Moudood Fiaz - Cowlar ML/CV Tasks Report

I have added documentation in my code to explain my methodology however will also add text explanation with pictures if needed.

Task 1:

Requirements:

1. Loading and preprocessing the data.

```
# Load and preprocess data
data = pd.read_csv('fea.csv', header=None)
data = data.to_numpy()

# Calculate norm and normalize X on it
magnitudes = np.linalg.norm(data, axis=1)
X = data / magnitudes[:, np.newaxis]

# Generating labels 0 to 9 for 170 samples
y = np.repeat(np.arange(n_subjects), n_total_per_subject)

# Reduce features to pca_components
if pca_components:
    pca = PCA(n_components=pca_components)
    X = pca.fit_transform(X)
```

Selecting 5 chunks of data randomly picked from all subjects one by one.

```
# Initialize arrays to store the training and testing indices for each repeat
train_indices_all = np.zeros((n_repeats, n_subjects * n_train_per_subject), dtype=int)
test_indices_all = np.zeros((n_repeats, n_subjects * n_val_per_subject), dtype=int)

# Loop over the repeats
for r in range(n_repeats):
    # Initialize arrays to store the training and testing indices for this repeat
    train_indices = np.zeros(n_subjects * n_train_per_subject, dtype=int)
    test_indices = np.zeros(n_subjects * n_val_per_subject, dtype=int)

    # Loop over the subjects
    for s in range(n_subjects):
        # Randomly select the indices for training and testing samples for this subject
        all_indices = np.arange(s * n_total_per_subject, (s + 1) * n_total_per_subject)
        train_indices_sub = np.random.choice(all_indices, size=n_train_per_subject, replace=False)
        test_indices_sub = np.setdiff1d(all_indices, train_indices_sub)

        # Add the indices for this subject to the overall training and testing indices
        train_indices[s * n_train_per_subject:(s + 1) * n_train_per_subject] = train_indices_sub
        test_indices[s * n_val_per_subject:(s + 1) * n_val_per_subject] = test_indices_sub

    # Shuffle the indices for this repeat and add them to the overall indices
    perm = np.random.permutation(len(train_indices))
    train_indices_all[r] = train_indices[perm]
    perm = np.random.permutation(len(test_indices))
    test_indices_all[r] = test_indices[perm]
```

Running experiments across 5 selections of data.

```
# Loop over the repeats
for split in range(n_repeats):
    # Training and Test data for this repeat
    X_train = X[train_indices_all[split]]
    y_train = y[train_indices_all[split]]
    X_test = X[test_indices_all[split]]
    y_test = y[test_indices_all[split]]

    # Initialize and fit the KNN classifier
    knn = KNNClassifier(k, distance_metric=dm)
    knn.fit(X_train, y_train)

    # Make predictions on the test set and record computational time
    t = time.time()
    y_pred = knn.predict(X_test)
    time_taken = time.time() - t
    times.append(time_taken)

    # Evaluate the performance of the model
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)

# Generating report
avg_accuracy = sum(accuracies) / len(accuracies)
std_dev = statistics.stdev(accuracies)
avg_comp_time = sum(times) / len(times)
print("Avg Accuracy: {:.3f}".format(avg_accuracy))
print("Standard Deviation: {:.3f}".format(std_dev))
print("Avg Comp Time: {:.3f}".format(avg_comp_time), "s")
```

