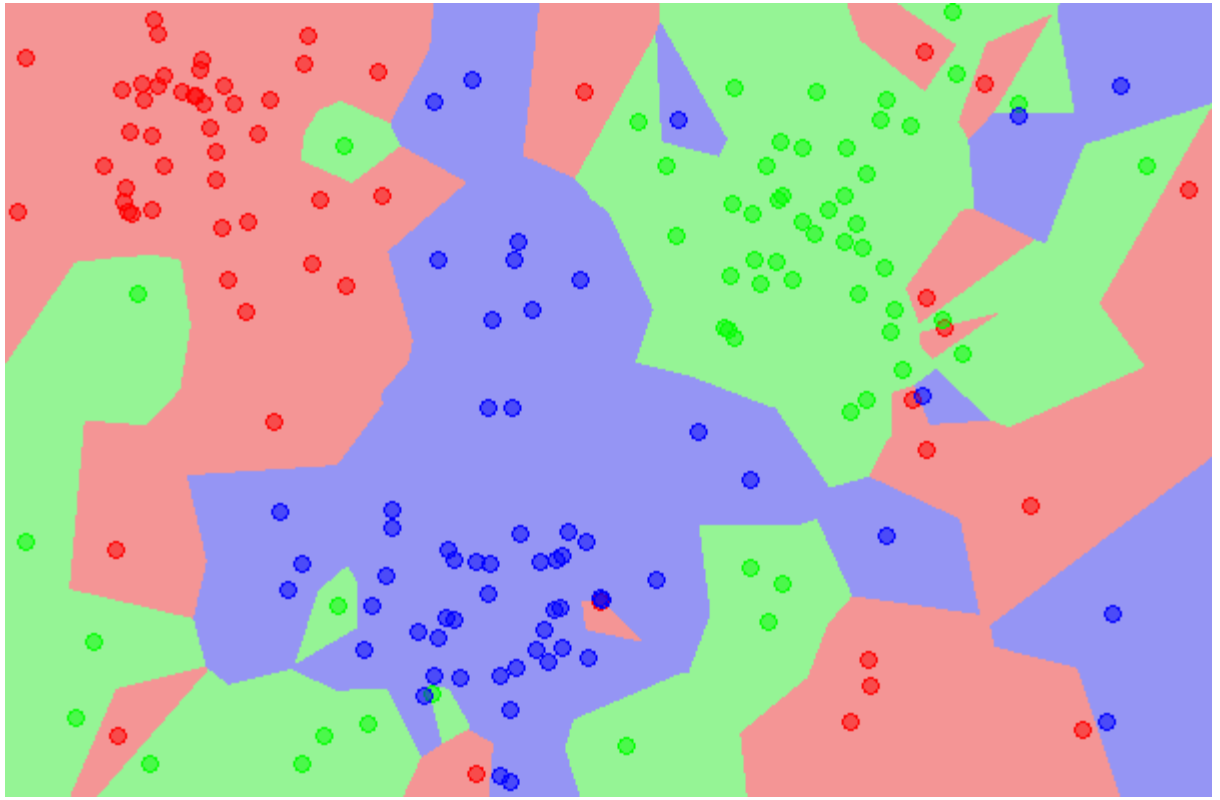
University of Ljubljana, Faculty of Computer and Information Science

Bias, variance and predictive models



Prof Dr Marko Robnik-Šikonja

Intelligent Systems, October 2018

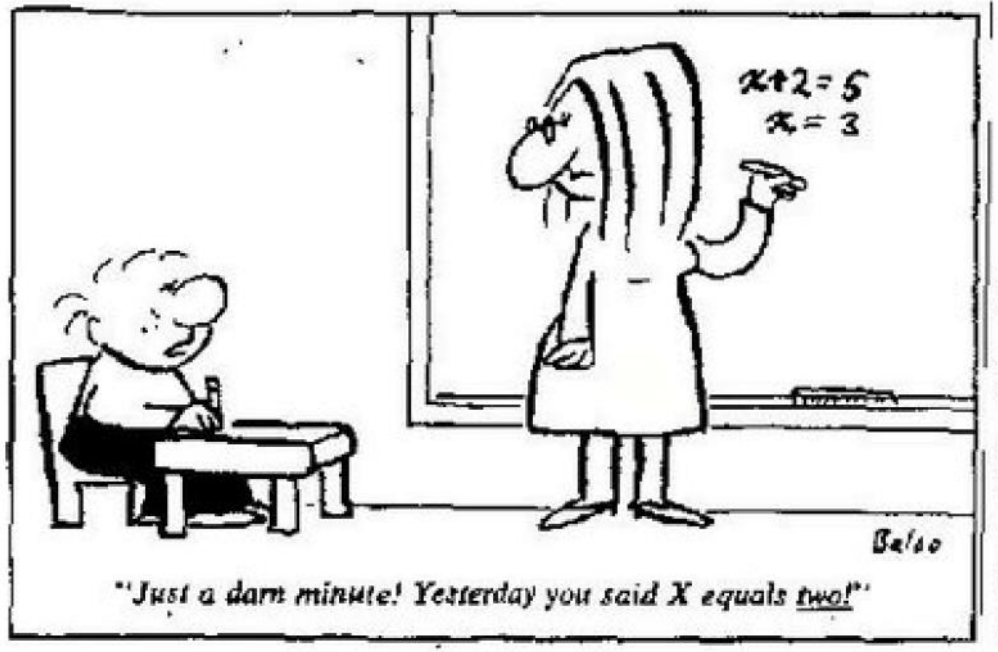
Contents

- Bias and variance of prediction models
- Simple regression models:
  - linear models, nearest neighbor, regression trees, regression rules.
- Simple classification models:
  - nearest neighbor, naïve Bayes, decision trees, decision rules, logistic regression.
- Biases in data

Assessing model accuracy

- Measuring the quality of fit
- The bias-variance trade-off
- Bias variance in the classification setting

Measuring the quality of fit



- Suppose we have a regression problem.

- One common measure of accuracy is the mean squared error (MSE), i.e.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Where $\hat{y}_i$ is the prediction our method gives for the observation in our training data.

A generalization problem

- Our method has generally been designed to make MSE small on the training data e.g. with linear regression we choose the line such that MSE is minimized.

- What we really care about is how well the method generalizes i.e. how well it works on new data. We call this new data "**Test data**".

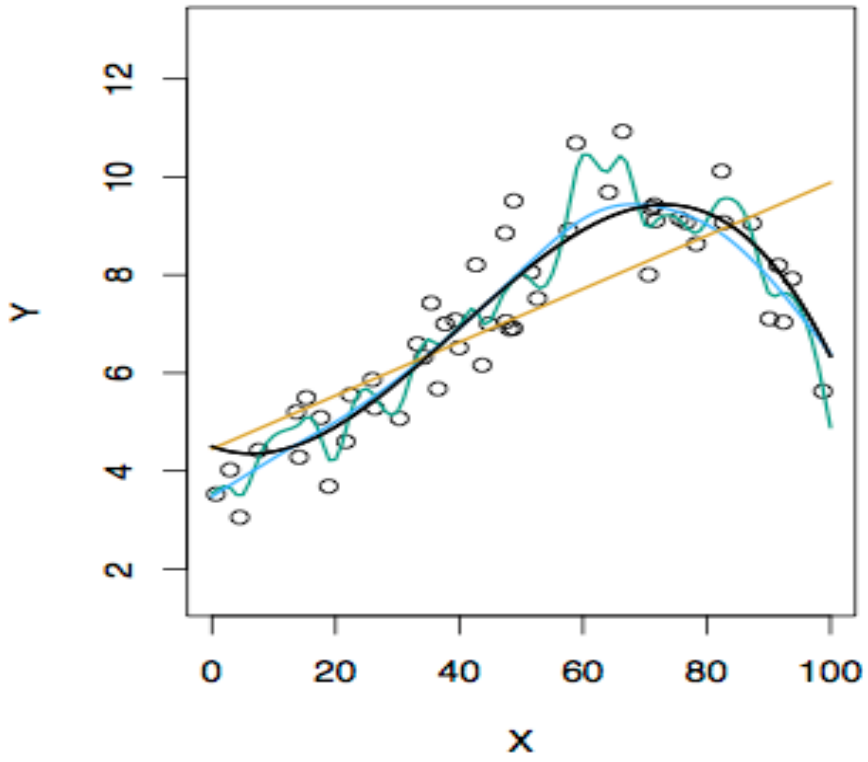
- There is no guarantee that the method with the smallest training MSE will have the smallest test (i.e. new data) MSE.

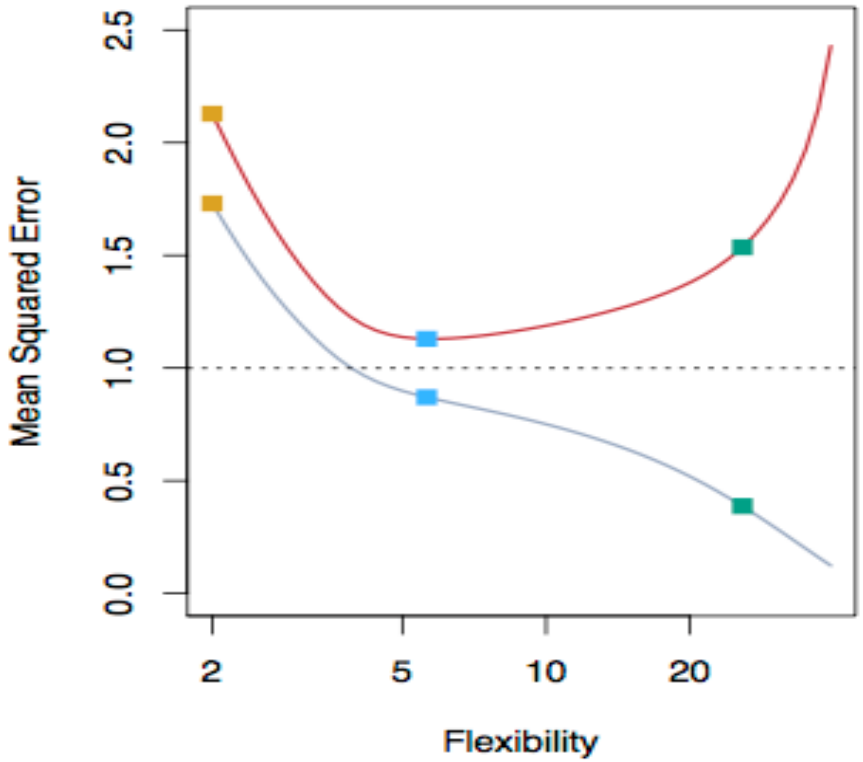
Training vs. test error

- In general the more flexible a method is the lower its training MSE will be, i.e. it will “fit” or explain the training data very well.
  - More Flexible methods (such as splines) can generate a wider range of possible shapes to estimate f as compared to less flexible and more restrictive methods (such as linear regression). The less flexible the method, the easier to interpret the model. Thus, there is a trade-off between flexibility and model interpretability.

- However, the test MSE may in fact be higher for a more flexible method than for a simple approach like linear regression.

Different levels of flexibility: example 1





LEFT

Black: Truth

Orange: Linear Estimate

Blue: smoothing spline

Green: smoothing spline (more flexible)

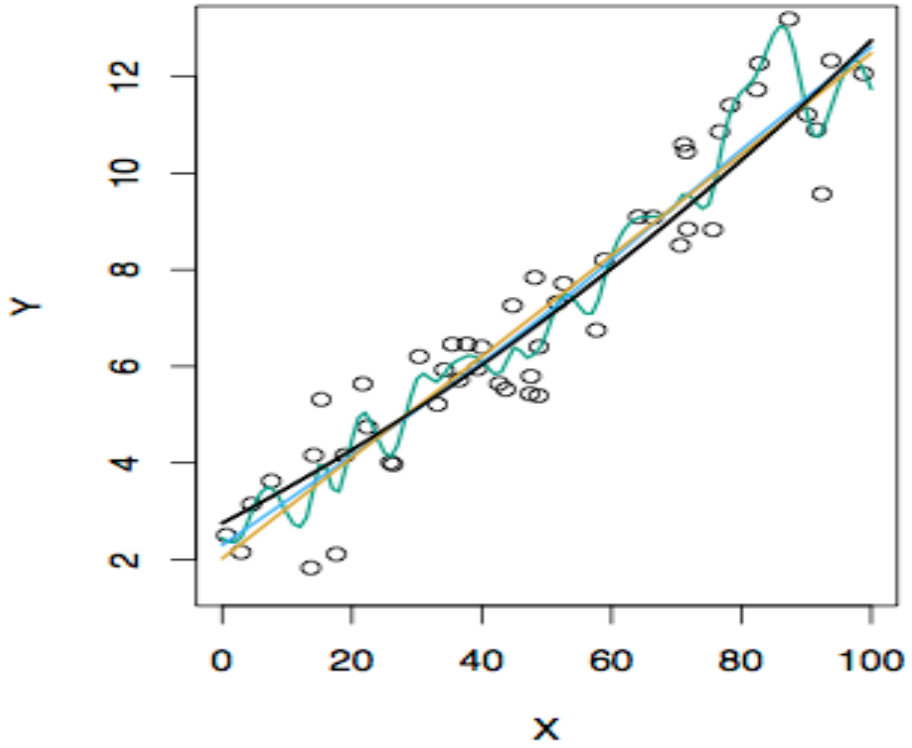
RIGHT

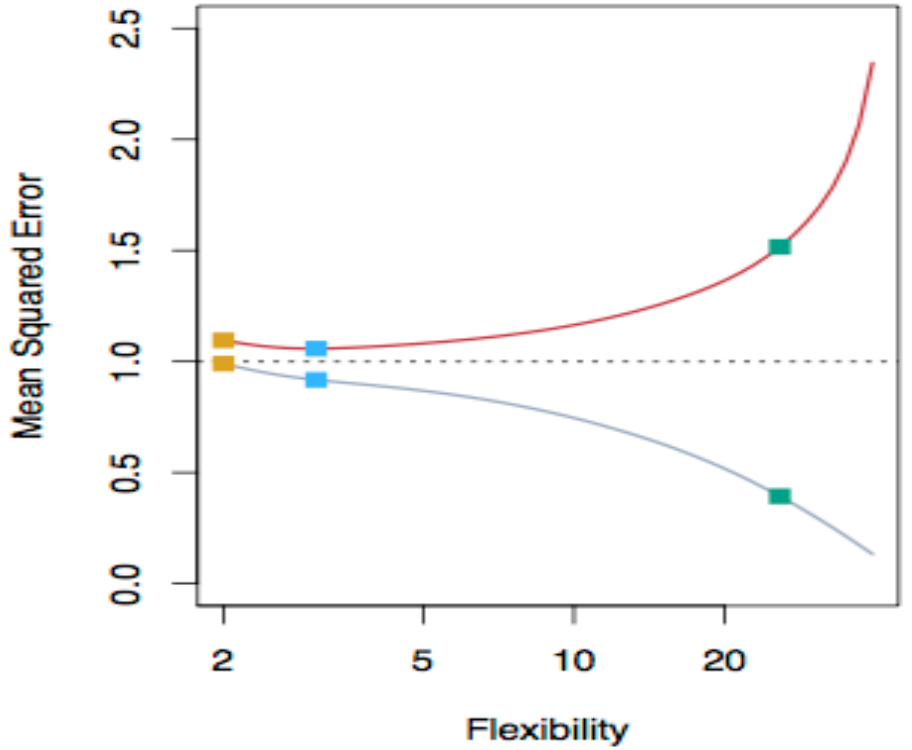
RED: Test MS

Grey: Training MSE

Dashed: Minimum possible test MSE (irreducible error)

Different levels of flexibility: example 2





LEFT

Black: Truth

Orange: Linear Estimate

Blue: smoothing spline

Green: smoothing spline (more flexible)

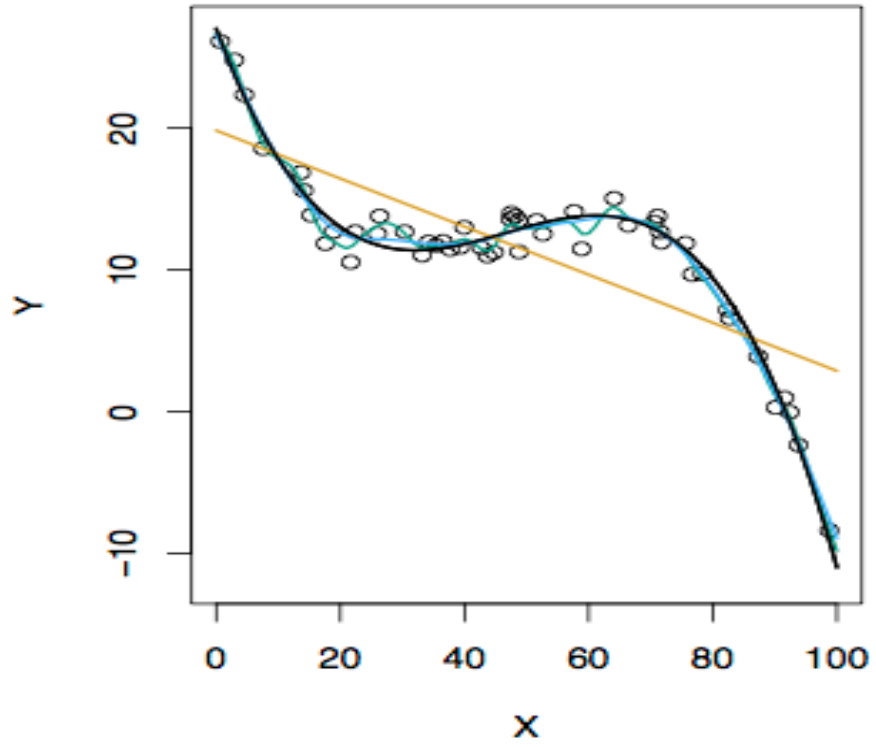
RIGHT

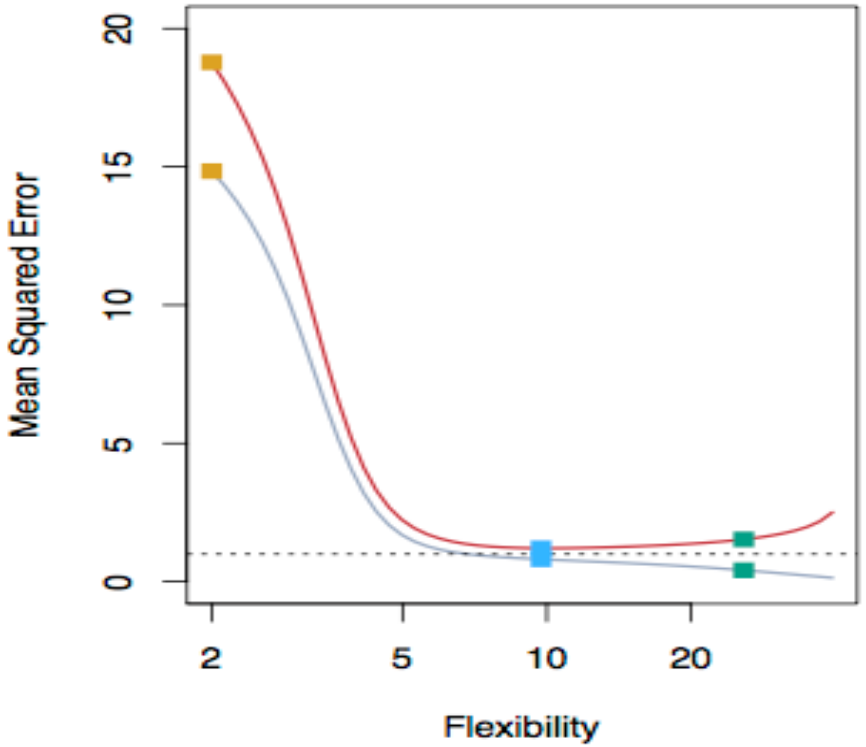
RED: Test MSE

Grey: Training MSE

Dashed: Minimum possible test MSE (irreducible error)

Different levels of flexibility: example 3





LEFT

Black: Truth

Orange: Linear Estimate

Blue: smoothing spline

Green: smoothing spline (more flexible)

RIGHT

RED: Test MSE

Grey: Training MSE

Dashed: Minimum possible test MSE (irreducible error)

Bias - variance trade-off

- The previous graphs of test versus training MSE's illustrates a very important tradeoff that governs the choice of statistical learning methods.

