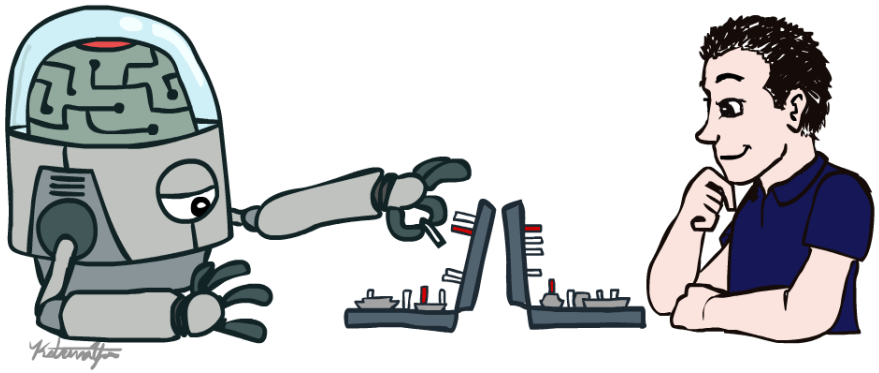
CSE 3521:
Introduction to Artificial Intelligence



[Many slides are adapted from the UC Berkeley. CS188 Intro to AI at UC Berkeley and previous CSE 3521 course at OSU.]

THE OHIO STATE UNIVERSITY

Decision Tree

ID3 Algorithm

ID3 (S, A, V)

Let:

S = Learning Set
A = Attibute Set
V = Attribute Values

Begin

Load learning sets and create decision tree root node(rootNode),
Add learning set S into root not as its subset

For rootNode, compute Entropy(rootNode.subset)

If Entropy(rootNode.subset) == 0 (subset is homogeneous)
return a leaf node

If Entropy(rootNode.subset)!= 0 (subset is not homogeneous)
compute Information Gain for each attribute left (not been used for spliting)
Find attibute A with Maximum(Gain(S,A))
Create child nodes for this root node and add to rootNode in the decision tree

For each child of the rootNode
Apply ID3(S,A,V)
Continue until a node with Entropy of 0 or a leaf node is reached

End

Decision Tree Equations

$$Entropy(S) = \sum_{i=1}^{c} -p_i log_2 p_i$$

S= Collection of Examples
P_i = proportion of S that belongs to class i

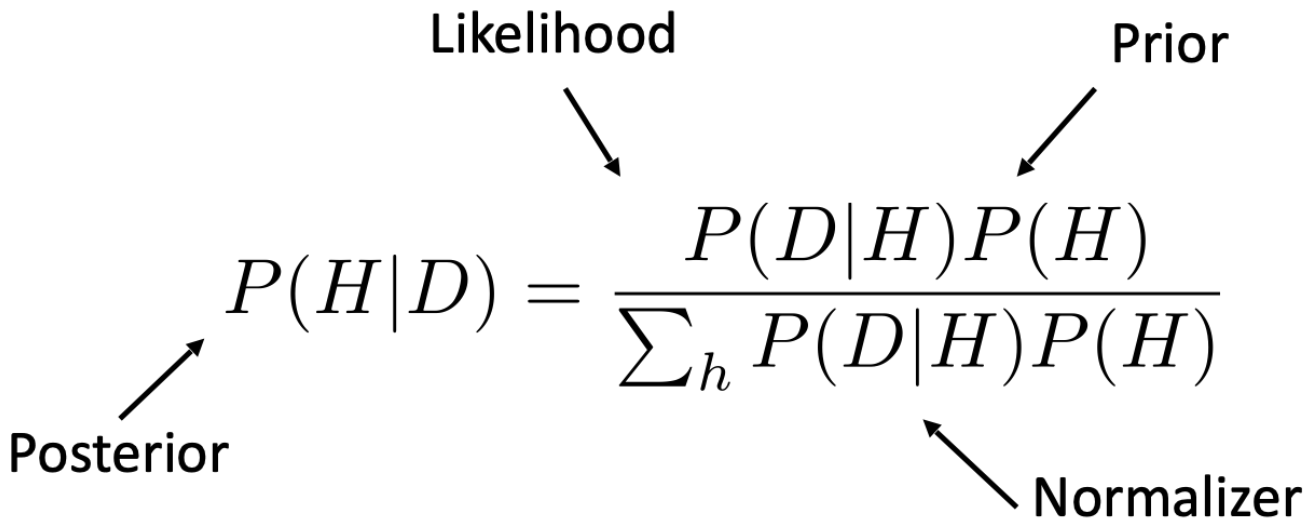
$$Gain(S, A) = Entropy(S) - \sum_{v \in values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

A= Attribute
values(A) = set of all possible values for A
|S| = number of example in S
$|S_v|$ =number of example in S where the value of A is v

Naïve Bayes

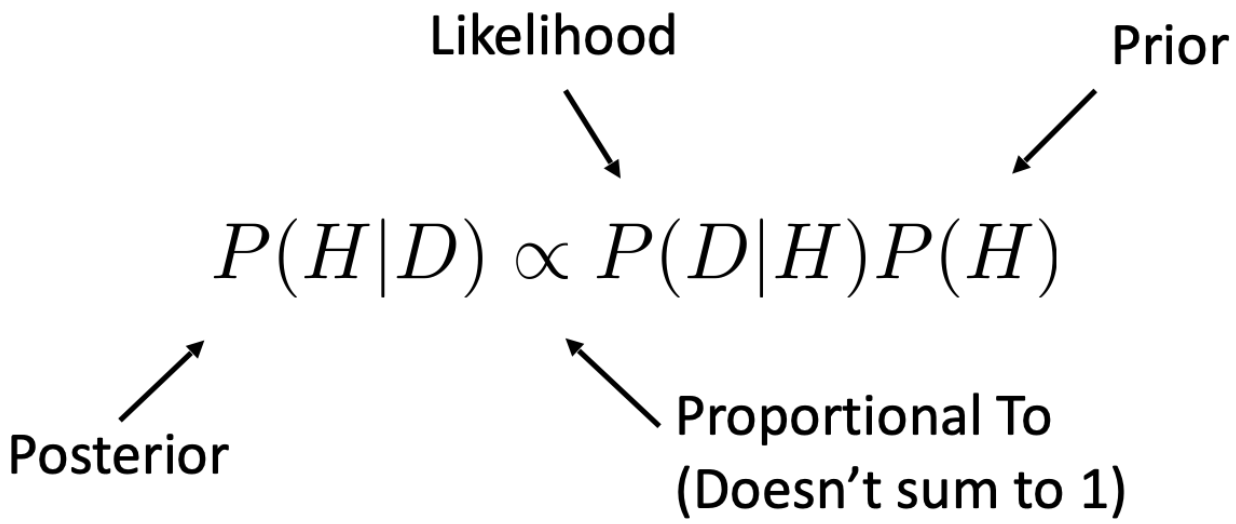
Bayes Rules

Bayes Rule tells us how to flip the conditional
Reason about effects to causes
Useful if you assume a generative model for your data



Bayes Rules

Bayes Rule tells us how to flip the conditional
Reason about effects to causes
Useful if you assume a generative model for your data



Learning: Naïve Bayes Classifier

- First attempt: maximum likelihood estimates
  - simply use the frequencies in the data

$$\hat{P}(c_j) = \frac{doccount(C = c_j)}{N_{doc}}$$

$$\hat{P}(w_i \mid c_j) = \frac{count(w_i, c_j)}{\sum_{w \in V} count(w, c_j)}$$

Smoothing: Naïve Bayes Parameter Estimation

$$\hat{P}(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w_i, c)}$$

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + \alpha}{\sum_{w \in V} count(w_i, c) + \alpha|V|}$$

Laplace Smoothing: Naïve Bayes Parameter Estimation

$$\hat{P}(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w_i, c)}$$

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + \alpha}{\sum_{w \in V} count(w_i, c) + \alpha|V|}$$

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w \in V} count(w_i, c) + |V|}$$

Naïve Bayes Parameter Learning: Step

- Calculate $P(c_j)$ terms
  - For each c_j in C do

    $docs_j \leftarrow$ all docs with class = c_j

    $$P(c_j) \leftarrow \frac{|docs_j|}{|\text{total \# documents}|}$$

- From training corpus, extract Vocabulary

- Calculate $P(w_k \mid c_j)$ terms
  - $\text{Text}_j \leftarrow$ single doc containing all sentences from class = c_j
  - For each word w_k in *Vocabulary*

    $n_k \leftarrow$ # of occurrences of w_k in $Text_j$

    $n \leftarrow$ # of words in class $Text_j$

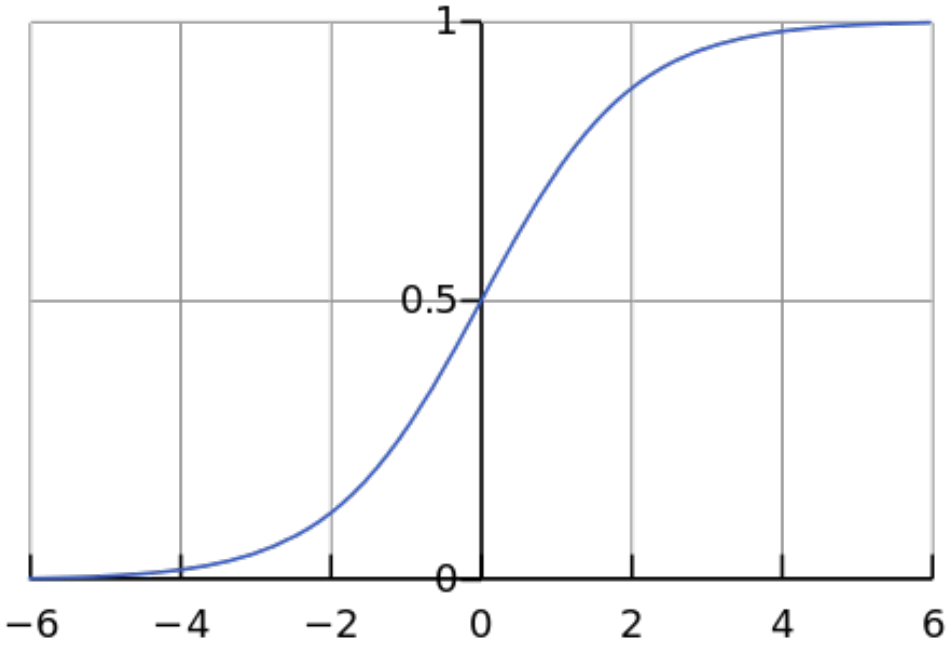
    $$P(w_k \mid c_j) \leftarrow \frac{n_k + \alpha}{n + \alpha\,|Vocabulary|}$$

Naïve Bayes Evaluation: Step

- # of correct prediction = 0
- For each t_i in Test data:
  - max_prob = 0
  - For each class c_j
    - Find p_j = probability of t_j to be in c_j
    - If pj > max_prob:
      - max_prob = p_j, max_prob_class= c_j
  - If max_prob_class == actual label of tj:
    - # of correct prediction += 1
- Accuracy = $\frac{\text{\# of correct prediction}}{\text{\# of test data}}$

Logistic Regression

The Logistic function



$$P(\text{spam}|x) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}}$$

The Dot Product

$$z = \sum_{i=0}^{|X|} w_i x_i$$

- Intuition: weighted sum of features
- All Linear models have this form

NB vs. LR

- Both compute the dot product
- NB: sum of log probabilities
- LR: logistic function

NB vs. LR:Parameter Learning

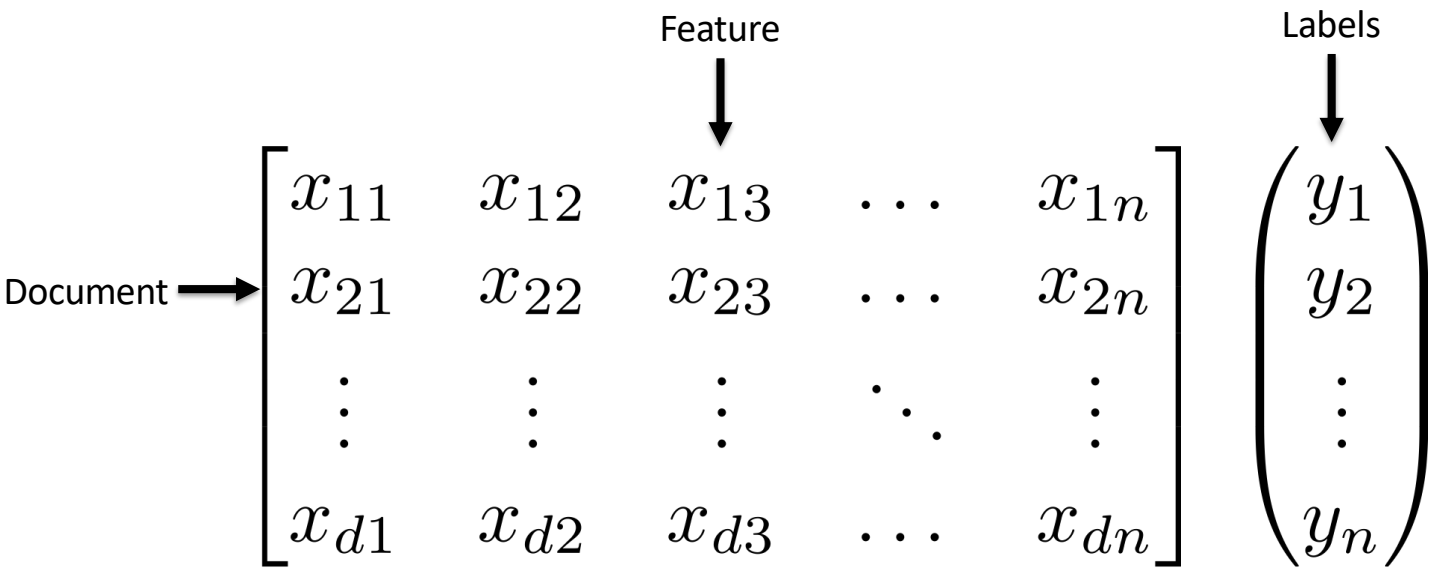
- Naïve Bayes:
  - Learn conditional probabilities **independently** by counting

- Logistic Regression:
  - Learn weights **jointly**

LR: Learning Weights

- Given: a set of feature vectors and labels
- Goal: learn the weights

Learning Weights



Q: what parameters should we choose?

