Applied Data Analysis (CS401)

Lecture 8
Supervised learning
2018/11/08

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Robert West

dlab

Double rainbow!

Announcements

- HW2 grades released
- HW3 peer reviews
  - due in one week (Thu, Nov 15, 23:59)
  - distributed one day later (Fri, Nov 16)
- HW4 released today (due Wed, Nov 21)
- Tomorrow's lab session:
  - HW2 post mortem
  - HW4 prae vitam
  - Tutorial: Applied ML
  - Office hours (HW + projects)

Feedback

Give us feedback on this lecture here:
https://go.epfl.ch/ada2018-lec8-feedback

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

Machine Learning

- **Supervised:** We are given input/output samples (X, y) which we relate with a function y = f(X). We would like to "learn" f, and evaluate it on new data. Types:
  - **Classification:** y is discrete (class labels)
  - **Regression:** y is continuous, e.g. linear/logistic regression
- **Unsupervised:** Given only samples X of the data, we compute a function f such that y = f(X) is a "simpler" representation.
  - **Clustering:** y is discrete
  - y is continuous: **Matrix factorization, topic modeling, unsupervised neural networks, word2vec, …**

Machine Learning: Examples

- **Supervised:**
  - Is this image a cat, dog, car, house?
  - How would this user score that restaurant?
  - Is this email spam?
  - Is this blob a supernova?
- **Unsupervised:**
  - Cluster some 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:**
  - kNN (k nearest neighbors)
  - Naïve Bayes
  - Linear + logistic regression
  - Support vector machines
  - Random forests
  - Supervised neural networks
  - etc.
- **Unsupervised Learning:**
  - Clustering
  - Dimensionality reduction; e.g., topic modeling, PCA, MDS
  - Hidden Markov models (HMM)
  - etc.

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

Intro to supervised learning: k nearest neighbors (kNN)

k Nearest Neighbors

Given a query item:

Find k closest matches in a labeled dataset ↓

k Nearest Neighbors

Given a query item:

Find k closest matches

Return the most

Frequent label

k-Nearest Neighbors

k = 3 votes for “cat”

k-NN issues

The data is the model

- No training needed.
- Accuracy generally improves with more data.
- Matching is simple and fairly fast if data fits in memory.
- Usually need data in memory, but can be run off disk.

Minimal configuration:

- Only parameter is k (number of neighbors)
- But two other choices are important:
  - Weighting of neighbors (e.g. inverse distance)
  - Similarity metric

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.

k-NN distance 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 metrics

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

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

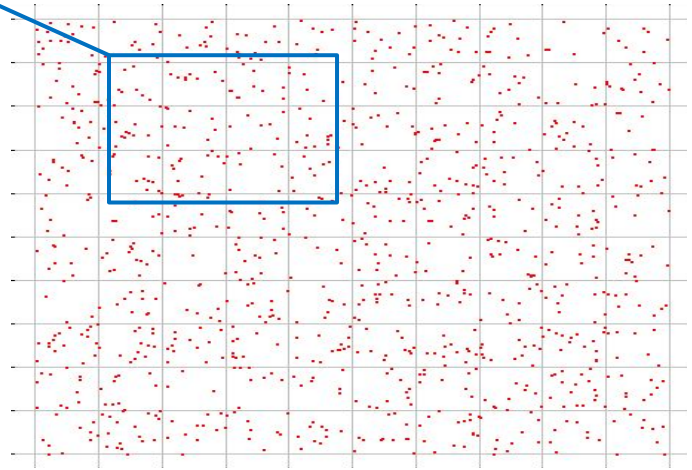
- **Edit Distance:** for strings, especially genetic data.

- **Mahalanobis Distance:** Normalized by the sample covariance matrix – unaffected by coordinate transformations.

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

Predicting from Samples

- Most datasets are **samples** from an **infinite population**.
- We are most interested in **models of the population**, but we have access only to a **sample** of it.



For datasets consisting of (X,y)

- features X, label y

For the true model f:

- $y = f(X)$

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

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.

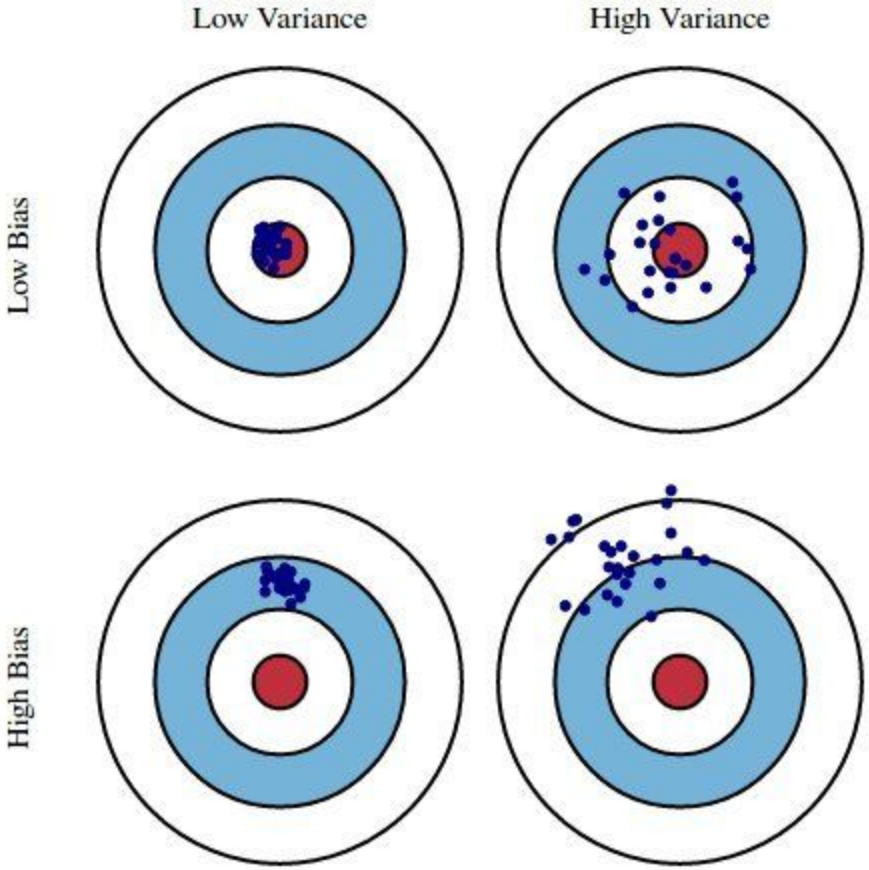
i.e. $$Bias = \mathrm{E}[f_D(X) - y]$$

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.

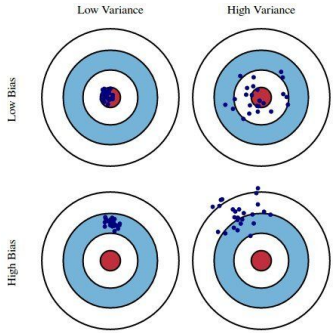

Low Variance
High Variance
Low Bias
High Bias

Bias and Variance Tradeoff

There is usually a bias-variance tradeoff caused by model complexity.

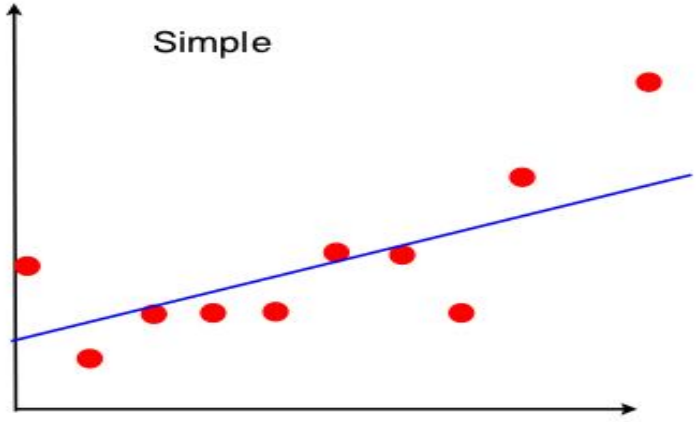
Complex models (many parameters) usually have lower bias, but higher variance.

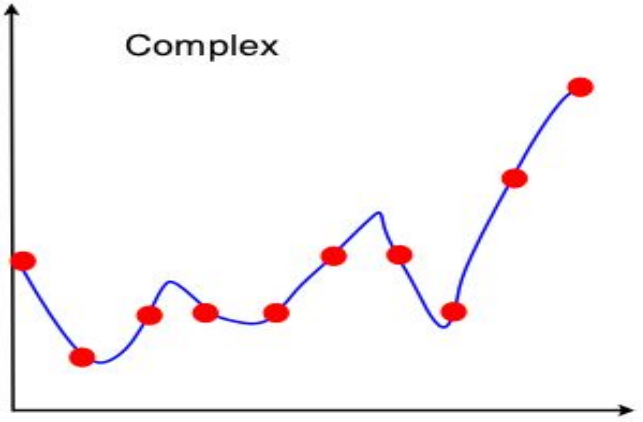
Simple models (few parameters) have higher bias, but lower variance.

