Learning with multiple models
  • Mixture of experts
  • Bagging and Boosting

Milos Hauskrecht
milos@pitt.edu
5329 Sennott Square
Learning with multiple models

We know how to build different classification or regression models from data

• **Question:**
  – Is it possible to learn and combine multiple (classification/regression) models and improve their predictive performance?

• **Answer:** yes

• There are different ways of how to do it…
Learning with multiple models

• **Question:**
  – Is it possible to learn and combine multiple (classification/regression) models and improve their predictive performance?

• There are different ways of how to do it…

• Assume you have models M1, M2, … Mk

• **Approach 1:** use different models (classifiers, regressors) to cover the different parts of the input (x) space

• **Approach 2:** use different models (classifiers, regressors) that cover the complete input (x) space, and combine their predictions
Approach 1

• Recall the decision tree:
  – It partitions the input space to regions
  – Picks the class independently for each partition

• What if we define a more general partitions of the input space and learn models specific to these partitions

\[
\begin{array}{c|c|c}
\hline
x_2 & \text{Model 1} & \text{Model 2} & \text{Model 3} \\
\hline
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 \\
\hline
\end{array}
\]
Learning with multiple models: Approach 1

Define a more general partitions of the input space and learn a model specific to these partitions

Example:
• 2 linear functions covering two regions of the input space

Mixture of expert model:
• Expert = learner (model)
• Different input regions are covered with a different learner/model
• A “soft” switching between learners
Mixture of experts model

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

\[ g_1, g_2, \ldots, g_k \] - gating functions

\[ y = \sum_{i=1}^{k} g_i(x)y_i \]
Mixture of experts model

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

\[ g_1, g_2, \ldots, g_k \] - gating functions

Assume
\[ g_1 = 1 \]
\[ g_2 = 0 \]
\[ g_k = 0 \]

\[ y = y_1 = f_1(x) \]
Learning mixture of experts

• **Learning consists of two tasks:**
  – Learn the parameters of individual expert networks
  – Learn the parameters of the gating (switching) network
    • Decides where to make a split
• **Assume:** gating functions give probabilities
  
  \[
  0 \leq g_1(x), g_2(x), \ldots g_k(x) \leq 1 \quad \sum_{u=1}^{k} g_u(x) = 1
  \]

  \[
  y = \sum_{u=1}^{k} g_u(x) f_u(x)
  \]

• Based on the probability we partition the space
  – partitions belongs to different experts
• How to model the gating network?
  – **A multi-class classifier model:**
    • softmax model
Learning mixture of experts

- Assume we have a **set of k linear experts**
  \[ y_i = w_i^T x + \varepsilon \quad \varepsilon \sim N(0, \sigma) \] (Note: bias terms are hidden in \( x \))
- Assume a **softmax gating network**
  \[
g_i(x) = \frac{\exp(\eta_i^T x)}{\sum_{u=1}^{k} \exp(\eta_u^T x)} \approx p(\omega_i \mid x, \eta)\]

![Diagram of learning mixture of experts](image-url)
Learning mixture of experts

- Assume we have a set of linear experts
  \[ y_i = w_i^T x + \varepsilon \quad \varepsilon \sim N(0, \sigma) \]  
  (Note: bias terms are hidden in x)

- Assume a softmax gating network
  \[ g_i(x) = \frac{\exp(\eta_i^T x)}{\sum_{u=1}^{k} \exp(\eta_u^T x)} \approx p(\omega_i \mid x, \eta) \]

- Likelihood of \( y \) (linear regression – assume errors for different experts are normally distributed with the same variance)
  \[
P(y \mid x, W, \eta) = \sum_{i=1}^{k} P(\omega_i \mid x, \eta) p(y \mid x, \omega_i, W) = \sum_{i=1}^{k} \left[ \frac{\exp(\eta_i^T x)}{\sum_{j=1}^{k} \exp(\eta_j^T x)} \right] \frac{1}{\sqrt{2\pi \sigma}} \exp \left( - \frac{\|y - w_i^T x_i\|^2}{2\sigma^2} \right) \]
Learning mixture of experts

Learning of parameters of expert models:

On-line update rule for parameters $w_i$ of expert $i$

- If we know the expert that is responsible for $x$

$$w_{ij} \leftarrow w_{ij} + \alpha_{ij} (y - w_i^T x) x_j$$

- If we do not know the expert

$$w_{ij} \leftarrow w_{ij} + \alpha_{ij} h_i (y - w_i^T x) x_j$$

$h_i$ - responsibility of the $i$th expert for $x$ = a kind of posterior

$$h_i(x, y) = \frac{g_i(x)p(y \mid x, \omega_i, W)}{\sum_{u=1}^{k} g_u(x)p(y \mid x, \omega_u, W)} = \frac{g_i(x)\exp\left(-\frac{1}{2}\|y - w_i^T x\|^2\right)}{\sum_{u=1}^{k} g_u(x)\exp\left(-\frac{1}{2}\|y - w_u^T x\|^2\right)}$$

$g_i(x)$ - a prior \hspace{5cm} \exp(...) - a likelihood
Learning mixtures of experts

Learning of parameters of the gating/switching network:

- **On-line learning of gating network parameters** \( \eta_i \)

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

- The learning with conditional 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 with multiple models: Approach 2

- **Approach 2:** use multiple models (classifiers, regressors) that cover the complete input (x) space and combine their outputs

- **Committee machines:**
  - Combine predictions of all models to produce the output
    - **Regression:** averaging
    - **Classification:** a majority vote
  - **Goal:** Improve the accuracy of the ‘base’ model

- **Methods:**
  - **Bagging (the same base models)**
  - **Boosting (the same base models)**
  - **Stacking (different base model) not covered**
Bagging (Bootstrap Aggregating)

• **Given:**
  – Training set of $N$ examples
  – A base learning model (e.g. decision tree, neural network, …)

• **Method:**
  – Train multiple (k) base models on slightly different datasets
  – 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**
- For each model M1, M2, … Mk
  - Randomly sample with replacement $N$ samples from the training set (bootstrap)
  - Train a chosen “base model” (e.g. neural network, decision tree) on the samples
Bagging algorithm

• **Training**
  • For each model M1, M2, … Mk
    • Randomly sample with replacement \( N \) samples from the training set
    • Train a chosen “base model” (e.g. a neural network, or a decision tree) on the samples

• **Test**
  – For each test example
    • Run all base models M1, M2, … Mk
    • Predict by combining results of all T trained models:
      – **Regression**: averaging
      – **Classification**: a majority vote
Class decision via majority voting

Test examples

\[ \text{model}_1 \]

\[ \text{model}_2 \]

\[ \text{model}_3 \]

\[ \text{Final} \]

Class “yes”

Class “no”
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 a squared discrepancy between averaged estimated and true function
    \[ \left( E[\hat{f}(X)] - E[f(X)] \right)^2 \]
  - **Variance** is an expected divergence of the estimated function vs. its average value
    \[ E \left[ \left( f(X) - E[\hat{f}(X)] \right)^2 \right] \]
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 a random variable $x$ with a $N(\mu, \sigma^2)$ distribution

\[
\text{Case 1: we draw one example/measurement } x_1 \text{ and use it to estimate the mean } \mu' = x_1
\]
  - The expected mean of the estimate $E[\mu'] = E[x_1] = \mu$
  - The variance of the mean estimate $\text{Var}(\mu') = \text{Var}(x_1) = \sigma^2$
Averaging decreases variance

- **Example** Assume a random variable $x$ with a $N(\mu, \sigma^2)$ distribution

- **Case 2:** a variable $x$ is measured independently $K$ times \((x_1, x_2, \ldots, x_k)\) and the mean is estimated as:

  $$\mu' = \frac{x_1 + x_2 + \ldots + x_k}{K},$$

- The expected mean of the estimate $E[\mu'] = \mu$
- But, the variance of the mean estimate $\text{Var}(\mu')$ is smaller:

  $$\text{Var}(\mu') = \frac{\text{Var}(x_1) + \ldots + \text{Var}(x_k)}{K^2} = \frac{K \sigma^2}{K^2} = \frac{\sigma^2}{K}$$
When Bagging works

Relation of the previous example to bagging:
• Bagging is a kind of averaging!

**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
  – Example: fully grown decision trees

**Bagging does not help much when**
• Applied to models with a high bias. When the base model is robust to the changes in the training data (due to sampling)
Boosting

• **Bagging**
  – Multiple models covering the complete space, a learner is not biased to any region
  – Learners are learned independently

• **Boosting**
  – Every learner covers the complete space
  – Learners are biased to regions not predicted well by other learners
  – Learners are dependent
Boosting. Theoretical foundations.