- There are always two competing forces that govern the choice of learning method, i.e. bias and variance.

Bias of learning methods

- Bias refers to the error that is introduced by modeling a real life problem (that is usually extremely complicated) by a much simpler problem.

- For example, linear regression assumes that there is a linear relationship between Y and X. It is unlikely that, in real life, the relationship is exactly linear so some bias will be present.

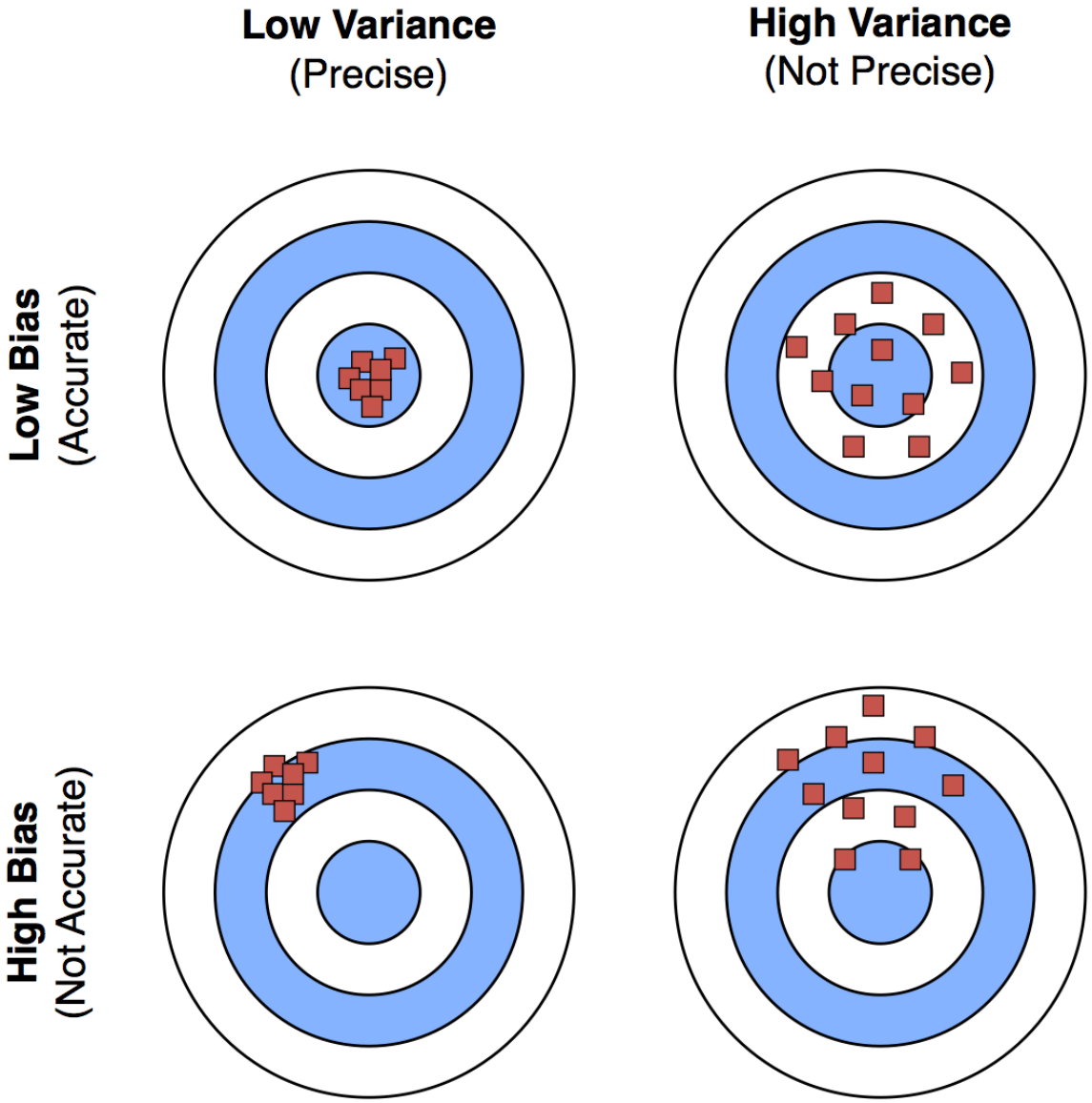
- The more flexible/complex a method is the less bias it will generally have.

Variance of learning methods

- Variance refers to how much your estimate for f would change if you had a different training data set.

- Generally, the more flexible a method is the more variance it has.

Bias-variance illustration



The trade-off

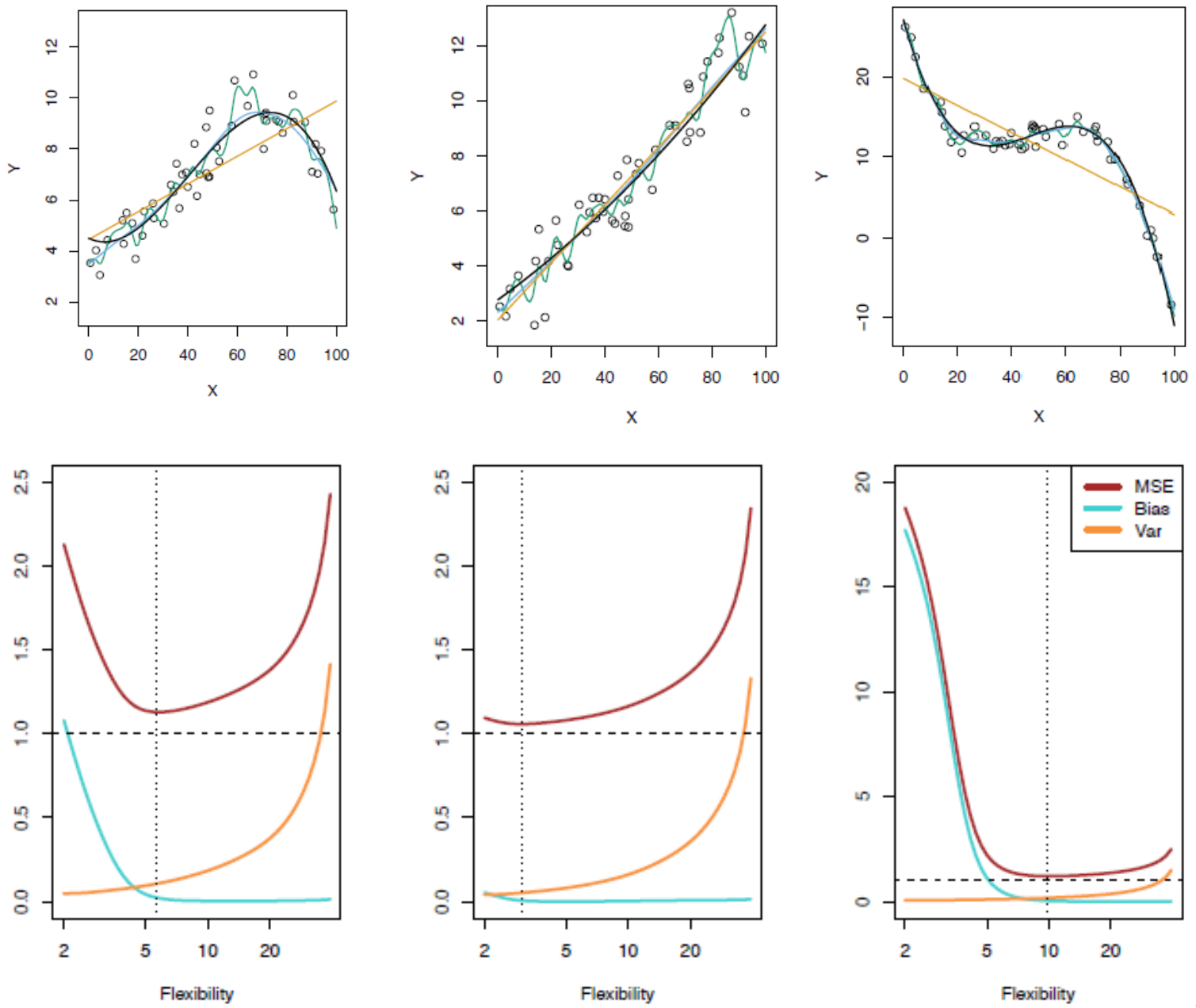
- It can be shown that for any given, $X=x_0$, the expected test MSE for a new Y at x_0 will be equal to

$$Expected\,Test\,MSE = E\left(Y - f(x_0)\right)^2 = Bias^2 + Var + \underbrace{s^2}_{\text{Irreducible Error}}$$

where Bias = E[Y]-f(x) and Var = $E[(Y - E[Y])^2]$

- What this means is that as a method gets more complex the bias will decrease and the variance will increase but expected test MSE may go up or down!

Test MSE, bias and variance



The classification setting

- For a regression problem, we used the MSE to assess the accuracy of the statistical learning method
- For a classification problem we can use the error rate i.e.

$$Error\ Rate = \sum_{i=1}^{n} I(y_i \neq \hat{y}_i)/n$$

- $I(y_i \neq \hat{y}_i)$ is an indicator function, which will give 1 if the condition $(y_i \neq \hat{y}_i)$ is correct, otherwise it gives a 0.

- Thus the error rate represents the fraction of incorrect classifications, or misclassifications

Bayes error rate

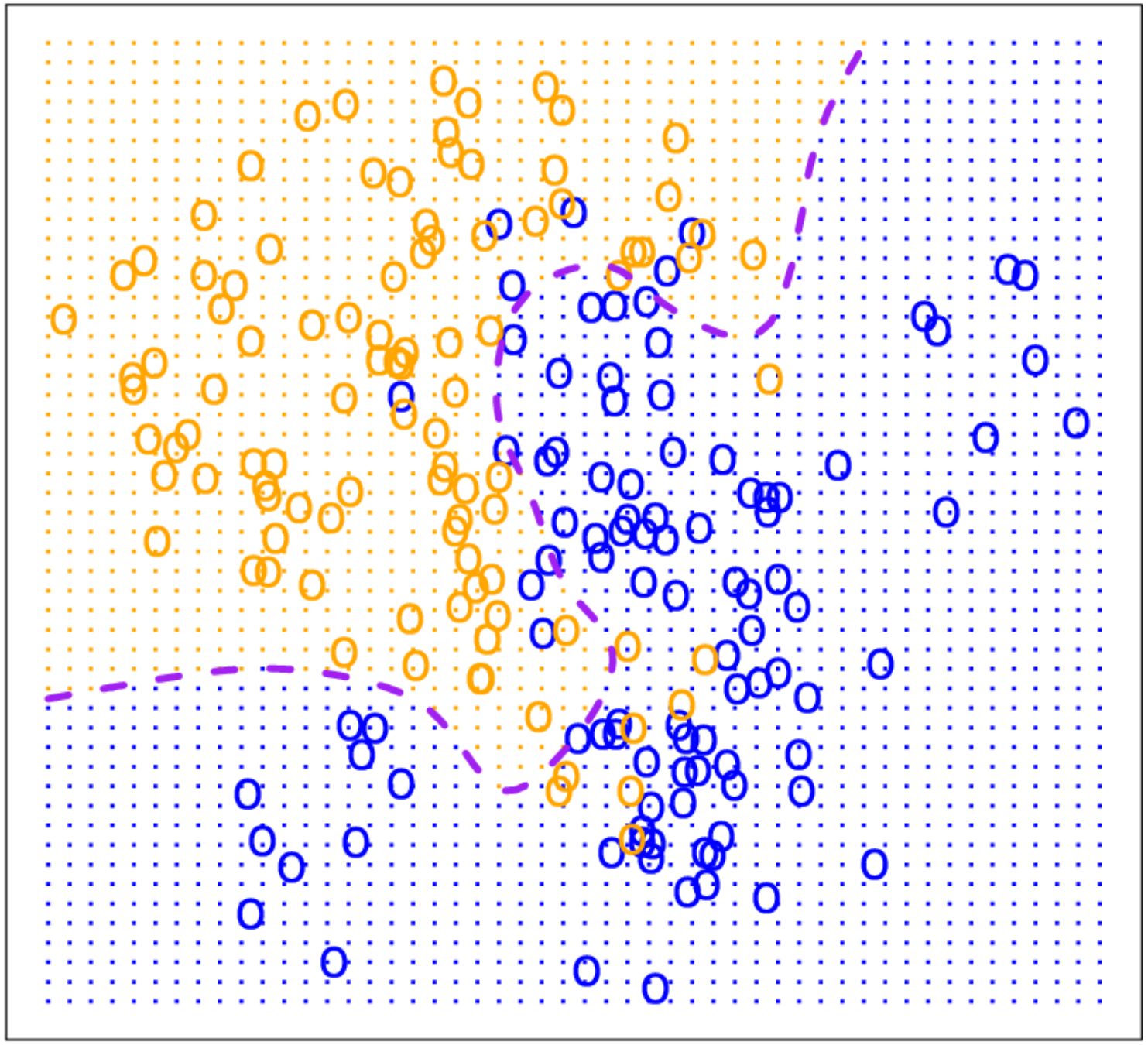
- The Bayes error rate refers to the lowest possible error rate that could be achieved if somehow we knew exactly what the “true” probability distribution of the data looked like.

- On test data, no classifier (or stat. learning method) can get lower error rates than the Bayes error rate.

- Of course in real life problems the Bayes error rate can’t be calculated exactly.

Bayes optimal classifier

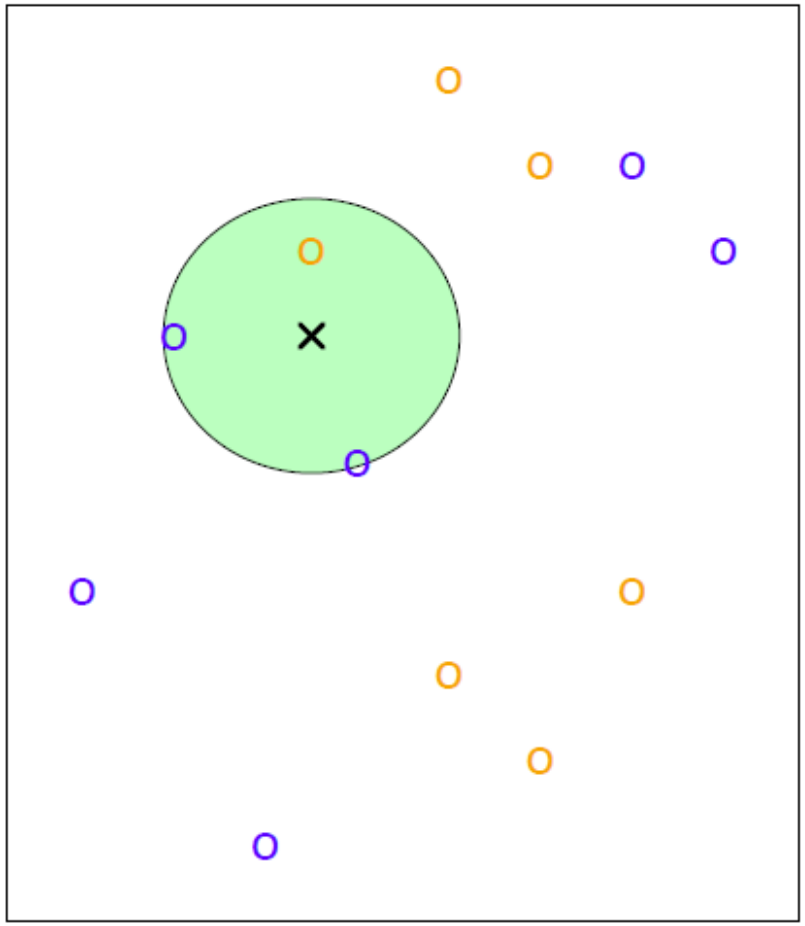
- for new x_0 returns the maximally probable prediction value $P(Y=y \mid X=x_0)$
- in classification $\arg\max_j P(Y=y_j \mid X=x_0)$

