Instructor: Herke van Hoof (herke.vanhoof@mcgill.ca)

Slides mostly by: Joelle Pineau (jpineau@cs.mcgill.ca)

Class web page: www.cs.mcgill.ca/~hvanho2/comp551
Main types of machine learning problems

- **Supervised learning**
  - Classification
  - Regression
  - Ensemble methods

- **Unsupervised learning**

- **Reinforcement learning**
Next topic: Ensemble methods

• Recently seen supervised learning methods:
  – Logistic regression, Naïve Bayes, LDA/QDA
  – Decision trees, Instance-based learning

• Decision trees? Build complex classifiers from simpler ones. (Linear separator)

• Ensemble methods use this idea with other ‘simple’ methods

• Several ways to do this.
  – Bagging
  – Random forests
  – Boosting
  – Stacking (Next lecture)
Ensemble learning in general

- **Key idea**: Run one or more base learning algorithms multiple times, then combine the predictions of the different learners to get a final prediction.
  - What’s a base learning algorithm?
    - Naïve Bayes, LDA, Decision trees, SVMs, …
Ensemble learning in general

• Key idea: Run one or more base learning algorithms multiple times, then combine the predictions of the different learners to get a final prediction.
  – What’s a base learning algorithm?
    • Naïve Bayes, LDA, Decision trees, SVMs, …

• First attempt: Construct several classifiers independently.
  – Bagging.
  – Randomizing the test selection in decision trees (Random forests).
  – Using a different subset of input features to train different trees.
Ensemble learning in general

- **Key idea**: Run one or more base learning algorithms multiple times, then combine the predictions of the different learners to get a final prediction.

  - What’s a base learning algorithm?
    - Naïve Bayes, LDA, Decision trees, SVMs, ...

- **First attempt**: Construct several classifiers **independently**.
  - Bagging.
  - Randomizing the test selection in decision trees (Random forests).
  - Using a different subset of input features to train different trees.

- **More complex approach**: Coordinate the construction of the hypotheses in the ensemble.
Ensemble methods in general

• Training models independently on same dataset tends to yield same result!

• For an ensemble to be useful, trained models need to be different
  1. Use slightly different (randomized) datasets
  2. Use (slightly) different (e.g. randomized) training procedure
Recall bootstrapping

• Given dataset $D$, construct a bootstrap replicate of $D$, called $D_k$, which has the same number of examples, by drawing samples from $D$ with replacement.

• Use the learning algorithm to construct a hypothesis $h_k$ by training on $D_k$.

• Compute the prediction of $h_k$ on each of the remaining points, from the set $T_k = D - D_k$.

• Repeat this process $K$ times, where $K$ is typically a few hundred.
Estimating bias and variance

• For each point $x$, we have a set of estimates $h_1(x), \ldots, h_K(x)$, with $K \leq B$ (since $x$ might not appear in some replicates).

• The average empirical prediction of $x$ is:
  \[
  \hat{h}(x) = \frac{1}{K} \sum_{k=1}^{K} h_k(x).
  \]

• We estimate the bias as: $y - \hat{h}(x)$.

• We estimate the variance as: $(1/(K-1)) \sum_{k=1}^{K} (\hat{h}(x) - h_k(x))^2$. 
Bagging: Bootstrap aggregation

- If we did all the work to get the hypotheses $h_b$, why not use all of them to make a prediction? (as opposed to just estimating bias/variance/error).

- All hypotheses get to have a vote.
  - For classification: pick the majority class.
  - For regression, average all the predictions.

- Which hypotheses classes would benefit most from this approach?
Bagging

• For each point $x$, we have a set of estimates $h_1(x), \ldots, h_K(x)$, with $K \leq B$ (since $x$ might not appear in some replicates).
  – The average empirical prediction of $x$ is: $\hat{h}(x) = \frac{1}{K} \sum_{k=1}^{K} h_k(x)$.
  – We estimate the bias as: $y - \hat{h}(x)$.
  – We estimate the variance as: $\frac{1}{(K-1)} \sum_{k=1}^{K} (\hat{h}(x) - h_k(x))^2$.

• In theory, bagging eliminates variance altogether.
• In practice, bagging tends to reduce variance and increase bias.
• Use this with “unstable” learners that have high variance, e.g. decision trees, neural networks, nearest-neighbour.
Random forests (Breiman, 2001)

- Basic algorithm:
  - Use $K$ bootstrap replicates to train $K$ different trees.
  - At each node, pick $m$ variables at random (use $m<M$, the total number of features).
  - Determine the best test (using normalized information gain).
  - Recurse until the tree reaches maximum depth (no pruning).
Random forests (Breiman, 2001)

- Basic algorithm:
  - Use $K$ bootstrap replicates to train $K$ different trees.
  - At each node, pick $m$ variables at random (use $m<M$, the total number of features).
  - Determine the best test (using normalized information gain).
  - Recurse until the tree reaches maximum depth (no pruning).

