# STAT4402 Tutorial

Michael Ciccotosto-Camp
*University of Queensland*

## 1. Introduction

Most computational work done within science is impractical to perform on commercial laptops and desktops, typically due to the extremely high memory and processing demands. Hence, almost every university and industry has its own high-performance computer to carry out such strenuous problems. A lot of research within the realm of Machine Learning benefits from access to high performance machinery as most of them require matrix computation (which can be efficiently carried out on GPU clusters) and can be processed in parallel.

In this tutorial we will revisit two models covered in the first few weeks of lectures, these being the SDG linear regressor and KNN classifier. The naive implementation of both these algorithms are fairly inefficient, so we shall look at some ways in which these two methods can be decomposed and parallelised.

## 2. Parallel KNN

- The $k^{th}$ Nearest-Neighbor (k-NN) methods use observations in the training set $T$ closest in feature space to a given unknown sample $\boldsymbol{x}$ to directly find its corresponding prediction $\overline{y}$
- The prediction for the k-NN classifier is usually calculated as

$$\overline{y}\left(\boldsymbol{x}\right) = \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} y_i$$

- The notion of 'clostest' implies the use of some sort of meteric. More often than not, feature vectors belong to $\mathbb{R}^n$ allowing us to use commonly used metrics to define distance between vectors in our feature space. For our purposes, we shall use the

Euclidean norm as a measurement of determining how close two feature values are to each other. The Euclidean norm is simply defined as

$$d(\boldsymbol{x}, \boldsymbol{y}) = \left( \sum_{i=1}^{n} (x_i - y_i)^2 \right)^{\frac{1}{2}}$$

- When a unknown sample $\boldsymbol{x}$ is to be classified, a k-NN classifier computes the distance between $\boldsymbol{x}$ and the other points within the training set $T$. The training data is then sorted by distance and the $k^{th}$ closests training samples are then used to predict $\boldsymbol{x}$.
- A simple K-NN algorithm works as follows

**Algorithm 1:** Serial k-NN

**input :** Training data $T$, an unlabelled sample $\boldsymbol{x}$ and a value $k$
**output:** Predicted class $\overline{y}(\boldsymbol{x})$

1 Computes distance $d(\boldsymbol{x}, \boldsymbol{x}_{t_i})$ for each $\boldsymbol{x}_{t_i} \in T$;
2 $N_k(\boldsymbol{x}) \leftarrow$ the $k^{th}$ closest $\boldsymbol{x}_{t_i}$ determined by $d(\boldsymbol{x}, \boldsymbol{x}_{t_i})$;
3 $\overline{y}(\boldsymbol{x}) \leftarrow \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} y_i$;
**Result:** $\overline{y}(\boldsymbol{x})$

- While this method is simple, computing the distance between $\boldsymbol{x}$ and each $\boldsymbol{x}_{t_i} \in T$ can incur a large time overhead, especially for large training sets
- One important observation is that computing the distances $d(\boldsymbol{x}, \boldsymbol{x}_{t_i})$ and $d\left(\boldsymbol{x}, \boldsymbol{x}_{t_j}\right)$ where $\boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_j} \in T$ and $i \neq j$ may be done completely independently of each other meaning these computations may be carried out on separate processes.
- CITATION takes advantage of this independence to have distance computation carried out on different processes.

This algorithm is shown pictorially below

## 3. Parallel SGD

- As before let $T$ a set of training samples $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{m} = \{z_i\}_{i=1}^{m}$
- Let $C(\boldsymbol{w})$ be the cost function

$$\arg\min_{\boldsymbol{w} \in \mathbb{R}^n} \sum_{i=0}^{m} C_{z_i}(\boldsymbol{w})$$

**Algorithm 2:** Parallel k-NN

**input :** Training data $T$, an unlabelled sample $\boldsymbol{x}$, a value $k$ and the number of processes to perform the algorithm $p$

**output:** Predicted class $\overline{y}(\boldsymbol{x})$

$\{T_1, T_2, \ldots, T_p\} \leftarrow$ an equal partition of $T$;

**for** $T_i \in \{T_1, T_2, \ldots, T_p\}$ **concurrently do**

  $N_{k_i}(\boldsymbol{x}) \leftarrow$ the $k^{th}$ nearest neighbors from $T_i$;

**end**

$N_k(\boldsymbol{x}) \leftarrow$ the $k^{th}$ closest neighbors from $N_{k_1}(\boldsymbol{x}), N_{k_2}(\boldsymbol{x}), \ldots N_{k_p}(\boldsymbol{x})$;

$\overline{y}(\boldsymbol{x}) \leftarrow \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} y_i$;

**Result:** $\overline{y}(\boldsymbol{x})$