Introduction to Machine Learning (IML)

LECTURE #3: CLASSIFICATION - METHODS

236756 – 2023-2024 WINTER – TECHNION

LECTURER: YONATAN BELINKOV

Today

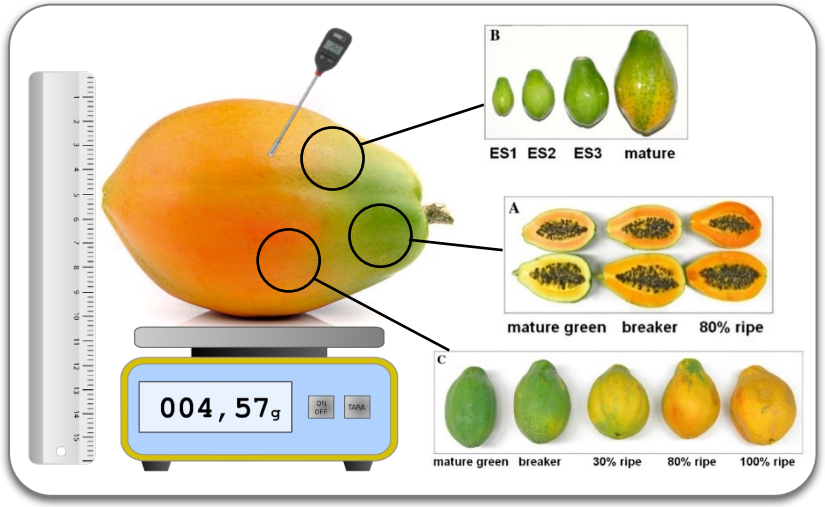
Recall from last time:

- Univariate (scalar) inputs, $x \in \mathbb{R}$
- Threshold models: $\mathrm{sign}(x > \theta)$
- Modeling, optimization, statistics
- Distribution-independent learning
- Restricted model classes

<u>Today</u>: Multivariate (vector) inputs, $x \in \mathbb{R}^d$

Multivariate inputs

- Features are *vectors* $x \in \mathbb{R}^d$

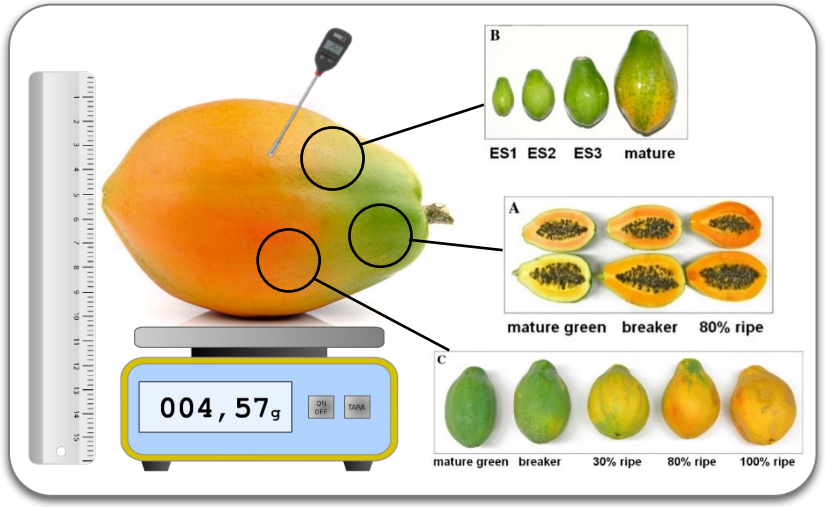


measure()

$$x = (height, weight, color) \in \mathbb{R}^3$$

Multivariate inputs

- Features are *vectors* $x \in \mathbb{R}^d$
- **Q**: How useful are vector features as descriptors of data?

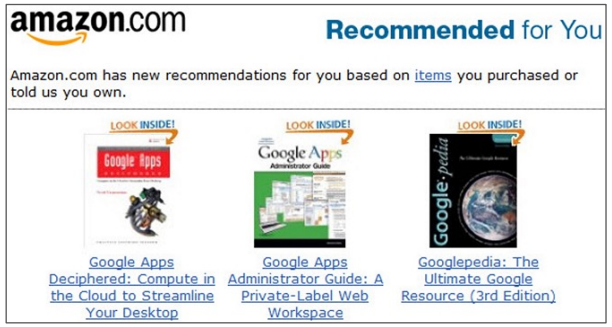


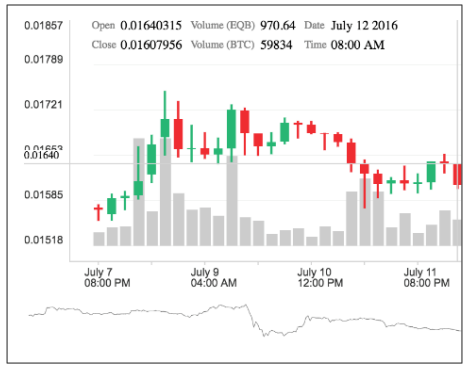
measure()

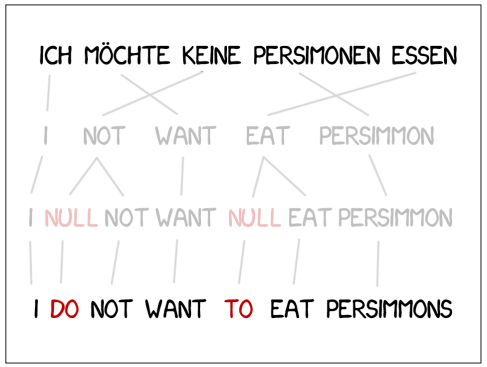
$$x = (height, weight, color) \in \mathbb{R}^3$$

Multivariate inputs

- Features are *vectors* $x \in \mathbb{R}^d$
- **Q**: How useful are vector features as descriptors of data?
- **A**: Very! Examples:



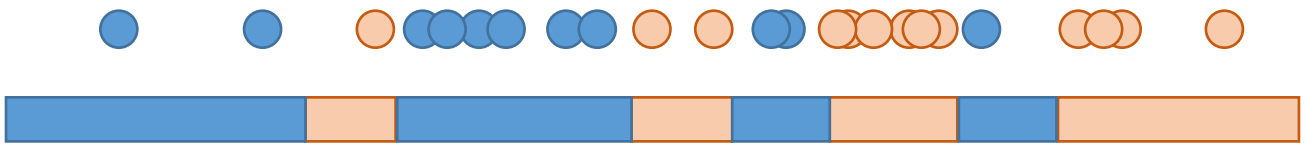




- Feature vectors are very, very useful.
- (So much that many modern methods include learning to transform objects into vectors)
- For clarity, let's think of vectors as describing **attributes** (think movies)

Multivariate inputs

- **Recall**: 1D intervals

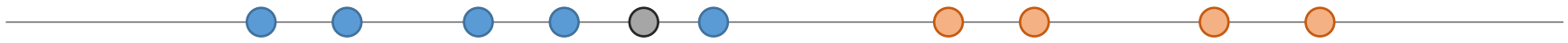


- For 1D, can consider directly parameterizing models with intervals
- But this approach will break for high-dimensional (vector) data.
- **Today – three alternative learning approaches:**
  1. Similarity-based (kNN)
  2. Rule-based (decision trees)
  3. Linear models (SVM)

Similarity-based learning

Similarity-based learning

- Let's start with 1D data.
- How would you classify this new point?



- *Why?* (we'll return to this later)
- Is there an underlying principle?
- Does it apply to higher-dimensional data?

