10-701 Midterm Review

Mar.19, 2018

Model-based Approach

Data: $X_1, X_2, \ldots, X_n$.

Model: $P(X;\theta)$ with parameters θ.

Assumption: Data drawn $i.i.d$ from distribution $P(X;\theta^*)$ for some unknown θ^*.

Mission (should you choose to accept it): recover θ^* from data $X_1, X_2, \ldots, X_n$.

- Estimate Model Parameter
  - Different ways: MLE/MAP
- Use model in production

Maximum Likelihood Estimation

Choose θ that maximizes the probability of observed data

$$\widehat{\theta}_{MLE} = \arg\max_{\theta} P(D \mid \theta)$$

$$\begin{aligned}\widehat{\theta}_{MLE} &= \underset{\theta}{\operatorname{argmax}}\, P(X_1, X_2, \ldots, X_n; \theta) \\ &= \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{n} P(X_i; \theta) \\ &= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \log(P(X_i; \theta))\end{aligned}$$

$\longrightarrow$ Take first derivative
Put to zero

How good is MLE?

- Asymptotically, unbiased
- Consistent: under some constraints
- Finite Sample:
  - Sample Question: flipped a coin once and observed a head.
    - What is MLE for p, where p is probability of heads showing.
    - Using MLE, give an interval such that there's a 95% chance of p lying in it.

MAP

Choose \theta whose probability given data is highest,
ie maximize posterior distribution

$$\widehat{\theta}_{MAP} = \underset{\theta}{\operatorname{argmax}}\, P(\theta | X_1, X_2, \ldots, X_n)$$

$$P(\theta \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \theta) P(\theta)}{P(\mathcal{D})}$$

$$\underbrace{P(\theta \mid \mathcal{D})}_{\text{posterior}} \propto \underbrace{P(\mathcal{D} \mid \theta)}_{\text{likelihood}} \underbrace{P(\theta)}_{\text{prior}}$$

$$\begin{aligned}\widehat{\theta}_{MAP} &= \underset{\theta}{\operatorname{argmax}} \left(\prod_{i=1}^{n} P(X_i; \theta) \right) P(\theta) \\ &= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \log(P(X_i; \theta)) + \log(P(\theta))\end{aligned}$$

Regularizer

Questions?

- Practice calculating MLE's and MAPs
  - Coin flips (prior: Beta)
  - Dice Flips (prior: Dirichlet)
  - Linear regression (prior: gaussian, doubly-exponential)
  - …..

Generalization via Risk Minimization

Risk Minimization

True Risk

Target performance measure

$$R(f) := \mathbb{E}(\ell(f(X), Y))$$

Classification

Probability of misclassification

$$P(f(X) \neq Y)$$

Empirical Risk

Performance on training data

$$\widehat{R}_D(f) := \frac{1}{|D|} \sum_{i \in D} \ell(f(X_i), Y_i)$$

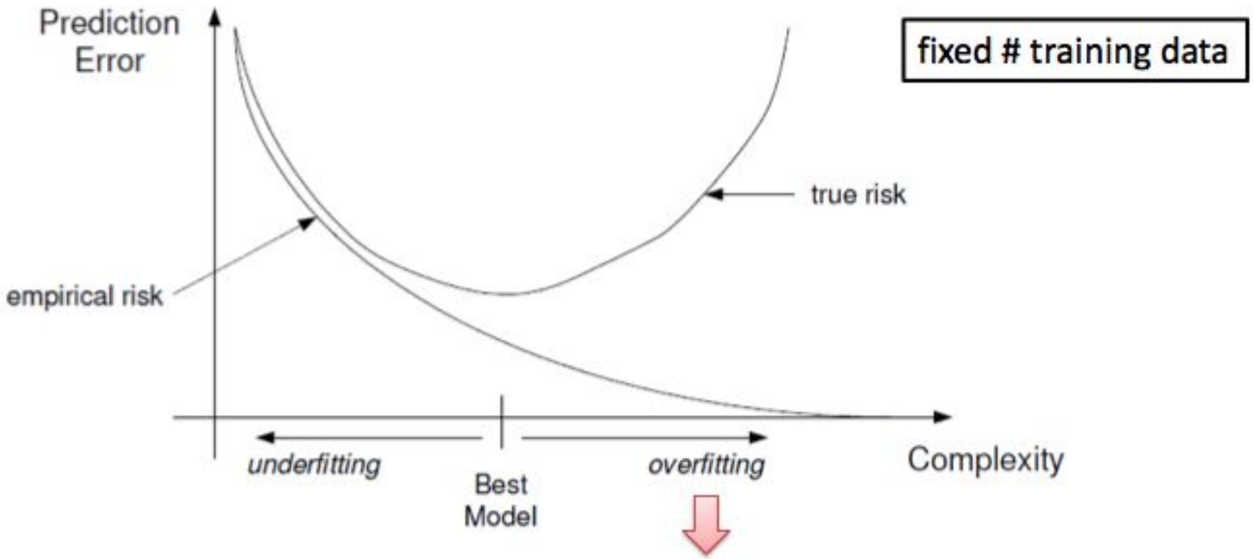
Classification

Proportion of misclassified examples

$$\frac{1}{n} \sum_{i=1}^{n} 1_{f(X_i) \neq Y_i}$$

Overfitting: Effect of discrepancy between empirical and true risks

If we allow very complicated predictors, we could overfit the training data.



Empirical risk is no longer a good indicator of true risk

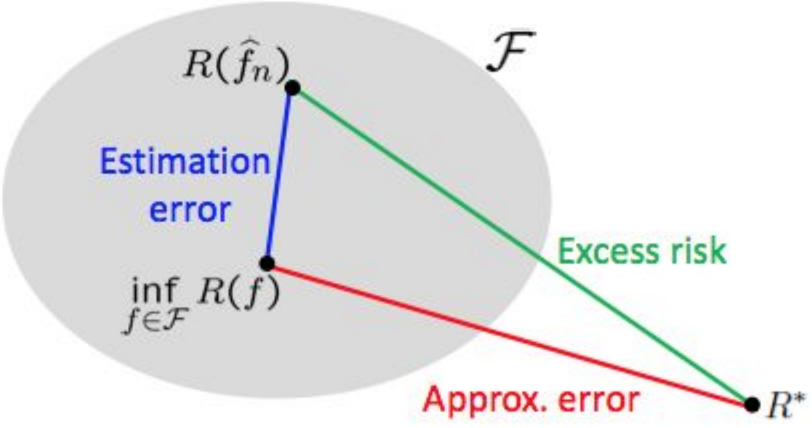
Behavior of True Risk

Want $\widehat{f}_n$ to be as good as optimal predictor f^*

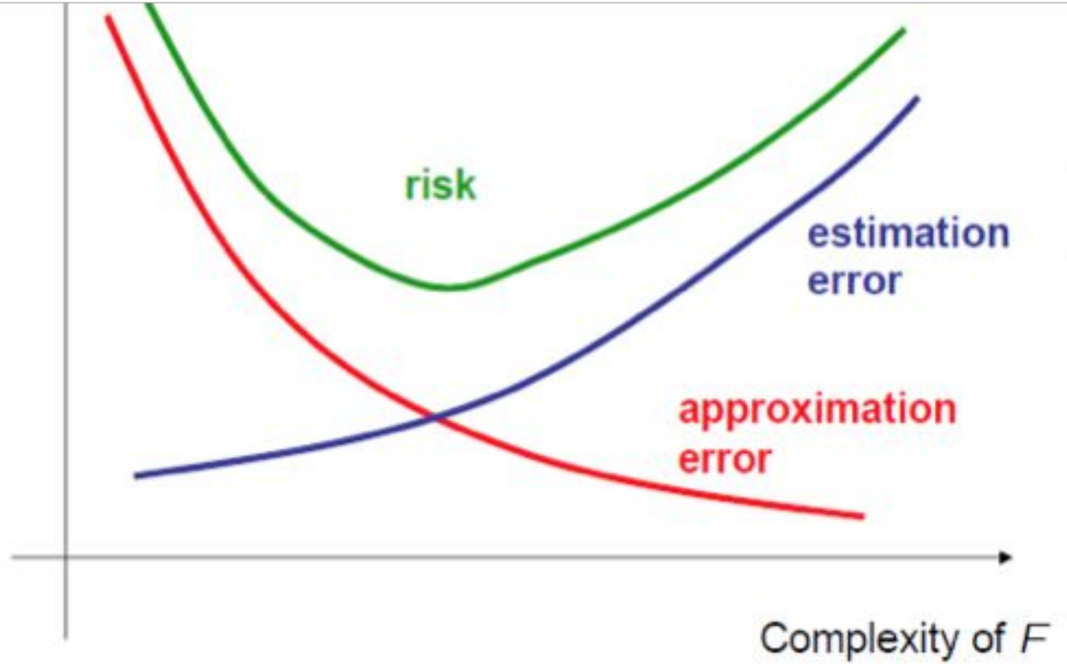
Excess Risk $$E\left[R(\widehat{f}_n)\right] - R^* = \underbrace{\left(E[R(\widehat{f}_n)] - \inf_{f\in\mathcal{F}} R(f)\right)}_{\text{estimation error}} + \underbrace{\left(\inf_{f\in\mathcal{F}} R(f) - R^*\right)}_{\text{approximation error}}$$

finite sample size + noise ← Due to randomness of training data

Due to restriction of model class



Behavior of True Risk



fixed # training data

Questions?

Neural Networks

Consider a prediction problem in which we want to predict some **outputs y** from some **inputs x**.

The **goal** is to learn a **function f** such that

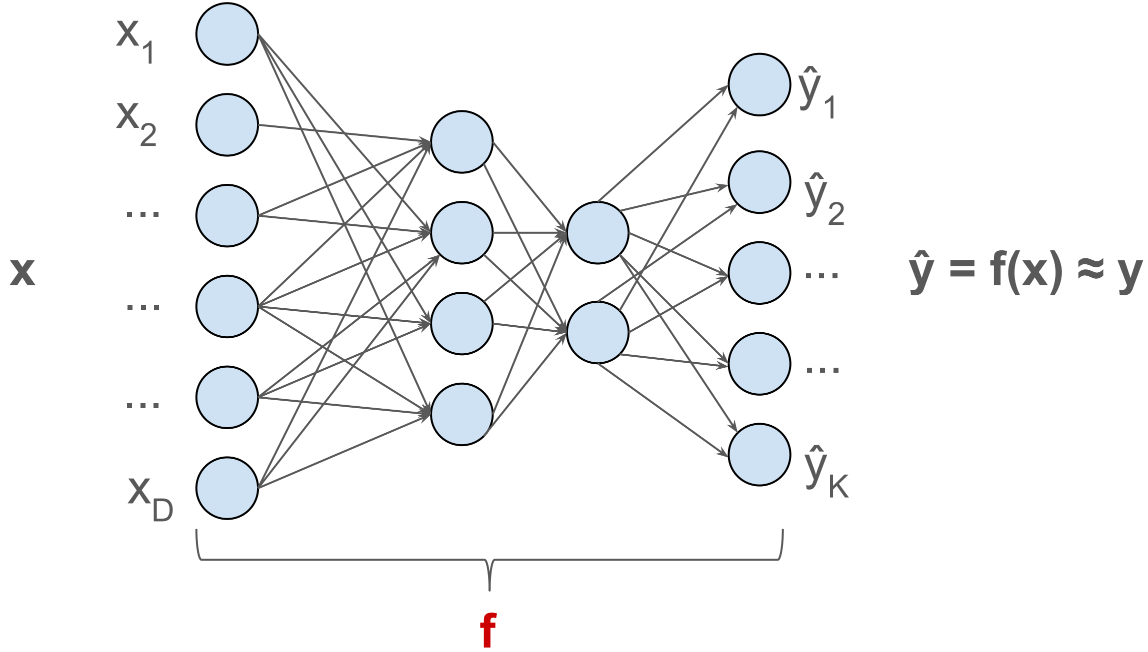
f(x) = y

Examples of such functions learnt in class:

- Linear regression: $f(x) = w^T x + b$
- Logistic regression: $f(x) = P(y|x) = \dfrac{\exp(w^T x + b)}{1 + \exp(w^T x + b)}$

Neural Networks

A neural network is also such a function **f**, whose parameters are the weights of the network. For example:



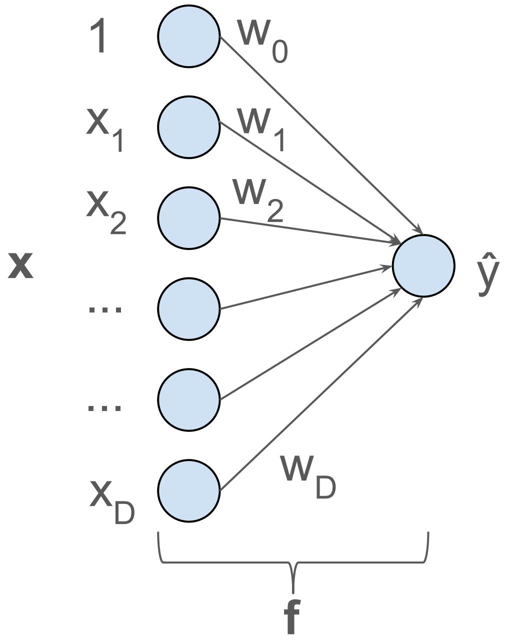
Neural Networks: Artificial neuron

Neuron pre-activation:

$$g(x) = \underbrace{w_0}_{\text{bias}} + \sum_i w_i x_i = w_0 + \underbrace{w^T}_{\text{weights}} x$$

Neuron post-activation:

$$\hat{y} = \underbrace{a}_{\text{activation function}}(g(x)) = a(w_0 + w^T x)$$



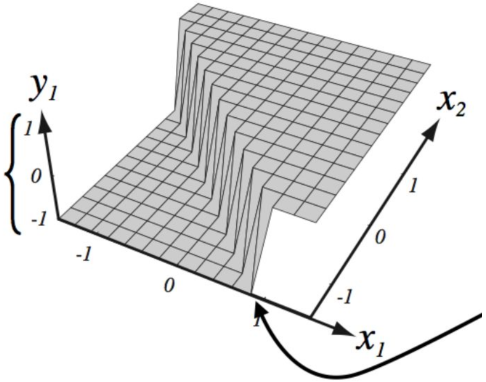
Neural Networks: Artificial neuron

Neuron post-activation:

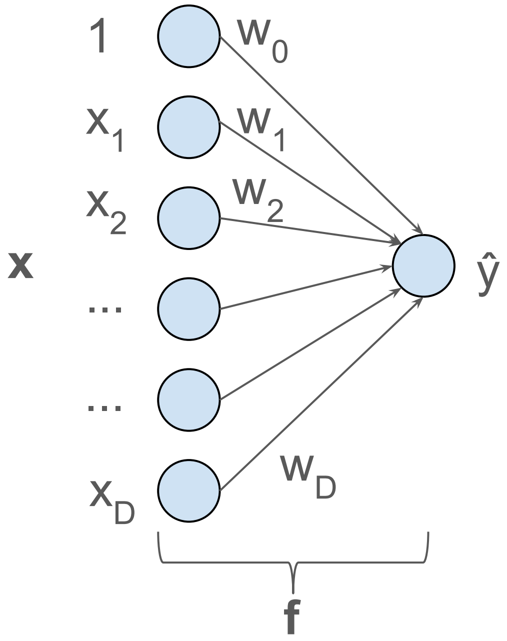
$$\hat{y} = a(g(x)) = a(w_0 + w^T x)$$

activation function

The range of y is determined by the function a(.)

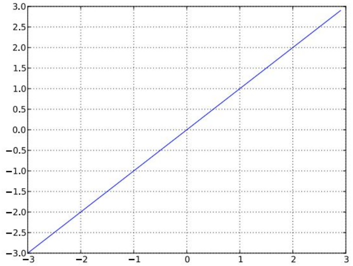


The bias changes only the position

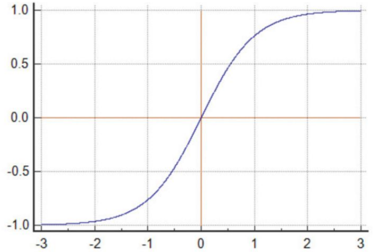


Neural Networks: Activation functions

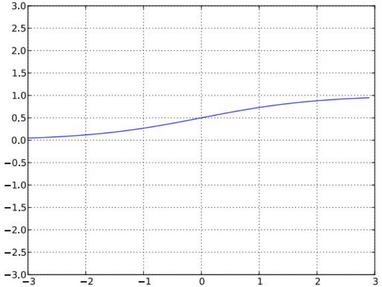
- **Linear:** $a(x) = c \cdot x + b$



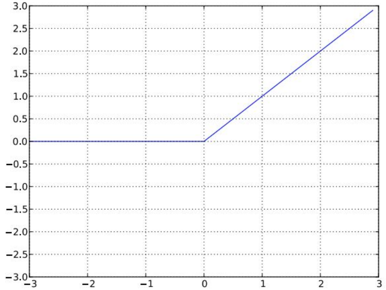
- **Tanh:** $a(x) = \dfrac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$



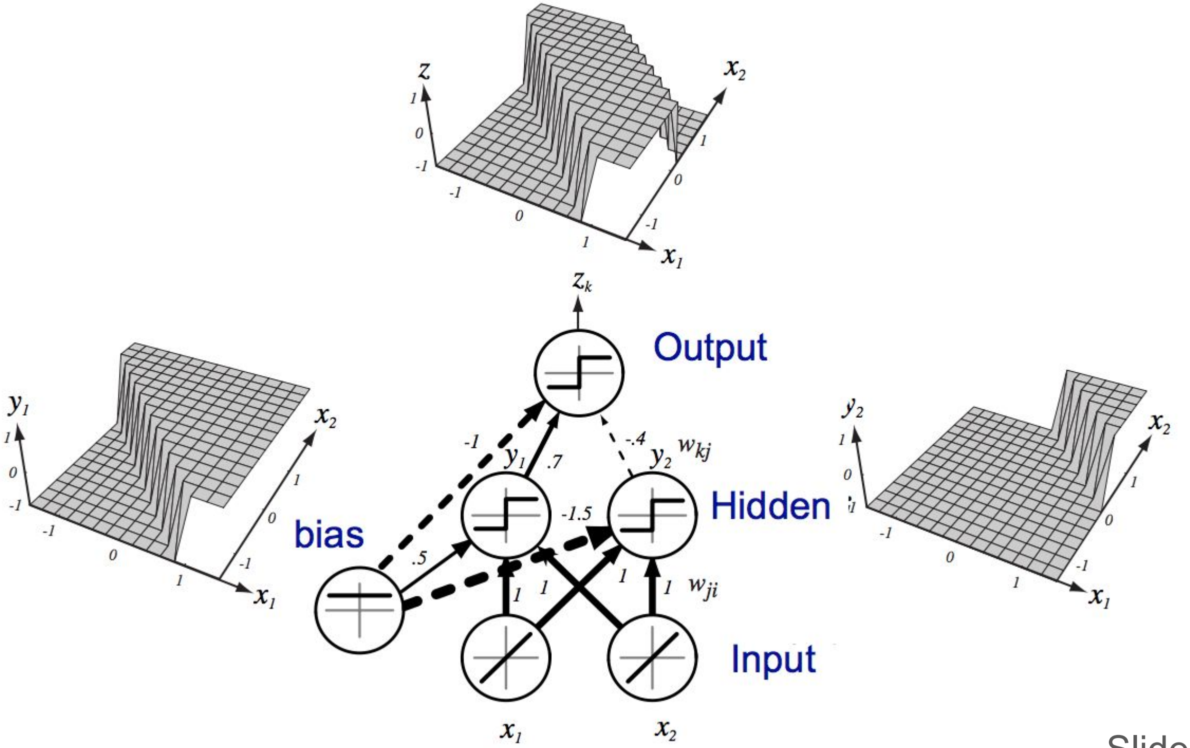
- **Sigmoid:** $a(x) = \dfrac{1}{1 + \exp(-x)}$



