# Introduction

Note: Unless otherwise noted all references including images are from the required textbook, Machine Learning: A Probabilistic Perspective by Kevin P. Murphy.

## Machine Learning

Machine Learning provides the computers with the ability to learn without being explicitly programmed.

*Arthur Samuel, 1959*

Machine learning is programming computers to optimize a performance criterion using example data or past experience.

*Ethem Alpaydin, 2014*

## Machine Learning

Learning: Improving with experience at some task

✓ Improve over task *T*

✓ with respect to performance measure *P*

✓ based on experience *E*

*Tom Mitchell, 1997*

## Why Learn?

Learning is used when:

- ✓ Human expertise does not exist (navigating on Mars)
- ✓ Humans are unable to explain their expertise (speech recognition)
- ✓ Solution changes in time (routing on a computer network)
- ✓ Solution needs to be adapted to particular cases (user biometrics)

*Ethem Alpaydin, 2014*

## Types of Machine Learning

- ✓ Supervised (Predictive) Learning
  - o Classification
  - o Regression
- ✓ Unsupervised (Descriptive) Learning
- ✓ Reinforcement Learning*

* Beyond the scope of this course

## Supervised Learning

The goal is to learn a mapping from inputs $x$ to outputs $y$, given a labeled set of input-output pairs $\mathcal{D} = \{(\mathrm{x}_i, y_i)\}_{i=1}^{N}$, where

$\mathcal{D}$ is called the training set and

$N$ is the number of training examples

## Supervised Learning

In the simplest setting, each training input $\mathrm{x}_i$ is a D-dimensional vector of numbers, representing say, the height and weight of a person. These are called **features**, **attributes**, or **covariates**. They are often stored in an $N \times D$ **design matrix** $\boldsymbol{X}$. In general, $\mathrm{x}_i$ could be complex structured object, such as an image, a sentence, an email message, a time series, a molecular shape, a graph, etc.

## Supervised Learning

Similarly the form of the output or **response variable** can in principle be anything, but most methods assume that $y_i$ is a **categorical** or **nominal** variable from some finite set, $y_i \in \{1, \ldots, C\}$ (such as male or female), or that $y_i$ is a real-valued scalar (such as income level).

## Supervised Learning

When $y_i$ is categorical, the problem is known as **classification** or **pattern recognition**.

When $y_i$ is real-valued, the problem is known as **regression**.

Another variant known as **ordinal regression**, occurs where the label space has some natural ordering, such as grades A-F.

## Classification

The goal is to learn a mapping from inputs $x$ to outputs $y$, where $y \in \{1, \ldots, C\}$, with $C$ being the number of classes.

If $C = 2$, this is called **binary classification** (in which case we often assume $y \in \{0, 1\}$).

If $C > 2$, this is called **multiclass classification**.

## Classification

If the class labels are not mutually exclusive (e.g., somebody may be classified as tall and strong), we call it **multi-label classification**, but this is best viewed as predicting multiple related binary class labels (a so-called **multiple output model**). When we use the term “classification”, we will mean multiclass classification with a single output, unless we state otherwise.

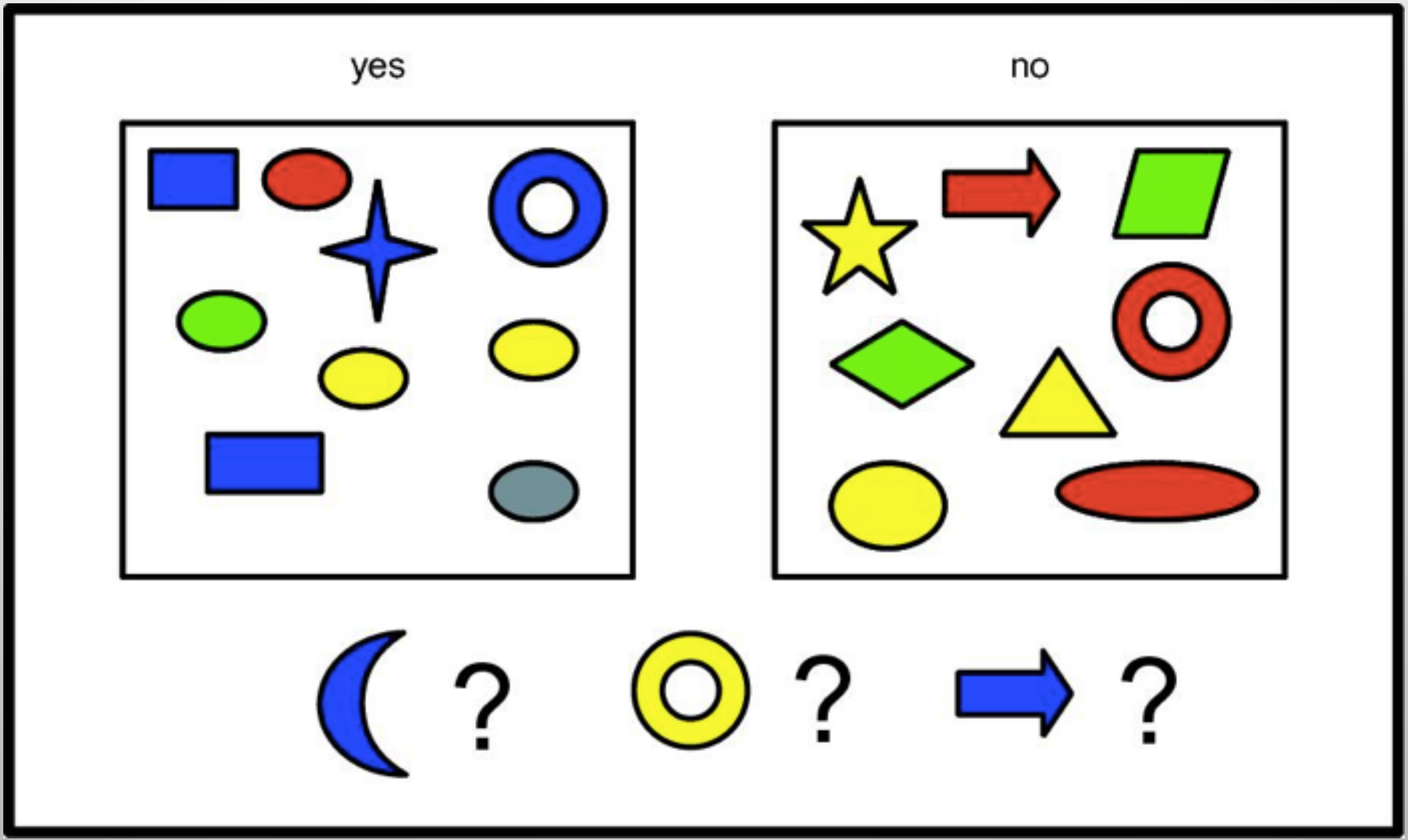
## Classification

One way to formalize the problem is as **function approximation**. We assume $y = f(\mathrm{x})$ for some unknown function $f$, and the goal of learning is to estimate the function f given a labeled training set, and then to make predictions using $\hat{y} = \hat{f}(\mathrm{x})$. (We use the hat symbol to denote an estimate.)

