MACHINE LEARNING IN BIOINFORMATICS

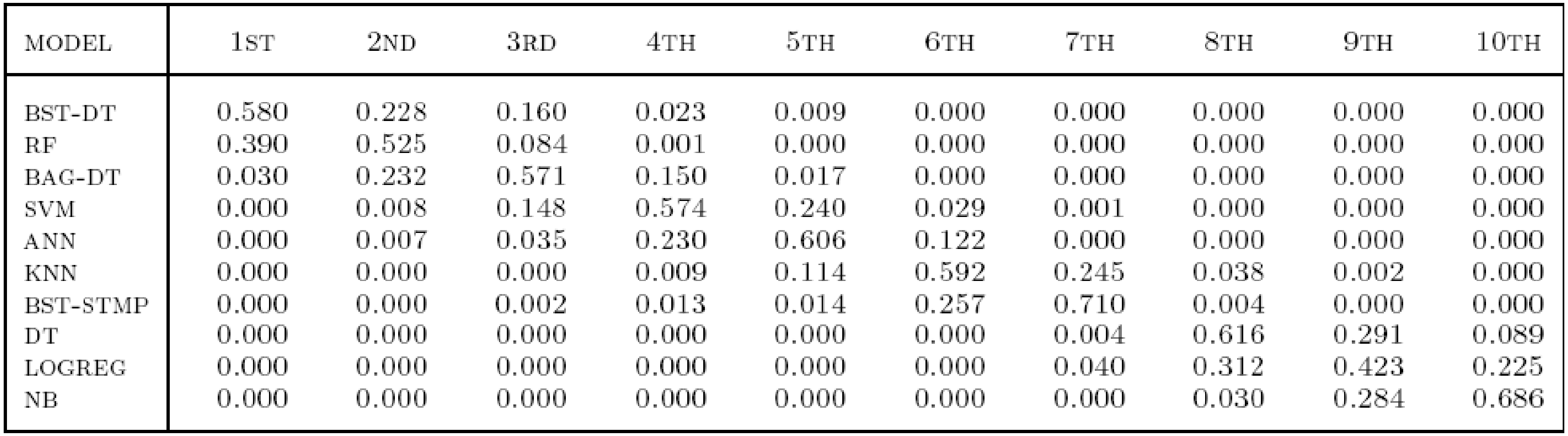
Part 6: Ensemble Classification Methods: Bagging, Boosting, and Random Forests

František Mráz
KSVI MFF UK

(Adapted slides by Junming Yin)

Empirical Comparisons of Different Classification Algorithms

Caruana and Niculesu-Mizil, ICML 2006

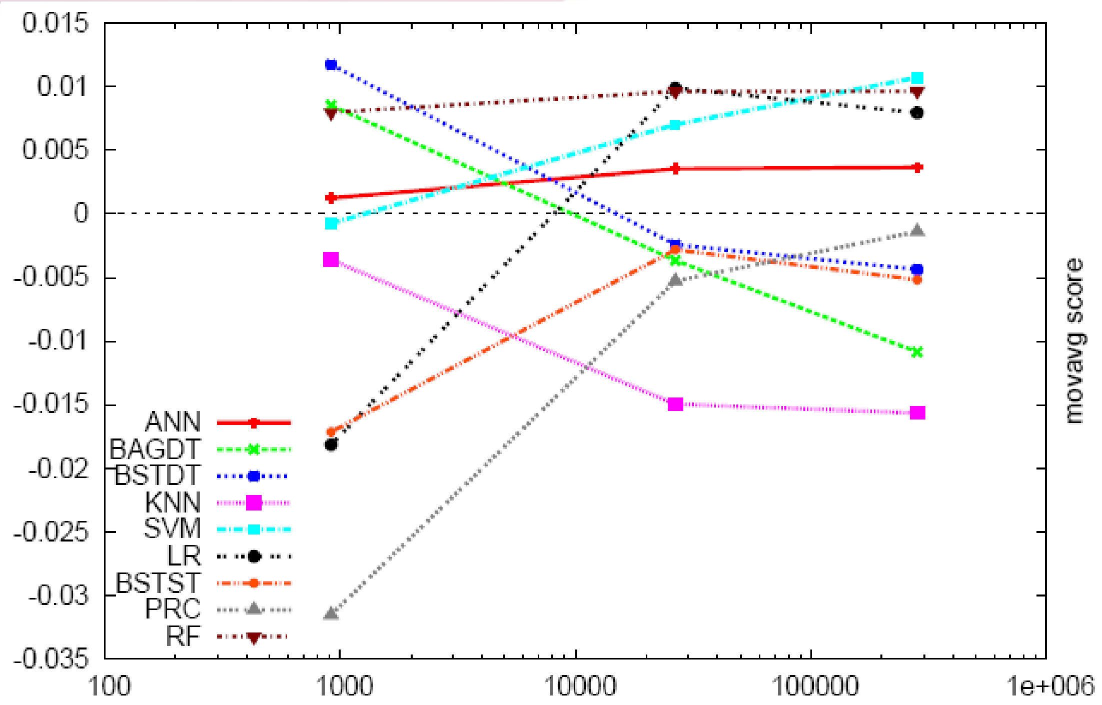


MODEL	1ST	2ND	3RD	4TH	5TH	6TH	7TH	8TH	9TH	10TH
BST-DT	0.580	0.228	0.160	0.023	0.009	0.000	0.000	0.000	0.000	0.000
RF	0.390	0.525	0.084	0.001	0.000	0.000	0.000	0.000	0.000	0.000
BAG-DT	0.030	0.232	0.571	0.150	0.017	0.000	0.000	0.000	0.000	0.000
SVM	0.000	0.008	0.148	0.574	0.240	0.029	0.001	0.000	0.000	0.000
ANN	0.000	0.007	0.035	0.230	0.606	0.122	0.000	0.000	0.000	0.000
KNN	0.000	0.000	0.000	0.009	0.114	0.592	0.245	0.038	0.002	0.000
BST-STMP	0.000	0.000	0.002	0.013	0.014	0.257	0.710	0.004	0.000	0.000
DT	0.000	0.000	0.000	0.000	0.000	0.000	0.004	0.616	0.291	0.089
LOGREG	0.000	0.000	0.000	0.000	0.000	0.000	0.040	0.312	0.423	0.225
NB	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.030	0.284	0.686

Overall rank by mean performance across problems and metrics (based on bootstrap analysis).

