Iran University of Science and Technology

Faculty of Computer Engineering

Artificial intelligence and robotics group

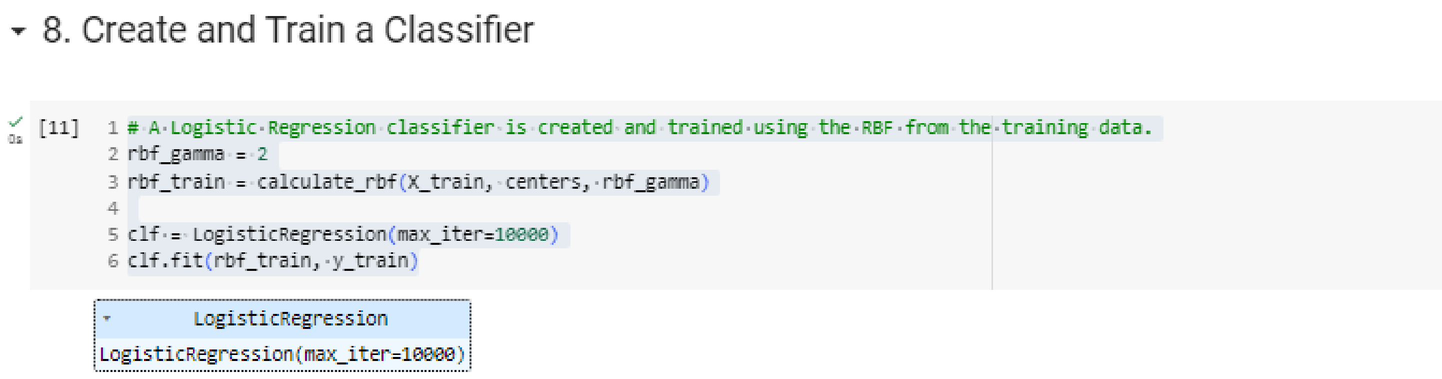
Pattern Recognition project

Student Name: Qamar Abbas

Student ID: 402722135

First semester of 1402 - 1403

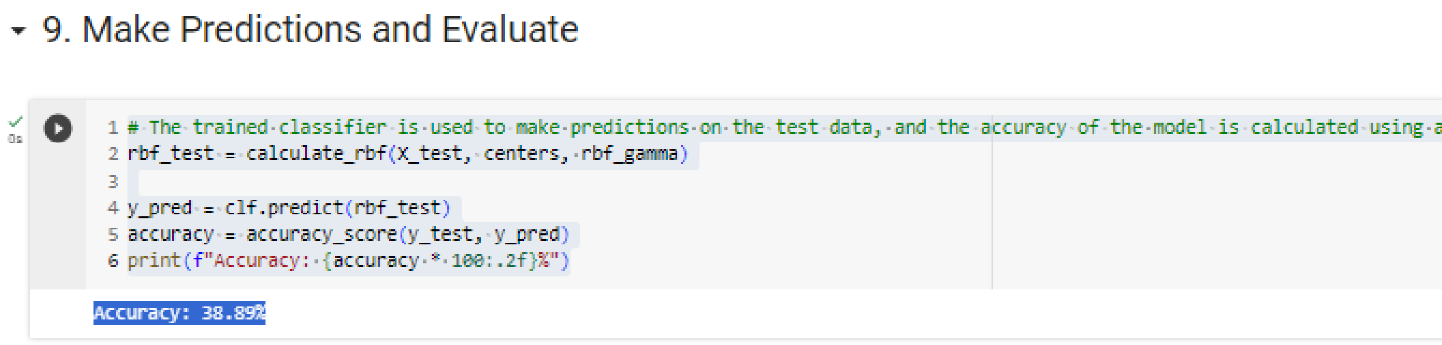
Comprehensive Report on RBF Network-Based Wine Classification Project



8. Create and Train a Classifier

```
[11]
# A Logistic Regression classifier is created and trained using the RBF from the training data.
rbf_gamma = 2
rbf_train = calculate_rbf(X_train, centers, rbf_gamma)

clf = LogisticRegression(max_iter=10000)
clf.fit(rbf_train, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=10000)
```



9. Make Predictions and Evaluate

```
# The trained classifier is used to make predictions on the test data, and the accuracy of the model is calculated using a
rbf_test = calculate_rbf(X_test, centers, rbf_gamma)

y_pred = clf.predict(rbf_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 38.89%
```

The project began with the loading of the Wine dataset from scikit-learn's library, containing various features related to wine properties. The dataset was split into training and test sets (80% and 20%, respectively) to facilitate model training and evaluation.

