ENSEMBLES Y MULTICLASIFICACIÓN

Minería de Datos: Preprocesamiento y clasificación

Descripción

- **Temario**:
  - Tema 1. Modelos no lineales, lineales avanzados y redes neuronales.
  - Tema 2. Árboles de decisión. Aprendizaje de Reglas.
  - Tema 3 y 4. Multiclasificación, Ensembles y Descomposición de problemas multiclase.
  - Tema 5. Máquinas soporte vectorial (SVM).
  - Tema 6. Preprocesamiento de datos.

- **Bibliografía**:

  - “The Elements of Statistical Learning: Data Mining, Inference, and Prediction”, Trevor Hastie, Robert Tibshirani, Jerome Friedman. Second Edition, Springer, 2009.
  - "Data Preparation for Data Mining". Dorian Pyle. Morgan Kaufmann, 1999.
  - "Data Preprocessing in Data Mining". Salvador García, Julián Luengo, Francisco Herrera. Springer, 2015.
  - "Data Mining: Concepts and Techniques. Jiawei Han, Jian Pei, Micheline Kamber, 2011.
  - “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems”. Aurélien Géron. O’Reilly, 2019.
  - “Hands-On Data Preprocessing in Python: Learn how to effectively prepare data for successful data analytics”. Roy Jafair. Packt, 2022.

Descripción

- **Temario**:
  - Tema 1. Modelos no lineales, lineales avanzados y redes neuronales.
  - Tema 2. Árboles de decisión. Aprendizaje de Reglas.
  - Tema 3 y 4. Multiclasificación, Ensembles y Descomposición de problemas multiclase.
  - Tema 5. Máquinas soporte vectorial (SVM).
  - Tema 6. Preprocesamiento de datos.

- **Bibliografía**:
  - ”The Elements of Statistical Learning: Data Mining, Inference, and Prediction”, Trevor Hastie, Robert Tibshirani, Jerome Friedman. Second Edition, Springer, 2009.
  - "Data Preparation for Data Mining". Dorian Pyle. Morgan Kaufmann, 1999.
  - "Data Preprocessing in Data Mining". Salvador García, Julián Luengo, Francisco Herrera. Springer, 2015.
  - "Data Mining: Concepts and Techniques. Jiawei Han, Jian Pei, Micheline Kamber, 2011.
  - ”Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems”. Aurélien Géron. O’Reilly, 2019.
  - “Hands-On Data Preprocessing in Python: Learn how to effectively prepare data for successful data analytics”. Roy Jafair. Packt, 2022.

Rationale for Ensemble Learning

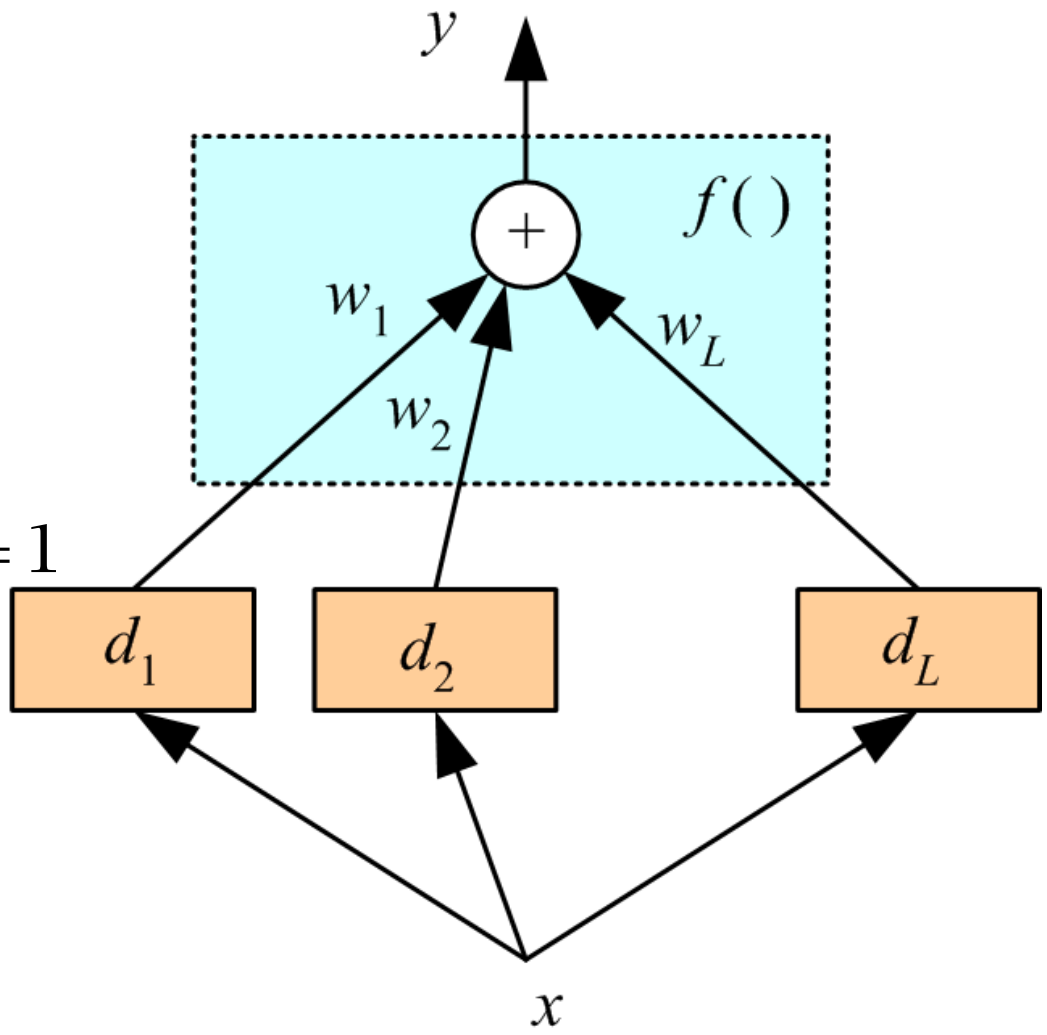
- No Free Lunch theorem: There is no algorithm that is always the most accurate
- Generate a group of base-learners which when combined have higher accuracy
- **Diversity** and **Accuracy** among classifiers are the key points for the success of ensembles.
- Different learners use different
  - Algorithms / Parameters
  - Training sets / Subproblems

Voting

- Linear combination

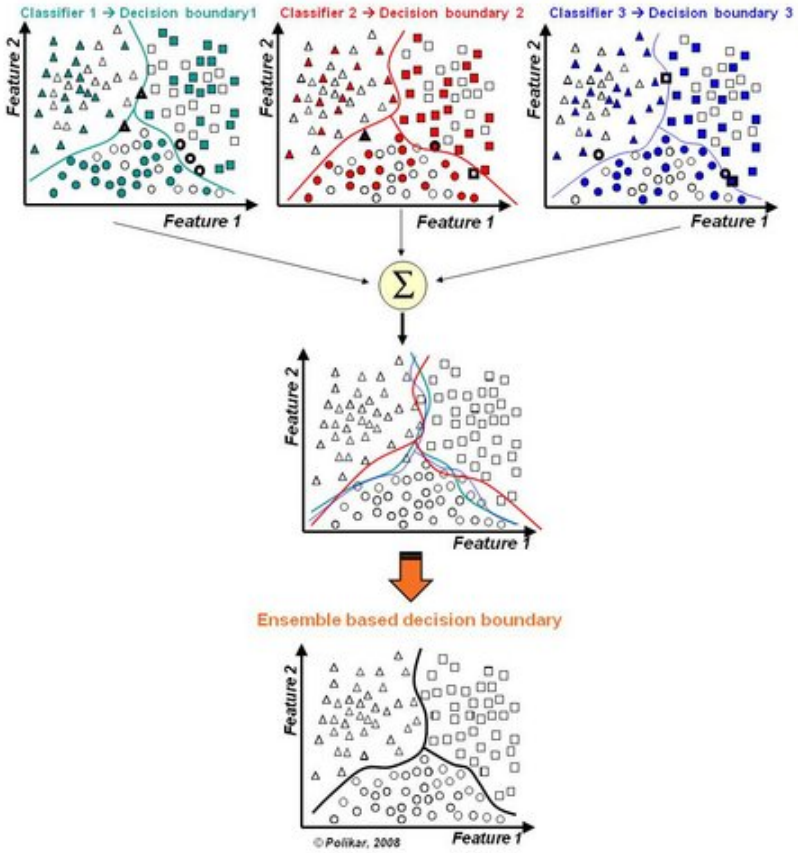
$$y = \sum_{j=1}^{L} w_j d_j$$

$$w_j \geq 0 \text{ and } \sum_{j=1}^{L} w_j = 1$$

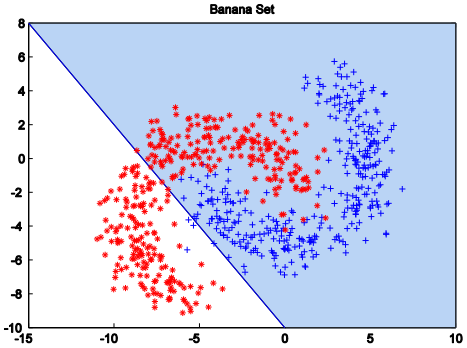


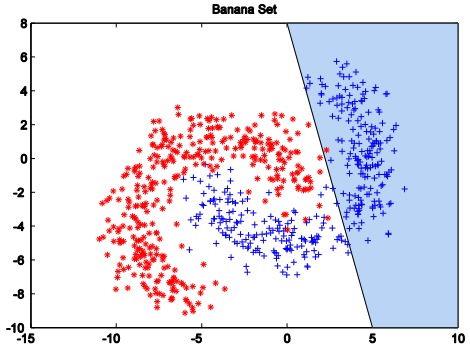
Ensembles, multiclasificadores,…

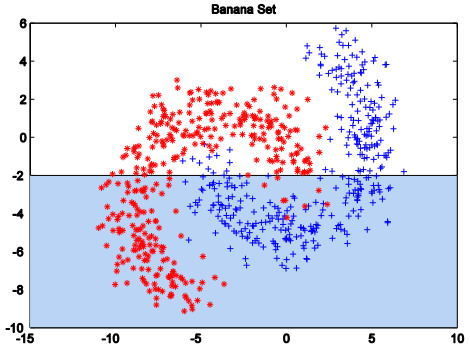
- La idea es inducir *n* clasificadores en lugar de uno solo.
- Para clasificar se utilizará una combinación de la salida que proporciona cada clasificador.
- Los clasificadores pueden estar basados en distintas técnicas (p.e. árboles, reglas, instancias,…).
- Se puede aplicar sobre el mismo clasificador o con diferentes.

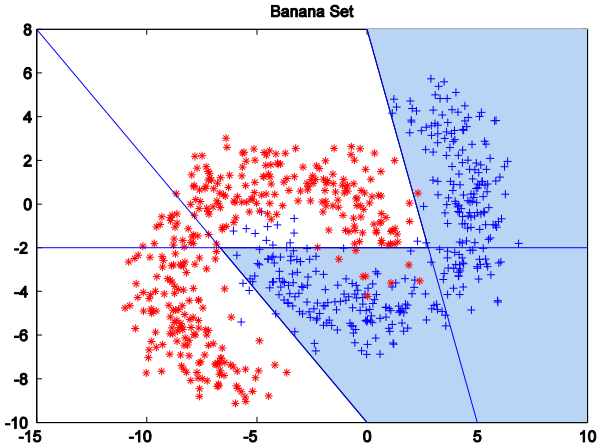


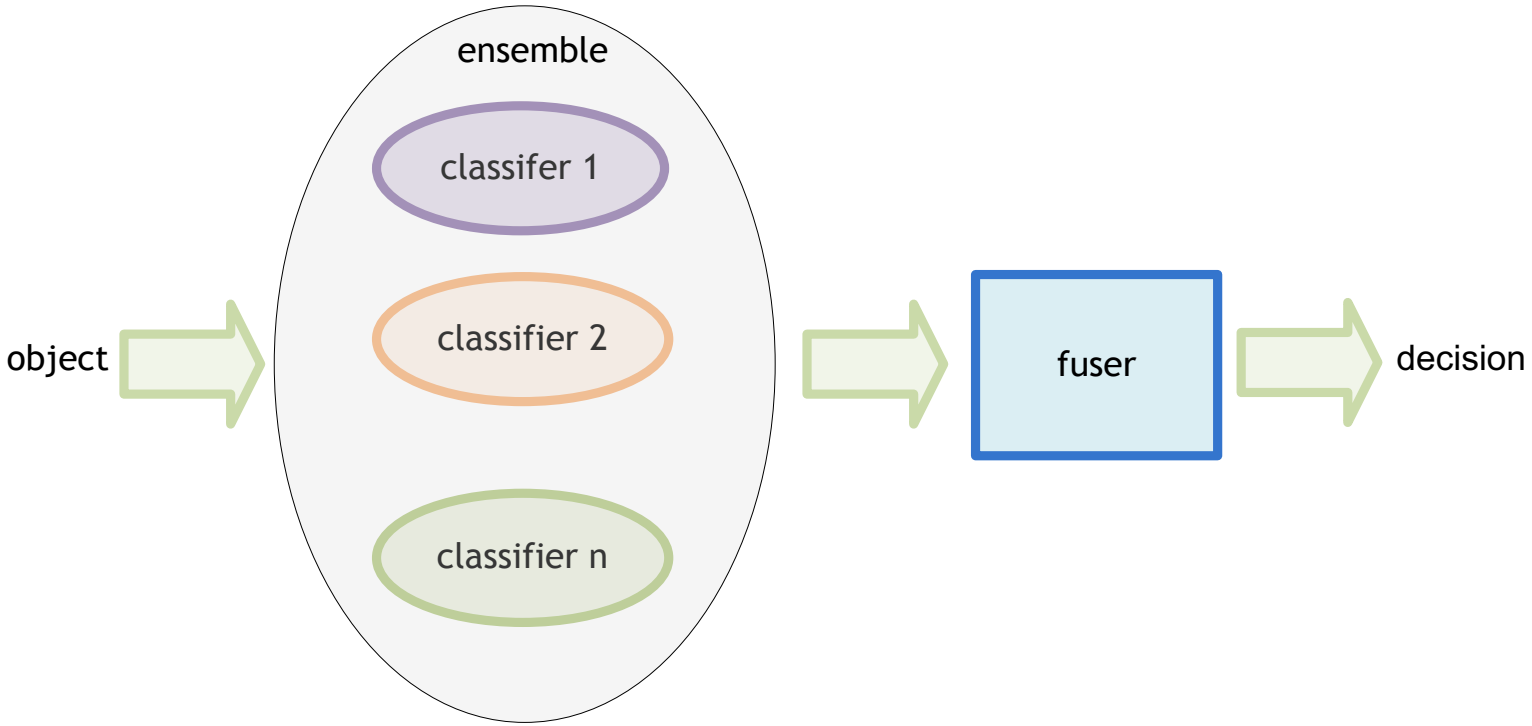
Bagging
Random Forest
Boosting


Banana Set


Banana Set


Banana Set


Banana Set


ensemble
classifer 1
classifier 2
classifier n
object
fuser
decision

Bagging

- Bootstrap aggregation, or bagging [Breiman, 94], is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.

- Recall that given a set of n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n.

- In other words, averaging a set of observations reduces variance. Of course, this is not practical because we generally do not have access to multiple training sets.

Bagging — continued

- Instead, we can bootstrap, by taking repeated samples from the (single) training data set.
- In this approach we generate B different bootstrapped training data sets. We then train our method on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$, the prediction at a point x. We then average all the predictions to obtain

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x).$$

This is called bagging.

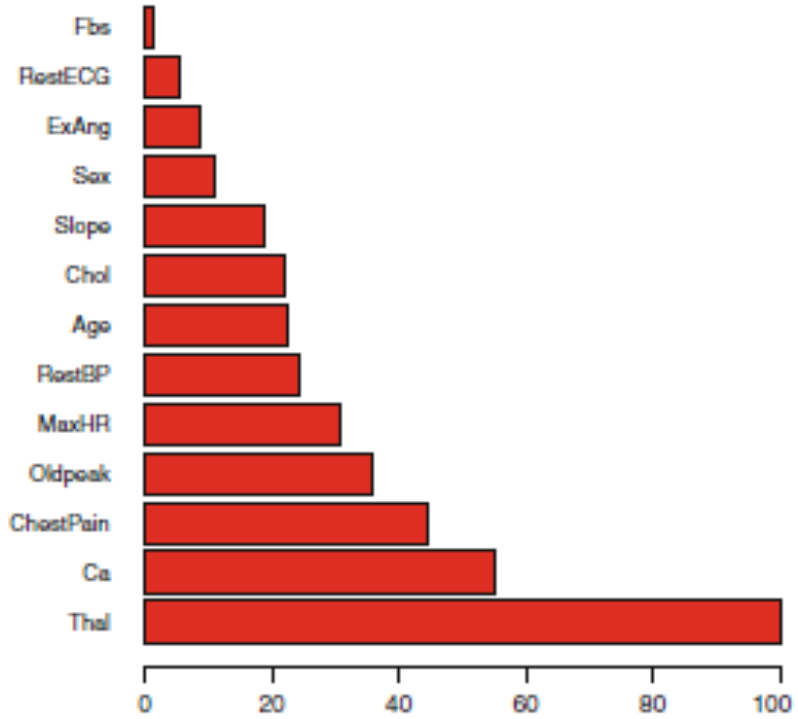
Bagging classification trees

- The above prescription applied to regression trees.
- For classification trees: for each test observation, we record the class predicted by each of the B trees, and take a majority vote: the overall prediction is the most commonly occurring class among the B predictions.