- Comments:
  - Each tree has high variance, but the ensemble uses averaging, which reduces variance.
  - Random forests are very competitive in both classification and regression, but still subject to overfitting.
Extremely randomized trees (Geurts et al., 2006)

• Basic algorithm:
  – Construct $K$ decision trees.
  – Pick $m$ attributes at random (without replacement) and pick a random test involving each attribute.
  – Evaluate all tests (using a normalized information gain metric) and pick the best one for the node.
  – Continue until a desired depth or a desired number of instances ($n_{\text{min}}$) at the leaf is reached.
Extremely randomized trees (Geurts et al., 2005)

- **Basic algorithm:**
  - Construct $K$ decision trees.
  - Pick $m$ attributes at random (without replacement) and pick a random test involving each attribute.
  - Evaluate all tests (using a normalized information gain metric) and pick the best one for the node.
  - Continue until a desired depth or a desired number of instances ($n_{\text{min}}$) at the leaf is reached.

- **Comments:**
  - Very reliable method for both classification and regression.
  - The smaller $m$ is, the more randomized the trees are; small $m$ is best, especially with large levels of noise. Small $n_{\text{min}}$ means less bias and more variance, but variance is controlled by averaging over trees.
  - Compared to single trees, can pick smaller $n_{\text{min}}$ (less bias)
Randomization

- For an ensemble to be useful, trained models need to be different
  1. Use slightly different (randomized) datasets
     - Bootstrap Aggregation (Bagging)
  2. Use slightly different (randomized) training procedure
     - Extremely randomized trees, Random Forests
Randomization in general

- Instead of searching very hard for the best hypothesis, generate lots of random ones, then average their results.
- Examples:
  - Random feature selection  Random projections.
- Advantages?

- Disadvantages?
Randomization in general

• Instead of searching very hard for the best hypothesis, generate lots of random ones, then average their results.

• Examples:

• Advantages?
  – Very fast, easy, can handle lots of data.
  – Can circumvent difficulties in optimization.
  – Averaging reduces the variance introduced by randomization.

• Disadvantages?
Randomization in general

- Instead of searching very hard for the best hypothesis, generate lots of random ones, then average their results.

- Examples:
  - Random feature selection  Random projections.

- Advantages?
  - Very fast, easy, can handle lots of data.
  - Can circumvent difficulties in optimization.
  - Averaging reduces the variance introduced by randomization.

- Disadvantages?
  - New prediction may be more expensive to evaluate (go over all trees).
  - Still typically subject to overfitting.
  - Low interpretability compared to standard decision trees.
Randomization

- For an ensemble to be useful, trained models need to be different
  1. Use slightly different (randomized) datasets
     • Bootstrap Aggregation (Bagging)
  2. Use slightly different (randomized) training procedure
     • Extremely randomized trees, Random Forests
  3. Alternative method to randomization?
Additive models

• In an ensemble, the output on any instance is computed by averaging the outputs of several hypotheses.

• **Idea:** Don’t construct the hypotheses independently. Instead, new hypotheses should focus on instances that are problematic for existing hypotheses.
  – If an example is difficult, more components should focus on it.
Boosting:

- Use the training set to train a simple predictor.
- Re-weight the training examples, putting more weight on examples that were not properly classified in the previous predictor.
- Repeat $n$ times.
- Combine the simple hypotheses into a single, accurate predictor.
Notation

• Assume that examples are drawn independently from some probability distribution $P$ on the set of possible data $D$.

• Let $J_P(h)$ be the expected error of hypothesis $h$ when data is drawn from $P$:

$$J_P(h) = \sum_{<x,y>} J(h(x),y) P(<x,y>)$$

where $J(h(x),y)$ could be the squared error, or 0/1 loss.
Weak learners

• Assume we have some “weak” binary classifiers:
  – A decision stump is a single node decision tree: \( x_i > t \)
  – A single feature Naïve Bayes classifier.
  – A 1-nearest neighbour classifier.

• “Weak” means \( J_P(h) < \frac{1}{2} - \gamma \) (assuming 2 classes), where \( \gamma > 0 \)
  – So true error of the classifier is only slightly better than random.

• Questions:
  – How do we re-weight the examples?
  – How do we combine many simple predictors into a single classifier?
Example
Example: First step

\[ h_1 \]

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

\[ D_2 \]
Example: Second step

$\varepsilon_2 = 0.21$
$\alpha_2 = 0.65$
Example: Third step

\[ h_3 \]

\[ \epsilon_3 = 0.14 \]

\[ \alpha_3 = 0.92 \]
Example: Final hypothesis

\[ H_{\text{final}} = \text{sign} \begin{pmatrix} 0.42 \ + \ 0.65 \ + \ 0.92 \end{pmatrix} \]
AdaBoost (Freund & Schapire, 1995)

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 \(t = 1, \ldots, T:\)
AdaBoost (Freund & Schapire, 1995)

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 weak learner using distribution \(D_t\).
- Get weak hypothesis \(h_t : X \rightarrow \{-1, +1\}\) with error

\[
\epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i].
\]
AdaBoost (Freund & Schapire, 1995)

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 weak learner using distribution \(D_t\).
- Get weak hypothesis \(h_t: X \rightarrow \{-1, +1\}\) with error
  \[
  \epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i].
  \]