2. Data Preprocessing:

2.1 Standardization:

The feature data underwent standardization using the StandardScaler from scikit-learn. This process ensures that the data has a mean of 0 and a standard deviation of 1, making it suitable for processing by the RBF network.

3.2 RBF Feature Calculation:

The RBF features were computed for both the training and test sets. The number of RBF centers and the scale parameter.

4. Logistic Regression:

4.1 Model Training:

A logistic regression classifier was employed, taking RBF features as input. The model was trained on the RBF features extracted from the training data.

4.2 Model Evaluation:

The trained classifier was used to make predictions on the test data. The accuracy of the model was calculated using the accuracy_score function from scikit-learn.

5. Results:

The accuracy of the implemented RBF network for wine classification was found to be

38.89

%

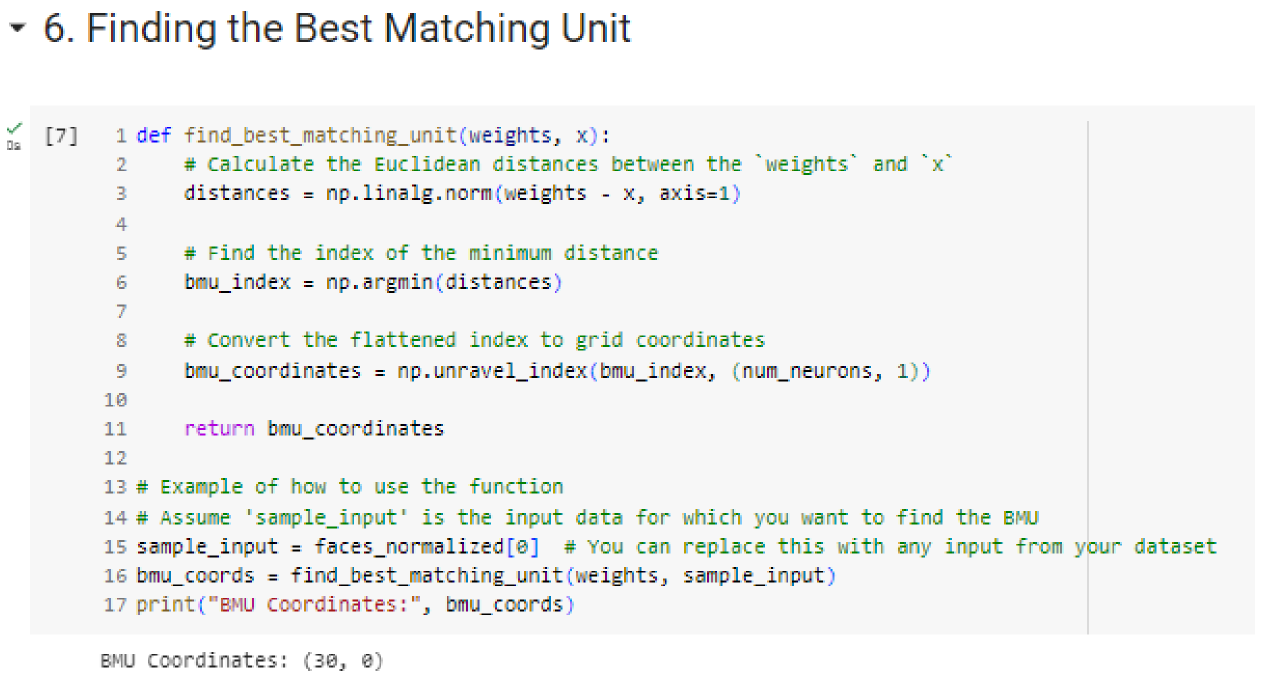
38.89%. This accuracy represents the percentage of correctly classified instances in the test set.