- **ReLU:** $a(x) = \max(0, x)$

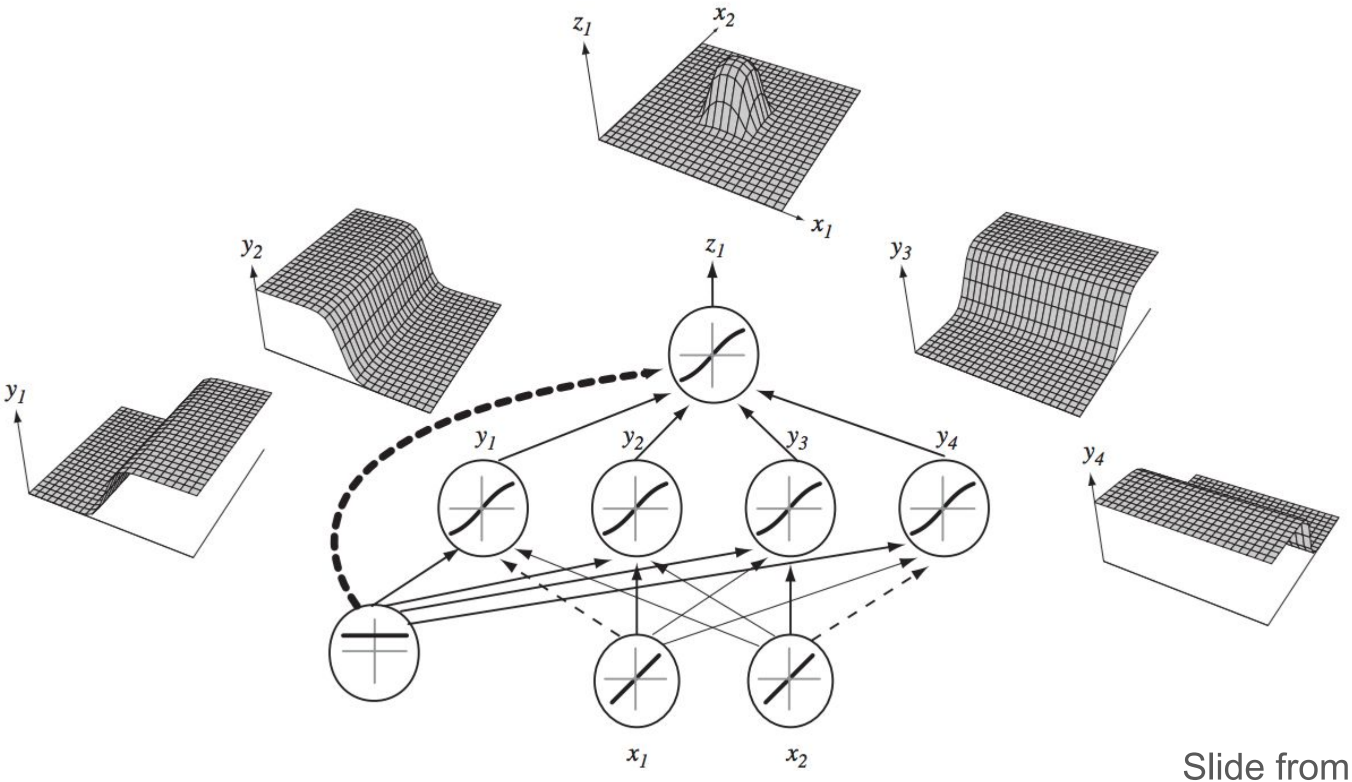


Neural Networks: Multiple layers



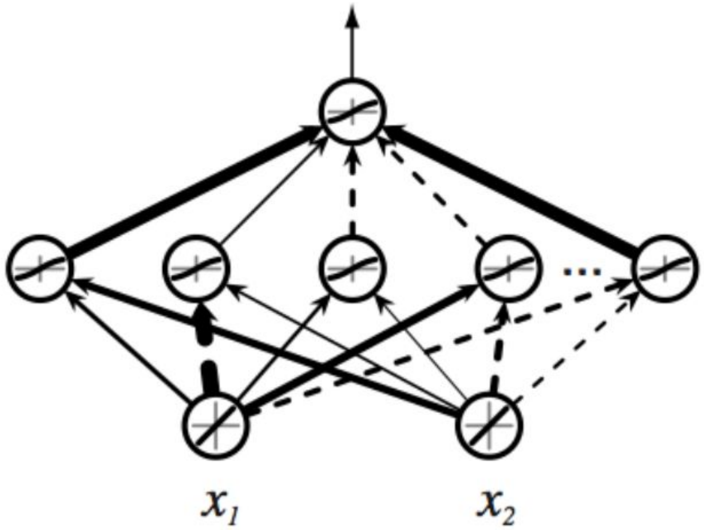
Slide from Pascal Vincent

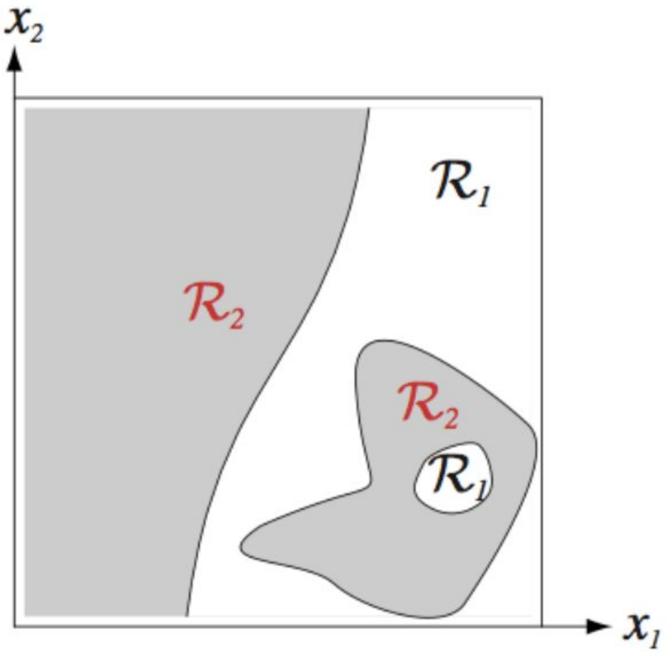
Neural Networks: Multiple layers



Slide from Pascal Vincent

Neural Networks: Multiple layers





Slide from Pascal Vincent

Neural Networks: Universal approximation

Universal Approximation Theorem (Hornik, 1991):

A single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units.

Neural Networks: Universal approximation

Universal Approximation Theorem (Hornik, 1991):

A single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units.

This does not mean that there is learning algorithm that can find the right parameter values!

Neural Networks: Training

- Learning is posed as an optimization problem.
- The goal is to minimize some loss function:

$$\min_{\theta=\{W,b\}} \ell(\underbrace{\hat{y}_{\text{pred}}}_{=f(x)}, y)$$

- One approach we learnt is to compute the gradients of the loss wrt. model parameters, and use them to update the parameters accordingly.

- Gradient descent: compute the gradient given N samples and update the parameters:

$$\theta^{+} \leftarrow \theta - \alpha \nabla_{\theta} \sum_{i=1}^{N} \ell(\hat{y}(x^{(i)}), y^{(i)})$$

Neural Networks: Training

- How do we compute the gradients?
  Backpropagation!

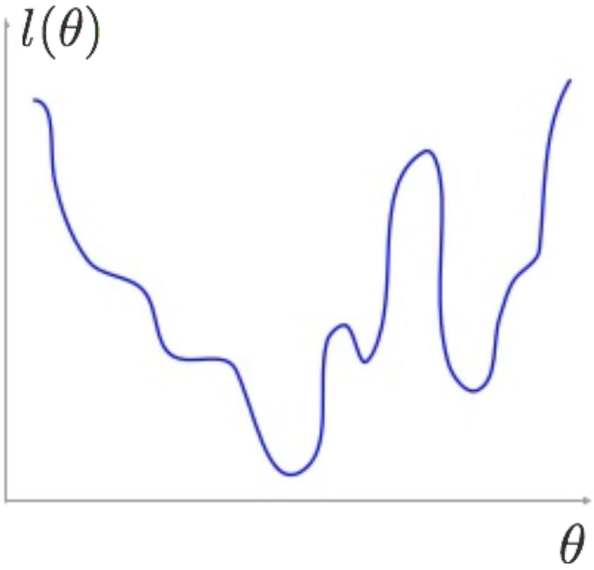
Useful resource:
http://colah.github.io/posts/2015-08-Backprop/

- Neural network training involves two phases:

1. **Forward pass**:
   Compute the function value f(x) given some input x, and our current estimate of the network parameters.

2. **Backward pass (backpropagation)**:
   Compute all gradients $\nabla_\theta \ell$ with just one pass and update the parameters.

Neural Networks: Why we can't always find the optimal parameters



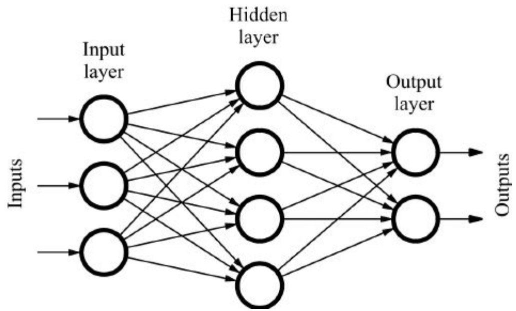
The loss function is usually not convex.

→ gradient descent will converge to a local optimum!

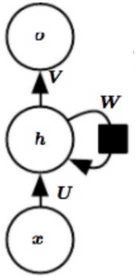
But that's ok!

Neural Networks: Examples

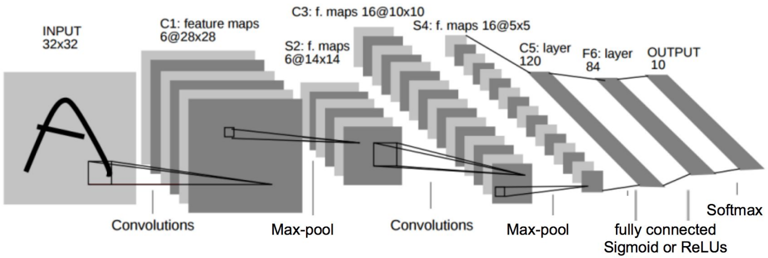
Fully-connected



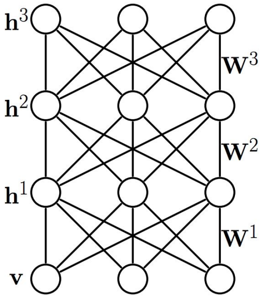
Recurrent



Convolutional



Deep Boltzmann Machine



Neural Networks

Questions?

Linear Regression

- Assume $Y = f(X) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$, and f does not need to be a linear function.

- Linear regression finds the best linear approximator:

$$\beta^* = \text{argmin}_\beta \, \mathbb{E}[(Y - X\beta)^2]$$ ← Ideally

$$\hat{\beta} = \text{argmin}_\beta \, \frac{1}{n} \sum_{i=1}^n (Y_i - X_i \beta)^2$$ ← Empirically

where $X \in \mathbb{R}^{n \times p}, Y \in \mathbb{R}^n, \beta \in \mathbb{R}^p$.

- A polynomial regression "is" a linear regression. Instead, one would want to replace X with a polynomial basis (e.g. $\phi(X) = (X_1, X_2, X_1 X_2, X_1^2, \ldots)$)

Linear Regression: Solving

- Fortunately, this is a convex optimization problem. Define

$$J(\beta) = (Y - X\beta)^T (Y - X\beta) = \sum_{i=1}^{n} (Y_i - X_i\beta)^2$$

  Then the solution is given by $\nabla_\beta J(\beta) = \frac{\partial J}{\partial \beta} = 0.$

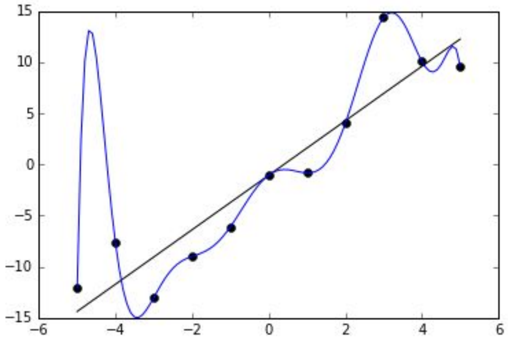
- Usually assume $X \in \mathbb{R}^{n \times p}, n > p$ (i.e. we have enough data). Why?

- $\hat{\beta} = (X^T X)^{-1} X^T Y$

- What if $X^T X$ is not invertible? We can constrain our answers by regularization.

Linear Regression: Overfitting & Regularization

- Let's say we are doing a polynomial regression



- One way to reduce overfitting is to regularize the model. In this case, in linear regression, we typically do this by adding sparsity constraints to β.

- How do we measure sparsity?

