Introduction to Machine Learning
NPFL 054

http://ufal.mff.cuni.cz/course/npfl054

Barbora Hladká
hladka@ufal.mff.cuni.cz

Martin Holub
holub@ufal.mff.cuni.cz

Charles University,
Faculty of Mathematics and Physics,
Institute of Formal and Applied Linguistics

Ensemble learning methods

Outline

- Decision Trees – deeper learning details and overfitting
- Combining classifiers into ensembles – general scheme
- Generating random samples by bootstrapping
- Bagging vs. boosting
- Bagging – example classifier
- Random Forests
- Simple boosting – the regression case
- Adaptive boosting – classification with AdaBoost

Historical excursion

$$\begin{tikzcd}
\text{Decision trees concept (Hunt, 1962)} \arrow[d] & \text{AID (Morgan, 1964)} \arrow[d] \\
\text{ID3 (Quinlan, 1979)} \arrow[d] & \text{CART (Breiman, 1984)} \\
\text{C4.5 (Quinlan, 1993)} &
\end{tikzcd}$$

- ID3 $\sim$ Iterative Dichotomiser
- AID $\sim$ Automatic Interaction Detection
- CART $\sim$ Classification and Regression Trees

Probably most well-known is the "C 5.0" algorithm (Quinlan), which has become the industry standard.
Packages in R: `rpart`

Building a classification tree from training data

We work with decisions on the value of only a single feature

- For each categorical feature A_j having values $Values(A_j) = \{b_1, b_2, ..., b_L\}$

$$\boxed{\text{is } x_j = b_i\text{? as } i = 1, ..., L}$$

- For each categorical feature A_j

$$\boxed{\text{is } x_j \in \text{a subset} \in 2^{Values(A_j)}\text{?}}$$

- For each numerical feature A_j

$$\boxed{\text{is } x_j \leq k\text{?}, \ k \in (-\infty, +\infty)}$$

Building a classification tree from training data

Which decision is the best?

- Focus on the distribution of target class values in the associated subset of training examples.
- Then select the decision that splits training data into subsets as pure as possible.

Building a classification tree from training data

Which decision is the best?

We say a data set is **pure** (or **homogenous**) if it contains only a single class. If a data set contains several classes, then the data set is **impure** (or **heterogenous**).

Building a classification tree from training data

Which decision is the best?

We say a data set is **pure** (or **homogenous**) if it contains only a single class. If a data set contains several classes, then the data set is **impure** (or **heterogenous**).

Example:

$\oplus$: 5, $\ominus$: 5		$\oplus$: 9, $\ominus$: 1
heterogenous high degree of impurity		almost homogenous low degree of impurity

Building a classification tree from training data

Which decision is the best?

1. **Define** a candidate set S of splits at each node using possible decisions. $s \in S$ splits t into two subsets t_1 and t_2.
2. **Define** the node proportions $p(y_j|t), j = 1, \dots, k$, to be the proportion of instances $\langle \mathbf{x}, y_j \rangle$ in t.
3. **Define** an **impurity measure** $i(t)$, i.e. **splitting criterion**, as a non-negative function Φ of the $p(y_1|t), p(y_2|t), \dots, p(y_k|t)$,

$$i(t) = \Phi(p(y_1|t), p(y_2|t), \dots, p(y_k|t)), \tag{1}$$

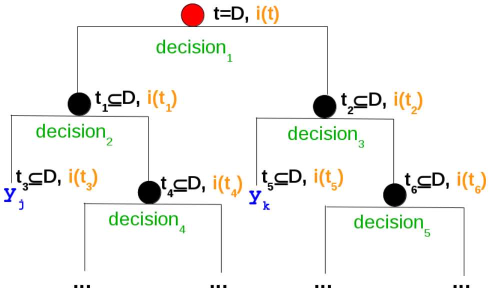
   such that

   - $\Phi(\frac{1}{k}, \frac{1}{k}, ..., \frac{1}{k}) = max$, i.e. the node impurity is largest when all examples are equally mixed together in it.
   - $\Phi(1, 0, ..., 0) = 0, \Phi(0, 1, ..., 0) = 0, ..., \Phi(0, 0, ..., 1) = 0$, i.e. the node impurity is smallest when the node contains instances of only one class

Building a classification tree from training data

Which decision is the best?