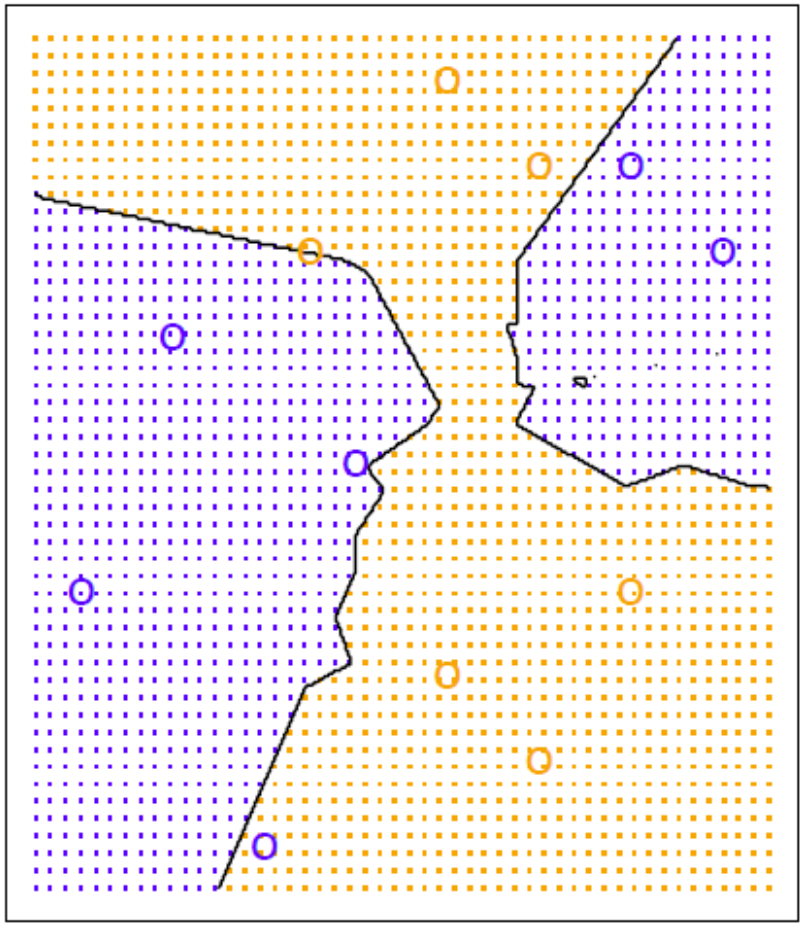


K-Nearest Neighbors (KNN)

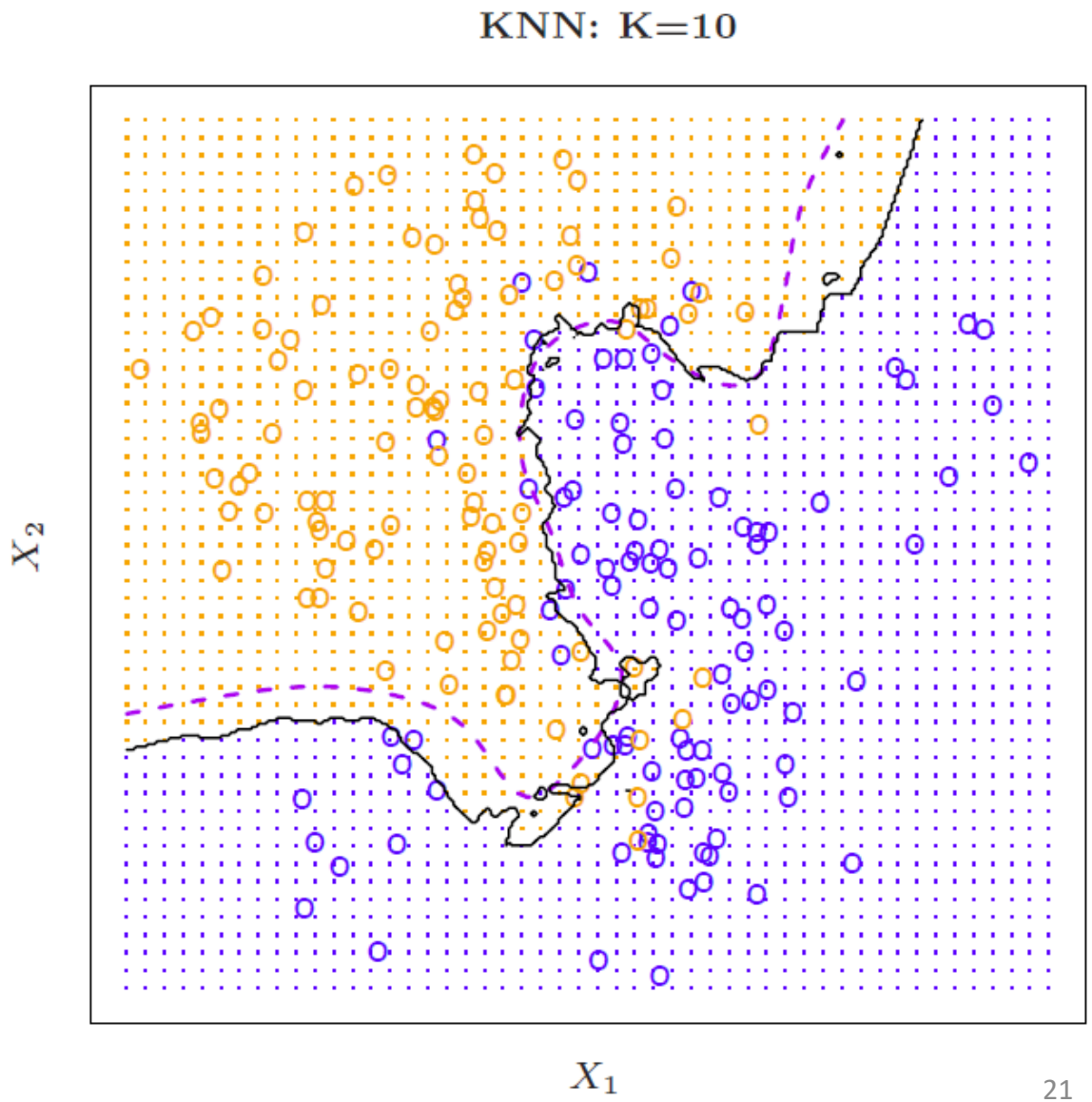
- k Nearest Neighbors is a flexible approach to estimate the Bayes classifier.
- For any given X we find the k closest neighbors to X in the training data, and examine their corresponding Y.
- If the majority of the Y's are orange we predict orange otherwise guess blue.
- The smaller that k is the more flexible the method will be.

KNN example with k = 3

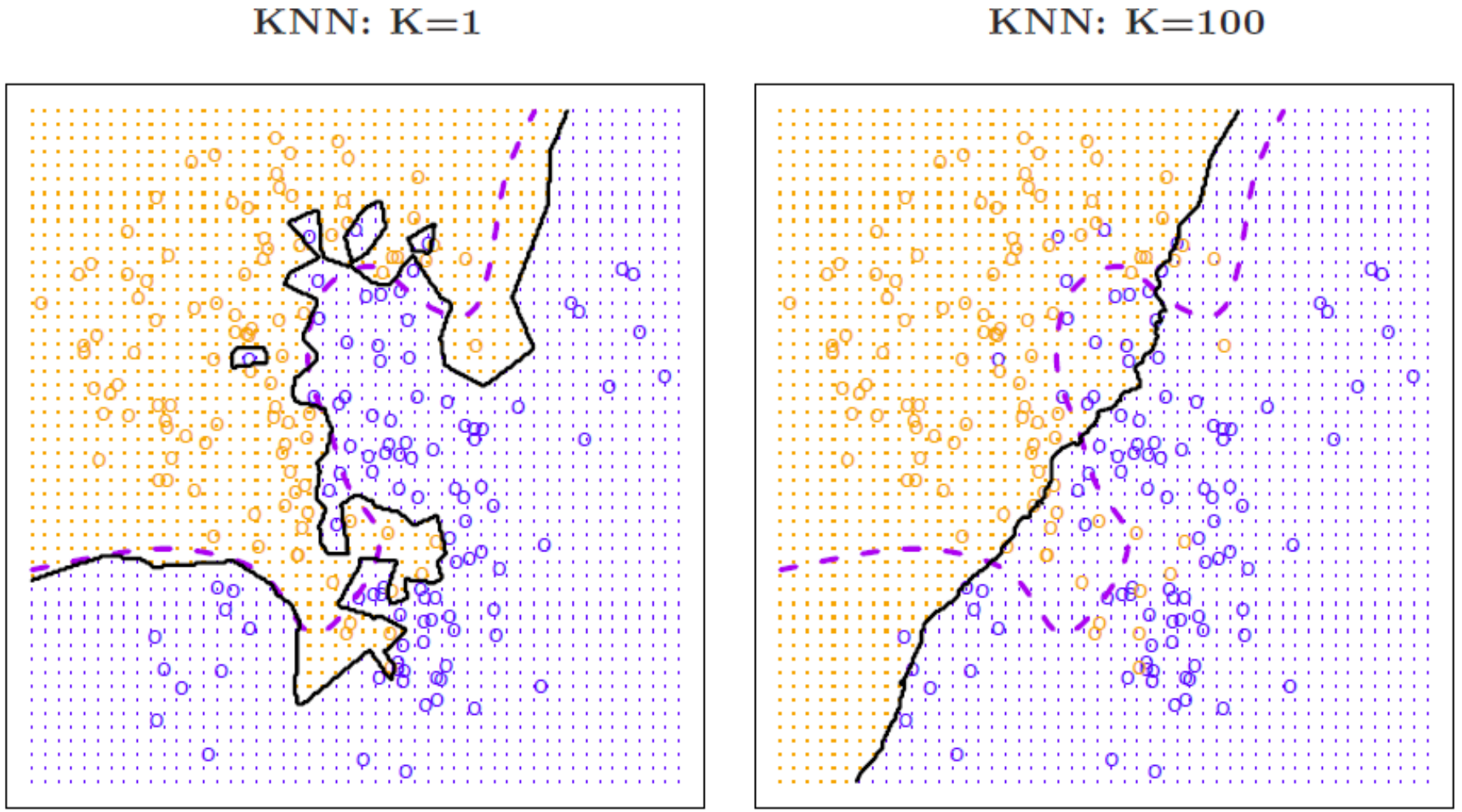




Simulated data:
K = 10

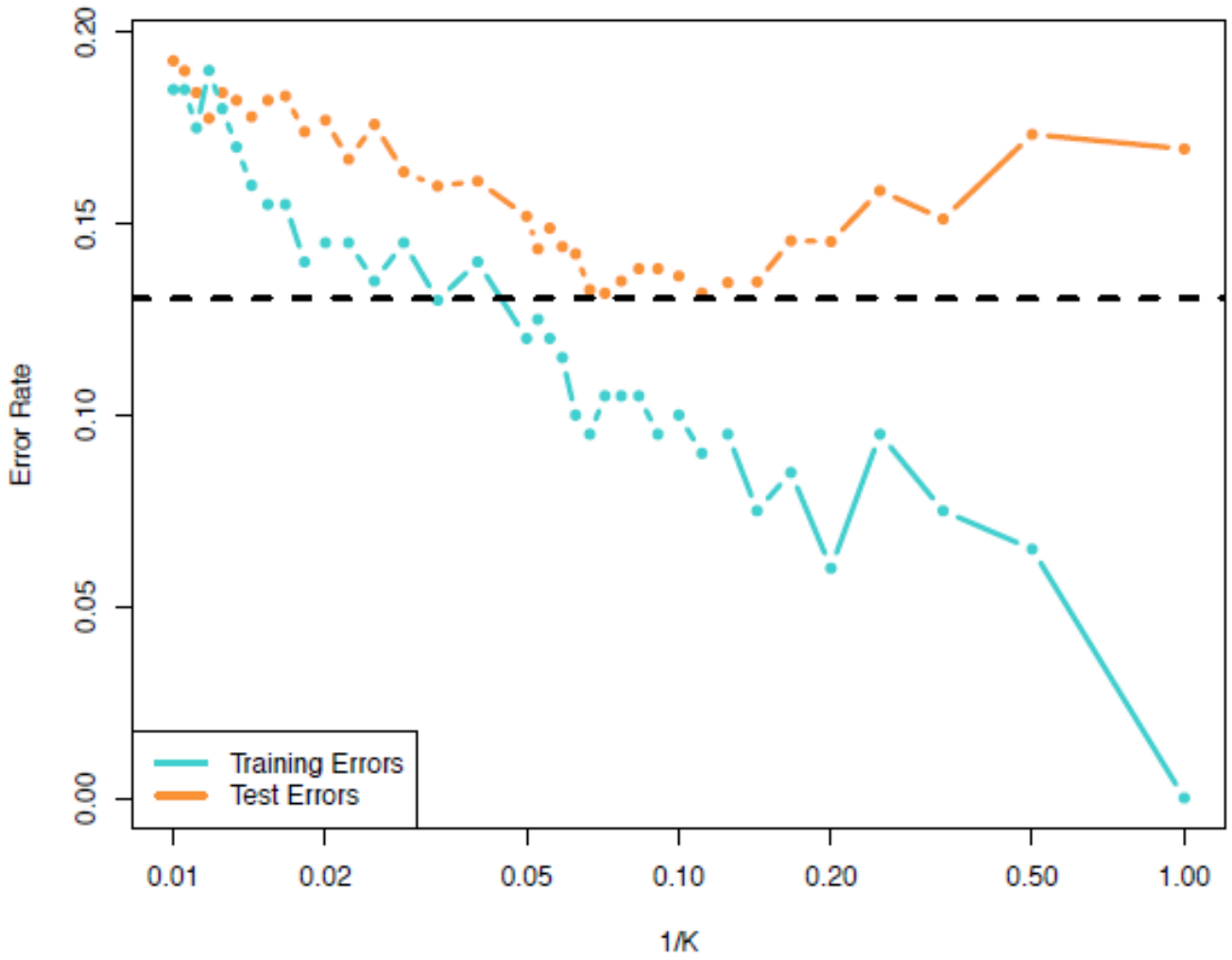


K = 1 and K = 100



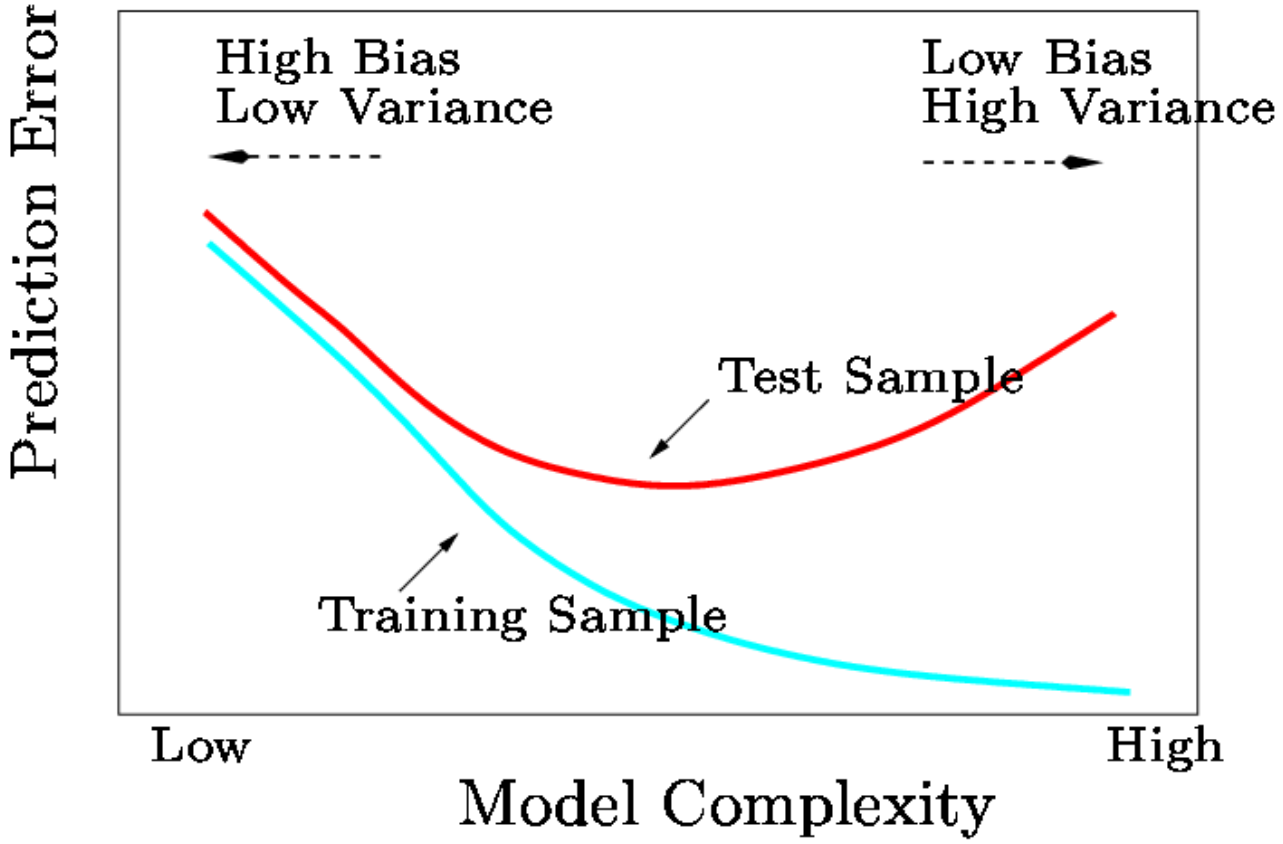
Training vs. test error rates on the simulated data

- Notice that training error rates keep going down as k decreases or equivalently as the flexibility increases.
- However, the test error rate at first decreases but then starts to increase again.