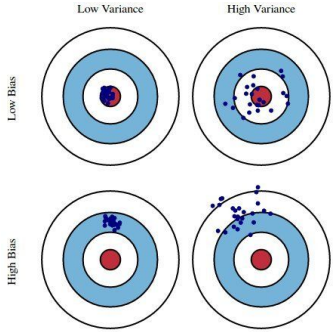


Bias and Variance Tradeoff

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







Bias and Variance Tradeoff

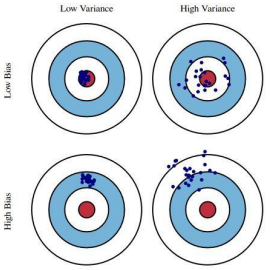
The total expected error is

$$Bias^2 + Variance$$

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**.

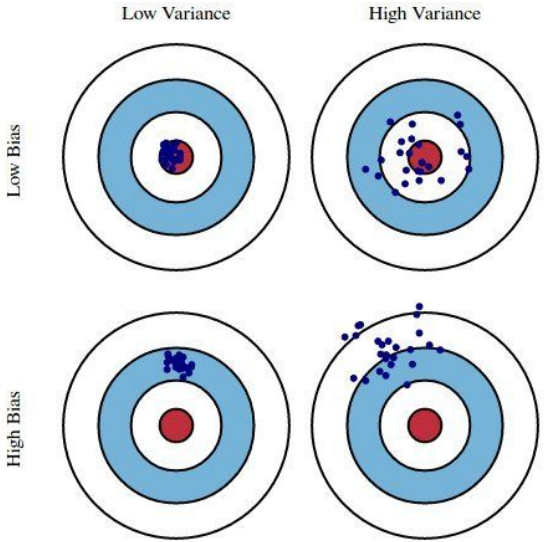


```
kNN = stack.pop()
```

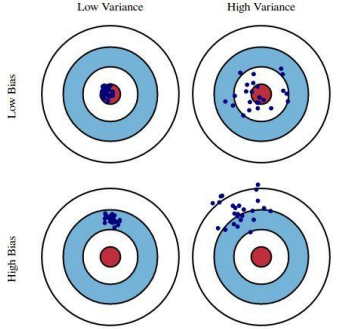
Choosing k for k nearest neighbors

We have a bias/variance tradeoff:

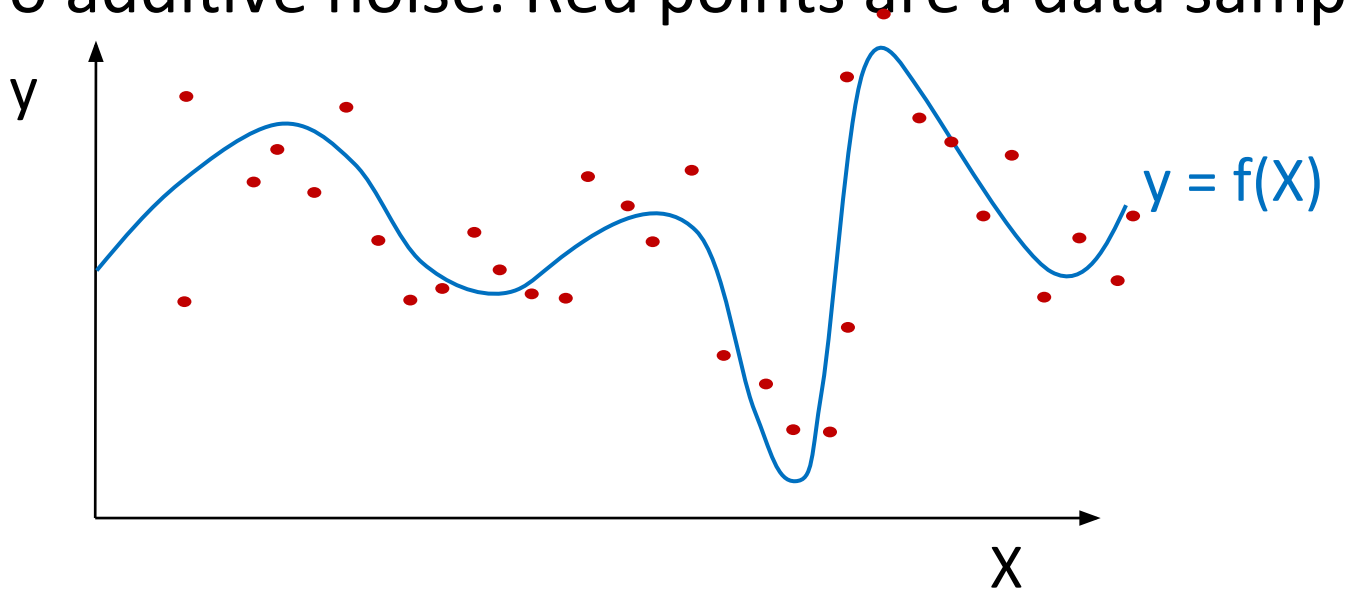
- Small k ⟶ ?
- Large k ⟶ ?



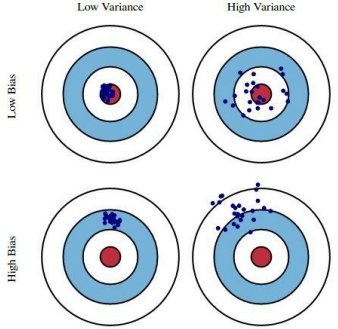
Choosing k



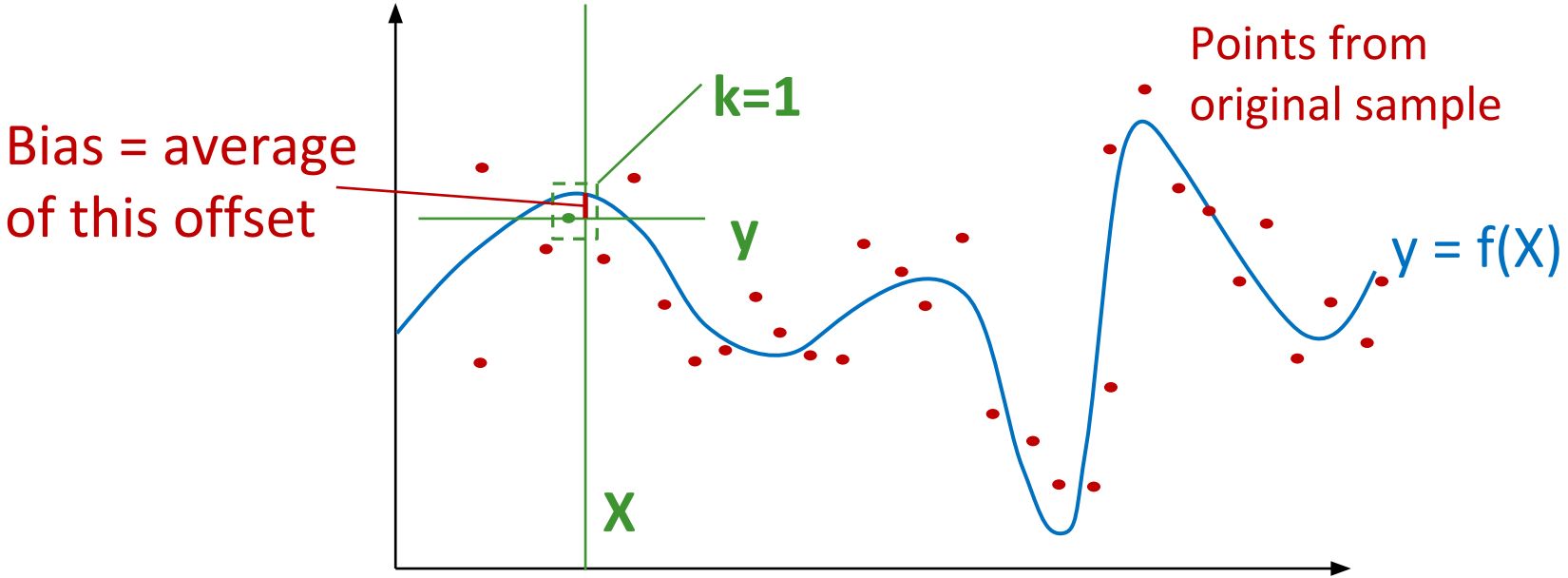
- Small k ⟶ low bias, high variance
- Large k ⟶ high bias, low variance
- Assume the real data follows the blue curve, with some mean-zero additive noise. Red points are a data sample.



