Lecture Applied Data Science Part 1

Nutarelli F.

IMT School for Advanced Studies, Lucca

December 30, 2023

Outline

k-NN

k-means clustering

k Nearest Neighbors for Classification (Theory)

Let $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ be a sequence of training examples. For each $\mathbf{x} \in \mathcal{X}$, let $\pi_1(\mathbf{x}), \ldots, \pi_m(\mathbf{x})$ be a reordering of $\{1, \ldots, m\}$ according to their distance to $\mathbf{x}$, $\rho(\mathbf{x}, \mathbf{x}_i)$. That is, for all $i < m$,

$$\rho(\mathbf{x}, \mathbf{x}_{\pi_i(\mathbf{x})}) \leq \rho(\mathbf{x}, \mathbf{x}_{\pi_{i+1}(\mathbf{x})}).$$

k-NN

input: a training sample $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
output: for every point $\mathbf{x} \in \mathcal{X}$,
return the majority label among $\{y_{\pi_i(\mathbf{x})} : i \leq k\}$

Key hints:

- Nearer points must be similar
- Hence the label of a point $\mathbf{x}$ is the same as the label of its neighbor points
- denoting $h_{\mathbf{S}}(\mathbf{x})$ the NN predictor...

For regression problems, namely, $\mathcal{Y} = \mathbb{R}$, one can define the prediction to be the average target of the k nearest neighbors. That is, $h_S(\mathbf{x}) = \frac{1}{k} \sum_{i=1}^{k} y_{\pi_i(\mathbf{x})}$. More generally, for some function $\phi : (\mathcal{X} \times \mathcal{Y})^k \to \mathcal{Y}$, the k-NN rule with respect to ϕ is:

$$h_S(\mathbf{x}) \;=\; \phi\left((\mathbf{x}_{\pi_1(\mathbf{x})}, y_{\pi_1(\mathbf{x})}), \ldots, (\mathbf{x}_{\pi_k(\mathbf{x})}, y_{\pi_k(\mathbf{x})})\right).$$

One would like to learn from finite training samples (m) and to understand the generalization performance as a function of the size of such finite training sets and clear prior assumptions on the data distribution

In other words: can we find a general rule for asymptotic consistency as a function of m without making assumption on the distribution of the data? $\longrightarrow$ YES!
Simpler to study for 1-NN rule (can be generalized for k-NN)

We now analyze the true error of the 1-NN rule for binary classification with the 0-1 loss, namely, $\mathcal{Y} = \{0, 1\}$ and $\ell(h, (\mathbf{x}, y)) = \mathbb{1}_{[h(\mathbf{x}) \neq y]}$. We also assume throughout the analysis that $\mathcal{X} = [0, 1]^d$ and ρ is the Euclidean distance.

We start by introducing some notation. Let $\mathcal{D}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$. Let $\mathcal{D}_{\mathcal{X}}$ denote the induced marginal distribution over $\mathcal{X}$ and let $\eta : \mathbb{R}^d \to \mathbb{R}$ be the conditional probability[1] over the labels, that is,

$$\eta(\mathbf{x}) = \mathbb{P}[y = 1 | \mathbf{x}].$$

Recall that the Bayes optimal rule (that is, the hypothesis that minimizes $L_{\mathcal{D}}(h)$ over all functions) is

$$h^\star(\mathbf{x}) = \mathbb{1}_{[\eta(\mathbf{x}) > 1/2]}.$$

We assume that the conditional probability function η is c-Lipschitz for some $c > 0$: Namely, for all $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, $|\eta(\mathbf{x}) - \eta(\mathbf{x}')| \leq c \|\mathbf{x} - \mathbf{x}'\|$. In other words, this assumption means that if two vectors are close to each other then their labels are likely to be the same.

