Lecture 2

Basic Concepts

Luigi Freda

ALCOR Lab
DIAG
University of Rome "La Sapienza"

December 4, 2016

Outline

Outline

Parametric vs Non-parametric Models

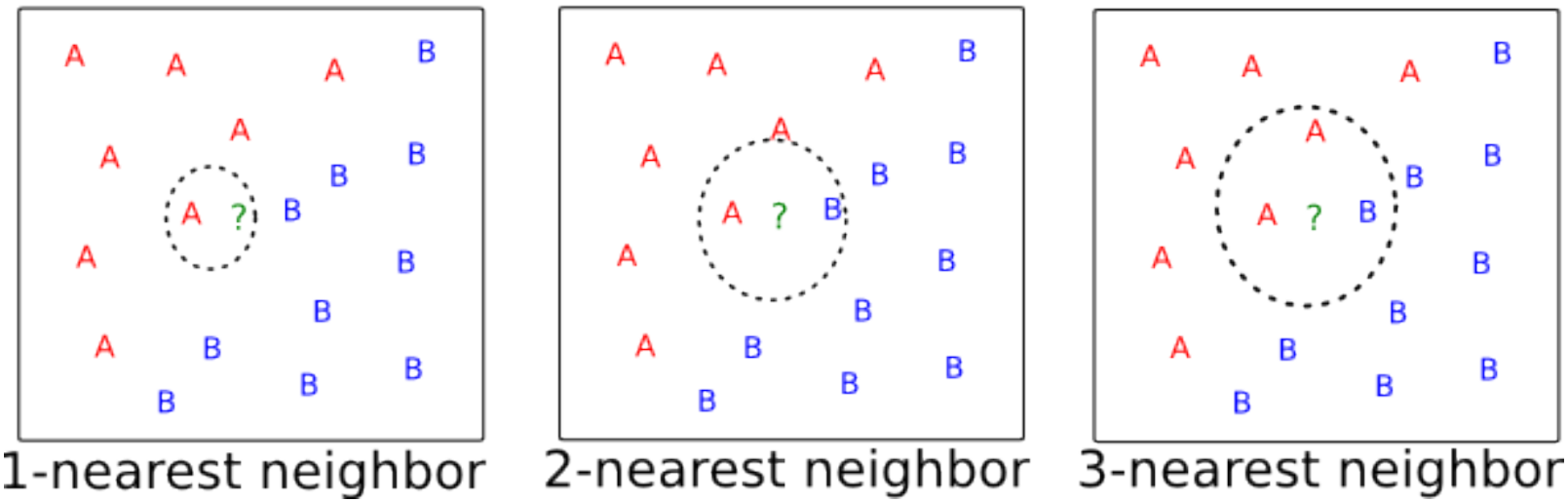
- we will focus on probabilistic models of the **form**:
  - ✓ $p(y|\mathbf{x})$ for supervised learning
  - ✓ $p(\mathbf{x})$ for unsupervised learning

  there are many ways to define such models
- one of the most important distinction:
  - ✓ **parametric models**: have a fixed number of parameters
  - ✓ **non-parametric models**: the number of parameters grow with the amount of training data
- **pros and cons**
  - ✓ **parametric models** have the advantage of often being **faster** to use, but the disadvantage of making **stronger assumptions** about the nature of the data distributions
  - ✓ **non-parametric models** are **more flexible**, but often **computationally intractable** for large datasets

Outline

K-nearest Neighbors Classifier

- a simple example of a non-parametric classifier is the **K nearest neighbor** (KNN) **classifier**
- this simply "looks at" the K points in the training set that are nearest to the test input x
- **memory-based learning**, it can be derived from probabilistic framework



