METACLASSIFIERS

Concha Bielza, Pedro Larrañaga

Computational Intelligence Group
Departmento de Inteligencia Artificial
Universidad Politica de Madrid

Machine Learning

Outline

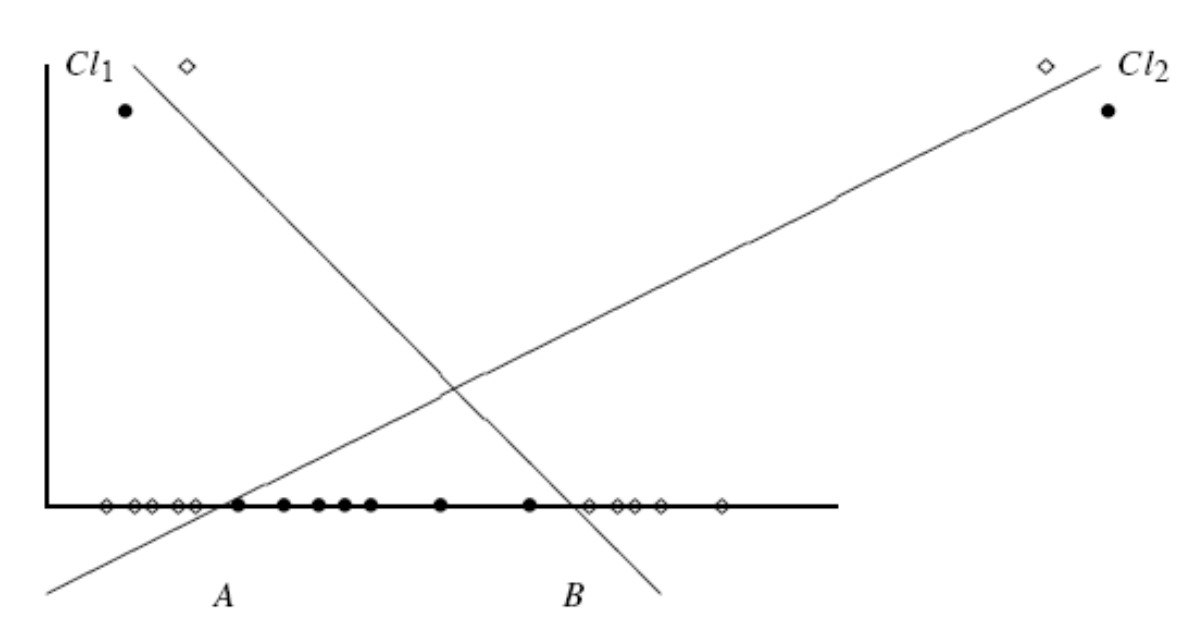
Outline

Motivation

- *Non free-lunch theorem* (Wolpert and MacReady, 1996) in machine learning: there is no a learning algorithm that in any domain always induces the most accurate classifier
  - Each algorithm converges to a different solution and fails under different circumstances
  - Even when refined for a validation set, there may exist samples where it is not accurate
  - Perhaps in those samples there is another algorithm which is well-behaved
  - Search for algorithms that make different decisions to complement each other
- Combine expert opinions (models) before making a final decision
- Each paradigm is associated with a decision region of a certain kind
- When combining different paradigms we try to obtain the suitable decision region for the problem

