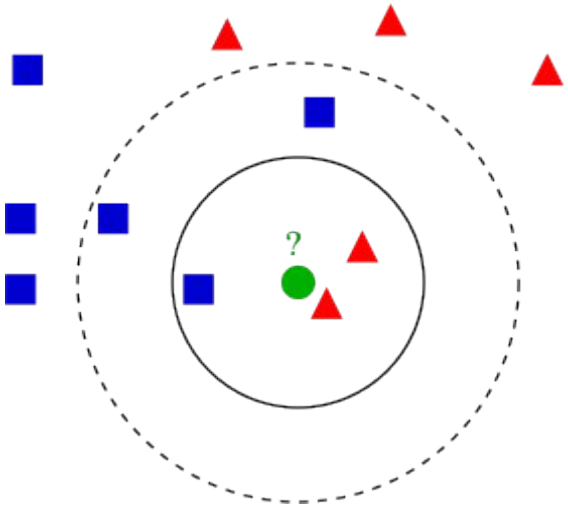
KNN ALGORITHM

KNN ALGORITHM



- CLASSIFICATION (SUPERVISED).
- High dimensional feature space.
- Find distance between all points in the space. And assign the class label as the most frequent among the k nearest train data.
- SIMPLE ALGORITHM yet SO POWERFUL.
- Has very high accuracies than most of state of the art algorithms and easy to implement.
- What's wrong?

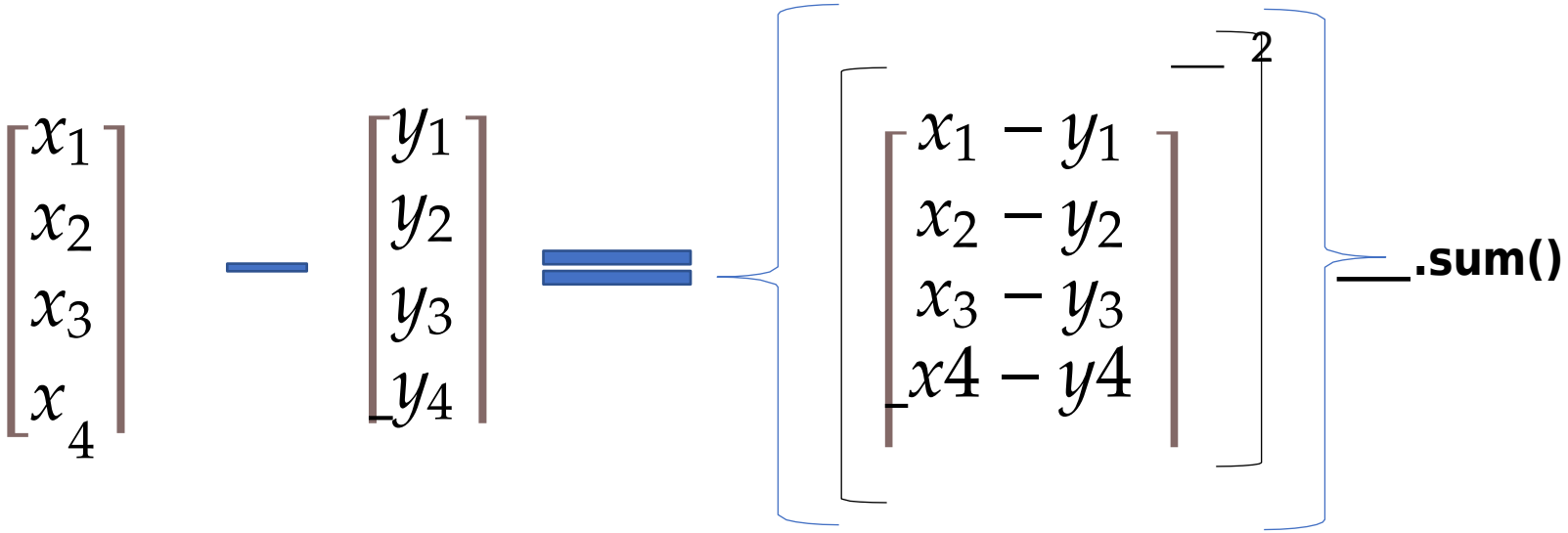
Problems

- Distance is to be calculated for all the points in the HD space.
- Euclidian distance itself is time consuming process.
- KNN is computationally very expensive.
- distance calculation is bottleneck for KNN.
- paralysing the distance calculations saves a lot of time.

- 1. Select 1 processor to be the master, the other N-processors are slaves.
- 2. Master divides the training samples to N subsets, and distributes 1 subset for each processor, keeping 1 subset for local processing (Master participates in distance computation too).
- 3. Each individual processor now computes the distance measures independently and storing the computes measures in a local array
- Time complexity drops from O(n) to O(n/p)

DISTANCE(EUCLEADIAN)

:



We can paralyze further for each process by assigning it into threads. As there is no synchronization problems here .In fact NumPy arrays are paralyzed and optimized internally

Parallelizing sort finally:

- Will use radix sort/quick sort and paralyze it

EVALUATION METRICS:

1. CPU paralyzed execution time vs GPU paralysed execution time
2. SPEEDUP: Ratio between sequential knn vs paralyzed knn
3. EFFICIENCY: S/p
4. Overhead: To = p TP - TS

RESEARCH PAPERS:

- CUKNN: A parallel implementation of K-nearest neighbor on CUDA-enabled GPU

- A Practical GPU Based KNN Algorithm Quansheng Kuang, and Lei Zhao*