K-nearest Neighbors Classifier

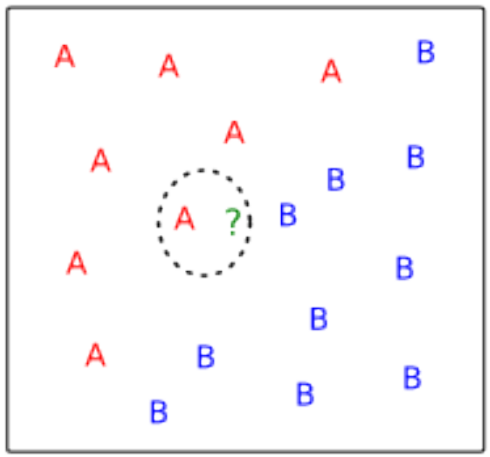
more formally

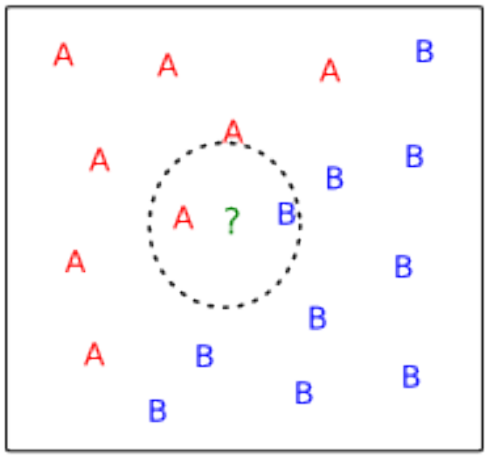
$$p(y = c|\mathbf{x}, \mathcal{D}, K) = \frac{1}{N} \sum_{i \in N_K(x,D)} \mathbb{I}(y_i = c)$$

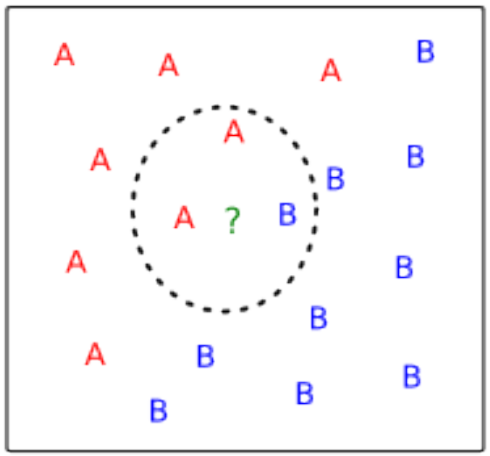
where

- $N_K(x, D)$ are the K nearest points to $\mathbf{x}$
- $\mathbb{I}(e) = \begin{cases} 1 & \text{if } e = \text{true} \\ 0 & \text{if } e = \text{false} \end{cases}$ is the **indicator function**

N.B.: the higher the value of K, the more we average local data







Outline

The Curse of Dimensionality

1/2

- in general KNN classifier is simple and works well
- problem: it has poor performance with **high dimensional inputs**

why?

consider an high-dimensional input space ($D >> 1$)

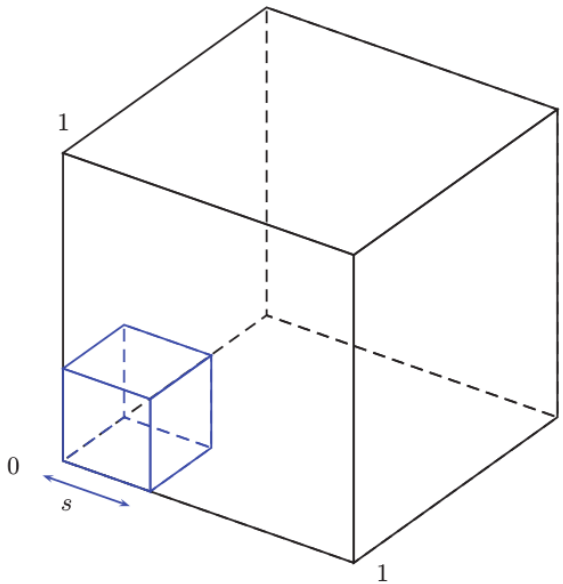
- the number of training instances needs to grow **exponentially** with the number of dimensions D to maintain a given **accuracy**
- the method becomes no longer local

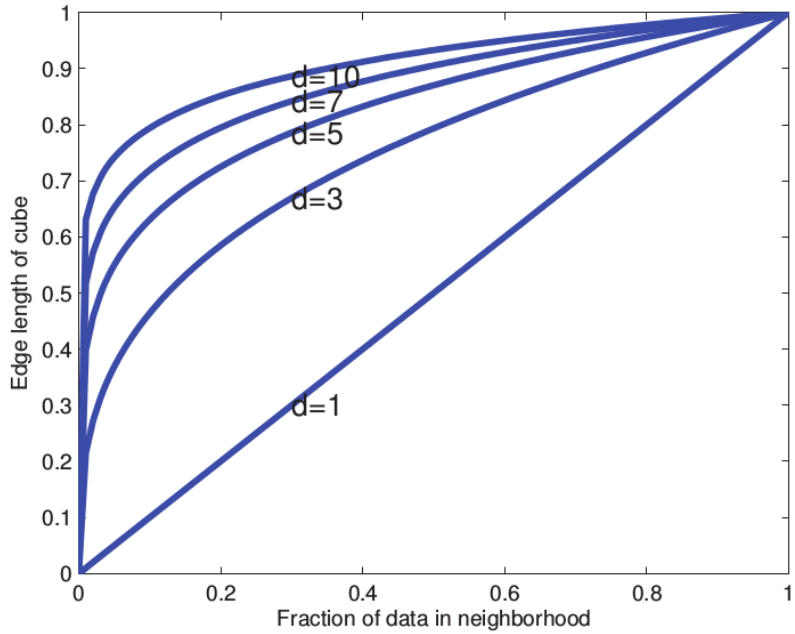
let's see this in more detail..

