Ciência de Dados Quântica 2021/22

Kernel Based Methods:

Fundamentals

LUÍS PAULO SANTOS

Material de Consulta

- [Schuld2021] – Secs. 2.5.4, 3.6.1; Chap. 6
- "Quantum Unsupervised and Supervised Learning on Superconducting Processors"; A. Sarma, R. Chatterjee, K. Gili, and T. Yu
  arXiv: quantum-ph, 2022
  https://arxiv.org/pdf/1909.04226.pdf
- "Building a quantum kNN classifier with Qiskit: theoretical gains put to practice"; D.J. Kok; MsC Thesis RadBoud University, 2021
  https://www.ru.nl/publish/pages/913395/daniel_kok_4_maart_2021.pdf
- "Quantum k-nearest neighbor machine learning algorithm"; Afham, Afrad Basheer and Sandeep K. Goyal ; In: arXiv: 2003.09187 [quant-ph]
  https://arxiv.org/pdf/2003.09187.pdf

Kernel Methods: concept

- Kernel methods are based on a **similarity measure** between data points
- **Definition:** for a data domain χ a kernel is a positive semi-definite bivariate function $\varkappa : \chi \times \chi \longrightarrow \mathbb{R}$
  - positive semi-definite means:
    - $\varkappa(x, x') \geq 0$
    - $\varkappa(x, x') = \varkappa(x', x)^*$

Examples of classical kernels

Name	Kernel
Linear	$\boldsymbol{x}^T\boldsymbol{x}'$
Gaussian	$e^{\lVert x-x' \rVert^2}$
Sigmoid	$\tanh(\boldsymbol{x}^T\boldsymbol{x}' + c)$
Euclidean distance	$\sqrt{\sum_{i=1}^{N}(x_i - x_i')^2}$
Hamming distance	Different bits $(\boldsymbol{x}, \boldsymbol{x}')$

Quantum Kernel: states overlap

- $|\langle \boldsymbol{\varphi} | \boldsymbol{\phi} \rangle|^{\mathbf{2}}$ - The absolute square value of the inner product of two quantum states, called the **overlap**, can be used as a **measure of similarity** between $\boldsymbol{\varphi}$ and $\boldsymbol{\phi}$
  - $\varphi = \phi \Rightarrow |\langle \varphi | \phi \rangle|^2 = 1$
  - $\varphi \perp \phi \Rightarrow |\langle \varphi | \phi \rangle|^2 = 0$

- The SWAP test is commonly used to measure the overlap

SWAP test

- Let $|\varphi\rangle$ and $|\phi\rangle$ be two n qubits quantum states and consider an additional single qubit ancilla: $|0\rangle|\varphi\rangle|\phi\rangle = |0\rangle\otimes|\varphi\rangle\otimes|\phi\rangle$
- Apply an Hadamard to the ancilla:

$$(H\otimes\mathbb{I}^n\otimes\mathbb{I}^n)|0\rangle|\varphi\rangle|\phi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|\varphi\rangle|\phi\rangle = \frac{1}{\sqrt{2}}(|0\rangle|\varphi\rangle|\phi\rangle + |1\rangle|\varphi\rangle|\phi\rangle)$$

- Swap $|\varphi\rangle$ and $|\phi\rangle$ conditioned on the ancilla being $|1\rangle$:

$$CSWAP\left(\frac{1}{\sqrt{2}}(|0\rangle|\varphi\rangle|\phi\rangle + |1\rangle|\varphi\rangle|\phi\rangle)\right) = \frac{1}{\sqrt{2}}(|0\rangle|\varphi\rangle|\phi\rangle + |1\rangle|\phi\rangle\varphi\rangle)$$

- Applying another Hadamard to the ancilla results in :

$$|\psi\rangle = (H\otimes\mathbb{I}^n\otimes\mathbb{I}^n)\left(\frac{1}{\sqrt{2}}(|0\rangle|\varphi\rangle|\phi\rangle + |1\rangle|\phi\rangle\varphi\rangle)\right)$$

$$= \frac{1}{2}|0\rangle\otimes(|\varphi\rangle|\phi\rangle + |\phi\rangle\varphi\rangle) + \frac{1}{2}|1\rangle\otimes(|\varphi\rangle|\phi\rangle - |\phi\rangle\varphi\rangle)$$