2. Custom implementation of KNN Algorithm.

```python
# Custom KNN Classifier Class
class KNNClassifier:
    # Initializing the classifier
    def __init__(self, k=5, distance_metric='euclidean'):
        self.k = k
        self.distance_metric = distance_metric
    # Training data and calculating co-variance
    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train
        if self.distance_metric == 'mahalanobis':
            self.covariance = np.cov(X_train.T)
    # Predicting results
    def predict(self, X_test):
        y_pred = []
        # Loop over each row in test set
        for x_test in X_test:
            distances = []
            # Loop over each row in train set and find distance
            for i, x_train in enumerate(self.X_train):
                if self.distance_metric == 'euclidean':
                    dist = np.sqrt(np.sum((x_test - x_train) ** 2))
                elif self.distance_metric == 'mahalanobis':
                    dist = mahalanobis(x_test, x_train, self.covariance)
                elif self.distance_metric == 'cosine':
                    dist = cosine(x_test, x_train)
                else:
                    raise ValueError('Invalid distance metric')
                distances.append((i, dist))
            # Sort distances so that shortest ones are in front
            distances.sort(key=lambda x: x[1])
            neighbors = distances[:self.k]
            neighbor_labels = [self.y_train[i] for i, _ in neighbors]
            # Most frequent label in k nearest neighbours
            y_pred.append(max(set(neighbor_labels), key=neighbor_labels.count))
        return y_pred
```

a. Comparing different distance metrics. Accuracy drops in mahalanobis because it uses covariance matrix to calculate distance and therefore it is beneficial for highly correlated data and since we have applied PCA the data is uncorrelated. Cosine metric is very slow as compared to euclidean distance therefore we will be using it for future experiments.

```
Distance Metric: euclidean, K: 5, NC: 10, n_pca: 75
Total(170) = Train(150) + Val(20)
Avg Accuracy: 0.957
Standard Deviation: 0.024
Avg Comp Time: 1.924 s
(carai) moudood@moudood-G3-3579:~/Documents/Projects/f
Distance Metric: mahalanobis, K: 5, NC: 10, n_pca: 75
Total(170) = Train(150) + Val(20)
Avg Accuracy: 0.634
Standard Deviation: 0.025
Avg Comp Time: 2.070 s
(carai) moudood@moudood-G3-3579:~/Documents/Projects/f
Distance Metric: cosine, K: 5, NC: 10, n_pca: 75
Total(170) = Train(150) + Val(20)
Avg Accuracy: 0.948
Standard Deviation: 0.015
Avg Comp Time: 7.749 s
```

Comparing different values of k. Accuracy tends to decrease with increasing value of K giving best results when k=1 or k=3. However keeping K very small is overfitting to the training data therefore we will keep K=5 in future experiments. Increasing K beyond a certain value underfits training data therefore reducing accuracy.

```
Distance Metric: euclidean, K: 1, NC: 10, n_pca: 75
Total(170) = Train(150) + Val(20)
Avg Accuracy: 0.990
Standard Deviation: 0.004
Avg Comp Time: 1.785 s
(carai) moudood@moudood-G3-3579:~/Documents/Projects
Distance Metric: euclidean, K: 35, NC: 10, n_pca: 75
Total(170) = Train(150) + Val(20)
Avg Accuracy: 0.890
Standard Deviation: 0.030
Avg Comp Time: 1.848 s
(carai) moudood@moudood-G3-3579:~/Documents/Projects
Distance Metric: euclidean, K: 67, NC: 10, n_pca: 75
Total(170) = Train(150) + Val(20)
Avg Accuracy: 0.819
Standard Deviation: 0.031
Avg Comp Time: 1.827 s
```

b. Reducing the number of classes increases the accuracy and decreases the computation time.

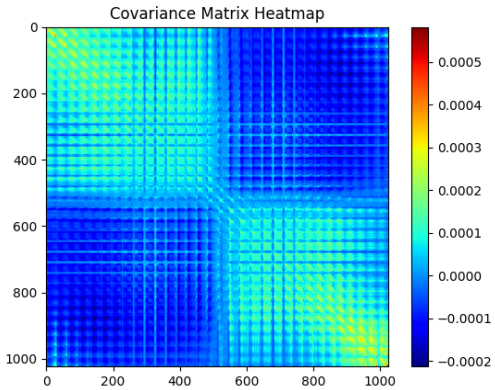
```
Distance Metric: euclidean, K: 5, NC: 10, n_pca: 75
Total(170) = Train(150) + Val(20)
Avg Accuracy: 0.953
Standard Deviation: 0.016
Avg Comp Time: 1.768 s
(carai) moudood@moudood-G3-3579:~/Documents/Project
Distance Metric: euclidean, K: 5, NC: 5, n_pca: 75
Total(170) = Train(150) + Val(20)
Avg Accuracy: 0.980
Standard Deviation: 0.007
Avg Comp Time: 0.453 s
(carai) moudood@moudood-G3-3579:~/Documents/Project
Distance Metric: euclidean, K: 5, NC: 2, n_pca: 75
Total(170) = Train(150) + Val(20)
Avg Accuracy: 0.980
Standard Deviation: 0.021
Avg Comp Time: 0.085 s
```