Motivation

Example

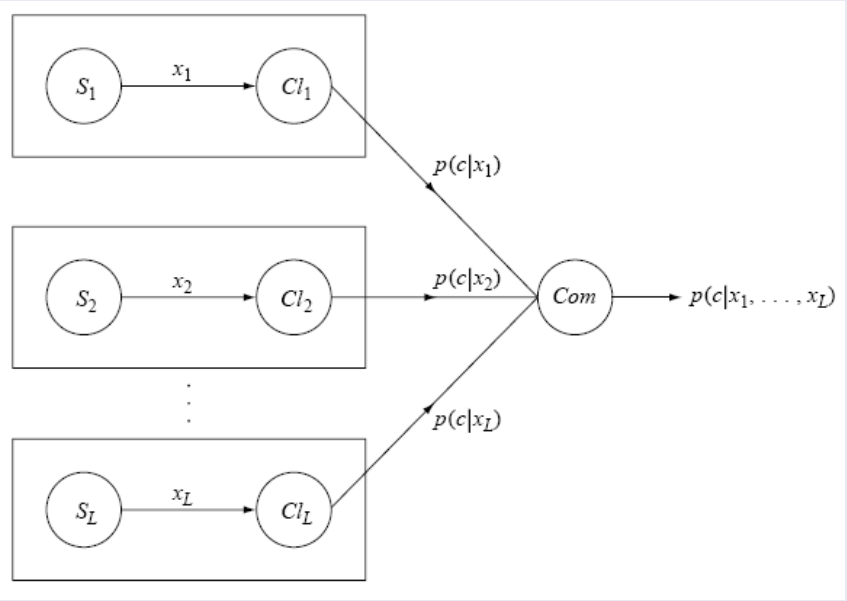


- Potential of combining classifiers for data sets with complex decision boundaries
- No single classifier is best, but misclassified samples may differ for each one
- Classifiers give complementary info: more useful when combined
- Cl_1 (• to the left of B) and Cl_2 (• to the right of A) don't obtain 100%. But better if combined with the rule: assign • if Cl_1 and Cl_2 predict •; else assign ◊.

General ideas

Different ways to characterize metaclassifiers

- Different feature spaces (perhaps using different sensors: people identification)

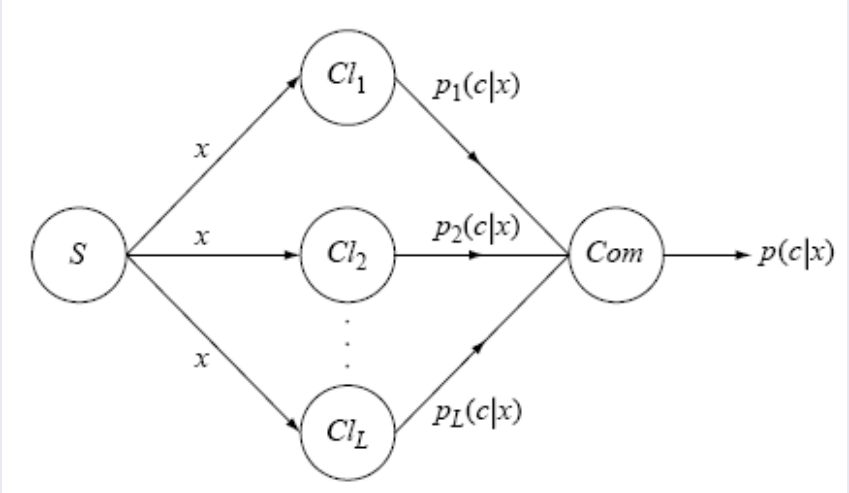


  - *Com* is the combination rule; it's itself a classifier that combines the prob. to estimate $p(c|x_1, ..., x_L)$
  - What's the best combination rule?

General ideas

Different ways to characterize metaclassifiers

- Common feature space



⇒ Classifiers may be of different kind (tree, logreg,...)

⇒ Of the same kind but trained with different samples: random (bagging), serially stressing on those misclassified by the previous classifier (boosting and cascade)

⇒ Of the same kind but with different parameters

General ideas

Different ways to characterize metaclassifiers

- Which is the best set of component classifiers?
  - Important accuracy of each one (error rate lower than random guessing)
  - Important diversity of each one (make different errors in predicting the class of a pattern **x**)
  - If all of them produce identical outputs, nothing is gained by combining the outputs

Individual accuracy are not very important because what is important is the result of the combination

General ideas

Different ways to characterize metaclassifiers

- Depending on the structure of the metaclassifier (often dictated by a practical application)
  - Parallel: results from each are passed to the combiner (voting, stacking)
  - Serial: each classifier is invoked sequentially, and uses the results from the previous classifier (cascade). They become more complex (they're used only if the previous are unreliable)
  - Hierarchical: are combined in a hierarchy, with the outputs feeding as inputs to a parent node...
- Depending on the part of the combining scheme that is optimized
  - Optimize combiner alone
  - Constituent classifiers (their parameters), for a fixed combiner rule and the number and type of constituent class.
  - Both
  - No optimization