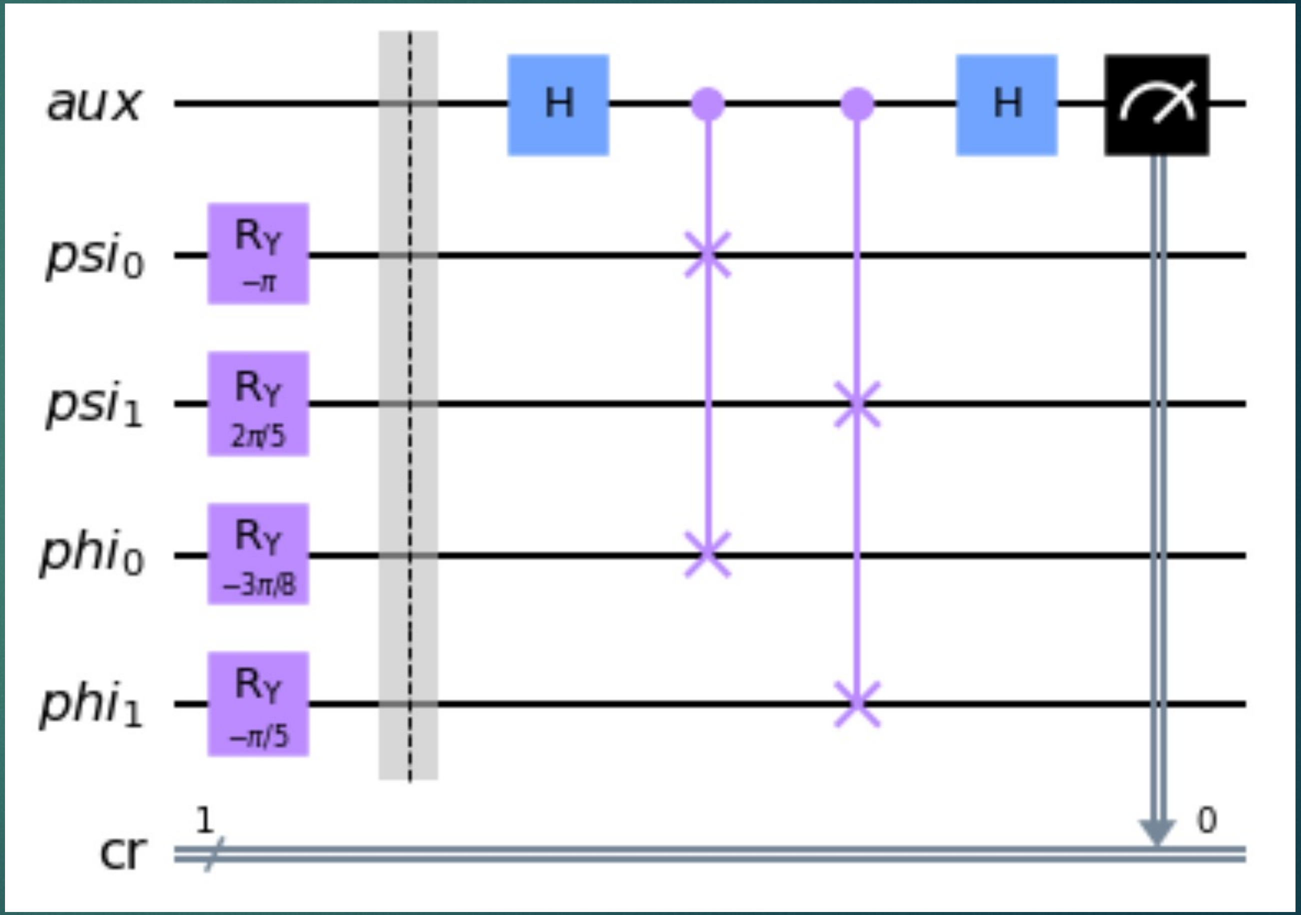
SWAP test

$$|\psi\rangle = \frac{1}{2}|0\rangle\otimes(|\varphi\rangle|\phi\rangle + |\phi\rangle\varphi\rangle) + \frac{1}{2}|1\rangle\otimes(|\varphi\rangle|\phi\rangle - |\phi\rangle\varphi\rangle)$$

- Let $p_0(|\psi\rangle)$ (respectively $p_1(|\psi\rangle)$) be the probability of measuring $|0\rangle$ (respectively $|1\rangle$) in the ancilla. It can be shown that:
  - $p_0(|\psi\rangle) = \frac{1}{2} + \frac{1}{2}|\langle\varphi|\phi\rangle|^2$ and $p_1(|\psi\rangle) = \frac{1}{2} - \frac{1}{2}|\langle\varphi|\phi\rangle|^2$
- Therefore $|\langle\boldsymbol{\varphi}|\boldsymbol{\phi}\rangle|^2 = \boldsymbol{p_0}(|\boldsymbol{\psi}\rangle)$ **-** $\boldsymbol{p_1}(|\boldsymbol{\psi}\rangle)$

SWAP test

```
def overlap:
    counts = execute_swap (shots)
    probs[0] = counts[0]/shots
    probs[1] = counts[1]/shots
    return probs[0] - probs[1]
```



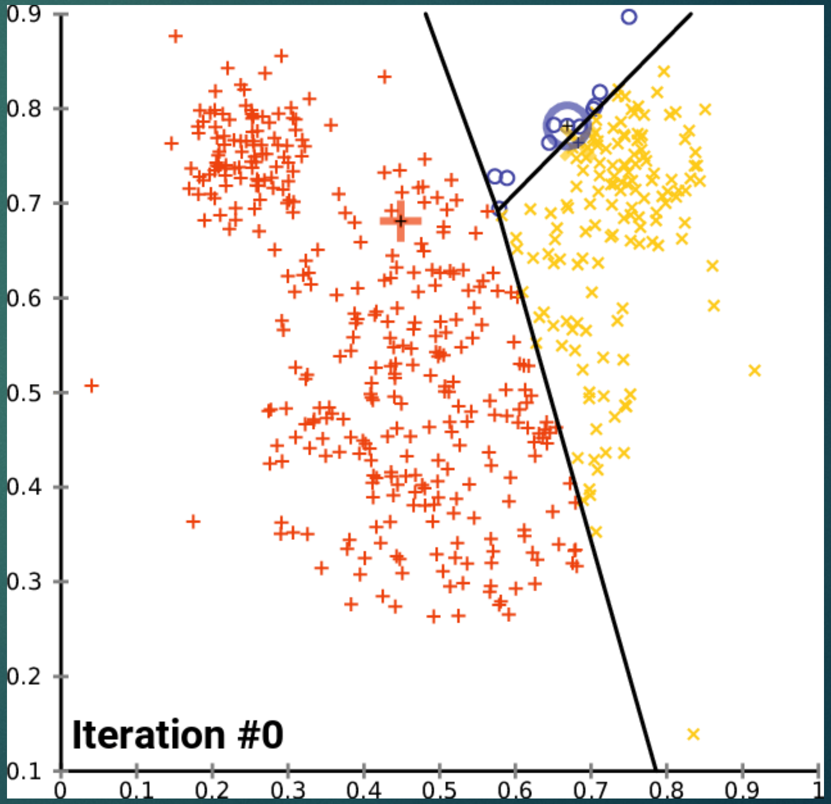
Clustering

- **clustering** is an unsupervised learning algorithm **partitioning** N data points (or feature vectors) $\boldsymbol{x}^i$ into subsets, or **clusters.**
  The algorithm seeks to find the clusters which **minimize the dissimilarity** between each cluster's members.

K-means clustering

- Given a data non-labelled data set $\mathcal{D} = \{\boldsymbol{x}^1, \cdots, \boldsymbol{x}^M\}$ of M data points and the number K of desired clusters, partition $\mathcal{D}$ into K subsets, minimizing the distance among cluster members.

