Nearest Neighbor and Metric Learning

Saeed Saremi

assigned reading: 3.5, 6.3.5

October 17, 2024

outline

- K nearest neighbor (KNN) classification
  - non-parametric approach to classification
  - What is the role of K in KNN?
  - A generative derivation of KNN via kernel density estimation (KDE)
  - How good is the nearest neighbor classifier? (Cover and Hart, 1967)
  - The curse of dimensionality

- Contrastive learning as Metric learning
  - Pairwise loss and Siamese networks
  - Triplet loss
  - N-pair loss
  - Learning joint embeddings (OpenAI's CLIP model)

parametric vs. non-parametric statistical models

- So far in the course we have mostly focused on parametric models with the aim of learning $p(y|x)$:
$$p_\theta(y|x) \approx p(y|x),$$
where θ is a fixed-dimensional vector of parameters.
- The parameters are estimated knowing a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$.
- After model fitting, the data is thrown away.

- Next, we consider the first (and arguably simplest) example of a non-parametric model for classification.
- In this approach, we must keep/store the training examples $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$.
- Effectively, the number of "parameters" grows with $n = |\mathcal{D}|$.

metric space[1]

A metric space is a set X together with a notion of distance d between its elements:

$$d : X \times X \to \mathbb{R},$$

satisfying the following axioms:

1. $d(x, x) = 0$
2. If $x_1 \neq x_2$, then $d(x_1, x_2) > 0$.
3. $d(x_1, x_2) = d(x_2, x_1)$.

⚠ triangle inequality:

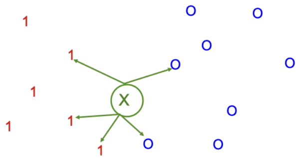
The Mahalanobis distance has a historical importance in ML (below $X = \mathbb{R}^d$):

$$d_M(x_1, x_2) = \sqrt{(x_1 - x_2)^\top M \, (x_1 - x_2)}$$

[1] Mathematician's way of quantifying "near"

K nearest neighbor (KNN) classifier

1. store the training set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$
2. to classify a test point x, we first find the K closest (using a pre-specific metric) examples to x in the training set $\mathcal{D}$
3. let's denote this set by $N_K(x, \mathcal{D})$ (Q: what is $|N_K(x, \mathcal{D}|$?)



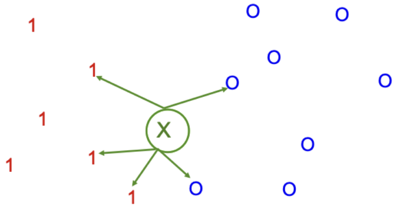
4. we then look at the labels y_i for the points in $N_K(x, \mathcal{D})$ to estimate $p(y|x)$:

$$p(y = c|x, \mathcal{D}) = \frac{1}{K} \sum_{i \in N_K(x,\mathcal{D})} \mathbb{I}(y_i = c), \text{ where } y \in [C].$$

? prove that the above is a proper conditional probability:

K nearest neighbor (KNN) classifier

? In the example below ($d = 2, C = 2, K = 5$) what is $p(y = 1|x, \mathcal{D})$?



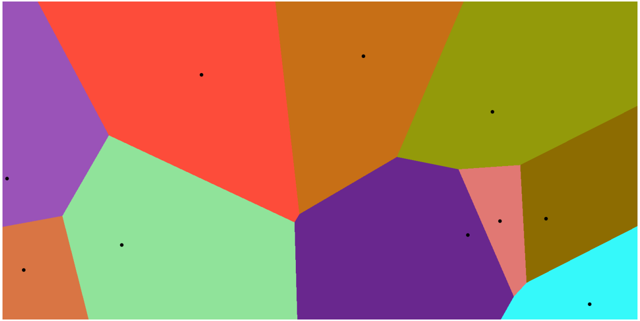
? What should we do for regression?