Outline

Basic methods

- Fusion of label outputs
  - Majority vote
  - Simple majority
  - Majority vote with a threshold
  - Weighted majority vote
- Fusion of continuous-valued outputs
  - Arithmetic mean, Minimum, Maximum, Median...
- Stacked generalization
- Cascading

Fusion of label outputs

- L classifiers: $Cl_1, \ldots, Cl_L$
- Class C with R possible values: $c_1, \ldots, c_R$
- $d_{i,j}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } Cl_i \text{ classifies } \boldsymbol{x} \text{ as } c_j \\ 0, & \text{otherwise} \end{cases}$
- $Cl_i(\boldsymbol{x}) = (d_{i,1}(\boldsymbol{x}), \ldots, d_{i,R}(\boldsymbol{x}))$ with $i = 1, \ldots, L$

Example: $R = 3; L = 5$

$\boldsymbol{x}$	0 1 0	1 0 0	1 0 0	0 1 0	0 1 0

Output of Cl1 is c2

Fusion of label outputs

x	0	1	0	1	0	0	1	0	0	0	1	0	0	1	0

- Majority vote: class with more votes: $x \hookrightarrow \arg_{j=1,\ldots,R} \max \sum_{i=1}^{L} d_{i,j}(\boldsymbol{x})$

  $\sum_{i=1}^{L} d_{i,1}(\boldsymbol{x}) = 2; \quad \sum_{i=1}^{L} d_{i,2}(\boldsymbol{x}) = 3; \quad \sum_{i=1}^{L} d_{i,3}(\boldsymbol{x}) = 0$

  The sum is the number of votes cj receives

- Simple majority: the class has to have more than half of the votes
- Majority vote with a threshold

  Most genera rule

  $$x \hookrightarrow \begin{cases} c_k, & \text{if } \sum_{i=1}^{L} d_{i,k} \geq \alpha L \\ c_{R+1}, & \text{otherwise} \end{cases}$$

  where $0 < \alpha \leq 1$, $c_{R+1} \equiv$ metaclass. has no enough confidence or it's a tie

  - Simple majority: $\alpha = \frac{1}{2} + \epsilon$
  - Unanimously vote: $\alpha = 1$

- Weighted majority vote

  $x \hookrightarrow \arg_{j=1,\ldots,R} \max \sum_{i=1}^{L} w_i d_{i,j}(\boldsymbol{x})$

  where w_i is proportional to the ith-classifier goodness (normalization is usually done)

Fusion of continuous-valued outputs

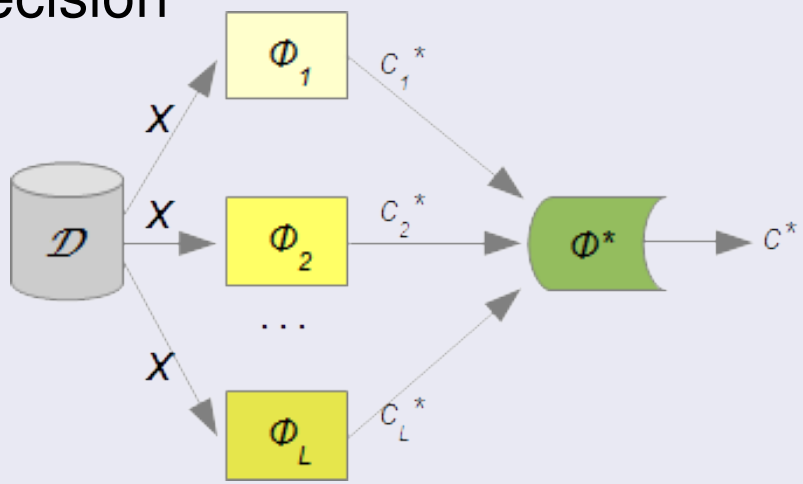
0.1	0.5	0.4	0.0	0.0	1.0	0.4	0.3	0.4	0.2	0.7	0.1	0.1	0.8	0.1

