Ensemble methods

• Why learn one classifier when you can learn many?

• Ensemble: combine many predictors
  – (Weighted) combinations of predictors
  – May be same type of learner or different
Ensemble methods

• Why learn one classifier when you can learn many?

• Ensemble: combine many predictors
  – (Weighted) combinations of predictors
  – May be same type of learner or different

“Who wants to be a millionaire?”
Ensemble methods

- Why learn one classifier when you can learn many?

- Ensemble: combine many predictors
  - (Weighted) combinations of predictors
  - May be same type of learner or different

“Who wants to be a millionaire?”

Various options for getting help:
Simple ensembles

- “Committees”
  - Unweighted average / majority vote
Simple ensembles

- “Committees”
  - Unweighted average / majority vote

- Weighted averages
  - Up-weight “better” predictors
  - Ex: Classes: +1 , -1 , weights \( \alpha \):
    \[
    \hat{y}_1 = f_1(x_1, x_2, \ldots) \\
    \hat{y}_2 = f_2(x_1, x_2, \ldots) \\
    \Rightarrow \hat{y}_e = \text{sign}(\sum \alpha_i \hat{y}_i)
    \]
Simple ensembles

• One option: train a “predictor of predictors”
  – Treat individual predictors as features
    \[ \hat{y}_1 = f_1(x_1, x_2, \ldots) \]
    \[ \hat{y}_2 = f_2(x_1, x_2, \ldots) \]
    \[ \Rightarrow \hat{y}_e = f_e(\hat{y}_1, \hat{y}_2, \ldots) \]
  …

  – Similar to multi-layer perceptron idea
  – Special case: binary, \( f_e \) linear \( \Rightarrow \) weighted vote
  – Can train ensemble weights \( f_e \) on validation data
Mixtures of experts

- Can make weights depend on $x$
  - Weight $\alpha_i(x)$ indicates “expertise”
  - Combine: weighted avg or just pick largest

Mixture of three linear predictor experts
Machine Learning and Data Mining

Ensembles: Bagging

Prof. Alexander Ihler
Fall 2012
Ensemble methods

- Why learn one classifier when you can learn many?
  - “Committee”: learn K classifiers, average their predictions

- “Bagging” = bootstrap aggregation
  - Learn many classifiers, each with only part of the data
  - Combine through model averaging

- Remember overfitting: “memorize” the data
  - Used test data to see if we had gone too far
  - Cross-validation
    - Make many splits of the data for train & test
    - Each of these defines a classifier
    - Typically, we use these to check for overfitting
    - Could we instead combine them to produce a better classifier?
Bagging

- **Bootstrap**
  - Create a random subset of data by sampling
  - Draw $N'$ of the $N$ samples with replacement (sometimes w/o)

- **Bagging**
  - Repeat $K$ times
    - Create a training set of $N' < N$ examples
    - Train a classifier on the random training set
  - To test, run each trained classifier
    - Each classifier votes on the output, take majority
    - For regression: each regressor predicts, take average

- **Notes**:
  - Some complexity control: harder for each to memorize data
    - Doesn’t work for linear models (e.g. linear regression)
    - Perceptrons OK (linear + threshold = nonlinear)
Bias / Variance

We only see a little bit of data

Can decompose error into two parts
- Bias – error due to model choice
  - Can our model represent the true best predictor?
  - Gets better with more complexity
- Variance – randomness due to data size
  - Better w/ more data, worse w/ complexity

“We only see a little bit of data”

Data we observe

$y(x) = \theta_0 + \theta_1 x + \nu$

$\hat{y}(x) = \hat{\theta}_0 + \hat{\theta}_1 x$

Predictive Error

Error on test data

(High bias)

(High variance)

Model Complexity
Bagged decision trees

- Randomly resample data
- Learn a decision tree for each

Simulates “equally likely” data sets we could have observed instead, & their classifiers
Bagged decision trees

- Average over collection
  - Classification: majority vote

- Reduces memorization effect
  - Not every predictor sees each data point
  - Lowers “complexity” of the overall average
  - Usually, better generalization performance

Avg of 5 trees  Avg of 25 trees  Avg of 100 trees
Bagging in Matlab

% Data set  X, Y
[N,D] = size(X);
Classifiers = cell(1,Nbag);   % Allocate space
for i=1:Nbag
    ind = ceil( N*rand(Nuse, 1) );   % Bootstrap sample data
    Xi = X(ind, :); Yi = Y(ind, :);   % Select those indices
    Classifiers{i} = Train_Classifier(Xi, Yi); % Train
end;

