

Hyperplane Based classifiers

Loss Functions for Classification

- In regression (assuming linear model $\hat{y} = \mathbf{w}^T \mathbf{x}$), some common loss fn

$$\ell(y, \hat{y}) = (y - \hat{y})^2$$

$$\ell(y, \hat{y}) = |y - \hat{y}|$$

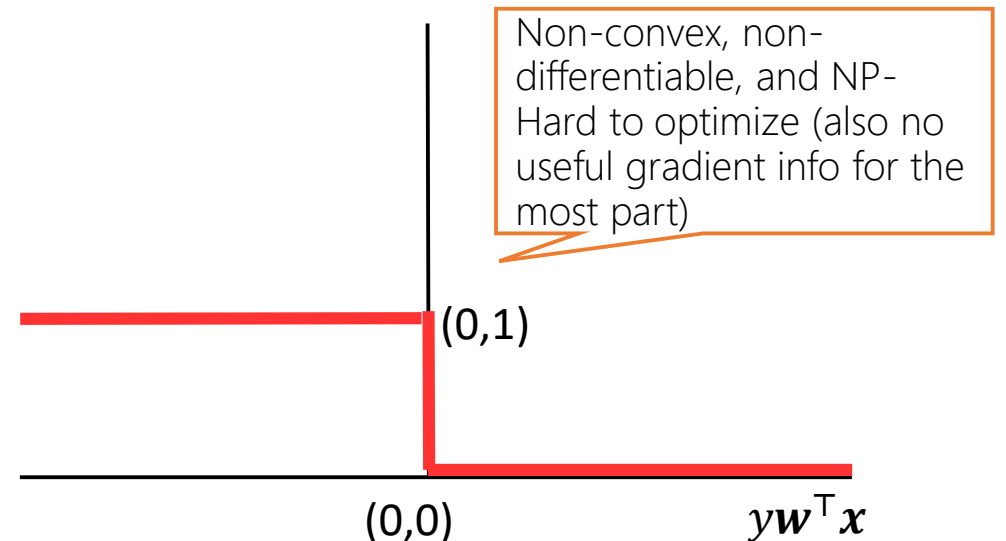
- These measure the difference between the true output and model's prediction
- What about loss functions for classification where $\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{x})$?
- Perhaps the most natural classification loss function would be a "0-1 Loss"

- Loss = 1 if $\hat{y} \neq y$ and Loss = 0 if $\hat{y} = y$.

- Assuming labels as +1/-1, it means

$$\ell(y, \hat{y}) = \begin{cases} 1 & \text{if } y\mathbf{w}^T \mathbf{x} < 0 \\ 0 & \text{if } y\mathbf{w}^T \mathbf{x} \geq 0 \end{cases}$$

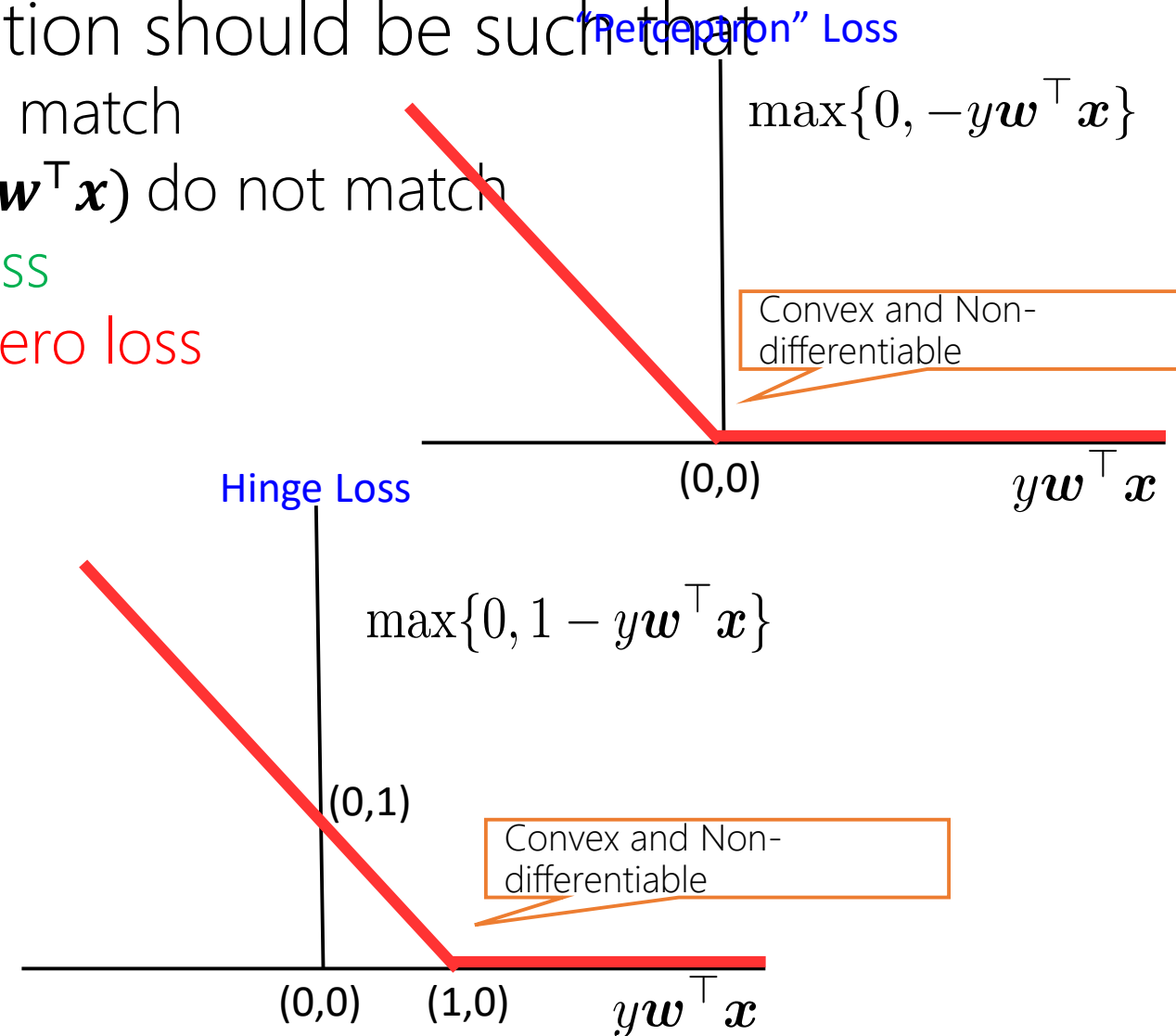
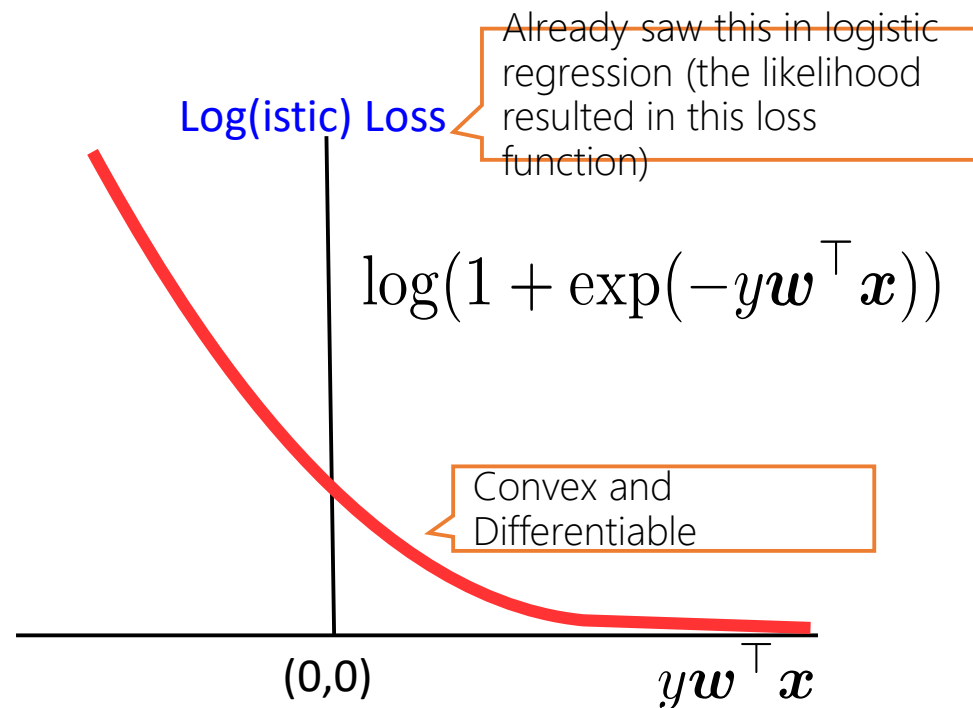
Same as $\mathbb{I}[y\mathbf{w}^T \mathbf{x} < 0]$ or $\mathbb{I}[\text{sign}(\mathbf{w}^T \mathbf{x}) \neq y]$