- Assign c_j with highest support $\mu_j(\boldsymbol{x}) = F(d_{1,j}(\boldsymbol{x}), \ldots, d_{L,j}(\boldsymbol{x}))$
- Examples of combination function F:
  - Arithmetic mean: $\mu_j(\boldsymbol{x}) = \frac{1}{L}\sum_{i=1}^{L} d_{i,j}(\boldsymbol{x})$ c2 has the biggest mean -> x is assigned to c2
    $\mu_1(\boldsymbol{x}) = \frac{1}{5}(0.1 + \ldots + 0.1) = 0.16$, $\mu_2(\boldsymbol{x}) = 0.46$, $\mu_3(\boldsymbol{x}) = 0.42$
  - Minimum: $\mu_j(\boldsymbol{x}) = \min_{i=1,\ldots,L}\{d_{i,j}(\boldsymbol{x})\}$ We select the maximum of the minimums
    $\mu_1(\boldsymbol{x}) = 0$; $\mu_2(\boldsymbol{x}) = 0$; $\mu_3(\boldsymbol{x}) = 0.1$
  - Maximum: $(0.4, 0.8, 1.0)$
  - Median: $(0.1, 0.5, 0.4)$ The median is not sensitive to extreme values
  - Trimmed mean (competition jury): mean after dropping extreme %
    $\mu_1^{Tri,20\%}(\boldsymbol{x}) = \frac{1}{3}(0.1 + 0.1 + 0.2) = 0.13$
    $\mu_2^{Tri,20\%}(\boldsymbol{x}) = \frac{1}{3}(0.3 + 0.5 + 0.7) = 0.5$
    $\mu_3^{Tri,20\%}(\boldsymbol{x}) = \frac{1}{3}(0.1 + 0.4 + 0.4) = 0.3$
  - Product: $\mu_j(\boldsymbol{x}) = \prod_{i=1}^{L} d_{i,j}(\boldsymbol{x})$
    $\mu_1(\boldsymbol{x}) = 0$; $\mu_2(\boldsymbol{x}) = 0$; $\mu_3(\boldsymbol{x}) = 0.0016$
  - Generalized mean: $\mu_j^{\alpha}(\boldsymbol{x}) = (\frac{1}{L}\sum_{i=1}^{L}(d_{i,j}(\boldsymbol{x}))^{\alpha})^{\frac{1}{\alpha}}$, where α is the optimism level of the combiner

Stacked generalization (Wolpert, 1992)

General idea

- Several layers (or levels) of classifiers
- Each layer uses the results obtained in the last layer
- The last layer consists of only one classifier which makes the final decision



Stacking (Wolpert, 1992)

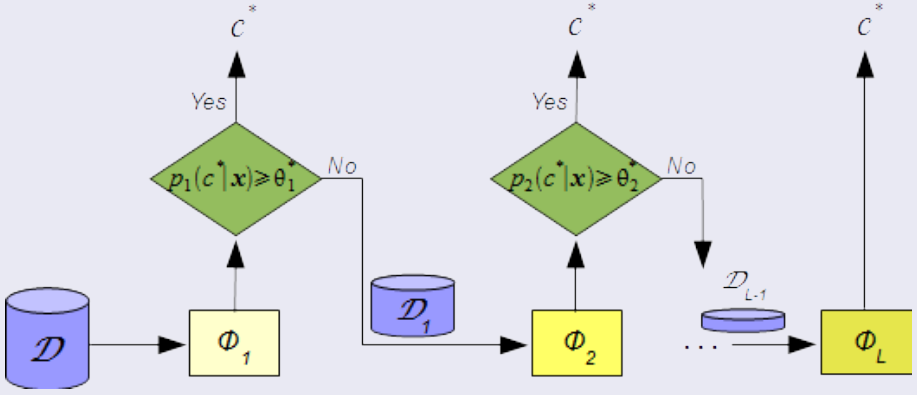
- Less widely used than bagging and boosting partly because it's difficult to analyze theoretically and partly because there are many variations
- Normally used to combine classifiers of different type (unlike bagging and boosting)
- How to combine the classifier outputs? By voting, as in bagging? $\Rightarrow$ Makes sense when all classifiers perform comparably well
- By voting it's not clear which classifier to trust; Stacking tries to learn which classifiers are the reliable ones discovering how to best combine their outputs
- Stacking builds another classifier with the outputs of the component classifiers, rather than voting
- Base models: level-0
- Predictions of these models are the input to the metamodel: level-1
- For level-1, not use the same data used to train the base learners because stacking just wants to correct their biases; learn how they make errors

