Support Vector Machine Regression for forecasting electricity demand for large commercial buildings by using kernel parameter and storage effect

Submitted By
Gurmeet Singh(IIT2017073)
*Indian Institute Of Information Technology,Allahabad

Abstract—**This Paper Provides the details about the forecasting tool based on support vector regression analysis.In order to optimize benefits and utilities we need ti know energy forecasting of the building.The real world case studies have been done developed and tested on four randomly selectd buildings using their monthly utility bills.We will measure our performance of SVM with respect to three parameters C,γ and ε.The analysis shows that all the forecasting models give acceptable result for all four buildings with low percentage error.**

I. Introduction

In a Matrix the coordinates having same x coordinate are horizontally aligned and the coordinates having same y coordinates are vertically aligned.Since searching horizontal and vertical points are independent events ,it can be done in parallel.

One thread looks for the points which are horizontally aligned while in parallel another thread runs which looks for the vertically aligned points. After both the threads are done completing their work, both are joined to get the desired output

Here , We work in parallel ways as , at the same time our program looks for both horizontal and vertically aligned points at the same time , out time reduces to almost half of the regular program.

This report further investigates the:

II Algorithmic Design.
III Algorithmic Analysis.
IV Experimental Study.
V Parallel Algorithm Discussion
VI Conclusion.
VII References.

II. Algorithmic Design

At First , we take the input from the user and store it in our user constructed struct which consists of x,y and val(basically a flag variable). A thread is created and is given to fhorizontal function while another thread is also created which is given the fvertical function. At last, both are joined to get the desired output.

Part1:

1.Take user input and store in struct array with val=0.

Part2:
2.Create one thread and assign it to the horizontal aligned searching points function.

Part3:
3.Create another thread and assign it to the vertical aligned searching points function.

Part4:
4.After both thread have completed their work,Join them.

Part5:
5.After threads are joined ,desired output is achieved.

III. Algorithmic Analysis

Pseudocode 1 *fhorizontal(p[])*

```
1:  for i ←0 to n do
2:      count ←0
3:      p[i].val=1
4:      for j ←i+1 to n  p[j].val==0 do
5:          if p[i].x== p[j].x then
6:              count ←count +1
7:              p[j].val== 1
8:              print points j
9:      if count > 0 then
10:         print point i
```

Explaination:

Looping over i till n , at first we initialised count to 0 and made the current coordinates flag as "visited". Now , by looping over suitable indexes we searched for it's horizontally aligned points. If we found a horizontally aligned point,we marked it as visited and printed it out. Finally by checking the count value we print the current point's value.Since we are marking each point as visited or not,we'll be able to compute all the points without repeating any point.

Pseudocode 2 *fvertical(p[])*

```
for i ←0 to n do
    count ←0
    p[i].val=1
    for j ←i+1 to n  p[j].val==0 do
        if p[i].x== p[j].x then
            count ←count +1
            p[j].val== 1
            print points j
    if count > 0 then
        print point i
=0
```

Explaination:
Looping over i till n , at first we initialised count to 0 and made the current coordinates flag as "visited". Now , by looping over suitable indexes we searched for it's vertically aligned points. If we find a vertically aligned point,we marked it as visited and printed it out. Finally by checking the count value we print the current point's value.Since we are marking each point as visited or not,we'll be able to compute all the points without repeating any point.

IV. Experimental Study

The Best case will occur when all the points given are either horizontally or vertically alligned. The worst case will occur when the points are neither horizontally nor vertically alligned. Average case lies between best and worst. In the best case, if all the points are horizontally or vertically alligned, the visited index of each element will be 1 in first iteration only. So the complexity is O(n). But in the worst case, all combinations of two points will be checked, hence the time complexity is O(n*n).

By using the parallel programming, we reduced the time complexity of the problem. The fhorizontal() and fvertical() are computed at the same time.So the time reduces by half by using parallel programming(via threads).

complexity of algorithm:
$t_{best}\alpha$ n
$t_{worst}\alpha$ n*n
$t_{avg}\alpha$ Between best and worst case

Time Complexity

Best Case: $\Omega(n)$
$WorstCase: O(n^2)$

$AverageCase: \Theta(n^2)$

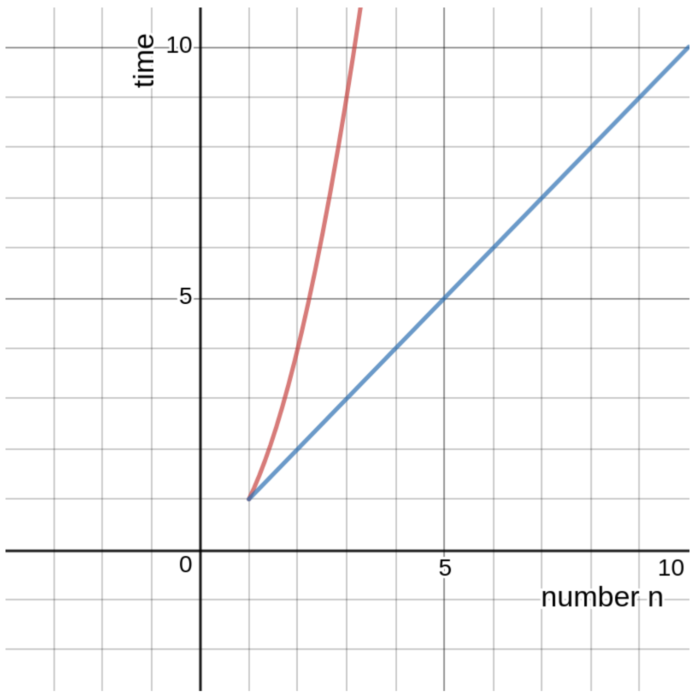

Figure 1. Time v/s length of array

V. Conclusion

Reducing energy consumption wasting of energy is affecting mankind.In near future energy can be a limiting factor.So this research is primarily focused on reducing energy consumption and environmental pollution.While building undergoes it's retrofitting then prediction of energy consumption is necessary. .

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, "Introduction to Algorithms,3Ed.(International Edition) (The MIT Press)".
[2] Concept of parallel programming and finding horizontally and vertically alligned points optimally. ,"www.geeksforgeeks.org".
[3] Concept of parallel programming ,"www.wikipedia.org".