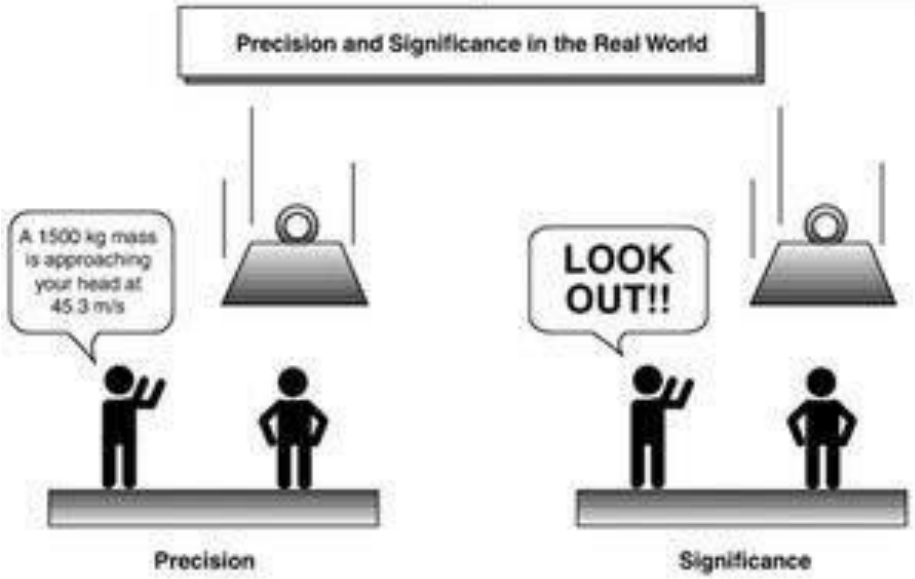
A fundamental picture

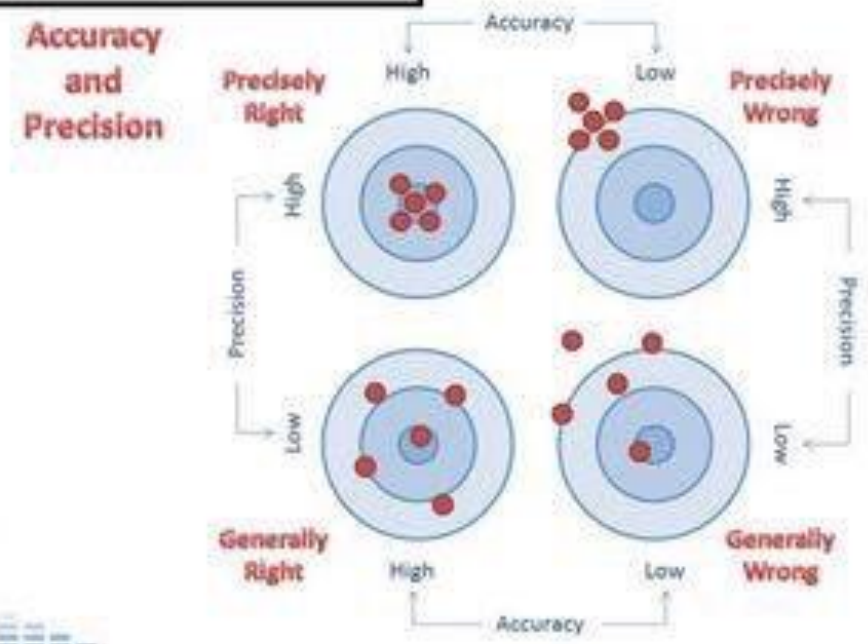
- In general training errors will always decline.
- However, test errors will decline at first (as reductions in bias dominate) but will then start to increase again (as increases in variance dominate).

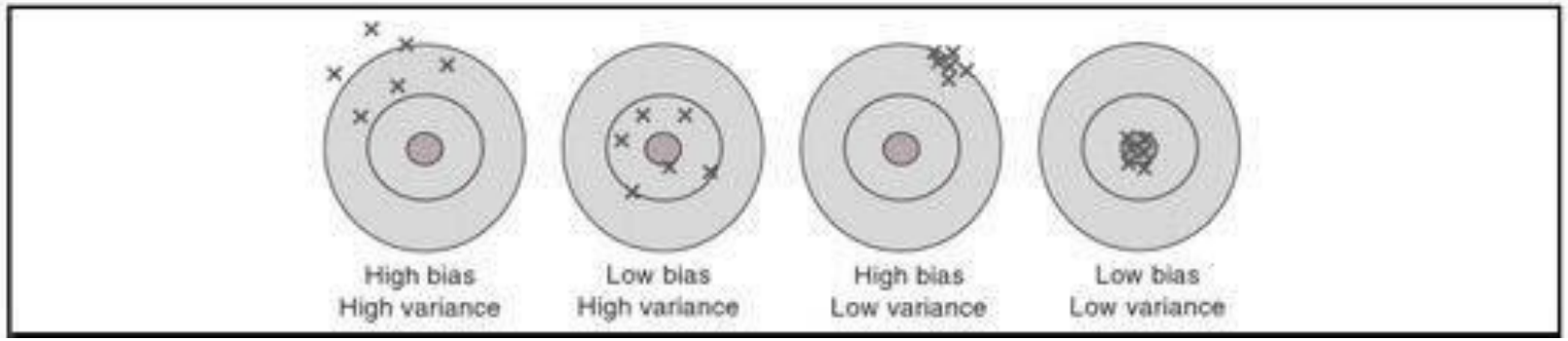


We must always keep this picture in mind when choosing a learning method. More flexible/complicated is not always better!

Precision vs. Significance
Accuracy vs. Precision
Bias vs. Variance







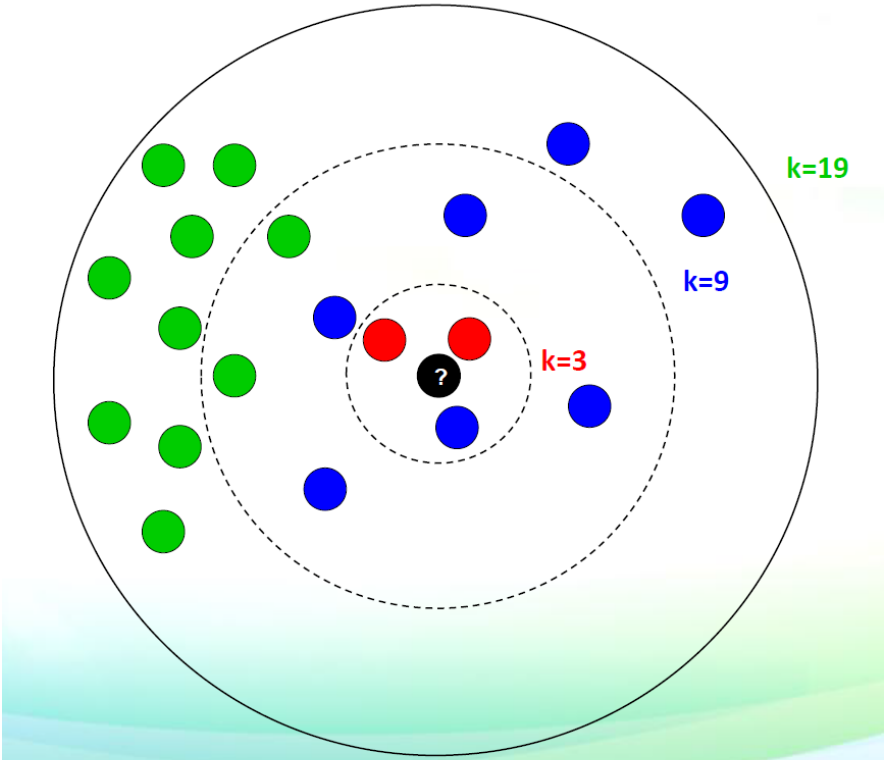
K-nearest neighbor for regression

- kNN regression is similar to the kNN classifier.
- To predict Y for a given value of X, consider k closest points to X in training data and take the average of the responses. i.e.

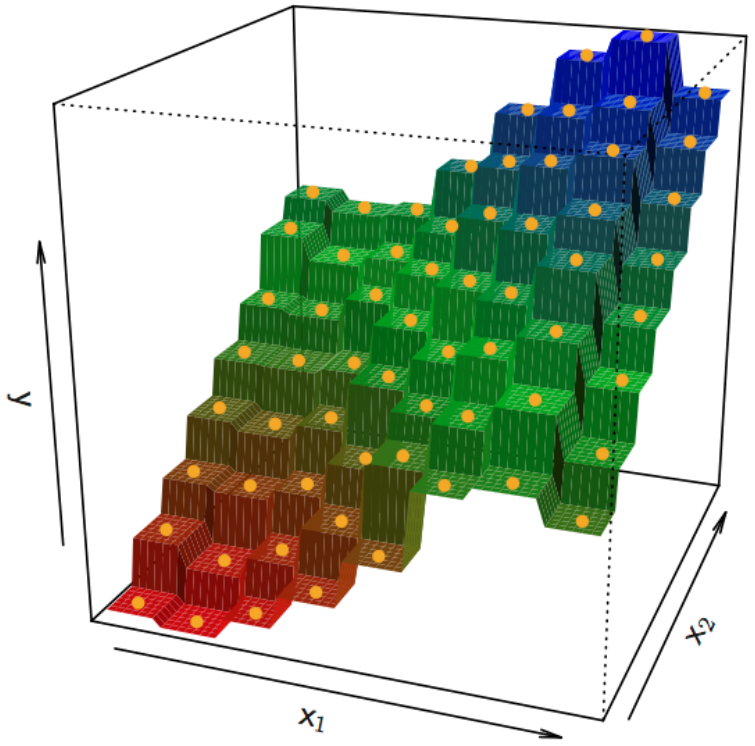
$$f(x) = \frac{1}{K} \sum_{x_i \in N_i} y_i$$

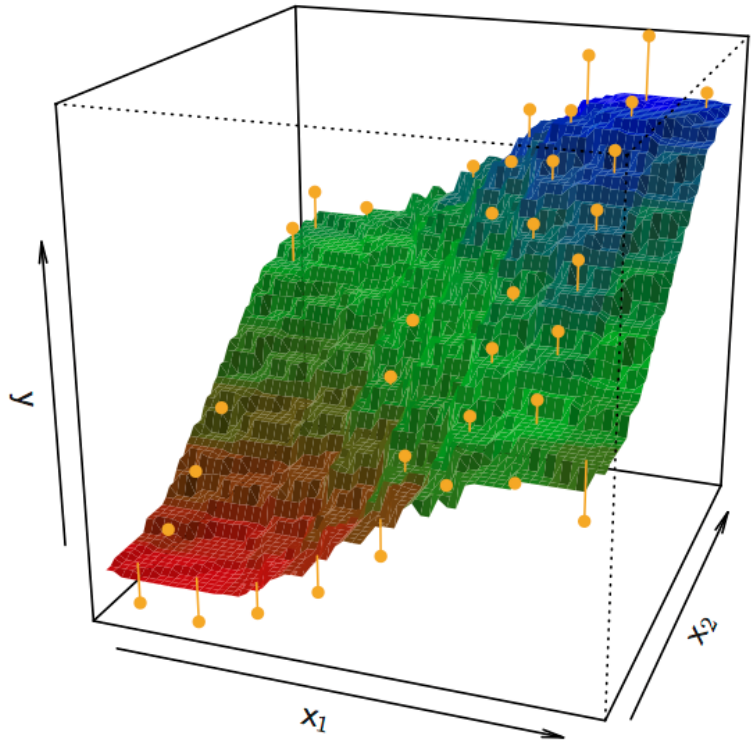
- If k is small kNN is much more flexible than linear regression.
- Is that better?

- The results may be highly dependent on the choice of k.



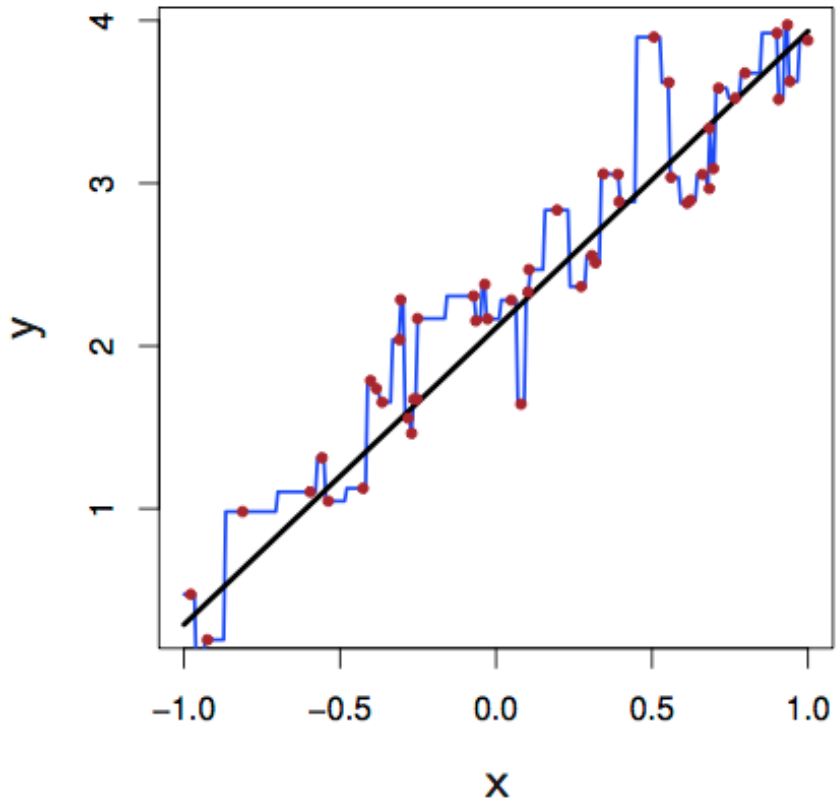
KNN Fits for k =1 and k = 9

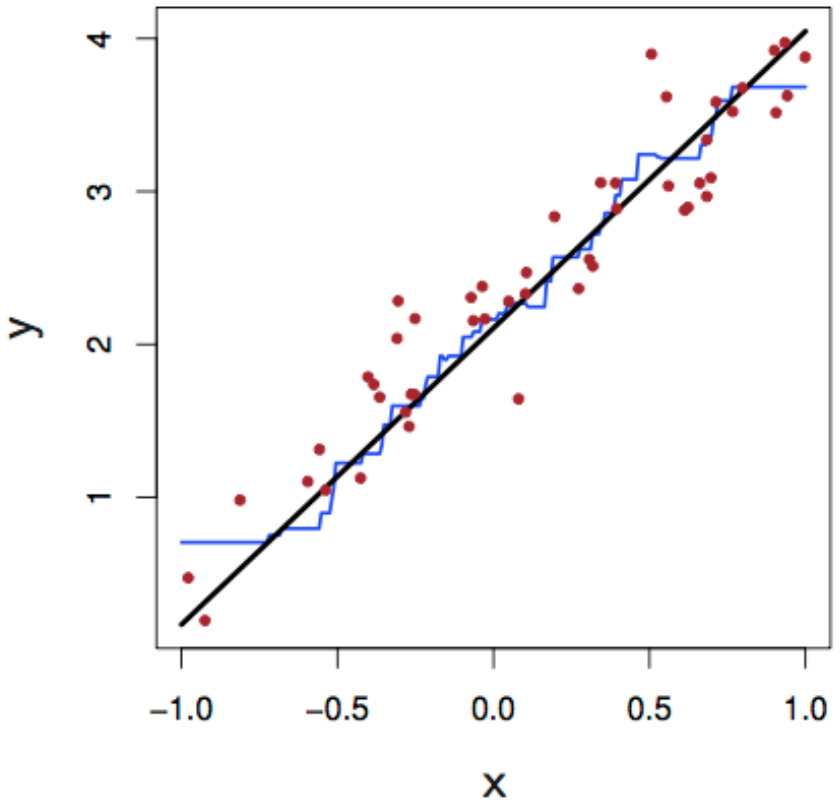




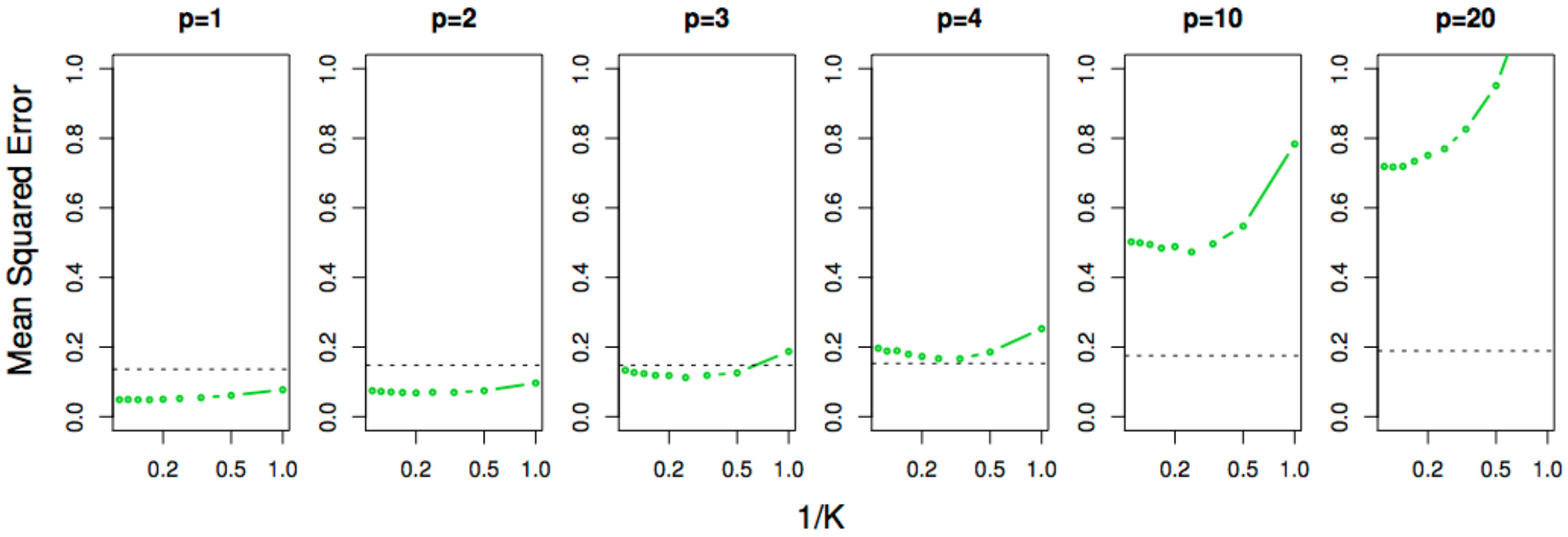
KNN fits in one dimension (k =1 and k = 9)

