Meta Learning methods

Mario Martin

UPC - Computer Science Dept.
Outline

Introduction
  Definition

Voting schemes
  Stacking
  Weighted majority
  Bagging and Random Forests
  Boosting
Introduction
  Definition

Voting schemes
  Stacking
  Weighted majority
  Bagging and Random Forests
  Boosting
Multiclassifiers, Meta-learners, Ensemble Learners

- Combining several *weak learners* to give a *strong learner*

- A kind of *multiclassifier* systems and *meta-learners*

- *Ensemble* typically applied to a single type of weak learner
  - All built by same algorithm, with different data or parameters

- Lots of what I say applies to multiclassifier systems in general
Why?

1. They achieve higher accuracy in practice
   - We trade computation time for classifier weakness

Why?

1. They achieve higher accuracy in practice
   - We trade computation time for classifier weakness
   - Netflix competition (2009) won by a combination of 107 hybrid classifiers
   - More: Most of the top teams were multi-classifiers

Why?

1. They achieve higher accuracy in practice
   - We trade computation time for classifier weakness
   - Netflix competition (2009) won by a combination of 107 hybrid classifiers
   - More: Most of the top teams were multi-classifiers

2. Combine strengths of different classifier builders
   - And we can incorporate domain knowledge into different learners

Why?

1. They achieve higher accuracy in practice
   ▶ We trade computation time for classifier weakness
   ▶ Netflix competition (2009) won by a combination of 107 hybrid classifiers
   ▶ More: Most of the top teams were multi-classifiers

2. Combine strengths of different classifier builders
   ▶ And we can incorporate domain knowledge into different learners

3. May help avoiding overfitting
   ▶ This is paradoxical because more expressive than weak learners!

Condorcet’s jury theorem

- **Condorcet’s jury theorem** states that when independent predictors with probability $p$ of successful output ($p > 0.5$), combining the outputs using majority vote have probability of success $p_{mv}$ such that $p_{mv} > p$.

- Example: 3 classifiers $c_1, c_2, c_3$ with $p = 0.7$
Condorcet’s jury theorem states that when independent predictors with probability $p$ of successful output ($p > 0.5$), combining the outputs using majority vote have probability of success $p_{mv}$ such that $p_{mv} > p$.

- Example: 3 classifiers $c_1, c_2, c_3$ with $p = 0.7$
- Example: 3 classifiers $c_1, c_2, c_3$ with $p_1 = 0.7, p_2 = 0.8$ and $p_3 = 0.75$
Combining weak learners

▶ Voting
  ▶ Each weak learner votes, and votes are combined

▶ Experts that abstain
  ▶ A weak learner only counts when it’s expert on this kind of instances
  ▶ Otherwise it abstains (or goes to sleep)
Outline

Introduction
Definition

Voting schemes
  Stacking
  Weighted majority
  Bagging and Random Forests
  Boosting
Voting

- Simple majority vote
- Weights depend on errors: 
  \[ (1 - e_i)^{1/e_i} \exp(-e_i) \ldots \]
- Weights depend on confidences
- Maximizing diversity
Voting

How to combine votes?

- Simple majority vote
- Weights depend on errors
  \[ (1 - e_i)^{-1} \exp(-e_i) \cdots \]
- Weights depend on confidences
- Maximizing diversity
How to combine votes?

- Simple *majority vote*
- Weights depend on *errors* \((1 - e_i)? 1/e_i? \exp(-e_i)? \ldots\)
Voting

How to combine votes?

- Simple *majority vote*
- Weights depend on *errors* \((1 - e_i) ? 1/e_i ? \exp(-e_i) ? \ldots\)
- Weights depend on *confidences*
- Maximizing *diversity*
Outline

Introduction
Definition

Voting schemes
Stacking
Weighted majority
Bagging and Random Forests
Boosting
Stacking (Wolpert 92)

A meta-learner that learns to weight its weak learner

- Dataset with instances \((x,y)\)
- Transform dataset to have instances \((x,c_1(x),\ldots,c_N(x),y)\)
- Train meta-classifier \(M\) with enriched dataset
Stacking (Wolpert 92)

A meta-learner that learns to weight its weak learner

- Dataset with instances \((x, y)\)
- Transform dataset to have instances \((x, c_1(x), \ldots, c_N(x), y)\)
- Train meta-classifier \(M\) with enriched dataset

