k-Nearest Neighbors

Wiebke Köpp

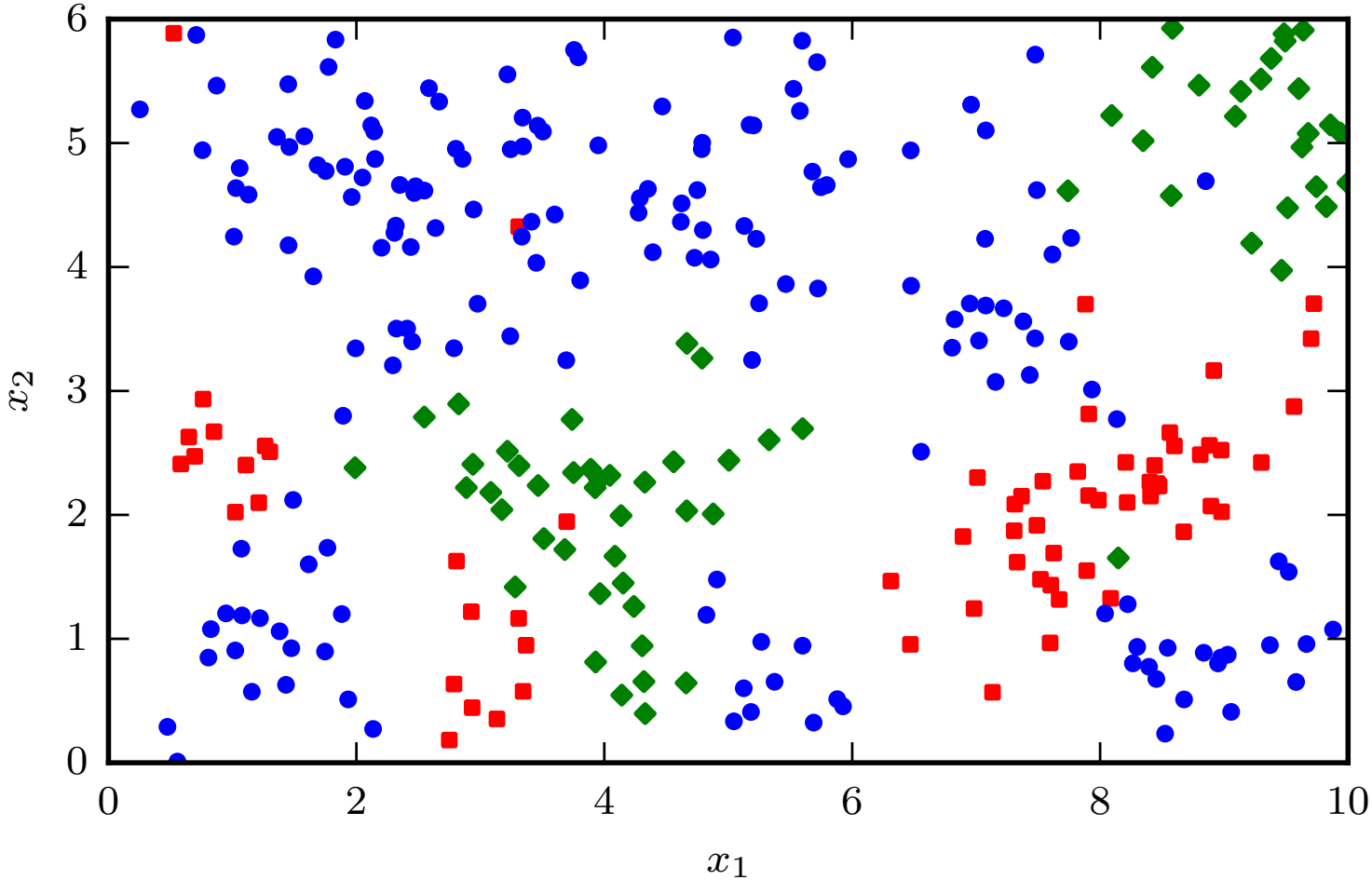
Technische Universität München

Reading Material:
"Machine Learning: A Probabilistic Perspective" by Murphy [ch. 1.4.1 - 1.4.3]

Further extra reading:
"Bayesian Reasoning and Machine Learning" by Barber [ch. 14]