Similarity-based learning

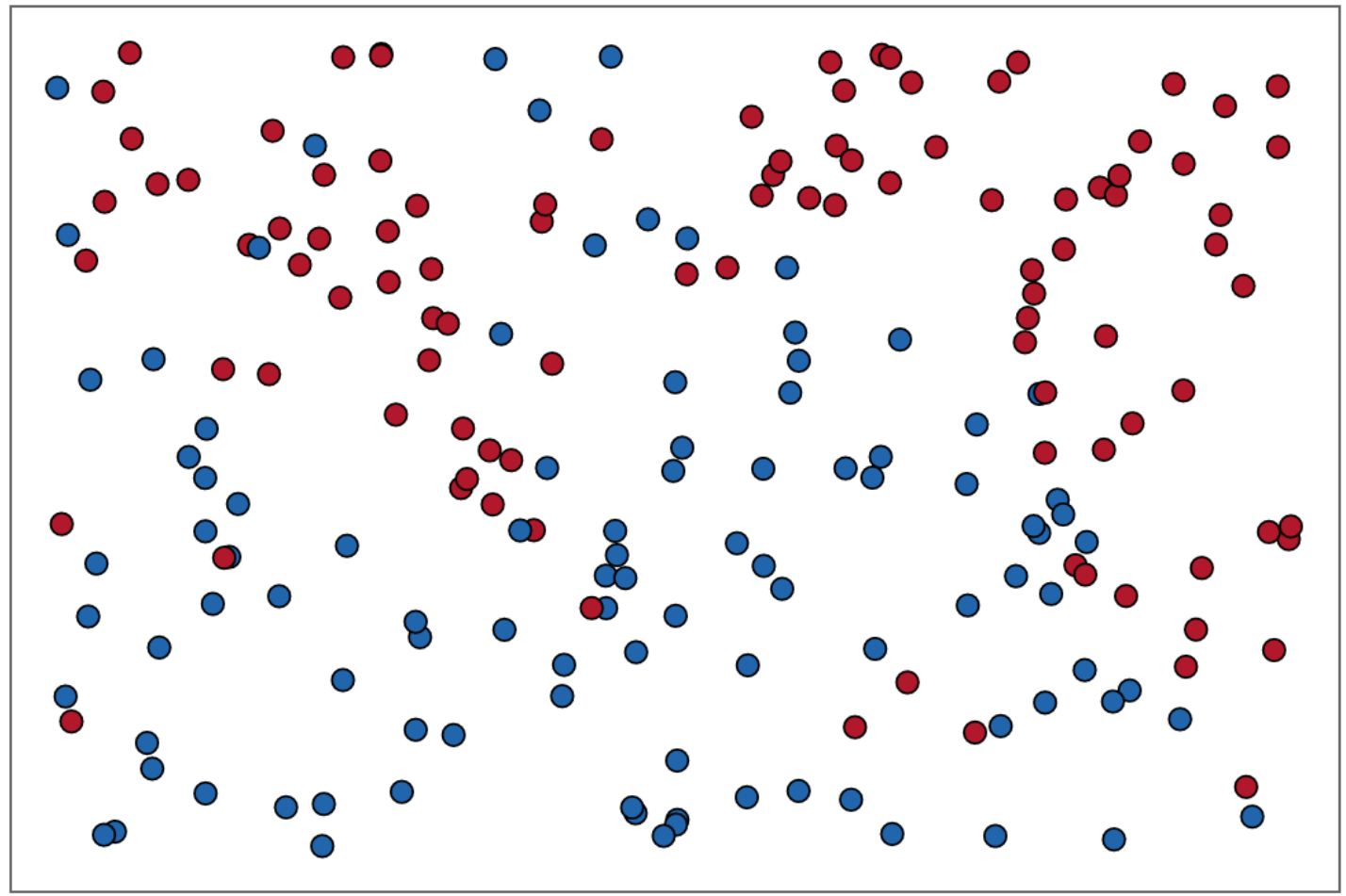
Quote:

"Show me who your friends are
and I will tell you who you are"

similarity-based classification:

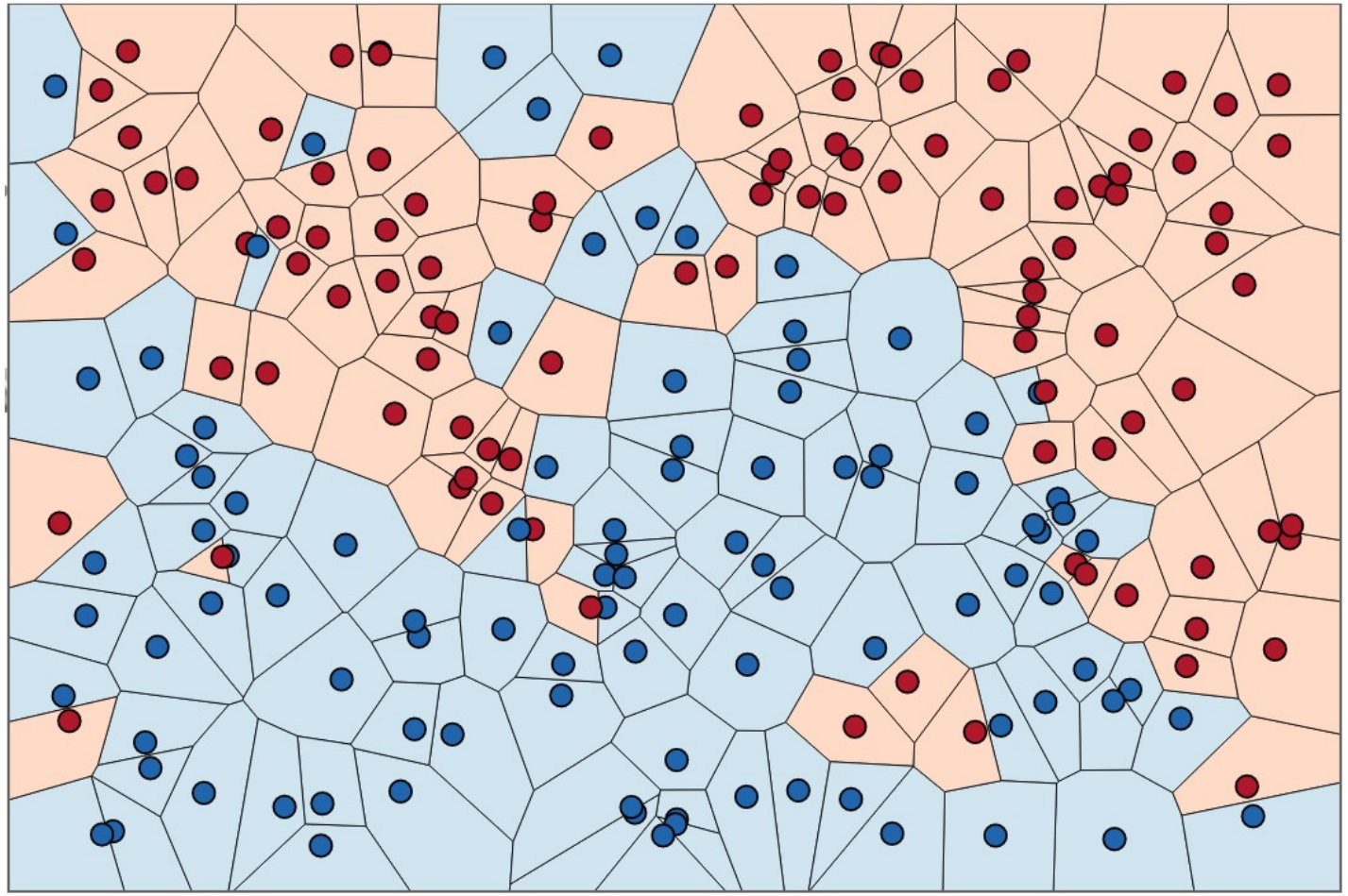
1. find closest examples in data
2. predict using their labels

