Applied Data Analysis (CS401)

ada

Lecture 7
Supervised learning
2 Nov 2022

EPFL

Robert West

dlab

Annowwwwwncements

- Happy (belated) Halloween!
- Homework H1 is being graded. Feedback to be released next week.
- Project milestone P2 due on Fri 18 Nov
- Friday’s lab session: two parallel tracks:
  - Track 1: exercise on supervised learning (in BCH 2201)
  - Track 2: project office hours (on Zoom)
    - Logistics: see Ed post
    - Do come and ask for feedback -- everyone will win!

Feedback

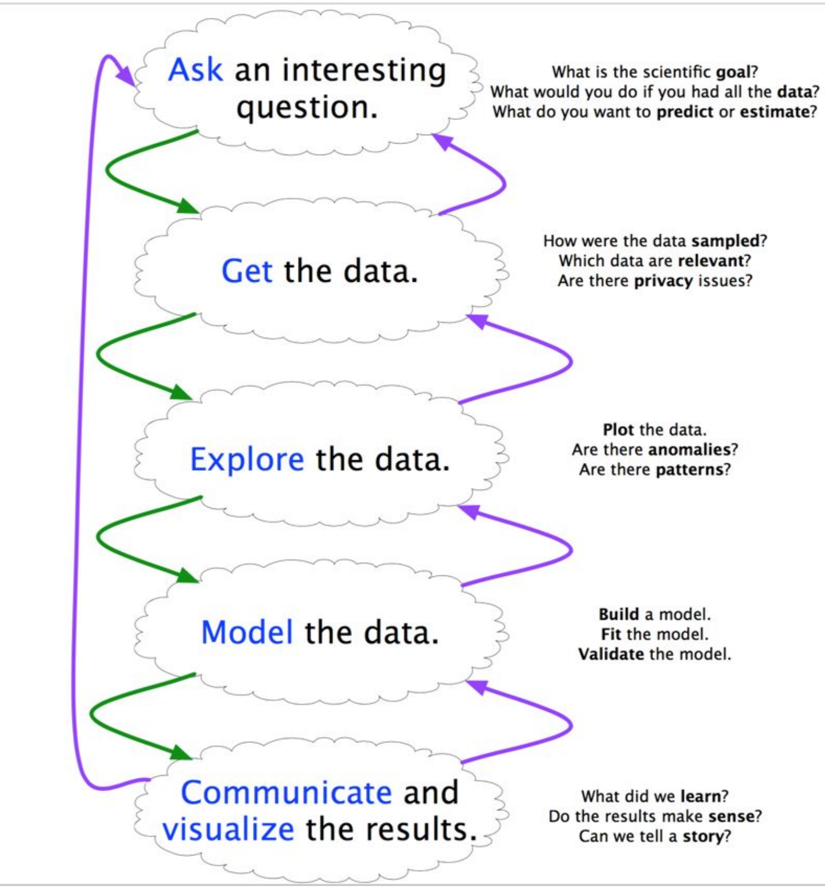
Give us feedback on this lecture here:
https://go.epfl.ch/ada2022-lec7-feedback

- What did you (not) like about this lecture?
- What was (not) well explained?
- On what would you like more (fewer) details?
- …

Why ML as part of data science?

- ML can facilitate most steps of the data analysis cycle

- But stay critical: “Can I trust my ML model?”



Ask an interesting question.

What is the scientific **goal**?
What would you do if you had all the **data**?
What do you want to **predict** or **estimate**?

Get the data.

How were the data **sampled**?
Which data are **relevant**?
Are there **privacy** issues?

Explore the data.

Plot the data.
Are there **anomalies**?
Are there **patterns**?

Model the data.

Build a model.
Fit the model.
Validate the model.

Communicate and visualize the results.

What did we **learn**?
Do the results make **sense**?
Can we tell a **story**?

Machine learning

- **Supervised:** We are given input/output pairs (X, y) (a.k.a. "samples") that are related via a function y = f(X). We would like to "learn" f, and evaluate it on new data. Types:
  - Discrete y (class labels): **"classification"**
  - Continuous y: **"regression"** (e.g., linear regression)
- **Unsupervised:** Given only samples X of the data, we compute a function f such that y = f(X) is a "simpler" representation.
  - Discrete y (cluster labels): **"clustering"**
  - Continuous y: **"dimensionality reduction"**

Machine learning: examples

- **Supervised (*lecture 7, i.e., today*):**
  - Is this image a cat, dog, car, house?
  - How would this user rate that restaurant?
  - Is this email spam?
  - Is this blob on a telescope image a supernova?
- **Unsupervised (*lecture 9*):**
  - Cluster handwritten digit data into 10 classes
  - What are the top 20 topics in Twitter right now?
  - Find the best 2D visualization of 1000-dimensional data

Machine learning: techniques

- **Supervised learning:**
  - k-NN (k nearest neighbors)
  - Tree-based models: decision trees, random forests
  - Linear + logistic regression
  - Naïve Bayes
  - Support vector machines
  - Supervised neural networks
  - etc.
- **Unsupervised learning:**
  - Clustering
  - Dimensionality reduction: topic modeling, matrix factorization (PCA, SVD, word2vec)
  - Hidden Markov models (HMM)
  - etc.

Today

(particularly in light of bias/variance tradeoff)

Lectures 9, 10, 11

Intro to supervised learning: k nearest neighbors (k-NN)

k nearest neighbors (k-NN)

Given a query item:

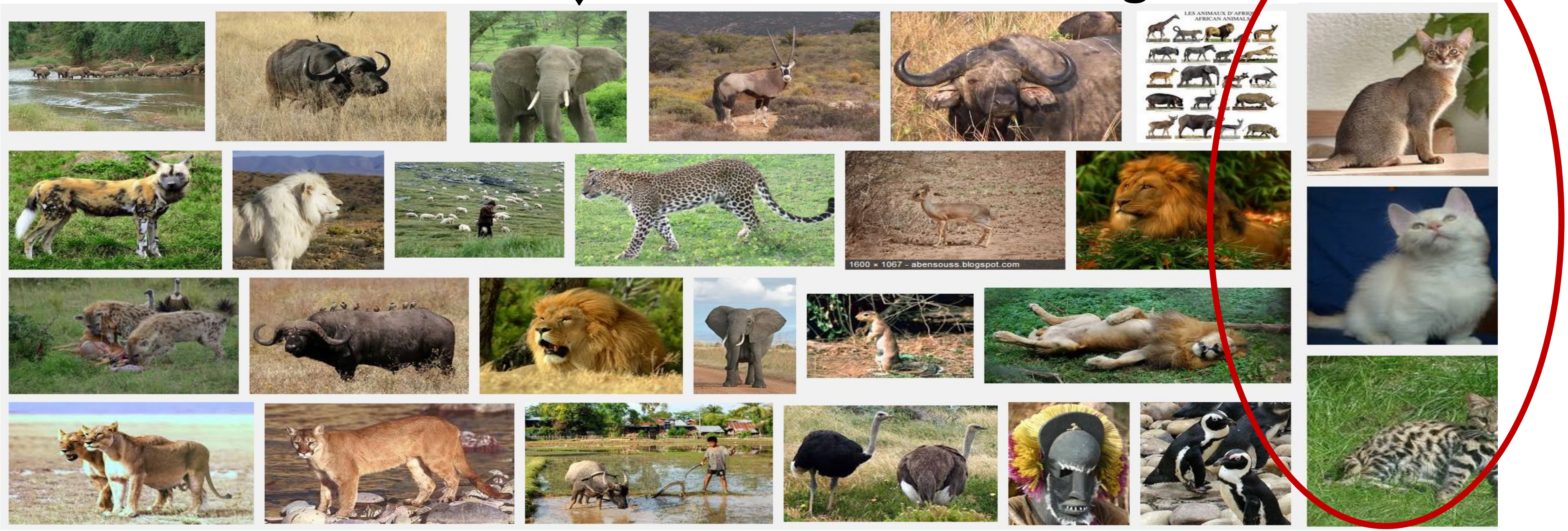
Find k closest matches in a labeled dataset ↓

k nearest neighbors (k-NN)

Given a query item:

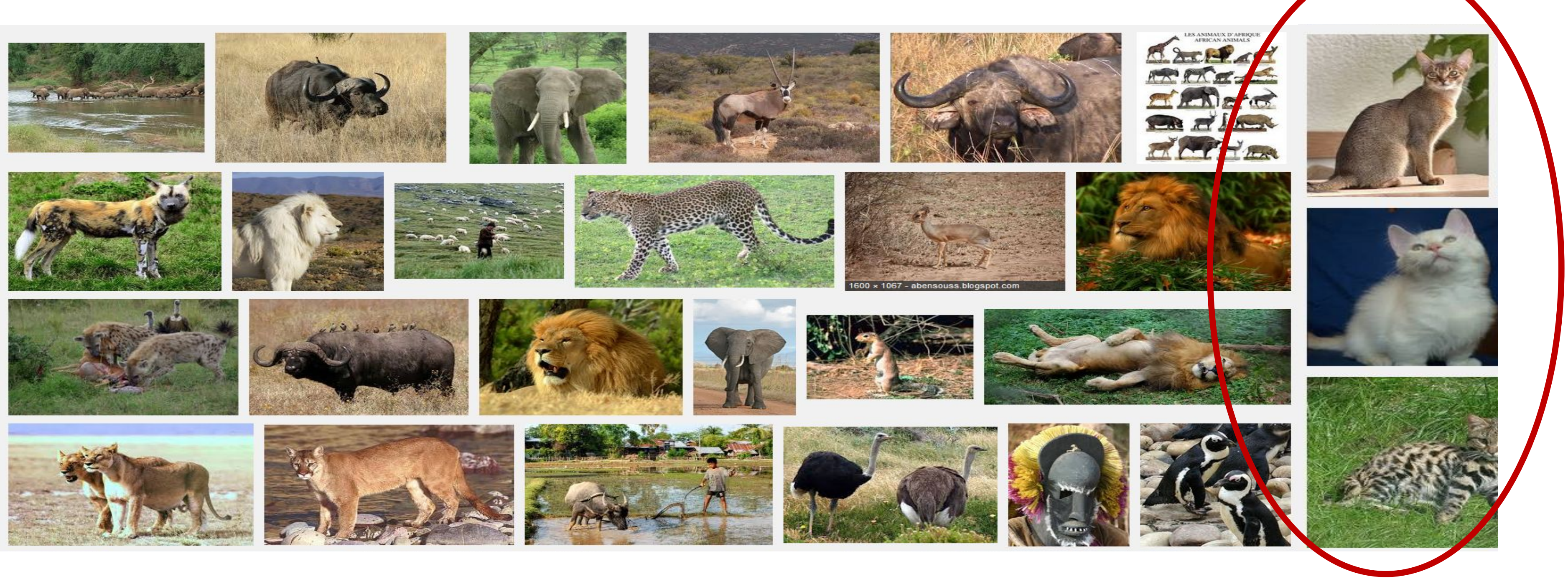
Return the most frequent label among the k

Find k closest matches in a labeled dataset ↓



k nearest neighbors (k-NN)

k = 3 votes for “cat”



Properties of k-NN

The data is the model

- No training needed.
- Conceptually simple algorithm.
- Accuracy generally improves with more data.
- Usually need data in memory, but can also be run from disk.

Minimal configuration:

- Only one parameter: k (number of neighbors)
- But two other choices are also important:
  - Similarity metric
  - Weighting of neighbors in voting (e.g. by similarity)

k-NN flavors

Classification:

- Model is y = f(X), y is from a discrete set (labels).
- Given X, compute y = majority vote of the k nearest neighbors.
- Can also use a weighted vote* of the neighbors.