- **PAC:** *Probably Approximately Correct* framework
  - *(\(\varepsilon, \delta\))* solution

- **PAC learning:**
  - Learning with a *pre-specified error* \( \varepsilon \) and a *confidence parameter* \( \delta \)
  - the probability that the misclassification error (ME) is larger than \( \varepsilon \) is smaller than \( \delta \)

\[
P(ME(c) > \varepsilon) \leq \delta
\]

**Alternative rewrite:**

\[
P(Acc(c) > 1 - \varepsilon) > (1 - \delta)
\]

- **Accuracy** (1-\(\varepsilon\)): Percent of correctly classified samples in test
- **Confidence** (1-\(\delta\)): The probability that in one experiment some target accuracy will be achieved
PAC Learnability

Strong (PAC) learnability:
• There exists a learning algorithm that efficiently learns the classification with a pre-specified error and confidence values

Strong (PAC) learner: A learning algorithm $P$ that
• Given an arbitrary:
  – classification error $\epsilon (< 1/2)$, and
  – confidence $\delta$ ($< 1/2$)
  or in other words:
    • classification accuracy $>(1-\epsilon)$
    • confidence probability $>(1-\delta)$
• Outputs a classifier that satisfies this parameters
• Efficiency: runs in time polynomial in $1/\delta$, $1/\epsilon$
  – Implies: number of samples $N$ is polynomial in $1/\delta$, $1/\epsilon$
Weak Learner

Weak learner:

• A learning algorithm (learner) $M$ that gives some fixed (not arbitrary !!!!):
  – error $\varepsilon_o (<1/2)$ and
  – confidence $\delta_o (<1/2)$

• Alternatively:
  – a classification accuracy $> 0.5$
  – with probability $> 0.5$

and this on an arbitrary distribution of data entries
Weak learnability=Strong (PAC) learnability

• Assume there exists a **weak learner**
  – it is better that a random guess (> 50 %) with confidence higher than 50 % on any data distribution

• **Question:**
  – Is the problem also strongly PAC-learnable?
  – Can we generate an algorithm $P$ that achieves an arbitrary $(\varepsilon, \delta)$ accuracy?

• **Why is this important?**
  – Usual classification methods (decision trees, neural nets), have good, but uncontrollable performances.
  – Can we improve their performance to achieve any pre-specified accuracy (confidence)?
Weak=Strong learnability!!!

- **Proof due to R. Schapire**
  An arbitrary $(\varepsilon, \delta)$ improvement is possible

**Idea:** combine multiple weak learners together
- Weak learner $W$ with confidence $\delta_0$ and maximal error $\varepsilon_0$
- It is possible:
  - To improve (boost) the confidence
  - To improve (boost) the accuracy
  by training different weak learners on slightly different datasets
Boosting accuracy
Training

Distribution of examples

Correct classification
Wrong classification

$H_1$ and $H_2$ classify differently
Boosting accuracy

• **Training**
  – Sample randomly from the distribution of examples
  – Train hypothesis $H_1$ on the sample
  – Evaluate accuracy of $H_1$ on the distribution
  – Sample randomly such that for the half of samples $H_1$ provides correct, and for another half, incorrect results; Train hypothesis $H_2$.
  – Train $H_3$ on samples from the distribution where $H_1$ and $H_2$ classify differently

• **Test**
  – For each example, decide according to the majority vote of $H_1$, $H_2$ and $H_3$
Theorem

• If each classifier has an error $< \varepsilon_o$, the final ‘voting’ classifier has error $< g(\varepsilon_o) = 3 \varepsilon_o^2 - 2\varepsilon_o^3$

• Accuracy improved !!!!

