GHENT UNIVERSITY

FACULTY OF ENGINEERING AND ARCHITECTURE

E016350 - Artificial Intelligence

Lecture 3

Machine learning

Optimization Algorithms and Logistic Regression

Aleksandra Pizurica

Ghent University
Spring 2025

Outline

These slides are based on: Andrew Ng and Tengyu Ma: Lecture Notes (CS229) Machine learning `https://cs229.stanford.edu/main_notes.pdf` and Moses Charikar and Sanmi Koyejo: (CS221) Artificial Intelligence: Principles and Techniques (Stanford)

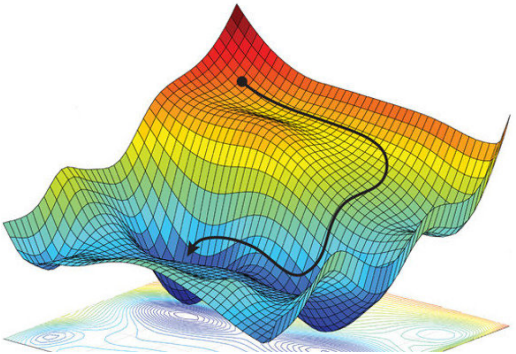
Outline

Optimization in machine learning

Our learning task: determine the parameters (weights) $\mathbf{w}$ of a hypothesis $h_{\mathbf{w}}(\mathbf{x})$ that approximates true, unknown function $y = f(\mathbf{x})$.

To find the optimal $\mathbf{w}$ we minimize a loss function.

– Why using gradient descent?

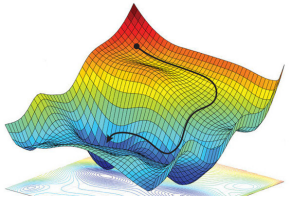


Picture credit: N. Azizan R. and B. Hassibi. Stochastic Gradient/Mirror Descent: Minimax Optimality and Implicit Regularization. ICLR 2019.

What is the Steepest Direction?

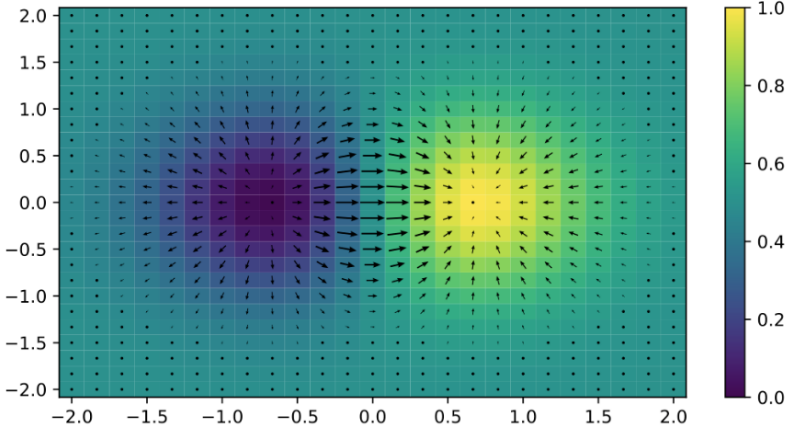
Goal: take a step $\Delta : \Delta_1^2 + \Delta_2^2 < \epsilon$ such that $Loss(\mathbf{w} + \Delta) \leq Loss(\mathbf{w})$

$$\min_{\Delta:\Delta_1^2+\Delta_2^2<\epsilon} Loss(\mathbf{w}+\Delta); \qquad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}; \qquad \Delta = \begin{bmatrix} \Delta_1 \\ \Delta_2 \end{bmatrix}$$



