Machine Learning

Lecture 2: k-Nearest Neighbors

Prof. Dr. Aleksandar Bojchevski

16.04.24

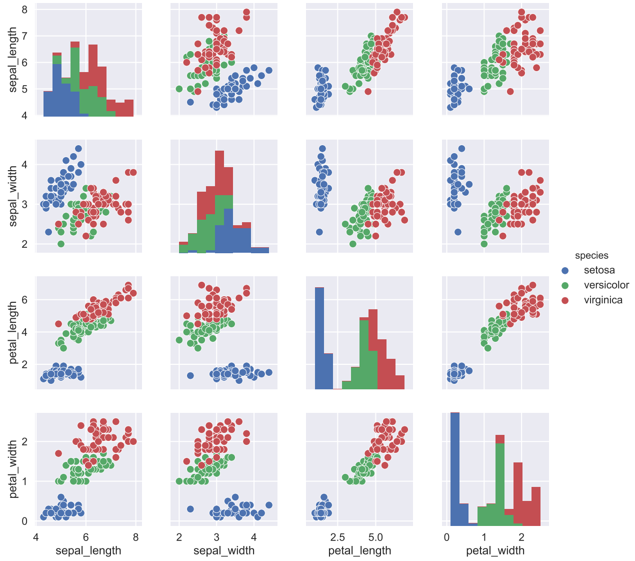
Outline

Algorithm

Curse of dimensionality

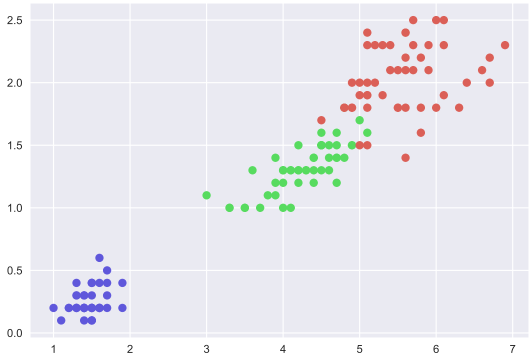
Generalization

Iris dataset



Iris dataset: 2 features

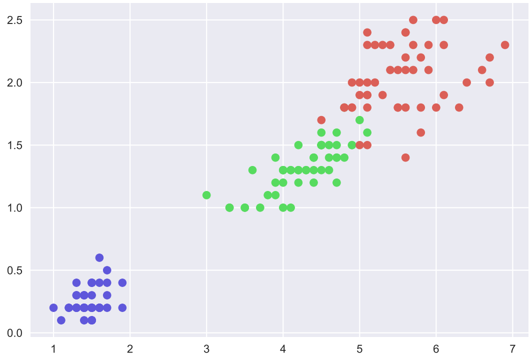
How do we intuitively label new samples by hand?



Iris dataset: 2 features

How do we intuitively label new samples by hand?

Look at the *surrounding* points. Do as your **neighbor** does.



Notation

Symbol	Meaning
x	scalar is lowercase and not bold
$\boldsymbol{x}$	vector is lowercase and bold
$\boldsymbol{\Sigma}$	matrix is uppercase and bold
$\boldsymbol{y}$	vector of labels (targets)
$\mathcal{D}$	sets are calligraphic, e.g. training dataset
$\boldsymbol{x}_i, y_i$	features and labels of the i'th example
$f(\boldsymbol{x})$	function, e.g. predicted value for input $\boldsymbol{x}$
N	number of samples (examples, instances)
D	number of features (attributes, predictors)
$\hat{y}$	predicted labels (targets)

Unless otherwise mentioned vectors are column vectors, e.g. $\boldsymbol{x} \in \mathbb{R}^{D \times 1}$.

1-Nearest Neighbor algorithm

Given a training dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ where $\boldsymbol{x}_i$ are features and y_i targets.

To classify new observations:

1. Define a distance measure (e.g. Euclidean distance)
2. Compute the nearest neighbor for a new data point
3. Label it with the label of its nearest neighbor

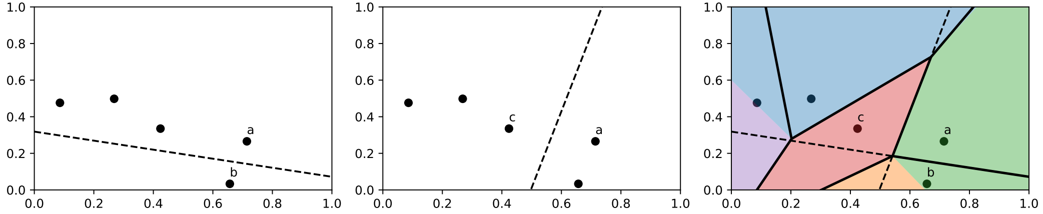
This works for both *classification* $y_i \in \{1, \ldots, C\}$, and *regression* $y_i \in \mathbb{R}$.