- black line: actual function,
- blue line: regressional kNN





kNN is not so good in high dimensional situations



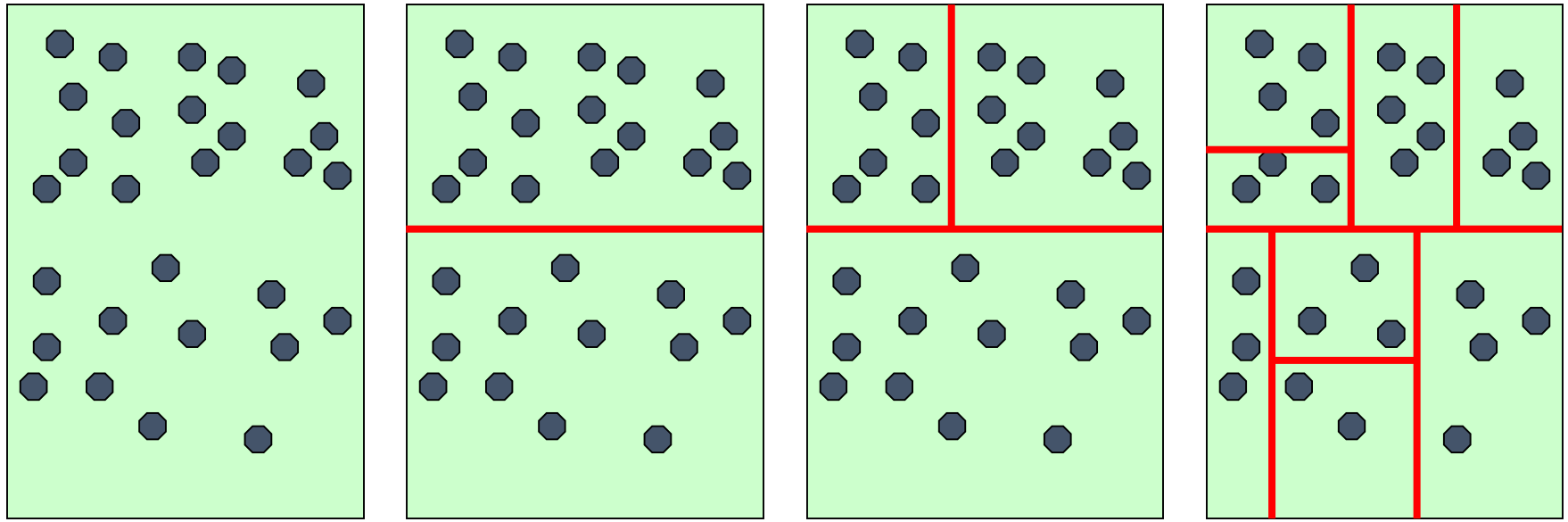
- p is the number of dimensions

Speeding up KNN algorithm

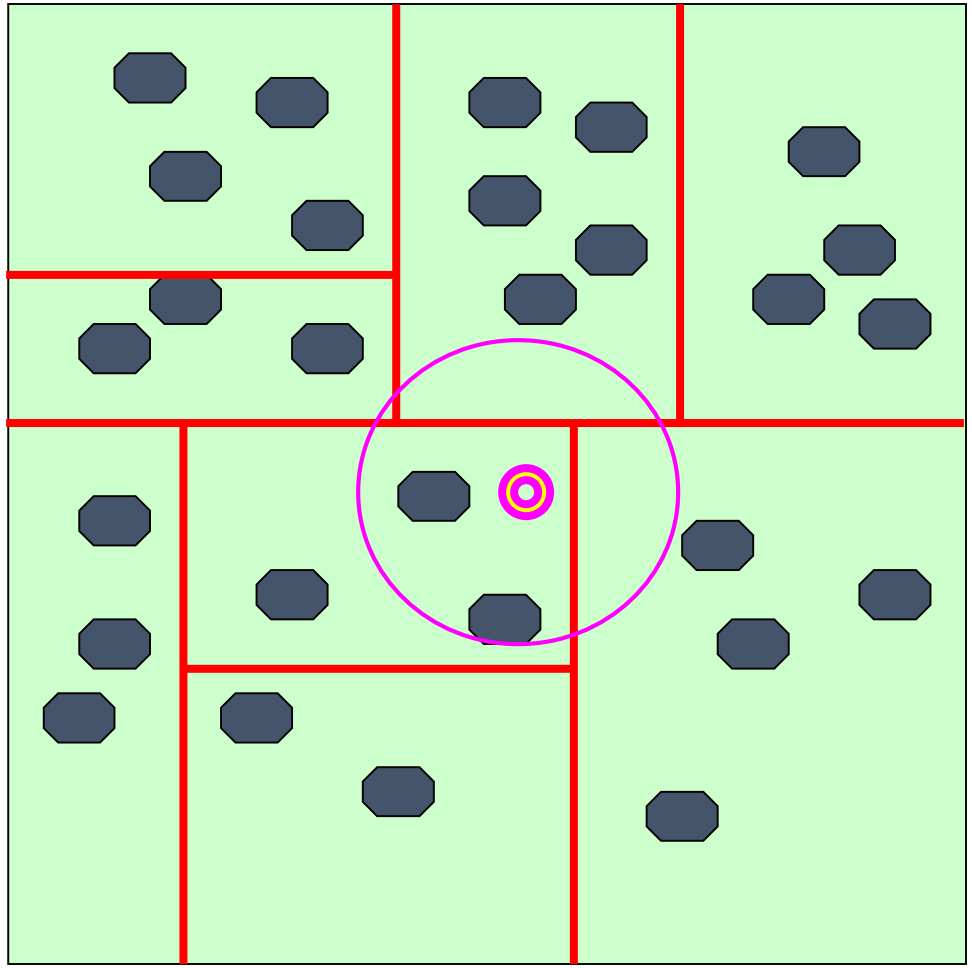
- precondition: normalization of dimensions, e.g., to [0, 1]
- naive search for nearest neigbors: $O(n \cdot d \cdot t)$
  - n is number of instances
  - d is number of dimensions
  - t is number of nearest neighbors
- exact search for low dimensional spaces
  - k-d trees for (d is around 10)
  - quad-trees (d=2), octrees (d=3)
  - R-tree (rectangular tree), also R+, R*, ...), d=2 or 3
  - M-tree (metric tree), also PM-tree, d=2 or 3
- approximate search
  - RKD-tree (random k-d tree)
  - locally sensitive hashing (LSH),
  - hierarchical k-means,
  - boundary forest

Speeding up KNNK-d trees

- K-d tree = binary tree with k keys



k-d trees and NN search



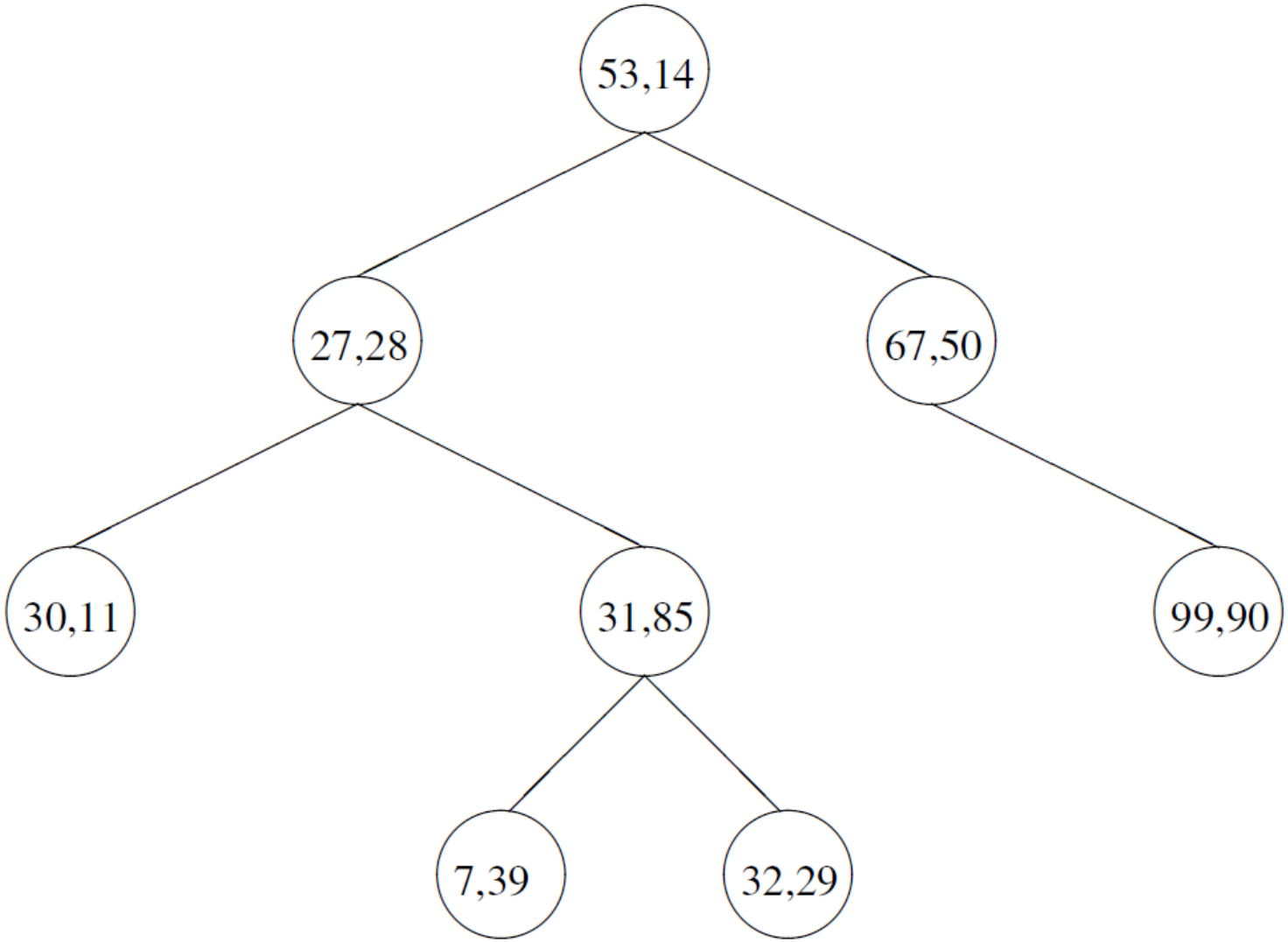
- Selection of:
  - split dimension
  - split value
- Optimized tree
- Finds t NN in

  O(log N)

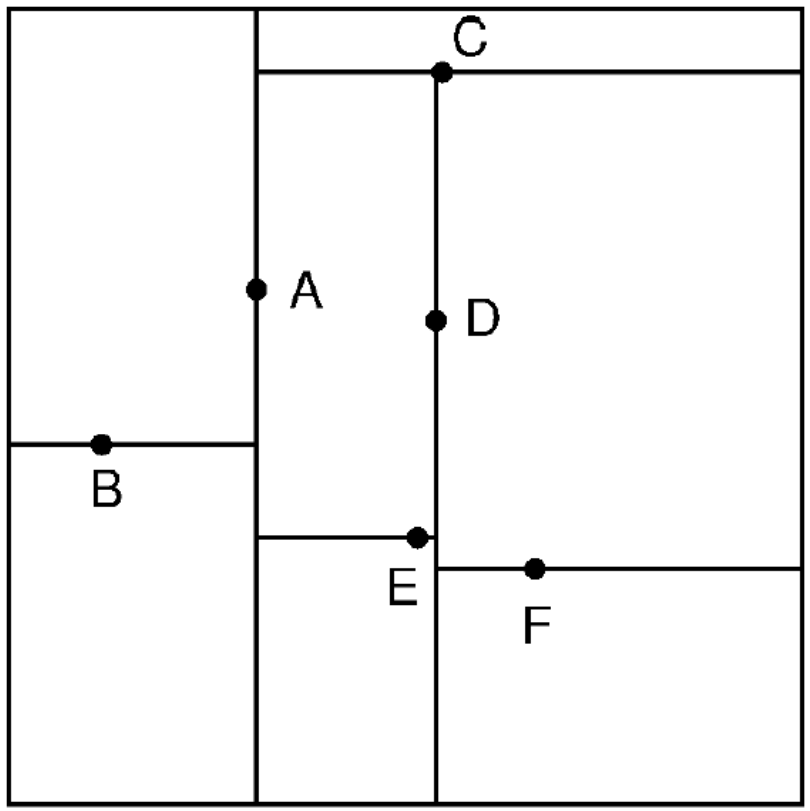
k-d tree

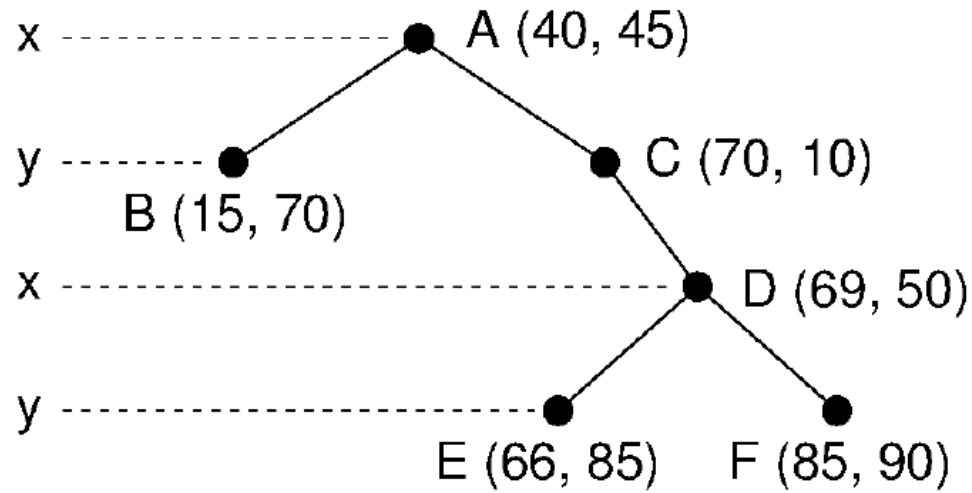
- k-d tree (k-dimensional tree)
- generalization of binary search tree to k-dimensional keys
- the simplest spliting criterion: on level d split on dimension 1+ (d mod k)

Example of 2-d tree



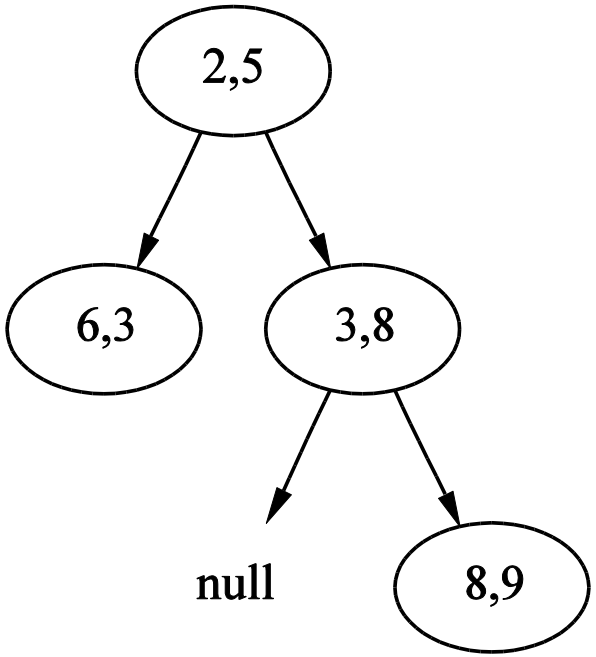
Example of 2-d tree

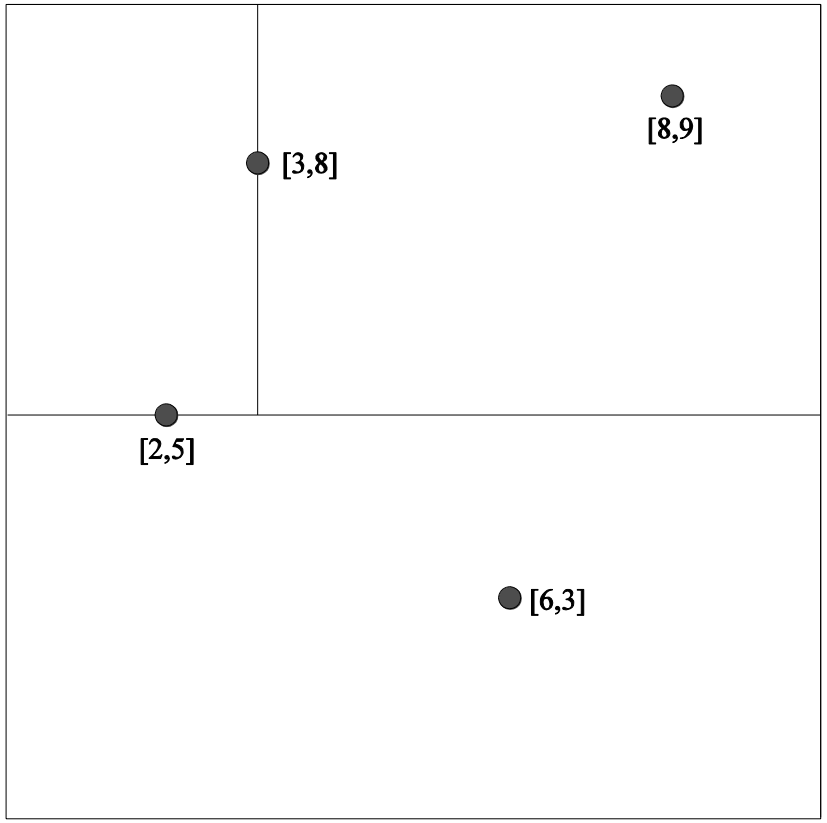




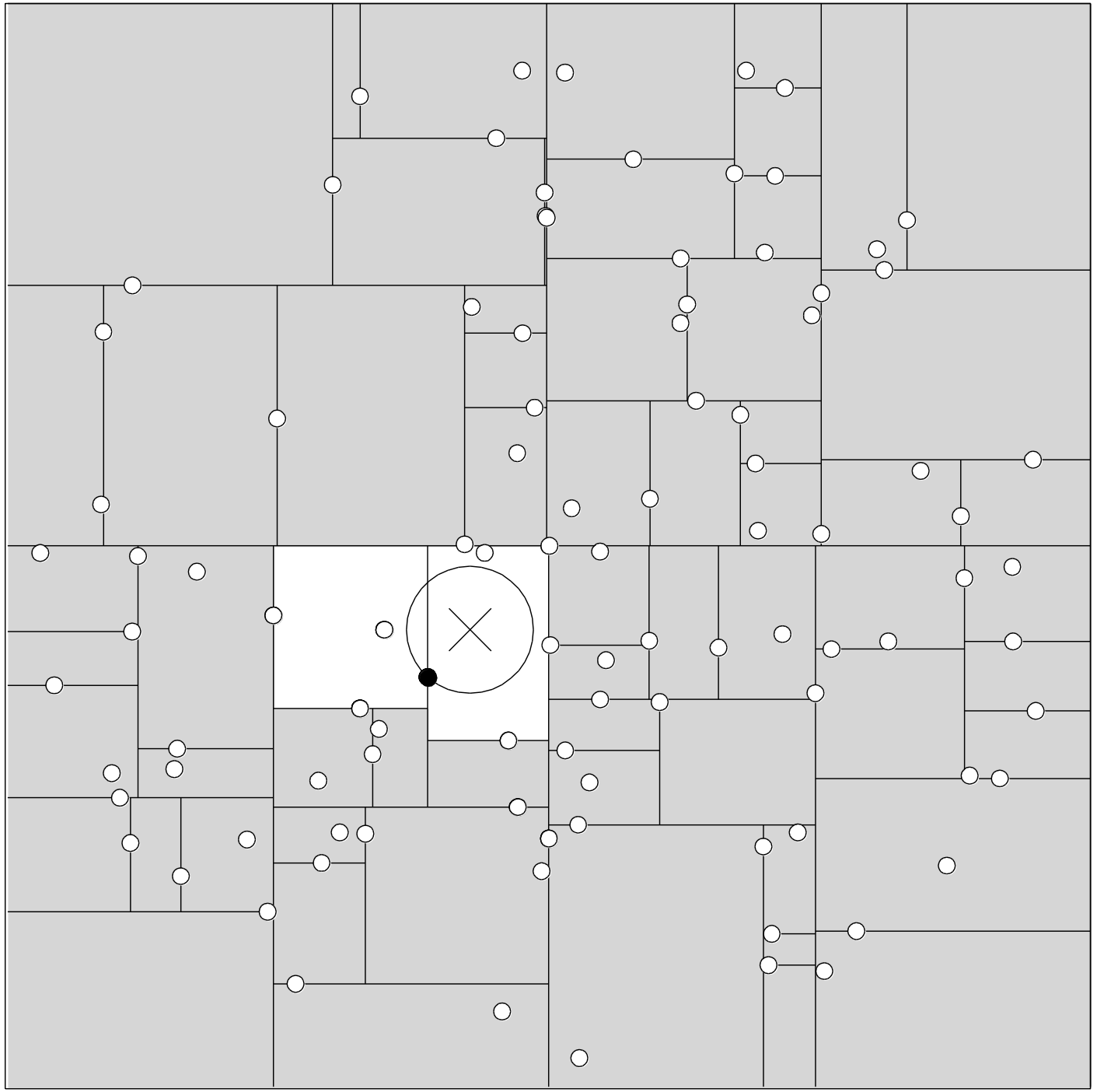
k-d trees for NN search

- structure the search space
- dimension are chosen based on variance of dimensions in each node separately (e.g., start with 2, then 1)





k-d trees for NN search



k-d tree node

- the nodes stores also the index of splitting dimension and the splitting value
- keys and pointers to children nodes