Often, \(x\) not given to \(M\), just the votes
Often, just linear classifier
Can simulate most other voting schemes
Outline

Introduction
  Definition

Voting schemes
  Stacking
  Weighted majority
  Bagging and Random Forests
  Boosting
Weighted majority (Littlestone-Warmuth 92)

initialize classifiers $c_1 \ldots c_N$ with weight $w_i = 1/N$ each;

for each example $x$ in sequence do
collect predictions $c_1(x) \ldots c_N(x)$;

$\text{prediction}(x) = \text{sign}\left[w_1 c_1(x) + \ldots + w_N c_N(x)\right] - 1/2$;

get true label $y$ for $x$;

for each $i = 1 \ldots N$,
if ($c_i(x) \neq y$) then $w_i = w_i/2$;
renormalize weights to sum 1;

$\triangleright$ Weights depend exponentially on error
$\triangleright$ At least as good as best weak learner in time $O(\log N)$
$\triangleright$ Often much better; more when classifiers are uncorrelated
$\triangleright$ Good for online prediction and when many classifiers
$\triangleright$ E.g. when 1 classifier = 1 feature
Weighted majority (Littlestone-Warmuth 92)

initialize classifiers $c_1...c_N$ with weight $w_i = 1/N$ each;
initialize classifiers \( c_1 \ldots c_N \) with weight \( w_i = 1/N \) each;
for each example \( x \) in sequence do

\[
\text{prediction}(x) = \text{sign}\left[ w_1 c_1(x) + \ldots + w_N c_N(x) \right] - 1/2
\]

get true label \( y \) for \( x \);
for each \( i = 1 \ldots N \),
if \( c_i(x) \neq y \) then \( w_i = w_i/2 \);
renormalize weights to sum 1;

▶ Weights depend exponentially on error
▶ At least as good as best weak learner in time \( O(\log N) \)
▶ Often much better; more when classifiers are uncorrelated
▶ Good for online prediction and when many classifiers
▶ E.g. when 1 classifier = 1 feature
Weighted majority (Littlestone-Warmuth 92)

initialize classifiers $c_1...c_N$ with weight $w_i = 1/N$ each;
for each example $x$ in sequence do
  collect predictions $c_1(x)...c_N(x)$;
Weighted majority (Littlestone-Warmuth 92)

initialize classifiers $c_1 \ldots c_N$ with weight $w_i = 1/N$ each;
for each example $x$ in sequence do
  collect predictions $c_1(x) \ldots c_N(x)$;
  $\text{prediction}(x) = \text{sign}[ w_1 c_1(x) + \ldots + w_N c_N(x) - 1/2 ]$
initialize classifiers $c_1 \ldots c_N$ with weight $w_i = 1/N$ each;
for each example $x$ in sequence do
    collect predictions $c_1(x) \ldots c_N(x)$;
    prediction$(x) = \text{sign}[ w_1c_1(x)+\ldots+w_Nc_N(x)-1/2 ]$
get true label $y$ for $x$;
initialize classifiers c1...cN with weight wi = 1/N each;
for each example x in sequence do
    collect predictions c1(x)...cN(x);
prediction(x) = sign[ w1*c1(x)+...+wN*cN(x))-1/2 ]
get true label y for x;
for each i=1..N,
initialize classifiers c1...cN with weight \( w_i = 1/N \) each;  
for each example x in sequence do  
  collect predictions c1(x)...cN(x);  
prediction(x) = \text{sign}[w_1c_1(x)+...+w_Nc_N(x)]-1/2  
get true label y for x;  
for each i=1..N,  
  if (ci(x) != y) then \( w_i = w_i/2 \);
initialize classifiers $c_1 \ldots c_N$ with weight $w_i = 1/N$ each;

for each example $x$ in sequence do
  collect predictions $c_1(x) \ldots c_N(x)$;
  prediction($x$) = sign[$w_1c_1(x)+\ldots+w_Nc_N(x)$]-1/2 ]
  get true label $y$ for $x$;
  for each $i=1..N$,
    if ($c_i(x) \neq y$) then $w_i = w_i/2$;
renormalize weights to sum 1;
Weighted majority (Littlestone-Warmuth 92)

