Ensemble methods: Boosting

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Ensemble methods

Train and use multiple base models for either classification or regression problems

• Mixture of experts
  – each ‘base’ model (classifier, regressor) covers a different part (region) of the input space
  – All models are trained together on the same training data

• Committee machines:
  – each ‘base’ model (classifier, regressor) covers the complete input space
  – Each base model is trained on a slightly different train set
  – Combine predictions of all models to produce the output
    • Goal: Improve the accuracy of the ‘base’ model
    • Methods: Bagging, Boosting, Stacking (not covered)
Mixture of experts model

• **Ensamble methods:**
  - Use a combination of simpler learners/model to improve their predictions

• **Mixture of expert model:**
  - Different input regions covered with different learners
  - A “soft” switching between learners

• **Mixture of experts**
  Expert = learner

----

Mixture of experts model

• **Gating network**: decides what expert to use
  \( g_1, g_2, \ldots, g_k \) - gating functions

---
Bagging (Bootstrap Aggregating)

• **Given:**
  – Training set of \( N \) examples
  – A class of learning models (e.g. decision trees, neural networks, …)

• **Method:**
  – Train multiple (\( k \)) models on different samples (data splits) and average their predictions
  – Predict (test) by averaging the results of \( k \) models

• **Goal:**
  – Improve the accuracy of one model by using its multiple copies
  – Average of misclassification errors on different data splits gives a better estimate of the predictive ability of a learning method

---

Bagging algorithm

• **Training**
  – In each iteration \( t \), \( t=1,\ldots,T \)
    • Randomly sample with replacement \( N \) samples from the training set
    • Train a chosen “base model” (e.g. neural network, decision tree) on the samples

• **Test**
  – For each test example
    • Start all trained base models
    • Predict by combining results of all \( T \) trained models:
      – **Regression:** averaging
      – **Classification:** a majority vote
Analysis of Bagging

- **Expected error** = **Bias** + **Variance**
  - *Expected error* is the expected discrepancy between the estimated and true function
    \[ E[(\hat{f}(X) - E[f(X)])^2] \]
  - *Bias* is squared discrepancy between averaged estimated and true function
    \[ (E[\hat{f}(X)] - E[f(X)])^2 \]
  - *Variance* is expected divergence of the estimated function vs. its average value
    \[ E[(\hat{f}(X) - E[\hat{f}(X)])^2] \]

Under-fitting and over-fitting

- **Under-fitting:**
  - High bias (models are not accurate)
  - Small variance (smaller influence of examples in the training set)

- **Over-fitting:**
  - Small bias (models flexible enough to fit well to training data)
  - Large variance (models depend very much on the training set)
When Bagging works

- **Main property of Bagging** (proof omitted)
  - Bagging **decreases variance** of the base model without changing the bias!!!
  - Why? averaging!
- **Bagging typically helps**
  - When applied with an **over-fitted base model**
    - High dependency on actual training data
- **It does not help much**
  - High bias. When the base model is robust to the changes in the training data (due to sampling)

Boosting

- **Mixture of experts**
  - One expert per region
  - Expert switching
- **Bagging**
  - Multiple models on the complete space, a learner is not biased to any region
  - Learners are **learned independently**
- **Boosting**
  - Every learner covers the complete space
  - During training the learners are biased to regions not predicted well by other learners
  - **Learners are dependent**
Boosting. Theoretical foundations.

- **PAC**: Probably Approximately Correct framework
  - (ε-δ) solution
- **PAC learning**:
  - Learning with pre-specified error ε and confidence δ parameters
  - The probability that the misclassification error is larger than ε is smaller than δ
  
  $P(ME(c) > \varepsilon) \leq \delta$

- **Accuracy (1-ε)**: Percent of correctly classified samples in test
- **(1-δ)**: The probability that in one experiment some minimum accuracy will be achieved
  
  $P(\text{Acc}(c) > 1 - \varepsilon) > (1 - \delta)$

---

PAC Learnability

**Strong (PAC) learnability**:
- There exists a learning algorithm that efficiently learns the classification with a pre-specified accuracy and confidence

**Strong (PAC) learner**:
- A learning algorithm $P$ that given an arbitrary
  - classification error $\varepsilon$ ($< 1/2$), and
  - confidence parameter $\delta$ ($< 1/2$)
  - Outputs a classifier that satisfies this parameters
    - In other words the classifier gives:
      - classification accuracy $> (1-\varepsilon)$
      - confidence probability $> (1- \delta)$
    - And runs in time polynomial in $1/ \delta$, $1/\varepsilon$
      - Implies: number of samples $N$ is polynomial in $1/ \delta$, $1/\varepsilon$
**Weak Learner**

**Weak learner:**
- A learning algorithm (learner) $W$ that gives:
  - error $\varepsilon_0 (<1/2)$
  - confidence $\delta_0 (<1/2)$
  - For some fixed and uncontrollable $\varepsilon_0, \delta_0$
  - In other words:
    - a classification accuracy $> 1 - \varepsilon_0 (> 1/2)$
    - with probability $> 1 - \delta_0 (> 1/2)$
  - and this on an arbitrary distribution of data entries

---

**Weak learnability=Strong (PAC) learnability**

- Assume there exists a weak learner
  - it is better that a random guess ($> 50\%$) with confidence higher than $50\%$ on any data distribution
- **Question:**
  - Is the problem also PAC-learnable?
  - Can we generate an algorithm $P$ that achieves an arbitrary $(\varepsilon, \delta)$ accuracy?
- **Why is important?**
  - Usual classification methods (decision trees, neural nets), have specified, but uncontrollable performances.
  - Can we improve performance to achieve any pre-specified accuracy (confidence)?
Weak=Strong learnability!!!