Regression:

- Model is y = f(X), y is a real value.
- Given X, compute y = average value of the k nearest neighbors.
- Can also use a weighted average* of the neighbors.

* Weight function is usually the similarity (inverse distance).

k-NN distance (opposite of similarity) measures

- **Euclidean Distance:** Simplest, fast to compute

$$d(x, y) = \|x - y\|$$

- **Cosine Distance:** Good for documents, images, etc.

$$d(x, y) = 1 - \frac{x \cdot y}{\|x\|\|y\|}$$

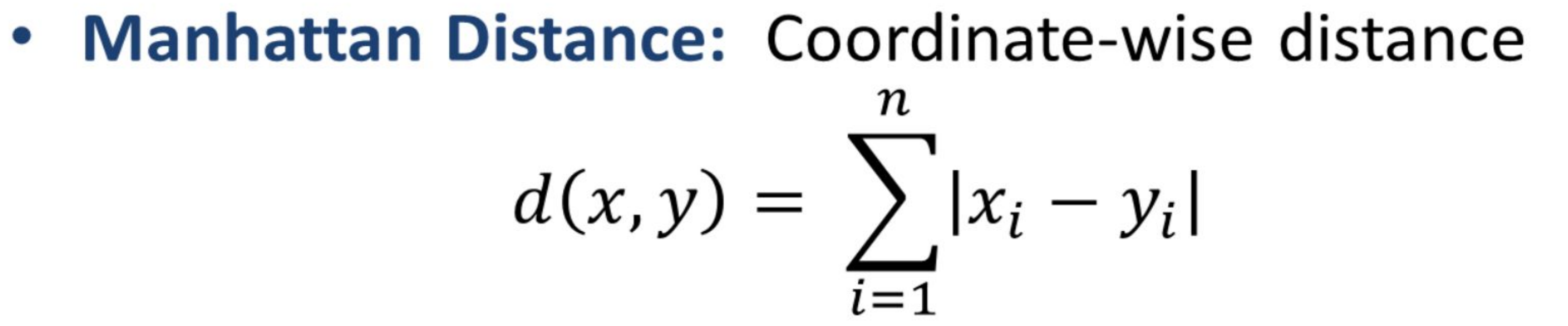
- **Jaccard Distance:** For set data:

$$d(X, Y) = 1 - \frac{|X \cap Y|}{|X \cup Y|}$$

- **Hamming Distance:** For string data:

$$d(x, y) = \sum_{i=1}^{n} (x_i \neq y_i)$$

k-NN distance (opposite of similarity) measures



- **Manhattan Distance:** Coordinate-wise distance

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

- **Edit Distance:** for strings, especially genetic data.

```
stack.push(kNN)
```

Predicting from samples

- Most datasets are **samples** from a (maybe infinite) **population**.
- We are most interested in **models of the population**, but we only have access to a **sample** (blue points) from the population.

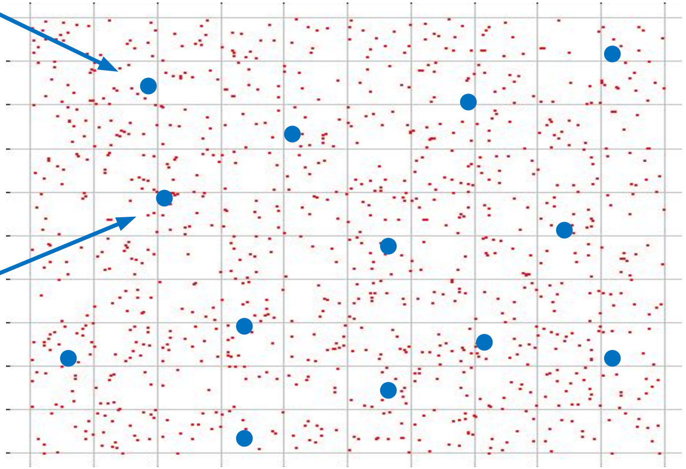
For a dataset consisting of pairs (X, y):

- features X, label y,

we aim to find the true model f:

- y = f(X).

We train on a training sample D and denote the fitted model as $f_D(X)$



Bias and variance

- Given a random training sample D, obtain model f_D
- For a new data point (X, y), prediction is $f_D(X)$
- (Squared) error = $E[(f_D(X) - y)^2]$ (E is expectation over D!)
- Fact: error can be decomposed into two parts (derivation)
  - $\text{Error}^2 = \text{Bias}^2 + \text{Variance}$
  - Bias = $E[f_D(X) - y]$
  - Variance = $E[(f_D(X) - E[f_D(X)])^2]$

Bias and variance

Our data-generated model $f_D(X)$ is a **statistical estimate** of the true function $f(X)$.

Because of this, its subject to bias and variance:

Bias: if we train models $f_D(X)$ on many training sets D, bias is the expected difference between their predictions and the true y's.

i.e. $$Bias = \mathrm{E}[f_D(X) - y]$$

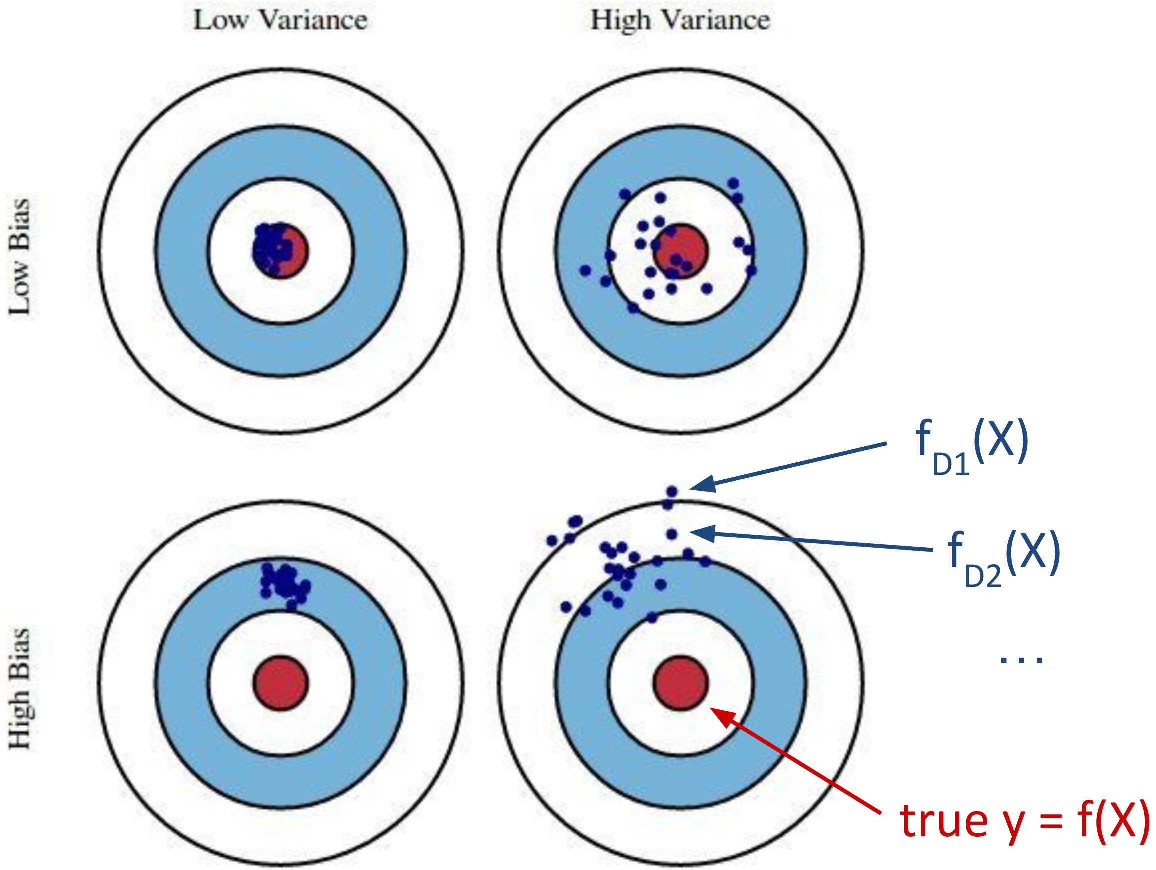
E[] is taken over points X and datasets D

Variance: if we train models $f_D(X)$ on many training sets D, variance is the variance of the estimates:

$$Variance = \mathrm{E}\left[\left(f_D(X) - \bar{f}(X)\right)^2\right]$$

Where $\bar{f}(X) = \mathrm{E}[f_D(X)]$ is the average prediction on X.

Consider a fixed testing point (X, y)



$f_{D1}(X)$

$f_{D2}(X)$

…

true y = f(X)

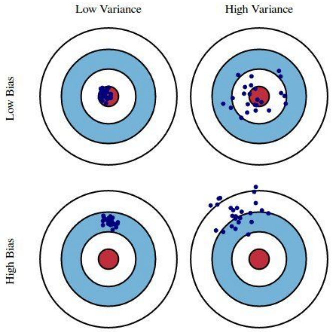
“Full” bias/variance: average this picture over all testing points (X, y)

Bias/variance tradeoff

Since Error^2 = Bias^2 + Variance, there is a tradeoff, usually modulated via model complexity:

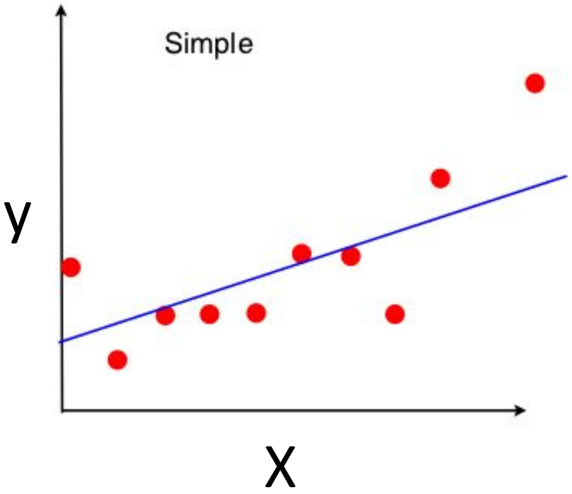
Complex models (many parameters) usually have lower bias, but higher variance.

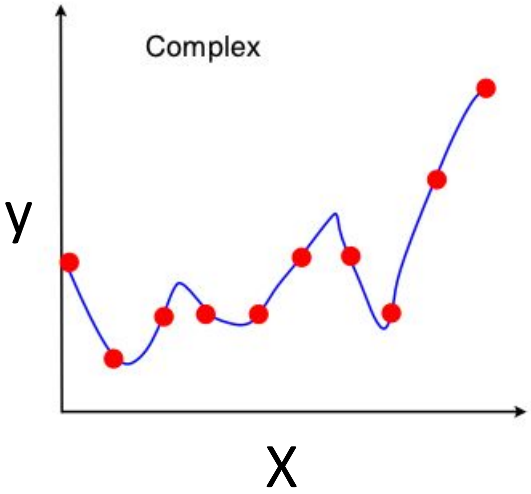
Simple models (few parameters) have higher bias, but lower variance.

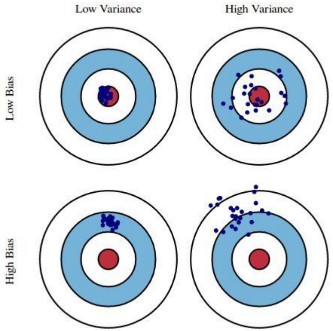


Bias/variance tradeoff

e.g. a linear model can only fit a straight line. A high-degree polynomial can fit a complex curve. But the polynomial will fit the individual training sample, rather than the full population. Its shape can vary from sample to sample, so it has high variance.