BST-DT: boosting with decision tree weak classifier	RF: random forest
BAG-DT: bagging with decision tree weak classifier	SVM: support vector machine
ANN: neural nets	KNN: *k*-nearest neighborhood
BST-STMP: boosting with decision stump weak classifier	DT: decision tree
LOGREG: logistic regression	NB: naïve Bayesian

Empirical Study on High-Dimension Tasks



Caruana et al., ICML 2008

Moving average standardized scores of each learning algorithm as a function of the dimension.

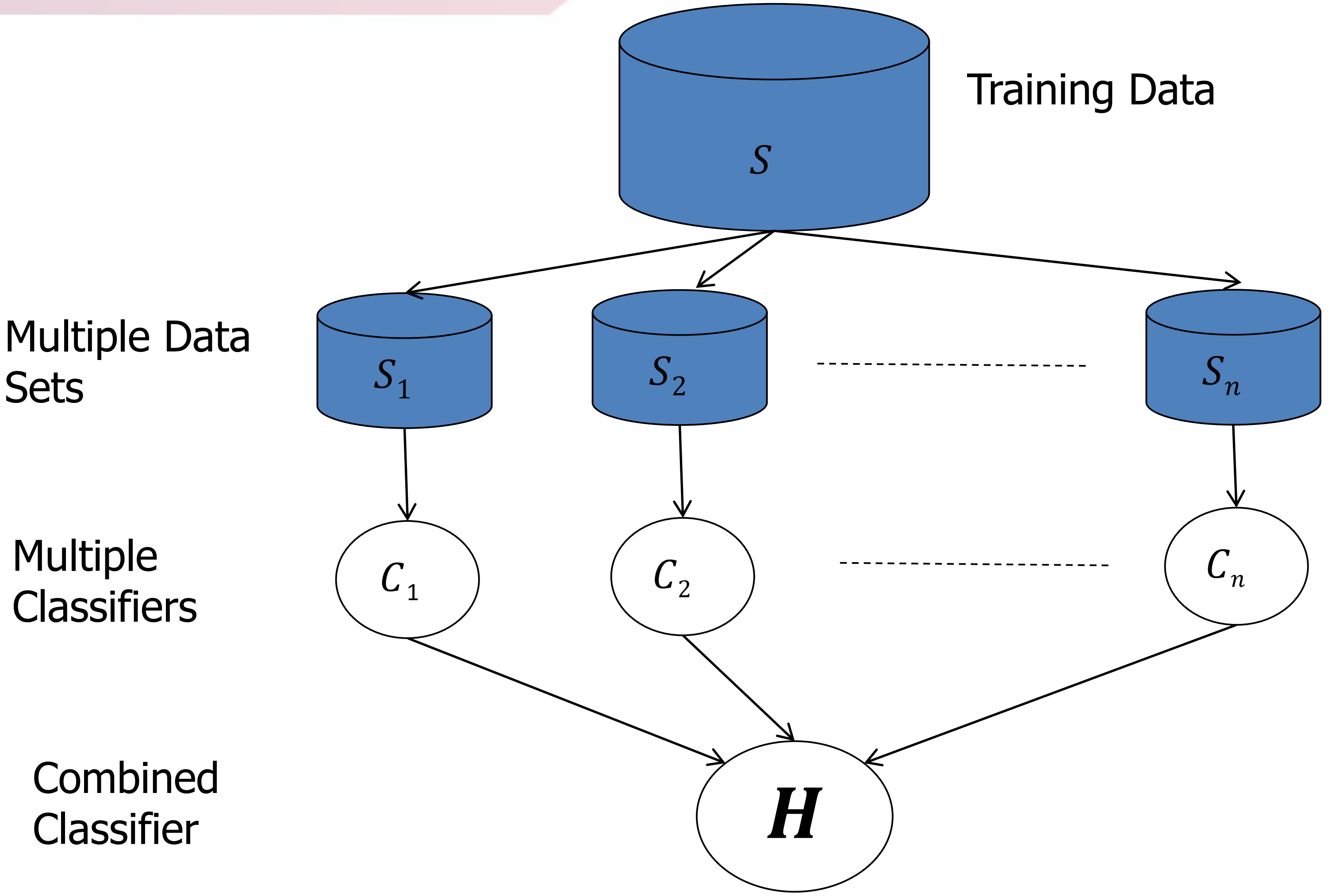
The rank for the algorithms to perform consistently well:
(1) random forest (2) neural nets (3) boosted tree (4) SVMs

Ensemble Methods

1. Bagging (Breiman 1994,…)
2. Boosting (Freund and Schapire 1995, Friedman et al. 1998,…)
3. Random forests (Breiman 2001,…)

- **Idea:** Predict class label for unseen data by aggregating a set of predictions (classifiers learned from the training data) – build different "experts", and let them vote
- **Advantages:**
  - Improved predictive performance
  - Other types of classifiers can be directly included
  - Easy to implement
  - No too much parameter tuning
- **Disadvantages:**
  - The combined classifier is not so transparent (black box)
  - Not a compact representation

General Idea



Why does it work?

