مقرر التعلم التلقائي | السنة الخامسة – هندسة المعلوماتية / الذكاء الصنعي | Week 6

KNN & NB

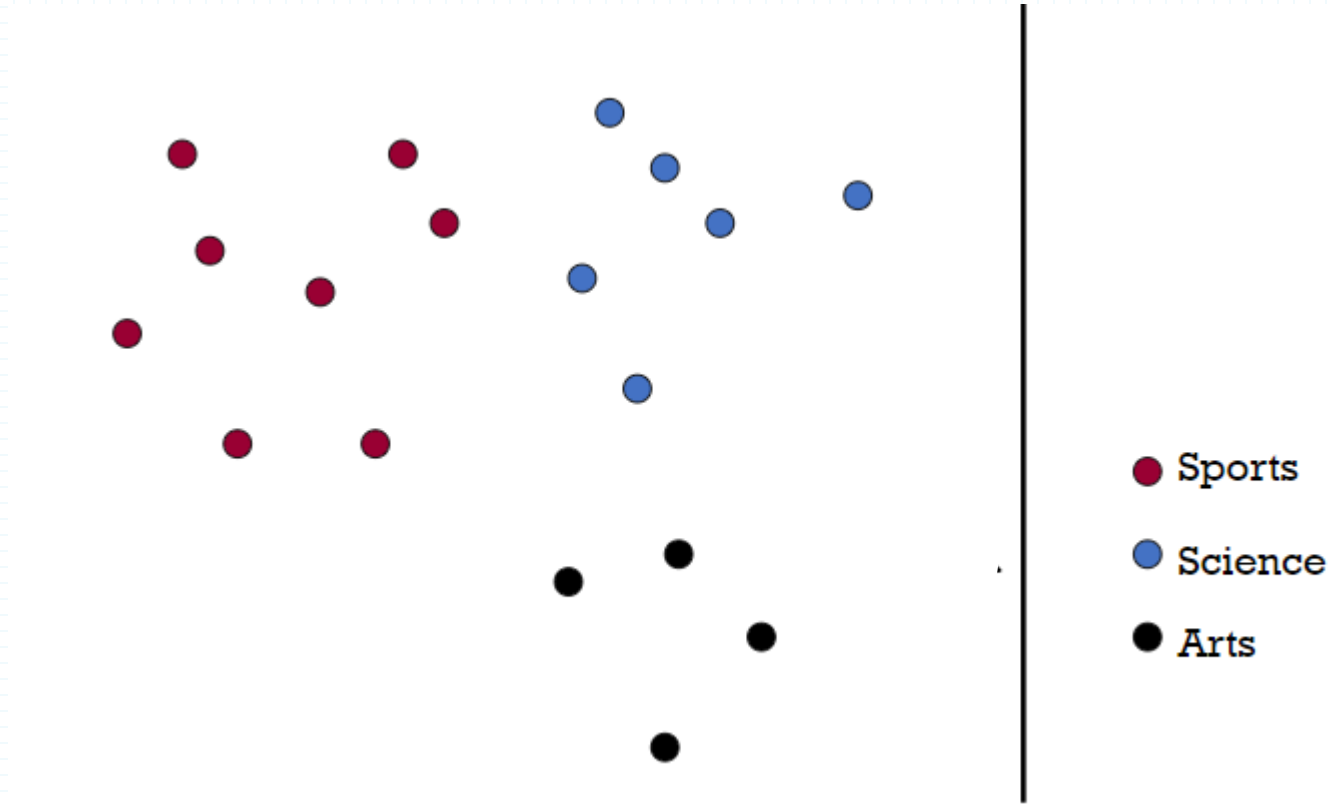
د. رياض سنبل

Access Course Materials

K-nearest-neighbor

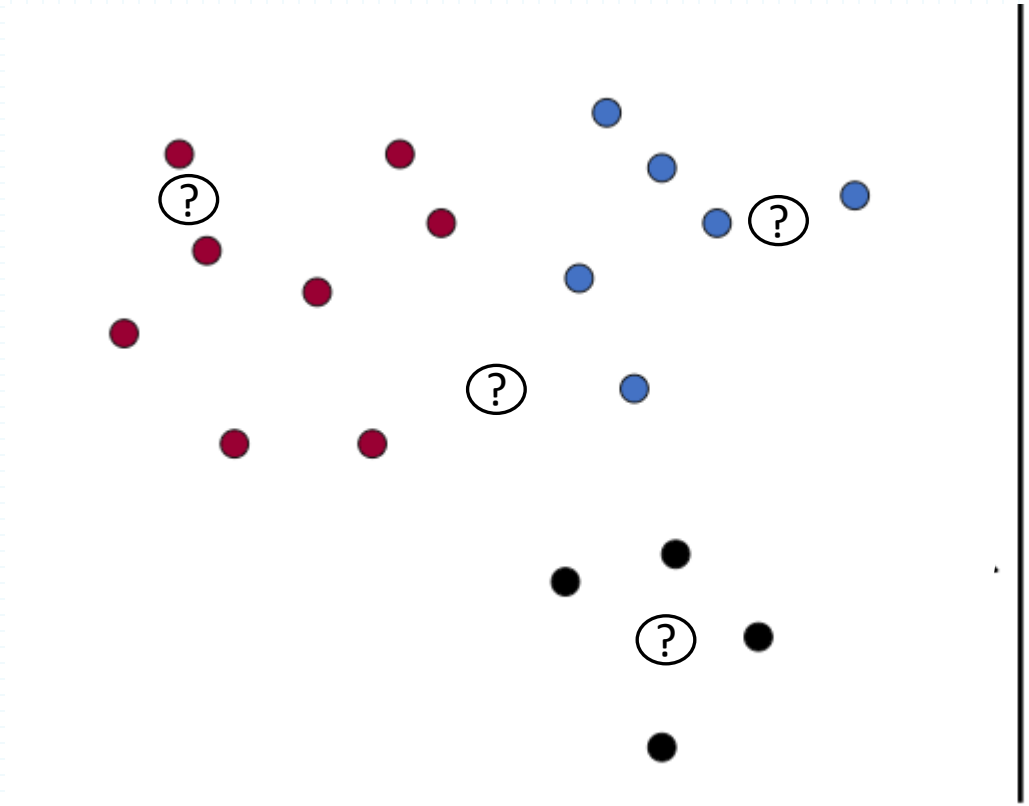
Vector Space Representation

- Each instance is represented as a vector of features.



Vector Space Representation

- Each instance is represented as a vector of features.



- Use closet training instances to predict the class of a new instance
- The instances themselves represent the knowledge!

