K-nearest neighbors

Parametric methods

- Supervised learning setting: training data

$$\mathcal{D} = \{\ldots(\boldsymbol{x}_i, y_i)\ldots\}$$

- We consider a **parameterized family of functions** $f(\boldsymbol{x}; \boldsymbol{\theta}) \rightarrow \hat{y}$
- We consider a loss: $\mathcal{L}(\boldsymbol{\theta})$, roughly defined as average error (plus regularization)
- **Learning**:find the $\boldsymbol{\theta}$ that minimizes the loss
- **Inference**: evaluate $f(\boldsymbol{x}; \boldsymbol{\theta})$ for new x from test data
- As $\boldsymbol{\theta}$ is a parameter of f, this type of methods are called **parametric**.

Some properties of parametric methods

- The **expressivity of the function family** f limits how good the learning can be.
- $\boldsymbol{\theta}$ is a vector of constant size
  - It's size doesn't depend on $\mathcal{D}$
  - In many classic ML approaches it is quite small!
- Once learning is done, we can dispose of the training data, and only keep $\boldsymbol{\theta}$
- Cost of inference: constant (and usually quite low)

Non-parametric methods

- Methods that don't use a parameterized family of functions and don't search for $\boldsymbol{\theta}$

K-nearest neighbors

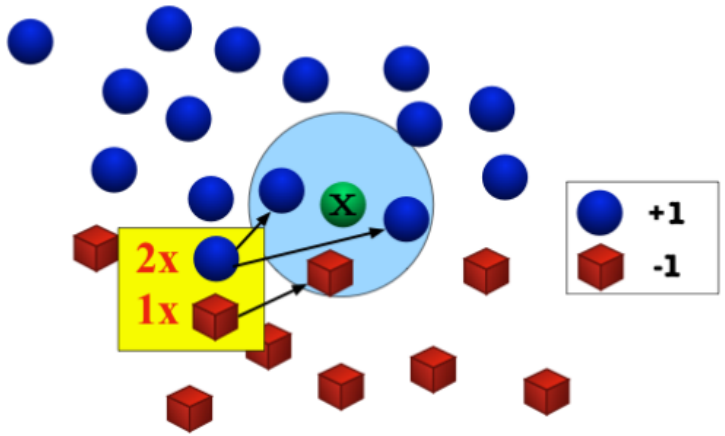
- Supervised learning setting: training data

$$\mathcal{D} = \{\ldots(\boldsymbol{x}_i, y_i)\ldots\}$$

- We consider:
  - k - small integer number (eg. 3, 5 etc.)
  - $dist$ - distance function $dist(\boldsymbol{x}_1, \boldsymbol{x}_2) \in \mathbb{R}$
- Learning: nothing done during learning. (*)
- Inference for input data $\boldsymbol{x}$:
  - calculate the distance between $\boldsymbol{x}$ and all the $\boldsymbol{x}_i$ in $\mathcal{D}$
  - choose the k closest ones
  - if regression: return the average of the corresponding y_i-s
  - if classification: return the corresponding y_i with the highest occurence.

Intuition behind k-NN

- Similar inputs have similar outputs
- k is not something that we are adjusting during training - it is not a parameter but a **hyperparameter**
- The larger the k the smoother the output



The distance function

- What is the distance function we can use?
- Euclidean

$$dist(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

- Manhattan

$$dist(\boldsymbol{a}, \boldsymbol{b}) = \sum_{i=1}^{n} |a_i - b_i|$$

The distance function (cont'd)

- Minkovski distance

$$dist(\boldsymbol{a}, \boldsymbol{b}, p) = \left(\sum_{i=1}^{n} |a_i - b_i|^p \right)^{\frac{1}{p}}$$

- For p=1 $\rightarrow$ Manhattan distance
- For p=2 $\rightarrow$ Euclidean distance
- For p=∞ $\rightarrow$ Chebyshev distance, $\max_{i=1}^{n} |x_i - y_i|$

The distance function (cont'd)

- The different dimensions of the distance function might have different scales or different importances
  - 3 sqft difference in area vs 3 sq meter difference in area
  - vs a difference of 3 bedrooms (4 bedrooms vs one)
- The distance function is a critical component of k-NN
- It basically describes our intuition of "what matters"

Performance of kNN

- There is no training cost.
- We need to carry with us the complete training data $\mathcal{D}$
- The cost of inference increases with the size of $\mathcal{D}$
  - In a naive implementation we need to calculate the distance to every training data point
  - In practice, we can pre-index the training data for a faster search, but it will still increase with $\mathcal{D}$

Problem in high dimensions: "curse of dimensionality"

- Our intuitions that about "nearests neighbors are close" are based on our 3D world.
  - High dimensions are weird and our intuitions mislead us.
- Consider $n = 1000$ samples, with each being a vector $\boldsymbol{x}$ of d dimensions, each element being between $x_i \in [0, 1]$
  - This means that they are in a unit lenght hypercube
- Let us consider $k = 10$
- What size is a hypercube that contains the k-nearest neighbors?

Curse of dimensionality

d	l
1	0.01
2	0.1
10	0.63
100	0.955
1000	0.9954

As d increases, almost the entire space is needed to find the 10-NN. So they are not particularly closer than any other point!

Some conclusions

- k-NN is a very good model for low dimensional data
  - Especially if we can engineer a distance function that captures our intuition
- It breaks down for high dimensionality
- It breaks down if the training data is very large

Moving beyond k-NN

- The insight that similar inputs map to similar outputs is important, but it needs to be refined
- There are several ways we can move on from this:
  - **Manifold learning**: maybe the interesting data is on a manifold of lower dimensionality (eg. images)
  - **Metric learning**: maybe we can learn a distance function that captures what we want?
  - **Representation learning**: maybe we can learn a function that transforms the input into a representation where a simple distance function captures what we want?
  - **Prototype learning**: find a prototype x for every class, and measure distance only to that one...