Loss Functions for Classification

- An ideal loss function for classification should be such that

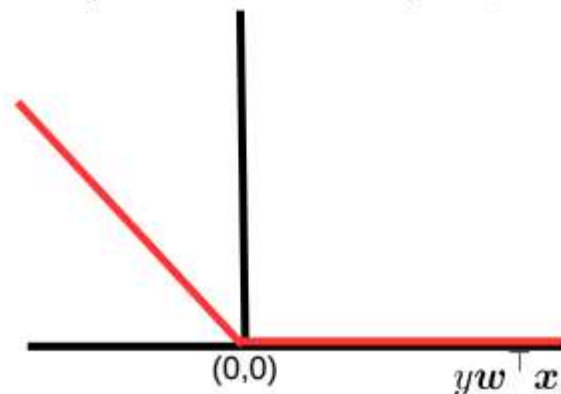
- Loss is small/zero if y and $\text{sign}(\mathbf{w}^\top \mathbf{x})$ match
- Loss is large/non-zero if y and $\text{sign}(\mathbf{w}^\top \mathbf{x})$ do not match
- Large positive $y\mathbf{w}^\top \mathbf{x} \Rightarrow$ small/zero loss
- Large negative $y\mathbf{w}^\top \mathbf{x} \Rightarrow$ large/non-zero loss



Learning by Optimizing Perceptron Loss

- Let's ignore the bias term b for now. So the hyperplane is simply $\mathbf{w}^\top \mathbf{x} = 0$
- The Perceptron loss function: $L(\mathbf{w}) = \sum_{n=1}^N \max\{0, -y_n \mathbf{w}^\top \mathbf{x}_n\}$. Let's do SG "Perceptron" Loss: $\max\{0, -y \mathbf{w}^\top \mathbf{x}\}$ Subgradients w.r.t. \mathbf{w}

One randomly chosen example in each iteration



$$\mathbf{g}_n = \begin{cases} 0, & \text{for } y_n \mathbf{w}^\top \mathbf{x}_n > 0 \\ -y_n \mathbf{x}_n & \text{for } y_n \mathbf{w}^\top \mathbf{x}_n < 0 \\ k y_n \mathbf{x}_n & \text{for } y_n \mathbf{w}^\top \mathbf{x}_n = 0 \quad (\text{where } k \in [-1, 0]) \end{cases}$$

- If we use $k = 0$ then $\mathbf{g}_n = 0$ for $y_n \mathbf{w}^\top \mathbf{x}_n \geq 0$, and $\mathbf{g}_n = -y_n \mathbf{x}_n$ for $y_n \mathbf{w}^\top \mathbf{x}_n < 0$
- Non-zero gradients only when the model makes a mistake on current

The Perceptron Algorithm

- Stochastic Sub-grad desc on Perceptron loss is also known as the Perceptron algorithm

Stochastic SubGD

- 1 Initialize $\mathbf{w} = \mathbf{w}^{(0)}$, $t = 0$, set $\eta_t = 1, \forall t$
- 2 Pick some (\mathbf{x}_n, y_n) randomly.
- 3 If current \mathbf{w} makes a **mistake** on (\mathbf{x}_n, y_n) , i.e., $y_n \mathbf{w}^{(t)\top} \mathbf{x}_n < 0$

$$\begin{aligned}\mathbf{w}^{(t+1)} &= \mathbf{w}^{(t)} + y_n \mathbf{x}_n \\ t &= t + 1\end{aligned}$$
- 4 If not converged, go to step 2.

Note: An example may get chosen several times during the entire run

Mistake condition

Updates are "corrective" : If $y_n = +1$ and $\mathbf{w}^\top \mathbf{x}_n < 0$, after the update $\mathbf{w}^\top \mathbf{x}_n$ will be less negative. Likewise, if $y_n = -1$ and $\mathbf{w}^\top \mathbf{x}_n > 0$, after the update $\mathbf{w}^\top \mathbf{x}_n$ will be less positive



If training data is linearly separable, the Perceptron algo will converge in a finite number of iterations (Block & Novikoff theorem)

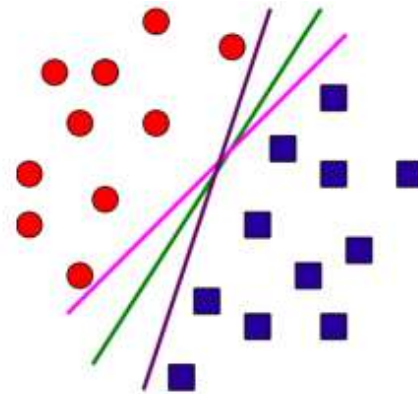
- An example of an **online learning** algorithm (processes one training ex. at a time)
- Assuming $\mathbf{w}^{(0)} = \mathbf{0}$, easy to see that the final \mathbf{w} has the form $\mathbf{w} = \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n$

Meaning of α_n may be different

α_n is total number of mistakes made by the algorithm on example (\mathbf{x}_n, y_n)

Perceptron and (lack of) Margins

- Perceptron would learn a hyperplane (of many possible) that separates the classes



Basically, it will learn the hyperplane which corresponds to the \mathbf{w} that minimizes the Perceptron loss

Kind of an "unsafe" situation to have – ideally would like it to be reasonably away from closest training examples from either class

- Doesn't guarantee any "margin" around the hyperplane
 - The hyperplane can get arbitrarily close to some training examples
 - This may not be good for generalization performance
- Can artificially introduce margin by changing the mistake condition to $y_n \mathbf{w}^T \mathbf{x}_n < \gamma$
 - $\gamma > 0$ is some pre-specified margin
- Support Vector Machine (SVM) does it directly by learning the **max. margin**

Support Vector Machines

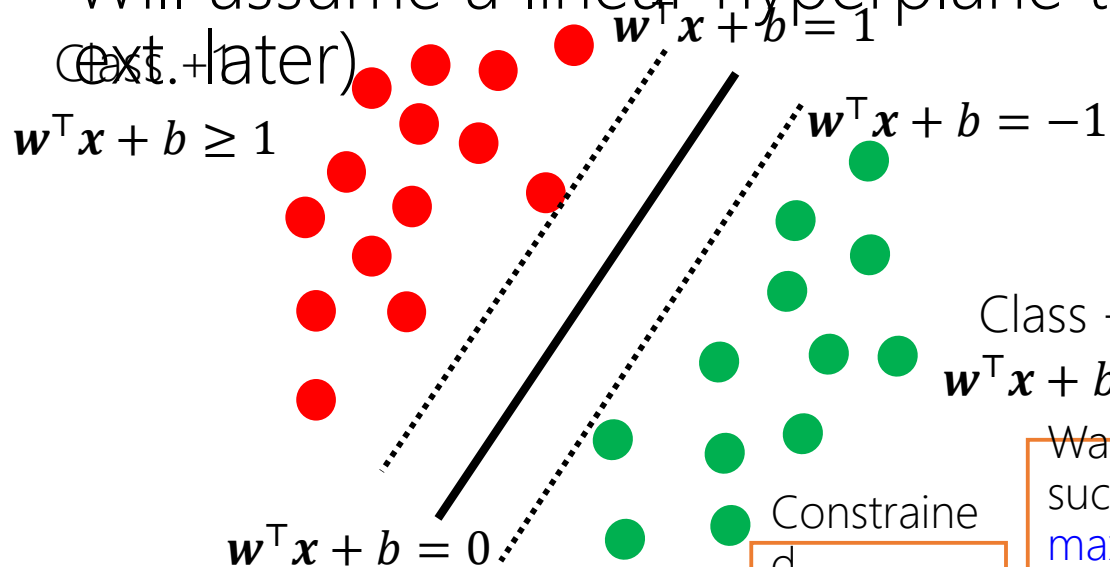
Support Vector Machine (SVM)