Linear Regression: Regularization

- A good measure to constrain how "large" β is is through L_p norm:

$$L_0 : \|\beta\|_0 = \sum_{i=1}^{p} 1_{\beta_i \neq 0}$$

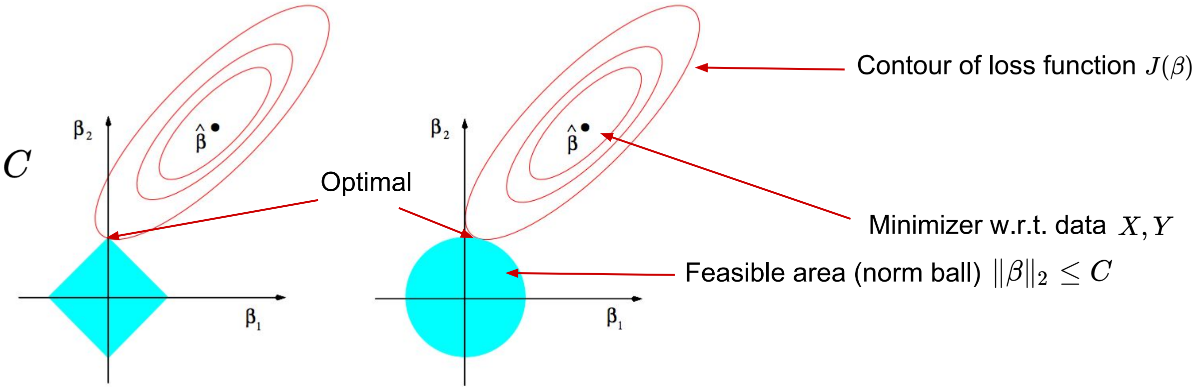
$$L_1 : \|\beta\|_1 = \sum_{i=1}^{p} |\beta_i|$$

$$L_2 : \|\beta\|_2 = \sqrt{\sum_{i=1}^{p} \beta_i^2}$$

- Typically one would use L0, L1 (lasso), or L2 (ridge) regularizations.

$$\min_\beta \; J(\beta) + \lambda \|\beta\|_p$$

$$\iff \min_\beta \; J(\beta) \text{ s.t. } \|\beta\|_p \leq C$$



Linear Regression: Summary & Questions

k Nearest Neighbors

- High-level idea: predict at test time by looking at the k nearest training data points. More formally: $P(Y=c|X=x)=\frac{1}{k}\sum_{i:X_i\in\mathcal{N}_k(x)}1_{Y_i=c}$.
- kNN **can** be used for regression as well: $\hat{f}(x)=\frac{1}{k}\sum_{i:X_i\in\mathcal{N}_k(x)}Y_i$.
- Efficient training: we only need to memorize all training data.
- Expensive evaluation: need to compute distance w.r.t. every training data point :-(
- **(T or F)** kNN algorithm works for **any** valid distance function/metric.
  - $d(x,y)\geq 0,\ d(x,y)=0 \text{ iff } x=y, d(x,y)=d(y,x),\ d(x,z)\leq d(x,y)+d(y,z)$

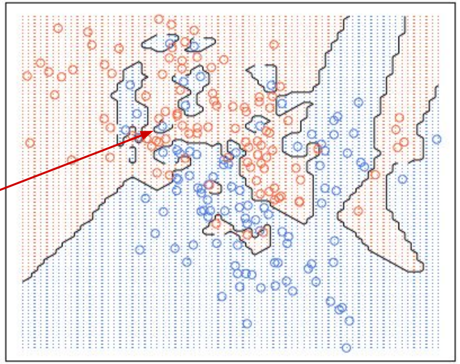
Non-negativity Identity of indiscernibles Symmetry Triangular inequality

k Nearest Neighbors: Overfitting

- kNN can overfit. One way to correct this is to pick a better k using cross validation.

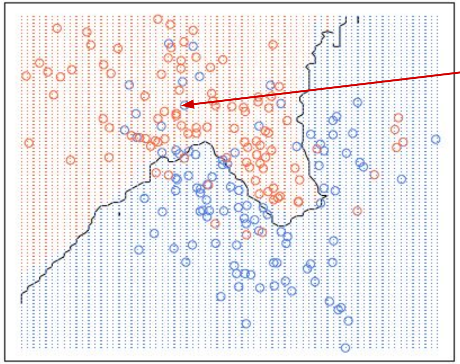
Likely to overfit the training data

nearest neighbour (k = 1)



Low bias, high variance

20-nearest neighbour



Lots of misclassifications

High bias, low variance

Figures from Kevin Zakka's blog

k Nearest Neighbors: Properties & Summary

- **Non-parametric**: kNN does not assume the function form of the decision boundary. Helps us avoid the danger of mis-modeling the distribution.

- **Discriminative**: kNN models $P(Y|X)$, instead of $P(X,Y)$.

- **Instance-based**: kNN does not learn an explicit model. Instead, it classifies (or make prediction) based on training instances.

- **Curse of dimensionality**: kNN is bad at dealing with high dimensional data (where distance can be easily affected).

Naive Bayes

- Bayes Classifier with additional "naïve" assumption:
  - Features are independent given class:

$$X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$$

$$P(X_1, X_2|Y) = P(X_1|X_2, Y)P(X_2|Y)$$
$$= P(X_1|Y)P(X_2|Y)$$

  - More generally:

$$P(X_1 ... X_d|Y) = \prod_{i=1}^{d} P(X_i|Y)$$

$$X = \begin{bmatrix} X_1 \\ X_2 \\ \dots \\ X_d \end{bmatrix}$$

- If conditional independence assumption holds, NB is optimal classifier! But worse otherwise.

Naive Bayes

- Naive Bayes is a generative model with a prior on each class:

$$P(X, Y) = P(Y) \cdot \prod_{i=1}^{n} P(X_i|Y)$$

- At training, we need to find all parameters (see above). Typically found them via MLE estimate. For instance,

$$P_{MLE}(Y = c) = \frac{1}{n} \sum_{i=1}^{n} 1_{Y_i = c}$$

- At evaluation, we apply the Naive Bayes assumption on the argmax:

$$\hat{y} = \operatorname{argmax}_y P(Y = y|X) = \operatorname{argmax}_y P(X|Y = y) \cdot P(Y = y)$$

Logistic Regression

Assumes the following functional form for P(Y|X):

$$P(Y=0|X) = \frac{1}{1+\exp(w_0 + \sum_i w_i X_i)}$$

$$\Rightarrow P(Y=1|X) = \frac{\exp(w_0 + \sum_i w_i X_i)}{1+\exp(w_0 + \sum_i w_i X_i)}$$

$$\Rightarrow \frac{P(Y=1|X)}{P(Y=0|X)} = \exp(w_0 + \sum_i w_i X_i) \underset{0}{\overset{1}{\gtrless}} 1$$

$$\Rightarrow \boxed{w_0 + \sum_i w_i X_i \underset{0}{\overset{1}{\gtrless}} 0}$$

If >0, then we predict 1; otherwise we predict 0

Logistic Regression

Assumes the following functional form for P(Y|X):

$$P(Y=0|X) = \frac{1}{1+\exp(w_0 + \sum_i w_i X_i)}$$

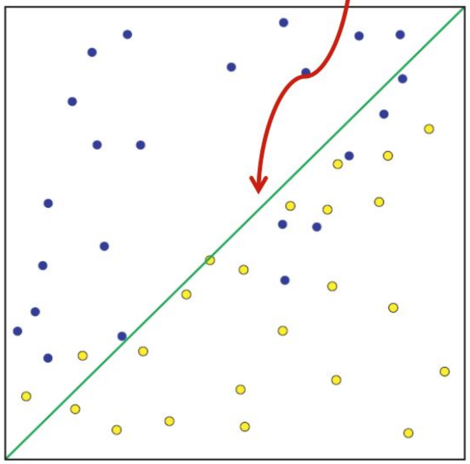
$$w_0 + \sum_i w_i X_i = 0$$

Decision boundary: Note - Labels are 0,1

$$P(Y=0|X) \underset{1}{\overset{0}{\gtrless}} P(Y=1|X)$$

$$w_0 + \sum_i w_i X_i \underset{0}{\overset{1}{\gtrless}} 0$$

(Linear Decision Boundary)



Logistic Regression

$$P(Y=0|\mathbf{X},\mathbf{w}) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

$$P(Y=1|\mathbf{X},\mathbf{w}) = \frac{exp(w_0+\sum_i w_i X_i)}{1+exp(w_0+\sum_i w_i X_i)}$$

$$\begin{aligned} l(\mathbf{w}) &\equiv \ln \prod_j P(y^j|\mathbf{x}^j,\mathbf{w}) \\ &= \sum_j \left[y^j(w_0+\sum_i^d w_i x_i^j) - \ln(1+exp(w_0+\sum_i^d w_i x_i^j)) \right] \end{aligned}$$

Bad news: no closed-form solution to maximize *l*(**w**)

Good news: *l*(**w**) is concave function of **w**

concave functions easy to maximize

Logistic Regression

$$p(\mathbf{w} \mid Y, \mathbf{X}) \;\propto\; P(Y \mid \mathbf{X}, \mathbf{w})p(\mathbf{w})$$

- Define priors on **w**
  - Common assumption: Normal distribution, zero mean, identity covariance
  - "Pushes" parameters towards zero

$$p(\mathbf{w}) = \prod_i \frac{1}{\kappa\sqrt{2\pi}} \; e^{\frac{-w_i^2}{2\kappa^2}}$$

Zero-mean Gaussian prior

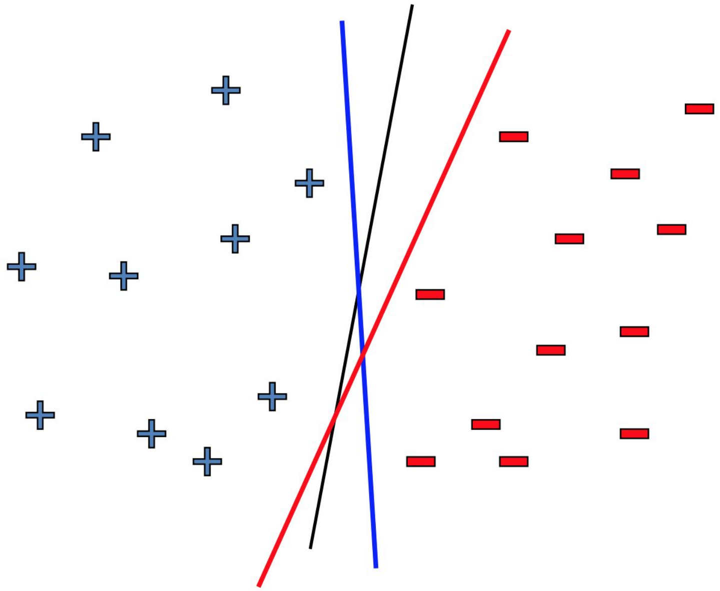
- M(C)AP estimate $\quad \mathbf{w}^* = \arg\max_{\mathbf{w}} \ln \left[p(\mathbf{w}) \prod_{j=1}^{n} P(y^j \mid \mathbf{x}^j, \mathbf{w}) \right]$

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \sum_{j=1}^{n} \ln P(y^j \mid \mathbf{x}^j, \mathbf{w}) - \underbrace{\sum_{i=1}^{d} \frac{w_i^2}{2\kappa^2}}_{\text{Penalizes large weights}}$$

Still concave objective!

Support Vector Machines (SVMs)

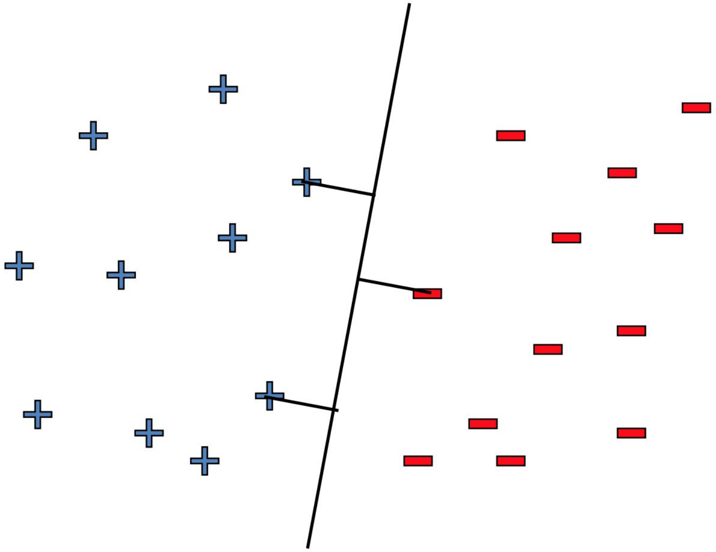
Consider a classification problem:



Often, the training data can be separated by multiple linear classifiers.
Which one to choose?