- Suppose there are 25 base classifiers
- Each classifier has error rate $\varepsilon = 0.35$
- Assume independence among classifiers
- Probability that the ensemble classifier makes a wrong prediction:

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^{i} (1-\varepsilon)^{25-i} = 0.06$$

Bagging
Bootstrap **Agg**regat**ing**

- **Training**
  - Given a dataset S, at each iteration i, a training set S_i is sampled with replacement from S (i.e. bootstraping)
  - A classifier C_i is learned for each S_i
- **Classification**: given an unseen sample X,
  - Each classifier C_i returns its class prediction
  - The bagged classifier H counts the votes and assigns the class with the most votes to X
- **Regression**: can be applied to the prediction of continuous values by taking the average value of each prediction.
- Bagging works because it reduces variance by voting/averaging
  - In some pathological hypothetical situations the overall error might increase
  - Usually, the more classifiers the better

Notes on Bootstrapping

- Given a dataset of size N, a bootstrapped replicate is a dataset of the same size (i.e. N) where instances are extracted with replacement
- Given the replacement, it is possible that an instance is present more than one time in the replicate
- The fraction of never selected instances $\left(1-\frac{1}{N}\right)^N$ can be estimated as $\frac{1}{e} \approx 0.368$
- The out-of-sample dataset has therefore size $\approx\ 0.368$
- The out-of-sample dataset can be used like a validation set to estimate the importance of features

Bagging

- Problem: we only have one dataset.
- Solution: generate new ones of size N by bootstrapping, i.e. sampling it with replacement
- Can help a lot if data is noisy.

Bias-Variance Decomposition

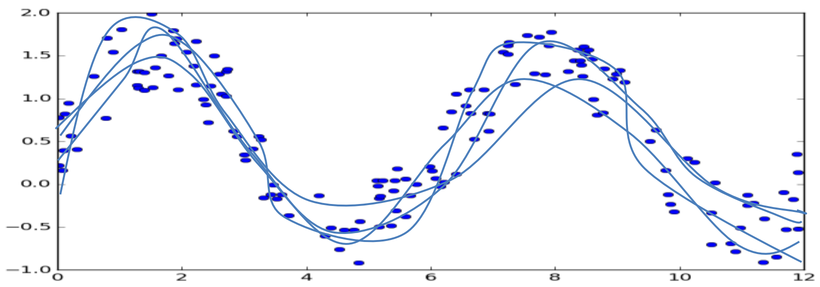
- Used to analyze how much selection of any specific training set affects performance
- Assume infinitely many classifiers, built from different training sets
- For any learning scheme,
  - ***Bias*** = expected error of the combined classifier on new data
  - ***Variance*** = expected error due to the particular training set used
- Total expected error ~ bias + variance

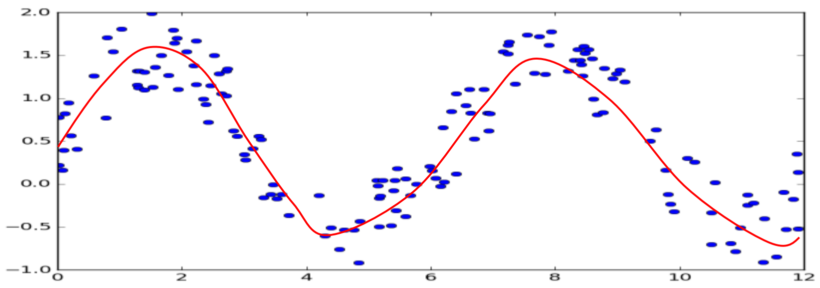
When does Bagging work?

- Learning algorithm is unstable: if small changes to the training set cause large changes in the learned classifier.
- If the learning algorithm is unstable, then Bagging almost always improves performance
- Some candidates:
  - Decision tree, decision stump (a decision tree of height 1), regression tree, linear regression, SVMs

Why Bagging works

- Let $S = \{(x_i, y_i), i = 1, \dots, N\}$ be the set of training samples
- Generate $S^{(1)}, \dots, S^{(m)}$ from bootstrapping
- Compute the $\varphi\left(x, S^{(k)}\right)$ for each $k = 1, \dots, m$
- Compute $\varphi(x, S) = E_k\left(\varphi\left(x, S^{(k)}\right)\right)$ by aggregation (voting, mean, etc.)





- We select some input samples
- We learn a regression model
- Very sensitive to the input selection
- m training sets (N items each) $= m$ different models

$$\left(x_1^{(1)}, y_1^{(1)}\right), \dots, \left(x_N^{(1)}, y_N^{(1)}\right) \rightarrow \hat{f}^{(1)}(x) = Y^{(1)}$$

$$\left(x_1^{(2)}, y_1^{(2)}\right), \dots, \left(x_N^{(2)}, y_N^{(2)}\right) \rightarrow \hat{f}^{(2)}(x) = Y^{(2)}$$

$$\vdots$$

$$\hat{f}^{(1)}, \hat{f}^{(2)}, \dots, \hat{f}^{(m)} \rightarrow Y^{(1)}, Y^{(2)}, \dots, Y^{(m)}$$

$$Z = \frac{1}{m} \sum_{i=1}^{m} Y^{(i)}$$

Proof of Convergence

- Hypothesis: The average will converge to something meaningful
  - Assumptions
    - $Y^{(1)}, Y^{(2)}, \dots, Y^{(m)}$ are iid (independent, identically distributed)
    - $E[Y] = y$ ($E[Y]$ is an unbiased estimator of y)
  - Expected error
    - $E[(Y - y)^2]$
  - With aggregation
    - $Z = \frac{1}{m}\sum_{i=1}^{m} Y^{(i)}$
    - $E[(Z - y)^2] = E\left[\left(\frac{1}{m}\sum_{i=1}^{m} Y^{(i)} - y\right)^2\right] = E\left[\left(\frac{1}{m}\sum_{i=1}^{m}\left(Y^{(i)} - y\right)\right)^2\right] =$