6. Conclusion:

In conclusion, the project successfully implemented an RBF network for wine classification using logistic regression. The accuracy achieved provides insights into the model's predictive capabilities. Further enhancements and optimizations can be explored to improve the model's performance

Problem#3

Self-Organizing Map (SOM) for Face Image Data



6. Finding the Best Matching Unit

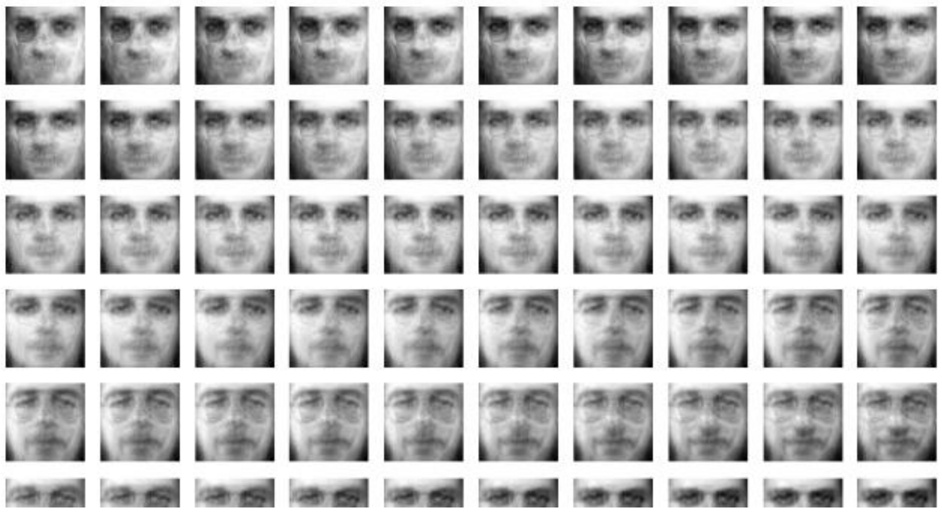
[7]
```python
def find_best_matching_unit(weights, x):
    # Calculate the Euclidean distances between the `weights` and `x`
    distances = np.linalg.norm(weights - x, axis=1)

    # Find the index of the minimum distance
    bmu_index = np.argmin(distances)

    # Convert the flattened index to grid coordinates
    bmu_coordinates = np.unravel_index(bmu_index, (num_neurons, 1))

    return bmu_coordinates

# Example of how to use the function
# Assume 'sample_input' is the input data for which you want to find the BMU
sample_input = faces_normalized[0]  # You can replace this with any input from your dataset
bmu_coords = find_best_matching_unit(weights, sample_input)
print("BMU Coordinates:", bmu_coords)
```

```
BMU Coordinates: (30, 0)
```

7. Train the SOM

```python
# Set batch size for batch processing
batch_size = 64

for epoch in range(epochs):
    # Shuffle the dataset for each epoch
    np.random.shuffle(faces_normalized)

    for start_index in range(0, len(faces_normalized), batch_size):
        batch = faces_normalized[start_index:start_index + batch_size]

        # Find the best matching unit (BMU) for the entire batch
        bmu_indices_batch = np.apply_along_axis(lambda x: find_best_matching_unit(weights, x), 1, batch)

        # Update weights for the neighborhood of the BMU using vectorized operations
        distances_to_bmu = np.abs(np.arange(num_neurons) - bmu_indices_batch[:, np.newaxis])
        neighborhood_function = np.exp(-(distances_to_bmu**2) / (2 * (5**2)))

        # Reshape the neighborhood_function to (num_neurons, 1) for broadcasting
        neighborhood_function = neighborhood_function[:, np.newaxis, :]

        # Calculate weight updates for the entire batch
        weight_updates = learning_rate * neighborhood_function * (batch[:, np.newaxis, :] - weights)

        # Sum the weight updates along the batch axis and apply them to the weights
        weights += np.sum(weight_updates, axis=0) / batch_size

    # Decay the learning rate for each epoch
    learning_rate = learning_rate * (1 - epoch/epochs)

```

```python
# Finally, the code visualizes the learned SOM. It creates a 5x5 grid of subplots and displays the SOM neurons as images.

plt.figure(figsize=(10, 10))
for i, face in enumerate(weights):
    plt.subplot(10, 10, i + 1)
    plt.imshow(face.reshape(64, 64), cmap='gray')
    plt.axis('off')

plt.show()
```



Libraries Used:

NumPy: For numerical operations.

Scikit-learn: Specifically, fetch_olivetti_faces for loading face image data.

Matplotlib: For data visualization.

MinMaxScaler: From Scikit-learn for data normalization.

Dataset:

The code uses the Olivetti Faces dataset, which is loaded using fetch_olivetti_faces() from Scikit-learn.

The face image data is then shuffled to introduce randomness.