initialize classifiers c1...cN with weight wi = 1/N each;
for each example x in sequence do
  collect predictions c1(x)...cN(x);
prediction(x) = sign[ w1*c1(x)+...+wN*cN(x))-1/2 ]
get true label y for x;
for each i=1..N,
  if (ci(x) != y) then wi = wi/2;
renormalize weights to sum 1;
initialize classifiers c1...cN with weight wi = 1/N each;
for each example x in sequence do
    collect predictions c1(x)...cN(x);
    prediction(x) = \text{sign}[ w1*c1(x)+...+wN*cN(x)-1/2 ]
    get true label y for x;
    for each i=1..N,
        if (ci(x) != y) then wi = wi/2;
    renormalize weights to sum 1;

- Weights depend exponentially on error
- At least as good as best weak learner in time $O(\log N)$
- Often much better; more when classifiers are uncorrelated
- Good for online prediction and when many classifiers
- E.g. when 1 classifier = 1 feature
Outline

Introduction
Definition

Voting schemes
Stacking
Weighted majority
Bagging and Random Forests
Boosting
Bagging I

- To reduce the variance of an estimator, it is helpful to average estimates from independent draws from the data.
- Assuming each $Y_b$ is an unbiased estimate of target value $y$:

$$
\begin{align*}
\mathbb{E} \left[ (y - Y_b)^2 \right] &= \text{Var} [Y_b] \\
\mathbb{E} \left[ \left( y - \frac{1}{B} \sum_b Y_b \right)^2 \right] &= \frac{1}{B^2} \sum_b \text{Var} [Y_b] \quad \text{(if all } Y_b \text{ are independent)} \\
 &= \frac{1}{B} \text{Var} [Y_b] \quad \text{(if all } Y_b \text{ have same variance)}
\end{align*}
$$
So, the idea of bagging is to combine the predictions of a high-variance predictor trained on independent bootstrap samples from the same dataset, to make the combined predictions more robust (i.e. with lower variance) and, therefore, more accurate.

Trees typically suffer from high variance (= overfitting) so it is specially useful in Decision trees (high variance or sensibility to training data set)
Bagging (Breiman 96)

1. Get a dataset $S$ of $N$ labeled examples on $A$ attributes;
2. Build $N$ bagging replicas of $S$: $S_1, \ldots, S_N$;
   - $S_i = \text{draw } N \text{ samples from } S \text{ with replacement}$;
3. Use the $N$ replicas to build $N$ weak learners $C_1, \ldots, C_N$;
4. Predict using majority vote of the $C_i$’s
Example of building training sets:

<table>
<thead>
<tr>
<th>Original:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Set1:</td>
<td>2</td>
<td>7</td>
<td>8</td>
<td>3</td>
<td>7</td>
<td>6</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Training Set2:</td>
<td>7</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Training Set3:</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Training Set4:</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

Any samples that are not chosen for the bootstrapped dataset are placed in a separate dataset called the **out-of-bag dataset** (OOB).
Bagging (Breiman 96)

1. Get a dataset $S$ of $N$ labeled examples on $A$ attributes;
2. Build $N$ bagging replicas of $S$: $S_1, \ldots, S_N$;
   - $S_i = \text{draw } N \text{ samples from } S \text{ with replacement}$;
3. Use the $N$ replicas to build $N$ weak learners $C_1, \ldots, C_N$;
4. Predict using majority vote of the $C_i$'s
Out-of-bag (OOB) error

- The **OOB error** is an estimation of generalization error that can be used as validation error to select appropriate values for the hyperparameters; as a direct consequence, **there is no need for cross-validation** for model selection (hyperparameter tuning).

- For each case in the OOB dataset compute the oob error:
  1. Find all models that are not trained by the OOB instance.
  2. Obtain prediction for each model
  3. Average all these predictions (regression) or take the majority vote (classification) to compute the oob error for each example, and

- Average OOB error across examples
Random Forests (Breiman 01, Ho 98)

1. Parameters $k$ and $a$;
2. Get a dataset $S$ of $N$ labeled examples on $A$ attributes;
3. Build $k$ bagging replicas of $S$: $S_1, \ldots, S_k$;
4. Use the $k$ replicas to build $k$ random trees $T_1, \ldots, T_k$;
   - At each node split, randomly select $a \leq A$ attributes, and choose best of these $a$;
   - Grow each tree as deep as possible: not pruning!!
5. Predict using majority vote of the $T_i$’s
Random Forests II