$$\frac{1}{m^2}\sum_{i=1}^{m}\sigma^2(Y^{(i)}) = \frac{1}{m}\sigma^2\left(Y^{(i)}\right) = \frac{1}{m}\sigma^2,$$

where $\sigma^2 = \sigma^2(Y^{(i)})$ for all $i = 1, \dots, m$

  - The variance (the expected error) of Z shrinks with m

Proof of Convergence

- If we drop the second assumption
- Assumptions
    - $Y^{(1)}, Y^{(2)}, \ldots, Y^{(m)}$ are iid (independent, identically distributed)
    - ~~$E[Y] = y$ ($E[Y]$is an unbiased estimator of y)~~

$$E[(Y-y)^2] = E[(Y - E[Y] + E[Y] - y)^2]$$
$$E[(Y-y)^2] = \boxed{E[(Y - E[Y])^2]} + E[(E[Y]-y)^2] + E[2(Y-E[Y])(E[Y]-y)]$$

$$\sigma^2(Y) \geq 0$$

$$\boxed{E\big[Y - E[Y]\big]} E[2(E[Y]-y)]$$

$$= 0$$

$$E[(Y-y)^2] \geq E[(E[Y]-y)^2]$$
$$E[(Y-y)^2] \geq E[(Z-y)^2]$$

- using Z gives us a smaller error (even if we can't prove convergence to zero)

Properties

- Instability is good
  - The more variable (unstable) the form of $\varphi(x, S)$ is, the more improvement can potentially be obtained
  - Low-variability methods (e.g. PCA, LDA) improve less than high-variability ones (e.g. Decision Trees)
- Loads of redundancy
  - Most predictors do roughly “the same thing”

From Bagging to Boosting

- Bagging: each model is trained independently
- Boosting: each model is built on top of the previous ones
  - Avoid redundancy: each learner complements the previous ones

- **Weak learner *WL***:
  - A learning algorithm $\mathcal{A}$ is a γ-weak-learner for a class of hypotheses $\mathcal{H}$ if there exists a function $m_{\mathcal{H}}: (0;\ 1) \to \mathbb{N}$ such that for every $\delta \in (0;1)$, for every distribution $\mathcal{D}$ over $\mathcal{X}$, and for every labeling function $f : \mathcal{X} \to \{-1,+1\}$, if there exists a hypothesis from $\mathcal{H}$ with zero error with respect to f and $\mathcal{D}$, then when running the learning algorithm on $m > m_{\mathcal{H}}(\delta)$ i.i.d. examples generated by $\mathcal{D}$ and labeled by f, the algorithm returns a hypothesis h such that, with probability of at least $1 - \delta$, the error of h (with respect to $\mathcal{D}$ and f) is at most $½ - \gamma$.

Weak learner

- A learning algorithm $\mathcal{A}$ is a γ-weak-learner for a class of hypotheses $\mathcal{H}$ if there exists a function $m_{\mathcal{H}}: (0;\ 1) \to \mathbb{N}$ such that for every $\delta \in (0;1)$, for every distribution $\mathcal{D}$ over $\mathcal{X}$, and for every labeling function $f : \mathcal{X} \to \{-1, +1\}$, if there exists a hypothesis from $\mathcal{H}$ with zero error with respect to f and $\mathcal{D}$, then when running the learning algorithm on $m > m_{\mathcal{H}}(\delta)$ i.i.d. examples generated by $\mathcal{D}$ and labeled by f, the algorithm returns a hypothesis h such that, with probability of at least $1 - \delta$, the error of h (with respect to $\mathcal{D}$ and f) is at most $½ - \gamma$.
- Providing classification accuracy $> 1 - \left(\frac{1}{2} - \gamma\right) = \frac{1}{2} + \gamma$
- With probability $> 1 - \delta$
- For some **fixed** and **uncontrollable**
  - classification error $½ - \gamma$
  - $\delta < \frac{1}{2}$
- **And this on an arbitrary distribution of data entries**

Decision Stumps

- Let $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{H}$ is the set of decision stumps
$$\mathcal{H} = \mathcal{H}_{DS} = \left\{x \to \operatorname{sign}\left(x_j - \theta\right) \cdot b; \theta \in \mathbb{R}, j \in \{1, \dots, d\}, b \in \{\pm 1\}\right\}$$
- A decision stump h is parameterized by a dimension j and a threshold θ
- Learning decision stump $h(x) = sign\left(x_j - \theta\right) \cdot b$
  - **Input:** training set $S = \{(x_1, y_1), \dots, (x_N, y_N)\}$
    distribution vector $D = (D_1, \dots, D_N)$

Probabilities of samples $x_1, \dots, x_N$

  - **Goal:** find b^*, j^*, θ^* solving
$$\min_{b \in \{\pm 1\}} \min_{j \in \{1,\dots,d\}} \min_{\theta \in \mathbb{R}} \left(\sum_{i; y_i = 1} D_i \mathbf{1}_{[h(x_{i,j}) = -1]} + \sum_{i; y_i = -1} D_i \mathbf{1}_{[h(x_{i,j}) = 1]} \right)$$
- Learning decision stump is possible in time $O(dN)$ – how?
- Can be used as weak learner

Learning Decision Stump

- **Goal:** find b^*, j^*, θ^* solving

$$\min_{b\in\{\pm 1\}} \min_{j\in\{1,\ldots,d\}} \min_{\theta\in\mathbb{R}} \left(\sum_{i;y_i=1} D_i \mathbf{1}_{[h(x_{i,j})=-1]} + \sum_{i;y_i=-1} D_i \mathbf{1}_{[h(x_{i,j})=1]} \right)$$

