



KENNESAW STATE UNIVERSITY

CS 7267
MACHINE LEARNING

PROJECT 1
UNSUPERVISED LEARNING

INSTRUCTOR
Dr. Zongxing Xie

Job King
000618086

1. ABSTRACT

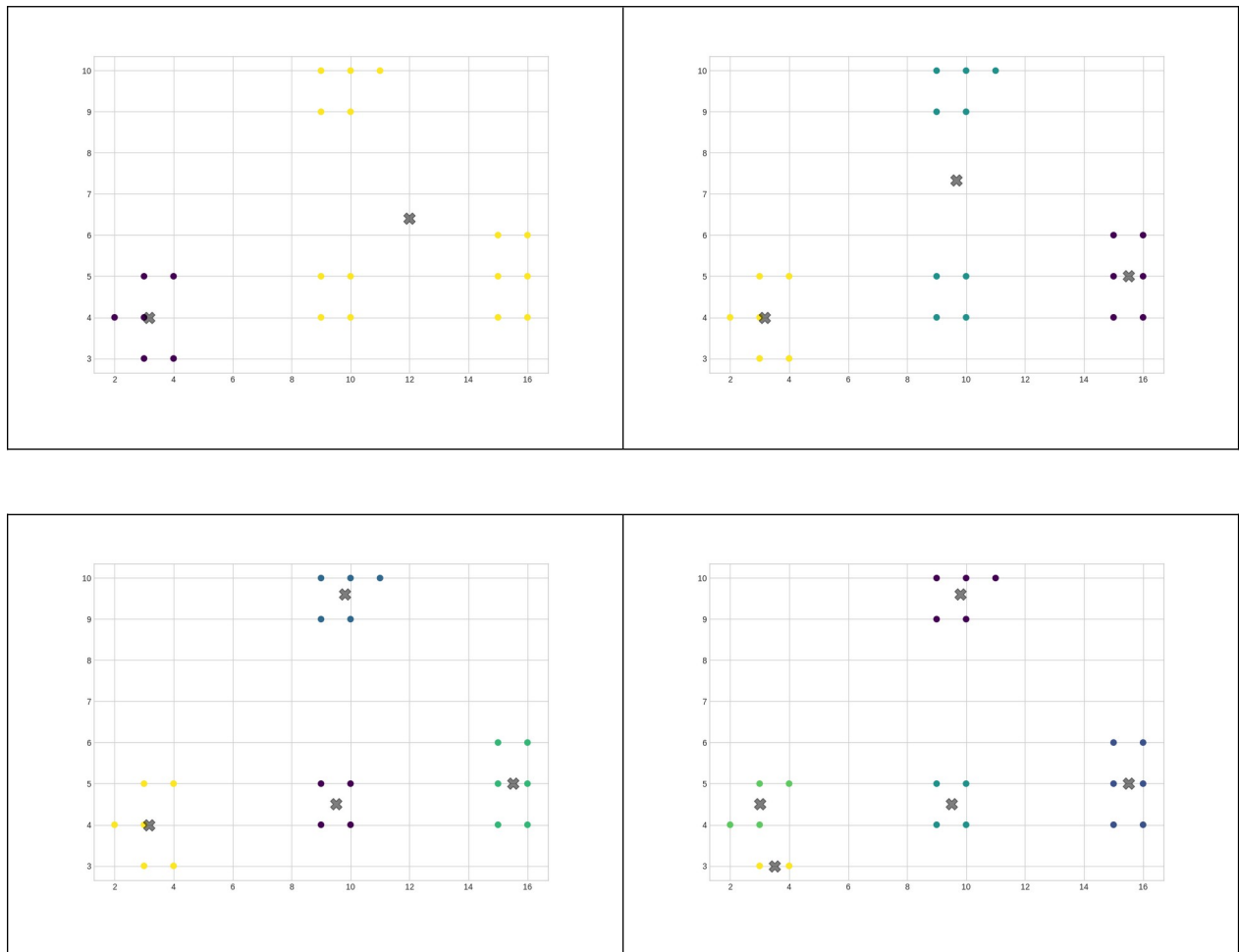
In this project I used unsupervised machine learning techniques to analyze the kmtest and iris datasets. While completing this process I was able to put into practice a number of things discussed in the machine learning course such as data cleaning, data transformation, data analysis and data visualization. In addition, as there were differences in both the required analysis tasks and the datasets themselves, it was necessary to research and utilize different techniques and apply them to successfully analyze each dataset to obtain useful results. As a result, this project made for a very effective initial experience with machine learning.