- First-order Taylor expansion
  $Loss(\mathbf{w}+\Delta) \approx Loss(\mathbf{w}) + \frac{\partial Loss}{\partial w_1}\Big|_{\mathbf{w}} \Delta_1 + \frac{\partial Loss}{\partial w_2}\Big|_{\mathbf{w}} \Delta_2 = Loss(\mathbf{w}) + \Delta^\top \nabla_{\mathbf{w}} Loss(\mathbf{w})$
- So, for maximum leverage out of moving along Δ:
  align Δ with $-\nabla_{\mathbf{w}} Loss(\mathbf{w})$
- I.e., steepest direction (down) = (negative) gradient direction in a given point

A visualization of the gradient field



The gradient of a scalar-valued differentiable function of several variables is the vector field whose value at a point p gives the direction and the rate of fastest increase at that point.
https://en.wikipedia.org/wiki/Gradient

Gradient in d dimensions

Same reasoning in arbitrary number of directions

In d dimensions, $\mathbf{w} \in \mathbb{R}^d$ and the gradient of the loss function in particular $\mathbf{w}$ point:

$$\nabla_{\mathbf{w}} Loss(\mathbf{w}) = \begin{bmatrix} \frac{\partial Loss}{\partial w_1}(\mathbf{w}) \\ \frac{\partial Loss}{\partial w_2}(\mathbf{w}) \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial Loss}{\partial w_d}(\mathbf{w}) \end{bmatrix}$$

Weight optimization

We optimize the $\mathbf{w} \in \mathbb{R}^d$ by applying the gradient descent to the training loss:

$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} TrainLoss(w_1, ..., w_d)$$

Sometimes we write this more compactly as

$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} TrainLoss(\mathbf{w})$$

or in a vector form

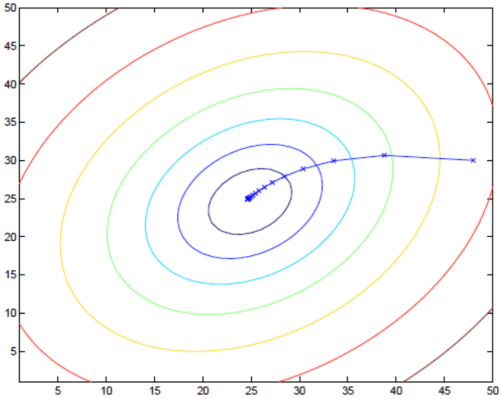
$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} TrainLoss(\mathbf{w})$$

- Perform update in downhill direction for each coordinate
- The steeper the slope (i.e. the larger the magnitude of the derivative) the bigger the step for that coordinate.

Gradient descent algorithm

Idea:

- Start somewhere
- Repeat: Take a step in the gradient direction



Credit: Stanford CS229 Course Notes. Trajectory for gradient descent is like climbing down into a valley

Optimization procedure: Gradient Descent (GD)

$$TrainLoss(\mathbf{w}) = \frac{1}{|\mathcal{D}_{train}|} \sum_{(x,y)\in\mathcal{D}_{train}} L(x, y, \mathbf{w})$$

- init $\mathbf{w} = [0, \ldots, 0]$
- for iter $1, 2, \ldots$

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} TrainLoss(\mathbf{w})$$

- α: learning rate — tweaking parameter that needs to be chosen carefully
- How? Try multiple choices
  - Crude rule of thumb: update changes $\mathbf{w}$ about 0.1–1%

