Meta-Learning methods

Mario Martin partially based on Ricard Gavaldà slides

UPC - Computer Science Dept.

Outline

Outline

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?

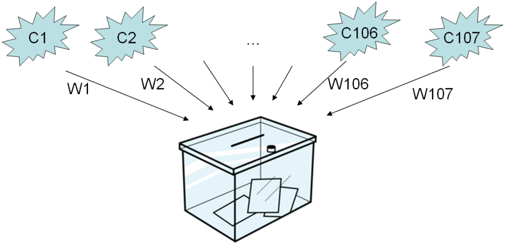
1. They achieve higher accuracy in practice
   - We trade computation time for classifier weakness
   - Netflix competition (2009) won by a combination of 107 hybrid classifiers
   - More: Most of the top teams were multi-classifiers
2. Combine strengths of different classifier builders
   - And we can incorporate domain knowledge into different learners
3. May help avoiding overfitting
   - This is paradoxical because more expressive than weak learners!

Combining weak learners

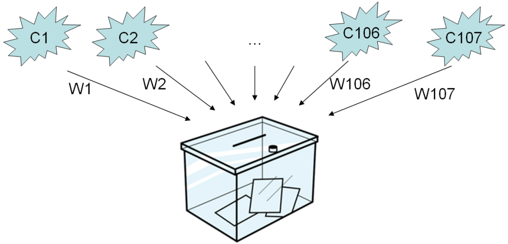
- Voting
  - Each weak learner votes, and votes are combined
- Experts that abstain
  - A weak learner only counts when it's expert on this kind of instances
  - Otherwise it abstains (or goes to sleep)

Outline

Voting

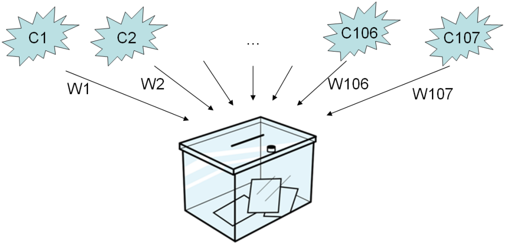


Voting



How to combine votes?

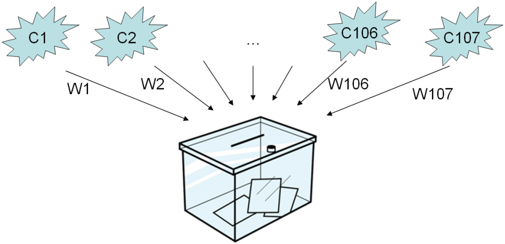
Voting



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* ($1 - e_i$? $1/e_i$? $\exp(-e_i)$? ...)
- Weights depend on *confidences*
- Maximizing *diversity*

Outline

Stacking (Wolpert 92)

A meta-learner that learns to weight its weak learner

- Dataset with instances `(x,y)`
- Transform dataset to have instances `(x,c1(x),...cN(x),y)`
- Train metaclassifier 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,c1(x),...cN(x),y)`
- Train metaclassifier M with enriched dataset

Often, `x` not given to M, just the votes
Often, just linear classifier
Can simulate most other voting schemes

Outline

Weighted majority (Littlestone-Warmuth 92)

Weighted majority (Littlestone-Warmuth 92)

```
initialize classifiers c1...cN with weight wi = 1/N each;
```

Weighted majority (Littlestone-Warmuth 92)

```
initialize classifiers c1...cN with weight wi = 1/N each;
for each example x in sequence do
```

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);
```

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 ]
```

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;
```

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,
```

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;
```

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;
```

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;
```

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;
```

- 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 (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 = draw N samples from S *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

Bagging I

Example of building training sets:

Original:	1	2	3	4	5	6	7	8
Training Set1:	2	7	8	3	7	6	3	1
Training Set2:	7	8	5	6	4	2	7	1
Training Set3:	3	6	2	7	5	6	2	2
Training Set4:	4	5	1	4	6	4	3	8

Specially useful in Decision trees and Perceptrons (high variance or sensibility to training data set)

Bagging II

Way to do majority voting when small samples or only one classifier builder

Improves unstable weak learners (= with high variance)

May degrade stable weak learners (= with low variance)

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

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

Outline

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

Boosting I (Schapire 92)

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

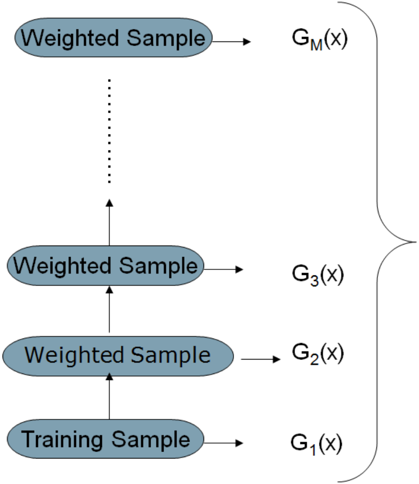
- Works on top of any *weak learner*. A weak learner is defined as any learning mechanism that works better than chance (accuracy > 0.5 when two equally probable classes)
- Adaptively, *sequentially*, creating classifiers
- Classifiers and instances have *varying weights*
- Increase weight of incorrectly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

Preliminaries

- Only two classes
- Output: $y \in \{-1, 1\}$
- Examples: X
- Weak Classifier: G(X)
- Error de training (err_{train})

$$err_{train} = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq G(x_i))$$

Preliminaries



$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x) \right)$$

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 = training set using weights for each example
```

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)
```