Note: These slides are adapted from slides originally by Daniala Korhammer

The data from last time



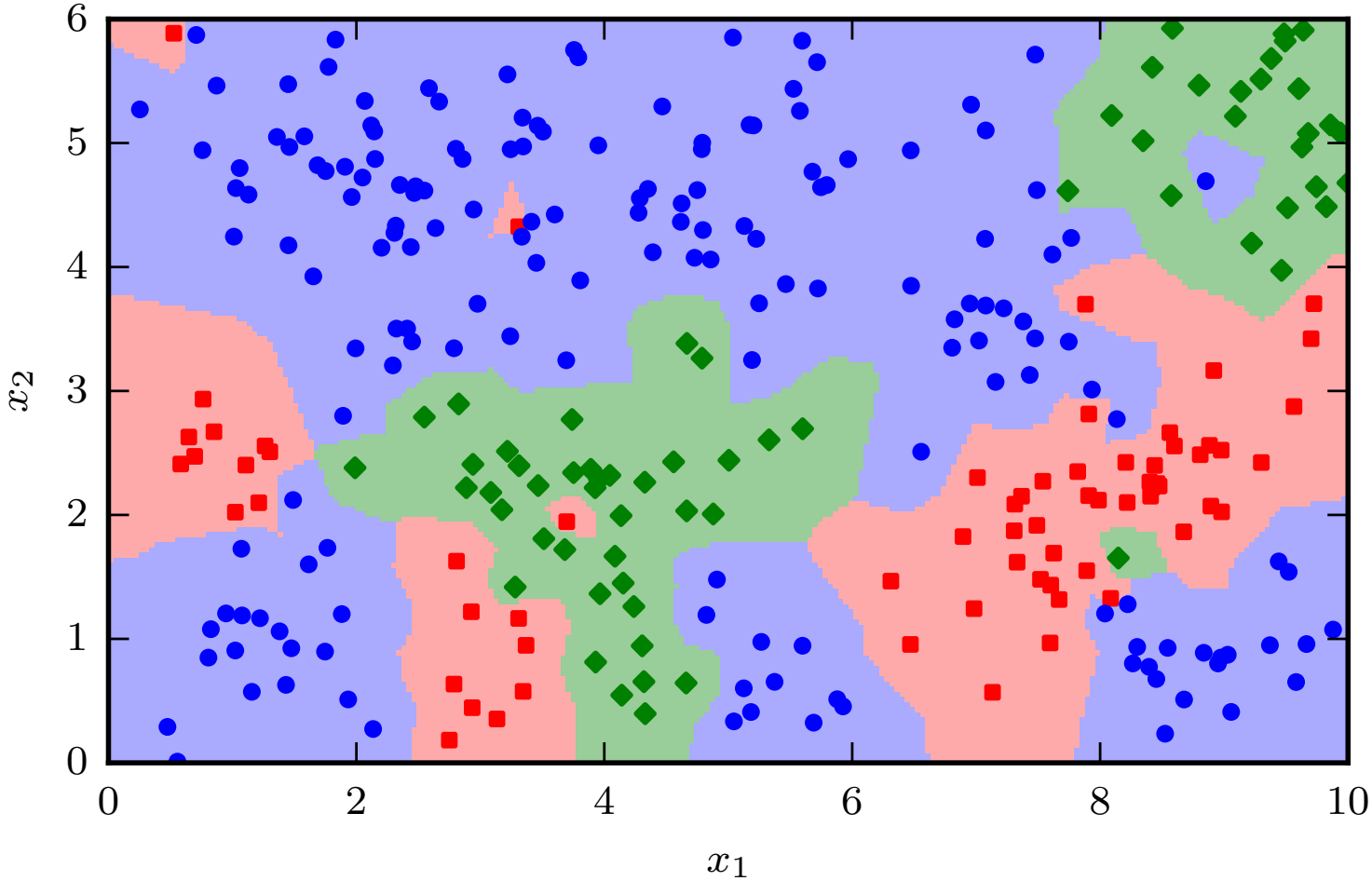
How do we intuitively label new samples by hand?
Look at the *surrounding* points.

1-NN Algorithm

- define a distance measure (e.g. Euclidean distance)
- compute the nearest neighbor for all new data points
- and label them with the label of their nearest neighbor

This works for both *classification* and *regression*.

1-NN



This corresponds to a Voronoi tesselation.
And results in overfitting...

k-Nearest Neighbor Classification

More *robust* against errors in the training set:

Look at a number of neighbors!

Let $\mathcal{N}_k(\boldsymbol{x})$ be the k nearest neighbors of a vector $\boldsymbol{x}$, then in classification tasks:

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

$$y = \arg\max_c p(z = c \mid \boldsymbol{x}, k)$$

with the *indicator variable* $\mathbb{I}(e)$ is defined as:

$$\mathbb{I}(e) = \begin{cases} 1 \text{ if } e \text{ is true} \\ 0 \text{ if } e \text{ is false.} \end{cases}$$

i.e., the vector will be labeled by the mode of its neighbors' labels.

k-Nearest-Neighbor Regression

Regression is similar:

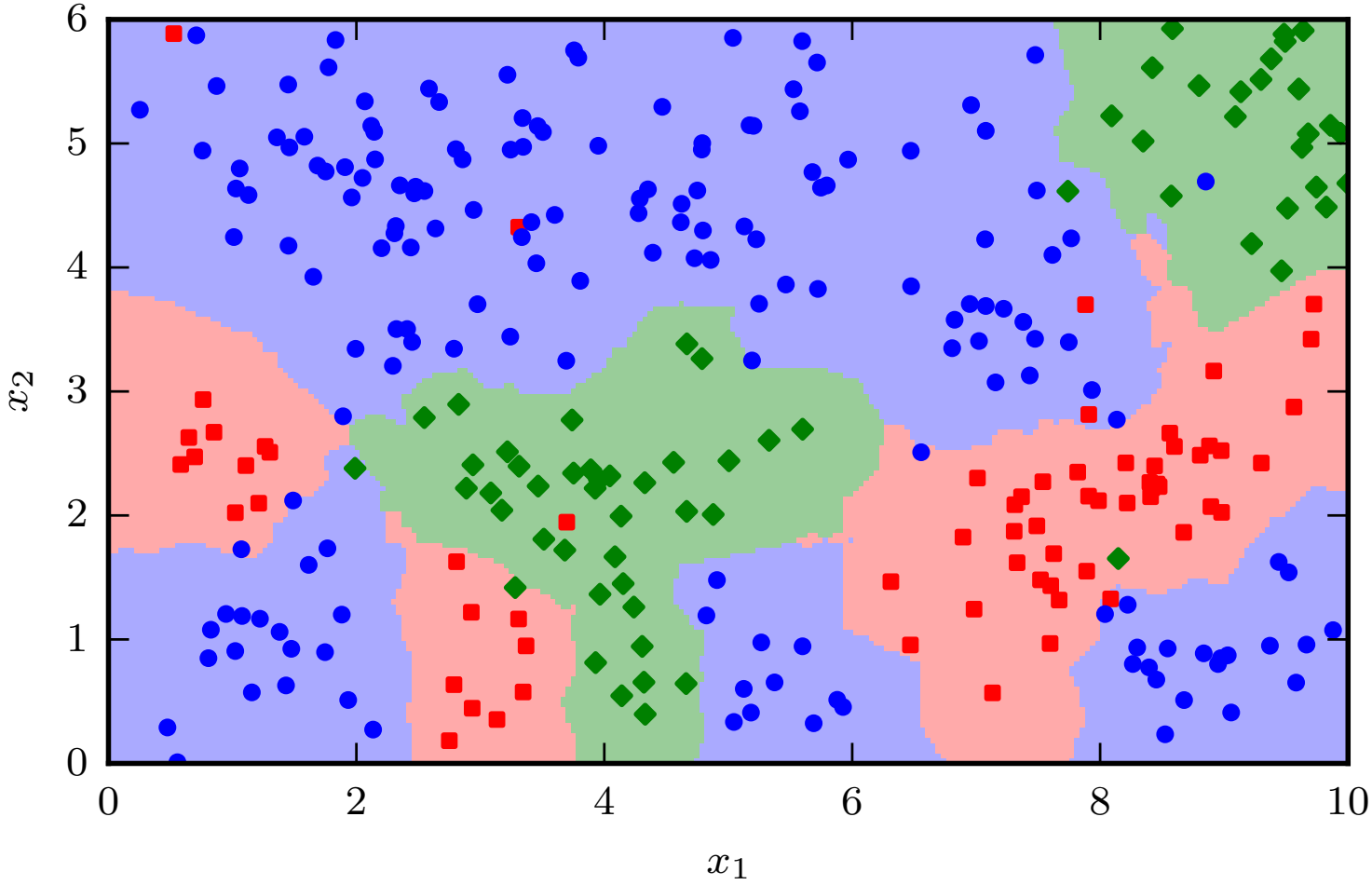
Let $\mathcal{N}_k(\boldsymbol{x})$ be the k nearest neighbors of a vector $\boldsymbol{x}$, then for regression:

$$y = \frac{1}{\mathcal{C}} \sum_{i \in \mathcal{N}_k(\boldsymbol{x})} \frac{1}{\mathrm{d}(\boldsymbol{x}, \boldsymbol{x_i})} \, z_i,$$

with $\mathcal{C} = \sum_{i \in \mathcal{N}_k(x)} \frac{1}{\mathrm{d}(\boldsymbol{x}, \boldsymbol{x_i})}$ the normalization constant and $\mathrm{d}(\boldsymbol{x}, \boldsymbol{x_i})$ being a distance measure between $\boldsymbol{x}$ and $\boldsymbol{x_i}$,

i.e., the vector will be labeled by a *weighted mean* of its neighbors' values.

3-NN

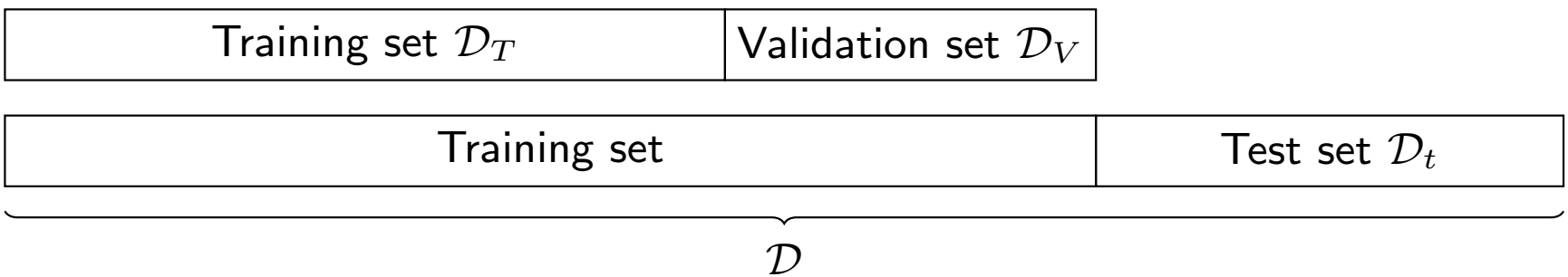


So, how many neighbors are best?