4. **Define** the **goodness of split s** to be the decrease in impurity
   $\Delta i(s,t) = i(t) - (p_1 * i(t_1) + p_2 * i(t_2))$,
   where p_i is the proportion of instances in t that go to t_i.
5. **Find** split s^* with the largest decrease in impurity:
   $\Delta i(s^*,t) = max_{s \in S} \Delta i(s,t)$.
6. **Use** splitting criterion $i(t)$ to compute $\Delta i(s,t)$ and get s^*.


t=D, i(t)
decision$_1$
$t_1 \subseteq D$, $i(t_1)$
$t_2 \subseteq D$, $i(t_2)$
decision$_2$
decision$_3$
$t_3 \subseteq D$, $i(t_3)$
y_j
$t_4 \subseteq D$, $i(t_4)$
$t_5 \subseteq D$, $i(t_5)$
y_k
$t_6 \subseteq D$, $i(t_6)$
decision$_4$
decision$_5$
...
...
...
...

Building a classification tree from training data

Which decision is the best?

Splitting criteria – examples that are really used

- Misclassification Error – $i(t)_{ME}$
- Information Gain – $i(t)_{IG}$
- Gini Index – $i(t)_{GI}$

Building a classification tree from training data

Which decision is the best?
Splitting criteria

$$i(t)_{ME} = 1 - max_{j=1,\ldots,k} p(y_j|t) \tag{2}$$

Building a classification tree from training data

Which decision is the best?
Splitting criteria

$$i(t)_{ME} = 1 - max_{j=1,\dots,k} p(y_j|t) \qquad (2)$$

Example:

	$\oplus$: 0, $\ominus$: 6	$\oplus$: 1, $\ominus$: 5	$\oplus$: 2, $\ominus$: 4	$\oplus$: 3, $\ominus$: 3
$i(t)_{ME}$	$1 - \frac{6}{6} = 0$	$1 - \frac{5}{6} = 0.17$	$1 - \frac{4}{6} = 0.33$	$1 - \frac{3}{6} = 0.5$

Building a classification tree from training data

Which decision is the best?
Splitting criteria

$$i(t)_{IG} = -\sum_{j=1}^{k} p(y_j|t) * \log p(y_j|t). \tag{3}$$

Recall the notion of entropy $H(t)$, $i(t)_{IG} = H(t)$.

$$Gain(s,t) = \Delta i(s,t)_{IG} \tag{4}$$

Building a classification tree from training data

Which decision is the best?
Splitting criteria

$$i(t)_{GI} = 1 - \sum_{j=1}^{k} p^2(y_j|t) = \sum_{j=1}^{k} p(y_j|t)(1 - p(y_j|t)). \qquad (5)$$

Building a classification tree from training data

Which decision is the best?
Splitting criteria

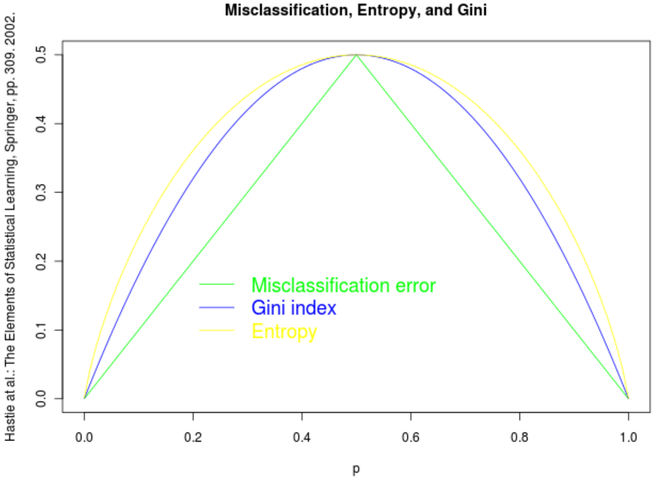
	$\oplus$: 0 $\ominus$: 6	$\oplus$: 1 $\ominus$: 5	$\oplus$: 2 $\oplus$: 4	$\oplus$: 3 $\oplus$: 3
Gini	0	0.278	0.444	0.5
Entropy	0	0.65	0.92	1.0
ME	0	0.17	0.333	0.5

For two classes ($k = 2$), if p is the proportion of the class "1", the measures are:

- Misclassification error: $1 - max(p, 1 - p)$
- Entropy: $-p * \log p - (1 - p) * \log(1 - p)$
- Gini: $2p * (1 - p)$

Building a classification tree from training data

Which decision is the best?
Splitting criteria



Building a regression tree from training data

Again, we work with decisions on the value of only a single feature

Which decision is the best?