The Curse of Dimensionality

2/2

- assume data are **uniformly distributed** in the D-dimensional **unit cube**
- suppose we estimate the density of class labels around a test point **x** by "**growing**" a hyper-cube around **x** until it contains a **desired fraction** f of the data points
- the expected edge length of this cube will be $e_D(f) = f^{1/D}$
- if $D = 10$, and we want to base our estimate on $f = 10\%$ of the data, then $e_{0.1} = 0.8$ and we need to extend the cube 80% along each dimension around **x**!!
- with $f = 10\%$ and $D = 10$ the method is **no more local** and we have to look at points that are far away

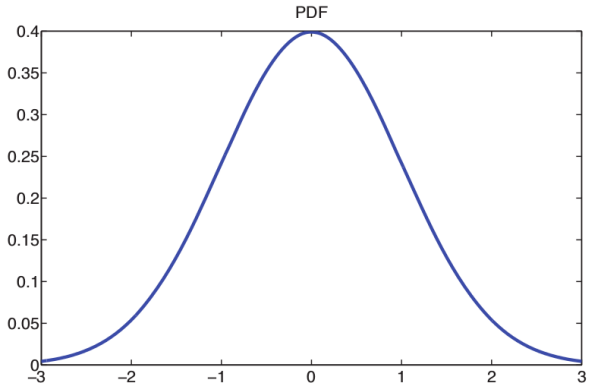




Outline

Linear Regression

$$y(x) = \mathbf{w}^T\mathbf{x} + \epsilon = \sum_{j=1}^{D} w_j x_j + \epsilon$$



- $\mathbf{w} \in \mathbb{R}^D$ is the **weight vector**
- $\epsilon \sim \mathcal{N}(\mu, \sigma^2)$ is the **residual error**
- $\mathcal{N}(\mu, \sigma^2)$ is the **Gaussian distribution**

This entails

$$p(y|\mathbf{x}, \theta) = \mathcal{N}(\mu(\mathbf{x}), \sigma^2(x)) = \mathcal{N}(\mathbf{w}^T\mathbf{x}, \sigma^2)$$

- $\mu(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = [w_0, \tilde{\mathbf{w}}^T\tilde{\mathbf{x}}]^T$ where $\mathbf{x} = [1, \tilde{\mathbf{x}}]^T$
- $\sigma(x) = \sigma$
- $\boldsymbol{\theta} = (\mathbf{w}, \sigma^2)$ are the **model parameters**