- Sancho Panza



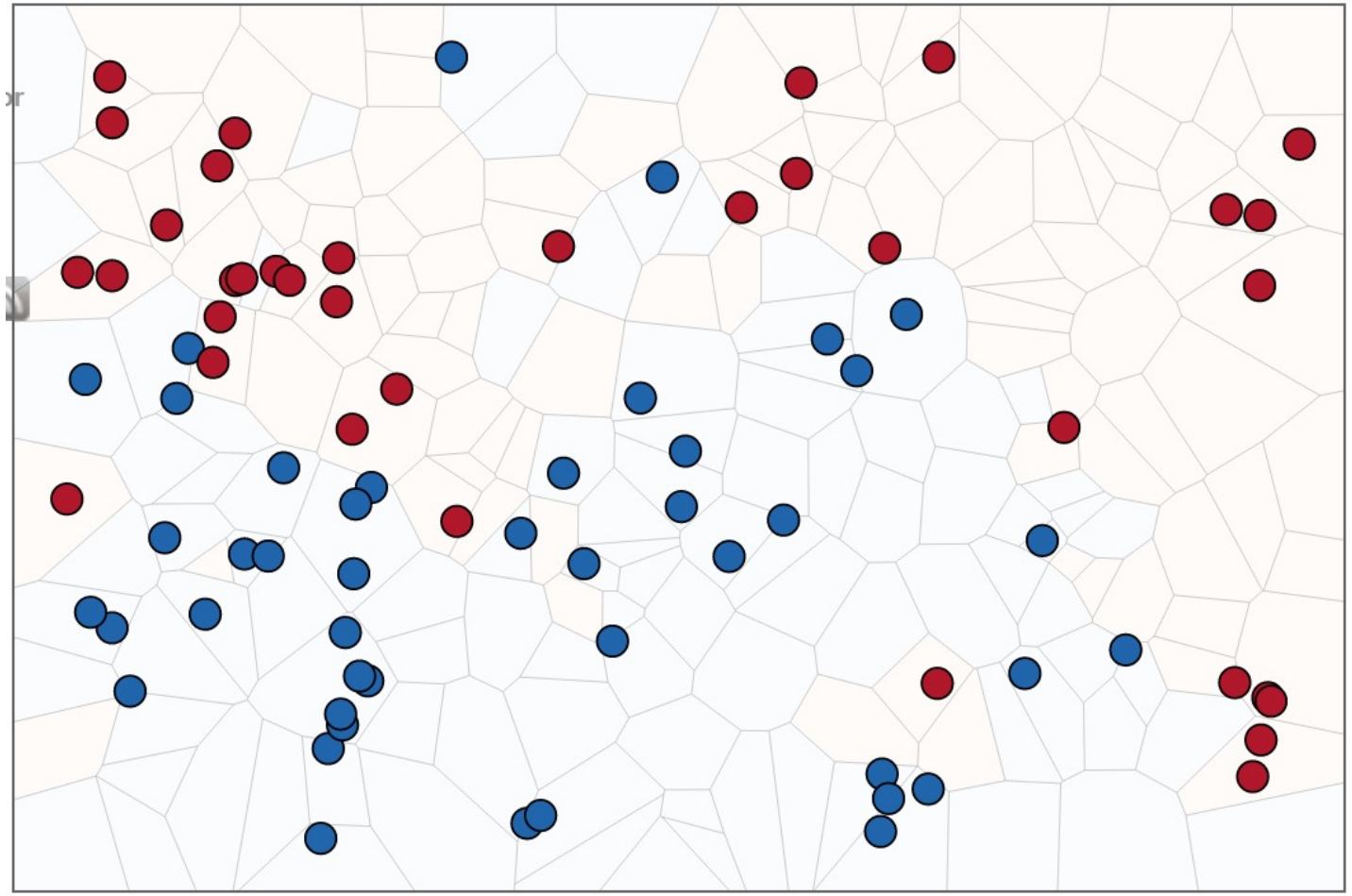
Voronoi diagram

http://scott.fortmann-roe.com/docs/BiasVariance.html



Voronoi diagram

http://scott.fortmann-roe.com/docs/BiasVariance.html



Voronoi diagram

http://scott.fortmann-roe.com/docs/BiasVariance.html

Nearest Neighbors

- **Classification rule:**

$$h(x) = y_i \text{ of closest neighbor } x_i \in S$$

- Works great when data is…

Nearest Neighbors

- **Classification rule:**

$$h(x) = y_i \text{ of closest neighbor } x_i \in S$$

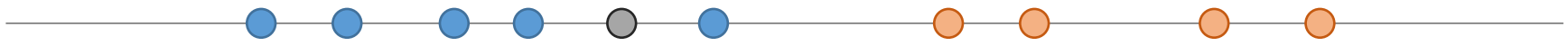
- Works great when data is… separable

Nearest Neighbors

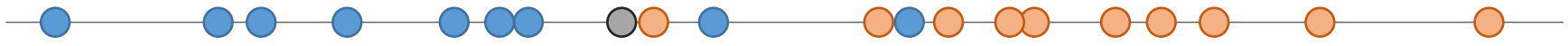
- **Classification rule:**

$$h(x) = y_i \text{ of closest neighbor } x_i \in S$$

- Works great when data is… separable



- What happens when it's not?



- **Solution**?
- Classify x using more than one neighbor

Nearest Neighbors

- **Classification rule:**

$$h(x) = \text{aggregate}\,\{y_i : x_i \in \text{Nei}(x)\}$$

- **Step I**: find closest examples. Need to define:
  1. Similarity measure: e.g., $\text{sim}(x, x') = \|x - x'\|_2$
  2. Inclusion criterion: e.g., top-k
  - Resulting "neighbors":
    $\text{Nei}_k(x; S) = \text{top}{-}k\{\text{sim}(x, x') : x' \in S\}$
- **Step II**: predict
  - Common choice for aggregation method:
    $\text{majority}\{y_1, \ldots, y_j\}$ = most frequent label in set
    (equivalent to $\text{sign}(\text{mean}(\{y_i\}))$)
- **Resulting classifier**: *k-nearest neighbors* (kNN)

In kNN: default norm is L2 unless we say otherwise

Nearest Neighbors

- **kNN classification rule:** $h_k(x; S) = \text{majority}\,\{y_i : x_i \in \text{Nei}_k(x; S)\}$
- **Q**: What is the model class?
- **A:** This is a trick question. Let's ask a different question.
- **Q:** Is it surprising that h depends on S?
- **A**: Supposedly, no – the output of a learning algorithm (whose input is S) is a function of S.
- But where is the learning algorithm?

- We've seen that learned models are often simple functions of (a subset of) the data.
- kNN takes this idea to the extreme and "memorizes" (=stores) the entire training set.
- This is called *instance-based learning* (kNN is a canonical example).
- But note that (at least for kNN), there is no actual learning going on (no optimization!).
- **Bonus**: current research suggests that, in effect, deep neural nets also memorize the data.

k-NN: Computational

<u>Runtime:</u>

- **Learning**: none
- **Prediction**:
  - Naïvely finding neighbors is $O(dm)$ – unrealistic in modern datasets
  - **But** – specialized data structure allow fast (approximate) retrieval in $O(\log m)$ using efficient indexing
  - Works well even for internet-sized data (e.g., search engines)

<u>Memory</u>:

- Classification requires storing all data – highly memory-intensive
- **But** – many tricks exist for reducing data dependence (at some cost in accuracy)

k-NN: Statistical

- Why/when does kNN work?
- As there is no actual "learning" going on, either:
  1. h_k is a closed form solution to a large class of models.
  2. The class of models includes only h_k (and so h_k is the only solution).
- Hence, in general, we would expect either overfitting or underfitting.
- **Conclusion**: we must be assuming something!

- Demonstration: http://scott.fortmann-roe.com/docs/BiasVariance.html

k-NN: Statistical

- **Recall**: our ideal classification rule is:

$$\text{argmax}_{y \in \{\pm 1\}} \, p(Y = y | x)$$

- Define *local empirical estimate* of conditional probability:

$$\hat{p}(y|x) = \frac{|\{(x_i, y_i) : \; y_i = y, x_i \in \text{Nei}(x)\}|}{|\text{Nei}(x)|}$$

- We can rewrite majority prediction $\hat{y} = h_k(x; S)$ as:

$$\hat{y} = \text{argmax}_{y \in \{\pm 1\}} \, \hat{p}(y|x)$$

$$= \text{sign}(\text{mean}(\{y_i : x_i \in \text{Nei}(x)\}))$$

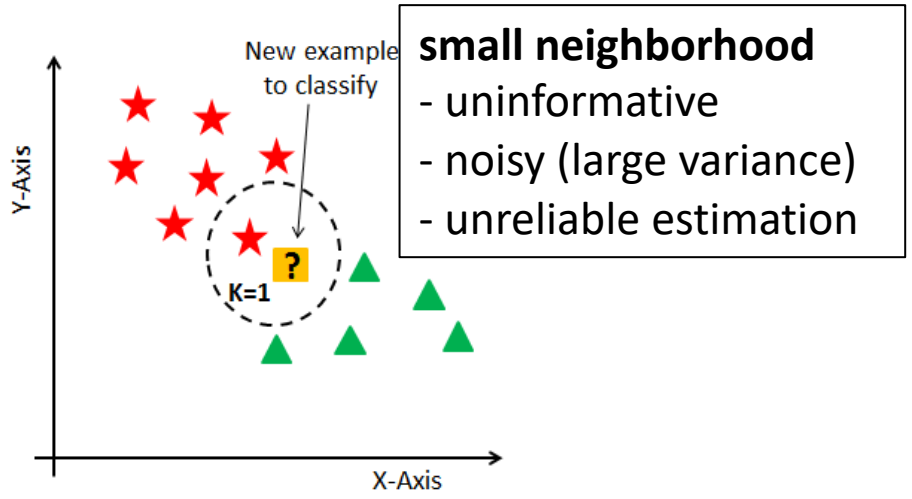
- k-NN makes sense if $\hat{p}(y|x)$ is a good estimate of $p(y|x)$ (for "most" x)
- **Q**: When does this hold?

