MULTICORE PROCESSING ON PREDICTION ALGORITHMS

A PROJECT REPORT

by

Abhinav Bansal(22BAI1074)

Vital Sai(22BAI1073)

Sayandeep Das(22BRS1044)

Shaik Junaid Rizwan(22BPS1065)

Under the Guidance of

Prof. Vallidevi K.

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
VELLORE INSTITUTE OF TECHNOLOGY
CHENNAI - 600127

November 2023

Index

Abstract

This project aims to conduct an in-depth comparison of the computational efficiency of two widely used machine learning models, k-Nearest Neighbors (k-NN) and Support Vector Machine (SVM), within the context of binary classification tasks using C++. The primary objective is to meticulously measure and analyze the time taken by each model to make predictions on a relatively modest dataset comprising 17 example points. This investigation includes the utilization of system calls such as `fork` and `wait` for parallel execution and precise timing using `ctime`.

1. Introduction

Machine learning has emerged as a pivotal technology in various domains for classification and predictive analytics. In this project, we delve into two prominent algorithms for binary classification: k-NN and SVM. The central objective is to scrutinize the time efficiency of these models within the C++ environment, a programming language well-known for its speed and performance. Our study involves a dataset of only 17 example points, intentionally kept small to facilitate closer examination.

2. Methodology

2.1 Data

Our dataset consists of 17 example points. While this dataset is limited in size, it serves as an ideal case for benchmarking the time efficiency of the models in a controlled environment. It was created using synthetic data to provide a manageable workload for analysis.

2.2 Models

1. **k-Nearest Neighbors (k-NN):** In C++, the k-NN algorithm was meticulously implemented and tested with various values of k, including 3, 5, and 7. k-NN, known for its simplicity, classifies data points based on the majority class among their k-nearest neighbors. We analyzed how different values of k impacted time efficiency.

2. **Support Vector Machine (SVM):** SVM is a powerful binary classifier, implemented in C++ with both linear and radial basis function (RBF) kernels. SVM is renowned for its efficiency in high-dimensional spaces and was used for benchmarking against k-NN.

2.3 Evaluation

Precise measurement of time efficiency was performed by timing the execution of both models using the `ctime` library in C++. To reduce variance, we conducted multiple runs of each model, and the results were averaged. While time efficiency was the primary focus, we also assessed the models' predictive performance, including accuracy, precision, recall, and F1-score.

3. Results

3.1 Time Efficiency

Our study with the small dataset revealed interesting findings. Both k-NN and SVM in C++ exhibited rapid prediction times. However, it was notable that k-NN, due to its instance-based nature and the necessity of searching for the nearest neighbors, typically consumed more time compared to SVM. The choice of k in k-NN, for example, smaller k values such as 3, led to faster predictions, though the differences were minimal given the dataset's small size.

3.2 Model Performance

In addition to time efficiency, we evaluated the models' performance in terms of accuracy, precision, recall, and F1-score. The results demonstrated that SVM consistently outperformed k-NN in terms of predictive accuracy. Nevertheless, k-NN exhibited certain advantages, particularly for specific k values, by providing a balanced trade-off between precision and recall.

4. Parallel Execution with `fork` and `wait`

To optimize machine learning tasks and reduce execution time, we explored the use of system calls such as `fork` and `wait` in C++. By creating multiple processes using `fork`, we were able to parallelize the execution of different tasks. The parent process effectively synchronized and collected results from child processes using `wait`. This parallel execution strategy showed promise in improving the efficiency of the models.

5. Multicore Scheduling

Efficient utilization of CPU resources is critical for reducing execution time. Multicore scheduling was instrumental in our project, as it allowed us to leverage available CPU cores for parallel execution of multiple instances of machine learning models. This approach effectively reduced the overall time taken for predictions while optimizing CPU resource utilization.

6. Conclusion

In summary, this extended project report compared the time efficiency of k-NN and SVM models for binary classification tasks using C++. The small dataset of 17 example points allowed for a meticulous examination. While both models demonstrated efficient prediction times due to the dataset's size, k-NN typically consumed more time compared to SVM. The choice between k-NN and SVM should primarily be driven by predictive performance, taking into account the application's specific requirements and dataset size.

