Learning with multiple models. Boosting.

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

Learning with multiple models: Approach 2

- **Approach 2**: use multiple models (classifiers, regressors) that cover the complete input (x) space and combine their outputs

- **Committee machines**:
  - Combine predictions of all models to produce the output
    - **Regression**: averaging
    - **Classification**: a majority vote
  - **Goal**: Improve the accuracy of the ‘base’ model

- **Methods**:
  - **Bagging** (the same base models)
  - **Boosting** (the same base models)
  - **Stacking** (different base model) not covered
Bagging algorithm

- **Training**
  - For each model M1, M2, … Mk
    - Randomly sample with replacement N samples from the training set (bootstrap)
    - Train a chosen “base model” (e.g. neural network, decision tree) on the samples

![Diagram of bagging algorithm](image)

- **Test**
  - For each test example
    - Run all base models M1, M2, … Mk
    - Predict by combining results of all T trained models:
      - **Regression**: averaging
      - **Classification**: a majority vote
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
    • Example: fully grown decision trees
• It does not help much
  – High bias. When the base model is robust to the changes in the training data (due to sampling)

Boosting

• Bagging
  – Multiple models covering the complete space, a learner is not biased to any region
  – Learners are learned independently

• Boosting
  – Every learner covers the complete space
  – Learners are biased to regions not predicted well by other learners
  – Learners are dependent
Boosting. Theoretical foundations.

- **PAC:** Probably Approximately Correct framework
  - \((\varepsilon, \delta)\) solution
- **PAC learning:**
  - Learning with a pre-specified error \(\varepsilon\) and a confidence parameter \(\delta\)
  - the probability that the misclassification error is larger than \(\varepsilon\) is smaller than \(\delta\)
  \[ P(ME(c) > \varepsilon) \leq \delta \]

Alternative rewrite:
\[ P(Acc(c) > 1 - \varepsilon) > (1 - \delta) \]

- **Accuracy (1-\(\varepsilon\)):** Percent of correctly classified samples in test
- **Confidence (1-\(\delta\)):** The probability that in one experiment some accuracy will be achieved

---

PAC Learnability

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

**Strong (PAC) learner:** A learning algorithm \(P\) that
- Given an arbitrary:
  - classification error \(\varepsilon\) (<1/2), and
  - confidence \(\delta\) (<1/2)
  - or in other words:
    - classification accuracy > (1-\(\varepsilon\))
    - confidence probability > (1- \(\delta\))
- Outputs a classifier that satisfies this parameters
- **Efficiency:** 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) $M$ that gives **some fixed (not arbitrary):**
  - error $\epsilon_o (<1/2)$ and
  - confidence $\delta_o (<1/2)$
- Alternatively:
  - a classification accuracy $> 0.5$
  - with probability $> 0.5$
  *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 strong PAC-learnable?
  - Can we generate an algorithm $P$ that achieves an arbitrary $(\epsilon, \delta)$ 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)?
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 different weak learners on slightly different datasets
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 $< \epsilon_o$, the final ‘voting’ classifier has error $< g(\epsilon_o) = 3 \epsilon_o^2 - 2 \epsilon_o^3$
- **Accuracy improved !!!!**
- **Apply recursively to get to the target accuracy !!!**

![Graph](attachment:image.png)
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 training

Training data

Distribution

$D_1$

Uniform distribution $D_1$ training examples

$P(\text{example } i) = 1/N$

AdaBoost training

Training data

Distribution

$D_1$

Learn

Model 1

Sample randomly according to $D_1$

And train the Model 1
AdaBoost training

Test the Model 1 and calculate errors

Use errors to recalculate the new distribution on data
More probability to pick examples with errors
**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 algorithm**

**Training (step t)**

- **Sampling Distribution** $D_t$
  
  $D_t(i)$ - a probability that example i from the original training dataset is selected
  
  $D_1(i) = 1/N$ 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 $h_t$: $\varepsilon_t = \sum_i D_t(i)$

- **Classifier weight**: $\beta_t = \varepsilon_t/(1-\varepsilon_t)$

- **New sampling distribution**
  
  $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} 
  \beta_t & h_t(x_i) = y_i \\
  1 & \text{otherwise} 
  \end{cases}$

  Norm. constant

---

**AdaBoost. Sampling Probabilities**

Example:
- Nonlinearly separable binary classification
- NN used as week 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
    \[ w_t = \log(1 / \beta_t) = \log \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) \]
    \[ \beta_t = \frac{\epsilon_t}{1 - \epsilon_t} \]
- **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)
    \[ h_{\text{final}}(x) = \arg \max_j \sum_{\forall h_t(x) = j} w_t \]
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 choice 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)
  - 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

Model Averaging

• An alternative way to combine weight multiple models
• Can be used for supervised and unsupervised frameworks
• For example:
  – Likelihood of the data can be expressed by averaging over the multiple models
    \[ P(D) = \sum_{i=1}^{N} P(D \mid M = m_i)P(M = m_i) \]
  – Prediction:
    \[ P(y \mid x) = \sum_{i=1}^{N} P(y \mid x, M = m_i)P(M = m_i) \]