- Fix b and j; w.l.o.g. let $b = +1$
- Sort training samples according to their j-th coordinate so that

$$x_{1,j} \le x_{2,j} \le \cdots \le x_{N,j}$$

- Define $\Theta_j = \left\{\frac{x_{i,j}+x_{i+1,j}}{2}; i \in \{1,\ldots,N\}\right\} \cup \{(x_{1,j}-1), (x_{N,j}+1)\}$
- Instead of minimizing over $\theta \in \mathbb{R}$, minimize over $\theta \in \Theta_j$
- Use that the samples are sorted according to j-th coordinate
- Time $O(dN)$

Adaptive Boosting
Adaboost

[Freund and Schapire, 1997]

```
input:
    training set $S = \{(x_1, y_1), \dots, (x_N, y_N)\}$
    weak learner $\mathrm{WL}(D, S)$
    number of rounds $T$
Initialize $D^{(1)} = (1/N, \dots, 1/N)$
For $t = 1, \dots, T$
    Invoke weak learner $h_t = \mathrm{WL}(D^{(t)}, S)$
    Compute its error $\varepsilon_t = \sum_{i=1}^{N} D_i^{(t)} \mathbf{1}_{[y_i \neq h_t(x_i)]}$ and
        weight $\alpha_t = \frac{1}{2} \ln\left(\frac{1}{\varepsilon_t} - 1\right)$
    Update $D$
        $D_i^{(t+1)} = \frac{D_t^{(t)} e^{-\alpha_t y_i h_t(x_i)}}{\sum_{j=1}^{N} D_j^{(t)} e^{-\alpha_t y_j h_t(x_j)}}$
Output the hypothesis $h_S = \mathrm{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$
```

Positive for error < 0.5

AdaBoost

- Distributions $D^{(t)}$ over training samples:
  - Originally uniform
  - At each round, the weight of a misclassified example is increased
  - Let $f_t = \sum_{p=1}^{t} \alpha_p h_p$
  - Observation: $D_i^{(t+1)} = \frac{e^{-y_i f_t(x_i)}}{\sum_{j=1}^{N} e^{-y_j f_t(x_j)}}$ since

$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{-\alpha_t y_i h_t(x_i)}}{\sum_{j=1}^{N} D_j^{(t)} e^{-\alpha_t y_j h_t(x_j)}} = \frac{D_i^{(t-1)} e^{-\alpha_{t-1} y_i h_{t-1}(x_i)} e^{-\alpha_t y_i h_t(x_i)}}{\sum_{j=1}^{N} D_j^{(t-1)} e^{-\alpha_t y_j h_t(x_j)} e^{-\alpha_t y_j h_t(x_j)}} = \cdots$$

$$= \frac{\frac{1}{N} e^{-y_i \sum_{p=1}^{t} \alpha_p h_p(x_i)}}{\sum_{j=1}^{N} \frac{1}{N} e^{-y_j \sum_{p=1}^{t} \alpha_p h_p(x_j)}} = \frac{e^{-y_i f_t(x_i)}}{\sum_{j=1}^{N} e^{-y_j f_t(x_j)}}$$

AdaBoost

- The weight α_t assigned to base classifier h_t directly depends on the accuracy of h_t at round t

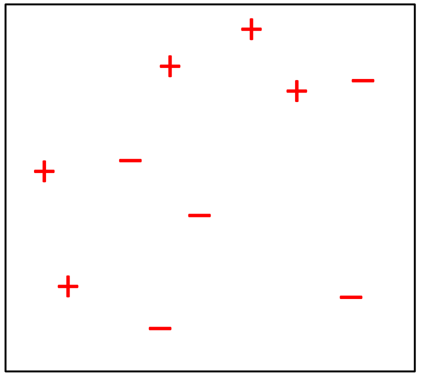
Theorem:

- Let S be a training set and assume that at each iteration of AdaBoost, the weak learner returns a hypothesis for which $\varepsilon_t < ½ - \gamma$. Then, the training error of the output hypothesis of AdaBoost is at most

$$L_S(h_S) = \frac{1}{N}\sum_{i=1}^{N} 1_{[h_S(x_i) \neq y_i]} \leq e^{-2\gamma^2 T}$$

Toy Example

Weak classifiers: decision stumps (horizontal and vertical half-planes)



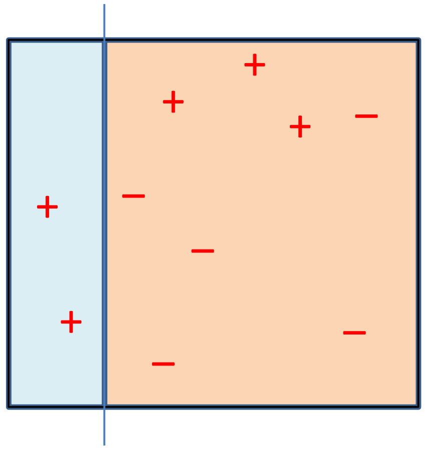
Toy Example

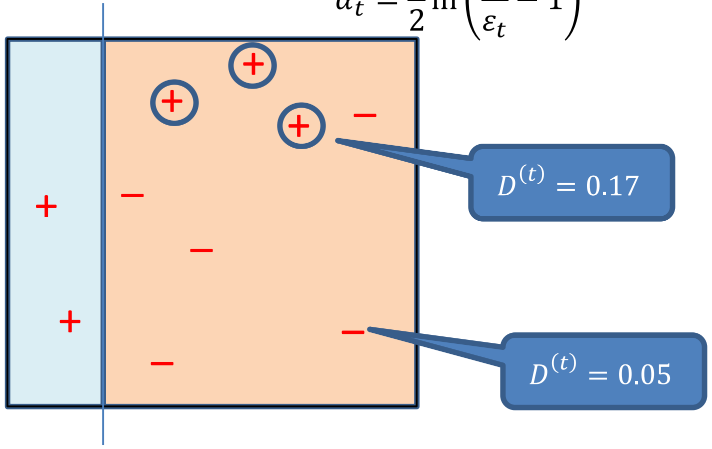
$$D_i^{(t+1)} = \frac{D_t^{(t)} e^{-\alpha_t y_i h_t(x_i)}}{\sum_{j=1}^{N} D_j^{(t)} e^{-\alpha_t y_j h_t(x_j)}}$$