Polynomial Regression

if we replace $\mathbf{x}$ by a non-linear function $\phi(\mathbf{x})$

$$y(x) = \mathbf{w}^T \phi(\mathbf{x}) + \epsilon$$

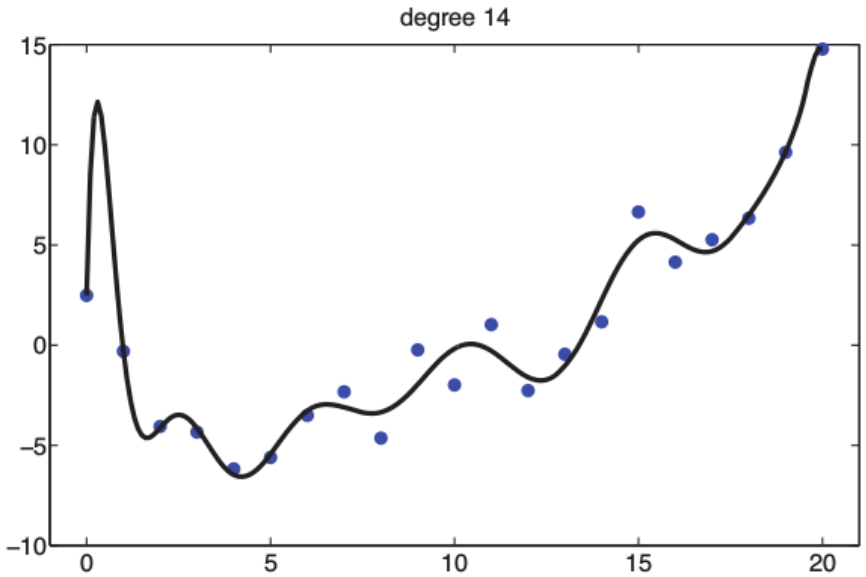
we now have

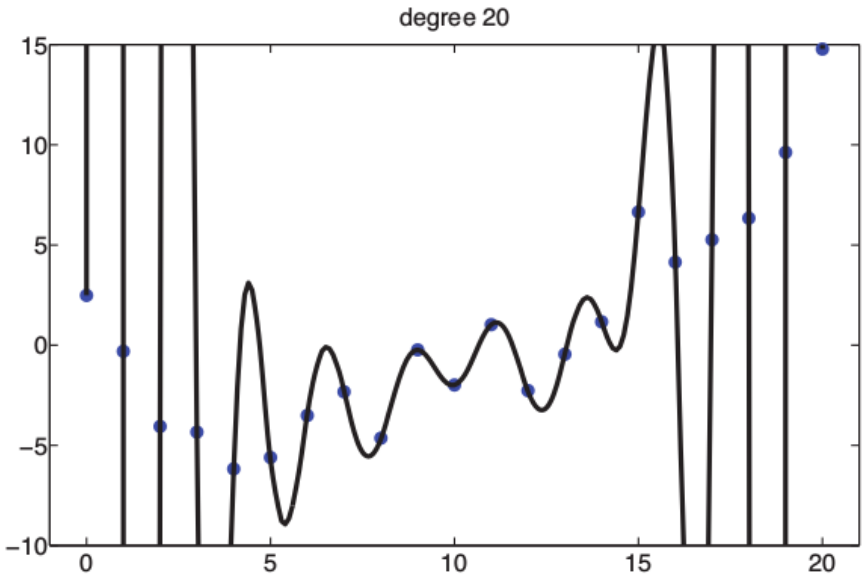
$$p(y|\mathbf{x}, \theta) = \mathcal{N}(\mathbf{w}^T \phi(\mathbf{x}), \sigma^2)$$

- $\mu(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$ (**basis function expansion**)
- if $x \in \mathbb{R}$ we can use $\phi(x) = [1, x, x^2, ..., x^d]$ which is the vector of **polynomial basis functions**
- in general if $\mathbf{x} \in \mathbb{R}^D$ we can use a **multivariate polynomial expansion** $\mathbf{w}^T \phi(\mathbf{x}) = \sum w_{i_1 i_2 \ldots i_D} \prod_{j=1}^{D} x_j^{i_j}$ up to a certain degree d
- $\boldsymbol{\theta} = (\mathbf{w}, \sigma^2)$ are the **model parameters**

Parametric Models

Linear Regression





- input: 21 data points (x_i, y_i)
- left: polynomial of degrees 14
- right: polynomial of degrees 20

?do we obtain a better result by increasing the model complexity?

Outline

Logistic Regression

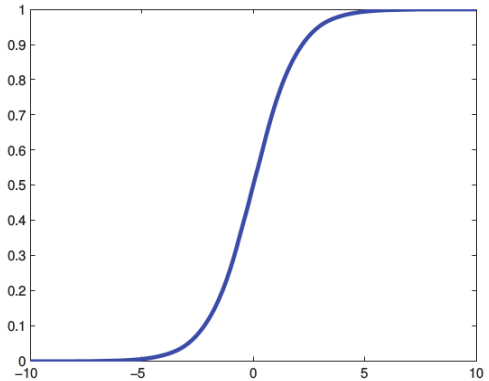
?can we generalize linear regression ($y \in \mathbb{R}$) to binary classification ($y \in \{0, 1\}$)?

two steps:

1. replace $\mathcal{N}(\mu(\mathbf{x}), \sigma^2(x))$ with $\mathrm{Ber}(y|\mu(\mathbf{x}))$ (we want $y \in \{0, 1\}$)
2. replace $\mu(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ with $\mu(\mathbf{x}) = \mathrm{sigm}(\mathbf{w}^T\mathbf{x})$ (we want $0 \leq \mu(\mathbf{x}) \leq 1$)

where

- $\mathrm{Ber}(y|\mu(\mathbf{x})) = \mu(\mathbf{x})^{\mathbb{I}(y=1)}(1 - \mu(\mathbf{x}))^{\mathbb{I}(y=0)}$ is the **Bernoulli distribution**
- $\mathbb{I}(e) = 1$ if e is true, $\mathbb{I}(e) = 0$ if e is false (**indicator function**)
- $\mathrm{sigm}(\eta) = \frac{1}{1+\exp(-\eta)}$ is **the sigmoid function** (aka logistic function)



