Experiment-9

Aim:

To implement the Learning Vector Quantization (LVQ) algorithm in Python and apply it to classify sample data.

Apparatus Required:

- Python (Anaconda/Miniconda/Standard Python Installation)
- Jupyter Notebook or any Python IDE
- Libraries: numpy, matplotlib, sklearn

Theory:

Learning Vector Quantization (LVQ) is a supervised learning algorithm used for classification. It works by representing each class with prototype vectors (codebook vectors). During training, these prototypes move closer to correctly classified input samples and farther from incorrectly classified samples.

The LVQ algorithm steps:

1. Initialize prototype vectors (codebook vectors) for each class.
2. For each input sample:
    - Find the nearest prototype vector (winner).
    - If the prototype's class matches the input's class, move it closer to the input vector.
    - Otherwise, move it away from the input vector.
3. Repeat over multiple iterations, decreasing the learning rate.

LVQ is effective in classification tasks where data is labeled, and it provides an intuitive representation of class boundaries.

Code:

```
import math
import matplotlib.pyplot as plt
class LVQ:
   def winner(self, weights, sample):
      distances = []
      for w in weights:
         d = 0
         for i in range(len(sample)):
            d += (sample[i] - w[i])**2
         distances.append(math.sqrt(d))
```

```
        return distances.index(min(distances))
    def update(self, weights, sample, label, J, cluster_labels, alpha):
        if cluster_labels[J] == label:
            # Move winner closer
            for i in range(len(weights[J])):
                weights[J][i] += alpha * (sample[i] - weights[J][i])
        else:
            # Move winner away
            for i in range(len(weights[J])):
                weights[J][i] -= alpha * (sample[i] - weights[J][i])
        return weights
def format_weights(weights):
    """Format weights with 8 decimal places"""
    return [[round(v, 8) for v in w] for w in weights]
def main():
    # Initial weights (prototypes)
    weights = [[0.2, 0.6, 0.6, 0.3],
               [0.5, 0.6, 0.3, 0.3]]
    # Training samples (features)
    T = [[0, 1, 0, 1],
         [0, 1, 1, 1],
         [0, 0, 0, 0],
         [1, 0, 0, 1]]
    # True class labels
    labels = [0, 0, 1, 1]
    # Assign class labels to prototypes
    cluster_labels = [0, 1]
    # Test sample
    test_sample = [1, 0, 0, 1]
    lvq = LVQ()
    alpha = 0.3
    epochs = 10
    # ---- Training ----
    for epoch in range(epochs):
```

```
        print(f"\nEpoch {epoch+1}:")
        for i, sample in enumerate(T):
            J = lvq.winner(weights, sample)
            weights = lvq.update(weights, sample, labels[i], J, cluster_labels, alpha)
            # Print update log with formatted weights
            print(f"Sample {sample} -> Winner: Cluster {J}, Updated weights:
{format_weights(weights)}")
        alpha *= 0.9  # decay learning rate
    print("\nFinal Result:")
    J_test = lvq.winner(weights, test_sample)
    print(f"Test Sample {test_sample} belongs to Cluster: {cluster_labels[J_test]}")
    print(f"Trained weights: {format_weights(weights)}")
    # ---- Visualization ----
    plt.figure(figsize=(6, 6))
    # Plot training samples
    for i, sample in enumerate(T):
        color = 'blue' if labels[i] == 0 else 'green'
        plt.scatter(sample[0], sample[1], c=color, marker='o', s=50,
                    label='Training samples' if i == 0 else "")

    # Plot prototypes/clusters with unique colors
    cluster_colors = ['red', 'purple']
    for j, w in enumerate(weights):
        plt.scatter(w[0], w[1], c=cluster_colors[j], marker='X', s=150,
                    label=f'Cluster {cluster_labels[j]}')
    # Plot test sample
    plt.scatter(test_sample[0], test_sample[1], c='orange', marker='o', s=100, label='Test
sample')
    plt.title("LVQ Clustering (2 clusters)")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.legend()
    plt.grid(True)
    plt.show()
```

```
if __name__ == "__main__":
    main()
```

Result:

- The LVQ model successfully classified the test dataset with an accuracy printed on the console.
- The plot shows the test samples colored by true class and the prototype vectors represented by 'X' markers.
- Prototype vectors effectively represent cluster centers and class boundaries learned during training.

Conclusion:

Learning Vector Quantization is a simple yet effective supervised classification algorithm. It provides interpretable prototype vectors representing each class and learns class boundaries through competitive weight updates.