Basic Idea

- *k*-NN classification rule is to assign to a test sample the majority category label of its *k* nearest training samples
- In practice, *k* is usually chosen to be odd, so as to avoid ties
- The *k* = 1 rule is generally called the nearest-neighbor classification rule

Bayes optimal classifier and NN

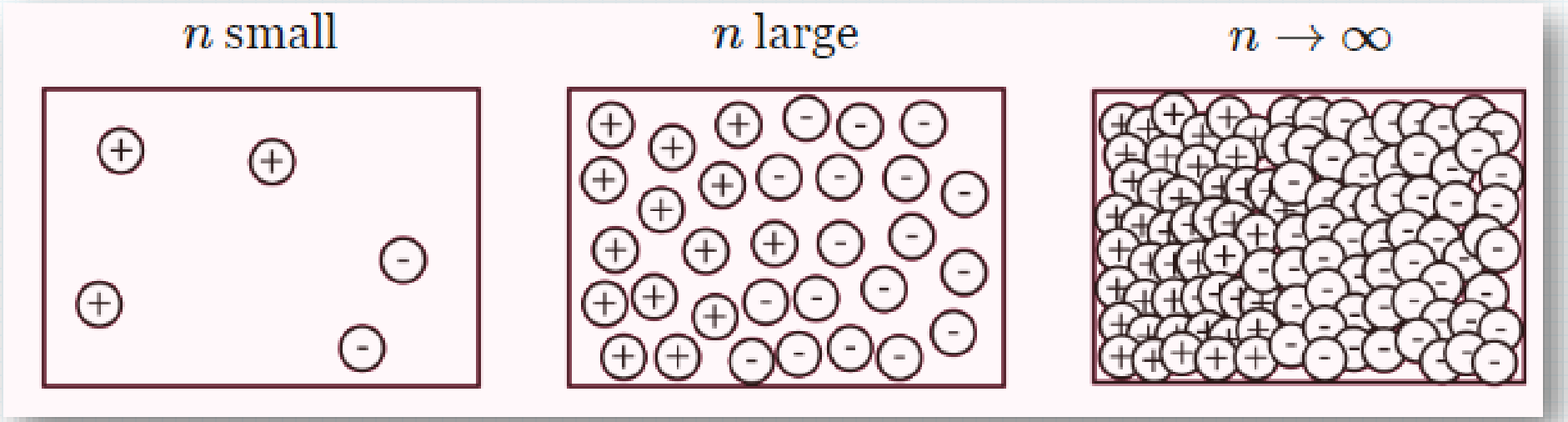
- Assume (and this is almost never the case) you knew P(y|x), then you would simply predict the most likely label.

$$\text{The Bayes optimal classifier predicts: } y^* = h_{\text{opt}}(\mathbf{x}) = \underset{y}{\text{argmax}}\, P(y|\mathbf{x})$$

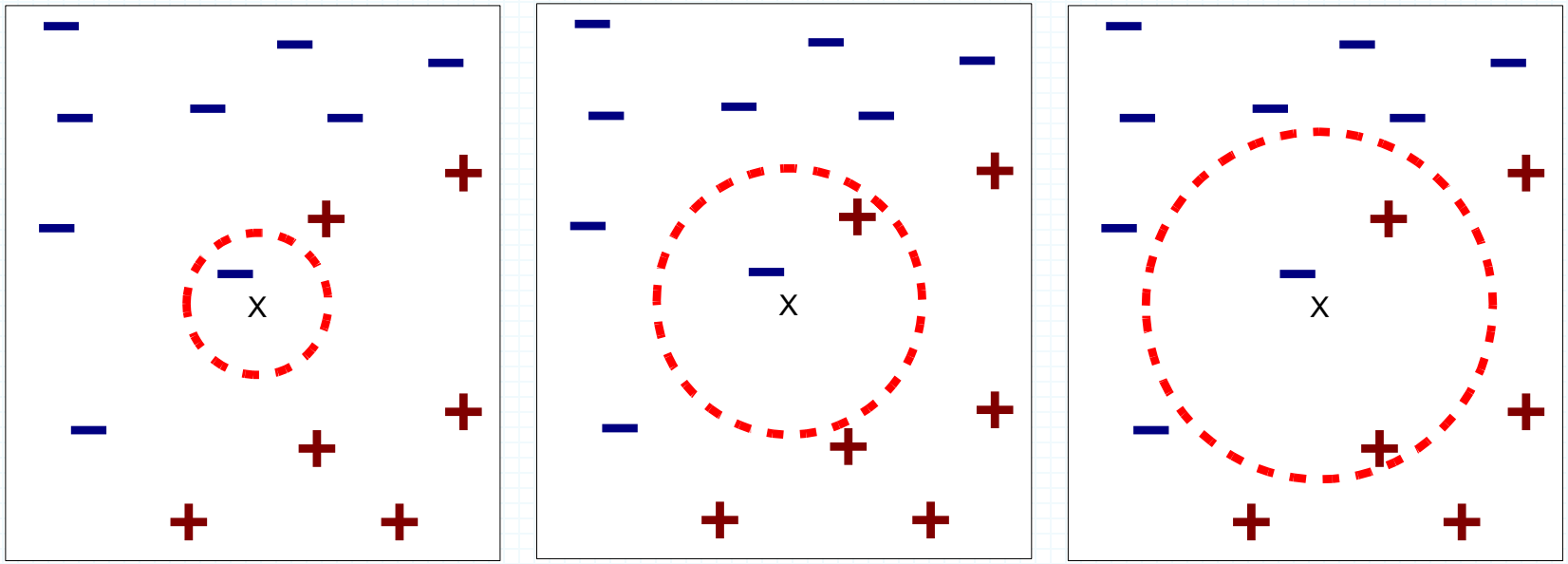
- Although the Bayes optimal classifier is as good as it gets, it still can make mistakes.
- **Why is the Bayes optimal classifier interesting, if it cannot be used in practice?** The reason is that it provides a highly informative lower bound of the error rate. With the same feature representation **no classifier can obtain a lower error.**

Bayes optimal classifier and NN

- As n→∞, the 1-NN classifier is only a factor 2 worse than the best possible classifier.