Splitting criterion – usually used

- Squared Error – $i(t)_{SE}$

$$i(t)_{SE} = \frac{1}{|t|} \sum_{\mathbf{x}_i \in t} (y_i - y^t)^2,$$

where $y^t = \frac{1}{|t|} \sum_{\mathbf{x}_i \in t} y_i$.

Building decision tree from training data

The recursive binary splitting is stopped when a stopping criterion is fulfilled. Then a leaf node is created with an output value.

Stopping criteria, e.g.

- the leaf node is associated with less than five training instances
- the maximum tree depth has been reached
- the best splitting criteria is not greater than a certain threshold

Building a decision tree from training data

Overfitting can be avoided by

- applying a stopping criterion that prevents some sets of training instances from being subdivided,
- removing some of the structure of the decision tree after it has been produced.

Preferred strategy
Grow a large tree T_0, stop the splitting process when only some minimum node size (say 5) is reached. Then prune T_0 using some pruning criteria.

Decision trees learning parameters

2 phases of decision tree learning:

- growing
- pruning

Learning parameters are used to control these two phases:

Decision trees learning parameters

2 phases of decision tree learning:

- growing
- pruning

Learning parameters are used to control these two phases:

- when to stop growing
- how much to prune the tree

Decision trees learning parameters

2 phases of decision tree learning:

- growing
- pruning

Learning parameters are used to control these two phases:

- when to stop growing
- how much to prune the tree

... to avoid overfitting and improve performance

Learning parameters in `rpart`

`rpart.control`

minsplit

- the minimum number of observations that must exist in a node in order for a split to be attempted

cp

- complexity parameter, influences the depth of the tree

... and others, see `?rpart.control`

T: try to set different `cp` and `minsplit` values in the M1 model learning and observe the resulting tree

cp parameter

Any split that does not decrease the **relative training error** by a factor of cp is not attempted

⇒ That means, the learning algorithm measures for each split how it improves the tree relative error and if the improvement is too small, the split will not be performed.

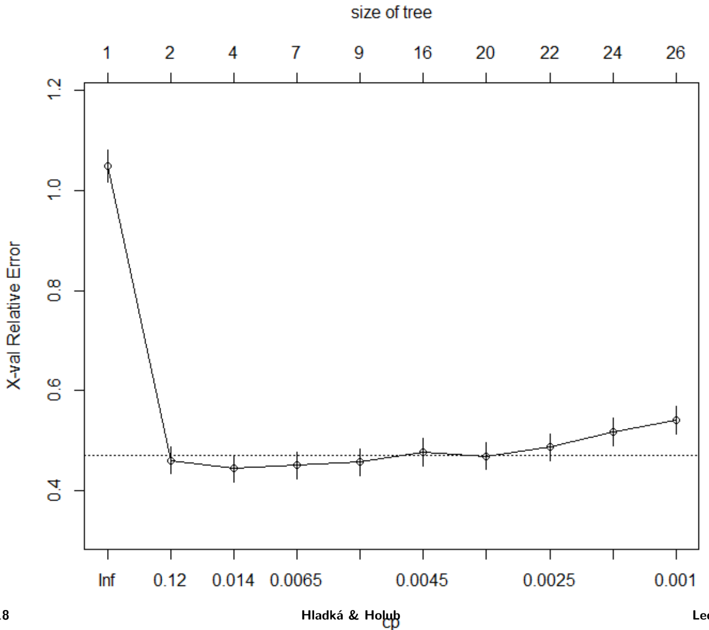
Relative error is the error relative to the misclassification error (without any splitting relative error is 100%)

How to choose the optimal cp value?

```
> m = rpart(profits ~ category + sales + assets + marketvalue,
            data=F[data.train, 1:8], cp=0.001)
> m$cptable
            CP nsplit rel error    xerror       xstd
1  0.543259557      0 1.0000000 1.0482897 0.03178559
2  0.027162978      1 0.4567404 0.4607646 0.02673551
3  0.007042254      3 0.4024145 0.4446680 0.02640028
4  0.006036217      6 0.3762575 0.4507042 0.02652763
5  0.005030181      8 0.3641851 0.4567404 0.02665301
6  0.004024145     15 0.3279678 0.4768612 0.02705703
7  0.003018109     19 0.3118712 0.4688129 0.02689795
8  0.002012072     21 0.3058350 0.4869215 0.02725122
9  0.001006036     23 0.3018109 0.5171026 0.02780383
10 0.001000000     25 0.2997988 0.5412475 0.02821490
```

rel error	relative error on training data
xerror	relative error in x-fold **cross-validation**
xstd	standard deviation of `xerror` on x validation folds

How to choose the optimal cp value?