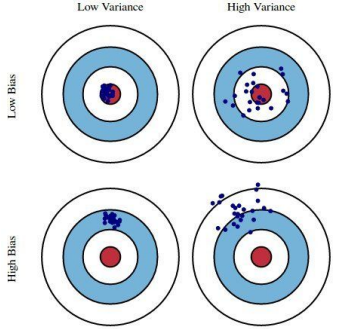
Choosing k



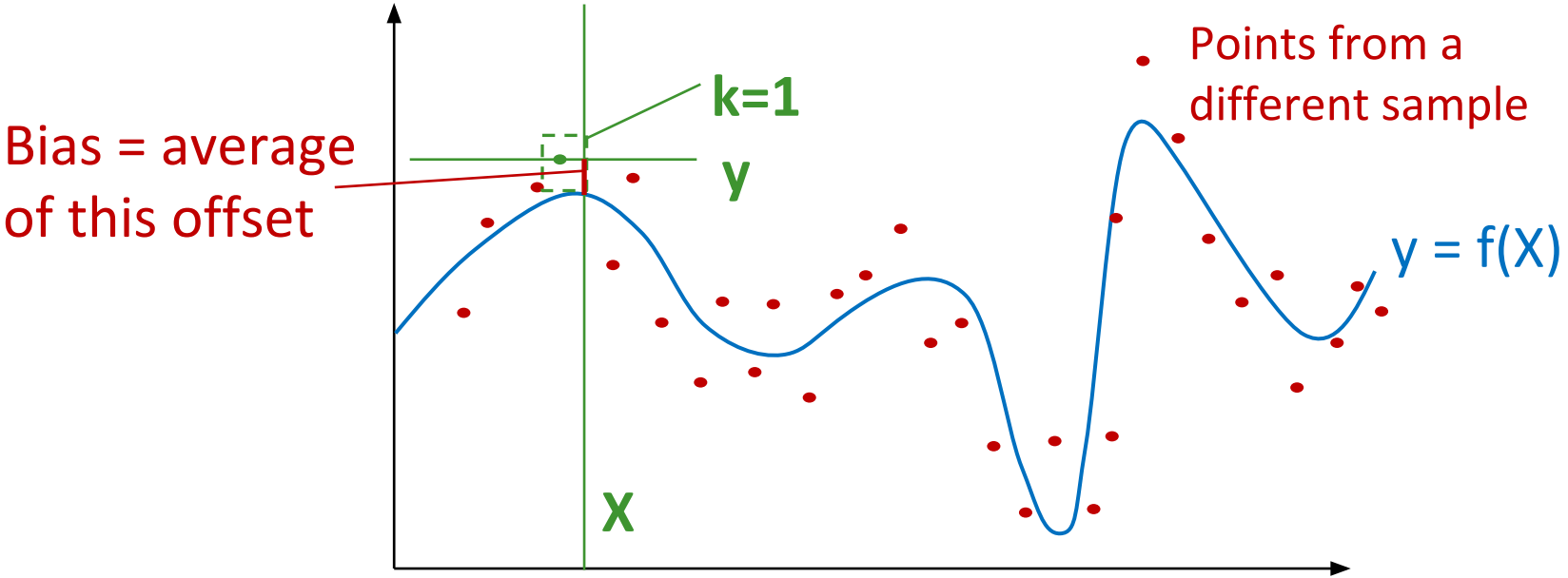
- **Small k** ⟶ low bias, high variance
- Large k ⟶ high bias, low variance



Choosing k

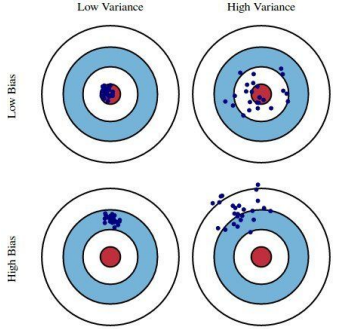


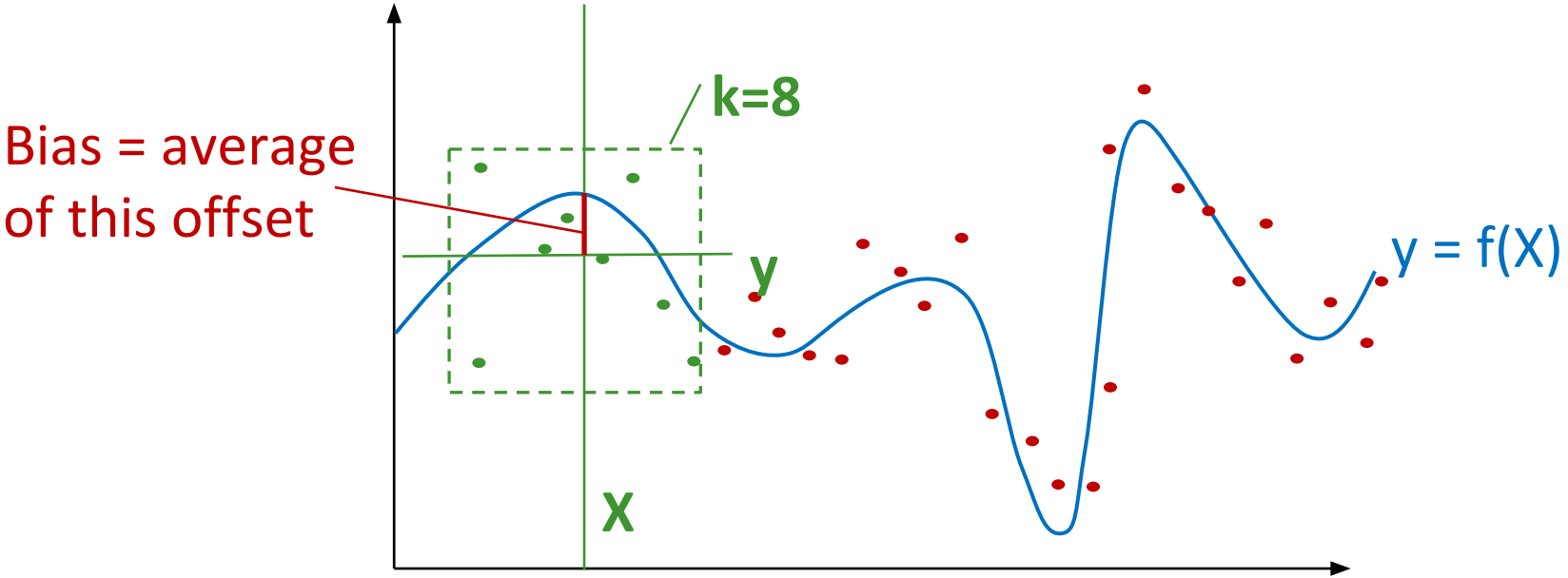
- **Small k** ⟶ low bias, high variance
- Large k ⟶ high bias, low variance



