Ensemble methods

COMP5318 Machine Learning and Data Mining

semester 2, 2024, week 5b

Nguyen Hoang Tran

Based on slides prepared by Irena Koprinska (irena.koprinska@sydney.edu.au**)**

Reference: Tan ch.4.10, Witten ch.12, Géron ch.7

THE UNIVERSITY OF
SYDNEY

Outline

- Motivation for creating ensembles
- Ensemble methods
  - Bagging
  - Boosting – AdaBoost
  - Random Forest

What is an ensemble method?

- An ensemble combines the predictions of multiple classifiers
- The classifiers that are combined are called base classifiers, they are created using the training data
- There are various ways to make a prediction for new examples, e.g. by taking the majority vote of the base classifiers, the weighed vote, etc.
- Ensembles tends to work better than the base classifiers they combine

- Example of an ensemble:
  - 3 base classifiers are trained on the training data, e.g. k nearest neighbor, logistic regression and decision tree
  - To classify a new example, the individual predictions are combined by taking the majority vote

Motivation for ensembles

- When do ensemble methods work?
- Let's consider an example which illustrates how ensembles can improve the performance of a single classifier:
- An ensemble of 25 binary classifiers. Each base classifier has an error rate ε =0.35 on the test set (i.e. accuracy=0.65). To predict the class of a new example, the predictions of the base classifiers are combined by majority vote.

- Case 1: The base classifiers are identical, i.e. make the same mistakes. What will be the error rate of the ensemble on the test set?
  - 0.35

Motivation for ensembles (2)

- Case 2: The base classifiers are independent, i.e. their errors are not correlated. What will be the error rate of the ensemble on the test set?

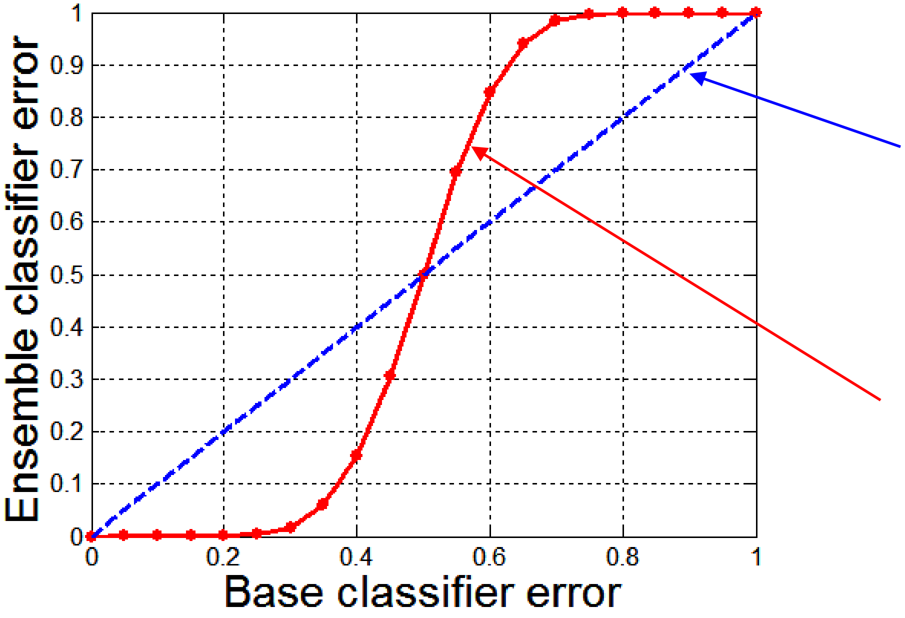
- When will a new example be misclassified? Only if more than half of the base classifiers predict incorrectly.
- It can be shown that the error rate of the ensemble will be:

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

- 0.06<0.35, i.e. the error rate of the ensemble is much lower than the error rate of the base classifiers

Error rate graph – ensemble vs base classifier

- For our example of 25 binary classifiers:

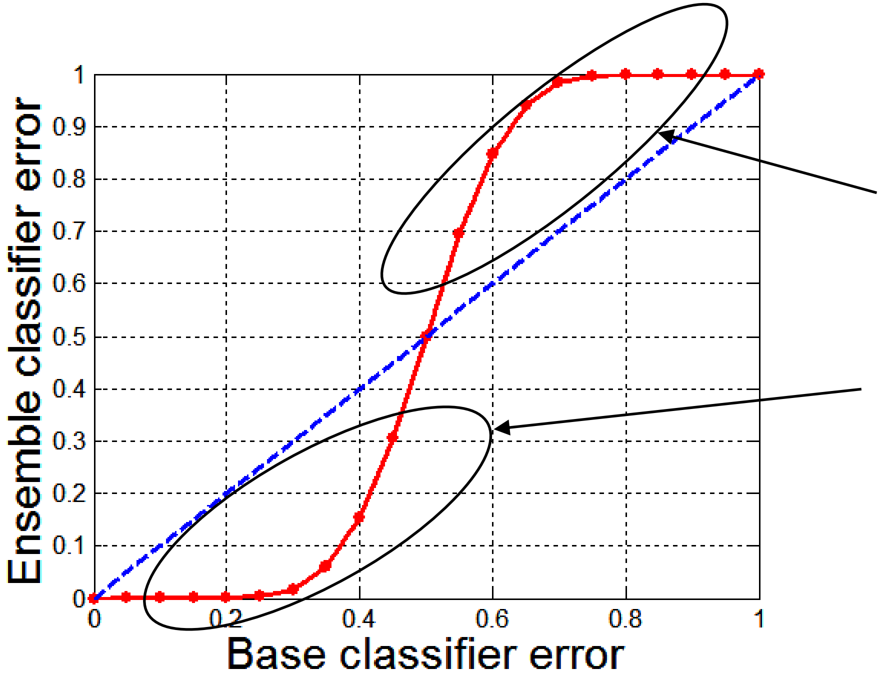


Case 1: base classifiers are identical; ensemble's error = base classifier's error

Case 2: base classifiers are independent; ensemble's error ≠ base classifier's error

Error rate graph – ensemble vs base classifier (2)

- When is the ensemble better and worse than the base classifier?



Ensemble is worse than a base classifier when $\varepsilon > 0.5$

Ensemble is better than a base classifier when $\varepsilon < 0.5$

- The ensemble is better if the error of the base classifier $\varepsilon < 0.5$, i.e. the predictions of the base classifier (which is binary) are better than random guess

When ensembles work well

- Conditions for an ensemble to perform better than a single classifier:
  - The base classifiers should be good enough, i.e. better than a random guessing (ε <0.5 for binary classifiers)
  - The base classifiers are independent of each other

- Independence – in practice:
  - It is not possible to ensure total independence among the base classifiers
  - Good results have been achieved in ensemble methods when the base classifiers are slightly correlated

Methods for constructing ensembles

- Effective ensemble consists of base classifiers that are reasonably correct and also diverse (i.e. independent)
- Methods for creating ensembles focus on generating disagreement among the base classifiers by:
  - Manipulating the training data – creating multiple training sets by resampling the original data according to some sampling distribution and constructing a classifier for each training set (e.g. Bagging and Boosting)
  - Manipulating the attributes – using a subset of input features (e.g. Random Forest and Random Subspace)
  - Manipulating the class labels – will not be covered (e.g. error-correcting output coding)
  - Manipulating the learning algorithm – e.g. building a set of classifiers with different parameters

Manipulating training data - examples

- Creating multiple training sets from the original training set by sampling
- Examples: Bagging and Boosting