8

SVM originally proposed by Vapnik and colleagues in early 90s

- Hyperplane based classifier. Ensures a large margin around the hyperplane

- Will assume a linear hyperplane to be of the form $\mathbf{w}^T \mathbf{x} + b = 0$ (nonlinear ext. later)



Distance from the closest point (on either side)

$$\mathbf{w}^T \mathbf{x}_n + b \geq 1 \quad \text{if } y_n = +1$$

$$\mathbf{w}^T \mathbf{x}_n + b \leq -1 \quad \text{if } y_n = -1$$

$$y_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1 \quad \forall n$$

Distance of an input \mathbf{x}_n from the h.p.

"Margin" of the hyper

$$\gamma = \min_{1 \leq n \leq N} \frac{|\mathbf{w}^T \mathbf{x}_n + b|}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$$

Want the hyperplane (\mathbf{w}, b) such that this margin is maximized (max-margin hyperplane) and

$$y_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1 \quad \forall n$$

Constrained optimization problem

$$\text{Total margin} = \frac{2}{\|\mathbf{w}\|}$$

The 1/-1 in supp. h.p. equations is arbitrary; can replace by any scalar m/-m and solution won't change, except a simple scaling of \mathbf{w}

- Two other "supporting" hyperplanes defining a "no man's land"

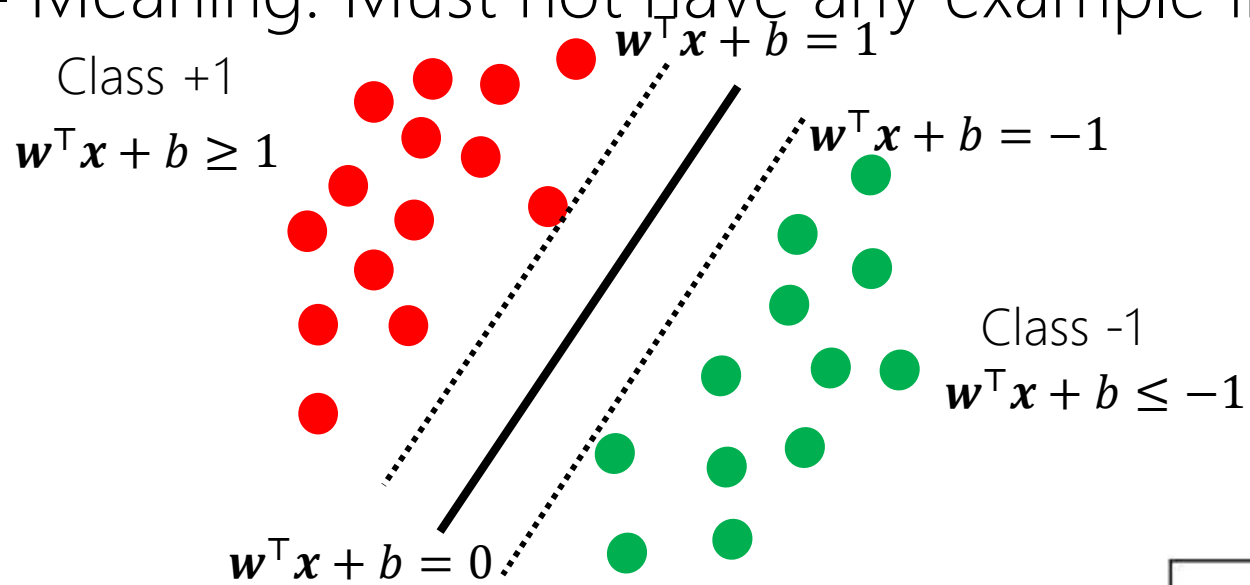
- Ensure that zero training examples fall in this region (will relax later)

$$\mathbf{w}^T \mathbf{x} + b = 1$$

Hard-Margin SVM

- Hard-Margin: Every training example must fulfil margin condition $y_n(\mathbf{w}^\top \mathbf{x}_n + b) \geq 1$

- Meaning: Must not have any example in the no-man's land

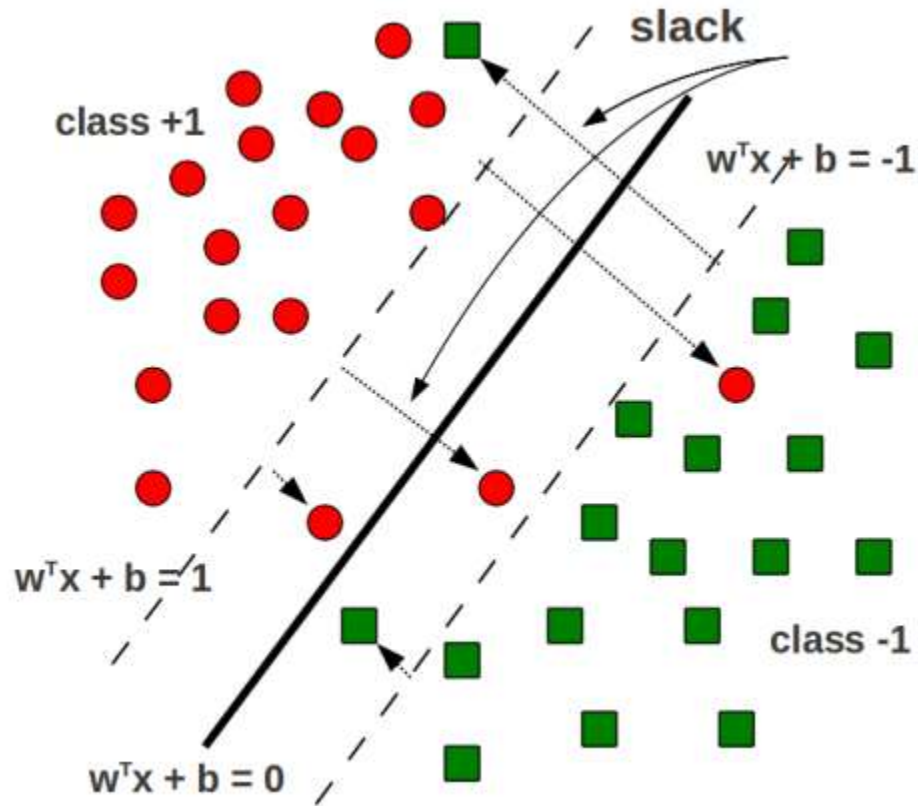


Constrained optimization problem with N inequality constraints. Objective and constraints both are convex

- Also want to maximize margin $2\gamma = \frac{2}{\|\mathbf{w}\|}$
- Equivalent to minimizing $\|\mathbf{w}\|^2$ or $\frac{\|\mathbf{w}\|^2}{2}$
- The objective func. for hard-margin SVM

$$\begin{aligned} \min_{\mathbf{w}, b} \quad & f(\mathbf{w}, b) = \frac{\|\mathbf{w}\|^2}{2} \\ \text{subject to} \quad & y_n(\mathbf{w}^\top \mathbf{x}_n + b) \geq 1, \quad n = 1, \dots, N \end{aligned}$$

