Lecture 19:

- What is Ensemble Learning?
- Bagging
- Random Forests
- Boosting
Last time... Decision Trees

- TriageTemp \( \leq 100.9500 \\) error = 0.381403449603 \\ samples = 1657 \\ value = [ 425, 1232 ]

- Age \( \leq 41.5000 \\) error = 0.465677281203 \\ samples = 729 \\ value = [ 269, 460 ]

- TriageSBP \( \leq 112.5000 \\) error = 0.279689357907 \\ samples = 928 \\ value = [ 156, 772 ]

- TriageHR \( \leq 108.5000 \\) error = 0.496029801 \\ samples = 303 \\ value = [ 138, 165 ]

- TriagePain \( \leq 0.5000 \\) error = 0.425896537283 \\ samples = 426 \\ value = [ 131, 295 ]

- error = 0.1413 \\ samples = 183 \\ value = [ 14, 169 ]

- TriageHR \( \leq 98.5000 \\) error = 0.462615966797 \\ samples = 256 \\ value = [ 93, 163 ]

- Age \( \leq 51.5000 \\) error = 0.366668906208 \\ samples = 244 \\ value = [ 59, 185 ]

- TriageRR \( \leq 18.5000 \\) error = 0.276445113764 \\ samples = 501 \\ value = [ 83, 418 ]

- error = 0.4861 \\ samples = 132 \\ value = [ 55, 77 ]

- error = 0.4251 \\ samples = 124 \\ value = [ 38, 101 ]

- error = 0.4064 \\ samples = 103 \\ value = [ 19, 84 ]

- error = 0.3118 \\ samples = 326 \\ value = [ 63, 263 ]

- error = 0.2024 \\ samples = 175 \\ value = [ 20, 155 ]
Last time... Information Gain

- Decrease in entropy (uncertainty) after splitting

\[
IG(X) = H(Y) - H(Y \mid X)
\]

In our running example:

\[
IG(X_1) = H(Y) - H(Y \mid X_1)
\]

\[
= 0.65 - 0.33
\]

\[
IG(X_1) > 0 \Rightarrow \text{we prefer the split!}
\]
Last time... Continuous features

- Binary tree, split on attribute $X$
  - One branch: $X < t$
  - Other branch: $X \geq t$
- Search through possible values of $t$
  - Seems hard!!!
- But only a finite number of $t$'s are important:
  - Sort data according to $X$ into \{x_1, ..., x_m\}
  - Consider split points of the form $x_i + (x_{i+1} - x_i)/2$
  - Moreover, only splits between examples from different classes matter!

Optimal splits for continuous attributes

Infinitely many possible split points $c$ to define node test $X_j > c$?

No! Moving split point along the empty space between two observed values has no effect on information gain or empirical loss; so just use midpoint $X_j c_1 c_2$

Moreover, only splits between examples from different classes can be optimal for information gain or empirical loss reduction $X_j c_2 c_1$
Last time… Decision trees will overfit

• Standard decision trees have no learning bias
  - Training set error is always zero!
    - (If there is no label noise)
  - Lots of variance
  - Must introduce some bias towards simpler trees

• Many strategies for picking simpler trees
  - Fixed depth
  - Fixed number of leaves

• Random forests
Today

• Ensemble Methods
  - Bagging
    • Random Forests
  - Boosting
Ensemble Methods

• High level idea
  – Generate multiple hypotheses
  – Combine them to a single classifier

• Two important questions
  – How do we generate multiple hypotheses
    • we have only one sample
  – How do we combine the multiple hypotheses
    • Majority, AdaBoost, ...
Bias/Variance Tradeoff

Hastie, Tibshirani, Friedman “Elements of Statistical Learning” 2001
Bias/Variance Tradeoff

Graphical illustration of bias and variance.

http://scott.fortmann-roe.com/docs/BiasVariance.html
Fighting the bias-variance tradeoff

- **Simple (a.k.a. weak) learners are good**
  - e.g., naïve Bayes, logistic regression, decision stumps (or shallow decision trees)
  - Low variance, don’t usually overfit

- **Simple (a.k.a. weak) learners are bad**
  - High bias, can’t solve hard learning problems
Reduce Variance Without Increasing Bias

• Averaging reduces variance:

\[ \text{Var}(\overline{X}) = \frac{\text{Var}(X)}{N} \quad \text{(when prediction are independent)} \]

• Average models to reduce model variance

• One problem:
  - Only one training set
  - Where do multiple models come from?
Bagging (Bootstrap Aggregating)

- Leo Breiman (1994)
- Take repeated bootstrap samples from training set D.
- **Bootstrap sampling:** Given set D containing N training examples, create D’ by drawing N examples at random with replacement from D.