Bias/variance tradeoff

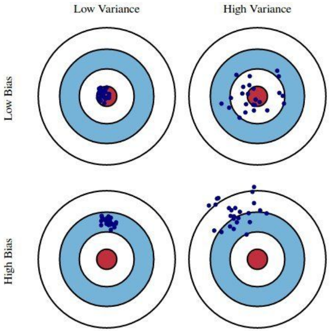
The total expected error is

$$Bias^2 + Variance$$

Because of the bias-variance trade-off, we want to **balance** these two contributions.

If $Variance$ strongly dominates, it means there is too much variation between models. This is called **over-fitting**.

If $Bias$ strongly dominates, then the models are not fitting the data well enough. This is called **under-fitting**.



```
kNN = stack.pop()
```

Choosing k for k nearest neighbors

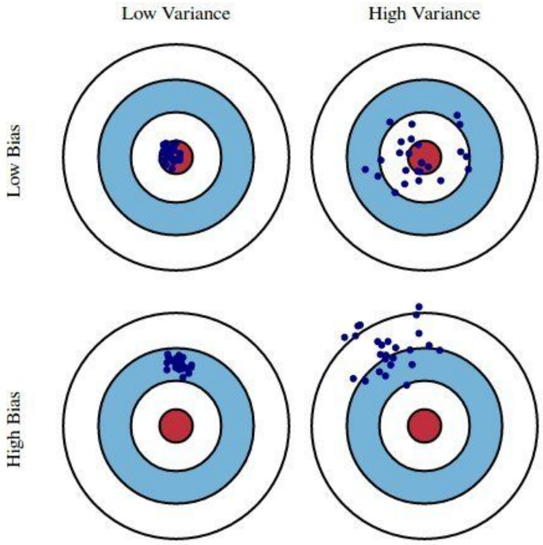
We have a bias/variance tradeoff:

- Small k → ?
- Large k → ?

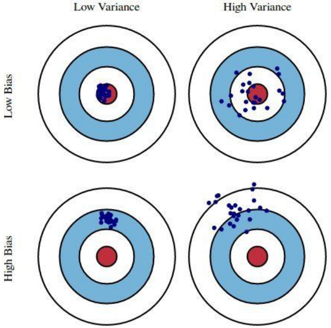
THINK FOR A MINUTE:

When *k* increases,
how do bias and variance change?

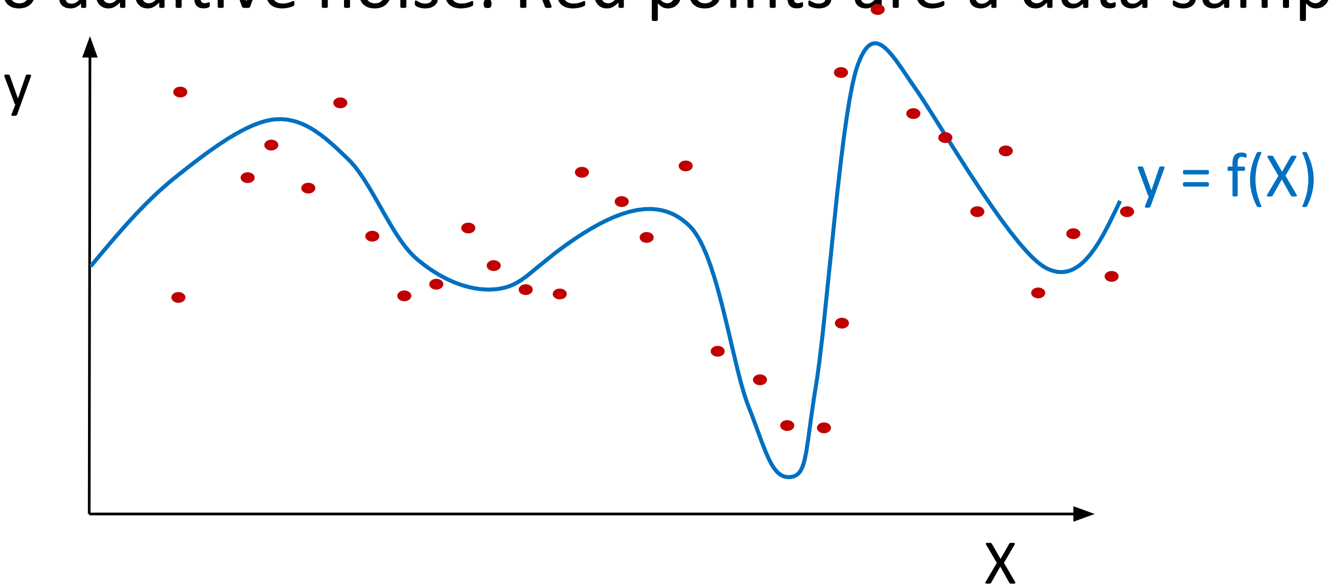
(Feel free to discuss with your neighbor.)



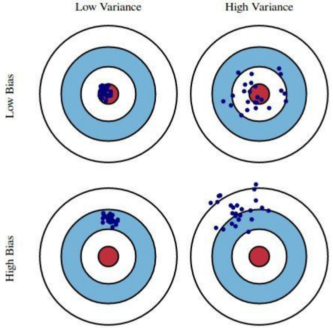
Choosing k



- Small k → low bias, high variance
- Large k → high bias, low variance
- Assume the real data follows the blue curve, with some mean-zero additive noise. Red points are a data sample.

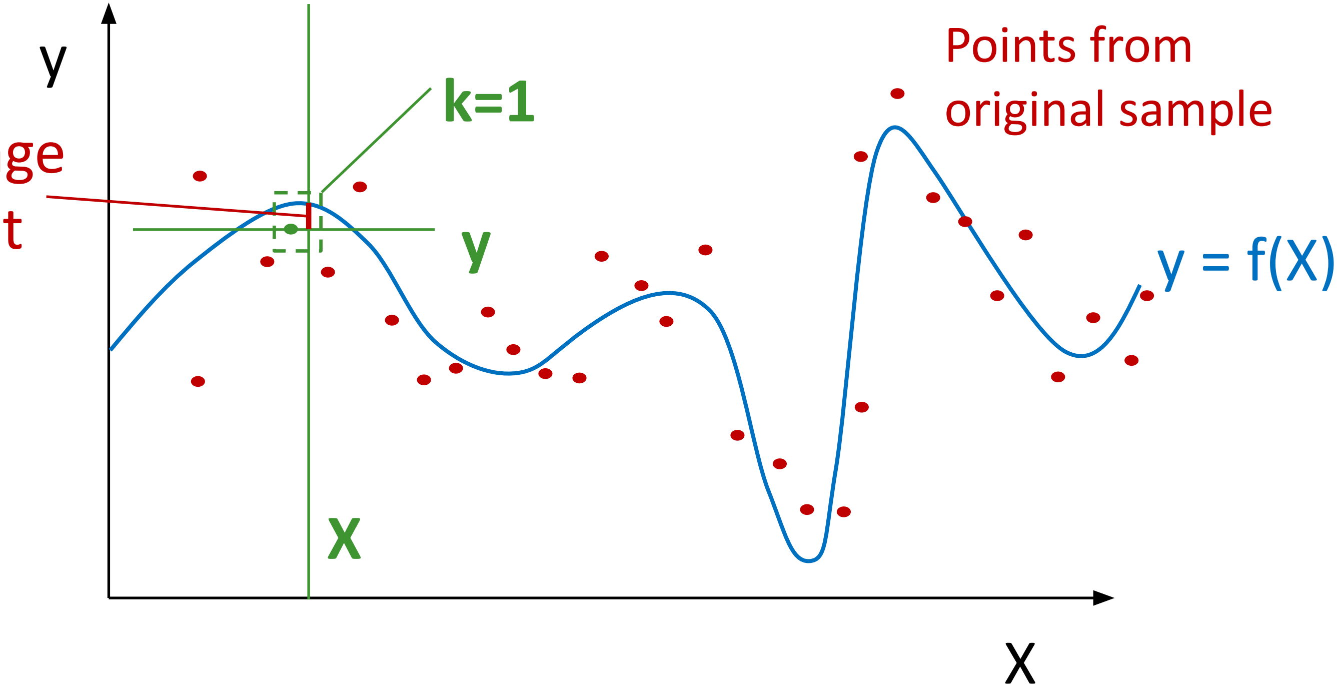


Choosing k



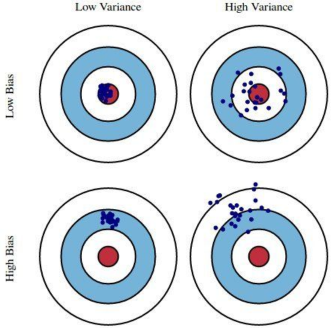
- **Small k** → low bias, high variance
- Large k → high bias, low variance