LEMMA 19.1 *Let $\mathcal{X} = [0,1]^d, \mathcal{Y} = \{0,1\}$, and $\mathcal{D}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$ for which the conditional probability function, η, is a c-Lipschitz function. Let $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ be an i.i.d. sample and let h_S be its corresponding 1-NN hypothesis. Let $h^\star$ be the Bayes optimal rule for η. Then,*

$$\mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}[L_{\mathcal{D}}(h_S)] \leq 2\, L_{\mathcal{D}}(h^\star) + c \mathop{\mathbb{E}}_{S\sim\mathcal{D}^m,\mathbf{x}\sim\mathcal{D}}[\|\mathbf{x} - \mathbf{x}_{\pi_1(\mathbf{x})}\|].$$

- $\mathop{\mathbb{E}}_{S\sim D^m}[L_D(h_S)] - 2L_D(h^\star)$ is the error of the 1-NN rule (where $h^\star$ is the ideal, infeasible rule according to which we should classify $\mathbf{x}$ as 1 when the true $\eta(\mathbf{x}) > 1/2$ and $L_D(.)$ is the loss function under D)
- this error is bounded by the distance between test point $\mathbf{x}$ and its (unique) nearest (remember notation $\pi_1(\mathbf{x})$) neighbor in the training set

Let's skip the proof of that!

Let's skip the proof of that!

PROBLEM: $\mathop{\mathbb{E}}_{S\sim D^m, x\sim D} ||x - x_{\pi_1(x)}||$ unknown!

Only way is to bound it from above. Not simple. Use a two step procedure:

Let's skip the proof of that!

PROBLEM: $\mathop{\mathbb{E}}_{S\sim D^m, x\sim D} ||x - x_{\pi_1(x)}||$ unknown!

Only way is to bound it from above. Not simple. Use a two step procedure:

1) **Step 1**: bound probability weight of subsets not hit by a random sample, i.e. such that $C_i \cap S = \emptyset$, as a function of m (sample size);

Let's skip the proof of that!

PROBLEM: $\mathop{\mathbb{E}}_{S\sim D^m, x\sim D} ||x - x_{\pi_1(x)}||$ unknown!

Only way is to bound it from above. Not simple. Use a two step procedure:

1) **Step 1**: bound probability weight of subsets not hit by a random sample, i.e. such that $C_i \cap S = \emptyset$, as a function of m (sample size);

2) **Step 2**: Use step 1 to bound $\mathop{\mathbb{E}}_{S\sim D^m, x\sim D} ||x - x_{\pi_1(x)}||$

Step 1

LEMMA 19.2 *Let $C_1, \ldots, C_r$ be a collection of subsets of some domain set, $\mathcal{X}$. Let S be a sequence of m points sampled i.i.d. according to some probability distribution, $\mathcal{D}$ over $\mathcal{X}$. Then,*

$$\mathop{\mathbb{E}}_{S \sim \mathcal{D}^m} \left[\sum_{i : C_i \cap S = \emptyset} \mathbb{P}[C_i] \right] \leq \frac{r}{m\, e}.$$

Step 2

THEOREM 19.3 *Let $\mathcal{X} = [0,1]^d, \mathcal{Y} = \{0,1\}$, and $\mathcal{D}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$ for which the conditional probability function, η, is a c-Lipschitz function. Let h_S denote the result of applying the 1-NN rule to a sample $S \sim \mathcal{D}^m$. Then,*

$$\mathop{\mathbb{E}}_{S \sim \mathcal{D}^m}[L_{\mathcal{D}}(h_S)] \leq 2\, L_{\mathcal{D}}(h^\star) + 4\, c \sqrt{d}\, m^{-\frac{1}{d+1}}.$$

upper bound to expected distance of x and its nearest neighbor

Step 2

THEOREM 19.3 *Let $\mathcal{X} = [0,1]^d, \mathcal{Y} = \{0,1\}$, and $\mathcal{D}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$ for which the conditional probability function, η, is a c-Lipschitz function. Let h_S denote the result of applying the 1-NN rule to a sample $S \sim \mathcal{D}^m$. Then,*

$$\mathop{\mathbb{E}}_{S \sim \mathcal{D}^m}[L_{\mathcal{D}}(h_S)] \leq 2\, L_{\mathcal{D}}(h^\star) + 4\, c \sqrt{d}\, m^{-\frac{1}{d+1}}.$$

upper bound to expected distance of x and its nearest neighbor

Bottom line: as $m \longrightarrow \infty$, error of 1-NN $\longrightarrow 2L_D(h^\star)$.
Generalization to *k*-NN: as $m \longrightarrow \infty$, error of *k*-NN $\longrightarrow$ $\frac{1}{\sqrt{8/k}} L_D(h^\star)$.