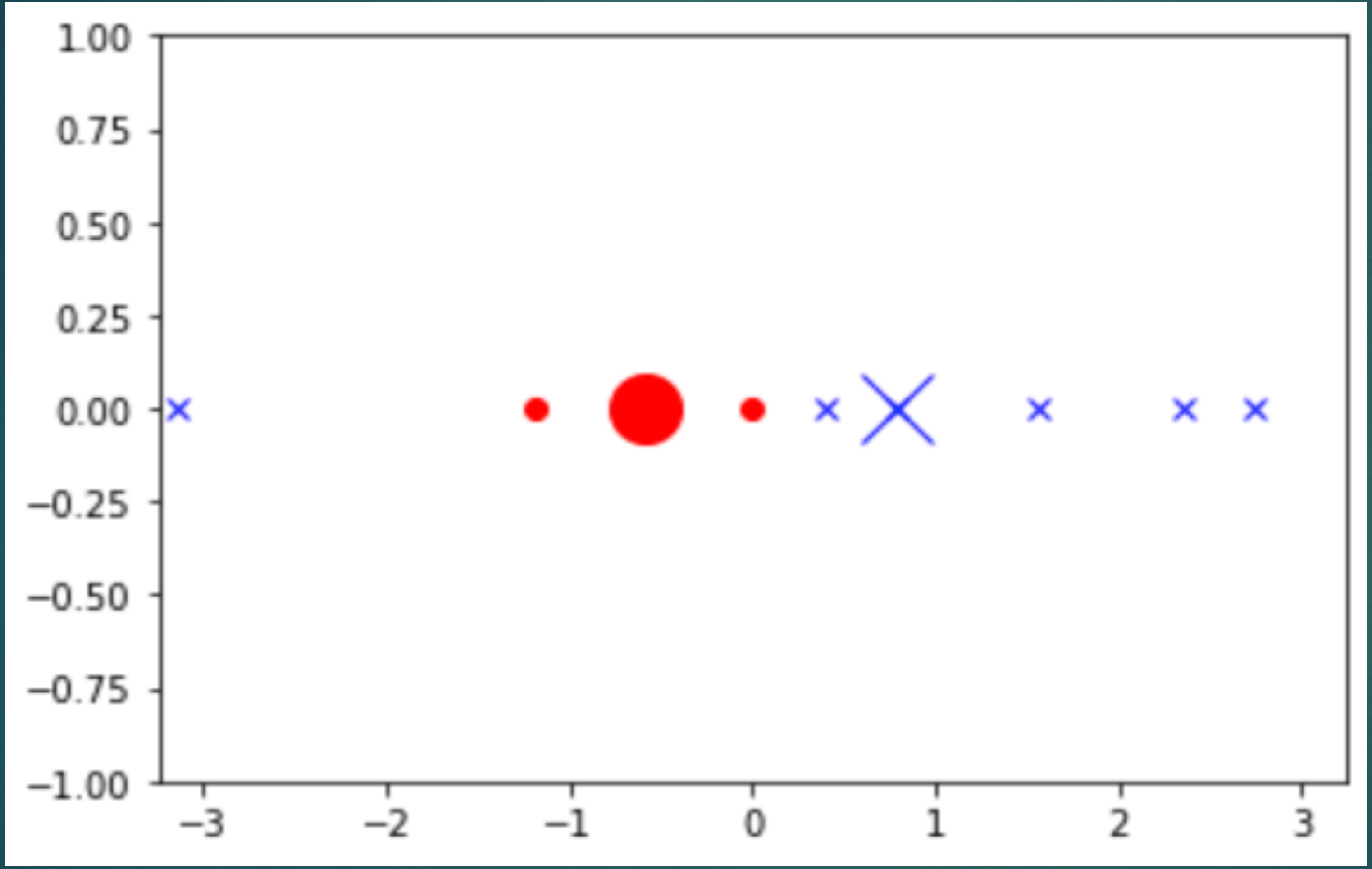
```
1. randomly select K centroids $C^k$ from $\mathcal{D} = \{\boldsymbol{x}^1, \cdots, \boldsymbol{x}^M\}$
2. while not stop
    1. for each $\boldsymbol{x}^i \in \mathcal{D}$
        1. for each cluster with centroid $C^k, k = 1 \cdots K$
            1. distance$_{i,k} = 1 - \boxed{\varkappa(C^k, x^i)}$ ⇐ QUANTUM
        2. assign $\boldsymbol{x}^i$ to cluster k with minimum distance$_{i,k}$
    2. for each cluster $k = 1 \cdots K$ compute the new centroid as the
       mean of all the cluster members
    3. if (new centroids == previous centroids) stop = True
```



K-means clustering

```
K = …                                                                   # set K

centroids = [$x^i$ for i in random.sample(range(len(D)),k=K)]          # initialize the centroids by randomly select K points within D

stop = False
iterations = 0
while not stop:
    iterations += 1
    clusters = [ [ ] for _ in range(K)]
    for x in D:
        distances = []                                                  # compute the distance of x to each centroid
        for centroid in centroids:
            distances.append(1-SWAP_test(centroid,x))        QUANTUM

        # find which centroid is at minimum distance
        index_min = np.argmin(distances)
        clusters[index_min].append(x)

    # compute new centroids
    centroid_change = False
    for ndx, cluster in enumerate(clusters):
        new_centroid = []                                               # average each feature across all points in the current cluster
        for f in range(n_features):
            avg_f = sum(member[f] for member in cluster)/len(cluster)
        new_centroid.append(avg_f)

      if new_centroid != centroids[ndx]:
        centroids[ndx] = new_centroid
        centroid_change = True

if not centroid_change: stop = True
```

