• Motivation: let's say we have a number of models for a problem
  • e.g. Regression with polynomials (different degree)
  • e.g. Classification with support vector machines (kernel type, parameters)
• Often, improved performance can be obtained by combining different models
• But how can we combine them together?

• A combination of models is often called a committee
• Simplest way to combine models is to just average them together:
  \[ y_{COM}(x) = \frac{1}{M} \sum_{m=1}^{M} y_m(x) \]
• It turns out this simple method is better than (or same as) the individual models on average (in expectation)
  • And usually slightly better
• But there are better methods, which we shall discuss
Error of Individual Models

- Consider individual models \( y_m(x) \), assume they can be written as true value plus error:
  \[
  y_m(x) = h(x) + \epsilon_m(x)
  \]

- The expected value of the error of an individual model is then:
  \[
  E_x \left[ (y_m(x) - h(x))^2 \right] = E_x [\epsilon_m(x)^2]
  \]

- The average error made by an individual model is then:
  \[
  E_{AV} = \frac{1}{M} \sum_{m=1}^{M} E_x [\epsilon_m(x)^2]
  \]

Error of Committee

- The committee
  \[
  y_{COM}(x) = \frac{1}{M} \sum_{m=1}^{M} y_m(x)
  \]
  has expected error
  \[
  E_{COM} = E_x \left[ \left( \frac{1}{M} \sum_{m=1}^{M} y_m(x) - h(x) \right)^2 \right]
  \]
  \[
  = E_x \left[ \left( \frac{1}{M} \sum_{m=1}^{M} h(x) + \epsilon_m(x) - h(x) \right)^2 \right]
  \]
  \[
  = E_x \left[ \left( \frac{1}{M} \sum_{m=1}^{M} \epsilon_m(x) \right)^2 \right] = E_x \left[ \left( \frac{1}{M} \sum_{m=1}^{M} \epsilon_m(x) \right)^2 \right]
  \]

Committee Error vs. Individual Error

- So, the committee error is
  \[
  E_{COM} = E_x \left[ \left( \frac{1}{M} \sum_{m=1}^{M} \epsilon_m(x) \right)^2 \right] = \frac{1}{M^2} \sum_{m=1}^{M} \sum_{n=1}^{M} E_x [\epsilon_m(x) \epsilon_n(x)]
  \]

- If we assume errors are uncorrelated, \( E_x [\epsilon_m(x) \epsilon_n(x)] = 0 \) when \( m \neq n \), then:
  \[
  E_{COM} = \frac{1}{M^2} \sum_{m=1}^{M} E_x [\epsilon_m(x)^2] = \frac{1}{M} E_{AV}
  \]

- However, errors are rarely uncorrelated
  - For example, if all errors are the same, \( \epsilon_m(x) = \epsilon_n(x) \), then
    \[
    E_{COM} = E_{AV}
    \]
    Using Jensen’s inequality (convex functions), can show
    \[
    E_{COM} \leq E_{AV}
    \]

Outline

- Boosting
- Decision Trees
- Mixture of Experts
Boosting is a technique for combining classifiers into a committee.

- We describe AdaBoost (adaptive boosting), the most commonly used variant.

Boosting is a meta-learning technique.

- Combines a set of classifiers trained using their own learning algorithms.
- Magic: can work well even if those classifiers only perform slightly better than random!

### Boosting Model

- We consider two-class classification problems, training data \((x_i, t_i)\), with \(t_i \in \{-1, 1\}\).
- In boosting we build a “linear” classifier of the form:

\[
y(x) = \sum_{m=1}^{M} \alpha_m y_m(x)
\]

- A committee of classifiers, with weights.
- In boosting terminology:
  - Each \(y_m(x)\) is called a weak learner or base classifier.
  - Final classifier \(y(x)\) is called strong learner.
- Learning problem: how do we choose the weak learners \(y_m(x)\) and weights \(\alpha_m\)?

### Choosing Weak Learners

- Boosting is a greedy strategy for building the strong learner.

\[
y(x) = \sum_{m=1}^{M} \alpha_m y_m(x)
\]

- Start by choosing the best weak learner, use it as \(y_1(x)\).
  - Best is defined as that which minimizes number of mistakes made (0-1 classification loss).
- i.e. Search over all \(p, \theta, i\) to find best:

\[
y_m(x) = px_i > p\theta
\]
The first weak learner $y_1(x)$ made some mistakes
Choose the second weak learner $y_2(x)$ to try to get those ones correct
Best is now defined as that which minimizes weighted number of mistakes made
Higher weight given to those $y_1(x)$ got incorrect
Strong learner now $y(x) = \alpha_1 y_1(x) + \alpha_2 y_2(x)$

Repeat: reweight examples and choose new weak learner based on weights
Green line shows decision boundary of strong learner

So exactly how should we choose the weights for the examples when classified incorrectly?
And what should the $\alpha_m$ be for combining the weak learners $y_m(x)$?
As usual, we define a loss function, and choose these parameters to minimize it

Boosting attempts to minimize the exponential loss
$$E_n = \exp\{-t_n y(x_n)\}$$
error on $n^{th}$ training example
Exponential loss is differentiable approximation to 0/1 loss
Better for optimization
Total error
$$E = \sum_{n=1}^{N} \exp\{-t_n y(x_n)\}$$

Exponential loss function
We will use the exponential loss to measure the quality of the classifier:
$$L(H(x), y) = \exp\{-y \cdot H(x)\}$$
– Easy to optimize
Other options are possible.