Determining k

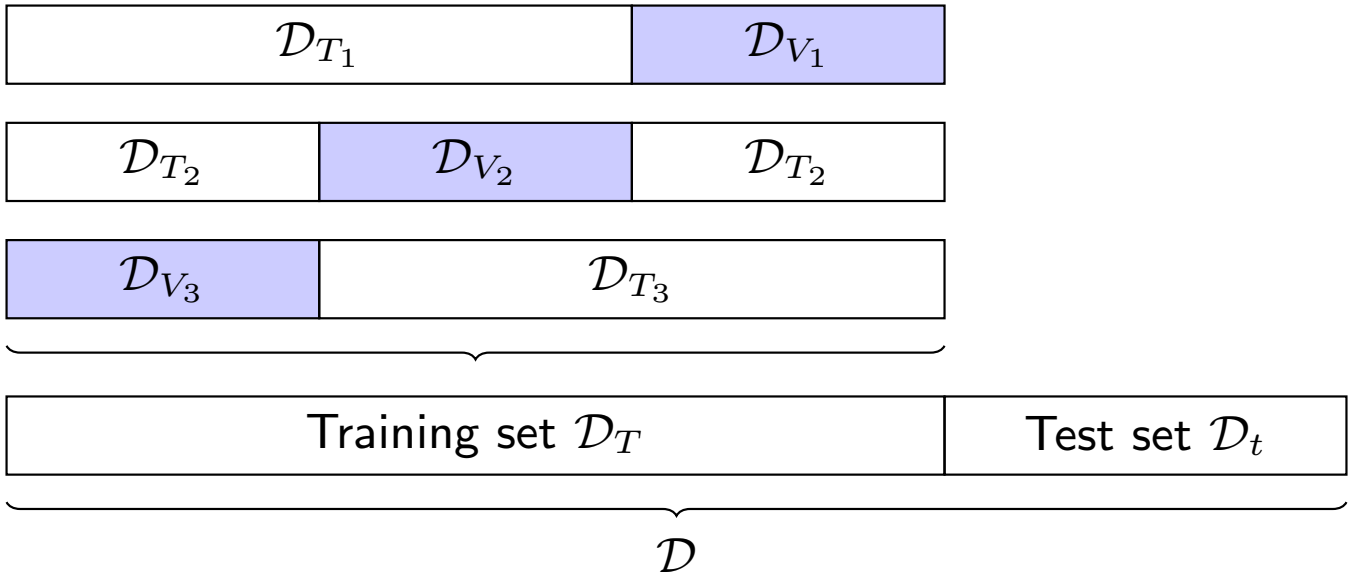
k is called a *hyper-parameter*.

Keep part of your training set separate and evaluate the algorithm with different k on this *validation set* to choose the best.



Better even: Do cross-validation!

K-fold Cross-Validation



- Split your training data into K folds (10-fold CV is common).
- Use $K-1$ folds for training and the remaining for evaluation.
- Average over all folds to get an estimate of the error for a certain setting of your *hyper-parameters* (such as k in k-NN).
- Try different settings for your hyper-parameters.
- Use all your training data and the best hyper-parameters for final training (and testing) of your model.

The extreme case - LOOCV

In *leave-one-out-cross validation* (LOOCV) we train on all but one sample.

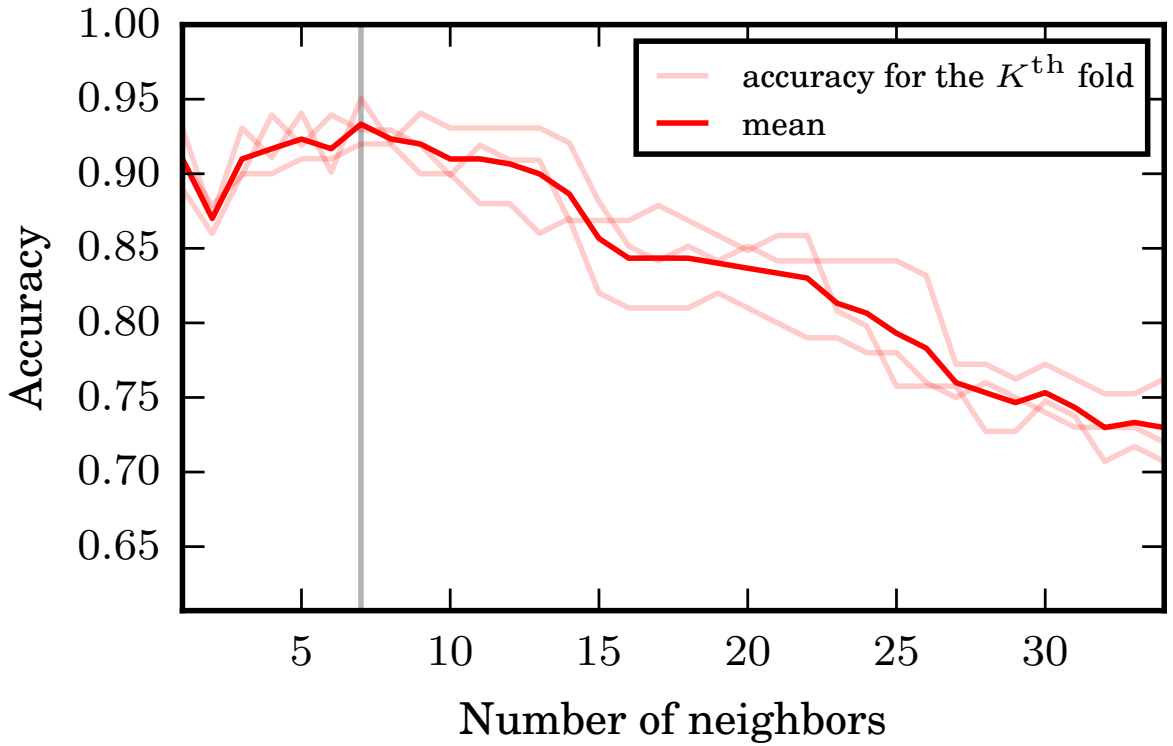
If we have N samples, this is the same as N-fold cross-validation.

LOOCV is interesting if we do not have a lot of data and we want to use as much of it for training as possible but still get a good estimate of model performance.

But it also means that we need to train our model N times...