- Bagging:
  - Create \( k \) bootstrap samples \( D_1 \ldots D_k \).
  - Train distinct classifier on each \( D_i \).
  - Classify new instance by majority vote / average.
Bagging

• Best case:
  \[ \text{Var}(\text{Bagging}(L(x, D))) = \frac{\text{Var}(L(x, D))}{N} \]

• In practice:
  - models are correlated, so reduction is smaller than 1/N
  - variance of models trained on fewer training cases usually somewhat larger
Bagging Example
CART* decision boundary

* A decision tree learning algorithm; very similar to ID3
100 bagged trees

- Shades of blue/red indicate strength of vote for particular classification
Random Forests
Random Forests

• Ensemble method specifically designed for decision tree classifiers

• Introduce two sources of randomness: “Bagging” and “Random input vectors”
  - **Bagging method:** each tree is grown using a bootstrap sample of training data
  - **Random vector method:** At each node, best split is chosen from a random sample of m attributes instead of all attributes
A generic data point is denoted by a vector $\mathbf{v} = (x_1, x_2, \cdots, x_d)$

$S_j = S_j^L \cup S_j^R$
Use information gain to decide splits

\[ I_j = H(S_j) - \sum_{i \in \{L,R\}} \frac{|S_j^i|}{|S_j|} H(S_j^i) \]

[Criminisi et al., 2011]
Advanced: Gaussian information gain to decide splits

\[ H(S) = \frac{1}{2} \log \left( (2\pi e)^d |\Lambda(S)| \right) \]

[Criminisi et al., 2011]
Each split node $j$ is associated with a binary split function $h(v, \theta) \in \{\text{true}, \text{false}\}$.

\[ \theta_j = \arg \max_{\theta \in \mathcal{T}_j} I \]

\[ I_j = H(S_j) - \sum_{i \in \{L,R\}} \frac{|S_j^i|}{|S_j|} H(S_j^i) \]

[Crinini et al., 2011]
**Alternative node decisions**

\[ \mathbf{v} = (x_1 \ x_2) \in \mathbb{R}^2 \quad \phi(\mathbf{v}) = (x_1 \ x_2 \ 1)^T \]

\[ h(\mathbf{v}, \theta) = [\tau_1 > \phi(\mathbf{v}) \cdot \psi > \tau_2] \]

axis aligned  
oriented line  
conic section

examples of weak learners
Building a random tree

$d=3$ features
$h=5$ data

\[ X = \begin{bmatrix}
  i=1 & i=2 & i=5 \\
 j=1 & 1 & 3 & 0 & 8 & 5 \\
 j=2 & 0 & 6 & 2 & 9 & 5 \\
 j=3 & 2 & 1 & 4 & 0 & 1 \\
\end{bmatrix} \quad Y = \begin{bmatrix}
  i=1 & i=2 & i=5 \\
  0 & 1 & 0 & 1 & 0 \\
\end{bmatrix} \]

Pick 2 features at random.

Split point
Random Forests algorithm

1. For \( b = 1 \) to \( B \):

   (a) Draw a bootstrap sample \( Z^* \) of size \( N \) from the training data.

   (b) Grow a random-forest tree \( T_b \) to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size \( n_{min} \) is reached.

      i. Select \( m \) variables at random from the \( p \) variables.

      ii. Pick the best variable/split-point among the \( m \).

      iii. Split the node into two daughter nodes.