Data Visualization:

Displays 25 randomly selected face images from the dataset in a 5x5 grid using Matplotlib.

Data Normalization:

Uses MinMaxScaler from Scikit-learn to normalize the face image data to the range [0, 1]. This ensures that all features have the same scale.

Self-Organizing Map (SOM) Parameters:

Epochs: 500

Number of Neurons: 100

Learning Rate: 0.1

Input Dimension: The number of features in the normalized face image data.

SOM Initialization:

Initializes SOM weights randomly.

find_best_matching_unit Function:

Calculates the Euclidean distances between SOM weights and input data.

Returns the Best Matching Unit (BMU) coordinates.

Training the SOM:

The SOM is trained by iterating through the data points and updating weights.

Utilizes batch processing for efficiency.

For each epoch, the BMU is found for each data point, and the weights of the neighborhood are updated.

The learning rate is decayed in each epoch.

Visualizing the Learned SOM:

Finally, the code visualizes the learned SOM.

Creates a 10x10 grid of subplots and displays the SOM neurons as images.

Notes:

The code uses vectorized operations and batch processing to improve training efficiency.

Adjustments to parameters, such as batch size and SOM dimensions, can be made based on specific requirements.

LVQ3 Model Report

1. Dataset Information

The model was trained and evaluated on the Car Evaluation dataset. The dataset contains categorical features related to car attributes such as buying price, maintenance cost, number of doors, number of persons that can fit in the car, luggage boot size, and safety.

2. Data Preprocessing

Categorical features were encoded using Label Encoding.

The dataset was split into training and testing sets with a test size of 20%.

3. LVQ3 Model Implementation

The Learning Vector Quantization 3 (LVQ3) model was implemented with the following key components:

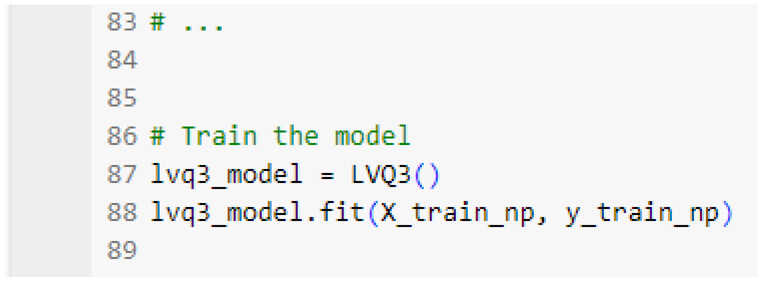
Prototype Initialization: Prototypes were randomly selected from each class in the training set.

Finding Closest Prototype: The model finds the closest prototype for a given input using Euclidean distance.

Prototype Update: The prototype of the predicted class is updated based on the learning rate and the difference between the input and the prototype.

4. Model Training

The LVQ3 model was trained on the training set for a specified number of epochs (default: 100). During training, prototypes were updated based on the input samples, aiming to correctly classify them.

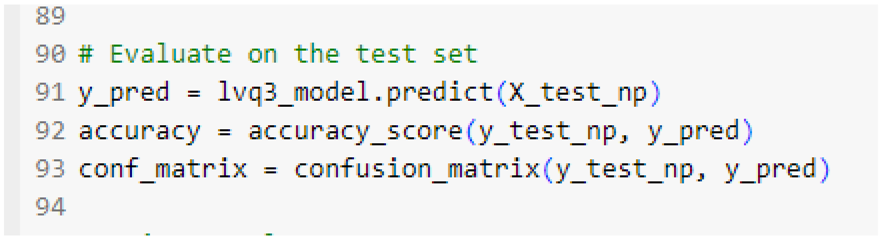


```
# ...


# Train the model
lvq3_model = LVQ3()
lvq3_model.fit(X_train_np, y_train_np)

