K-Nearest Neighbors (KNN) Implementation Report

Breast Cancer Detection & CIFAR-10 Classification

Group M

January 22, 2026

Abstract

This report details the implementation and evaluation of the K-Nearest Neighbors (KNN) algorithm on two distinct datasets: the Breast Cancer Wisconsin dataset (binary classification) and a subset of the CIFAR-10 dataset (multi-class image classification). Various distance metrics (Euclidean, Manhattan, Minkowski, Cosine, and Hamming) were tested across multiple K values. The results indicate that KNN performs significantly better on structured tabular data compared to high-dimensional raw image pixel data.

1 Introduction

The K-Nearest Neighbors (KNN) algorithm is a non-parametric, instance-based learning method used for classification and regression. It operates on the principle that similar data points exist in close proximity to each other. This project explores the effectiveness of KNN by implementing core distance functions from scratch and applying them to real-world datasets.

2 Methodology

2.1 Distance Metrics

The core of the KNN algorithm is the distance metric used to determine proximity. The following metrics were implemented:

- **Euclidean Distance**: The straight-line distance between two points.

$$d(x, y) = \sqrt{\sum (x_i - y_i)^2}$$

- **Manhattan Distance**: The sum of absolute differences.

$$d(x, y) = \sum |x_i - y_i|$$

- **Minkowski Distance**: A generalization of Euclidean and Manhattan distances (used here with $p = 3$).

$$d(x, y) = \left(\sum |x_i - y_i|^p\right)^{1/p}$$

- **Cosine Distance**: Measures the cosine of the angle between two non-zero vectors.

$$d(x, y) = 1 - \frac{x \cdot y}{||x||||y||}$$

- **Hamming Distance**: Measures the proportion of positions at which the corresponding symbols are different.

2.2 Algorithm Implementation

The `knn_predict` function was implemented to:

1. Calculate the distance between a test point and all training points.
2. Sort the distances in ascending order.
3. Select the top K nearest neighbors.
4. Determine the majority class (mode) among the neighbors to make a prediction.

3 Task 1: Breast Cancer Prediction

3.1 Experimental Setup

The Breast Cancer dataset was processed by mapping the diagnosis ('M' for Malignant, 'B' for Benign) to binary values (1 and 0). The data was split into 80% training and 20% testing sets. We evaluated the model using $K \in \{3, 4, 9, 20, 47\}$ across all distance metrics.

3.2 Results

The experiments revealed that the **Cosine** distance metric with $K = 3$ yielded the highest accuracy of approximately **90.35%**.

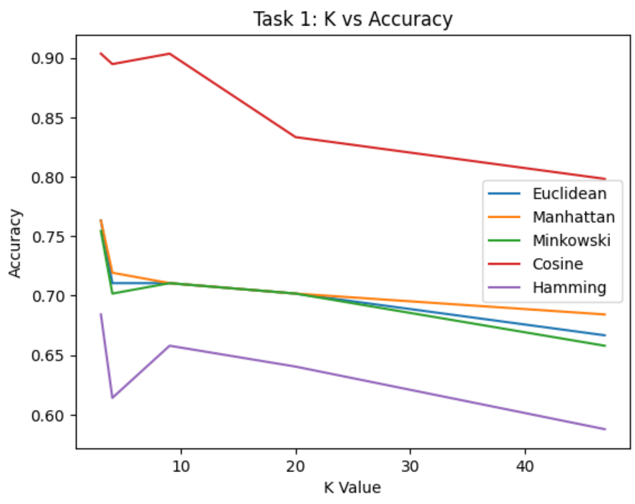

Figure 1: Impact of K and Distance Metric on Accuracy (Task 1)

The confusion matrix for the best model ($K = 3$, Cosine) indicated strong performance, particularly in identifying benign cases.

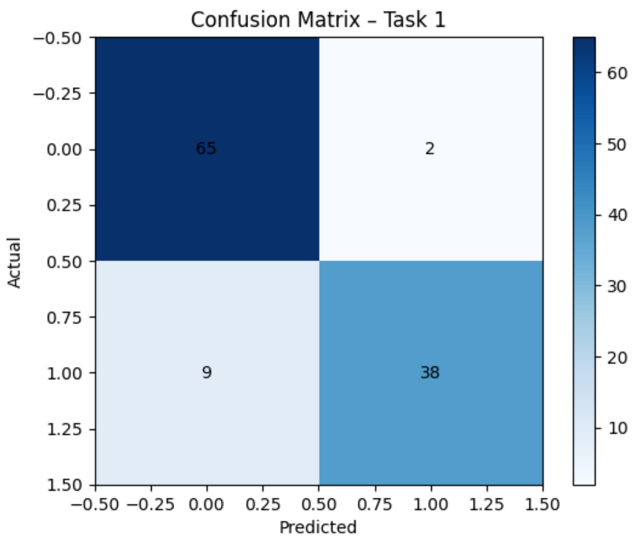

Figure 2: Confusion Matrix (Task 1)

Performance Metrics:

- **Precision**: 0.95

- **Recall**: 0.81

The high precision suggests a low false-positive rate, which is crucial in medical diagnosis to avoid unnecessary distress, though the recall indicates some malignant cases might be missed.

4 Task 2: CIFAR-10 Image Classification

4.1 Experimental Setup

For the image classification task, the CIFAR-10 dataset was utilized. Due to the high computational cost of KNN on pixel data, the dataset was subsampled to 2000 training images and 500 test images. The images were flattened into 1D vectors before processing.

4.2 Results

The results for image classification were significantly lower than the tabular data task, which is expected for a basic KNN implementation on raw pixels. The best performance was achieved using the **Manhattan** distance with $K = 4$, resulting in an accuracy of approximately **26.8%**.