2. Output the ensemble of trees \( \{ T_b \}_{1}^{B} \).

[From the book of Hastie, Friedman and Tibshirani]
Randomized node optimization. If $\mathcal{T}$ is the entire set of all possible parameters $\theta$ then when training the $j^{th}$ node we only make available a small subset $\mathcal{T}_j \subset \mathcal{T}$ of such values.

$$\theta_j^* = \arg \max_{\theta_j \in \mathcal{T}_j} I_j.$$
Building a forest (ensemble)

In a forest with $T$ trees we have $t \in \{1, \ldots, T\}$. All trees are trained independently (and possibly in parallel). During testing, each test point $v$ is simultaneously pushed through all trees (starting at the root) until it reaches the corresponding leaves.

$$p(c|v) = \frac{1}{T} \sum_{t} p_{t}(c|v)$$
Effect of forest size

(a) Training points

(b) Forest training

(c) Forest testing

T=1

T=8

T=200

Tree 43

Tree 121

Learnt decision boundaries

Learnt trees

x_1

x_2
Effect of forest size
Effect of more classes and noise

Example 1 - two-class spiral
Example 2 - four-class spiral
Example 3 - noisier four-class spiral

Training points

Testing posteriors

slide by Nando de Freitas
Effect of more classes and noise
Effect of tree depth (D)

Training points: 4-class mixed

D=3 (underfitting)

D=6

D=15 (overfitting)
Effect of bagging

Randomized node optimization (RNO)

Bagging (50%) and RNO

no bagging => max-margin
Random Forests and the Kinect
Random Forests and the Kinect

From a single input depth image, a per-pixel segmentation into body parts is performed. This segmentation is then used to generate 3D joint proposals for the 3D locations of each skeletal joint, even for multiple users. This approach handles self-occlusions and variations in appearance, making it suitable for real-time human body tracking.

The system runs at 200 frames per second on consumer hardware, making it suitable for gaming applications and telepresence, among others. It demonstrates high accuracy in comparison with related work, even for challenging cases.

This work is inspired by the Kinect gaming platform and introduces a new method for quick and accurate pose recognition using depth images.
Random Forests and the Kinect

- Use computer graphics to generate plenty of data

Real-Time Human Pose Recognition in Parts from Single Depth Images

CVPR 2011

Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, Andrew Blake
Microsoft Research Cambridge & Xbox Incubation
Reduce Bias$^2$ and Decrease Variance?

- Bagging reduces variance by averaging
- Bagging has little effect on bias
- Can we average and reduce bias?
- Yes: Boosting
Boosting
Boosting Ideas

- Main idea: use weak learner to create strong learner.
- Ensemble method: combine base classifiers returned by weak learner.
- Finding simple relatively accurate base classifiers often not hard.
- But, how should base classifiers be combined?
Example: “How May I Help You?”

- **Goal:** automatically categorize type of call requested by phone customer (Collect, CallingCard, PersonToPerson, etc.)
  - yes I’d like to place a collect call long distance please (Collect)
  - operator I need to make a call but I need to bill it to my office (ThirdNumber)
  - yes I’d like to place a call on my master card please (CallingCard)
  - I just called a number in sioux city and I musta rang the wrong number because I got the wrong party and I would like to have that taken off of my bill (BillingCredit)

- **Observation:**
  - easy to find “rules of thumb” that are “often” correct
    - e.g.: “IF ‘card’ occurs in utterance THEN predict ‘CallingCard’ ”
  - hard to find single highly accurate prediction rule

[Gorin et al.]
Boosting: Intuition

• Instead of learning a single (weak) classifier, learn many weak classifiers that are good at different parts of the input space

• Output class: (Weighted) vote of each classifier
  - Classifiers that are most “sure” will vote with more conviction
  - Classifiers will be most “sure” about a particular part of the space
  - On average, do better than single classifier!

• But how do you???
  - force classifiers to learn about different parts of the input space?
  - weigh the votes of different classifiers?
Boosting [Schapire, 1989]

- **Idea:** given a weak learner, run it multiple times on (rewighted) training data, then let the learned classifiers vote

- **On each iteration** $t$:
  - weight each training example by how incorrectly it was classified
  - Learn a hypothesis – $h_t$
  - A strength for this hypothesis – $a_t$

- **Final classifier**:
  - A linear combination of the votes of the different classifiers weighted by their strength $H(X) = \text{sign} \left( \sum a_t h_t(X) \right)$

- **Practically useful**
- **Theoretically interesting**
Boosting: Intuition

• Want to pick weak classifiers that contribute something to the ensemble

Greedy algorithm: for \( m=1, \ldots, M \)

• Pick a weak classifier \( h_m \)

• Adjust weights: misclassified examples get “heavier”