Weak learner strength vs. weak learner variance

- More attributes $a$ increases strength, overfits more
- More trees $k$ decreases variance, overfits less

Can be shown to be similar to weighted $k$-NN

Top performer in many tasks
Random Forests II

Weak learner strength vs. weak learner variance

- More attributes $a$ increases strength, overfits more
- More trees $k$ decreases variance, overfits less

Can be shown to be similar to weighted $k$-NN
Top performer in many tasks
If a random forest contains many trees, it can be difficult to comprehend what the model is doing (not interpretable by a person).

- Variable importance plot add interpretability to the model

1. Gini-based variable importance
   Add gini impurity gains for variables in splits in each tree in the forest, sort variables by their sum.

2. Permutation-based variable importance
   For each variable, permute values and compute difference in OOB error metrics before and after permutation. If variable is important, then accuracy in the permuted copy should decrease. Sort variables by this difference.

Permutation-based more reliable, but slower; gini-based is biased towards categorical variables with many splits.\(^2\)

\(^2\)If interested, you can read this article.
Outline

Introduction
Definition

Voting schemes
Stacking
Weighted majority
Bagging and Random Forests
Boosting
Boosting I

- Bagging tries to reduce variance of base classifiers by building different bootstrapping datasets
- Boosting tries to actively improve accuracy of weak classifiers
- How? By training a sequence of specialized classified based on previous errors
Adaptively, sequentially, creating classifiers

Classifiers and instances have varying weights

Increase weight of incorrectly classified instances
Boosting II

- Works on top of any *weak learner*. A weak learner is defined as any learning mechanism that works better than chance (accuracy > 0.5 when two equally probable classes)
Boosting II

- Works on top of any weak learner. A weak learner is defined as any learning mechanism that works better than chance (accuracy > 0.5 when two equally probable classes)
- Adaptively, *sequentially*, creating classifiers
Boosting II

- Works on top of any weak learner. A weak learner is defined as any learning mechanism that works better than chance (accuracy $> 0.5$ when two equally probable classes)
- Adaptively, sequentially, creating classifiers
- Classifiers and instances have varying weights
Boosting II

- Works on top of any weak learner. A weak learner is defined as any learning mechanism that works better than chance (accuracy > 0.5 when two equally probable classes)
- Adaptively, sequentially, creating classifiers
- Classifiers and instances have varying weights
- Increase weight of incorrectly classified instances
Boosting II

- Works on top of any *weak learner*. A weak learner is defined as any learning mechanism that works better than chance (accuracy $> 0.5$ when two equally probable classes)
- Adaptively, *sequentially*, creating classifiers
- Classifiers and instances have *varying weights*
- Increase weight of incorrectly classified instances
- Final label as weighting voting of sequence of classifiers
Only two classes
Output: \( y \in \{-1, 1\} \)
Exemples: \( X \)
Weak Classifier: \( G(X) \)
Error de training (\( \text{err}_{\text{train}} \))

\[
\text{err}_{\text{train}} = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq G(x_i))
\]
Preliminars

\[ G(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m G_m(x) \right) \]
Adaboost algorithm

Set weight of all examples to $1/n$
Adaboost algorithm

Set weight of all examples to $1/n$
For $t=1:L$
Adaboost algorithm

Set weight of all examples to $1/n$
For $t=1:L$
    $S_t = \text{training set using weights for each example}$

\[ \text{return classifier: } G(x) = \text{sign} \left( \sum_{t=1}^{L} \alpha_t G_t(x) \right) \]
Adaboost algorithm

Set weight of all examples to $1/n$
For $t=1:L$
    $S_t = \text{training set using weights for each example}$
    Learn $G_t(S_t)$
Adaboost algorithm

Set weight of all examples to $1/n$
For $t=1:L$

$S_t =$ training set using weights for each example
Learn $G_t(S_t)$
Compute $err_t$ for $G_t$
Adaboost algorithm

Set weight of all examples to $1/n$
For $t=1:L$

- $S_t =$ training set using weights for each example
- Learn $G_t(S_t)$
- Compute $err_t$ for $G_t$
- Compute $\alpha_m = \frac{1}{2} \ln \left( \frac{1 - err_t}{err_t} \right)$
Adaboost algorithm