1-Nearest Neighbor decision boundary

Corresponds to a Voronoi tesselation.

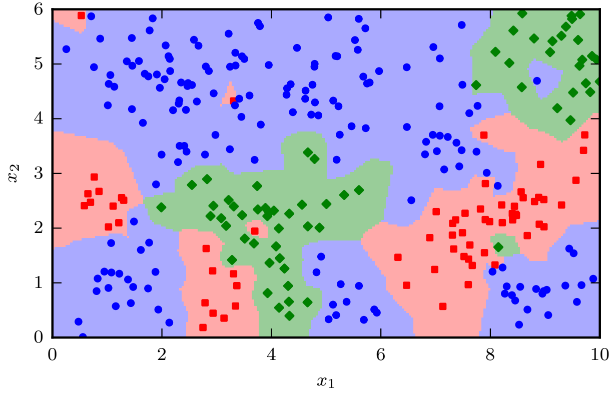
The bisecting line between each pair of points determines which one is closer.

The decision boundary is a set of connected, convex polyhedra.



1-Nearest Neighbor decision boundary

Tends to result in poor generalization.



k-Nearest Neighbors classification

Looking at multiple nearest neighbors and picking the **majority** label makes us more *robust* against errors in the training set.

Let $\mathcal{N}_k(\boldsymbol{x})$ be the k nearest neighbors of $\boldsymbol{x}$ in $\mathcal{D}$. The probability of class c is:

$$p(y = c \mid \boldsymbol{x}, k) = \frac{1}{k} \sum_{i \in \mathcal{N}_k(\boldsymbol{x})} \mathbb{I}(y_i = c),$$

and the prediction $\hat{y} = \arg\max_c p(y = c \mid \boldsymbol{x}, k)$ is the mode of its neighbors' labels.

Here $\mathbb{I}$ is the **indicator** function defined as $\mathbb{I}(e) = \begin{cases} 1 \text{ if } e \text{ is true} \\ 0 \text{ if } e \text{ is false.} \end{cases}$

k-Nearest Neighbors classification

Looking at multiple nearest neighbors and picking the **majority** label makes us more *robust* against errors in the training set.

Let $\mathcal{N}_k(\boldsymbol{x})$ be the k nearest neighbors of $\boldsymbol{x}$ in $\mathcal{D}$. The probability of class c is:

$$p(y = c \mid \boldsymbol{x}, k) = \frac{1}{k} \sum_{i \in \mathcal{N}_k(\boldsymbol{x})} \mathbb{I}(y_i = c),$$

and the prediction $\hat{y} = \arg\max_c p(y = c \mid \boldsymbol{x}, k)$ is the mode of its neighbors' labels.

Here $\mathbb{I}$ is the **indicator** function defined as $\mathbb{I}(e) = \begin{cases} 1 \text{ if } e \text{ is true} \\ 0 \text{ if } e \text{ is false.} \end{cases}$

k-Nearest Neighbors classification

Looking at multiple nearest neighbors and picking the **majority** label makes us more *robust* against errors in the training set.

Let $\mathcal{N}_k(\boldsymbol{x})$ be the k nearest neighbors of $\boldsymbol{x}$ in $\mathcal{D}$. The probability of class c is:

$$p(y = c \mid \boldsymbol{x}, k) = \frac{1}{k} \sum_{i \in \mathcal{N}_k(\boldsymbol{x})} \mathbb{I}(y_i = c),$$

and the prediction $\hat{y} = \arg\max_c p(y = c \mid \boldsymbol{x}, k)$ is the mode of its neighbors' labels.

Here $\mathbb{I}$ is the **indicator** function defined as $\mathbb{I}(e) = \begin{cases} 1 \text{ if } e \text{ is true} \\ 0 \text{ if } e \text{ is false.} \end{cases}$

k-Nearest Neighbors classification: weighted

Look at multiple nearest neighbors and pick the **weighted majority** label.

The weight is **inversely proportional** to the distance.