- Proof due to R. Schapire
  
  An arbitrary \((\varepsilon, \delta)\) improvement is possible

Idea: combine multiple weak learners together
- Weak learner \(W\) with confidence \(\delta_o\) and maximal error \(\varepsilon_o\)
- It is possible:
  - To improve (boost) the confidence
  - To improve (boost) the accuracy

by training the different weak learners on slightly different datasets and combining their results

---

Boosting accuracy

Training

Distribution samples

Learners

\(H_1\)

\(H_2\)

\(H_3\)

Correct classification
Wrong classification
\(H_1\) and \(H_2\) classify differently
Boosting accuracy

• **Training**
  – Sample randomly from the distribution of examples
  – Train hypothesis $H_1$ on the sample
  – Evaluate accuracy of $H_1$ on the distribution
  – Sample randomly such that for the half of samples $H_1$ provides correct, and for another half, incorrect results; Train hypothesis $H_2$.
  – Train $H_3$ on samples from the distribution where $H_1$ and $H_2$ classify differently

• **Test**
  – For each example, decide according to the majority vote of $H_1$, $H_2$ and $H_3$

---

Theorem

• If each hypothesis has an error $< \varepsilon_0$, the final ‘voting’ classifier has error $< g(\varepsilon_0) = 3\varepsilon_0^2 - 2\varepsilon_0^3$

• **Accuracy improved !!!!**

• **Apply recursively to get to the target accuracy !!!**

---

CS 2750 Machine Learning
Theoretical Boosting algorithm

- Similarly to boosting the accuracy we can boost the confidence at some restricted accuracy cost
- **The key result:** we can improve both the accuracy and confidence

- **Problems with the theoretical algorithm**
  - A good (better than 50%) classifier on all distributions and problems
  - We cannot get a good sample from data-distribution
  - The method requires a large training set

- **Solution to the sampling problem:**
  - Boosting by sampling
    - AdaBoost algorithm and variants

AdaBoost

- **AdaBoost:** boosting by sampling

- **Classification** (Freund, Schapire; 1996)
  - AdaBoost.M1 (two-class problem)
  - AdaBoost.M2 (multiple-class problem)

- **Regression** (Drucker; 1997)
  - AdaBoostR
AdaBoost

- **Given:**
  - A training set of \( N \) examples (attributes + class label pairs)
  - A “base” learning model (e.g. a decision tree, a neural network)

- **Training stage:**
  - Train a sequence of \( T \) “base” models on \( T \) different sampling distributions defined upon the training set (\( D \))
  - A sample distribution \( D_t \) for building the model \( t \) is constructed by modifying the sampling distribution \( D_{t-1} \) from the \((t-1)\)th step.
    - Examples classified incorrectly in the previous step receive higher weights in the new data (attempts to cover misclassified samples)

- **Application (classification) stage:**
  - Classify according to the **weighted majority** of classifiers

---

AdaBoost training

![AdaBoost training diagram]

- Distribution
- Learn
- Test
- \( D_1 \) → Model 1 → Errors 1
- \( D_2 \) → Model 2 → Errors 2
- \( \cdots \)
- \( D_T \) → Model T → Errors T

---
AdaBoost algorithm

**Training (step t)**

- **Sampling Distribution** $D_t$
  - $D_t(i)$ - a probability that example $i$ from the original training dataset is selected
  - $D_t(i) = 1 / N$ for the first step ($t=1$)
- Take $K$ samples from the training set according to $D_t$
- Train a classifier $h_t$ on the samples
- Calculate the error $\varepsilon_t$ of $h_t$: $\varepsilon_t = \sum_{i: h_t(x_i) \neq y_i} D_t(i)$
- Classifier weight: $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$
- New sampling distribution $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$

---

**AdaBoost. Sampling Probabilities**

Example:  
- Nonlinearly separable binary classification
- NN as weak learners
AdaBoost classification

- We have $T$ different classifiers $h_t$.
  - Weight $w_t$ of the classifier is proportional to its accuracy on the training set
    \[
    w_t = \log(1/\beta_t) = \log((1 - \epsilon_t)/\epsilon_t) \\
    \beta_t = \epsilon_t/(1 - \epsilon_t)
    \]

- **Classification:**
  For every class $j=0,1$
  - Compute the sum of weights $w$ corresponding to ALL classifiers that predict class $j$;
  - Output class that correspond to the maximal sum of weights (weighted majority)
    \[
    h_{final}(x) = \arg\max_j \sum_{t:h_t(x)=j} w_t
    \]
## Two-Class example. Classification.

- Classifier 1  “yes”  0.7
- Classifier 2  “no”  0.3
- Classifier 3  “no”  0.2

- Weighted majority  “yes”  
  \[ 0.7 - 0.5 = + 0.2 \]

- The final choose is “yes”  + 1

---

## What is boosting doing?

- Each classifier specializes on a particular subset of examples
- Algorithm is concentrating on “more and more difficult” examples

**Boosting can:**
  - Reduce variance (the same as Bagging)
  - But also to eliminate the effect of high bias of the weak learner (unlike Bagging)

**Train versus test errors performance:**
  - Train errors can be driven close to 0
  - But test errors do not show overfitting

- Proofs and theoretical explanations in **a number of papers**
Boosting. Error performances

CS 2750 Machine Learning