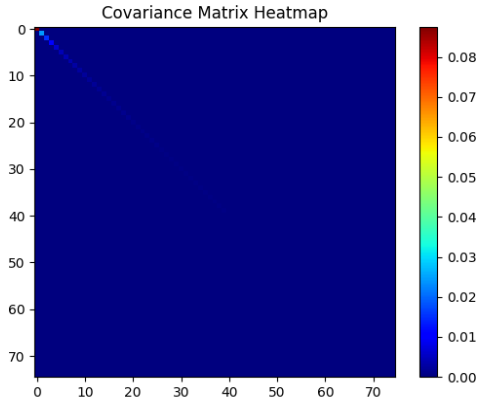
c. Reducing the training examples drops the accuracy.

```
Distance Metric: euclidean, K: 5, NC: 10, n_pca: 75
Total(170) = Train(100) + Val(70)
Avg Accuracy: 0.923
Standard Deviation: 0.013
Avg Comp Time: 4.199 s
(carai) moudood@moudood-G3-3579:~/Documents/Project
Distance Metric: euclidean, K: 5, NC: 10, n_pca: 75
Total(170) = Train(150) + Val(20)
Avg Accuracy: 0.961
Standard Deviation: 0.008
Avg Comp Time: 1.727 s
```

3. Using PCA to reduce the dimensions of the training data improved the computational performance of the model by up to 28% providing very similar accuracies. Moreover the correlation is also handled well. We lose features when reducing the number of pca_components and therefore loss of accuracy but it is not very much as compared to the performance gains.

```
Distance Metric: euclidean, K: 5, NC: 10, n_pca: None
Total(170) = Train(150) + Val(20)
The data is correlated
Avg Accuracy: 0.957
Standard Deviation: 0.003
Avg Comp Time: 2.589 s
(carai) moudood@moudood-G3-3579:~/Documents/Projects/t
Distance Metric: euclidean, K: 5, NC: 10, n_pca: 75
Total(170) = Train(150) + Val(20)
The data is uncorrelated
Avg Accuracy: 0.949
Standard Deviation: 0.015
Avg Comp Time: 1.862 s
```





Task 2:

My custom implementation of K-means clustering. My implementation was 92% faster than the library function.

```python
def kmeans(k, data, use_builtin=False):
    # Use builtin function
    if use_builtin:
        km = KMeans(n_clusters=k)
        km.fit(data)
        return km.labels_, km.cluster_centers_

    # Use custom implementation
    else:
        # Initialize centroids randomly
        centroids = data[np.random.choice(data.shape[0], k, replace=False), :]

        while True:
            # Assign each data point to the nearest centroid
            distances = np.linalg.norm(data[:, np.newaxis] - centroids, axis=2)
            cluster_idx = np.argmin(distances, axis=1)

            # Update centroids as the mean of the data points assigned to each cluster
            new_centroids = np.array([data[cluster_idx == i, :].mean(axis=0)
                                      if np.sum(cluster_idx == i) > 0 else centroids[i]
                                      for i in range(k)])

            # Check if centroids have moved
            if np.allclose(new_centroids, centroids):
                break

            # Updating current centroid to new for next iteration
            centroids = new_centroids

        return cluster_idx, centroids
```

```
Comp Time: 12.698 s
(carai) moudood@moud
libpng warning: iCCP
Comp Time: 1.666 s
```

Extraction of seed pixels from rgb image through seed masks.