Influence of the learning rate

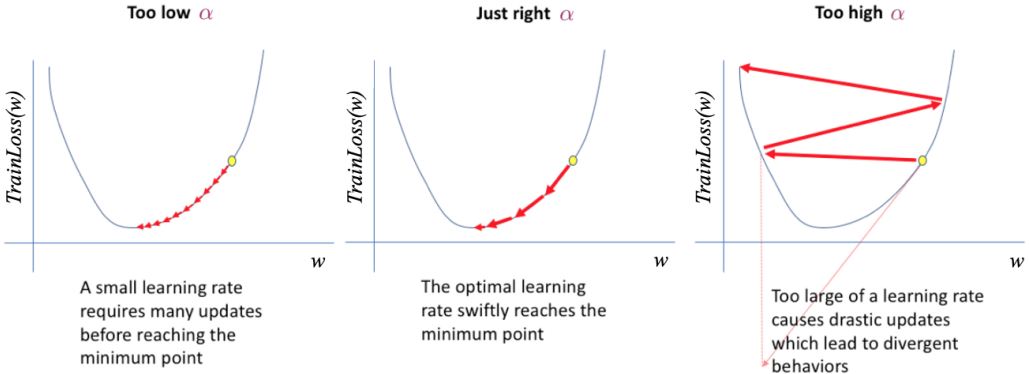


Illustration Credit: Edouard Duchesnay, NeuroSpin CEA Université Paris-Saclay, France.

Stochastic Gradient Descent (SGD)

$$TrainLoss(\mathbf{w}) = \frac{1}{|\mathcal{D}_{train}|} \sum_{(x,y)\in\mathcal{D}_{train}} L(x, y, \mathbf{w})$$

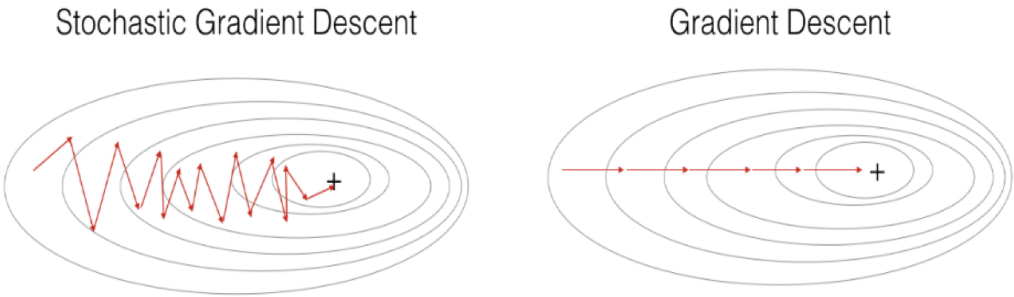
- init $\mathbf{w} = [0, \ldots, 0]$
- for iter $1, 2, ...$
  - For $(x, y) \in \mathcal{D}_{train}$:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} L(x, y, \mathbf{w})$$

Motivation:

- Gradient descent algorithm is slow: going over all the training examples in each iteration is expensive when lots of data!
- Rather than looping through all the training examples to compute a single gradient, update the weights based on each example $\rightarrow$ SGD

SGD vs. GD



"+" denotes a minimum of the cost. SGD leads to many oscillations to reach convergence. But each step is a lot faster to compute for SGD than for GD, as it uses only one training example (vs. the whole batch for GD).

Illustration Credit: Ankit-AI - Sharing AI: Optimization Algorithms for Machine Learning.

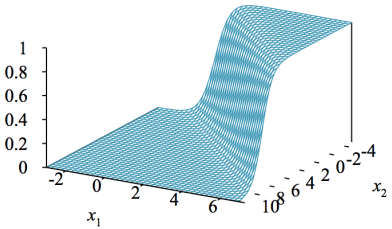
Outline

Logistic regression - Reminder

We consider binary classification: to each input data point $\mathbf{x} \in \mathbb{R}^d$ we assign a class label $y \in \{0, 1\}$.

Let $g(z)$ denote the logistic (sigmoid) function:

$$g(z) = Logistic(z) = \frac{1}{1 + e^{-z}}$$



For some weight vector $\mathbf{w} \in \mathbb{R}^d$, the hypothesis

$$h_{\mathbf{w}}(\mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

can be interpreted as the probability that $\mathbf{x}$ belongs to class 1, i.e., the probability that $y = 1$.

