CS 5806 Machine Learning II

Lecture 13 - Ensemble Learning 2
October 4th, 2023
Hoda Eldardiry

Recommended Reading [1] Ch. 13, [5] Ch. 10, 15-16, [6] Ch. 14, [7] Sec. 18.10, [8] Sec. 16.2.5, 16.4.5

Outline

- Boosting cont.
- Gradient Boosting
- Bagging
- Random Forest

Boosting Paradigm

- When we already have a bunch of hypotheses
  — Boosting provides a principled approach to combine them

- **Useful for**
  — Sensor fusion
  — Combining experts
  — Where multiple heuristics have been proposed
  — Some may be weak with no strong backing theory
  — Boosting provides a way to combine different classifiers or different predictors, perhaps based on different sensors

Classification vs. Regression

- AdaBoost designed for classification
- Boosting for regression?

Gradient Boosting

- One of several possible solutions

Gradient Boosting

- **Key idea**
  - Single Regression Model
    - Fit predictor to desired output
    - Fit to training data (x,y) or (x, desired output)
  - Gradient Boosting for Regression
    - Fit predictor, compute loss
    - Next:
    - Instead of fitting another predictor
    - Fit predictor to approximate negative gradient of loss (current - desired)
    - Fit predictor to approximate difference between target & current prediction
    - Next predictor = current prediction + step in negative gradient direction

Gradient Boosting

1. Start with

Regression technique

Loss function (general)

Prediction

desired output

- Predictor f_k at stage k incurs loss $L(f_k(\boldsymbol{x}), y)$

- Train h_{k+1} to approximate negative gradient:

3. Fit next predictor h_{k+1} to the negative gradient*

2. Compute negative gradient of loss function w.r.t. current predictor f_k

$$h_{k+1}(\boldsymbol{x}) \approx -\frac{\partial L(f_k(\boldsymbol{x}), y)}{\partial f_k(\boldsymbol{x})}$$

*In regression, we normally fit predictor to desired output,
but here we fit the predictor to approximate the negative gradient!

- Update predictor by adding a multiple η_{k+1} of h_{k+1}:

4. Update predictor*

$$f_{k+1}(\boldsymbol{x}) \leftarrow f_k(\boldsymbol{x}) + \eta_{k+1}\, h_{k+1}(\boldsymbol{x})$$

*This emulates a step of gradient descent

Gradient boosting idea:
f_k predicts => f_k not perfect => improve it => instead of fitting another predictor => fit predictor to approximate negative gradient => step in negative gradient direction => gradient descent

At every step, we have imperfect predictor, with some loss, fit next predictor to approximate difference between target & current prediction => gradient descent:

Next predictor = current prediction + step in negative gradient direction

Squared Loss

Popular for regression

- Consider **squared loss**

To cancel square in derivative

$$L(f_k(\boldsymbol{x}_n), y_n) = \frac{1}{2}(f_k(\boldsymbol{x}_n) - y_n)^2$$

- Negative gradient corresponds to **residual** r

Difference between target & predictor

$$-\frac{\partial L(f_k(\boldsymbol{x}_n), y_n)}{\partial f_k(\boldsymbol{x}_n)} = y_n - f_k(\boldsymbol{x}_n) = r_n$$

Fit next hypothesis to approximate negative gradient
= fit difference between current predictor & target

Add this new hypothesis to predictor
=> fill gap = reduce difference
=> improve overall accuracy

- Train **base learner** h_{k+1} with **residual dataset** $\{(\boldsymbol{x}_n, r_n)_{\forall n}\}$

Train base learner to predict **residual** of each data point instead of predicting target y

If predictor approximates residual well,
& we add to current predictor => accuracy improves

- Base learner h_{k+1} can be any **non-linear predictor** (often a small decision tree)

If base learner is linear
=> add next linear predictor
=> new predictor still linear
=> little chance for improvement

If new predictor is non-linear
=> better chance for improvement

Gradient Boosting Algorithm

- Initialize predictor with a constant c:

$$f_0(\boldsymbol{x}_n) = argmin_c \sum_n L(c, y_n)$$

Find constant c that minimizes sum of losses for all points
Simple optimization in one variable c

Advantages of initializing predictor to **constant**
++simple
++takes us to right range/scale
=> if values to predict should be large, c will be large
=> if values to predict should be small, c will be small