2. TEST RESULTS

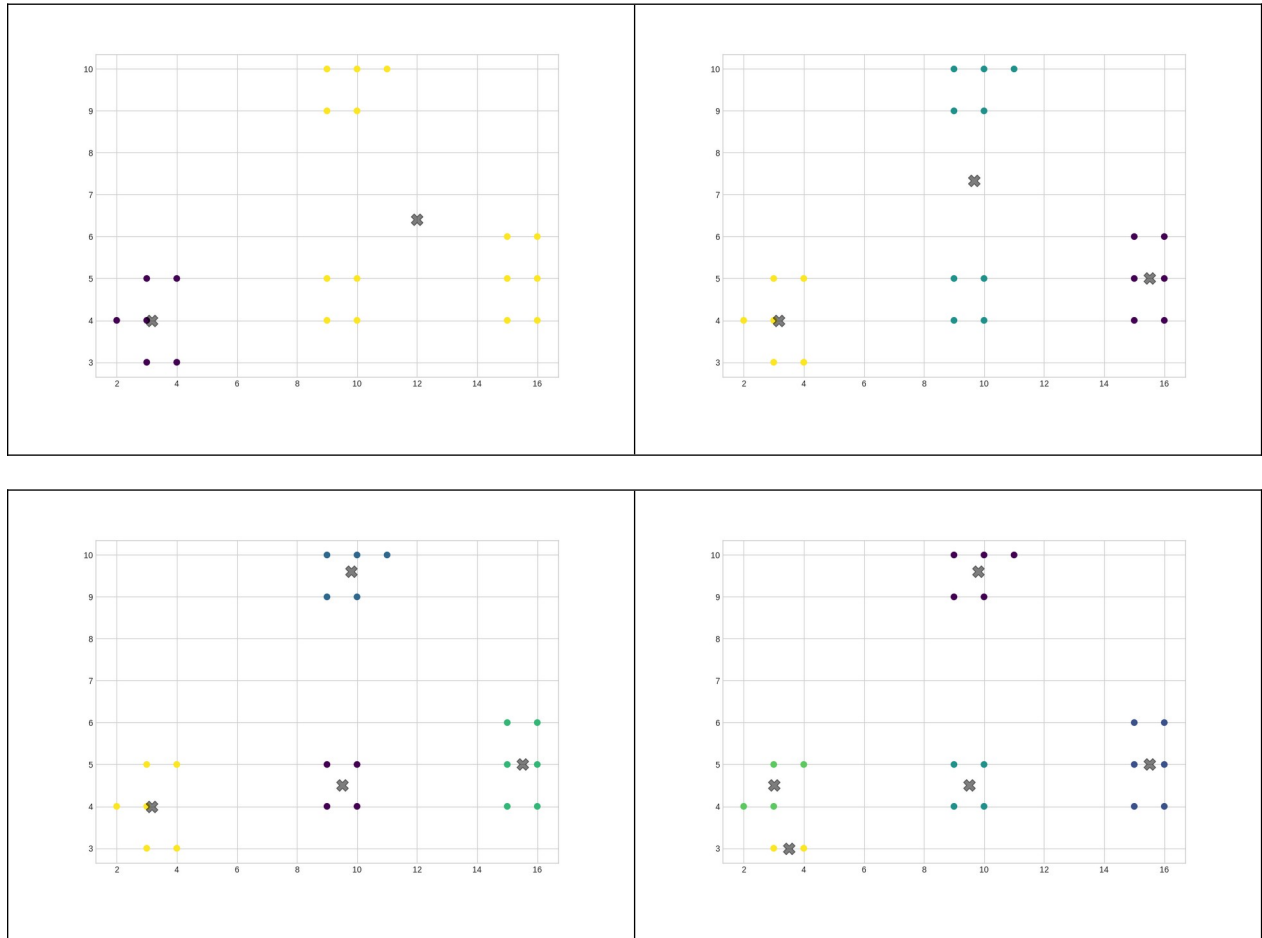
2.1 Clustering with K-means algorithm for kmtest dataset

This task had two parts: clustering the kmtest dataset using 2, 3, 4 and 5 clusters with the raw and z-score normalized data. Both sets of clustering results were plotted. The plots are below in ascending cluster order number with the raw and normalized results denoted.