Let $\mathcal{N}_k(\boldsymbol{x})$ be the k nearest neighbors of $\boldsymbol{x}$ in $\mathcal{D}$. The probability of class c is now:

$$p(y = c \mid \boldsymbol{x}, k) = \frac{1}{Z} \sum_{i \in \mathcal{N}_k(\boldsymbol{x})} \frac{1}{\mathrm{d}(\boldsymbol{x}, \boldsymbol{x}_i)} \mathbb{I}(y_i = c),$$

where $Z = \sum_{i \in \mathcal{N}_k(x)} \frac{1}{\mathrm{d}(\boldsymbol{x}, \boldsymbol{x}_i)}$ is the the normalization constant, and $\mathrm{d}(\boldsymbol{x}, \boldsymbol{x}_i)$ measures the distance between $\boldsymbol{x}$ and $\boldsymbol{x}_i$.

As before the prediction is $\hat{y} = \arg\max_c p(y = c \mid \boldsymbol{x}, k)$.

k-Nearest Neighbors classification: weighted

Look at multiple nearest neighbors and pick the **weighted majority** label.
The weight is **inversely proportional** to the distance.

Let $\mathcal{N}_k(\boldsymbol{x})$ be the k nearest neighbors of $\boldsymbol{x}$ in $\mathcal{D}$. The probability of class c is now:

$$p(y = c \mid \boldsymbol{x}, k) = \frac{1}{Z} \sum_{i \in \mathcal{N}_k(\boldsymbol{x})} \frac{1}{\mathrm{d}(\boldsymbol{x}, \boldsymbol{x}_i)} \mathbb{I}(y_i = c),$$

where $Z = \sum_{i \in \mathcal{N}_k(x)} \frac{1}{\mathrm{d}(\boldsymbol{x}, \boldsymbol{x}_i)}$ is the the normalization constant, and $\mathrm{d}(\boldsymbol{x}, \boldsymbol{x}_i)$ measures the distance between $\boldsymbol{x}$ and $\boldsymbol{x}_i$.

As before the prediction is $\hat{y} = \arg\max_c p(y = c \mid \boldsymbol{x}, k)$.

k-Nearest Neighbors regression

Regression, i.e. $y_i \in \mathbb{R}$ is a real number, is similar.

Let $\mathcal{N}_k(\boldsymbol{x})$ be the k nearest neighbors of $\boldsymbol{x}$ in $\mathcal{D}$, then for regression:

$$\hat{y} = \frac{1}{Z} \sum_{i \in \mathcal{N}_k(\boldsymbol{x})} \frac{1}{\mathrm{d}(\boldsymbol{x}, \boldsymbol{x}_i)} y_i,$$

where $Z = \sum_{i \in \mathcal{N}_k(x)} \frac{1}{\mathrm{d}(\boldsymbol{x}, \boldsymbol{x}_i)}$ is the the normalization constant, and $\mathrm{d}(\boldsymbol{x}, \boldsymbol{x}_i)$ measures the distance between $\boldsymbol{x}$ and $\boldsymbol{x}_i$.

The prediction is the **weighted mean** of its neighbors' values.

k-Nearest Neighbors regression

Regression, i.e. $y_i \in \mathbb{R}$ is a real number, is similar.

Let $\mathcal{N}_k(\boldsymbol{x})$ be the k nearest neighbors of $\boldsymbol{x}$ in $\mathcal{D}$, then for regression:

$$\hat{y} = \frac{1}{Z} \sum_{i \in \mathcal{N}_k(\boldsymbol{x})} \frac{1}{\mathrm{d}(\boldsymbol{x}, \boldsymbol{x}_i)} y_i,$$

where $Z = \sum_{i \in \mathcal{N}_k(x)} \frac{1}{\mathrm{d}(\boldsymbol{x}, \boldsymbol{x}_i)}$ is the the normalization constant, and $\mathrm{d}(\boldsymbol{x}, \boldsymbol{x}_i)$ measures the distance between $\boldsymbol{x}$ and $\boldsymbol{x}_i$.

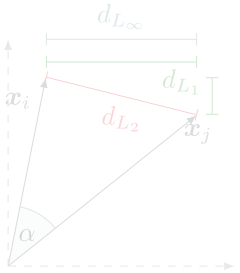
The prediction is the **weighted mean** of its neighbors' values.

Distance measures

K-NN can be used with various distance measures $\rightarrow$ highly flexible.

Usually the features are real vectors, $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathbb{R}^D$:

- L_2 norm (Euclidean): $d_{L_2}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{\sum_{d=1}^{D} (x_{id} - x_{jd})^2}$
- L_1 norm: $d_{L_1}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{d=1}^{D} |x_{id} - x_{jd}|$
- L_∞ norm: $d_{L_\infty}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \max_d |x_{id} - x_{jd}|$
- Angle: $d_{\cos}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \cos\alpha = \frac{\boldsymbol{x}_i^\top \boldsymbol{x}_j}{\|\boldsymbol{x}_i\| \|\boldsymbol{x}_j\|}$

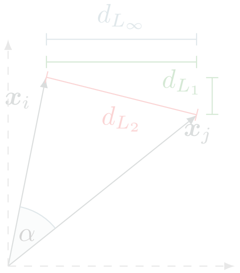


Distance measures

K-NN can be used with various distance measures $\rightarrow$ highly flexible.