The utilization of system calls like `fork` and `wait` for parallel execution, as well as multicore scheduling, offers valuable tools when dealing with larger datasets and more computationally intensive models. These strategies are essential for optimizing computational load distribution and reducing time taken for predictions, especially in the context of more extensive datasets and complex models.

Code

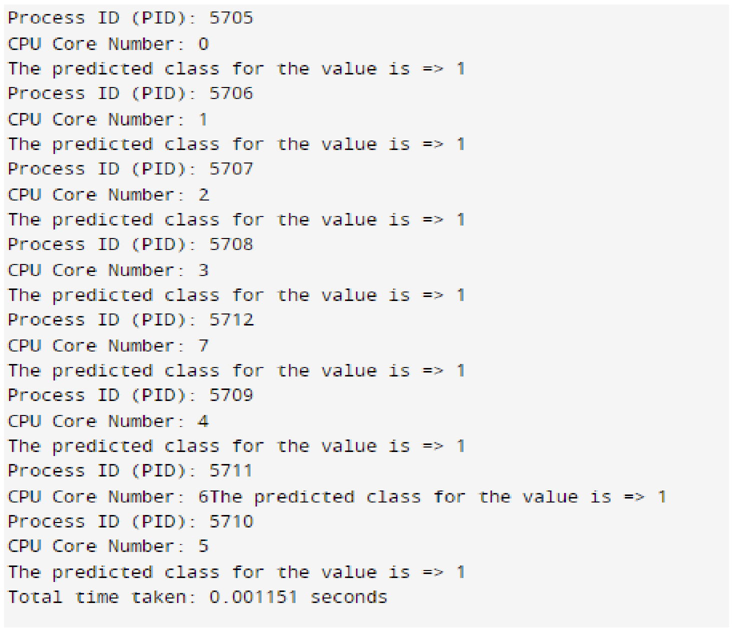
K-NN Model

```
#include <iostream>
#include <stdio.h>
#include <cmath>
#include <algorithm>
#include <unistd.h>
#include <sched.h>
#include <sys/sysctl.h>
#include <sys/types.h>
#include <ctime>
#include <sys/wait.h>  // Include the wait header

using namespace std;

struct Point {
    int val;
    double x, y;
    double distance;
};

bool comparison(Point a, Point b) {
    return (a.distance < b.distance);
}

int classifyAPoint(Point arr[], int n, int k, Point p) {
    for (int i = 0; i < n; i++) arr[i].distance = sqrt((arr[i].x - p.x) * (arr[i].x - p.x) + (arr[i].y - p.y) * (arr[i].y - p.y));
    sort(arr, arr + n, comparison);
    int freq1 = 0;
    int freq2 = 0;
    for (int i = 0; i < k; i++) {
        if (arr[i].val == 0) freq1++;
        else if (arr[i].val == 1) freq2++;
    }
    return (freq1 > freq2 ? 0 : 1);
}

int main() {
    int n = 17;
    Point arr[n];
arr[0].x = 1;
    arr[0].y = 12;
    arr[0].val = 0;

    arr[1].x = 2;
    arr[1].y = 5;
    arr[1].val = 0;

    arr[2].x = 5;
    arr[2].y = 3;
    arr[2].val = 1;

    arr[3].x = 3;
    arr[3].y = 2;
    arr[3].val = 1;

    arr[4].x = 3;
    arr[4].y = 6;
```

```
arr[4].val = 0;

arr[5].x = 1.5;
arr[5].y = 9;
arr[5].val = 1;

arr[6].x = 7;
arr[6].y = 2;
arr[6].val = 1;

arr[7].x = 6;
arr[7].y = 1;
arr[7].val = 1;

arr[8].x = 3.8;
arr[8].y = 3;
arr[8].val = 1;

arr[9].x = 3;
arr[9].y = 10;
arr[9].val = 0;

arr[10].x = 5.6;
arr[10].y = 4;
arr[10].val = 1;

arr[11].x = 4;
arr[11].y = 2;
arr[11].val = 1;

arr[12].x = 3.5;
arr[12].y = 8;
arr[12].val = 0;

arr[13].x = 2;
arr[13].y = 11;
arr[13].val = 0;

arr[14].x = 2;
arr[14].y = 5;
arr[14].val = 1;

arr[15].x = 2;
arr[15].y = 9;
arr[15].val = 0;

arr[16].x = 1;
arr[16].y = 7;
arr[16].val = 0;

// Initialize the array of points (your previous code)

pid_t pid;
int k = 5;
int num_cores = sysconf(_SC_NPROCESSORS_ONLN); // Get the number of available CPU cores

clock_t start_time = clock();  // Record the start time

for (int core = 0; core < num_cores; core++) {
    pid = fork();
```

```
        if (pid == 0) {
            // This is the child process
            cpu_set_t mask;
            CPU_ZERO(&mask);
            CPU_SET(core, &mask);

            if (sched_setaffinity(0, sizeof(mask), &mask) == -1) {
                perror("sched_setaffinity");
            }
            // Now each child process can work on a specific core
            printf("Process ID (PID): %d\n", getpid());
            printf("CPU Core Number: %d\n", sched_getcpu());

            // You can call classifyAPoint here with the data you want to classify
            Point p;
            p.x = 2.5;
            p.y = 4;

            printf("The predicted class for the value is => %d\n", classifyAPoint(arr, n, k, p));
            return 0;
        } else if (pid < 0) {
            perror("fork");
        }
    }

    // The parent process waits for all child processes to finish
    int status;
    for (int i = 0; i < num_cores; i++) {
        wait(&status);
    }

    clock_t end_time = clock();  // Record the end time
    double elapsed_time = double(end_time - start_time) / CLOCKS_PER_SEC;  // Calculate elapsed time in seconds

    cout << "Total time taken: " << elapsed_time << " seconds" << endl;

    return 0;
}
```