Choose \(\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)\). weight of weak learner \(t\)
AdaBoost (Freund & Schapire, 1995)

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 weak learner using distribution \(D_t\).
- Get weak hypothesis \(h_t : X \rightarrow \{-1, +1\}\) with error
  \[\epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i].\]
- Choose \(\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t}\right)\).
- Update:
  \[D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases} = \frac{D_t(i) \exp(-\alpha_t y_t h_t(x_i))}{Z_t}.\]

where \(Z_t\) is a normalization factor (chosen so that \(D_{t+1}\) will be a distribution).
AdaBoost (Freund & Schapire, 1995)

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 weak learner using distribution \(D_t.\)
- Get weak hypothesis \(h_t : X \rightarrow \{-1, +1\}\) with error
  \[ \epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i]. \]
- Choose \(\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right).\)
- Update:
  \[
  D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} 
    e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\
    e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i 
  \end{cases}
  = \frac{D_t(i) \exp(-\alpha_i y_i h_t(x_i))}{Z_t}
  \]
  where \(Z_t\) is a normalization factor (chosen so that \(D_{t+1}\) will be a distribution).

Output the final hypothesis:
\[
H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right).
\]
Why these equations?

• Loss function:

\[ L = \sum_{i=1}^{N} e^{-m_i} \]

\[ m_i = y_i \sum_{k=1}^{K} \alpha_k h_k(x_i) \]

• Has a gradient

• Upper bound on classification loss

• Stronger signal for wrong classifications

• Stronger signal if wrong and far from boundary
Why these equations?

- Loss function:

\[
L = \sum_{i=1}^{N} e^{-m_i}
\]

\[
m_i = y_i \sum_{k=1}^{K} \alpha_k h_k(x_i)
\]

- Update equations are derived from this loss function
Properties of AdaBoost

- Compared to other boosting algorithms, main insight is to automatically adapt the error rate at each iteration.
Properties of AdaBoost

• Compared to other boosting algorithms, main insight is to automatically adapt the weights at each iteration.

• Training error on the final hypothesis is at most:

\[
\prod_t \left[ 2\sqrt{\epsilon_t(1 - \epsilon_t)} \right] = \prod_t \sqrt{1 - 4\gamma_t^2} \leq \exp \left( -2 \sum_t \gamma_t^2 \right)
\]

recall: \( \gamma_t \) is how much better than random is \( h_t \)

• AdaBoost gradually reduces the training error exponentially fast.
Real data set: Text categorization

database: AP

database: Reuters
Boosting empirical evaluation

Comparison of C4.5, Boosting C4.5, Boosting decision stumps (depth 1 trees), 27 benchmark datasets

C4.5: Lecture 7 – Decision Trees
Bagging vs Boosting

- Bagging is typically faster, but may get a smaller error reduction (not by much).
- Bagging works well with “reasonable” classifiers.
- Boosting works with very simple classifiers.
  E.g., Boostexter - text classification using decision stumps based on single words.
- Boosting may have a problem if a lot of the data is mislabeled, because it will focus on those examples a lot, leading to overfitting.
Why does boosting work?
Why does boosting work?

- Weak learners have high bias. By combining them, we get more expressive classifiers. Hence, boosting is a bias-reduction technique.
Why does boosting work?

- Weak learners have high bias. By combining them, we get more expressive classifiers. Hence, boosting is a bias-reduction technique.

- Adaboost minimizes an upper bound on the misclassification error, within the space of functions that can be captured by a linear combination of the base classifiers.

- What happens as we run boosting longer? Intuitively, we get more and more complex hypotheses. How would you expect bias and variance to evolve over time?
A naïve (but reasonable) analysis of error

- Expect the training error to continue to drop (until it reaches 0).
- Expect the test error to increase as we get more voters, and $h_f$ becomes too complex.
Actual typical run of AdaBoost

- Test error **does not increase** even after 1000 runs! (more than 2 million decision nodes!)
- Test error **continues to drop** even after training error reaches 0!
- These are consistent results through many sets of experiments!
- **Conjecture:** Boosting does not overfit!

![Graph showing test error decrease](image)
Other methods

• Random forests, extremely randomized trees, boosting and bagging all combine many learners of a single type

• Advantage: we have a ‘recipe’ for generating many classifiers by randomizing the dataset or the training procedure

• Disadvantages: since classifiers are of the same family they might make similar errors

• Next lecture: combining different types of learners
  (E.g. combine SVM + decision tree + LDA )
What you should know

• Ensemble methods combine several hypotheses into one prediction.

• They work better than the best individual hypothesis from the same class because they reduce bias or variance (or both).

• Random forests, Extremely randomized trees and bagging
  – Average over multiple independently trained classifiers, thus lower variance
  – Bagging is thus useful for complex hypotheses.
  – Can use more aggressive settings that would normally overfit: lower bias

• The classifiers in boosting are coordinated to lower error
  – Focuses on harder examples
  – Gives a weighted vote to the hypotheses.
  – Reduces the bias of simple hypotheses (not so useful for complex models).