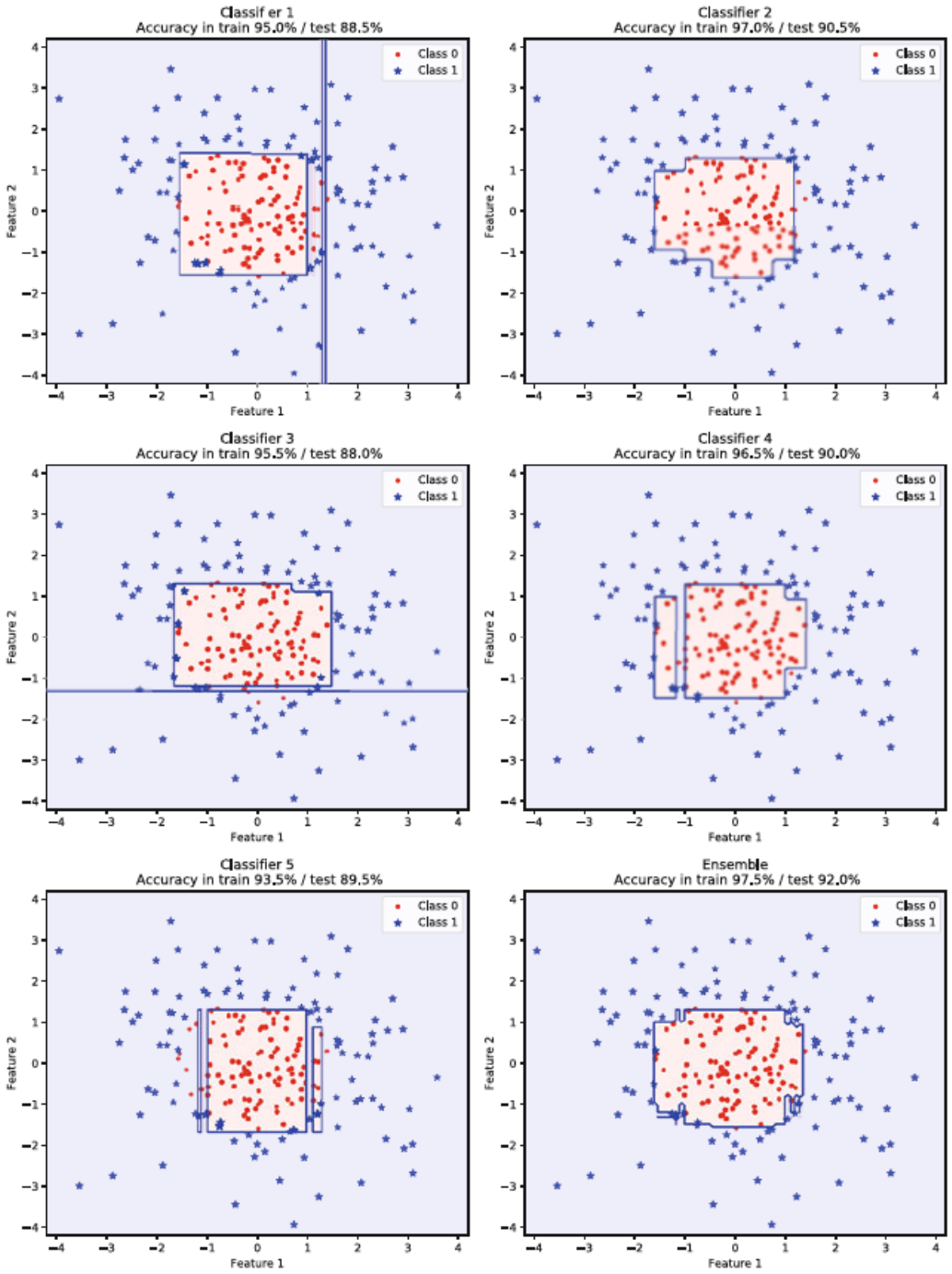
Out-of-Bag Error Estimation

- It turns out that there is a very straightforward way to estimate the test error of a bagged model.
- Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. One can show that on average, each bagged tree makes use of around two-thirds of the observations.
- The remaining one-third of the observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations.
- We can predict the response for the ith observation using each of the trees in which that observation was OOB. This will yield around B/3 predictions for the ith observation.
- The resulting OOB error is a valid estimate of the test error for the bagged model.

When we bag a large number of trees, it is no longer possible to represent the resulting statistical learning procedure using a single tree, and it is no longer clear which variables are most important to the procedure.

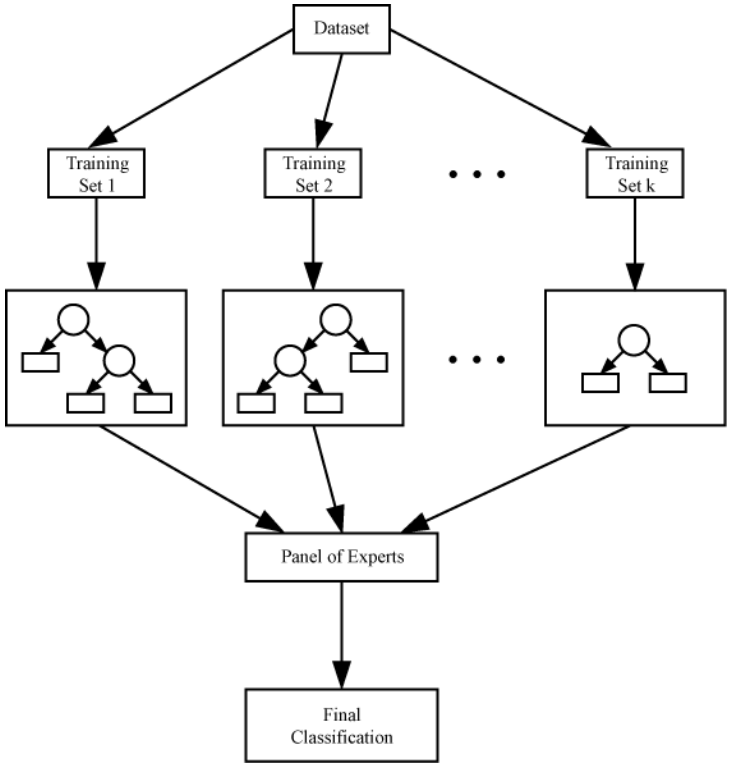


We can record the total amount that the RSS (or gain information) is decreased due to splits over a given predictor, averaged over all B trees.


Classif er 1
Accuracy in train 95.0% / test 88.5%
Classifier 2
Accuracy in train 97.0% / test 90.5%
Classifier 3
Accuracy in train 95.5% / test 88.0%
Classifier 4
Accuracy in train 96.5% / test 90.0%
Classifier 5
Accuracy in train 93.5% / test 89.5%
Ensemble
Accuracy in train 97.5% / test 92.0%
Class 0
Class 1
Feature 1
Feature 2

Random Forests

- Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.

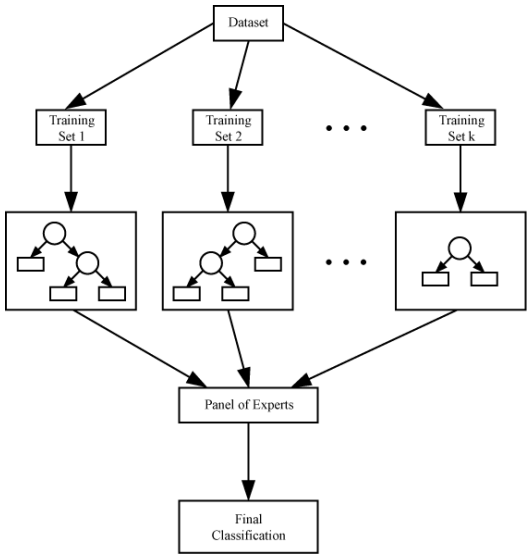


Breiman, Leo (2001). "Random Forests". *Machine Learning* **45** (1): pp. 5–32. doi:10.1023/A:1010933404324

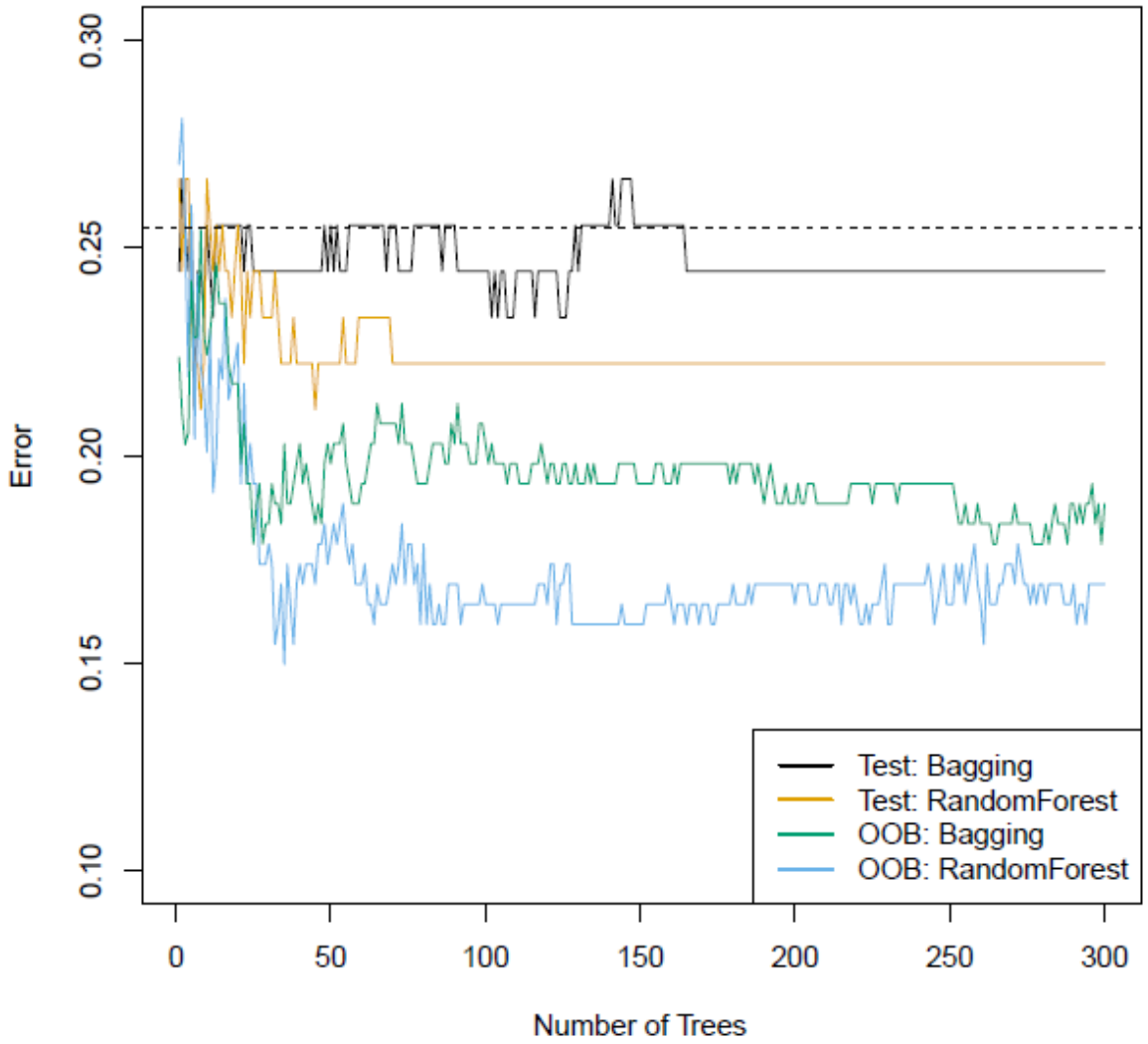
Random Forests

- But when building these decision trees, each time a split in a tree is considered, a random selection of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.

- A selection of m predictors is taken at each split, and typically we choose m $\approx\sqrt{p}$ - that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors (4 out of the 13 for the Heart data).



Random Forests: the heart data

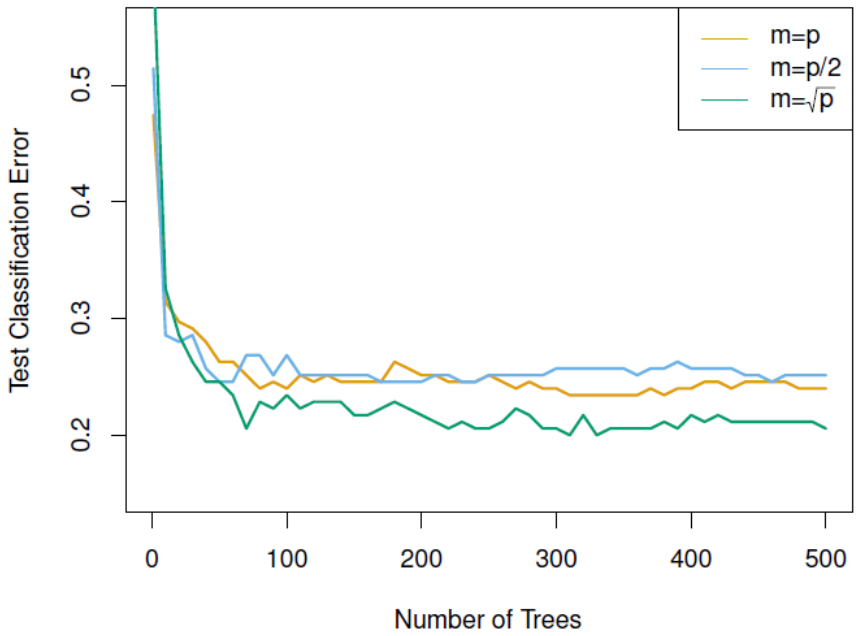


Example: gene expression data

- We applied random forests to a high-dimensional biological data set consisting of expression measurements of 4,718 genes measured on tissue samples from 349 patients.
- There are around 20,000 genes in humans, and individual genes have different levels of activity, or expression, in particular cells, tissues, and biological conditions.
- Each of the patient samples has a qualitative label with 15 different levels: either normal or one of 14 different types of cancer.
- We use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.
- We randomly divided the observations into a training and a test set, and applied random forests to the training set three different values of the number of splitting variables m.

Results: gene expression data

- Results from random forests for the fifteen-class gene expression data set with p = 500 predictors.
- The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m, the number of predictors available for splitting at each interior tree node.
- Random forests (m < p) lead to a slight improvement over bagging (m = p). A single classification tree has an error rate of 45.7%.



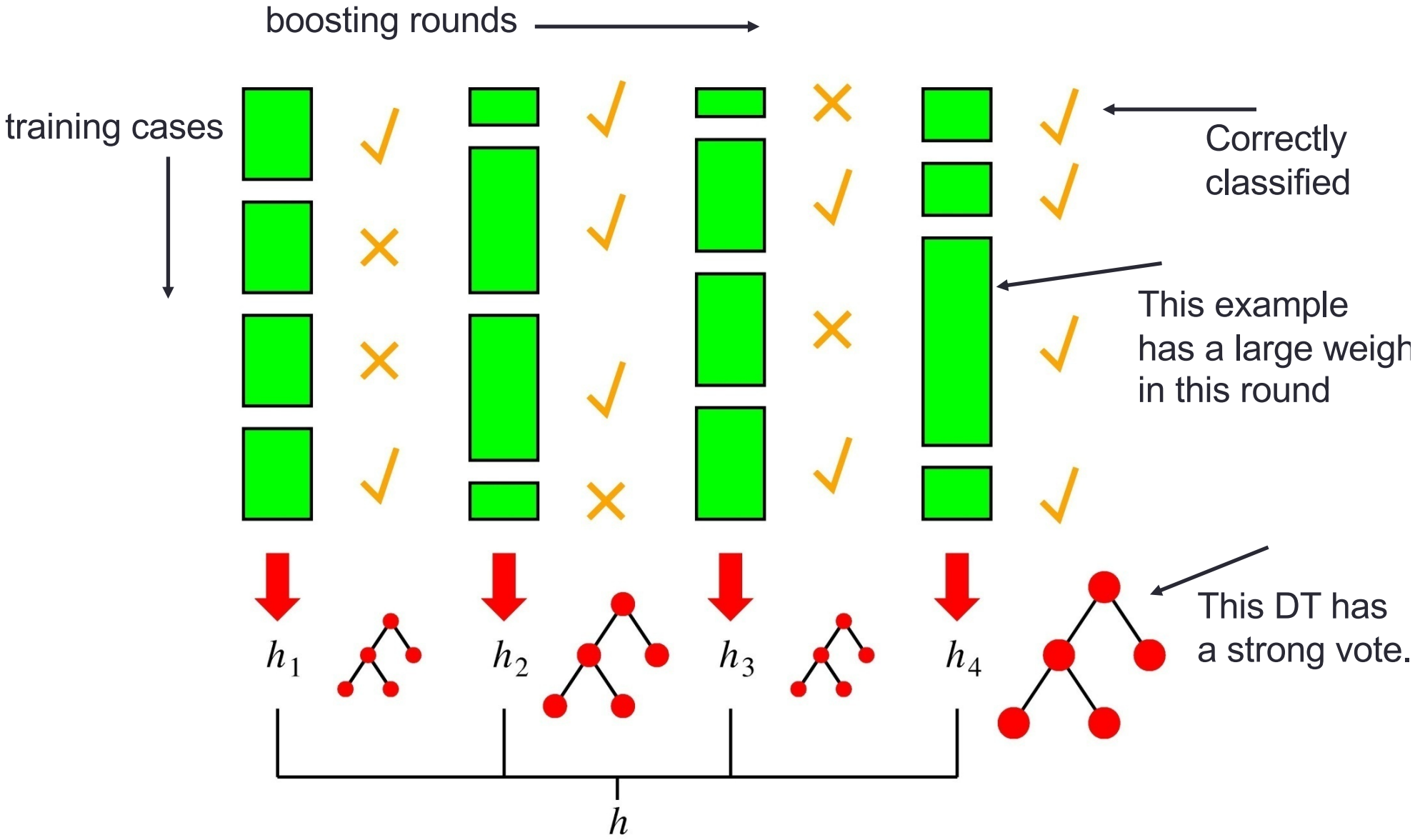
Boosting

- Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification.
- Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model.
- Notably, each tree is built on a bootstrap data set, independent of the other trees.
- Boosting works in a similar way, except that the trees are grown sequentially: each tree is grown using information from previously grown trees.

Boosting Intuition

- We adaptively weight each data case.
- Data cases which are wrongly classified get high weight (the algorithm will focus on them)
- Each boosting round learns a new (simple) classifier on the weighed dataset.
- These classifiers are weighed to combine them into a single powerful classifier.
- Classifiers that obtain low training error rate have high weight.