## Classification

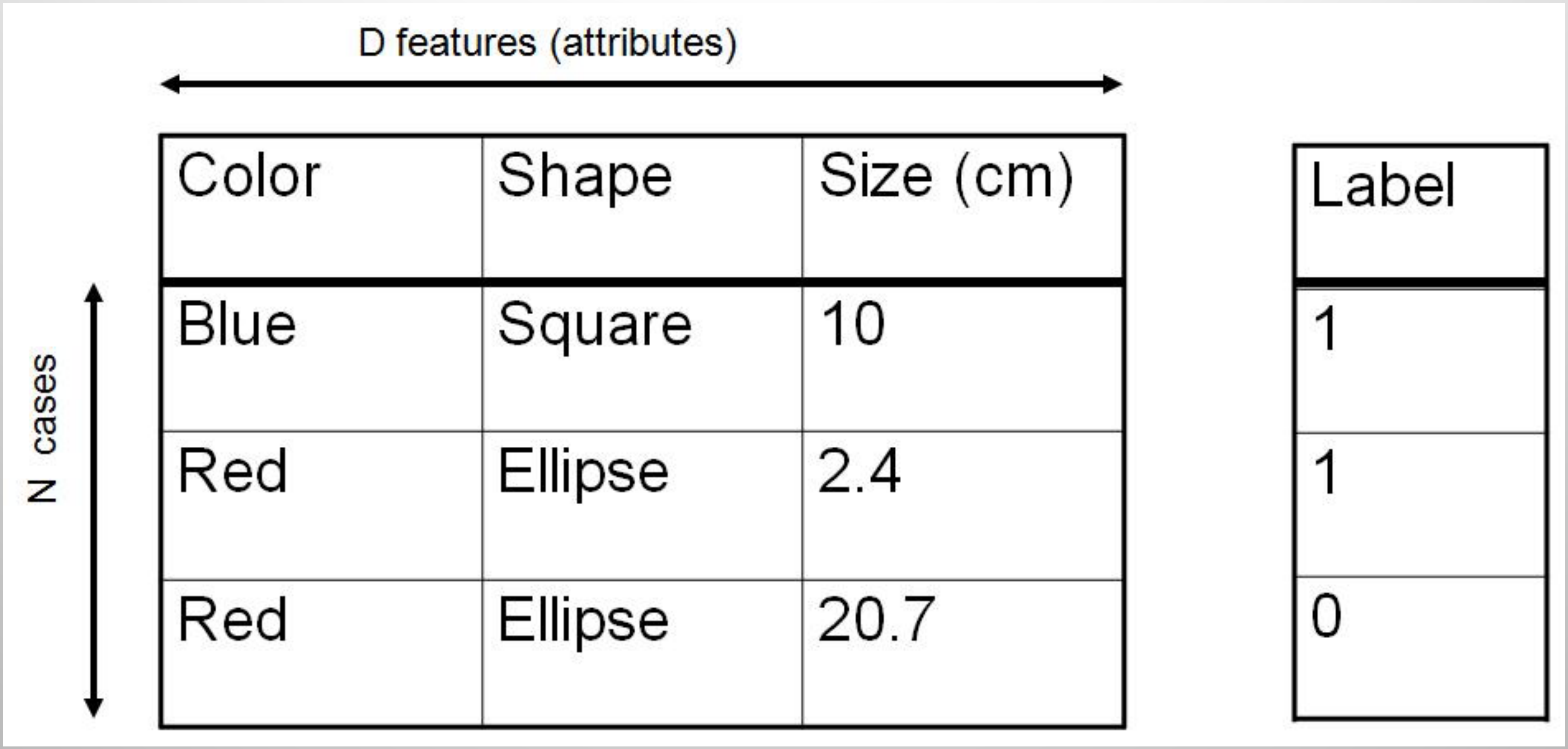
The main goal is to make predictions on novel inputs, meaning ones that were not seen before (this is called **generalization**), since predicting the response on the training set is easy.

# Classification



Some labeled training examples of colored shapes, along with 3 unlabed test cases

## Supervised Learning


D features (attributes)
N cases

| Color | Shape | Size (cm) |
|---|---|---|
| Blue | Square | 10 |
| Red | Ellipse | 2.4 |
| Red | Ellipse | 20.7 |

| Label |
|---|
| 1 |
| 1 |
| 0 |

Representing the training data as an *N x D* design matrix. Row *i* represents the feature vector $x_i$. The last column is the label, $y_i \in \{0,1\}$.

## Classification

To handle ambiguous cases, such as the yellow circle above, it is desirable to return a probability.

The probability distribution over possible labels, given the input vector $\mathrm{x}$ and training set $\mathcal{D}$ will be denoted by $p(y|\mathrm{x}, \mathcal{D})$. In general, this represents a vector of length $C$. (If there are just two classes, it is sufficient to return $p(y \ = 1|\mathrm{x}, \mathcal{D})$, since $p(y = 1|\mathrm{x}, \mathcal{D}) \ + \ p(y = 0|\mathrm{x}, \mathcal{D}) \ = \ 1$

## Classification

Given a probabilistic output, the “best guess” as to the “true label” can be computed using

$$\hat{y} = \hat{f}(\mathbf{x}) = \operatorname*{argmax}_{c=1}^{C} p(y = c|\mathbf{x}, \mathcal{D})$$

This corresponds to the most probable class label, and is called the **mode** of the distribution $p\ (y|\mathrm{x}, \mathcal{D})$; it is also known as a **maximum a posteriori (MAP) estimate.**

## Classification

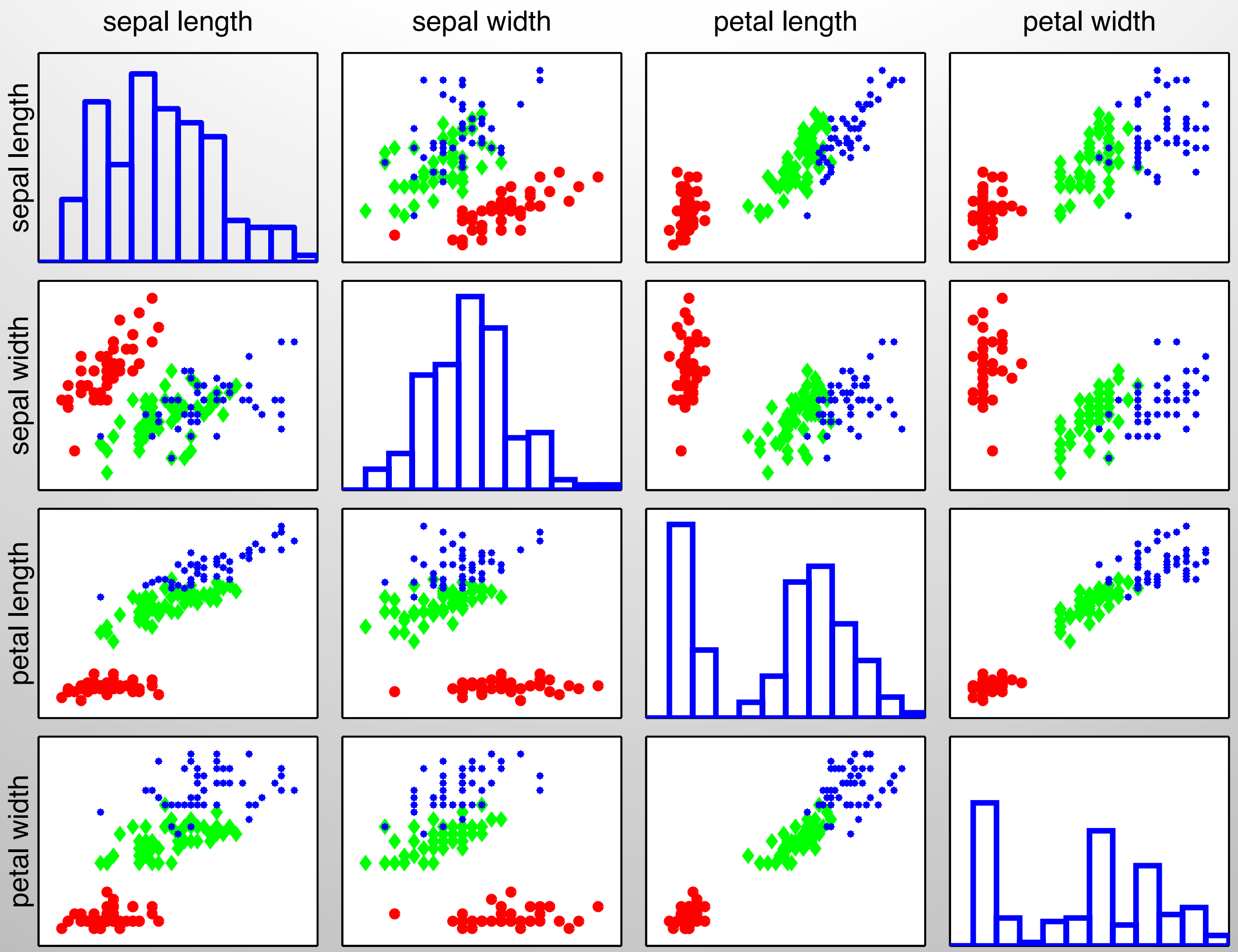
Real-world applications

- ✓ Document classification and email spam filtering
- ✓ *Classifying flowers*
- ✓ *Image classification and handwriting recognition*
- ✓ *Face Detection and Recognition*

