k-Nearest Neighbors (k-NN)

Ioannis Michalainas, TBD

October 2024

Abstract

This project implements the k-nearest neighbors (k-NN) algorithm to compute the nearest neighbors of query points based on their distances to a dataset. Utilizing optimized matrix operations and the CBLAS library, the implementation efficiently handles distance calculations in high-dimensional spaces.

1 Problem Statement

The objective of this project is to implement a subroutine that computes the **k-nearest neighbors (k-NN)** of each query point in Q based on their distances to the data points in C. In this problem, we assume that $Q = C$.

To achieve this, we will utilize the following distance calculation formula:

$$D = \sqrt{C^2 - 2CQ^T + (Q^2)^T} \tag{1}$$

where:

- C represents the set of data points.
- Q represents the set of query points.
- D is the distance matrix resulting from the calculations.

2 Example

In this section, we illustrate the process of generating random data points and calculating the distance matrix for k-NN using the following C code snippets. For the sake of clarity, let's assume we have a set of data points C and query points Q in d-dimensional space.

2.1 Random Data Generation

We begin by generating random data points for both the dataset C and query set Q. The following function creates a dataset containing a specified number of points and dimensions:

Listing 1: Generating random data points

```c
void random_data(DataSet *dataset, size_t points, size_t dimensions) {
  dataset->data = (double *)malloc(points*dimensions*sizeof(double));
  dataset->rows = points;
  dataset->cols = dimensions;

  for(size_t i=0; i<points*dimensions; i++) {
    dataset->data[i] = ((double)rand()/RAND_MAX)*100.0;
  }
}
```

For example, if we choose points = 4 and dimensions = 2, we might generate the following random dataset:

$$C = \begin{bmatrix} 23.45 & 12.34 \\ 65.23 & 43.67 \\ 32.98 & 77.54 \\ 54.21 & 11.29 \end{bmatrix}$$

And assume Q is identical to C:

$$Q = \begin{bmatrix} 23.45 & 12.34 \\ 65.23 & 43.67 \\ 32.98 & 77.54 \\ 54.21 & 11.29 \end{bmatrix}$$

2.2 Distance Calculation

Next, we calculate the distance matrix D using the following function. This function utilizes the CBLAS library to perform efficient matrix operations, leveraging the power of optimized linear algebra libraries for speed.

Listing 2: Calculating distances using CBLAS

```c
void calculate_distances(const DataSet* C, const DataSet* Q, double* D) {
  size_t n = C->rows;  // Number of points in C (and Q)
  size_t d = C->cols;  // Number of dimensions

  double* C_squared = (double*)malloc(n*sizeof(double));
  double* Q_squared = (double*)malloc(n*sizeof(double));

  // Compute squared terms
  for(size_t i=0; i<n; i++) {
    C_squared[i] = 0;
    for(size_t j=0; j<d; j++) {
      C_squared[i] += C->data[i*d + j]*C->data[i*d + j];
    }
  }

  for(size_t i=0; i<n; i++) {
    Q_squared[i] = 0;
    for(size_t j=0; j<d; j++) {
      Q_squared[i] += Q->data[i*d + j]*Q->data[i*d + j];
    }
  }

  // Calculate the distance matrix using CBLAS
  double* C_QT = (double*)malloc(n*n*sizeof(double));
  cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasTrans,
              n, n, d,
              -2.0, C->data, d, Q->data, d,
              0.0, C_QT, n);

  // Combine results to get distance matrix
  #pragma omp parallel for
  for (size_t i=0; i<n; i++) {
    for (size_t j=0; j<n; j++) {
```

```
        D[i*n + j] = C_squared[i] + Q_squared[j] + C_QT[i*n + j];
      }
    }

    free(C_squared);
    free(Q_squared);
    free(C_QT);
}
```

Given our dataset C and query set Q, we can compute the distance matrix D:

$$D = \begin{bmatrix} 0.00 & 68.56 & 60.17 & 5.26 \\ 68.56 & 0.00 & 96.54 & 52.16 \\ 60.17 & 96.54 & 0.00 & 66.30 \\ 5.26 & 52.16 & 66.30 & 0.00 \end{bmatrix}$$

Each entry $D[i][j]$ in this matrix represents the squared distance between the i-th point in C and the j-th point in Q. This matrix serves as a foundation for identifying the k-nearest neighbors based on distance metrics.

2.3 Finding k-Nearest Neighbors

To find the k-NN, we will use the **quiselect** algorithm. This will be further elaborated in a subsequent section.

3 Summary

In this project, we compute the distance matrix D using the formula provided in the problem statement. The following steps summarize what we have accomplished so far:

a. We calculated the squared terms for the data points C and query points Q.

b. We computed the dot product CQ^T to facilitate the distance calculation.

c. We will combine the results to obtain the distance matrix D using matrix operations.

d. Finally, we identified the k-nearest neighbors for each query point based on the computed distances.

4 Conclusion

This project aims to efficiently implement the k-NN algorithm using advanced mathematical and programming techniques, facilitating accurate neighbor searches in high-dimensional spaces. The systematic approach outlined ensures clarity and correctness in the computation of distances and neighbor identification.