Boosting in a Picture



Boosting algorithm for regression trees

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   - 2.1 Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).
   - 2.2 Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$

   - 2.3 Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$

3. Output the boosted model,

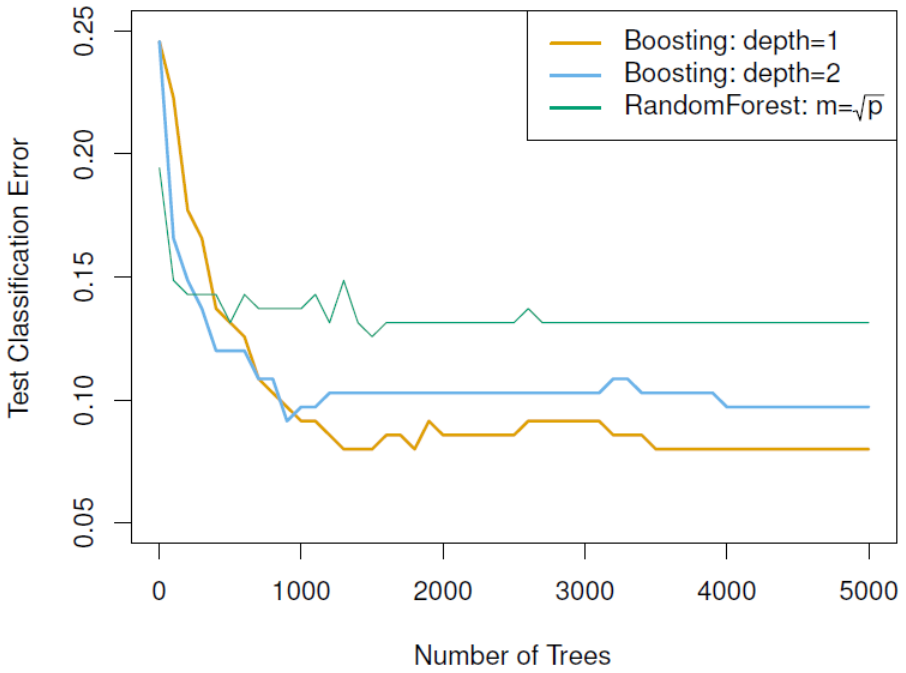
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

Tuning parameters for boosting

1. The number of trees B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
2. The shrinkage parameter λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.
3. The number of splits d in each tree, which controls the complexity of the boosted ensemble. Often d = 1 works well, in which case each tree is a stump, consisting of a single split and resulting in an additive model. More generally d is the interaction depth, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

Gene expression data

- Results from performing boosting and random forests on the fifteen-class gene expression data set in order to predict cancer versus normal.
- The test error is displayed as a function of the number of trees. For the two boosted models, $\lambda = 0.01$. Depth-1 trees slightly outperform depth-2 trees, and both outperform the random forest, although the standard errors are around 0.02, making none of these differences significant.



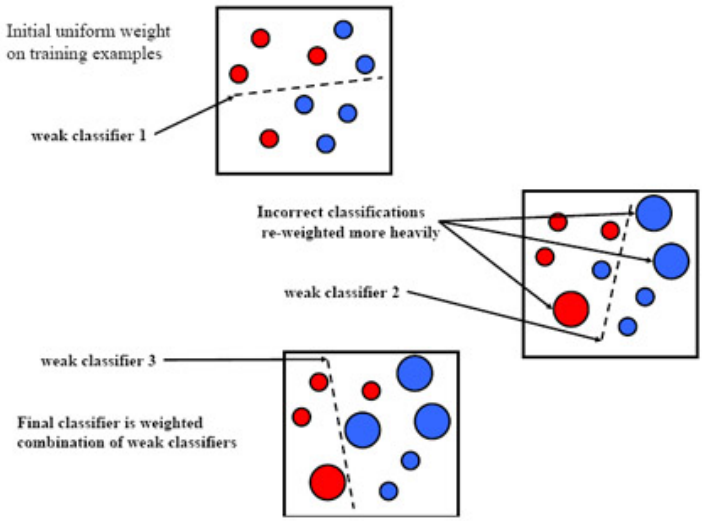
Boosting for classification

- **Muestreo ponderado (ejemplos):**
  - En lugar de hacer un muestreo aleatorio de los datos de entrenamiento, se ponderan las muestras para concentrar el aprendizaje en los ejemplos más difíciles.

- **Votos ponderados (clasificadores):**
  - En lugar de combinar los clasificadores con el mismo peso en el voto, se usa un voto ponderado.

Boosting for classification

AdaBoost, abreviatura de "Adaptive Boosting", es un algoritmo de aprendizaje formulado por Yoav Freund y Robert Schapire que ganó el prestigioso "Premio Gödel" en 2003 por su trabajo. Se puede utilizar en conjunción con muchos otros tipos de algoritmos de aprendizaje para mejorar su rendimiento.



$$H(x) = sign(\alpha_1 h_1(x) + \alpha_2 h_2(x) + \alpha_3 h_3(x))$$

Boosting for classification

AdaBoost. Adaptive Boosting [Freund,Schapire,96]

- Initialize distribution over training set $D_1(i) = 1/N$.
- For $t = 1, \ldots, T$

1. Train weak learner using distribution D_t and obtain h_t.
2. Choose a weight (confidence value) $\alpha_t \in R$.
3. Update distribution over training set:

$$D_{t+1}(i) = \frac{D_t(i)e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

- Set $H(x) = sign(f(x)) = sign\left(\sum_{i=1}^{T} \alpha_t h_t(x)\right)$

Explaining underlying mathematics

- How do we assign weight to observations?
- We always start with a uniform distribution assumption. Lets call it as D_1 which is 1/n for all n observations.
- Step 1 . We assume an alpha(t)
- Step 2: Get a weak classifier h(t)
- Step 3: Update the population distribution for the next step

where

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x}{Z_t}$$

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot A$$

$$Z_t = \sum_{i=1}^{m} D_t(i)\exp(-\alpha_t y_i h_t(x_i))$$

$$if \quad h_t(\mathbf{x}_i) = y_i \Longrightarrow A = e^{-\alpha_t}$$
$$if \quad h_t(\mathbf{x}_i) \neq y_i \Longrightarrow A = e^{\alpha_t}$$

Explaining underlying mathematics

- Simply look at the argument in exponent. Alpha is kind of learning rate, y is the actual response (+ 1 or -1) and h(x) will be the class predicted by learner. Essentially, if learner is going wrong, the exponent becomes 1*alpha and else -1*alpha. The weight will probably increase, if the prediction went wrong the last time.
- Step 4 : Use the new population distribution to again find the next learner
- Step 5 : Iterate step 1 – step 4 until no hypothesis is found which can improve further.
- Step 6 : Take a weighted average of the frontier using all the learners used till now. But what are the weights? Weights are simply the alpha values. Alpha is calculated as follows:

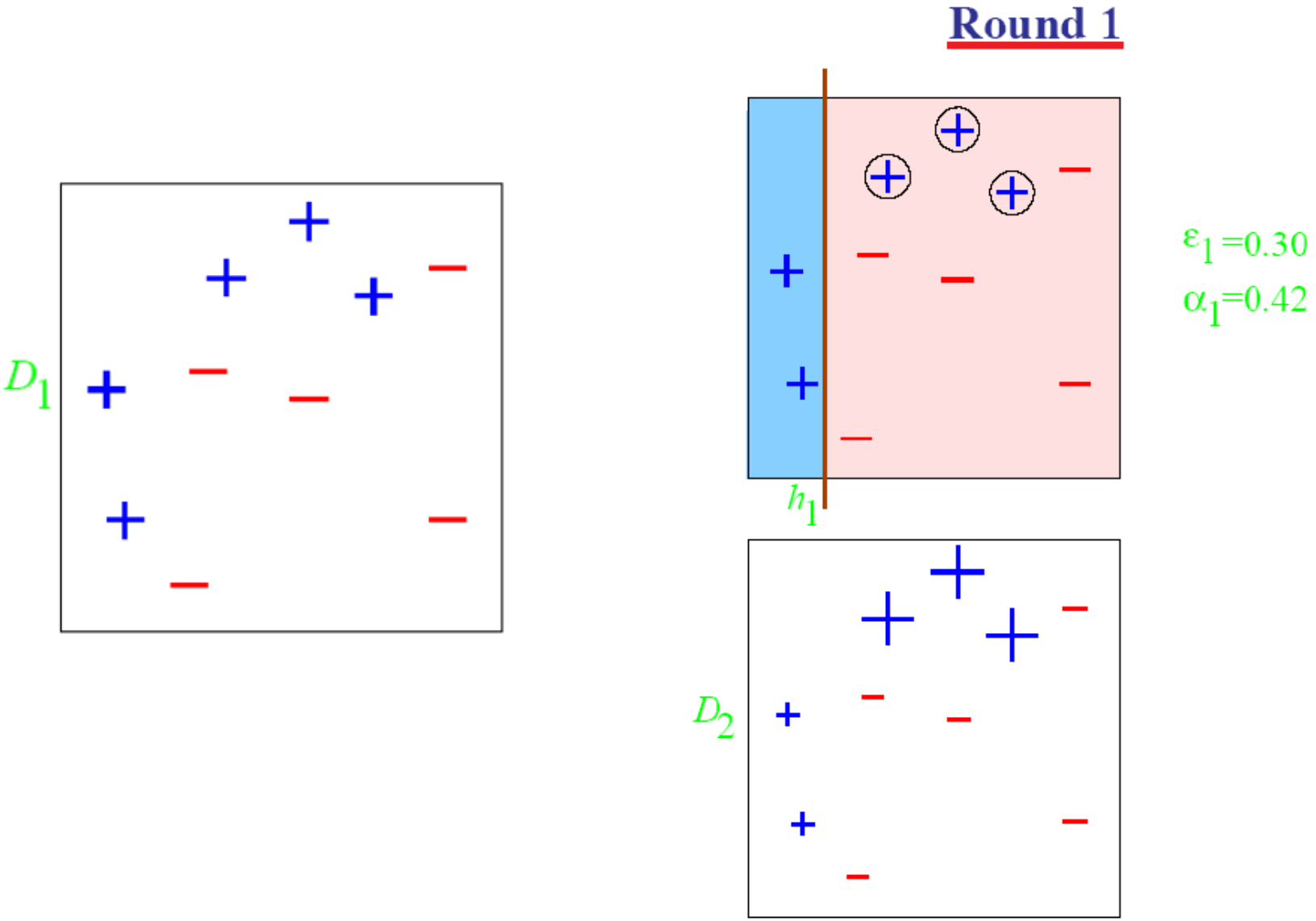
$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