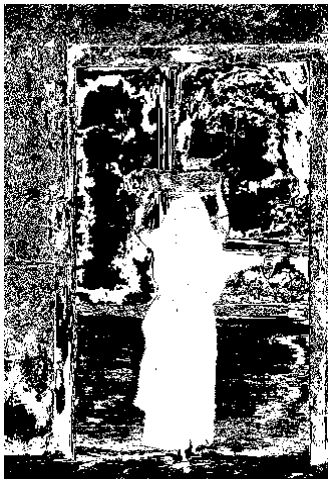
```python
def extract_seed_pixels(seed_img, rgb_img):
    # Extracting channels from seed
    blue_ch = seed_img[:, :, 0]  # Blue channel in BGR
    red_ch = seed_img[:, :, 2]  # Red channel in BGR

    # Extracting masks from rgb image
    background_mask = cv2.bitwise_and(rgb_img, rgb_img, mask=blue_ch)
    foreground_mask = cv2.bitwise_and(rgb_img, rgb_img, mask=red_ch)

    # Extracting only the colored pixels to reduce computational time
    non_black_pixels = background_mask.any(axis=-1)
    background = background_mask[non_black_pixels]
    non_black_pixels = foreground_mask.any(axis=-1)
    foreground = foreground_mask[non_black_pixels]

    return background, foreground
```

Function to compute likelihood given the image and k-means output for both classes.

```python
def compute_likelihood(rgb_img, bg_cluster_idx, bg_centroids, fg_cluster_idx, fg_centroids):
    # Calculating weights
    bg_weights = np.bincount(bg_cluster_idx, minlength=len(bg_centroids)) / len(bg_cluster_idx)
    fg_weights = np.bincount(fg_cluster_idx, minlength=len(fg_centroids)) / len(fg_cluster_idx)

    # Compute distance and exponential function
    bg_dist = np.linalg.norm(rgb_img[:, :, np.newaxis, :] - bg_centroids, axis=-1)
    bg_expo = np.exp(-bg_dist)**2 * bg_weights
    fg_dist = np.linalg.norm(rgb_img[:, :, np.newaxis, :] - fg_centroids, axis=-1)
    fg_expo = np.exp(-fg_dist)**2 * fg_weights

    # Compute probabilities for each pixel and centroid
    bg_prob = np.sum(bg_expo, axis=2)
    fg_prob = np.sum(fg_expo, axis=2)

    # Set foreground pixels to white and background pixels to black
    bw = np.zeros_like(rgb_img)
    bw[fg_prob > bg_prob] = 255

    return bw
```

Now I will show the results.

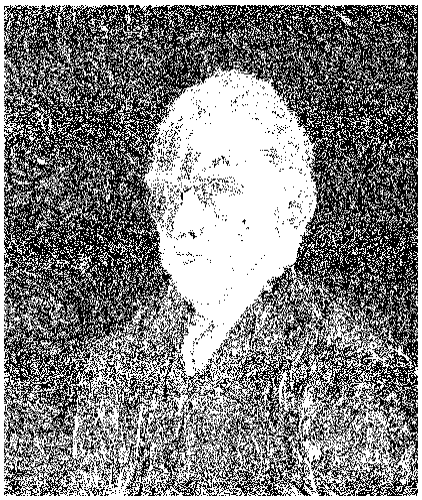
1. Trying various values of N i.e. N= 4, 16, 64



Results get better but computational times increase as well.

```
Comp Time: 0.110 s
(carai) moudood@mou
libpng warning: iCC
Comp Time: 0.568 s
(carai) moudood@mou
libpng warning: iCC
Comp Time: 2.076 s
```

2. Van Gogh results. It took more time for the clustering to converge for this picture.



3. Extra strokes vs normal strokes

4. Results using built-in function which was 92% slower vs my custom implementation

```
Comp Time: 12.698 s
(carai) moudood@moud
libpng warning: iCCP
Comp Time: 1.666 s
```