Bagging

THE UNIVERSITY OF SYDNEY

Bagging

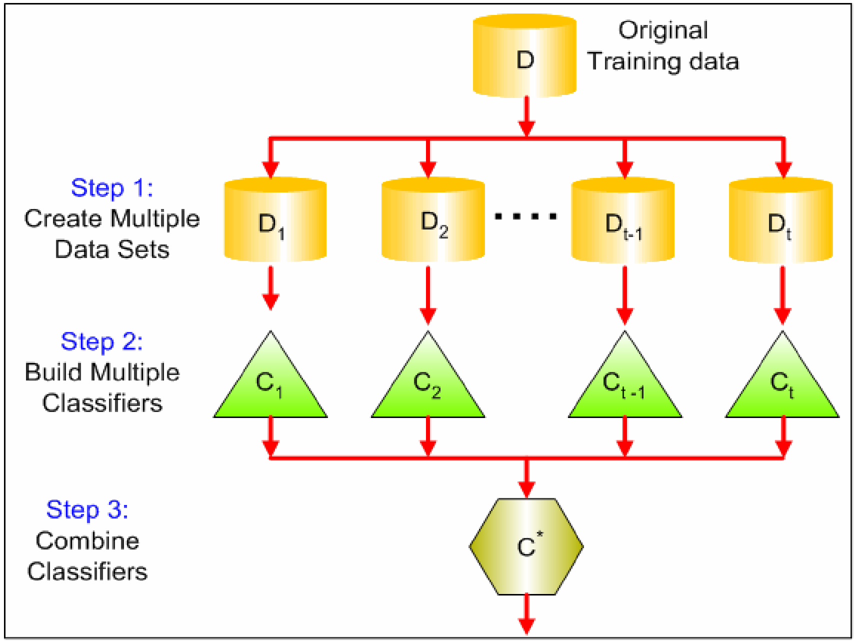
- Bagging is also called bootstrap aggregation
- A bootstrap sample - definition:
  - Given: a dataset D with n example (the original dataset)
  - Bootstrap sample D’ from D: contains also n examples, randomly chosen from D with replacement (i.e. some examples from D will appear more than once in D’, some will not appear at all)
- For n=10, on average 65% of the examples in D will also appear in D’ as it can be shown that the probability to choose an example is 1- $(1-1/n)^n$

Dataset with 10 examples:

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

Bagging

- Create M bootstrap samples
- Use each sample to build a classifier
- To classify a new example: get the predictions of each classifier and combine them with a majority vote
  - i.e. the individual classifiers receive equal weights



Bagging - pseudocode

Model generation

```
Let $n$ be the number of examples in the training data
For each of $M$ iterations:
  Sample $n$ examples with replacement from training data
  Apply the learning algorithm to the sample
  Store the resulting model
```

Classification

```
For each of the $M$ models:
  Predict class of testing example using model
Return class that has been predicted most often (majority vote)
```

Bagging

- Typically performs significantly better than the single classifier and is never substantially worse
- Especially effective for unstable classifiers
- Unstable classifiers: small changes in the training set result in large changes in predictions, e.g. decision trees, neural networks are considered unstable classifiers

- Applying Bagging to regression tasks - the individual predictions are averaged

Boosting

Boosting – main idea

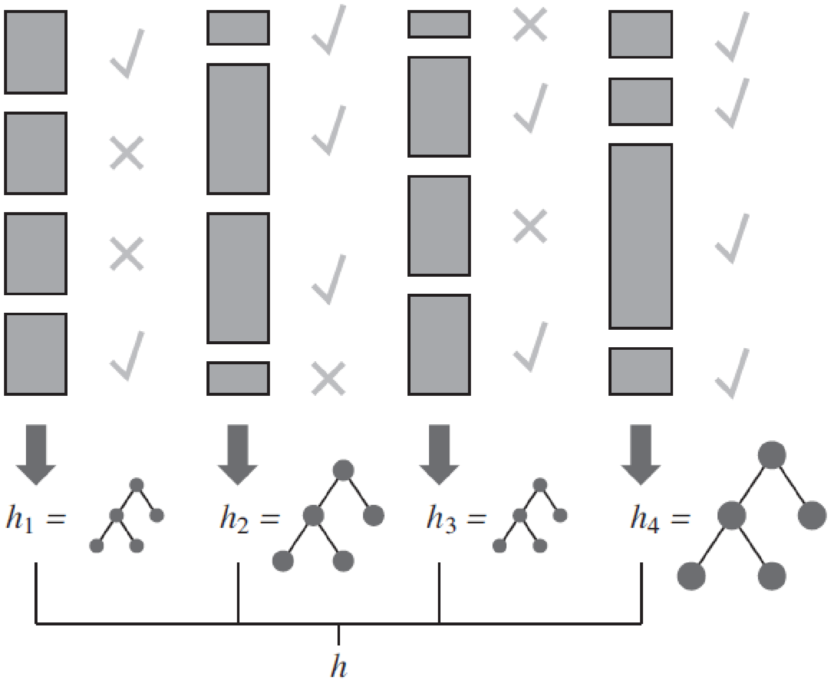
- The most widely used ensemble method
- Idea: Make the classifiers complement each other
- How: The next classifier should be created using examples that were difficult for the previous classifiers