$$\varepsilon_t = \sum_{i=1}^{N} D_i^{(t)} \mathbf{1}_{[y_i \neq h_t(x_i)]}$$

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

Weak classifiers: decision stumps (horizontal and vertical half-planes)





$$D_1 = \left(\frac{1}{10}, \cdots, \frac{1}{10}\right)$$

$$\varepsilon_1 = \frac{3}{10} = 0.3$$

$$\alpha_1 = 0.42$$

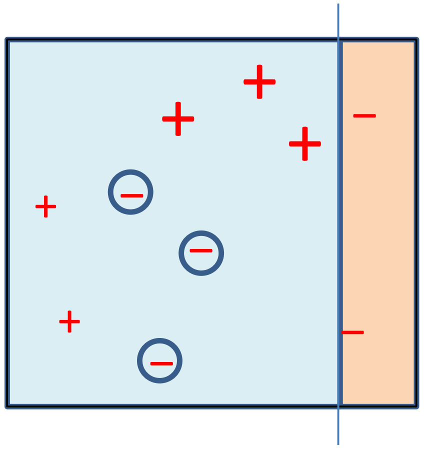
Toy Example

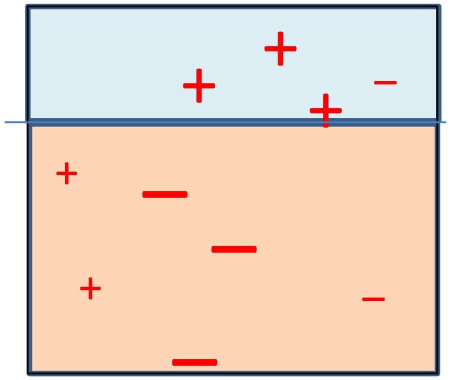
$$D_i^{(t+1)} = \frac{D_t^{(t)} e^{-\alpha_t y_i h_t(x_i)}}{\sum_{j=1}^{N} D_j^{(t)} e^{-\alpha_t y_j h_t(x_j)}}$$

$$\varepsilon_t = \sum_{i=1}^{N} D_i^{(t)} \mathbf{1}_{[y_i \neq h_t(x_i)]}$$

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

Weak classifiers: decision stumps (horizontal and vertical half-planes)

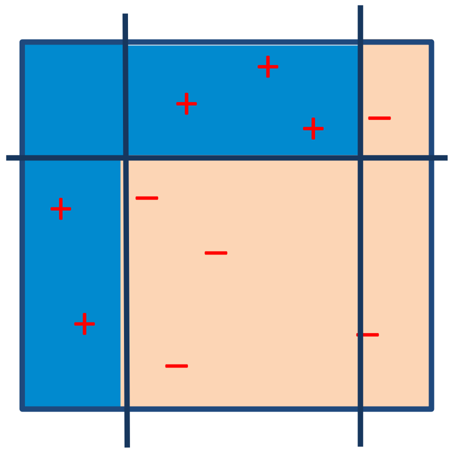




$D_2 = (0.05, \cdots, 0.17, \cdots)$
$\varepsilon_2 = 0.21$
$\alpha_2 = 0.65$

$\varepsilon_3 = 0.14$
$\alpha_3 = 0.92$
$D_3 = (\cdots)$

Toy Example



Advantages

- Simple and easy to implement
- Flexible – can combine with any learning algorithm
- No requirement on data metric – data features don't need to be normalized, like in kNN and SVMs (this has been a central problem in machine learning)
- Feature selection and fusion are naturally combined with the same goal for minimizing an objective error function
- Non-parametric – no parameters to tune (maybe T)
- No prior knowledge needed about weak learner
- Provably effective
- Versatile – can be applied on a wide variety of problems

Caveats

- Performance of AdaBoost depends on data and weak learner
- Consistent with theory, AdaBoost can fail if
  - weak classifier too complex – overfitting
  - weak classifier too weak – underfitting
- Empirically, AdaBoost seems especially susceptible to uniform noise

Random Forests Breiman 2001

- Combine tree predictors by voting
- Let us consider CART (classification and regression tree)
- We use bootstrapped replicates of the original dataset
- At each node, $m = \sqrt{p}$ candidate splitting variables are chosen uniformly at random out of the p features available
- All cut-off values are examined for all of the m variables
- The GINI impurity score is computed for each case and the (variable, cutoff) pair that empirically minimizes the score is selected
- The procedure continues recursively until a node is pure (w.r.t. the target)

GINI Impurity Score Interpretation

- GINI – a measure of how often a randomly chosen element from the set would be incorrectly labeled if it were randomly labeled according to the distribution of labels in the subset
- It can be computed by summing the probability of each item being chosen times the probability of being mistaken in choosing that label for categorizing that item
- It reaches its minimum (zero) when all cases in the set fall into a single category
- Let f_i be the fraction of items in a set labeled with label $i \in \{1,2,\ldots,m\}$, then:

$$\mathrm{Gini}(f) = \sum_{i=1}^{m} \underbrace{f_i}_{\text{Probability that an element with label } i \text{ is selected}} \underbrace{(1 - f_i)}_{\text{Probability that an element with label } i \text{ is wrongly labeled}} = \sum_{i=1}^{m} \left(f_i - f_i^2\right) = \sum_{i=1}^{m} f_i - \sum_{i=1}^{m} f_i^2 = 1 - \sum_{i=1}^{m} f_i^2$$