Raw Results



Normalized Results

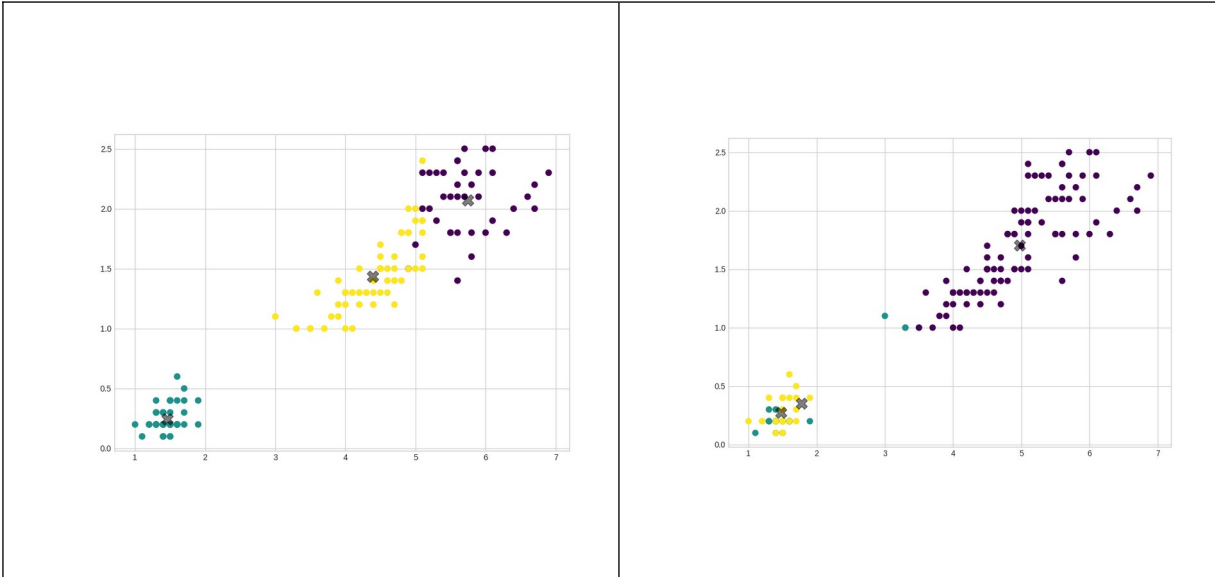


2.2 Test Results for Clustering with K-means algorithm for iris dataset

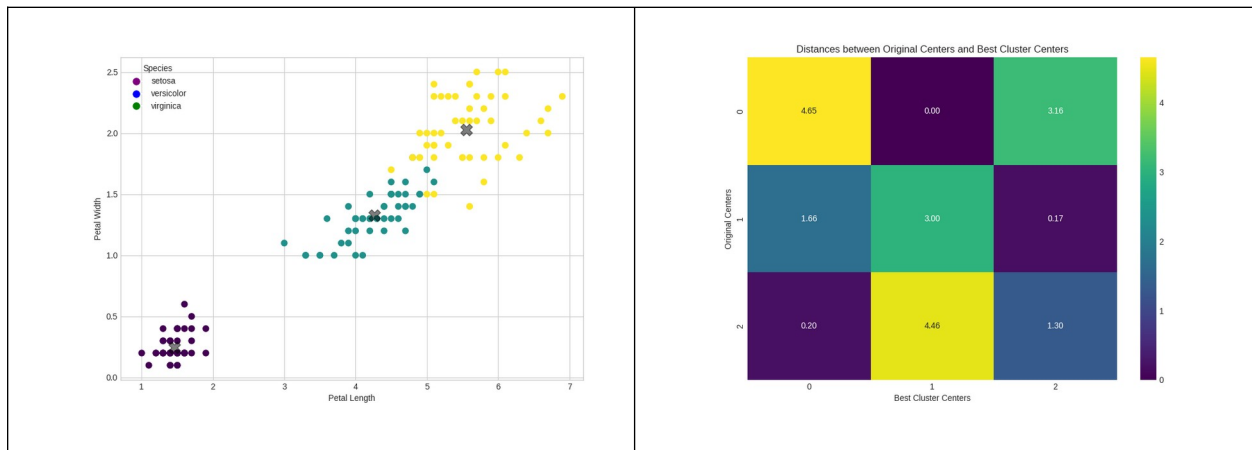
To evaluate the iris data set, I ran 10 iterations with 3 clusters using the sepal length, sepal width, petal length and petal with attributes to predict species. The total sum of squares (TSS), within-cluster sum of squares (WCSS) and between-cluster sum of squares (BCSS) data from those 10 iterations are below.