- What is the right value for the weights?

- Maximum Likelihood Principle:
  - Pick the parameters that maximize the probability of the data

Maximum Likelihood Estimation

$$
\begin{aligned}
w_{\text{MLE}} &= \text{argmax}_w \log P(y_1, \ldots, y_d | x_1, \ldots, x_d; w) \\
&= \text{argmax}_w \sum_i \log P(y_i | x_i; w) \\
&= \text{argmax}_w \sum_i \log \begin{cases} p_i, & \text{if } y_i = 1 \\ 1 - p_i, & \text{if } y_i = 0 \end{cases} \\
&= \text{argmax}_w \sum_i \log p_i^{\mathbb{I}(y_i=1)} (1 - p_i)^{\mathbb{I}(y_i=0)}
\end{aligned}
$$

Maximum Likelihood Estimation

$$= \text{argmax}_w \sum_i \log p_i^{\mathbb{I}(y_i=1)} (1-p_i)^{\mathbb{I}(y_i=0)}$$

$$= \text{argmax}_w \sum_i y_i \log p_i + (1-y_i) \log(1-p_i)$$

Maximum Likelihood Estimation

- Unfortunately there is no closed form solution
  - (like there was with naïve bayes)
- Solution:
  - Iteratively climb the log-likelihood surface through the derivatives for each weight

Derivative of Sigmoid

$$\frac{ds(x)}{dx} = \frac{d}{dx}\frac{1}{1+e^{-x}}$$

$$= -\left(\frac{1}{1+e^{-x}}\right)^2 \frac{d}{dx}(1+e^{-x})$$

$$= -\left(\frac{1}{1+e^{-x}}\right)^2 e^{-x}(-1)$$

$$= s(x)(1-s(x))$$

Gradient ascent

Loop While not converged:

For all features **j**, compute and add derivatives

$$w_j^{\text{new}} = w_j^{\text{old}} + \eta \frac{\partial}{\partial w_j} \mathcal{L}(w)$$

$\mathcal{L}(w)$: Training set log-likelihood

$\left(\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \ldots, \frac{\partial \mathcal{L}}{\partial w_n} \right)$: Gradient vector

Gradient ascent

Loop While not converged:

For all features **j**, compute and add derivatives

$$w_j^{\text{new}} = w_j^{\text{old}} + \eta \frac{\partial}{\partial w_j} \mathcal{L}(w)$$

$\mathcal{L}(w)$: Training set log-likelihood $\quad \frac{\partial \mathcal{L}}{\partial w_j} = \sum_i (y_i - p_i) x_j$

$\left(\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \ldots, \frac{\partial \mathcal{L}}{\partial w_n} \right)$: Gradient vector

LR Gradient

$$\frac{\partial \mathcal{L}}{\partial w_j} = \sum_i (y_i - p_i) x_j$$

j -> iterating over features
i -> iterating over training examples

Y_i = Ture label
P_i = Predicted Label

MultiClass Classification

- Q: what if we have more than 2 categories?
  - Sentiment: Positive, Negative, Neutral
  - Document topics: Sports, Politics, Business, Entertainment, …

MultiClass Classification

- Q: what if we have more than 2 categories?
  - Sentiment: Positive, Negative, Neutral
  - Document topics: Sports, Politics, Business, Entertainment, …
- Could train a separate logistic regression model for each category...
- Pretty clear what to do with Naive Bayes.

Log-Linear Models

$$P(y|x) \propto e^{w \cdot f(d,y)}$$

$$P(y|x) = \frac{1}{Z(w)} e^{w \cdot f(d,y)}$$

MultiClass Logistic Regression

$$P(y|x) \propto e^{w \cdot f(d,y)}$$

$$P(y|x) = \frac{1}{Z(w)} e^{w \cdot f(d,y)}$$

$$P(y|x) = \frac{e^{w \cdot f(d,y)}}{\sum_{y' \in Y} e^{w \cdot f(d,y')}}$$

MultiClass Logistic Regression

- Binary logistic regression:
  - We have one feature vector that matches the size of the vocabulary
- Multiclass in practice:
  - one weight vector for each category

$$w_{\text{pos}} \qquad w_{\text{neg}} \qquad w_{\text{neut}}$$

MultiClass Logistic Regression

- Binary logistic regression:
  - We have one feature vector that matches the size of the vocabulary
- Multiclass in practice:
  - one weight v

Can represent this in practice with one giant weight vector and repeated features for each category.

w_{pos} w_{neg} w_{neut}

Maximum Likelihood Estimation

$$w_{\text{MLE}} = \text{argmax}_w \log P(y_1, \ldots, y_n | x_1, \ldots, x_n; w)$$

$$= \text{argmax}_w \sum_i \log P(y_i | x_i; w)$$

$$= \text{argmax}_w \sum_i \log \frac{e^{w \cdot f(x_i, y_i)}}{\sum_{y' \in Y} e^{w \cdot f(x_i, y_i)}}$$

Multiclass Leaning

LR : $$\frac{\partial \mathcal{L}}{\partial w_j} = \sum_{i=1}^{D} f_j(y_i, d_i) - \sum_{i=1}^{D} \sum_{y \in Y} f_j(y, d_i) P(y|d_i)$$

Perceptron

Perceptron Algorithm

- Algorithm is Very similar to logistic regression
- Not exactly computing gradients

Perceptron Algorithm

- Algorithm is Very similar to logistic regression
- Not exactly computing gradients

Initialize weight vector $w = 0$

Loop for K iterations

  Loop For all training examples x_i

    $y_i' = \text{sign}(w * x_i)$

    if $y_i' != y_i$

      $w += (y_i - y_i') * x_i$

Perceptron Notes

- Guaranteed to converge if the data is linearly separable
- Only hyperparameter is maximum number of iterations
- Parameter averaging

Differences: LR vs. Perceptron

- Batch Learning vs. Online learning

- Perceptron doesn’t always make updates

Online Learning (perceptron)

- Rather than making a full pass through the data, compute gradient and update parameters after each training example.
- Gradients will be less accurate, but the overall effect is to move in the right direction
- Often works well and converges faster than batch learning

Multiclass Leaning

Perceptron : $$\frac{\partial \mathcal{L}}{\partial w_j} = \sum_{i=1}^{D} f_j(y_i, d_i) - \sum_{i=1}^{D} f_j(\arg\max_{y \in Y} P(y|d_i), d_i)$$

Multiclass Leaning

LR : $$\frac{\partial \mathcal{L}}{\partial w_j} = \sum_{i=1}^{D} f_j(y_i, d_i) - \sum_{i=1}^{D} \sum_{y \in Y} f_j(y, d_i) P(y|d_i)$$

Perceptron : $$\frac{\partial \mathcal{L}}{\partial w_j} = \sum_{i=1}^{D} f_j(y_i, d_i) - \sum_{i=1}^{D} f_j(\arg\max_{y \in Y} P(y|d_i), d_i)$$

Multiclass Leaning

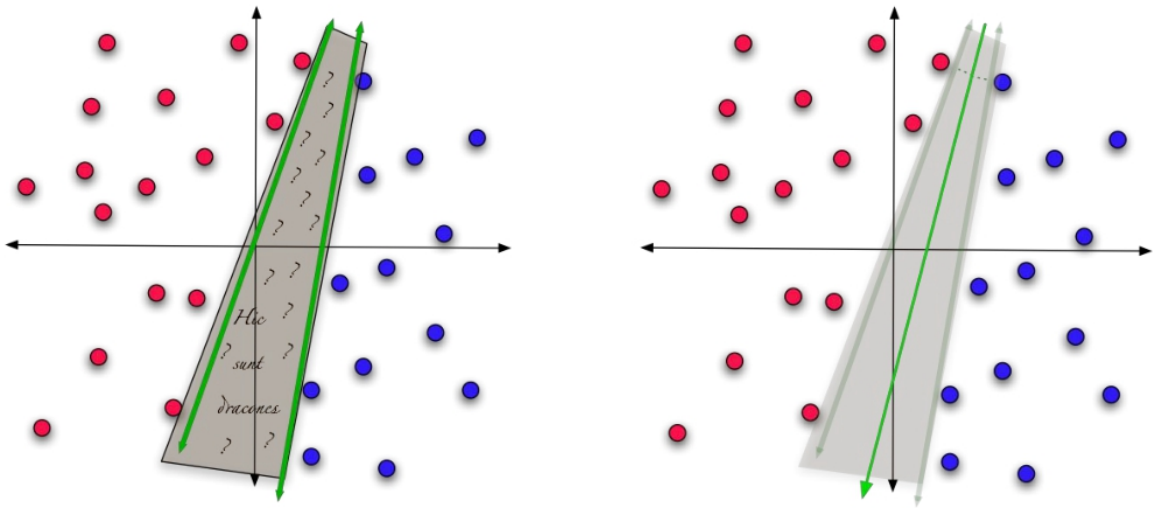
LR : $$\frac{\partial \mathcal{L}}{\partial w_j} = \sum_{i=1}^{D} f_j(y_i, d_i) - \sum_{i=1}^{D} \sum_{y \in Y} f_j(y, d_i) P(y|d_i)$$

Perceptron : $$\frac{\partial \mathcal{L}}{\partial w_j} = \sum_{i=1}^{D} f_j(y_i, d_i) - \sum_{i=1}^{D} f_j(\arg\max_{y \in Y} P(y|d_i), d_i)$$

SVM

Margin

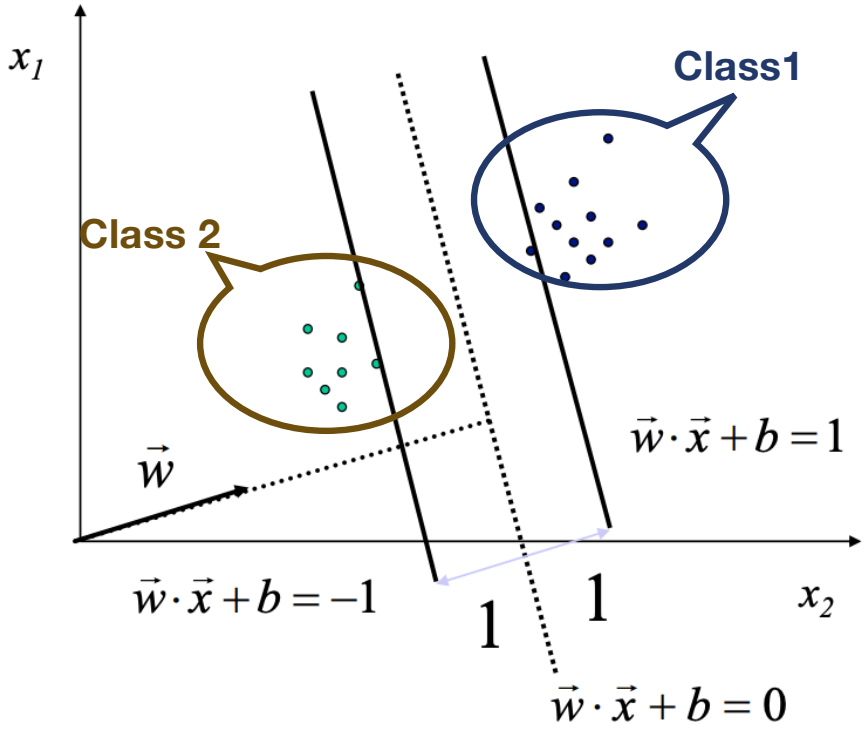
- the distance to closest point in the training data
- We tend to get better generalization to **unseen data** if we choose the separating hyperplane which *maximizes the margin*



Support Vector Machines

- A learning method which *explicitly calculates the maximum margin hyperplane* by solving a gigantic quadratic programming minimization problem.
- Among the very highest -performing traditional machine learning techniques.
- But it's relatively slow and quite complicated.