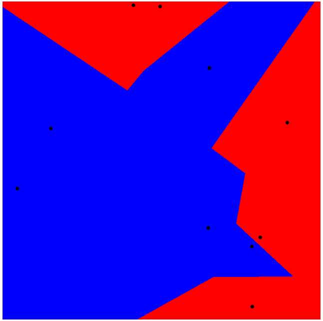
– Training:

– Inference:

Voronoi[2] tessellation

- KNN classifier with $K = 1$ is special
- It induces a Voronoi partition of the space



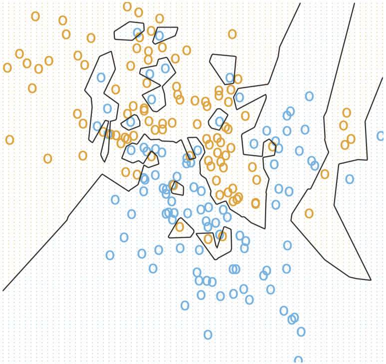


- by definition, any test point in a Voronoi region is classified by its "exemplar"

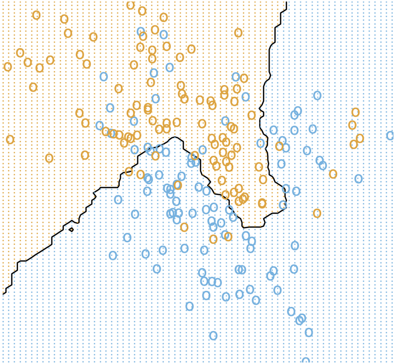
? prove that for the Euclidean metric in $\mathbb{R}^d$ all the boundaries are linear

[2] pronounced "VOH-roh-noy"

what is the role of K in KNN?



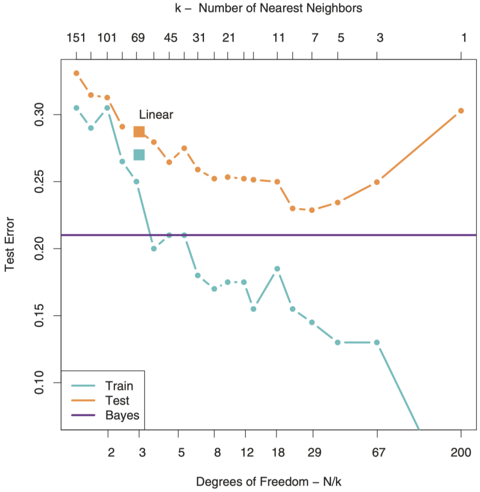
(a) K



(b) K'

- ❓ Can you guess K in (a)?
- ❓ Is $K' > K$? Why?
- ❓ What does $p(y = c|x, \mathcal{D})$ reduce to if $K \geq |\mathcal{D}|$?

n/K = "model complexity"



from KDE to KNN

- next, we arrive at KNN with a generative classifier
- we first need to estimate $p(x|c)$ and $p(c)$
- we "grow" a ball around each point x until we encounter K data points
- let's denote the volume of the ball by $V_K(x)$
- denote $n_c(x)$ as the number of samples from class c in the volume $V_K(x)$
- in this set up we have[3]

$$p(x|c, \mathcal{D}) = \frac{n_c(x)}{n_c}\frac{1}{V_K(x)}, \; p(c) = \frac{n_c}{n}.$$

✎ Bayes rule:

[3] $p(x|c, \mathcal{D})V_K(x)$ is roughly the fraction of c-class samples in $V_K(x)$, which is $n_c(x)/n_c$

how good is KNN?

- A result by Cover and Hart (1967) states that in the large sample limit $n \to \infty$, the *KNN error is never worse than twice the Bayes optimal classifier's error*.
- As a reminder for the Bayes optimal classifier we assume we know $p(x|y)$ and $p(y)$:
$$p(y = c|x) = \frac{p(x|c)p(c)}{\sum_{c \in [C]} p(x|c)p(c)}$$
- In KNN we know absolutely nothing about the underlying distribution!

the curse of dimensionality[4]

- All (non-parametric) local (distance-based) methods suffer from the curse of dimensionality in high dimensions.
- The fundamental problem is that the volume of space grows exponentially fast with dimension

$$V(h) = h^d$$

- This means to "fill up" the space we need exponentially large number of samples.
- Equivalently, with n samples we only get $O(n^{-1/d})$ closer to the target.
- Mathematica demonstration

