Project 1: Intermediate Classification Methods Comparison

Pelopidas-Nikolaos Tsiountsiouras
Department of Electrical and Computer Engineering
Aristotle University of Thessaloniki
Email: pelopidt@ece.auth.gr

***Abstract*—This work presents an implementation and comparison of three classical classification approaches: the 1-Nearest Neighbor (1-NN), 3-Nearest Neighbor (3-NN), and Nearest Centroid Classifier (NCC). Experiments were conducted on the CIFAR-10 dataset using various distance metrics (Euclidean, Manhattan, and Cosine) and Principal Component Analysis (PCA) for dimensionality reduction. Quantitative results highlight the trade-off between accuracy and computational cost across methods.**

I. Classifier Implementations

A. K-Nearest Neighbor (K-NN)

The K-NN classifier predicts a sample's label based on the k closest training samples according to a chosen distance metric. The distance function was implemented for Euclidean, Manhattan, and Cosine distances. For $k > 1$, the final prediction is determined by majority voting.

```
def knn_classifier(X_train, y_train, X_test, k=1,
    distance_type='euclidean'):
    n_test = X_test.shape[0]
    n_train = X_train.shape[0]
    predictions = np.zeros(n_test, dtype=int)

    print(f"\nExcecution␣{k}-NN␣classifier␣({
        distance_type}␣distance)...")
    print(f"Training␣samples:␣{n_train}")
    print(f"Test␣samples:␣{n_test}")

    start_time = time.time()

    for i in range(n_test):
        # Show progress every 1000 samples
        if (i + 1) % 1000 == 0:
            elapsed = time.time() - start_time
            print(f"␣␣␣Processed␣{i+1}/{n_test}␣
                samples␣({elapsed:.1f}s)")

        # Distance function choice
        if distance_type == 'manhattan':
            distances = manhattan_distance_batch(
                X_test[i], X_train)
        elif distance_type == 'cosine':
            distances = cosine_distance_batch(X_test
                [i], X_train)
        else:  # euclidean (default)
            distances = np.sqrt(np.sum((X_train -
                X_test[i]) ** 2, axis=1))

        # Find k nearest
        k_nearest_indices = np.argsort(distances)[:k
            ]
        k_nearest_labels = y_train[k_nearest_indices
            ]

        # Voting
        if k == 1:
            predictions[i] = k_nearest_labels[0]
        else:
            # Majority
            most_common = Counter(k_nearest_labels).
                most_common(1)
            predictions[i] = most_common[0][0]

    elapsed_time = time.time() - start_time
    print(f"Completed␣in␣{elapsed_time:.2f}␣seconds"
        )

    return predictions
```

B. Nearest Centroid Classifier (NCC)

NCC assigns each test sample to the class with the nearest mean vector (centroid) in the feature space.

```
def ncc_classifier(X_train, y_train, X_test):
    n_classes = len(np.unique(y_train))
    n_features = X_train.shape[1]

    # Calculating class centers
    print("\n␣Calculating␣class␣centers...")
    centroids = np.zeros((n_classes, n_features))

    for c in range(n_classes):
        # Find all samples of class c
        class_samples = X_train[y_train == c]

        # Average Calculation
        centroids[c] = np.mean(class_samples, axis
            =0)
        print(f"Class␣{c}␣({class_names[c]}):␣{
            class_samples.shape[0]}␣samples")

    # Classification of test samples
    print("\n␣Excecuting␣NCC␣classifier...")
    n_test = X_test.shape[0]
    predictions = np.zeros(n_test, dtype=int)

    start_time = time.time()

    for i in range(n_test):
        if (i + 1) % 1000 == 0:
            elapsed = time.time() - start_time
            print(f"Processed␣{i+1}/{n_test}␣samples
                ␣({elapsed:.1f}s)")

        #

        distances = np.sqrt(np.sum((centroids -
            X_test[i]) ** 2, axis=1))

        #
```

```
        predictions[i] = np.argmin(distances)

    elapsed_time = time.time() - start_time
    print(f"Completed in {elapsed_time:.2f} seconds"
        )

    return predictions
```

II. Results and Discussion

A. Performance Overview

Table I summarizes the performance across all classifiers. The 3-NN classifier using the Manhattan distance achieved the highest accuracy (39.22%), while the Nearest Centroid Classifier (NCC) was the fastest (1.34 seconds). PCA-based 1-NN models provided a strong balance, maintaining nearly the same accuracy with a 50–60× reduction in runtime.