## Classification

Three types of iris flowers: setosa, versicolor, and virginica.
Source: http://www.statlab.uni-heidelberg.de/data/iris/

# Classification



Visualization of the Iris data as a pairwise scatter plot.

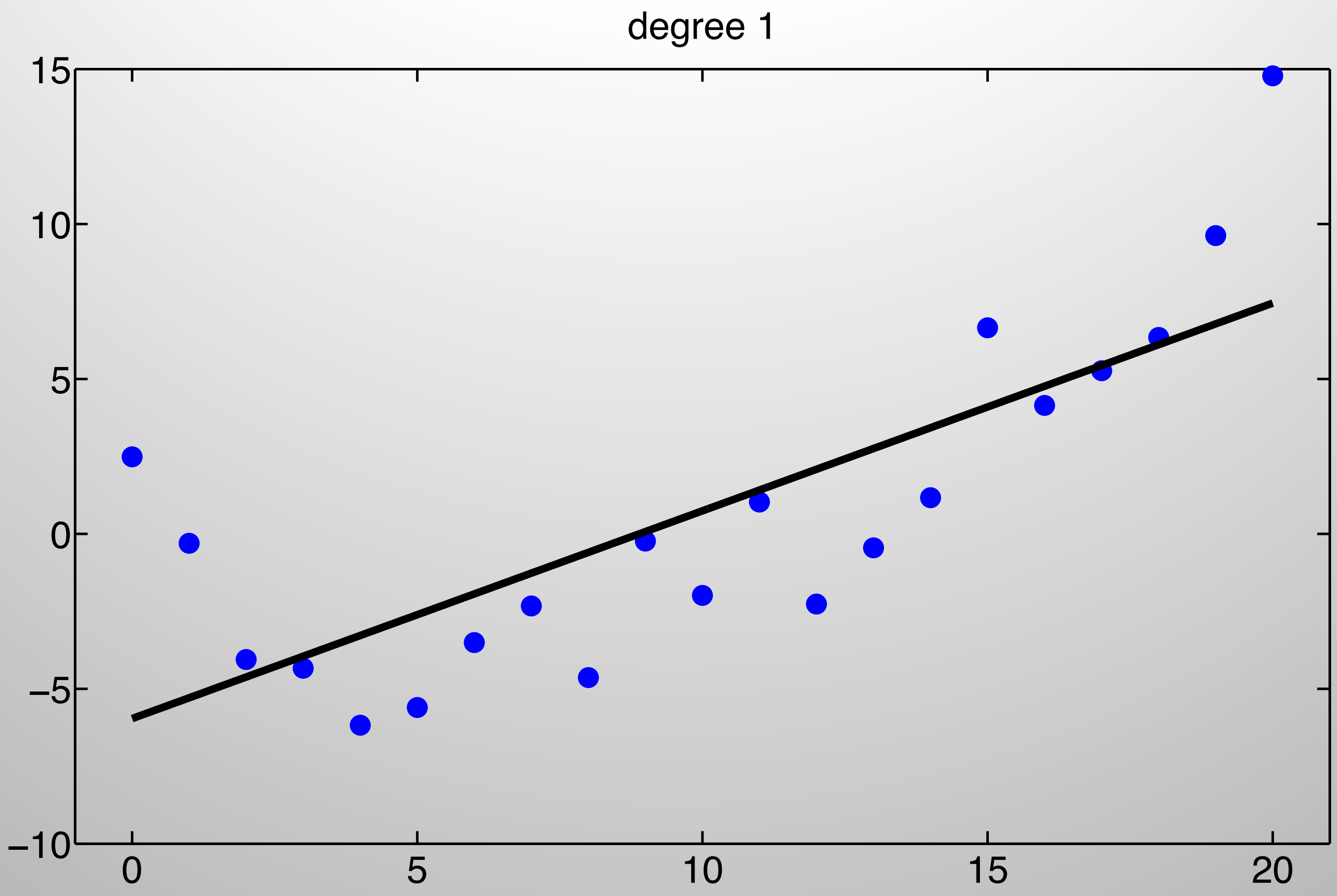
## Regression

Regression is just like classification except the response variable is continuous.

The following example has a single real-valued input $\mathrm{x}_i \in \mathbb{R}$, and a single real-valued response $y_i \in \mathbb{R}$.

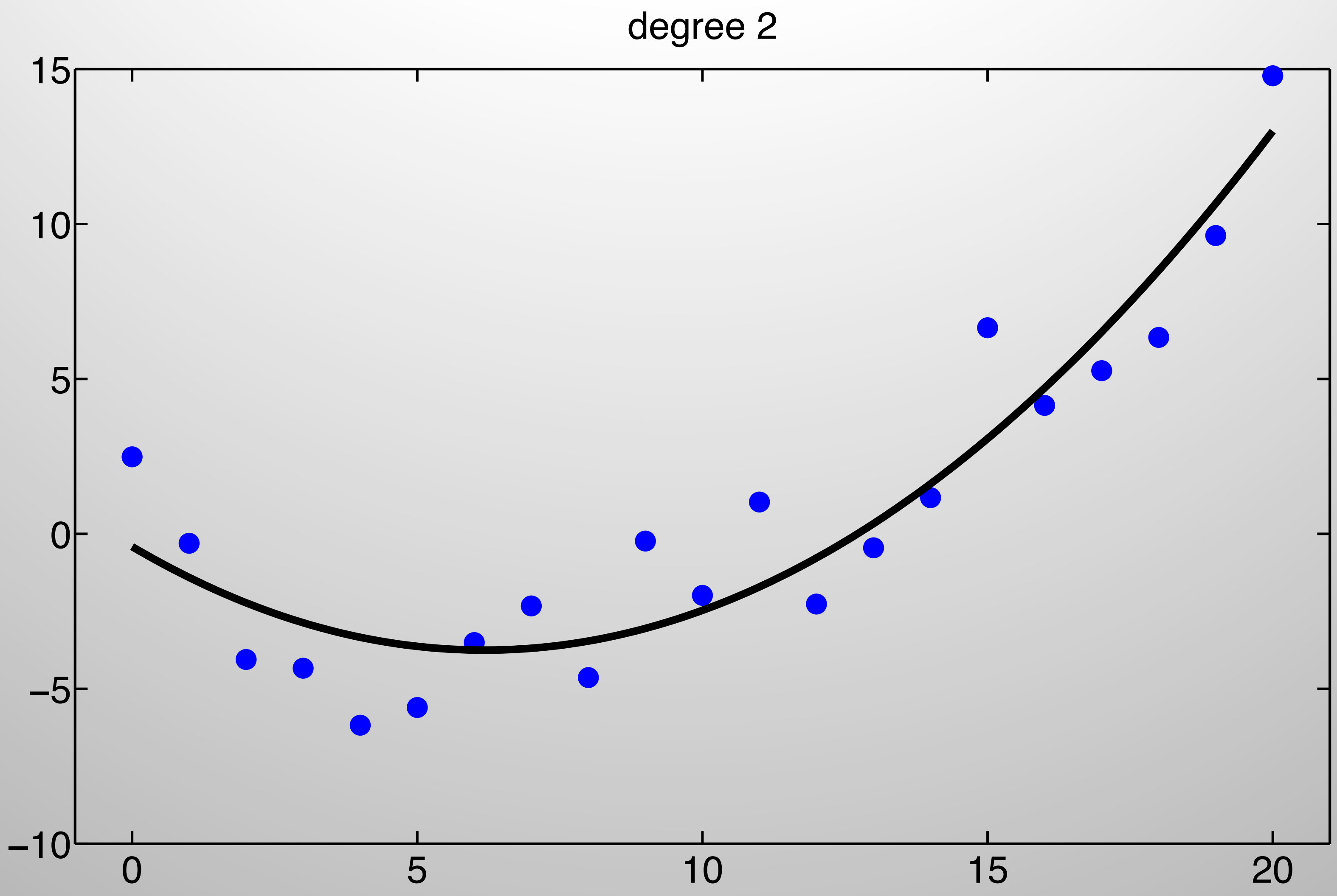
Two different models are fit to the data: a straight line and a quadratic function.

