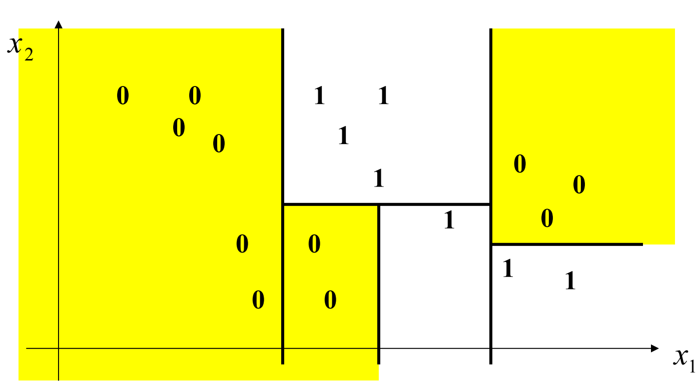
CS 2750 Machine Learning
Lecture 18

Ensemble methods:
• Mixtures of experts
• Bagging & Boosting

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Reviewing Decision trees

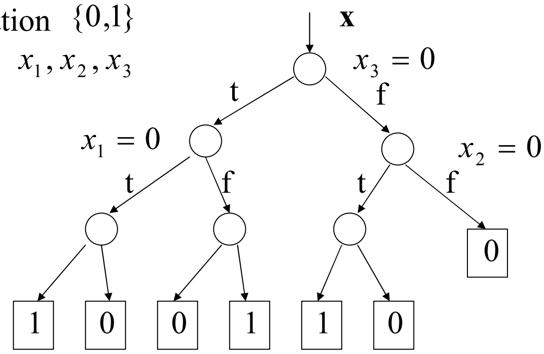
- An approach to classification that:
  - **Partitions the input space to regions**
  - **Classifies independently in every region**



Decision trees

- The partitioning idea is used in the **decision tree model**:
  - Split the space recursively according to inputs in **x**
  - Classify (assign class label) at the bottom of the tree

Example:
Binary classification $\{0,1\}$
Binary attributes x_1, x_2, x_3



Decision tree learning

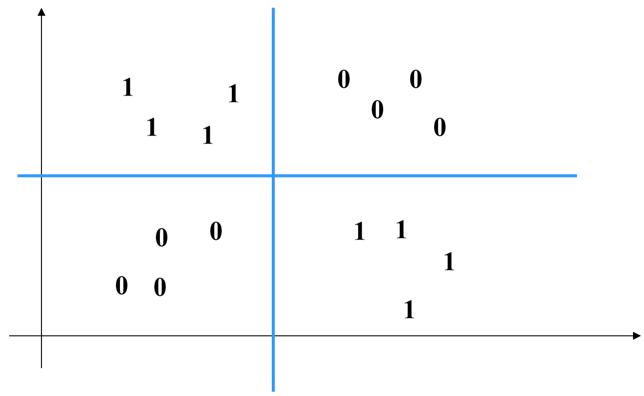
- **Greedy learning algorithm**:

  Repeat until no or small improvement in the purity
  - Find the attribute with the highest gain
  - Add the attribute to the tree and split the set accordingly

- Builds the tree in the top-down fashion
  - Gradually expands the leaves of the partially built tree
- The method is greedy
  - It looks at a single attribute and gain in each step
  - May fail when the combination of attributes is needed to improve the purity (parity functions)

Limitations of Decision trees

- **Greedy learning methods: a** combination of two or more attributes improves the impurity
- **Rectangular regions**

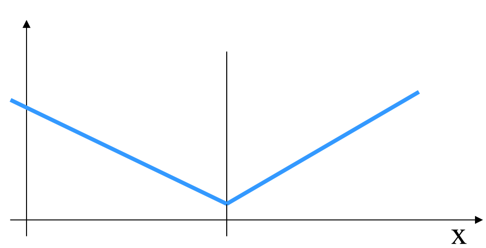


Mixture of experts model

- **Ensamble methods:**
  - Use a combination of simpler learners to improve predictions
- **Mixture of expert model:**
  - Different input regions covered with different learners
  - A "soft" switching between learners