Soft-Margin SVM (More Commonly Used)



- Allow some training examples to fall within the no-man's land (margin region)
- Even okay for some training examples to fall totally on the wrong side of h.p.
- Extent of "violation" by a training input (\mathbf{x}_n, y_n) is known as **slack** $\xi_n \geq 0$

- $\xi_n > 1$ means totally on the wrong side

$$\mathbf{w}^T \mathbf{x}_n + b \geq 1 - \xi_n \quad \text{if } y_n = +1$$

$$\mathbf{w}^T \mathbf{x}_n + b \leq -1 + \xi_n \quad \text{if } y_n = -1$$

Soft-margin constraint $y_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1 - \xi_n \quad \forall n$

Soft-Margin SVM (Contd)

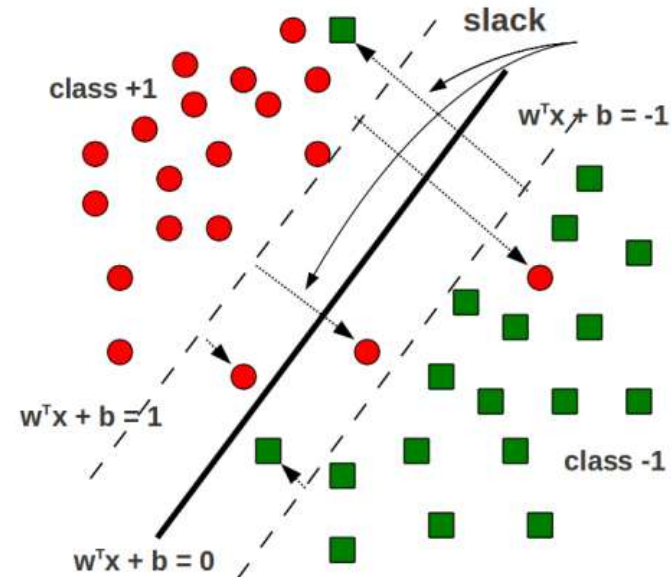
- Goal: Still want to maximize the margin such that

Sum of slacks is like the training error

- Soft-margin constraints $y_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1 - \xi_n$ are satisfied for all training ex.

- Do not have too many margin violations (sum of slacks $\sum_{n=1}^N \xi_n$ should be small)

- The objective func. for soft-margin SVM



$$\min_{\mathbf{w}, b, \xi} f(\mathbf{w}, b, \xi) = \frac{\|\mathbf{w}\|^2}{2} + C \sum_{n=1}^N \xi_n$$

subject to $y_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0 \quad n = 1, \dots, N$

Annotations:

- $\frac{\|\mathbf{w}\|^2}{2}$: Inversely prop. to margin
- C : Trade-off hyperparam
- $\sum_{n=1}^N \xi_n$: training error
- Overall problem: Constrained optimization problem with $2N$ inequality constraints. Objective and constraints both are convex.

- Hyperparameter C controls the trade off between large margin and small training error (need to tune)
 - Large C : small training error but also small margin (bad)
 - Small C : large margin but large training error (bad)

Solving the SVM Problem

Solving Hard-Margin SVM

- The hard-margin SVM optimization problem is

$$\begin{aligned} \min_{\mathbf{w}, b} \quad & f(\mathbf{w}, b) = \frac{\|\mathbf{w}\|^2}{2} \\ \text{subject to} \quad & 1 - y_n(\mathbf{w}^T \mathbf{x}_n + b) \leq 0, \quad n = 1, \dots, N \end{aligned}$$

- A constrained optimization problem. One option is to solve using Lagrange's method
- Introduce Lagrange multipliers α_n ($n = 1, \dots, N$), one for each constraint, and solve

$$\min_{\mathbf{w}, b} \max_{\alpha \geq 0} \mathcal{L}(\mathbf{w}, b, \alpha) = \frac{\|\mathbf{w}\|^2}{2} + \sum_{n=1}^N \alpha_n \{1 - y_n(\mathbf{w}^T \mathbf{x}_n + b)\}$$

- $\alpha = [\alpha_1, \alpha_2, \dots, \alpha_N]$ denotes the vector of Lagrange multipliers
- It is easier (and helpful; we will soon see why) to solve the dual: min and

Solving Hard-Margin SVM

- The dual problem (min then max) is

$$\max_{\alpha \geq 0} \min_{\mathbf{w}, b} \mathcal{L}(\mathbf{w}, b, \alpha) = \frac{\mathbf{w}^T \mathbf{w}}{2} + \sum_{n=1}^N \alpha_n \{1 - y_n(\mathbf{w}^T \mathbf{x}_n + b)\}$$

Note: if we ignore the bias term b then we don't need to handle the constraint $\sum_{n=1}^N \alpha_n y_n = 0$ (problem becomes a bit more easy to solve)



Otherwise, the α_n 's are coupled and some opt. techniques such as co-ordinate ascent can't easily be applied

- Take (partial) derivatives of \mathcal{L} w.r.t. \mathbf{w} and b and setting them to zero gives (verify)

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n \quad \frac{\partial \mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{n=1}^N \alpha_n y_n = 0$$

α_n tells us how important training example (\mathbf{x}_n, y_n) is

- The solution \mathbf{w} is simply a weighted sum of all the training inputs

This is also a "quadratic program" (QP) – a quadratic function of the variables α

$$\max_{\alpha \geq 0} \mathcal{L}_D(\alpha) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{m,n=1}^N \alpha_m \alpha_n y_m y_n (\mathbf{x}_m^T \mathbf{x}_n)$$

Note that inputs appear only as pairwise dot products. This will be useful later on when we make SVM nonlinear using kernel methods



$$\max_{\alpha \geq 0} \mathcal{L}_D(\alpha) = \alpha^T \mathbf{1} - \frac{1}{2} \alpha^T \mathbf{G} \alpha$$

\mathbf{G} is an $N \times N$ p.s.d. matrix, also called the Gram Matrix, $G_{nm} = y_n y_m \mathbf{x}_n^T \mathbf{x}_m$, and $\mathbf{1}$ is a vector of all 1s

Maximizing a concave function (or minimizing a convex function) s.t. $\alpha \geq 0$ and $\sum_{n=1}^N \alpha_n y_n = 0$. Many methods to solve it

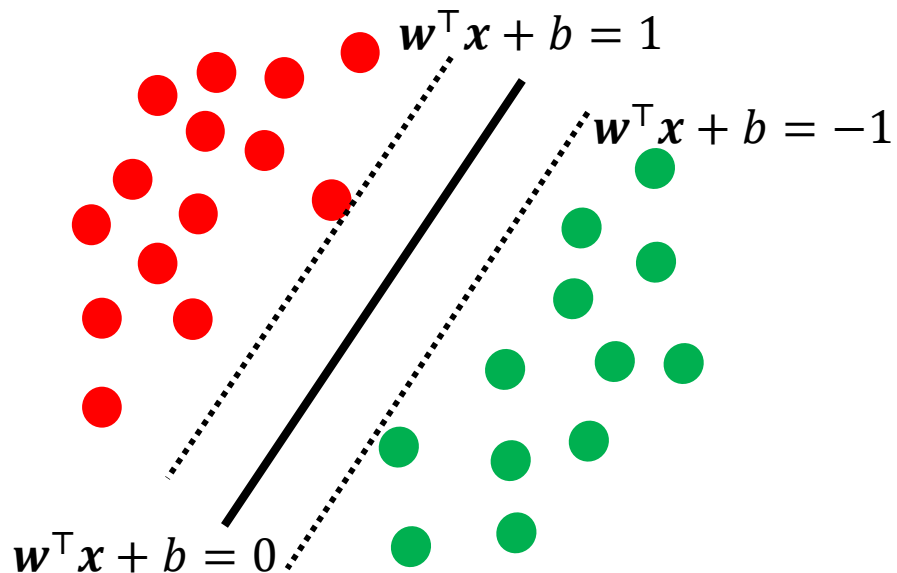
Solving Hard-Margin SVM

- Once we have the α_n' s by solving the dual, we can get \mathbf{w} and b as

$$\mathbf{w} = \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n \quad (\text{we already saw this})$$

$$b = -\frac{1}{2} (\min_{n:y_n=+1} \mathbf{w}^T \mathbf{x}_n + \max_{n:y_n=-1} \mathbf{w}^T \mathbf{x}_n) \quad (\text{exercise})$$

