("shuttle-landing-control.data", header=None)

# Replace asterisks with NaN
data = data.replace("*", np.nan)

# Perform one-hot encoding on categorical variables
data = pd.get_dummies(data, columns=[0, 1, 2, 3, 4, 5])

# Drop any rows with NaN values
data = data.dropna()

# Convert data to a NumPy array
X = data.to_numpy()


eps_range = np.arange(0.1, 1.0, 0.1)
min_samples_range = range(1, 10)

# Find optimal values for eps and min_samples
best_eps, best_min_samples, best_silhouette = find_optimal_dbscan_params(X, eps_range, r


# Print the optimal hyperparameters and clustering quality
print("Optimal eps:", best_eps)
print("Optimal min_samples:", best_min_samples)
print("Best silhouette score:", best_silhouette)
# clustering_pipeline(X, 'shuttle_landing_results.txt')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
c:\Users\alexp\Documents\GitHub\cs395_data_mining\HW9.ipynb Cell 10 in <module>
     <a href='vscode-notebook-cell:/c%3A/Users/alexp/Documents/GitHub
/cs395_data_mining/HW9.ipynb#X13sZmlsZQ%3D%3D?line=17'>18</a> min_samples_range =
range(1, 10)
     <a href='vscode-notebook-cell:/c%3A/Users/alexp/Documents/GitHub
/cs395_data_mining/HW9.ipynb#X13sZmlsZQ%3D%3D?line=19'>20</a> # Find optimal values
for eps and min_samples
---> <a href='vscode-notebook-cell:/c%3A/Users/alexp/Documents/GitHub
/cs395_data_mining/HW9.ipynb#X13sZmlsZQ%3D%3D?line=20'>21</a> best_eps,
best_min_samples, best_silhouette = find_optimal_dbscan_params(X, eps_range,
min_samples_range)
     <a href='vscode-notebook-cell:/c%3A/Users/alexp/Documents/GitHub
/cs395_data_mining/HW9.ipynb#X13sZmlsZQ%3D%3D?line=23'>24</a> # Print the optimal
hyperparameters and clustering quality
     <a href='vscode-notebook-cell:/c%3A/Users/alexp/Documents/GitHub
/cs395_data_mining/HW9.ipynb#X13sZmlsZQ%3D%3D?line=24'>25</a> print("Optimal eps:",
best_eps)

c:\Users\alexp\Documents\GitHub\cs395_data_mining\HW9.ipynb Cell 10 in
find_optimal_dbscan_params(X, eps_range, min_samples_range)
      <a href='vscode-notebook-cell:/c%3A/Users/alexp/Documents/GitHub
/cs395_data_mining/HW9.ipynb#X13sZmlsZQ%3D%3D?line=8'>9</a> labels =
dbscan.fit_predict(X)
     <a href='vscode-notebook-cell:/c%3A/Users/alexp/Documents/GitHub
/cs395_data_mining/HW9.ipynb#X13sZmlsZQ%3D%3D?line=9'>10</a> if len(set(labels)) >
1:
---> <a href='vscode-notebook-cell:/c%3A/Users/alexp/Documents/GitHub
/cs395_data_mining/HW9.ipynb#X13sZmlsZQ%3D%3D?line=10'>11</a>     silhouette =
silhouette_score(X, labels)
     <a href='vscode-notebook-cell:/c%3A/Users/alexp/Documents/GitHub
/cs395_data_mining/HW9.ipynb#X13sZmlsZQ%3D%3D?line=11'>12</a>     if silhouette >
best_silhouette:
     <a href='vscode-notebook-cell:/c%3A/Users/alexp/Documents/GitHub
/cs395_data_mining/HW9.ipynb#X13sZmlsZQ%3D%3D?line=12'>13</a>
best_silhouette = silhouette

File c:\Users\alexp\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn
\metrics\cluster\_unsupervised.py:117, in silhouette_score(X, labels, metric,
sample_size, random_state, **kwds)
    115     else:
    116         X, labels = X[indices], labels[indices]
--> 117 return np.mean(silhouette_samples(X, labels, metric=metric, **kwds))

File c:\Users\alexp\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn
\metrics\cluster\_unsupervised.py:231, in silhouette_samples(X, labels, metric,
```

Results:

make_moons:

K-Means Clustering Results: Silhouette Score: 0.48975082695298533 Calinski-Harabasz Index: 1481.4753424968233 Davies-Bouldin Index: 0.7817364605843775 Execution Time: 0.04900717735290527 seconds Memory Usage: 146.68359375 MB Hierarchical Clustering Results: Silhouette Score: 0.4391867809034311 Calinski-Harabasz Index: 1087.7713369249986 Davies-Bouldin Index: 0.837284087751716 Execution Time: 0.029980182647705078 seconds Memory Usage: 146.69140625 MB DBSCAN Clustering Results: Eps: 0.4503519041566185 Min Samples: 147 Silhouette Score: 0.3060017383266752 Calinski-Harabasz Index: 361.7530597734725 Davies-Bouldin Index: 2.8816508031309773 Execution Time: 0.24503803253173828 seconds Memory Usage: 147.47265625 MB

smartphone:

K-Means Clustering Results: Silhouette Score: 0.37196991401903456 Calinski-Harabasz Index: 5934.114761241196 Davies-Bouldin Index: 1.074520597574652 Execution Time: 0.3592417240142822 seconds Memory Usage: 311.51953125 MB Hierarchical Clustering Results: Silhouette Score: 0.3735621062678581 Calinski-Harabasz Index: 5765.523209652622 Davies-Bouldin Index: 1.0735359209035171 Execution Time: 8.188697576522827 seconds Memory Usage: 318.9765625 MB

shuttle-landing-control:

K-Means Clustering Results: Silhouette Score: 0.20685539935851324 Calinski-Harabasz Index: 5.1140495867768605 Davies-Bouldin Index: 1.518699240339859 Execution Time: 0.1279003620147705 seconds Memory Usage: 204.6953125 MB Hierarchical Clustering Results: Silhouette Score: 0.19435098907732273 Calinski-Harabasz Index: 4.711515151515 15 Davies-Bouldin Index: 1.6027259104482954 Execution Time: 0.1157076358795166 seconds Memory Usage: 205.296875 MB

Discussion

For the make_moons data, k-means had the best silhouette score by a small margin, though all 3 algorithms had positive scores indicating a decent clustering. K-means outerformed in the other metrics as well, and both hierarchical and k-means outperformed DBSCAN, except for in memory usage. Smartphone data had the hierarchical clustering slightly ahead of k-means in silhouette score but not enough to be significant. Hierarchical also took more memory and time. For the shuttle-landing-control dataset, the clustering scores were relatively low at around 0.2, indicating poor clustering poor clustering.

DBSCAN did not run for either the shuttle landing control or smartphone datasets. The smartphone dataset was much larger, and I let DBSCAN iterate randomly over eps values between 0.1 and 0.5, in combination with min_sample sizes between 2 and n/2. I let the code run over night for 635 minutes and it still hadn't found a valid clustering for the data. For the shuttle-landing-control data, I believe it was too noisy for DBSCAN to create meaningful clusters. I got the error "ValueError: Number of labels is 15. Valid values are 2 to n_samples - 1 (inclusive)" which indicates it was clustering into 15 different classes, which is clearly too many for the data.