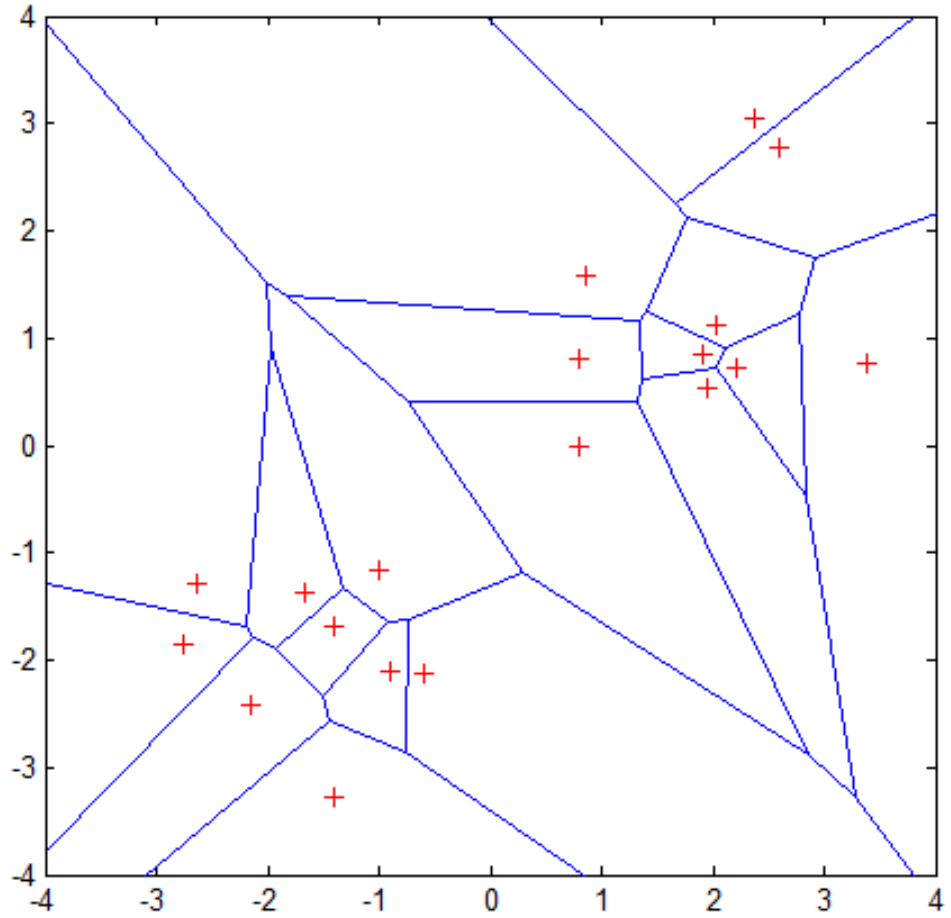
Definition of Nearest Neighbor



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

Voronoi Diagram

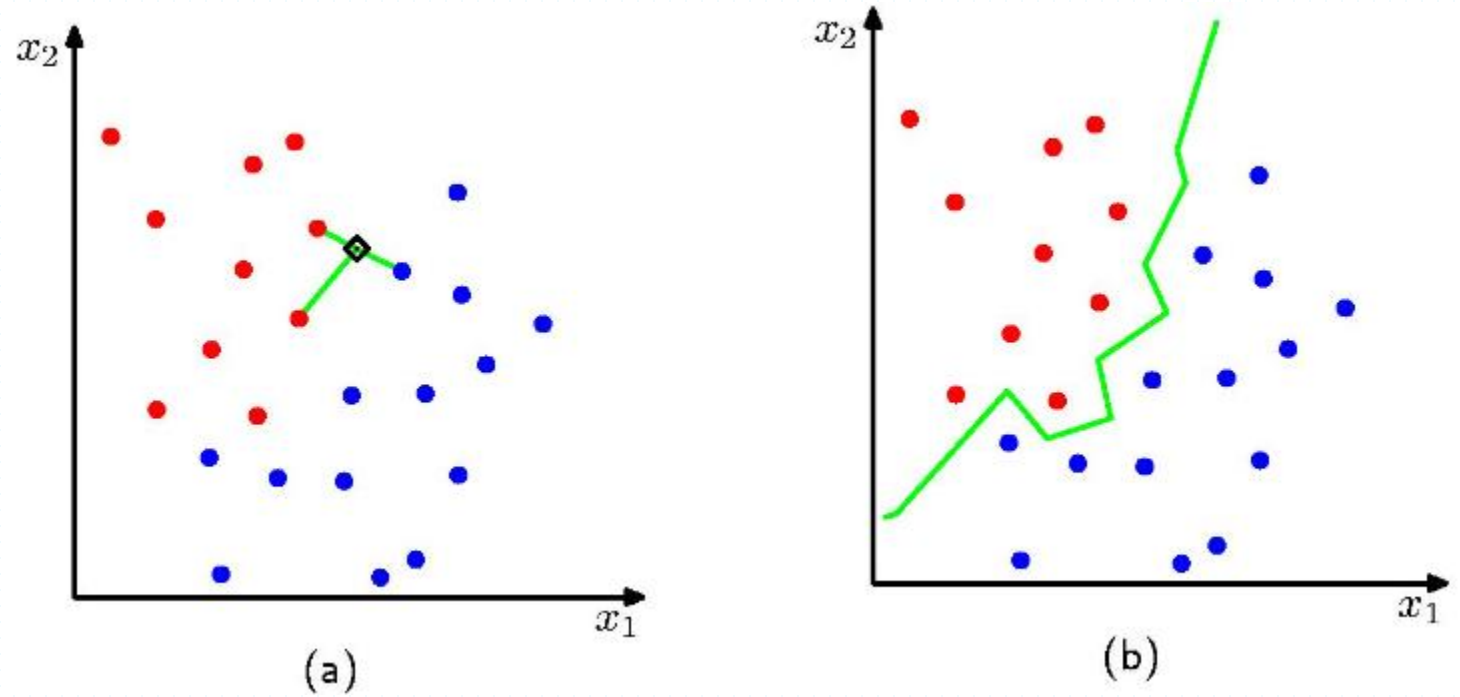


Properties:

1) All possible points within a sample's Voronoi cell are the nearest neighboring points for that sample
2) For any sample, the nearest sample is determined by the closest Voronoi cell edge

Decision boundary implemented by 3NN

- The boundary is always the perpendicular bisector of the line between two points (Vornoi tesselation)
  - k-nearest neighbors of a sample x are data points that have the k smallest distances to x



Nearest-Neighbor Classifiers: Issues

(1) The value of *k*, the number of nearest neighbors to retrieve

(2) Choice of Distance Metric to compute distance between records

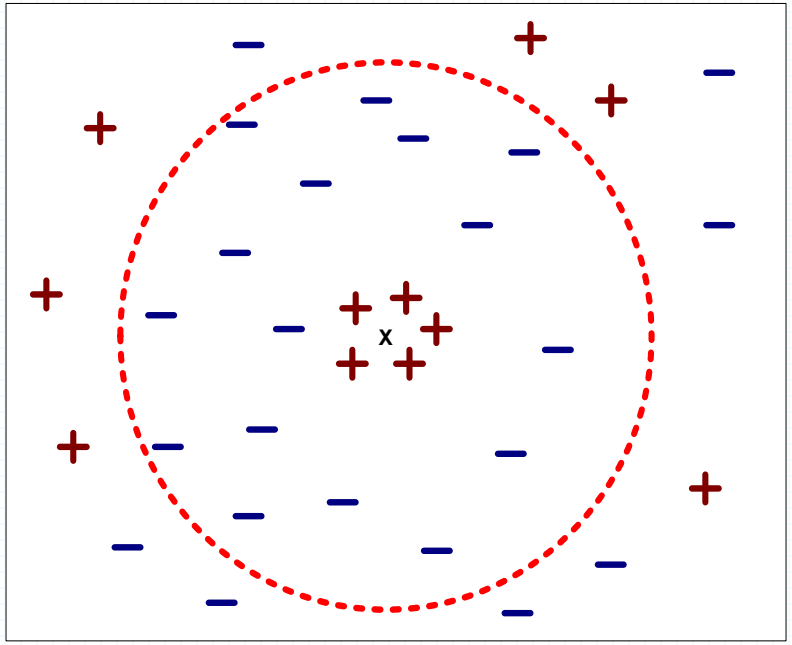
(3) The weight of each neighbor.

