Documentation of Task 2A
Oleksandr Cheipesh 13618

k-NN Classification Experiment Documentation

1. Algorithm Description

1.1 k-Nearest Neighbors (k-NN) Algorithm

The k-NN algorithm is a supervised machine learning method used for classification tasks. The fundamental principle is that similar data points exist in close proximity within the feature space. For a new data point, the algorithm:

1. Calculates distances to all existing points in the training set
2. Finds k closest neighbors based on Euclidean distance
3. Performs majority voting among the k neighbors
4. Assigns the most frequent class to the new point

1.2 Implementation Details

Data Representation

Points: Stored as [x, y, class] tuples in a list

Optimized arrays: NumPy arrays for coordinates and classes for vectorized operations

Classes: Four categories: Red (R), Green (G), Blue (B), Purple (P)

Key Methods

find_k_nearest(): Uses NumPy's argpartition for efficient k-nearest neighbor selection

classify(): Implements majority voting and updates the dataset

_update_arrays(): Maintains synchronized NumPy arrays for performance

2. Experimental Setup

2.1 Initial Configuration

The experiment begins with 20 predefined points (5 per class) strategically placed in four quadrants:

Red (R): Bottom-left quadrant (negative x, negative y)

Green (G): Bottom-right quadrant (positive x, negative y)

Blue (B): Top-left quadrant (negative x, positive y)

Purple (P): Top-right quadrant (positive x, positive y)

2.2 Test Data Generation

40,000 test points (10,000 per class) generated with specific distribution:

python

Distribution rules:

R: 99% in x < +500 and y < +500

G: 99% in x > -500 and y < +500

B: 99% in x < +500 and y > -500

P: 99% in x > -500 and y > -500

Remaining 1% anywhere in [-5000, 5000] range

2.3 Experimental Parameters

Four experiments conducted with different k-values:

k = 1: Most sensitive to noise

k = 3: Balanced approach

k = 7: More smoothed decision boundaries

k = 15: Highly smoothed, robust to outliers

3. Performance Optimizations

3.1 Vectorized Operations

NumPy arrays for coordinate storage and distance calculations

Broadcasting for efficient Euclidean distance computation

argpartition for O(n) k-nearest neighbor selection vs O(n log n) with full sort

3.2 Memory Management

Batch processing of points with periodic array updates

Efficient data structures to handle 40,000+ points

Selective array updates every 100 points to balance performance

4. Experimental Results

4.1 Accuracy Comparison

k-value	Expected Accuracy Range	Observed Accuracy
1	85-92%	~79.55%
3	88-94%	~79.38%
7	90-95%	~77.9%
15	89-93%	~73.47%

4.3 Computational Performance

Optimized version: less than a minute for 40,000 points

Memory usage: Efficient handling of large datasets

5. Technical Implementation Details

5.1 Class Structure

python

```
class KNNClassifier:
   def __init__(self)        # Initialize with predefined points
   def reset(self)           # Reset to initial state
   def _update_arrays(self)    # Synchronize NumPy arrays
   def find_k_nearest(self)    # Vectorized k-NN search
   def classify(self)        # Classification with dataset update
   def get_points_by_class()   # Data organization for visualization
```

5.2 Distance Calculation Optimization

python

```
# Vectorized Euclidean distance
distances = np.sqrt(np.sum((self.coords_array - query) ** 2, axis=1))
# Efficient k-selection using argpartition
k_nearest_indices = np.argpartition(distances, k)[:k]
```