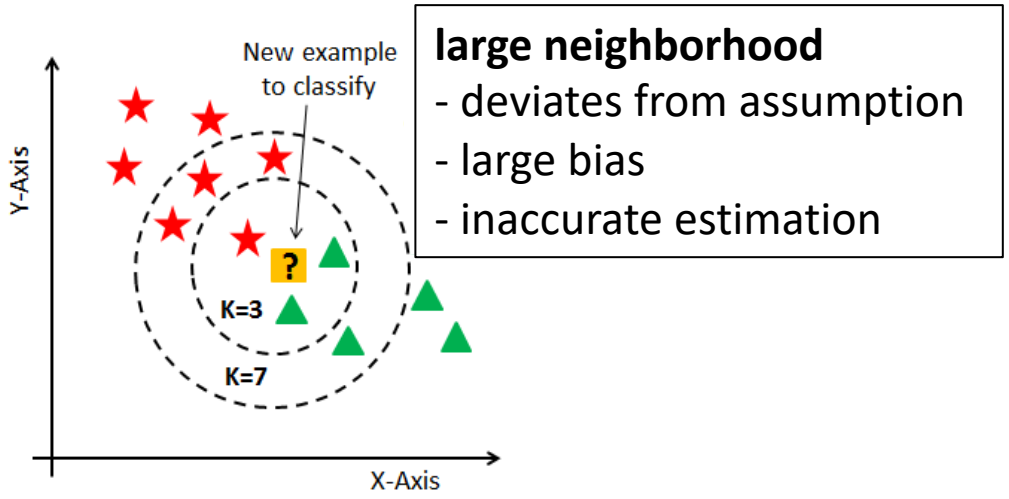
k-NN: Statistical

- **Informal**: if $p(y|x)$ changes smoothly in x, then $\hat{p}(y|x) \to p(y|x)$ as $m \to \infty$
- **Hand-wave**: "Smoothness" means $P(Y = y|x) \approx P(Y = y|x + \epsilon)$ (think Lipschitzness)

k-NN: Statistical

- **Informal**: if $p(y|x)$ changes smoothly in x, then $\hat{p}(y|x) \to p(y|x)$ as $m \to \infty$
- **Hand-wave**: "Smoothness" means $P(Y = y|x) \approx P(Y = y|x + \epsilon)$ (think Lipschitzness)
- But in practice, m is finite, and the choice of k introduces a tradeoff:





- The tradeoff here is between **bias** and **variance**; we will return to this later.
- The choice of k is crucial – but outside of our scope.

k-NN: Statistical

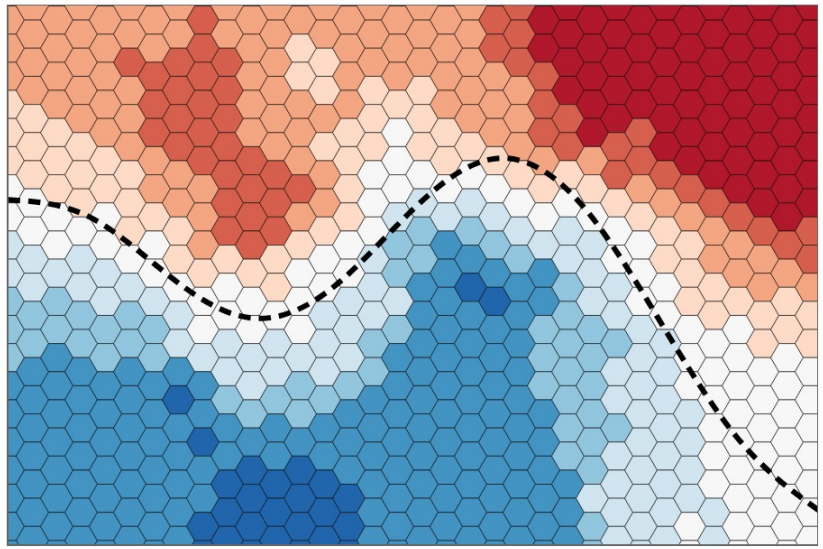
- Generalization bound (for 1-NN, with Lipschitz assumption on D):
- **Theorem**: $\mathbb{E}_{S\sim D^m}[L_D(h_S)] \leq 2L_D(h^*) + 4c\sqrt{d}m^{-1/(d+1)}$
- [Proof: UML book Sec. 19.2, p262]
- **Corollary**: $m \geq (4c\sqrt{d}/\epsilon)^{d+1}$
- If we want 2nd term of Theorem to be $\leq \epsilon$, we need very large m
- **Hence**: with large dimensionality d, we need many samples

k-NN: Discussion

- kNN classifiers are very expressive – use data as “mold”.
- Can work very well – when smoothness assumption holds.
- But we’ve hidden something!
- kNN requires two modeling choices:
  1. Number of neighbors, k
  2. Similarity measure
- Smoothness is w.r.t predefined similarity measure!

➢ What if we use the “wrong” measure?

➢ What happens to similarities in high dimensions?



Rule-based models

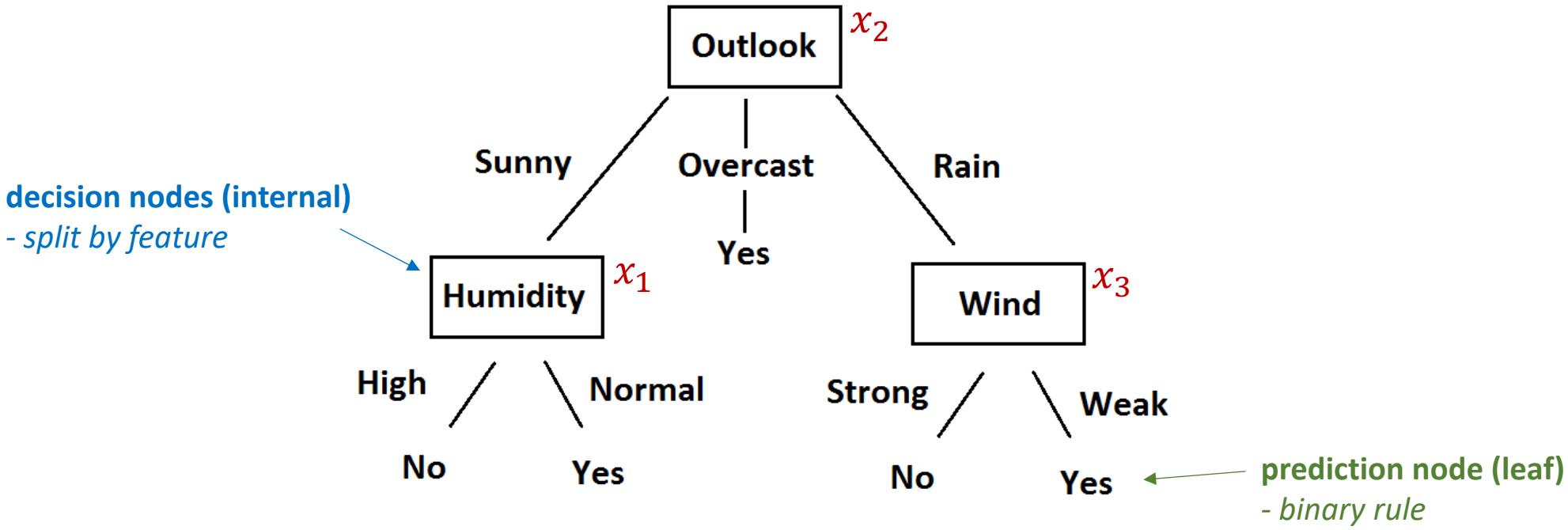
Hello, I am Akinator

Think about a real or fictional character.
I will try to guess who it is

akinator®

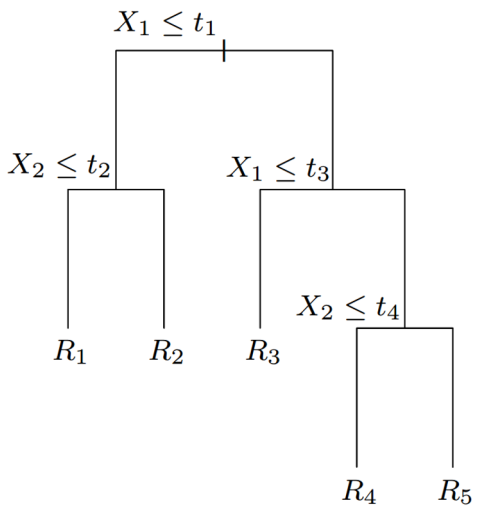
Decision trees

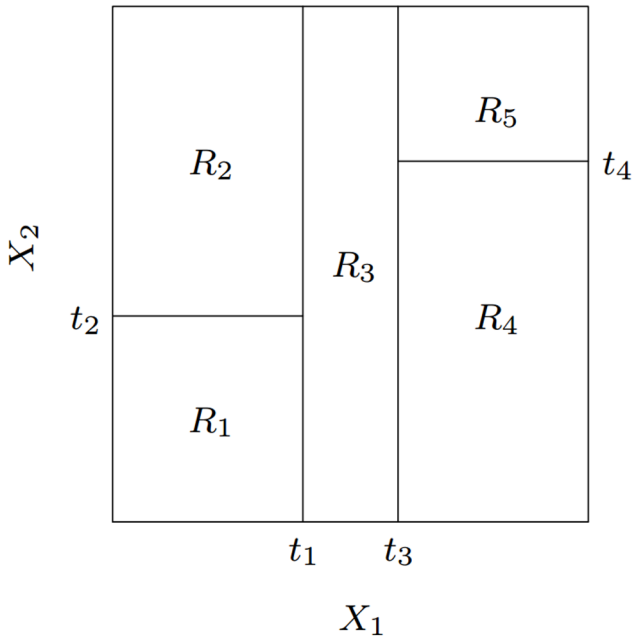
- Should I take an umbrella today?