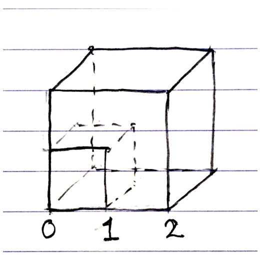


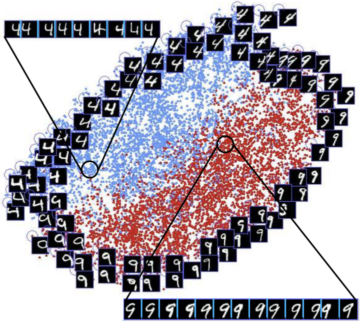
Figure: The abundance of room in high dimensions

[4] watch the lecture "Mysteries in High Dimensions" https://www.youtube.com/watch?v=B_Cmzc5I8ro by Michel Talagrand

KNN: the good, the bad, and the ugly

- fast/no training
- learns complex functions easily
- high storage cost
- slow at inference
- performs poorly in high dimensions

manifold hypothesis, and metric learning



- The manifold hypothesis is an assumption in machine learning that suggests high-dimensional data (such as images, videos, or text) lies on a "lower-dimensional manifold" embedded within the high-dimensional space.
- Next, we attempt to learn this manifold using some modern techniques.
- The general philosophy is that instead of learning a complex metric we embed the data using a neural network (the mapping is therefore complex), but in the embedding space we use a simple Euclidean metric.

contrastive learning: pull and push

- The key problem is that Euclidean metric (in the ambient space) is not good!
- Similar objects could be far apart according to the Euclidean metric and vice versa.

- The solution is simple: learn an embedding in which similar objects are pulled closer together, and dissimilar ones are apart.
- ? How do we define "similar"?

Siamese networks

- One of the earliest approaches to metric learning from similar/dissimilar pairs was based on minimizing the following contrastive loss:

$$\mathcal{L}_{ij}(\theta; m) = \mathbb{I}(y_i = y_j)\, d(z_\theta(x_i), z_\theta(x_j))^2 + \mathbb{I}(y_i \neq y_j)\, \max(0, m - d(z_\theta(x_i), z_\theta(x_j))^2)$$

- In this setup, the similar/dissimilar was "given to us" by the labels.
- We minimize the loss over all pairs. Naively this takes $O(n^2)$, but we use SGD.

? Why is this called Siamese network?

triplet loss

- One disadvantage of pairwise losses is that the optimization of the positive pairs is "independent" of the negative pairs
- The triplet loss brings the two terms together:

$$\mathcal{L}_i(\theta; m) = max(d(z_\theta(x_i), z_\theta(x_i^+))^2 - d(z_\theta(x_i), z_\theta(x_i^-))^2 + m, 0)$$

- In this literature, x_i is called the *anchor*, x_i^+ is the (similar) *positive* example, and x_i^- is the (dissimilar) *negative* example.
- pull and push: intuitively in minimizing the loss we want the distance from the anchor to the positive to be less (by some safety margin m) than the distance from the anchor to the negative.

N-pair loss

- One problem with the triplet loss is its inefficiency:
  each anchor is only compared to one negative example at a time.
- What if we want to take a batch of negative samples?[5]
- We can set up the following N-pair loss (Sohn, 2016):

$$\mathcal{L}(\theta; x, x^+, \{x_k^-\}_{k=1}^{N-1}) = \log\left(1 + \sum_{k=1}^{N-1} \exp\left(\hat{z}_\theta(x)^\top \hat{z}_\theta(x_k^-) - \hat{z}_\theta(x)^\top \hat{z}_\theta(x^+)\right)\right)$$

✎ interpretation in terms of the softmax classification

[5] typically, negative pairs are cheaper to gather; see the next slide

the frontier: learning joint embeddings

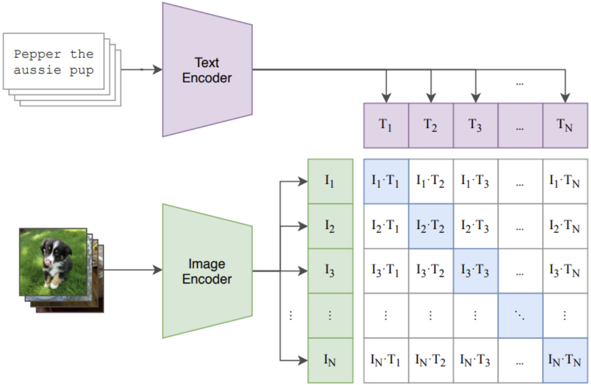


Figure: OpenAI's CLIP model was trained with the N-pair loss