Setting Up the Optimization Problem



The maximum margin can be characterized as a solution to an optimization problem:

$$\max. \frac{2}{\|w\|}$$

$$s.t.\ (w\cdot x+b)\geq 1,\ \forall x \text{ of class 1}$$

$$(w\cdot x+b)\leq -1,\ \forall x \text{ of class 2}$$

Define the margin (what ever it turns out to be) to be *one unit of width.*

Setting Up the Optimization Problem

- **If class 1 corresponds to 1 and class 2 corresponds to -1, we can rewrite**

$$(w \cdot x_i + b) \geq 1, \ \forall x_i \ \text{with} \ y_i = 1$$
$$(w \cdot x_i + b) \leq -1, \ \forall x_i \ \text{with} \ y_i = -1$$

- **as**

$$y_i (w \cdot x_i + b) \geq 1, \ \forall x_i$$

- **So the problem becomes:**

$$\max. \ \frac{2}{\|w\|} \quad \text{or} \quad \min. \ \frac{1}{2}\|w\|^2$$
$$s.t. \ y_i (w \cdot x_i + b) \geq 1, \ \forall x_i \qquad s.t. \ y_i (w \cdot x_i + b) \geq 1, \ \forall x_i$$

Linear, (Hard-Margin) SVM Formulation

- **Find *w,b* that solves**

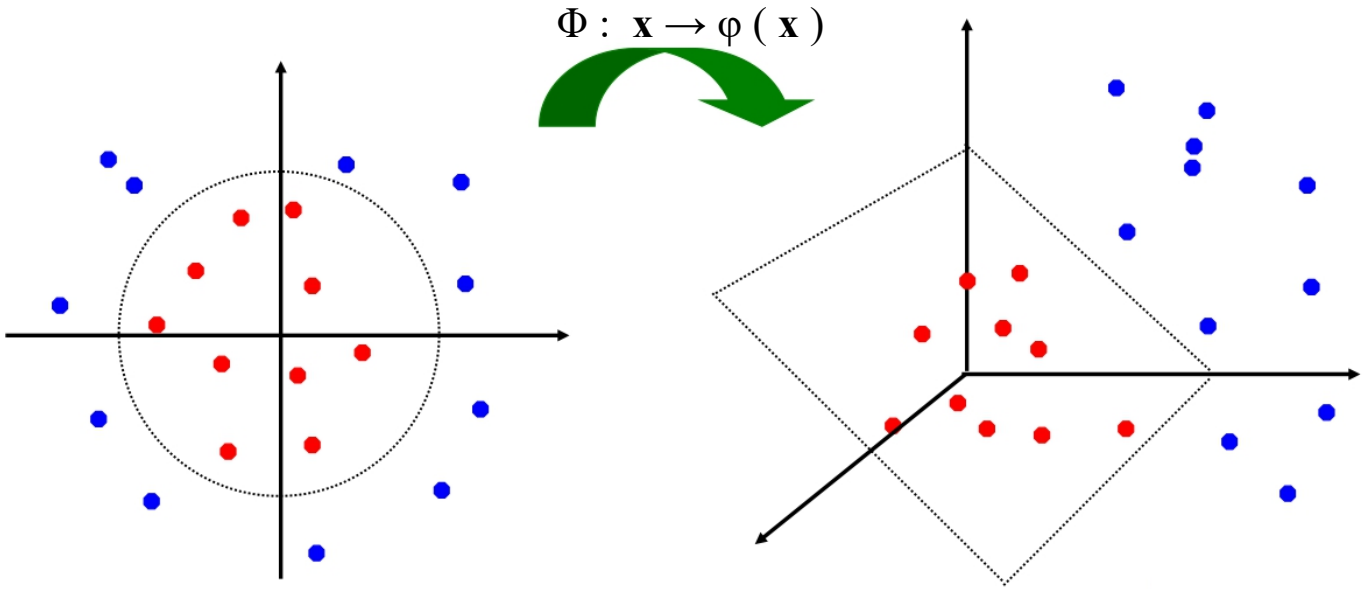
$$\min.\ \frac{1}{2}\|w\|^2$$

$$s.t.\ y_i(w \cdot x_i + b) \geq 1,\ \forall x_i$$

- **Problem is *convex*, so there is a unique global minimum value (when feasible)**
- **There is also a unique minimizer, i.e. weight and *b* value that provides the minimum**
- **Quadratic Programming**
  - very efficient computationally with procedures that take advantage of the special structure

Non - linear SVMs: Feature spaces

- General idea: the original feature space can *always* be mapped to some *higher - dimensional* feature space where the training set is *linearly* separable:



Kernel Trick

- **If our data isn't linearly separable, we can define a projection $\Phi(x_i)$ to map it into a much higher dimensional feature space where it is.**

- **For SVM where everything can be expressed as the dot products of instances this can be done efficiently using the `kernel trick':**
  - A kernel K is a function such that: $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$
  - Then, we never need to explicitly map the data into the high-dimensional space to solve the optimization problem – magic!!

Non-Linear Regression

Non-linear regression

- For many cases when $f(x; w)$ is not linear w.r.t. x, $f(x; w)$ is not linear w.r.t. w
- Examples:
  - $f(x; w) = e^{wx}$
  - $f(x; w) = \sin(wx)$

- For non-linear regression, $\boldsymbol{x}$ and $\boldsymbol{w}$ very likely have different dimensionality
- Example:
  - $f(x; w) = e^{w[1]x} + \sin(w[2]x)$

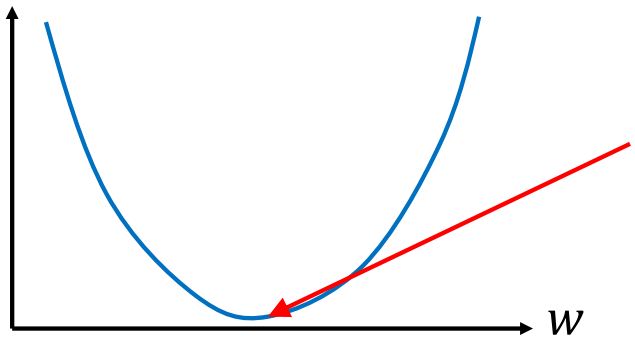
- We will say $\boldsymbol{x}$ is D-dimensional and $\boldsymbol{w}$ is K-dimensional

When $f(x; w)$ is nonlinear w.r.t. w

- Squared-error function is no longer parabolic w.r.t. the parameters
  - Can not assume local minimums of error functions are always global minimums

- Often no closed-form solution
  - Need numerical approximation

$$E(w) = \sum_i (wx_i - y_i)^2$$



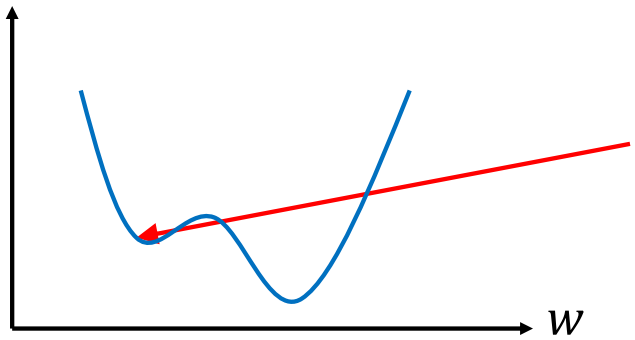
- $\frac{\partial E}{\partial w} = 0$ means global minimums
- Closed-forms to find w^* for $\frac{\partial E}{\partial w} = 0$

When $f(x; w)$ is nonlinear w.r.t. w

- Squared-error function is no longer parabolic w.r.t. the parameters
  - Can not assume local minimums of error functions are always global minimums

- Often no closed-form solution
  - Need numerical approximation

$$E(w) = \sum_i (f(x_i; w) - y_i)^2$$



- $\frac{\partial E}{\partial w} = 0$ does not mean global minimums
- No closed-forms to find w^* for $\frac{\partial E}{\partial w} = 0$

Gauss-Newton Algorithm

1. Get initial guess for parameters $\widehat{w}^{(1)}$
2. For t=1…∞ (or until we decide to stop)
   1. Find error for current guess ($\widehat{w}^{(t)}$)
      $$\Delta y_i = y_i - f(\mathbf{x}_i; \widehat{w}^{(t)})$$
   2. Find Jacobian around current guess
      $$J_i = \frac{df(\mathbf{x}_i; \widehat{w}^{(t)})}{dw}$$
   3. Calculate change in guess
      $$\Delta \boldsymbol{w} = (\boldsymbol{J}^T \boldsymbol{J})^{-1} \boldsymbol{J}^T \Delta \boldsymbol{y}$$
   4. Make new guess
      $$\widehat{w}^{(t+1)} = \widehat{w}^{(t)} + \Delta \widehat{w}$$

Gauss-Newton Algorithm (let $\boldsymbol{w}$ be K-dim)

1. Get initial guess for parameters $\widehat{\boldsymbol{w}}^{(1)}$
2. For t=1…∞ (or until we decide to stop)
   1. Find error for current guess ($\widehat{\boldsymbol{w}}^{(t)}$)
      $$\Delta y_i = y_i - f(\mathbf{x}_i; \widehat{\boldsymbol{w}}^{(t)})$$
   2. Find Jacobian around current guess
      $$J_{ik} = \frac{\partial f(\mathbf{x}_i; \widehat{\boldsymbol{w}}^{(t)})}{\partial w[k]}$$
   3. Calculate change in guess
      $$\Delta \boldsymbol{w} = (\boldsymbol{J}^T \boldsymbol{J})^{-1} \boldsymbol{J}^T \Delta \boldsymbol{y}$$
   4. Make new guess
      $$\widehat{\boldsymbol{w}}^{(t+1)} = \widehat{\boldsymbol{w}}^{(t)} + \Delta \widehat{\boldsymbol{w}}$$

Gradient Descent (GD)

- Simple (and faster?) method for optimization
  - $f(\boldsymbol{x}; \boldsymbol{w})$ linear or nonlinear w.r.t. parameters $\boldsymbol{w}$
  - Sum or square losses or other losses
- A general method to find local minimum for a multi-parameter error function
  - Search through "parameter space" to find minimum error
- Main idea
  - Evaluate error function for current guess of parameters
  - Determine change in parameters that decrease error
    - From gradient of error function
  - Update parameters in this new change
  - Repeat process until converge (or hit max iterations)

Algorithm

Initialize parameters: $\widehat{\boldsymbol{w}}^{(1)} = [w[1], w[2], \ldots, w[K]]^T$

Error function: $E(\boldsymbol{w})$

Compute gradients: $\nabla E\left(\widehat{\boldsymbol{w}}^{(t)}\right) = \begin{bmatrix} \frac{\partial E\left(\widehat{\boldsymbol{w}}^{(t)}\right)}{\partial w[1]} \\ \vdots \\ \frac{\partial E\left(\widehat{\boldsymbol{w}}^{(t)}\right)}{\partial w[K]} \end{bmatrix}$

Update parameters: $\widehat{\boldsymbol{w}}^{(t+1)} = \widehat{\boldsymbol{w}}^{(t)} - \eta \cdot \nabla E\left(\widehat{\boldsymbol{w}}^{(t)}\right)$

Learning rate

[Repeat until converges]

Compute Gradients: Least Squares (w is 1-dim)

Error function: $$E(\boldsymbol{w}) = \sum_i [f(\boldsymbol{x_i}; \boldsymbol{w}) - y_i]^2$$

Error gradient: $$\nabla E(\boldsymbol{w})$$

$$\frac{dE(\widehat{\boldsymbol{w}}^{(t)})}{dw} = \frac{\partial}{\partial w} \sum_i [f(\boldsymbol{x_i}; \widehat{\boldsymbol{w}}^{(t)}) - y_i]^2$$

$$= 2 \sum_i (f(\boldsymbol{x_i}; \widehat{\boldsymbol{w}}^{(t)}) - y_i) \frac{df(\boldsymbol{x_i}; \widehat{\boldsymbol{w}}^{(t)})}{dw}$$

Compute Gradients: Least Squares ($\boldsymbol{w}$ is K-dim)

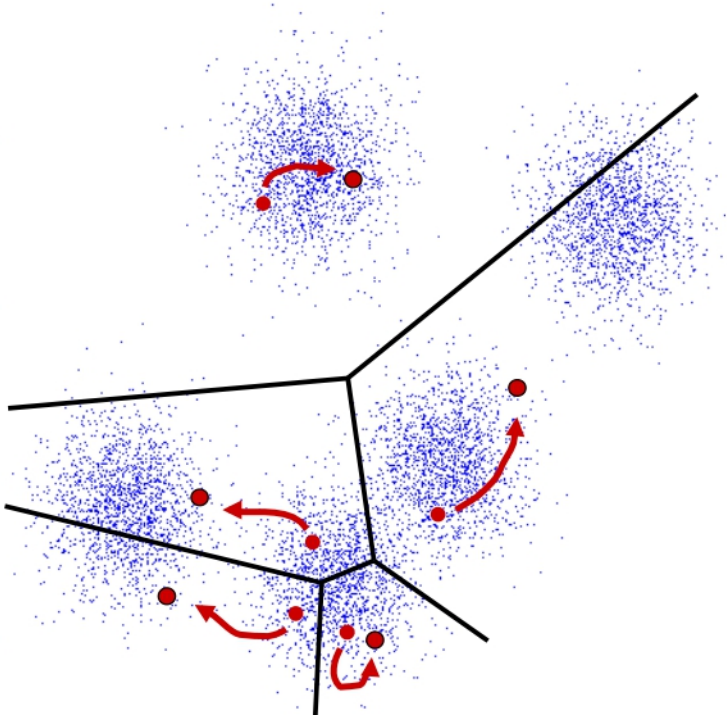
Error function: $$E(\boldsymbol{w}) = \sum_i [f(\boldsymbol{x_i}; \boldsymbol{w}) - y_i]^2$$