K-means clustering



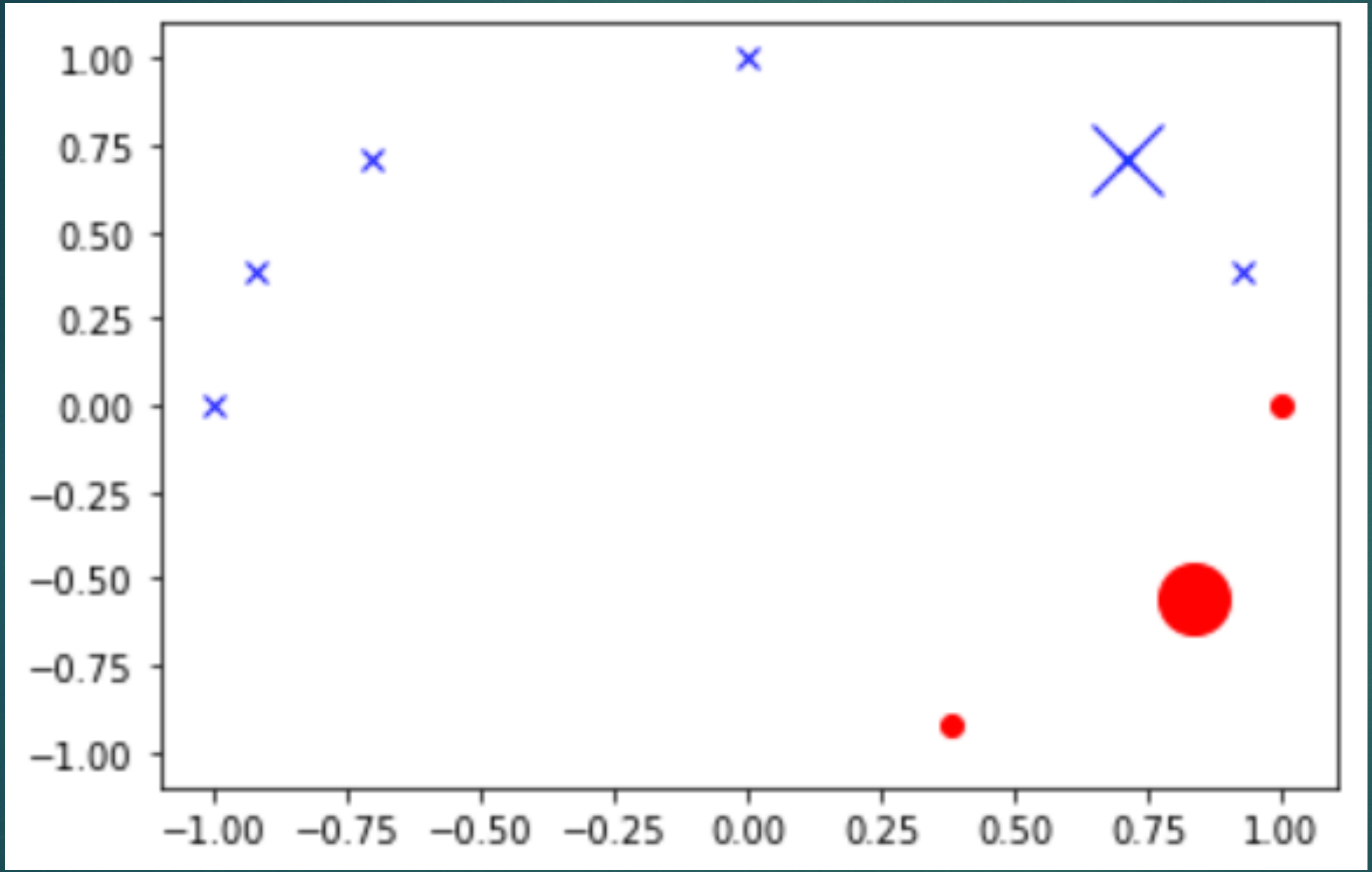
0 iterations

1 iterations

2 iterations

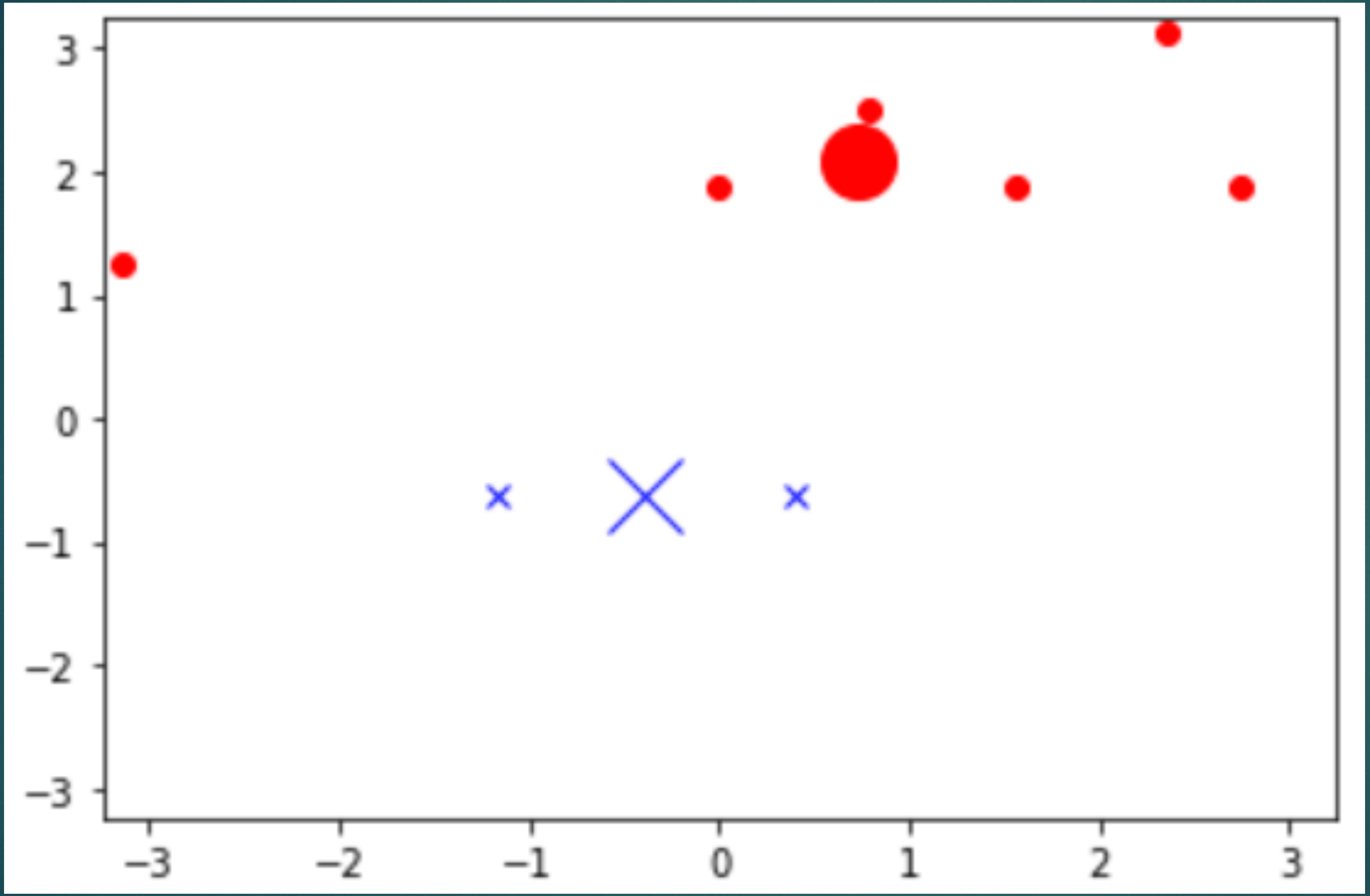
4 iterations

K-means clustering



4 iterations

K-means clustering



0 iterations

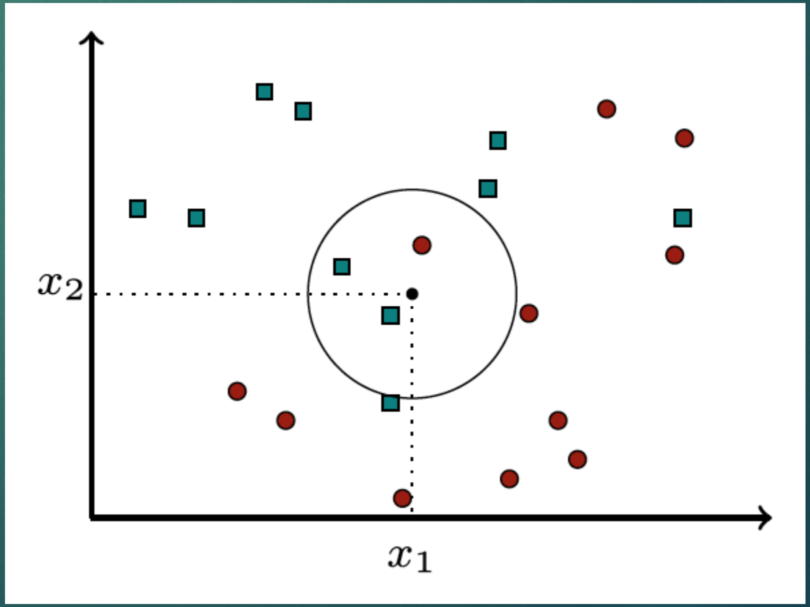
1 iterations

2 iterations

4 iterations

K-Nearest Neighbours

- Given a labelled dataset $\mathcal{D} = \{(\boldsymbol{x}^1, y^1), \cdots, (\boldsymbol{x}^M, y^M)\}$, select the K closest training inputs relative to the new input $\boldsymbol{x}$ according to the similarity metric of choice = $\varkappa(\boldsymbol{x}, \boldsymbol{x}^i)$
- The respective class y can be chosen as the majority class among the neighbours for a classification task