Usually the features are real vectors, $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathbb{R}^D$:

- L_2 norm (Euclidean): $d_{L_2}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{\sum_{d=1}^{D} (x_{id} - x_{jd})^2}$
- L_1 norm: $d_{L_1}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{d=1}^{D} |x_{id} - x_{jd}|$
- L_∞ norm: $d_{L_\infty}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \max_d |x_{id} - x_{jd}|$
- Angle: $d_{\cos}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \cos\alpha = \frac{\boldsymbol{x}_i^\top \boldsymbol{x}_j}{\|\boldsymbol{x}_i\|\|\boldsymbol{x}_j\|}$

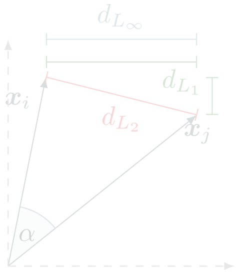


Distance measures

K-NN can be used with various distance measures $\rightarrow$ highly flexible.

Usually the features are real vectors, $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathbb{R}^D$:

- L_2 norm (Euclidean): $d_{L_2}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{\sum_{d=1}^{D}(x_{id} - x_{jd})^2}$
- L_1 norm: $d_{L_1}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{d=1}^{D} |x_{id} - x_{jd}|$
- L_∞ norm: $d_{L_\infty}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \max_d |x_{id} - x_{jd}|$
- Angle: $d_{\cos}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \cos\alpha = \frac{\boldsymbol{x}_i^\top \boldsymbol{x}_j}{\|\boldsymbol{x}_i\|\|\boldsymbol{x}_j\|}$

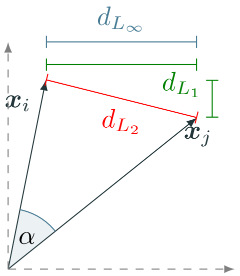


Distance measures

K-NN can be used with various distance measures $\rightarrow$ highly flexible.

Usually the features are real vectors, $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathbb{R}^D$:

- L_2 norm (Euclidean): $d_{L_2}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{\sum_{d=1}^{D} (x_{id} - x_{jd})^2}$
- L_1 norm: $d_{L_1}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{d=1}^{D} |x_{id} - x_{jd}|$
- L_∞ norm: $d_{L_\infty}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \max_d |x_{id} - x_{jd}|$
- Angle: $d_{\cos}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \cos\alpha = \frac{\boldsymbol{x}_i^\top \boldsymbol{x}_j}{\|\boldsymbol{x}_i\| \|\boldsymbol{x}_j\|}$



Distance measures

Mahalanobis distance, where $\mathbf{\Sigma}$ is *positive (semi) definite* and *symmetric*:

$$d_{\mathbf{\Sigma}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \mathbf{\Sigma}^{-1} (\boldsymbol{x}_i - \boldsymbol{x}_j)}$$

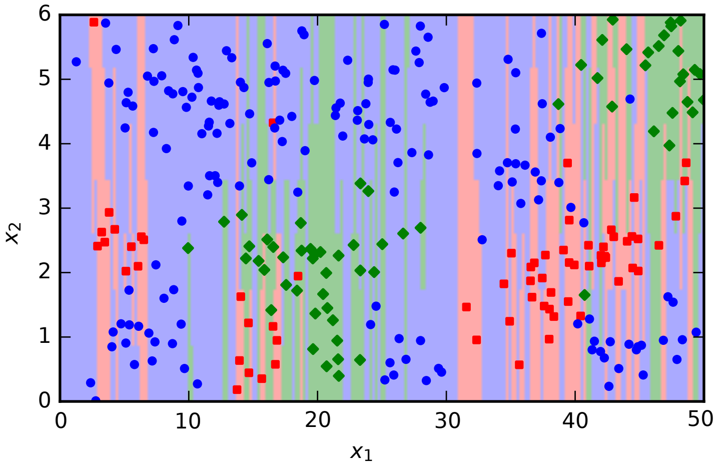
L_p norm, Hamming distance, String/Graph edit distance, Learned metric (e.g. learn $\mathbf{\Sigma}$), Jaccard, ...

How to choose? Depends on the problem. It should be semantically meaningful.

Example: $d_{\cos}$ for bag-of-words representation for text documents.