Error gradient: $$\nabla E(\boldsymbol{w})$$

$$\frac{\partial E(\widehat{\boldsymbol{w}}^{(t)})}{\partial w[k]} = \frac{\partial}{\partial w[k]} \sum_i \left[f\left(\boldsymbol{x_i}; \widehat{\boldsymbol{w}}^{(t)}\right) - y_i\right]^2$$

$$= 2 \sum_i \left(f\left(\boldsymbol{x_i}; \widehat{\boldsymbol{w}}^{(t)}\right) - y_i\right) \frac{\partial f\left(\boldsymbol{x_i}; \widehat{\boldsymbol{w}}^{(t)}\right)}{\partial w[k]}$$

Clustering



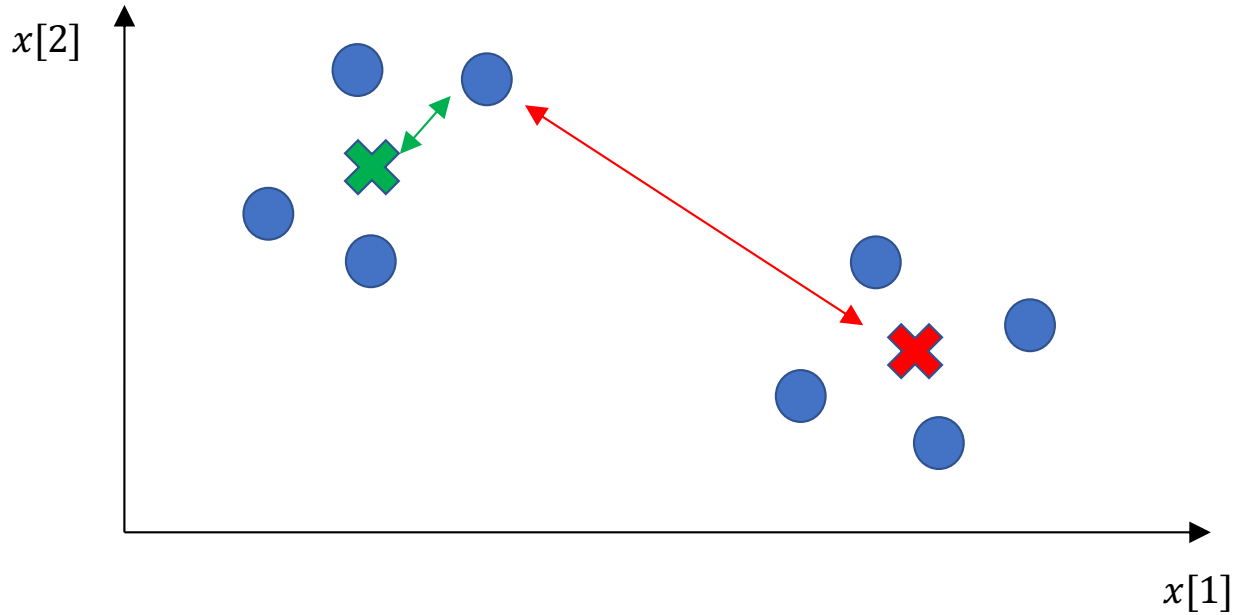
THE OHIO STATE UNIVERSITY

Clustering

- Clustering systems:
  - Unsupervised learning
  - Detect patterns in unlabeled data
    - E.g. group emails or search results
    - E.g. find categories of customers
    - E.g. detect anomalous program executions
  - Useful when we don’t know what you are looking for
    - Requires data, but no labels
    - May get gibberish

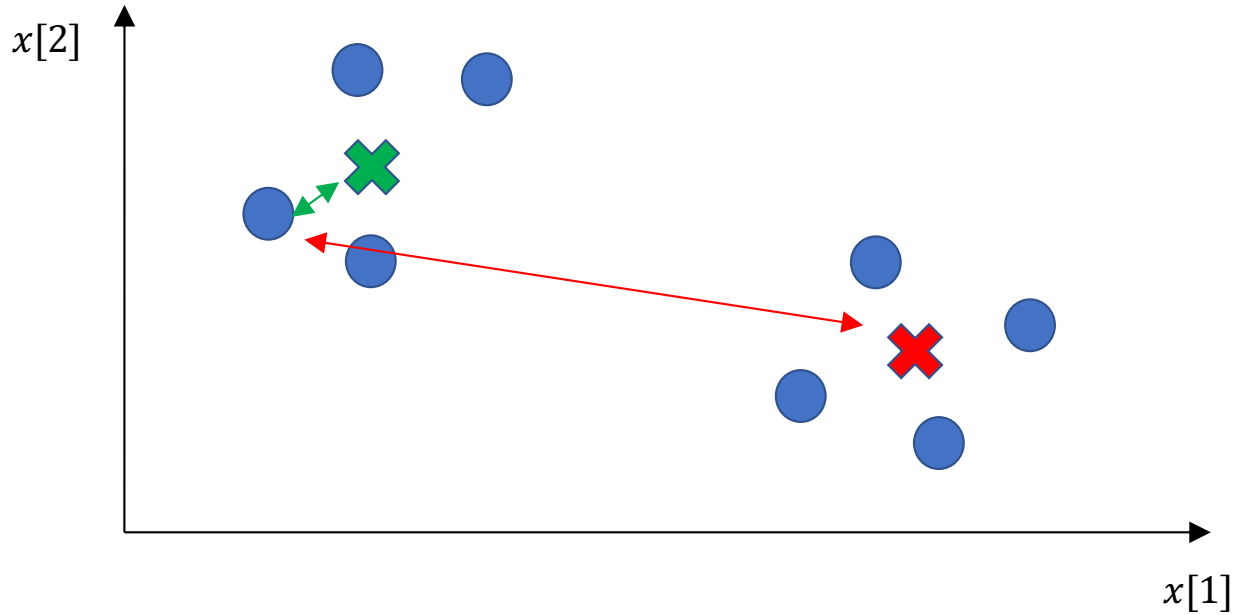
K-Means

- Find K centers and group data instances into the K centers
  - Example: K = 2
  - Assign each data instance to the closet (most similar, smallest distant) center



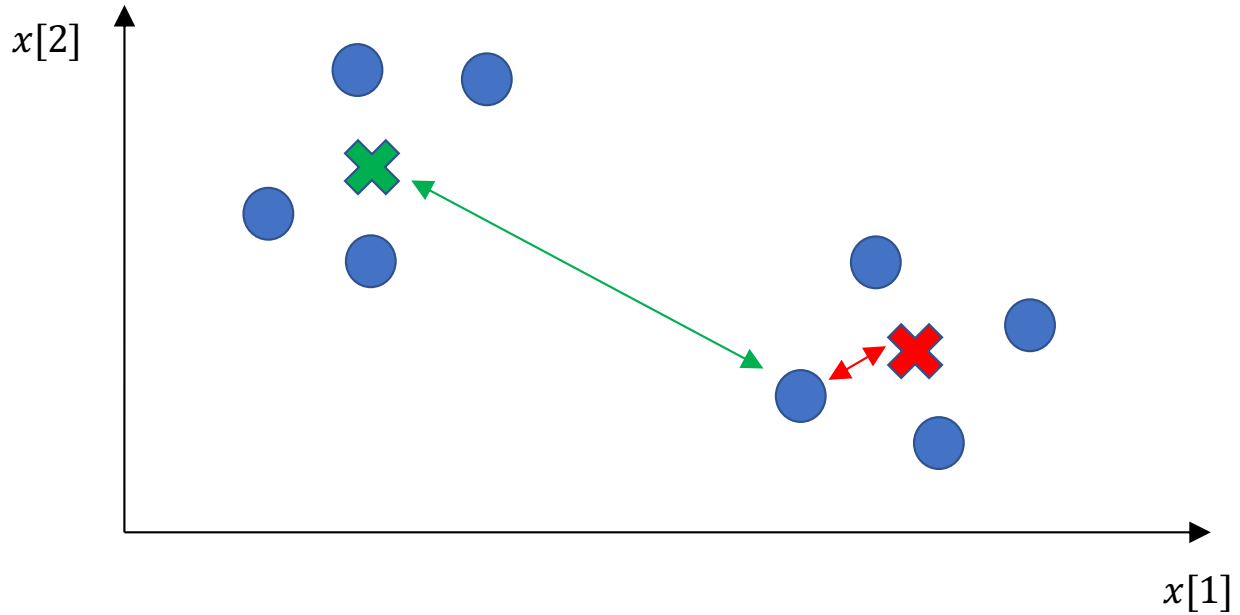
K-Means

- Find K centers and group data instances into the K centers
  - Example: K = 2
  - Assign each data instance to the closet (most similar, smallest distant) center



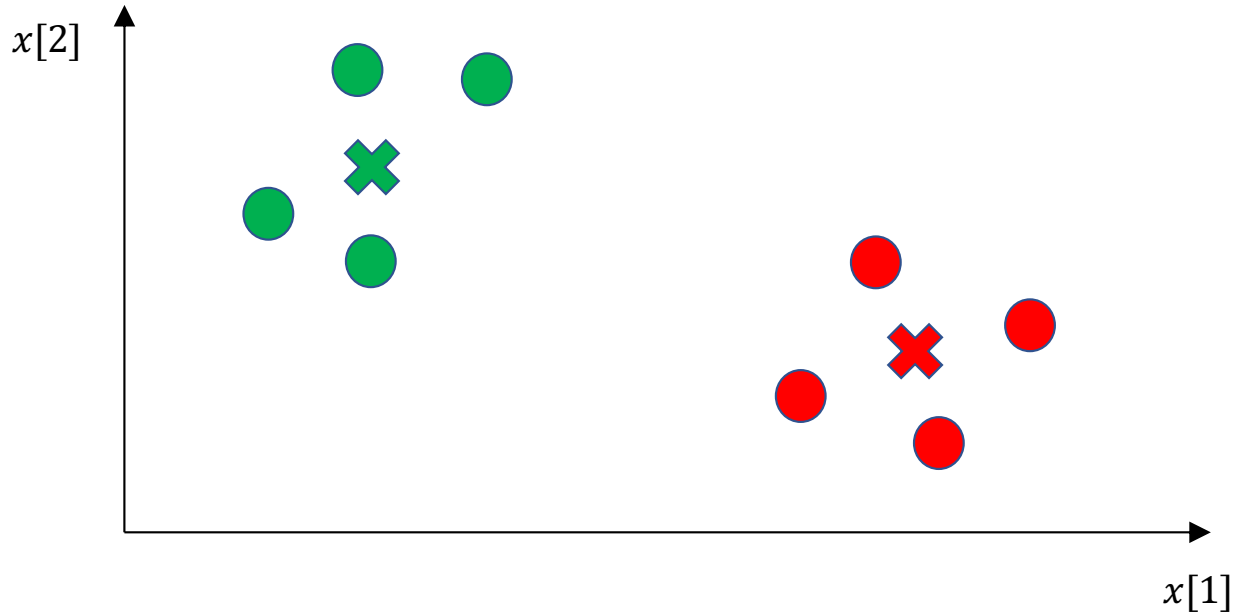
K-Means

- Find K centers and group data instances into the K centers
  - Example: K = 2
  - Assign each data instance to the closet (most similar, smallest distant) center



K-Means

- Find K centers and group data instances into the K centers
  - Example: K = 2
  - Assign each data instance to the closet (most similar, smallest distant) center



K-Means

- Formulate as an optimization problem (total distance to the centers)

$$\underset{\{y_i \in \{1,\ldots,K\}\}_{i=1}^{N},\ \{\boldsymbol{c}_k \in \mathbb{R}^D\}_{k=1}^{K}}{\operatorname{argmin}} \sum_{k=1}^{K} \sum_{\{i \mid y_i = k\}} \|\boldsymbol{x}_i - \boldsymbol{c}_k\|_2^2$$

All data instances assigned to group k

  - To find centers (*K* of them) and group assignments (*N* data instances) simultaneously

K-Means

- Formulate as an optimization problem (total distance to the centers)

$$\underset{\{y_i \in \{1,\ldots,K\}\}_{i=1}^{N},\, \{\boldsymbol{c}_k \in \mathbb{R}^D\}_{k=1}^{K}}{\operatorname{argmin}} \sum_{k=1}^{K} \sum_{\{i \mid y_i = k\}} \|\boldsymbol{x}_i - \boldsymbol{c}_k\|_2^2$$

All data instances assigned to group k

  - To find centers (*K* of them) and group assignments (*N* data instances) simultaneously
  - Given centers, then for each data point *i*, the assignment is independent

$$\underset{y_i \in \{1,\ldots,K\}}{\operatorname{argmin}} \sum_{k=1}^{K} \sum_{i \mid y_i = k} \|\boldsymbol{x}_i - \boldsymbol{c}_k\|_2^2 \Rightarrow y_i = \underset{k}{\operatorname{argmin}} \|\boldsymbol{x}_i - \boldsymbol{c}_k\|_2^2$$

K-Means

- Formulate as an optimization problem (total distance to the centers)

$$\underset{\{y_i \in \{1,\ldots,K\}\}_{i=1}^{N},\ \{\boldsymbol{c}_k \in \mathbb{R}^D\}_{k=1}^{K}}{\operatorname{argmin}} \sum_{k=1}^{K} \sum_{\{i \mid y_i = k\}} \|\boldsymbol{x}_i - \boldsymbol{c}_k\|_2^2$$