If we have sufficiently large amounts of data and training our model is computationally expensive, we better stick to lower numbers of K or a single validation set.

Cross-validation for determining k

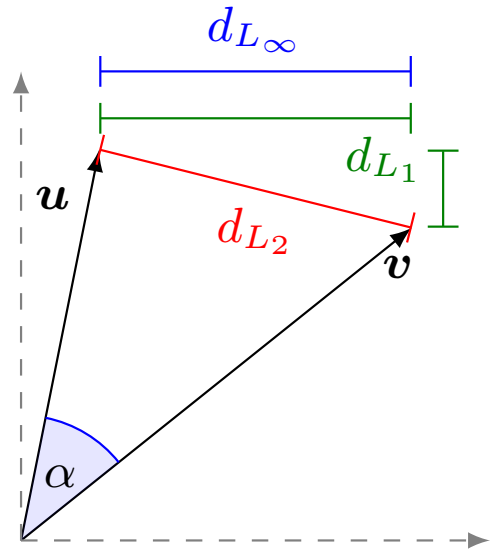


We choose $k = 7$.

Distance Measures

- Euclidean distance (L_2 norm): $\sqrt{\sum_i (u_i - v_i)^2}$
- L_1 norm: $\sum_i |u_i - v_i|$
- L_∞ norm: $\max_i |u_i - v_i|$
- Angle:

$$\cos\alpha = \frac{\boldsymbol{u}^T \boldsymbol{v}}{\|\boldsymbol{u}\|\|\boldsymbol{v}\|}$$

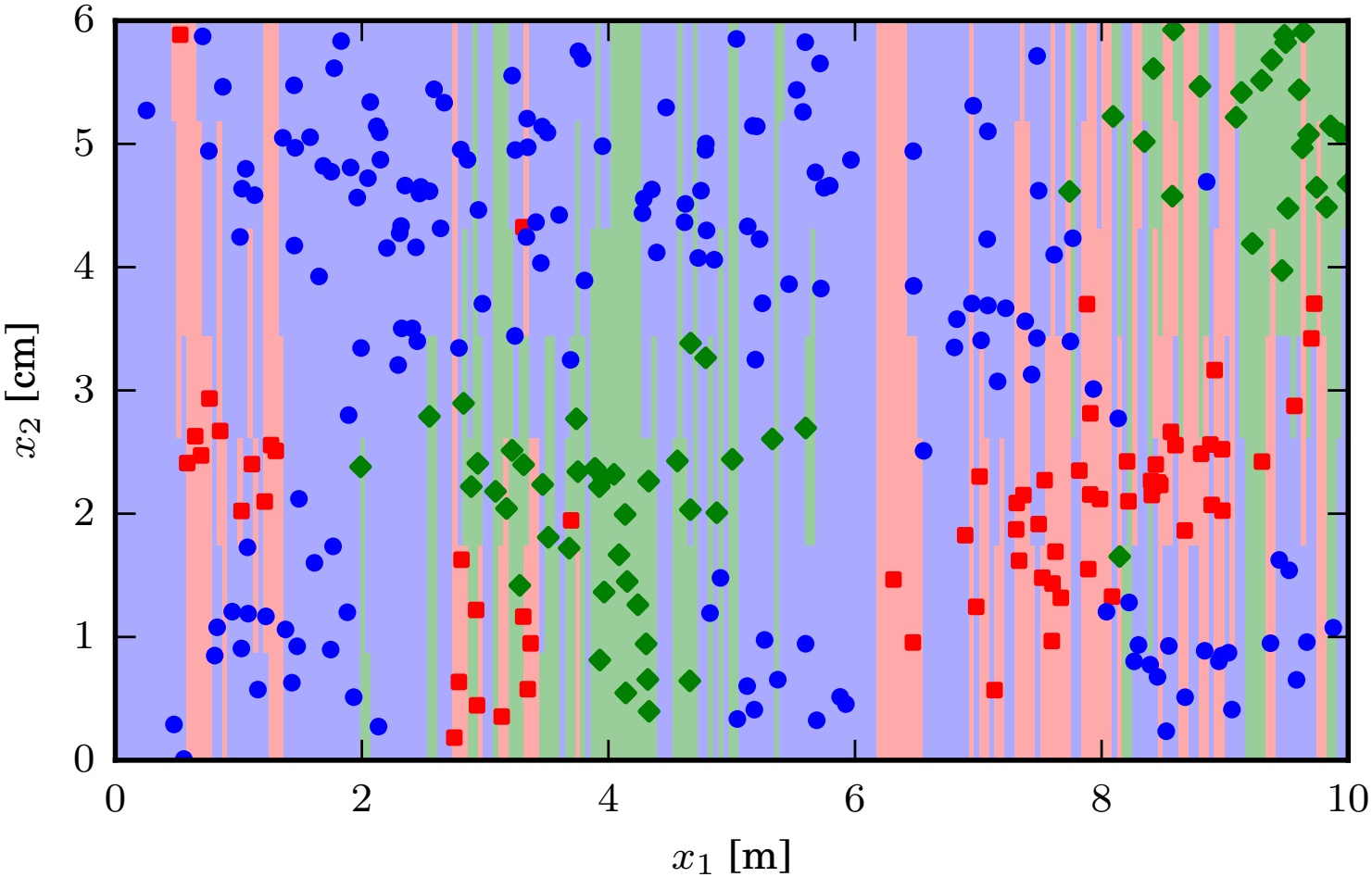


- *Mahalanobis distance* ($\boldsymbol{\Sigma}$ is *positive (semi) definite* and *symmetric*):

$$\sqrt{(\boldsymbol{u} - \boldsymbol{v})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{u} - \boldsymbol{v})}$$

- Hamming distance, Edit distance, ...