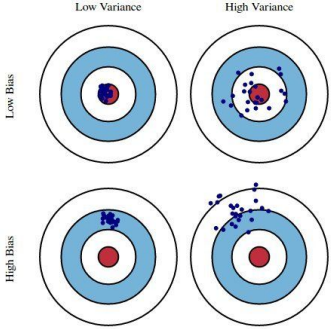
Choosing k

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

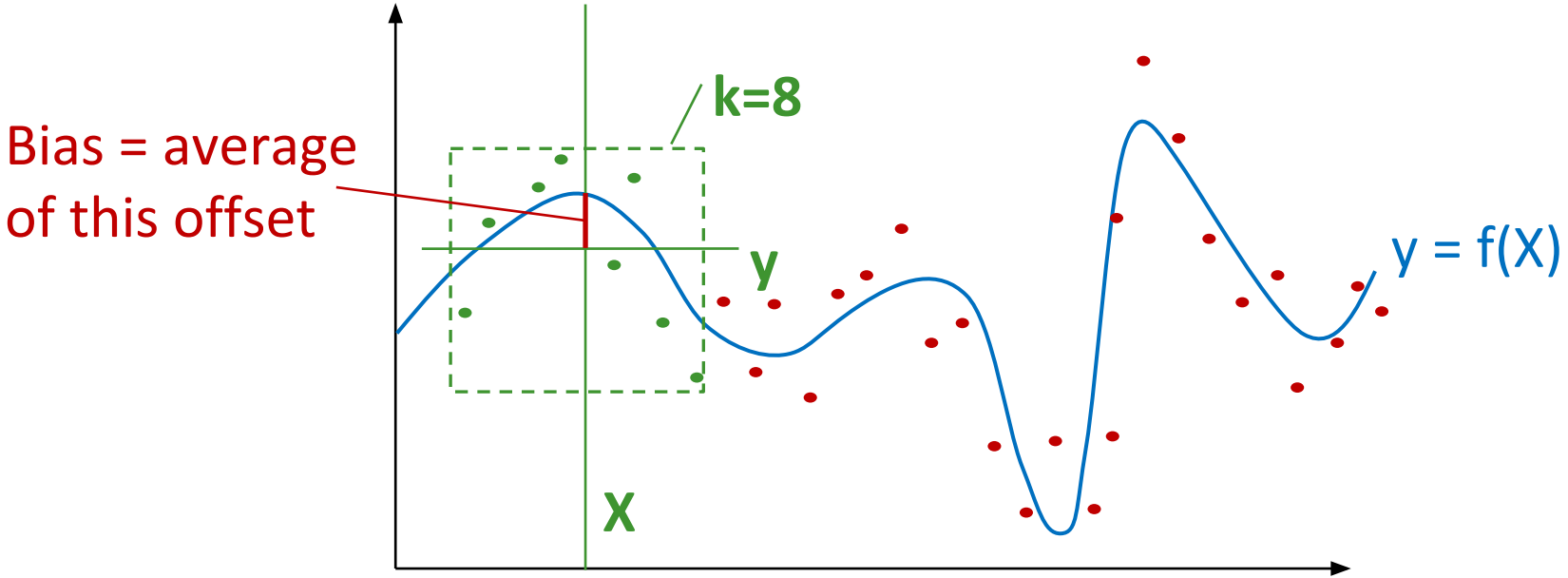




Choosing k



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



Choosing k in practice

Use leave-one-out cross-validation (LOO): 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 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: test on the test set to measure performance.

kNN and the curse of dimensionality

The curse of dimensionality refers to 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 kNN, that means there are fewer points that are very close in feature space (very similar), to the point we want to predict.

kNN 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 $\sqrt{N}$

kNN and the curse of dimensionality

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

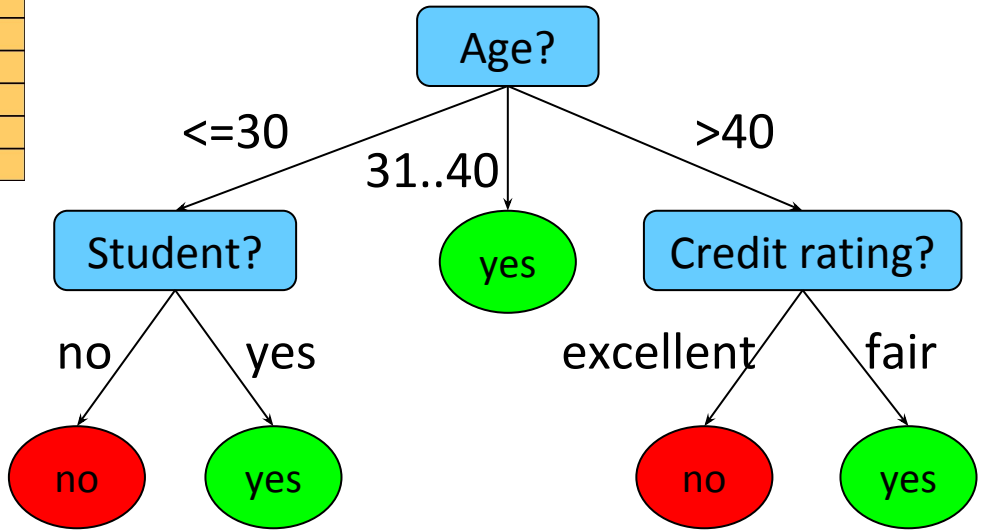
Luckily real data are not like random points in a high-dimensional cube. Instead they live in **dense clusters** and near **much lower-dimensional surfaces**.

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)

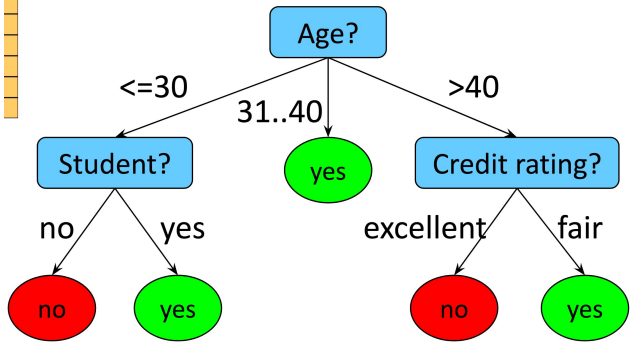
Decision Trees

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

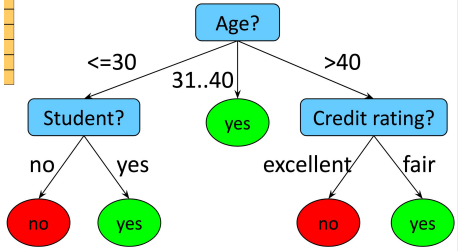
- Nodes are tests on a single attribute
- Branches are attribute values
- Leaves are marked with class labels

Score function: classification accuracy

Optimization:

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

Decision tree induction



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

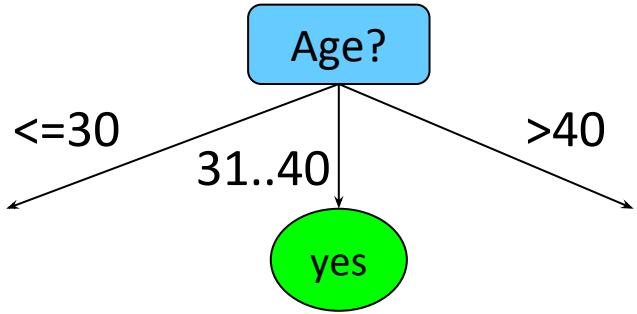
- At the beginning, all training samples belong to the root
- Examples are partitioned recursively based on selected “most discriminative” attributes
- Discriminative power based on information gain (ID3/C4.5) or Gini impurity (CART)

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

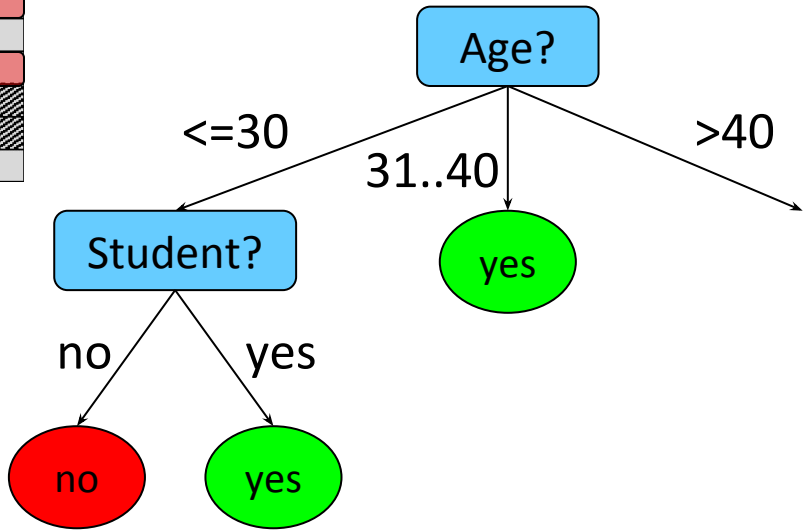
Decision tree induction

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



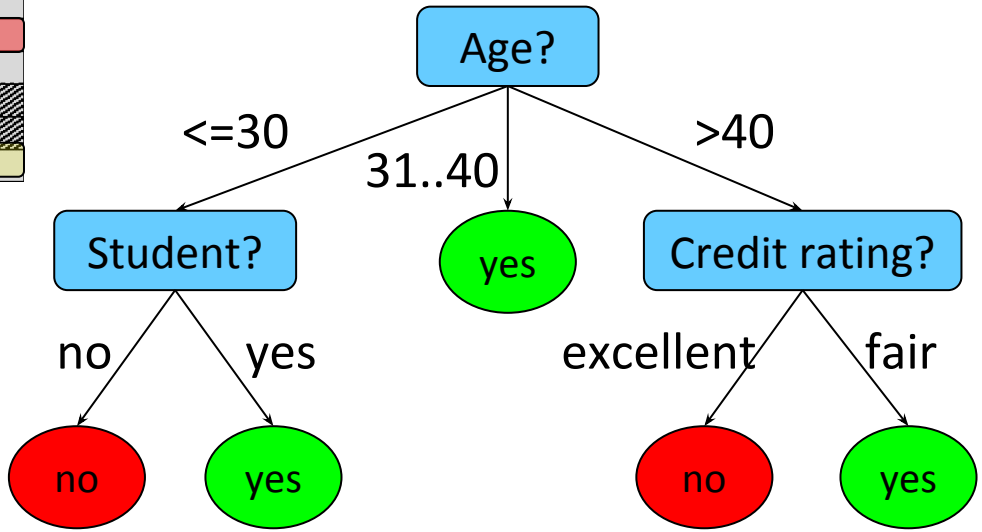
Decision tree induction

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 $\longrightarrow$ no uncertainty
- If P=N, H(P, N) = 1 $\longrightarrow$ max uncertainty