Logistic regression - Probabilistic interpretation formally

We said: $h_{\mathbf{w}}(\mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x})$ can be interpreted as the probability that $y = 1$. Formally:

$$\begin{aligned} P(y = 1|\mathbf{x}, \mathbf{w}) &= h_{\mathbf{w}}(\mathbf{x}) \\ P(y = 0|\mathbf{x}, \mathbf{w}) &= 1 - h_{\mathbf{w}}(\mathbf{x}) \end{aligned}$$

Or, more compactly:

$$P(y|\mathbf{x}, \mathbf{w}) = (h_{\mathbf{w}}(\mathbf{x}))^y (1 - h_{\mathbf{w}}(\mathbf{x}))^{(1-y)}$$

If the training examples were generated independently, the likelihood of the weights is:

$$\mathcal{L}(\mathbf{w}) = \prod_{i=1}^{N} P(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) = \prod_{i=1}^{N} \left(h_{\mathbf{w}}(\mathbf{x}^{(i)}) \right)^{y^{(i)}} \left(1 - h_{\mathbf{w}}(\mathbf{x}^{(i)}) \right)^{1-y^{(i)}}$$

it is easier to maximize the log likelihood:

$$\ell(\mathbf{w}) = \log \mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} y^{(i)} \log h_{\mathbf{w}}(\mathbf{x}^{(i)}) \; + \; (1 - y^{(i)}) \log(1 - h_{\mathbf{w}}(\mathbf{x}^{(i)})$$

Logistic regression under the maximum likelihood optimization

Now we can determine the update rule for the logistic regression by maximizing the log-likelihood of the weights. This is the most common form of the logistic regression.

Note that now $TrainLoss(\mathbf{w}) = -\ell(\mathbf{w})$, so we are applying the gradient descent algorithm to $-\ell(\mathbf{w})$, or equivalently, we are applying the **gradient ascent** to $\ell(\mathbf{w})$:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \nabla_{\mathbf{w}} \ell(\mathbf{w})$$

We start with one training example $(\mathbf{x}, y)$:

$$\begin{aligned}
\frac{\partial}{\partial w_j}\ell(\mathbf{w}) &= \Big(y\frac{1}{g(\mathbf{w}\cdot\mathbf{x})} - (1-y)\frac{1}{1-g(\mathbf{w}\cdot\mathbf{x})}\Big)\frac{\partial}{\partial w_j} g(\mathbf{w}\cdot\mathbf{x}) \\
&= \Big(y\frac{1}{g(\mathbf{w}\cdot\mathbf{x})} - (1-y)\frac{1}{1-g(\mathbf{w}\cdot\mathbf{x})}\Big) g(\mathbf{w}\cdot\mathbf{x})(1-g(\mathbf{w}\cdot\mathbf{x}))\frac{\partial}{\partial w_j}(\mathbf{w}\cdot\mathbf{x}) \\
&= (y(1-g(\mathbf{w}\cdot\mathbf{x})) \;-\; (1-y)g(\mathbf{w}\cdot\mathbf{x}))x_j \\
&= (y - h_{\mathbf{w}}(\mathbf{x}))x_j
\end{aligned}$$

Logistic regression under the maximum likelihood optimization

The maximum likelihood estimation gives us the following update rule

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \nabla_{\mathbf{w}} \ell(\mathbf{w})$$

where for one training example we derived

$$\frac{\partial}{\partial w_j} \ell(\mathbf{w}) \quad = \quad (y - h_{\mathbf{w}}(\mathbf{x})) x_j$$