Step 2

THEOREM 19.3 *Let $\mathcal{X} = [0,1]^d, \mathcal{Y} = \{0,1\}$, and $\mathcal{D}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$ for which the conditional probability function, η, is a c-Lipschitz function. Let h_S denote the result of applying the 1-NN rule to a sample $S \sim \mathcal{D}^m$. Then,*

$$\mathop{\mathbb{E}}_{S \sim \mathcal{D}^m}[L_{\mathcal{D}}(h_S)] \leq 2\, L_{\mathcal{D}}(h^\star) + 4\, c \sqrt{d}\, m^{-\frac{1}{d+1}}.$$

upper bound to expected distance of x and its nearest neighbor

Bottom line: as $m \longrightarrow \infty$, error of 1-NN $\longrightarrow 2L_D(h^\star)$.
Generalization to *k*-NN: as $m \longrightarrow \infty$, error of *k*-NN $\longrightarrow$ $\frac{1}{\sqrt{8/k}} L_D(h^\star)$.

Crucial implication: for the last term of Theorem 19.3 to be lower than an arbitrary $\varepsilon > 0$, it must be that $m \geq (4c\sqrt{d}/\varepsilon)^{d+1}$, i.e. the size of the training set should increase exponentially with the dimension $d \longrightarrow$ CURSE OF DIMENSIONALITY

Summary and recommendations

The *k*-NN rule is a very simple learning algorithm that relies on the assumption that "things that look alike must be alike." We formalized this intuition using the Lipschitzness of the conditional probability. Notice: the fact that $\eta = \mathbb{P}[Y = 1|X = x]$ is Lipschitz-continuous means that if $\mathbf{x} = \mathbf{x_i}$ is near to $\mathbf{x} = \mathbf{x_j}$ they will have very similar $\mathbb{P}[Y = 1|X = x]$

Summary and recommendations

The *k*-NN rule is a very simple learning algorithm that relies on the assumption that "things that look alike must be alike." We formalized this intuition using the Lipschitzness of the conditional probability. Notice: the fact that $\eta = \mathbb{P}[Y = 1|X = x]$ is Lipschitz-continuous means that if $\mathbf{x} = \mathbf{x_i}$ is near to $\mathbf{x} = \mathbf{x_j}$ they will have very similar $\mathbb{P}[Y = 1|X = x]$

We have shown that, practically, with a sufficiently large training set, the risk of the 1-NN is upper bounded by twice the risk of the Bayes optimal rule.

Summary and recommendations

The *k*-NN rule is a very simple learning algorithm that relies on the assumption that "things that look alike must be alike." We formalized this intuition using the Lipschitzness of the conditional probability. Notice: the fact that $\eta = \mathbb{P}[Y = 1|X = x]$ is Lipschitz-continuous means that if $\mathbf{x} = \mathbf{x_i}$ is near to $\mathbf{x} = \mathbf{x_j}$ they will have very similar $\mathbb{P}[Y = 1|X = x]$

We have shown that, practically, with a sufficiently large training set, the risk of the 1-NN is upper bounded by twice the risk of the Bayes optimal rule.

We have also derived a lower bound that shows the "curse of dimensionality" – the required sample size might increase exponentially with the dimension. Recommendation: in practice perform *k*-NN after a dimensionality reduction pre-processing step

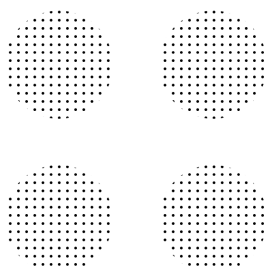
k-means

Please do not confuse k-means and k-NN:

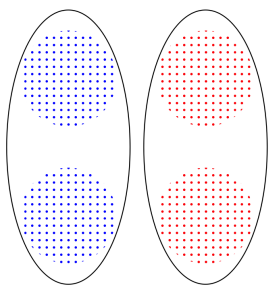
- k-NN is for CLASSIFICATION, i.e. $y \in \{0, 1\}$ (given);
- k-means is for CLUSTERING i.e. no label y is given;

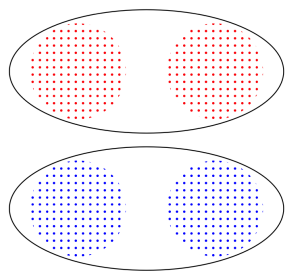
Clustering is a problematic task. (a) and (b) look both correct:

Consider, for example, the following set of points in $\mathbb{R}^2$:



(a)





(b)

Common setup in clustering

Input — a set of elements, $\mathcal{X}$, and a distance function over it. That is, a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ that is symmetric, satisfies $d(x, x) = 0$ for all $x \in \mathcal{X}$ and often also satisfies the triangle inequality. Alternatively, the function could be a similarity function $s : \mathcal{X} \times \mathcal{X} \to [0, 1]$ that is symmetric and satisfies $s(x, x) = 1$ for all $x \in \mathcal{X}$. Additionally, some clustering algorithms also require an input parameter k (determining the number of required clusters).

Output — a partition of the domain set $\mathcal{X}$ into subsets. That is, $C = (C_1, \ldots C_k)$ where $\bigcup_{i=1}^{k} C_i = \mathcal{X}$ and for all $i \neq j$, $C_i \cap C_j = \emptyset$. In some situations the clustering is "soft," namely, the partition of $\mathcal{X}$ into the different clusters is probabilistic where the output is a function assigning to each domain point, $x \in \mathcal{X}$, a vector $(p_1(x), \ldots, p_k(x))$, where $p_i(x) = \mathbb{P}[\mathbf{x} \in C_i]$ is the probability that $\mathbf{x}$ belongs to cluster C_i. Another possible output is a clustering *dendrogram* (from Greek dendron = tree, gramma = drawing), which is a hierarchical tree of domain subsets, having the singleton sets in its leaves, and the full domain as its root. We shall discuss this formulation in more detail in the following.

Common setup in clustering

Several clustering techniques out there! Examples:

- Linkage-based clustering: they start from the trivial clustering that has each data point as a single-point cluster. Then, repeatedly, these algorithms merge the "closest" clusters of the previous clustering. Principal problem: define a distance between clusters!
- Cost Minimization Clusterings. Define a cost function over a parameterized set of possible clusterings and the goal of the clustering algorithm is to find a partitioning (clustering) of minimal cost. Most popular: k-means clustering.

Objective function

- The objective function is a function from pairs of an input, $(\mathcal{X}, d)$, where d is a distance and a proposed clustering solution $C = (C_1, ..., C_k)$, to positive real numbers:

$$G : \mathbb{R}^n \times \mathbb{R}^k \longmapsto \mathbb{R}^+,$$

  i.e. $G((\mathcal{X}, d), C) \in \mathbb{R}^+$ must be minimized

- The solution to such a clustering problem is determined by a set of cluster centers $((\mu_1, \ldots, \mu_k))$, and the clustering assigns each instance to the center closest to it. More generally, the center-based objective is determined by choosing some monotonic function $f : \mathbb{R}^+ \longmapsto \mathbb{R}^+$ and then defining

$$G_f((\mathcal{X}, d), (C_1, \ldots C_k)) = \min_{\mu_1, \ldots \mu_k \in \mathcal{X}'} \sum_{i=1}^{k} \sum_{x \in C_i} f(d(x, \mu_i)),$$

where $\mathcal{X}'$ is either $\mathcal{X}$ or some superset of $\mathcal{X}$.

Objective function for k-means

The k-means objective function is one of the most popular clustering objectives. In k-means the data is partitioned into disjoint sets $C_1, \ldots, C_k$ where each C_i is represented by a centroid μ_i. It is assumed that the input set $\mathcal{X}$ is embedded in some larger metric space $(\mathcal{X}', d)$ (so that $\mathcal{X} \subseteq \mathcal{X}'$) and centroids are members of $\mathcal{X}'$. The k-means objective function measures the squared distance between each point in $\mathcal{X}$ to the centroid of its cluster. The centroid of C_i is defined to be

$$\mu_i(C_i) = \operatorname*{argmin}_{\mu \in \mathcal{X}'} \sum_{x \in C_i} d(x, \mu)^2.$$

Then, the k-means objective is

$$G_{\mathrm{k-means}}((\mathcal{X}, d), (C_1, \ldots, C_k)) = \sum_{i=1}^{k} \sum_{x \in C_i} d(x, \mu_i(C_i))^2.$$

This can also be rewritten as

$$G_{\mathrm{k-means}}((\mathcal{X}, d), (C_1, \ldots, C_k)) = \min_{\mu_1, \ldots \mu_k \in \mathcal{X}'} \sum_{i=1}^{k} \sum_{x \in C_i} d(x, \mu_i)^2.$$

Specification

What does it mean that centroids could belong to a superset $\mathcal{X}'$ of $\mathcal{X}$?