Ensemble classifiers – a motivation exercise

Consider the following task – we have a binary classification problem and a number of predictors, each with error less than 0.5. Will the resulting majority voting ensemble have an error lower than the single classifers?

Ensemble classifiers – a motivation exercise

Consider the following task – we have a binary classification problem and a number of predictors, each with error less than 0.5. Will the resulting majority voting ensemble have an error lower than the single classifers?

Depends on the *accuracy* and the *diversity* of the base learners!

Ensemble classifiers – a motivation exercise

Consider the following task – we have a binary classification problem and a number of predictors, each with error less than 0.5. Will the resulting majority voting ensemble have an error lower than the single classifers?

Depends on the *accuracy* and the *diversity* of the base learners!

Illustrative example

Particular settings – assume that you have

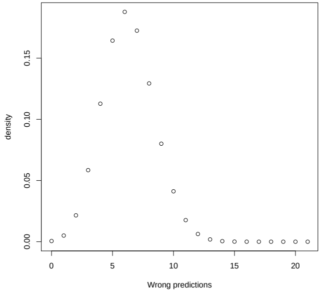
- 21 classifiers
- each with error $p = 0.3$
- their outputs are *statistically independent*

Compute the error of the ensemble under these conditions!

Solution of the exercise

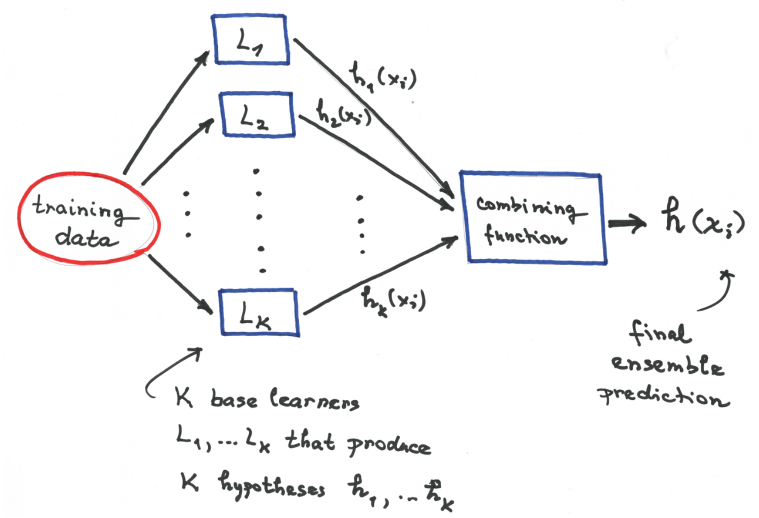
How many classifiers will produce error output?
Key idea: The number of them will be binomially distributed! $\sim$ Bi(21, 0.3)

```
> plot(0:21, dbinom(0:21, 21, 0.3))
> dbinom(11, 21, 0.3)
[1] 0.01764978
> pbinom(10, 21, 0.3)
[1] 0.9736101
```



Conslusion: Accuracy of the ensemble will be more than 97.3 %!

General scheme of combining classifiers



Resampling approach

Resampling can be used as a way to produce diversity among base learners

- Distribute the training data into K portions
- Run the learning process to get K different models
- Collect the output of the K models use a combining function to get a final output value

Bootstrapping principle

- New data sets $Data_1, \ldots, Data_K$ are drawn from *Data* with replacement, each of the same size as the original *Data*, i.e. n.
- In the i-th step of the iteration, $Data_i$ is used as a training set, while the examples $\{\mathbf{x} \,|\, \mathbf{x} \in Data \wedge \mathbf{x} \notin Data_i\}$ form the test set.