Hence, the MLE update rule for the logistic regression, with one example, is

$$w_j \leftarrow w_j + \alpha (y - h_{\mathbf{w}}(\mathbf{x})) x_j$$

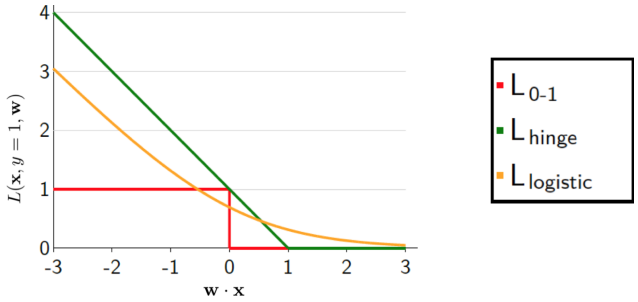
and with all training examples

$$w_j \leftarrow w_j + \alpha \sum_{(\mathbf{x},y) \in \mathcal{D}_{train}} (y - h_{\mathbf{w}}(\mathbf{x})) x_j$$

Note: Looks exactly the same as for the least-squares linear regression ($h_{\mathbf{w}}$ is different)

Classification losses

Some common losses for binary classification (shown for $y = 1$).

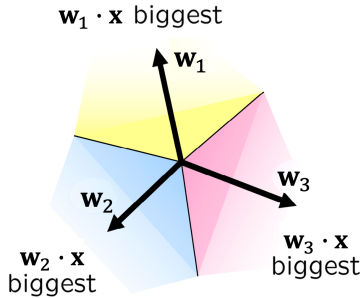


- **Logistic** loss:
  $L_{\text{logistic}}(\mathbf{x}, y, \mathbf{w}) = -\ell(\mathbf{w}) = -y \log h_{\mathbf{w}}(\mathbf{x}) - (1 - y) \log(1 - h_{\mathbf{w}}(\mathbf{x}))$
  - For $y = 1$, this becomes: $L_{\text{logistic}}(\mathbf{x}, y = 1, \mathbf{w}) = \log(1 + e^{-\mathbf{w} \cdot \mathbf{x}})$
- **Hinge** loss:
  - $L_{\text{hinge}}(\mathbf{x}, y = 1, \mathbf{w}) = \max(1 - \mathbf{w} \cdot \mathbf{x}, 0)$ (and $L_{\text{hinge}}(\mathbf{x}, 0, \mathbf{w}) = \max(1 + \mathbf{w} \cdot \mathbf{x}, 0)$)
  - This loss is often used with support vector machines (SVM)

Outline

Multiclass classification

- Multi-class linear classification
  - A weight vector for each class: $\mathbf{w}_y$
  - Score (activation) of a class y : $\mathbf{w}_y \cdot \mathbf{x}$
  - Prediction "the highest score wins": $\arg\max_y \mathbf{w}_y \cdot \mathbf{x}$



- How to turn the scores into probabilities?

$$\underbrace{z_1, z_2, z_3}_{\text{original activations}} \quad \rightarrow \quad \underbrace{\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}, \frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}, \frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}}_{\text{softmax activations}}$$

- In general, for K classes:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

Multiclass perceptron

$$h_{\mathbf{w}}(\mathbf{x}) = \arg\max_i \ \mathbf{w}_i \cdot \mathbf{x}$$

one-hot encoding:

$$\mathbf{t} = \underbrace{[0, \ldots, 0, 1, 0, \ldots, 0]^\top}_{\text{entry } k \text{ is one}} \in \mathbb{R}^K$$

Let $\mathbf{o} \in \mathbb{R}^K$ be the classifier's output vector. For the multiclass perceptron

$$o_k = \begin{cases} 1 & \text{if } k = \arg\max_i \ \mathbf{w}_i \cdot \mathbf{x} \\ 0 & \text{otherwise} \end{cases}$$

Multiclass Logistic Regression

softmax rule:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}, \quad i = 1, \ldots, K$$

softmax regression:

$$h_{\mathbf{w}}(\mathbf{x}) = \begin{bmatrix} P(y=1|\mathbf{x},\mathbf{w}) \\ P(y=2|\mathbf{x},\mathbf{w}) \\ \vdots \\ P(y=K|\mathbf{x},\mathbf{w}) \end{bmatrix} = \frac{1}{\sum_{k=1}^{K} e^{\mathbf{w}_k \cdot \mathbf{x}}} \begin{bmatrix} e^{\mathbf{w}_1 \cdot \mathbf{x}} \\ e^{\mathbf{w}_2 \cdot \mathbf{x}} \\ \vdots \\ e^{\mathbf{w}_K \cdot \mathbf{x}} \end{bmatrix}$$