Scaling issues

The same example $(k = 1)$ but one feature is in meters, the other in centimeters.



Circumventing scaling issues

Data **standardization**: scale each feature to zero mean and unit variance.

$$\boldsymbol{x}_i' = \frac{\boldsymbol{x}_i - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

where $\boldsymbol{\mu}$ is the mean vector and $\boldsymbol{\sigma}$ is the standard variance vector.

Commonly used since many models are sensitive to differences in scale.

The Mahalanobis distance $\mathrm{d}_{\boldsymbol{\Sigma}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_j - \boldsymbol{x}_j)}$

with $\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \cdots & 0 \\ 0 & 0 & \sigma_n^2 \end{bmatrix}$ is equal to Euclidean distance on normalized data.

Circumventing scaling issues

Data **standardization**: scale each feature to zero mean and unit variance.

$$\boldsymbol{x}_i' = \frac{\boldsymbol{x}_i - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

where $\boldsymbol{\mu}$ is the mean vector and $\boldsymbol{\sigma}$ is the standard variance vector.

Commonly used since many models are sensitive to differences in scale.

The Mahalanobis distance $\mathrm{d}_{\boldsymbol{\Sigma}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_j - \boldsymbol{x}_j)}$

with $\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \cdots & 0 \\ 0 & 0 & \sigma_n^2 \end{bmatrix}$ is equal to Euclidean distance on normalized data.

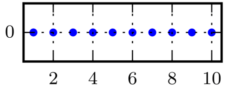
Outline

Algorithm

Curse of dimensionality

Generalization

The curse of dimensionality



Given a discrete one-dimensional input space $x \in \{1, 2, \ldots, 10\}$

For $N = 20$ uniformly distributed samples the data covers 100% of the input space.

Add a second dimension (now $\boldsymbol{x} \in \{1, \ldots, 10\}^2$) and your data only covers 18% of the space.

Once you add a third dimension you only cover 2%.

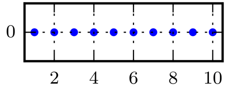
The curse of dimensionality

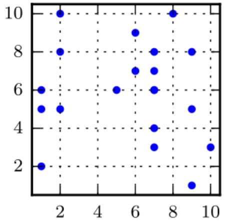
Given a discrete one-dimensional input space $x \in \{1, 2, \ldots, 10\}$

For $N = 20$ uniformly distributed samples the data covers 100% of the input space.

Add a second dimension (now $\boldsymbol{x} \in \{1, \ldots, 10\}^2$) and your data only covers 18% of the space.

Once you add a third dimension you only cover 2%.





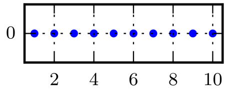
The curse of dimensionality

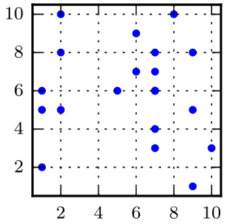
Given a discrete one-dimensional input space $x \in \{1, 2, \ldots, 10\}$

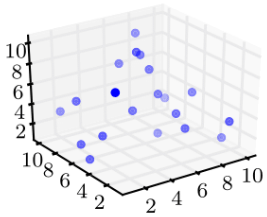
For $N = 20$ uniformly distributed samples the data covers 100% of the input space.

Add a second dimension (now $\boldsymbol{x} \in \{1, \ldots, 10\}^2$) and your data only covers 18% of the space.

Once you add a third dimension you only cover 2%.







The curse of dimensionality

Sample data uniformly at random in the unit cube $[0,1]^D$.

Let l be the edge length of the smallest hyper-cube that contains all k-nearest neighbors of a test point. Then $l^D \approx \frac{k}{N}$ and $l \approx (\frac{k}{N})^{1/D}$.

How big does l have to be for just $k = 10$ neighbors for different D?

D	l
2	0.1
10	0.63
100	0.955
1000	0.9954

Spans almost the entire space, so the nearest neighbor will be far away.

The curse of dimensionality

Divide the interval into $[0, \epsilon, 1 - \epsilon, 1]$ for some $\epsilon > 0$. The probability of landing in the interior is $(1 - 2\epsilon)^D$ which quickly converges to 0 as D grows since $1 - 2\epsilon < 1$.

Can we just use a larger dataset, i.e. larger N?

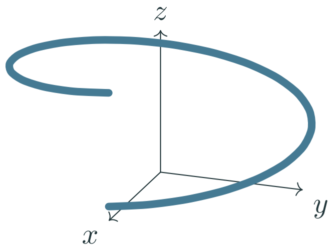
Let $l = 0.1$ so it is relatively small.
Then $N = \frac{k}{(0.1)^D}$, $\implies$ N has to grow exponentially with the number of features.