## Regression



Linear regression on some 1-dimensional data

## Regression



Same data as the previous slide with polynomial regression

## Regression

Real-world applications

- ✓ Predict tomorrow’s stock market price given current market conditions and other possible side information.
- ✓ Predict the age of a viewer watching a given video on YouTube.
- ✓ Predict the temperature at any location inside a building using weather data, time, door sensors, etc.

# Unsupervised Learning

- Given only inputs $\mathcal{D} = \{\mathrm{x}_i\}_{i=1}^{N}$, the goal is to discover "interesting structure" in the data, which is sometimes called **knowledge discovery**. Unlike supervised learning, the desired output for each input is not given. Instead, the task is formalized as one of **density estimation**, that is, models of the form $p(\mathrm{x}_i|\theta)$.

## Unsupervised Learning

There are two differences from the supervised case.

- ✓ First, the probability distribution is $p(\mathrm{x}_i|\theta)$ instead of $p(y_i|x_i, \theta)$; that is, supervised learning is conditional density estimation, whereas unsupervised learning is unconditional density estimation.

## Unsupervised Learning

- ✓ Second, $\mathrm{x}_i$ is a vector of features, so we need to create multivariate probability models. By contrast, in supervised learning, $y_i$ is usually just a single variable that we are trying to predict.

## Unsupervised Learning

Some canonical examples of unsupervised learning are:

- ✓Discovering clusters
- ✓Discovering latent factors
- ✓Discovering graph structures
- ✓Matrix completion

## Clustering

Let $K$ denote the number of clusters. The first goal is to estimate the distribution over the number of clusters, $p(K|\mathcal{D})$; this tells us if there are subpopulations within the data.
For simplicity, we often approximate the distribution $p(K|\mathcal{D})$ by its mode,
$K^* = \operatorname{argmax}_K p(K|\mathcal{D})$.

## Clustering

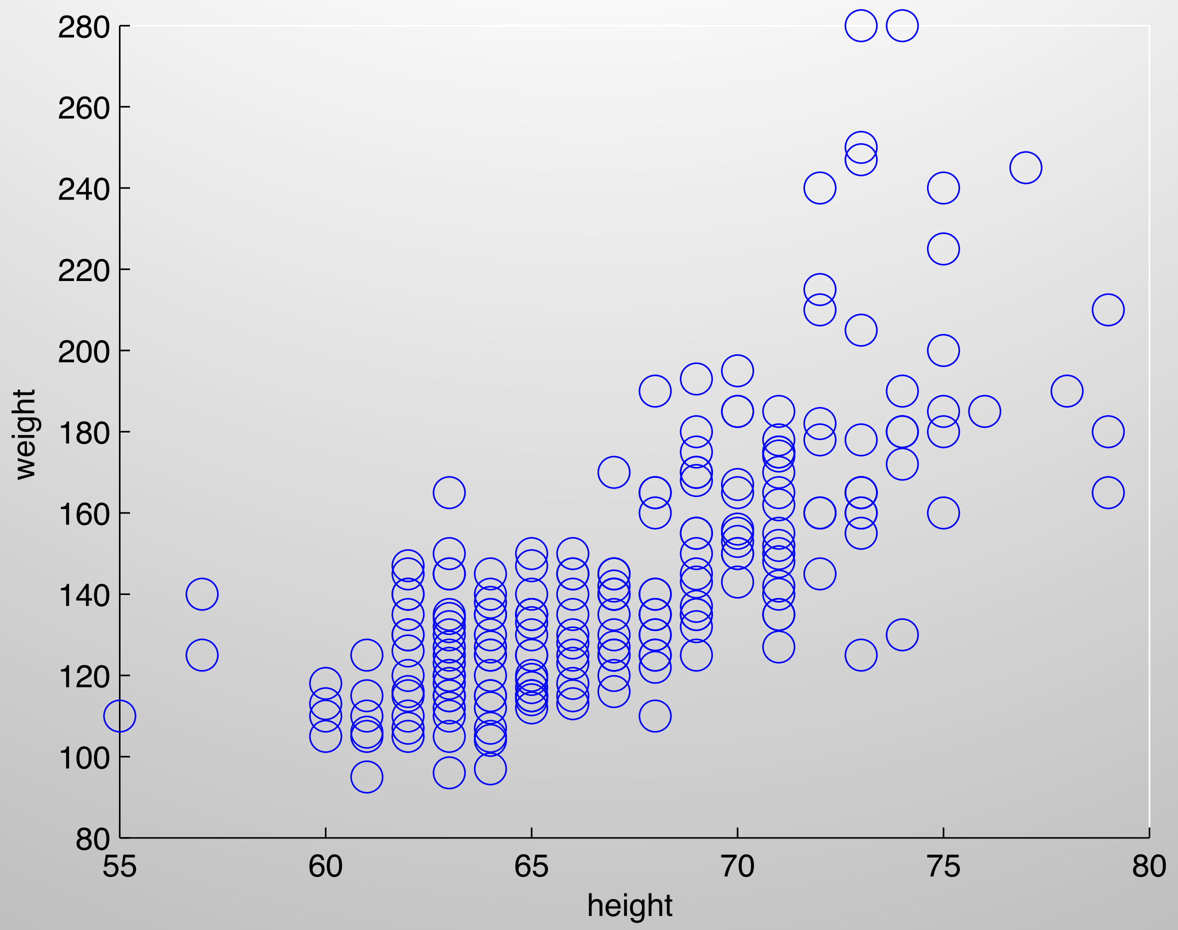
In the supervised case, we are told how many classes there are, but in the unsupervised case, we are free to choose as many or few clusters as we like. Picking a model of the “right” complexity is called **model selection**.

## Clustering

The second goal is to estimate which cluster each point belongs to. Let $z_i \in \{1, \ldots, K\}$ represent the cluster to which data point $i$ is assigned. ($z_i$ is an example of a **hidden** or **latent** variable, since it is never observed in the training set.) We can infer which cluster each data point belongs to by computing.