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 α_t = 1/2 ln( (1 - err_t) / err_t )
```

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

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 α_t = 1/2 ln( (1 − err_t) / err_t )
    Compute new weights w_i ← (w_i / Z_t) · e^(−[α_t · y_i · G_t(x_i)])
```

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

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 α_t = 1/2 ln((1 - err_t)/err_t)
    Compute new weights w_i ← (w_i / Z_t) · e^(−[α_t · y_i · G_t(x_i)])
Return classifier:
```

$$G(x) = \operatorname{sign}\left(\sum_{t=1}^{L} \alpha_t \, G_t(x)\right)$$

Adaboost algorithm

$$\alpha_m = \frac{1}{2} \ln \left(\frac{1 - err_m}{err_m} \right) > 0$$

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

$$G(x) = \operatorname{sign}\left(\sum_{m=1}^{L} \alpha_m G_m(x) \right)$$

Adaboost algorithm

$$\alpha_m = \frac{1}{2} \ln \left(\frac{1 - err_m}{err_m} \right) > 0$$

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

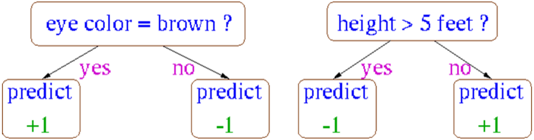
$$G(x) = \operatorname{sign} \left(\sum_{m=1}^{L} \alpha_m G_m(x) \right)$$

Simple example

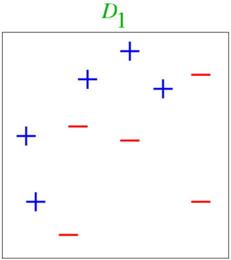
We will use *Decision stumps* as the weak learner

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

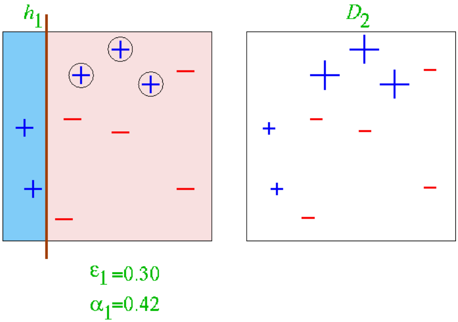
Two examples of decision stumps.



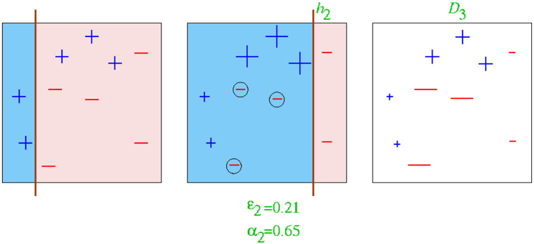
Simple example



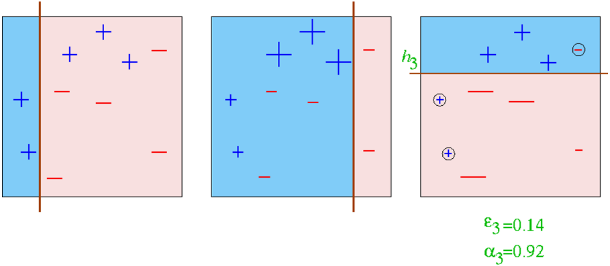
Simple example



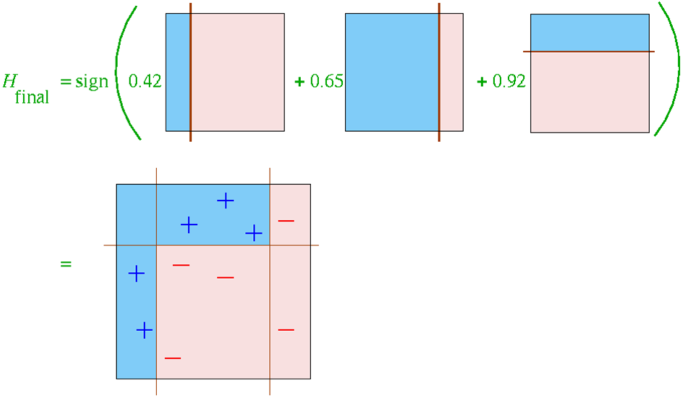
Simple example



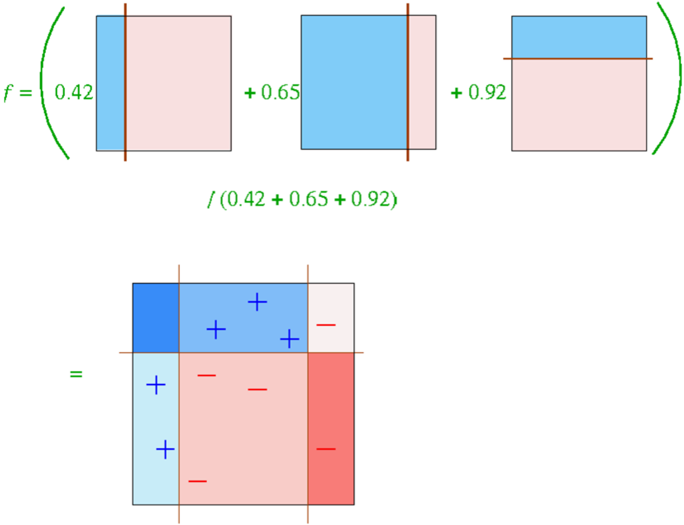
Simple example



Simple example



Simple example



AdaBoost III

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 γ

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

Possible 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

Possible reasons why this works

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

Possible 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.

Possible 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 that SVMs.