- A nice property: Most α_n' s in the solution will be zero (sparse solution)



- Reason: KKT conditions
- For the optimal α_n' s, we must have
- Thus α_n nonzero only if $y_n(\mathbf{w}^T \mathbf{x}_n + b) = 1$, i.e., the training example lies on the boundary

$$\alpha_n \{1 - y_n(\mathbf{w}^T \mathbf{x}_n + b)\} = 0$$

- These examples are called support vectors

Solving Soft-Margin SVM

- Recall the soft-margin SVM optimization problem

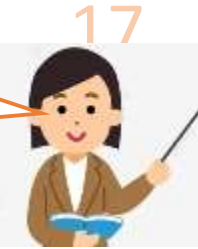
$$\begin{aligned} \min_{\mathbf{w}, b, \boldsymbol{\xi}} \quad & f(\mathbf{w}, b, \boldsymbol{\xi}) = \frac{\|\mathbf{w}\|^2}{2} + C \sum_{n=1}^N \xi_n \\ \text{subject to} \quad & 1 \leq y_n(\mathbf{w}^T \mathbf{x}_n + b) + \xi_n, \quad -\xi_n \leq 0 \quad n = 1, \dots, N \end{aligned}$$

- Here $\boldsymbol{\xi} = [\xi_1, \xi_2, \dots, \xi_N]$ is the vector of **slack variables**
- Introduce Lagrange multipliers α_n, β_n for each constraint and solve

$$\min_{\mathbf{w}, b, \boldsymbol{\xi}} \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta} \geq 0} \mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{\|\mathbf{w}\|^2}{2} + C \sum_{n=1}^N \xi_n + \sum_{n=1}^N \alpha_n \{1 - y_n(\mathbf{w}^T \mathbf{x}_n + b) - \xi_n\} - \sum_{n=1}^N \beta_n \xi_n$$

- The terms in red color above were not present in the hard-margin SVM
- Two set of dual variables $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \dots, \alpha_N]$ and $\boldsymbol{\beta} = [\beta_1, \beta_2, \dots, \beta_N]$
- Will eliminate the primal var $\mathbf{w}, b, \boldsymbol{\xi}$ to get dual problem containing the dual variables

Solving Soft-Margin SVM



- The Lagrangian problem to solve

Note: if we ignore the bias term b then we don't need to handle the constraint $\sum_{n=1}^N \alpha_n y_n = 0$ (problem becomes a bit more easy to solve)

Otherwise, the α_n 's are coupled and some opt. techniques such as co-ordinate aspect can't easily applied

$$\min_{\mathbf{w}, b, \xi} \max_{\alpha \geq 0, \beta \geq 0} \mathcal{L}(\mathbf{w}, b, \xi, \alpha, \beta) = \frac{\|\mathbf{w}\|^2}{2} + C \sum_{n=1}^N \xi_n + \sum_{n=1}^N \alpha_n \{1 - y_n(\mathbf{w}^T \mathbf{x}_n + b) - \xi_n\} - \sum_{n=1}^N \beta_n \xi_n$$

- Take (partial) derivatives of \mathcal{L} w.r.t. \mathbf{w} , b , and ξ_n and setting to zero gives

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n, \quad \frac{\partial \mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{n=1}^N \alpha_n y_n = 0, \quad \frac{\partial \mathcal{L}}{\partial \xi_n} = 0 \Rightarrow C - \alpha_n - \beta_n = 0$$

Weighted sum of training inputs

- Using $C - \alpha_n - \beta_n = 0$ and $\beta_n \geq 0$, we have $\alpha_n \leq C$ (for hard-margin, $\alpha_n \geq 0$)

The dual variables β don't appear in the dual problem!

Given α , \mathbf{w} and b can be found just like the hard-margin SVM case

$$\max_{\alpha \leq C, \beta \geq 0} \mathcal{L}_D(\alpha, \beta) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{m,n=1}^N \alpha_m \alpha_n y_m y_n (\mathbf{x}_m^T \mathbf{x}_n) \quad \text{s.t.} \quad \sum_{n=1}^N \alpha_n y_n = 0$$

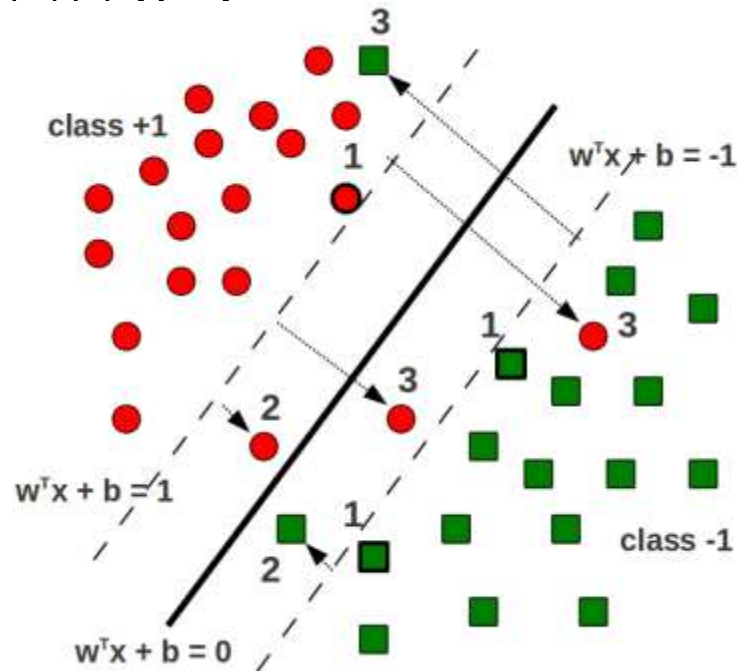
Maximizing a concave function (or minimizing a convex function) s.t. $\alpha \leq C$ and $\sum_{n=1}^N \alpha_n y_n = 0$. Many methods to solve it

$$\max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \alpha^T \mathbf{1} - \frac{1}{2} \alpha^T \mathbf{G} \alpha$$

In the solution, α will still be sparse just like the hard-margin SVM case. Nonzero α_n correspond to the support vectors

Support Vectors in Soft-Margin SVM

- The hard-margin SVM solution had only one type of support vectors
 - All lied on the supporting hyperplanes $\mathbf{w}^T \mathbf{x}_n + b = 1$ and $\mathbf{w}^T \mathbf{x}_n + b = -1$
- The soft-margin SVM solution has three types of support vectors (with non-zero α)



1. Lying on the supporting hyperplanes
2. Lying within the margin region but still on the correct side of the hyperplane
3. Lying on the wrong side of the hyperplane (misclassified training examples)

SVMs via Dual Formulation: Some Comments

- Recall the final dual objectives for hard-margin and soft-margin SVM

Hard-Margin SVM: $\max_{\alpha \geq 0} \mathcal{L}_D(\alpha) = \alpha^\top \mathbf{1} - \frac{1}{2} \alpha^\top \mathbf{G} \alpha$

Soft-Margin SVM: $\max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \alpha^\top \mathbf{1} - \frac{1}{2} \alpha^\top \mathbf{G} \alpha$

Note: Both these ignore the bias term b otherwise will need another constraint $\sum_{n=1}^N \alpha_n y_n = 0$

- The dual formulation is nice due to two primary reasons
 - Allows conveniently handling the margin based constraint (via Lagrangians)
 - Allows learning nonlinear separators by replacing inner products in G_{nm} $= y_n y_m \mathbf{x}_n^\top \mathbf{x}_m$ by general kernel-based similarities (more on this when we talk about kernels)
- However, dual formulation can be expensive if N is large (esp. compared to D)
 - Need to solve for N variables $\alpha = [\alpha_1, \alpha_2, \dots, \alpha_N]$
 - Need to pre-compute and store $N \times N$ gram matrix \mathbf{G}
- Lot of work on speeding up SVM in these settings (e.g., can use co-ord.