Decision trees: statistics

- Model class: $H_{DT} = \{h_T(x) : T \text{ is a decision tree}\}$
- **Q**: how expressive are trees?
- **A**: Very.
- For 1D – relation to intervals.
- Without restrictions, always exists a tree T having zero empirical error, $L_S(h_T) = 0$.
- Beware overfitting!
- **Q**: What determines model complexity?
- **A**: Number of decision nodes (as proxy – tree depth).
- Controlling depth as a means to combat overfitting – more in Tirgul.





Decision trees: optimization

- Model class: $H_{DT} = \{h_T(x) : T \text{ is a decision tree}\}$
- **Q**: Can we efficiently solve ERM?
- **A**: No. This is a hard discrete optimization problem (combinatorially many trees).
- **Standard solution**: greedy + recursion = "grow tree"
- Greedy = at each step, "split" in way that *locally* minimizes empirical error.
- Template (recursive) algorithm:
  - Start with a root node (as a leaf node), representing S
  - Turn it into a decision node + two new leaf nodes – this partitions ("splits") the data
  - Recurse on new leaf nodes (and data subset)
- This is a heuristic approach, so no global optimality guarantees.
- **Main question**: how should we determine how to best split?

Decision trees: optimization

- **Observation**: $L_S(h) = 0$ when all leaf nodes are "pure".
- A leaf node v is "pure" if all examples $x \in S$ assigned to it are of the same class.

purity $\longrightarrow$ $$p_v := \frac{|\{(x, y) \in v : y = 1\}|}{|v|}$$

notational overload:
v = subset of S assigned to it

- Conversely, if a node is "impure", we should split it!

Decision trees: optimization

- **Observation**: $L_S(h) = 0$ when all leaf nodes are "pure".
- A leaf node v is "pure" if all examples $x \in S$ assigned to it are of the same class.

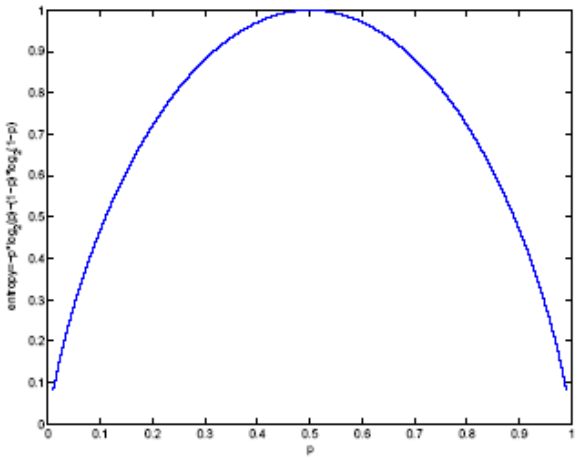
purity

$$p_v := \frac{|\{(x,y) \in v : y = 1\}|}{|v|}$$

notational overload:
v = subset of S assigned to it

- Conversely, if a node is "impure", we should split it!
- Reasonable measure of impurity – **entropy:**

$$H = -p_v \log p_v - (1 - p_v)\log(1 - p_v) = -\sum_{c \in C} p_v^c \log p_v^c$$

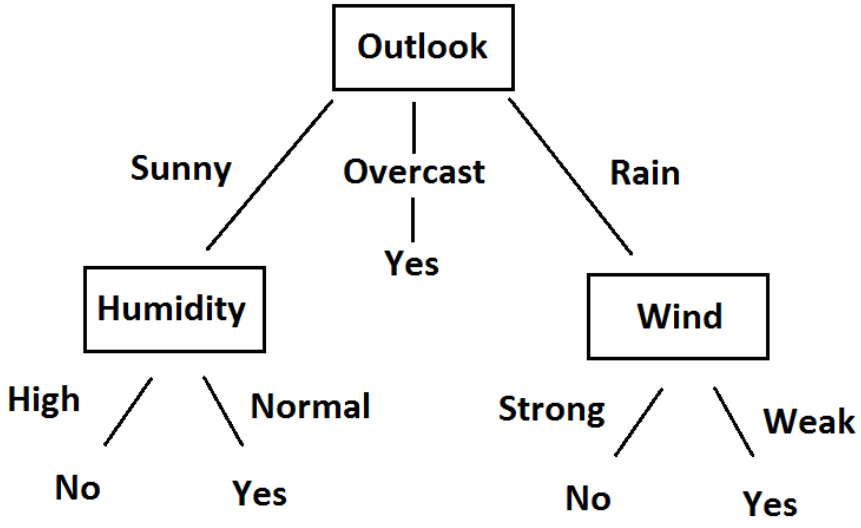


- **Greedy step**: partition dataset in way that minimizes entropy.

Decision trees: optimization

- **Classic algorithm**: ID3 (*Iterative Dichotomiser 3*)
  - Uses entropy as measure of purity
  - Splits according to attributes

- **Algorithm:** (full details in tirgul)
  1. Start with root as leaf.
  2. For each feature, compute best possible split (using entropy).
  3. Create decision node (and split data) using best feature.
  4. Recurse.



Linear models

Return of the thresholds

- Recall 1D thresholds: $h_\theta(z) = \mathrm{sign}(z - \theta), \quad z \in \mathbb{R}$
- For vector data, $x \in \mathbb{R}^d$, use scalar *score function*:

$$h_{f,\theta}(x) = \mathrm{sign}(f(x) - \theta), \quad f: \mathcal{X} \to \mathbb{R}$$

- Simplest model: *linear* score functions

$$f_{w,b}(x) = w^\top x + b$$

- Parameters: *weights* $w \in \mathbb{R}^d$, bias/offset $b \in \mathbb{R}$

Return of the thresholds

- Recall 1D thresholds: $h_\theta(z) = \text{sign}(z - \theta), \quad z \in \mathbb{R}$
- For vector data, $x \in \mathbb{R}^d$, use scalar *score function*:

$$h_{f,\theta}(x) = \text{sign}(f(x) - \theta), \quad f: \mathcal{X} \to \mathbb{R}$$

- Simplest model: *linear* score functions

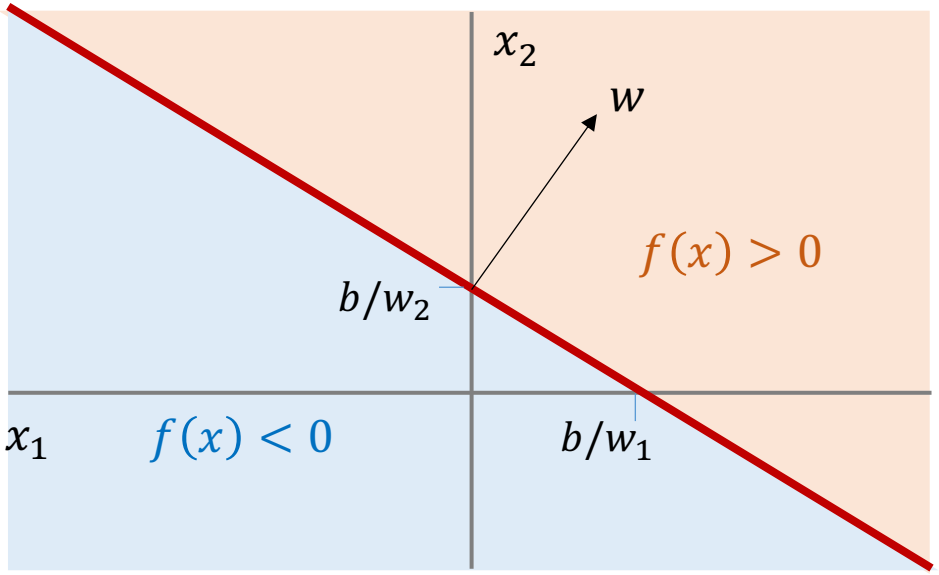
$$f_{w,b}(x) = w^\top x + b$$

- Parameters: *weights* $w \in \mathbb{R}^d$, bias/offset $b \in \mathbb{R}$
- This results in a *linear threshold classifiers*:

$$h_{w,b}(x) = \text{sign}(w^\top x + b)$$

- (θ went away because we can just set $b' = b - \theta$)

Geometric interpretation:



Return of the thresholds

- Recall 1D thresholds: $h_\theta(z) = \text{sign}(z - \theta), \quad z \in \mathbb{R}$
- For vector data, $x \in \mathbb{R}^d$, use scalar *score function*:

$$h_{f,\theta}(x) = \text{sign}(f(x) - \theta), \quad f: \mathcal{X} \to \mathbb{R}$$