$\Rightarrow$ Use holdout or k-fold cross-validation

Cascaded classifiers

General idea

- Sequence of classifiers $\phi_1, \ldots, \phi_j, \phi_{j+1}, \ldots$ sorted in terms of their space or time complexity, or the cost of the representation they use: ϕ_{j+1} is costlier than ϕ_j
- Use classifier ϕ_j for case $\boldsymbol{x}$ iff all preceding learners $\phi_1, \ldots, \phi_{j-1}$ are not confident (e.g. probabil. greater than a given threshold) when $\boldsymbol{x}$ is classified



As we move to the right, the classifie are more complex in some way

Usual decision: last classifier is KNN because it is non parametric

Cascading

Procedure

1. Train ϕ_1 and validate it with other set
2. Instances from a validation set on which ϕ_1 is not confident (because they don't surpass the threshold or they're misclassified), constitute the training set of ϕ_2, etc.
   - Classifiers are more and more complex because an early simple classifier handles the majority of instances, and a more complex is only used for a small percentage, thereby not significantly increasing the overall complexity
   - In order not to increase the number of base-classifiers, the few instances not covered by any are stored as they are and are treated by a nonparametric classifier, as k-NN
   - Classes can be explained by a small number of "rules" in increasing complexity, with an additional small set of "exceptions" not covered by the rules (best handled by a nonparametric model)

Outline

Advanced methods

- Bagging
- Randomization
- Boosting
- Hybrid

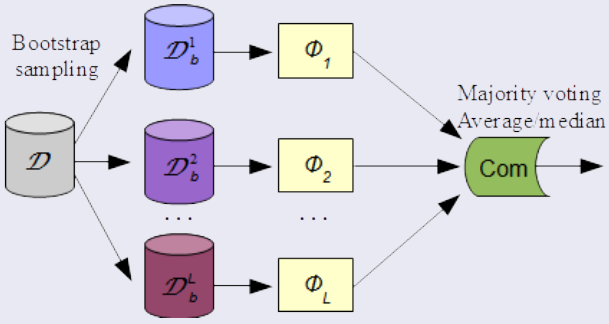
Bagging=Bootstrap AGGregatING (Breiman, 1996)

Basic ideas

- Voting method whereby base-learners are made different by training them over slightly different training sets
- Generating L slightly different samples from a given sample is done by bootstrap: given a training set $\mathcal{D}$ of size N, draw N instances randomly from $\mathcal{D}$ with replacement $\Rightarrow$ some instances repeated, others deleted
  - Prob. of being selected for the bootstrap sample is $1-(1-\frac{1}{N})^N \approx 0.63$ $\Rightarrow$ Each bootstrap sample is expected to have 63% of different instances from the training set
- Makes sense with unstable classifiers (slight changes in the training data may easily cause a large difference in the generated classifier –high variance–): trees, neural nets, rules (unstable); k-NN, Bayesian, logistic, SVMs (stable)
- For large N, bootstrap replicates will be too similar and bagging is not worth it

Bagging (Breiman, 1996)

Procedure



- Training stage
  - Let N be the number of examples in the training set
  - For $k = 1, ..., L$ (L classifiers):
    - Draw a bootstrap sample S_k from the training set
    - Learn a classifier ϕ_k by using such a sample S_k
    - Add it to the metaclassifier
- Classification stage for **x**
  - For each classifier, predict the label for **x** using each model
  - Return the class label with more votes

Bagging (Breiman, 1996)

Types of outputs to be applied

- Class labels (ok)
- Posterior probabilities: 2 options, with similar error rates
  1. Use predicted labels (with highest probability). Vote later on
  2. Average the probabilities obtaining new probabilities for each class and then choose that with the highest probability
- Numeric predictions, like model trees: take the average or the median for combining outputs