| TSS | BCSS | WCSS |
|----------|------------------|------------------|
| 681.3706 | 602.519158573854 | 78.851441426146 |
| 681.3706 | 602.514934174023 | 78.8556658259773 |
| 681.3706 | 602.514934174023 | 78.8556658259773 |
| 681.3706 | 602.519158573854 | 78.851441426146 |
| 681.3706 | 602.519158573854 | 78.851441426146 |
| 681.3706 | 602.519158573854 | 78.851441426146 |
| 681.3706 | 535.91790823515 | 145.45269176485 |
| 681.3706 | 602.519158573854 | 78.851441426146 |
| 681.3706 | 602.514934174023 | 78.8556658259773 |
| 681.3706 | 535.91790823515 | 145.45269176485 |

According to Varsha (2025) the goal is to minimize WCSS. The iteration with the lowest WCSS was chosen as the best result. The iteration with the highest WCSS was chosen as the worst. The best WCSS: 78.85144142614601, iteration 4. The worst WCSS: 142.7540625 iteration 5. Their plots according to petal_length and petal_width, attributes 4 and 5, are below respectively.



The previous results predicted the iris flower species using sepal_length, sepal_width, petal_length and petal_width. The next step was to analyze the original results, the iris flower species label data, in attribute 5. A plot of the species data against petal_length and petal_width was done. A further step was to calculate the distance between the centers of the best result, the lowest WCSS iteration, and the original result. To accomplish this, the species character labels were encoded as integers. Next the encoded integers were normalized using linear transformation. The normalized iris integer data was then loaded into a pandas dataframe, whose groupby function was used to estimate the centers. Finally, the Euclidean distance algorithm was used to compute the distance between the species label original centers and the best centers. A visualization for this was done as well. First the species against petal_length and petal_width plot and the distance between the species and best k-Means visualization:



The visualization represented this data for the distances between the original centers and best centers:

| | | |
|-------------|------------|-------------|
| 4.65296875 | 0 | 3.16307025 |
| 1.65883677 | 2.99920056 | 0.171672122 |
| 0.195370803 | 4.46054929 | 1.30100997 |

3. DISCUSSION

For the kmtest data it appears that k-means is better at predicting the clusters with the raw data than the normalized data. It further appears that the ideal K value is 4. If given more time, I would have liked to try more data normalization techniques such as linear or standard deviation. And although I determined that 4 was the best cluster value from visual inspection of the plots, I would have liked to see this confirmed using the elbow method.

For the iris data analysis, visual inspection revealed that there was not much difference between the best result and the original centers. However, there was a great deal of difference between the worst clusters and both. There was not very much variation in the WCSS scores save for outliers in 2 of the runs, and both outliers were very similar. Given more time, I would have researched various methods of generating random clusters to provide a better variety of results. As the WCSS analysis performed well in this instance, there was no need to perform other methods of choosing the best and worst clustering iterations. However, the original results approach requires much revision. The distance calculation between the original and best iteration centers results are a curiosity as the smallest values create a triangle shape and the largest values an inverted triangle shape. Given more time, I would have investigated more ways to visualize the data. Also, the species were originally nominal data that was encoded into 0, 1 and 2 values, adding an additional step after encoding and normalizing the data so that the number ranges were more consistent with the centers produced by k-Means is an area that I would have dedicated more time to.