Attribute selection

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

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

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

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

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

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

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

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

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 **validation set (!)** does not get worse this way

Comments

Decision trees are just an example of classification algorithm

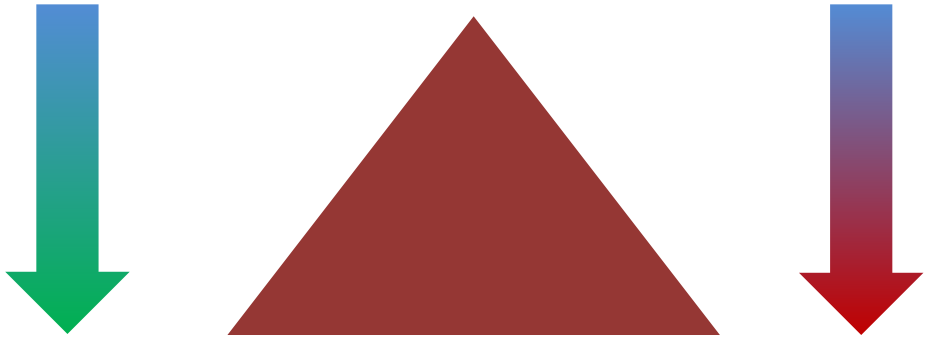
- Many other out there (Naive Bayes, SVM, neural networks, logistic regression, kNN, random forest ...)

Maybe not the best one ...

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

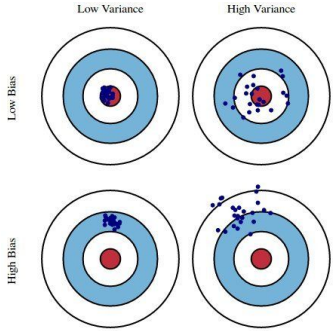
Decision Tree Models

- As tree depth increases, 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 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 (discrete output) or by averaging (continuous output).
- **Stacking:** combine model outputs using a second-stage learner like 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.
- 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. These 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.

Random Forests

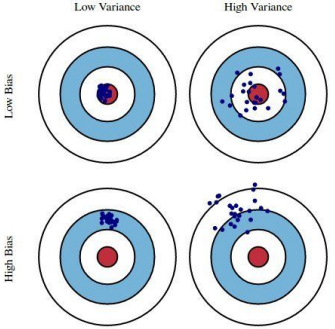
- **Very popular in practice,** probably the most popular classifier for dense data (up to a few thousand features)
- **Easy to implement** (simply train many normal decision trees)
- **Parallelizes easily**

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

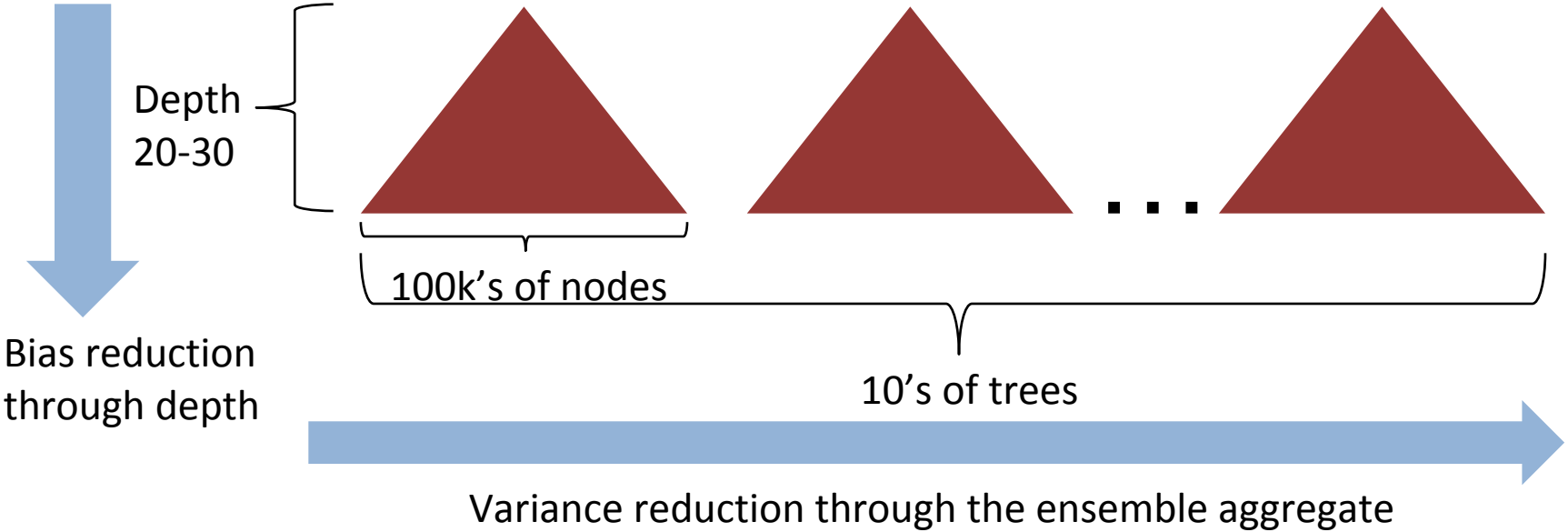
Boosted Decision Trees

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

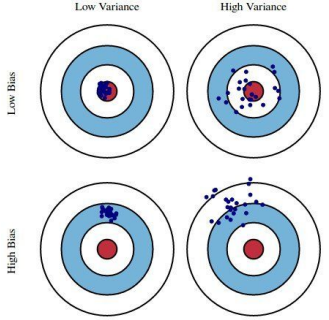
Random Forests vs. Boosted Trees



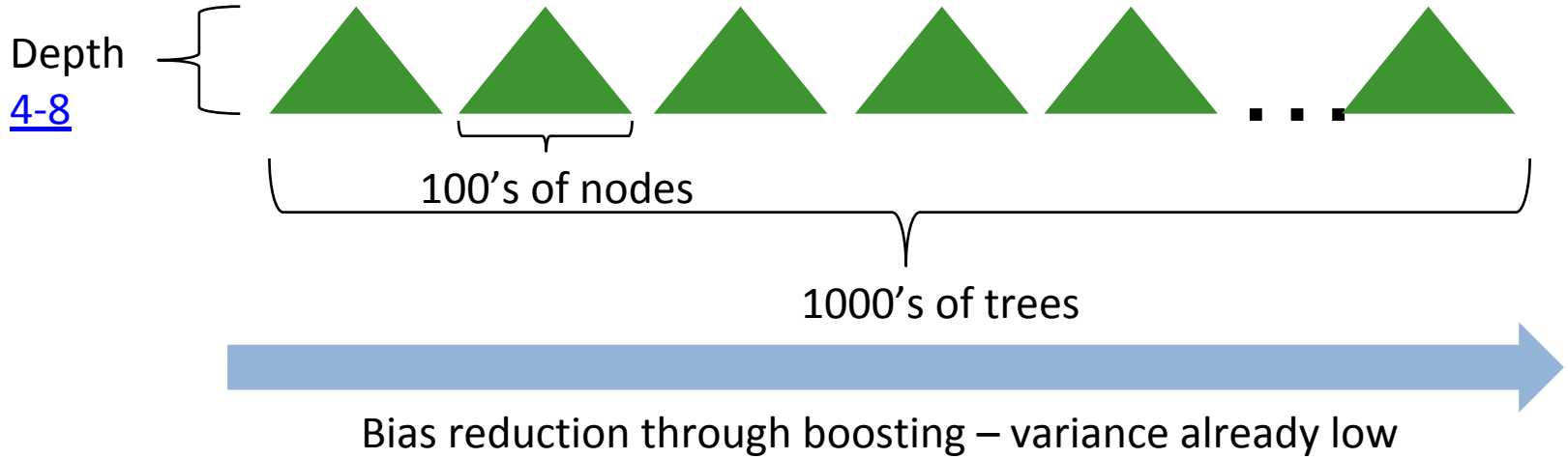
- The “geometry” of the methods is very different:
- Random forests use 10’s of deep, large trees:



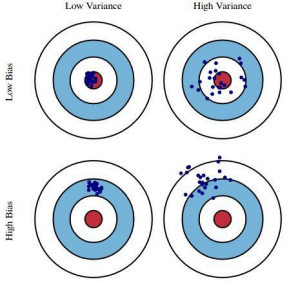
Random Forests vs Boosted Trees



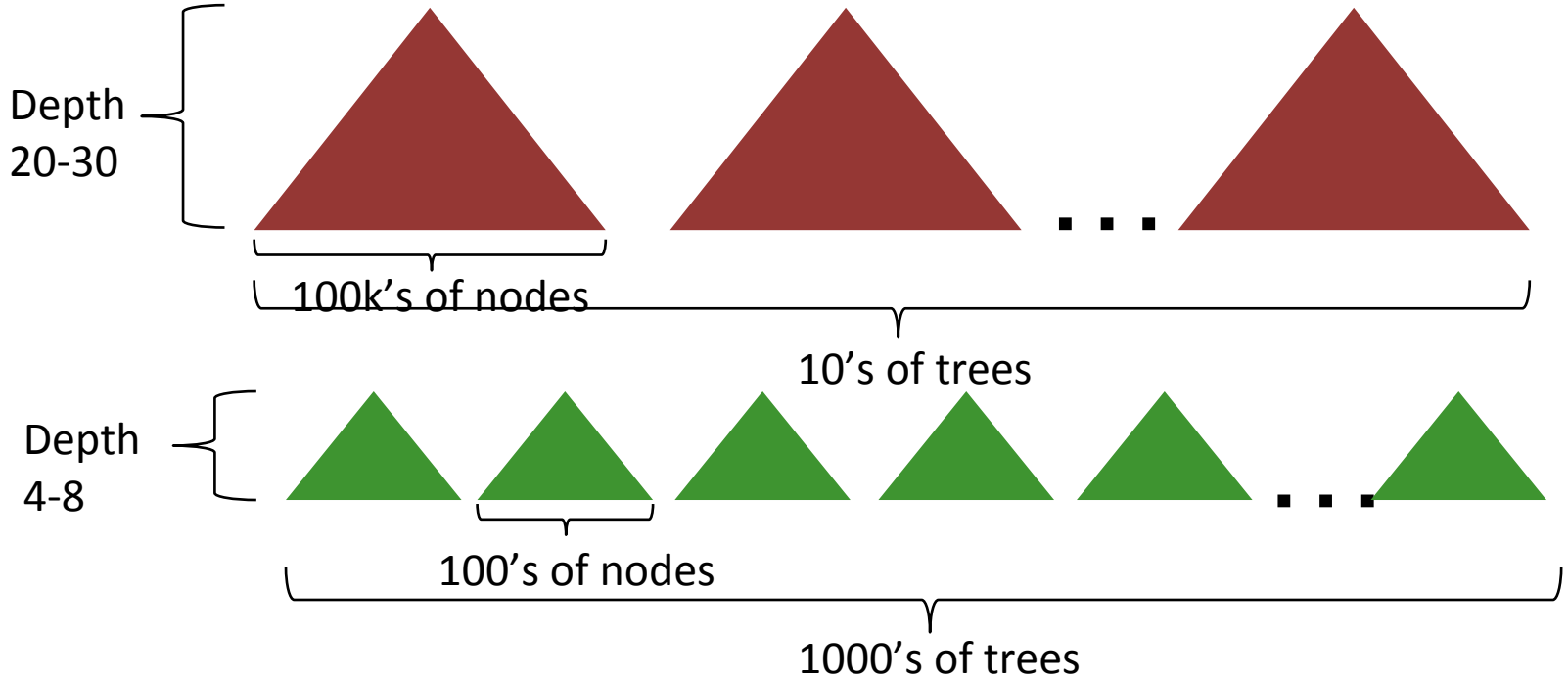
- The “geometry” of the methods is very different:
- Boosted decision trees use 1000’s of shallow, small trees:



Random Forests vs Boosted Trees



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



On model transparency

DNNs are often considered as “opaque” boxes

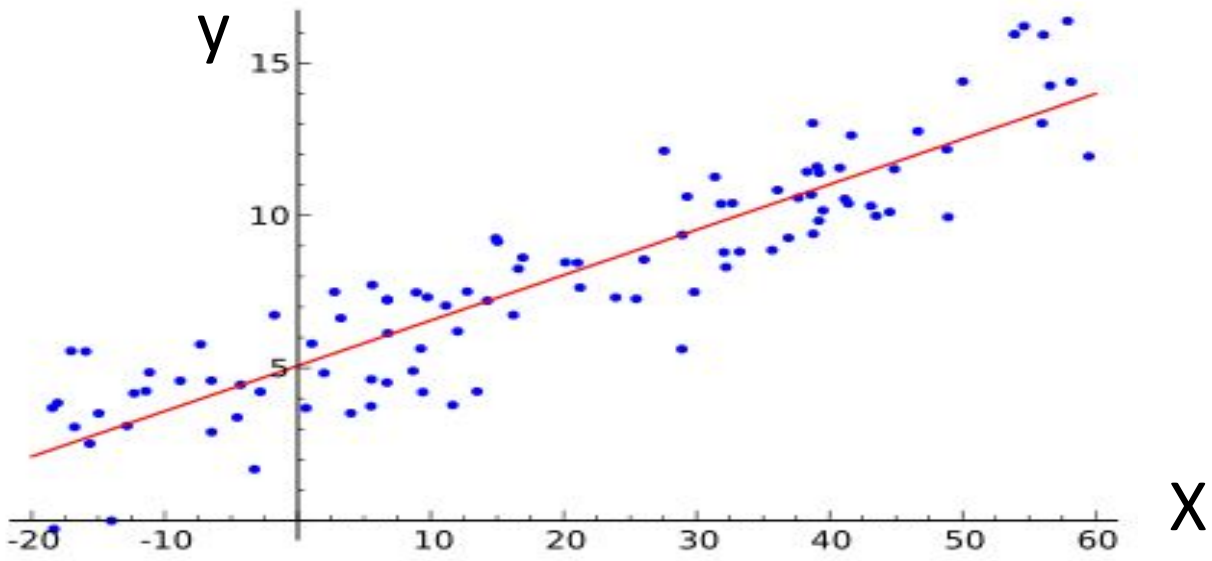
Do you think it’s easier to interpret a model with 1000s of trees?!

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

Linear and logistic regression

Linear Regression

We want to find the “best” line (linear function y=f(X)) to explain the data.

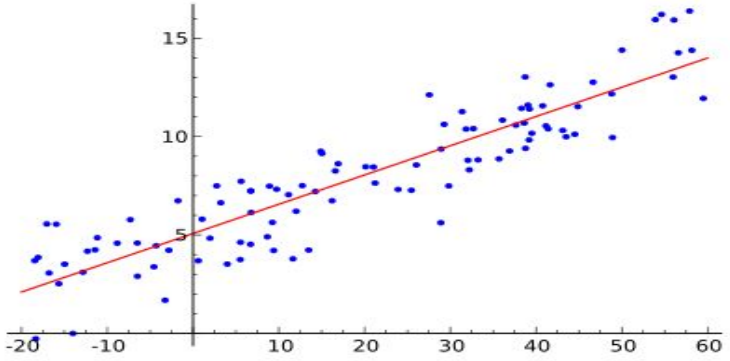


Linear Regression

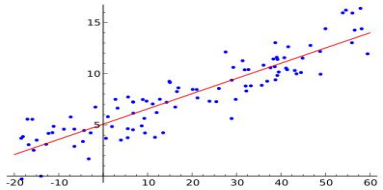
The predicted value of y is given by:

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

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



Least Squares Solution



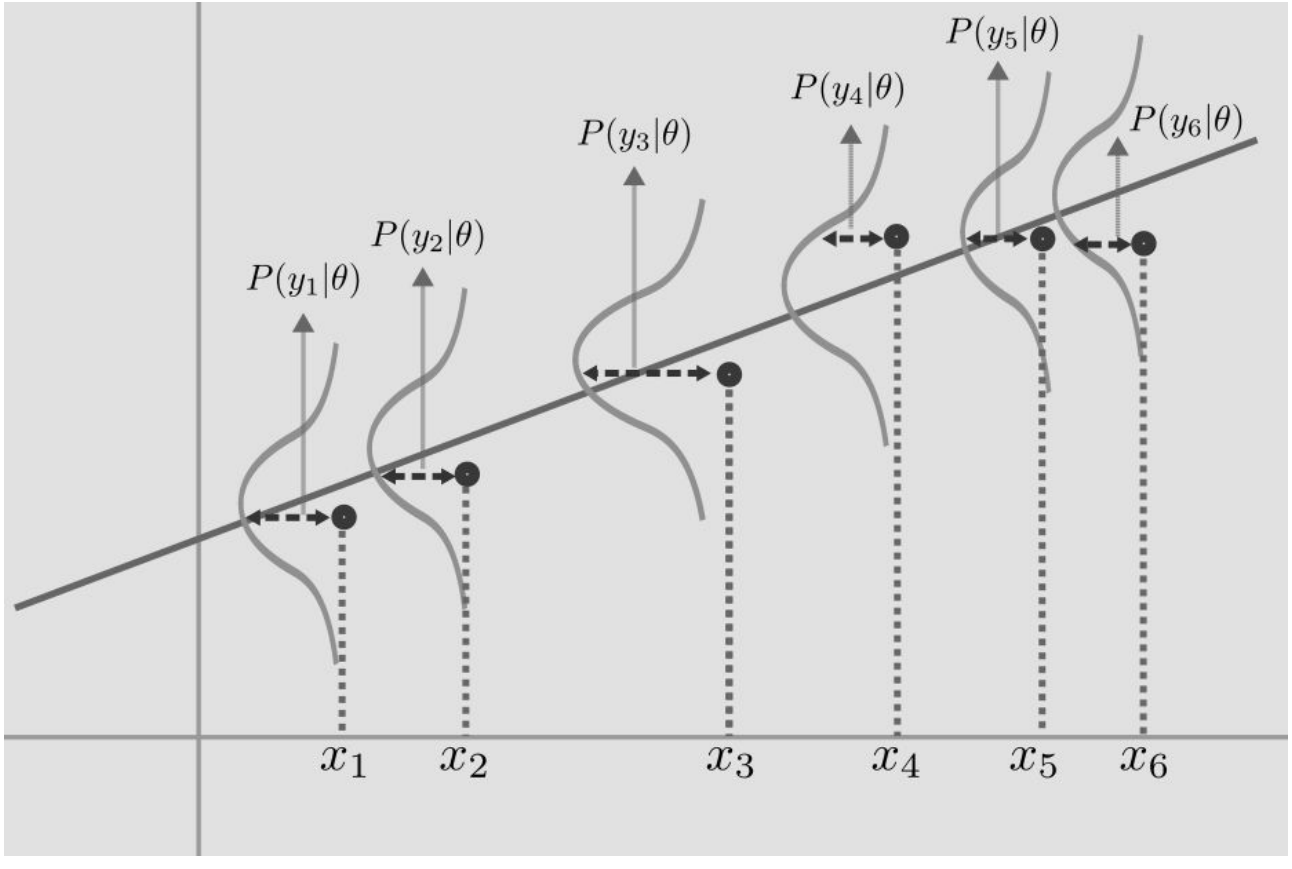
The most common measure of fit between the line and the data is the **least-squares fit**.

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 $\Pr(y_j)$.

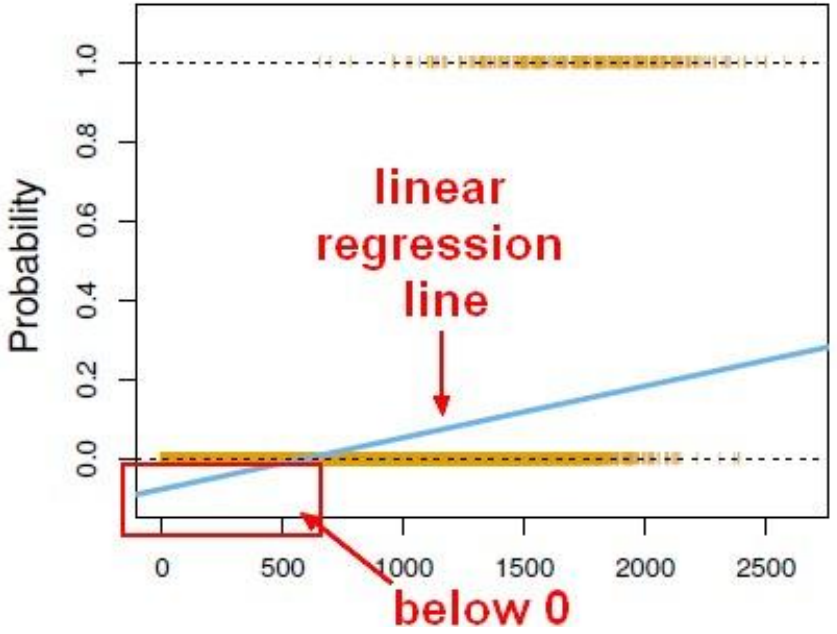
We can **easily maximize the log** of this expression $\frac{-(y_j - X_j\beta)^2}{2\sigma^2}$ for one point, or the sum of this expression at all points.

Least Squares Solution

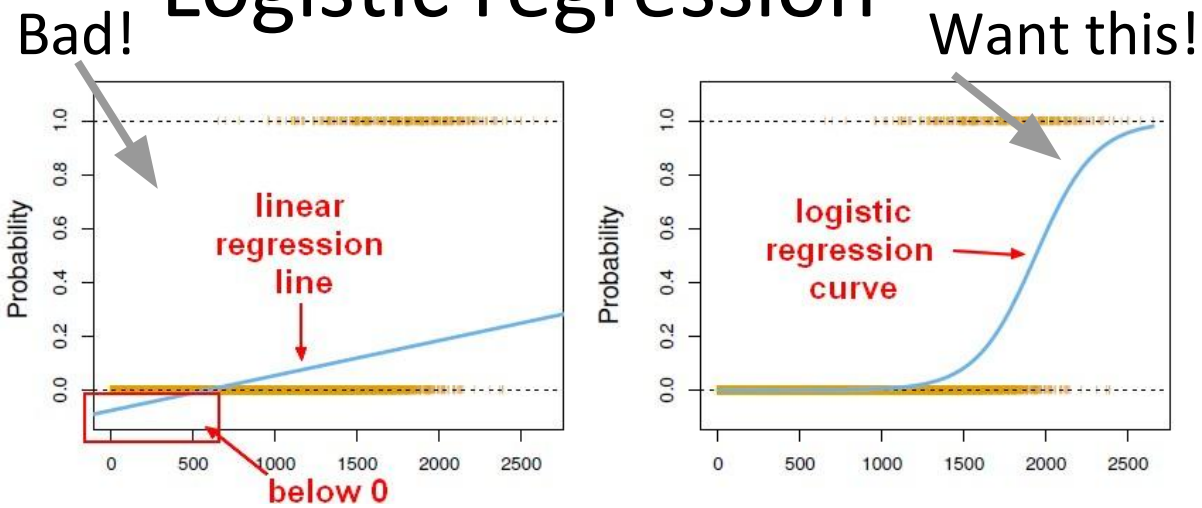


How to model binary events?

- 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



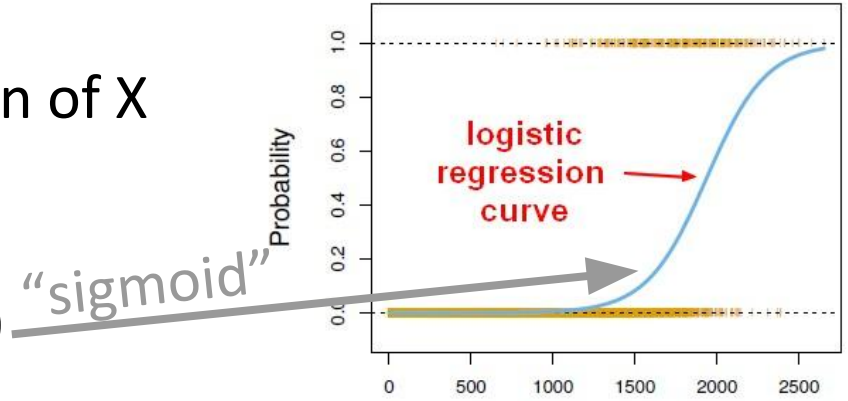
Logistic regression



- 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: $y = 1 / (1 + \exp(-\beta^T X))$
- Finding best model β via maximum likelihood:
  - Don't use square loss as in linear regression
  - Use cross-entropy loss instead



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)

Feedback

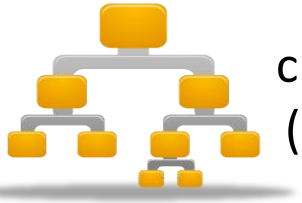
Give us feedback on this lecture here:
https://go.epfl.ch/ada2018-lec8-feedback

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

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”