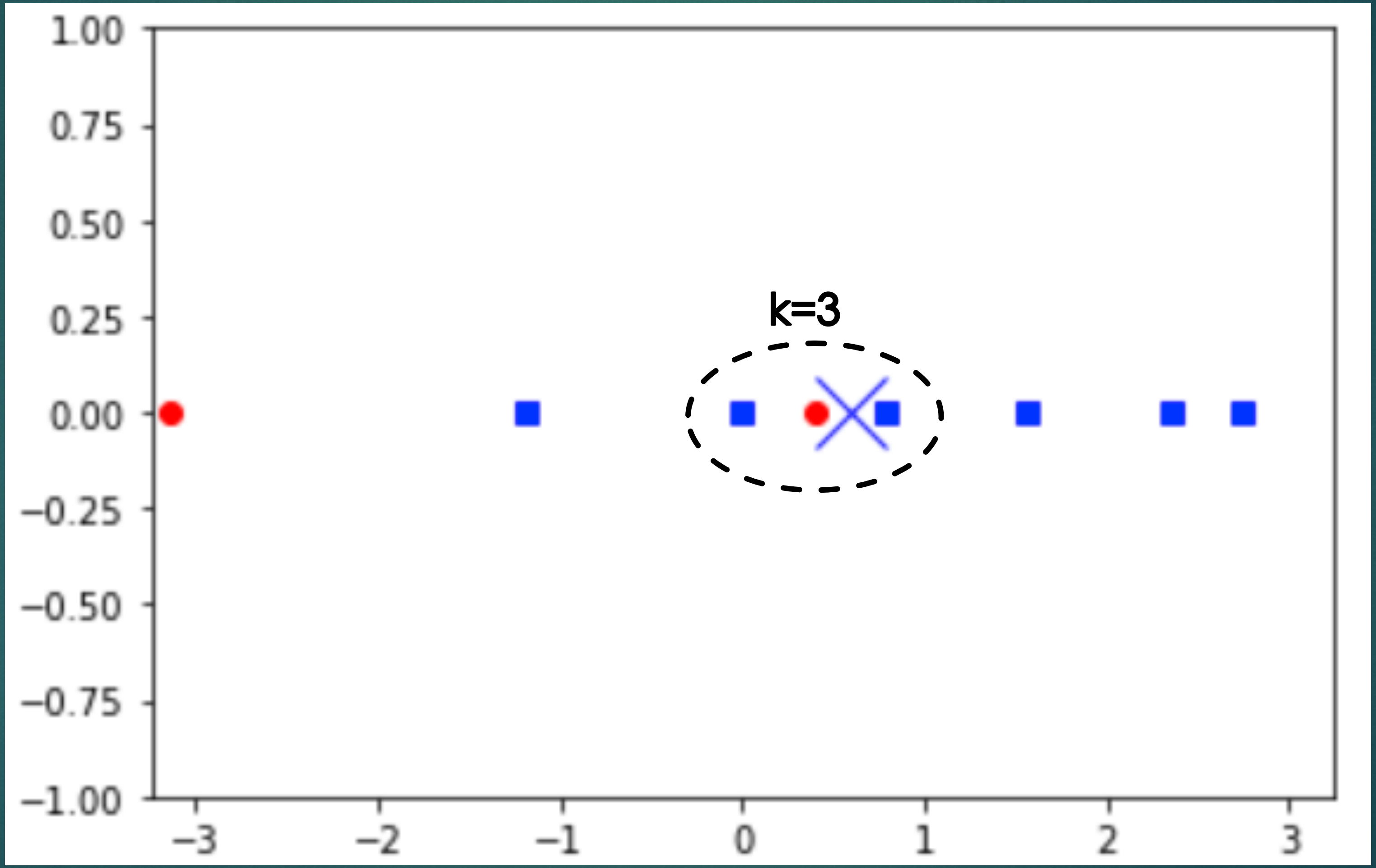


$k = 3$

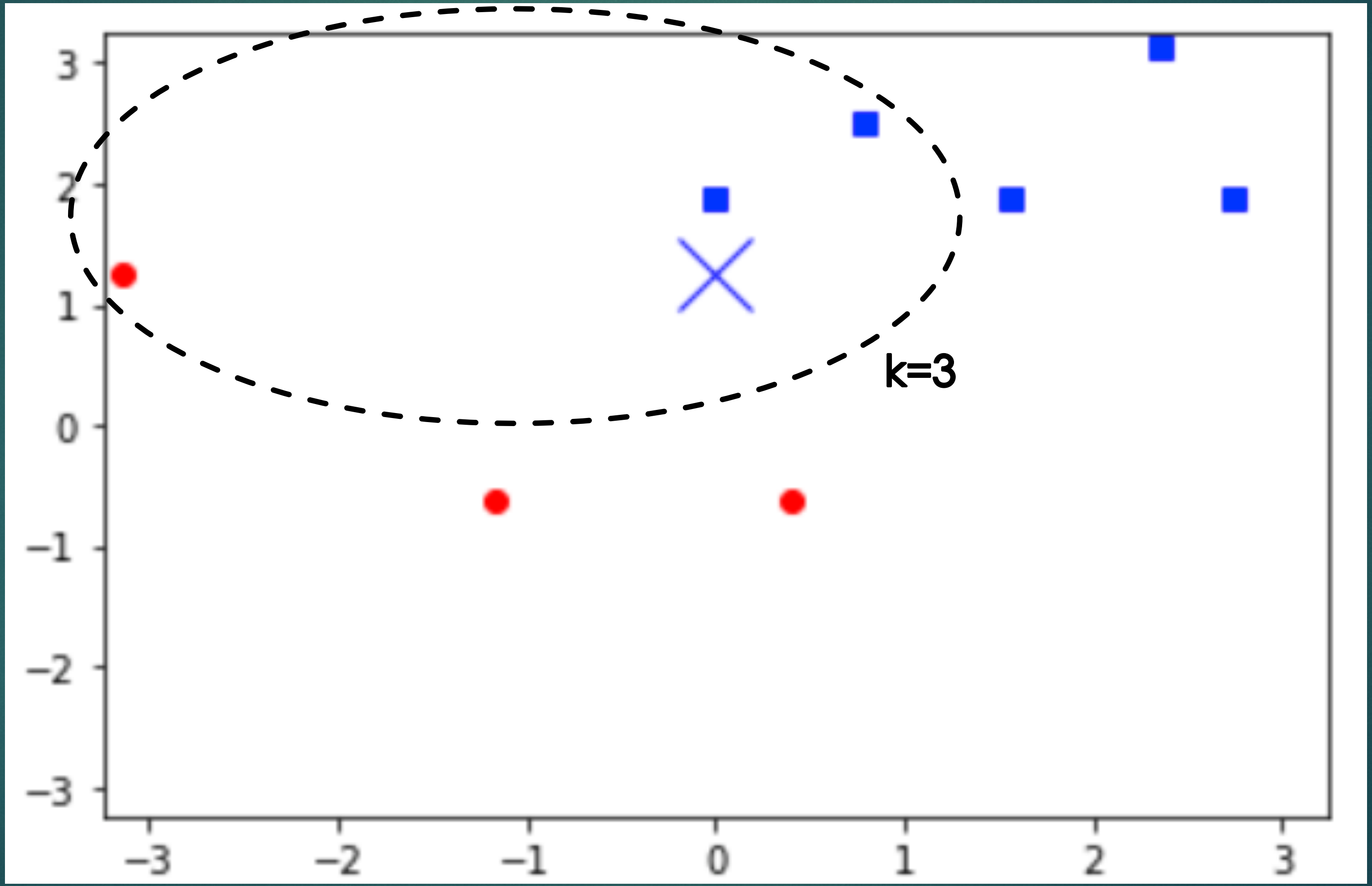
K-Nearest Neighbours

```
Classes = 2                                  # set number of classes
K = 3                                        # set K

# p is the point being classified
distances = [1-SWAP_test(p, x) for x in X]

# sort the indexes and get the K smallest distances
indexes=np.argsort(distances)[:K]

count = [0] * Classes
K_neigh = [Y[ndx] for ndx in indexes]
for y in K_neigh:
    count[y] += 1
classification = np.argmax(count)
```