Solving for SVM in the Primal

- Maximizing margin subject to constraints led to the soft-margin formulation of SVM

$$\begin{aligned} \arg \min_{\mathbf{w}, b, \xi} \quad & \frac{\|\mathbf{w}\|^2}{2} + C \sum_{n=1}^N \xi_n \\ \text{subject to} \quad & y_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0 \quad n = 1, \dots, N \end{aligned}$$

- Note that slack ξ_n is the same as $\max\{0, 1 - y_n(\mathbf{w}^T \mathbf{x}_n + b)\}$, i.e., hinge loss for (\mathbf{x}_n, y_n)

- Thus the above is $\mathcal{L}(\mathbf{w}, b) = \sum_{n=1}^N \max\{0, 1 - y_n(\mathbf{w}^T \mathbf{x}_n + b)\} + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$ regularized hinge loss

- Sum of slacks is like sum of hinge losses, C and λ play similar roles
- Can learn (\mathbf{w}, b) directly by minimizing $\mathcal{L}(\mathbf{w}, b)$ using (stochastic)(sub)grad

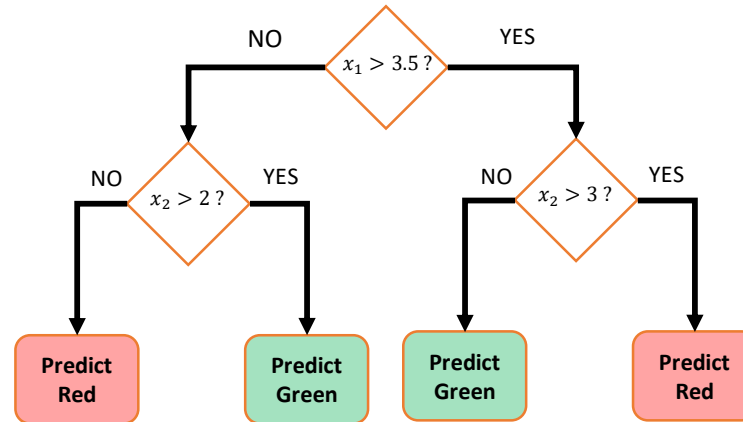
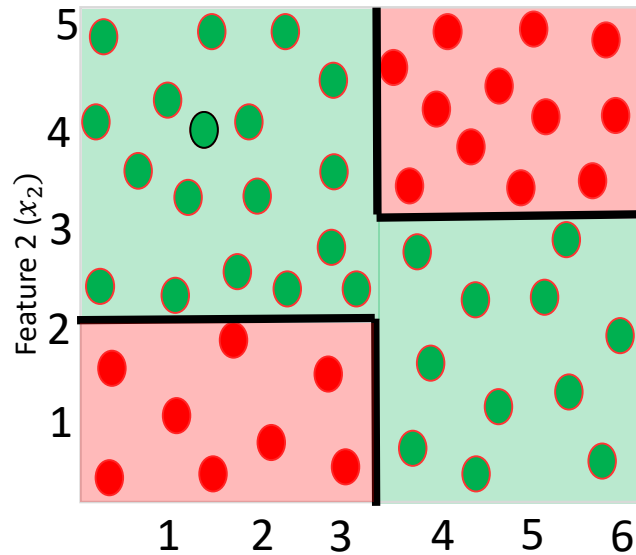
SVM: Summary

- A hugely (perhaps the most!) popular classification algorithm
- Reasonably mature, highly optimized SVM softwares freely available (perhaps the reason why it is more popular than various other competing algorithms)
- Some popular ones: libSVM, LIBLINEAR, sklearn also provides SVM
- Lots of work on scaling up SVMs* (both large N and large D)
- Extensions beyond binary classification (e.g., multiclass, structured outputs)
- Can even be used for regression problems (Support Vector Regression)
- Nonlinear extensions possible via kernels

* See: "Support Vector Machine Solvers" by Bottou and Lin

Decision Trees

Constructing Decision Trees



The rules are organized in the DT such that **most informative rules are tested first**

Informativeness of a rule is related to the extent of the purity of the split arising due to that rule. **More informative rules yield more pure splits**

Hmm.. So DTs are like the "20 questions" game (ask the **most useful questions first**)



Given some training data, what's the "optimal" DT?

How to decide which rules to test for and in what order?

How to assess informativeness of a rule?

In general, constructing DT is an intractable problem (NP-hard)

Often we can use some "greedy" heuristics to construct a "good" DT

To do so, we use the training data to figure out which rules should be tested at each node

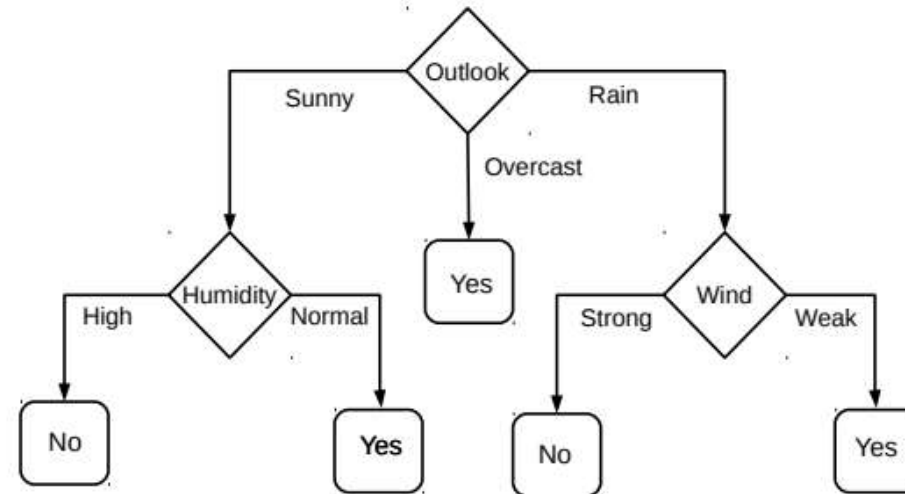
The same rules will be applied on the test inputs to route them along the tree until they reach some leaf node where the prediction is made



Decision Tree Construction: An Example

- Let's consider the playing Tennis example
- Assume each internal node will test the value of one of the features

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



- Question: Why does it make more sense to test the feature "outlook" first?
- Answer: Of all the 4 features, it's the most informative
 - It has the highest **information gain** as the root node

Entropy and Information Gain

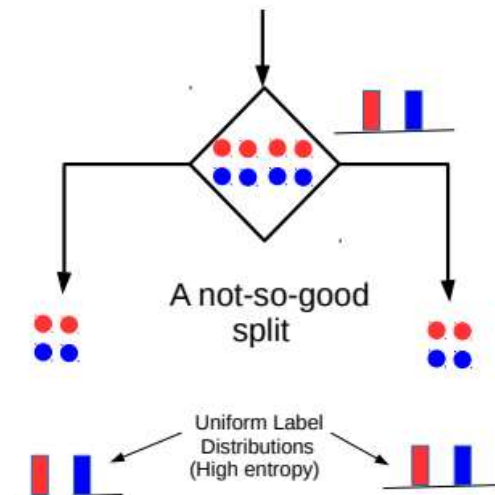
- Assume a set of labelled inputs \mathbf{S} from \mathcal{C} classes, p_c as fraction of class c inputs
- Entropy of the set \mathbf{S} is defined as $H(\mathbf{S}) = -\sum_{c \in \mathcal{C}} p_c \log p_c$
- Suppose a rule splits \mathbf{S} into two smaller disjoint sets \mathbf{S}_1 and \mathbf{S}_2
- Reduction in entropy after the split is called information gain

