# Supervised learning

Data Science, AU, Fall 2023

Ira Assent

# Where we are…

1. 11/10 Introduction, data preprocessing, PCA, clustering
2. 13/10 Unsupervised learning, more clustering, outlier detection
3. 23/10 Supervised learning, classical machine learning: DT, SVMs,…
4. 26/10 Neural networks, pitfalls, outlook

- We looked at unsupervised learning
  - Density-based clustering:
    - robust to noise, arbitrary cluster shapes,
    - complex, difficult parameter choices,
    - hierarchical approaches: overview, help with parameter settings
  - Outlier detection: find issues or rare events in data
    - Inlier / outlier: e.g. statistics
    - Ranking approaches: e.g. LOF, density-based local

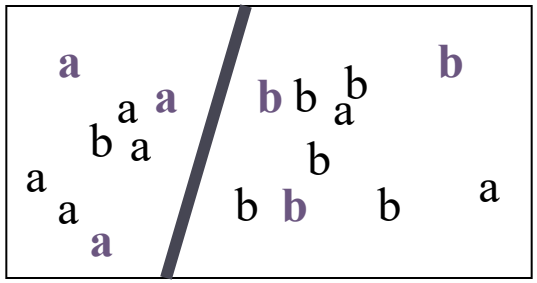
Any comments? Questions?

# Today's learning goals

- Classification
  - Supervised learning
    - Training data to learn about pre-defined classes, make predictions
- Classical machine learning models
  - Before current wave of deep neural networks
  - Decision Trees, Naïve Bayes,…
  - Still have their value
    - In particular, small(er) data sets
    - Interpretable
    - efficient
- Classification evaluation measures
  - How do we know that the model is good?
  - Different types depending on what is important in the performance
- Revisiting outlier detection
  - Using decision trees for the purpose of finding outliers

# Classification

- Class labels are known for a small set of “training data”: Find models/functions/rules (based on attribute values of the training examples) that
  - describe and distinguish classes
  - predict class membership for “new” objects




- Applications
  - Classify gene expression values for tissue samples to predict disease type and suggest best possible treatment
  - Automatic assignment of categories to large sets of newly observed celestial objects
  - Predict unknown or missing values (→ KDD pre-processing step)
  - . . .

# Classification Problem

- Given
  - a $d$-dimensional data space $D$ with attributes $a_i$, $i = 1, \ldots, d$
  - a set $C = \{c_1, \ldots, c_k\}$ of $k$ distinct class labels $c_j$, $j = 1, \ldots, k$
  - a set $O \subseteq D$ of objects, $o = (o_1, \ldots, o_d)$, with known class labels from $C$
- Searched
  - class label for objects from $D - O$, i.e. for objects with unknown class
  - a classifier $K: D \rightarrow C$
- Demarcation from clustering
  - Classification: classes are known in advance (a priori)
  - Clustering: classes are determined
- Related problem: prediction
  - Determine the value of a numerical attribute
  - Method: e.g., regression analysis

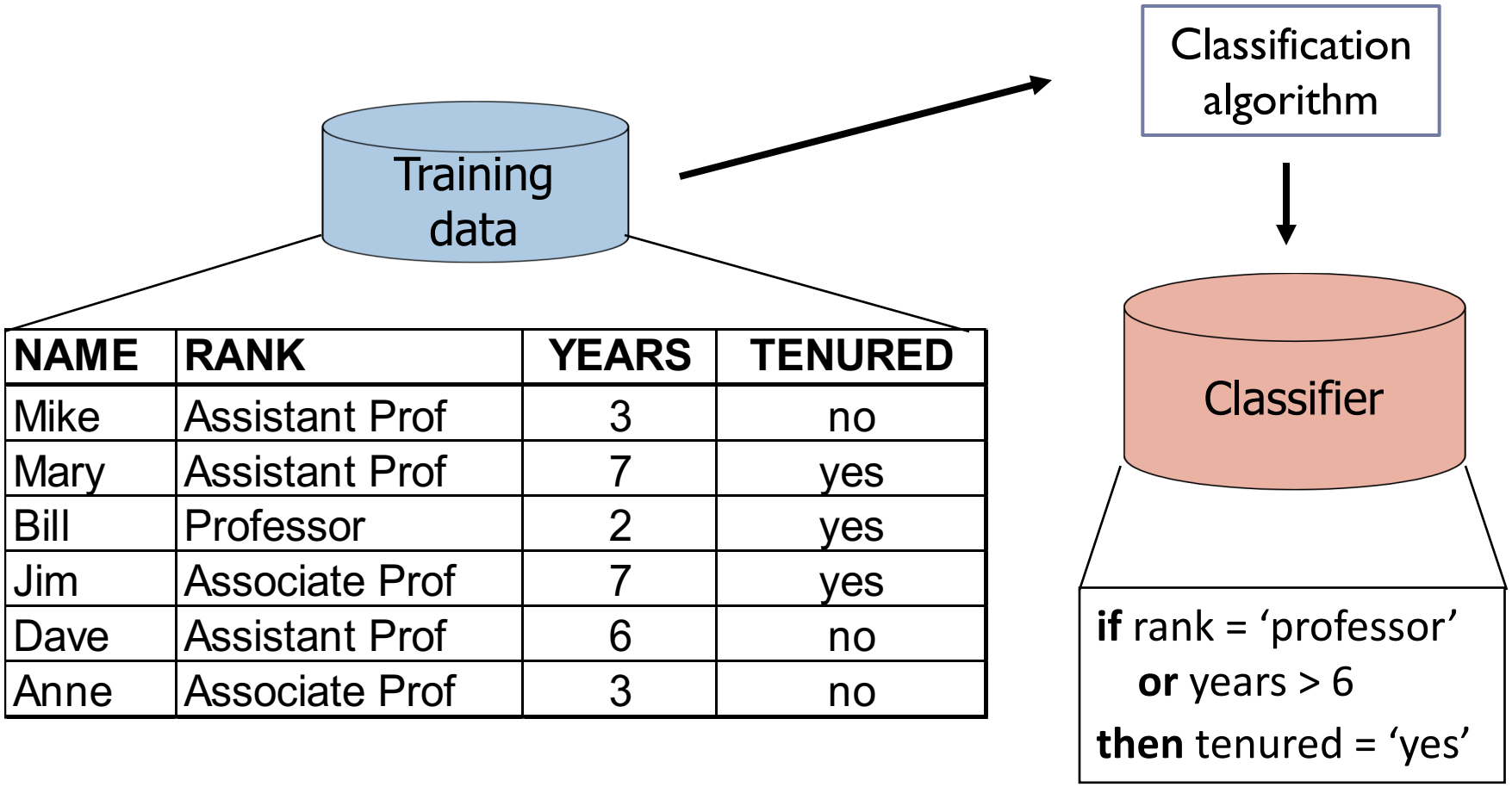
# Classification Example

| ID | age | car type | risk |
|---|---|---|---|
| 1 | 23 | family | high |
| 2 | 17 | sportive | high |
| 3 | 43 | sportive | high |
| 4 | 68 | family | low |
| 5 | 32 | truck | low |

- Simple Classifier

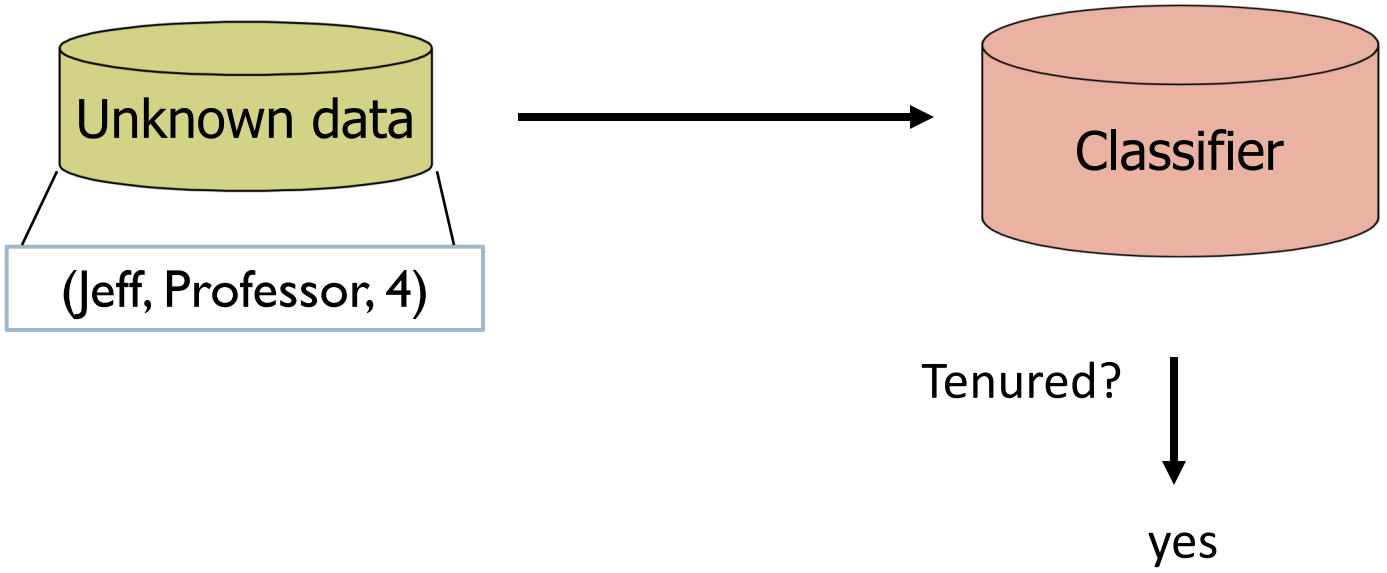
```
if age > 50 then risk = low;
if age ≤ 50 and car type = truck then risk = low;
if age ≤ 50 and car type ≠ truck then risk = high.
```

# Classification Process: Model Construction (= training phase)




| 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 |

# Classification Process: Model Application (= test phase)




- Goal is sometimes not to classify unknown data but to get a better understanding of the data

# Evaluation of Classifiers – Accuracy

- *Classification Accuracy*
  - Predict the class label for each object from a data set *o* (= test set)
  - Determine the fraction of correctly predicted class labels:

$$classification\ accuracy = \frac{count(\text{correctly predicted class label})}{count(o)}$$

  - Classification error = 1 – classification accuracy

# Evaluation of Classifiers – Notions

- *Overfitting*
  - Classifier is optimized to training data
  - May yield worse results for entire data set
  - Potential reasons
    - bad quality of training data (noise, missing values, wrong values)
    - different statistical characteristics of training data and test data

- *Train-and-Test*
  Decomposition of data set o into two partitions
  - *Training data* to train the classifier
    - construction of the model by using information about the class labels
  - *Test data* to evaluate the classifier
    - temporarily hide class labels, predict them anew and compare results with original class labels

# Evaluation of Classifiers – Cross Validation

- Train-and-Test is not applicable if the set of objects for which the class label is known is very small

- *m*-fold *Cross Validation*
  - Decompose data set evenly into *m* subsets of (nearly) the same size.
  - Iteratively use *m* – 1 partitions as training data and the remaining single partition as test data.
  - Combine the *m* classification accuracy values to an overall classification accuracy, and combine the *m* generated models to an overall model for the data.