K-Nearest Neighbours



K-Nearest Neighbours

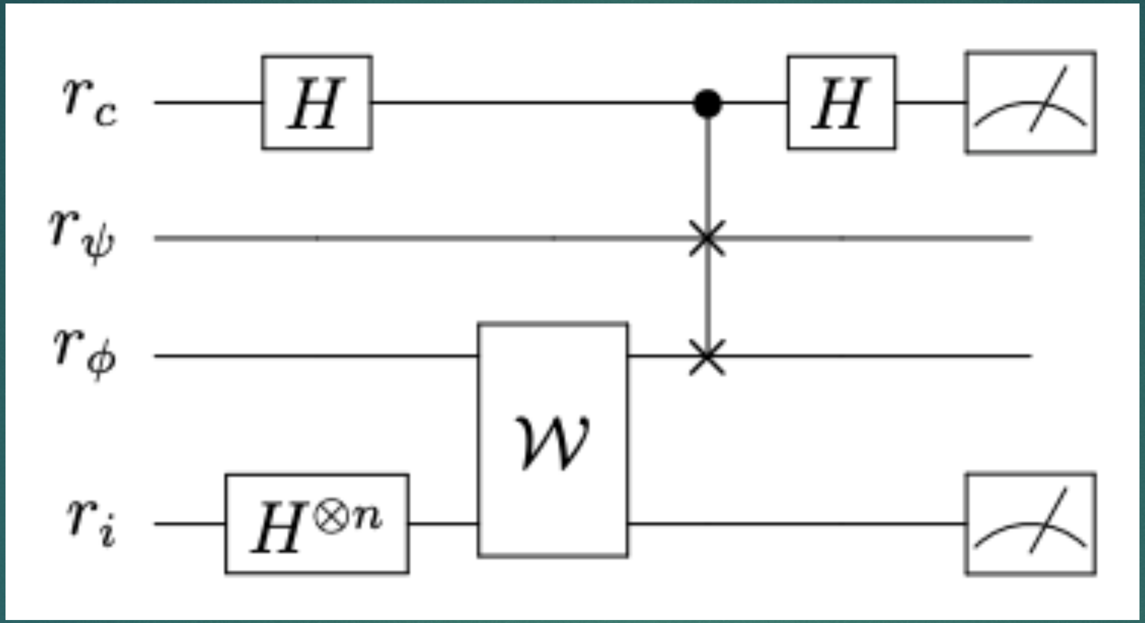


K-Nearest Neighbours

- The presented approach has a drawback:
  it requires a different circuit for each different $\varkappa(\boldsymbol{x}, \boldsymbol{x}^i), i = 1 \cdots M$

- If all the $\boldsymbol{x}^i$ can be loaded into a superposition is it possible to simultaneously compute all $\varkappa(\boldsymbol{x}, \boldsymbol{x}^i), i = 1 \cdots M$?

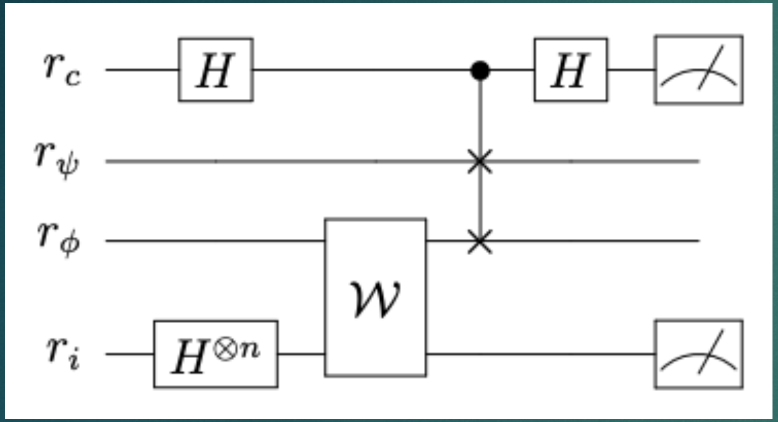
K-Nearest Neighbours



where:

- after the Hadamards, $|r_i\rangle = \frac{1}{\sqrt{M}} \sum_{i=0}^{M-1} |i\rangle$
- $\mathcal{W}$ acts upon $\left|r_\phi\right\rangle$: $\mathcal{W}|i\rangle|0\rangle = \mathcal{W}|i\rangle\left|x^i\right\rangle$