Tree construction in practice

- on average we search O(log n) leaves, worst case is O(n) leaves
- we would like the leaves to be approximately hypercubes (not hyperrectangles)
- splitting dimension is selected to maximize variance in that dimension
- splitting value is usually the median of chosen dimension
- normalize data to [0,1]:
  val = (data[i][j]-low[j])/(max[j] – min[j])
- in leaves we store more than 1 element (typically bucket size is around 10)
- tree construction takes O(*n* log *n*)
- only makes sense if searching for large number of NN

Searching NNs

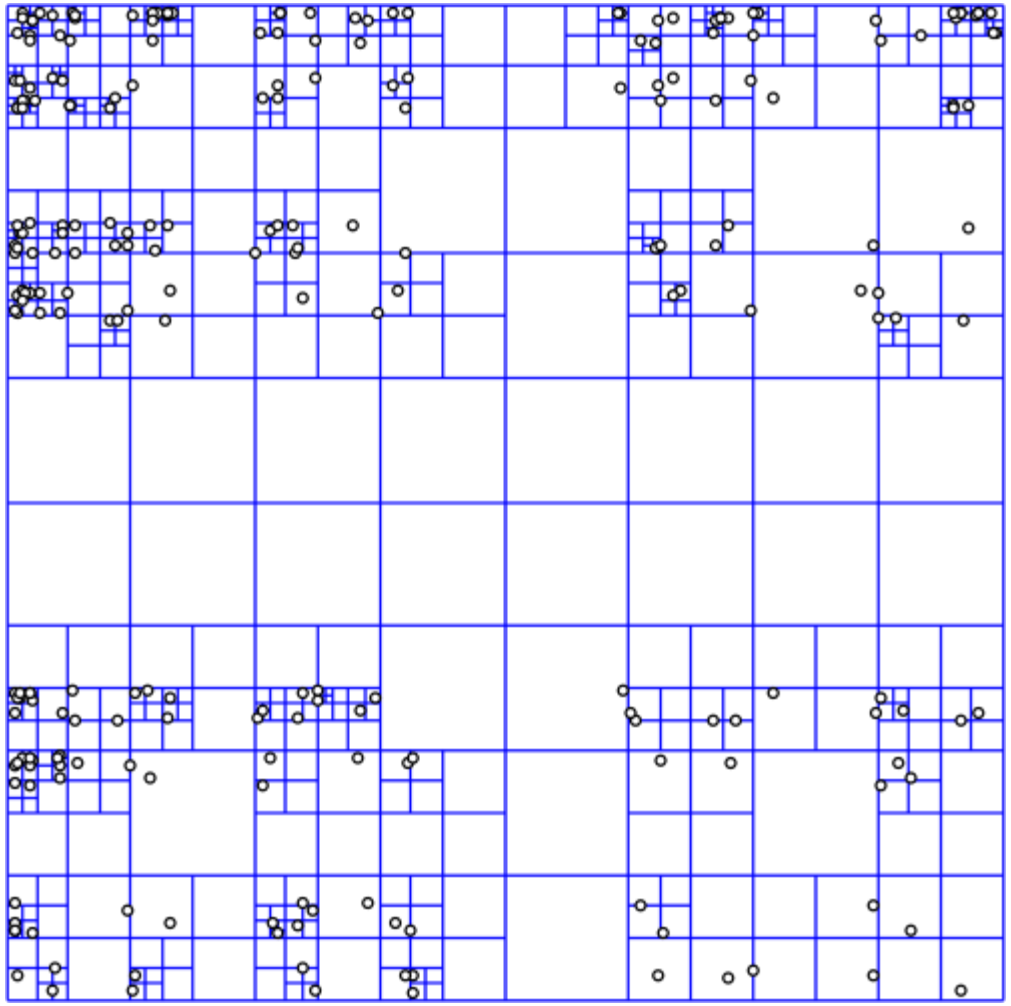
- recursively to the leaf containing query point q
- (the nearest) instance from that leaf is the first approximation for the nearest neighbor, the distance from that point to q is the radius of hypersphere
- in recursive backtracking we check if hyperspere with center in q would intersect the hyperrectangle of the other subtree; if yes we enter that subtree and search in it
- if we find a new closest element, this is the new radius of hypersphere
- when searching for t nearest: the radius of hypersphere is the t-th closest element
- works when the number of instances is much larger than the dimensionality , $n >> 2^k$

Algoritem 4 Iskanje k-najbližjih sosedov v k-d drevesu

```
1: procedure KNN_SEARCH(node, p, k)
2:     ▷ node represents current node, p is the query point and k is the
           number of nearest neighbors to return
3:     if node is Leaf then
4:         if node is closer to p than current k-th NN then
5:             add node to current NNs and remove previous k-th NN
6:     else
7:         ▷ d is splitting dimension of current node
8:         d ← node.d
9:         if p_d < node_d then
10:            knn_search(node.left, p, k)
11:            if node is closer to p than current k-th NN then
12:                add node to current NNs and remove previous k-th NN
13:            if hypersphere(node, k-thNN) ∩ hyperplane(node.right) ≠ ∅
               then
14:                ▷ the right subtree might contain a point which is closer to
               p than the current k-th NN
15:                knn_search(node.right, p, k)
16:        else
17:            knn_search(node.right, p, k)
18:            if node is closer to p than current k-th NN then
19:                add node to current NNs and remove previous k-th NN
20:            if hypersphere(node, k-thNN) ∩ hyperplane(node.left) ≠ ∅
               then
21:                ▷ the left subtree might contain a point which is closer to
               p than the current k-th NN
22:                knn_search(node.left, p, k)
```

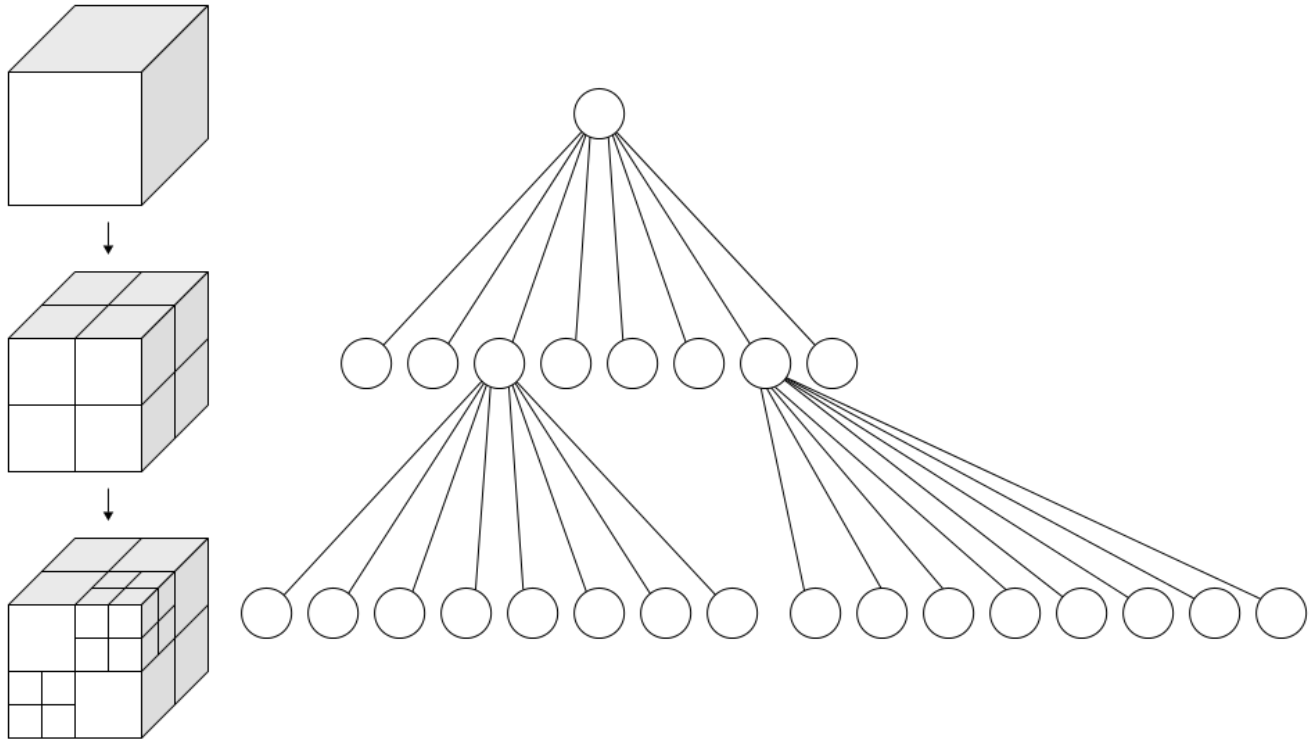
Quad-tree

- generalization of k-d tree
- each node has 4 successors
- used in spliting of 2D space
- leaves store points, curves, areas, polygons etc.



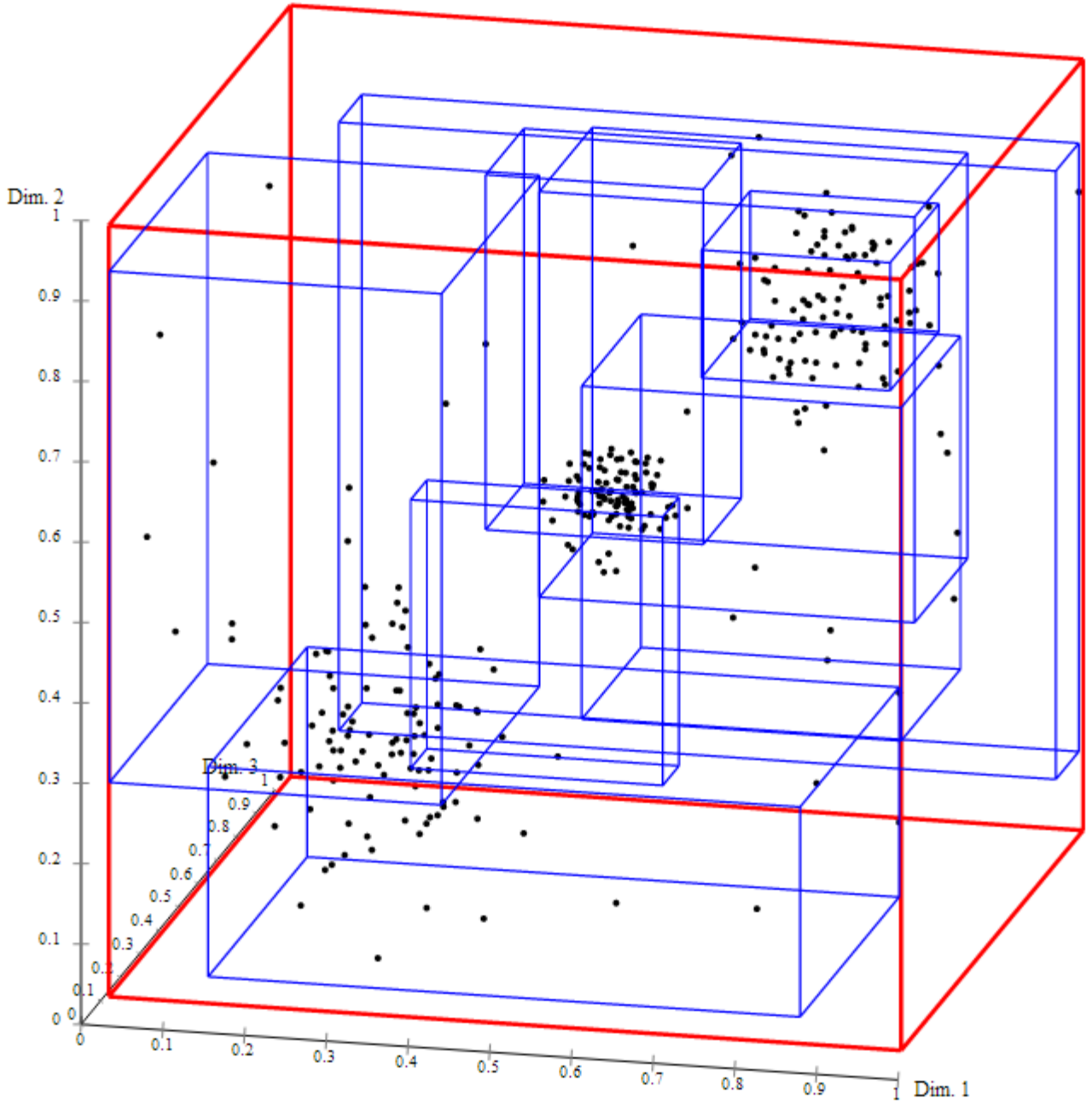
Octree

- as quad-tree, but each node has eight successors,
- to be used in 3D spacesa

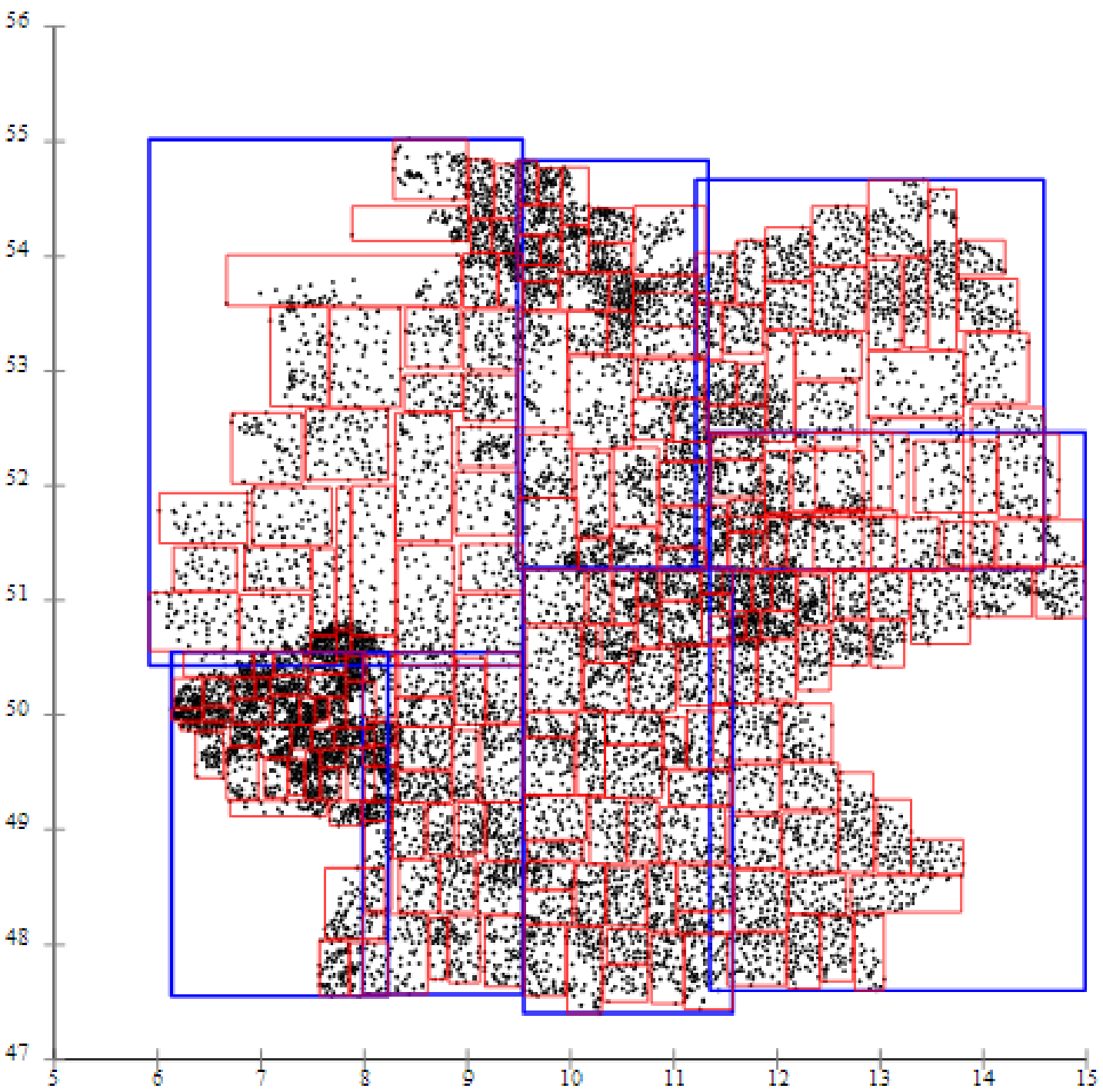


Example of R-tree in 3D

- nodes store minimally bounding, possibly overlapping (hyper) rectangles

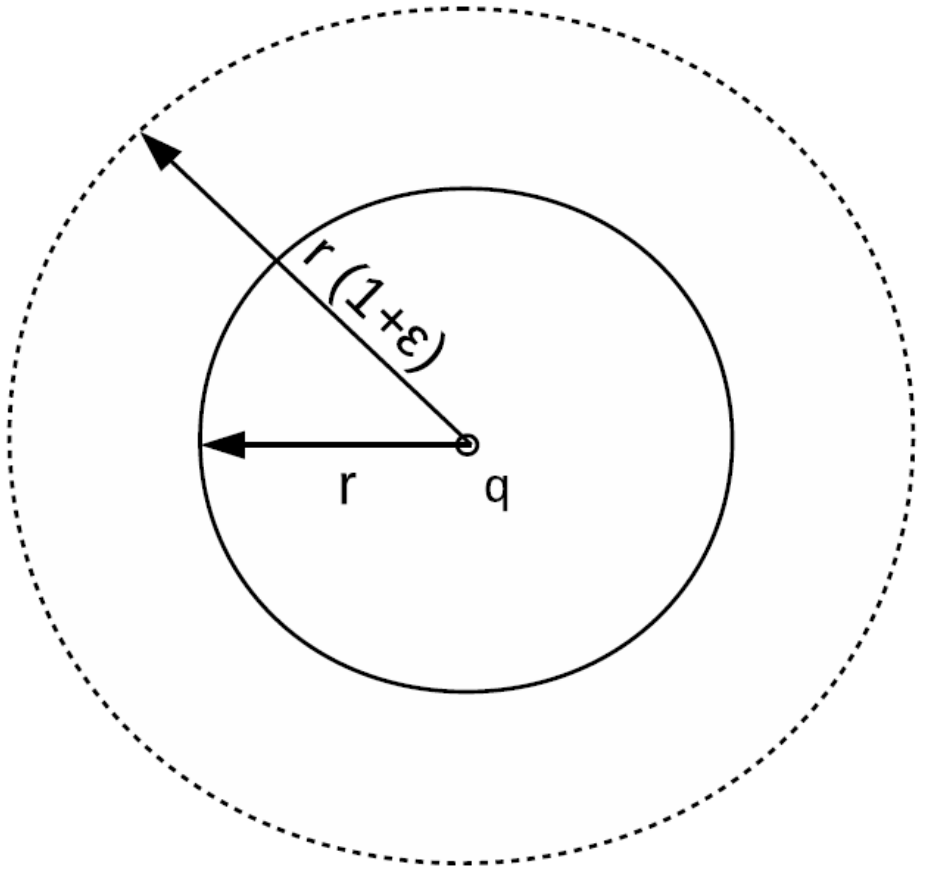


R*-tree
more balanced than
R-tree due to different
splitting heuristics



Approximate methods

- ε-approximate methods



RKD-tree

- kd-tree for high dimensional data
- we build a set of m kd-trees which differ in their splitting dimensions
- we take into account only v dimensions with the largest variance
- in each tree we randomly select splitting dimensions in nodes
- if the probability not to find the nearest element with a single tree is p, for m trees this probability is p^m
- easy to use several threads, sharing the common list of nearest instances,
- several variants how t choose dimensions, splitting points

