Use KNN for KDD'99

Kazi Abir Adnan (2014280162)

Wednesday 29th April, 2015

Introduction

Use KNN to solve the KDDCup 1999

Algorithm

Algorithm 1 KDD 99

```
procedure KNN
    $TrainData$ ← Load All Train Data
    $TestData$ ← Load All Test Data
    $float[41]max$ ← Find max value to normalize
    $float[41]min$ ← Find min value to normalize
    $TrainData$ ← Normalize Train Data
    $TestData$ ← Normalize Test Data
    $BestDistance$ ← smallest distance points for each TestData
    for <each instance in Test Data> do
        for <each instance in Train Data> do
            $Distance$ ← distances of train and test data
            $BestDistance$ ← Best Distances in ascending order
    Print accuracy matrix for labelled or unlabelled test data
```

Environment

- Java
- JDK 1.8

Read Me

- You can provide labeled or unlabeled test data. If you provide labeled test data output will be accuracy matrix. And if you provide unlabeled data output will be count of each attack.
- Make sure you have JDK installed. You can check it by typing ”java -version” in cmd.
- Go to the folder containing the jar file from cmd.
- Run the .jar(KNN.jar) file using the following command: java -jar [Jar file Name with jar extension] [Training Data File Location] [Testing Data File Location]
- The input file (training_attack_types) must be in the same folder.

Result

Labeled Data

I have used 50% of the 10% train data to train and used 50% of the rest of the data to test the system.

- Train Data Size : 247012
- Test Data Size : 123506
- Running Time : 23 minutes

FOR K = 1

	Actual	normal	u2r	r21	probe	dos	accuracy
Predicted	-	-	-	-	-	-	-
normal	|	5987	0	13	4	2	99.68%
u2r	|	0	4	0	0	0	100.00%
r21	|	0	0	0	0	0	0.0%
probe	|	12	0	0	464	1	97.27%
dos	|	29	0	0	75	116915	99.91%
accuracy	|	99.31%	100%	0.0%	85.45%	99.99%	

FOR K = 3

	Actual	normal	u2r	r21	probe	dos	accuracy
Predicted	-	-	-	-	-	-	-
normal	|	5990	0	11	3	2	99.74%
u2r	|	0	4	0	0	0	100.00%
r21	|	0	0	0	0	0	0.0%
probe	|	21	0	0	455	1	95.39%
dos	|	8	0	0	75	116936	99.93%
accuracy	|	99.51%	100%	0.0%	85.36%	99.99%	

Unlabeled Data

- Train Data Size : 494021 (kddcup.data_10_percent)
- Test Data Size : 311079 (kddcup.newtestdata_10_percent_unlabeled)
- Running Time : 100 mins

FOR K = 1

normal	u2r	r21	probe	dos
83017	79	861	3734	223388

FOR K = 3

normal	u2r	r21	probe	dos
82708	42	1028	3815	223486

Analysis

- Read and parsed test and train data (O(n), from disk).
- Normalized data for test and train(O(n)). I got the max and min values to normalize the data while train data loaded. So no need to scan whole array to find the max or min value. Used also Threading.
- Calculated the distances(scan, floating point operations and double point operation including sqrt) using Threading. I have calculated three best points with smallest distances here. So, no need to sort and find K smallest distances anymore scanning the results.
- Calculated accuracy using Threading.

Improvement

- As data are normalized, the range of value is from 0 to 1. So, we do not need to use float data type for this small range of data. If we can use more small size data type I think the program will be more faster.
- I couldn't find single precision SQRT function in Java. I had to use double precision SQRT function to calculate Euclidian distance. Otherwise it would be more faster.