(4) Computational complexity

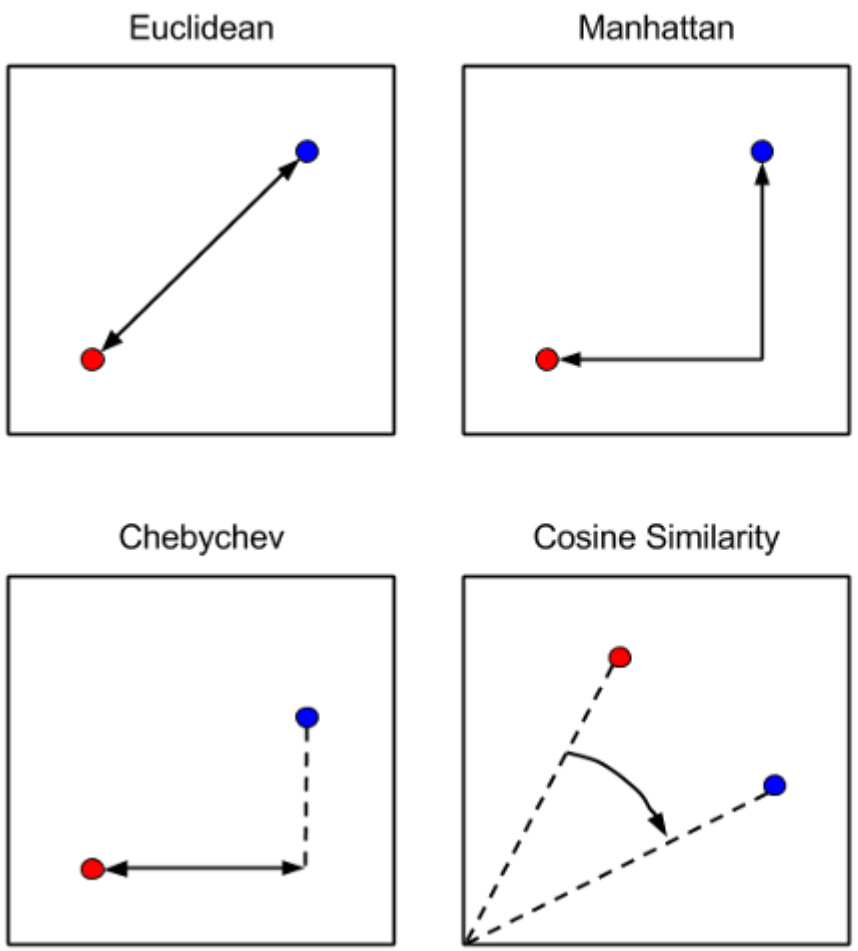
(1) Value of K

- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes

Rule of thumb:
K = sqrt(N)
N: number of training points



(2) Distance Metrics



Minkowsky:

$$D(\boldsymbol{x},\boldsymbol{y}) = \left(\sum_{i=1}^{m} |x_i - y_i|^r\right)^{1/r}$$

Euclidean:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan / city-block:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} |x_i - y_i|$$

Camberra:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|}$$

Chebychev:

$$D(\boldsymbol{x},\boldsymbol{y}) = \max_{i=1}^{m} |x_i - y_i|$$

Quadratic:

$$D(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x} - \boldsymbol{y})^T Q(\boldsymbol{x} - \boldsymbol{y}) = \sum_{j=1}^{m}\left(\sum_{i=1}^{m}(x_i - y_i)q_{ji}\right)(x_j - y_j)$$

Q is a problem-specific positive definite $m \times m$ weight matrix

Mahalanobis:

$$D(\boldsymbol{x},\boldsymbol{y}) = [\det V]^{1/m}(\boldsymbol{x} - \boldsymbol{y})^{\mathrm{T}} V^{-1}(\boldsymbol{x} - \boldsymbol{y})$$

V is the covariance matrix of $A_1..A_m$, and A_j is the vector of values for attribute j occuring in the training set instances $1..n$.

Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum_{i=1}^{m}(x_i - \overline{x_i})(y_i - \overline{y_i})}{\sqrt{\sum_{i=1}^{m}(x_i - \overline{x_i})^2 \sum_{i=1}^{m}(y_i - \overline{y_i})^2}}$$

$\overline{x_i} = \overline{y_i}$ and is the average value for attribute i occuring in the training set.

Chi-square:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{1}{sum_i}\left(\frac{x_i}{size_x} - \frac{y_i}{size_y}\right)^2$$

sum_i is the sum of all values for attribute i occuring in the training set, and $size_x$ is the sum of all values in the vector $\boldsymbol{x}$.

Kendall's Rank Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = 1 - \frac{2}{n(n-1)}\sum_{i=1}^{m}\sum_{j=1}^{i-1}\mathrm{sign}(x_i - x_j)\mathrm{sign}(y_i - y_j)$$

sign(x)=-1, 0 or 1 if $x < 0$, $x = 0$, or $x > 0$, respectively.

Figure 1. Equations of selected distance functions. ($\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors of m attribute values).

(2) Distance Measure: Scale Effects

- Different features may have different measurement scales
  - E.g., patient weight in kg (range [50,200]) vs. blood protein values in ng/dL (range [-3,3])
- Consequences
  - Patient weight will have a much greater influence on the distance between samples
  - May bias the performance of the classifier

(2) Distance Measure: Standardization

- Transform raw feature values into z-scores

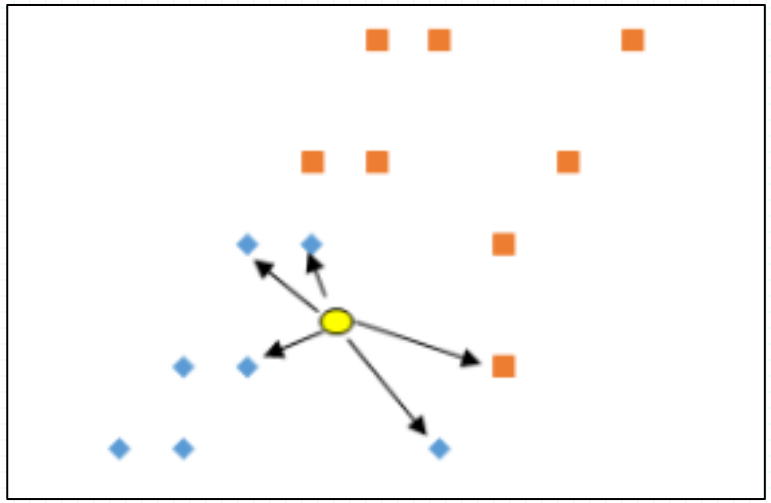
$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}$$

  - x_{ij} is the value for the i^{th} sample and j^{th} feature
  - μ_j is the average of all x_{ij} for feature j
  - σ_j is the standard deviation of all x_{ij} over all input samples

- Range and scale of z-scores should be similar (providing distributions of raw feature values are alike)

(3) The weight of each neighbor.

Closer neighbors should have more influence than farther ones.



- Each neighbor's **contribution to the final prediction** is **scaled by its distance** to the query point.

(4) Computational Complexity

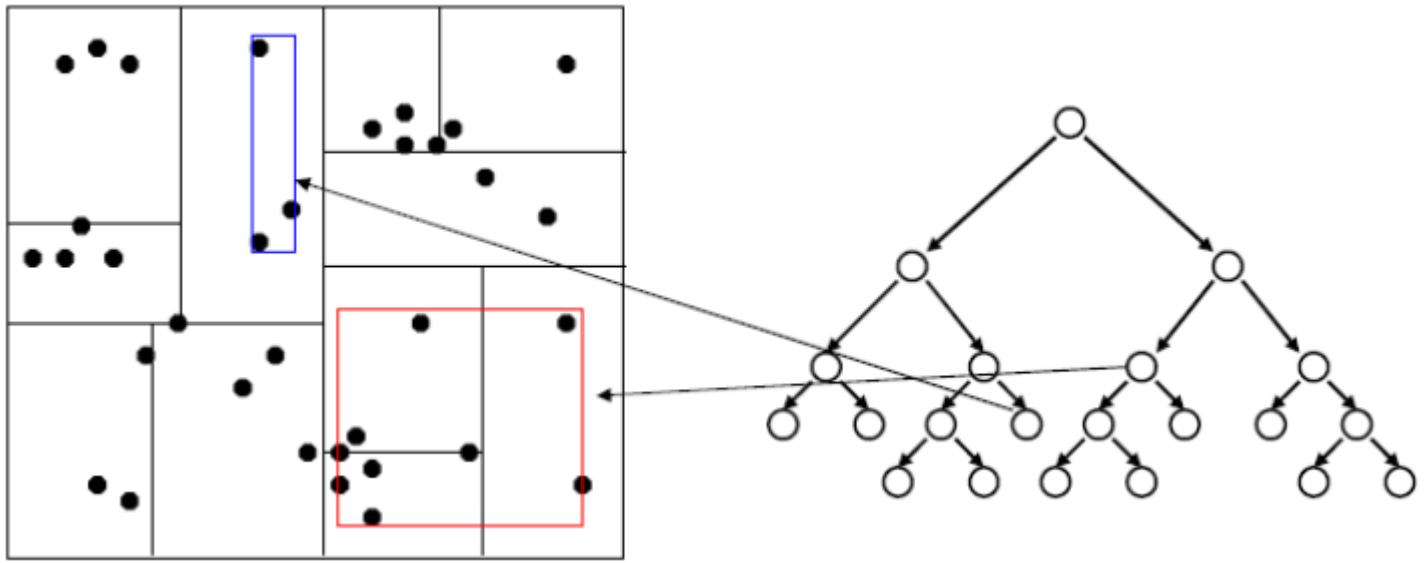
- Expensive
  - To determine the nearest neighbour of a query point *q*, must compute the distance to all *N* training examples
- Solutions!