- **Mixture of experts**

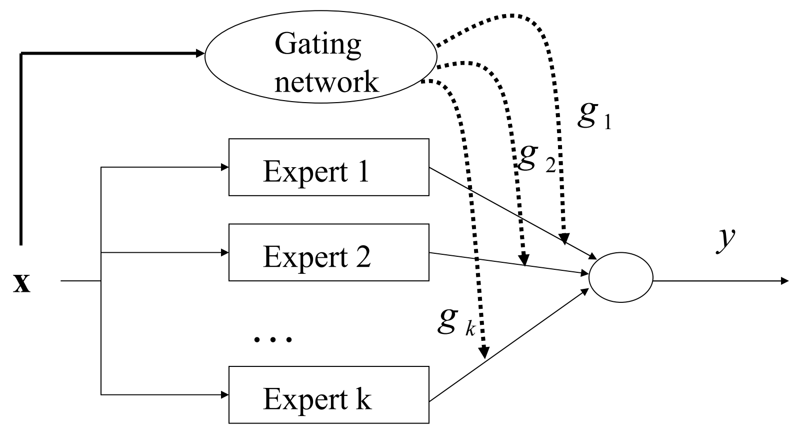
  Expert = learner



Mixture of experts model

- **Gating network** : decides what expert to use

  $g_1, g_2, \dots g_k$ - gating functions



Learning mixture of experts

- **Learning consists of two tasks:**
  - Learn the parameters of individual expert networks
  - Learn the parameters of the gating network
    - Decides where to make a split
- **Assume:** gating functions give probabilities

$$0 \leq g_1(\mathbf{x}), g_2(\mathbf{x}), \dots g_k(\mathbf{x}) \leq 1 \qquad \sum_{u=1}^{k} g_u(\mathbf{x}) = 1$$

- Based on the probability we partition the space
  - partitions belongs to different experts
- How to model the gating network?
  - **A multi-way classifier model:**
    - **softmax model**
    - **a generative classifier model**

Learning mixture of experts

- Assume we have a **set of linear experts**

$$\mu_i = \boldsymbol{\theta}_i^T \mathbf{x} \qquad \text{(Note: bias terms are hidden in x)}$$

- Assume a **softmax gating network**

$$g_i(\mathbf{x}) = \frac{\exp(\boldsymbol{\eta}_i^T \mathbf{x})}{\sum_{u=1}^{k} \exp(\boldsymbol{\eta}_u^T \mathbf{x})} \approx p(\omega_i \mid \mathbf{x}, \boldsymbol{\eta})$$

- Likelihood of y (assumed that errors for different experts are normally distributed with the same variance)

$$P(y \mid \mathbf{x}, \Theta, \boldsymbol{\eta}) = \sum_{i=1}^{k} P(\omega_i \mid \mathbf{x}, \boldsymbol{\eta}) p(y \mid \mathbf{x}, \omega_i, \boldsymbol{\Theta})$$

$$= \sum_{i=1}^{k} \left[\frac{\exp(\boldsymbol{\eta}_i^T \mathbf{x})}{\sum_{j=1}^{k} \exp(\boldsymbol{\eta}_j^T \mathbf{x})} \right] \left[\frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(- \frac{\| y - \mu_i \|^2}{2\sigma^2} \right) \right]$$

Learning mixture of experts

Gradient learning.

On-line update rule for parameters $\boldsymbol{\theta}_i$ of expert *i*

– If we know the expert that is responsible for **x**

$$\theta_{ij} \leftarrow \theta_{ij} + \alpha_{ij} (y - \mu_i) x_j$$

– If we do not know the expert

$$\theta_{ij} \leftarrow \theta_{ij} + \alpha_{ij} h_i (y - \mu_i) x_j$$

h_i - **responsibility of the *i*th expert** = a kind of posterior

$$h_i(\mathbf{x}, y) = \frac{g_i(\mathbf{x}) p(y \mid \mathbf{x}, \omega_i, \boldsymbol{\theta})}{\sum_{u=1}^{k} g_u(\mathbf{x}) p(y \mid \mathbf{x}, \omega_u, \boldsymbol{\theta})} = \frac{g_i(\mathbf{x}) \exp\left(-1/2 \| y - \mu_i \|^2\right)}{\sum_{u=1}^{k} g_u(\mathbf{x}) \exp\left(-1/2 \| y - \mu_u \|^2\right)}$$