# Evaluation of Classifiers – Leave-One-Out

- If data set is very small
- *Leave-one-out* is, in some sense, a degenerate variant of cross validation
  - For each of the objects *o* in the data set *D*:
    - Use set *D* – *o* as training set
    - Use the singleton set {*o*} as test set
    - Predict the class label of object *o*
  - Compute classification accuracy by dividing the number of correct predictions through the database size |D|
- Particularly well applicable to nearest-neighbor classifiers

# Quality Measures for Classifiers

- Classification accuracy
- Compactness of the model
  - decision tree size; number of decision rules
- Interpretability of the model
  - Insights and understanding provided by the model
- Efficiency
  - Time to generate the model (training time)
  - Time to apply the model (prediction time)
- Scalability for large databases
  - Efficiency in disk-resident databases
- Robustness
  - Robust against noise or missing values

# Evaluation

- Idea:
  - Train a classification model, then check how many of the training samples it gets correct

a) Good evaluation: accuracy of classifier
b) Poor evaluation: we may be too optimistic

# Quality Measures for Classifiers

- Let *K* be a classifier, $TR \subseteq O$ a training set, and $TE \subseteq O$ a test set. Let C(o) denote the correct class label of an object $o \in O$.
- *Classification Accuracy* of *K* on *TE:*

$$G_{TE}(K) = \frac{|\{o \in TE, K(o) = C(o)\}|}{|TE|}$$

- True classification error

$$F_{TE}(K) = \frac{|\{o \in TE, K(o) \neq C(o)\}|}{|TE|}$$

- Apparent classification error (not to be used in evaluation!)

$$F_{TR}(K) = \frac{|\{o \in TR, K(o) \neq C(o)\}|}{|TR|}$$

# Classifier Evaluation: Confusion Matrix

Confusion Matrix = Contingency Matrix:

| Actual class\Predicted class | $C_1$ | ¬ $C_1$ |
|---|---|---|
| $C_1$ | **True Positives (TP)** | **False Negatives (FN)** |
| ¬ $C_1$ | **False Positives (FP)** | **True Negatives (TN)** |

Example of Confusion Matrix:

| Actual class\Predicted class | buy_computer = yes | buy_computer = no | Total |
|---|---|---|---|
| buy_computer = yes | **6954** | **46** | 7000 |
| buy_computer = no | **412** | **2588** | 3000 |
| Total | 7366 | 2634 | 10000 |

- Given *m* classes, an entry, $\boldsymbol{CM_{i,j}}$ in a **confusion matrix** indicates # of tuples in class *i* that were labeled by the classifier as class *j*
- May have extra rows/columns to provide totals

# Classifier Evaluation Metrics

| A\P | C | ¬C | |
|---|---|---|---|
| C | **TP** | **FN** | **P** |
| ¬C | **FP** | **TN** | **N** |
| | **P'** | **N'** | **All** |

- **Classifier Accuracy,** or recognition rate: percentage of test set tuples that are correctly classified

  **Accuracy = (TP + TN)/All**

- **Error rate:** *1 – accuracy*, or

  **Error rate = (FP + FN)/All**

- **Class Imbalance Problem**:
  - One class may be *rare*, e.g. fraud, or HIV-positive
  - Significant *majority of the negative class* and minority of the positive class
  - **Sensitivity**: True Positive recognition rate
    - **Sensitivity = TP/P**
  - **Specificity**: True Negative recognition rate
    - **Specificity = TN/N**

# Classifier Evaluation Metrics seen before

- **Precision**: exactness – what % of tuples that the classifier labeled as positive are actually positive

$$precision = \frac{TP}{TP + FP}$$

- **Recall:** completeness – what % of positive tuples did the classifier label as positive?
- Perfect score is 1.0

$$recall = \frac{TP}{TP + FN}$$

- Inverse relationship between precision & recall
- ***F* measure (*$F_1$* or *F*-score)**: harmonic mean of precision and recall,

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

- ***$F_\beta$*:** weighted measure of precision and recall
  - assigns ß times as much weight to recall as to precision

$$F_\beta = \frac{(1 + \beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$

# Classifier Evaluation Metrics: Example

| Actual Class\Predicted class | cancer = yes | cancer = no | Total | Recognition(%) |
|---|---|---|---|---|
| cancer = yes | **90** | **210** | 300 | 30.00 (*sensitivity* |
| cancer = no | **140** | **9560** | 9700 | 98.56 (*specificity)* |
| Total | 230 | 9770 | 10000 | 96.40 (*accuracy*) |

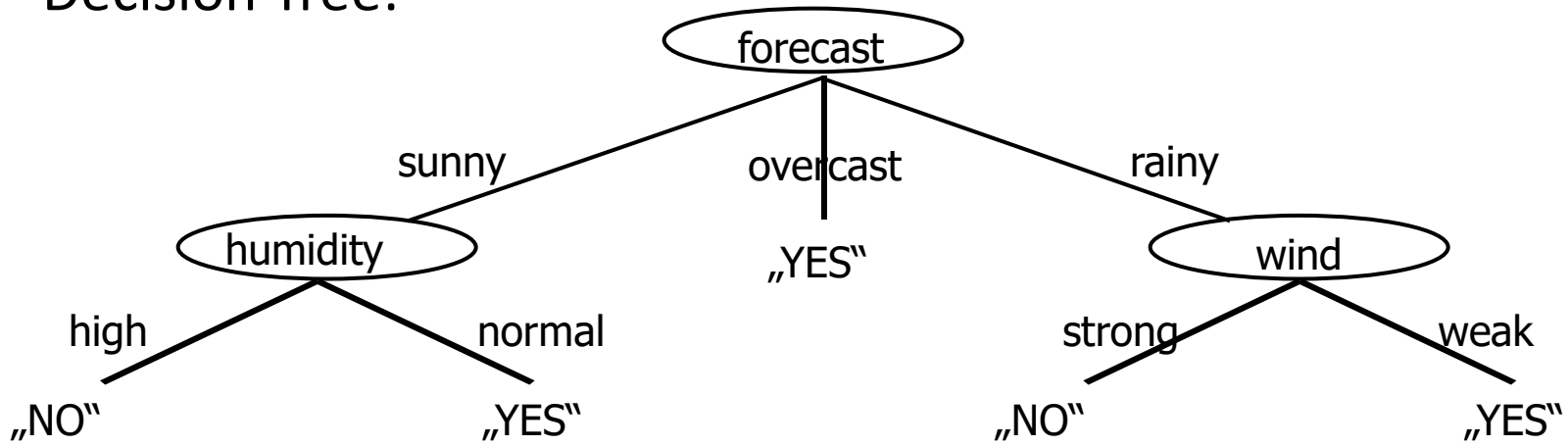
- *Precision* = 90/230 = 39.13% *Recall* = 90/300 = 30.00%

# Decision Tree Classifiers

- Query: How about playing tennis today?
- Training data set:

| day | forecast | temperature | humidity | wind | tennis decision |
|---|---|---|---|---|---|
| 1 | sunny | hot | high | weak | no |
| 2 | sunny | hot | high | strong | no |
| 3 | overcast | hot | high | weak | yes |
| 4 | rainy | mild | high | weak | yes |
| 5 | rainy | cool | normal | weak | yes |
| 6 | rainy | cool | normal | strong | no |
| 7 | . . . | . . . | . . . | . . . | . . . |

- Decision Tree:



# Decision Tree Classifiers: Basics

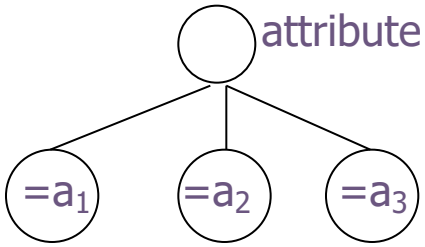
- Decision tree
  - A flow-chart-like tree structure
  - Internal node denotes a test on an attribute
  - Branch represents an outcome of the test
  - Leaf nodes represent class labels or class distribution
- Decision tree generation consists of two phases
  - Tree construction
    - At start, all the training examples are at the root
    - Partition examples recursively based on selected attributes
  - Tree pruning
    - Identify and remove branches that reflect noise or outliers
- Use of decision tree: Classifying an unknown sample
  - Traverse the tree and test attribute values of the sample against decision tree
  - Assign the class label of the respective leaf to the query object

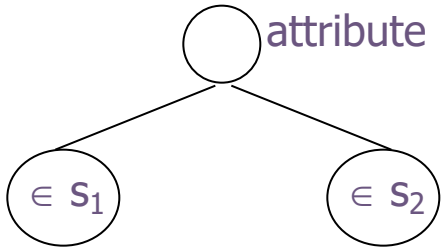
# Algorithm for Decision Tree Induction

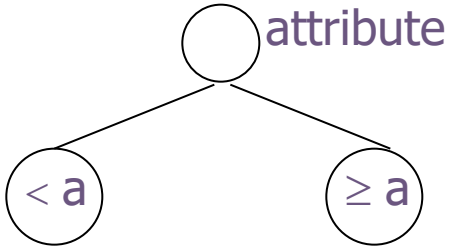
- Basic algorithm (a greedy algorithm)
  - Tree is created in a top-down recursive divide-and-conquer manner
  - Attributes may be categorical or continuous-valued
  - At start, all the training examples are assigned to the root node
  - Recursively partition the examples at each node and push them down to the new nodes
    - Select test attributes and determine split points or split sets for the respective values on the basis of a heuristic or statistical measure (*split strategy,* e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

# Split Strategies: Types of Splits

- Categorical attributes
  - split criteria based on equality „*attribute* = *a*“ or
  - based on subset relationships „*attribute* $\in$ *set*“
  - many possible choices (subsets)




- Numerical attributes
  - split criteria of the form „*attribute* < *a*“
  - many possible choices for the split point

# Split Strategies: Quality of Splits

- Given
  - a set $T$ of training objects
  - a (disjoint, complete) partitioning $T_1, T_2, \ldots, T_m$ of $T$
  - the relative frequencies $p_i$ of class $c_i$ in $T$
- Define
  - a measure for the heterogeneity of a set $S$ of training objects with respect to the class membership
  - a split of $T$ into partitions $T_1, T_2, \ldots, T_m$ such that the heterogeneity is minimized

- Proposals: Information gain, Gini index (based on entropy)
  - We skip details here, but they assess the goodness of the split in separating classes, i.e., reaching a clear classification

# Building Decision Trees

- Tree is created top-down
  - We greedily try to reduce our uncertainty about the class outcome (YES/NO)
  - Training examples T recursively partitioned into $T_1$, $T_2$, …, $T_m$
    - Entropy for $k$ classes with frequencies $p_i$ (information theory: measure of uncertainty)

$$entropy(T) = -\sum_{i=1}^{k} p_i \cdot log_2 p_i$$

    - Compute the information gain of a split using an attribute, such as humidity, by comparing the entropy before the split with the entropy of the split

$$information\ gain(T, A) = entropy(T) - \sum_{i=1}^{m} \frac{|T_i|}{|T|} \cdot entropy(T_i)$$

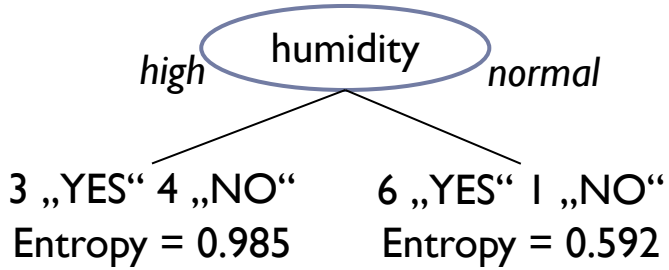
# Building Decision Trees

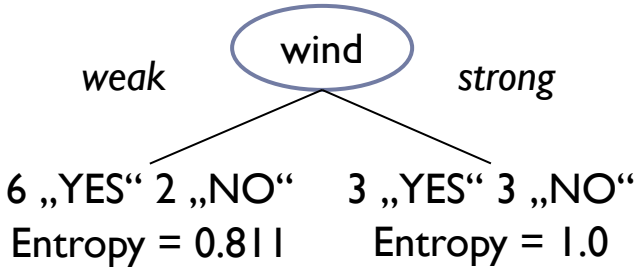
$$information\ gain(T, A) = entropy(T) - \sum_{i=1}^{m} \frac{|T_i|}{|T|} \cdot entropy(T_i)$$

$$entropy(T) = -\sum_{i=1}^{k} p_i \cdot log_2 p_i$$

For all training data: 9 „YES“ 5 „NO“ Entropy = 0.940

For humidity or wind:




$$IG(T, hum) = 0.94 - \frac{7}{14} * 0.985 - \frac{7}{14} * 0.592 = 0.151$$

$$IG(T, wind) = 0.94 - \frac{8}{14} * 0.811 - \frac{6}{14} * 1.0 = 0.048$$

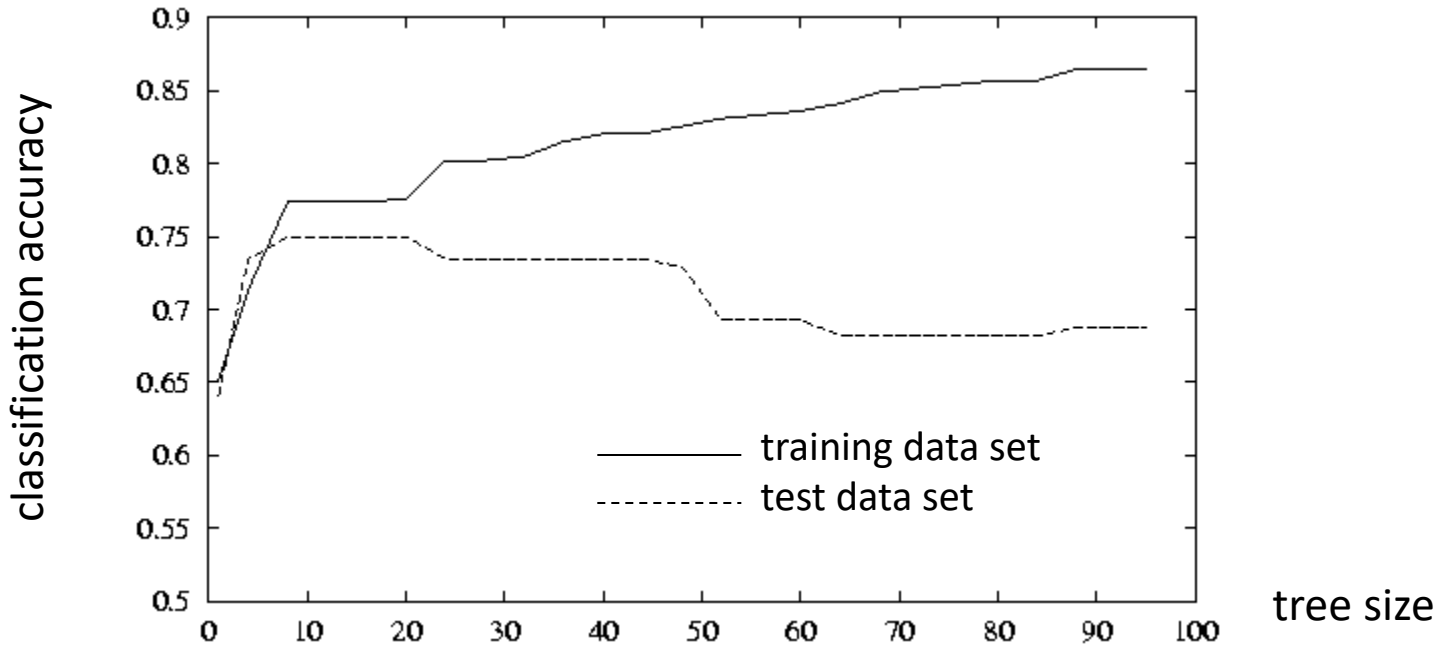
Result: humidity yields the highest information gain

# Avoid Overfitting in Classification

- The generated tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Result is in poor accuracy for unseen samples
- Two approaches to avoid overfitting
  - Prepruning: Halt tree construction early—do not split a node if this would result in the goodness measure falling below a threshold
    - Difficult to choose an appropriate threshold
  - Postpruning: Remove branches from a "fully grown" tree—get a sequence of progressively pruned trees
    - Use a set of data different from the training data to decide which is the "best pruned tree"

# Overfitting: Notion

- *Overfitting* occurs at the creation of a decision tree, if there are two trees *E* and *E´* for which the following holds:
  - on the training set, *E* has a smaller error rate than *E´*
  - on the overall data set, *E´* has a smaller error rate than *E*



# Overfitting: Avoidance

- Removal of noisy and **erroneous** training data
  - in particular, remove contradicting training data
- Choice of an appropriate size of the training set
  - not too small, not too large
- Choice of an appropriate value for minimum support
  - *minimum support:* minimum number of data objects a leaf node contains
  - in general, *minimum support* >> 1
- Choice of an appropriate value for minimum confidence
  - *minimum confidence:* minimum fraction of the majority class in a leaf node
  - typically, minimum confidence << 100%
  - leaf nodes can errors or noise in data records absorb
- Post pruning of the decision tree
  - pruning of overspecialized branches

# Pruning of Decision Trees: Approach

***Error-Reducing Pruning*** [Mitchell 1997]

- Decompose classified data into training set and test set
- Creation of a decision tree *E* for the training set
- Pruning of *E* by using the test set *T*
  - determine the subtree of *E* whose pruning reduces the classification error on *T* the most
  - remove that subtree
  - finished if no such subtree exists

- only applicable if a sufficient number of classified data is available

# Pruning of Decision Trees: Approach

***Minimal Cost Complexity Pruning***
[Breiman, Friedman, Olshen & Stone 1984]

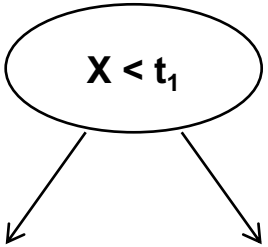
- Does not require a separate test set
  - applicable to small training sets as well

- New quality measure for decision trees: $CC_T(E,\alpha) = F_T(E) + \alpha \cdot |E|$
  - trade-off of classification error and tree size (| *E* | number of leaf nodes)
  - Smaller decision trees tend to yield better generalization
- complexity parameter $\alpha \geq 0$
  - zero means only error matters, large value: only tree size matters
- Prune always the weakest link (wrt CC), then select best E($\alpha_i$) using classification error on the overall data set by an l-fold cross validation on the training set

# Breathe deep

# Revisiting outlier detection

- Decision trees can help here as well!
- Basic idea: look at how isolated points are in different decision trees
  - Use the number of branches it takes to isolate them as a measure

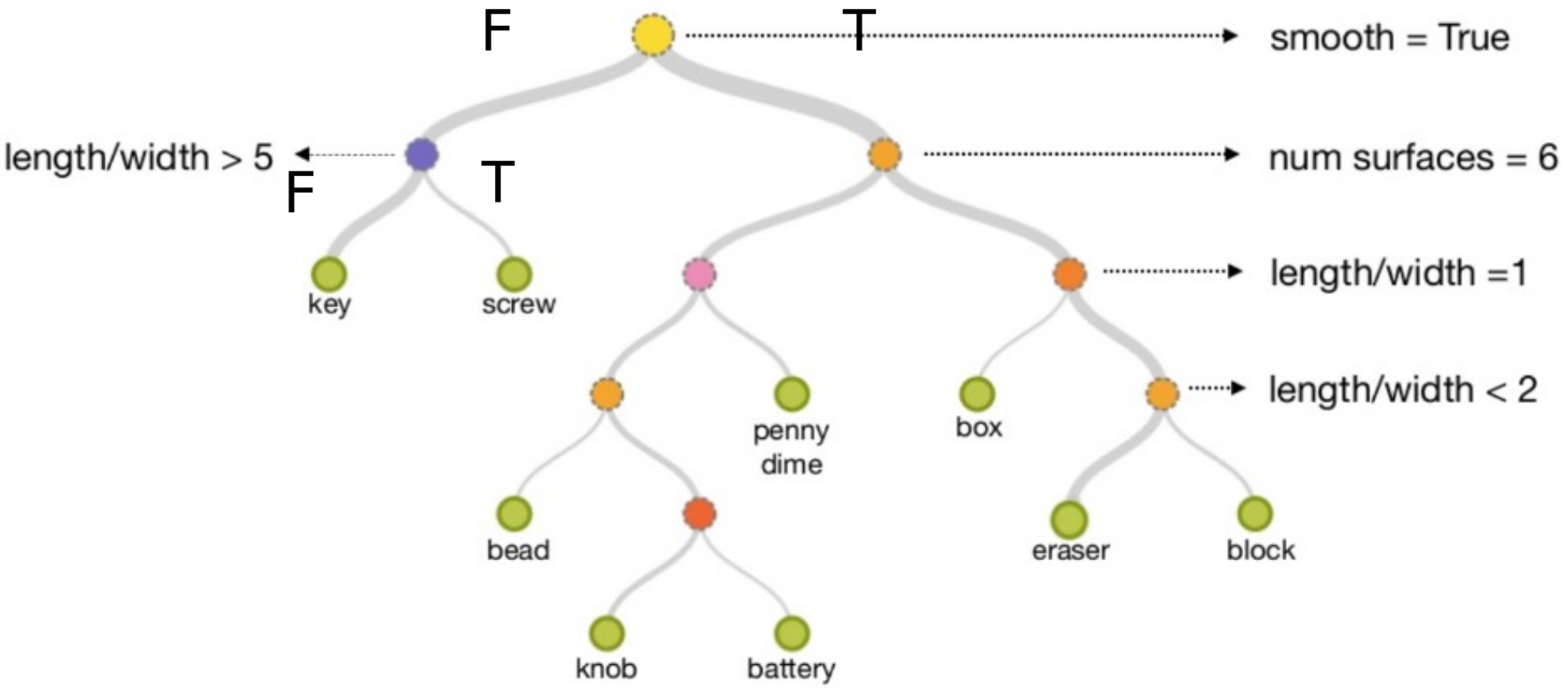




For outliers, their path lengths to the root node tend to be small

  - It's not really a classification problem – no classes or class labels known
    - Train a number of random trees using random data samples

# Random Trees

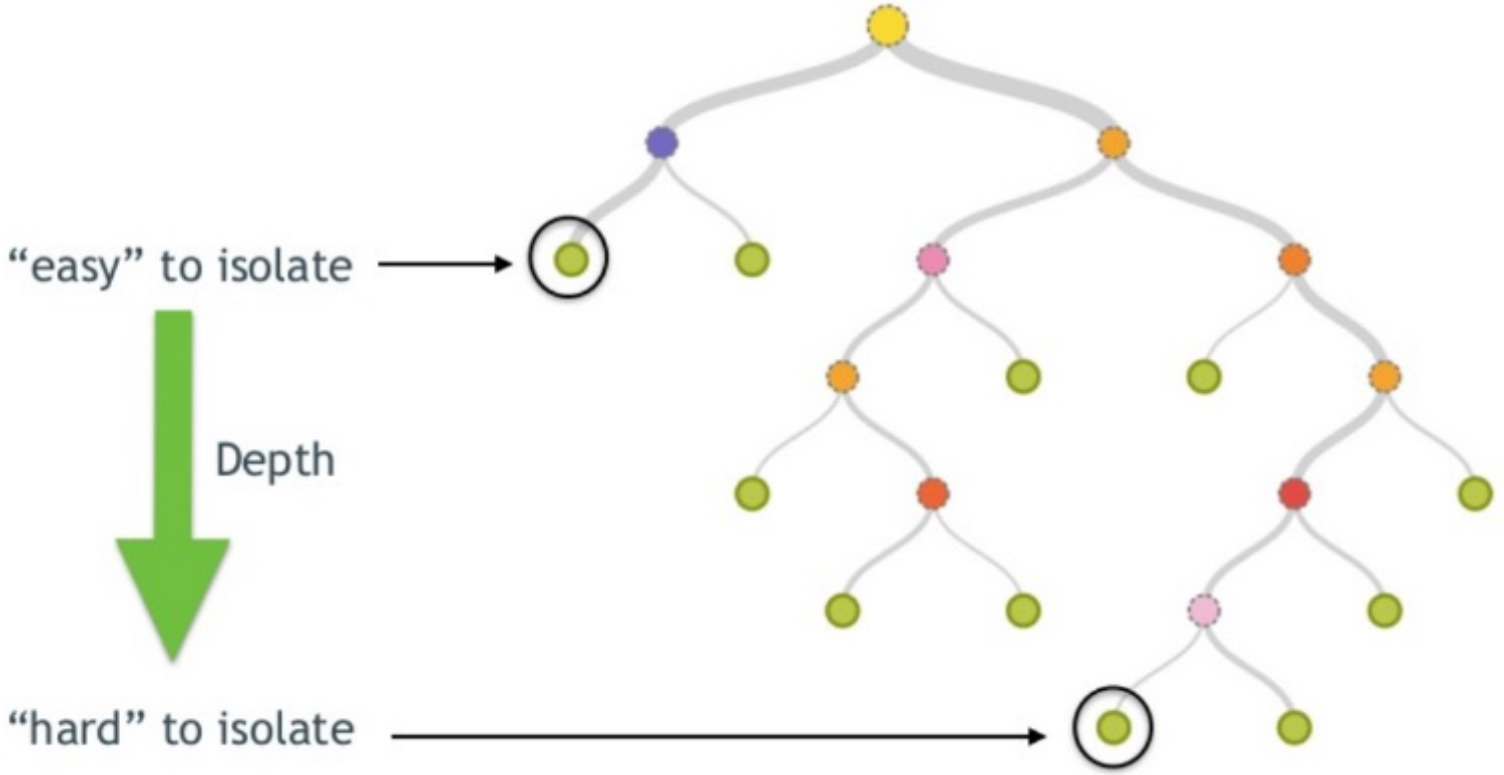
- Build a decision tree from a random data subsample