Simply that, if we are given a set of points $\in \mathcal{X}$, centroids are no necessarily chosen over that set, but can be chosen on artificial points $\notin \mathcal{X}$.

Example: say we are given points $\{1, 2, 3, 4, 5\} \in \mathbb{Z}^+$. The first centroid could be $2.25 \in \mathbb{R}^+$

k-means objective $\neq$ k-means algorithm

Problem: k-means problem is NP-hard, i.e. loosely speaking, it is computationally expensive to find its optimal solution. So?

k-means objective $\neq$ *k*-means algorithm

Problem: *k*-means problem is NP-hard, i.e. loosely speaking, it is computationally expensive to find its optimal solution. So?

ADOPT *k*-means algorithm. Notice: the outcome of this algorithm is not necessarily the same clustering that minimizes the k-means objective cost.

The *k*-means algorithm is usually based on the Euclidean distance $d(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}||$

k-means algorithm

k-Means

input: $\mathcal{X} \subset \mathbb{R}^n$; Number of clusters k

initialize: Randomly choose initial centroids $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$

repeat until convergence

$\forall i \in [k]$ set $C_i = \{\mathbf{x} \in \mathcal{X} : i = \operatorname{argmin}_j \|\mathbf{x} - \boldsymbol{\mu}_j\|\}$

(break ties in some arbitrary manner)

$\forall i \in [k]$ update $\boldsymbol{\mu}_i = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}$

k-means algorithm, Lemma

LEMMA 22.1 *Each iteration of the k-means algorithm does not increase the k-means objective function*

This Lemma is very important as it ensures that at every iteration of the algorithm, the cost function $G((\mathcal{X}, d), C)$ does not increase. However, the are no guarantees the global minimum is found!

Proof outline

- Define $G(C_1^{(t)}, \ldots, C_k^{(t)})$ as cost function at iteration t as the function whose centroids μ_i^t minimize the distance between a generic point $x \in C_i^{(t)}$ and μ_i^t;
- Define an auxiliary function, say $A = \sum_{i=1}^{k} \sum_{\mathbf{x} \in \mathbf{C_i^{(t)}}} ||x - \mu_i^{(t-1)}||$ which is greater than $G(C_1^{(t)}, \ldots, C_k^{(t)})$;
- Show that $A \leq G(C_1^{(t-1)}, \ldots, C_k^{(t-1)})$;
- exploit transitive property: $G(C_1^{(t)}, \ldots, C_k^{(t)}) \leq A \leq G(C_1^{(t-1)}, \ldots, C_k^{(t-1)}) \longrightarrow$ $G(C_1^{(t)}, \ldots, C_k^{(t)}) \leq G(C_1^{(t-1)}, \ldots, C_k^{(t-1)})$

Proof part 1

Proof To simplify the notation, let us use the shorthand $G(C_1, \ldots, C_k)$ for the k-means objective, namely,

$$G(C_1, \ldots, C_k) = \min_{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k \in \mathbb{R}^n} \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2. \tag{22.2}$$

It is convenient to define $\boldsymbol{\mu}(C_i) = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}$ and note that $\boldsymbol{\mu}(C_i) = \operatorname{argmin}_{\boldsymbol{\mu} \in \mathbb{R}^n} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \boldsymbol{\mu}\|^2$. Therefore, we can rewrite the k-means objective as

$$G(C_1, \ldots, C_k) = \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \boldsymbol{\mu}(C_i)\|^2. \tag{22.3}$$

Consider the update at iteration t of the k-means algorithm. Let $C_1^{(t-1)}, \ldots, C_k^{(t-1)}$ be the previous partition, let $\boldsymbol{\mu}_i^{(t-1)} = \boldsymbol{\mu}(C_i^{(t-1)})$, and let $C_1^{(t)}, \ldots, C_k^{(t)}$ be the new partition assigned at iteration t. Using the definition of the objective as given in Equation (22.2) we clearly have that

$$G(C_1^{(t)}, \ldots, C_k^{(t)}) \leq \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i^{(t)}} \|\mathbf{x} - \boldsymbol{\mu}_i^{(t-1)}\|^2. \tag{22.4}$$