TABLE I
Classifier Performance Summary

Classifier	Accuracy (%)	Time (s)
1-NN (Euclidean)	35.39	3998.20
3-NN (Euclidean)	35.61	3722.56
NCC	27.74	1.34
3-NN (Manhattan)	39.22	5130.59
3-NN (Cosine)	37.74	3544.48
1-NN (PCA 25)	38.54	68.65
1-NN (PCA 50)	39.10	88.63
1-NN (PCA 100)	38.52	143.18

B. Accuracy Comparisons

The following figures show the baseline accuracy comparison between classifiers and the effect of distance metric choice on K-NN performance.

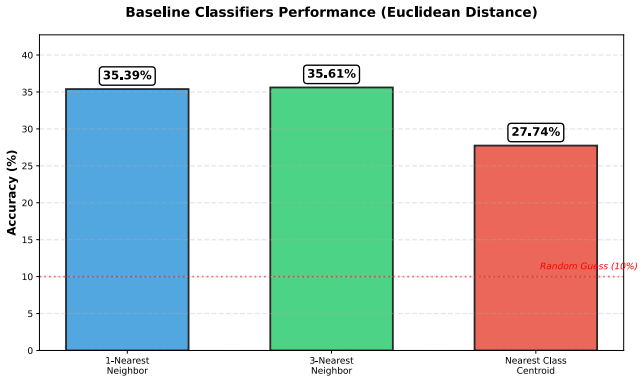

Fig. 1. Baseline accuracies of 1-NN, 3-NN, and NCC classifiers.

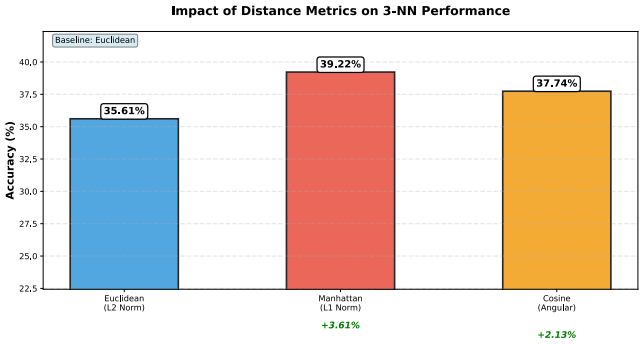

Fig. 2. Accuracy comparison across different distance metrics.

C. Correct and Incorrect Classification Examples

Figures **??** and **??** present visual examples of the classifiers' predictions. The first figure shows correctly classified samples from different categories, demonstrating cases where the models capture key visual cues accurately. The second figure illustrates incorrectly classified samples, highlighting typical sources of error such as color or texture similarity between classes.

Fig. 3. Correct classifications per class.

Fig. 4. Incorrect classifications per class.

D. Analysis

Overall, 3-NN consistently improved performance compared to 1-NN due to better neighborhood voting, but at the cost of computation time. The Manhattan distance provided the most discriminative measure for this dataset, while PCA reduction effectively balanced runtime and accuracy, making it the most practical approach for high-dimensional data.

III. Conclusion

This study compared three classification techniques using different distance metrics and dimensionality reduction. The best-performing configuration was 3-NN with Manhattan distance, achieving 39.22% accuracy. However, PCA-based 1-NN models achieved a comparable 39.10% accuracy with drastically lower runtime, highlighting the benefit of feature compression. Overall, the experiments illustrate the trade-off between computational efficiency and predictive performance in instance-based classifiers.