- Select the split feature *A*, randomly and uniformly
- Select the split value $V_A$, uniformly as the min(A)+(max(A)–min(A))*rand(1)
- Grow random tree until each data point in its own leaf or tree reaches max height

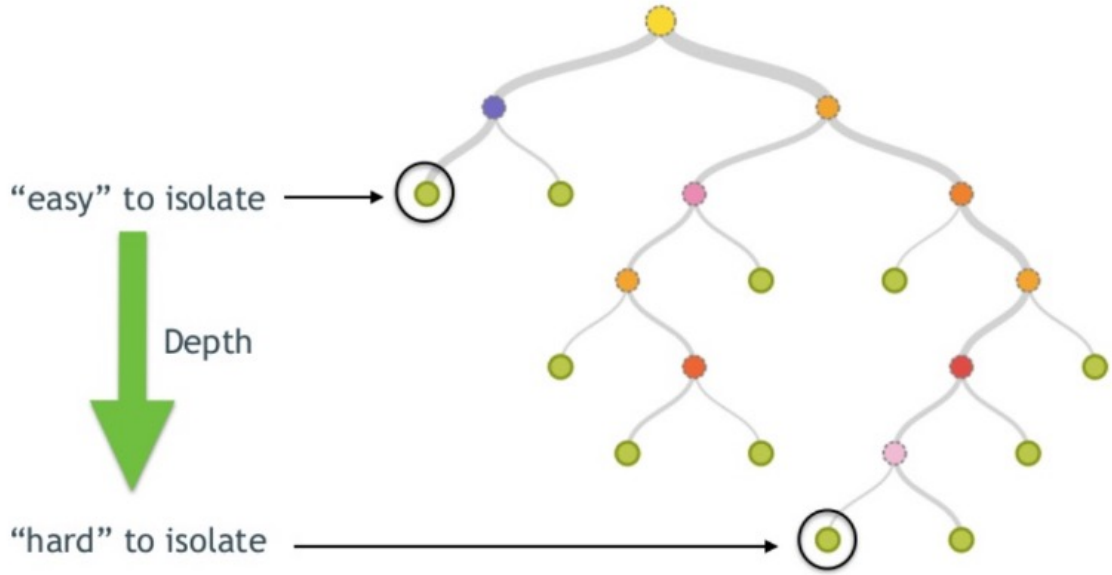
# Isolation Forest

- To score a data point, find the height of the leaf node
  - The smaller the height the more anomalous is the data



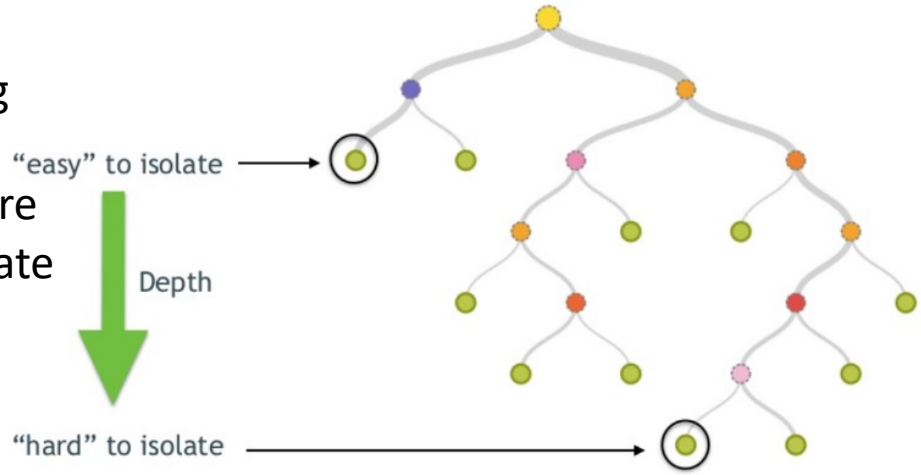
# Isolation forest

- Can we use this score for outlier detection?
  - Score=height



# Ensembles

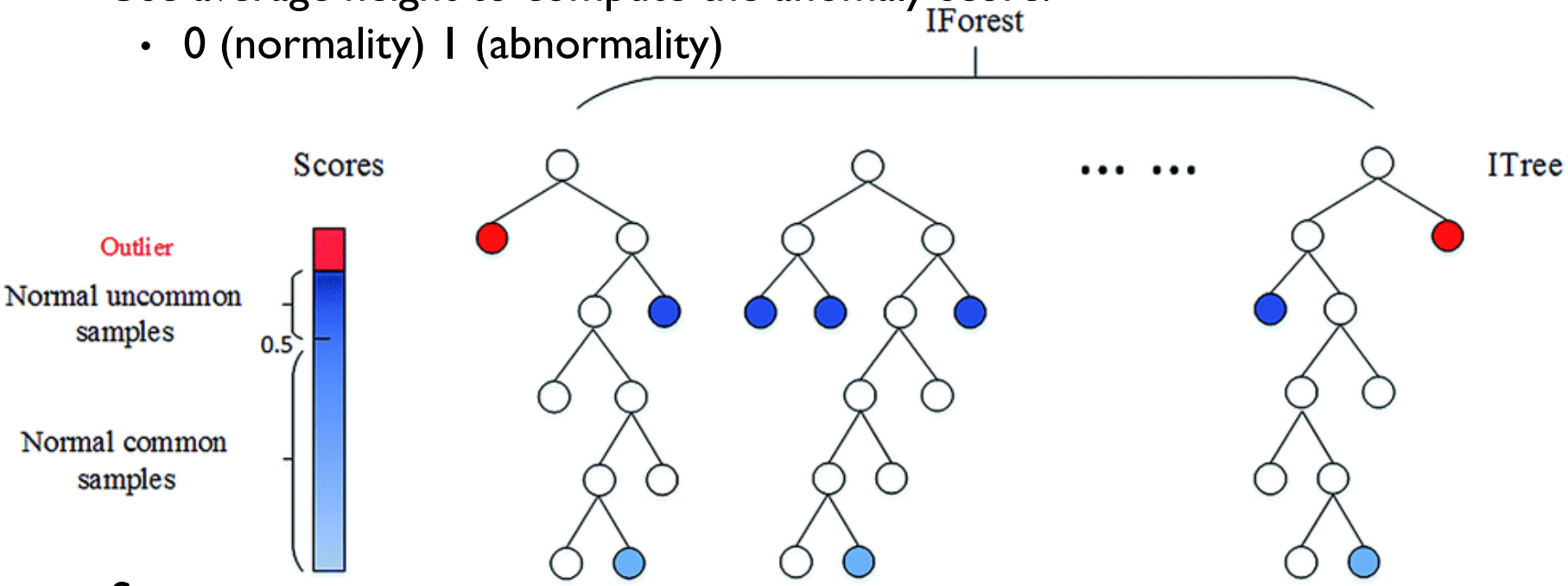
- Observation
  - Any single tree can be arbitrarily wrong
  - Random process of creating trees
  - But, if you try often enough, chances are that the outliers will be easier to separate more often




- Solution
  - Build an emsemble
  - Approach used in supervised and also (sometimes) in unsupervised learning
  - Idea: combine several models
  - Typical approach: average
  - Known to generate robust models even if composed of weakly performing models (as long as they are better than random)
- Build an ensemble of decision trees
  - from randomly selected subsamples of size n
  - To score a data point, find the height of the leaf node (higher=anomalous)

# Isolation Forest Scores

- Use average height to compute the anomaly score:
  - 0 (normality) 1 (abnormality)



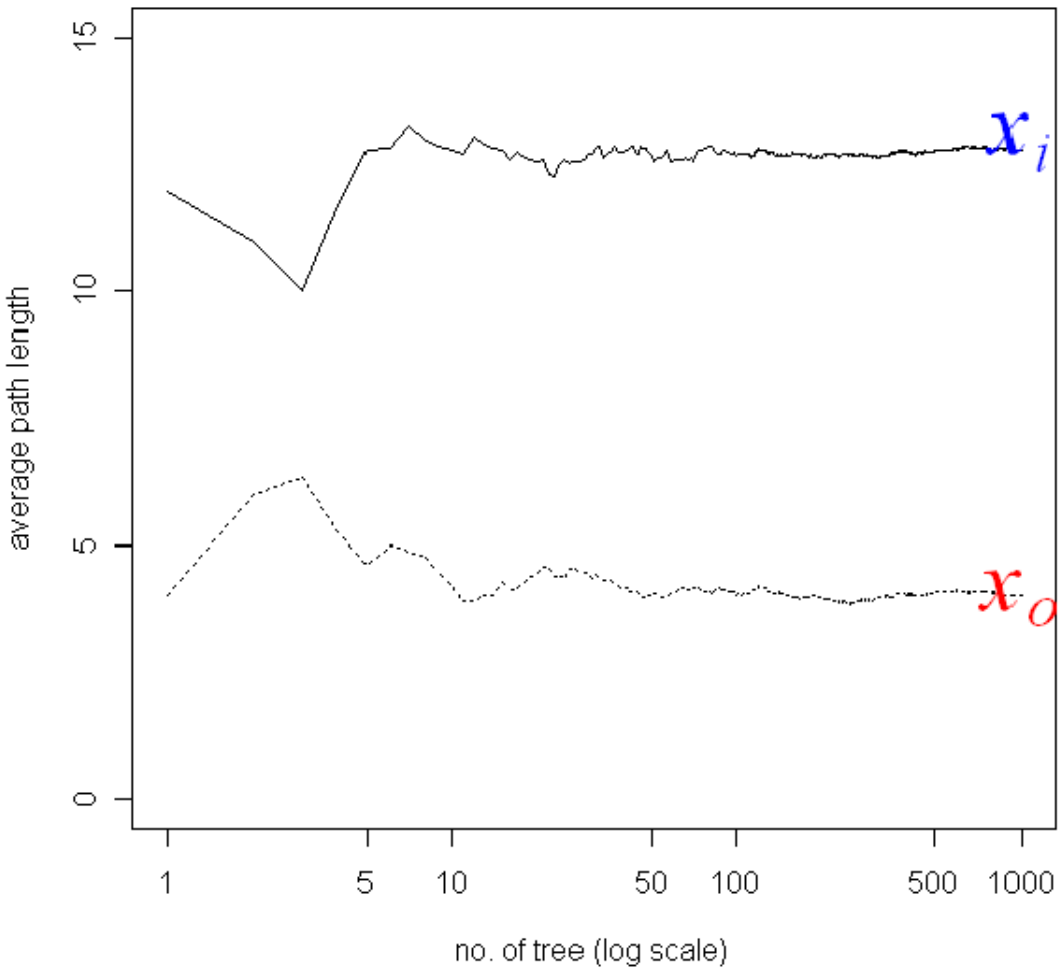

- Score
  - Ensemble average path length to a data point
  - Normalized by the expected path length of balanced binary search tree
  - H(x) path length of sample x

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}$$



https://towardsdatascience.com/a-brief-overview-of-outlier-detection-techniques-1e0b2c19e561