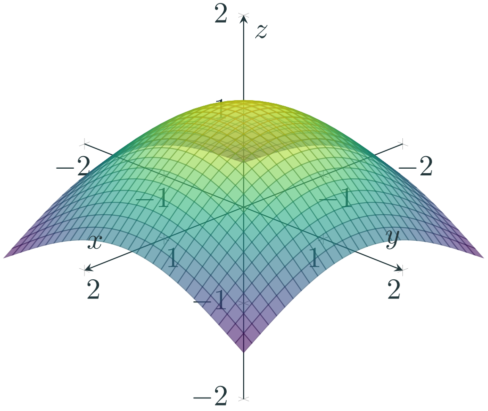
Always keep the curse of dimensionality in mind when using k-NN.

Real data has low-dimensional structure

The ”true” dimensionality can be much lower than the ambient space.

Data often lies on a lower-dimensional manifold (*manifold hypothesis*).





We can perform dimensionality reduction or learn a good representation.

Outline

Algorithm

Curse of dimensionality

Generalization

How many neighbors should we use?

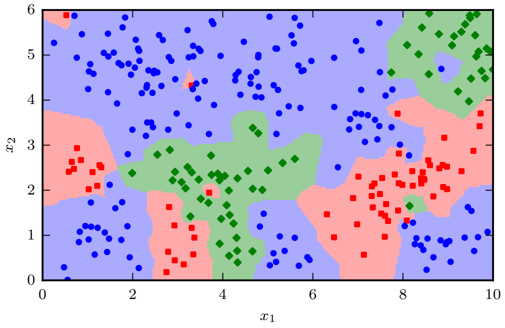


Figure 1: 1-NN decision boundary.

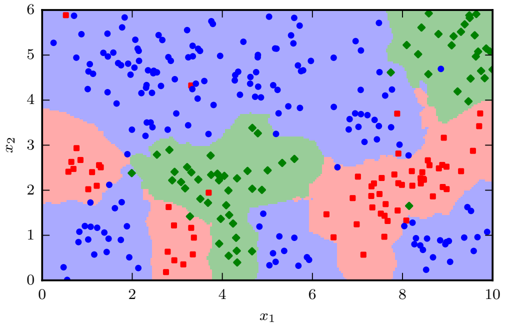


Figure 2: 3-NN decision boundary.

Generalization

Goal is **generalization**: find a model that performs best on unseen (future) data.

Given a datasets $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ where instances are drawn from some (unknown) distribution $(\boldsymbol{x}_i, y_i) \sim p$.

We want to learn a function h[1] such that $h(\boldsymbol{x}) \approx y$ for a **new** instance $(\boldsymbol{x}, y) \sim p$

To do so we specify:

- Class of functions $\mathcal{H}$, e.g. all possible values of k, all neural networks
- Loss function ℓ that tells us how good is a given hypothesis $h \in \mathcal{H}$

[1] Usually we denote the model with f here we have h for hypothesis.

Loss functions

Let y be the ground-truth target and $\hat{y} = h(\boldsymbol{x})$ the prediction. We have:

- **Zero-one loss**: $\ell(y, \hat{y}) = \mathbb{I}(y \neq \hat{y})$
- **Squared loss**: $\ell(y, \hat{y}) = (y - \hat{y})^2$
- **Absolute loss**: $\ell(y, \hat{y}) = |y - \hat{y}|$

We can also compute the average loss for a given dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$

$$\mathcal{L}(h, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, h(\boldsymbol{x}_i))$$

With the zero-one loss $\mathcal{L}(h, \mathcal{D})$ computes the error.

We want to learn (find) $h^* = \arg\min_{h \in \mathcal{H}} \mathcal{L}(h, \mathcal{D})$.

Generalization

We want to learn (find) $h^* = \arg\min_{h \in \mathcal{H}} \mathcal{L}(h, \mathcal{D})$.

If we find a function with low loss on our data $\mathcal{D}$, how do we know if it will make correct predictions on unseen examples not in $\mathcal{D}$?

Bad example: Define memorizer $h(\cdot)$ as:

$$h(\boldsymbol{x}) = \begin{cases} y_i, & \text{if } \exists (\boldsymbol{x}_i, y_i) \in \mathcal{D}, \text{s.t. } \boldsymbol{x} = \boldsymbol{x}_i \\ 0, & \text{otherwise.} \end{cases}$$

h has 0% error rate, i.e. $\mathcal{L}(h, \mathcal{D}) = 0$, but it performs horribly on samples not in $\mathcal{D}$.