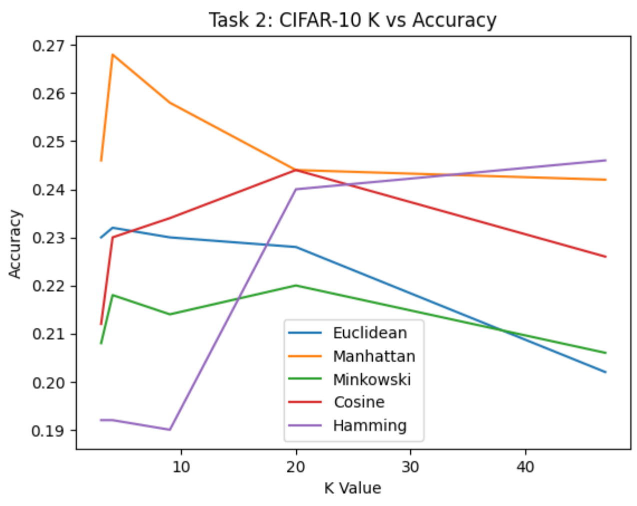

Figure 3: Impact of K and Distance Metric on Accuracy (Task 2)

The multi-class confusion matrix below highlights the difficulty the model faced in distinguishing between visually similar classes (e.g., cats vs. dogs).

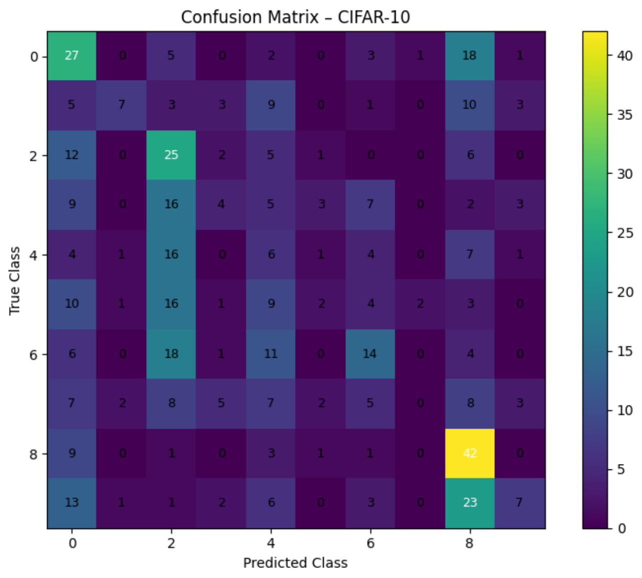

Figure 4: Multi-class Confusion Matrix (Task 2)

5 Conclusion

This project demonstrated the versatility and limitations of the KNN algorithm.

- **Distance Metrics Matter**: Cosine distance proved superior for the high-dimensional features of the breast cancer dataset, likely due to its focus on orientation rather than magnitude. In contrast, Manhattan distance performed better for the grid-like pixel data of CIFAR-10.
- **Data Type Sensitivity**: KNN achieved high accuracy ($> 90\%$) on structured tabular data but struggled (26%) with raw image data. This highlights the need for feature extraction or more advanced algorithms (like CNNs) for image classification tasks.

A Code Implementation

A.1 Core KNN Functions

```python
import numpy as np

def euclidean(x, y):
    return np.sqrt(np.sum((x - y) ** 2))

def knn_predict(X_train, y_train, X_test, k, distance_fn):
    predictions = []
    for test_point in X_test:
        distances = []
        for i in range(len(X_train)):
            dist = distance_fn(test_point, X_train[i])
            distances.append((dist, y_train[i]))
        distances.sort(key=lambda x: x[0])
        neighbors = distances[:k]
        labels = [label for _, label in neighbors]
        predictions.append(max(set(labels), key=labels.count))
    return np.array(predictions)

def accuracy(y_true, y_pred):
    return np.mean(y_true == y_pred)
```

A.2 Evaluation Metrics

```python
def confusion_matrix(y_true, y_pred):
    cm = np.zeros((2,2), dtype=int)
    for t, p in zip(y_true, y_pred):
        cm[t][p] += 1
    return cm

def precision_recall(cm):
    precision = cm[1,1] / (cm[0,1] + cm[1,1])
    recall = cm[1,1] / (cm[1,0] + cm[1,1])
    return precision, recall

def confusion_matrix_multi(y_true, y_pred, classes=10):
    cm = np.zeros((classes, classes), dtype=int)
    for t, p in zip(y_true, y_pred):
        cm[t][p] += 1
    return cm
```

A.3 Task 1: Breast Cancer Pipeline (Snippet)

```python
import pandas as pd

# Data Loading
df = pd.read_csv('data.csv')
df.drop('Unnamed:␣32', axis=1, inplace=True)
df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})

X = df.drop('diagnosis', axis=1).values
y = df['diagnosis'].values

# ... Splitting and Training Loop ...

best_metric_1 = max(results_task1, key=lambda x: max(results_task1[x]))
best_k_1 = Ks[np.argmax(results_task1[best_metric_1])]
```

```python
print(f"Best␣Metric:␣{best_metric_1},␣Best␣K:␣{best_k_1}")
```

A.4 Task 2: CIFAR-10 Pipeline (Snippet)

```python
from tensorflow.keras.datasets import cifar10

(X_train, y_train), (X_test, y_test) = cifar10.load_data()
# Flattening and Subsampling
X_train_s = X_train[:2000]
y_train_s = y_train[:2000].flatten()
X_test_s = X_test[:500]
y_test_s = y_test[:500].flatten()

# ... Training Loop ...
```