Logistic Regression

1. replace $\mathcal{N}(\mu(\mathbf{x}), \sigma^2(x))$ with $\mathrm{Ber}(y|\mu(\mathbf{x}))$ (we want $y \in \{0, 1\}$)
2. replace $\mu(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ with $\mu(\mathbf{x}) = \mathrm{sigm}(\mathbf{w}^T\mathbf{x})$ (we want $0 \leq \mu(\mathbf{x}) \leq 1$)

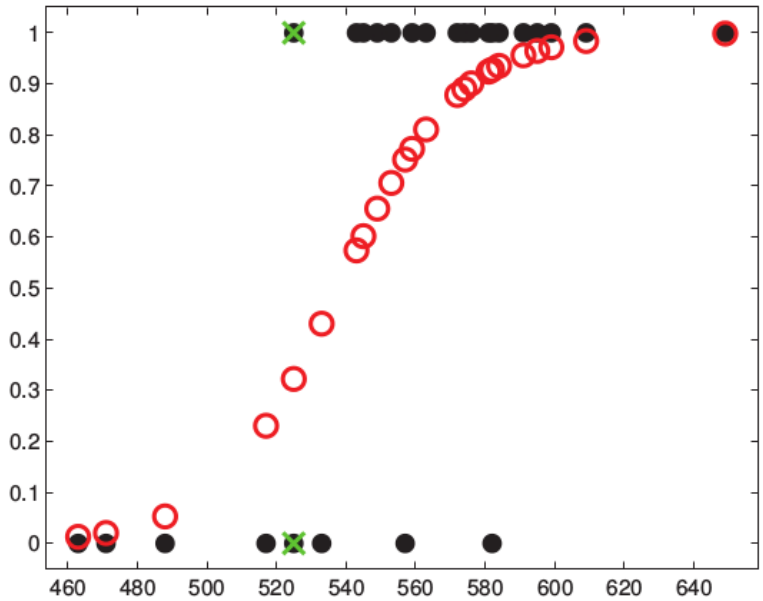
hence, we started from a linear regression

$$p(y|\mathbf{x}, \theta) = \mathcal{N}(\mathbf{w}^T\mathbf{x}, \sigma^2) \qquad \text{where } y \in \mathbb{R}$$

to obtain a logistic regression

$$p(y|\mathbf{x}, \mathbf{w}) = \mathrm{Ber}(y|\mathrm{sigm}(\mathbf{w}^T\mathbf{x})) \qquad \text{where } y \in \{0, 1\}$$

Logistic regression - an example



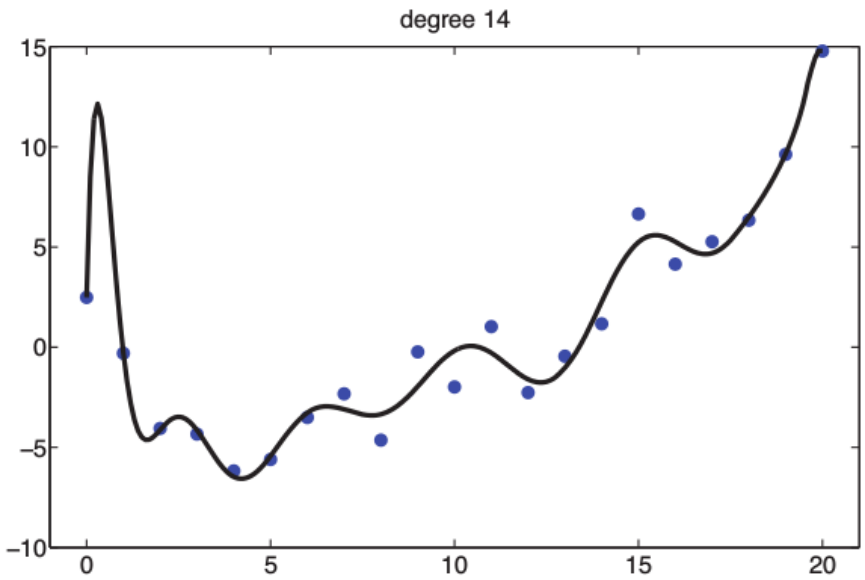
- solid black dots are data (x_i, y_i)
- open red circles are **predicted probabilities**: $p(y_i = 1|x_i, \mathbf{w}) = \text{sigm}(w_0 + w_1 x_i)$
- data is **not** linearly separable
- in particular, here we have different y_i for a same value x_i