% Test data Xtest
[Ntest,D] = size(Xtest);
predict = zeros(Ntest,Nbag);   % Allocate space
for i=1:Nbag,                  % Apply each classifier
    predict(:,i)=Apply_Classifier( Xtest, Classifiers{i});
end;
predict = (mean(predict,2) > 1.5); % Vote on output (1 vs 2)
Random forests

- Bagging applied to decision trees
- Problem
  - With lots of data, we usually learn the same classifier
  - Averaging over these doesn’t help!
- Introduce extra variation in learner
  - At each step of training, only allow a subset of features
  - Enforces diversity (“best” feature not available)
  - Average over these learners (majority vote)

In `decisionTreeSplitData2(X,Y)`:  
For each of a subset of features  
  For each possible split  
    Score the split (e.g. information gain)  
    Pick the feature & split with the best score  
  Recurse on each subset
Summary

• Ensembles: collections of predictors
  – Combine predictions to improve performance

• Bagging
  – “Bootstrap aggregation”
  – *Reduces* complexity of a model class prone to overfit
  – In practice
    • Resample the data many times
    • For each, generate a predictor on that resampling
  – Plays on bias / variance trade off
  – Price: more computation per prediction
Machine Learning and Data Mining

Ensembles: Gradient Boosting

Prof. Alexander Ihler
Fall 2012
Ensembles

• Weighted combinations of predictors
• “Committee” decisions
  – Trivial example
  – Equal weights (majority vote / unweighted average)
  – Might want to weight unevenly – up-weight better predictors

• Boosting
  – Focus new learners on examples that others get wrong
  – Train learners sequentially
  – Errors of early predictions indicate the “hard” examples
  – Focus later predictions on getting these examples right
  – Combine the whole set in the end
  – Convert many “weak” learners into a complex predictor
Gradient Boosting

- Learn a regression predictor
- Compute the error residual
- Learn to predict the residual

Learn a simple predictor…

Then try to correct its errors
Gradient Boosting

- Learn a regression predictor
- Compute the error residual
- Learn to predict the residual

Combining gives a better predictor… Can try to correct its errors also, & repeat
Gradient Boosting

- Learn sequence of predictors
- Sum of predictions is increasingly accurate
- Predictive function is increasingly complex

Data & prediction function

Error residual
Gradient boosting

- Make a set of predictions $\hat{y}[i]$
- The “error” in our predictions is $J(y, \hat{y})$
  - For MSE: $J(.) = \sum (y[i] - \hat{y}[i])^2$
- We can “adjust” $\hat{y}$ to try to reduce the error
  - $\hat{y}[i] = \hat{y}[i] + \text{alpha } f[i]$
  - $f[i] \approx \nabla J(y, \hat{y}) = (y[i]-\hat{y}[i])$ for MSE
- Each learner is estimating the gradient of the loss f’n
- Gradient descent: take sequence of steps to reduce $J$
  - Sum of predictors, weighted by step size alpha
Gradient boosting in Matlab

% Data set  X, Y
mu = mean(Y);  % Often start with constant "mean" predictor
dY = Y - mu;  % subtract this prediction away
For k=1:Nboost,
    Learner{k} = Train_Regressor(X,dY);
    alpha(k) = 1;  % alpha: a "learning rate" or "step size"
    % smaller alphas need to use more classifiers, but tend to
    % predict better given enough of them

    % compute the residual given our new prediction
    dY = dY - alpha(k) * predict(Learner{k}, X)
end;

% Test data Xtest
[Ntest,D] = size(Xtest);
predict = zeros(Ntest,1);  % Allocate space
For k=1:Nboost,
    % Predict with each learner
    predict = predict + alpha(k)*predict(Learner{k}, Xtest);
end;
Summary

- **Ensemble methods**
  - Combine multiple classifiers to make “better” one
  - Committees, average predictions
  - Can use weighted combinations
  - Can use same or different classifiers

- **Gradient Boosting**
  - Use a simple regression model to start
  - Subsequent models predict the error residual of the previous predictions
  - Overall prediction given by a weighted sum of the collection
Machine Learning and Data Mining

Ensembles: Boosting

Prof. Alexander Ihler
Fall 2012
Ensembles

- Weighted combinations of classifiers
- “Committee” decisions
  - Trivial example
  - Equal weights (majority vote)
  - Might want to weight unevenly – up-weight good experts