$g_i(\mathbf{x})$ - a prior $\qquad$ $\exp(...)$ - a likelihood

Learning mixtures of experts

Gradient methods

- **On-line learning of gating network parameters** $\boldsymbol{\eta}_i$

$$\eta_{ij} \leftarrow \eta_{ij} + \beta_{ij}(h_i(\mathbf{x}, y) - g_i(\mathbf{x}))x_j$$

- The learning with conditioned mixtures can be extended to learning of parameters of an **arbitrary expert network**
  - e.g. logistic regression, multilayer neural network

$$\theta_{ij} \leftarrow \theta_{ij} + \beta_{ij} \frac{\partial l}{\partial \theta_{ij}}$$

$$\frac{\partial l}{\partial \theta_{ij}} = \frac{\partial l}{\partial \mu_i} \frac{\partial \mu_i}{\partial \theta_{ij}} = h_i \frac{\partial \mu_i}{\partial \theta_{ij}}$$

Learning mixture of experts

EM algorithm offers an alternative way to learn the mixture

Algorithm:

Initialize parameters Θ

Repeat

Set $\Theta' = \Theta$

1. **Expectation step**

$$Q(\Theta \mid \Theta') = E_{H|\mathbf{X},\mathbf{Y},\mathbf{\Theta}'} \log P(\mathbf{H}, \mathbf{Y} \mid \mathbf{X}, \Theta, \xi)$$

2. **Maximization step**

$$\Theta = \arg\max_{\Theta} Q(\Theta \mid \Theta')$$

until no or small improvement in $Q(\Theta \mid \Theta')$

- **Hidden variables are identities of expert networks responsible for (x,y) data points**

Learning mixture of experts with EM

- Assume we have a **set of linear experts**

$$\mu_i = \boldsymbol{\theta}_i^T \mathbf{x}$$

- Assume a **softmax gating network**

$$g_i(\mathbf{x}) = P(\omega_i \mid \mathbf{x}, \boldsymbol{\eta})$$

- **Q function to optimize**

$$Q(\Theta \mid \Theta') = E_{H|\mathbf{X},\mathbf{Y},\boldsymbol{\Theta}'} \log P(\mathbf{H}, \mathbf{Y} \mid \mathbf{X}, \Theta, \xi)$$

- **Assume:**
  - l indexes different data points
  - δ_i^l an indicator variable for the data point l to be covered by an expert i

$$Q(\Theta \mid \Theta') = \sum_l \sum_i E(\delta_i^l \mid \mathbf{x}^l, y^l, \Theta', \boldsymbol{\eta}') \log(P(y^l, \omega_i \mid \mathbf{x}^l, \Theta, \boldsymbol{\eta}))$$

Learning mixture of experts with EM

- **Assume:**
  - l indexes different data points
  - δ_i^l an indicator variable for data point l and expert i

$$Q(\Theta \mid \Theta') = \sum_l \sum_i E(\delta_i^l \mid \mathbf{x}^l, y^l, \Theta', \boldsymbol{\eta}') \log(P(y^l, \omega_i \mid \mathbf{x}^l, \Theta, \boldsymbol{\eta}))$$

$$E(\delta_i^l \mid \mathbf{x}^l, y^l, \Theta', \boldsymbol{\eta}') = h_i^l(\mathbf{x}^l, y^l) = \frac{g_i(\mathbf{x}^l) p(y \mid \mathbf{x}^l, \omega_i, \boldsymbol{\theta}')}{\sum_{u=1}^{k} g_u(\mathbf{x}^l) p(y^l \mid \mathbf{x}^l, \omega_u, \boldsymbol{\theta}')}$$

Responsibility of the expert i for *(x,y)*

$$Q(\Theta \mid \Theta') = \sum_l \sum_i h_i^l(\mathbf{x}^l, y^l) \log(P(y^l, \omega_i \mid \mathbf{x}^l, \Theta, \boldsymbol{\eta}))$$