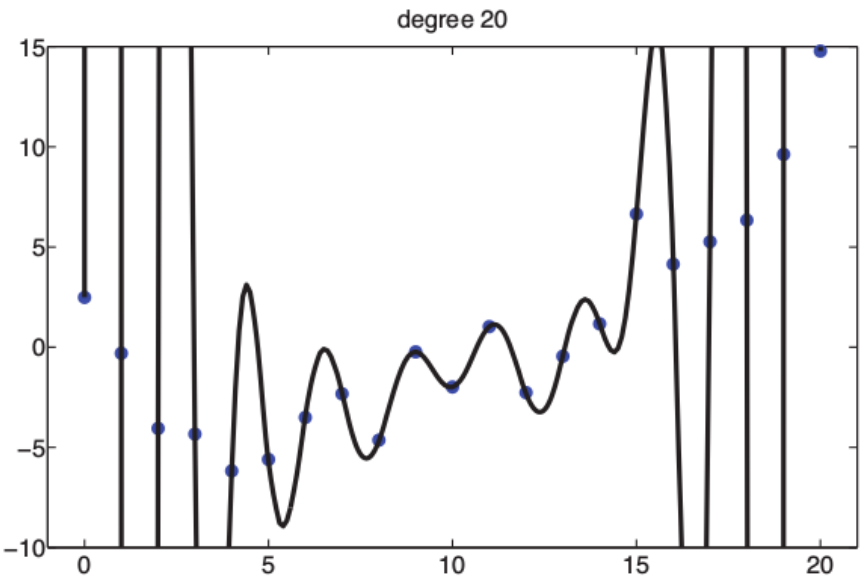
in general when data is not linearly separable, we can try to use the basis function expansion as a further step

Outline

Overfitting

- when we fit **highly flexible models**, we should avoid trying to model every minor variation in the input
- these minor variations **are more likely to be noise than "true" signal**