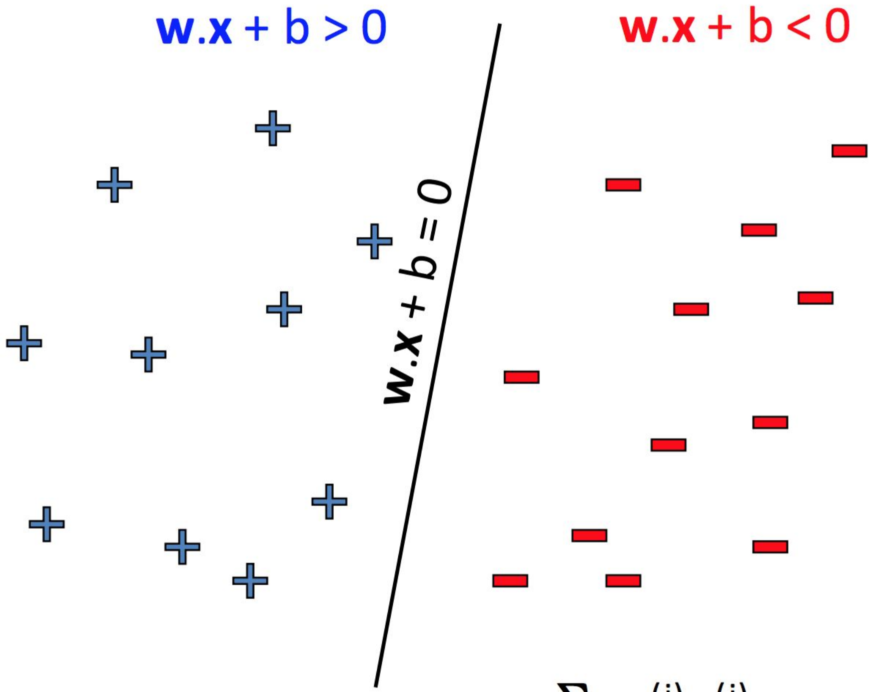
Support Vector Machines (SVMs)

Consider a classification problem:



SVMs choose the one with the largest margin.

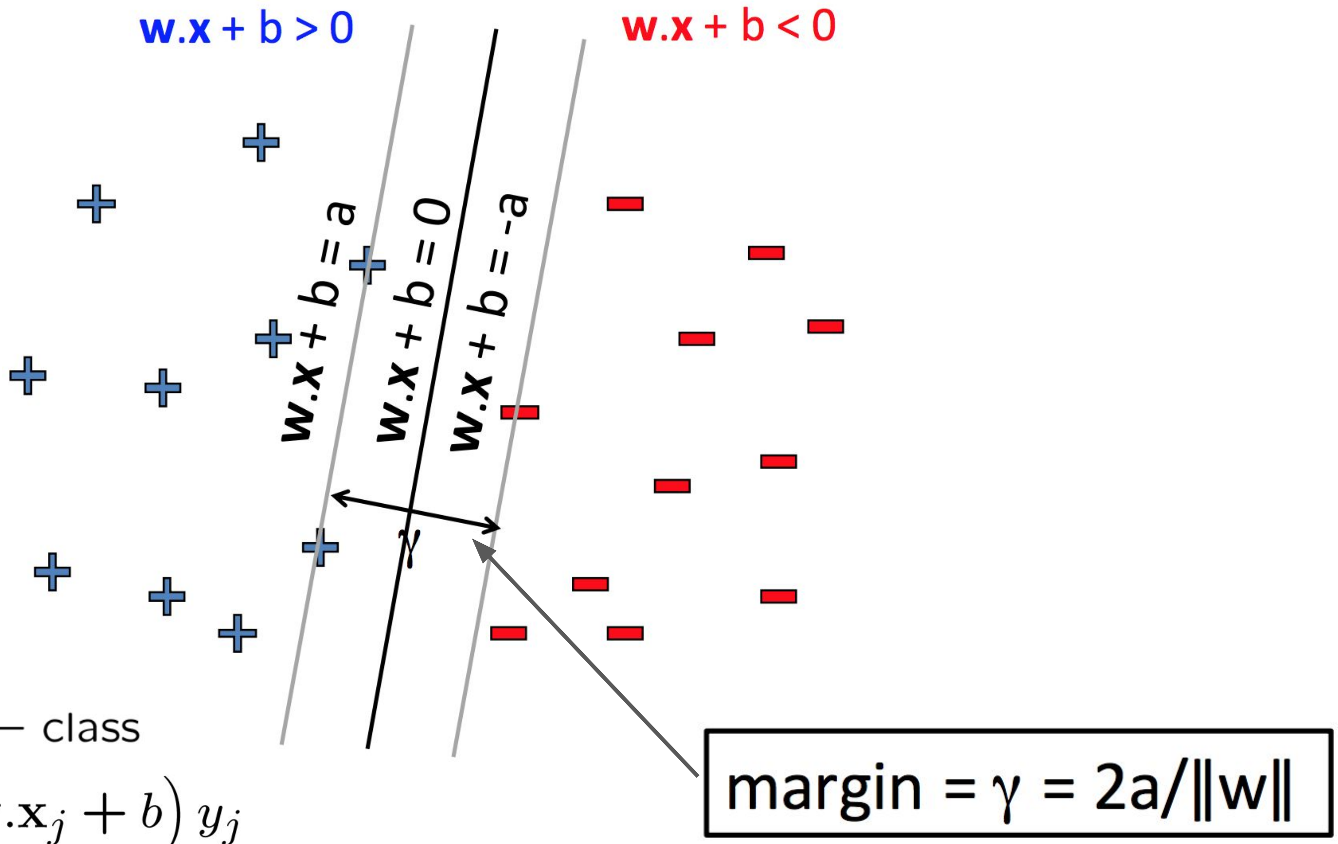
Support Vector Machines (SVMs)



$y_j \in \{-1, +1\}$ — class

$\mathbf{w}.\mathbf{x} = \sum_j w^{(j)} x^{(j)}$

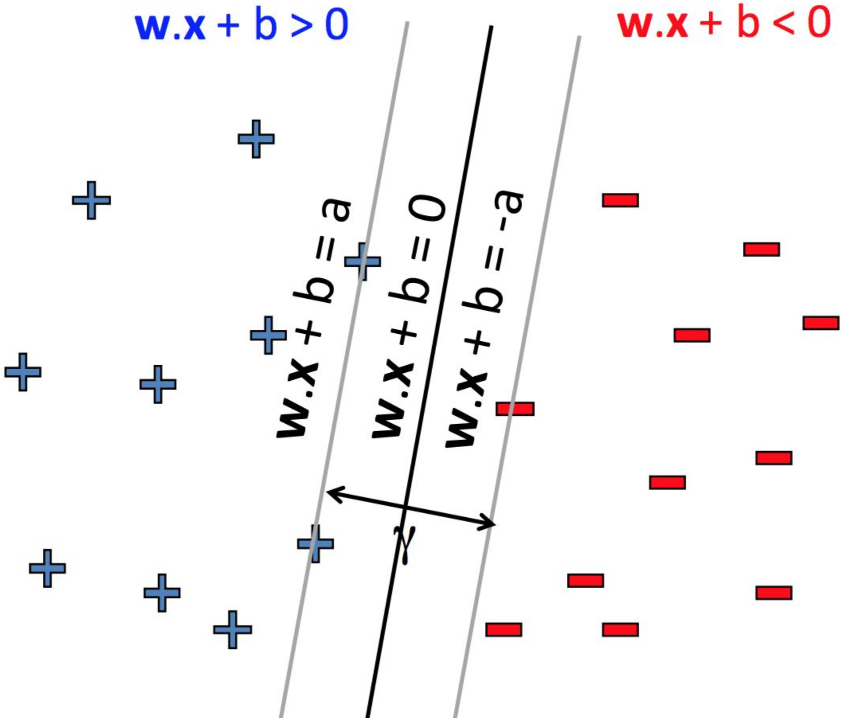
Support Vector Machines (SVMs)



$y_j \in \{-1, +1\}$ — class

"confidence" $= \left(\mathbf{w}.\mathbf{x}_j + b\right) y_j$

Support Vector Machines (SVMs)



$y_j \in \{-1, +1\}$ — class

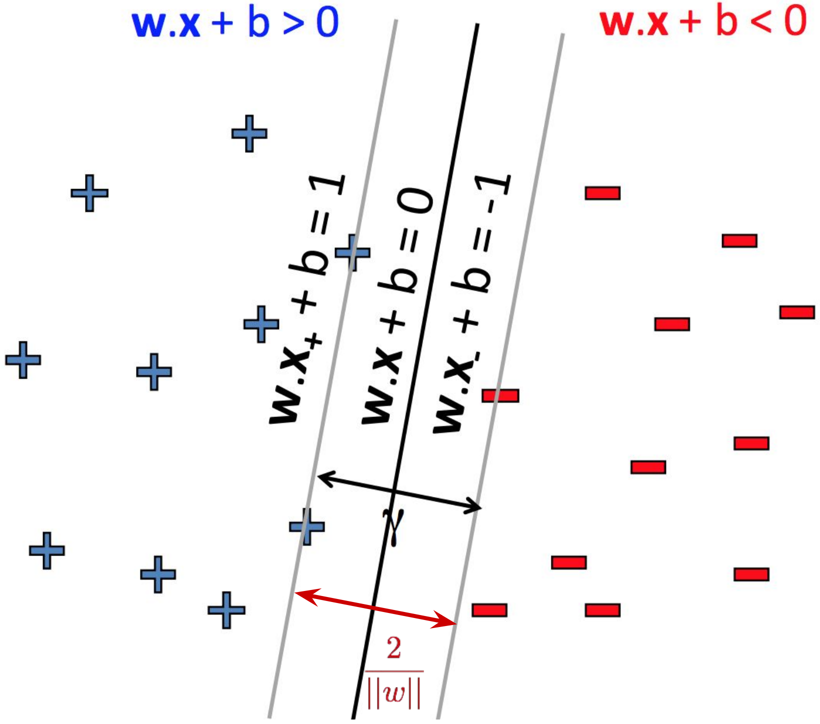
"confidence" $= \left(\mathbf{w}.\mathbf{x}_j + b\right) y_j$

Find w and b by solving:

$$\boxed{\begin{gathered}\max_{w,b} \gamma = \frac{2a}{||w||} \\ \text{s.t. } (w^T x_j + b) y_j \geq a \; \forall j\end{gathered}}$$

$$\boxed{\text{margin} = \gamma = 2a/\|w\|}$$

Support Vector Machines (SVMs)



$y_j \in \{-1, +1\}$ — class

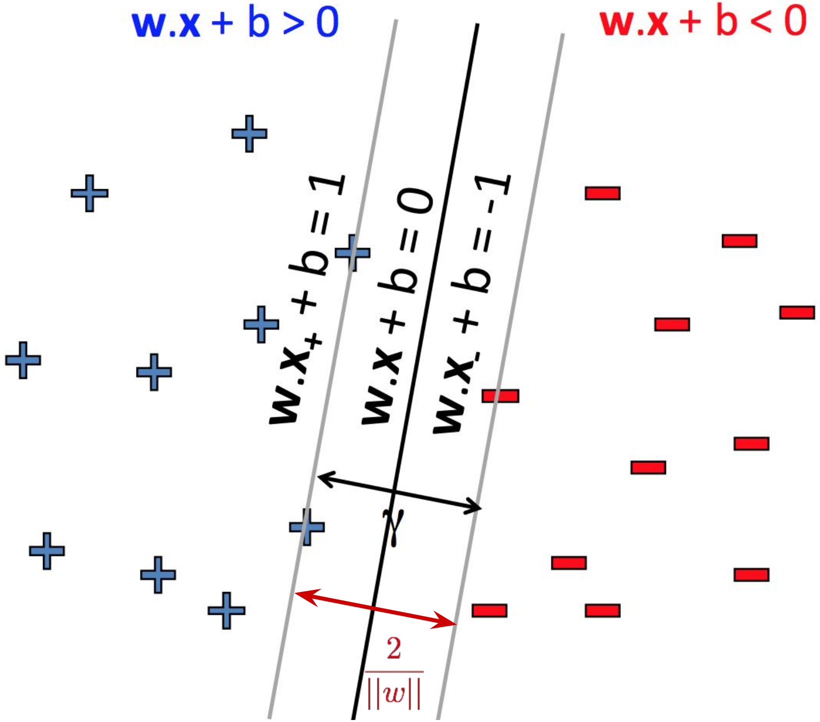
"confidence" $= \left(\mathbf{w}.\mathbf{x}_j + b\right) y_j$