Scaling Issues



The same old example but one of our features is in the order of meters, the other in the order of centimeters. $(k = 1)$

Circumventing Scaling Issues

- Data *standardization*
  Scale each feature to zero mean and unit variance.

$$x_{i,\text{std}} = \frac{x_i - \mu_i}{\sigma_i}$$

  (This is a standard procedure in machine learning. Many models are sensitive to differences in scale.)

- Use the Mahalanobis distance.

$$\text{mahalanobis}(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sqrt{(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_1 - \boldsymbol{x}_2)}$$

If $\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \cdots & 0 \\ 0 & 0 & \sigma_n^2 \end{bmatrix}$, this is equal to *normalized Euclidean distance*

The curse of dimensionality

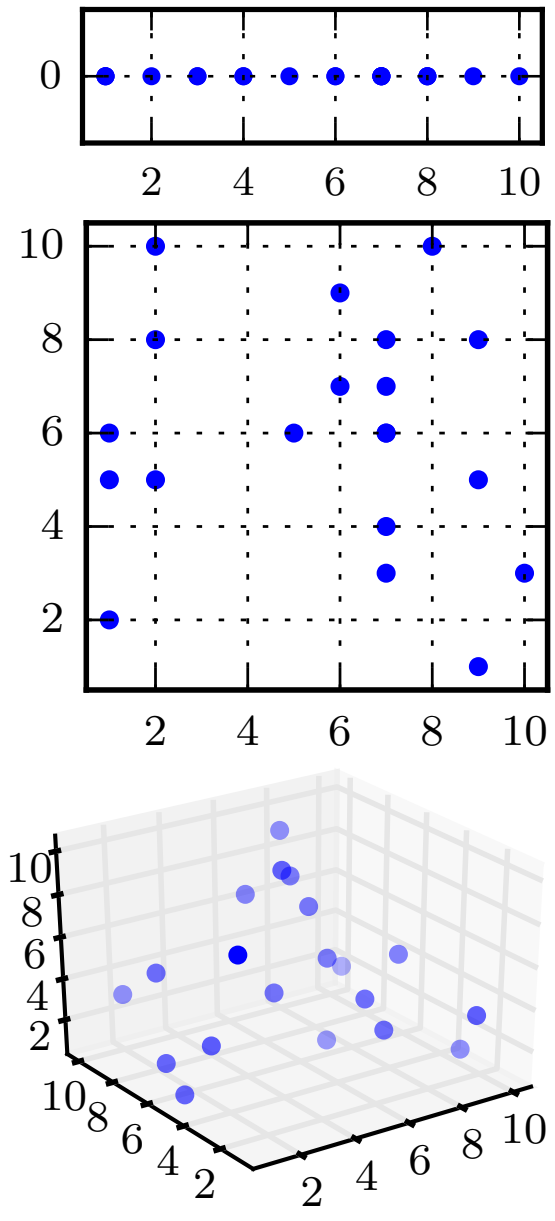
Say, you have a discrete one-dimensional input space $x \in \{1, 2, \ldots, 10\}$. Your data is uniformly distributed over this space.

You have $N = 20$ samples and your 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 input space.

Once you add a third dimension you only cover 2%. The nearest neighbor will now be pretty far away..

N has to grow exponentially with the number of features. Consider this when using k-NN on high-dimensional data.