Locally sensitive hash functions

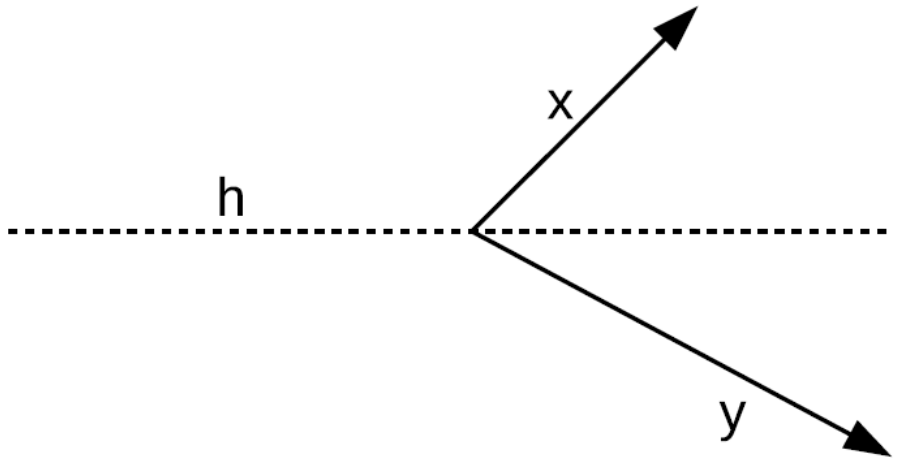
- LSH (locally sensitive hashing)
- among the fastest methods
- a family of hash functions H with parameters (r1, r2, P_1, P_2), r1<r2, P1>P2 is (r1, r2, P_1, P_2)-sensitive

$$if\,||p-q|| \leq r1 \implies Pr_H[h(p)=h(q)] \geq P_1$$
$$if\,||p-q|| \geq r2 \implies Pr_H[h(p)=h(q)] \leq P_2$$

- we generate l tables with m hash functions
- each table generates one hash value for an object
- with enough tables two neighboring points have different hash values with high probability

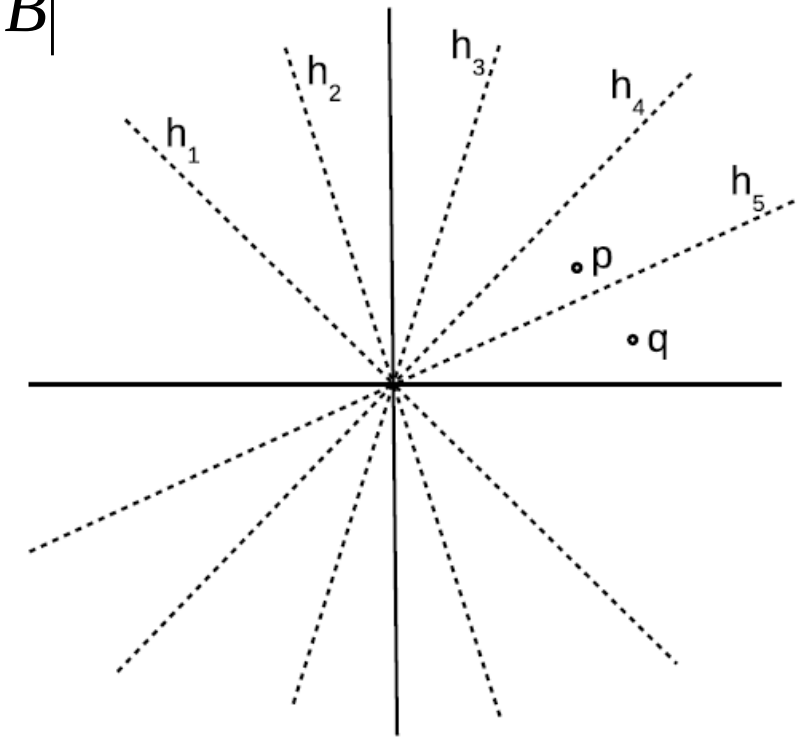
An example: hashing with random hyperplanes

- hyperplane is determined with a normal vector n, which is rectangular to the plane
- if for a dot product of vectors sign(xn) ≠ sign(yn), then x and y are on the opposite sides of the plane h determined by n



Hash functions for cosine distance

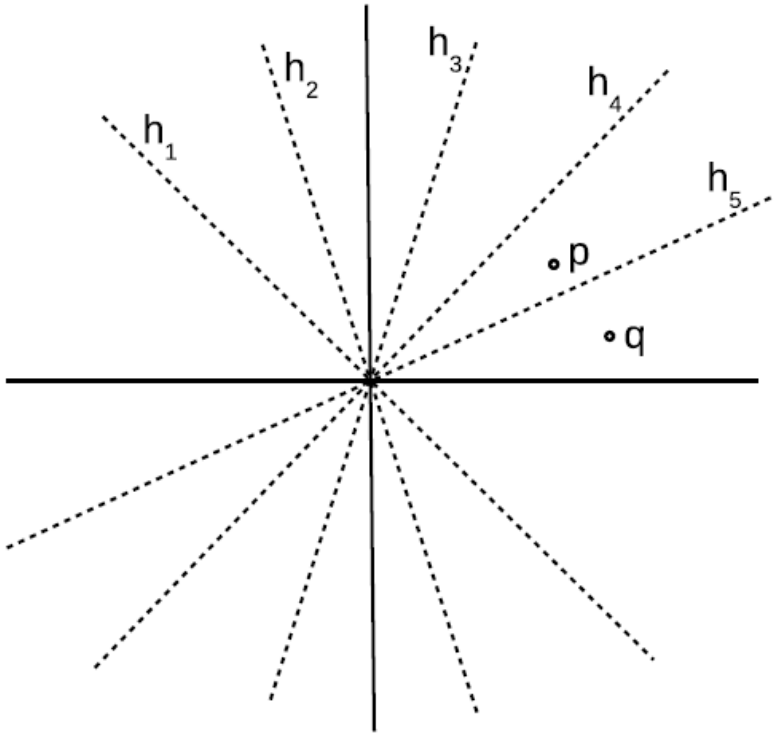
- cosine distance between vectors: $\cos(\Theta) = \frac{A \cdot B}{|A||B|}$
- let Θ be an angle between x and y
- the probability that a random hyperplane separates x and y is $\Theta/180$
- this gives a family of locally sensitive functions for cosine distance: each hash function f(x) is a random hyperplane
- f(x) = f(y) if and only if sign(xn) = sign(yn)
- this family is (r1, r2, (180-r1)/180,(180-r2)/180)-sensitive



LSH insertion

- for each hash table, compute the hash value of a point and insert it into a bucket representing that value
- repeat for all hash tables and insert each point into / buckets

Insertion into a single table



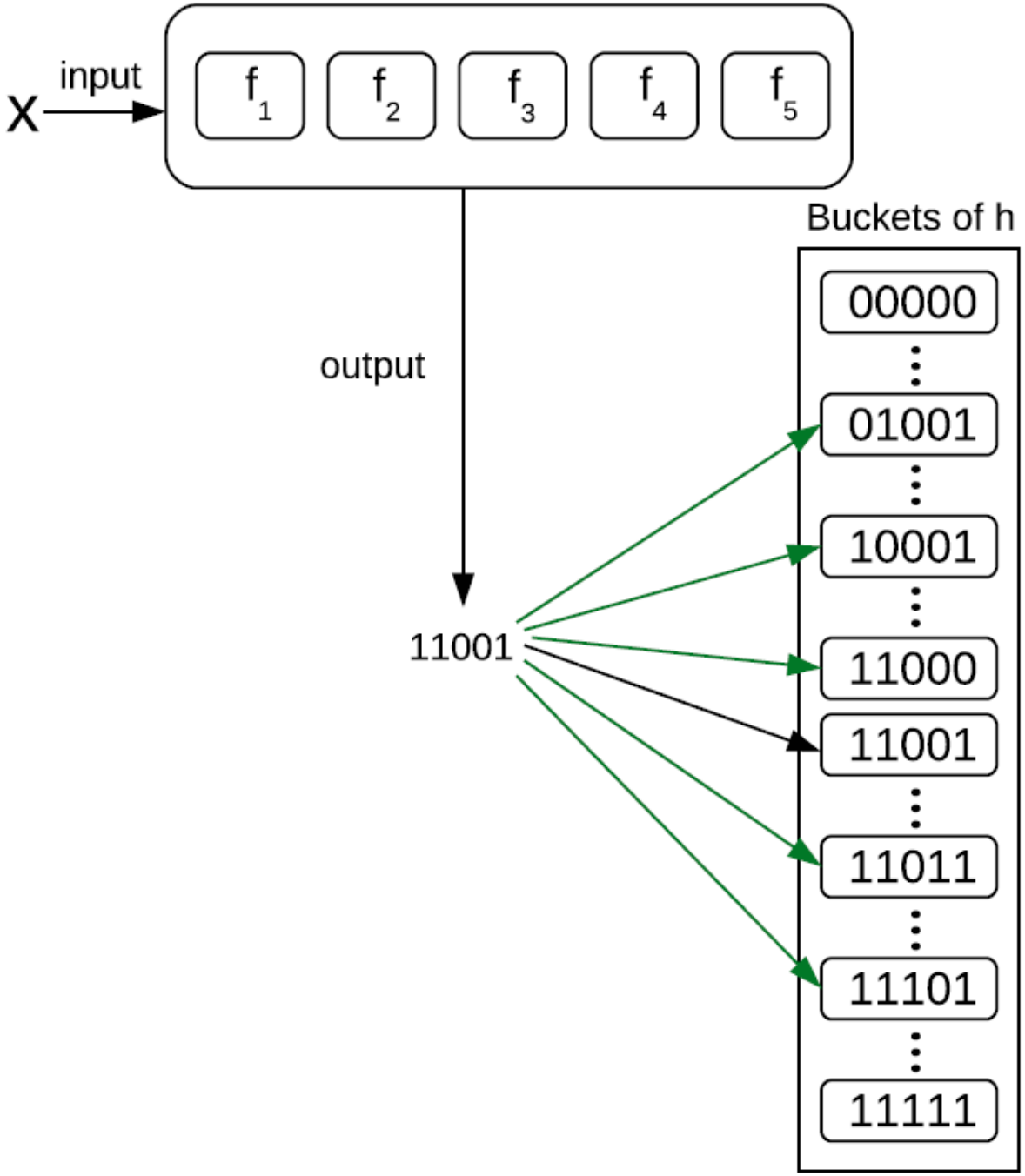
	h_1	h_2	h_3	h_4	h_5
hash(p):	1	1	1	1	0
hash(q):	1	1	1	1	1

LSH search

- for a point u compute hash values with all functions and get hash values and bucket indices
- compare u with elements in that buckets and choose the nearest one (or more)
- a weakness: for larger k there may not be enough instances in that buckets
- improvement: multiprobe LSH search

Multiprobe-LSH

- similar elements have similar hash vaues
- we set the allowed difference and insert an element in all buckets with allowed difference
- we can use smaller tables
- an example for allowed difference of 1



Hierarchical k-means

- repeat several times
  - recursively run k-means clustering, while clusters are small enough
  - in one clustering compare the element with centers of the clusters and find the nearest neighbor in the clusters
  - for each cluster we also store its radius (distance to the farthest member), to detect overlapping

Algoritem 5 K-means

```
1:  procedure K-MEANS(points, k)
2:      clusters ← ChooseCenters(points, k)
3:      while centers have not converged do
4:          for each p in points do
5:              c ← null
6:              mindist ← ∞
7:              for each cluster in clusters do
8:                  d ← distance(p, cluster)
9:                  if d < mindist then
10:                     mindist ← d
11:                     c ← cluster
12:             add point p to cluster c
13:         for each i = 1 .. k do
14:             clusters[i] = mean of points in cluster i
15:     return clusters
```

Rule learning

- Using IF-THEN Rules for Classification
- Represent the knowledge in the form of IF-THEN rules

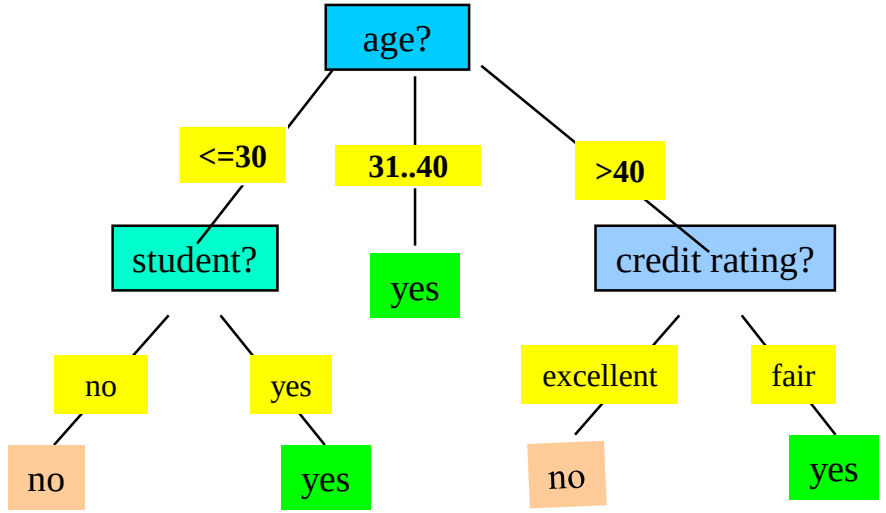
  R: IF *age* = youth AND *student* = yes THEN *buys_computer* = yes
  - Rule antecedent/precondition vs. rule consequent
- Assessment of a rule: *coverage* and *accuracy*
  - n_{covers} = # of tuples covered by R
  - $n_{correct}$ = # of tuples correctly classified by R

  coverage(R) = n_{covers} /|D| /* D: training data set */

  accuracy(R) = $n_{correct}$ / n_{covers}
- If more than one rule are triggered, need **conflict resolution**
  - Size ordering: assign the highest priority to the triggering rules that has the "toughest" requirement (i.e., with the *most attribute tests*)
  - Class-based ordering: decreasing order of *prevalence or misclassification cost per class*
  - Rule-based ordering (**decision list**): rules are organized into one long priority list, according to some measure of rule quality or by experts

Rule extraction from a decision tree

- Rules are *easier to understand* than large trees
- One rule is created *for each path* from the root to a leaf
- Each attribute-value pair along a path forms a conjunction: the leaf holds the class prediction
- Rules are mutually exclusive and exhaustive



- Example: Rule extraction from our *buys_computer* decision-tree

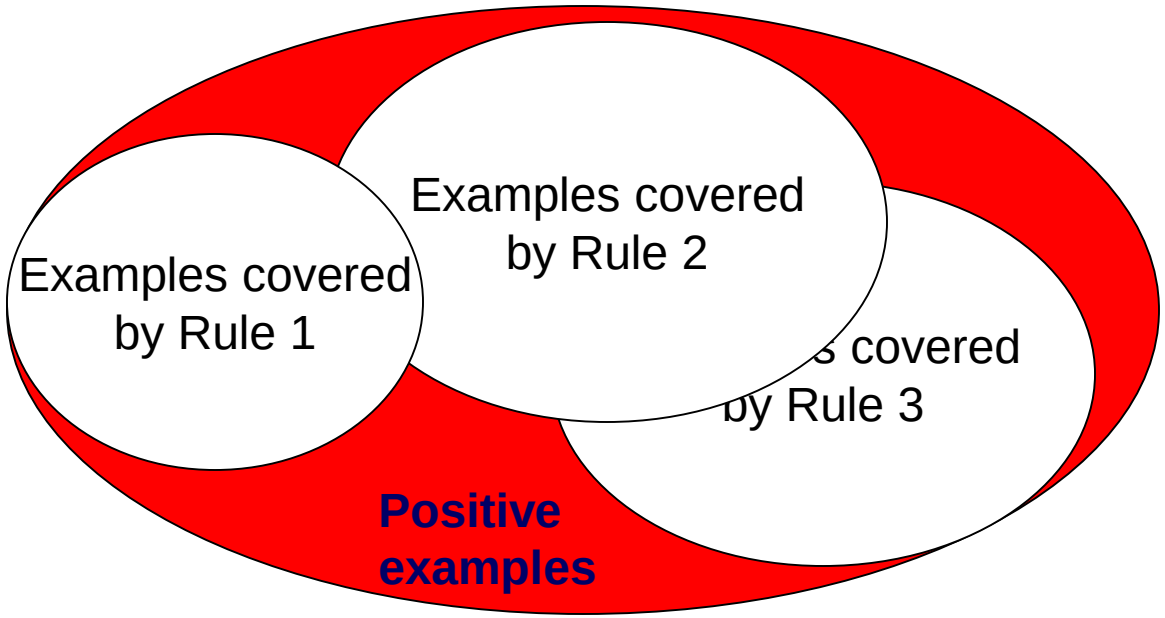
IF *age* = young AND *student* = *no* THEN *buys_computer* = *no*
IF *age* = young AND *student* = *yes* THEN *buys_computer* = *yes*
IF *age* = mid-age THEN *buys_computer* = *yes*
IF *age* = old AND *credit_rating* = *excellent* THEN *buys_computer* = *no*
IF *age* = old AND *credit_rating* = *fair* THEN *buys_computer* = *yes*