Bias = average of this offset

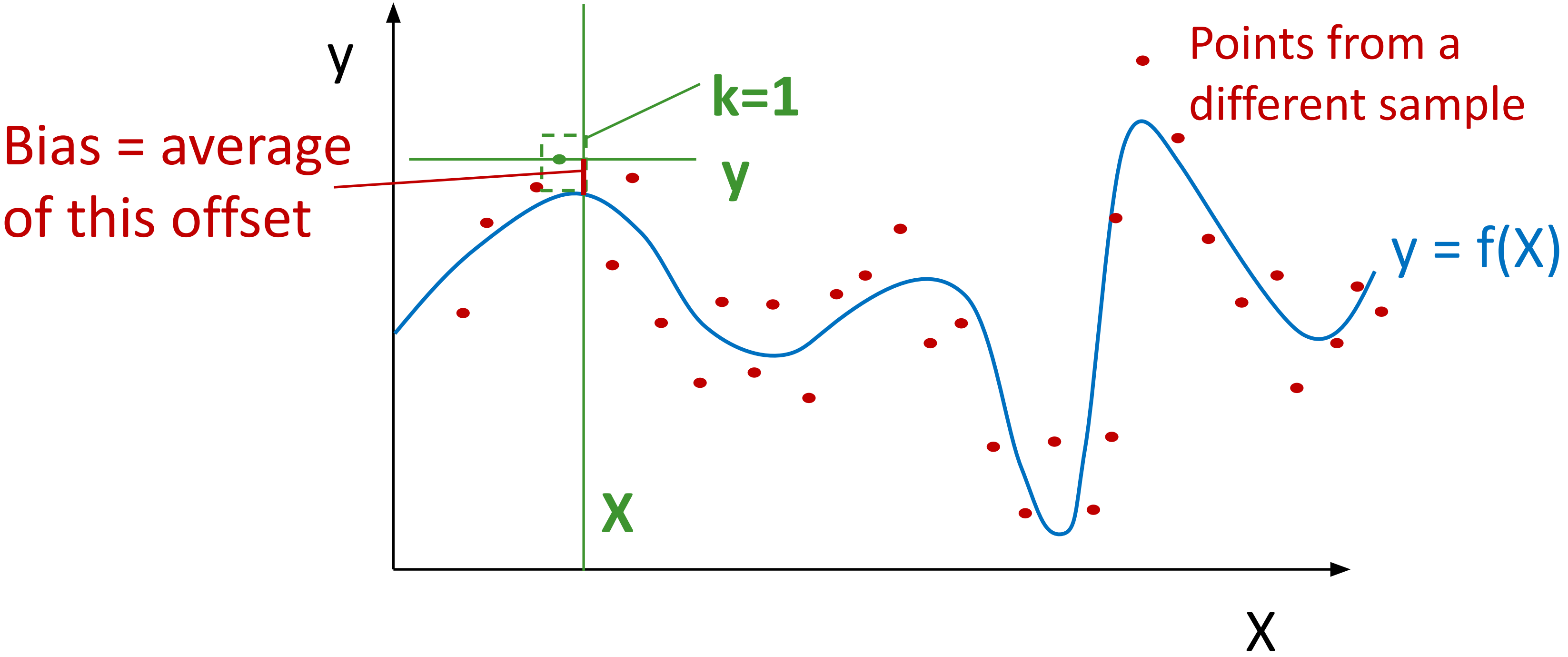


Points from original sample

Choosing k



- **Small k** → low bias, high variance
- Large k → high bias, low variance

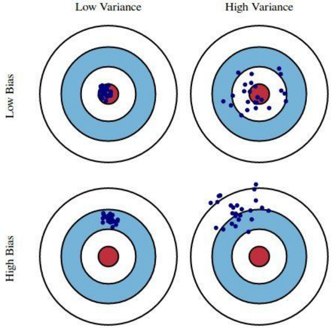


Bias = average of this offset

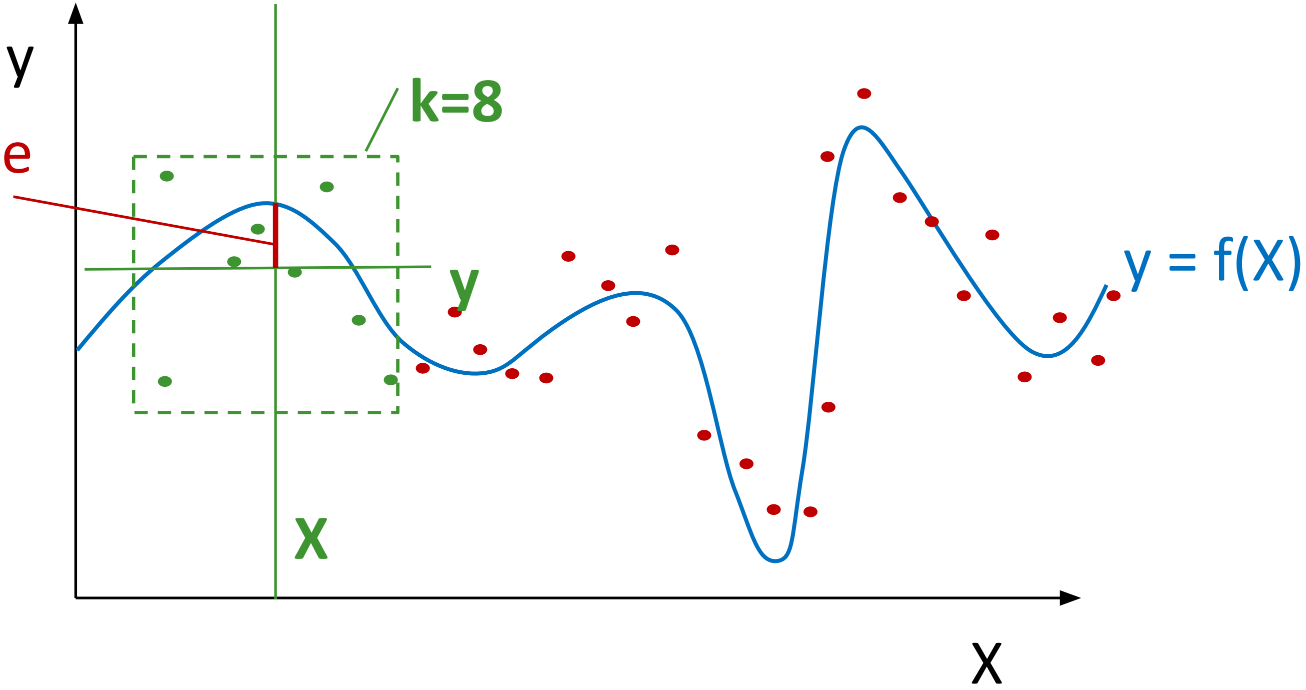
Points from a different sample

y = f(X)

Choosing k

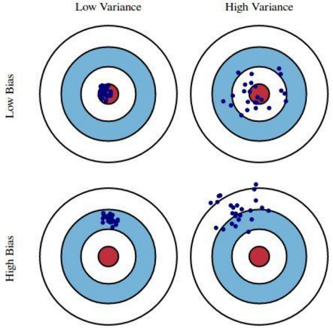


- Small k ⟶ low bias, high variance
- **Large k** ⟶ high bias, low variance

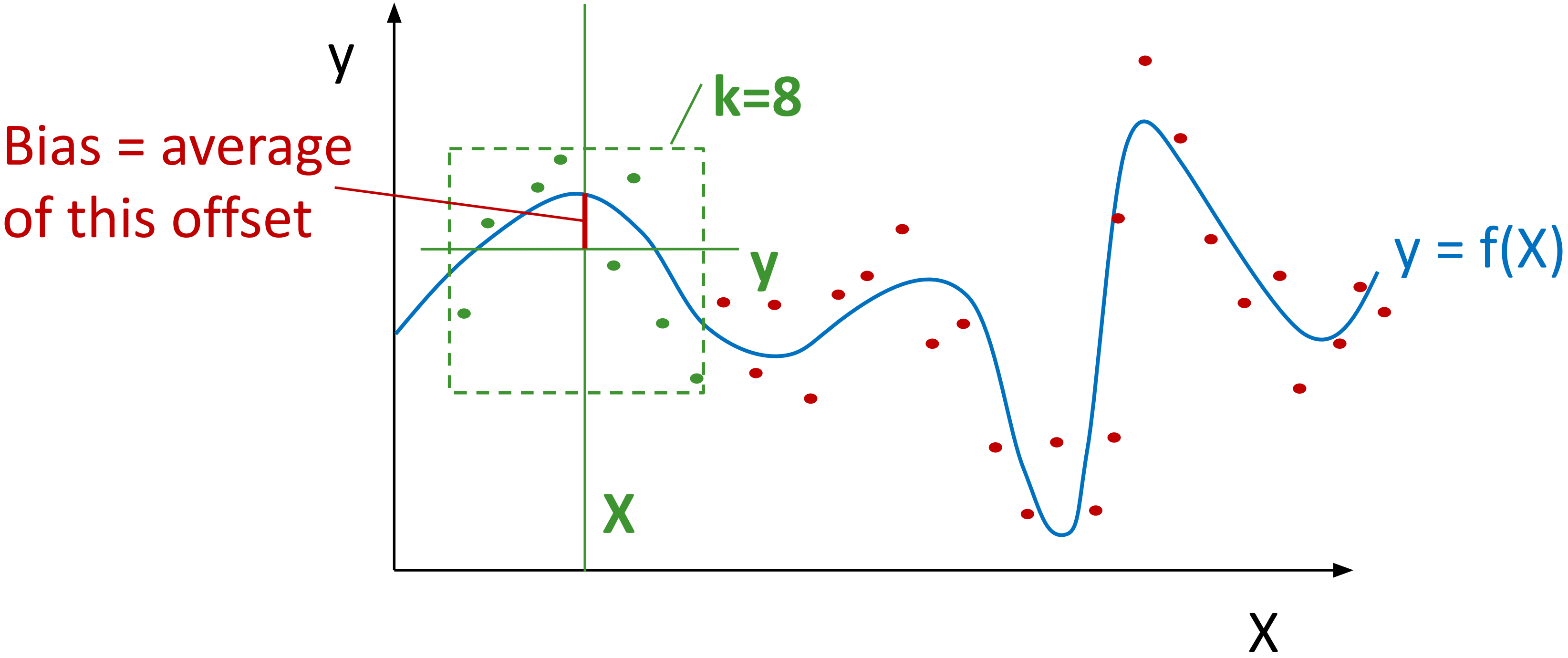


Bias = average of this offset

Choosing k



- Small k → low bias, high variance
- **Large k** → high bias, low variance



Choosing k in practice

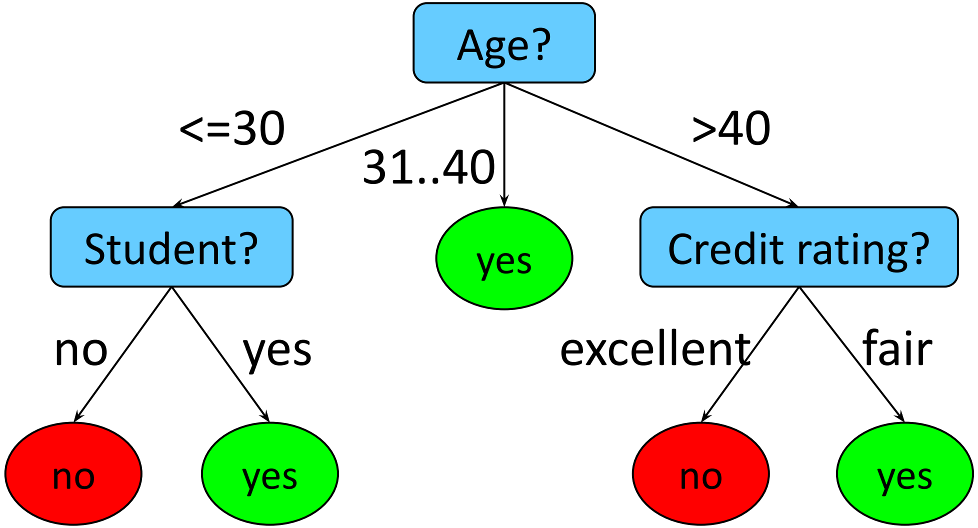
Use leave-one-out (LOO) cross-validation:

- **Split:** Break data into train and test subsets, e.g. 80-20 % random split.
- **Predict:** For each point in the training set, predict using the k nearest neighbors from the set of all *other* points in training set. Measure the LOO error rate (classification) or squared error (regression).
- **Tune:** Try different values of k, and use the one that gives minimum leave-one-out error.
- **Evaluate:** Measure error on the test set to quantify performance.

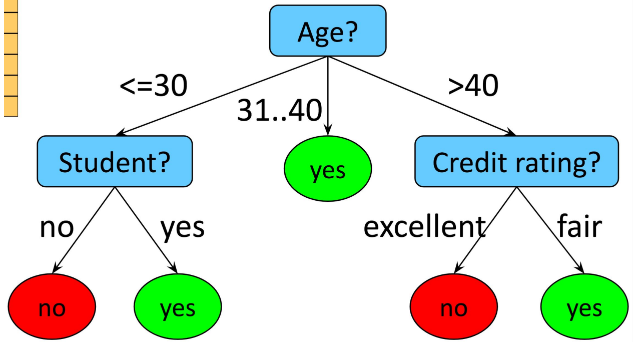
Decision trees

Decision trees: example

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no



Decision trees



Model: flow-chart-like tree structure

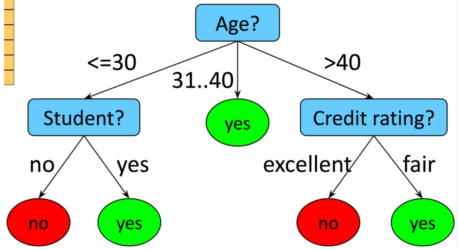
- Nodes are tests on a single attribute
- Branches are attribute values of parent node
- Leaves are marked with class labels

Goal: find decision tree that maximizes classification accuracy on given dataset

Optimization:

- NP-hard
- Heuristic: greedy top-down tree construction + pruning

Decision tree induction



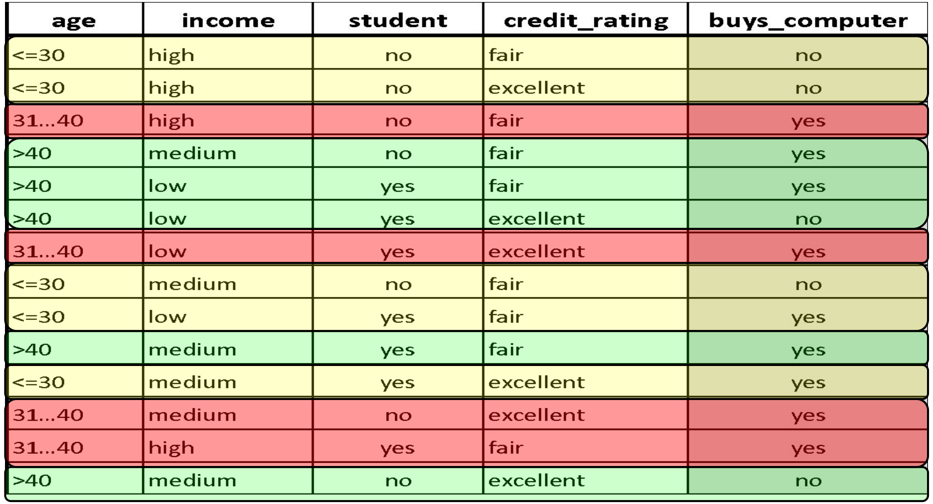
Tree construction (top-down divide-and-conquer strategy)

- At the beginning, all training samples belong to the root
- Examples are partitioned recursively based on selected “most discriminative” attributes
- Discriminative power based on information gain (in ID3 and C4.5 algorithms) or Gini impurity (in CART algorithm)

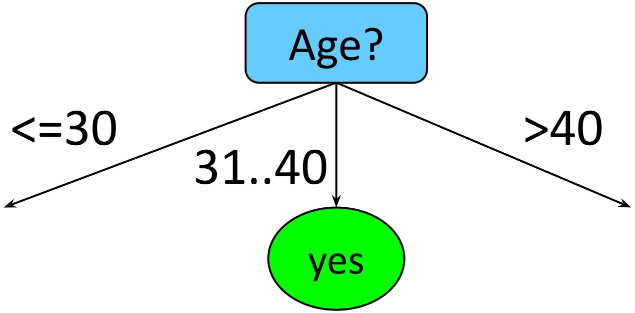
Partitioning stops if

- All samples belong to the same class → assign the class label to the leaf
- There are no attributes left → majority voting to assign the class label to the leaf

Decision tree induction

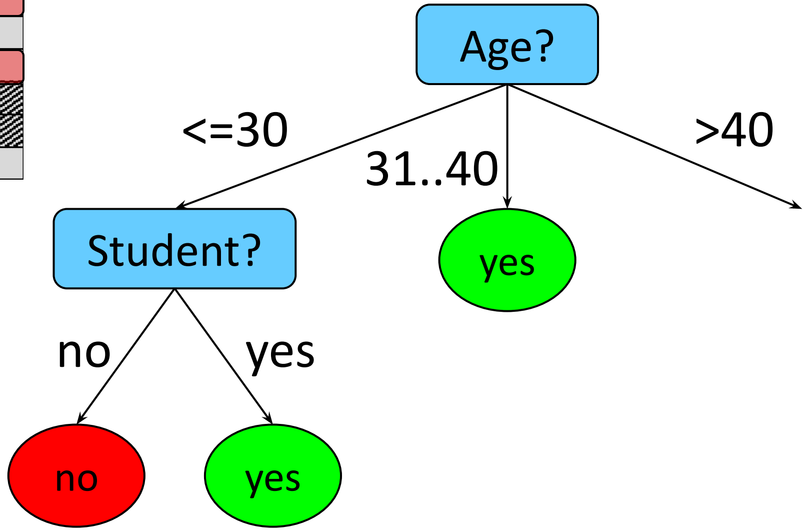


age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no



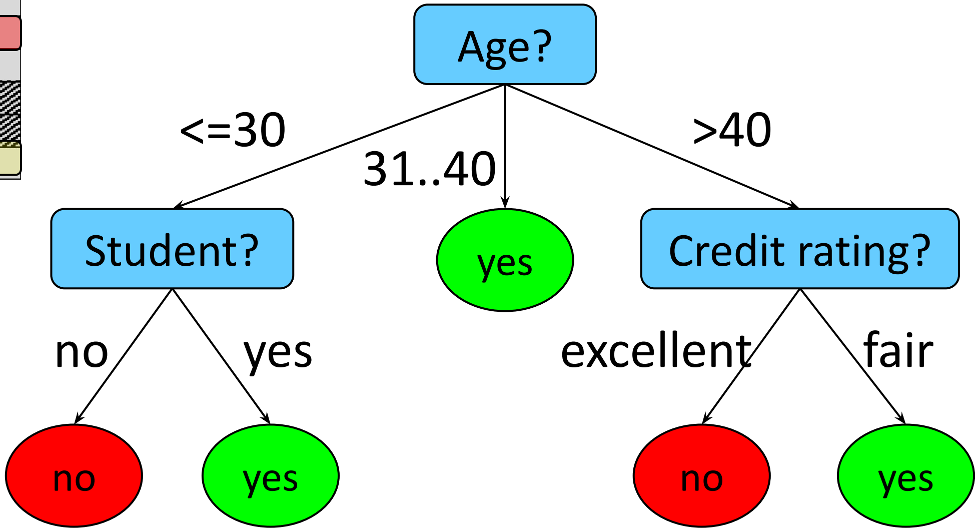
Decision tree induction

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no



Decision tree induction

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31..40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31..40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31..40	medium	no	excellent	yes
31..40	high	yes	fair	yes
>40	medium	no	excellent	no



Attribute selection

At a given branch in the tree, the set of samples S to be classified has P positive and N negative samples

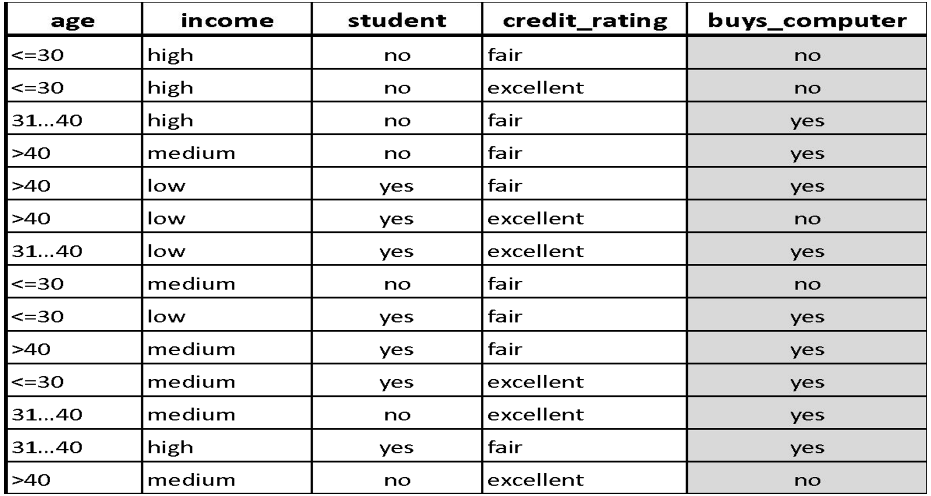
The amount of entropy in the set S is

$$H(P,\ N) = -\frac{P}{P+N}\log_2\frac{P}{P+N} - \frac{N}{P+N}\log_2\frac{N}{P+N}$$

Note that:

- If P=0 (or N=0), H(P, N) = 0 → no uncertainty
- If P=N, H(P, N) = 1 → max uncertainty

Attribute selection



age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$H_S = H(9, 5) = 0.94$

Age [<=30] $H(2, 3) = 0.97$

Age [31...40] $H(4, 0) = 0$

Age [>40] $H(3, 2) = 0.97$

Income [high] $H(2, 2) = 1$

Income [med] $H(4, 2) = 0.92$

Income [low] $H(3, 1) = 0.81$

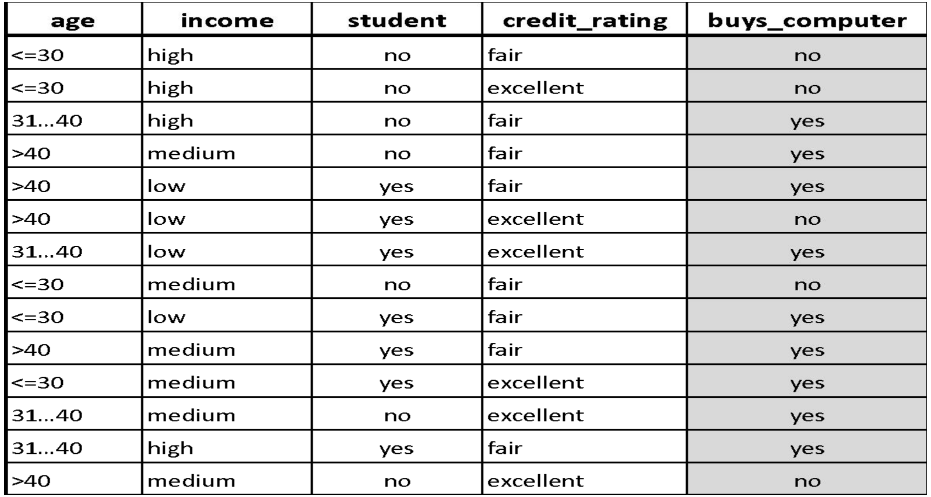
Student [yes] $H(6, 1) = 0.59$

Student [no] $H(3, 4) = 0.98$

Rating [fair] $H(6, 2) = 0.81$

Rating [exc] $H(3, 3) = 1$

Attribute selection



age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$$H_S = H(9, 5) = 0.94$$

$$\boxed{\begin{aligned} H_{Age} &= p([<=30]) \cdot H(2, 3) + p([31...40]) \cdot H(4, 0) + p([>40]) \cdot H(3, 2) = \\ &= 5/14 \cdot 0.97 + 4/14 \cdot 0 + 5/14 \cdot 0.97 = 0.69 \end{aligned}}$$

$$\begin{aligned} H_{Income} &= p([high]) \cdot H(2, 2) + p([med]) \cdot H(4, 2) + p([low]) \cdot H(3, 1) = \\ &= 4/14 \cdot 1 + 6/14 \cdot 0.92 + 4/14 \cdot 0.81 = 0.91 \end{aligned}$$

$$H_{Student} = p([yes]) \cdot H(6, 1) + p([no]) \cdot H(3, 4) = 7/14 \cdot 0.59 + 7/14 \cdot 0.98 = 0.78$$

$$H_{Rating} = p([fair]) \cdot H(6, 2) + p([exc]) \cdot H(3, 3) = 8/14 \cdot 0.81 + 6/14 \cdot 1 = 0.89$$

Attribute selection

Attribute A partitions S into S_1, S_2, ... S_v

Entropy of attribute A is
$$H(A) = \sum_{i=1}^{v} \frac{P_i + N_i}{P + N} H(P_i, N_i)$$

The *information gain* obtained by splitting S using A is

$$Gain(A) = H(P, N) - H(A)$$

Gain(Age) = 0.94 – 0.69 = 0.25 ← split on age

Gain(Income) = 0.94 – 0.91 = 0.03

Gain(Student) = 0.94 – 0.78 = 0.16

Gain(Rating) = 0.94 – 0.89 = 0.05

Pruning

The construction phase does not filter out noise
→ **overfitting**

Many possible pruning strategies

- Stop partitioning a node when the corresponding number of samples assigned to a leaf goes below a threshold
- Bottom-up cross validation: Build the full tree and replace nodes with leaves labeled with the majority class if classification accuracy on a **validation set (not seen during training!)** does not get worse this way

Comments

Decision trees are an example of a classification algorithm

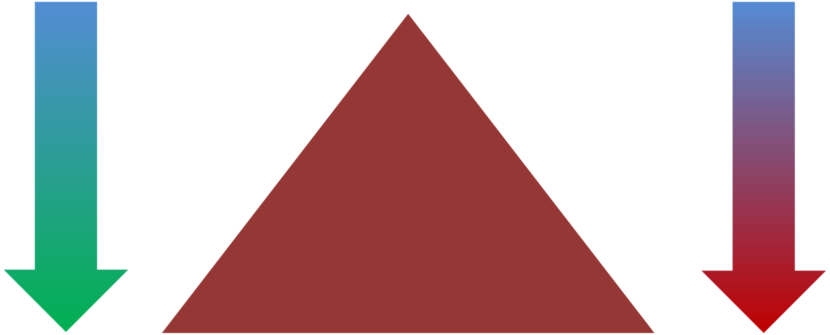
- Many other out there (k-NN, naive Bayes, SVM, neural networks, logistic regression, random forest ...)