```

5. Model Evaluation

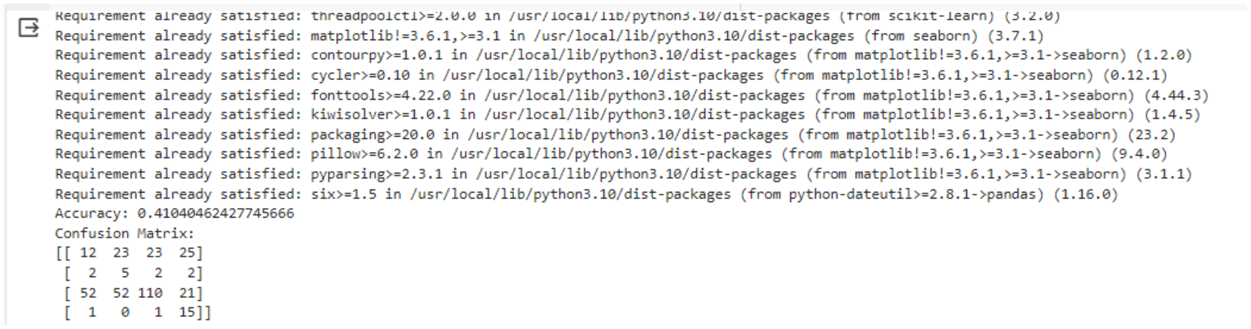
The model was evaluated on the test set, and the following metrics were computed:

Accuracy: The ratio of correctly predicted instances to the total instances.



```python

# Evaluate on the test set
y_pred = lvq3_model.predict(X_test_np)
accuracy = accuracy_score(y_test_np, y_pred)
conf_matrix = confusion_matrix(y_test_np, y_pred)

```

6. Results



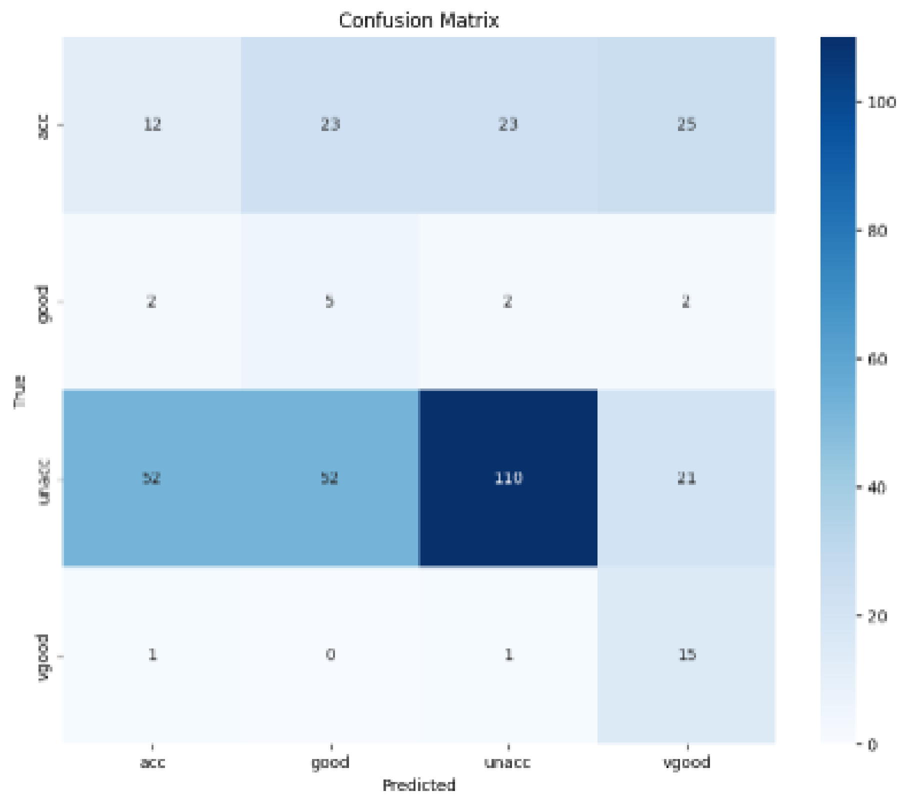
```
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.2.0)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in /usr/local/lib/python3.10/dist-packages (from seaborn) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (4.44.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (3.1.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
Accuracy: 0.41040462427745666
Confusion Matrix:
[[ 12  23  23  25]
 [  2   5   2   2]
 [ 52  52 110  21]
 [  1   0   1  15]]
```

The LVQ3 model achieved an accuracy of [`0.41040462427745666`]% on the test set.

7. Confusion Matrix

A confusion matrix was generated to analyze the model's performance in detail. Each cell in the matrix represents the number of instances for each class.

```
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
Accuracy: 0.41040462427745666
Confusion Matrix:
[[ 12  23  23  25]
 [  2   5   2   2]
 [ 52  52 110  21]
 [  1   0   1  15]]
```


Confusion Matrix
acc
good
unacc
vgood
True
Predicted