All data instances assigned to group k

  - To find centers (*K* of them) and group assignments (*N* data instances) simultaneously
  - Given assignments, then finding the center for each group *k* is independent

$$\underset{\boldsymbol{c}_k \in \mathbb{R}^D}{\operatorname{argmin}} \sum_{\{i \mid y_i = k\}} \|\boldsymbol{x}_i - \boldsymbol{c}_k\|_2^2 \implies \boldsymbol{c}_k = \frac{1}{|\{i \mid y_i = k\}|} \sum_{\{i \mid y_i = k\}} \boldsymbol{x}_i$$

K-Means: an iterative clustering algorithm

- Pick *K* random points as cluster centers (means): $c_1 \ldots c_K$
- Alternate (the error is monotonically decreasing):
  - Assign each example x_i to the mean c_k that is closest to it

$$\underset{y_i \in \{1,\ldots,K\}}{\operatorname{argmin}} \sum_{k=1}^{K} \sum_{i|y_i=k} \|\boldsymbol{x}_i - \boldsymbol{c}_k\|_2^2 \Rightarrow y_i = \underset{k}{\operatorname{argmin}} \|\boldsymbol{x}_i - \boldsymbol{c}_k\|_2^2$$

  - Set each mean c_k to the average of its assigned points

$$\underset{\boldsymbol{c}_k \in \mathbb{R}^D}{\operatorname{argmin}} \sum_{\{i \mid y_i=k\}} \|\boldsymbol{x}_i - \boldsymbol{c}_k\|_2^2 \Rightarrow \boldsymbol{c}_k = \frac{1}{|\{i \mid y_i = k\}|} \sum_{\{i \mid y_i=k\}} \boldsymbol{x}_i$$

  - Stop when no points assignments change

K-Means

- Data: $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$
- An iterative clustering algorithm
  - Pick K random cluster centers, $c_1 \ldots c_K$
  - For t=1 … T: [or, stop if assignments don t change]
    - for i = 1 ... N: [update cluster assignments]

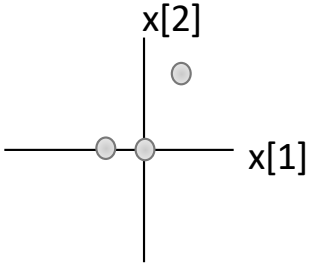
$$y_i = \underset{k}{\operatorname{argmin}} \|\boldsymbol{x}_i - \boldsymbol{c}_k\|_2^2$$

    - for k= 1 … K: [update cluster centers]

$$\boldsymbol{c}_k = \frac{1}{|\{i \mid y_i = k\}|} \sum_{\{i \mid y_i = k\}} \boldsymbol{x}_i$$

K-Means

	x[1]	x[2]
i=1	-1	0
i=2	0	0
i=3	2	2



Random cluster means:
c_1=[-1,0], c_2=[0,0]
t = 0:

	i=1	i=2	i=3
c_1	0	1	13
c_2	1	0	8

- $y_1 = \text{argmin}_k \text{ dist}(x_1,c_k) = 1$
- $y_2 = \text{argmin}_k \text{ dist}(x_2,c_k) = 1$
- $y_3 = \text{argmin}_k \text{ dist}(x_3,c_k) = 2$

- $c_1 = (1/1) * [-1,0] = [-1,0]$
- $c_2 = ((1/2) * ([0,0]+[2,2]) = [1,1]$

c_1=[-1,0], c_2=[1,1]
t = 1 :

	i=1	i=2	i=3
c_1	0	1	13
c_2	4	4	18

- $y_1 = \text{argmin}_k \text{ dist}(x_1,c_k) = 1$
- $y_2 = \text{argmin}_k \text{ dist}(x_2,c_k) = 1$
- $y_3 = \text{argmin}_k \text{ dist}(x_3,c_k) = 2$

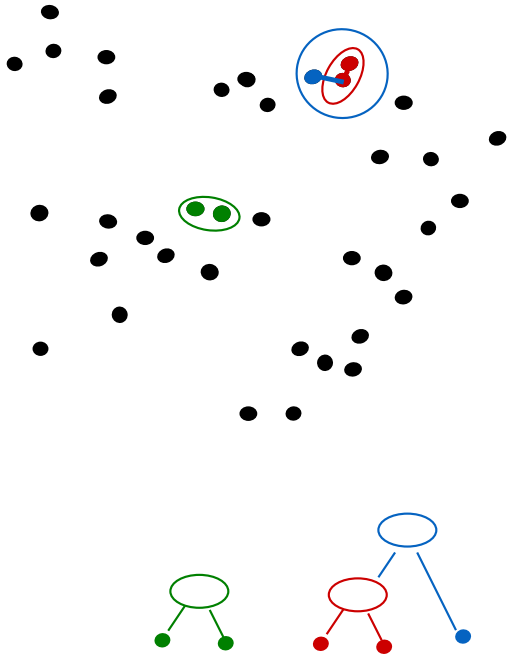
- $c_1 = (1/2) * ([-1,0]+[0,0]) = [-0.5,0]$
- $c_2 = (1/1) * ([2,2]) = [2,1]$

- t=2:
- Stop!!
- y_i won't change (verify!)

Agglomerative Clustering

- Agglomerative clustering:
  - First merge very similar instances
  - Incrementally build larger clusters out of smaller clusters

- Algorithm:
  - Maintain a set of clusters
  - Initially, each instance in its own cluster
  - Repeat:
    - Pick the two closest clusters
    - Merge them into a new cluster
    - Stop when there is only one cluster left



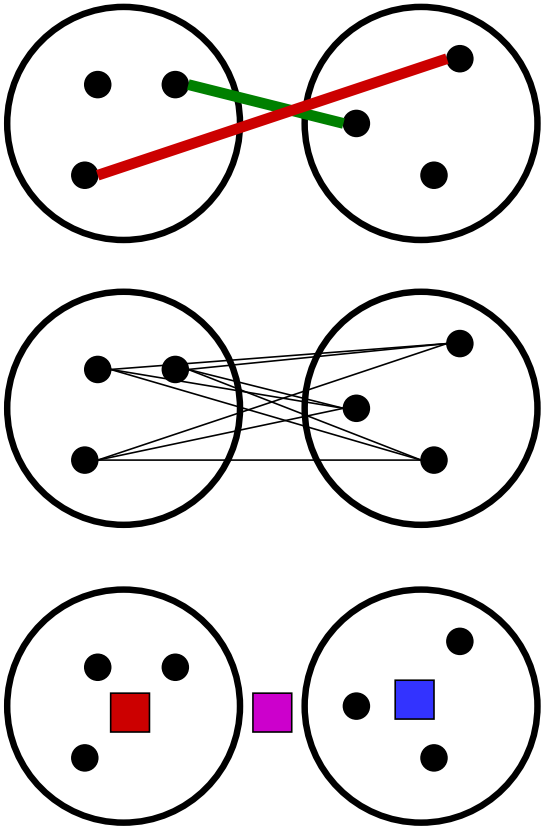
- Produces not one clustering, but a family of clusters represented by a dendrogram

Agglomerative Clustering

- How should we define “closest” for clusters with multiple elements?

- Many options
  - Closest pair (single-link clustering)
  - Farthest pair (complete-link clustering)
  - Average of all pairs
  - Ward’s method (min variance, like k-means)

- Different choices create different clustering behaviors



Agglomerative Clustering

complete link

single link

Each discrete value alone the x-axis is treated as one data instance.

Agglomerative Clustering

complete link

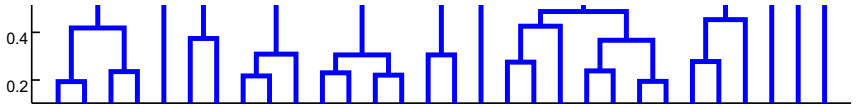
single link

Each discrete value alone the x-axis is treated as one data instance.

Agglomerative Clustering

complete link

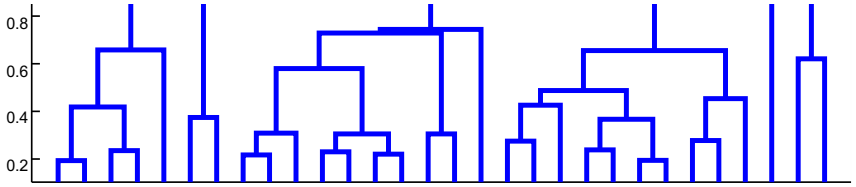
single link



Each discrete value alone the x-axis is treated as one data instance.

Agglomerative Clustering

complete link

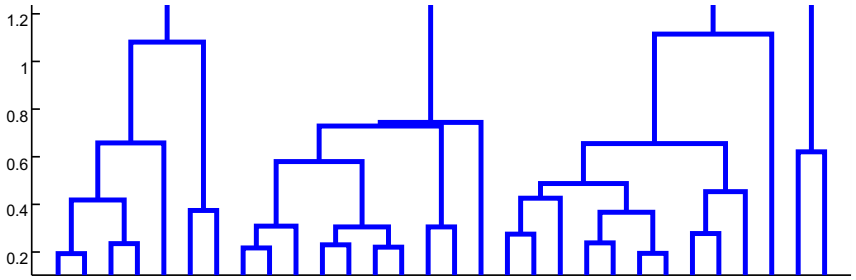


single link

Each discrete value alone the x-axis is treated as one data instance.

Agglomerative Clustering

complete link

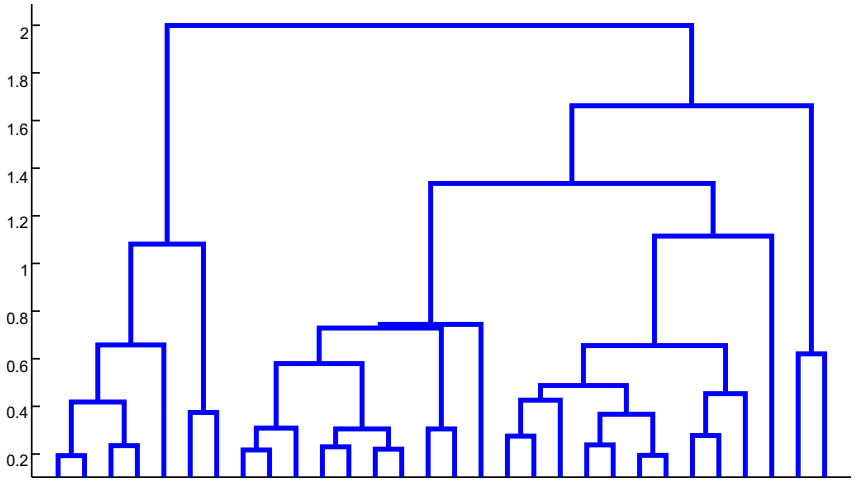


single link

Each discrete value alone the x-axis is treated as one data instance.

Agglomerative Clustering

complete link

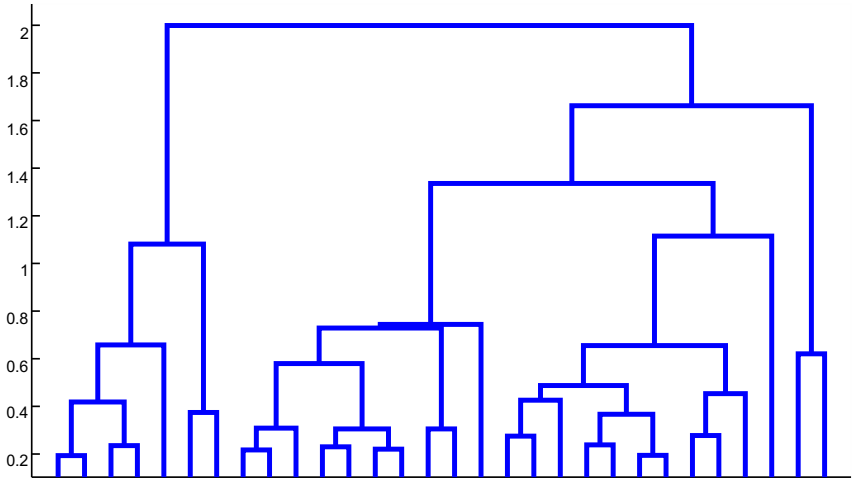


single link

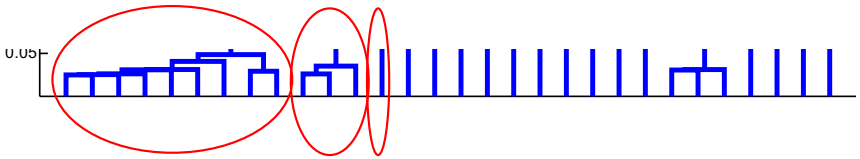
Each discrete value alone the x-axis is treated as one data instance.

