Learning with multiple models
Mixture of experts
Bagging and Boosting

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

<table>
<thead>
<tr>
<th>$x_2$</th>
<th>Model 1</th>
<th>Model 2</th>
<th>Model 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$\rightarrow x_1$
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 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:

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

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

\[ 0 \leq g_1(x), g_2(x), \ldots, g_k(x) \leq 1 \]
\[ \sum_{u=1}^{k} g_u(x) = 1 \]

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

- Based on the probability we partition the space
  - partitions belongs to different experts
- How to model the gating network?
  - **A multi-way classifier model**:
    - softmax model
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
  \[ g_i(x) = \frac{\exp(\eta_i^T x)}{\sum_{u=1}^{k} \exp(\eta_u^T x)} \approx p(\omega_i \mid x, \eta) \]

\[
\begin{align*}
\text{Gating network} & \quad \eta \\
\text{Expert 1} & \quad w_1 \\
\text{Expert 2} & \quad w_2 \\
\ldots & \\
\text{Expert k} & \quad w_k \\
& \Rightarrow y
\end{align*}
\]

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

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

$g_i(x)$ - a prior \quad \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 $M_1$, $M_2$, … $M_k$
    - 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

**Diagram**

- **Data**
  - Data 1
  - Data 2
  - Data k

- **Bootstrap**

- Model $M_1$
- Model $M_2$
- Model $M_k$

- **Test**
  - For each test example
    - Run all base models $M_1$, $M_2$, … $M_k$
    - Predict by combining results of all $T$ trained models:
      - Regression: averaging
      - **Classification**: a majority vote
Class decision via majority voting

Test examples

- model_1
- model_2
- model_3
- Final

Class “yes”
Class “no”

Analysis of Bagging

- Expected error = Bias + Variance
  - Expected error is the expected discrepancy between the estimated and true function
  \[ E[(\hat{f}(X) - E[f(X)])^2] \]

  It decomposes to two terms Bias + Variance
  - Bias is a squared discrepancy between averaged estimated and true function
    \[ (E[\hat{f}(X)] - E[f(X)])^2 \]
  - Variance is an expected divergence of the estimated function vs. its average value
    \[ E[(\hat{f}(X) - E[\hat{f}(X)])^2] \]
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

  - **Case 1:** we draw one example/measurement $x_1$ 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

  ![Distribution Diagram]

  - **Case 2**: a variable $x$ is measured independently $K$ times $(x_1, x_2, \ldots, x_k)$ and the mean is estimated as:
    $$\mu' = (x_1 + x_2 + \ldots + x_k) / K,$$
    - The expected mean of the estimate $E[\mu'] = \mu$
    - But, the variance of the mean estimate $\text{Var}(\mu')$ is smaller:
      $$\text{Var}(\mu') = \frac{\text{Var}(x_1) + \ldots + \text{Var}(x_k)}{K^2} = \frac{\sigma^2}{K}$$

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
  – \((\epsilon, \delta)\) solution

• **PAC learning**:
  – Learning with a pre-specified error \(\epsilon\) and a confidence parameter \(\delta\)
  – the probability that the misclassification error (ME) is larger than \(\epsilon\) is smaller than \(\delta\)
    \[ P(ME(\epsilon) > \epsilon) \leq \delta \]

Alternative rewrite:

\[ P(Acc(\epsilon) > 1 - \epsilon) > (1 - \delta) \]

• **Accuracy** (1-\(\epsilon\)): 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 $\varepsilon < 1/2$, and
  – confidence $\delta < 1/2$
    or in other words:
    • classification accuracy $> (1-\varepsilon)$
    • confidence probability $> (1-\delta)$
• Outputs a classifier that satisfies this parameters
• Efficiency: runs in time polynomial in $1/\delta, 1/\varepsilon$
  – Implies: number of samples $N$ is polynomial in $1/\delta, 1/\varepsilon$

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 $\delta_o$ and maximal error $\varepsilon_o$
- 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

- 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 $< \varepsilon_0$, the final ‘voting’ classifier has error $< g(\varepsilon_0) = 3 \varepsilon_0^2 - 2\varepsilon_0^3$

• Accuracy improved !!!!

• Apply recursively to get to the target accuracy !!!

![Graph showing error vs. \varepsilon_0]

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**

![Diagram of AdaBoost training](image)

- **Distribution**
  - Uniform distribution $D_1$ of training examples
  - $P(\text{example i}) = 1/N$
AdaBoost training

1. **Distribution**:
   - Sample randomly according to $D_1$
   - And train Model 1

2. **Learn**:

3. **Test**:
   - Test Model 1 and calculate errors
AdaBoost training

Training data

Distribution

Learn

Test

D_1

Model 1

Errors 1

D_2

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

D_T

Model T

Errors T

Model 2

Errors 2

...
**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

![Sampling Probabilities](image1)

![Sampling Probabilities](image2)
AdaBoost classification

- We have $T$ different classifiers $h_t$.
  - weight $w_t$ of the classifier is proportional to its accuracy on the training set
    \[ w_t = \log(1 / \beta_t) = \log \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right) \]
    \[ \beta_t = \frac{\varepsilon_t}{1 - \varepsilon_t} \]
- **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)
    \[ h_{final}(x) = \arg \max_j \sum_{t: h_t(x) = j} w_t \]

Two-Class example. Classification.

- Classifier 1 \("yes"\) 0.7
- Classifier 2 \("no"\) 0.3
- Classifier 3 \("no"\) 0.2

\[ \begin{align*}
\text{Weighted majority \("yes"\)} & \quad 0.7 - 0.5 = +0.2 \\
\text{The final choice is \("yes"\) + 1} & 
\end{align*} \]
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 training error, test error, and single-learner error over iterations.](image)