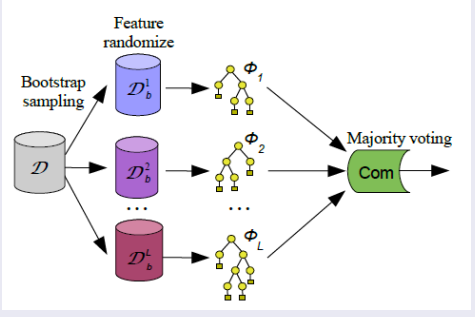
Randomization

- Bagging introduces randomness into the learning algorithm's input, often with excellent results
- There are other ways of creating diversity by introducing randomization
- Almost every learning method is amenable to some kind of randomization
  - E.g.: decision tree that randomly picks one of the *I* best options to be the next node, instead of a single winner
  - Or by choosing a random subset of options and pick the best from that

Randomization

Random forests (Breiman, 2001)

- Bagging and randomization can be combined if they introduce randomness in a complementary and different form
- E.g.: random forests, with decision trees as base classifiers: each tree is built from a random vector
- The random vector may consist of:
  - Randomness in the instances
  - Randomness in the variables, or both, by randomizing the tree in each iteration of bagging



- May be applied to many classifiers, stable ones included (unlike in bagging): the trick is to randomize to have diversity among the classifiers
- E.g.: k-NN (stable) depends on distances between instances, which in turn depends heavily on the variables used to compute them $\Rightarrow$ k-NN can be randomized by using different, randomly chosen subsets of variables

Boosting (Freund and Shapire, 1996)

It is a chained classifier

Basic ideas

- The metaclassifier builds incrementally, adding one at each iteration
- Boosting actively tries to generate complementary base-learners by training the next learner on the mistakes of the previous learners
- Training set is selectively sampled
- At the beginning, all instances have the same probability of being chosen
- Those misclassified instances in the previous iteration increase their probability of being chosen

We are going to choose the samples using bootstrap, but from the second classifier, the samples will have different probabilities of being chosen (more probability if a previous classifier misclassified the sample)

Boosting (Freund and Shapire, 1996)

AdaBoost=Adaptive Boosting

Training stage

1. Initialize parameters: weight of each of the N instances, capturing their importance: $w_j^1 = \frac{1}{N}$; number of iterations L; set of classifiers $\mathcal{C} = \emptyset$
2. For $k = 1, \ldots, L$ [*at each iteration one classifier is built*]
   - Draw a sample S_k from the data set by using distribution w_j^k
   - Learn classifier Cl_k by using S_k as training set
   - Compute Cl_k error: $\varepsilon_k = \sum_{j \text{ misclassified}} w_j^k$
   - If $\varepsilon_k = 0$ or $\varepsilon_k \geq 0.5$, terminate model generation and ignore Cl_k
   - Else, compute: $\beta_k = \frac{\varepsilon_k}{1-\varepsilon_k} (< 1)$
   - Update weights [*increase if misclassified*]:
     1. If correctly classified, $w_j^{k+1} = w_j^k \beta_k$ (decreases); Else, maintain the same weight
     2. Normalize all weights
3. Return $\mathcal{C}$ and $\beta_1, \ldots, \beta_L$

Boosting (Freund and Shapire, 1996)