Uniform sets (all classes roughly equally present) have high entropy;
skewed sets low

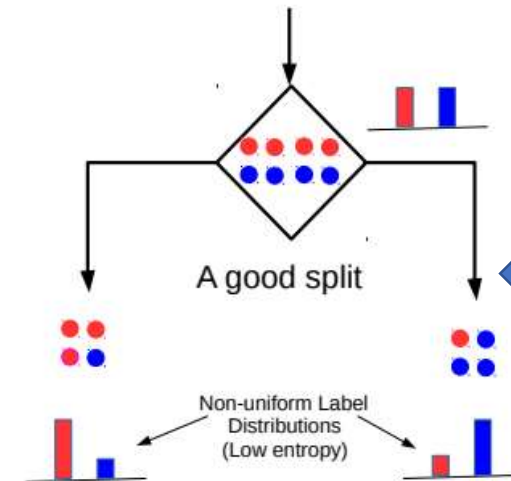


$$IG = H(\mathbf{S}) - \frac{|\mathbf{S}_1|}{|\mathbf{S}|} H(\mathbf{S}_1) - \frac{|\mathbf{S}_2|}{|\mathbf{S}|} H(\mathbf{S}_2)$$

This split has a low IG
(in fact zero IG)



VS



This split has higher IG

Entropy and Information Gain

- Let's use IG based criterion to construct a DT for the Tennis example
- At root node, let's compute IG of each of the 4 features
- Consider feature "wind". Root contains all examples $S =$

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no

$$H(S) = -(9/14) \log_2(9/14) - (5/14) \log_2(5/14) = 0.94$$

$$S_{\text{weak}} = [6+, 2-] \Rightarrow H(S_{\text{weak}}) = 0.811$$

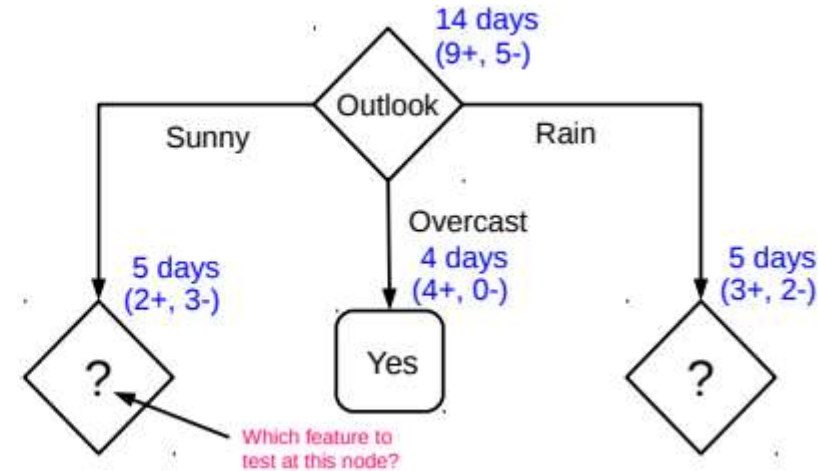
$$S_{\text{strong}} = [3+, 3-] \Rightarrow H(S_{\text{strong}}) = 1$$

$$IG(S, \text{wind}) = H(S) - \frac{|S_{\text{weak}}|}{|S|} H(S_{\text{weak}}) - \frac{|S_{\text{strong}}|}{|S|} H(S_{\text{strong}}) = 0.94 - 8/14 * 0.811 - 6/14 * 1 = 0.048$$

- Likewise, at root: $IG(S, \text{outlook}) = 0.246$, $IG(S, \text{humidity}) = 0.151$, $IG(S, \text{temp}) = 0.029$
- Thus we choose "outlook" feature to be tested at the root node
- Now how to grow the DT, i.e., what to do at the next level? Which feature to test next?

Growing the tree

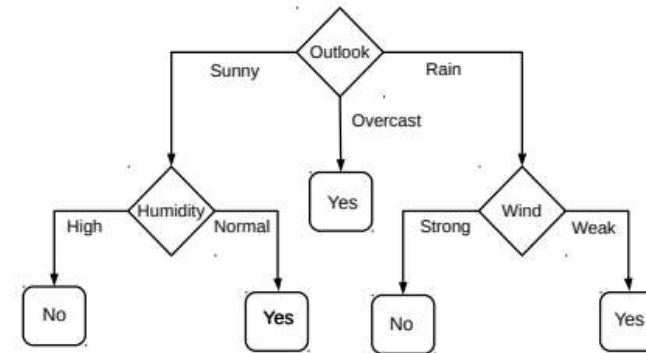
day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



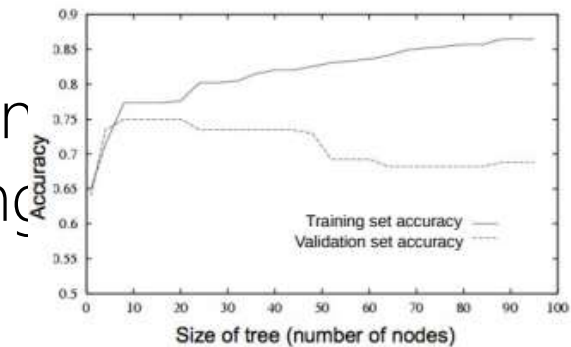
- Proceeding as before, for level 2, left node, we can verify that
 - $IG(S, temp) = 0.570$, $IG(S, humidity) = 0.970$, $IG(S, wind) = 0.019$
- Thus humidity chosen as the feature to be tested at level 2, left node
- No need to expand the middle node (already “pure” - all “yes” training examples 😊)
- Can also verify that wind has the largest IG for the right node
- Note: If a feature has already been tested along a path earlier, we don't consider



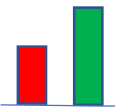

When to stop growing the tree?

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



- Stop expanding a node further (i.e., make it a leaf node) when
 - It consist of all training examples having the same label (the node becomes "pure")
 - We run out of features to test along the path to that r
 - The DT starts to overfit (can b To help prevent the tree from growing too much! oring, the validation set accuracy)



- Important:** No need to obsess too much for purity
 - It is okay to have a leaf node that is not fully pure, e.g., this    OR 
 - At test inputs that reach an impure leaf, can predict probability of belonging to each class (in above example, $p(\text{red}) = 3/8$, $p(\text{green}) = 5/8$), or simply predict

Avoiding Overfitting in DTs

- Desired: a DT that is not too big in size, yet fits the training data reasonably
- Note: An example of a very simple DT is “decision-stump”
 - A decision-stump only tests the value of a single feature (or a simple rule)
 - Not very powerful in itself but often used in large ensembles of decision stumps
- Mainly two approaches to prune a complex DT
 - Prune while building the tree (stopping early)
 - Prune after building the tree (post-pruning)
- Criteria for judging which nodes could potentially be pruned
 - Use a validation set (separate from the training set)
 - Prune each possible node that doesn't hurt the accuracy on the validation set
 - Greedily remove the node that improves the validation accuracy the most
 - Stop when the validation set accuracy starts worsening
 - Use model complexity control, such as Minimum Description Length (will see

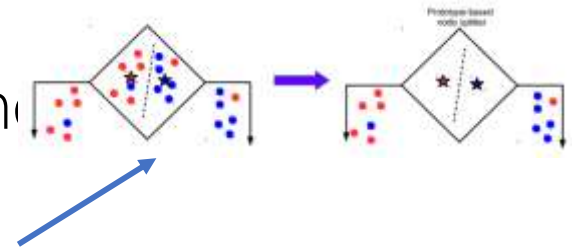
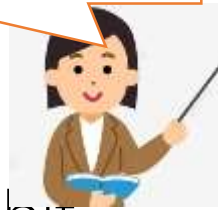
Either can be done
using a validation
set