Find w and b by solving:

$$\max_{w,b} \gamma = \frac{2a}{||w||}$$
$$\text{s.t.} \;\; (w^T x_j + b) y_j \geq a \;\; \forall j$$

$$\max_{w,b} \frac{2}{||w||}$$
$$\text{s.t.} \;\; (w^T x_j + b) y_j \geq 1 \;\; \forall j$$

Support Vector Machines (SVMs)



$y_j \in \{-1, +1\}$ — class

"confidence" $= \left(\mathbf{w}.\mathbf{x}_j + b\right) y_j$

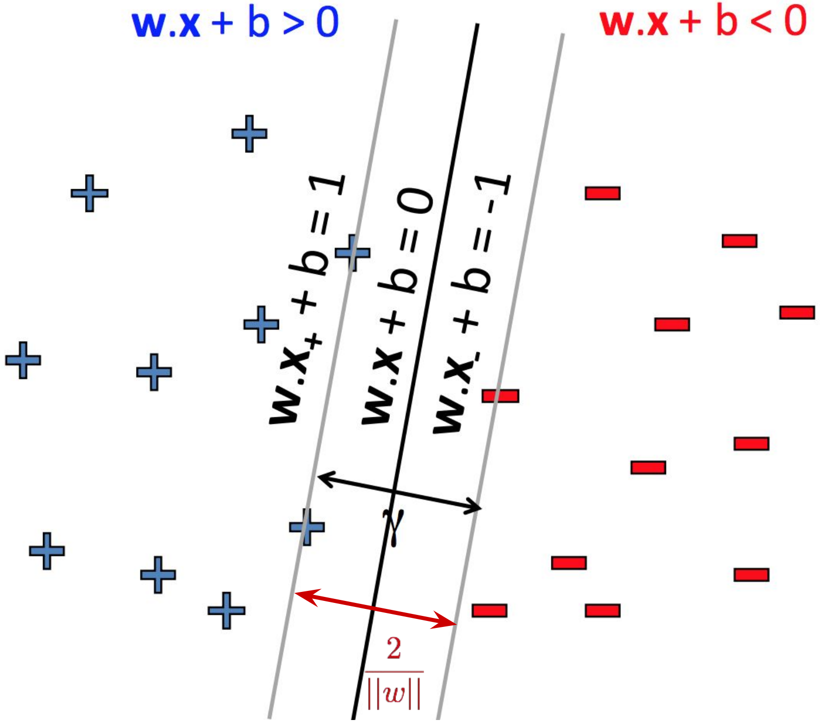
Find w and b by solving:

$$\boxed{\begin{array}{c} \max_{w,b} \gamma = \dfrac{2a}{||w||} \\ \text{s.t. } (w^T x_j + b) y_j \geq a \ \forall j \end{array}}$$

$\updownarrow$

$$\boxed{\begin{array}{c} \min_{w,b} \dfrac{||w||}{2} \\ \text{s.t. } (w^T x_j + b) y_j \geq 1 \ \forall j \end{array}}$$

Support Vector Machines (SVMs)



$y_j \in \{-1, +1\}$ — class

"confidence" $= \left(\mathbf{w}.\mathbf{x}_j + b\right) y_j$

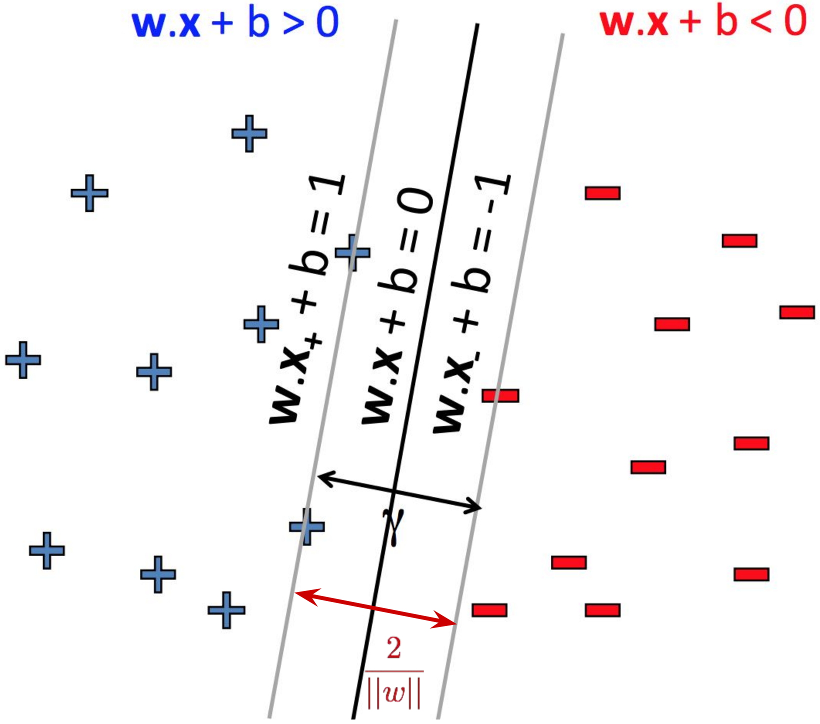
Find w and b by solving:

$$\max_{w,b} \gamma = \frac{2a}{||w||}$$
$$\text{s.t.} \ \ (w^T x_j + b) y_j \geq a \ \forall j$$

$\updownarrow$

$$\min_{w,b} ||w||$$
$$\text{s.t.} \ \ (w^T x_j + b) y_j \geq 1 \ \forall j$$

Support Vector Machines (SVMs)



$y_j \in \{-1, +1\}$ — class

"confidence" $= \left(\mathbf{w}.\mathbf{x}_j + b\right) y_j$

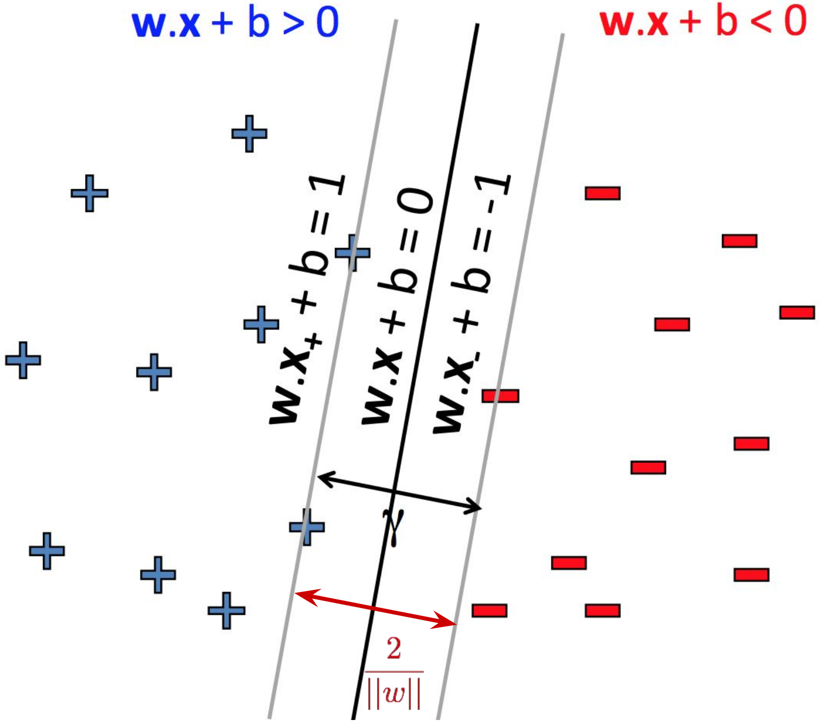
Find w and b by solving:

$$\boxed{\begin{gathered}\max_{w,b} \gamma = \frac{2a}{||w||} \\ \text{s.t. } (w^T x_j + b) y_j \geq a \ \forall j\end{gathered}}$$

$$\updownarrow$$

$$\boxed{\begin{gathered}\min_{w,b} w^T w \\ \text{s.t. } (w^T x_j + b) y_j \geq 1 \ \forall j\end{gathered}}$$

Support Vector Machines (SVMs)



$y_j \in \{-1, +1\}$ — class

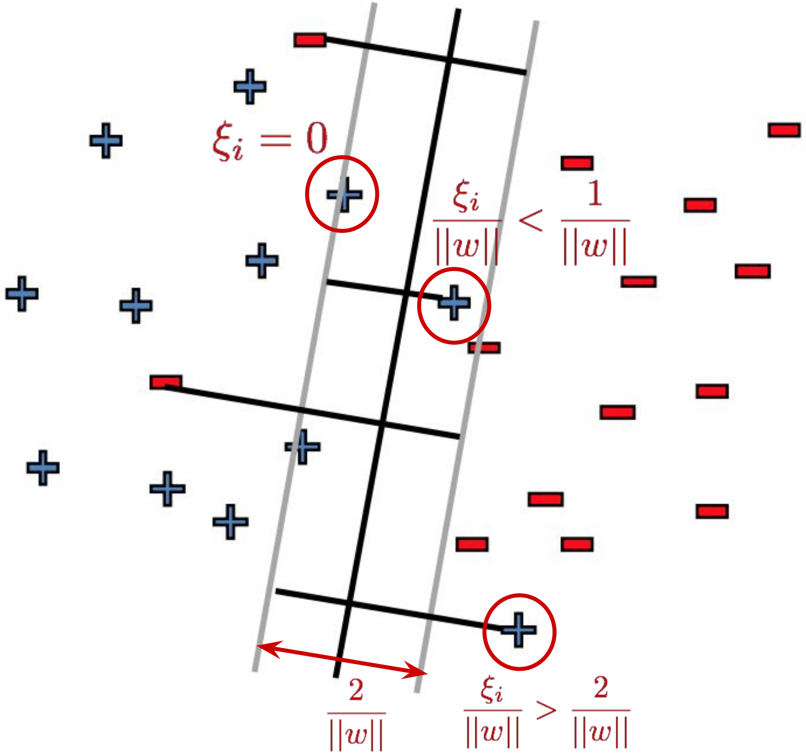
"confidence" $= \left(\mathbf{w}.\mathbf{x}_j + b\right) y_j$

Find w and b by solving:

$$\boxed{\begin{array}{c} \min\limits_{w,b} w^T w \\ \text{s.t. } (w^T x_j + b) y_j \geq 1 \ \forall j \end{array}}$$

We can solve this efficiently using quadratic programming (QP).

SVMs: What if data is not linearly separable?