Explaining underlying mathematics

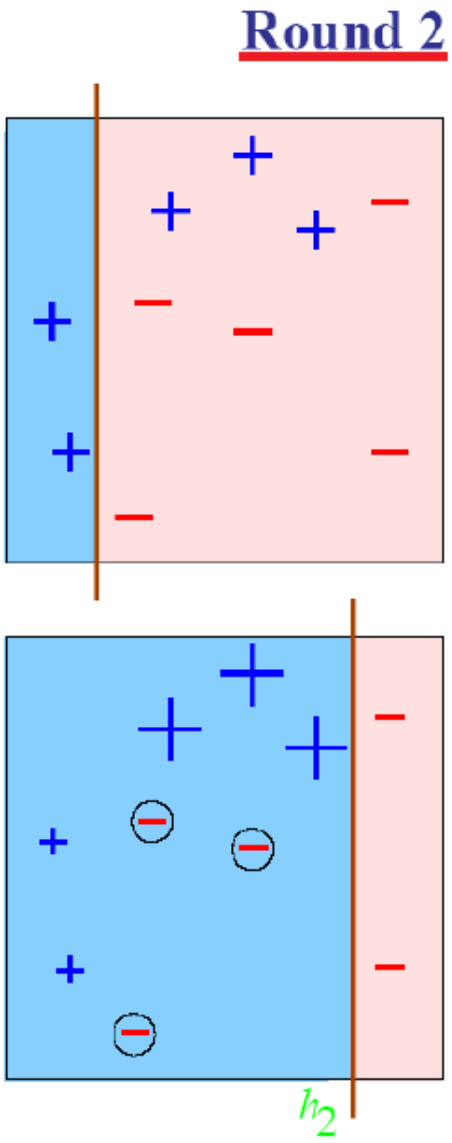
- Output the final hypothesis

$$H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) .$$

Boosting for classification

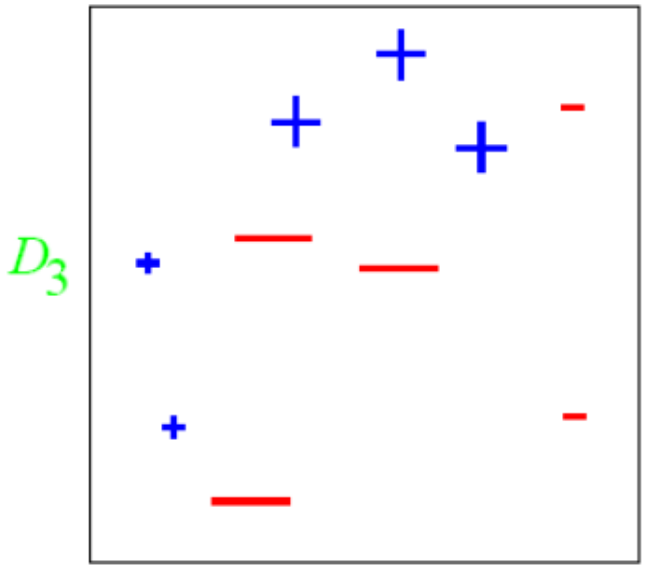


Boosting for classification

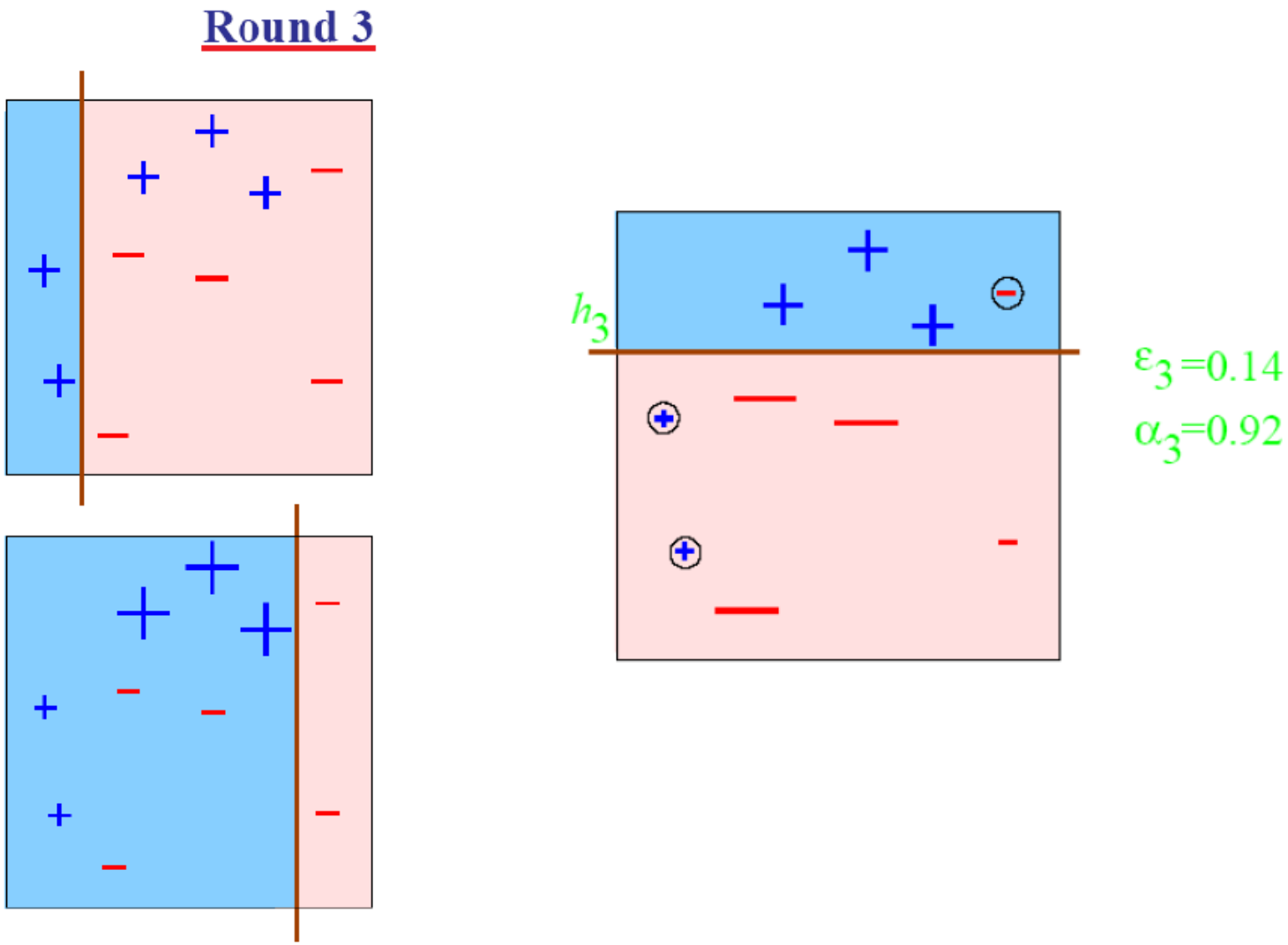


$\varepsilon_2 = 0.21$

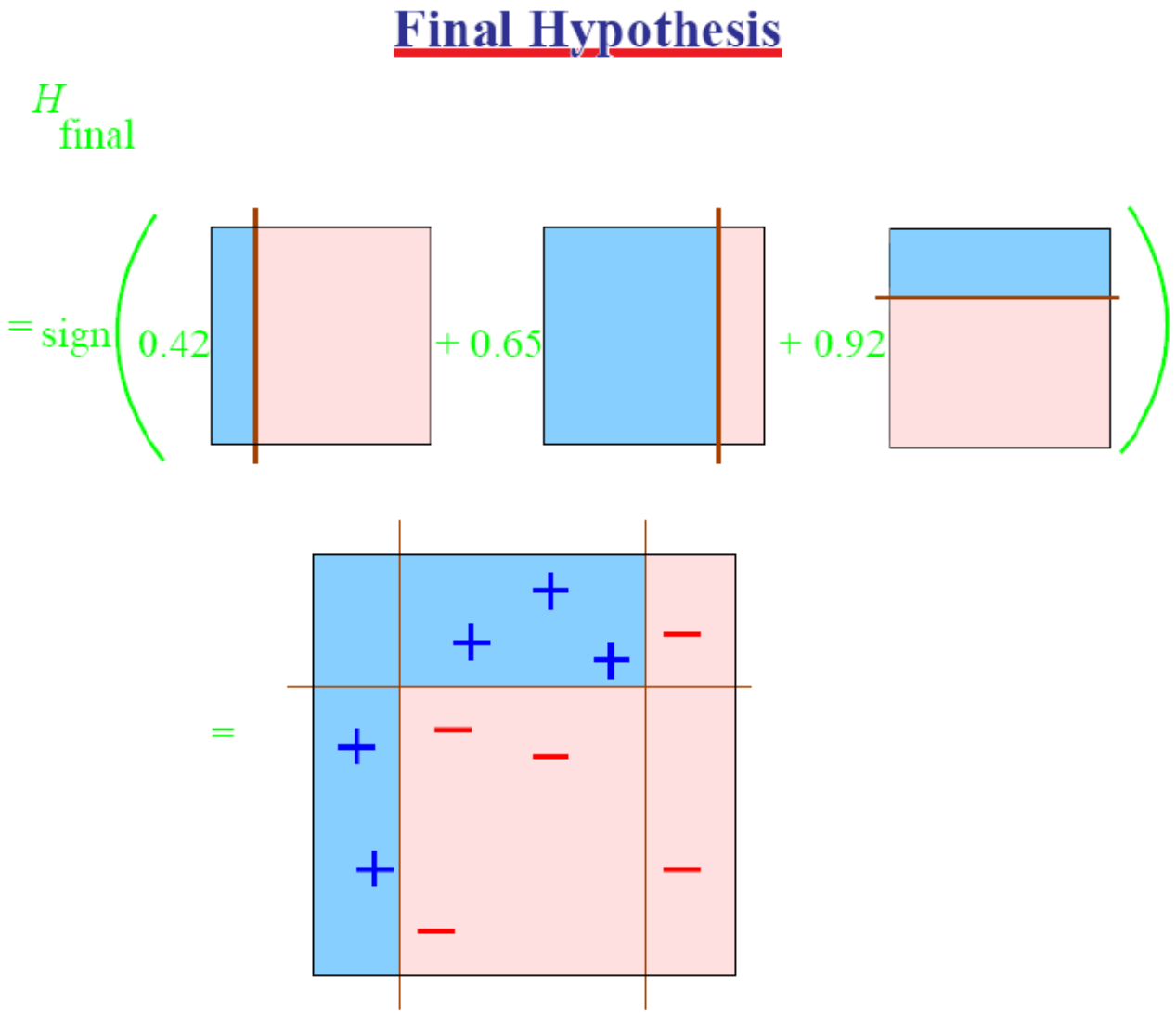
$\alpha_2 = 0.65$



Boosting for classification



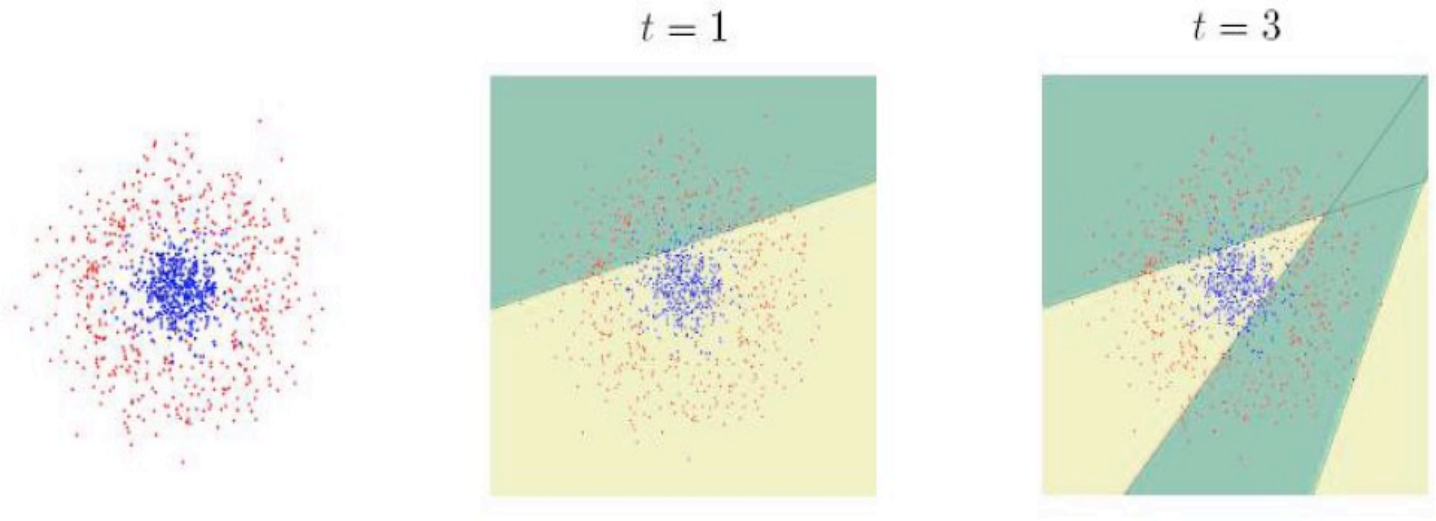
Boosting for classification



AdaBoost On a linear Classifier (e.g. Fisher)

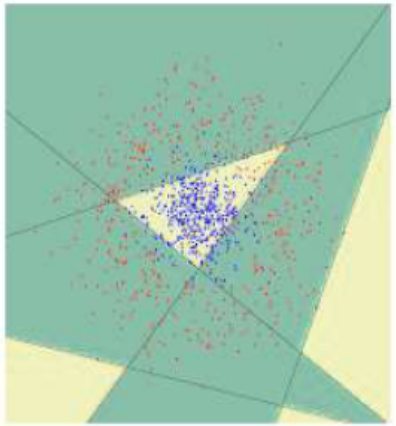
J. Sochman, J. Matas, cmp.felk.cvut.cz

Start with a problem for which a linear classifier is weak:

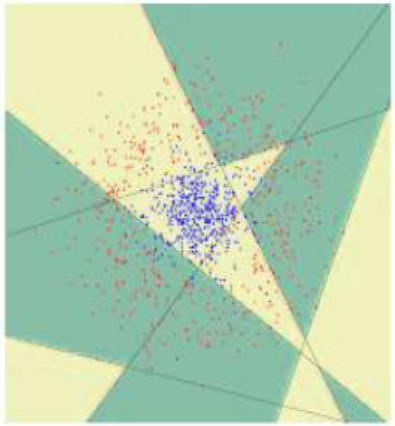


AdaBoost On a linear Classifier (e.g. Fisher)

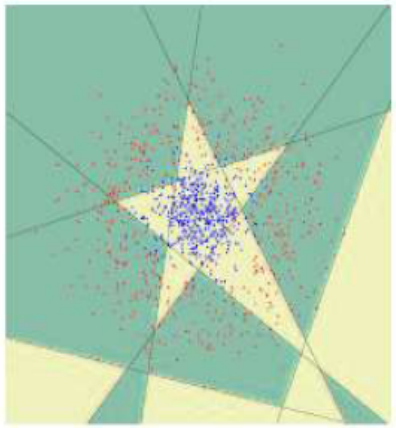
$t = 5$



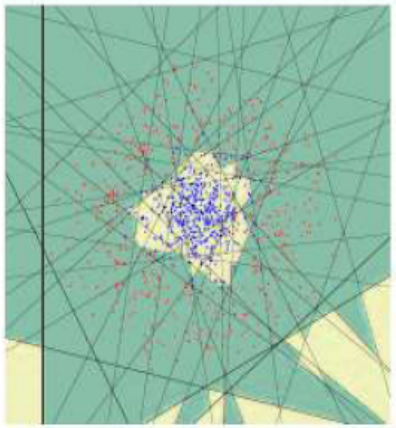
$t = 6$

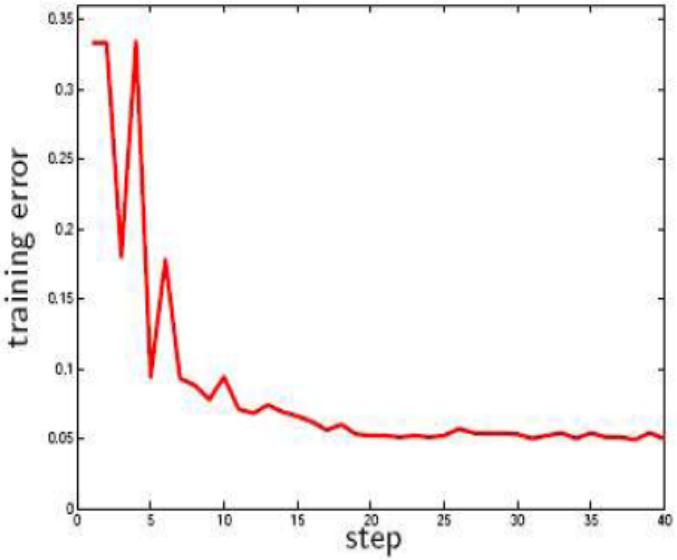


$t = 7$



$t = 40$

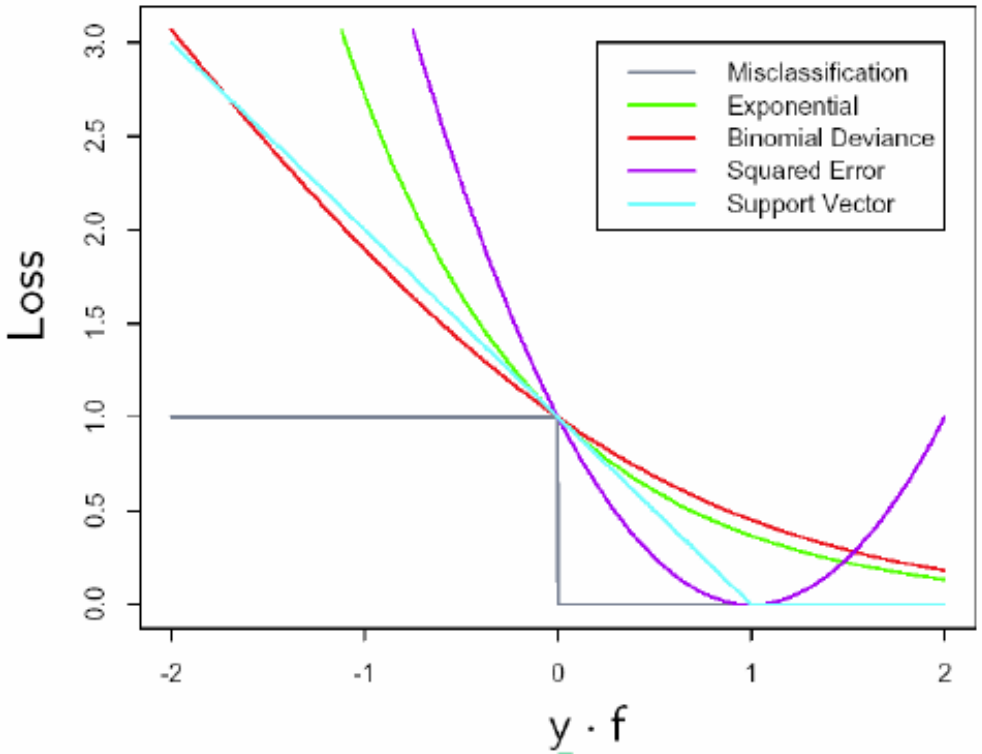




Types of Boosting Algorithms

- Underlying engine used for boosting algorithms can be anything. It can be decision stamp, margin-maximizing classification algorithm etc. There are many boosting algorithms which use other types of engine such as:
  - AdaBoost (**Ada**ptive **Boost**ing)
  - Gradient Tree Boosting
  - XGBoost
  - CatBoost
  - Lightgbm

Loss Functions for $y \in \{-1, +1\}, f \in \Re$



Incorrect Classification | **Correct Classification**

- Misclassification

$$I\left(\operatorname{sgn}(f) \neq y\right)$$

- Exponential (Boosting)

$$\exp(-yf)$$

- Binomial Deviance
(Cross Entropy)

$$\log\left(1+\exp(-2yf)\right)$$

- Squared Error

$$(y-f)^2$$

- Support Vectors

$$(1-yf)\bullet I(yf>1)$$

Gradient Boosting

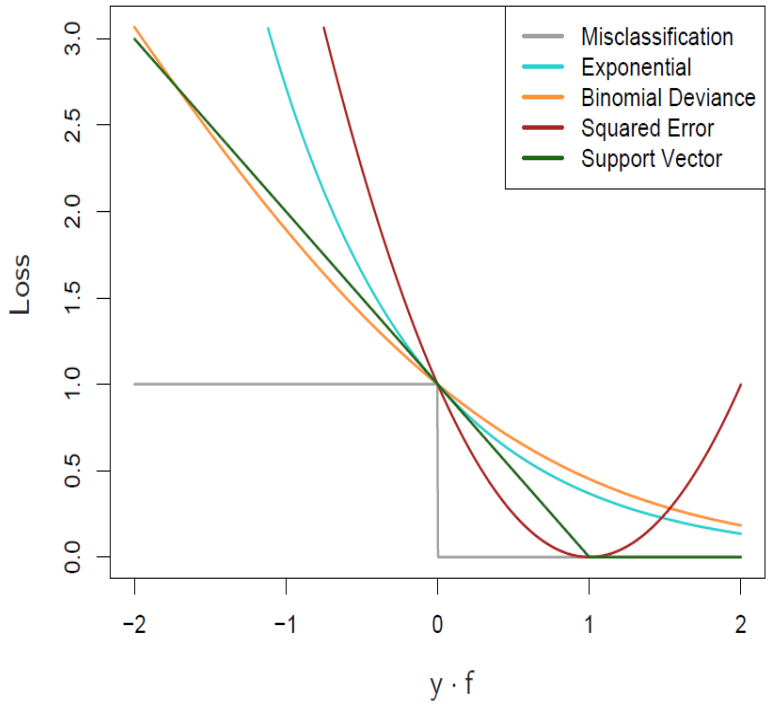
- In gradient boosting, it trains many models sequentially. Each new model gradually minimizes the loss function (y = ax + b + e, e needs special attention as it is an error term) of the whole system using Gradient Descent method.
- The learning procedure consecutively fit new models to provide a more accurate estimate of the response variable.
- The principle idea behind this algorithm is to construct new base learners which can be maximally correlated with negative gradient of the loss function, associated with the whole ensemble.

Gradient Boost

Gradient Boost is a way to implement “boosting” with arbitrary “loss functions” by approximating “somehow” the gradient of the loss function

AdaBoost: Exponential loss $\exp(-y_0 y(\alpha,x))$ → theoretically sensitive to outliers

Binomial log-likelihood loss $\ln(1 + \exp(-2y_0 y(\alpha,x))$ → more well behaved loss function.



Gradient Boosting

- Type of Problem – You have a set of variables vectors x1 , x2 and x3. You need to predict y which is a continuous variable.

- **Steps of Gradient Boost algorithm**

  Step 1 : Assume mean is the prediction of all variables.

  Step 2 : Calculate errors of each observation from the mean (latest prediction).

  Step 3 : Find the variable that can split the errors perfectly and find the value for the split. This is assumed to be the latest prediction.

  Step 4 : Calculate errors of each observation from the mean of both the sides of split (latest prediction).

  Step 5 : Repeat the step 3 and 4 till the objective function maximizes/minimizes.

  Step 6 : Take a weighted mean of all the classifiers to come up with the final model.

Example

- Assume, you are given a previous model M to improve on. Currently you observe that the model has an accuracy of 80% (any metric). How do you go further about it?

- One simple way is to build an entirely different model using new set of input variables and trying better ensemble learners. On the contrary, I have a much simpler way to suggest. It goes like this:

$$Y = M(x) + \text{error}$$

- What if I am able to see that error is not a white noise but have same correlation with outcome(Y) value. What if we can develop a model on this error term? Like,

$$\text{error} = G(x) + \text{error2}$$

Example

- Probably, you'll see error rate will improve to a higher number, say 84%. Let's take another step and regress against error2.

$$error2 = H(x) + error3$$

- Now we combine all these together :

$$Y = M(x) + G(x) + H(x) + error3$$

- This probably will have an accuracy of even more than 84%. What if I can find an optimal weights for each of the three learners,

$$Y = alpha * M(x) + beta * G(x) + gamma * H(x) + error4$$

Example

- If we found good weights, we probably have made even a better model. This is the underlying principle of a boosting learner.
- Boosting is generally done on weak learners, which do not have a capacity to leave behind white noise.
- Boosting can lead to overfitting, so we need to stop at the right point.

How to improve regression results

- You are given (x_1, y_1), (x_2, y_2), …, (x_n, y_n), and the task is to t a model F(x) to minimize square loss.
- Suppose your friend wants to help you and gives you a model F.
- You check his model and find the model is good but not perfect. There are some mistakes: $F(x_1) = 0.8$, while $y_1 = 0.9$, and $F(x_2) = 1.4$ while $y_2 = 1.3$. How can you improve this model?
- **Rule of the game:**

  You are not allowed to remove anything from F or change any parameter in F.

  You can add an additional model (regression tree) h to F, so the new prediction will be F(x) + h(x).

- Simple solution:
- You wish to improve the model such that

$$F(x_1) + h(x_1) = y_1$$
$$F(x_2) + h(x_2) = y_2$$
$$\vdots$$
$$F(x_n) + h(x_n) = y_n$$

Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$
$$h(x_2) = y_2 - F(x_2)$$
$$\vdots$$
$$h(x_n) = y_n - F(x_n)$$

- Can any regression tree h achieve this goal perfectly?

- Maybe not....
- But some regression tree might be able to do this approximately.
- How?
- Just fit a regression tree h to data

$$(x_1, y_1 - F(x_1)), (x_2, y_2 - F(x_2)), \ldots, (x_n, y_n - F(x_n))$$

- Congratulations, you get a better model!
- $y_i - F(x_i)$ are residuals. These are the parts that existing model F cannot do well.
- The role of h is to compensate the shortcoming of existing model F. If the new model F + h is still not satisfactory, we can add another regression tree...
- We are improving the predictions of training data, but is the procedure also useful for test data?
- Yes! Because we are building a model, and the model can be applied to test data as well.
- How is this related to gradient descent?