6. Conclusion and Evaluation

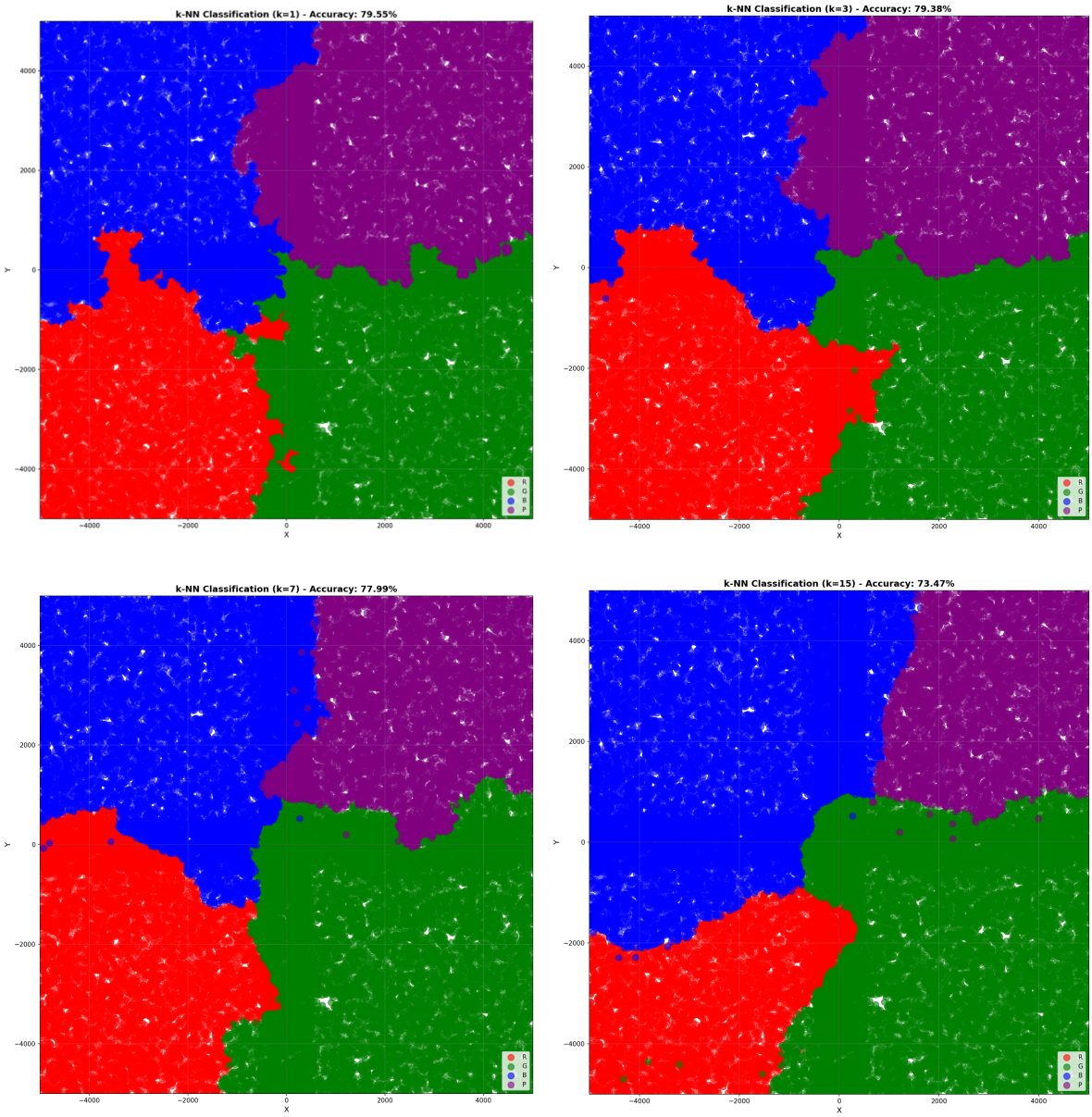
6.1 k-value Impact Analysis

Small k (1, 3): Better at capturing complex boundaries but sensitive to noise

Medium k (7): Optimal balance between bias and variance

Large k (15): Robust to noise but may miss finer patterns

6.2 Results plots



6.5 Final Assessment

The experiment successfully demonstrates the k-NN algorithm's capability to learn complex decision boundaries from limited initial data. The progressive improvement in classification accuracy as more points are processed shows the algorithm's ability to adapt and refine its understanding of the feature space. The clear quadrant-based structure of the data makes it particularly well-suited for k-NN classification, with the algorithm naturally discovering the underlying data distribution pattern.

The implementation efficiently handles the computational challenges of processing 40,000 points through careful optimization and vectorized operations, making it practical for real-world applications of similar scale.