Generalization

We want to learn (find) $h^* = \arg\min_{h \in \mathcal{H}} \mathcal{L}(h, \mathcal{D})$.

If we find a function with low loss on our data $\mathcal{D}$, how do we know if it will make correct predictions on unseen examples not in $\mathcal{D}$?

Bad example: Define memorizer $h(\cdot)$ as:

$$h(\boldsymbol{x}) = \begin{cases} y_i, & \text{if } \exists(\boldsymbol{x}_i, y_i) \in \mathcal{D}, \text{s.t. } \boldsymbol{x} = \boldsymbol{x}_i \\ 0, & \text{otherwise.} \end{cases}$$

h has 0% error rate, i.e. $\mathcal{L}(h, \mathcal{D}) = 0$, but it performs horribly on samples not in $\mathcal{D}$.

Generalization

We actually care about the population risk $\mathcal{L}(h) = \mathbb{E}_{(\boldsymbol{x},y)\sim p}[\ell(y, h(\boldsymbol{x}))]$, while $\mathcal{L}(h, \mathcal{D})$ computes the empirical risk on a fixed sample $\mathcal{D}$.

The difference $\mathcal{L}(h) - \mathcal{L}(h, \mathcal{D})$ is called the **generalization gap**.

A large generalization gap indicates that we are **overfitting**.

In practice we don't know the distribution p so how can we get an **unbiased** estimate of $\mathcal{L}(h)$?

Split $\mathcal{D}$ into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$. Learn h with $\mathcal{D}_{\text{train}}$ and evaluate with $\mathcal{D}_{\text{test}}$.

Why does $\mathcal{L}(h, \mathcal{D}_{\text{test}}) \to \mathcal{L}(h)$ as $|\mathcal{D}_{\text{test}}| \to \infty$?

Generalization

We actually care about the population risk $\mathcal{L}(h) = \mathbb{E}_{(\boldsymbol{x},y)\sim p}[\ell(y, h(\boldsymbol{x}))]$, while $\mathcal{L}(h, \mathcal{D})$ computes the empirical risk on a fixed sample $\mathcal{D}$.

The difference $\mathcal{L}(h) - \mathcal{L}(h, \mathcal{D})$ is called the **generalization gap**.

A large generalization gap indicates that we are **overfitting**.

In practice we don't know the distribution p so how can we get an **unbiased** estimate of $\mathcal{L}(h)$?

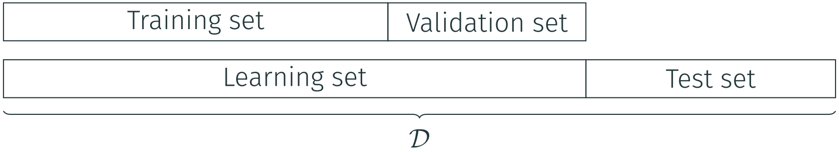
Split $\mathcal{D}$ into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$. Learn h with $\mathcal{D}_{\text{train}}$ and evaluate with $\mathcal{D}_{\text{test}}$.

Why does $\mathcal{L}(h, \mathcal{D}_{\text{test}}) \to \mathcal{L}(h)$ as $|\mathcal{D}_{\text{test}}| \to \infty$? (Weak) law of large numbers.

Choosing k

Generalization: pick the **hyper-parameter** k that performs best on unseen data.

Unfortunately, no access to unseen future data, so split the dataset $\mathcal{D}$:

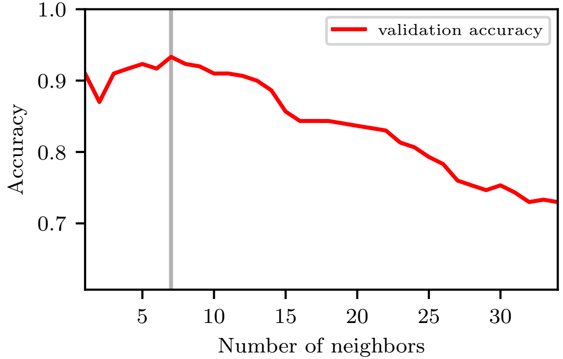


Hyper-parameter tuning procedure:

1. Learn the model using the training set (e.g. for k-NN just store the data)
2. Pick k that leads to the best performance on the *validation set*
3. Report final performance on the test set

Example: Using the validation set to choose k

We choose $k = 7$ since it has highest validation accuracy.



Detour: Bayes optimal classifier

Assume that we know the true $p(y \mid \boldsymbol{x})$, what is the optimal model h?