XGBoosting (Extreme Gradient Boosting)

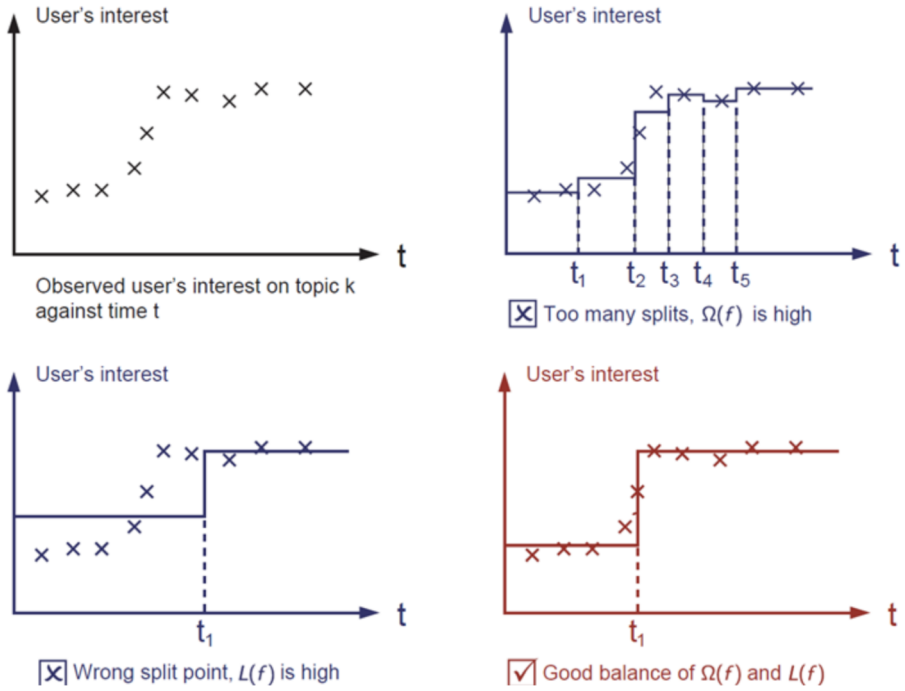
- Ever since its introduction in 2014, XGBoost has been lauded as the holy grail of machine learning hackathons and competitions. From predicting ad click-through rates to classifying high energy physics events, XGBoost has proved its mettle in terms of performance – and speed.
- Execution Speed: Generally, XGBoost is fast. Really fast when compared to other implementations of gradient boosting. But newly introduced LightGBM is faster than XGBoosting.
- Model Performance: XGBoost dominates structured or tabular datasets on classification and regression predictive modeling problems. The evidence is that it is the go-to algorithm for competition winners on the Kaggle competitive data science platform.

What Algorithm Does XGBoost Use?

- The XGBoost library implements the gradient boosting decision tree algorithm.
- This algorithm goes by lots of different names such as gradient boosting, multiple additive regression trees, stochastic gradient boosting or gradient boosting machines.
- Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.
- This approach supports both regression and classification predictive modeling problems.

XGBoosting (Extreme Gradient Boosting)

- **What is the difference between the R gbm (gradient boosting machine) and XGBoost (extreme gradient boosting)?**
- Both XGBoost and gbm follows the principle of gradient boosting. There are however, the difference in modeling details. Specifically, XGBoost used a more regularized model formalization to control over-fitting, which gives it better performance.
- Objective Function : Training Loss + Regularization
- The regularization term controls the complexity of the model, which helps us to avoid overfitting.



XGBoosting

- In the XGBoost package, at the t^{th} step we are tasked with finding the tree F_t that will minimize the following objective function:

$$Obj(F_t) = L(F_{t-1} + F_t) + \Omega(F_t)$$

where $L(F_t)$ is our loss function and $\Omega(F_t)$ is our regularization function.

- Regularization is essential to prevent overfitting to the training set. Without any regularization, the tree will split until it can predict the training set perfectly. This will usually mean that the tree has lost generality and will not do well on new test data. In XGBoost, the regularization function shows the model complexity.

$$\Omega(F_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

where T is the number of leaves in the tree, w_j is the score of leaf j, γ is the leaf weight penalty parameter, and λ is the tree size penalty parameter.

- Determining how to find the function to optimize the above objective function is not clear.

$$Obj(F_t) = -\frac{1}{2}\sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

$$G_j = \sum_{i \in I_j} \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

$$H_j = \sum_{i \in I_j} \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

$$I = \{i | q(x_i) = j\}$$

where q(x) maps input features to a leaf node in the tree and $l(y_i; \hat{y})$ is our loss function. This objective function is much easier to work with because it is now gives a score that we can use to determine how good a tree structure is.

Unique features of XGBoost

- XGBoost is a popular implementation of gradient boosting. Let's discuss some features of XGBoost that make it so interesting.
- **Regularization:** XGBoost has an option to penalize complex models through both L1 and L2 regularization. Regularization helps in preventing overfitting
- **Handling sparse data:** Missing values or data processing steps like one-hot encoding make data sparse. XGBoost incorporates a sparsity-aware split finding algorithm to handle different types of sparsity patterns in the data
- **Weighted quantile sketch:** Most existing tree based algorithms can find the split points when the data points are of equal weights (using quantile sketch algorithm). However, they are not equipped to handle weighted data. XGBoost has a distributed weighted quantile sketch algorithm to effectively handle weighted data

- Norm based regularizers are used as approximations to $R^{cnt}(\mathbf{w}, b)$
  - ℓ_2 squared norm: $||\mathbf{w}||_2^2 = \sum_{d=1}^{D} w_d^2$
  - ℓ_1 norm: $||\mathbf{w}||_1 = \sum_{d=1}^{D} |w_d|$
  - ℓ_p norm: $||\mathbf{w}||_p = (\sum_{d=1}^{D} w_d^p)^{1/p}$

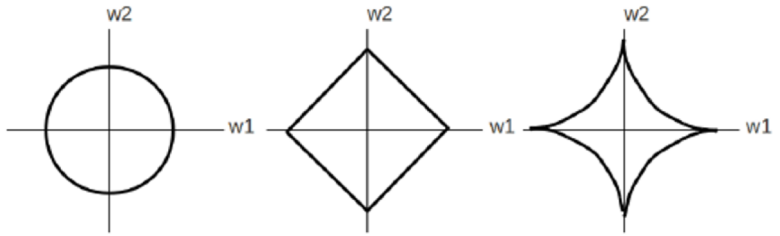


Figure: Contour plots. Left: ℓ_2 norm, Center: ℓ_1 norm, Right: ℓ_p norm (for $p < 1$)

Unique features of XGBoost

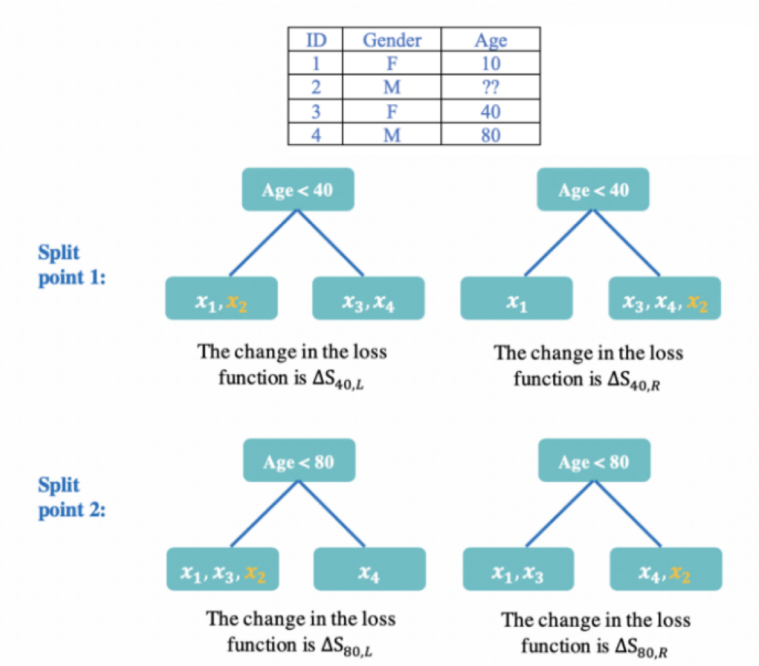
- **Block structure for parallel learning:** For faster computing, XGBoost can make use of multiple cores on the CPU. This is possible because of a block structure in its system design. Data is sorted and stored in in-memory units called blocks. Unlike other algorithms, this enables the data layout to be reused by subsequent iterations, instead of computing it again. This feature also serves useful for steps like split finding and column sub-sampling
- **Cache awareness:** In XGBoost, non-continuous memory access is required to get the gradient statistics by row index. Hence, XGBoost has been designed to make optimal use of hardware. This is done by allocating internal buffers in each thread, where the gradient statistics can be stored
- **Out-of-core computing:** This feature optimizes the available disk space and maximizes its usage when handling huge datasets that do not fit into memory

Parameters

- XGBoost requires a number of parameters to be selected. The following is a list of all the parameters that can be specified:
- **(eta) Shrinkage term.** Each new tree that is added has its weight shrunk by this parameter, preventing overfitting, but at the cost of increasing the number of rounds needed for convergence.
- **(gamma)** Tree size penalty
- **max depth** The maximum depth of each tree
- **min child weight** The minimum weight that a node can have. If this minimum is not met, that particular split will not occur.
- **subsample** Gives us the opportunity to perform "bagging," which means randomly sampling with replacement a proportion specified by parameter of the training examples to train each round on. The value is between 0 and 1 and is a method that helps prevent overfitting.
- **colsample bytree** Allows us to perform "feature bagging," which picks a proportion of the features to build each tree with. This is another way of preventing overfitting.
- **(lambda)** This is the L2 leaf node weight penalty

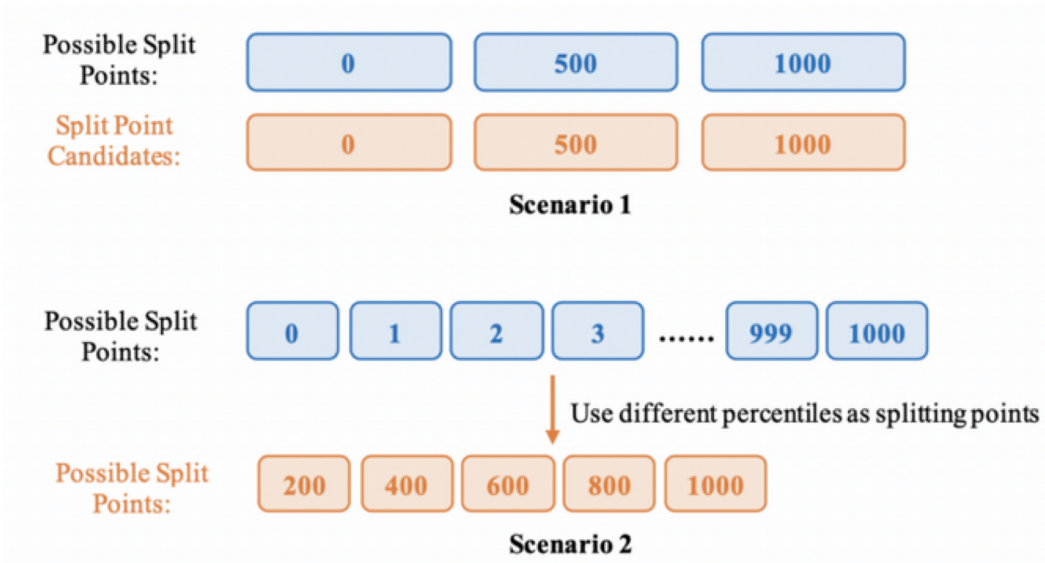
XGBoost

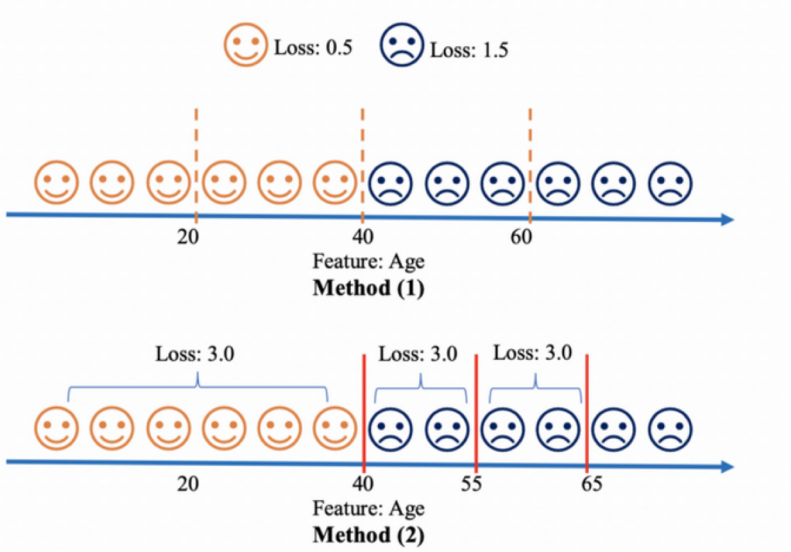
- **Summary**
  - KEEP missing values as they are. When splitting a node, XGBoost compares the two scenarios where the missing values were put to right node and the left node; then selects the method which minimizes the loss function.
  - CANNOT handle categorical features. Need to convert categorical features to numeric, otherwise there will be errors.
  - RELATIVELY efficient, as XGBoost uses a less intensive way to find splitting points for nodes, and also parallel computation.
- **Missing Values**



XGBoost

- **Split Finding Algorithm**
  - When there are a limited number of split point candidates, XGBoost uses Extract Greedy Algorithm to find optimal split points. It enumerates over all the possible splits (values) of all the features.
  - However, when there are numerous split point candidates, the algorithm above is too costly in computation and requires large memory space. In this scenario, XGBoost uses *Weighted Quantile Sketch in Approximate Algorithm* to expedite the calculation.
  - We'd better set more splitting points in the place where the loss is high, and less splitting points where the loss is low. The split point is selected to make the similar sum of loss for the observations in different intervals.





CatBoost

- **Summary**
  - CANNOT handle missing values. Need to impute the missing values.
  - TREAT categorical variables well, with many methods available: e.g. One-Hot-Encode (much faster in CATBOOST), target statistics, frequency statistics as well as different combinations of features as splitting points.
  - VERY efficient, as CatBoost is more efficient in dealing with categorical variables besides the advantages of XGBoost.

- **Categorical Features**
  - One-hot encoding: Appropriate for low-cardinality categorical feature. Drawback: high-cardinality categorical features use very large memory space, and cause insufficient training examples.
  - Target statistics: use target values to replace each level of categorical feature. Appropriate for categorical features of any cardinality. Drawback: target leakage (will be discussed below)

CatBoost

- **Target Statistics (TS)**
  - The basic idea of TS is to use the expected target value of each category to replace the categorical feature. That is, computationally, for any i-th observation, its value of the categorical feature k will be replaced with the average target value of all the observations with the same categorical value as its:

$$\hat{x}_{i,k} = \frac{\sum_{j=1}^{n} 1(x_{j,k} = x_{i,k}) * y_j}{\sum_{j=1}^{n} 1(x_{j,k} = x_{i,k})},$$

where:

- $\hat{x}_{i,k}$ is the new value of the categorical feature k for the i-th observation.
- n is the number of observations (in the training set).
- $x_{i,k}$: the original value of the feature k for the i-th observation.
- y_i: the target value of the i-th observation.

  - **what if the denominator is zero?**

CatBoost

- **Greedy TS**

$$\hat{x}_{i,k} = \frac{\sum_{j=1}^{n} 1(x_{j,k} = x_{i,k}) * y_j + a * P}{\sum_{j=1}^{n} 1(x_{j,k} = x_{i,k}) + a} \qquad (1)$$

where:

- a is the weight on the prior target value P
- P could be the average target value under the prior probability distribution
- n is the number of observations

However, **Greedy TS** still has a problem: target leakage. For example, if every observation has a unique value $x_{i,k}$ of feature k. Then, the new value of feature k in the training set $\hat{x}_{i,k} = \frac{y_i + a*P}{1+a}$ fully reveals of the target value the value, which perfectly classifies all the training examples. However, the performance on the test set will be worst, as $\hat{x}_{i,k} = P$ for all observations. (Attention, n is the number of observations in the training set, thus $\sum_{j=1}^{n} 1(x_{j,k} = x_{i,k}) = 0$ for all observations in the test set under this scenario.)

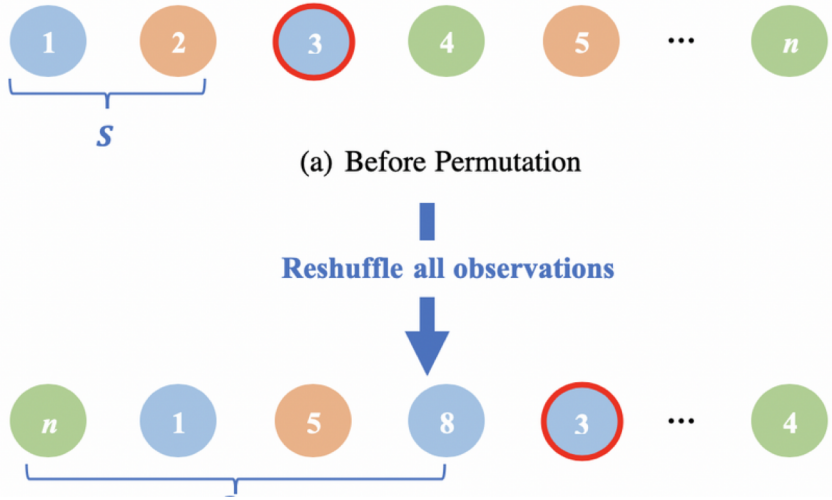
CatBoost

- **Ordered TS**

$$\hat{x}_{i,k} = \frac{\sum_{j<i} 1(x_{j,k} = x_{i,k}) * y_j + a * P}{\sum_{j<i} 1(x_{j,k} = x_{i,k}) + a}$$

  - But if we simply use this Equation to replace the categorical value, there is one problem: the proceeding observations have their TS with much larger variances than the subsequent ones. Ordered TS has made an improvement by using permutations, as shown:

This figure shows an example of using Ordered TS to replace the categorical value of 3rd observation (the blue ball with the red circle). In the normal way to calculate the new value for 3rd observation, we only look at the two observations (set S) in front of it and apply the equation. Ordered TS uses the same logics but after reshuffle all the observations. Then, to calculate the new value for 3rd observation, we will look at the four observations

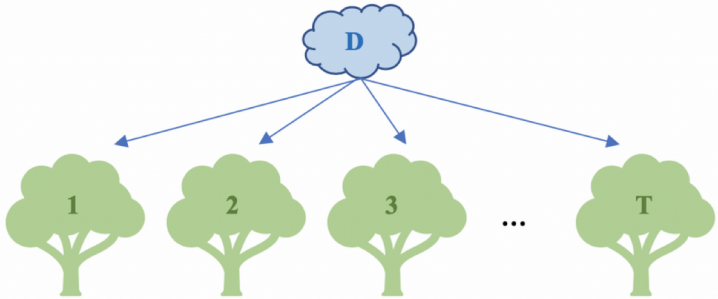


CatBoost

- **Prediction Shift**
  - In Boosting, if fitting many models on the same training dataset which will be used in prediction, it will cause a biased estimation, i.e. Prediction Shift.
  - In the paper of CatBoost, it gives a concrete example to show Prediction Shift (Read the original paper for more details):

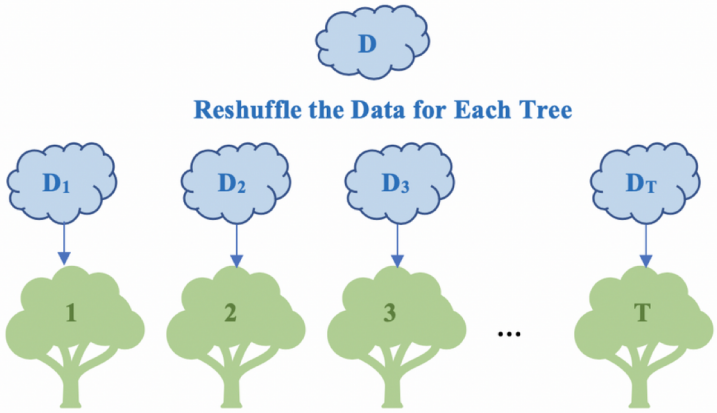
Models h^1 and h^2 are trained on two training datasets D_1 and D_2, where both D_1 and D_2 satisfy Bernoulli distribution (p=0.5), and the final prediction is $h^1 + h^2$.