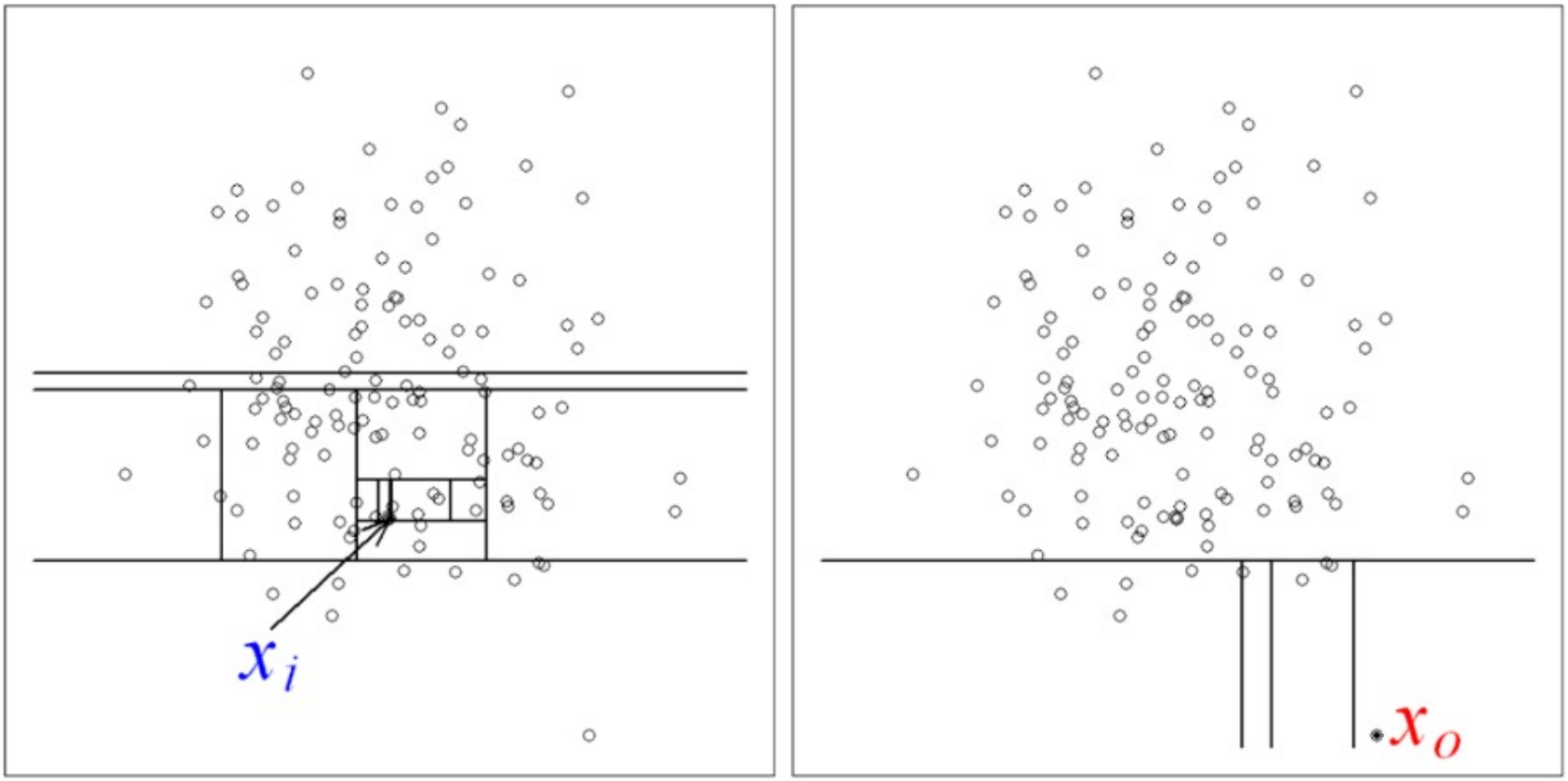
# Isolation Forest




Repeat

1. Randomly sample a subset of the data
2. Build a tree from random sample
   - Each tree is generated by randomly choosing a splitting attribute and the split point
   - Tree is grown either until maxdepth is reached, only 1 point remains, or all attributes have the same values

# Isolation Forest Scores




(a) Isolating $x_i$ (b) Isolating $x_o$

12 partitions (not an anomaly) 4 partitions (anomaly)

- https://www.depends-on-the-definition.com/detecting-network-attacks-with-isolation-forests/

# iTree Algorithm

Sample S randomly drawn from $X \leq R^d$

**Algorithm 1:** Function iTree($S, l, l_{max}$)

**Input** : $S \subset X$, $l$ the current depth level, $l_{max}$ the maximal depth limit
**Output**: an iTree

```
1  if l ≥ l_max or |S| ≤ 1 then
2      return exNode(S)
3  end
4  else
5      randomly select a dimension q ∈ {1, ··· , n}
6      randomly select a split value p between max and min values along dimension q in S
7      S_l ← filter(S, q < p)
8      S_r ← filter(S, q ≥ p)
9      return inNode(Left ← iTree(S_l, l + 1, l_max),
10                   Right ← iTree(S_r, l + 1, l_max),
11                   splitDim ← q,
12                   splitVal ← p)
13 end
```

# iForest: Pros and Cons

- Pros
  - Very easy to construct (no distance/density function needed) avoiding hard decisions whether a data point is an anomaly or not
    - assigns an anomalous score to each of the testing data point
  - Achieve a sublinear time-complexity and a small memory-footprint
    - By exploiting subsampling
    - By eliminating major computational cost of distance calculation in all the distance-based and density-based AD methods
  - Can provide anomaly explanations
- Cons
  - Hyper-parameter tuning (e.g. number/height of trees, sample size)
    - Large datasets will need more isolation trees (how many?)
  - Requires a *high percentage of relevant features to identify anomalies*
    - In presence of features that do not provide information over the anomaly, iForest increases height randomly by ignoring this fact

# Bayes Classifier

- *Decision rule:* $c_{max} = \arg\max_{c_j \in C} \{P(c_j \mid o)\}$
- Since, typically, the values of P($c_j$ | *o*) are not known, the rule is transformed by using *Bayes´ theorem:*

Bayes´ rule: $p(c_j \mid o) \cdot p(o) = p(o \mid c_j) \cdot p(c_j)$

- Final decision rule for the *optimal Bayes classifier* (called *Maximum Likelihood* classifier)

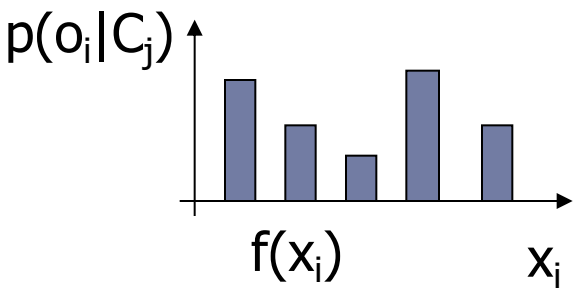
$$\arg\max_{c_j \in C} \{p(c_j \mid o)\} = \arg\max_{c_j \in C} \left\{ \frac{p(o \mid c_j) \cdot p(c_j)}{p(o)} \right\} = \arg\max_{c_j \in C} \{p(o \mid c_j) \cdot p(c_j)\}$$

$$c_{max} = \arg\max_{c_j \in C} \{P(o \mid c_j) \cdot P(c_j)\}$$

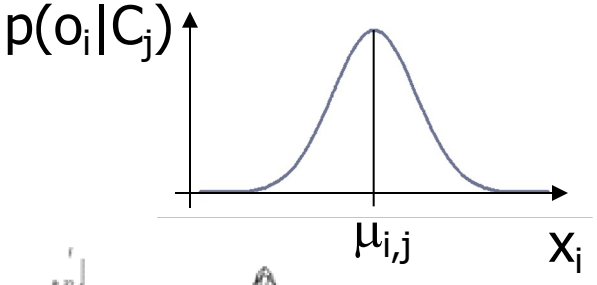
# Bayesian Classifier (2)

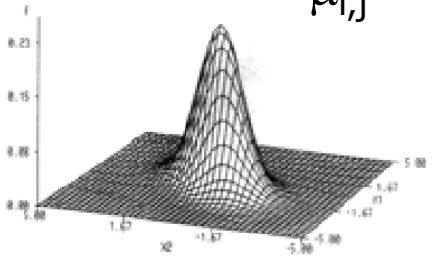
- Estimate the values of $p(c_j)$ by using the observed frequency of the individual class labels $c_j$

- If i-th attribute is categorical:
  $p(o_i|C)$ is estimated as the relative frequency of samples having value $x_i$ as i-th attribute in class C




- If i-th attribute is continuous:
  $p(o_i|C)$ is estimated through e.g.:
  - Gaussian density function (multivariate)
    determine $(\mu_{i,j}, \sigma_{i,j})$

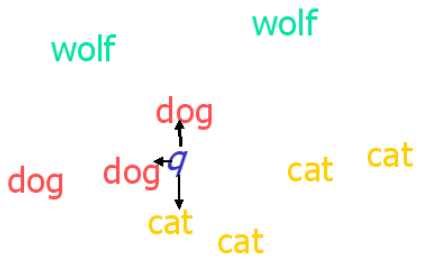


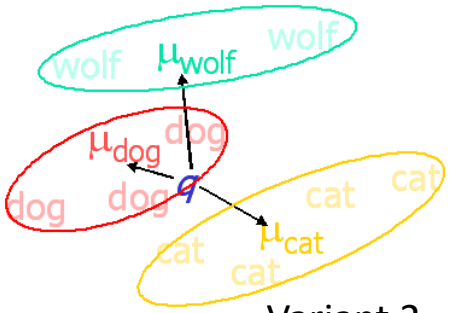
➔ $$p(o_i \mid C_j) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{o_i - \mu_{i,j}}{\sigma_{i,j}}\right)^2}$$

# Nearest Neighbor Classifiers

- Instance-based learning
- Fundamental procedure
  - Use attribute vectors $o = (o_1, \ldots, o_d)$ as training objects
    - Do nothing for training! (lazy)
  - Variant 1:
    - Assign query object to the class $c_j$ of the closest training object
  - Variant 2:
    - Determine mean vector $\mu_i$ for each class $c_j$ (in training phase) (not quite as lazy)
    - Assign query object to the class $c_j$ of the nearest mean vector $\mu_i$
  - Generalizations:
    - Consider $k > 1$ neighbors for the class assignment decision (Var. 1)
    - Use weights for the classes of the $k$ nearest neighbors
    - Use more than one representative per class (Var. 2)



Variant 1

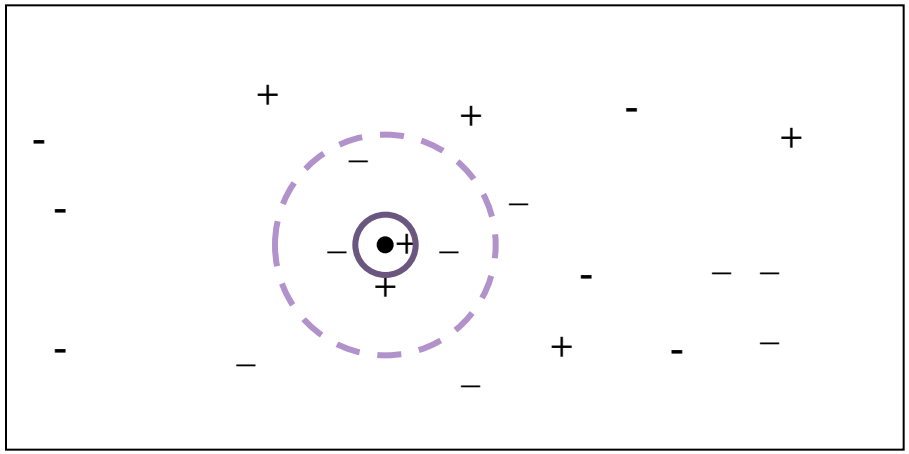


Variant 2

# Nearest Neighbor Classifiers: Notions

- Distance function
  - Defines the (dis-)similarity for pairs of objects
- Number *k* of neighbors to be considered
- Decision set
  - Set of *k* nearest neighboring objects to be used in the decision rule
- Decision rule
  - Given the class labels of the objects from the decision set, how to determine the class label to be assigned to the query object?