- Simplest model: *linear* score functions

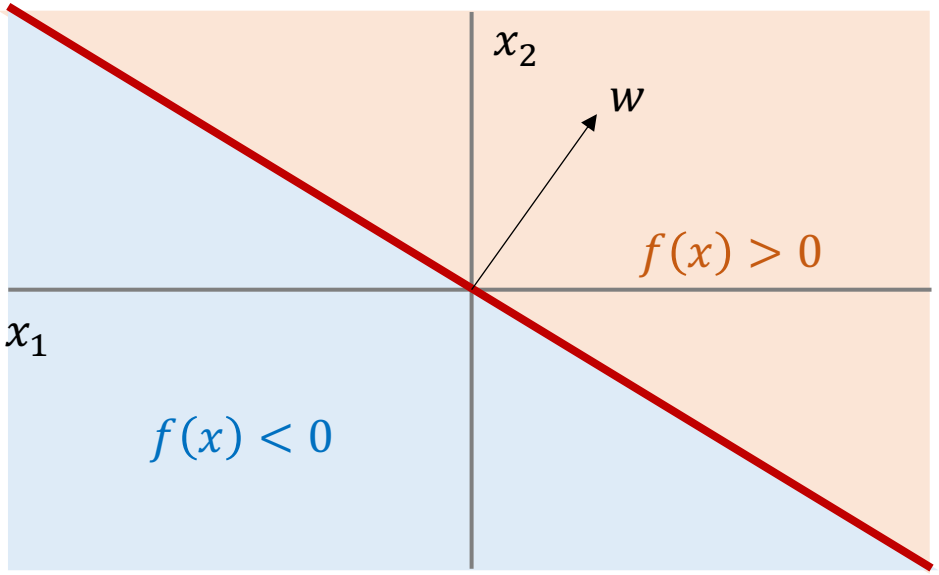
$$f_{w,b}(x) = w^\top x + b$$

- Parameters: *weights* $w \in \mathbb{R}^d$, bias/offset $b \in \mathbb{R}$
- This results in a *linear threshold classifiers*:

$$h_{w,b}(x) = \text{sign}(w^\top x + b)$$

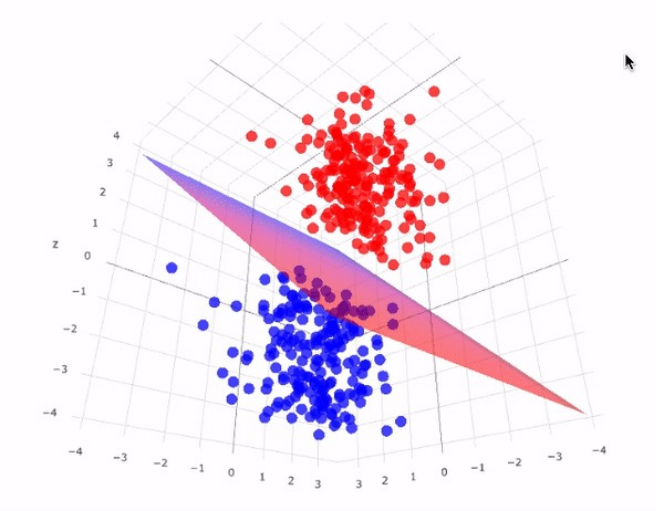
- (θ went away because we can just set $\theta = -b$)
- For simplicity, consider *homogeneous models*: $b = 0$
- (W.l.o.g*: can add feature with $x_{d+1} = 1$ to all examples)

Geometric interpretation:



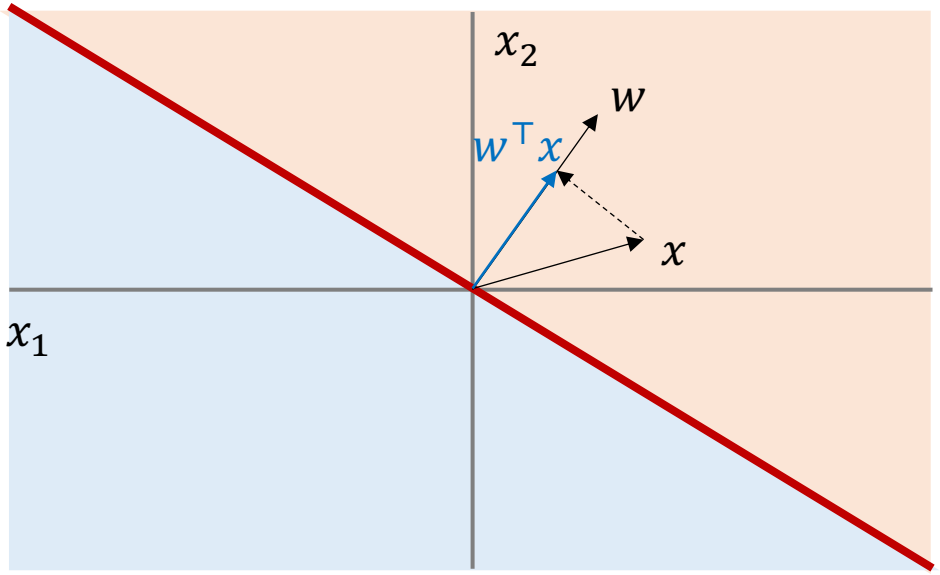
Return of the thresholds

- $w^\top x$ = project x on w
- $\mathrm{sign}(w^\top x)$ = same/different direction
- w is vector, but effectively "halves" space
  - Size of w doesn't matter
- Thus, w defines a *separating hyperplane*:



- h_w is called a *halfspace classifier*.

Geometric interpretation:



Linear learning

- **Model class:**

$$H = \{h_w(x) = \mathrm{sign}(w^\top x) : w \in \mathbb{R}^d\}$$

- **Remember**: this is a *modeling choice* with pros and cons.
- Expected error: $\underset{h \in H}{\mathrm{argmin}}\, \mathbb{E}_{(x,y)\sim D}[\mathbb{1}\{y \neq h(x)\}]$

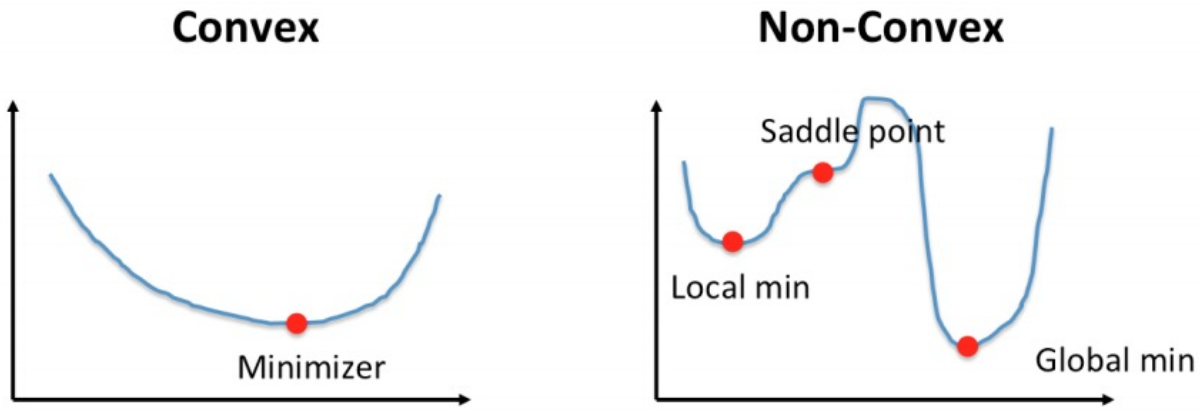
$$= \underset{w \in \mathbb{R}^d}{\mathrm{argmin}}\, \mathbb{E}_{(x,y)\sim D}[\mathbb{1}\{y \neq \mathrm{sign}(w^\top x)\}]$$

- Empirical error: $\underset{w \in \mathbb{R}^d}{\mathrm{argmin}} \sum_{i=1}^{m} \mathbb{1}\{y_i \neq \mathrm{sign}(w^\top x_i)\}$
- Can we effectively minimize the empirical error?
- **Claim**: Solving ERM for linear classifiers is NP-hard (won't prove; see UML p.111)
- **Who's to blame**: the $\mathbb{1}$ (makes it a discrete "subset selection" optimization)

Optimization

- What is an "optimization problem"?
- **Optimization template**: $\underset{w \in \mathbb{R}^d}{\text{argmin}}$ [objective] s.t. [constraints]

- What is a "nice" optimization problem?
- The nicest you can likely ask for is:
  1) continuous variables
  2) continuous and **convex** objective
  3) no constraints, or at least linear



- For our problem, we'll aim for all of the above!
- This will allow us to use **gradient descent** as an optimization algorithm