Learning mixture of experts with EM

- The maximization step boils down to the problem that is equivalent to the problem of finding the ML estimates of the parameters of the expert and gating networks

$$Q(\Theta \mid \Theta') = \sum_{l} \sum_{i} h_i^l(\mathbf{x}^l, y^l) \log(P(y^l, \omega_i \mid \mathbf{x}^l, \Theta, \boldsymbol{\eta}))$$

$$\log(P(y^l, \omega_i \mid \mathbf{x}^l, \Theta, \boldsymbol{\eta})) = \log P(y^l \mid \omega_i, \mathbf{x}^l, \Theta) + \log P(\omega_i \mid \mathbf{x}^l, \boldsymbol{\eta})$$

Expert network i (Linear regression) — Gating network (Softmax)

- Note that any optimization technique can be applied in this step

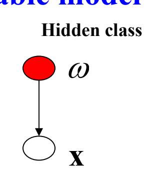
Learning mixture of experts

- **Note that we can use different expert and gating models**
- **For example:**
  - Experts: logistic regression models

$$y_i = 1/(1 + \exp(-\boldsymbol{\theta}_i^T \mathbf{x}))$$

  - Gating network: a **generative latent variable model**

$$g_i(\mathbf{x}) = P(\omega_i \mid \mathbf{x}, \boldsymbol{\eta})$$

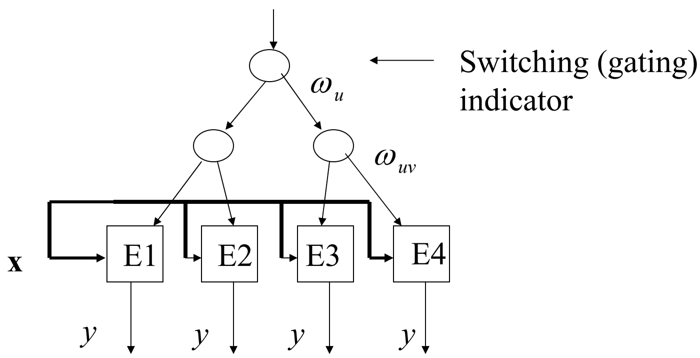


- Likelihood of y:

$$P(y \mid \mathbf{x}, \Theta, \boldsymbol{\eta}) = \sum_{u=1}^{k} P(\omega_u \mid \mathbf{x}, \boldsymbol{\eta}) p(y \mid \mathbf{x}, \omega_u, \boldsymbol{\Theta})$$

Hierarchical mixture of experts

- **Mixture of experts**: define a probabilistic split
- The idea can be extended to a **hierarchy of experts** (a kind of a probabilistic decision tree)



Hierarchical mixture model

An output is conditioned (gated) on multiple mixture levels

$$P(y \mid \mathbf{x}, \Theta) = \sum_u P(\omega_u \mid \mathbf{x}, \eta) \sum_v p(\omega_{uv} \mid \mathbf{x}, \omega_u, \xi_u) .. \sum_s P(\omega_{uv..s} \mid \mathbf{x}, \omega_u, \omega_{uv}, \ldots) \underline{P(y \mid \mathbf{x}, \omega_u, \omega_{uv}, .., \theta_{uv..s})}$$

Individual experts

- **Define** $\Omega_{uv..s} = \{\omega_u, \omega_{uv}, \ldots \omega_{uv..s}\}$

$$P(\Omega_{uv..s} \mid \mathbf{x}, \Theta) = P(\omega_u \mid \mathbf{x}) P(\omega_{uv} \mid \mathbf{x}, \omega_u) .. P(\omega_{uv..s} \mid \mathbf{x}, \omega_u, \omega_{uv}, \ldots)$$

- **Then**

$$P(y \mid \mathbf{x}, \Theta) = \sum_u \sum_v \ldots \sum_s P(\Omega_{uv..s} \mid \mathbf{x}, \Theta) P(y \mid \mathbf{x}, \Omega_{uv..s}, \Theta)$$

- Mixture model is a kind of soft decision tree model

- with a fixed tree structure !!