Bootstrapping principle

- New data sets $\mathit{Data}_1, \ldots, \mathit{Data}_K$ are drawn from Data with replacement, each of the same size as the original Data, i.e. n.
- In the i-th step of the iteration, Data_i is used as a training set, while the examples $\{\mathbf{x} \mid \mathbf{x} \in \mathit{Data} \wedge \mathbf{x} \notin \mathit{Data}_i\}$ form the test set.
- The probability that we pick an instance is $1/n$, and the probability that we do not pick an instance is $1 - 1/n$. The probability that we do not pick it after n draws is $(1 - 1/n)^n \approx e^{-1} \doteq 0.368$.
- It means that for training the system will not use 36.8 % of the data, and the error estimate will be pessimistic. So the solution is to repeat the process many times.

Same algorithm, different classifiers

Combining classifiers to improve the performance

Ensemble methods – key ideas

- combining the classification results from different classifiers to produce the final output
- using (un)weighted voting
- different training data – e.g. bootstrapping
- different features
- different values of the relevant paramaters
- performance: complementarity $\longrightarrow$ potential improvement

Two fundamental approaches

- **Bagging** works by taking a bootstrap sample from the training set
- **Boosting** works by changing the weights on the training set

Bagging and boosting — the difference

- **Bagging**: each predictor is trained independently
- **Boosting**: each predictor is built on the top of previous predictors trained

  – Like bagging, boosting is also a voting method. In contrast to bagging, boosting actively tries to generate complementary learners by training the next learner on the mistakes of the previous learners.

Are ensembles effective?

Combining multiple learners

- the more **complementary** the learners are, the more useful their combining is
- the simpliest way to combine multiple learners is **voting**
- in **weighted voting** the voters (= base-learners) can have different weights

Unstable learning

- learning algorithm is called unstable if small changes in the training set cause large differences in generated models
- typical unstable algorithm is the decision trees learning
- bagging or boosting techniques are a natural remedy for unstable algorithms

Bagging

- Bagging is a voting method that uses slightly different training sets (generated by bootstrap) to make different base-learners. Generating complementary base-learners is left to chance and to unstability of the learning method.
- Generally, bagging can be combined with any approach to learning.

Bagging – algorithm

Bootstrap **AGG**regat**ING**

```
1 for $i \leftarrow 1$ to K do
2 $Train_i \leftarrow \text{bootstrap}(Data)$
3 $h_i \leftarrow \text{TrainPredictor}(Train_i)$
```

Combining function

- **Classification:** $h_{final}(\mathbf{x}) = \text{MajorityVote}(\text{h}_1(\mathbf{x}), \text{h}_2(\mathbf{x}), \dots, \text{h}_\text{K}(\mathbf{x}))$
- **Regression:** $h_{final}(\mathbf{x}) = \text{Mean}(\text{h}_1(\mathbf{x}), \text{h}_2(\mathbf{x}), \dots, \text{h}_\text{K}(\mathbf{x}))$

Random Forests

- an ensemble method based on decision trees and bagging
- builds a number of random decision trees and then uses voting
- introduced by L. Breiman (2001), then developed by L. Breiman and A. Cutler
- very good (state-of-the-art) prediction performance
- a nice page with description
  `www.stat.berkeley.edu/∼breiman/RandomForests/cc_home.htm`
- important: Random Forests helps to
  - avoid overfitting (by random sampling the training data set)
  - select important/useful features (by random sampling the feature set)

Building Random Forests

The algorithm for building a tree in the ensemble

1. Having a training set of the size n, sample n cases at random – with replacement, and use the sample to build a decision tree.
2. If there are M input features, choose a less number $m \ll M$. When building the tree, at each node a random sample of m features is selected as split candidates from the full set of M available features. Then the best split on these m features is used to split the node. A fresh sample of m features is taken at each split.
   – m is fixed for the whole procedure
3. Each tree is grown to the largest extent possible. There is no pruning.

The more trees in the ensemble, the better.
There is no risk of overfitting!