Figure from G. Shakhnarovich
Boosting Decision Trees Mixture of Experts

Minimizing Exponential Loss

- Let's assume we've already chosen weak learners $y_1(x), \ldots, y_{m-1}(x)$ and their weights $\alpha_1, \ldots, \alpha_{m-1}$
- Define $f_{m-1}(x) = \alpha_1 y_1(x) + \ldots + \alpha_{m-1} y_{m-1}(x)$
- Just focus on choosing $y_m(x)$ and $\alpha_m$
- Greedy optimization strategy
- Total error using exponential loss is:

$$E = \sum_{n=1}^{N} \exp\{-t_n y(x_n)\} = \sum_{n=1}^{N} \exp\{-t_n f_{m-1}(x_n) + \alpha_m y_m(x_n)\}$$

$$= \sum_{n=1}^{N} \exp\{-t_n f_{m-1}(x_n)\} - t_n \alpha_m y_m(x_n)$$

$$= \sum_{n=1}^{N} \exp\{-t_n f_{m-1}(x_n)\} \exp\{-t_n \alpha_m y_m(x_n)\}$$

weight $\alpha_m$

Minimization wrt $y_m$

- Consider the weighted loss

$$E = \sum_{n=1}^{N} w_n^{(m)} \exp\{-t_n \alpha_m y_m(x_n)\} = e^{-\alpha_m} \sum_{n \in \mathcal{T}_m} w_n^{(m)} + e^{\alpha_m} \sum_{n \in \mathcal{N}_m} w_n^{(m)}$$

where $\mathcal{T}_m$ is the set of points correctly classified by the choice of $y_m(x)$, and $\mathcal{N}_m$ those that are not

$$E = e^{\alpha_m} \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n) + e^{-\alpha_m} \sum_{n=1}^{N} w_n^{(m)} (1 - I(y_m(x_n) \neq t_n))$$

$$= (e^{\alpha_m} - e^{-\alpha_m}) \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n) + e^{-\alpha_m} \sum_{n=1}^{N} w_n^{(m)}$$

- Since the second term is a constant wrt $y_m$ and $e^{\alpha_m} - e^{-\alpha_m} > 0$ if $\alpha_m > 0$, best $y_m$ minimizes weighted 0-1 loss

Weighted Loss

- On the $m^{th}$ iteration of boosting, we are choosing $y_m$ and $\alpha_m$ to minimize the weighted loss:

$$E = \sum_{n=1}^{N} w_n^{(m)} \exp\{-t_n \alpha_m y_m(x_n)\}$$

where $w_n^{(m)} = \exp\{-t_n f_{m-1}(x_n)\}$

- Can define these as weights since they are constant wrt $y_m$ and $\alpha_m$
- We'll see they're the right weights to use

Choosing $\alpha_m$

- So best $y_m$ minimizes weighted 0-1 loss regardless of $\alpha_m$
- How should we set $\alpha_m$ given this best $y_m$?
- Recall from above:

$$E = e^{\alpha_m} \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n) + e^{-\alpha_m} \sum_{n=1}^{N} w_n^{(m)} (1 - I(y_m(x_n) \neq t_n))$$

$$= e^{\alpha_m} \epsilon_m + e^{-\alpha_m} (1 - \epsilon_m)$$

where we define $\epsilon_m$ to be the weighted error of $y_m$

- Calculus: $\alpha_m = \frac{1}{2} \log \frac{1 - \epsilon_m}{\epsilon_m}$
AdaBoost Summary

- Initialize weights \( w_1^{(1)} = \frac{1}{N} \)
- For \( m = 1, \ldots, M \) (and while \( \epsilon_m < \frac{1}{2} \))
  - Find weak learner \( y_m(x) \) with minimum weighted error
    \[
    \epsilon_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n)
    \]
  - Set \( \alpha_m = \frac{1}{2} \log \frac{1}{\epsilon_m} \)
  - Update weights
    \[
    w_n^{(m+1)} = w_n^{(m)} \exp \{-\alpha_m t_n y_m(x_n)\}
    \]
  - Normalize weights to sum to one
- Final classifier is
  \[
  y(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m y_m(x) \right)
  \]

Adaboost Behaviour

- Typical behaviour:
  - Test error decreases even after training error is flat (even zero!)
  - Tends not to overfit