# Nearest Neighbor Classifiers: Example



Classes + and –

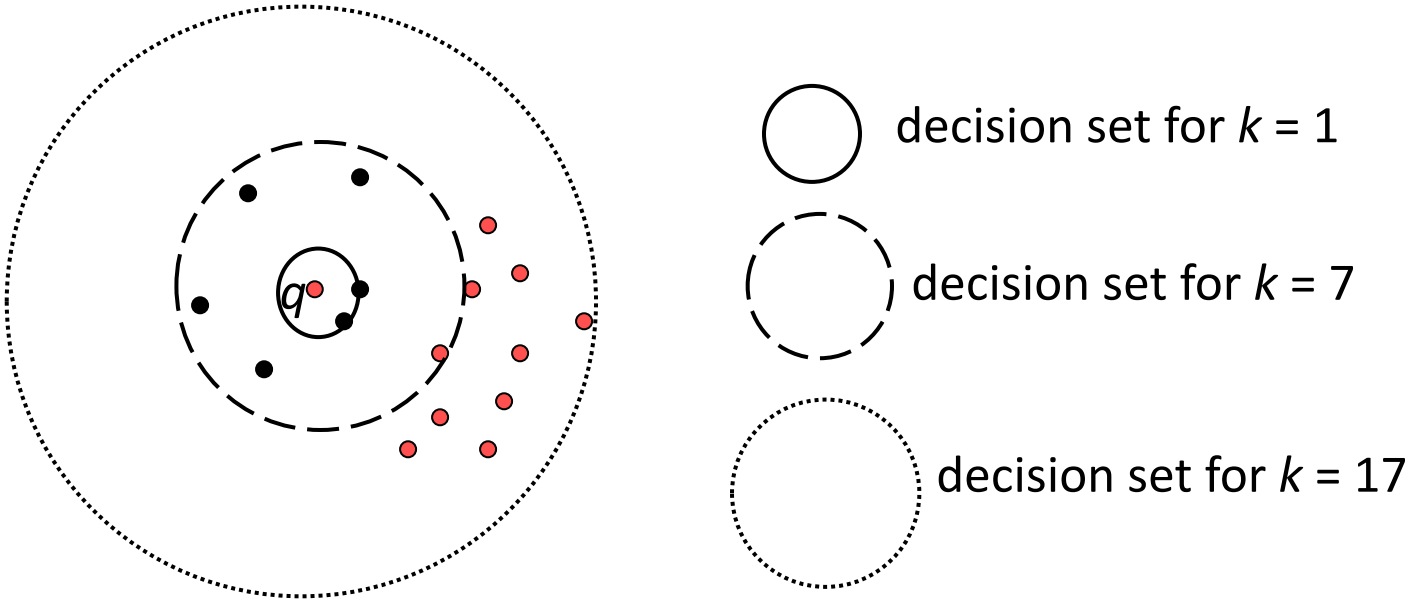
decision set for $k = 1$

decision set for $k = 5$

- Using unit weights (i.e., no weights) for the decision set
  - Simply called *"majority criterion"*
  - for $k = 5$, rule yields class „–“
- Using the reciprocal square of the distances as weights
  - for $k = 5$, rule yields class „+“
- Using a-priori probability (=frequency) of classes as weights
  - for $k = 5$, rule yields class „+“

# Nearest Neighbor Classifiers: Parameters

- Problem of choosing an appropriate value for parameter *k*
  - *k* too small: high sensitivity against outliers
  - *k* too large: decision set contains many objects from other classes
  - Empirically, 1 << *k* < 10 yields a high classification accuracy in many cases



# NN Classification: Discussion

+ applicability: training data required only
+ high classification accuracy in many applications
+ easy incremental adaptation to new training objects
+ useful also for prediction
+ robust to noisy data by averaging *k*-nearest neighbors

– naïve implementation is inefficient
  - requires k-nearest neighbor query processing
  - support by database techniques may help to reduce from O(n) to O(log n) for n training objects
– does not produce explicit knowledge about classes
  - But provides some explanation information
– Curse of dimensionality: distance between neighbors could be dominated by irrelevant attributes
  - To overcome it, stretch axes or eliminate least relevant attributes

# Remarks on Lazy vs. Eager Learning

- Instance-based learning: lazy evaluation
- Decision-tree and Bayesian classification: eager evaluation
- Key differences
  - Lazy method may consider query instance *xq* when deciding how to generalize beyond the training data *D*
  - Eager method cannot since they have already chosen global approximation when seeing the query
- Efficiency
  - Lazy - less time training but more time predicting
- Accuracy
  - Lazy method effectively uses a richer hypothesis space since it uses many local linear functions to form its implicit global approximation to the target function
  - Eager: must commit to a single hypothesis that covers the entire instance space

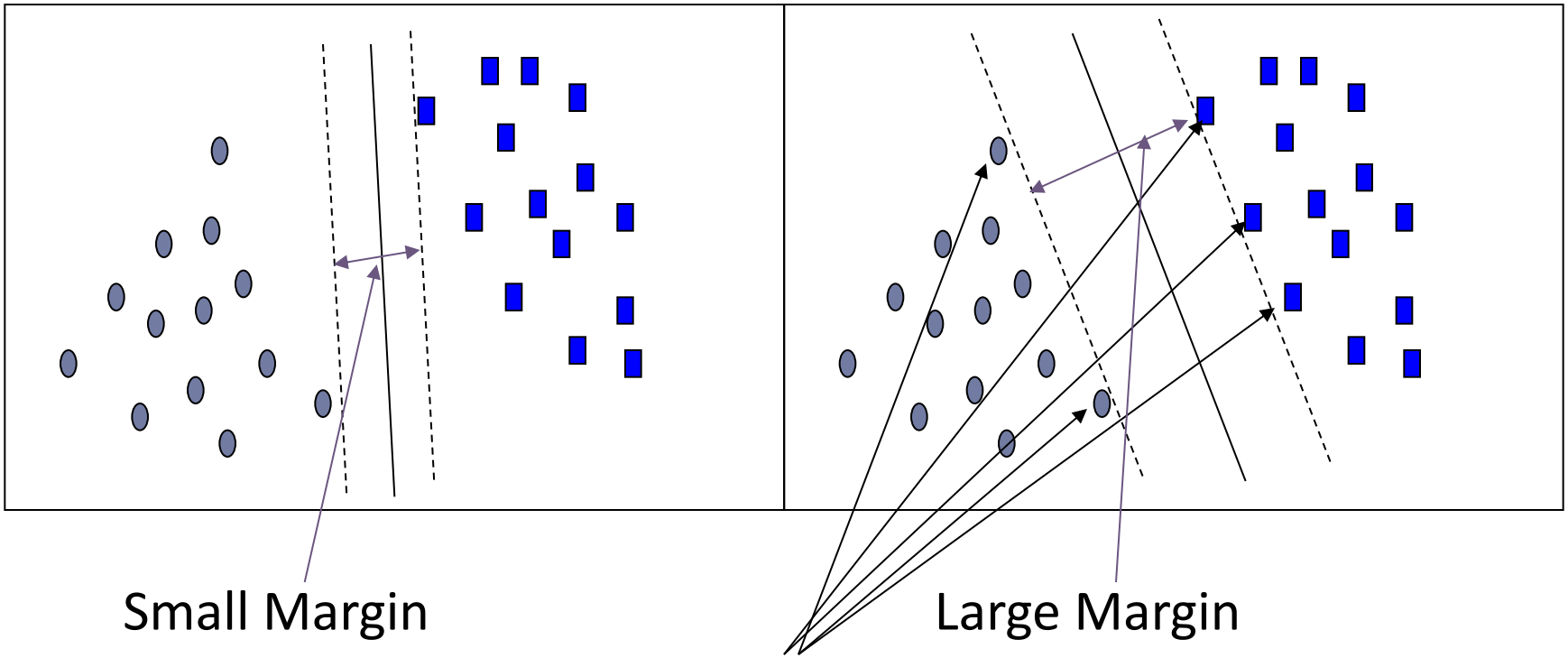
# SVM—Support Vector Machines

- A classification method for both linear and nonlinear data
- It uses a nonlinear mapping to transform the original training data into a higher dimension
- With the new dimension, it searches for the linear optimal separating hyperplane (i.e., “decision boundary”)
- With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane
- SVM finds this hyperplane using support vectors (“essential” training tuples) and margins (defined by the support vectors)

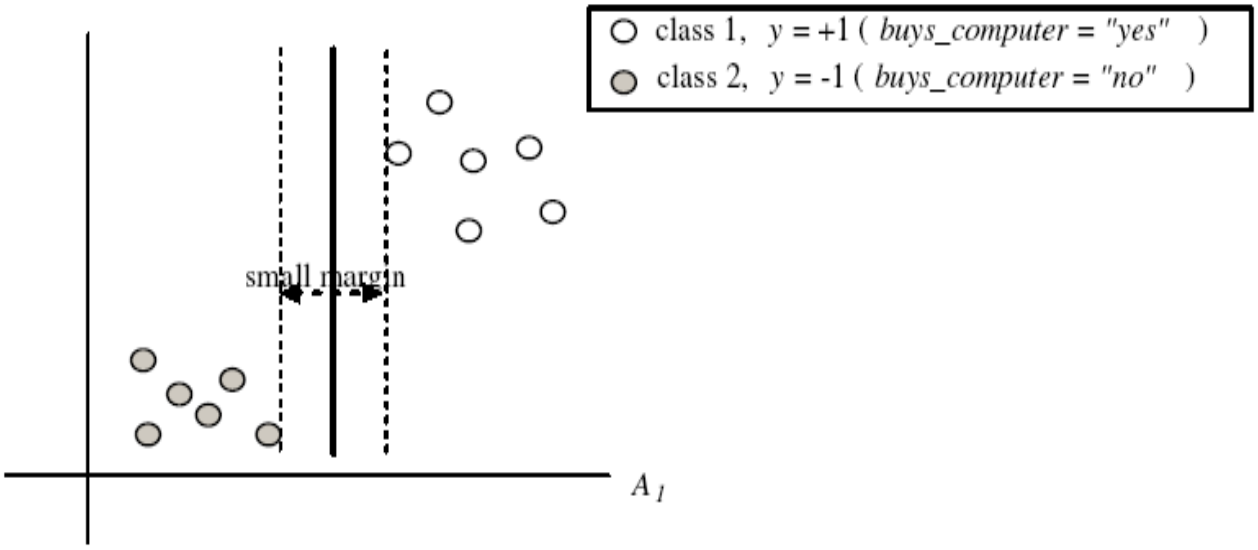
# SVM—History and Applications

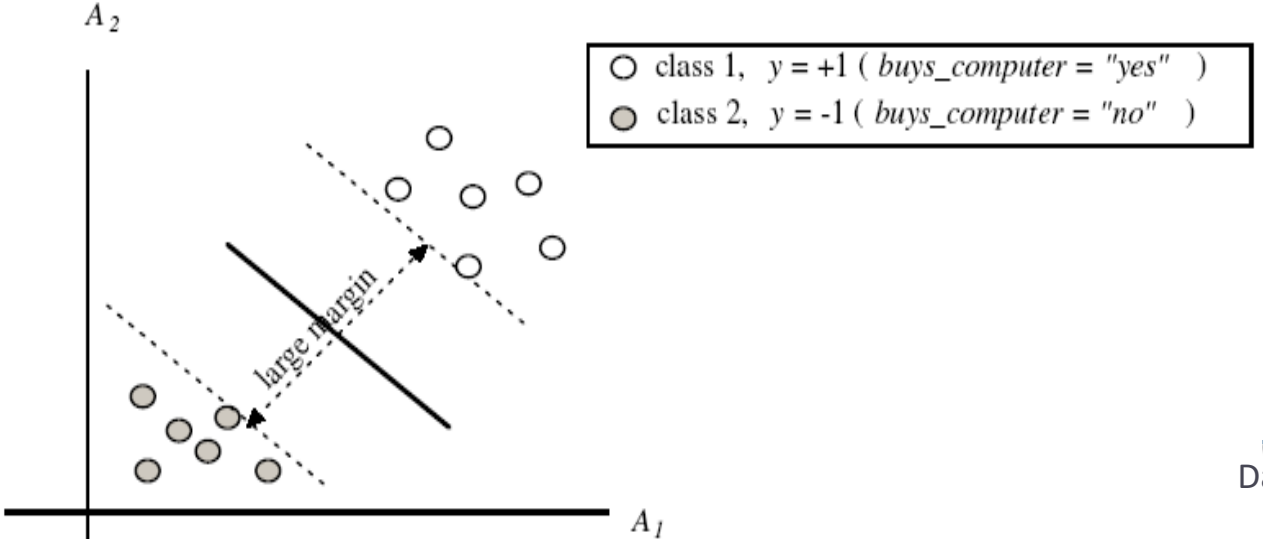
- Vapnik and colleagues (1992)—groundwork from Vapnik & Chervonenkis' statistical learning theory in 1960s
- Features: training can be slow but accuracy is high owing to their ability to model complex nonlinear decision boundaries (margin maximization)
- Used both for classification and prediction
- Applications:
  - handwritten digit recognition, object recognition, speaker identification, benchmarking time-series prediction tests