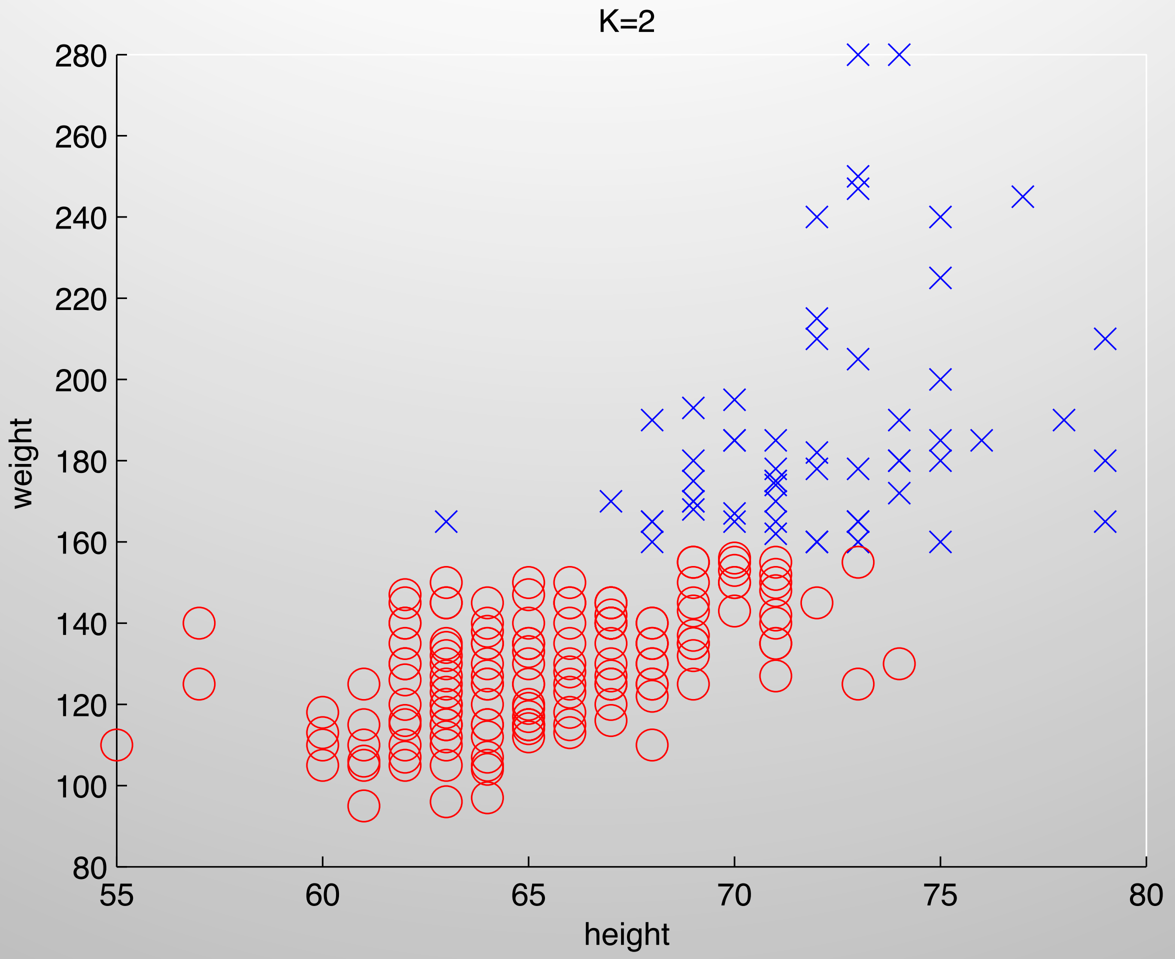
$$z_i^* = \operatorname{argmax}_k p\,(z_i = k | \mathrm{x}_i, \mathcal{D})$$

# Clustering



The height and weight of some people

# Clustering



A possible clustering using K = 2 clusters

## Clustering

Real-world applications:

- ✓ In astronomy, the autoclass system (Cheeseman et al. 1988) discovered a new type of star, based on clustering astrophysical measurements.
- ✓ In e-commerce, it is common to cluster users into groups, based on their purchasing or web-surfing behavior, and then to send customized targeted advertising to each group (see e.g., (Berkhin 2006)).
- ✓ In biology, it is common to cluster flow-cytometry data into groups, to discover different sub-populations of cells (see e.g., (Lo et al. 2009)).

## Discovering Latent Factors

When dealing with high dimensional data, it is often useful to reduce the dimensionality by projecting the data to a lower dimensional subspace which captures the “essence” of the data. This is called **dimensionality reduction**.

## Discovering Latent Factors

The motivation behind this technique is that although the data may appear high dimensional, there may only be a small number of degrees of variability, corresponding to latent factors . For example, when modeling the appearance of face images, there may only be a few underlying **latent factors** which describe most of the variability, such as lighting, pose, identity, etc.

## Discovering Latent Factors

When used as input to other statistical models, such low dimensional representations often result in better predictive accuracy, because they focus on the “essence” of the object, filtering out inessential features.
The most common approach to dimensionality reduction is called **principal components analysis (PCA)**.

## Discovering Graph Structures

Sometimes we measure a set of correlated variables, and we would like to discover which ones are most correlated with which others. This can be represented by a graph $G$, in which nodes represent variables, and edges represent direct dependence between variables. We can then learn this graph structure from data, i.e., we compute $\hat{G} = \operatorname{argmax} p(G|\mathcal{D})$.

## Matrix Completion

Sometimes we have missing data, that is, variables whose values are unknown. For example, we might have conducted a survey, and some people might not have answered certain questions. Or we might have various sensors, some of which fail.

## Matrix Completion

The corresponding design matrix will then have "holes" in it; these missing entries are often represented by **NaN**, which stands for "not a number". The goal of **imputation** is to infer plausible values for the missing entries. This is sometimes called **matrix completion**.

## Matrix Completion

Real-world applications
- ✓ Image inpainting
- ✓ Collaborative filtering
- ✓ Market basket analysis

## Parametric vs. Non-Parametric Models

Does the model have a fixed number of parameters, or does the number of parameters grow with the amount of training data?
The former is called a **parametric model**, and the latter is called a **nonparametric model**.

## Parametric vs. Non-Parametric Models

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.

Nonparametric models are more flexible, but often computationally intractable for large datasets.

## Parametric vs. Non-Parametric Models

Non-parametric model example:

✓ K-nearest neighbors

Parametric model example:

✓ Linear regression

✓ Logistic regression

## *K*-nearest Neighbors (KNN)

KNN classifier simply looks at the *K* points in the training set that are nearest to the test input x, counts how many members of each class are in this set, and returns that empirical fraction as the estimate.