Hierarchical mixture of experts

- Multiple levels of probabilistic gating functions

$$g_u(\mathbf{x}) = P(\omega_u \mid \mathbf{x}, \Theta) \qquad g_{v|u}(\mathbf{x}) = P(\omega_{uv} \mid \mathbf{x}, \omega_u \Theta)$$

- Multiple levels of responsibilities

$$h_u(\mathbf{x}, y) = P(\omega_u \mid \mathbf{x}, y, \Theta) \qquad h_{v|u}(\mathbf{x}, y) = P(\omega_{uv} \mid \mathbf{x}, y, \omega_u, \Theta)$$

- How they are related?

$g_{v|u}(\mathbf{x})$

responsibility

$$P(\omega_{uv} \mid \mathbf{x}, y, \omega_u, \Theta) = \frac{P(y \mid \mathbf{x}, \omega_u, \omega_{uv}, \Theta) P(\omega_{uv} \mid \mathbf{x}, \omega_u, \Theta)}{\sum_v P(y \mid \mathbf{x}, \omega_u, \omega_{uv}, \Theta) P(\omega_{uv} \mid \mathbf{x}, \omega_u, \Theta)}$$

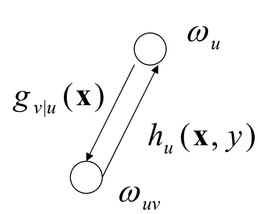
$$= \sum_v P(y, \omega_{uv} \mid \mathbf{x}, \omega_u, \Theta) = P(y \mid \mathbf{x}, \omega_u, \Theta)$$

Hierarchical mixture of experts

- **Responsibility for the top layer**

$$h_u(\mathbf{x}, y) = P(\omega_u \mid \mathbf{x}, y, \Theta) = \frac{P(y \mid \mathbf{x}, \omega_u, \Theta) P(\omega_u \mid \mathbf{x}, \Theta)}{\sum_u P(y \mid \mathbf{x}, \omega_u, \Theta) P(\omega_u \mid \mathbf{x}, \Theta)}$$

- But $P(y \mid \mathbf{x}, \omega_u \Theta)$ is computed while computing $h_{v|u}(\mathbf{x}, y) = P(\omega_{uv} \mid \mathbf{x}, y, \omega_u, \Theta)$
- **General algorithm:**
  - **Downward sweep; calculate** $g_{v|u}(x) = P(\omega_{uv} \mid \mathbf{x}, \omega_u, \Theta)$
  - **Upward sweep; calculate** $h_u(\mathbf{x}, y) = P(\omega_u \mid \mathbf{x}, y, \Theta)$



On-line learning

- Assume linear experts $\mu_{uv} = \boldsymbol{\theta}_{uv}{}^{T}\mathbf{x}$
- **Gradients (vector form):**

$$\frac{\partial l}{\partial \boldsymbol{\theta}_{uv}} = h_u h_{v|u}(y - \mu_{uv})\mathbf{x}$$

$$\frac{\partial l}{\partial \boldsymbol{\eta}} = (h_u - g_u)\mathbf{x} \qquad \text{Top level (root) node}$$

$$\frac{\partial l}{\partial \boldsymbol{\xi}} = h_u(h_{v|u} - g_{v|u})\mathbf{x} \qquad \text{Second level node}$$

- Again: can it can be extended to different expert networks

Ensemble methods

- **Mixture of experts**
  - Multiple 'base' models (classifiers, regressors), each covers a different part (region) of the input space
- **Committee machines:**
  - Multiple 'base' models (classifiers, regressors), each covers the complete input space
  - Each base model is trained on a slightly different train set
  - Combine predictions of all models to produce the output
    - **Goal:** Improve the accuracy of the 'base' model
  - **Methods:**
    - **Bagging**
    - **Boosting**
    - Stacking (not covered)

Bagging (Bootstrap Aggregating)