Warm up: Separability

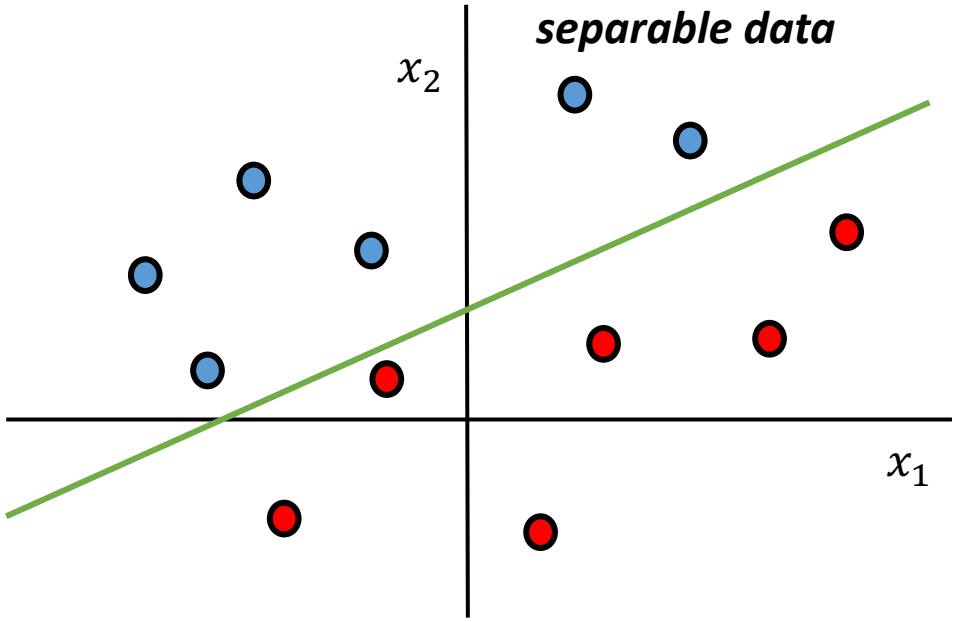
- **First step**: let's look at an easier problem
- Assume the data is linearly separable

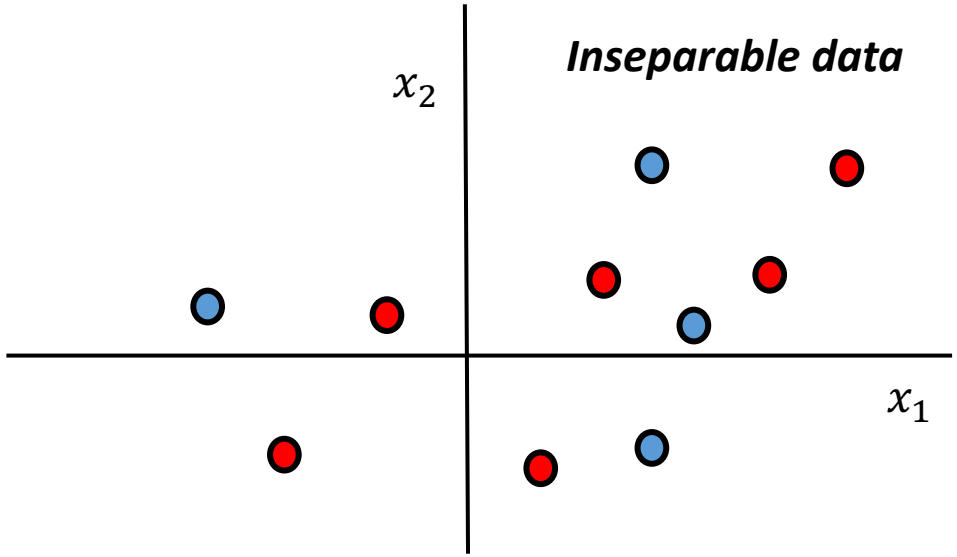
Definition: The dataset $S = \{(x_i, y_i)\}_{i=1}^m$ is (homogeneously) *linearly separable* if exists $w \in \mathbb{R}^d$ such that:

$$\forall i \in [m] \qquad w^\top x_i > 0 \text{ iff } y_i = 1$$

or, equivalently:

$$\forall i \in [m] \qquad y_i \cdot w^\top x_i > 0$$





*Note: separability is a property of data+model class, *not* of distribution, as we (incorrectly) defined before.

Optimization under separability

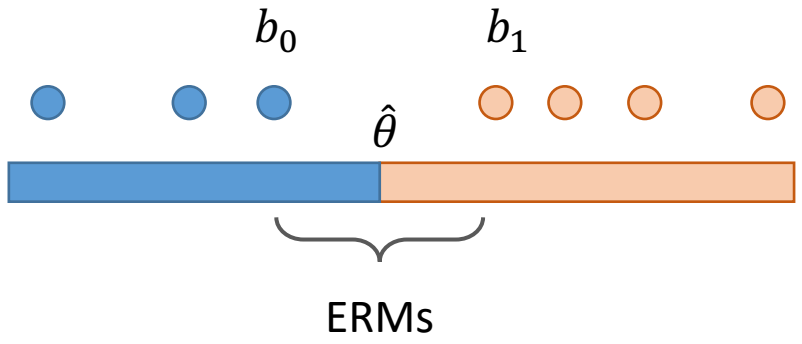
- Linear separability: $\exists w \;\; \forall i \in [m] \;\; y_i \cdot w^\top x_i > 0$
- **Observation**: any w that satisfies these inequalities is an ERM minimizer ($L_S(h_w) = 0$)
- **Idea**: use inequalities as constraints!
- Any feasible solution will be an ERM minimizer
- Template (surrogate) learning objective:

$$\underset{w}{\operatorname{argmin}}[objective] \;\; st \; \forall i \in [m] \;\; y_i \cdot w^\top x_i > 0$$

- **Note**: constraints are linear in w, maintaining convexity
- **Next**: choose an objective

Margins

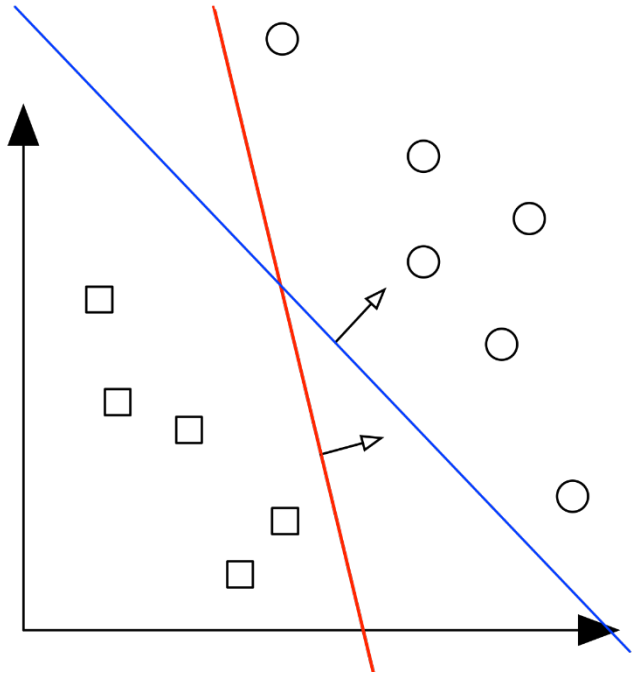
- Recall 1D separable from last time
- We could have chosen as θ any point in $[b_0, b_1]$
- But something felt "fishy" about choosing $\theta = b_0$
- And somehow the average $\theta = \frac{b_0+b_1}{2}$ "feels" better
- Why?



Margins

- Recall 1D separable from last time
- We could have chosen as θ any point in $[b_0, b_1]$
- But something felt "fishy" about choosing $\theta = b_0$
- And somehow the average $\theta = \frac{b_0+b_1}{2}$ "feels" better
- Why?

- This also happens in high-dimensional halfspaces
- Intuitively – which hyperplane would you choose?
- Now can you say why?
  (are you making implicit additional assumptions?)



Margins

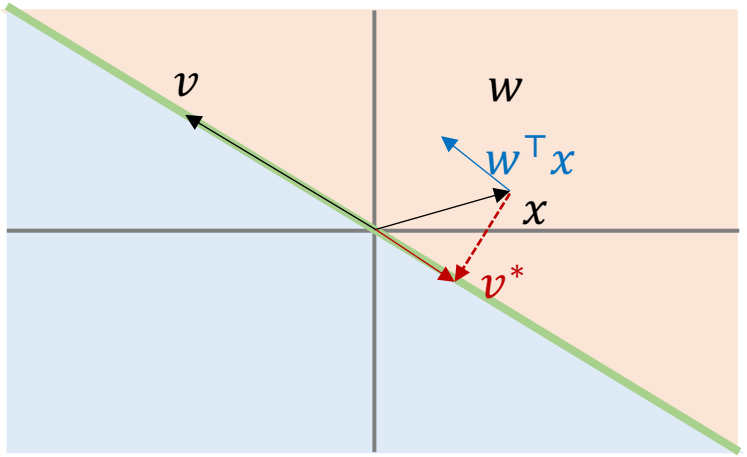
- **Claim** (won't prove):
  the distance between a point x and a hyperplane w is:

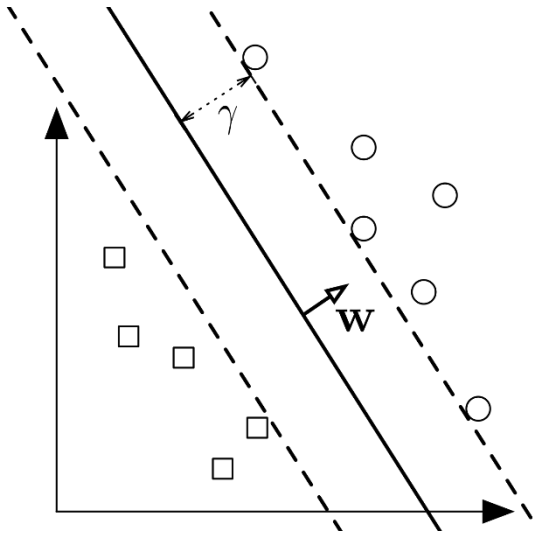
$$\mathrm{dist}(x, w) \stackrel{\text{def}}{=} \min\{\|x - v\| : w^\top v = 0\} = \frac{|w^\top x|}{\|w\|}$$

- **Def**: the **margin** of a linear classifier h_w is the distance to w from the closest point in dataset,

$$\mathrm{margin}(w; S) = \min_{i \in [m]} \frac{|w^\top x_i|}{\|w\|} := \gamma(w; S)$$

- (normalization is because hyperplanes are scale-invariant objects)
- **Observation**: one characterization of our "solution" is that it has the *largest margin* of all classifiers
- =largest (of classifiers) smallest (of points) distance
- **Next step**: use max margin as an objective





Max-margin classification

- The max-margin optimization problem (for separable data):

$$\text{argmax}_w \, \gamma(w; S) \;\; \text{s.t.} \; y_i \cdot w^\top x_i > 0 \qquad \forall i \in [m]$$

$$\equiv \text{argmax}_w \, \min_{i \in [m]} \frac{|w^\top x_i|}{\|w\|} \;\; \text{s.t.} \;\; y_i \cdot w^\top x_i > 0 \qquad \forall i \in [m]$$

$$\equiv \text{argmax}_w \, \frac{1}{\|w\|} \min_{i \in [m]} |w^\top x_i| \;\; \text{s.t.} \;\; y_i \cdot w^\top x_i > 0 \qquad \forall i \in [m]$$

- **Still hard to solve!**
- **Recall**: hyperplane is scale-invariant
- This means that for every "direction" $\vec{w}$ (with $\|\vec{w}\| = 1$),
  we can choose a "representative" $w = \alpha \vec{w}$ for some α

Max-margin classification

- The max-margin optimization problem (for separable data):

$$\equiv \operatorname{argmax}_w \frac{1}{\|w\|} \min_{i \in [m]} |w^\top x_i| \text{ s.t. } y_i \cdot w^\top x_i > 0 \qquad \forall i \in [m]$$

- **Recall**: hyperplane is scale-invariant
- Idea: for each $\vec{w}$, choose α so that our representative $w = \alpha \vec{w}$ has $\min_{i \in [m]} |w^\top x_i| = 1$
- **Dubious move**: add constraint $\min_{i \in [m]} |w^\top x_i| = 1$
- This means we rescaled w to have a margin of size 1 (~ scaling the entire objective)

Max-margin classification

(1) $w_1 = \text{argmax}_w \frac{1}{\|w\|_2} \min_{i\in[m]} |w^\top x_i|$ s.t. $y_i \cdot w^\top x_i \geq 0 \quad \forall i \in [m]$

(2) $w_2 = \text{argmax}_w \frac{1}{\|w\|_2}$ s.t. $\min_{i\in[m]} |w^\top x_i| = 1, \; y_i \cdot w^\top x_i \geq 0 \; \forall i \in [m]$

F_i - objective
C_i - constraints

- **Claim**: solution of (2) is optimal for (1)
- **Proof**: $C_2 \subseteq C_1 \Rightarrow w_2 \in C_1$, so remains to show $F_1(w_2) \geq F_1(w_1)$. Assume contrarily $F_1(w_1) > F_1(w_2)$ (*).
- Observation: (1) does not have unique optimum, but "classes" of same-direction solutions $\{\alpha w\}$.
- **Lemma**: if w is optimal for (1), then for any $\alpha \in \mathbb{R}$, αw is also optimal for (1)
- **Proof**: scale invariance:
  - Objective: $F_1(w) = \min_{i\in[m]} \left|\left(\frac{w}{\|w\|_2}\right)^\top x_i\right| = \min_{i\in[m]} \left|\left(\frac{\alpha w}{\|\alpha w\|_2}\right)^\top x_i\right| = F_1(\alpha w)$
  - Constraints: $y_i w^\top x_i \geq 0$ iff $y_i \alpha w^\top x_i \geq 0$

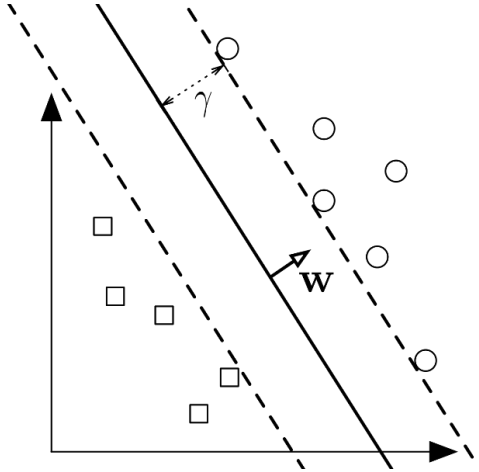
Think of (2) as (1) but with the twist that for each "direction", we choose as a representative the w with $\min_{i\in[m]} |w^\top x_i| = 1$.

- Denote $\frac{1}{\alpha_1} = \min_{i\in[m]} |w_1^\top x_i|$, then $\frac{1}{\|w_1\|_2} \cdot \frac{1}{\alpha_1} = F_1(\alpha_1 w_1) =_{lemma} F_1(w_1) >_* F_1(w_2) = \frac{1}{\|w_2\|_2} \cdot 1 = F_2(w_2)$
- But $\alpha_1 w_1 \in C_2$ and $F_2(\alpha_1 w_1) = \frac{1}{\alpha_1 \|w_1\|_2} > F_2(w_2)$, which contradicts the optimality of w_2 for (2).

Hard SVM

(2) $w_2 = \text{argmax}_w \frac{1}{\|w\|_2}$ s.t. $\min_{i\in[m]} |w^\top x_i| = 1, \;\; y_i \cdot w^\top x_i \geq 0 \;\; \forall i \in [m]$

(3) $w_3 = \text{argmin}_w \|w\|_2^2$ s.t. $y_i \cdot w^\top x_i \geq 1 \;\; \forall i \in [m]$



- **Claim**: (2),(3) have same optima
- **Proof:**
- Objective:
  - $F_3(w) = \left(\frac{1}{F_2(w)}\right)^2$ and $\frac{1}{z^2}$ is decreasing monotone so $\max F_2 = \min F_3$
- Constraints:
  - $w_2 \in C_3$: $y_i w^\top x_i \geq 0 \Rightarrow \min_{i\in[m]} |w^\top x_i| = \min_{i\in[m]} y_i w^\top x_i \Rightarrow y_i w^\top x_i \geq 1$
  - $w_3 \in C_2$: denote $\beta = y_i \cdot w_3^\top x_i \geq 1$.
    Assume contradictorily that $\beta > 1$, but then $\frac{w_3}{\beta} \in C_3$ and $F_3\left(\frac{w_3}{\beta}\right) < F_3(w_3)$, which contradicts the optimality of w_3.

Discussion

- **Final Hard SVM objective**: $\boxed{w_{\text{H-SVM}} = \text{argmin}_w \|w\|_2^2 \quad \text{s.t.} \;\; y_i \cdot w^\top x_i \geq 1 \;\; \forall i \in [m]}$
- **Main insight**: increasing margin ≡ reducing norm

(=1 as constraint implicit)

$$\gamma(w; S) := \min_{i \in [m]} \frac{|w^\top x_i|}{\|w\|_2} \Leftrightarrow \frac{1}{\|w\|_2}$$

what we *want* vs. what we *do*

- Results in simple, convex objective with linear constraints (easy to optimize + unique solution)

- **Claim**: at least one example (but possibly more) "touches" margin (=constraint is tight)
- Margin-toucing examples are called "**support vectors**": (hence the name "SVM")
  removing "support" examples changes learned model (removing other examples does not)
- These will pop up again later

Linear models

- **Modeling:** linear models have (apparently) limited expressive power.
- (Think decision boundaries, e.g., vs. kNN, trees)
- **Optimization**: But this simplicity makes learning tractable – to be seen
- **Statistics**: We'll see some generalization bounds later on

- And, for many problems, linear models work very well

- **Next week**:

1. Drop separability assumption (Soft SVM)
2. Expand expressivity but remain linear (kernels)

- **See you next week!**