• \( \alpha_m \) set according to weighted error of \( h_m \)

[Source: G. Shakhnarovich]
Boosting: Intuition

- Want to pick weak classifiers that contribute something to the ensemble

Greedy algorithm: for $m=1,...,M$

- Pick a weak classifier $h_m$
- Adjust weights: misclassified examples get “heavier”
- $\alpha_m$ set according to weighted error of $h_m$

[Source: G. Shakhnarovich]
Boosting: Intuition

• Want to pick weak classifiers that contribute something to the ensemble

Greedy algorithm: for $m=1,...,M$

• Pick a weak classifier $h_m$
• Adjust weights: misclassified examples get “heavier”
• $\alpha_m$ set according to weighted error of $h_m$

[Source: G. Shakhnarovich]
Boosting: Intuition

• Want to pick weak classifiers that contribute something to the ensemble

Greedy algorithm: for $m=1,...,M$

• Pick a weak classifier $h_m$
• Adjust weights: misclassified examples get “heavier”
• $\alpha_m$ set according to weighted error of $h_m$

[Source: G. Shakhnarovich]
Boosting: Intuition

• Want to pick weak classifiers that contribute something to the ensemble

Greedy algorithm: for $m=1,...,M$

• Pick a weak classifier $h_m$

• Adjust weights: misclassified examples get “heavier”

• $\alpha_m$ set according to weighted error of $h_m$

[Source: G. Shakhnarovich]
Boosting: Intuition

• Want to pick weak classifiers that contribute something to the ensemble

Greedy algorithm: for $m=1,...,M$

• Pick a weak classifier $h_m$
• Adjust weights: misclassified examples get “heavier”
• $\alpha_m$ set according to weighted error of $h_m$

[Source: G. Shakhnarovich]
Boosting: Intuition

- Want to pick weak classifiers that contribute something to the ensemble

Greedy algorithm: for $m=1,\ldots,M$
- Pick a weak classifier $h_m$
- Adjust weights: misclassified examples get “heavier”
- $\alpha_m$ set according to weighted error of $h_m$
First Boosting Algorithms

- [Schapire ’89]:
  - first provable boosting algorithm
- [Freund ’90]:
  - “optimal” algorithm that “boosts by majority”
- [Drucker, Schapire & Simard ’92]:
  - first experiments using boosting
  - limited by practical drawbacks
- [Freund & Schapire ’95]:
  - introduced “AdaBoost” algorithm
  - strong practical advantages over previous boosting algorithms
The AdaBoost Algorithm

Given: \((x_1, y_1), \ldots, (x_m, y_m)\) where \(x_i \in X, y_i \in Y = \{-1, +1\}\)

Initialize \(D_1(i) = 1/m\).

For \(t = 1, \ldots, T\):

- Train \textbf{weak} learner using distribution \(D_t\).
- Get \textbf{weak} classifier \(h_t : X \to \mathbb{R}\).
- Choose \(\alpha_t \in \mathbb{R}\).
- Update:

\[
D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}
\]

where \(Z_t\) is a normalization factor

\[
Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i))
\]

Output the final classifier:

\[
H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right).
\]
Minimize the error

\[ \epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i] \]

For binary \( h_t \), typically use

\[ \alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) \]

weak hypotheses = vertical or horizontal half-planes
Round 1

$h_1$
Round 1

$h_1$

$\varepsilon_1 = 0.30$

$\alpha_1 = 0.42$
Round 1

\[ h_1 \]

\[ \varepsilon_1 = 0.30 \]
\[ \alpha_1 = 0.42 \]
Round 2

\[ \alpha_2 = 0.21 \]

\[ \varepsilon_2 = 0.65 \]
Round 2

\[ \epsilon_2 = 0.21 \]
\[ \alpha_2 = 0.65 \]
Round 2

\[ \alpha_2 = 0.65 \]

\[ \varepsilon_2 = 0.21 \]
Round 3

\[ \alpha = 0.92 \]

\[ \varepsilon = 0.14 \]
Round 3

\[ h_3 \]

\[ \varepsilon_3 = 0.14 \]

\[ \alpha_3 = 0.92 \]
Final Hypothesis

\[ H_{\text{final}} = \text{sign} \left( 0.42 + 0.65 + 0.92 \right) \]
Demo time…