Decision Trees: Some Comments

- **Gini-index** defined as $\sum_{c=1}^C p_c(1 - p_c)$ can be an alternative
- For DT regression¹, variance in the outputs can be used to assess purity
- When **features are real-valued** (no finite possible values to try), things are a bit more tricky
 - Can use tests based on **thresholding** feature values (recall our synthetic data examples)
 - Need to be careful w.r.t. number of threshold points, how fine etc.
- More sophisticated decision rules at the internal nodes can also be used
 - Basically, need some rule that splits inputs at an internal node into homogeneous groups
 - The rule can even be a machine learning classification algo (e.g., LwP or a deep learner)

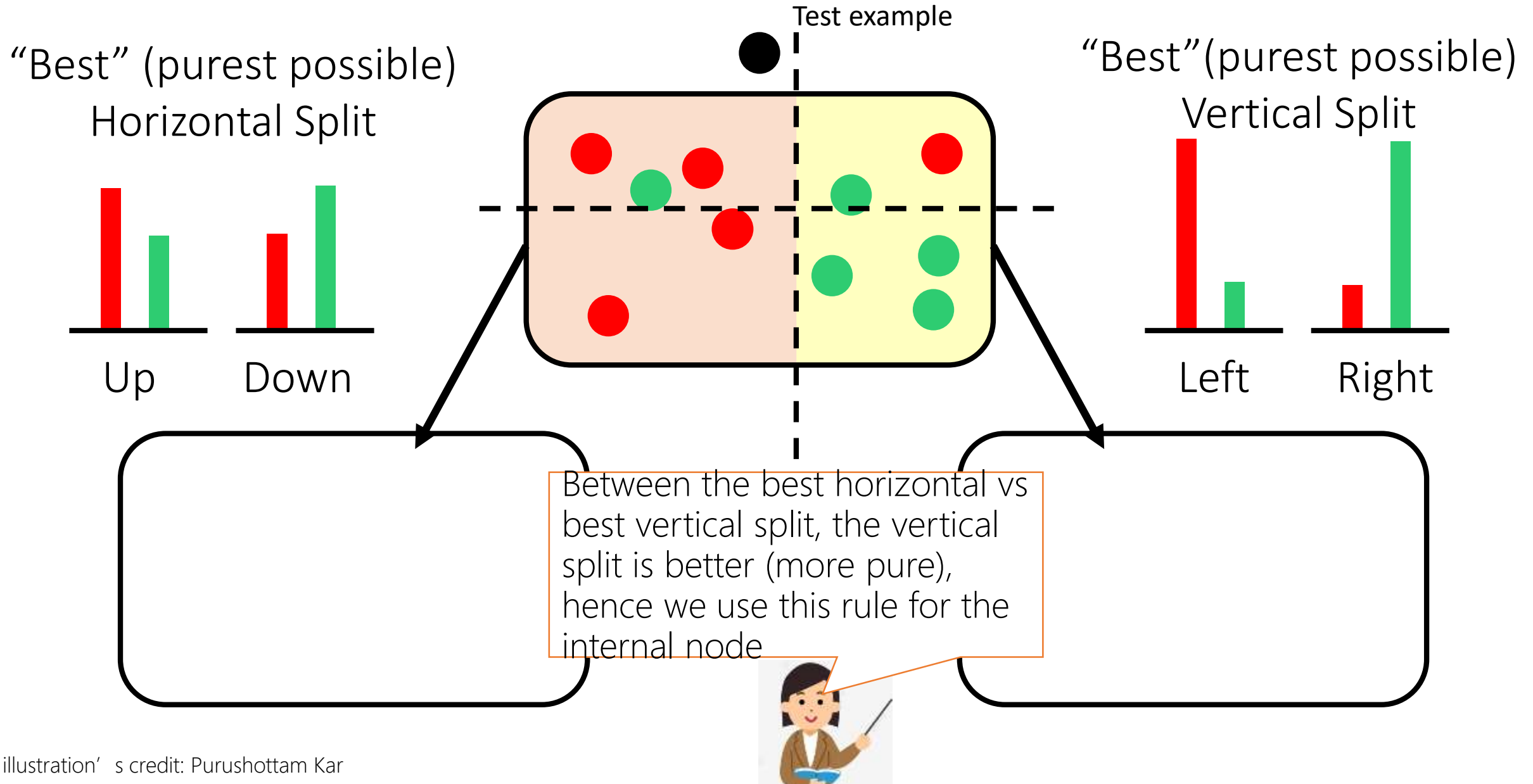
For regression, outputs are real-valued and we don't have a "set" of classes, so quantities like entropy/IG/gini etc. are undefined

An illustration on the next slide



¹Breiman, Leo; Friedman, J. H.; Olshen, R. A.; Stone, C. J. (1984). Classification and regression trees

An Illustration: DT with Real-Valued Features

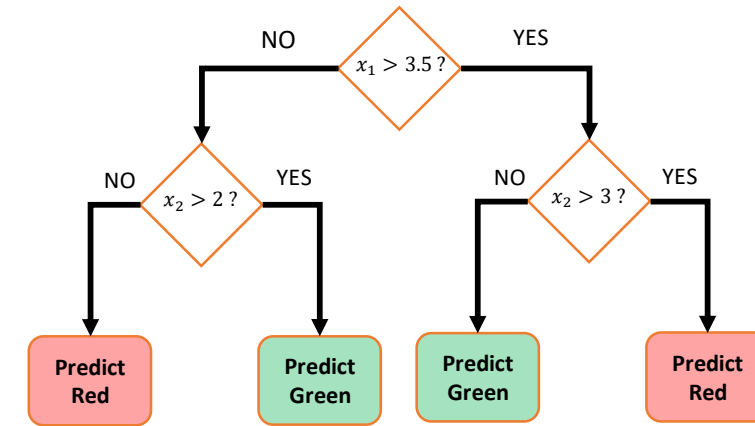
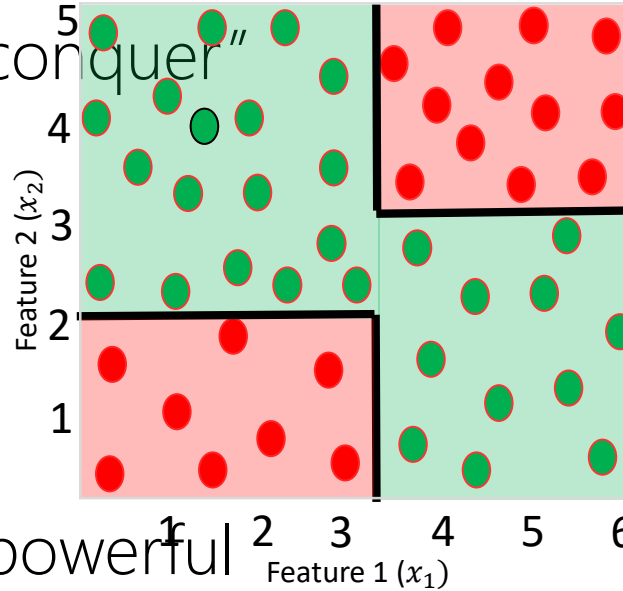


Decision Trees: A Summary

Some key strengths:

- Simple and easy to interpret
- Nice example of “divide and conquer” paradigm in machine learning
- Easily handle different types of features (real, categorical, etc.)
- Very fast at test time
- Multiple DTs can be combined via **ensemble methods**: more powerful (e.g., Decision Forests; will see later)
- Used in several real-world ML applications, e.g., recommender systems, gaming (Kinect)

.. thus helping us learn complex rule as a combination of several simpler rules



Human-body pose estimation

Some key weaknesses:

- Learning optimal DT is (NP-hard) intractable. Existing algos mostly greedy heuristics