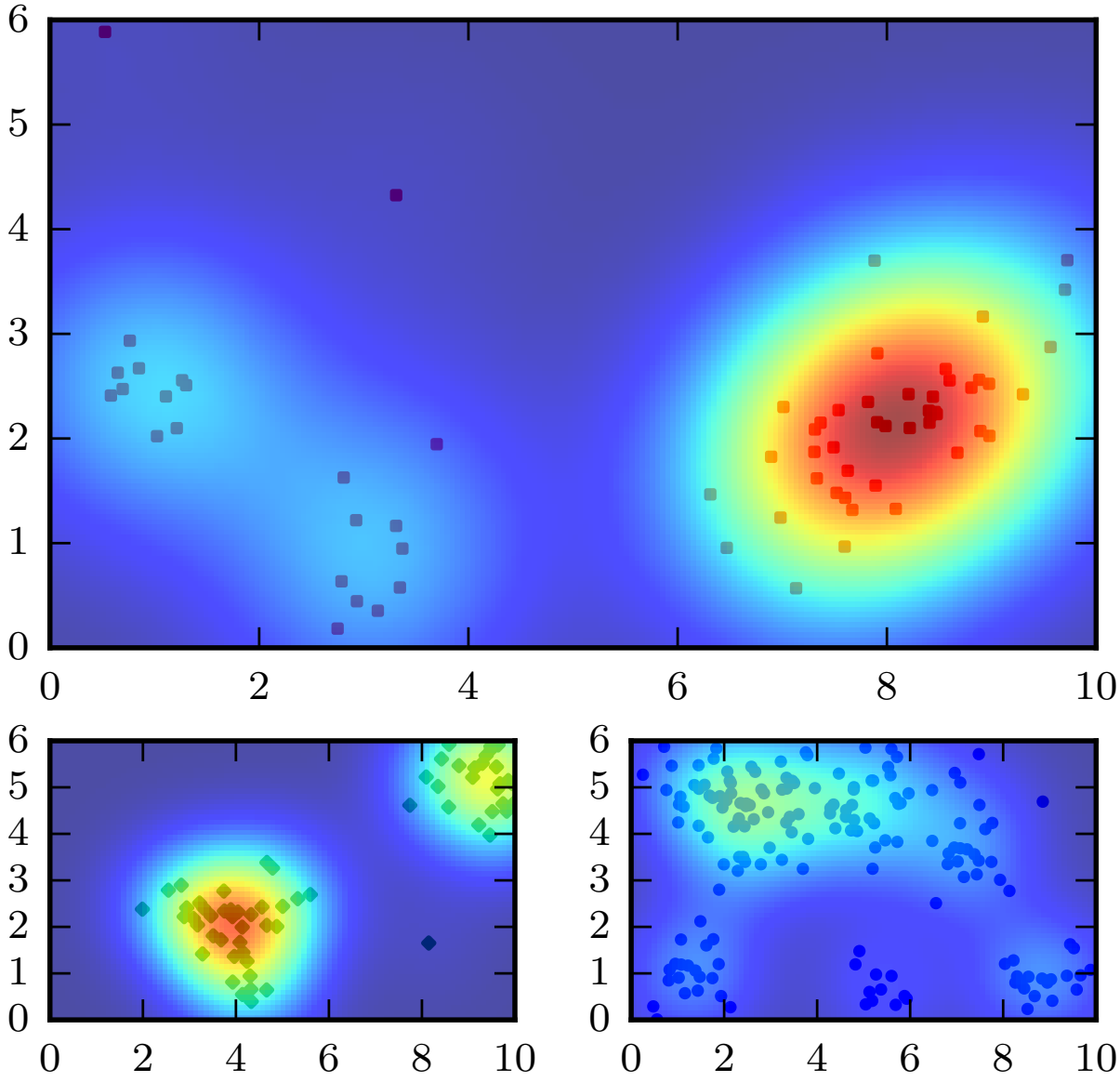
A probabilistic interpretation of k-NN

Assumption: Data points are distributed normally.
A *generative* model for data from class c:

$$
\begin{aligned}
p(\boldsymbol{x} \mid z = c) &= \frac{1}{N_c} \sum_{\boldsymbol{x}_n \in \text{class } c} \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{x}_n, \sigma^2 \boldsymbol{I}) \\
&= \frac{1}{N_c} \frac{1}{(2\pi\sigma^2)^{D/2}} \sum_{\boldsymbol{x}_n \in \text{class } c} e^{-\frac{\|\boldsymbol{x}-\boldsymbol{x}_n\|^2}{(2\sigma^2)}},
\end{aligned}
$$

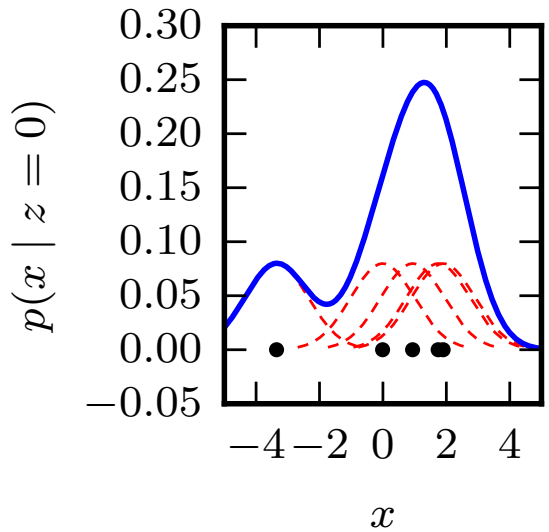
where D is the dimension of the data points, N_c is the number of training points in class c.

Generative models for classes red, blue and green



A probabilistic interpretation of k-NN

A *Parzen density estimator*: Uniform weighted sum of Gaussian distributions centered on the training points.



Bayes' rule helps with a new data point $\boldsymbol{x}^*$:

$$p(z = c \mid \boldsymbol{x}^*) = \frac{p(\boldsymbol{x}^* \mid z = c)\ p(z = c)}{\sum_{i \in C} p(\boldsymbol{x}^* \mid z = i)\ p(z = i)}$$

$$p(z = c \mid \boldsymbol{x}^*) \propto p(\boldsymbol{x}^* \mid z = c)\ p(z = c)$$

Neighborhood Component Analysis - NCA

Instead of defining the distance metric for k-NN, we can also learn it!
We want to find a *good* $\boldsymbol{Q}$ for the (squared) Mahalanobis distance:

$$\text{mahalanobis}(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\boldsymbol{x}_1 - \boldsymbol{x}_2)^T \boldsymbol{Q} (\boldsymbol{x}_1 - \boldsymbol{x}_2)$$

For *any* such $\boldsymbol{Q}$ we can find $\boldsymbol{A}$, such that $\boldsymbol{Q} = \boldsymbol{A}^T \boldsymbol{A}$, so we can rewrite the distance as:

$$\text{mahalanobis}(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\boldsymbol{A}\boldsymbol{x}_1 - \boldsymbol{A}\boldsymbol{x}_2)^T (\boldsymbol{A}\boldsymbol{x}_1 - \boldsymbol{A}\boldsymbol{x}_2)$$

J. Goldberger, S. Roweis, G. Hinton, R. Salakhutdinov. *Neighborhood Component Analysis*. 2005.

NCA - a good metric

How do we define *good*?