Random Forest Randomization

- Different types of random forests (RF) can be obtained by changing the source of randomization
  1. on training set: bootstrapped replicates
  2. on outputs: flipped for classification, with added Gaussian noise for regression
  3. on split: select randomly one from S best splits on all features
  4. on feature selection: select m features at random on each node and then find the best split
  5. on subspace: select m features at random for the whole decision tree
  6. on all: on each node select two instances of opposite class at random, select a feature at random, select a split point at random

Random Forest Key Points

- One key notion in RF is that of margin, i.e. the difference between the average vote for the true class and the highest average vote for any other class
- It can be shown that RF do not over-fit as more trees are added, but rather converge to the limiting value of the generalization error
- Moreover, given the set of decision trees, the error is bound by their strength (i.e. the expected margin) and inversely by their mean correlation
    - ➢ large strength and low correlation → good generalization
- These results explain the reason behind the randomization procedures: to encourage a low correlation between predictors

Random Forest Key Points

- RF has better prediction accuracy as it smooths the hard cut decision boundaries (compared to a single decision tree)

  ⇒ it reduces variance
- The random selection of splitting variables allows weaker variables to be considered

  ⇒ interaction effects can be revealed with other variables that would have been otherwise overshadowed

Variable Importance (VI)

- Given an induced RF, one can rank the importance of each feature according to:
  1. **Gini Score**: at each split, the decrease in Gini impurity for the variable chosen for splitting is recorded; the importance of a variable is its *average Gini impurity decrease* (averaged over all the trees in the forest)

$$\mathrm{Gini}(f) = 1 - \sum_{i=1}^{m} f_i^2$$

  1. **OOB Accuracy:** the importance is defined as the mean decrease in accuracy averaged over the bootstrapped validation replicates
     - Given that bootstrapped replicates are used to induce each decision tree, one can use the remaining 30% of data called out-of-bag (OOB), as validation set (no need to run an external cross-validation)
     - We call it the *Permutation Variable Importance Estimate* $\Rightarrow$ see the next slide

Permutation VI Estimate

- For each bootstrapped replicate
  - Identify the OOB instances
  - Predict the class membership and cumulate the correct prediction counts
  - For each feature $j \in \{1,2,\ldots,d\}$
    - permute the values of $\mathbf{x}_j$ in the OOB sample
    - use the learned decision tree to make predictions
    - cumulate the correct prediction counts
  - the importance for variable j is the average difference in accuracy on the OOB between the original setting vs. the permuted case

Permutation VI Estimate

- The permutation VI can be used in a backward elimination loop for increased reliability:
  - Compute the importance of each feature on the full dataset
  - Remove a fixed percentage of the low scoring features (say 10-20%)
  - Iterate the training and VI estimate until only one predictor is left

    This procedure allows to find a nested set of variables and as a consequence a more robust variable ranking
  - ...or, iterate until the out-of-bag error is within one standard deviation from the minimum out-of-bag error estimate

Permutation VI Estimate

- Rational for the Permutation VI Estimate:
    - When we randomly permute a predictor variable, its original association with the target is lost
    - We then use the permuted variable, together with the remaining non-permuted predictor variables, to predict the target
    - In this case, the prediction accuracy is likely to decrease substantially *if the original variable was associated with the response*

Remarks on VI Estimates in general

- Gini Impurity Score is affected by multiple testing effects and has been shown to be biased when variables vary in their number of categories or scale of measurement (variables with more categories are preferred because multiple testing is not corrected for)
- The permutation importance is reliable when sub-sampling without replacement (instead of bootstrap sampling) is used
- The performance of VI measures may not be reliable when variables are correlated: random forests show a preference for correlated variables

Remarks on VI Estimates in general

- The performance of VI measures may not be reliable when variables are correlated: random forests show a preference for correlated variables
  - To remove this bias, one can permute the values of the selected feature only within the intervals that result from the partition made by all the other cut-offs choices in the decision tree[*] for all the variables that are related (e.g. use the correlation coefficient)
  - The idea is that VI really estimates the independence of a variable from the target AND from the other variables – if there exist correlated variables this is not true and the resulting indication is an overestimate

[*] Strobl, C., Boulesteix, A. L., Kneib, T., Augustin, T., & Zeileis, A. (2008). Conditional variable importance for random forests. BMC bioinformatics

Remarks on VI Estimates in general

- Variable importance $\propto$ the loss of accuracy (caused by randomly permuting attribute values between objects)
- computed separately for all trees in RF, averaged and standard deviation of the accuracy loss is computed. Alternatively, the Z-score $\frac{average\ loss}{std(average\ loss)}$ can be used as the importance measure.
- unfortunately, the Z score cannot measure the statistical significance of the feature importance in RF, since its distribution is not $\mathcal{N}(0;1)$ [Rudnicki, Kierczak, Koronacki, and Komorowski 2006].
- we need something to decide whether the VI of a given attribute is significant, i.e. distinguishable from importance obtained by random fluctuations

Boruta algorithm

Boruta comes from the mythological Slavic figure that embodies the spirit of the forest.

1. Add shadow variables as copies of all variables (always at least 5 shadow attributes) and randomly shuffle values of each shadow variable
2. Run a RF classifier on the extended set of attributes and compute the Z-scores for all variables.
3. Find the maximum Z-score among shadow attributes (MZSA), and then assign a hit to every attribute that scored better than MZSA.
4. For each attribute with undetermined importance perform a two-sided test of equality with the MZSA.