4. CODE

3.1 Functions module containing k-means algorithm, plotting functions and WCSS, BCSS and TCSS calculations.

```
# Name: Job King
# Number: 000618086
# Project 1
# File: assignment1_functions.py, imported as module af
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import pairwise_distances_argmin
import random
```

```

def simplekmeans(X_df, K, max_runs=300):
    pseudo = 100 * random.randint(1, 100) # generate a pseudo-random seed
    input_df = X_df #preserve original data frame so the original data can be
    used for plotting
    centroid_indices = input_df.sample(K, random_state=pseudo,
    replace=False).index.values # Randomly initialize centroids
    centroids = input_df.values[centroid_indices]

    for x in range(max_runs):
        labels = pairwise_distances_argmin(input_df.values, centroids)

        new_centroids = np.array([input_df.values[labels == i].mean(axis=0)
        for i in range(K)])

        if np.all(new_centroids == centroids):
            break

        centroids = new_centroids

    return centroids, labels
####

find_tss = lambda X: np.sum((X - np.mean(X, axis=0))**2) #usage: tss =
find_tss(X)
find_bcscs = lambda tss, wcss: tss - wcss #usage: bcscs = find_bcscs(tss,
wcss)

def find_wcss(X, K, labels, centers):
    wcss = 0
    m = centers
    for i in range(K):
        x = X[labels == i]
        if x.size > 0:
            wcss += np.sum((x - m[i]) ** 2)
    return wcss
####

def kmtest_plot(X, centers, labels, title):
    plt.style.use('seaborn-v0_8-whitegrid')
    plt.figure(figsize=(10, 7))
    plt.scatter(X[0:, 0], X[0:, 1], c=labels, s=50, cmap='viridis')

```

```

plt.scatter(centers[0:, 0], centers[0:, 1], c='black', s=200, alpha=0.5,
marker='X');
#plt.show()
plt.savefig(title)
plt.close()

def iris_plot(X, centers, labels, title):
plt.style.use('seaborn-v0_8-whitegrid')
plt.figure(figsize=(10, 7))
plt.scatter(X[0:, 2], X[0:, 3], c=labels, s=50, cmap='viridis')
plt.scatter(centers[0:, 2], centers[0:, 3], c='black', s=200, alpha=0.5,
marker='X');
#plt.show()
plt.savefig(title)
plt.close()

def cluster_distance_plot(distances, title):
plt.style.use('seaborn-v0_8-whitegrid')
plt.figure(figsize=(10, 7))
sns.heatmap(distances, annot=True, fmt=".2f", cmap='viridis')
plt.title('Distances between Original Centers and Best Cluster Centers')
plt.xlabel('Best Cluster Centers')
plt.ylabel('Original Centers')
#plt.show()
plt.savefig(title)
plt.close()

def plot_original_result(df, centers, encoder, title):
plt.style.use('seaborn-v0_8-whitegrid')
plt.figure(figsize=(10, 7))

# Use the encoded species to color the data points with a colormap
encoded_species = encoder.transform(df['species'])

plt.scatter(df['petal_length'], df['petal_width'], c=encoded_species,
s=50, cmap='viridis')
plt.scatter(centers['petal_length'], centers['petal_width'], c='black',
s=200, alpha=0.5, marker='X')
# Hard code the legend to match species names with colors
color_scheme = {'setosa': 'purple', 'versicolor': 'blue', 'virginica':
'green'}

legend_elements = []

```

```

for species, color in color_scheme.items():
    legend_elements.append(plt.Line2D([0], [0], marker='o', color='w',
    label=species, markerfacecolor=color, markersize=10))
plt.legend(handles=legend_elements, title='Species')

plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
#plt.title(title)
#plt.show()
plt.savefig(title)
plt.close()

```

3.2 Code for K-means algorithm for kmtest dataset

```

# Name: Job King
# Number: 000618086
# Project 1
# File: kmtest.py, used for question 1

import pandas as pd
import assignment1_functions as af

kmtest = pd.read_csv('kmtest.csv', sep=r'\s+', encoding='utf-8-sig',
header=None, dtype=float)
kmtest_normalized = (kmtest - kmtest.mean()) / kmtest.std()

for K in range(2,6):
    centers, labels = af.simplekmeans(kmtest, K)
    af.kmtest_plot(kmtest.values, centers, labels, f'kmtest-with-K-value-
    {K}.png')
    centers_n, labels_n = af.simplekmeans(kmtest_normalized, K)
    af.kmtest_plot(kmtest_normalized.values, centers_n, labels_n, f'kmtest-
    normalized-with-K-value-{K}.png')