Agglomerative Clustering

complete link



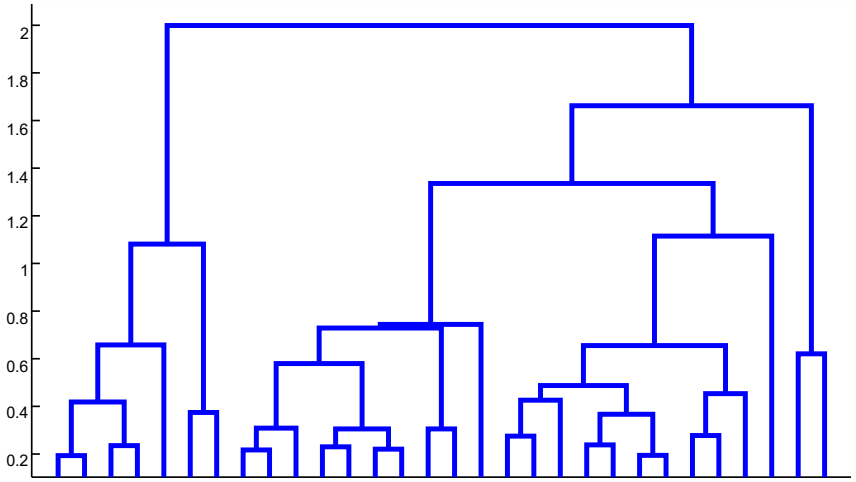
single link



Each discrete value alone the x-axis is treated as one data instance.

Agglomerative Clustering

complete link

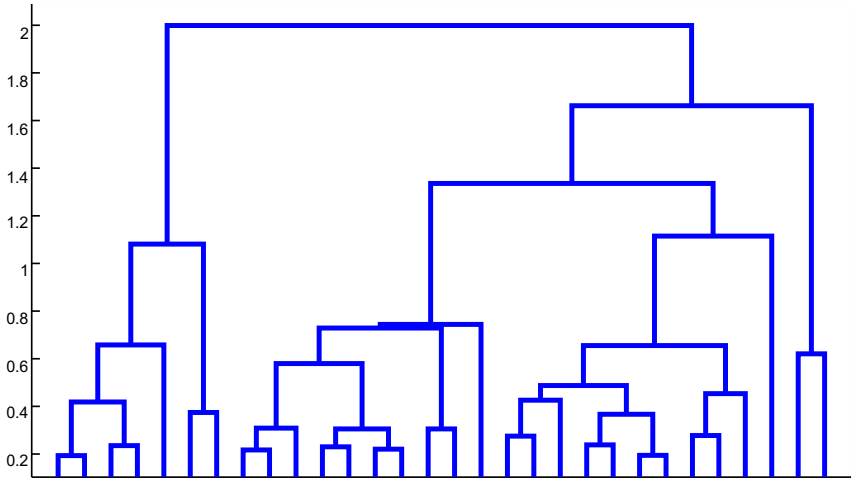


single link

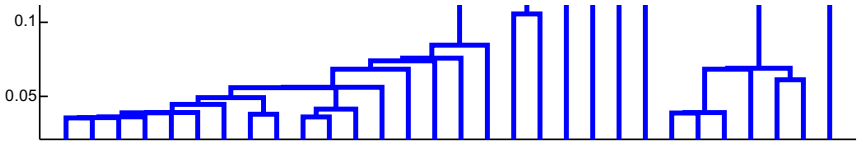
Each discrete value alone the x-axis is treated as one data instance.

Agglomerative Clustering

complete link



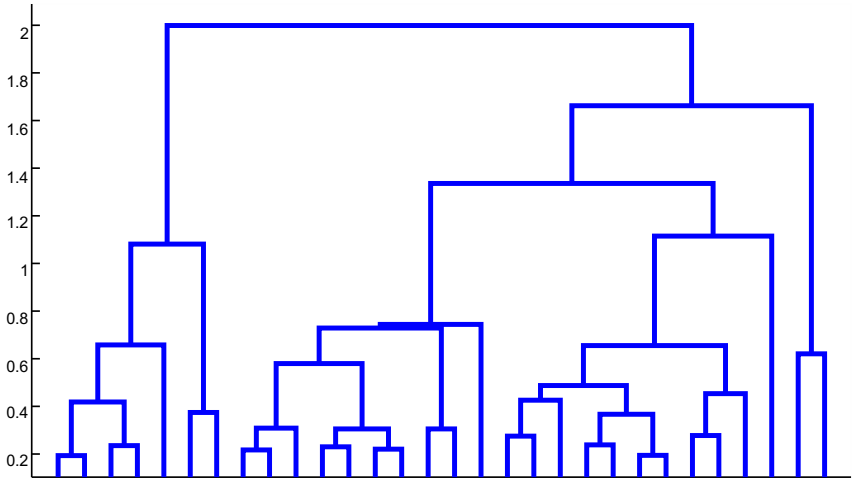
single link



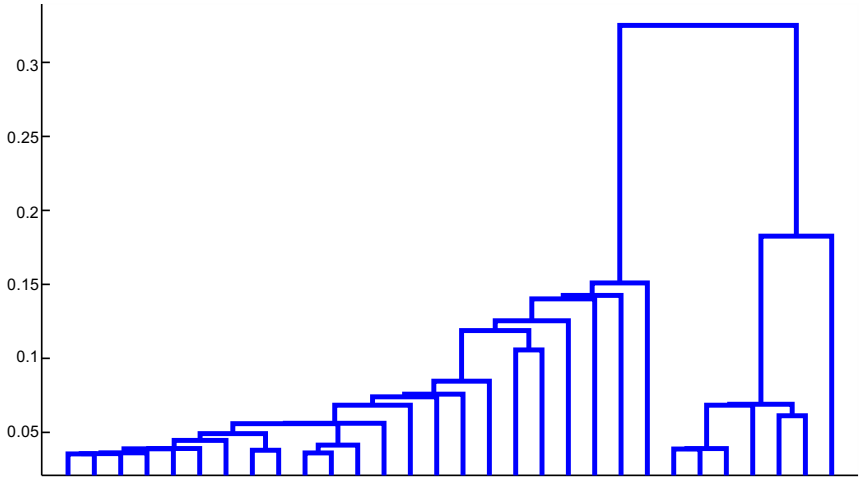
Each discrete value alone the x-axis is treated as one data instance.

Agglomerative Clustering

complete link



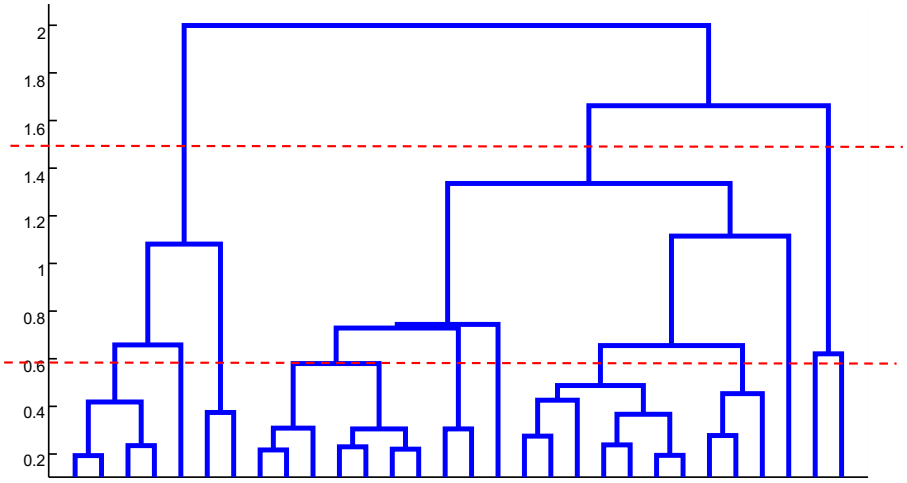
single link



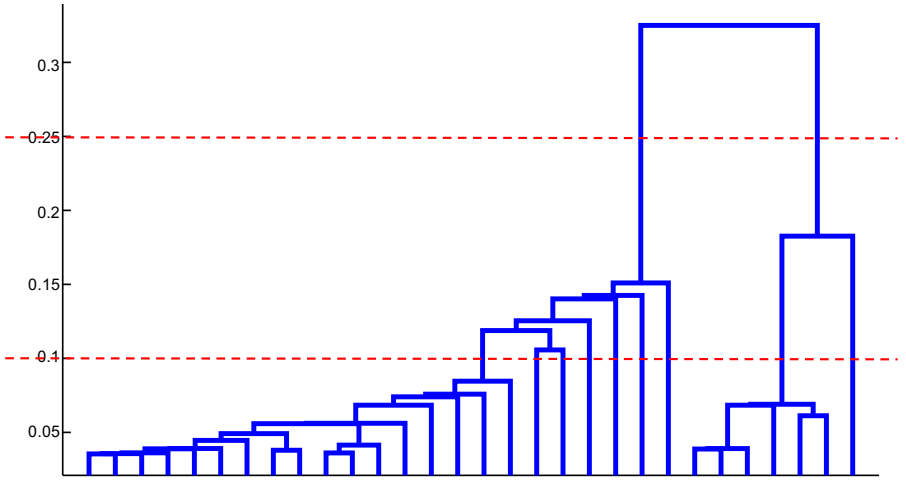
Each discrete value alone the x-axis is treated as one data instance.

Agglomerative Clustering

complete link



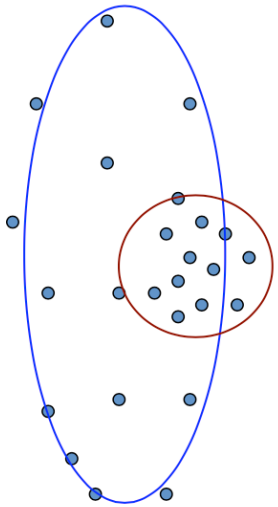
single link



Each discrete value alone the x-axis is treated as one data instance.

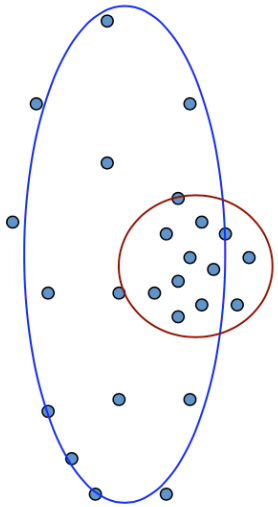
(One) bad case for “hard assignments”?

- Clusters may overlap
- Some clusters may be “wider” than others



(One) bad case for “hard assignments”?

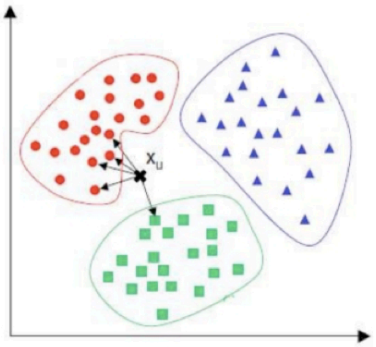
- Clusters may overlap
- Some clusters may be “wider” than others
- *Distances can be deceiving!*



KNN

K-Nearest Neighbor

- Given training data $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$ and a test point
- Prediction Rule: Look at the K most similar training examples



  - For classification: assign the majority class label (majority voting)
  - For regression: assign the average response
- The algorithm requires:
  - Parameter K: number of nearest neighbors to look for
  - Distance function: To compute the similarities between examples

K-Nearest Neighbor Algorithm

- Compute the test point's distance from each training point
- Sort the distances in ascending (or descending) order
- Use the sorted distances to select the K nearest neighbors
- Use majority rule (for classification) or averaging (for regression)

Note: K-Nearest Neighbors is called a *non-parametric* method

- Unlike other supervised learning algorithms, K-Nearest Neighbors doesn't learn an explicit mapping f from the training data
- It simply uses the training data at the test time to make predictions

K-NN: Compute Distance

- The K-NN algorithm requires computing distances of the test example from each of the training examples
- Several ways to compute distances
- The choice depends on the type of the features in the data
- Real-valued features ($\mathbf{x}_i \in \mathbb{R}^D$): Euclidean distance is commonly used

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{m=1}^{D} (x_{im} - x_{jm})^2} = \sqrt{||\mathbf{x}_i||^2 + ||\mathbf{x}_j||^2 - 2\mathbf{x}_i^T \mathbf{x}_j}$$

- Generalization of the distance between points in 2 dimensions
- $||\mathbf{x}_i|| = \sqrt{\sum_{m=1}^{D} x_{im}^2}$ is called the **norm** of $\mathbf{x}_i$
  - Norm of a vector $\mathbf{x}$ is also its **length**

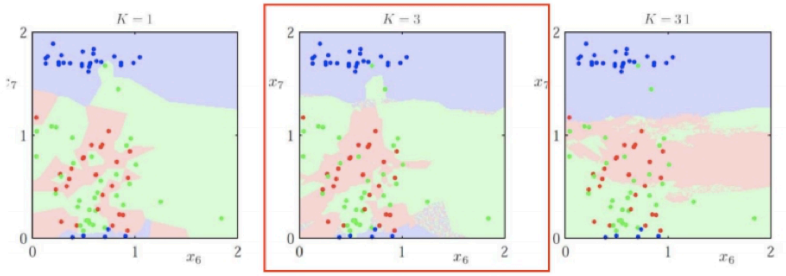
K-NN: Feature Normalization

- Note: Features should be on the same scale
- Example: if one feature has its values in millimeters and another has in centimeters, we would need to normalize
- One way is:
  - Replace x_{im} by $z_{im} = \frac{(x_{im} - \bar{x_m})}{\sigma_m}$ (make them zero mean, unit variance)
  - $\bar{x_m} = \frac{1}{N} \sum_{i=1}^{N} x_{im}$: empirical mean of m^{th} feature
  - $\sigma_m^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{im} - \bar{x_m})^2$: empirical variance of m^{th} feature

K-NN: Other Distance Measure

- Binary-valued features
  - Use Hamming distance: $d(x_i, x_j) = \sum_{m=1}^{D} \mathbb{I}(x_{im} \neq x_{jm})$
  - Hamming distance counts the number of features where the two examples disagree
- Mixed feature types (some real-valued and some binary-valued)?
  - Can use mixed distance measures
  - E.g., Euclidean for the real part, Hamming for the binary part
- Can also assign **weights** to features: $d(x_i, x_j) = \sum_{m=1}^{D} w_m d(x_{im}, x_{jm})$

K-NN: Choice of K



- Small K
  - Creates many small regions for each class
  - May lead to non-smooth) decision boundaries and overfit
- Large K
  - Creates fewer larger regions
  - Usually leads to smoother decision boundaries (caution: too smooth decision boundary can underfit)
- Choosing K
  - Often data dependent and heuristic based
  - Or using cross-validation (using some **held-out data**)
  - In general, a K too small or too big is bad!