K-Nearest Neighbours



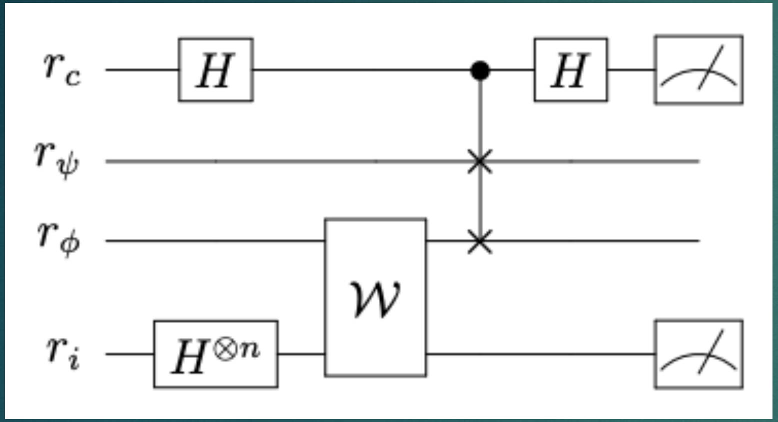
- $p_0 = \frac{1}{2} + \frac{1}{2M}\sum_{i=1}^{M}\left\|\langle x|x^i\rangle\right\|^2$; $p_1 = \frac{1}{2} - \frac{1}{2M}\sum_{i=1}^{M}\left\|\langle x|x^i\rangle\right\|^2$
- $p_0(\mathrm{i}) = \frac{1+\left\|\langle x|x^i\rangle\right\|^2}{M+\sum_{i=1}^{M}\left\|\langle x|x^i\rangle\right\|^2}$ $p_1(\mathrm{i}) = \frac{1-\left\|\langle x|x^i\rangle\right\|^2}{M-\sum_{i=1}^{M}\left\|\langle x|x^i\rangle\right\|^2}$
- $q(i) = p_0(\mathrm{i}) - p_1(\mathrm{i}) = \frac{2(F_i - \langle F\rangle)}{M(1-\langle F\rangle^2)}$

where $F_i = \left\|\langle x|x^i\rangle\right\|^2$ is the fidelity

$\langle F\rangle = \sum_{i=1}^{M} F_i/M$ is the average fidelity

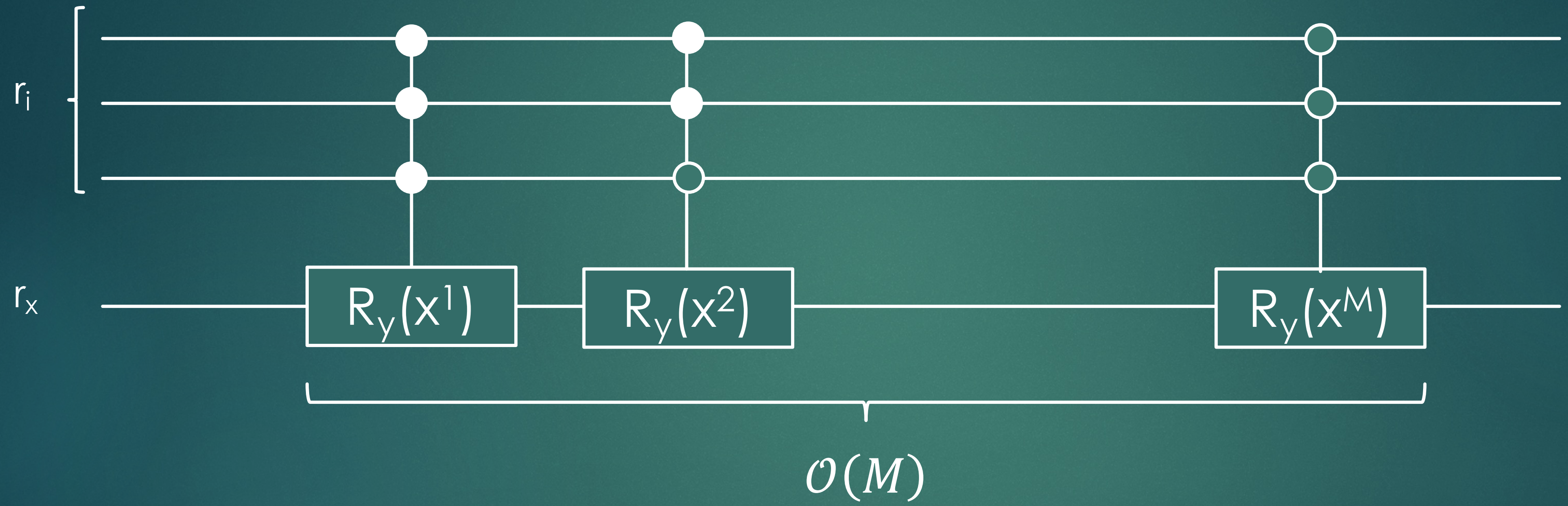
- $F_i = \left\|\langle x|x^i\rangle\right\|^2 = \frac{M}{2}q(i)[1-(p_0-p_1)^2] + (p_0-p_1)$

K-Nearest Neighbours



- Execute the circuit T times and let:
  - T_n be the count of $r_c = n$, $n \in \{0,1\}$
  - $c_n(i)$ be the count of $r_i = i$, given $r_c = n$, $n \in \{0,1\}$
- Then:
  - $p_n \approx \overline{p_n} = T_n/T$ $n \in \{0,1\}$
  - $p_n(i) \approx \overline{p_n}(i) = c_n(i)/T_n$ $n \in \{0,1\}$
  - $q(i) \approx \bar{q}(i) = \overline{p_0}(i) - \overline{p_1}(i)$
- $F_i = \left\|\langle x|x^i\rangle\right\|^2 \approx \frac{M}{2}\bar{q}(i)[1-(\bar{p}_0-\bar{p}_1)^2] + (\bar{p}_0 - \bar{p}_1)$