Regularized Random Forests

- a recent extension of the original Random Forest
  - introduced by Houtao Deng and George Runger (2012)
- produces a compact feature subset
- provides an effective and efficient feature selection solution for many practical problems
- overcomes the weak spot of the ordinary RF: Random Forest importance score is biased toward the variables having more (categorical) values
- a useful page: `https://sites.google.com/site/houtaodeng/rrf`
  - a presentation
  - a sample code
  - links to papers
  - a brief explanation of the difference between RRF and guided RRF

R packages for Random Forests

- **randomForest**: Breiman and Cutler's random forests for classification and regression
  – Classification and regression based on a forest of trees using random inputs.
- **RRF**: Regularized Random Forest
  – Feature Selection with Regularized Random Forest. This package is based on the 'randomForest' package by Andy Liaw. The key difference is the RRF function that builds a regularized random forest.
  – `http://cran.r-project.org/web/packages/RRF/index.html`
- **party**: A Laboratory for Recursive Partytioning
  – a computational toolbox for recursive partitioning
  – `cforest()` provides an implementation of Breiman's random forests
  – extensible functionality for visualizing tree-structured regression models is available

Boosting

Boosting combines the outputs of many “weak” classifiers (“rules of thumb”) to produce a powerfull “commitee.”

Motivation

- How to extract rules of thumb that will be the most useful?
- How to combine moderately accurate rules of thumb into a single highly accurate prediction rule?

Basic idea

- Boosting is a method that produces a very accurate predictor by combininig rough and moderately accurate predictors.
- It is based on the observation that finding many rough predictors (rules of thumb) can be easier than finding a single, highly accurate predictor.

Simple boosting with regression trees

1. Initialization: Set $h(x) = 0$ and $r_i = y_i$ for all $i = 1, \dots, n$ in the training set
2. For $b = 1, \dots, B$, repeat
   - **(a)** Fit a tree h^b with only d splits to the training set (X, r)
   - **(b)** Update h by adding the new tree

     $$h(x) \longleftarrow h(x) + \lambda h^b(x)$$

   - **(c)** Update the residuals

     $$r_i \longleftarrow r_i - \lambda h^b(x_i)$$

3. Output the boosted model

$$h(x) = \sum_{b=1}^{B} \lambda h^b(x)$$

Boosting with regression trees – tuning parameters

- The number of trees B
- The shrinkage parameter λ
- The number d of splits in each tree
  — trees with just $d = 1$ split are called “stumps”

Boosting — Adaboost (Adaptive Boosting)

AdaBoost is a boosting method that repeatedly calls a given weak learner, each time with different distribution over the training data. Then we combine these weak learners into a strong learner.

Boosting — Adaboost (Adaptive Boosting)

AdaBoost is a boosting method that repeatedly calls a given weak learner, each time with different distribution over the training data. Then we combine these weak learners into a strong learner.

- originally proposed by Freund and Schapire (1996)
- great success
  — "AdaBoost with trees is the best off-the-shelf classifier in the world." (Breiman 1998)

  — "Boosting is one of the most powerful learning ideas introduced in the last twenty years." (Hastie et al, 2009)

Boosting — Adaboost (Adaptive Boosting)

Key questions

- How to choose the distribution?
- How to combine the weak predictors into a single predictor?
- How many weak predictors should be trained?

Schapire's strategy: Change the distribution over the examples in each iteration, feed the resulting sample into the weak learner, and then combine the resulting hypotheses into a voting ensemble, which, in the end, would have a boosted prediction accuracy.

Binary classification and AdaBoost.M1

AdaBoost.M1 (Freund and Schapire, 1997) is the most popular boosting algorithm

- Consider a binary classification task with the training data

$$Data = \{\langle \mathbf{x}_i, y_i \rangle : \mathbf{x}_i \in \mathbf{X}, y_i \in \{-1, +1\}, i = 1, \dots, n\}$$

- We need to define distribution $\mathcal{D}$ over *Data* such that $\sum_{i=1}^{n} \mathcal{D}_i = 1$.

- Assumption: a weak classifier h_t has the property

$$\mathrm{error}_{\mathcal{D}}(h_t) < 1/2.$$

Adaboost (Adaptive Boosting) — key idea

Classifiers are trained on weighted versions of the original training data set, and then combined to produce a final prediction

$$\begin{tikzcd}
\boxed{\text{Training examples}} \arrow[r] \arrow[d] & h_1(x) \\
\boxed{\text{Weighted examples}} \arrow[r] \arrow[d] & h_2(x) \\
\boxed{\text{Weighted examples}} \arrow[r] \arrow[d] & h_3(x) \\
\vdots & \\
\boxed{\text{Weighted examples}} \arrow[r] & h_M(x)
\end{tikzcd}$$