  (A) Pre-sort training examples into fast data structures (kd-trees)

  (B) Remove redundant data (condensing)

Solution (A) kd-tree: Building the tree

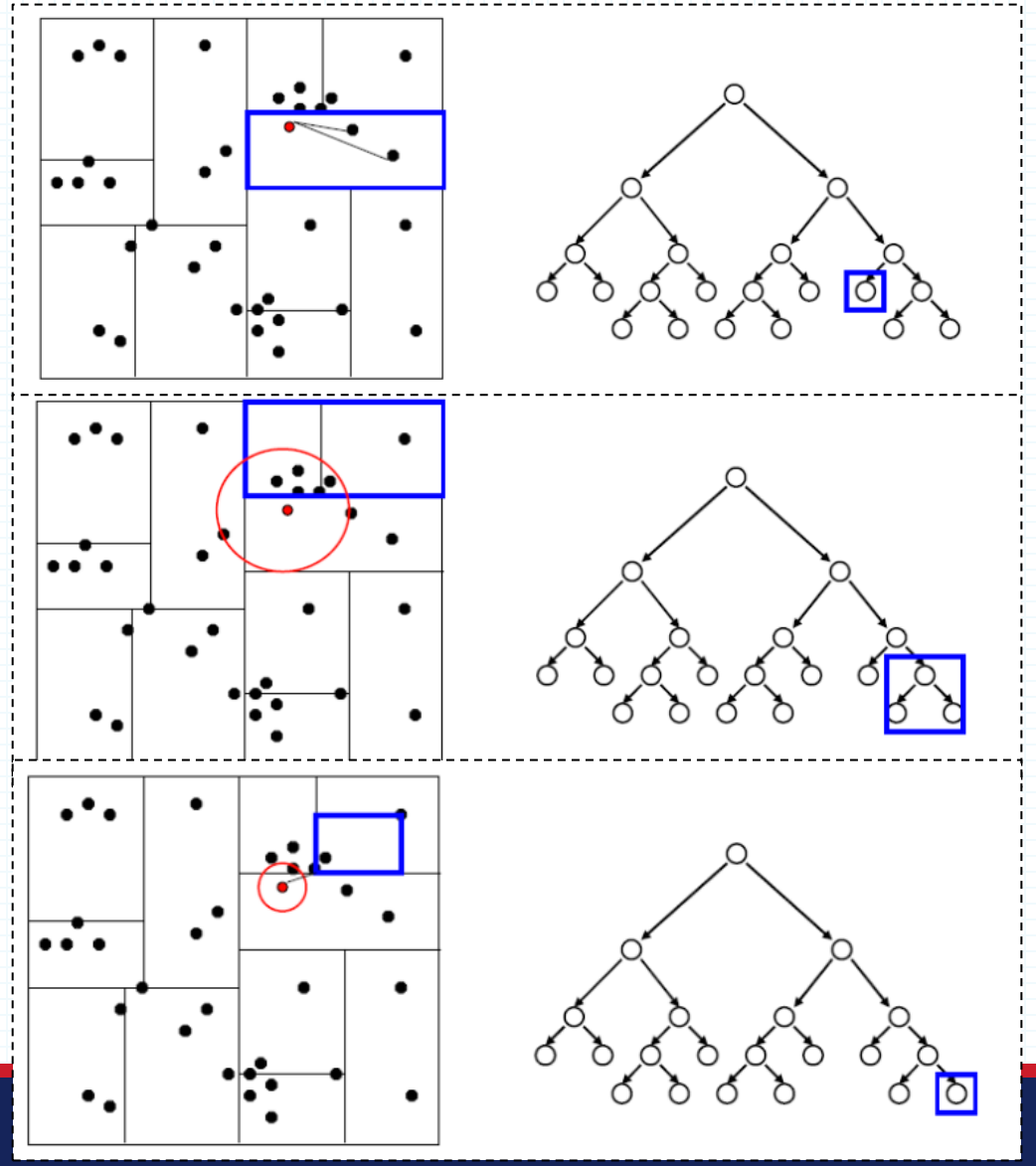
- Recursively divide the space by selecting a dimension and a median point.
- At each level, alternate the splitting dimension.
- Each node represents a hyperplane splitting the space into two halves.



Solution (B) kd-tree: Find NN

1.Traverse Tree: Go down to the leaf node where the target point would be.

2.Backtrack: On the way up, check whether sibling branches could contain closer points (using distance to splitting hyperplane).
If the distance is less than the distance to the current furthest neighbor in the heap, the other branch might have a closer point — so explore it.



(4) Computational Complexity

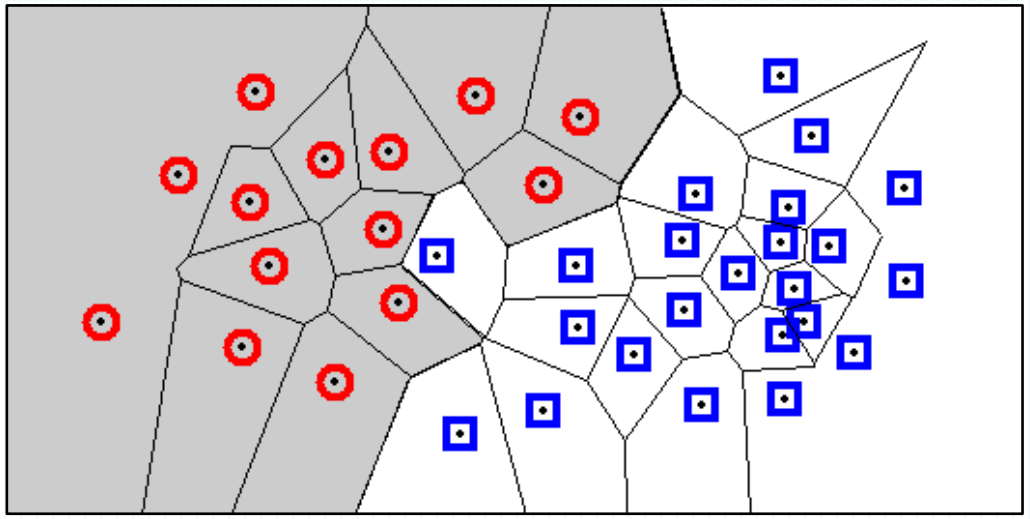
- Expensive
  - To determine the nearest neighbour of a query point *q*, must compute the distance to all *N* training examples
- Solutions!