- For $k = 1$ to K do

Compute a new predictor

  – Compute pseudo residuals: $r_n = -\frac{\partial L(f_{k-1}(\boldsymbol{x}_n), y_n)}{\partial f_{k-1}(\boldsymbol{x}_n)}$

Residual: defined in context of squared loss
pseudo residual: more general for other loss functions
=> negative gradient still gives some indication of error

  – Train a base learner h_k with residual dataset $\{(\boldsymbol{x}_n, r_n)_{\forall n}\}$

Find h to approximate -ve gradient: residual

  – Optimize step length:

$$\eta_k = argmin_\eta \sum_n L(f_{k-1}(\boldsymbol{x}_n) + \eta h_k(\boldsymbol{x}_n), y_n)$$

Add to predictor f, new h multiplied by step
Optimize step to minimize error
=> gradient descent

  – Update predictor: $f_k(\boldsymbol{x}) \leftarrow f_{k-1}(\boldsymbol{x}) + \eta_k h_k(\boldsymbol{x})$

Update predictor => weighted combination of predictors
=> f: predict y => h: predict difference

This leads to a predictor that is gradually improving (loss on training set is decreasing)
No guarantee that it reaches zero error, depends on:
=> expressiveness of base learner hypothesis space
=> noise in data

We don’t change predictor weights, we just add more predictors to create a larger ensemble

Gradient Boosting -> Susceptible to overfitting
To address overfitting => introduce randomization => reduce k => use validation set to select k

<<Ada-boost robust w.r.t. overfitting>>

XGBoost

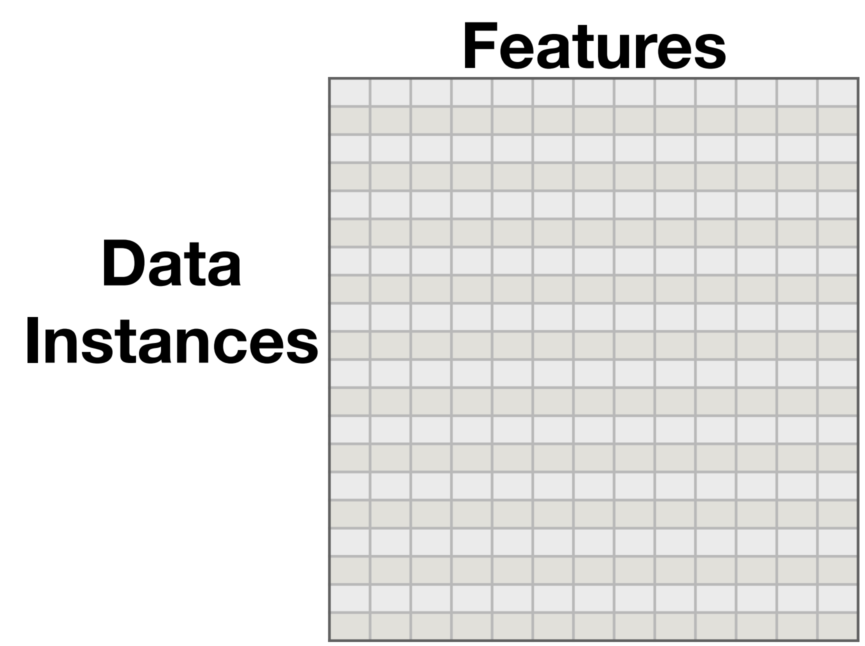
- eXtreme Gradient Boosting
  — Gradient Boosting package optimized for speed & accuracy
  — XGBoost used in >12 winning entries for various Kaggle challenges
  https://github.com/dmlc/xgboost/tree/master/demo#machine-learning-challenge-winning-solutions

Boosting vs Bagging

	Bagging	Boosting (ADA)	Boosting (GRADIENT)
Prediction	Majority vote	Weighted predictions	Linear combination
Assumptions	— Hypotheses must be independent — Must have similar accuracies	— Allows correlated hypotheses — Allows hypotheses with imbalanced accuracies	
Flexibility	— Limiting assumptions —> Less Flexible — Can we engineer a set up to guarantee these assumptions hold?	— More flexible ++ Weights counter effect of correlated hypotheses ++ Weights can balance accuracies	
Overfitting	No	No	Yes
Improvement	Reduces variance	Reduces bias	
Task	Classification	Classification	Regression
Base learner			Preferably nonlinear