## *K*-nearest Neighbors (KNN)

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

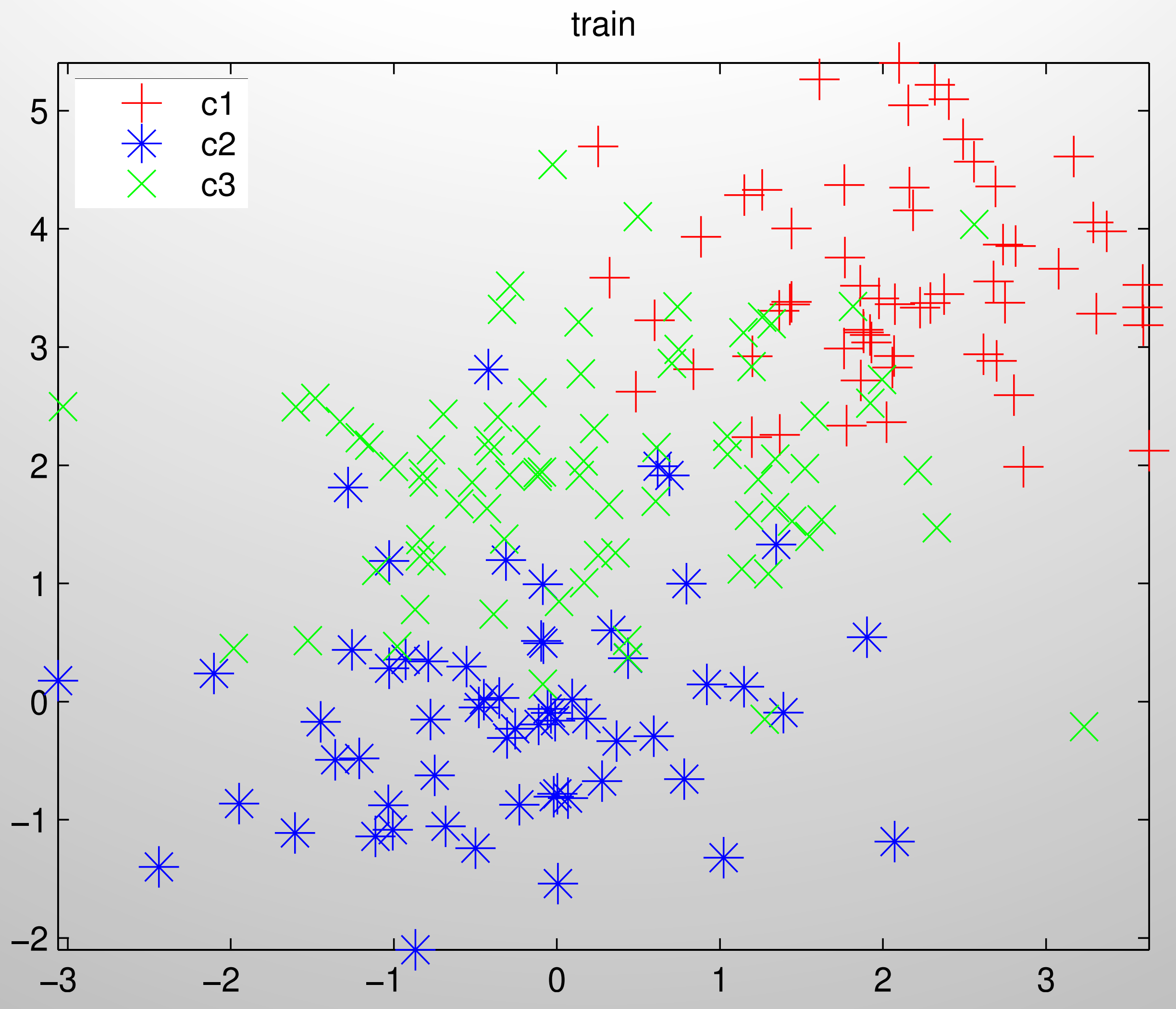
where $N_K(\mathrm{x}, \mathcal{D})$ are the (indices of the) K nearest points to x in $\mathcal{D}$ and $\mathbb{I}(e)$ is the **indicator function** defined as

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

## *K*-nearest Neighbors (KNN)

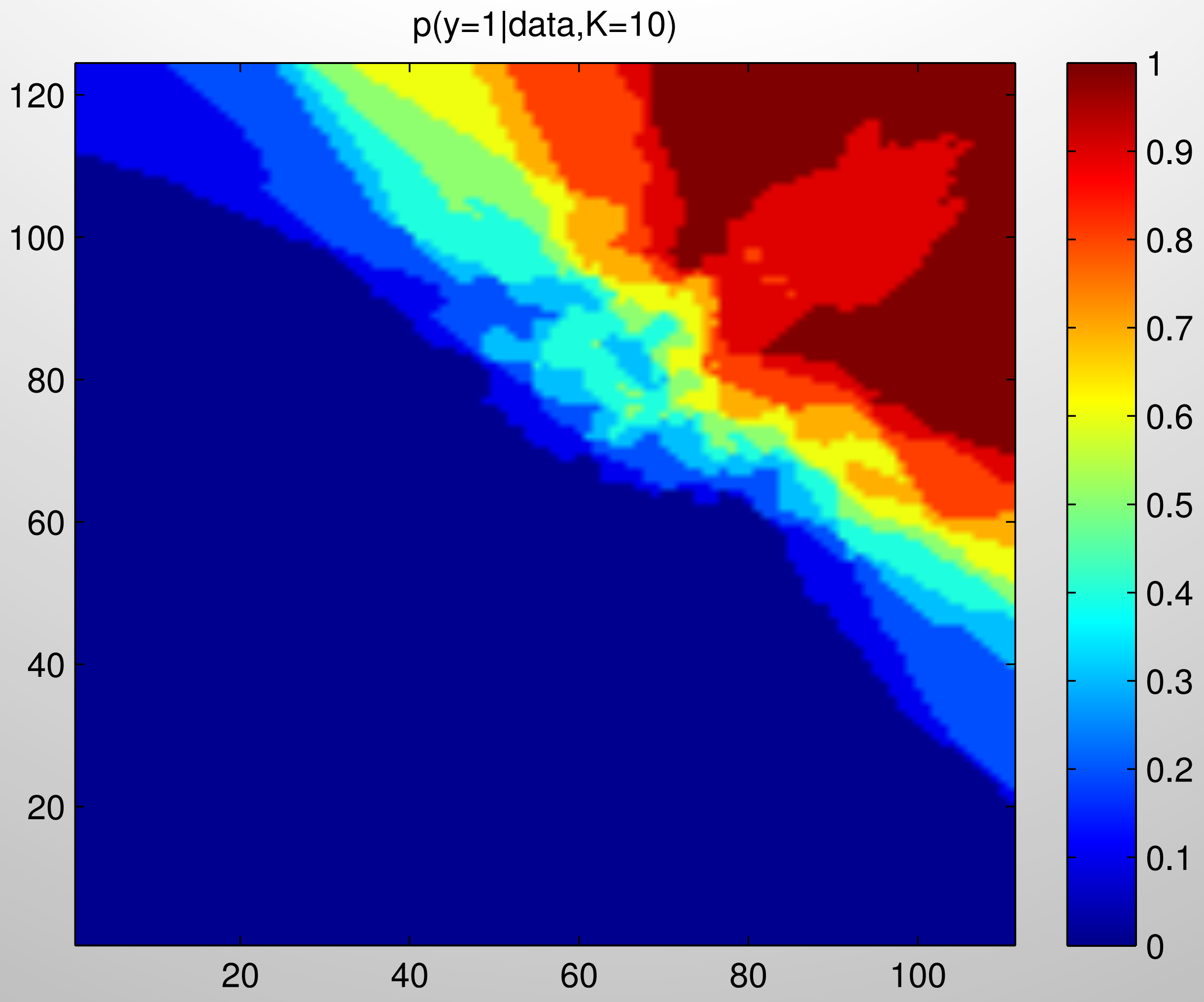
This method is an example of **memory-based learning** or **instance-based learning**. It can be derived from a probabilistic framework. The most common distance metric to use is Euclidian distance (which limits the applicability of the technique to data which is real-valued), although other metrics can be used.