  (A) Pre-sort training examples into fast data structures (kd-trees)

  (B) Remove redundant data (condensing)

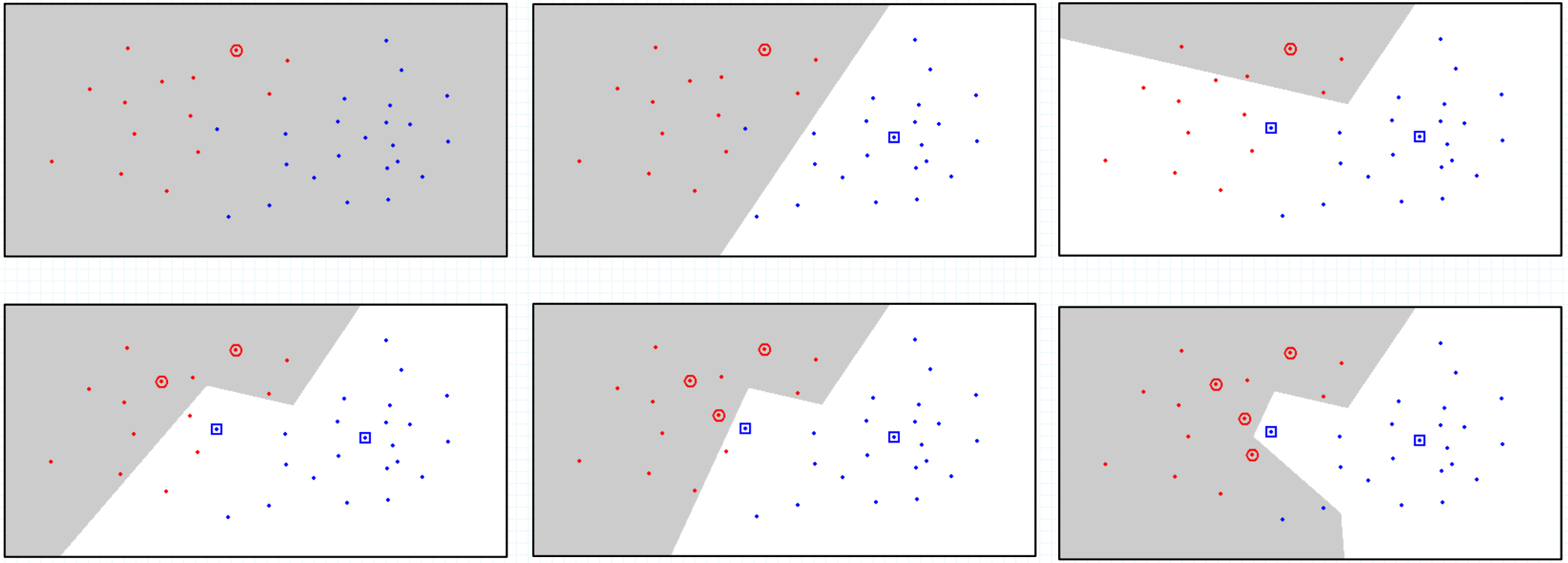
Solution (B) Condensation

- Reducing the size of a KD-Tree by removing unnecessary or redundant nodes.
- Keep only the samples that are needed to define the decision boundary
- Minimum Consistent Set – the smallest subset of the training data that correctly classifies all of the original training data



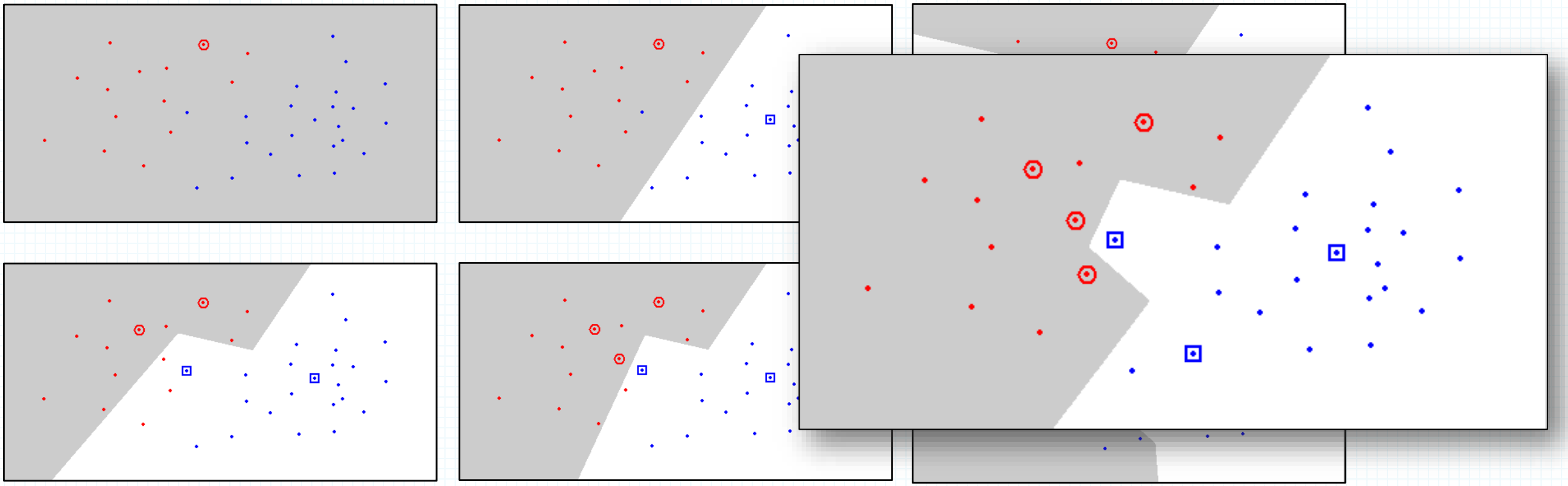
Solution (B) Condensing

1. Initialize subset with a single (or K) training example
2. Classify all remaining samples using the subset, and transfer any incorrectly classified samples to the subset
3. Return to 2 until no transfers occurred or the subset is full