Maybe not the best one ...

- Sensitive to small perturbation in the data (high variance)
- Tend to overfit
- Non-incremental: Need to be re-trained from scratch if new training data becomes available

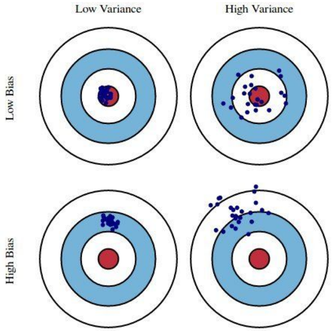
Decision tree models

- As tree depth increases, how do bias and variance change? (Hint: think about k-NN)



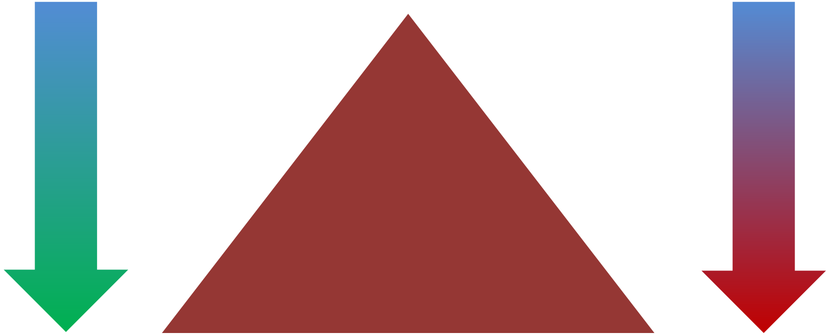
POLLING TIME

- Scan QR code or go to https://web.speakup.info/room/join/66626



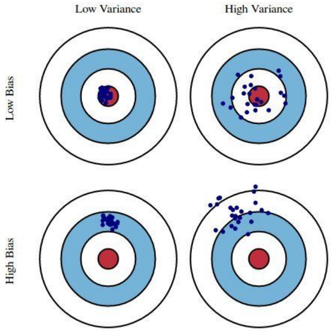
Decision tree models

- As tree depth increases, bias decreases and variance generally increases. Why? (Hint: think about k-NN)



Bias decreases with tree depth

Variance increases with tree depth



Ensemble methods

Are, metaphorically, like **crowdsourced machine learning algorithms**:

- Take a collection of simple or *weak* learners
- Combine their results to make a single, better learner

Types:

- **Bagging:** train learners in parallel on different samples of the data, then combine by voting (for discrete output) or by averaging (for continuous output).
- **Stacking:** combine outputs from various models using a second-stage learner (e.g., linear regression).
- **Boosting:** train learner again, but after filtering/weighting samples based on output of previous train/test runs.

Random forests

Grow K trees on datasets sampled from the original dataset (size N) with replacement (bootstrap samples), p = number of features.

- Draw K bootstrap samples of size N
- Grow each decision tree by selecting a random set of m out of p features at each node and choosing the best feature to split on.
- At testing time, aggregate the predictions of the trees (most popular vote, or average) to produce the final class (example of bagging).

Typically m might be e.g. sqrt(p), but can be smaller.

Random forests

Principles: we want to take a vote between different learners so we don’t want the models to be too similar. The following two criteria ensure diversity in the individual trees:

- Draw K bootstrap samples of size N:
  - Each tree is trained on different data.
- Grow a decision tree by selecting a random set of m out of p features at each node, and choosing the best feature to split on.
  - Corresponding nodes in different trees (usually) can’t use the same feature to split on.

Random forests

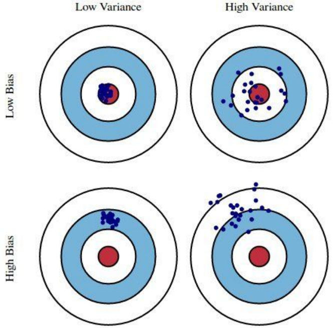
- **Very popular in practice,** probably the most popular classifier for dense data (up to a few thousand features)
- **Easy to implement** (simply train many normal decision trees)
- **Easy to parallelize**

- **Needs many passes over the data** – at least the max depth of the trees (<< boosted trees though, cf. next slide)

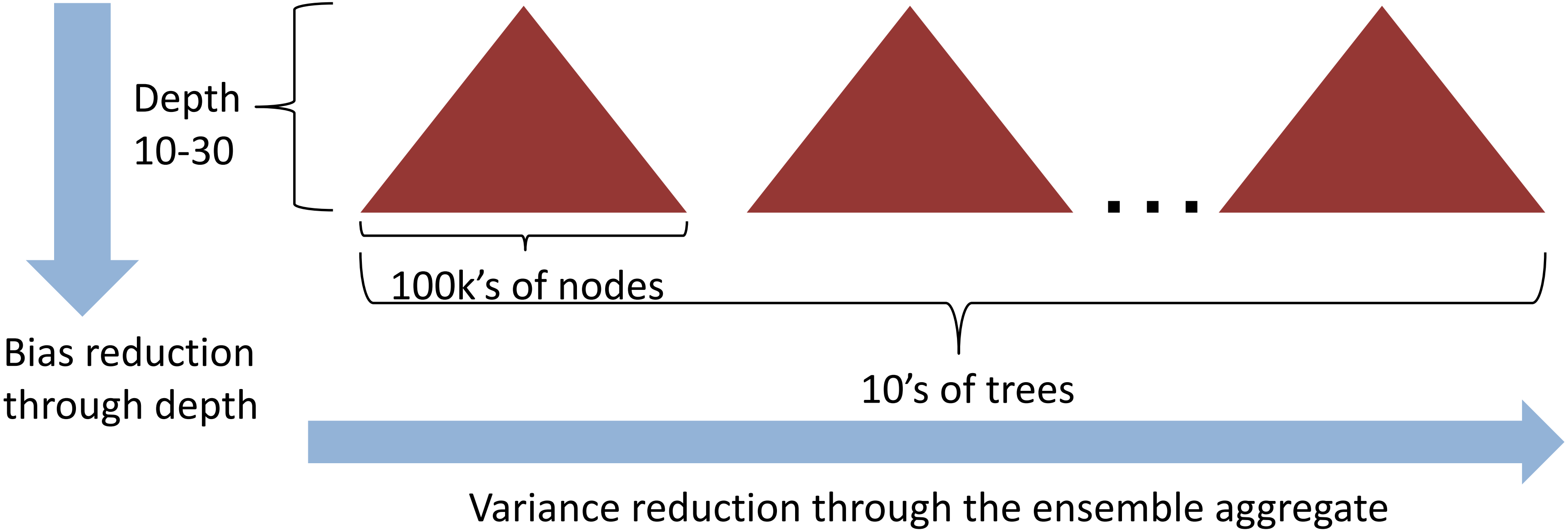
Boosted decision trees

- A more recent alternative to random forests (RF) [good intro here]
- In contrast to RFs, whose trees are trained **independently** by bagging, BDT trees are trained **sequentially** by **boosting**: Each tree is trained to predict ("correct") residual errors of previous trees (--> bias reduction).
- Final prediction: sum of predictions made by individual trees.
- Both RF and boosted trees can produce very high-quality models. Superiority of one method or the other is dataset-dependent.
- Resource requirements are very different as well, so it's actually non-trivial to compare the methods.

Random forests vs. boosted trees



- The “geometry” of the methods is very different:
- Random forests use 10’s of deep, large trees:



Depth 10-30

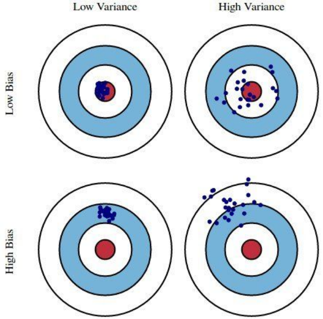
100k’s of nodes

10’s of trees

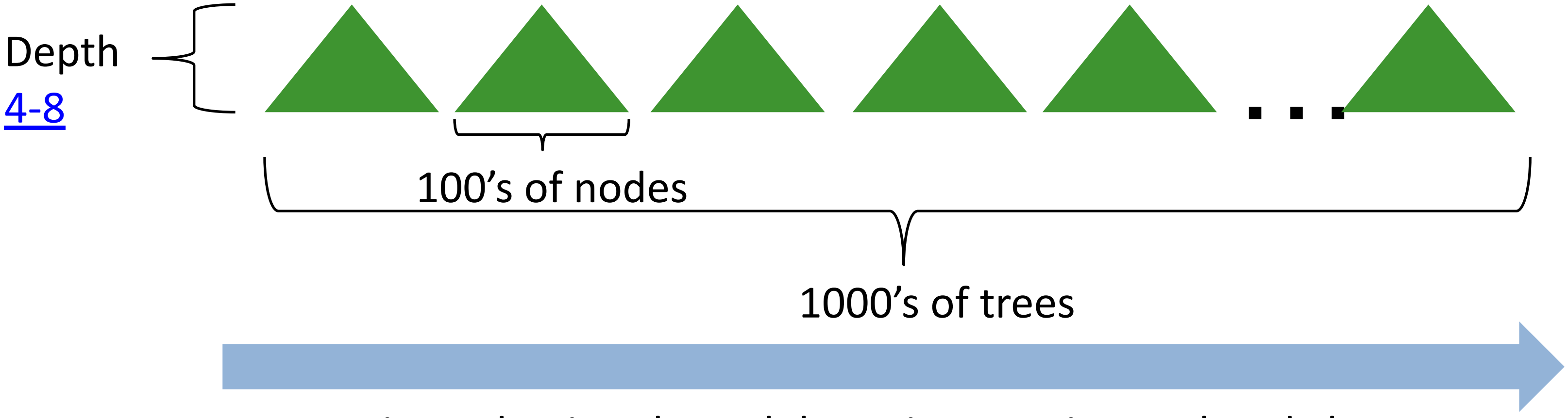
Bias reduction through depth

Variance reduction through the ensemble aggregate

Random forests vs. boosted trees



- The "geometry" of the methods is very different:
- Boosted decision trees use 1000's of shallow, small trees:



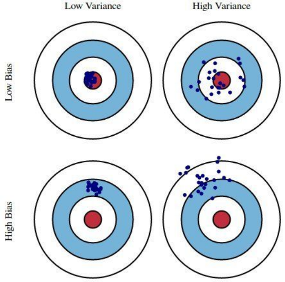
Depth
4-8

100's of nodes

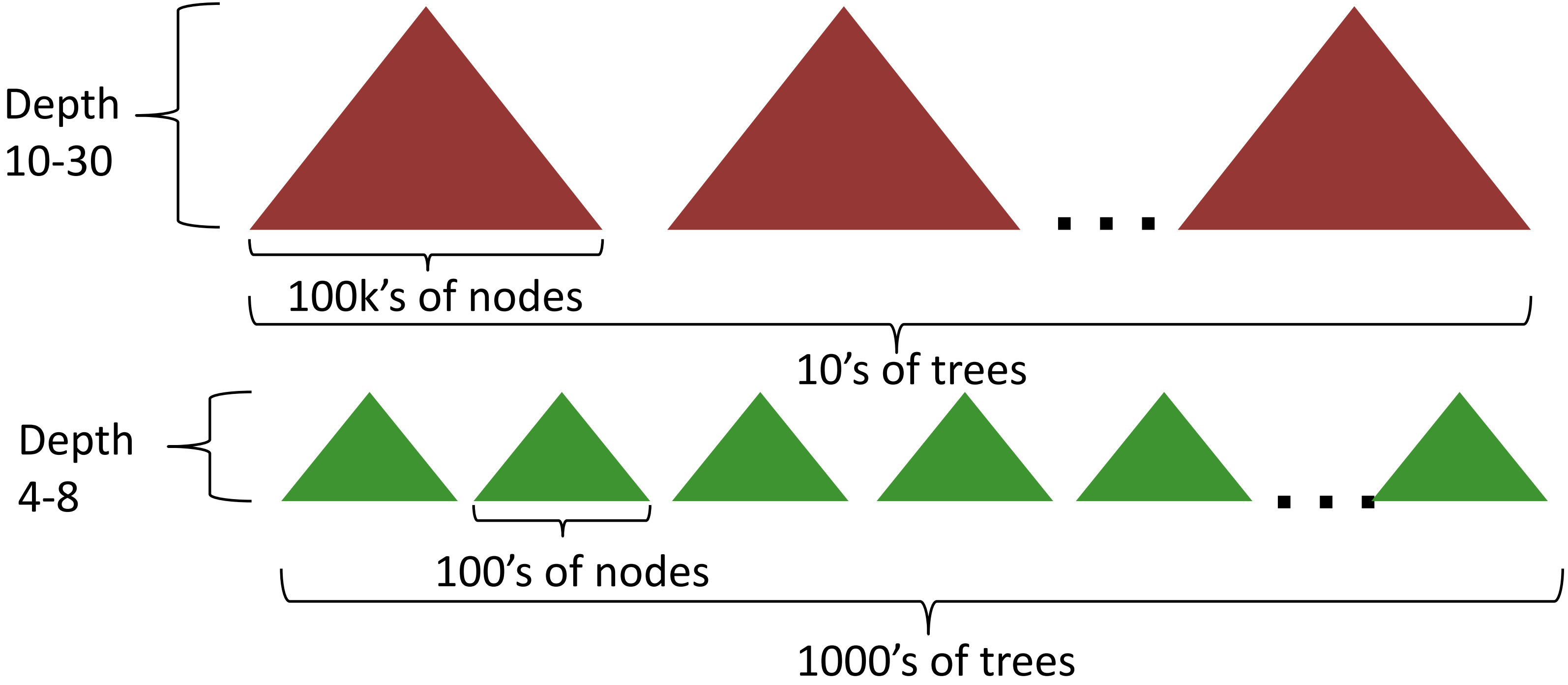
1000's of trees

Bias reduction through boosting – variance already low

Random forests vs. boosted trees



- RF training embarrassingly parallel, can be very fast
- Evaluation of trees (runtime) also much faster for RFs



Depth 10-30

100k's of nodes

10's of trees

Depth 4-8

100's of nodes

1000's of trees

For your personal perusal:

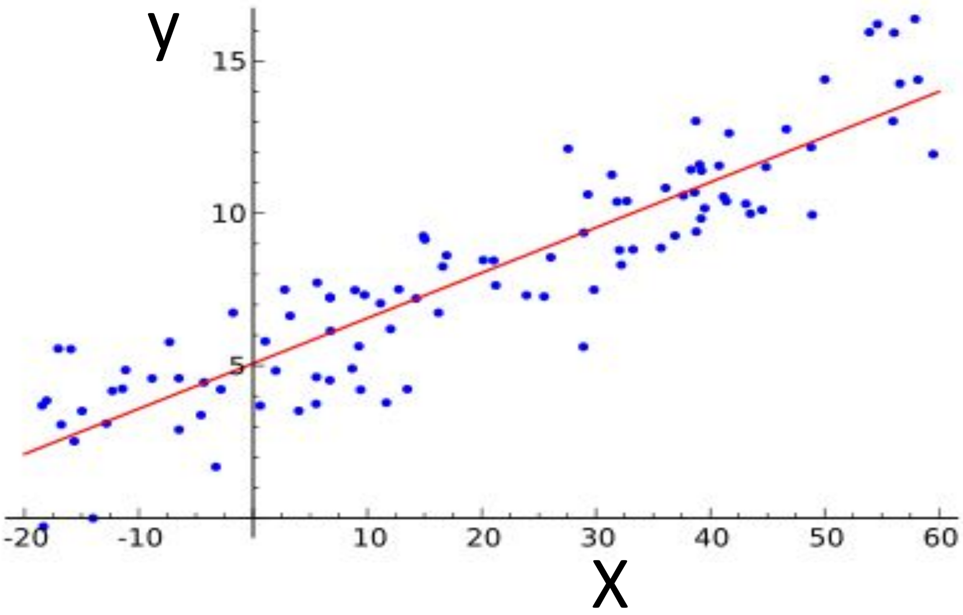
“A visual introduction to machine learning”

http://www.r2d3.us/visual-intro-to-machine-learning-part-1/

Linear and logistic regression

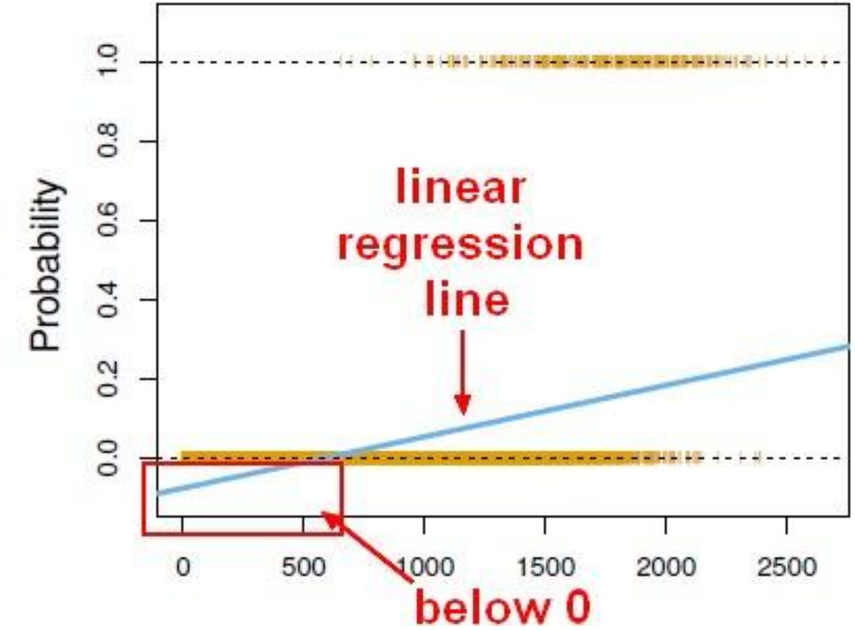
Linear regression

- Your good friend from lecture 5 on regression analysis
- Goal: find the “best” line (linear function y=f(X)) to explain the data

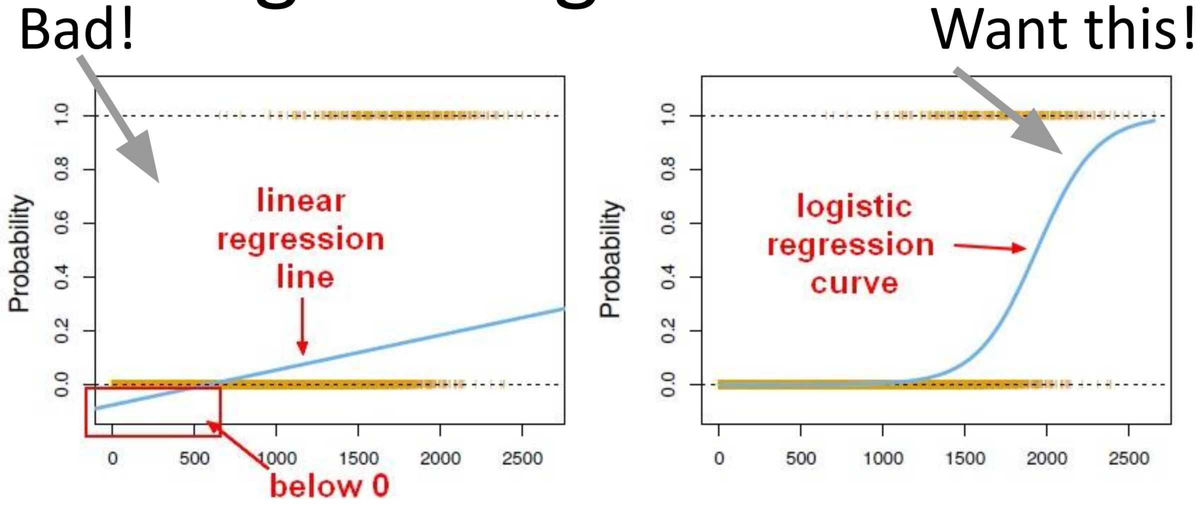


How to model binary events?

- E.g., X: student features; y: did student pass ADA?
- Desired output: f(X) = probability of passing ADA, given feats X
- Problem with linear regression:
  f(X) can be below 0 or above 1



Logistic regression



Bad!

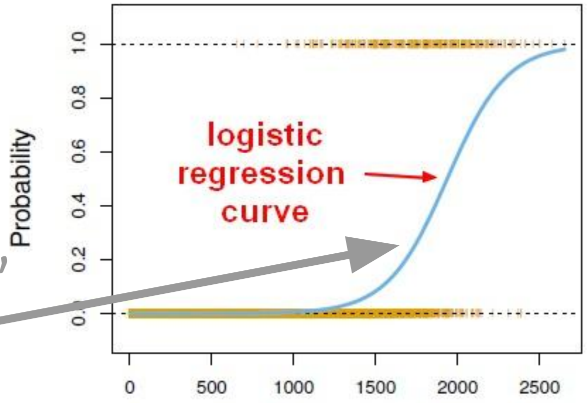
Want this!

- Trick: don’t deal with probabilities, which range from 0 to 1, but with log odds, which range from -inf to +inf
- Probability y ⇔ odds y/(1-y) ⇔ log odds log[y/(1-y)]
- Model log odds as a linear function of X

Logistic regression

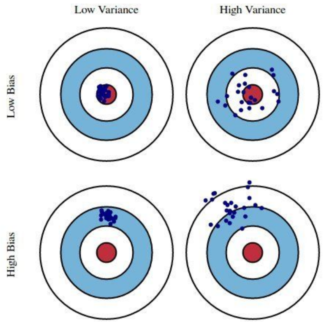
- Model log odds as a linear function of X
- $\beta^T X = \log[y/(1-y)]$
- Solve for y: $y = 1 / (1 + \exp(-\beta^T X))$

"sigmoid"



- Finding best model β via maximum likelihood:
  - Don't use square loss as in linear regression (where y is assumed to be generated from Normal distribution)
  - Use cross-entropy loss instead (y assumed to be generated from Bernoulli distribution, i.e., biased coin)

Overfitting



- The more features the better?
  - **NO!**
  - More features mean less bias, but more variance
  - Overfitting
- Carefully selected features can improve model accuracy
  - E.g., keep features that correlate with the label y
  - Forward/backward feature selection
  - Regularization (e.g., penalize norm of weight vector)
- More on such practical aspects: next lecture (“applied ML”)

Feedback

Give us feedback on this lecture here:
https://go.epfl.ch/ada2022-lec7-feedback

- What did you (not) like about this lecture?
- What was (not) well explained?
- On what would you like more (fewer) details?
- …

Criteria

Predictive performance (accuracy, AUC/ROC, precision, recall, F1-score, etc.)