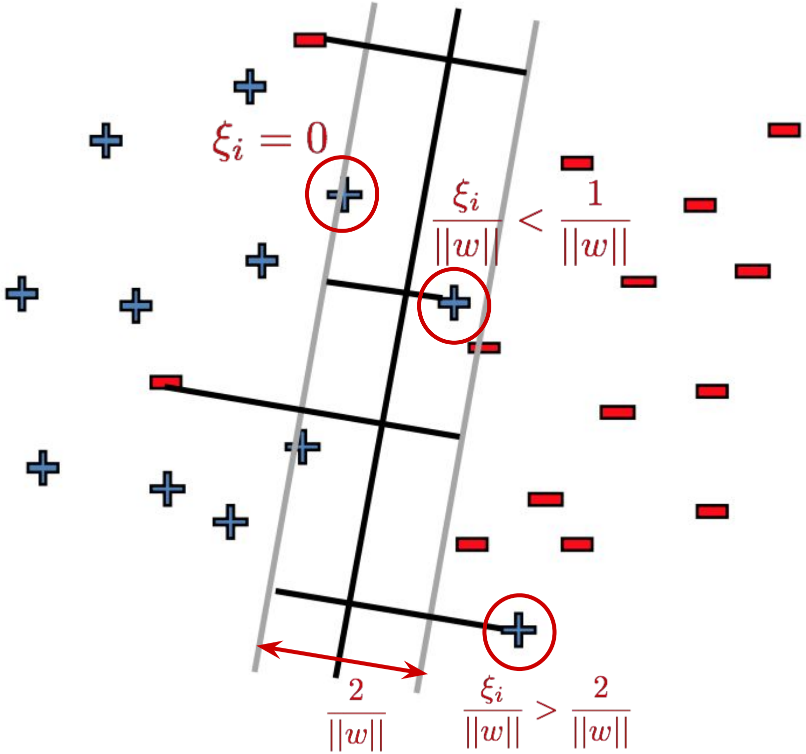
Introduce "slack" variables!

$$\xi_i \geq 0$$

such that:

- $\xi_i = 0$ if the point is on the margin
- $0 < \xi_i \leq 1$ if point is between the margin and the correct side of the hyperplane
- $\xi_i > 1$ if the point is misclassified

SVMs: What if data is not linearly separable?



Introduce "slack" variables!

$$\xi_i \geq 0$$

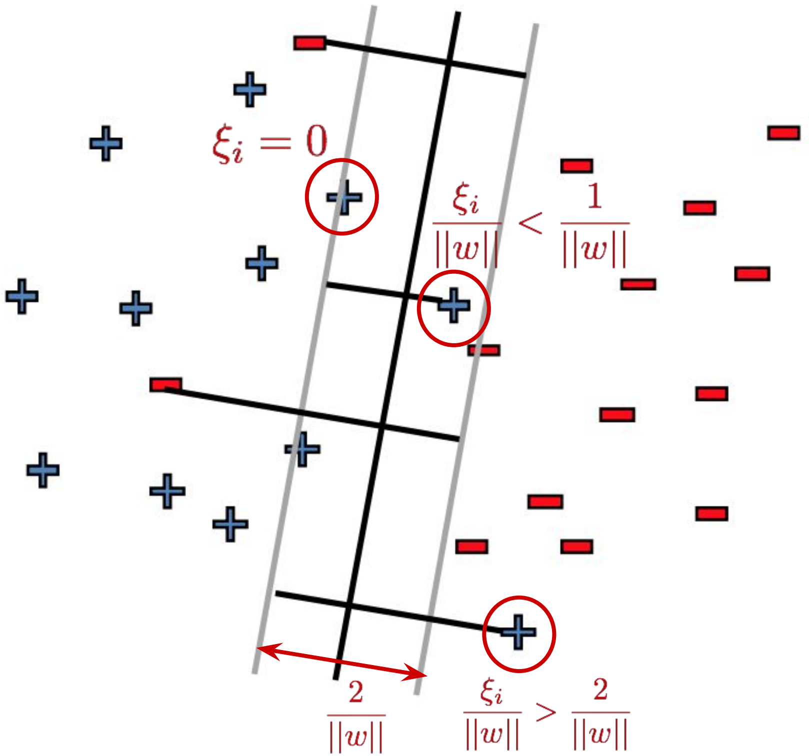
such that:

- $\xi_i = 0$ if the point is on the margin
- $0 < \xi_i \leq 1$ if point is between the margin and the correct side of the hyperplane
- $\xi_i > 1$ if the point is misclassified

The optimization problem becomes:

$$\boxed{\begin{aligned} &\min_{w,b,\xi_j} w^T w + C \sum_j \xi_j \\ &\text{s.t. } (w^T x_j + b) y_j \geq 1 - \xi_j \ \forall j \\ &\qquad \xi_j \geq 0 \ \forall j \end{aligned}}$$

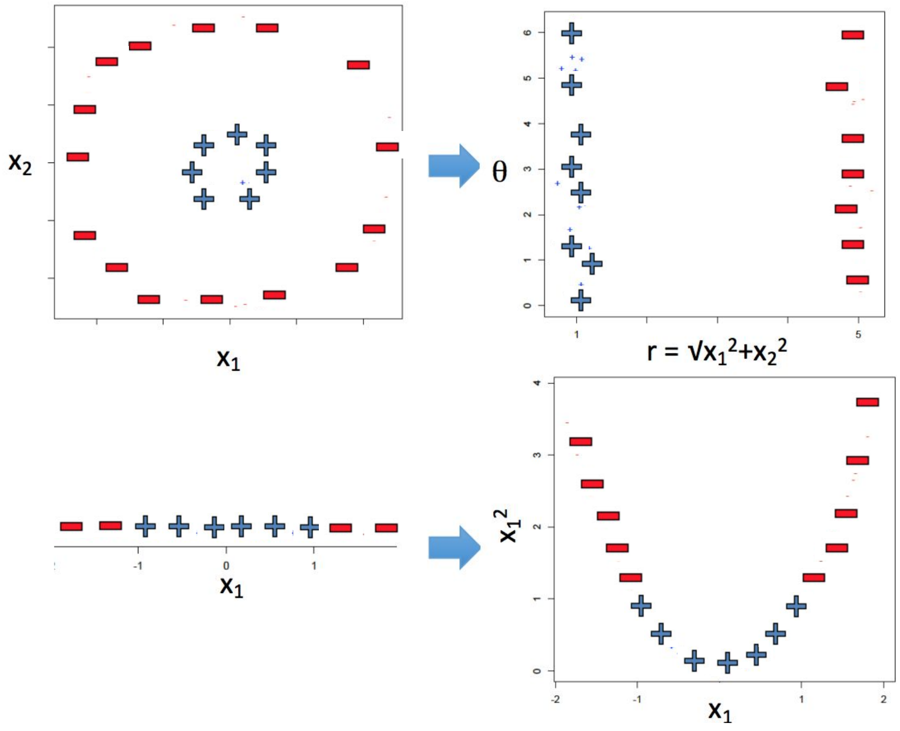
SVMs: What if data is not linearly separable?



$$\boxed{\begin{aligned} &\min_{w,b,\xi_j} w^T w + C \sum_j \xi_j \\ \text{s.t. } &(w^T x_j + b) y_j \geq 1 - \xi_j \ \forall j \\ &\xi_j \geq 0 \ \forall j \end{aligned}}$$

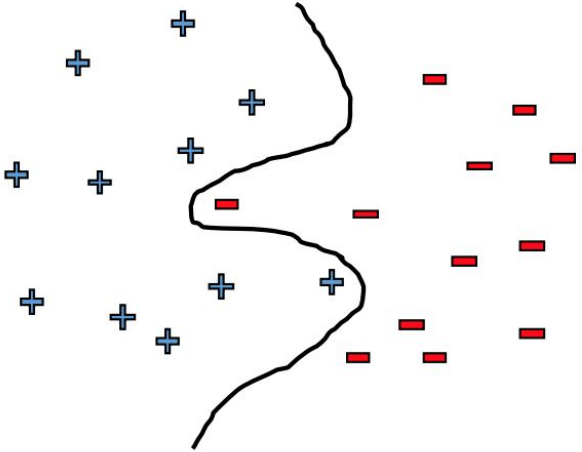
- Every constraint can be satisfied if ξ_i is sufficiently large.
- **C** is a **regularization** parameter:
  - C small allows constraints to be easily ignored → large margin
  - C large makes constraints hard to ignore → narrow margin
  - C = ∞ enforces all constraints → hard margin
- This is still a **quadratic optimization problem** and there is a **unique minimum**.

SVMs: What if data is not linearly separable?



What if data is linearly separable using higher-order features ?

SVMs: What if data is not linearly separable?



Use a feature mapping $\Phi(x)$ to convert the data to a different space.

Example:

$$\Phi(\mathbf{x}) = (x_1^2, x_2^2, x_1 x_2, \ldots., \exp(x_1))$$

What if data is linearly separable using higher-order features ?

SVMs: What if data is not linearly separable?

In the original space:

$$\min_{w,b} w^T w$$
$$\text{s.t. } (w^T x_j + b) y_j \geq 1 \ \forall j$$

In feature space:

$$\min_{w,b} w^T w$$
$$\text{s.t. } (w^T \phi(x_j) + b) y_j \geq 1 \ \forall j$$

Use a feature mapping $\Phi(x)$ to convert the data to a different space.

Example:

$$\Phi(\mathbf{x}) = (x_1^2, x_2^2, x_1 x_2, \ldots., \exp(x_1))$$

SVMs: What if data is not linearly separable?

In the original space:

$$\min_{w,b} w^T w$$
$$\text{s.t. } (w^T x_j + b) y_j \geq 1 \ \forall j$$

In feature space:

$$\min_{w,b} w^T w$$
$$\text{s.t. } (w^T \phi(x_j) + b) y_j \geq 1 \ \forall j$$

Use a feature mapping $\Phi(x)$ to convert the data to a different space.

Example:

$$\Phi(\mathbf{x}) = (x_1^2, x_2^2, x_1 x_2, \ldots, \exp(x_1))$$

Problem:

- $\Phi(x)$ can be very high dimensional (e.g. think of polynomial mapping).
- $\Phi(x)$ can be expensive to compute.

SVMs: What if data is not linearly separable?

In the original space:

$$\min_{w,b} w^T w$$
$$\text{s.t. } (w^T x_j + b) y_j \geq 1 \ \forall j$$

In feature space:

$$\min_{w,b} w^T w$$
$$\text{s.t. } (w^T \phi(x_j) + b) y_j \geq 1 \ \forall j$$

Use a feature mapping $\Phi(x)$ to convert the data to a different space.

Example:

$$\Phi(\mathbf{x}) = (x_1^2, x_2^2, x_1 x_2, \ldots, \exp(x_1))$$

Problem:

- $\Phi(x)$ can be very high dimensional (e.g. think of polynomial mapping).
- $\Phi(x)$ can be expensive to compute.

Solution:

- Maybe we don't need to compute $\Phi(x)$ explicitly!