The softmax output

$$o_k = softmax(\mathbf{w}_k \cdot \mathbf{x}), \quad k = 1, \ldots, K$$

Finding the best weights

Maximum likelihood estimation

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \arg\max_{\mathbf{w}} \prod_{i=1}^{N} P(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w})$$

with

$$P(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) = \frac{e^{\mathbf{w}_{y^{(i)}} \cdot \mathbf{x}^{(i)}}}{\sum_{y} e^{\mathbf{w}_{y^{(i)}} \cdot \mathbf{x}^{(i)}}}$$

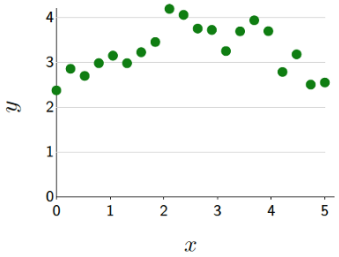
(softmax activations serve as probabilities)

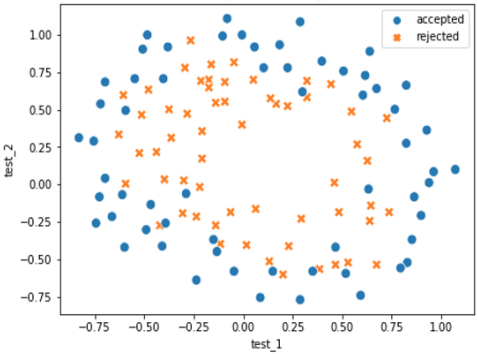
Outline

More complex data

So far we analysed linear regression and linear classification (with logistic regression).

– But in many cases data show nonlinear trends / nonlinear separability.





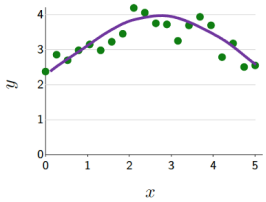
Can we handle such more complex data with the machinery of linear predictors?

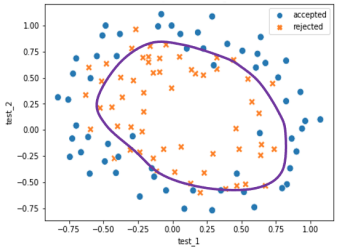
Linear predictors with nonlinear features

Idea: extract a vector of nonlinear features $\phi(x)$ from input x (no matter its dimension) and feed $\phi(x)$ to a linear predictor. It's going to be non-linear in x!

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \phi(x)$$

$$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \phi(x)}}$$

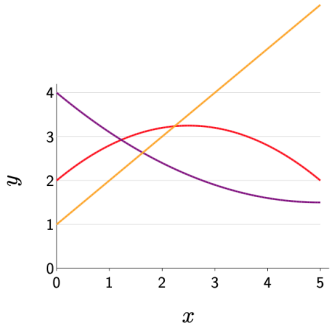




Quadratic predictors

$\phi(x) = [1, x, x^2]$ (Example: $\phi(3) = [1, 3, 9]$)

$h_{\mathbf{w}}(x) = [2, 1, -0.2] \cdot \phi(x)$
$h_{\mathbf{w}}(x) = [4, -1, 0.1] \cdot \phi(x)$
$h_{\mathbf{w}}(x) = [1, 1, 0] \cdot \phi(x)$



Hypothesis space:
$\mathcal{H} = \{h_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x) : \mathbf{w} \in \mathbb{R}^3\}$

Non-linear predictors just by changing ϕ !