Independent classifiers/ predictors

- How can we obtain independent classifiers/predictors for bagging?
- Bootstrap sampling
  – Sample subset of data
- Random projection
  – Sample subset of features



- Learn different classifiers/predictors based on each data subset & feature subset

Bagging

- For k = 1 to K
  $D_k \leftarrow$ sample data subset
  $F_k \leftarrow$ sample feature subset
  $h_k \leftarrow$ train classifier/predictor based on D_k & F_k

- Classification: majority(h_1(x), … , h_k(x))

- Regression: average(h_1(x), … , h_k(x))

- Random forest: bagging of decision trees

- Works well for distributed computing

Applications

- Any supervised learning task
  - Collaborative filtering (Netflix challenge - movie recommendation)
    => ensemble learning is key
  - Body part recognition (Kinect by Microsoft)
    => ensemble of decision trees
  - Spam filtering
  - Speech recognition/natural language processing
  - Data mining
  - Etc.

Application: Xbox 360 Kinect

- Microsoft Cambridge
- **Body part recognition:** supervised learning: random forest: before Deep learning (CNN) revolution
- Amongst earliest games that do not require holding a controller or joystick to issue commands
- Instead recognize body movement
- Track motion
- Infer posture
- Infer what players are doing

Depth Camera

- Kinect low cost depth camera

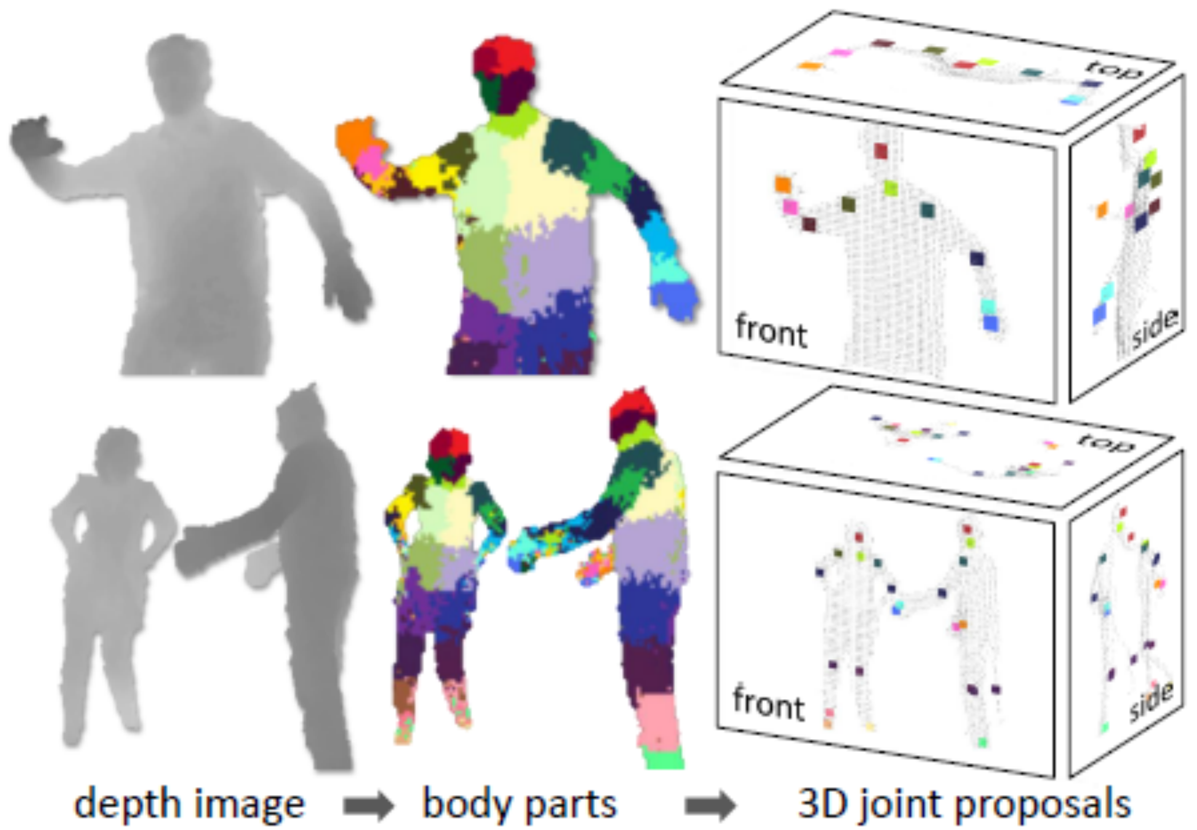
- Returns RGB color pixels & also depth information
- Distance of pixel from camera