Boosting the Margin

- Define the margin of an example:
  \[
  \gamma(x_i) = t_i \frac{\alpha_1 y_1(x_i) + \ldots + \alpha_m y_m(x_i)}{\alpha_1 + \ldots + \alpha_m}
  \]
- Margin is 1 if all \( y_i \) classify correctly, -1 if none do
- Iterations of AdaBoost increase the margin of training examples (even after training error is zero)

Loss Functions for Classification

- We revisit a graph from earlier: 0-1 loss, SVM hinge loss, logistic regression cross-entropy loss, and AdaBoost exponential loss are shown
- All are approximations (upper bounds) to 0-1 loss
- Exponential loss leads to simple greedy optimization scheme
- But it has problems with outliers: note behaviour compared to logistic regression cross-entropy loss for badly mis-classified examples
The boosting method for building a committee builds a model:

\[ y(x) = \sum_{m=1}^{M} \alpha_m y_m(x) \]

Note that the committee is built over all input space

- Though it can of course behave differently in different regions

Instead, we could explicitly carve up input space into different regions \( R_m \) and have different committee members act in different regions:

\[ y(x) = \sum_{m=1}^{M} 1_{R_m}(x) y_m(x) \]

where \( 1_{R_m}(\cdot) \) is the indicator function (0 or 1)

A common method for carving up input space is to use axis-aligned cuboid-shaped regions

- Each model \( y_m(x) \) would only be responsible for one subregion

These splits are commonly chosen in a top-down fashion to form a binary tree

- These are known as decision trees

Given a dataset, the learning problem is to decide which is the best tree

- There are (exponentially-exponentially) many different trees to choose from

Brute force impossible, so use a greedy strategy

- Start with an empty tree
- Choose a dimension \( i \) and value \( \theta \) on which to split
- Make recursive calls
  - Some training examples \( X_L \) go down left branch, recursive call with those
  - Other training examples \( X_R \) go down right branch, a second recursive call with those
Example - Waiting for Table

<table>
<thead>
<tr>
<th>Example</th>
<th>Full</th>
<th>Waiter</th>
<th>Egg</th>
<th>Soup</th>
<th>Time</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>S5</td>
</tr>
<tr>
<td>X2</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>X3</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>10–60</td>
</tr>
<tr>
<td>X4</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>S</td>
<td>F</td>
<td>35</td>
</tr>
<tr>
<td>X5</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>S5</td>
</tr>
<tr>
<td>X6</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>35</td>
</tr>
<tr>
<td>X7</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>35</td>
</tr>
<tr>
<td>X8</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>35</td>
</tr>
<tr>
<td>X9</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>35</td>
</tr>
<tr>
<td>X10</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>35</td>
</tr>
<tr>
<td>X11</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>35</td>
</tr>
<tr>
<td>X12</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>35</td>
</tr>
</tbody>
</table>

Choosing a Dimension

- Of all the dimensions one could choose to put at root of decision tree, which is best?
- Compare using Patrons? versus Type?
  - Patrons? looks better – more information about classification

Information

- Information answers questions
- The more clueless I am about the answer initially, the more information is in the answer
- Scale: 1 bit = answer to Boolean question with prior\( p(x = \text{true}) = 0.5 \)
- For a K-class classification problem, we have a prior\( p(x = k) = \pi_k \)
- Information in answer is
  \[
  H(x) = - \sum_{k=1}^{K} \pi_k \log_2 \pi_k
  \]

known as entropy of prior
- A good dimension produces a split that reduces entropy

Entropy

- Entropy for binary classification (boolean prior)
- \( H(x) = -\pi_1 \log_2 \pi_1 - (1 - \pi_1) \log_2(1 - \pi_1) \)
Choosing an attribute

Idea: a good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative.”

- Compare using Patrons? versus Type?
  - Patrons? has average entropy of 0.459 bits
  - Type? has average entropy of 1 bit
- Put Patrons? at root of tree
  - Make recursive calls using training examples that fall down each path

Mixture of Experts

- The mixture of experts model takes the idea of splitting up regions of space in a probabilistic direction.
- The decision on which model to use is probabilistic:
  \[ p(t|x) = \sum_{m=1}^{M} \pi_m(x)p_m(t|x) \]
- Note that all models \( p_m(t|x) \) are used
- But contributions \( \pi_m(x) \) depend on input variable \( x \)
  - These coefficients \( \pi_m(x) \) are known as gating functions
  - Each \( p_m(t|x) \) is an expert in a region of input space, the gating functions determine when to use each expert

Learnt Tree

- At each leaf have an expert
- In this case, just report what type of examples are in this region of input space
  - More generally, could stop earlier, build classifier in each region

Conclusion

- Readings: Ch. 14.3, 14.4
- Methods for combining models
  - Simple averaging into a committee
  - Greedy selection of models to minimize exponential loss (AdaBoost)
  - Select models which are good at particular regions of input space (decision trees, mixture of experts)