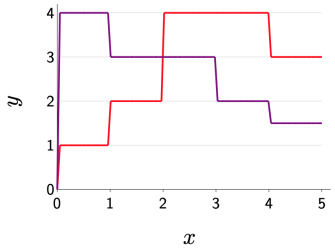
Slide credit: M. Charikar and Koyejo: Artificial Intelligence: Principles and Techniques (Stanford)

Piece-wise constant predictors

$$\phi(x) = [\mathbf{1}[0 < x \leq 1], \mathbf{1}[1 < x \leq 2], \mathbf{1}[2 < x \leq 3], \mathbf{1}[3 < x \leq 4], \mathbf{1}[4 < x \leq 5]]$$

(Example: $\phi(2.3) = [0, 0, 1, 0, 0]$)

$$h_{\mathbf{w}}(x) = [1, 2, 4, 4, 3] \cdot \phi(x)$$
$$h_{\mathbf{w}}(x) = [4, 3, 3, 2, 1.5] \cdot \phi(x)$$



Hypothesis space:

$$\mathcal{H} = \{h_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x) : \mathbf{w} \in \mathbb{R}^5\}$$

Expressive non-linear predictors by partitioning the input space.

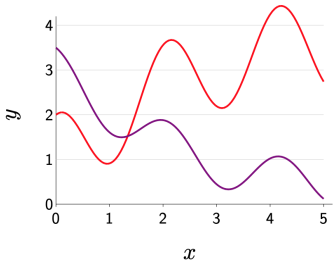
Slide credit: M. Charikar and Koyejo: Artificial Intelligence: Principles and Techniques (Stanford)

Predictors with periodicity structure

$\phi(x) = [1, x, x^2, \cos(3x)]$

(Example: $\phi(2) = [1, 2, 4, 0.96]$)

$h_{\mathbf{w}}(x) = [1, 1, -0.1, 1] \cdot \phi(x)$
$h_{\mathbf{w}}(x) = [3, -1, 0.1, 0.5] \cdot \phi(x)$



Hypothesis space:
$\mathcal{H} = \{h_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x) : \mathbf{w} \in \mathbb{R}^4\}$

Just throw in any features you want!

Slide credit: M. Charikar and Koyejo: Artificial Intelligence: Principles and Techniques (Stanford)

Quadratic classifiers

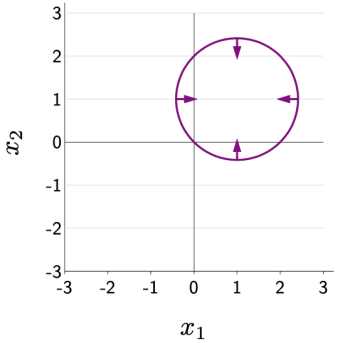
$$\phi(x) = [x_1, x_2, x_1^2 + x_2^2]$$

A linear classifier with a hard threshold:

$$h_{\mathbf{w}}(\mathbf{x}) = Threshold\Big([2, 2, -1]) \cdot \phi(x)\Big)$$

is now equivalent to:

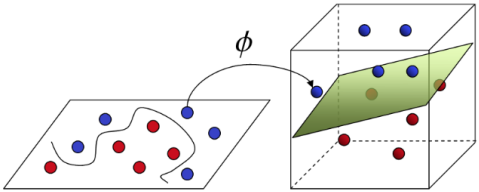
$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & (x_1 - 1)^2 + (x_2 - 1)^2 \leq 2 \\ 0 & \text{otherwise} \end{cases}$$



Decision boundary is a circle.

Slide credit: M. Charikar and Koyejo: Artificial Intelligence: Principles and Techniques (Stanford)

A general concept: Kernel method



D. Wilimitis: The Kernel Trick in Support Vector Classification

- Kernel machines involve using linear classifiers to solve nonlinear problems.
- Best known representative: Support Vector Machines (SVM)

Next time

- Decision trees
- Information gain & Decision tree learning