- Boosting
  - Focus new experts on examples that others get wrong
  - Train experts sequentially
  - Errors of early experts indicate the “hard” examples
  - Focus later classifiers on getting these examples right
  - Combine the whole set in the end
  - Convert many “weak” learners into a complex classifier
Boosting Example

Original data set, $D_1$

Trained classifier

Update weights, $D_2$

Trained classifier

Update weights, $D_3$

Trained classifier

Classes $+1$, $-1$
Aside: minimizing weighted error

- So far we’ve mostly minimized unweighted error
- Minimizing weighted error is no harder:

Unweighted average loss:

\[ J(\theta) = \frac{1}{N} \sum_i J_i(\theta, x^{(i)}) \]

Weighted average loss:

\[ J(\theta) = \sum_i w_i J_i(\theta, x^{(i)}) \]

For any loss (logistic MSE, hinge, …)

\[ J(\theta, x^{(i)}) = \left( \sigma(\theta x^{(i)}) - y^{(i)} \right)^2 \]

\[ J(\theta, x^{(i)}) = \max \left[ 0, 1 - y^{(i)} \theta x^{(i)} \right] \]

For e.g. decision trees, compute weighted impurity scores:

- \( p(+1) = \) total weight of data with class +1
- \( p(-1) = \) total weight of data with class -1  \( \Rightarrow \) \( H(p) = \) impurity
Boosting Example

Weight each classifier and combine them:

\[ 0.33 \times \begin{array}{c}
\end{array} + 0.57 \times \begin{array}{c}
\end{array} + 0.42 \times \begin{array}{c}
\end{array} \geq 0 \]

Combined classifier

1-node decision trees
“decision stumps”
very simple classifiers
AdaBoost = adaptive boosting

- Pseudocode for AdaBoost

```matlab
for i=1:Nboost
    C{i} = train(X,Y,wts);       % Train a weighted classifier
    Yhat = predict(C{i},X);     % Compute predictions
    e = wts*(Y~=Yhat)';         % Compute weighted error rate
    alpha(i) = .5 log (1-e)/e;  % Compute coefficient
    wts *= exp(-alpha(i)*Y*Yhat); % Update weights
    wts=wts/sum(wts);
end;

% Final classifier:
( \sum alpha(i)*predict(C{i},Xtest) ) > 0
```

- Notes
  - $e > .5$ means classifier is not better than random guessing
  - $Y \times Yhat > 0$ if $Y == Yhat$, and weights decrease
  - Otherwise, they increase

Classes +1, -1
AdaBoost theory

- Minimizing classification error was difficult
  - For logistic regression, we minimized MSE instead
  - Idea: low MSE $\Rightarrow$ low classification error
- Example of a surrogate loss function
- AdaBoost also corresponds to a surrogate loss $f'$

\[ C_{ada} = \sum_{i} \exp[-y^{(i)} f(x^{i})] \]

- Prediction is $\hat{y} = \text{sign}(f(x))$
  - If same as $y$, loss $< 1$; if different, loss $> 1$; at boundary, loss $= 1$
- This loss function is smooth & convex (easier to optimize)
AdaBoost Example: Face Detection

- Viola-Jones face detection algorithm
- Combine lots of very weak classifiers
  - Decision stumps = threshold on a single feature
- Define lots and lots of features
- Use AdaBoost to find good features
  - And weights for combining as well
Haar wavelet features

- Four basic types.
  - They are easy to calculate.
  - The white areas are subtracted from the black ones.
  - A special representation of the sample called the integral image makes feature extraction faster.
Training a face detector

- Wavelets give ~100k features
- Each feature is one possible classifier
- To train: iterate from 1:T
  - Train a classifier on each feature using weights
  - Choose the best one, find errors and re-weight

- This can take a long time… (lots of classifiers)
  - One way to speed up is to not train very well…
  - Rely on adaboost to fix “even weaker” classifier

- Lots of other tricks in “real” Viola-Jones
  - Cascade of decisions instead of weighted combo
  - Apply at multiple image scales
  - Work to make computationally efficient
Summary

• Ensemble methods
  – Combine multiple classifiers to make “better” one
  – Committees, majority vote
  – Weighted combinations
  – Can use same or different classifiers

• Boosting
  – Train sequentially; later predictors focus on mistakes by earlier

• Boosting for classification (e.g., AdaBoost)
  – Use results of earlier classifiers to know what to work on
  – Weight “hard” examples so we focus on them more
  – Example: Viola-Jones for face detection