K-NN: Pseudocode

1. Calculate "d(x, x_i)" i =1, 2, ….., **n**; where **d** denotes the Euclidean distance between the points.
2. Arrange the calculated **n** Euclidean distances in non-decreasing order.
3. Let **k** be a +ve integer, take the first **k** distances from this sorted list.
4. Find those **k**-points corresponding to these **k**-distances.
5. Let k_i denotes the number of points belonging to the i^{th} class among **k** points i.e. $k \geq 0$
6. If $k_i > k_j \; \forall \; i \neq j$ then put x in class i.

Let (X_i, C_i) where i = 1, 2……., n be data points. X_i denotes feature values & C_i denotes labels for X_i for each i.

Assuming the number of classes as 'c'
$C_i \in \{1, 2, 3, ……, c\}$ for all values of i

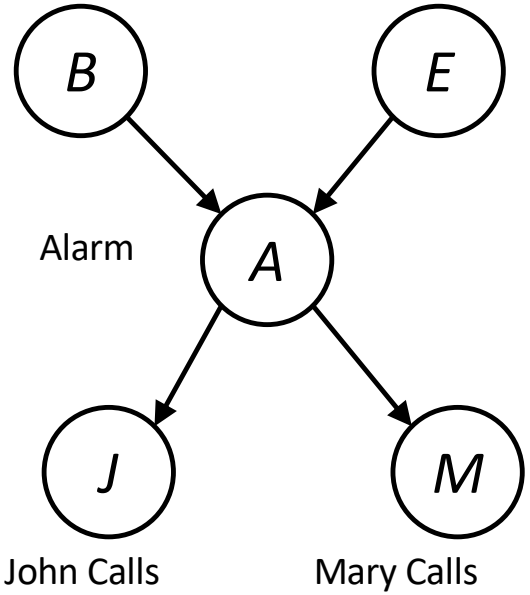
Let x be a point for which label is not known, and we would like to find the label class using k-nearest neighbor algorithms.

Bayes Network

Inference on PGMs

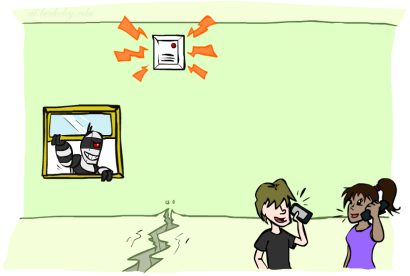
Burglary

B	P(B)
+b	0.001
-b	0.999



E	P(E)
+e	0.002
-e	0.998

Earthquake



A	J	P(J|A)
+a	+j	0.9
+a	-j	0.1
-a	+j	0.05
-a	-j	0.95

A	M	P(M|A)
+a	+m	0.7
+a	-m	0.3
-a	+m	0.01
-a	-m	0.99

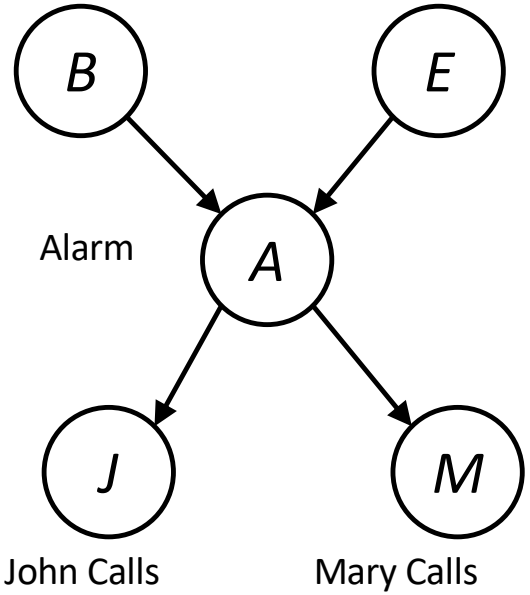
B	E	A	P(A|B,E)
+b	+e	+a	0.95
+b	+e	-a	0.05
+b	-e	+a	0.94
+b	-e	-a	0.06
-b	+e	+a	0.29
-b	+e	-a	0.71
-b	-e	+a	0.001
-b	-e	-a	0.999

$$P(+b, -e, +a, -j, +m) =$$
$$P(+b)P(-e)P(+a|+b,-e)P(-j|+a)P(+m|+a) =$$

Inference on PGMs

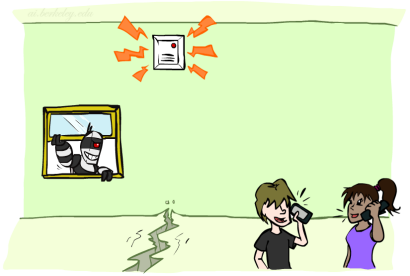
Burglary

B	P(B)
+b	0.001
-b	0.999



E	P(E)
+e	0.002
-e	0.998

Earthquake



A	J	P(J|A)
+a	+j	0.9
+a	-j	0.1
-a	+j	0.05
-a	-j	0.95

A	M	P(M|A)
+a	+m	0.7
+a	-m	0.3
-a	+m	0.01
-a	-m	0.99

B	E	A	P(A|B,E)
+b	+e	+a	0.95
+b	+e	-a	0.05
+b	-e	+a	0.94
+b	-e	-a	0.06
-b	+e	+a	0.29
-b	+e	-a	0.71
-b	-e	+a	0.001
-b	-e	-a	0.999

$$P(+b, -e, +a, -j, +m) =$$

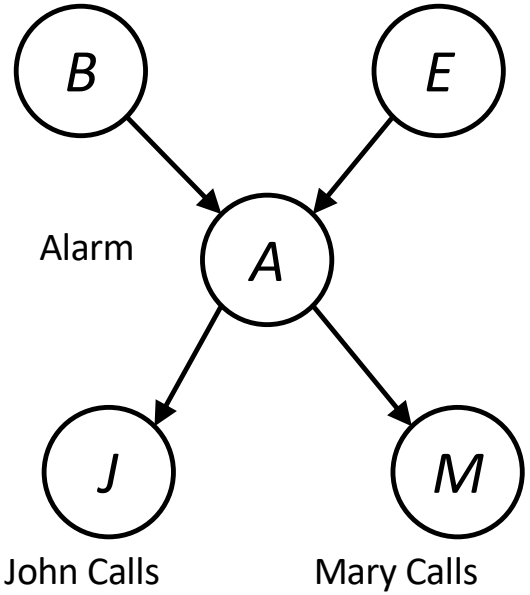
$$P(+b)P(-e)P(+a|+b,-e)P(-j|+a)P(+m|+a) =$$

$$0.001 \times 0.998 \times 0.94 \times 0.1 \times 0.7$$

Inference on PGMs

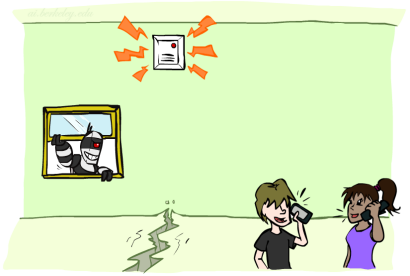
Burglary

B	P(B)
+b	0.001
-b	0.999



E	P(E)
+e	0.002
-e	0.998

Earthquake



A	J	P(J|A)
+a	+j	0.9
+a	-j	0.1
-a	+j	0.05
-a	-j	0.95

A	M	P(M|A)
+a	+m	0.7
+a	-m	0.3
-a	+m	0.01
-a	-m	0.99

B	E	A	P(A|B,E)
+b	+e	+a	0.95
+b	+e	-a	0.05
+b	-e	+a	0.94
+b	-e	-a	0.06
-b	+e	+a	0.29
-b	+e	-a	0.71
-b	-e	+a	0.001
-b	-e	-a	0.999

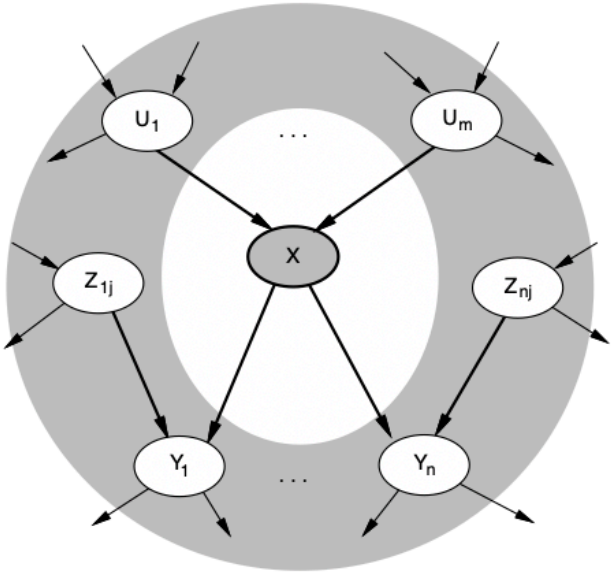
$$P(B \mid +j, +m) = \frac{P(B, +j, +m)}{P(+j, +m)} = ?$$

Markov Blanket

- Definition:
  - The smallest set of “observed” nodes that renders a node t conditionally independent of all the other nodes in the graph.
- Markov blanket in DAG is:
  - Parents
  - Children
  - Co-parents (other nodes that are also parents of the children)

Markov Blanket

- Each node is conditionally independent of all others given its Markov blanket:
  - parents + children + children's parents

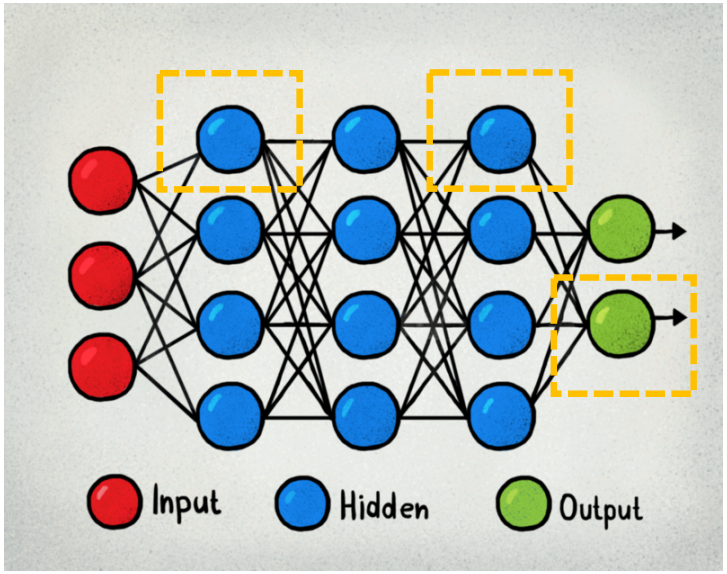


Neural Network

“Artificial” Neuron

- A basic “computational unit” of neural networks

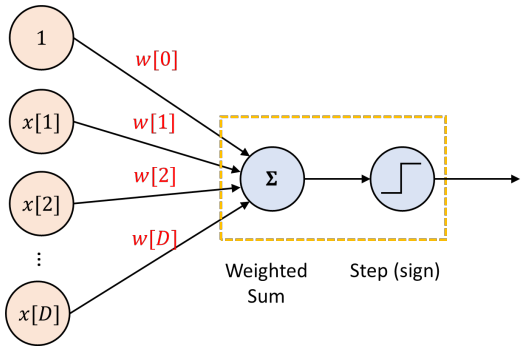
x $z^{(1)}$ $z^{(2)}$ $z^{(3)}$ y

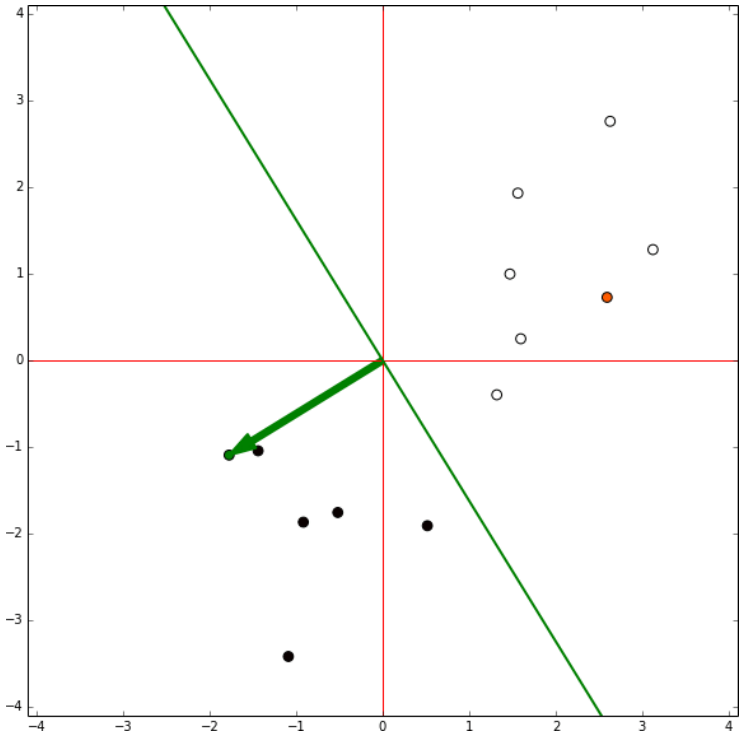


Perceptron Algorithm

- Let $\boldsymbol{x} \in \mathbb{R}^D$ and $\boldsymbol{w} \in \mathbb{R}^D$ (b is merged into $\boldsymbol{w}$), $y \in \{-1,1\}$