# *K*-nearest Neighbors (KNN)



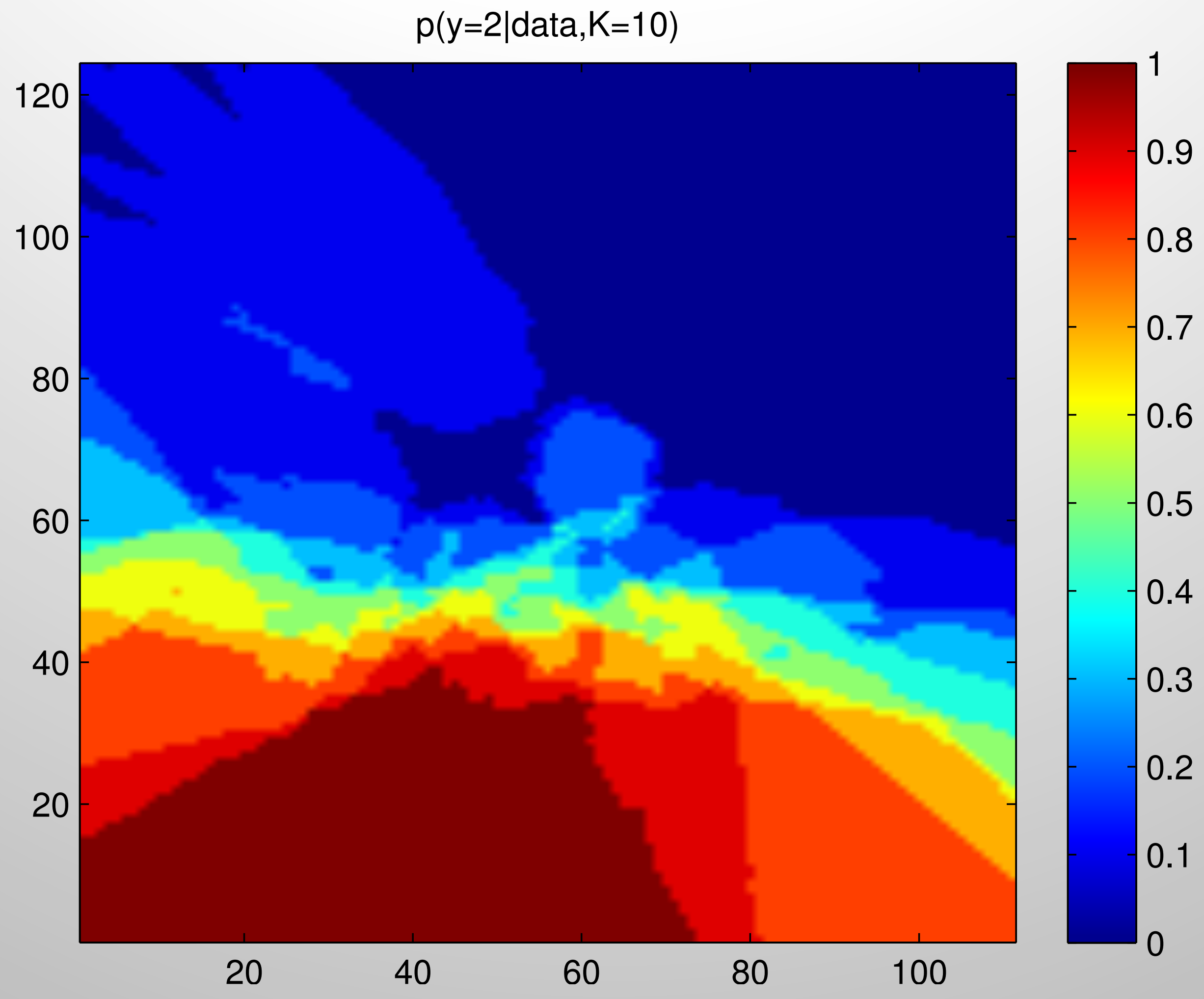
Some synthetic 3-class data in 2d

# *K*-nearest Neighbors (KNN)



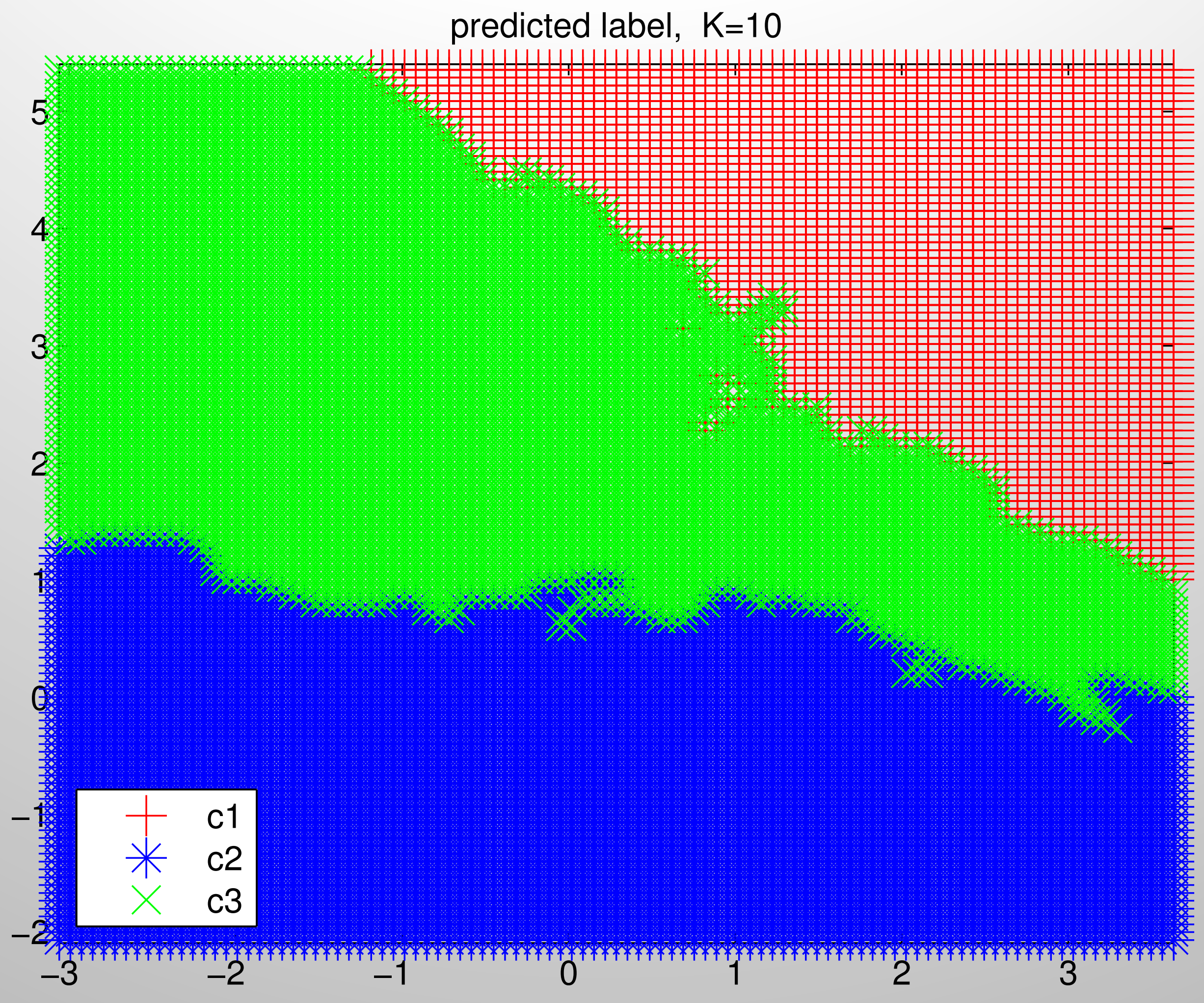
Probability of class 1 for KNN with K = 10