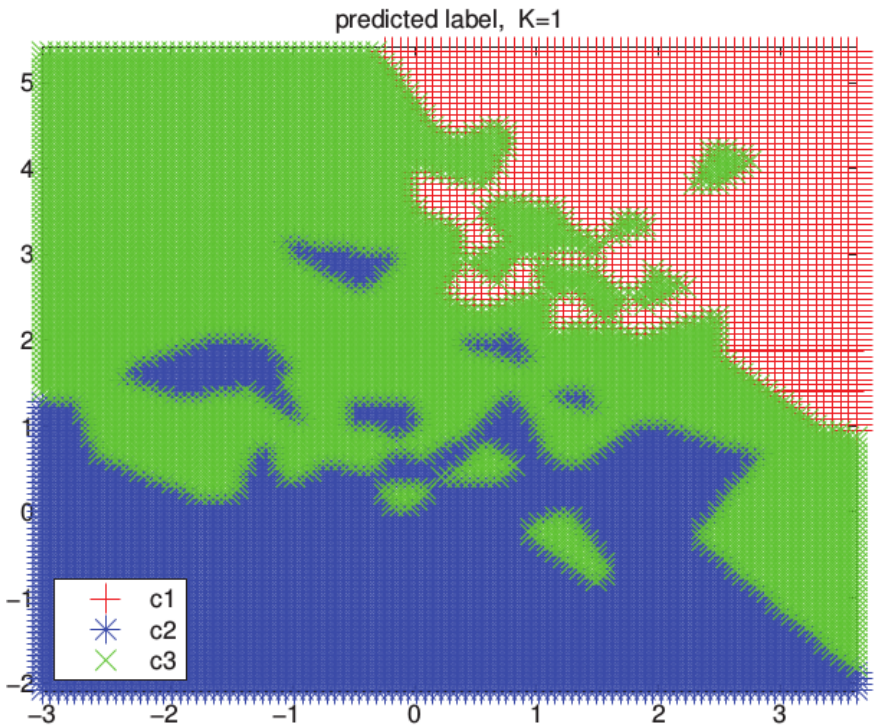


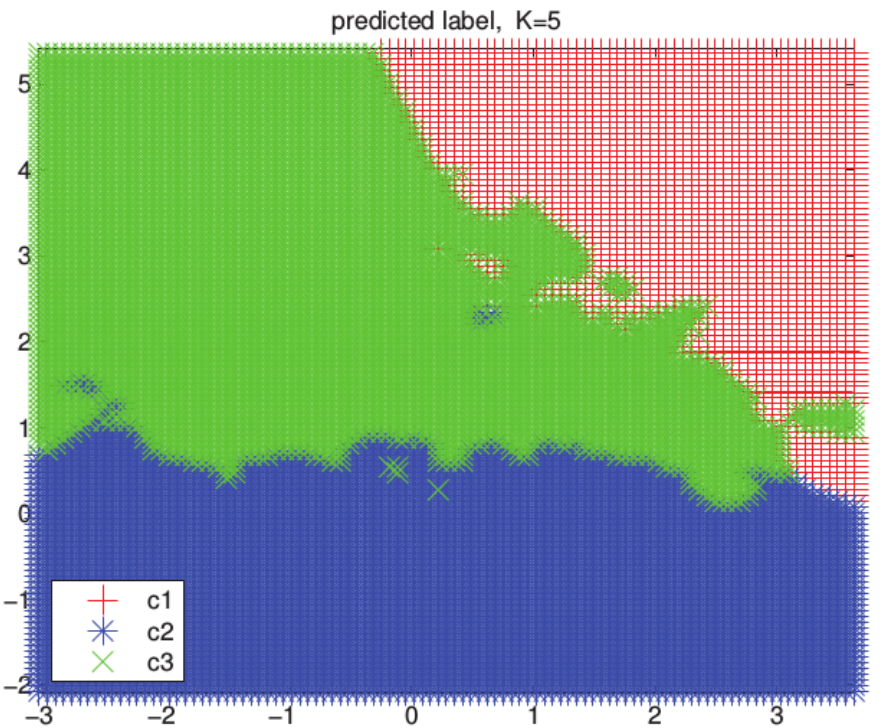


!**pay attention**: do not to fit noise!

Overfitting

an example with KNN





N.B.: the higher the value of K, the more we average local data

Outline

Model Selection

- suppose we have different models M_i, how to choose? (e.g. we have to select K for the KNN classifier)
- if $f(\mathbf{x})$ is a classifier we can compute its **misclassification rate**

$$\text{err}(f, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(f(\mathbf{x}_i) \neq y_i)$$

- consider a KNN classifier: in principle, we can select K so as to have the **minimum misclassification rate** on the training set
- but our model is valuable if it returns a low misclassification rate over future data (**generalization error**) and not on the training set itself

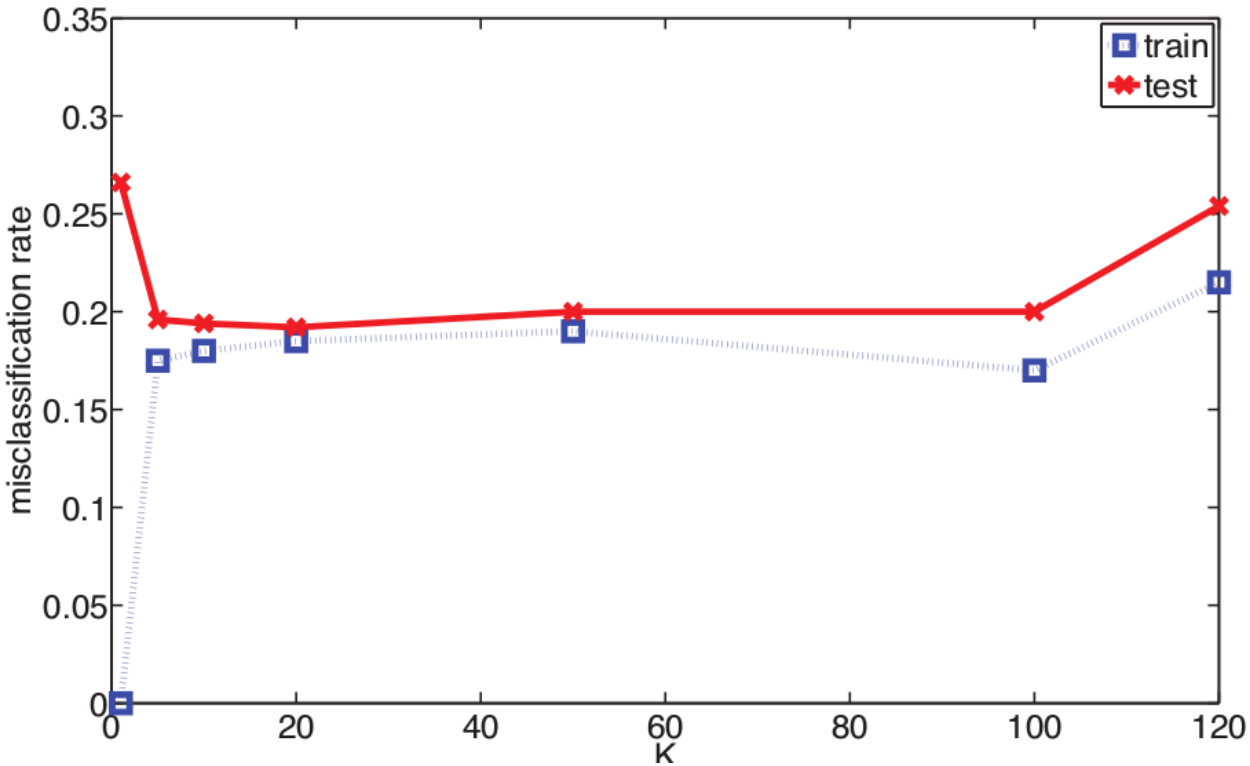
- **training set** $\mathcal{D} \longrightarrow$ for estimating the model
- **test set** $\mathcal{T} \longrightarrow$ for computing the generalization error
- $\mathcal{D} \cap \mathcal{T} = \emptyset$