- If D_1 and D_2 are independent, the bias in estimation is $O(1/2^n)$.
- If $D_1 = D_2$, the bias in estimation is $-(1/(n-1))\, c_2\, (x^2-1/2) + O(1/2^n)$.



(a) Normal Boosting

To solve the problem of Prediction Shift, CatBoost uses Ordered Boosting. The figure shows the idea of Order Boosting. Figure (a) is the normal way Boosting used to train models, which leads to Prediction Shift. Compared with that, Figure (b) shows Ordered Boosting: each model is trained on a reshuffled dataset. Within each tree, there will be n different supporting models M_1, M_2, …, M_n where each supporting model Mi is trained only on first (i-1) observations. To compute the residual for observation i, we only use model M_{i-1}.



(b) Ordered Boosting

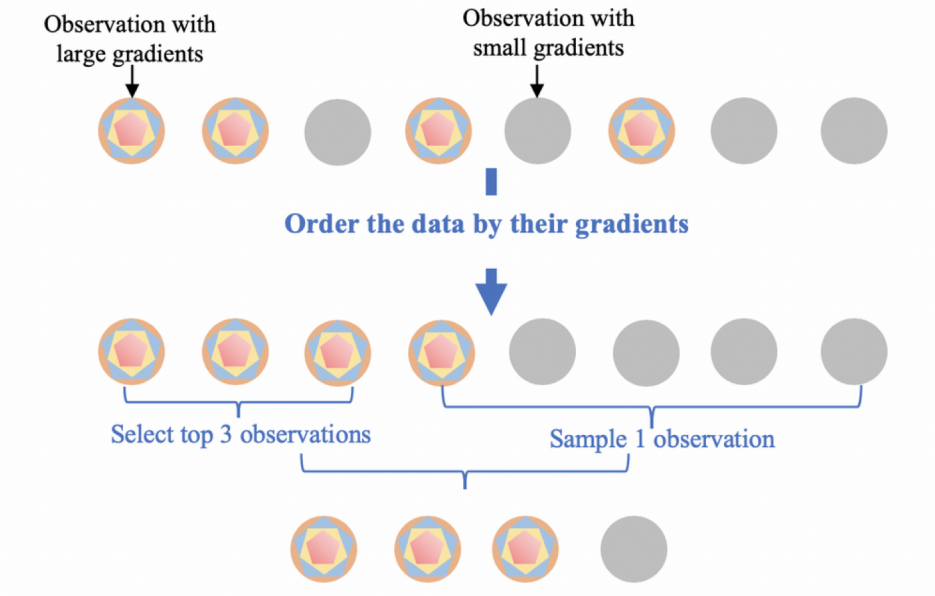
LightGBM

- **Summary**
  - 1. KEEP missing values as they are. Treat them in the same way as XGBoost.
  - 2. CAN treat categorical variables, but only in two ways one-hot-encoding or label-encoding.
  - 3. MOST efficient, as LightGBM reduces the sample size and the number of features in training by Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB), and also saves time by using leaf-wise growth strategy (we won't cover it in this lecture).

- **Reduce Data Size (Gradient-based One-Side Sampling, GOSS)**
  - The fundamental reason why it is possible to reduce data size is that not all observations are equally important in training.
  - The idea of Gradient-based One-Side Sampling (GOSS) is that the observations with the larger gradients (i.e. residuals) play an more important role in training.
  - Thus, GOSS down samples the data by looking at their gradients. However, if simply discard the observations with small gradients, it will alter the data distribution and hurt the accuracy of the training model.
  - To avoid that, besides select a number of observations with the large gradients, GOSS samples the observations with the small gradients, and then amplify their information gain with different factors.

LightGBM

- **Reduce Data Size (Gradient-based One-Side Sampling, GOSS)**



Step 1: Compute the gradients (i.e. residuals) of all the observations.

Step 2: Sort the observations according to their gradients.

Step 3: Select top *a**100% observations.

Step 4: Randomly sample *b* * 100% observations from the rest of data.

Step 5: when calculate the information gain, amplify the randomly selected data (*i.e.* in *Step 4*) by (1-*a*)/*b.*

LightGBM

- **Reduce Feature Numbers (Exclusive Feature Bundling, EFB)**

- The idea of Exclusive Feature Bundling (EFB) is straightforward.
- The big data are usually high-dimensional and very sparse thus many features could be almost mutually exclusive.
- We can bundle these exclusive features together as “one feature” in splitting nodes, by allowing some level of conflicts.
- It uses the algorithm related to graph to reduce the complexity. We won’t discuss it in this lecture.

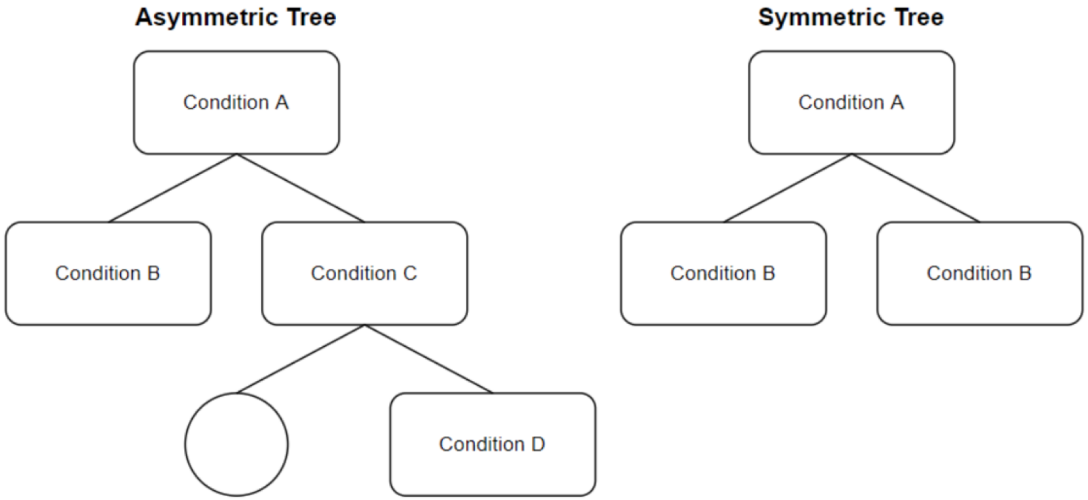
CatBoost / LightGBM / XGBoost

- Main properties

	CatBoost	LightGBM	XGBoost
Developer	Yandex	Microsoft	DMLC
Release Year	2017	2016	2014
Tree Symmetry	Symmetric	Asymmetric Leaf-wise tree growth	Asymmetric Level-wise tree growth
Splitting Method	Greedy method	Gradient-based One-Side Sampling (GOSS)	Pre-sorted and histogram-based algorithm
Type of Boosting	Ordered	-	-
Numerical Columns	Support	Support	Support
Categorical Columns	Support Perform one-hot encoding (default) Transforming categorical to numerical columns by border, bucket, binarized target mean value, counter methods available	Support, but must use numerical columns Can interpret ordinal category	Supports, but must use numerical columns Cannot interpret ordinal category, users must convert to one-hot encoding, label encoding or mean encoding
Text Columns	Support Support Bag-of-Words, Naïve-Bayes or BM-25 to calculate numerical features from text data	Do not support	Do not support
Missing values	Handle missing value Interpret as NaN (default) Possible to interpret as error, or processed as minimum or maximum values	Handle missing value Interpret as NaN (default) or zero Assign missing values to side that reduces loss the most in each split	Handle missing value Interpret as NaN (tree booster) or zero (linear booster) Assign missing values to side that reduces loss the most in each split

CatBoost / LightGBM / XGBoost

- **Simmetry**
  - In CatBoost, symmetric trees, or balanced trees, refer to the splitting condition being consistent across all nodes at the same depth of the tree. LightGBM and XGBoost, on the other hand, results in asymmetric trees, meaning splitting condition for each node across the same depth can differ.
  - For symmetric trees, this means that the splitting condition must result in the lowest loss across all nodes of the same depth. Benefits of balanced tree architecture include faster computation and evaluation and control overfitting.
  - Even though LightGBM and XGBoost are both asymmetric trees, LightGBM grows leaf-wise (horizontally) while XGBoost grows level-wise (vertically). To put it simply, we can think of LightGBM as growing the tree selectively, resulting in smaller and faster models compared to XGBoost.



CatBoost / LightGBM / XGBoost

- **Splitting Method**
  - In CatBoost, a greedy method is used such that a list of possible candidates of feature-split pairs are assigned to the leaf as the split and the split that results in the smallest penalty is selected.
  - In LightGBM, Gradient-based One-Side Sampling (GOSS) keeps all data instances with large gradients and performs random sampling for data instances with small gradients. Gradient refers to the slope of the tangent of the loss function. Data points with larger gradients have higher errors and would be important for finding the optimal split point, while data points with smaller gradients have smaller errors and would be important for keeping accuracy for learned decision trees. This sampling technique results in lesser data instances to train the model and hence faster training time.
  - In XGBoost, the pre-sorted algorithm considers all feature and sorts them by feature value. After which, a linear scan is done to decide the best split for the feature and feature value that results in the most information gain. The histogram-based algorithm works the same way but instead of considering all feature values, it groups feature values into discrete bins and finds the split point based on the discrete bins instead, which is more efficient than the pre-sorted algorithm although still slower than GOSS.
- **Categorical Attributes**
  - In CatBoost, a greedy method is used such that a list of possible candidates of feature-split pairs are assigned to the leaf as the split and the split that results in the smallest penalty is selected.
    - CatBoost: cat_features, one_hot_max_size
    - LightGBM: categorical_feature
    - XGBoost: NA

CatBoost / LightGBM / XGBoost

- **Parameters**

	CatBoost	LightGBM	XGBoost
Parameters to tune	*iterations* : number of trees *depth* : depth of tree *min_data_in_leaf* : control depth of tree	*num_leaves* : value should be less than 2^max_depth *min_data_in_leaf* : control depth of tree *max_depth* : depth of tree	*n_estimators* : number of trees *max_depth* : depth of tree *min_child_weight* : control depth of tree
Parameters for better accuracy		*max_bin* : maximum number of bins feature values will be bucketed in *num_leaves* Use bigger training data	
Parameters for faster speed	*subsample* : fraction of number of instances used in a tree *rsm* : random subspace method; fraction of number of features used in a split selection *iterations* *sampling_frequency* : frequency to sample weights and objects when building trees	*feature_fraction* : fraction of number of features used in a tree *bagging_fraction* : fraction of number of instances used in a tree *bagging_freq* : frequency for bagging *max_bin* *save_binary* : indicator to save dataset to binary file Use parallel learning	*colsample_bytree* : fraction of number of features used in a tree *subsample* : fraction of number of instances used in a tree *n_estimators*
Parameters to control overfitting	*early_stopping_rounds* : stop training after specified number of iterations since iteration with optimal metric value *od_type* : type of overfitting detector *learning_rate* : learning rate for reducing gradient step *depth* *l2_leaf_reg* : regularization parameter	*max_bin* *num_leaves* *max_depth* *bagging_fraction* *bagging_freq* *feature_fraction* *lambda_l1 / lambda_l2 / min_gain_to_split* Use bigger training data Regularization	*learning_rate* *gamma* : regularization parameter, higher gamma for more regularization *max_depth* *min_child_weight* *subsample*

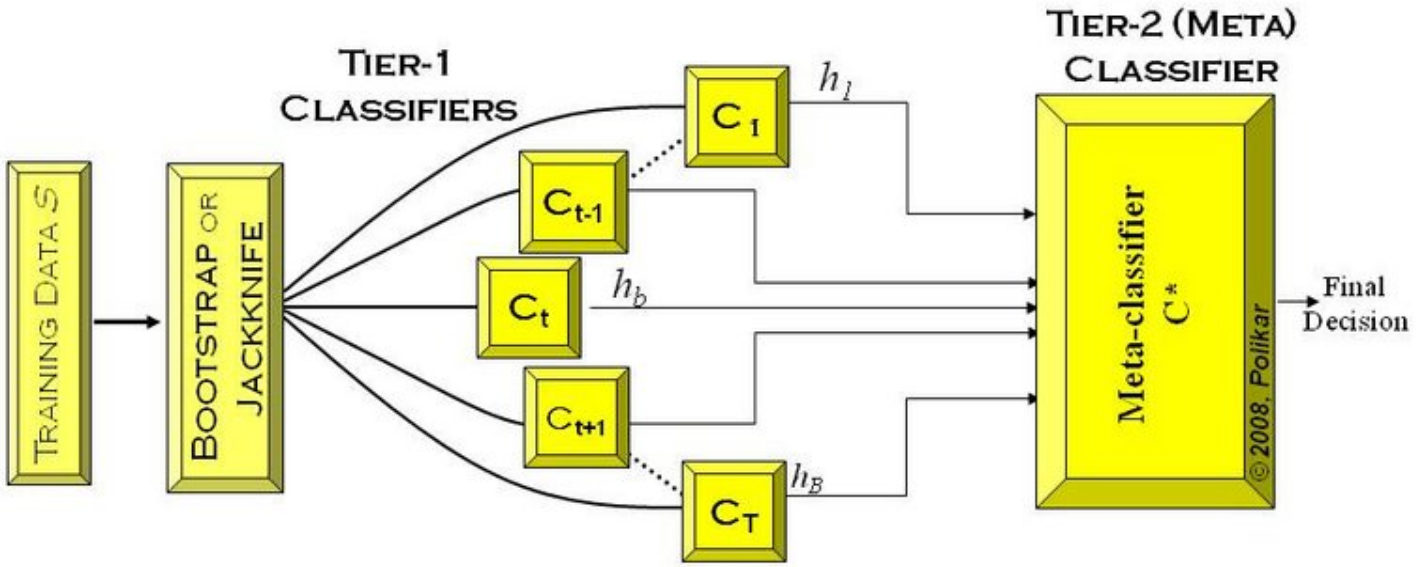
Stacking

- It involves combining the predictions from multiple machine learning models on the same dataset, like bagging and boosting.
- Stacking addresses the question: Given multiple machine learning models that are skillful on a problem, but in different ways, how do you choose which model to use (trust)?
- The approach to this question is to use another machine learning model that learns when to use or trust each model in the ensemble.
- Unlike bagging, in stacking, the models are typically different (e.g. not all decision trees) and fit on the same dataset (e.g. instead of samples of the training dataset).
- Unlike boosting, in stacking, a single model is used to learn how to best combine the predictions from the contributing models (e.g. instead of a sequence of models that correct the predictions of prior models).
- The architecture of a stacking model involves two or more base models, often referred to as level-0 models, and a meta-model that combines the predictions of the base models, referred to as a level-1 model.
- Level-0 Models (Base-Models): Models fit on the training data and whose predictions are compiled.
- Level-1 Model (Meta-Model): Model that learns how to best combine the predictions of the base models.