SVMs: Deriving the dual problem

Primal problem:

$$\boxed{\begin{aligned}&\min_{w,b} w^T w \\ &\text{s.t. } (w^T x_j + b) y_j \geq 1 \ \forall j\end{aligned}}$$

Introducing Lagrange multipliers:

$$L(w, b, \alpha) = \min_{w,b} \frac{1}{2} w^T w - \sum_j \alpha_j [(w^T x_j + b) y_j - 1]$$

$$\text{s.t. } \alpha_j \geq 0 \ \forall j$$

weights on constraints
(one per training point)

Dual problem:

$$\boxed{\begin{aligned}&\max_{\alpha} \min_{w,b} L(w, b, \alpha) \\ &\text{s.t. } \alpha_j \geq 0 \ \forall j\end{aligned}}$$

SVMs: Deriving the dual problem

Primal problem:

$$\boxed{\begin{aligned} &\min_{w,b} w^T w \\ &\text{s.t. } (w^T x_j + b) y_j \geq 1 \ \forall j \end{aligned}}$$

Introducing Lagrange multipliers:

$$L(w, b, \alpha) = \min_{w,b} \frac{1}{2} w^T w - \sum_j \alpha_j [(w^T x_j + b) y_j - 1]$$

$$\text{s.t. } \alpha_j \geq 0 \ \forall j$$

weights on constraints
(one per training point)

Dual problem:

$$\boxed{\begin{aligned} &\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i x_j \\ &\text{s.t. } \sum_i \alpha_i y_i = 0 \ \forall j \\ &\qquad \alpha_j \geq 0 \ \forall j \end{aligned}}$$

SVMs: Deriving the dual problem

Primal problem:

$$\min_{w,b} w^T w$$
$$\text{s.t. } (w^T x_j + b) y_j \geq 1 \; \forall j$$

→

Primal in feature space:

$$\min_{w,b} w^T w$$
$$\text{s.t. } (w^T \phi(x_j) + b) y_j \geq 1 \; \forall j$$

Dual problem:

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i x_j$$
$$\text{s.t. } \sum_i \alpha_i y_i = 0 \; \forall j$$
$$\alpha_j \geq 0 \; \forall j$$

→

Dual in feature space:

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \phi(x_i) \phi(x_j)$$
$$\text{s.t. } \sum_i \alpha_i y_i = 0 \; \forall j$$
$$\alpha_j \geq 0 \; \forall j$$

SVMs: Deriving the dual problem

Primal problem:

$$\min_{w,b} w^T w$$
$$\text{s.t. } (w^T x_j + b) y_j \geq 1 \ \forall j$$

→

Primal in feature space:

$$\min_{w,b} w^T w$$
$$\text{s.t. } (w^T \phi(x_j) + b) y_j \geq 1 \ \forall j$$

Dual problem:

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i x_j$$
$$\text{s.t. } \sum_i \alpha_i y_i = 0 \ \forall j$$
$$\alpha_j \geq 0 \ \forall j$$

→

Dual in feature space:

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \underbrace{\phi(x_i) \phi(x_j)}$$
$$\text{s.t. } \sum_i \alpha_i y_i = 0 \ \forall j$$
$$\alpha_j \geq 0 \ \forall j$$

No need to compute Φ(x), only $\Phi(x)^T\Phi(x)$!

SVMs: Kernel trick

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \phi(x_i)\phi(x_j)$$

$$\text{s.t.} \quad \sum_i \alpha_i y_i = 0 \ \forall j$$

$$\alpha_j \geq 0 \ \forall j$$

No need to compute Φ(x), only $\Phi(x)^T\Phi(x)$!

Kernel trick: replace these dot products with an equivalent kernel function

$$K(x_i, x_j) = \phi(x_i)\phi(x_j)$$

No need to know Φ(x), we can choose K directly from one of the commonly used kernel functions (e.g., polynomial, RBF, Gaussian, sigmoid).

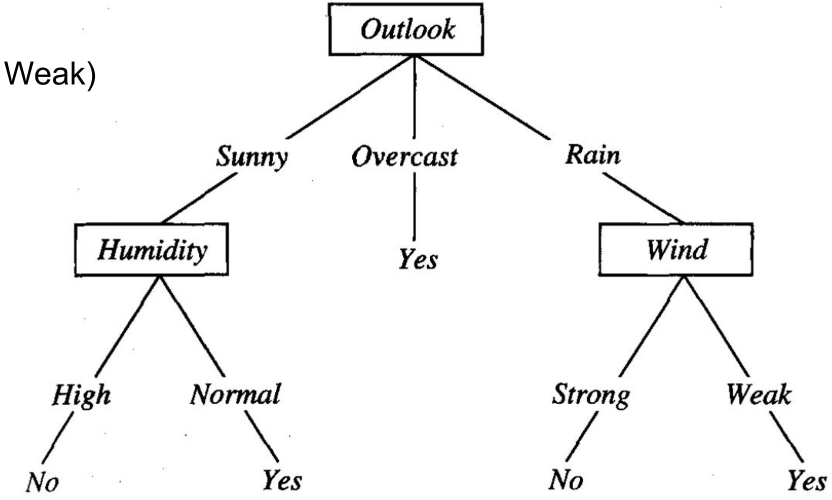
SVMs

Questions?

Decision Tree Learning: Introduction

- Approximate discrete valued target functions where outcome is in form of decision tree
  - Sort observations by attributes from root to leaf

(Outlook = Sunny, Humidity = High, Wind = Weak)



Decision Tree Learning: Usages

- Observations are represented by a fixed set of attribute pairs

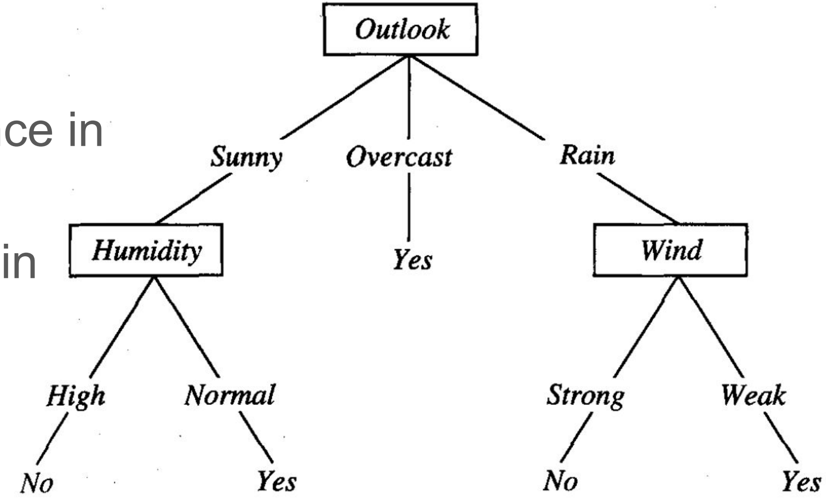
- Target function has discrete action values

- Training Data may contain errors/is noisy/missing attribute values

Decision Tree Learning: Example Learner - ID3

- Learns decision trees in a top down manner
  - Sort instance attributes in descending order of importance from root to leaves.
  - Attribute order dependency contained within branch

- Each attribute can appear at most once in a branch
- Importance Measure: Information Gain



Decision Tree Learning: ID3 Importance Measure

Information Gain

- Advantage of attribute = decrease in uncertainty
  - Entropy of Y before split

$$H(Y) = -\sum_y P(Y = y) \log_2 P(Y = y)$$

  - Entropy of Y after splitting based on X_i
    - Weight by probability of following each branch

$$\begin{aligned} H(Y \mid X_i) &= \sum_x P(X_i = x) H(Y \mid X_i = x) \\ &= -\sum_x P(X_i = x) \sum_y P(Y = y \mid X_i = x) \log_2 P(Y = y \mid X_i = x) \end{aligned}$$

- Information gain is difference

$$I(Y, X_i) = H(Y) - H(Y \mid X_i)$$

Max Information gain = min conditional entropy

Decision Tree Learning: ID3 Importance Measure

Pick the attribute/feature which yields maximum information gain:

$$\arg\max_i I(Y, X_i) = \arg\max_i [H(Y) - H(Y|X_i)]$$

$$= \arg\min_i H(Y|X_i)$$

Entropy of Y $\quad H(Y) = -\sum_y P(Y = y) \log_2 P(Y = y)$

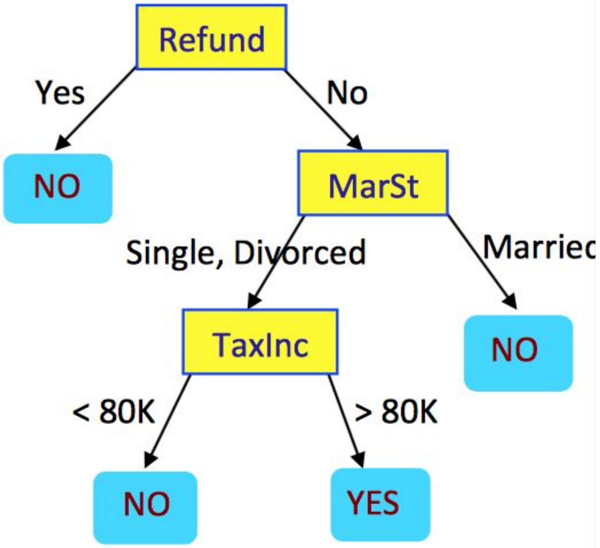
Conditional entropy of Y $\quad H(Y \mid X_i) = \sum_x P(X_i = x) H(Y \mid X_i = x)$

Feature which yields maximum reduction in entropy (uncertainty) provides maximum information about Y

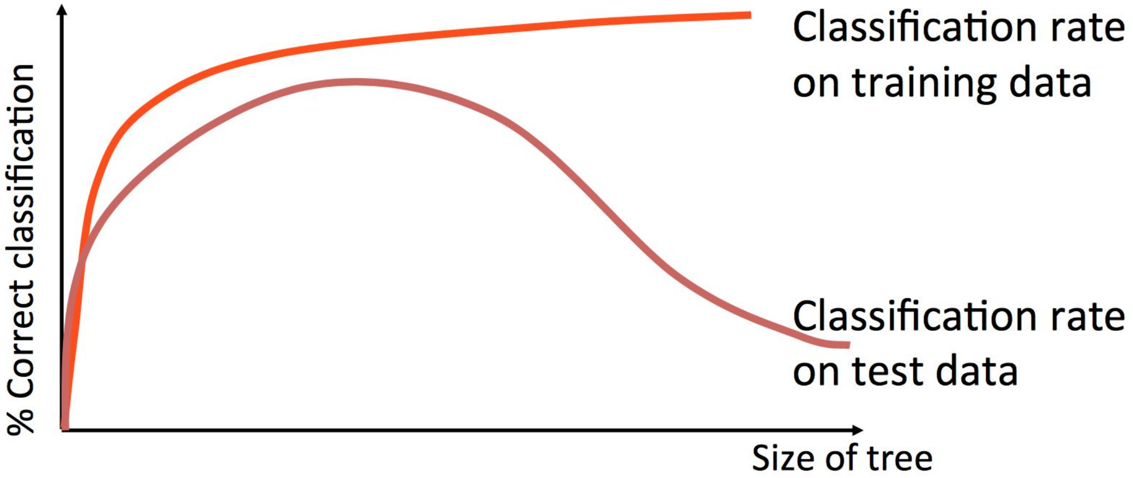
Decision Tree Learning: ID3 Build-Up

Main loop: C4.5

1. $X \leftarrow$ the $\boxed{\text{“best”}}$ decision feature for next *node*
2. Assign X as decision feature for *node*
3. For $\boxed{\text{“best”}}$ split of X, create new descendants of *node*
4. Sort training examples to leaf nodes
5. If training examples perfectly classified, Then STOP, Else iterate over new leaf nodes



Decision Tree Learning: Overfitting



- Can be reduced by pruning

Decision Tree Learning: General Takeaways

- In general a decision tree can represent any discrete valued function
- Decision Trees perform a complete search through the Hypothesis space (consisting of decision trees)
    - The target optimal decision tree is guaranteed to be in this space
- ID3 Algorithm creates trees by placing attributes with highest IG closest to root

Acknowledgements

- Neural network slides inspired by Russ Salakhutdinov' s slides