Proof part 2

In addition, the definition of the new partition $(C_1^{(t)}, \ldots, C_k^{(t)})$ implies that it minimizes the expression $\sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \boldsymbol{\mu}_i^{(t-1)}\|^2$ over all possible partitions $(C_1, \ldots, C_k)$. Hence,

$$\sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i^{(t)}} \|\mathbf{x} - \boldsymbol{\mu}_i^{(t-1)}\|^2 \leq \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i^{(t-1)}} \|\mathbf{x} - \boldsymbol{\mu}_i^{(t-1)}\|^2. \tag{22.5}$$

Using Equation (22.3) we have that the right-hand side of Equation (22.5) equals $G(C_1^{(t-1)}, \ldots, C_k^{(t-1)})$. Combining this with Equation (22.4) and Equation (22.5), we obtain that $G(C_1^{(t)}, \ldots, C_k^{(t)}) \leq G(C_1^{(t-1)}, \ldots, C_k^{(t-1)})$, which concludes our proof. □

Cons of k-means algorithm

While the preceding lemma tells us that the k-means objective is monotonically non-increasing, there is no guarantee on the number of iterations the k-means algorithm needs in order to reach convergence.

k-means algorithm might converge to a point which is not even a local minimum

Why? Example (next page)

why k-means might not converge to local minimum

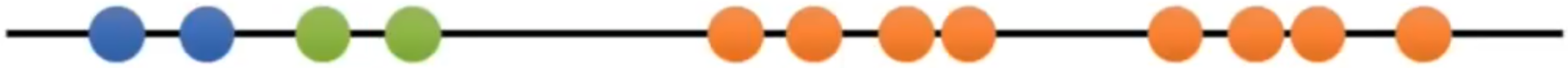


Figure 1: here is the solution of the algorithm. Variability of clusters not minimal

Figure 2: here is the solution by minimizing $G(.)$. Variability of clusters is minimal

Random Initialization Trap. Introduction

The fact that k-means might not converge to local minimum highly depends on the so called **Random Initialization Trap**. This leads to the use of **"kmeans++"** correction.

Random initialization trap is a problem that occurs in the k-means algorithm. In random initialization trap when the centroids of the clusters to be generated are explicitly defined by the user then inconsistency may be created and this may sometimes lead to generating wrong clusters in the dataset. So the random initialization trap may sometimes prevent us from developing the correct clusters.

Random Initialization Trap: kmeans++ (intuition)

The intuition behind this approach is that spreading out the k initial cluster centers $\mu_1, \ldots, \mu_k$ is a good thing: the first cluster center is chosen uniformly at random from the data points that are being clustered, after which each subsequent cluster center is chosen from the remaining data points with probability proportional to its squared distance from the point's closest existing cluster center.

Random Initialization Trap: kmeans++ (algorithm)

- Choose one center uniformly at random among the data points
- For each data point x not chosen yet, compute $d(x)$, the distance between x and the nearest center that has already been chosen
- Choose one new data point at random as a new center, using a weighted probability distribution where a point x is chosen with probability proportional to $d(x)^2$, i.e. **more distant points are chosen as centers with more probability**
- Repeat Steps 2 and 3 until k centers have been chosen
- Now that the initial centers have been chosen, proceed using standard *k*-means clustering.

Summary

The k-means problem is to find cluster centers that minimize the intra-class variance, i.e. the sum of squared distances from each data point being clustered to its cluster center (the center that is closest to it). Although finding an exact solution to the k-means problem for arbitrary input is NP-hard, the standard approach to finding an approximate solution is used widely and frequently finds reasonable solutions quickly.
However, the k-means algorithm has at least two major theoretic shortcomings:

- First, long computational time $\longrightarrow$ adaptive search of centroids
- Second, the approximation found can be arbitrarily bad with respect to the objective function compared to the optimal clustering.

The kmeans++ algorithm addresses the second of these obstacles by specifying a procedure to initialize the cluster centers before proceeding with the standard k-means optimization iterations.