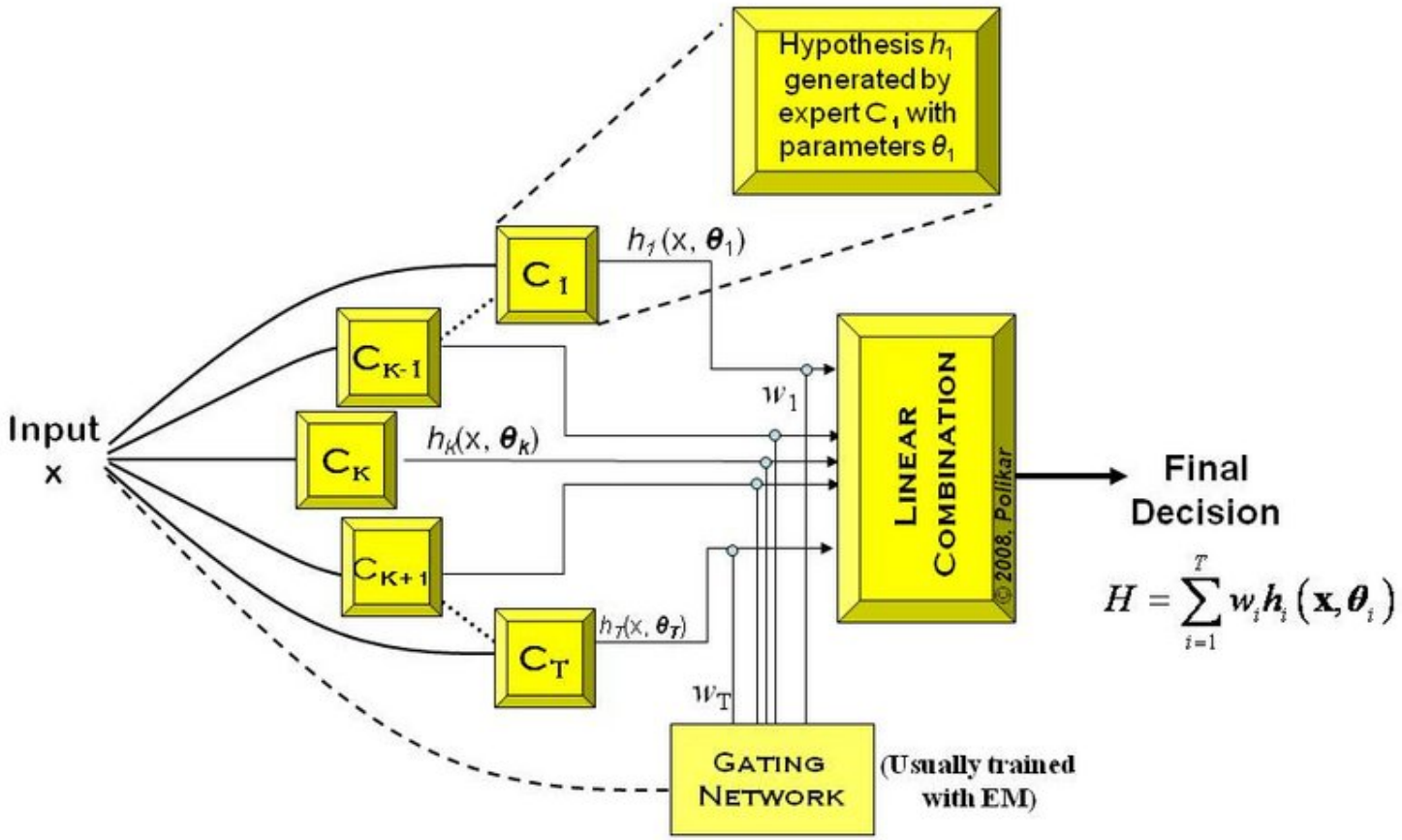
Stacking

- The meta-model is trained on the predictions made by base models on out-of-sample data. That is, data not used to train the base models is fed to the base models, predictions are made, and these predictions, along with the expected outputs, provide the input and output pairs of the training dataset used to fit the meta-model.
- The outputs from the base models used as input to the meta-model may be real value in the case of regression, and probability values, probability like values, or class labels in the case of classification.
- The training data for the meta-model may also include the inputs to the base models, e.g. input elements of the training data. This can provide an additional context to the meta-model as to how to best combine the predictions from the meta-model.
- Stacking is appropriate when multiple different machine learning models have skill on a dataset, but have skill in different ways. Another way to say this is that the predictions made by the models or the errors in predictions made by the models are uncorrelated or have a low correlation.
- Base-models are often complex and diverse.
- The meta-model is often simple, providing a smooth interpretation of the predictions made by the base models.
- **Regression Meta-Model: Linear Regression.**
- **Classification Meta-Model: Logistic Regression.**

Stacking


TRAINING DATA S
BOOTSTRAP OR JACKKNIFE
TIER-1 CLASSIFIERS
C 1
C t-1
C t
C t+1
C T
h1
hb
hB
TIER-2 (META) CLASSIFIER
Meta-classifier C*
© 2008, Polikar
Final Decision

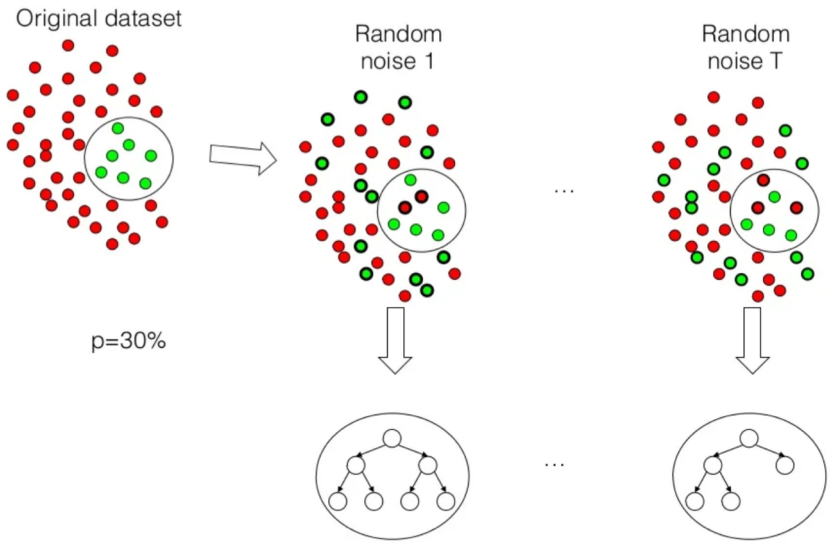
Mixture of experts

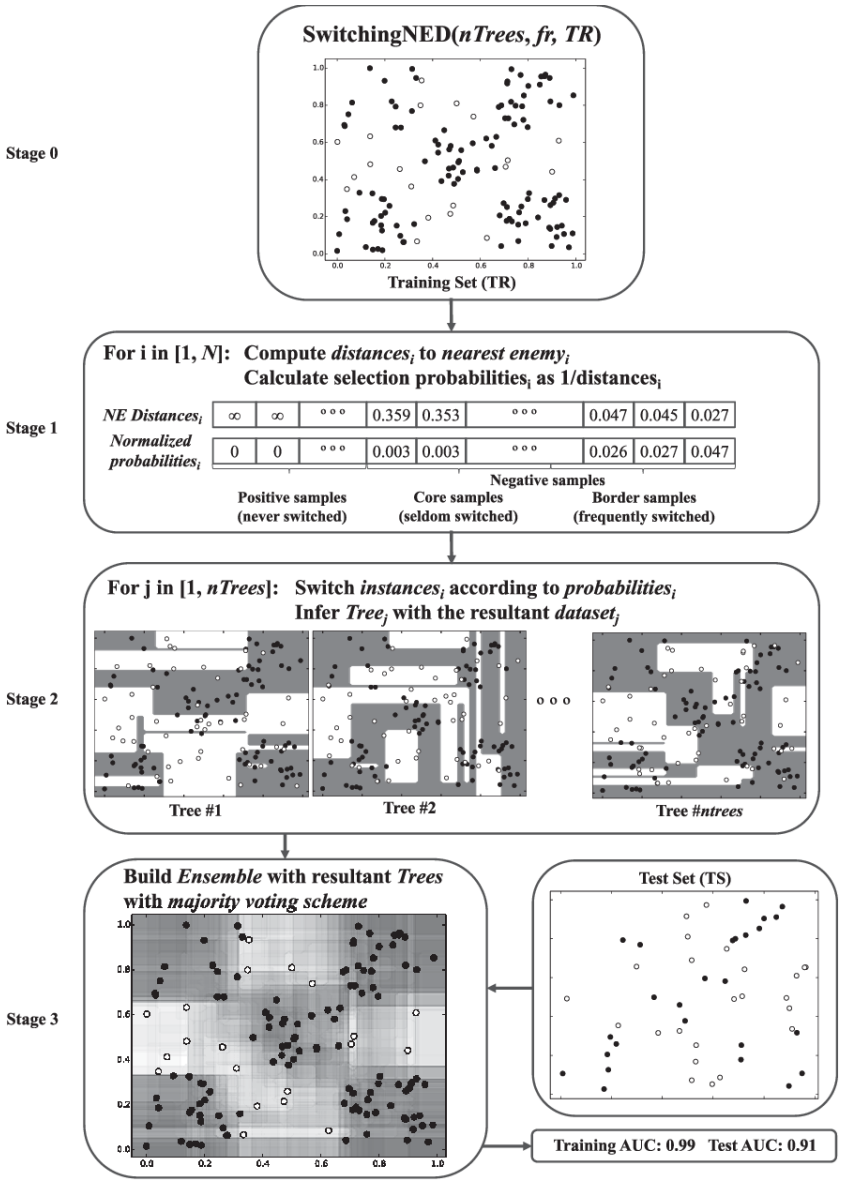


Class-Switching Ensembles

- Class switching is an ensemble method in which diversity is obtained by using different versions of the training data polluted with class label noise.

- Specifically, to train each base learner, the class label of each training point is changed to a different class label with probability p.





Ensemble Pruning

1.- Random ordering produced by bagging

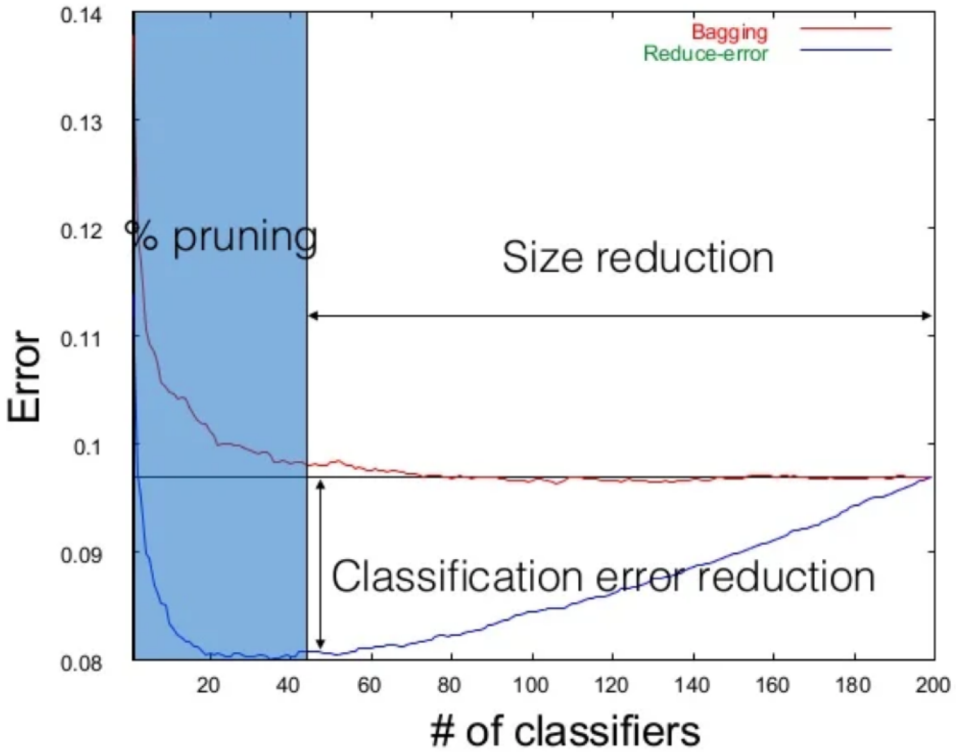
$h_1, h_2, h_3, .., h_T$

2.- New ordering

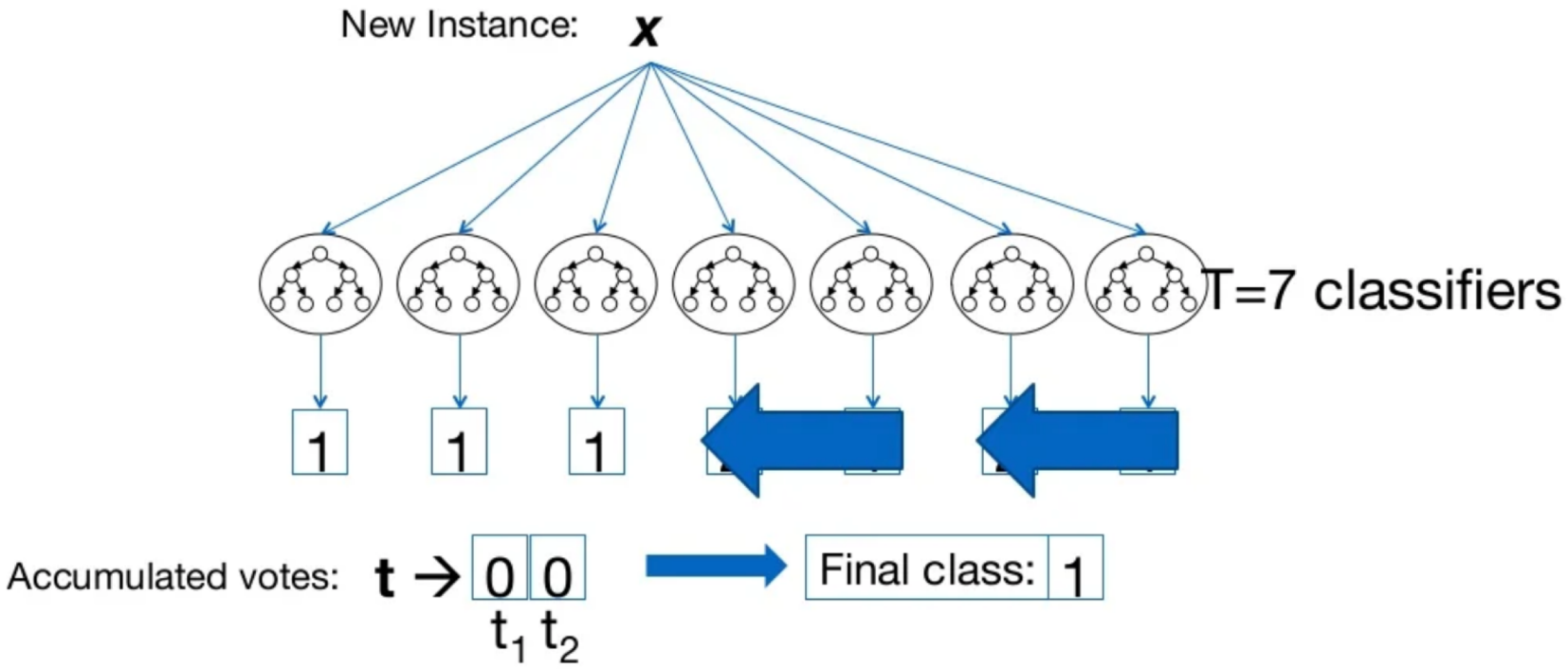
$h_{s1}, h_{s2}, h_{s3}, .., h_{sT}$

3.- Pruning

$h_{s1}, .., h_{sM}$



Dynamic Ensemble Pruning



Do we really need to query all classifiers in the ensemble?

NO

Summary

- Decision trees are simple and interpretable models for regression and classification.
- However they are sometime not competitive with other methods in terms of prediction accuracy.
- Bagging, random forests and boosting are good methods for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.
- Two methods— random forests and gradient boosting (any form)— are among the state-of-the-art methods for supervised learning. However their results can be difficult to interpret.

Summary

Manuel Fernandez-Delgado, Eva Cernadas, Senen Barro, Dinani Amorim, Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?, Journal of Machine Learning Research 15 (2014) 3133-3181

"We evaluate 179 classifiers arising from 17 families (discriminant analysis, Bayesian, neural networks, support vector machines, decision trees, rule-based classifiers, boosting, bagging, stacking, random forests and other ensembles, generalized linear models, nearest neighbors, partial least squares and principal component regression, logistic and multinomial regression, multiple adaptive regression splines and other methods), …

We use 121 data sets, which represent the whole UCI data base (excluding the large-scale problems) and other own real problems, in order to achieve significant conclusions about the classier behavior, not dependent on the data set collection.

The classifiers most likely to be the bests are the random forest (RF) versions, the best of which achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets. However, the difference is not statistically significant with the second best, the SVM with Gaussian kernel, which achieves 92.3% of the maximum accuracy.

DESCOMPOSICIÓN DE PROBLEMAS MULTICLASE

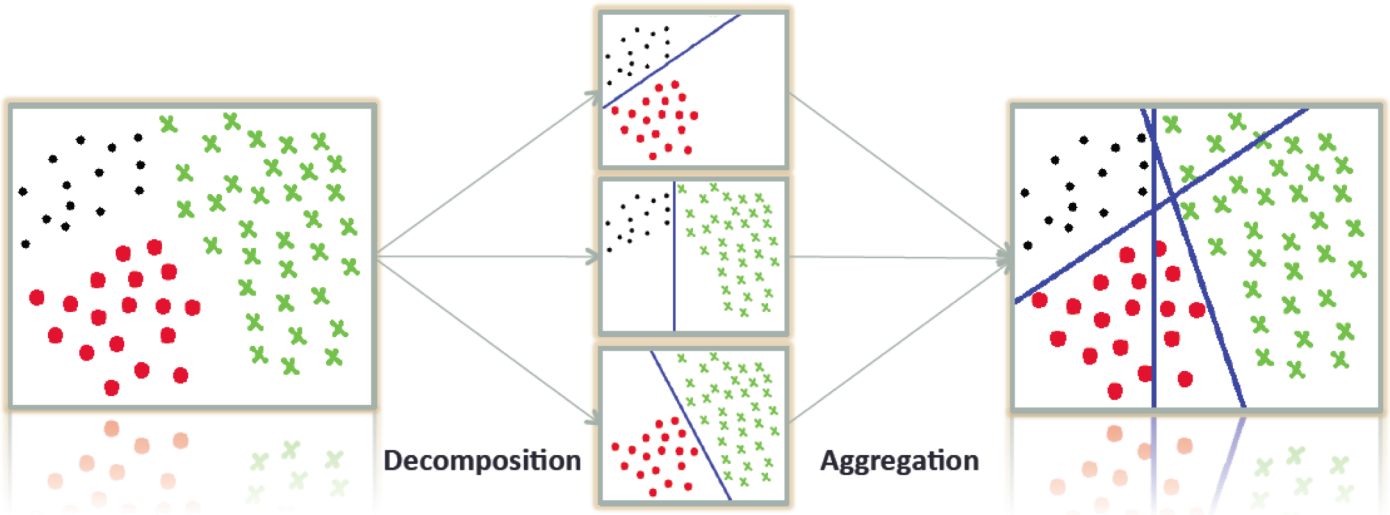
Minería de Datos: Preprocesamiento y clasificación

Descomposición de problemas multiclase

- Diversos algoritmos de clasificación están limitados a resolver problemas binarios, es decir, con dos clases.
- Una manera de abordar el problema de clasificación multi-clase es descomponerlo en problemas binarios y construir un modelo de clasificación para cada uno.
- Dos estrategias comunes de descomposición de problemas multi-clase son:
  - Uno contra uno (OVO, one-versus-one).
  - Uno contra todos (OVA, one-versus-all).
- Otra estrategia menos común es la denominada "todos y uno" (A&O, all-and-one), la cual es una combinación de OVO y OVA.