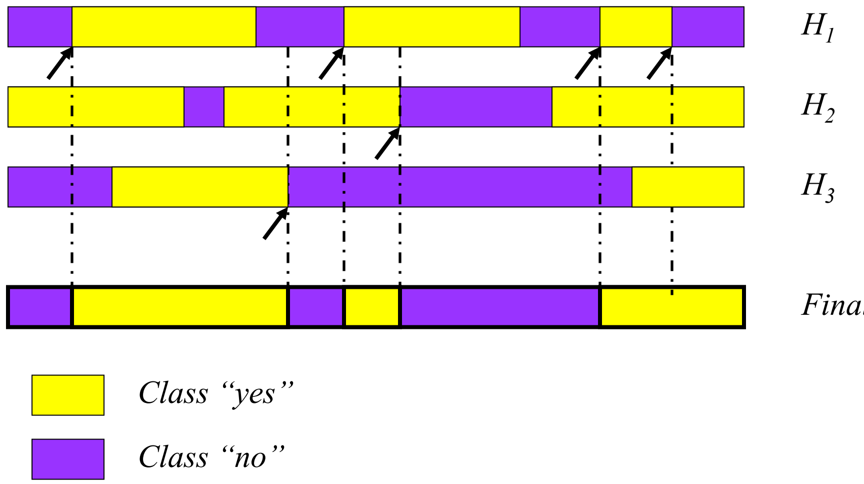
- **Given:**
  - Training set of *N* examples
  - A class of learning models (e.g. decision trees, neural networks, …)
- **Method:**
  - Train multiple (k) models on different samples (data splits) and average their predictions
  - Predict (test) by averaging the results of k models
- **Goal:**
  - Improve the accuracy of one model by using its multiple copies
  - Average of misclassification errors on different data splits gives a better estimate of the predictive ability of a learning method

Bagging algorithm

- **Training**
  - In each iteration t, t=1,…T
    - Randomly sample with replacement *N* samples from the training set
    - Train a chosen "base model" (e.g. neural network, decision tree) on the samples
- **Test**
  - For each test example
    - Start all trained base models
    - Predict by combining results of all T trained models:
      - **Regression:** averaging
      - **Classification:** a majority vote

Simple Majority Voting

Test examples



Analysis of Bagging

- **Expected error= Bias+Variance**
  - *Expected error* is the expected discrepancy between the estimated and true function

$$E\left[\left(\hat{f}(X) - E[f(X)]\right)^2\right]$$

  - *Bias* is squared discrepancy between *averaged* estimated and true function

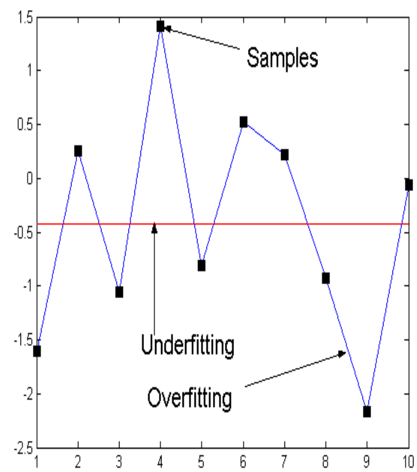
$$\left(E\left[\hat{f}(X)\right] - E[f(X)]\right)^2$$

  - *Variance* is expected divergence of the estimated function vs. its average value

$$E\left[\left(\hat{f}(X) - E\left[\hat{f}(X)\right]\right)^2\right]$$

When Bagging works?
Under-fitting and over-fitting

- **Under-fitting:**
  - High bias (models are not accurate)
  - Small variance (smaller influence of examples in the training set)
- **Over-fitting:**
  - Small bias (models flexible enough to fit well to training data)
  - Large variance (models depend very much on the training set)



Averaging decreases variance

- **Example**
  - Assume we measure a random variable x with a $N(\mu,\sigma^2)$ distribution
  - If only one measurement x_1 is done,
    - The expected mean of the measurement is μ
    - Variance is $Var(x_1)=\sigma^2$
  - If random variable x is measured K times $(x_1,x_2,\ldots x_k)$ and the value is estimated as: $(x_1+x_2+\ldots+x_k)/K$,
    - Mean of the estimate is still μ
    - But, variance is smaller:
      - $[Var(x_1)+\ldots Var(x_k)]/K^2=K\sigma^2 / K^2 = \sigma^2/K$
- Observe: **Bagging is a kind of averaging!**

When Bagging works

- **Main property of Bagging** (proof omitted)
  - Bagging **decreases variance** of the base model without changing the bias!!!
  - Why? averaging!
- **Bagging typically helps**
  - When applied with an **over-fitted base model**
    - High dependency on actual training data
- **It does not help much**
  - High bias. When the base model is robust to the changes in the training data (due to sampling)