AdaBoost

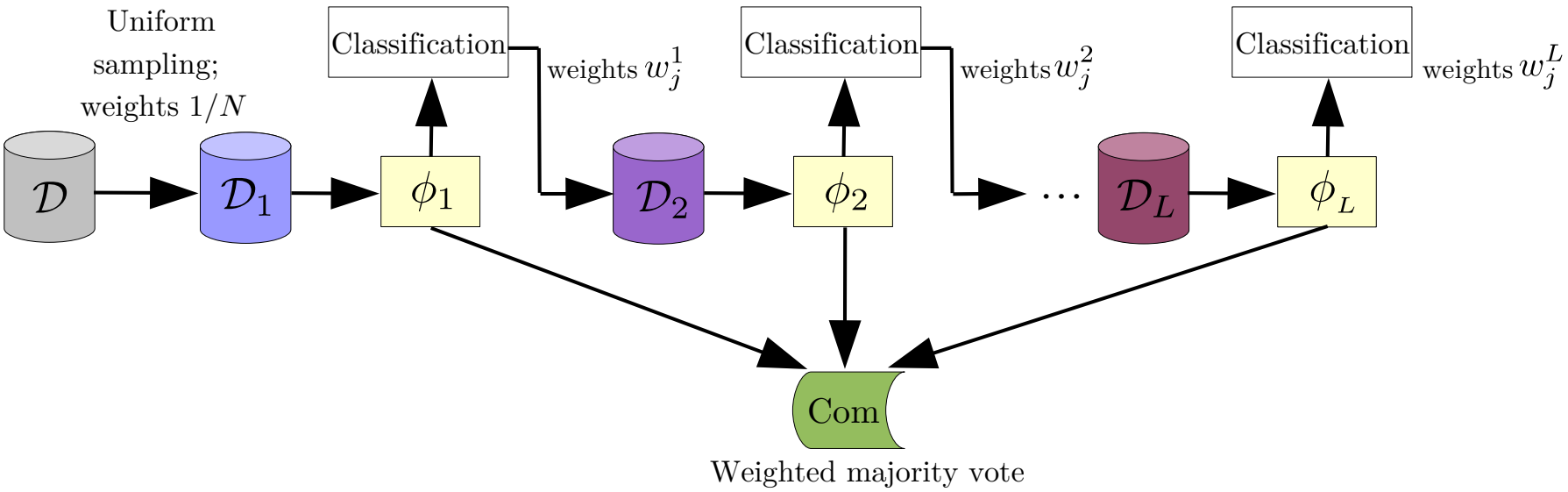
- Requires that error rates $\varepsilon_k < 1/2, \forall k$ (weak learners). If not, we stop adding new base-learners
- Each Cl_{k+1} focuses more on instances misclassified by $Cl_k \Rightarrow$ base-learners are chosen to be simple and not accurate
  (otherwise, the next sample would contain only a few outlier and noisy instances repeated many times)

Clasification stage: Weighted voting, with weights proportional to the base-learner's accuracies (on the training set): weight=$\log(1/\beta_k)$
[*classifiers with lower error should receive high weight*]

- Initialize as 0 the weights of all classes
- If Cl_k predicts c_r, sum $\log(\frac{1}{\beta_k})$ to the weight of c_r label
- Classify instance $\boldsymbol{x}$ as the class with the highest weight

$\Rightarrow$ AdaBoost considered as one of the best classifiers, and almost automatic once the base-learners are chosen

Boosting (Freund and Shapire, 1997)



- Gradient boosting (Friedman, 2001): more general; needs a loss function (log-likelihhod in classification) and uses gradient descent. Also for regression
- In scikit-learn, XGBoost...

Boosting (Freund and Shapire, 1996)

Example of AdaBoost

Training:

5 instances: $\mathbf{x}_1, ..., \mathbf{x}_5$

Weights in the beginning: $w_j^1 = 1/5 \quad \forall j$

$k = 1$: Generate S_1 from $\mathcal{U}\ \{\mathbf{x}_1, ..., \mathbf{x}_5\}$

Learn Cl_1 from $S_1 \Rightarrow$ Suppose $\mathbf{x}_2, \mathbf{x}_3$ are misclassified

Compute $e_1 = 2/5 = 0.4 \Rightarrow \beta_1 = 0.4/0.6 = 0.66$

Update weights: 1/5+1/5

		Normalize
$w_1^2 = \frac{1}{5}0.66 = 0.133$	$\Rightarrow$	0.166
$w_2^2 = \frac{1}{5} = 0.2$	$\Rightarrow$	0.25
$w_3^2 = \frac{1}{5} = 0.2$	$\Rightarrow$	0.25
$w_4^2 = \frac{1}{5}0.66 = 0.133$	$\Rightarrow$	0.166
$w_5^2 = \frac{1}{5}0.66 = 0.133$	$\Rightarrow$	0.166

Boosting (Freund and Shapire, 1996)

Example of AdaBoost (cont.)