Rule induction: sequential covering method

- Sequential covering algorithm: extracts rules directly from training data
- Typical sequential covering algorithms: FOIL, AQ, CN2, RIPPER
- Rules are learned *sequentially*, each for a given class C_i will cover many tuples of C_i but none (or few) of the tuples of other classes
- Steps:
  - Rules are learned one at a time
  - Each time a rule is learned, the tuples covered by the rules are removed
  - Repeat the process on the remaining tuples until *termination condition*, e.g., when no more training examples or when the quality of a rule returned is below a user-specified threshold
- Compare with decision-tree induction: learning a set of rules *simultaneously*

Sequential covering algorithm

while (enough target tuples left)
    generate a rule
    remove positive target tuples satisfying this rule



Rule generation

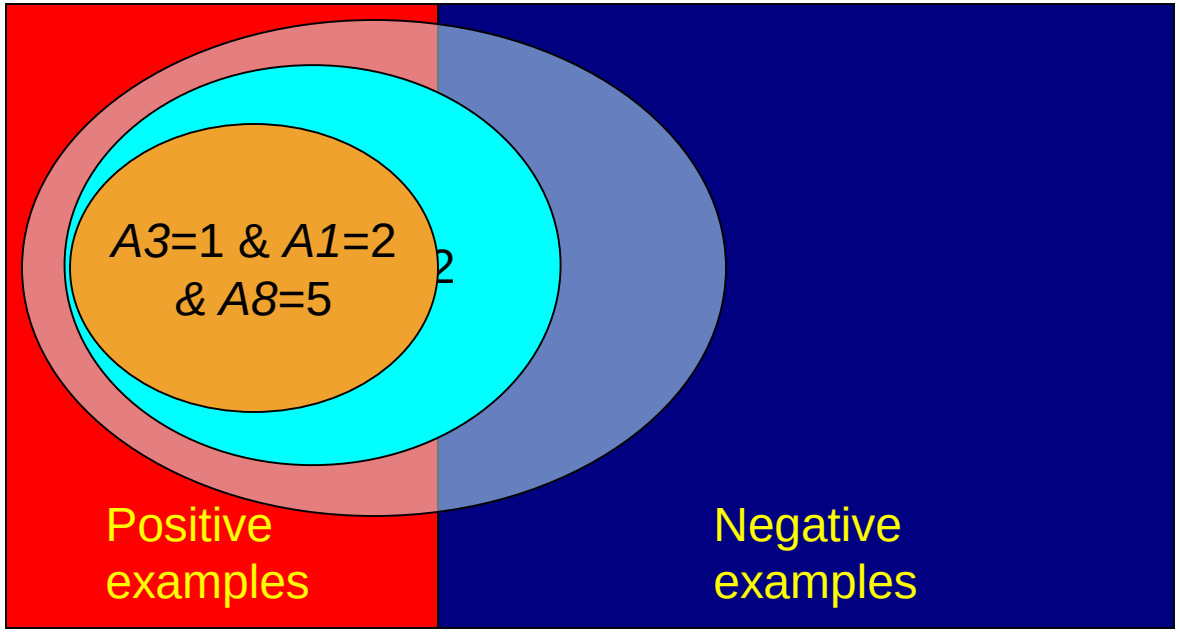
- To generate a rule

  while(true)

    find the best predicate *p*

    if ruleQuality(*p*) > threshold **then** add *p* to current rule

    else break



How to learn one rule?

- Start with the *most general rule* possible: condition = empty
- *Adding new attributes* by adopting a greedy depth-first strategy
  - Picks the one that most improves the rule quality
- Rule-quality measures: consider both coverage and accuracy
  - Foil-gain (in FOIL & RIPPER): assesses information gain by extending condition

$$FOIL\_Gain = pos' \times (\log_2 \frac{pos'}{pos'+neg'} - \log_2 \frac{pos}{pos+neg})$$

    - favors rules that have high accuracy and cover many positive tuples
- Rule pruning based on an independent set of test tuples

$$FOIL\_Prune(Rule) = \frac{pos-neg}{pos+neg}$$

Pos/neg are # of positive/negative tuples covered by Rule

If *FOIL_Prune* is higher for the pruned version of Rule, prune Rule

ACO for rule learning

- IF-THEN rules are comprehensible
- usually achieve lower classification accuracy compared to the best black-box machine learning approaches
- ACO based rule mining algorithms build a discrete search space, represented by a graph, in which ants try to find the best rule set by discrete optimization.
- good for discrete optimization but can be problematic when datasets are described with numeric or mixed attribute types
- can use discretization as a pre-processing step

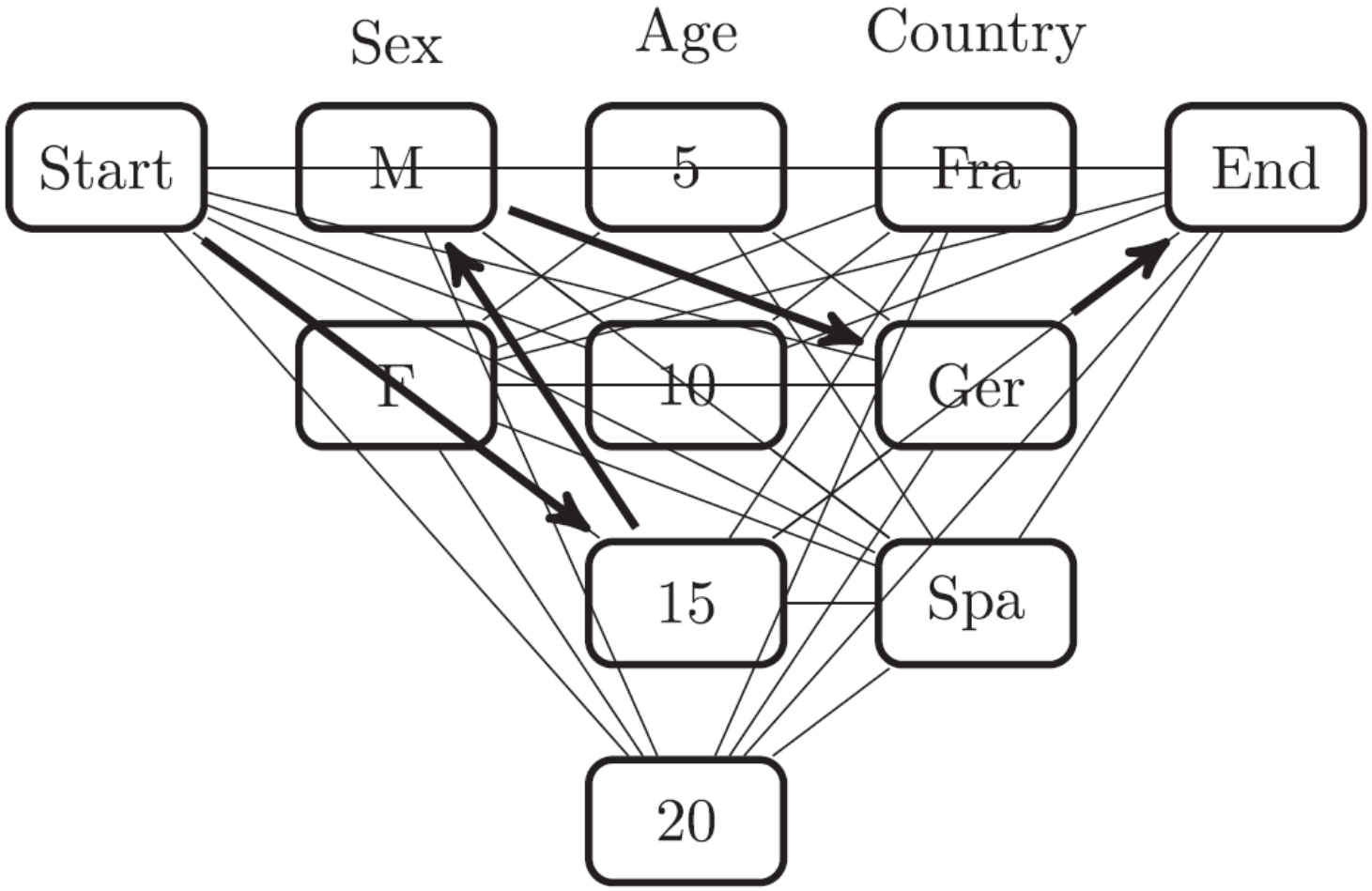
Ant-Miner idea

- Parpinelli (2002)
- separate and conquer approach for rule generation
  - generate one rule
  - remove (separates) the covered examples from the dataset
  - learn the remaining rules (conquers) from the remaining
- can only use nominal attributes

Ant-Miner algorithm

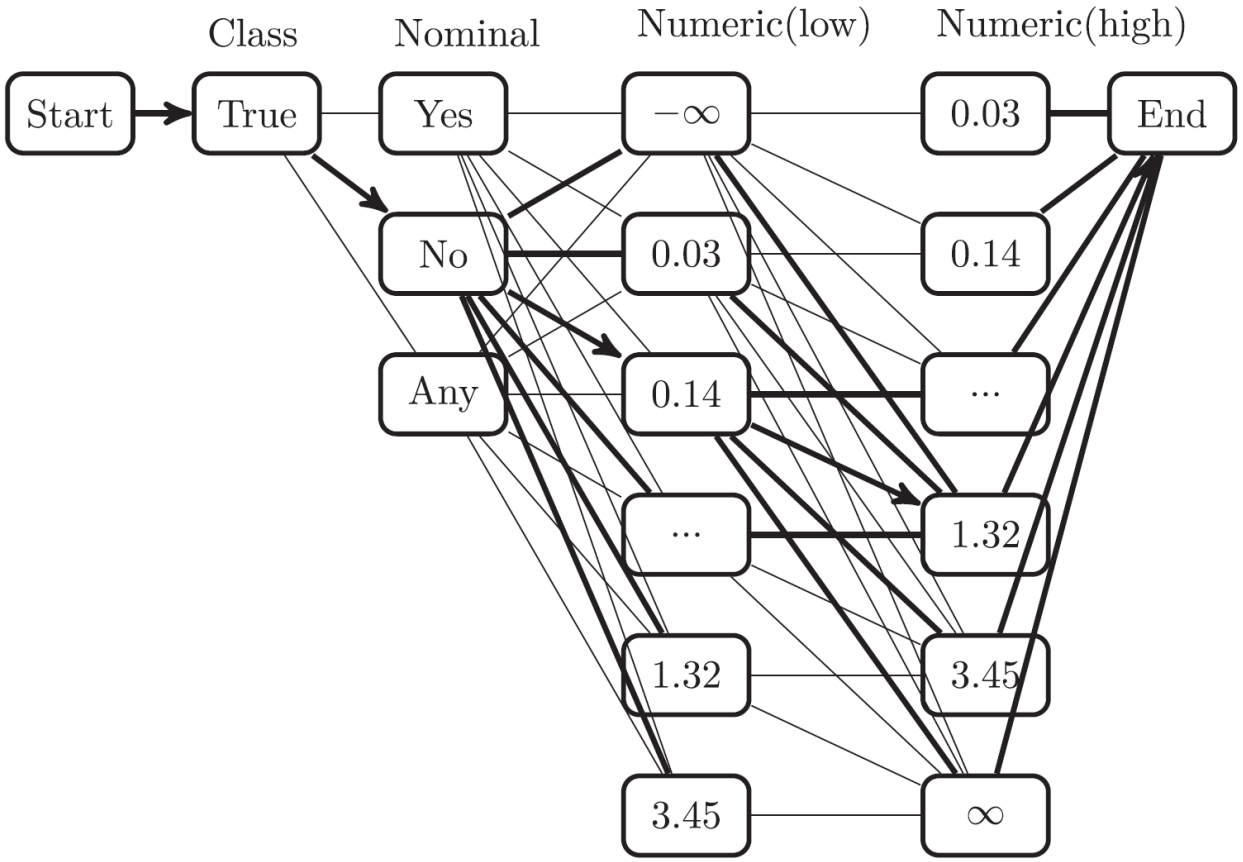
- construct a discrete search space from given data
- ants forage the graph from the start to the end node and the path they make describes a classification rule
- the found rules are evaluated and based on their quality, and the paths by which they were constructed are strengthened by artificial pheromones
- the process is repeated until all or most of the ants converge to a single path and then the corresponding rule is added to the rule set
- the examples covered by this rule are removed from the training data and the process is repeated until no more data remains.

Ant-Miner graph



nAntMiner

- Pičulin & Robnik-Šikonja (2014)
- handles numeric attributes directly
- uses Max-Min ant system



nAntMiner pseudocode

Algorithm 1. nAnt-Miner algorithm

```
1: TrainingSet ← {All training instances}
2: RuleList ← {}
3: while not stopping criterion do
4:    construct graph
5:    Init: heuristic (η), pheromones (τ), probabilities (P)
6:    while not converged do
7:       Let ants run from Start to End nodes
8:       Keep 'elite' number of rules
9:       Prune rules
10:      Update global best rule
11:      Update pheromone values on paths defined by elite
   rules and global best rule
12:   end while
13:   Add R_best to RuleList
14:   TrainingSet = TrainingSet \ {instances covered by R_best}
15: end while
16: return Rulelist
```

Bias in models, bias in data

Biases in the data

- Machine learning models are not inherently objective. Engineers train models by feeding them a data set of training examples, and human involvement in the provision and curation of this data can make a model's predictions susceptible to bias.
- When building models, it's important to be aware of common human biases that can manifest in your data, so you can take proactive steps to mitigate their effects.
- The biases listed provide just a small selection of biases that are often uncovered in machine learning data sets; this list *is not intended to be exhaustive*. Wikipedia's catalog of cognitive biases enumerates over 100 different types of human bias that can affect our judgment. When auditing your data, you should be on the lookout for any and all potential sources of bias that might skew your model's predictions.

Reporting bias

- **Reporting bias** occurs when the frequency of events, properties, and/or outcomes captured in a data set does not accurately reflect their real-world frequency. This bias can arise because people tend to focus on documenting circumstances that are unusual or especially memorable, assuming that the ordinary can "go without saying."
- **EXAMPLE**: A sentiment-analysis model is trained to predict whether book reviews are positive or negative based on a corpus of user submissions to a popular website. The majority of reviews in the training data set reflect extreme opinions (reviewers who either loved or hated a book), because people were less likely to submit a review of a book if they did not respond to it strongly. As a result, the model is less able to correctly predict sentiment of reviews that use more subtle language to describe a book.

Automation bias

- **Automation bias** is a tendency to favor results generated by automated systems over those generated by non-automated systems, irrespective of the error rates of each.
- **EXAMPLE**: Software engineers working for a sprocket manufacturer were eager to deploy the new "groundbreaking" model they trained to identify tooth defects, until the factory supervisor pointed out that the model's precision and recall rates were both 15% lower than those of human inspectors.

Selection bias

- **Selection bias** occurs if a data set's examples are chosen in a way that is not reflective of their real-world distribution. Selection bias can take many different forms:
  - **Coverage bias**: Data is not selected in a representative fashion.
  - **EXAMPLE**: A model is trained to predict future sales of a new product based on phone surveys conducted with a sample of consumers who bought the product. Consumers who instead opted to buy a competing product were not surveyed, and as a result, this group of people was not represented in the training data.
  - **Non-response bias** (or **participation bias**): Data ends up being unrepresentative due to participation gaps in the data-collection process.
  - **EXAMPLE**: A model is trained to predict future sales of a new product based on phone surveys conducted with a sample of consumers who bought the product and with a sample of consumers who bought a competing product. Consumers who bought the competing product were 80% more likely to refuse to complete the survey, and their data was underrepresented in the sample.
  - **Sampling bias**: Proper randomization is not used during data collection.
  - **EXAMPLE**: A model is trained to predict future sales of a new product based on phone surveys conducted with a sample of consumers who bought the product and with a sample of consumers who bought a competing product. Instead of randomly targeting consumers, the surveyer chose the first 200 consumers that responded to an email, who might have been more enthusiastic about the product than average purchasers.

Group attribution bias

- **Group attribution bias** is a tendency to generalize what is true of individuals to an entire group to which they belong. Two key manifestations of this bias are:
  - **In-group bias**: A preference for members of a group to which *you also belong*, or for characteristics that you also share.
  - **EXAMPLE**: Two engineers training a resume-screening model for software developers are predisposed to believe that applicants who attended the same computer-science academy as they both did are more qualified for the role.
  - **Out-group homogeneity bias**: A tendency to stereotype individual members of a group to which *you do not belong*, or to see their characteristics as more uniform.
  - **EXAMPLE**: Two engineers training a resume-screening model for software developers are predisposed to believe that all applicants who did not attend a computer-science academy do not have sufficient expertise for the role.

Implicit bias

- **Implicit bias** occurs when assumptions are made based on one's own mental models and personal experiences that do not necessary apply more generally.
- **EXAMPLE**: An engineer training a gesture-recognition model uses a head shake as a feature to indicate a person is communicating the word "no." However, in some regions of the world, a head shake actually signifies "yes." A common form of implicit bias is **confirmation bias**, where model builders unconsciously process data in ways that affirm preexisting beliefs and hypotheses. In some cases, a model builder may actually keep training a model until it produces a result that aligns with their original hypothesis; this is called **experimenter's bias**.
- **EXAMPLE**: An engineer is building a model that predicts aggressiveness in dogs based on a variety of features (height, weight, breed, environment). The engineer had an unpleasant encounter with a hyperactive toy poodle as a child, and ever since has associatated the breed with aggression. When the trained model predicted most toy poodles to be relatively docile, the engineer retrained the model several more times until it produced a result showing smaller poodles to be more violent.