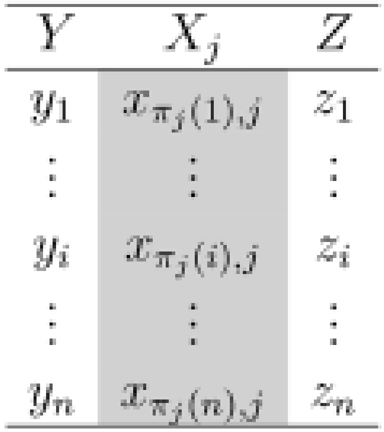
- Deem the attributes which have importance significantly lower than MZSA as ‘unimportant’ and permanently remove them
- Deem the attributes which have importance significantly higher than MZSA as ‘important’.
- Remove all shadow attributes.
- Repeat the procedure until the importance is assigned for all the attributes, or the algorithm has reached the previously set limit of the random forest runs.

Z-score used in Boruta Algorithm

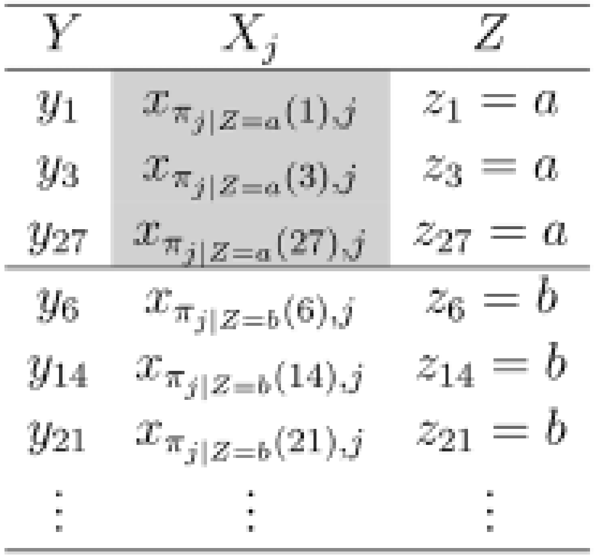
- The importance measure of an attribute = the loss of accuracy of classification caused by the random permutation of attribute values between objects.
- It is computed separately for all trees in the forest which use a given attribute for classification.
- Then the average and standard deviation of the accuracy loss are computed.
- Alternatively, the Z score computed by dividing the average loss by its standard deviation can be used as the importance measure.
- The Z score is not directly related to the statistical significance of the feature importance returned by the random forest algorithm, since its distribution is not $\mathcal{N}(0,1)$ [Rudnicki, Kierczak, Koronacki, and Komorowski 2006].
- Nevertheless, in Boruta Z score is used as the importance measure since it takes into account the fluctuations of the mean accuracy loss among trees in the forest.

Conditional Variable Importance

- Use a conditional permutation scheme, where x_j is permuted only within groups of observations with $Z = z$, to preserve the correlation structure between x_j and the other predictor variables



Y	X_j	Z
y_1	$x_{\pi_j(1),j}$	z_1
$\vdots$	$\vdots$	$\vdots$
y_i	$x_{\pi_j(i),j}$	z_i
$\vdots$	$\vdots$	$\vdots$
y_n	$x_{\pi_j(n),j}$	z_n

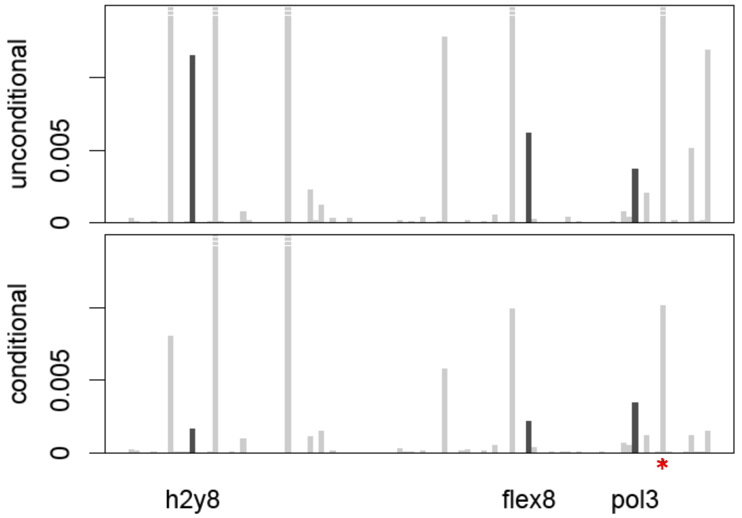


Y	X_j	Z
y_1	$x_{\pi_{j\|Z=a}(1),j}$	$z_1 = a$
y_3	$x_{\pi_{j\|Z=a}(3),j}$	$z_3 = a$
y_{27}	$x_{\pi_{j\|Z=a}(27),j}$	$z_{27} = a$
y_6	$x_{\pi_{j\|Z=b}(6),j}$	$z_6 = b$
y_{14}	$x_{\pi_{j\|Z=b}(14),j}$	$z_{14} = b$
y_{21}	$x_{\pi_{j\|Z=b}(21),j}$	$z_{21} = b$
$\vdots$	$\vdots$	$\vdots$

*

Experimental Results

- Dataset of $N = 300$ amino acid sequences described by 13 properties for each amino acid in each of 8 specific positions ($p = 104$) for a classification task (binding)



- The VI of h2y8 seems higher than that of pol3 however the VI of h2y8 is over-estimated by the unconditional approach since it is highly correlated with another important variable (the one marked with *); the rank is corrected in the conditional permutation scheme