$k = 2$: Generate S_2 from $\{\mathbf{x}_1, ..., \mathbf{x}_5\}$ with those weights
Learn Cl_2 from $S_2 \Rightarrow$ Suppose $\mathbf{x}_3$ is misclassified
Compute $e_2 = 0.25 \Rightarrow \beta_2 = 0.25/0.75 = 0.33$
Update weights:

$$\begin{array}{lll} w_1^3 = 0.166 \cdot 0.33 = 0.055 & \Rightarrow & 0.11 \\ w_2^3 = 0.25 \cdot 0.33 = 0.083 & \Rightarrow & 0.166 \\ w_3^3 = 0.25 & \Rightarrow & 0.5 \\ w_4^3 = 0.166 \cdot 0.33 = 0.055 & \Rightarrow & 0.11 \\ w_5^3 = 0.166 \cdot 0.33 = 0.055 & \Rightarrow & 0.11 \end{array}$$

$k = 3$: Generate S_3 from $\{\mathbf{x}_1, ..., \mathbf{x}_5\}$ with those weights
Learn Cl_3 from $S_3 \Rightarrow$ Suppose none is misclassified
Compute $e_3 = 0 \Rightarrow \beta_3 = 0$, Stop and delete Cl_3
Classification of $\mathbf{x}$: 3 classes c_1, c_2, c_3; Cl_1 says c_2 and Cl_2 says $c_1 \Rightarrow$
$\text{weight}(c_1) = -\log \frac{e_2}{1-e_2} = -\log 0.33$
$\text{weight}(c_2) = -\log \frac{e_1}{1-e_1} = -\log 0.66$
$\text{weight}(c_3) = 0$
Then, c_1

Boosting (Freund and Shapire, 1996)

Boosting versus bagging

- Similarities:
  - Combine models of the same type
  - Use votes to combine outputs
  - Base classifiers are generated by using the same training sample
- Differences:
  - In bagging, training samples drawn randomly; classifiers in parallel; in boosting, selectively sampled and sequential generation
  - In boosting, final prediction is based on its accuracy; bagging gives the same weights to all classifiers

	Input data	FSS	Homogeneous	Combiner	Topology
Stacking	original (layer-0), predicted (layer-1)	no	no	another classifier ϕ^*	layered
Cascading	non-confident instances	no	no	confident prediction	sequential
Bagging	boostrap samples	no	yes	voting	parallel
Random forest	boostrap samples	yes	yes (trees)	voting	parallel
Boosting	weighted samples	no	yes	weighted voting	sequential

Random forest has FSS

Hybridations

Hybrid classifiers: metaclassifier is induced taking into account 2 or more paradigms

- Lazy Bayesian Rules (Zheng and Webb, 2000): for each instance to be classified it obtains a rule with a local NB model as its consequent, created from those training examples that satisfy the antecedent of the rule
- Naive Bayes Tree (Kohavi, 1996): it induces trees with leaves that are NB classifiers, to be used for instances that reach such leaves
- Logistic Model Trees (Landwehr, Hall and Frank, 2005): as NB trees but with logistic regressions instead of NB

Outline

Combining classifiers in Weka

Stacking	Meta	Stacking and StackingC
Bagging	Meta	Bagging
Boosting	Meta	AdaBoostM1
Boosting with a Bagging variant	Meta	MultiBoostAB
Random Forest	Trees	RandomForest and RandomTree
Fusion(mean)	Meta	Vote
Fusion(artificial training smpl esp. built)	Meta	Decorate
Lazy Bayesian Rules	Lazy	LBR
Naive Bayes Tree	Trees	NBTree
Logistic Model Trees	Trees	LMT

Conclusions

- Combining classifiers to try to find decision boundaries suitable for the problem
  - Many possibilities
  - We lose interpretability (SHAP, LIME...)
- Active field of empirical research (still discussing about how and why it works)

Bibliography

Texts

- Alpaydin, E. (2004) *Introduction to Machine Learning*, The MIT Press [Chap. 15]
- Bielza, C., Larrañaga, P. (2021) *Data-Driven Computational Neuroscience. Machine Learning and Statistical Models*, Cambridge University Press [Chap. 9]
- Friedman, J.H. (2001) Greedy function approximation: A gradient boosting machine, *The Annals of Statistics*, 29(5), 1189-1232
- Kuncheva, L. (2004) *Combining Pattern Classifiers*, Wiley [especially Chap. 3, 4, 5, 7]
- Webb, A. (2002) *Statistical Pattern Recognition*, Wiley, 2nd ed. [Section 8.4]
- Witten, I., Frank, E. (2005) *Data Mining*, Morgan Kaufmann, 2nd ed. [Sections 7.5 and 10.5]