Speed and scalability

- Time to build the model
- Time to use the model
- In memory vs. on disk processing
- Communication cost

Robustness

- Handling noise, outliers, missing values

Interpretability

- Understanding the model and its decisions (black box vs. white box)

Compactness of the model

- Mobile and embedded devices

k-NN and the curse of dimensionality

The curse of dimensionality refers to “weird” phenomena that occur in high dimensions (100s to millions) that do not occur in low-dimensional (e.g. 3-dimensional) space.

In particular data in high dimensions are much sparser (less dense) than data in low dimensions.

For k-NN, this means there are fewer points that are very close in feature space (very similar) to the point X whose y we want to predict.

k-NN and the curse of dimensionality

From this perspective, it’s surprising that kNN works at all in high dimensions.

Luckily real data are not like random points in a high-dimensional cube. Instead they live in **dense clusters** and near **much lower-dimensional surfaces**.

Also, points can be very “similar” even if their Euclidean distance is large. E.g. documents with the same few dominant words are likely to be on the same topic (--> use different distance)

k-NN and the curse of dimensionality

Example: Consider a collection of uniformly random points in the unit cube. In one dimension, the average squared Euclidean distance between any two points is:

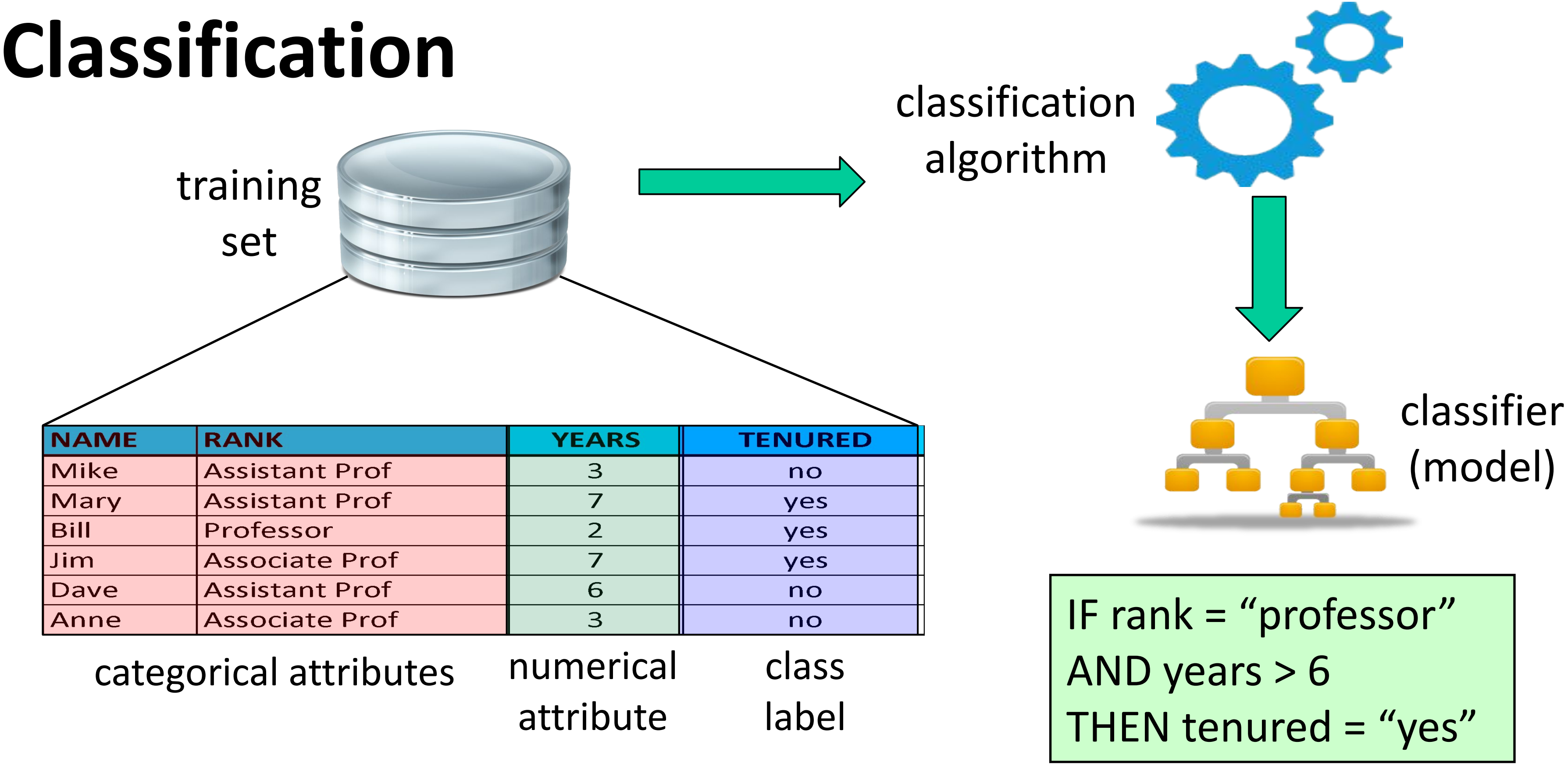
$$\int_0^1 \int_0^1 (x-y)^2 dx\, dy = \frac{1}{6}$$

In N dimensions, we add up the squared differences for all N coordinates (because the coordinates are independent in a uniform random cube), giving:

$$d^2 = \mathrm{E}[\|x-y\|^2] = \frac{N}{6}$$

So the euclidean distance scales as $\sqrt{N}$

Classification



training set

classification algorithm

classifier (model)

NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

categorical attributes

numerical attribute

class label

IF rank = “professor”
AND years > 6
THEN tenured = “yes”

On model transparency

Deep neural networks are often considered as opaque, inscrutable black boxes

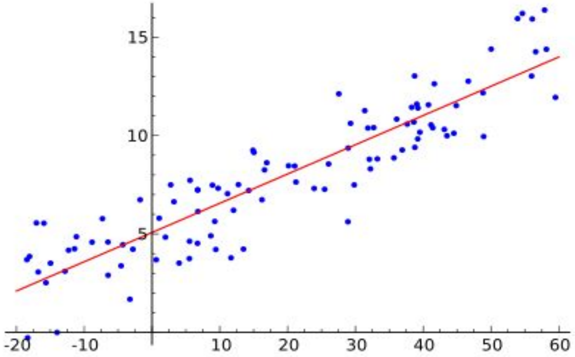
Do you think it’s easier to interpret a model with 1000’s of trees?!

Linear regression

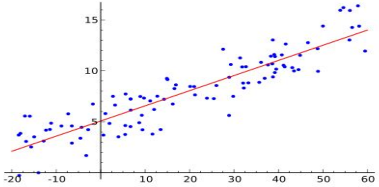
The predicted value of y is given by:

$$\hat{y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j$$

The vector of coefficients $\hat{\beta}$ is the regression model.



Least-squares solution



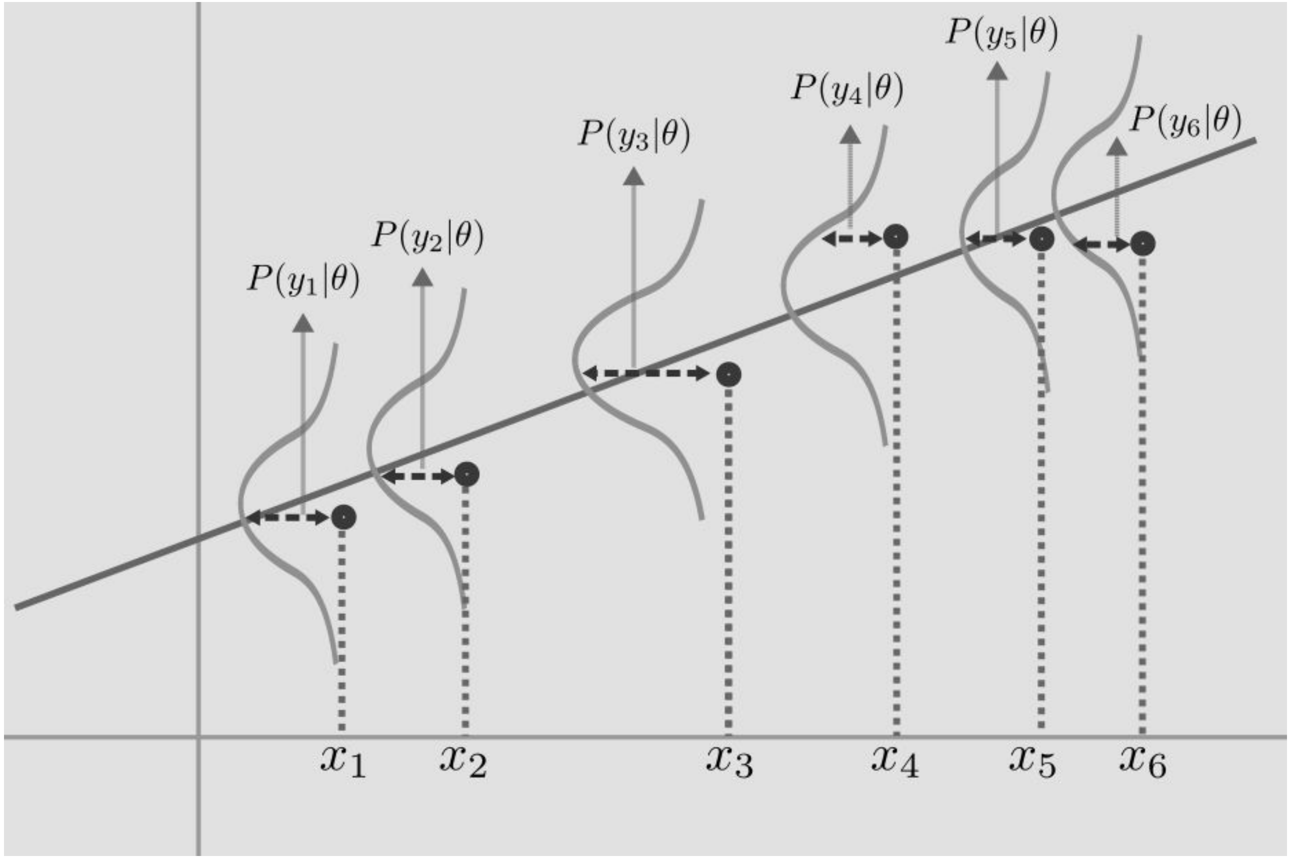
The most common measure of fit between the line and the data is the **least-squares fit**.

There is a good reason for this: If the points are generated by an ideal line with additive Gaussian noise, the least squares solution is the ***maximum likelihood solution***.

Probability of a point y_j is $\Pr(y_j) = \exp\left(\frac{-(y_j - X_j\beta)^2}{2\sigma^2}\right)$ and the probability for all points is the product over j of $\Pr(y_j)$.

We can **easily maximize the log** of this expression $\frac{-(y_j - X_j\beta)^2}{2\sigma^2}$ for one point, or the sum of this expression at all points.

Least-squares solution


$P(y_1|\theta)$
$P(y_2|\theta)$
$P(y_3|\theta)$
$P(y_4|\theta)$
$P(y_5|\theta)$
$P(y_6|\theta)$
x_1
x_2
x_3
x_4
x_5
x_6