• Apply recursively to get to the target accuracy !!!!
Theoretical Boosting algorithm

• Similarly to boosting the accuracy we can boost the confidence at some restricted accuracy cost

• **The key result:** we can improve both the accuracy and confidence

• **Problems with the theoretical algorithm**
  – A good (better than 50 %) classifier on all distributions and problems
  – We cannot get a good sample from data-distribution
  – The method requires a large training set

• **Solution to the sampling problem:**
  – Boosting by sampling
    • AdaBoost algorithm and variants
AdaBoost

- **AdaBoost**: boosting by sampling

- **Classification** (Freund, Schapire; 1996)
  - AdaBoost.M1 (two-class problem)
  - AdaBoost.M2 (multiple-class problem)

- **Regression** (Drucker; 1997)
  - AdaBoostR
AdaBoost training

Training data

Distribution

Uniform distribution $D_1$ of training examples

$P(\text{example } i) = 1/N$
AdaBoost training

Training data \( \rightarrow \) Distribution \( D_1 \) \( \rightarrow \) Learn Model 1

Sample randomly according to \( D_1 \)
And train Model 1
AdaBoost training

Training data → Distribution → Learn → Test

Model 1

Errors 1

Test Model 1 and calculate errors
AdaBoost training

Training data

D1

Model 1

Errors 1

D2

Use errors to recalculate the new distribution on data
Give more probability to pick examples with errors
AdaBoost training

Training data -> Distribution D₁ -> Learn Model 1 -> Test Errors 1

Distribution D₂ -> Learn Model 2 -> Test Errors 2

... 

Distribution Dₜ -> Learn Model T -> Test Errors T
AdaBoost

• **Given:**
  - A training set of \( N \) examples (attributes + class label pairs)
  - A “base” learning model (e.g. a decision tree, a neural network)

• **Training stage:**
  - Train a sequence of \( T \) “base” models on \( T \) different sampling distributions defined upon the training set \( (D) \)
  - A sample distribution \( D_t \) for building the model \( t \) is constructed by modifying the sampling distribution \( D_{t-1} \) from the \((t-1)\)th step.
    - Examples classified incorrectly in the previous step receive higher weights in the new data (attempts to cover misclassified samples)

• **Application (classification) stage:**
  - Classify according to the **weighted majority** of classifiers
AdaBoost algorithm

Training (step t)

- **Sampling Distribution** \( D_t \)
  \( D_t(i) \) - a probability that example i from the original training dataset is selected
  \( D_1(i) = 1 / N \) for the first step (t=1)
- Take \( K \) samples from the training set according to \( D_t \)
- Train a classifier \( h_t \) on the samples
- Calculate the error \( \varepsilon_t \) of \( h_t \):
  \( \varepsilon_t = \sum_{i: h_t(x_i) \neq y_i} D_t(i) \)
- **Classifier weight:** \( \beta_t = \varepsilon_t / (1 - \varepsilon_t) \)
- **New sampling distribution**
  \[
  D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} 
  \beta_t & h_t(x_i) = y_i \\
  1 & \text{otherwise}
\end{cases}
\]

Norm. constant
AdaBoost. Sampling Probabilities

Example:  - Nonlinearly separable binary classification
          - NN used as a week learner
AdaBoost: Sampling Probabilities
AdaBoost classification

- We have $T$ different classifiers $h_t$
  - weight $w_t$ of the classifier is proportional to its accuracy on the training set
    \[
    w_t = \log\left(\frac{1}{\beta_t}\right) = \log\left(\frac{(1 - \varepsilon_t)}{\varepsilon_t}\right)
    \]
    \[
    \beta_t = \frac{\varepsilon_t}{1 - \varepsilon_t}
    \]

- **Classification:**
  For every class $j=0,1$
  - Compute the sum of weights $w$ corresponding to ALL classifiers that predict class $j$;
  - Output class that correspond to the maximal sum of weights (weighted majority)
    \[
    h_{\text{final}}(x) = \arg \max_j \sum_{t: h_t(x) = j} w_t
    \]
Two-Class example. Classification.

- Classifier 1 "yes" 0.7
- Classifier 2 "no" 0.3
- Classifier 3 "no" 0.2

Weighted majority "yes"

\[0.7 - 0.5 = + 0.2\]

The final choice is "yes" + 1
What is boosting doing?

• Each classifier specializes on a particular subset of examples
• Algorithm is concentrating on “more and more difficult” examples

• **Boosting can:**
  – Reduce variance (the same as Bagging)
  – Eliminate the effect of high bias of the weak learner (unlike Bagging)

• **Train versus test errors performance:**
  – Train errors can be driven close to 0
  – But test errors do not show overfitting

• Proofs and theoretical explanations in a number of papers
Boosting. Error performances

![Graph showing error performances for Training error, Test error, and Single-learner error.](image)