Model Selection

- **misclassification rate**

$$\mathrm{err}(f, \mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}(f(\mathbf{x}_i) \neq y_i)$$

- select K so as to have the **minimum misclassification rate**



N.B.: on the left (small K) **overfitting**, on the right (large K) **underfitting**

Model Selection

- unfortunately we have not access to the test set (future data) to pick the model of the right complexity K
- we can create a "test set" by partitioning the available training set $\mathcal{D}$ in two parts:
  1. the part actually used for training the model $\tilde{\mathcal{D}}$
  2. the part used for selecting the model complexity, the **validation set** $\mathcal{V}$
- then we have a partition $\mathcal{D} = \tilde{\mathcal{D}} \cup \mathcal{V}$ with $\tilde{\mathcal{D}} \cap \mathcal{V} = \emptyset$

common procedure

- use 80% of the data for $\tilde{\mathcal{D}}$ and 20% for $\mathcal{V}$
- fit all the models M_i by using $\tilde{\mathcal{D}}$
- pick the **best model** M^* by evaluating all the M_i on $\mathcal{V}$ (find the model M^* with minimum misclassification error)
- fit the selected model M^* on the full training set $\mathcal{D}$ (now use full info)

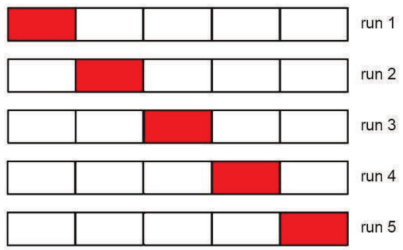
Model Selection

problem: if $N = |\mathcal{D}|$ is very small, we won't have enough data to train the model

cross validation

- split the data $\mathcal{D}$ in K equal folds $\{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_K\}$
- for each model M_i:
  for each $k \in \{1, 2, ..., K\}$ use $\tilde{\mathcal{D}}_k \triangleq \mathcal{D} \setminus \mathcal{D}_k$ to train model M_i and evaluate it on $\mathcal{V}_k \triangleq \mathcal{D}_k$ by computing the misclassification rate $err(M_i, \mathcal{V}_k)$
- for each model M_i: compute the average error $err(M_i) = \sum_{k=1}^{K} err(M_i, \mathcal{V}_k)$ and use it as an approx. for the test/generalization error of M_i
- select the best model $M^* = \text{argmin } err(M_i)$ and fit it on the full dataset $\mathcal{D}$

N.B.: in general $K = 5$, if $K = N$ we get a method called **leave-one out cross validation**



Outline

No Free Lunch Theorem

All models are wrong, but some models are useful – George Box

- machine learning is concerned with devising
  - different **models**
  - different **algorithms** to fit them
- there is no single best model that works optimally for all kinds of problems!
- why? assumptions limit our domain of application!
- we have to design **speed-accuracy-complexity** tradeoffs selecting a suitable model and an appropriate algorithm

Credits

- Kevin Murphy's book