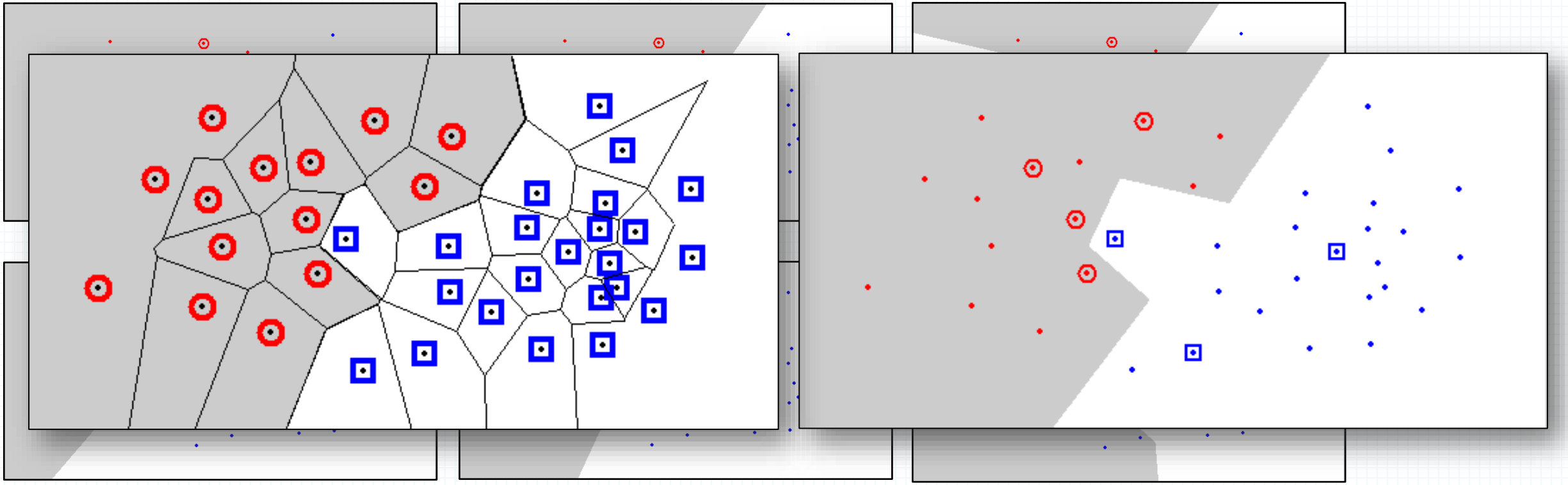
Solution (B) Condensing

1. Initialize subset with a single (or K) training example
2. Classify all remaining samples using the subset, and transfer any incorrectly classified samples to the subset
3. Return to 2 until no transfers occurred or the subset is full



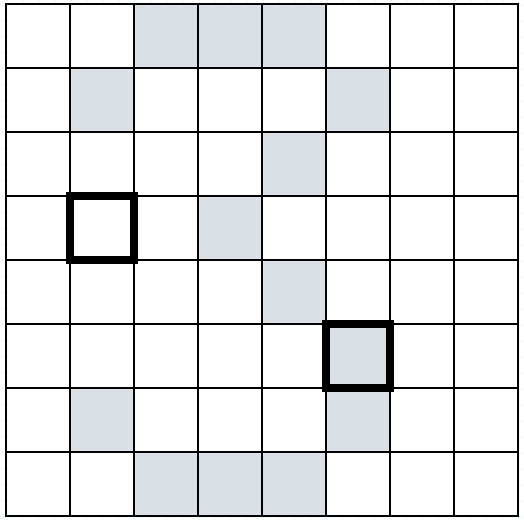
Solution (B) Condensing

1. Initialize subset with a single (or K) training example
2. Classify all remaining samples using the subset, and transfer any incorrectly classified samples to the subset
3. Return to 2 until no transfers occurred or the subset is full



Naïve Bayes Classifier

Example: OCR



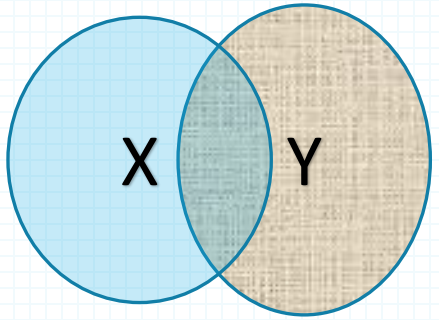
The theory!

- **Our main goal in machine learning:**

  Estimate P(Y|D) Y is the output, D is the data.

- For a specific input X,

  What is the meaning of P(Y|X)? And how can we calculate it using D?
  Remember: P(Y|X)=P(X,Y)/P(X)



- Is it feasible? Why?
- Btw, Why KNN is a good estimator for the last formula?

The theory!

- Naïve Bayes solution :

$$P(Y|X) = \frac{P(X|Y).P(Y)}{P(X)}$$

Posterior Probability

prior probability
Not a big problem

The likelihood
Generative Probability
(the main challenge)

We can forgot it, it is just a normalizer!

So:

$$\hat{y} = \underset{y}{arg\max}\big(P(Y|X)\big) = \underset{y}{arg\max}\left(\frac{P(X|Y).P(Y)}{P(X)}\right) = \underset{y}{arg\max}\big(P(X|Y).P(Y)\big)$$

Naïve Bayes solution

$$P(X = x|Y = y) = P(X_1 = x_1, \dots, X_n = x_n|Y = y) \approx P(X_1 = x_1|Y = y) \dots . P(X_n = x_n|Y = y)$$

The theory!

Multiplying lots of probabilities, which are between 0 and 1 by definition, can result in floating-point underflow.

Then:

$$\hat{y}=\underset{y}{arg\max}(P(X|Y).P(Y)) = \underset{y}{arg\max}(P(Y).\prod_i P(X_i|Y)) =$$

$$= \underset{y}{arg\max}\left(\log(P(Y)) + \sum_i \log(P(X_i|Y))\right)$$

What should we learn from the data in Naïve Bayes?

Example: Play Tennis

What do we need in order to use Naïve Bayes?

PlayTennis: training examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Example: Play Tennis

$P(\text{Play}=Yes) =$ $P(\text{Play}=No) =$

$P(X|Y)$

Outlook	Play=*Yes*	Play=*No*
Sunny		
Overcast		
Rain		

Temp	Play=*Yes*	Play=*No*
Hot		
Mild		
Cool		

Humidity	Play=*Yes*	Play=No
High		
Normal		

Wind	Play=*Yes*	Play=*No*
Strong		
Weak		

PlayTennis: training examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Example: Play Tennis

$P(\text{Play}=Yes) = 9/14$ $P(\text{Play}=No) = 5/14$

$P(X|Y)$

Outlook	Play=Yes	Play=No
Sunny	2/9	3/5
Overcast	4/9	0/5
Rain	3/9	2/5

Temp	Play=Yes	Play=No
Hot	2/9	2/5
Mild	4/9	2/5
Cool	3/9	1/5

Humidity	Play=Yes	Play=No
High	3/9	4/5
Normal	6/9	1/5

Wind	Play=Yes	Play=No
Strong	3/9	3/5
Weak	6/9	2/5

PlayTennis: training examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Example

- Test Phase
  - Given a new instance,
    $\mathbf{x}'$=(Outlook=*Sunny*, Temperature=*Cool*, Humidity=*High*, Wind=*Strong*)
  - Prediction

$P(Yes|\mathbf{x}')$: [P(*Sunny*|Yes)P(*Cool*|*Yes*)P(*High*|Yes)P(*Strong*|*Yes*)]P(Play=*Yes*) = 0.0053

$P(No|\mathbf{x}')$: [P(*Sunny*|No) P(*Cool*|No)P(*High*|*No*)P(*Strong*|*No*)]P(Play=*No*) = 0.0206

Given the fact $P(Yes|\mathbf{x}') < P(No|\mathbf{x}')$, we label $\mathbf{x}'$ to be "*No*".