Linear Regression

The regression formula

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

if $X_0 = 1$, can be written as a matrix product with X a row vector:

$$\hat{y} = \mathrm{X}\,\hat{\beta}$$

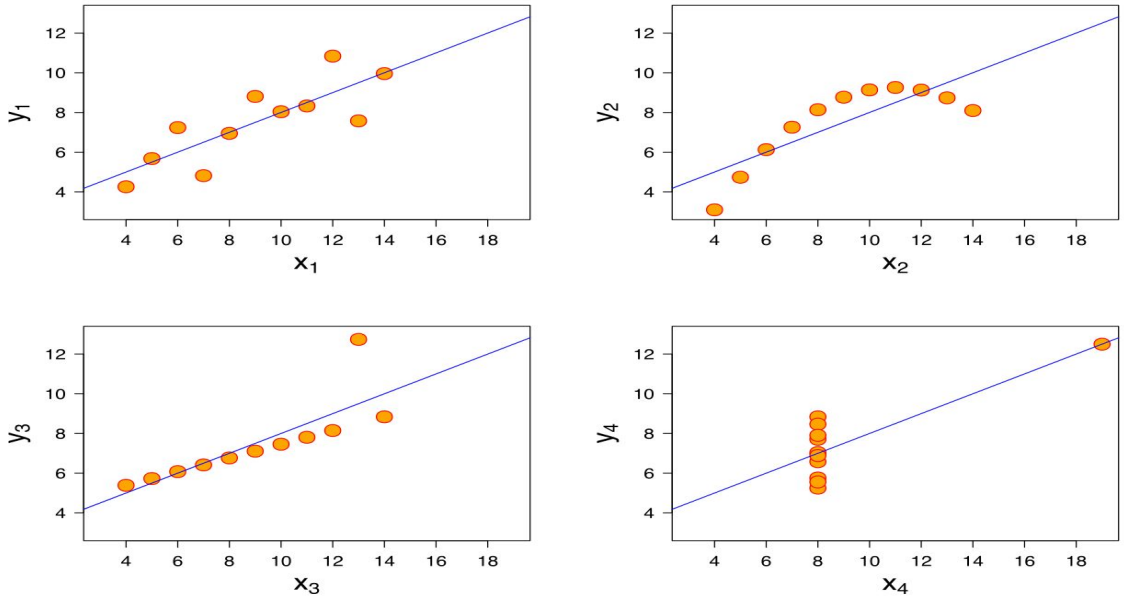
We get this by writing all of the input samples in a single matrix **X**:

i.e. **rows of X** $= \begin{pmatrix} X_{11} & \cdots & X_{1n} \\ \vdots & \ddots & \vdots \\ X_{m1} & \cdots & X_{mn} \end{pmatrix}$

are **distinct observations**, **columns of X** are **input features**.

R^2-values and P-values

We can **always** fit a linear model to any dataset, but how do we know if there is a **real linear relationship**?



R^2-values

Approach: Measure how much the total "noise" (variance) is reduced when we include the line as an offset.

R-squared: a suitable measure. Let $\hat{y} = \mathrm{X}\,\hat{\beta}$ be a predicted value, and $\bar{y}$ be the sample mean. Then the R-squared value is

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

And can be described as the fraction of the total variance explained by the model.

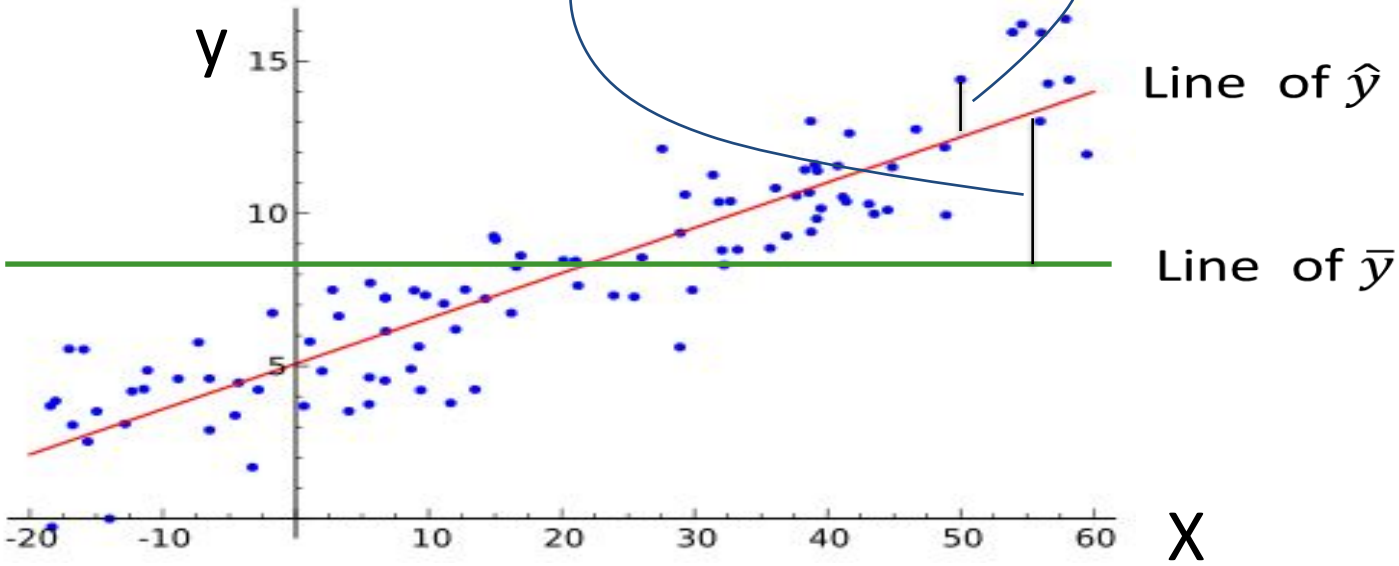
$R^2 = 0$: bad model. No evidence of a linear relationship.

$R^2 = 1$: good model. The line perfectly fits the data.

R-squared

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

Small if good fit



R^2-values and P-values

Statistic: From R-squared we can derive another statistic (using degrees of freedom) that has a standard distribution called an **F-distribution**.

From the CDF for the F-distribution, we can derive a **P-value** for the data.

The P-value is, as usual, the probability of observing the data under the null hypothesis of no linear relationship.

If **p is small**, say less than 0.05, we conclude that **there is a linear relationship**.

Presence of people (1)[1]	
Population density[1]	0.155**
Employees density[1]	0.328**
Deprivation	-0.022
Distance centre	-0.257**
Safety appearence	0.105**
Spatial Eigenvectors	11
Adj-R^2	0.91
Moran's I (p-value)	0.07 (0.08)

[1] log transformed variable.

Table 3: OLS regression model between presence of people and safety perception. The β coefficients are reported in the table. *p $<$ 0.01, **p $<$ 0.001.

Are Safer Looking Neighborhoods More Lively?

Ex. of correlation (and impact of each feature) explained via regression

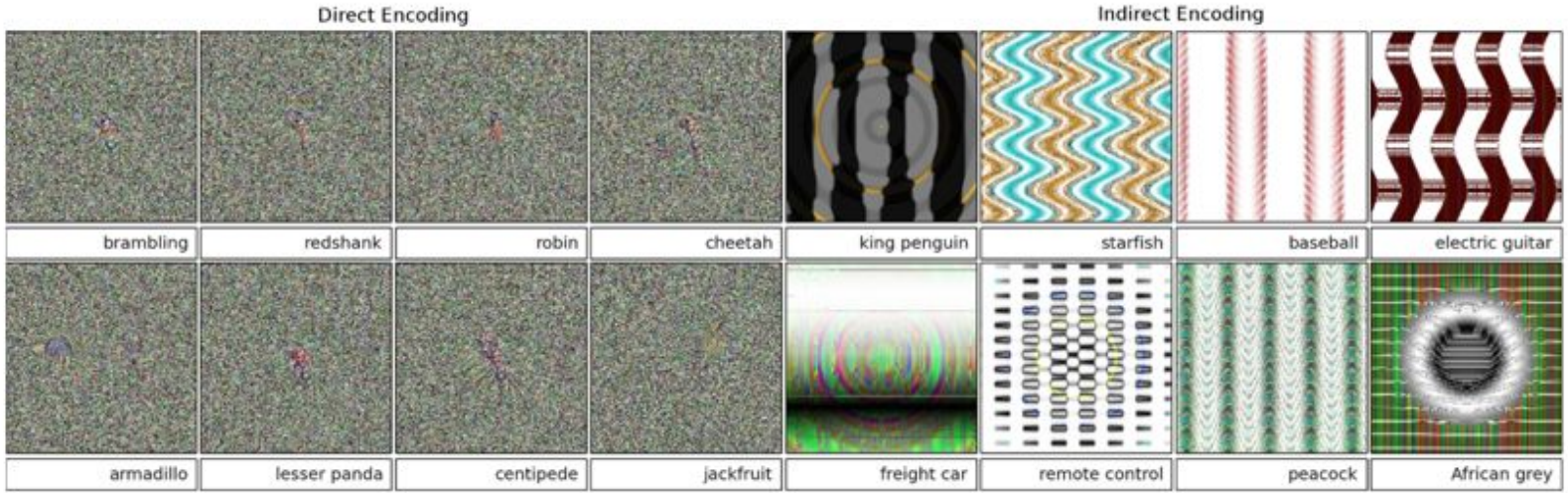


Figure 1: Evolved images that are unrecognizable to humans, but that state-of-the-art DNNs trained on ImageNet believe with >= 99.6% certainty to be a familiar object. This result highlights differences between how DNNs and humans recognize objects. Left: Directly encoded images. Right: Indirectly encoded images.