Infrared image

Gray scale depth map

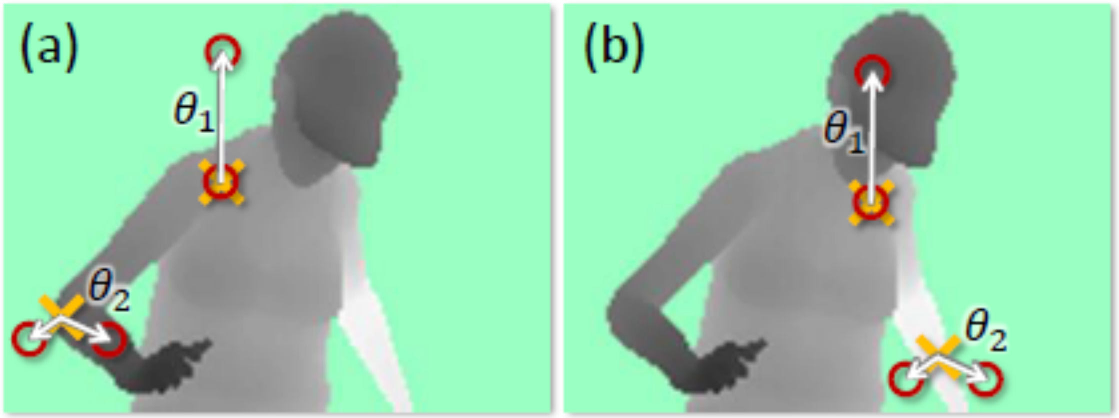
Kinect Body Part Recognition

- Problem: label each pixel with a body part

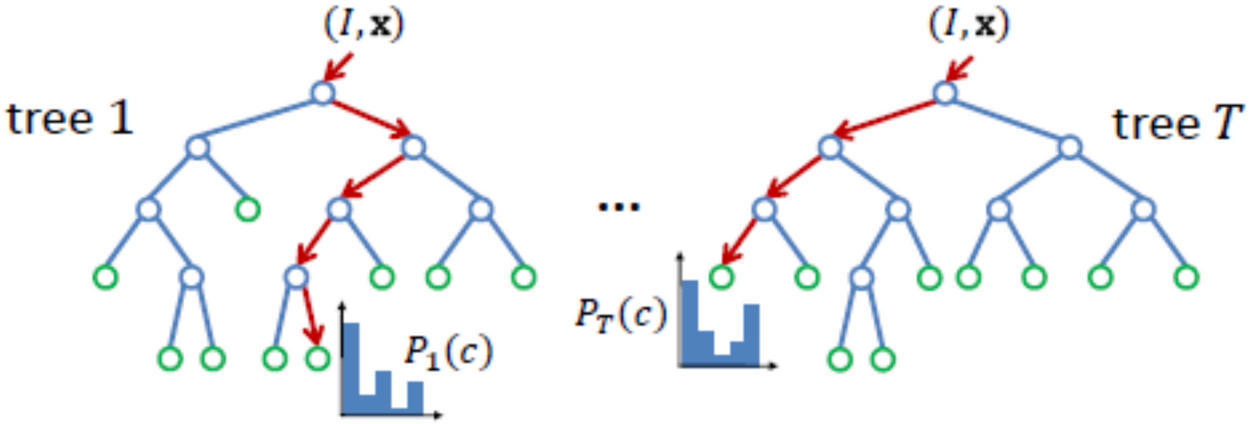


Kinect Body Part Recognition

- Features: depth differences between pairs of pixels



- Classification: Random Forest (forest of decision trees)



Safely Combining Predictions

- Safe approach to ensemble learning:
  — Combine predictions (not parameters)
- Classification: majority vote of classes predicted by the classifiers
- Regression: average of predictions computed by the regressors

Large Scale Machine Learning

- Big Data
  — Large number of data instances
  — Large number of features

- Solution: Distribute (parallel) computation
  — GPU (Graphics Processing Unit)
  — Multiple cores

Summary

- Ways to improve predictions
  - **Ensemble Learning** — combine models

Discussion

- What model are you considering for your project?
- Can you consider an ensemble of models?
- How would you do it?
- Start a discussion topic on canvas
  if you like to brainstorm further with others