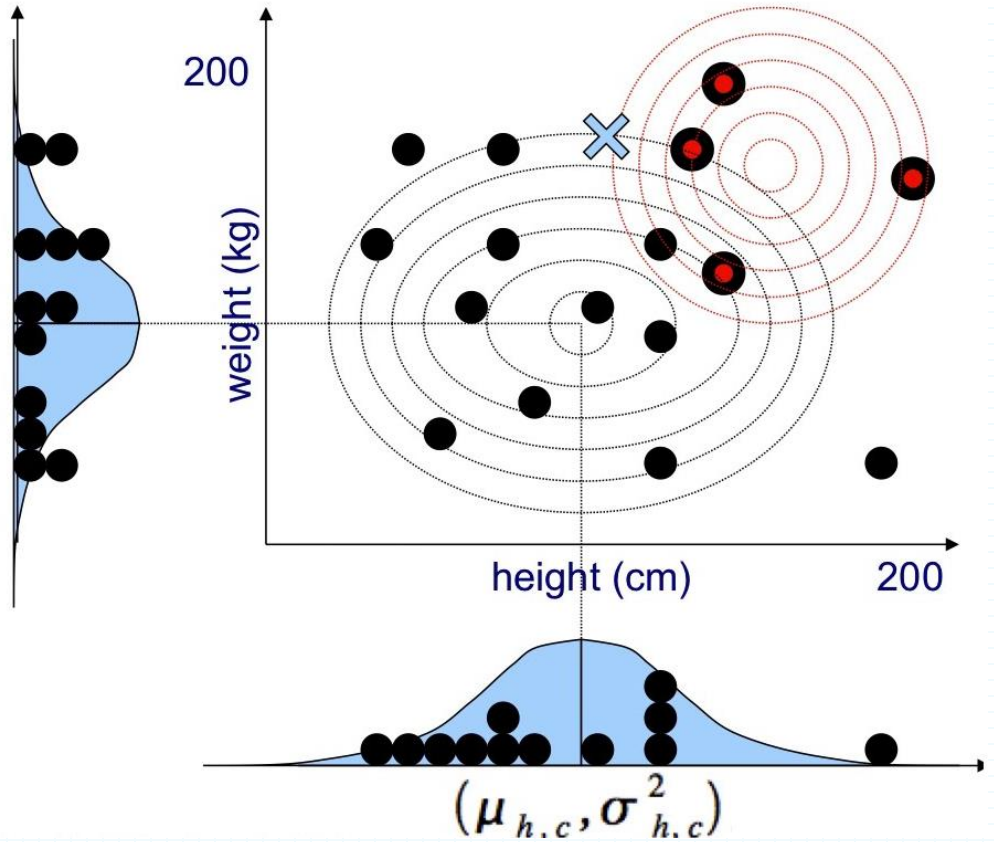
Gaussian Naïve Bayes (Continuous features)

- Assumes that each class follow a Gaussian distribution.

$$P(x_\alpha \mid y = c) = \mathcal{N}\left(\mu_{\alpha c}, \sigma^2_{\alpha c}\right)$$

$$= \frac{1}{\sqrt{2\pi}\sigma_{\alpha c}} e^{-\frac{1}{2}\left(\frac{x_\alpha - \mu_{\alpha c}}{\sigma_{\alpha c}}\right)^2}$$

- Each feature α comes from a class-conditional Gaussian distribution.



Gaussian Naïve Bayes (Continuous features)

- **Parameter estimation:**
  - As always, we estimate the parameters of the distributions for each dimension and class independently.
  - Gaussian distributions only have two parameters, the mean and variance.
  - The mean is estimated by the average feature value of dimension α from all samples with label y.
  - The (squared) standard deviation is simply the variance of this estimate.

$$\mu_{\alpha c} \leftarrow \frac{1}{n_c} \sum_{i=1}^{n} I(y_i = c) x_{i\alpha} \qquad \text{where } n_c = \sum_{i=1}^{n} I(y_i = c)$$

$$\sigma^2_{\alpha c} \leftarrow \frac{1}{n_c} \sum_{i=1}^{n} I(y_i = c)(x_{i\alpha} - \mu_{\alpha c})^2$$

Overfitting and Naïve Bayes Classifier

Overfitting and NB

$P(\text{features}, C = 2)$

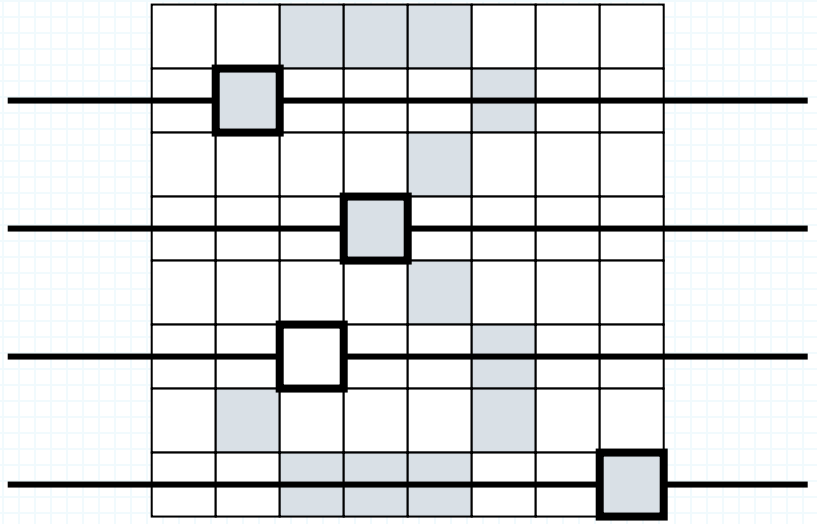
$P(C = 2) = 0.1$

$P(\text{on}|C = 2) = 0.8$

$P(\text{on}|C = 2) = 0.1$

$P(\text{off}|C = 2) = 0.1$

$P(\text{on}|C = 2) = 0.01$



$P(\text{features}, C = 3)$

$P(C = 3) = 0.1$

$P(\text{on}|C = 3) = 0.8$

$P(\text{on}|C = 3) = 0.9$

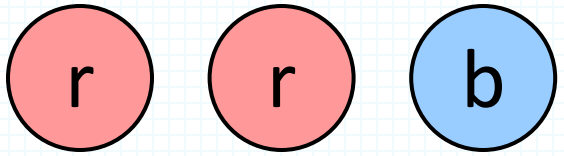
$P(\text{off}|C = 3) = 0.7$

$P(\text{on}|C = 3) = 0.0$

2 wins!!

Laplace Smoothing

- Laplace's estimate:
  - Pretend you saw every outcome once more than you actually did



$$P_{LAP}(x) = \frac{c(x)+1}{\sum_x [c(x)+1]}$$

$$= \frac{c(x)+1}{N+|X|}$$

$$P_{ML}(X) =$$

$$P_{LAP}(X) =$$

Laplace Smoothing

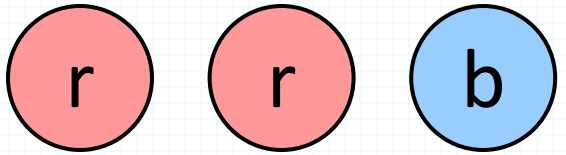
- Laplace's estimate (extended):
  - Pretend you saw every outcome k extra times

$$P_{LAP,k}(x) = \frac{c(x) + k}{N + k|X|}$$

  - What's Laplace with k = 0?
  - k is the strength of the prior

- Laplace for conditionals:
  - Smooth each condition independently:

$$P_{LAP,k}(x|y) = \frac{c(x,y) + k}{c(y) + k|X|}$$



$$P_{LAP,0}(X) =$$

$$P_{LAP,1}(X) =$$

$$P_{LAP,100}(X) =$$

Estimation: Linear Interpolation*

- In practice, Laplace often performs poorly for P(X|Y):
  - When |X| is very large
  - When |Y| is very large

- Another option: linear interpolation
  - Also get the empirical P(X) from the data
  - Make sure the estimate of P(X|Y) isn't too different from the empirical P(X)

$$P_{LIN}(x|y) = \alpha \hat{P}(x|y) + (1.0 - \alpha)\hat{P}(x)$$

  - What if α is 0? 1?