$\boldsymbol{A}$ should be chosen in such a way that when we take one exemplar from the training set and try to classify it based on the remaining training set, it should be classified into its *known* category. And this should hold for *all* exemplars in the training set (*Leave One Out Cross Validation—LOOCV*).

But an *infinitesimal* change in $\boldsymbol{A}$ can effect the classification performance by a *finite* amount.

NCA - the probabilistic perspective

A better-behaving performance measure (still based on LOOCV):
Don't assign classes directly but use probabilities instead.

$$p_{ij} = \frac{\exp\left(-\|\boldsymbol{A}\,\boldsymbol{x}_i - \boldsymbol{A}\,\boldsymbol{x}_j\|^2\right)}{\sum_{k \neq i} \exp\left(-\|\boldsymbol{A}\,\boldsymbol{x}_i - \boldsymbol{A}\,\boldsymbol{x}_k\|^2\right)}, \qquad p_{ii} = 0$$

Each training point i selects another point j as its neighbor with some probability p_{ij}.

We compute the probability p_i that point i will be correctly classified:

$$p_i = \sum_{j \in C_i} p_{ij}, \qquad C_i = \{j \mid c_i = c_j\}$$

NCA - Finding $\boldsymbol{A}$

The goal is now to find some $\boldsymbol{A}$ that maximizes the *expected number of correctly classified points*:

$$f(\boldsymbol{A}) = \sum_i \sum_{j \in C_i} p_{ij} = \sum_i p_i$$

A function (*objective*) that depends solely on $\boldsymbol{A}$.
How to *maximize* it? Follow the *gradient* until the top (*gradient ascent*).
A gradient with respect to a matrix!

$$\frac{\partial f}{\partial A} = 2A \sum_i \left(p_i \sum_k p_{ik} \boldsymbol{x}_{ik} \boldsymbol{x}_{ik}^T - \sum_{j \in C_i} p_{ij} \boldsymbol{x}_{ij} \boldsymbol{x}_{ij}^T \right)$$

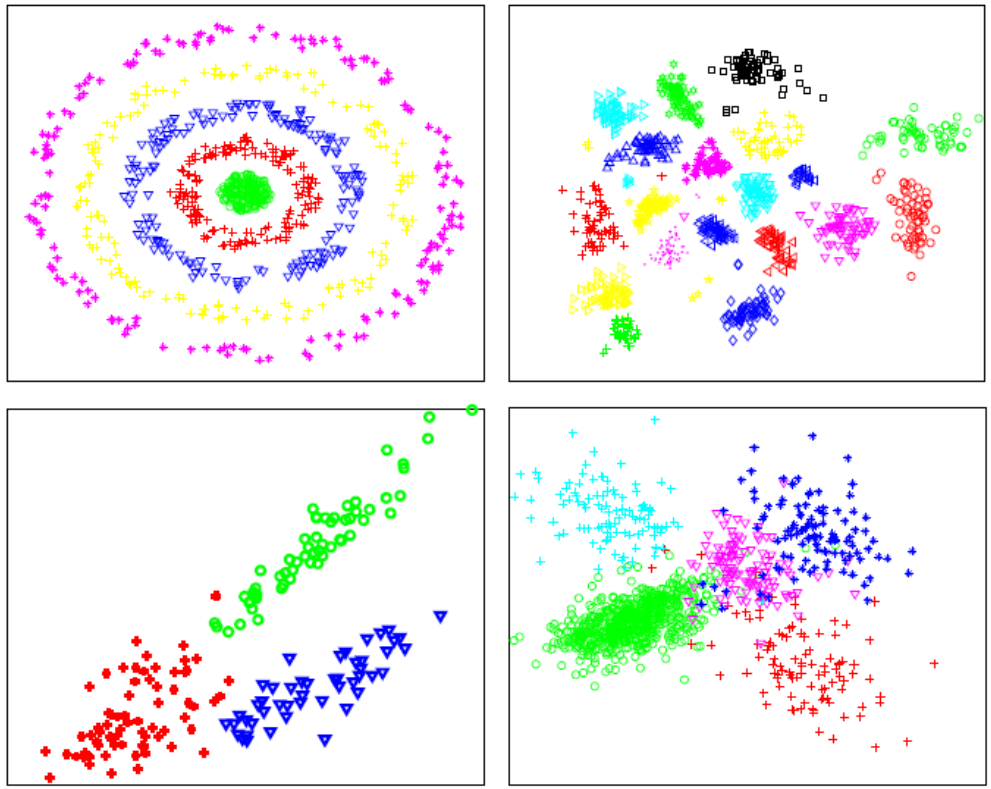
with $\boldsymbol{x}_{ij} = \boldsymbol{x}_i - \boldsymbol{x}_j$.

NCA - Impact of $\boldsymbol{A}$

The algorithm also learns automatically a real-valued estimate of the optimal number of neighbors: by scaling up $\boldsymbol{A}$ uniformly, the method consults fewer neighbors and vice versa.

By restricting $\boldsymbol{A}$ to be a non-square matrix of size $d \times D$, NCA can also do *linear dimensionality reduction* (thus, the metric will be of low rank if $d \leq D$).

NCA for dimensionality reduction



NCA on data that was originally 3-D (top left), 13-D (lower left), 560-D (top right, face data) and 256-D (lower right, digits data)

What we learned

- k-NN Algorithm
- Cross-Validation
- Distance Metrics
- Curse of Dimensionality
- Probabilistic k-NN and NCA