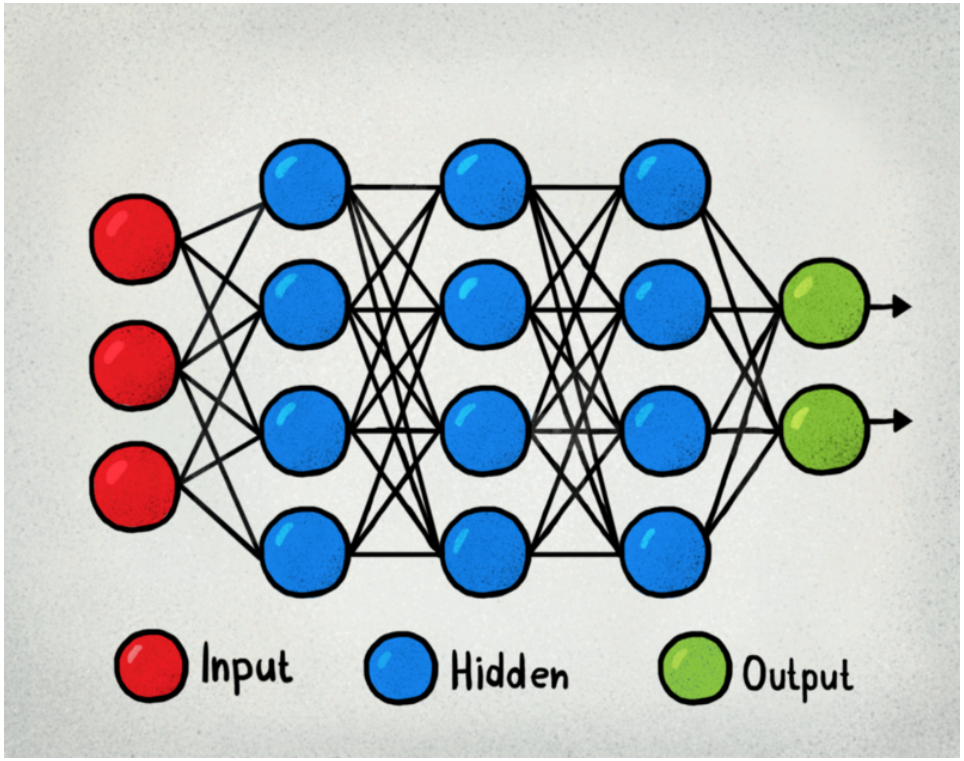
- Initialize weight vector $\boldsymbol{w} = \boldsymbol{0}$
- Loop for T iterations
  - Loop for all training examples $\boldsymbol{x}_n$ (random order!)
  - Predict $\hat{y}_n = \text{sign}(\boldsymbol{w}^T \boldsymbol{x}_n)$
  - If $\hat{y}_n \neq y_n$
    - Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(y_n \boldsymbol{x}_n)$





Multi-layer perceptron

- How to learn?
- Perceptron algorithm?
  - Only look at one instance a time
  - Can update only a single perceptron
  - Not converge for not linear separable data



Training a neural network: gradient descent

- Let $\boldsymbol{x} \in \mathbb{R}^D$; y as true label, and $\{(\boldsymbol{x}_n, y_n)\}$ as the training data
- $\boldsymbol{\theta}$ as all parameters, $\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$ as the neural network's prediction

- Initialize $\boldsymbol{\theta}$ with [with some specific methods]

- Loop for T "epochs"
  - $\nabla_{\boldsymbol{\theta}} L = \frac{1}{N} \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}} l\,(y_n, \hat{y}_n)$
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \times \nabla_{\boldsymbol{\theta}} L$

Stochastic gradient descent

- Let $\boldsymbol{x} \in \mathbb{R}^D$; y as true label, and $\{(\boldsymbol{x}_n, y_n)\}$ as the training data
- $\boldsymbol{\theta}$ as all parameters, $\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$ as the neural network's prediction

- Initialize $\boldsymbol{\theta}$ with [with some specific methods]

- Loop for T "epochs"
  - Loop for all training examples $\boldsymbol{x}_n$ (random order!)
  - $\nabla_{\boldsymbol{\theta}} L = \nabla_{\boldsymbol{\theta}} l(y_n, \hat{y}_n)$
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \times \nabla_{\boldsymbol{\theta}} L$

“Mini-batch” Stochastic gradient descent

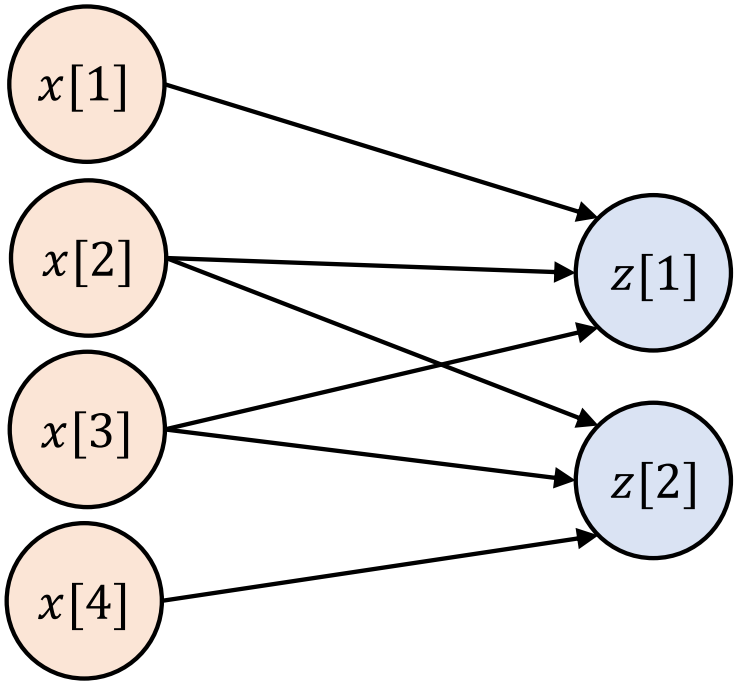
- Let $\boldsymbol{x} \in \mathbb{R}^D$; y as true label, and $\{(\boldsymbol{x}_n, y_n)\}$ as the training data
- $\boldsymbol{\theta}$ as all parameters, $\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$ as the neural network’s prediction

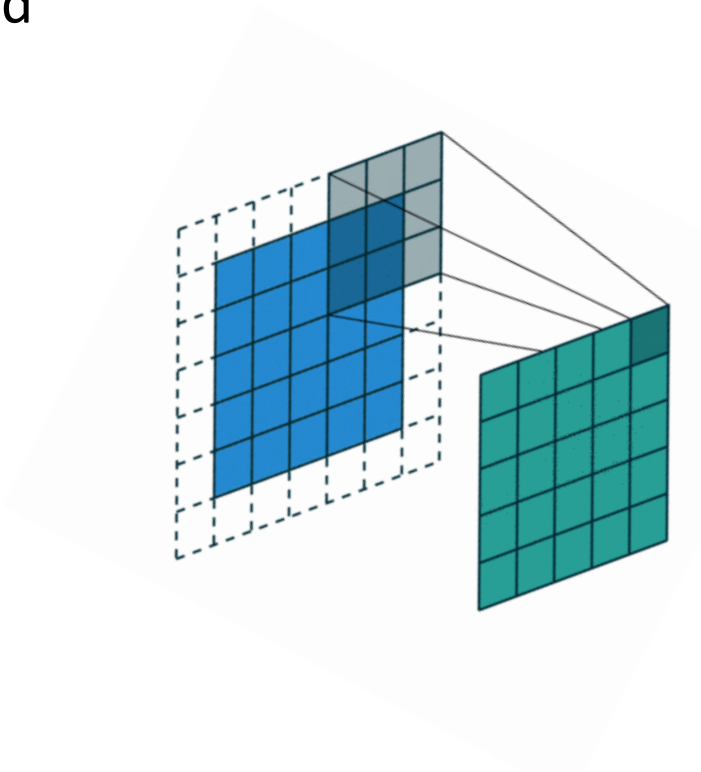
- Initialize $\boldsymbol{\theta}$ with [with some specific methods]

- Loop for T “epochs”
  - Loop for “B sampled examples from $\{(\boldsymbol{x}_n, y_n)\}$” (called batch) without replacement (random order!)
  - $\nabla_{\boldsymbol{\theta}} L = \frac{1}{N} \sum_{b=1}^{B} \nabla_{\boldsymbol{\theta}} l\,(y_b, \hat{y}_b)$
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \times \nabla_{\boldsymbol{\theta}} L$

Convolutional Neural Networks

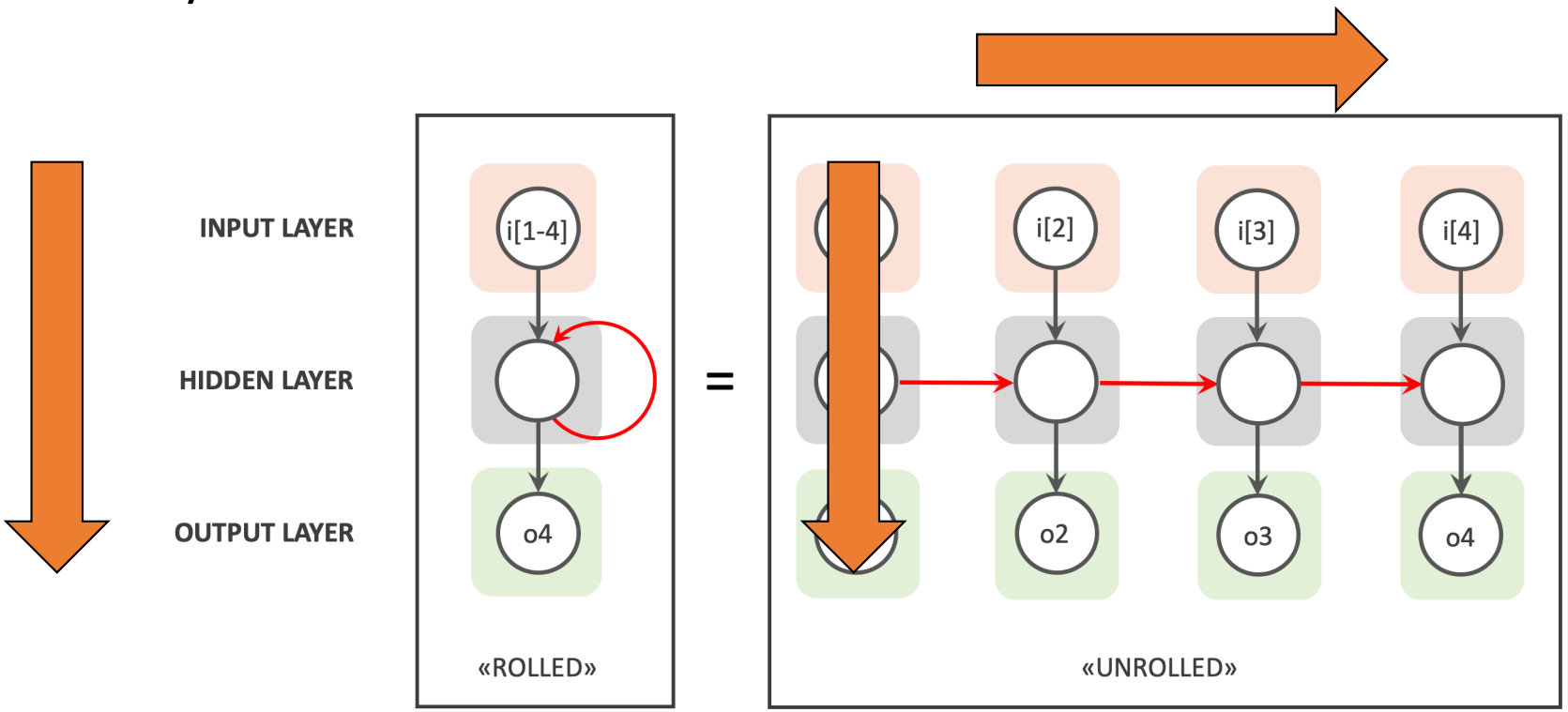
- A current node is not directly affected by "all nodes in the previous layer"
- The network "weights" on the edges can be "re-used"





Recurrent Neural Network

- Other styles: Recurrent neural network




GOOD LUCK
ON YOUR EXAMS!
I BELIEVE IN YOU,
AND I THINK YOU
CAN DO REALLY WELL!