# *K*-nearest Neighbors (KNN)



Probability of class 2 for KNN with K = 10

# *K*-nearest Neighbors (KNN)



MAP estimate of class label

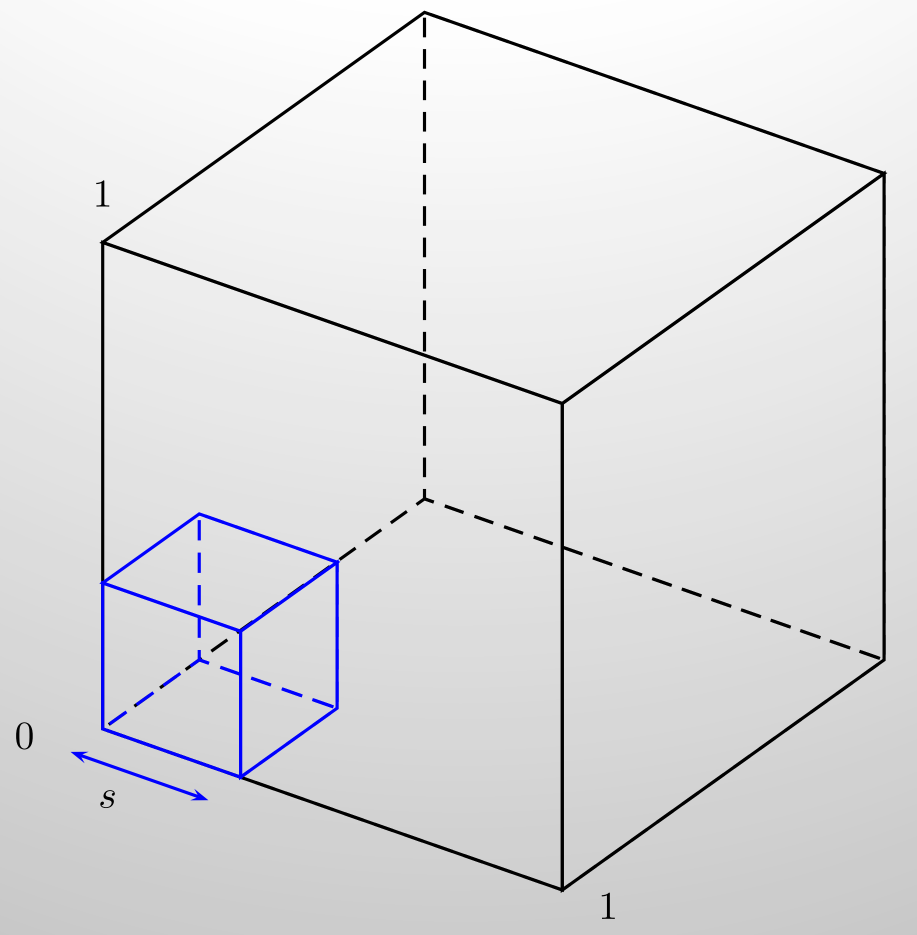
## Curse of Dimensionality

The KNN classifier is simple and can work quite well, provided it is given a good distance metric and has enough labeled data. However, the main problem with KNN classifiers is that they do not work well with high dimensional inputs. The poor performance in high dimensional settings is due to the **curse of dimensionality**.

# Curse of Dimensionality

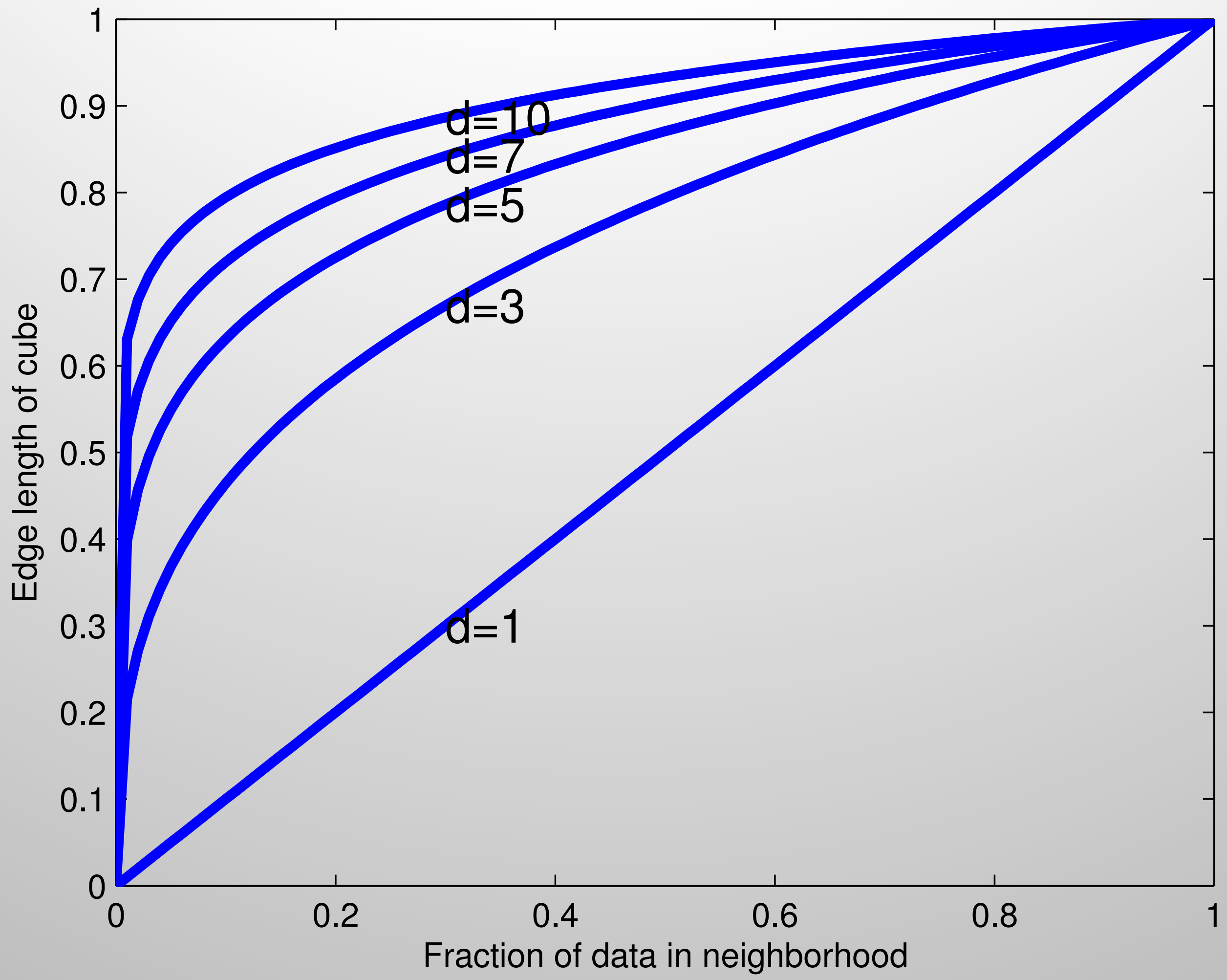
Consider applying a KNN classifier to data where the inputs are uniformly distributed in the *D*-dimensional unit cube. Suppose we estimate the density of class labels around a test point x by growing a hyper-cube around x until it contains a desired fraction *f* of the data points, The expected edge length of this cube will be $e_D(f) = f^{1/D}$.

# Curse of Dimensionality



We embed a small cube of side *s* inside a larger unit cube

## Curse of Dimensionality



Edge length of a cube needed to cover a given volume of the unit cube as a function of the number of dimensions

## Parametric Models

The main way to combat the curse of dimensionality is to make some assumptions about the nature of the data distribution. These assumptions, known as **inductive bias**, are often embodied on the form of a **parametric model**, which is a statistical model with a fixed number of parameters.

## Overfitting

When we fit highly flexible models, we need to be careful that we do not **overfit** the data, that is, we should avoid trying to model every minor variation in the input, since this is more likely to be noise than true signal.

## Model Selection

When we have a variety of models of different complexity (e.g. linear or logistic regression models with different degree polynomials, or KNN classifiers with different values of *K*), how should we pick the right one?

A natural approach is to compute the **misclassification rate** on the training set for each method. <u>**What is the problem with this?**</u>

## Model Selection

When we are building a model, what we care about is generalization. This can be approximated by computing the misclassification rate on a large independent *test set*, not used model training.

Unfortunately, when training the model, (by assumption) we don’t have access to the test set, so we cannot use the test set to pick the model of the right complexity.

## Model Selection

However, we can create a test set by partitioning the training set into two: the part used for training the model, and a second part, called the **validation set**, used for selecting the model complexity. We then fit all the models on the training set, and evaluate their performance on the validation set, and pick the best. Once we picked the best, we can refit it to all the available data.

## No Free Lunch Theorem

Most of machine learning is concerned with devising different models, and different algorithms to fit them. We can use methods such as cross validation to empirically choose the best method for our particular problem. However, there is no single best model that works optimally for all kinds of problems, this is sometimes called the **no free lunch theorem** (Wolpert, 1996).