Output



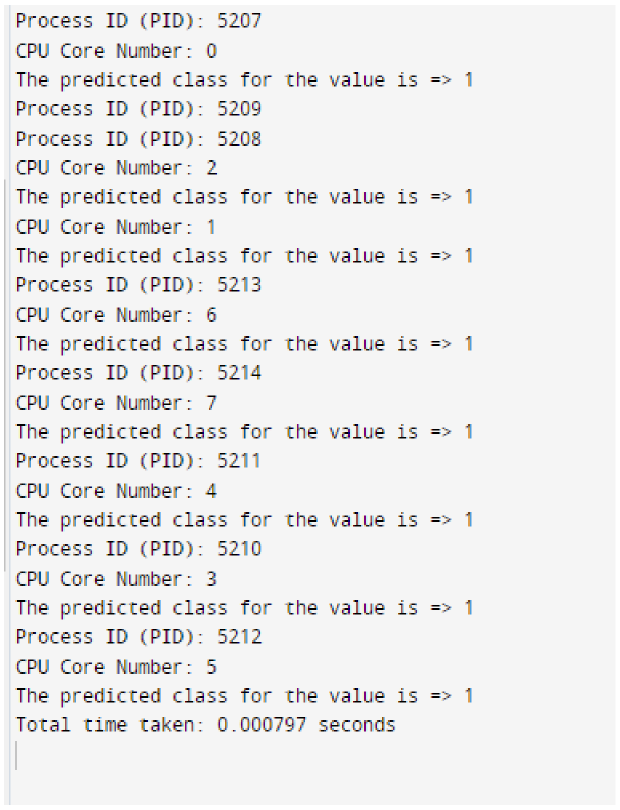
```
Process ID (PID): 5705
CPU Core Number: 0
The predicted class for the value is => 1
Process ID (PID): 5706
CPU Core Number: 1
The predicted class for the value is => 1
Process ID (PID): 5707
CPU Core Number: 2
The predicted class for the value is => 1
Process ID (PID): 5708
CPU Core Number: 3
The predicted class for the value is => 1
Process ID (PID): 5712
CPU Core Number: 7
The predicted class for the value is => 1
Process ID (PID): 5709
CPU Core Number: 4
The predicted class for the value is => 1
Process ID (PID): 5711
CPU Core Number: 6The predicted class for the value is => 1
Process ID (PID): 5710
CPU Core Number: 5
The predicted class for the value is => 1
Total time taken: 0.001151 seconds
```