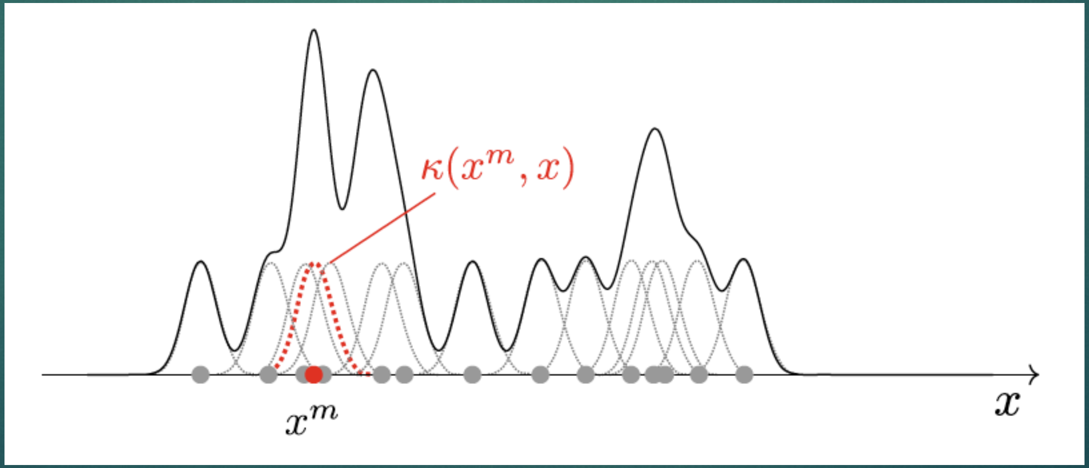
K-Nearest Neighbours – W operator



Kernel density estimation

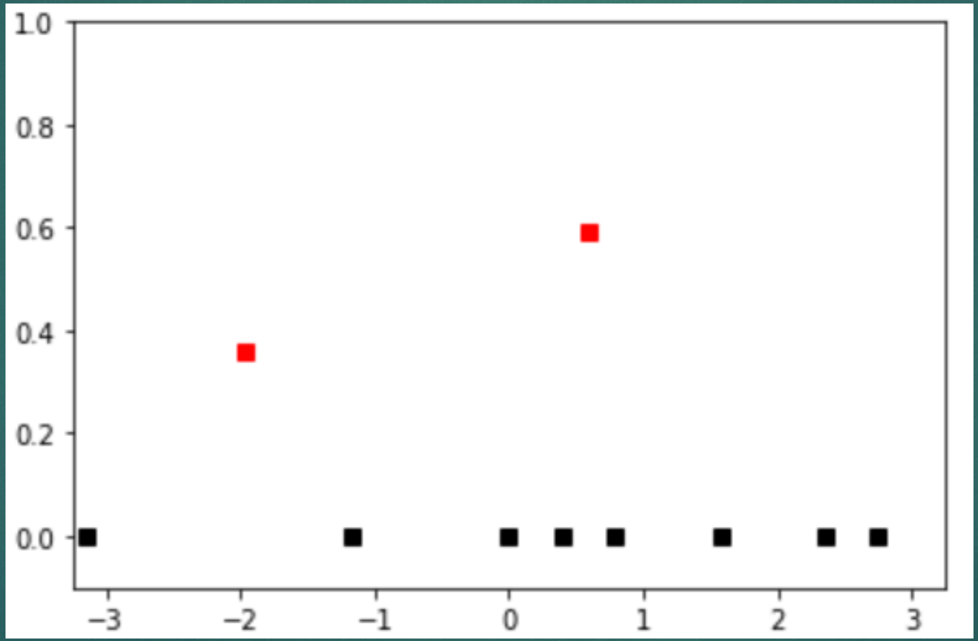
- Kernel density estimation constructs a probabilistic model from data as the sum of a similiraty measure κ between all M training inputs:

$$p(x) = \frac{1}{M} \sum_{m=1}^{M} \kappa(x\,, x^m)$$



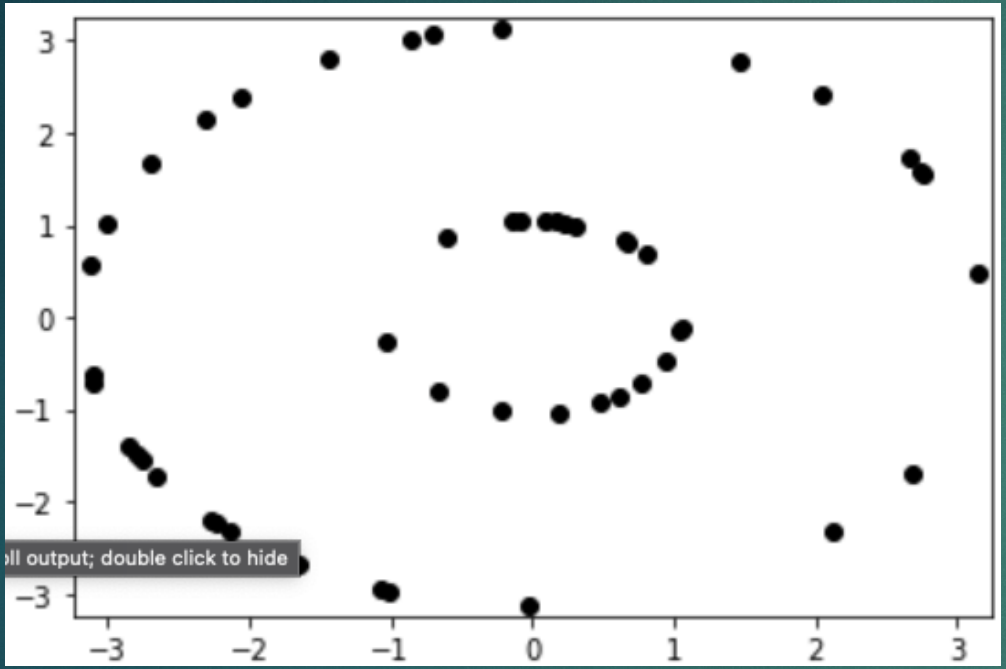
Kernel Density Estimation

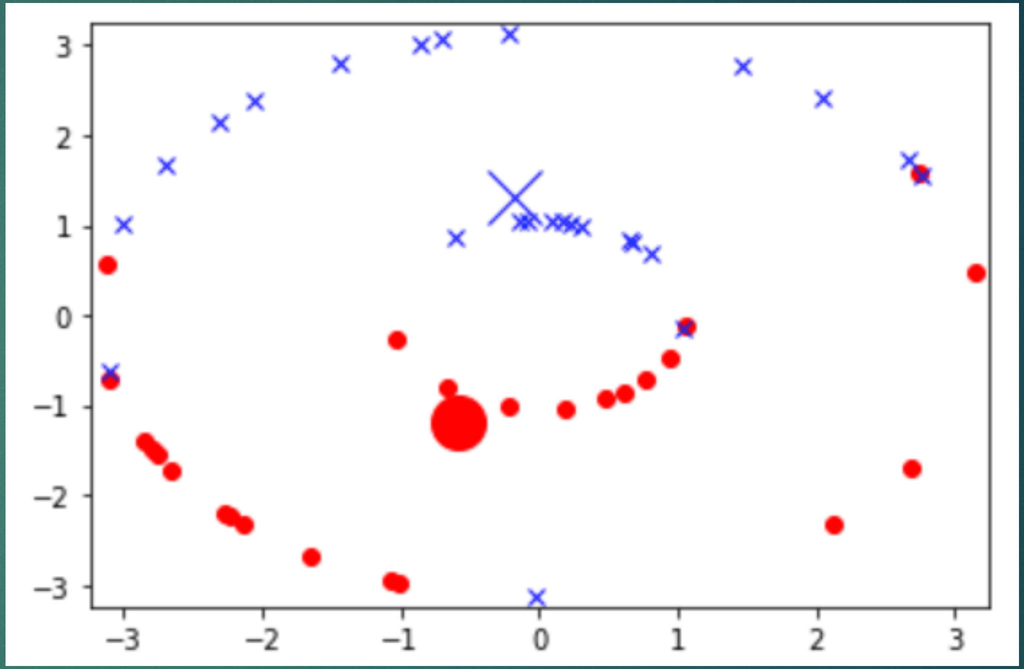
```
# q is the point whose probability we want
summation = sum ( [SWAP_test(p, x) for x in X] )
p_q = summation / M
```



$\mathcal{D}$=[[-8], [-3], [0], [1], [2], [4], [6], [7]]
q = [[1.5], [-5]]

Data separability





Is this really the expected result?