Set weight of all examples to \(1/n\)

For \(t=1:L\)

\(S_t = \) training set using weights for each example

Learn \(G_t(S_t)\)

Compute \(err_t\) for \(G_t\)

Compute \(\alpha_m = \frac{1}{2} \ln \left( \frac{1 - err_t}{err_t} \right)\)

Compute new weights \(w_i \leftarrow \frac{w_i}{Z_t} \cdot e^{-[\alpha_t \cdot y_i \cdot G_t(x_i)]}\)
Adaboost algorithm

Set weight of all examples to $1/n$

For $t=1:L$

$S_t =$ training set using weights for each example

Learn $G_t(S_t)$

Compute $err_t$ for $G_t$

Compute $\alpha_m = \frac{1}{2} \ln \left( \frac{1 - err_t}{err_t} \right)$

Compute new weights $w_i \leftarrow \frac{w_i}{Z_t} \cdot e^{-[\alpha_t \cdot y_i \cdot G_t(x_i)]}$

Return classifier: $G(x) = \text{sign} \left( \sum_{t=1}^{L} \alpha_m G_t(x) \right)$
Adaboost algorithm

\[
\alpha_m = \frac{1}{2} \ln \left( \frac{1 - \text{err}_m}{\text{err}_m} \right) > 0
\]

\[
w_i \leftarrow \frac{w_i}{Z_m} \cdot e^{-[\alpha_m \cdot y_i \cdot G(x_i)]}
\]

\[
G(x) = \text{sign} \left( \sum_{m=1}^{L} \alpha_m G_m(x) \right)
\]
Adaboost algorithm

\[ \alpha_m = \frac{1}{2} \ln \left( \frac{1 - \text{err}_m}{\text{err}_m} \right) > 0 \]

\[ w_i \leftarrow \frac{w_i}{Z_m} \cdot \begin{cases} e^{-\alpha_t} & \text{if } y_i = G_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq G_t(x_i) \end{cases} \]

\[ G(x) = \text{sign} \left( \sum_{m=1}^{L} \alpha_m G_m(x) \right) \]
We will use *Decision stumps* as the weak learner.

Decision stumps are decision trees pruned to only one level. Good candidates to weak learners: above 0.5 accuracy and high variance.

Two examples of decision stumps.
Simple example
Simple example

\[ h_1 \]

\[ D_2 \]

\[ \varepsilon_1 = 0.30 \]

\[ \alpha_1 = 0.42 \]
Simple example
Simple example
\[ H_{\text{final}} = \text{sign} \begin{pmatrix} 0.42 & + 0.65 & + 0.92 \end{pmatrix} \]
Simple example

\[ f = \left( \begin{array}{c} 0.42 \\ + 0.65 \\ + 0.92 \end{array} \right) / (0.42 + 0.65 + 0.92) \]
**Theorem.** Suppose that the error of classifier $h_t$ is $1/2 - \gamma_t$, $t = 1..T$. Then the error of the combination $H$ of $h_1, \ldots, h_T$ is at most

$$\exp \left( - \sum_{t=1}^{T} \gamma_t^2 \right)$$

Note: It tends to 0 if we can guarantee $\gamma_i \geq \gamma$ for fixed $\gamma$.
Boosting vs. Bagging

- Fruitful investigation on how and why they differ
- On average, Boosting provides a larger increase in accuracy than Bagging
- But Boosting fails sometimes (particularly in noisy data)
- while bagging consistently gives an improvement
Reasons why this works

1. Statistical reasons: We do not rely on one classifier, so we reduce variance

2. Computational reasons: A weak classifier can be stuck in local minima. When starting from different training data sets, we can find better solution

3. Representational reasons: Combination of classifiers return solutions outside the initial set of hypothesis, so they adapt better to the problem
Reasons why this works

All the previous reasons seem to drive us to an overfitting on the training data set.
Reasons why this works

All the previous reasons seem to drive us to an overfitting on the training data set.

However, in practice this is not the case. Not well understood theoretical reasons.
Reasons why this works

All the previous reasons seem to drive us to an overfitting on the training data set.

However, in practice this is not the case. Not well understood theoretical reasons.

In practice, they work very well, sometimes better than SVMs.