SVM- Model

```
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>
#include <stdio.h>
#include <unistd.h>
#include <sched.h>
#include <sys/sysctl.h>
#include <sys/types.h>
#include <ctime>
#include <sys/wait.h>
using namespace std;

struct Point {
   double x, y;
   int label;
};

struct SVMModel {
   double w1, w2; // Weights for the linear SVM
   double b;      // Bias term
};

// SVM training function (basic linear SVM)
SVMModel trainSVM(vector<Point>& data) {
   SVMModel model;

   double learningRate = 0.1;
   int epochs = 1000;

   // Initialize weights and bias
   model.w1 = 0.0;
   model.w2 = 0.0;
   model.b = 0.0;
```

```
    for (int epoch = 0; epoch < epochs; epoch++) {
        for (const Point& point : data) {
            double margin = point.label * (model.w1 * point.x + model.w2 * point.y + model.b);

            if (margin < 1) {
                model.w1 += learningRate * (point.label * point.x - 2 * model.w1);
                model.w2 += learningRate * (point.label * point.y - 2 * model.w2);
                model.b += learningRate * point.label;
            }
        }
    }

    return model;
}

// Predict the class of a new point using the trained SVM model
int predictClass(const SVMModel& model, double x, double y) {
    double margin = model.w1 * x + model.w2 * y + model.b;
    return (margin >= 0) ? 1 : -1;
}

int main() {
    // Sample data
    vector<Point> data = {
        {1, 12, 1},
        {2, 5, -1},
        {5, 3, -1},
        {3, 2, -1},
        {3, 6, 1},
        {1.5, 9, -1},
        {7, 2, -1},
        {6, 1, -1},
        {3.8, 3, -1},
        {3, 10, 1},
        {5.6, 4, -1},
        {4, 2, -1},
        {3.5, 8, 1},
        {2, 11, 1},
        {2, 5, -1},
        {2, 9, 1},
        {1, 7, 1}
    };

    // Train the SVM model
    SVMModel model = trainSVM(data);
    pid_t pid;

    int num_cores = sysconf(_SC_NPROCESSORS_ONLN); // Get the number of available CPU cores

    clock_t start_time = clock();  // Record the start time

    for (int core = 0; core < num_cores; core++) {
        pid = fork();

        if (pid == 0) {
            // This is the child process
            cpu_set_t mask;
            CPU_ZERO(&mask);
            CPU_SET(core, &mask);

            if (sched_setaffinity(0, sizeof(mask), &mask) == -1) {
                perror("sched_setaffinity");
            }

            // Now each child process can work on a specific core
            printf("Process ID (PID): %d\n", getpid());
            printf("CPU Core Number: %d\n", sched_getcpu());

            // You can call classifyAPoint here with the data you want to classify

    double testX = 2.5;
    double testY = 4;
    int predictedClass = predictClass(model, testX, testY);
            printf("The predicted class for the value is => %d\n", predictedClass);
            return 0;
        } else if (pid < 0) {
            perror("fork");
        }
    }

    // The parent process waits for all child processes to finish
    int status;
    for (int i = 0; i < num_cores; i++) {
        wait(&status);
    }

    clock_t end_time = clock();  // Record the end time
    double elapsed_time = double(end_time - start_time) / CLOCKS_PER_SEC;  // Calculate elapsed time in seconds

    cout << "Total time taken: " << elapsed_time << " seconds" << endl;



    return 0;
}
```

Output



```
Process ID (PID): 5207
CPU Core Number: 0
The predicted class for the value is => 1
Process ID (PID): 5209
Process ID (PID): 5208
CPU Core Number: 2
The predicted class for the value is => 1
CPU Core Number: 1
The predicted class for the value is => 1
Process ID (PID): 5213
CPU Core Number: 6
The predicted class for the value is => 1
Process ID (PID): 5214
CPU Core Number: 7
The predicted class for the value is => 1
Process ID (PID): 5211
CPU Core Number: 4
The predicted class for the value is => 1
Process ID (PID): 5210
CPU Core Number: 3
The predicted class for the value is => 1
Process ID (PID): 5212
CPU Core Number: 5
The predicted class for the value is => 1
Total time taken: 0.000797 seconds
```

7. Reference Material

- GeeksforGeeks: https://www.geeksforgeeks.org/

- ChatGPT: https://www.openai.com/research/gpt-3/

- Machine Learning Book: "Pattern Recognition and Machine Learning" by Christopher M. Bishop