```

3.3 Code for K-means algorithm for iris dataset

```

# Name: Job King
# Number: 000618086
# Project 1
# File: iris.py, used for question 2
import pandas as pd
import assignment1_functions as af
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from scipy.spatial.distance import cdist

```

```

iris_full = pd.read_csv('iris.csv', header=None, names=['sepal_length',
'sepal_width', 'petal_length', 'petal_width', 'species'])

iris = iris_full.iloc[:, 0:4].astype(float) # need separate dataframe
containing first 4 attributes

# low wcss is required for good clustering

center_list = []
label_list = []
wcss_list = []
bcss_list = []
tss_list = []

tss = af.find_tss(iris.values) # calculate total sum of squares
K = 3
for o in range(1,11): # run k-means 10 times to get different results
centers, labels = af.simplekmeans(iris, K)
wcss = af.find_wcss(iris.values, K, labels, centers)
bcss = af.find_bcss(tss, wcss)
center_list.append(centers)
label_list.append(labels)
wcss_list.append(wcss)
bcss_list.append(bcss)
tss_list.append(tss)

# find best and worst results based on wcss
best_results = min(wcss_list)
worst_results = max(wcss_list)
best_bcss = af.find_bcss(best_results, tss)
worst_bcss = af.find_bcss(worst_results, tss)

# find index of best and worst results to use to retrieve corresponding
centers and labels
best_cluster_index = wcss_list.index(best_results)
worst_cluster_index = wcss_list.index(worst_results)
best_centers = center_list[best_cluster_index]
worst_centers = center_list[worst_cluster_index]
best_labels = label_list[best_cluster_index]
worst_labels = label_list[worst_cluster_index]

# plot best and worst clustering results

```

```

af.iris_plot(iris.values, best_centers, best_labels, 'iris-best-
clusters.png')
af.iris_plot(iris.values, worst_centers, worst_labels, 'iris-worst-
clusters.png')

# Save results of running k-means against iris.csv 10 times to CSV and
text files
df = pd.DataFrame({'TSS': tss_list, 'BCSS': bcss_list, 'WCSS': wcss_list})
df.to_csv('iris-kmeans-results.csv', index=False)

with open('iris_clustering_analysis.txt', 'w') as f:
f.write(f'Best WCSS: {best_results}, Best Iteration: {best_cluster_index +
1}, \
Worst WCSS: {worst_results} Worst Iteration: {worst_cluster_index + 1}\n')

#Species data is character. Encode species to integers for analysis
encode_the_species = LabelEncoder()
iris_full['species_encoded'] =
encode_the_species.fit_transform(iris_full['species'])

# Normalize the encoded species data
normalize_the_species = MinMaxScaler()
iris_full['species_normalized'] =
normalize_the_species.fit_transform(iris_full[['species_encoded']])

#Calculate original centers and values using groupby with respect to
attribute 3 and attribute 4
original_centers = iris_full.groupby('species')[['petal_length',
'petal_width']].mean().reset_index()
original_centers_values = original_centers[['petal_length',
'petal_width']].values

#Calculate distances between original centers and best centers
distances = cdist(original_centers_values, best_centers[:, 2:4],
metric='euclidean')

# Save distances to text file
with open('distance_original_best_clusters.txt', 'w') as f:
f.write(f'Distances between original centers and best centers:\
n{distances}\n')

# Plot distances between original centers and best centers

```

```
af.cluster_distance_plot(distances, 'iris-distance-original-best-  
clusters.png')  
  
# Plot original species with original centers overlayed  
af.plot_original_result(iris_full, original_centers.set_index('species'),  
encode_the_species, 'iris-original-centers.png')
```

References

Varsha, I. (2025, May 18). *K-means clustering: Your friendly guide to unlocking hidden patterns in data!*. Medium. <https://medium.com/@inkollusrivarsha0287/k-means-clustering-your-friendly-guide-to-unlocking-hidden-patterns-in-data-aad4b8eb9814>