Descomposición OVO

- Para un problema de clasificación con c clases se crean $c(c-1)/2$ modelos de clasificación binarios, los cuales corresponden a todas las posibles combinaciones entre pares de clases.
- Cada clasificador binario aprende a distinguir entre las i-ésima y j-ésima clases, por lo que se entrena únicamente con instancias de dichas clases.
- Para clasificar un patrón, se combinan las respuestas de todos los $c(c-1)/2$ clasificadores construidos.
- Clasificación con OVO para tres clases:



Descomposición OVO

- Sea $r_{i,j}$ la respuesta de un clasificador entrenado para discriminar entre las clases ω_i y ω_j, tal que $r_{i,j} \in [-\infty, \infty]$ y $r_{j,i} = -r_{i,j}$.
- La regla de decisión es:

$$\text{Decidir} \begin{cases} \omega_i & \text{si } r_{i,j} > 0 \\ \omega_j & \text{otro caso} \end{cases}$$

- Sea R la matriz de respuestas de todos los clasificadores creados mediante la descomposición OVO para un patrón arbitrario:

$$R = \begin{pmatrix} 0 & r_{1,2} & \cdots & r_{1,j} & \cdots & r_{1,c} \\ r_{2,1} & 0 & \cdots & r_{2,j} & \cdots & r_{2,c} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ r_{i,1} & r_{i,2} & \cdots & 0 & \cdots & r_{i,c} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{c,1} & r_{c,2} & \cdots & r_{c,j} & \cdots & 0 \end{pmatrix}$$

- La predicción de clase se hace usando la información de la matriz R aplicando una estrategia de agregación.

Descomposición OVO-MV

- En la estrategia de agregación “voto mayoritario” (MV) la clase ganadora es aquella que haya tenido un mayor número de respuestas positivas y se expresa como:

$$\omega_i = \arg\max_{i=1,\ldots,c} \sum_{1 \le j \le c} v_{i,j}, \quad \text{para } i \neq j$$

- donde

$$v_{i,j} = \begin{cases} 1 & \text{si } r_{i,j} > 0 \\ 0 & \text{otro caso} \end{cases}$$

- Ventaja: es una estrategia simple con resultados competitivos.
- Desventaja: es posible que resulten zonas en el espacio de características que produzcan empates en el número de votos.

Descomposición OVO-WV

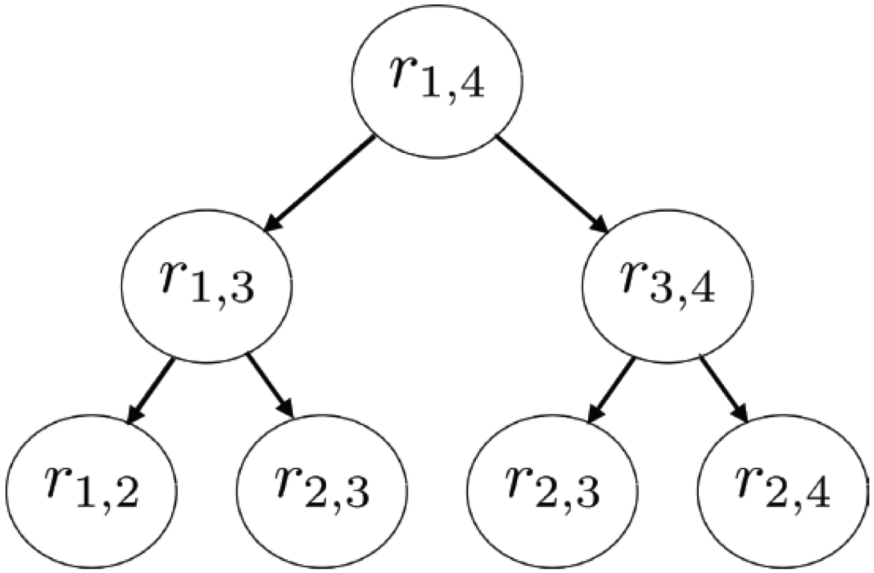
- En la estrategia de agregación "voto ponderado" (WV) la clase ganadora es aquella cuya suma de respuestas sea la de mayor magnitud y se expresa como:

$$\omega_i = \arg\max_{i=1,\ldots,c} \sum_{1 \le j \le c} r_{i,j}, \quad \text{para } i \neq j$$

- Ventaja: es poco probable que existan zonas en el espacio de características que produzcan empates con una misma suma de respuestas.
- Desventaja: es posible que la clase con máxima suma de respuestas no sea la clase verdadera.

Descomposición OVO-DDAG

- En la estrategia de agregación “grafo acíclico dirigido de decisión” (DDAG) se crea un árbol binario de decisión donde cada nodo representa la respuesta de un clasificador.
- De esta manera, cuando una se predice una clase, otra clase es descartada, por lo que es necesario descartar la pertenencia a $c-1$ clases.
- La clase ganadora es aquella con respuesta positiva en el último nodo.

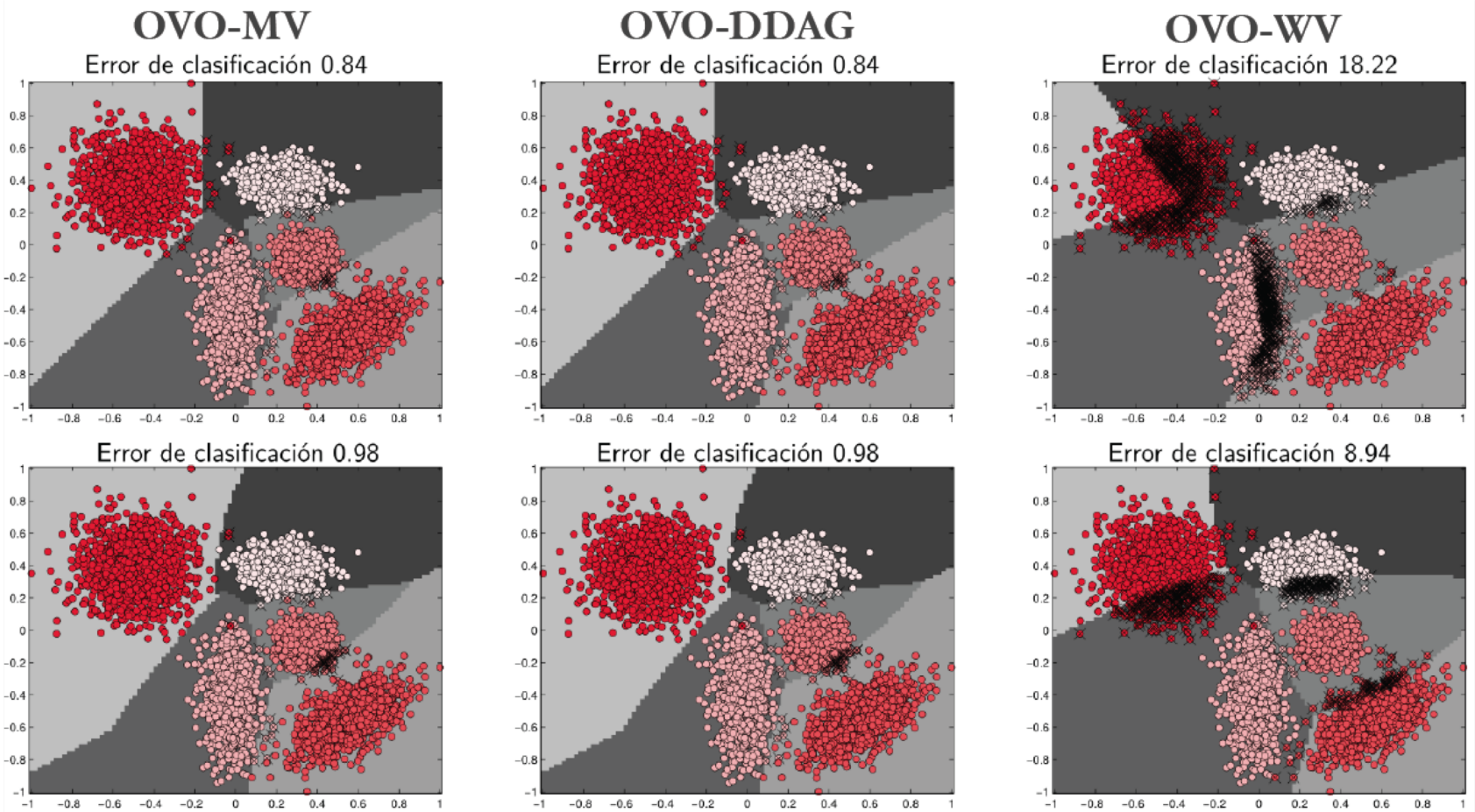


Árbol binario de decisión para cuatro clases.

Descomposición OVO-DDAG

- En los nodos superiores se colocan los clasificadores que produzcan una mayor separación entre clases, aunque no hay evidencia sustancial de que el ordenamiento de los nodos con alguna preferencia mejore la clasificación.
- Ventajas: tiene el mismo desempeño que OVO-MV para los casos en que el número de respuestas positivas para una clase es $c-1$. Además, no hay ambigüedad en la predicción de clase.
- Desventaja: el proceso de construcción y evaluación del grafo puede resultar más costoso en comparación de la estrategia OVO-MV.

Comparativo OVO



Regiones de decisión creadas por la estrategia de descomposición OVO para dos tipos de clasificadores binarios: LDA (fila superior) y SVM-RBF (fila inferior).

Example of prediction

Classify x, whose real class is c_1

$$R(x) = \begin{pmatrix} & c1 & c2 & c3 & c4 & c5 \\ c1 & - & 0{,}55 & 0{,}6 & 0{,}75 & 0{,}7 \\ c2 & 0{,}45 & - & 0{,}4 & 1 & 0{,}8 \\ c3 & 0{,}4 & 0{,}6 & - & 0{,}5 & 0{,}4 \\ c4 & 0{,}25 & 0{,}0 & 0{,}5 & - & 0{,}1 \\ c5 & 0{,}30 & 0{,}2 & 0{,}6 & 0{,}9 & - \end{pmatrix}$$

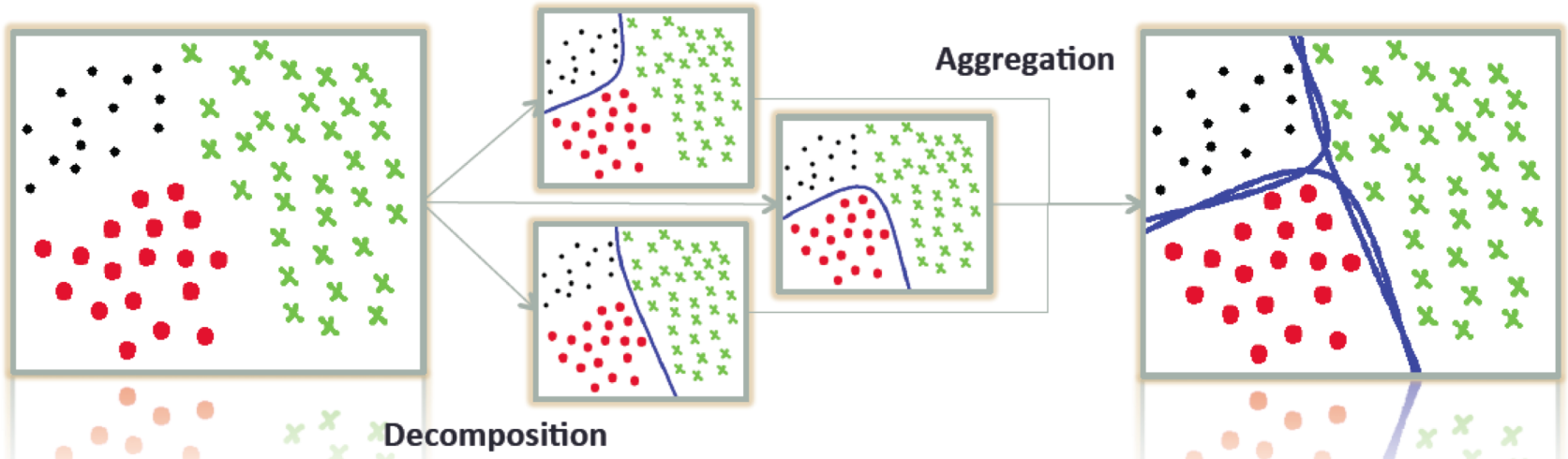
Example of prediction

- Beware of **Non-Competent** Classifiers:
  - Consider WV aggregation, c_2 is predicted
  - **None** of the classifiers **considering c_1 failed**
  - **Non-competent** classifiers **strongly voted for** c_2
  - $R(x) =$

	$c1$	$c2$	$c3$	$c4$	$c5$	WV
$c1$	–	0,55	0,6	0,75	0,7	2,6
$c2$	0,45	–	0,4	1	0,8	**2,65**
$c3$	0,4	0,6	–	0,5	0,4	1,9
$c4$	0,25	0,0	0,5	–	0,1	0,85
$c5$	0,30	0,2	0,6	0,9	–	2,1

Descomposición OVA

- Para un problema de clasificación con c clases se crean c modelos de clasificación binarios, donde cada uno de ellos aprende a distinguir una clase determinada del resto.
- El vector de respuestas es $\mathbf{r}=[r_1,\ldots,r_c]$, tal que $r_i \in [-\infty,\infty]$ y $r_i>0$ para muestras de la clase ω_i.
- La principal desventaja de esta descomposición es el alto desbalance en el número de muestras de entrenamiento para cada clasificador base.



Descomposición OVA-MAX

- La estrategia de agregación más común se denomina de “máxima confianza” (MAX), en la cual la clase ganadora es aquella con la máxima respuesta positiva:

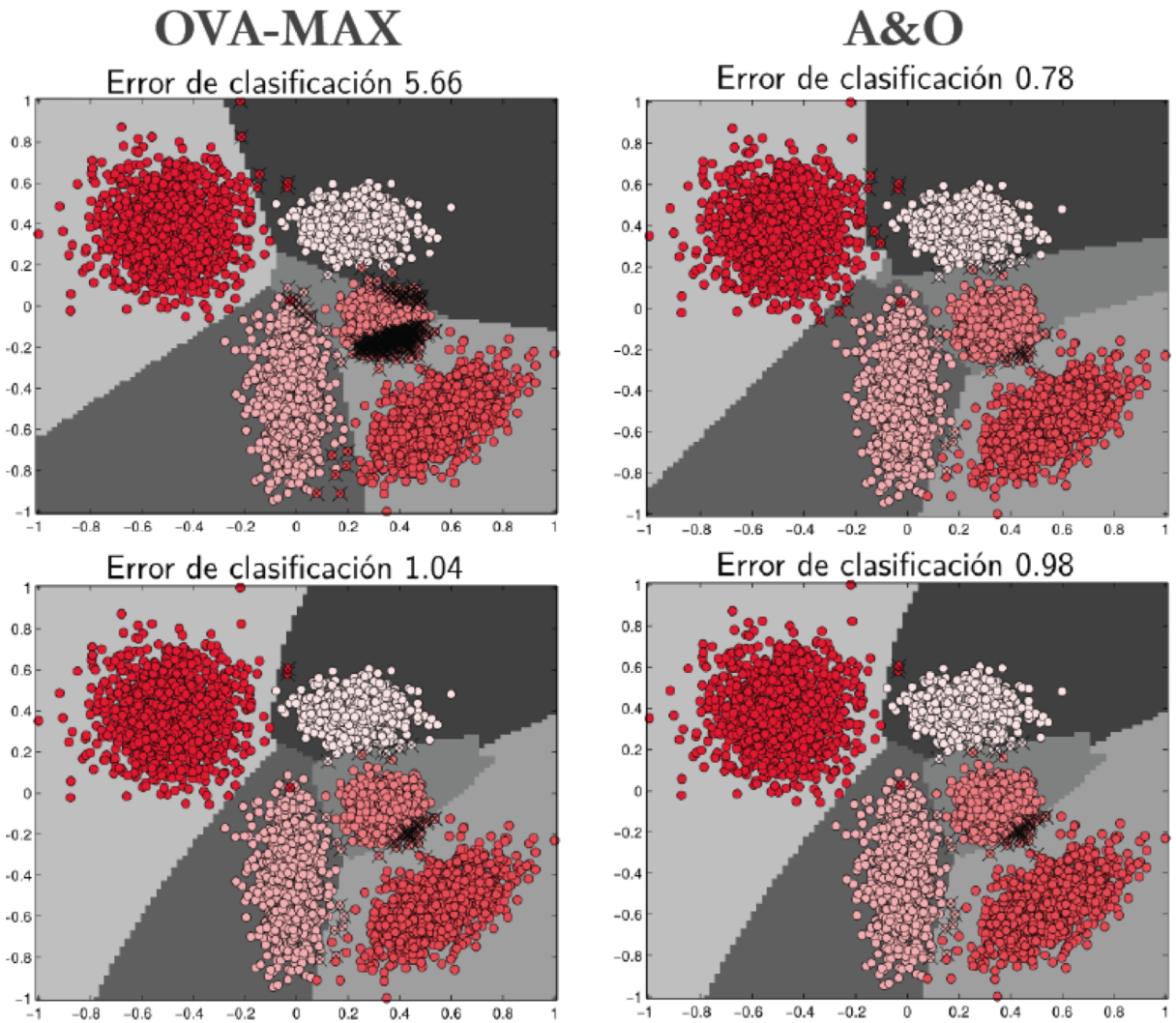
$$\omega_i = \arg\max_{i=1,\ldots,c} r_i$$

- Ventaja: es poco probable que haya empates en la máxima respuesta positiva.
- Desventaja: tomar el valor máximo no garantiza una clasificación correcta. Puede suceder que no existan respuestas positivas, aunque se toma la respuesta negativa más cercana a cero.

Descomposición A&O

- En la estrategia A&O es necesario entrenar los clasificadores base de las estrategias OVO y OVA.
- La clasificación de una instancia se realiza como:
  - Obtener las dos clases, ω_a y ω_b, cuyos correspondientes clasificadores en la estrategia OVA, c_a y c_b, generan las respuestas más altas.
  - Clasificar la instancia con el clasificador $c_{a,b}$ en la estrategia OVO.
- Desventaja: es necesario entrenar $c(c+1)/2$ clasificadores, por lo que aumenta el costo espacial y temporal.
- Ventajas: en algunos casos se obtienen mejores resultados que usar de manera independiente OVO ó OVA.

Comparativa OVA



Estudios sobre la descomposición binaria

- M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, F. Herrera, An Overview of Ensemble Methods for Binary Classifiers in Multi-class Problems: Experimental Study on One-vs-One and One-vs-All Schemes. *Pattern Recognition 44:8 (2011) 1761-1776, doi: 10.1016/j.patcog.2011.01.017*

- Anderson Rocha and Siome Goldenstein. Multiclass from Binary: Expanding One-vs-All, One-vs-One and ECOC-based Approaches. IEEE Transactions on Neural Networks and Learning Systems 25:2 (2014) 289–302 doi:10.1109/TNNLS.2013.2274735