# SVM—General Philosophy

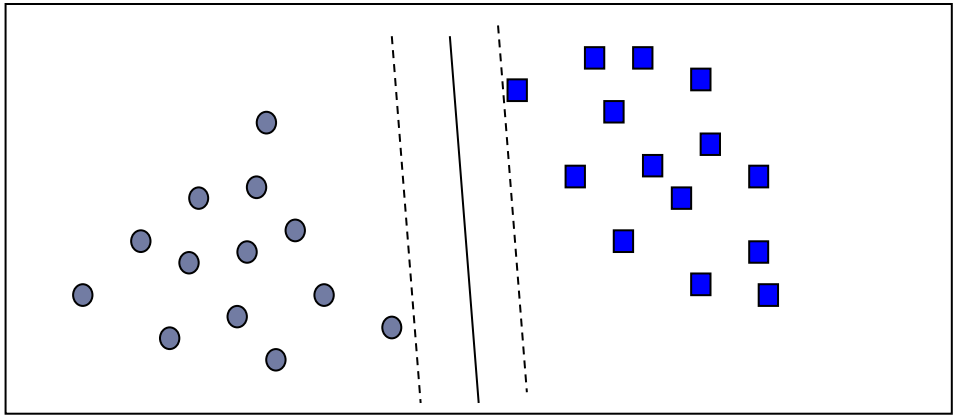


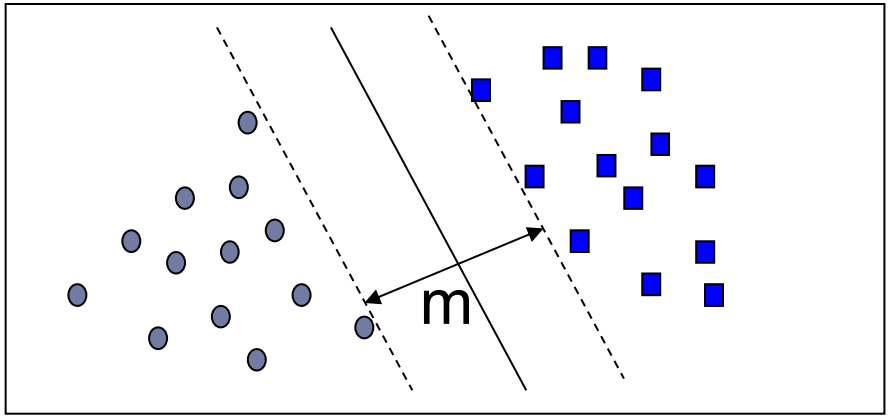
# SVM—Margins and Support Vectors



# SVM—When Data Is Linearly Separable






- Let data D be (X1, y1), …, (X|D|, y|D|), where Xi is the set of training tuples associated with the class labels yi
- There are infinite lines (hyperplanes) separating the two classes, but we want to find the best one (the one that minimizes classification error on unseen data)
- SVM searches for the hyperplane with the largest margin, i.e., maximum marginal hyperplane (MMH)

# SVM—Linearly Separable

- A separating hyperplane can be written as

  $\mathbf{W} \bullet \mathbf{X} + b = 0$

  where $\mathbf{W}=\{w_1, w_2, \ldots, w_n\}$ is a weight vector and b a scalar (bias)

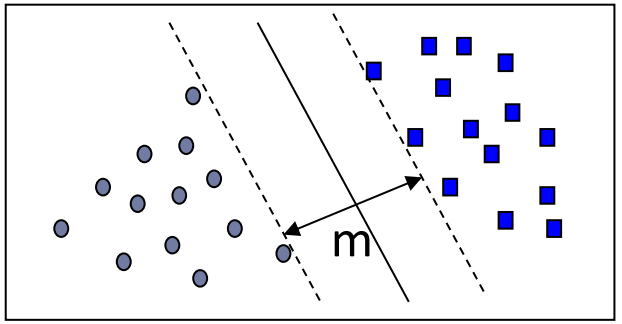
- For 2-D it can be written as

  $w_0 + w_1 x_1 + w_2 x_2 = 0$

- The hyperplane defining the sides of the margin:

  $H_1: w_0 + w_1 x_1 + w_2 x_2 \geq 1$ for $y_i = +1$, and

  $H_2: w_0 + w_1 x_1 + w_2 x_2 \leq -1$ for $y_i = -1$



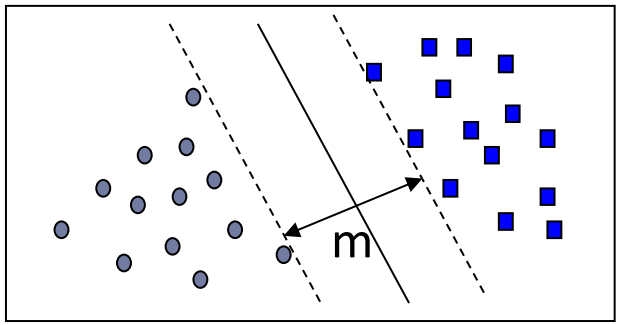

- Any training tuples that fall on hyperplanes $H_1$ or $H_2$ (i.e., the sides defining the margin) are **support vectors**
- This becomes a **constrained (convex) quadratic optimization** problem: Quadratic objective function and linear constraints → *Quadratic Programming (QP)* → Lagrangian multipliers

# Why Is SVM Effective on High Dimensional Data?

- The complexity of trained classifier is characterized by the # of support vectors rather than the dimensionality of the data
- The support vectors are the essential or critical training examples —they lie closest to the decision boundary (MMH)
- If all other training examples are removed and the training is repeated, the same separating hyperplane would be found
- The number of support vectors found can be used to compute an (upper) bound on the expected error rate of the SVM classifier, which is independent of the data dimensionality
- Thus, an SVM with a small number of support vectors can have good generalization, even when the dimensionality of the data is high

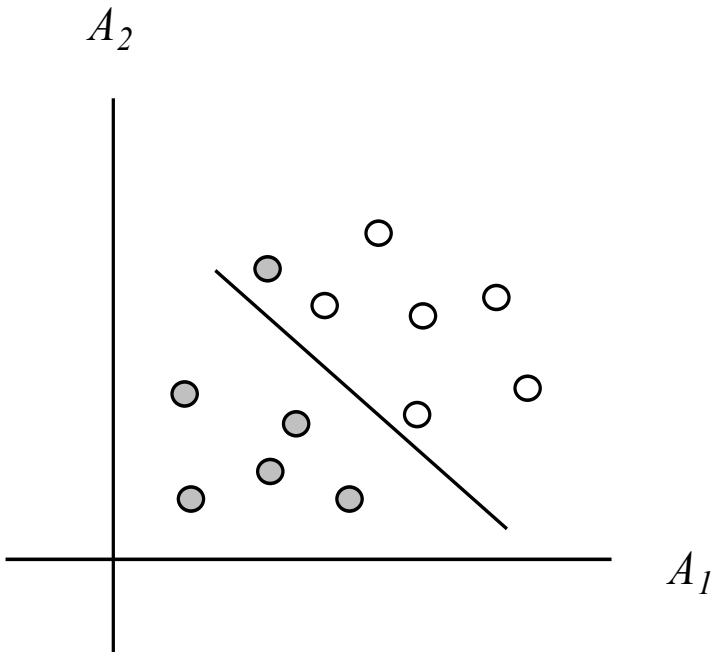
# But for real data…

- Any issues with this setup?



# SVM—Linearly Inseparable

- Transform the original input data into a higher dimensional space




- Search for a linear separating hyperplane in the new space

# SVM—Kernel functions

- Instead of computing the dot product on the transformed data tuples, it is mathematically equivalent to instead applying a kernel function $K(\mathbf{X}_i, \mathbf{X}_j)$ to the original data, i.e., $K(\mathbf{X}_i, \mathbf{X}_j) = \Phi(\mathbf{X}_i)\ \Phi(\mathbf{X}_j)$
- Typical Kernel Functions

$$\text{Polynomial kernel of degree } h: \quad K(X_i, X_j) = \quad (X_i \cdot X_j + 1)^h$$

$$\text{Gaussian radial basis function kernel}: \quad K(X_i, X_j) = \quad e^{-\|X_i - X_j\|^2/2\sigma^2}$$

$$\text{Sigmoid kernel}: \quad K(X_i, X_j) = \quad \tanh(\kappa X_i \cdot X_j - \delta)$$

- SVM can also be used for classifying multiple (> 2) classes and for regression analysis (with additional user parameters)

# What Is Prediction?

- Prediction is *similar* to classification
  - First, construct a model
  - Second, use model to predict unknown value
    - Major method for prediction is regression
      - Linear regression
      - Logistic regression
- Prediction is *different* from classification
  - Classification refers to predict categorical class label
  - Prediction models continuous-valued functions

# Regression

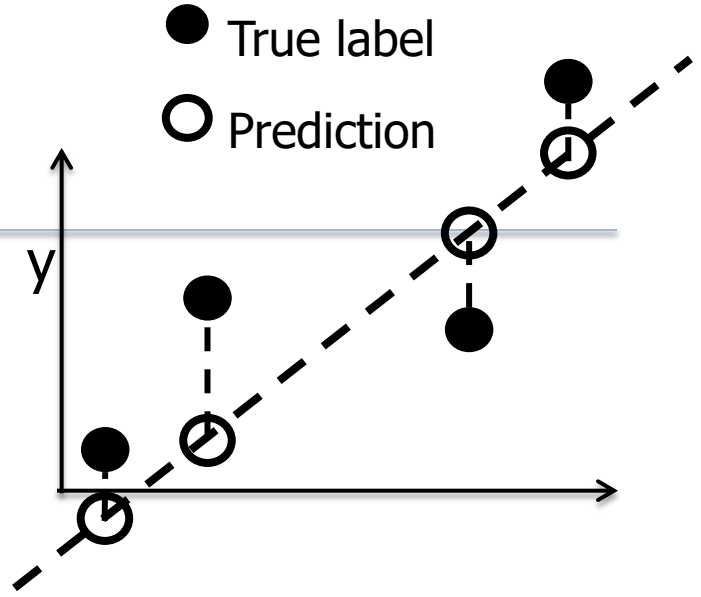
- Regression
  - Predicting value instead of class/category
  - Labels Y=R (real values).