Adaboost (Adaptive Boosting) — key idea

Classifiers are trained on weighted versions of the original training data set, and then combined to produce a final prediction

$$\begin{tikzcd}
\boxed{\text{Training examples}} \arrow[r] \arrow[d] & h_1(x) \\
\boxed{\text{Weighted examples}} \arrow[r] \arrow[d] & h_2(x) \\
\boxed{\text{Weighted examples}} \arrow[r] \arrow[d] & h_3(x) \\
\vdots & \\
\boxed{\text{Weighted examples}} \arrow[r] & h_M(x)
\end{tikzcd}$$

Final hypothesis $h(x) = \text{sign} \sum_{t=1}^{M} \alpha_t h_t(x)$, where α_t are computed by the boosting algorithm, and weight the contribution of each respective h_t

AdaBoost – iterative algorithm

- Initialize the training distribution $\mathcal{D}_1(i) = 1/n \quad \text{for } i = 1, \ldots, n$
- At each step t
  - Learn h_t using $\mathcal{D}_t$: find the weak classifier h_t with the minimum weighted sample error $\mathrm{error}_{\mathcal{D}_t}(h_t) = \sum_{i=1}^{n} \mathcal{D}_t(i)\, \delta(h(\mathbf{x}_i) \neq y_i)$
  - Set weight α_t of h_t based on the sample error
$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \mathrm{error}_{\mathcal{D}_t}(h_t)}{\mathrm{error}_{\mathcal{D}_t}(h_t)} \right)$$
  - Update the training distribution
$$\mathcal{D}_{t+1} = \mathcal{D}_t\, \mathrm{e}^{-\alpha_t y_i h_t(\mathbf{x}_i)} / Z_t \quad \text{where } Z_t \text{ is a normalization factor}$$
  - Stop when impossible to find a weak classifier being better than chance
- Output the final classifier $h_{final}(\mathbf{x}) = \mathrm{sign} \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$

AdaBoost – training data weighting

Constructing $\mathcal{D}_t$

- On each round, the weights of incorrectly classified instances are increased so that the algorithm is forced to focus on the hard training examples.
- $\mathcal{D}_1(i) = 1/n \quad \text{for } i = 1, \dots, n$
- given $\mathcal{D}_t$ and h_t (i.e. update $\mathcal{D}_t$):

$$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i)}{Z_t} \cdot \begin{cases} e^{-\alpha_t} & \text{if} \quad y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if} \quad y_i \neq h_t(x_i) \end{cases} = \frac{\mathcal{D}_t(i)}{Z_t} \, \mathrm{e}^{-\alpha_t y_i h_t(x_i)},$$

where Z_t is normalization constant $Z_t = \sum_i \mathcal{D}_t(i) \, \mathrm{e}^{-\alpha_t y_i h_t(x_i)}$

- α_t measures the importance that is assigned to h_t

As the iterations proceed, examples that are difficult to classify correctly receive ever-increasing influence

AdaBoost – base learners weighting

Weights of the base learners α_t

- $error_{\mathcal{D}_t}(h_t) < \frac{1}{2} \Rightarrow \alpha_t > 0$
- the smaller the error, the bigger the weight of the (weak) base learner
- the bigger the weight, the more impact on the (strong) resulting classifier

$$error_{\mathcal{D}_t}(h_1) < error_{\mathcal{D}_t}(h_2) \Longrightarrow \alpha_1 > \alpha_2$$

- $\mathcal{D}_{t+1} = \dfrac{1}{Z_t}\mathcal{D}_t\, \mathrm{e}^{-\alpha_t y_i h_t(\mathbf{x}_i)}$
  The weights of correctly classified instances are reduced while weights of misclassified instances are increased.

AdaBoost.M1 — multiclass problem

Multiclass problem – generalization of the two-class case

- Assume classification task where $Y = \{y_1, \dots, y_k\}$

$$h_t : X \to Y,$$

$$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i)}{Z_t} \cdot \begin{cases} e^{-\alpha_t} & \text{if} \quad y_i = h_t(\mathbf{x_i}) \\ e^{\alpha_t} & \text{if} \quad y_i \neq h_t(\mathbf{x_i}) \end{cases}$$

$$h_{final}(\mathbf{x}) = argmax_{y \in Y} \sum_{\{t \mid h_t(\mathbf{x}) = y\}} \alpha_t.$$

Summary of examination requirements

- Decision Trees – splitting criteria
- Decision Trees – pruning and overfitting
- Ensembles, bagging, boosting – general principles
- Random Forests
- Boosting with regression trees
- AdaBoost