The bayes optimal classifier predicts $\hat{y} = h_{\text{opt}}(\boldsymbol{x}) = \arg\max_y p(y \mid \boldsymbol{x})$.

This is a good as it gets, but we can still make some errors. For the zero-one loss:

$$\epsilon_{\text{BO}} = \mathcal{L}(h_{\text{opt}}) = \mathbb{E}_{(x,y)\sim p}[\ell(y, h_{\text{opt}}(\boldsymbol{x}))] = \mathbb{E}_{(x,y)\sim p}[\ell(y, \hat{y})] = \mathbb{E}_{x\sim p}[1 - p(\hat{y} \mid \boldsymbol{x})]$$

We have $\mathcal{L}(h_{\text{opt}}) \leq \mathcal{L}(h)$ for any h.

1-NN convergence

Theorem (informal). As $N \to \infty$ the 1-NN error is no more than twice the error of the Bayes optimal classifier.

Proof. Let $\boldsymbol{x}_{\text{NN}}$ be the nearest neighbor of $\boldsymbol{x}$.

As $N \to \infty, d(\boldsymbol{x}_{\text{NN}}, \boldsymbol{x}) \to 0$, i.e. the nearest neighbor is identical to $\boldsymbol{x}$.

The error then equals the probability of drawing two different labels:

$$\begin{aligned}\epsilon_{\text{NN}} &= \mathbb{E}_{x\sim p}[p(\hat{y} \mid \boldsymbol{x})(1 - p(\hat{y} \mid \boldsymbol{x}_{\text{NN}})) + p(\hat{y} \mid \boldsymbol{x}_{\text{NN}})(1 - p(\hat{y} \mid \boldsymbol{x}))] \\ &\leq \mathbb{E}_{x\sim p}[1 - p(\hat{y} \mid \boldsymbol{x}_{\text{NN}}) + 1 - p(\hat{y} \mid \boldsymbol{x})] = \mathbb{E}_{x\sim p}[2(1 - p(\hat{y} \mid \boldsymbol{x}_{\text{NN}}))] = 2\epsilon_{\text{BO}}\end{aligned}$$

We have $\epsilon_{\text{BO}} \leq \epsilon_{\text{NN}} \leq 2\epsilon_{\text{BO}}$. So $\epsilon_{\text{BO}} = 0 \implies \epsilon_{\text{NN}} = 0$.

However, 1-NN is *statistically inconsistent* – there are distributions for which it does not converge to the Bayes error rate.

What happens when $k \to N$?

What happens when $k \to N$?

We converge to the constant majority vote predictor $h(\boldsymbol{x}) = \arg\max_y \sum_i \mathbb{I}[y = y_i]$.

This is the simplest trivial baseline that you should always compare against.

This also shows you that high accuracy, e.g. 99%, is meaningless without context. If the class labels are highly imbalanced the baseline can trivially achieve this.[2]

[2] We will discuss proper evaluation and model selection in much more detail next week.

Speed and memory requirements

You need to store the entire datasets for k-NN. Alternative:

- Prototypes: select a few representative instances and discard the rest

Computing the nearest neighbors becomes expensive in large dimensions. Methods for nearest neighbor search[3]:

[3] FAISS (`https://github.com/facebookresearch/faiss`) is a useful library.

Speed and memory requirements

You need to store the entire datasets for k-NN. Alternative:

- Prototypes: select a few representative instances and discard the rest

Computing the nearest neighbors becomes expensive in large dimensions. Methods for nearest neighbor search[3]:

- Linear search, $O(DN)$ runtime
- Space partitioning, e.g. k-d tree, $O(\log N)$ average runtime
- Approximate search with Locality Sensitive Hashing (LSH)

[3] FAISS (`https://github.com/facebookresearch/faiss`) is a useful library.

k-NN in context

k-NN is **nonparametric** – no assumptions about the functional form, no fixed number of parameters (number of ”parameters” scales with the data size).

k-NN is **lazy** – it does not have an explicit training step.

k-NN is **instance-based** – predict by comparing with instances in the training set.

k-NN is **discriminative** – does not explicitly model the data generating process.

Summary

Predict the (weighted) majority of your k nearest neighbors.

Can use any distance measure (flexible) but the choice can be critical.

Can suffer from the curse of dimensionality.

Use train/validation/test split to select k and obtain a model that generalizes.

Reading material

Main reading

- ”Probabilistic Machine Learning: An Introduction” by Murphy [ch. 1.2.1, 1.2.3, 16.1]

Extra reading

- ”Bayesian Reasoning and Machine Learning” by Barber [ch. 14]