- Least Squares Loss Function
  - In regression, often use least squares loss
  - $L(y',y) = (y'-y)^2$

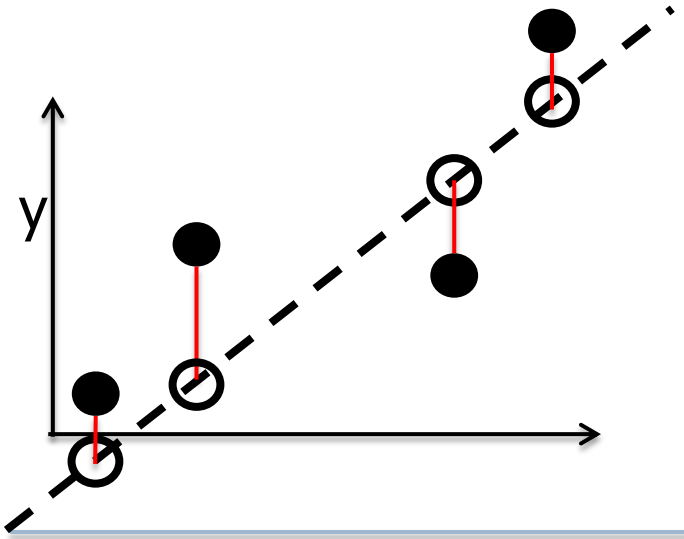
- $L(5.300.000, 5.150.000) = (150.000)^2$

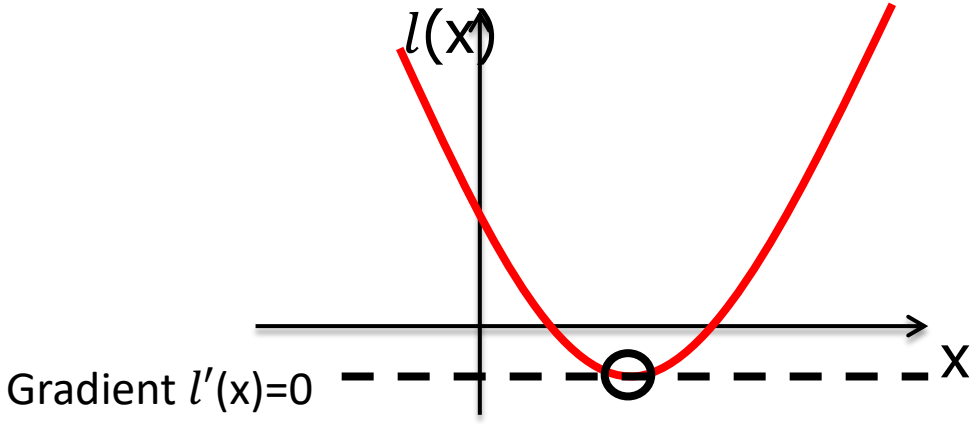
- Predict house prices

# Linear regression



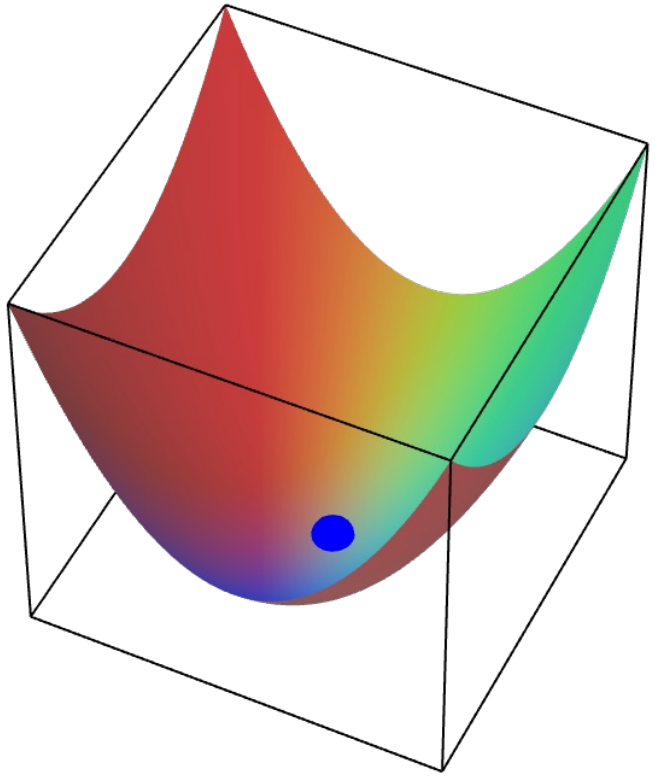

- Linear Model:
  - Predict: $w(x) = w_1x_1 + \ldots + w_dx_d + b = x^Tw+b$
  - Features: $x=(x_1,\ldots,x_d)$
  - Parameters to be learnt $(w_1, w_2,\ldots, w_d, b)$
  - Minimize: $L(w) = \sum_{i=1}^{n}\left(x_i^T w - y_i\right)^2$
  - To minimize, make use of gradient if differentiable



# Local Minima

- Functions of multiple variables:
  - If $l$ is a function of variables $x_1,\ldots x_d$, i.e. from $R^d$ to R.
  - Assume $l$ differentiable.



- Gradient (vector):
  - $\nabla_x\big(l(x)\big) = \begin{pmatrix} \frac{\partial}{\partial x_1} l(x) \\ \vdots \\ \frac{\partial}{\partial x_d} l(x) \end{pmatrix}$

- Local minima must have:
  - $\nabla_x\big(l(x)\big) = 0$

# Linear Models

- Logistic Regression:
  - Some applications not properly captured

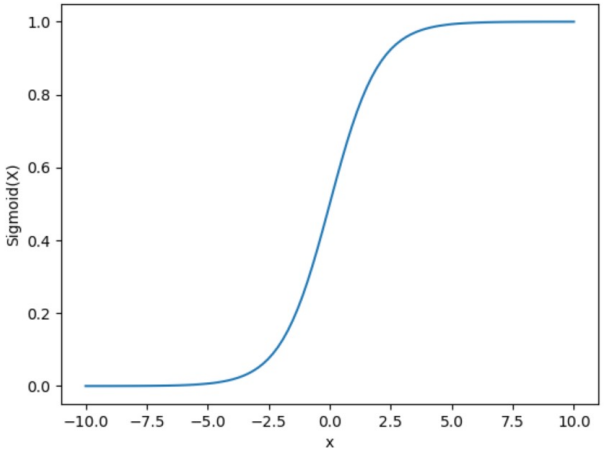
  - Probability of heart attack given medical conditions
  - Probability of failing to pay rates on mortgage
  - Probability of committing a crime if not imprisoned while awaiting trial

- Setup:
  - Two clases {-1,1}
  - Noisy target (not deterministic) P(y | x)
  - Goal: Learn Pr[y=1 | x], i.e. learn P(y | x)
  - Y=[0,1]

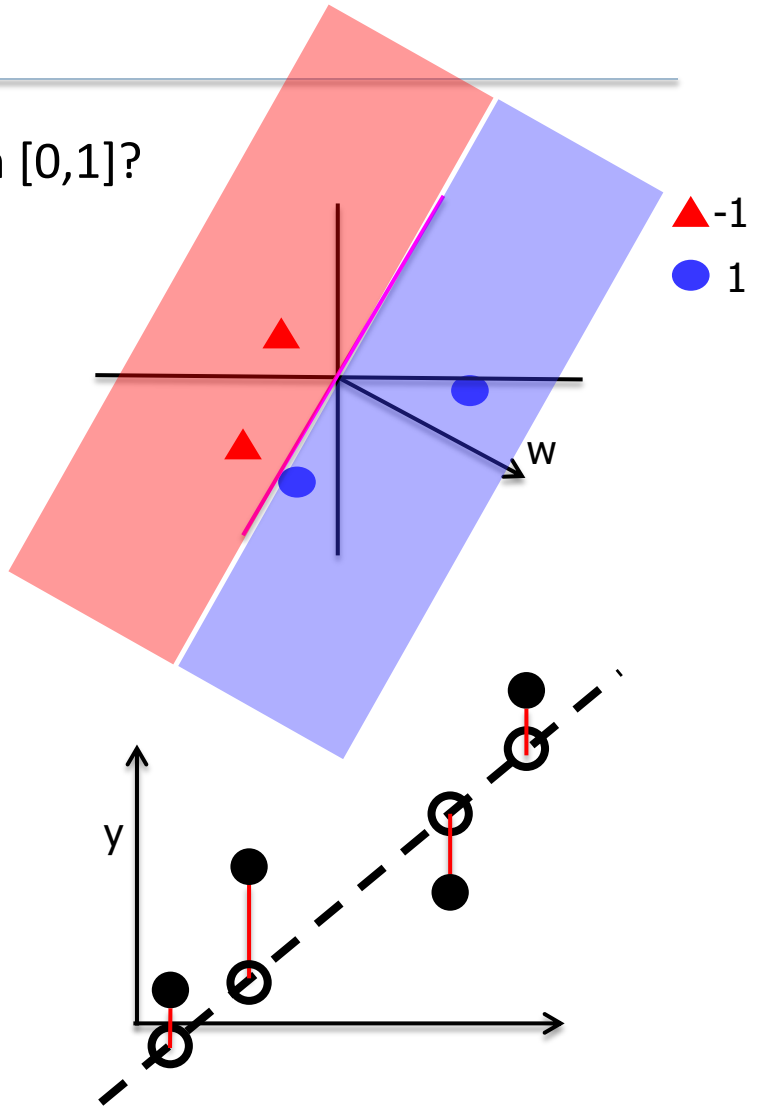
# Logistic regression

How do we make a linear model with outputs in [0,1]?

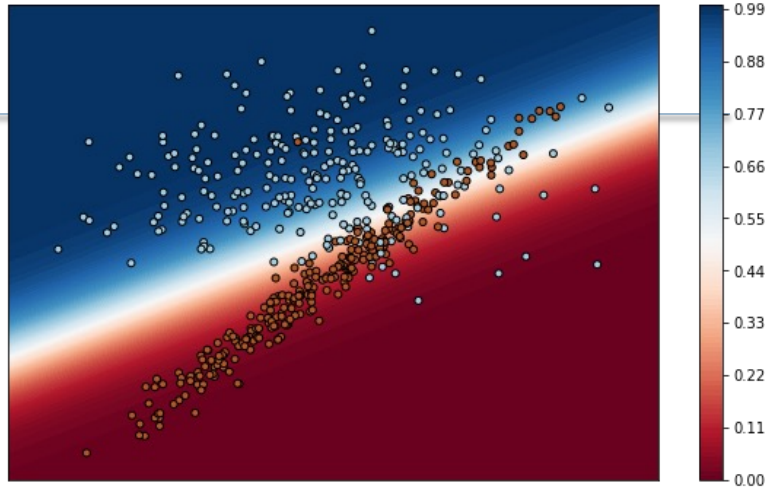
- Use sigmoid / logistic function $\theta(z) = \frac{1}{1+e^{-z}}$
  - Shape like an ”s”




  - Predict(x)= $\theta(w^T x) = \frac{1}{1+e^{-w^T x}}$
  - Y=[0,1]



# Learning




- Logistic Regression:
  - Predict(x)= $\theta(w^T x)= \frac{1}{1+e^{-w^T x}}$
- What loss should we minimize?
  - Classification:
    - 0-1 Loss:
      - $E_{in}(w) = \frac{1}{n}\sum_{i=1}^{n} 1_{sign(x_i^T w)\neq y_i}$
  - Regression:
    - Least squares:
      - $E_{in}(w) = \frac{1}{n}\sum_{i=1}^{n}\left(x_i^T w - y_i\right)^2$
- Idea:
  - For any hypothesis w, can estimate ”likelihood” of training labels, denoted Pr[S | w,X] where S=($x_1$,$y_1$),…,($x_n$,$y_n$).
  - Maximum likelihood approach to obtain best ”fit” (details omitted here)

# Summary

- Supervised learning
  - Build a model of the data for characterization
  - Or for making classification / prediction on new data
- Learning setup
  - Build model on training data
  - Evaluate on testing / validation data (more next time)
- Decision trees
- Isolation forests for outlier detection
  - Ensemble approach over several learners
- Naïve Bayes (probabilistic)
- Nearest neighbor (lazy)
- Support vector machines
- Regression: prediction

# References


- https://jakevdp.github.io/PythonDataScienceHandbook/ Data Science in Python
- https://wesmckinney.com/book/ Python for Data Science
- https://scikit-learn.org Scikit-learn
- https://pandas.pydata.org/ Pandas
- https://dataminingbook.info/book_html/ Mohammed J. Zaki, Wagner Meira, Jr., Data Mining and Machine Learning: Fundamental Concepts and Algorithms, 2nd Edition, Cambridge University Press, March 2020. ISBN: 978-1108473989.
- J. Han and M. Kamber. Data Mining: Concepts and Techniques. Morgan Kaufmann, 2000.
- M. H. Dunham. Data Mining: Introductory and Advanced Topics. Prentice Hall, 2003
- Kasper Green Larsen, Machine Learning course slides, Aarhus University