AdaBoost

- Uses a *weighed training set*
- Each training example has an associated weight (≥0)
- The higher the weight, the more difficult the example was to classify by the previous classifiers
- Examples with higher weight will have a higher chance to be selected in the training set for the next classifier

- AdaBoost was proposed by Freund and Schapire in 1996

Diagram



Combining decision trees

- 1 rectangle = 1 training example
- height of the rectangle corresponds to the weight of the example
- √ and X - how the example was classified by the current classifier (correctly or incorrectly)
- size of the tree corresponds to the weight of its prediction in the ensemble

AdaBoost algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current weights $w_t(n)$, by minimizing

     只能是一个 method 比如 选了 logistic 就一直用 logistic

     $$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \quad \text{(the weighted classification error)}$$

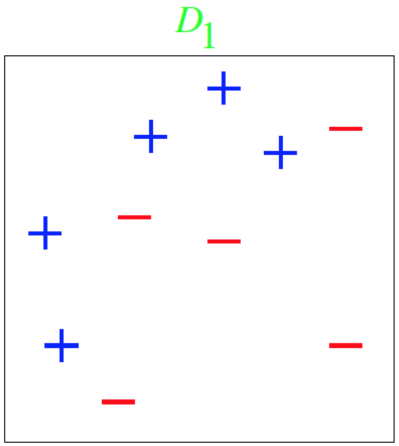
  2. Compute contribution for this classifier: $\beta_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$
  3. Update weights on training points

     $$w_{t+1}(n) \propto w_t(n)e^{-\beta_t y_n h_t(\boldsymbol{x}_n)}$$

     and normalize them such that $\sum_n w_{t+1}(n) = 1$
- Output the final classifier

$$h[\boldsymbol{x}] = \text{sign}\left[\sum_{t=1}^{T}\beta_t h_t(\boldsymbol{x})\right]$$

AdaBoost: Example



- The data points are clearly not linearly separable
- In the beginning, all data points have equal weights (the size of the data markers "+" or "-")
- Base classifier $h(\cdot)$: horizontal or vertical lines ('decision stumps')
  - Depth-1 decision trees, i.e., classify data based on a single attribute.

AdaBoost: Example

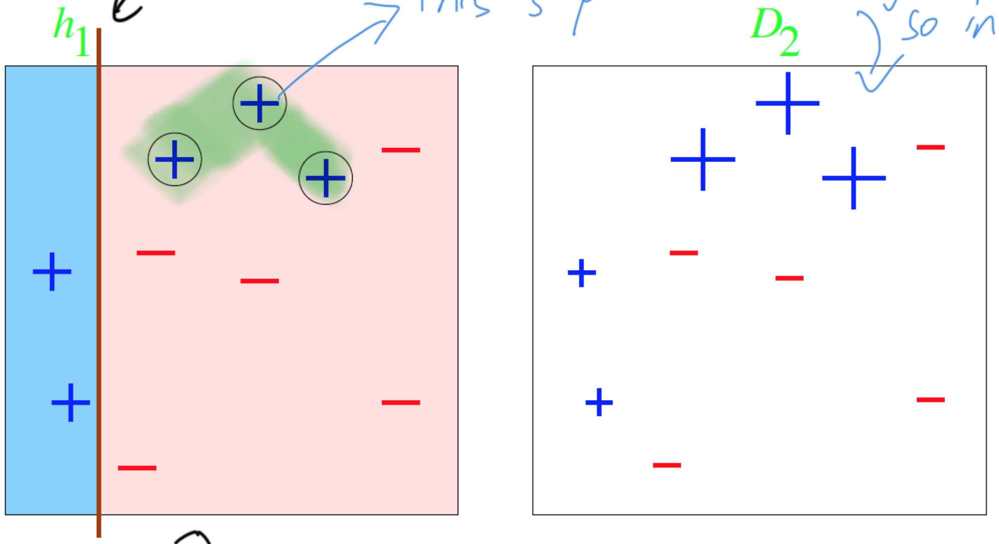
Round 1: t=1

This is our model

all left predict to be '+'

This 3 predict wrongly

so increase weight



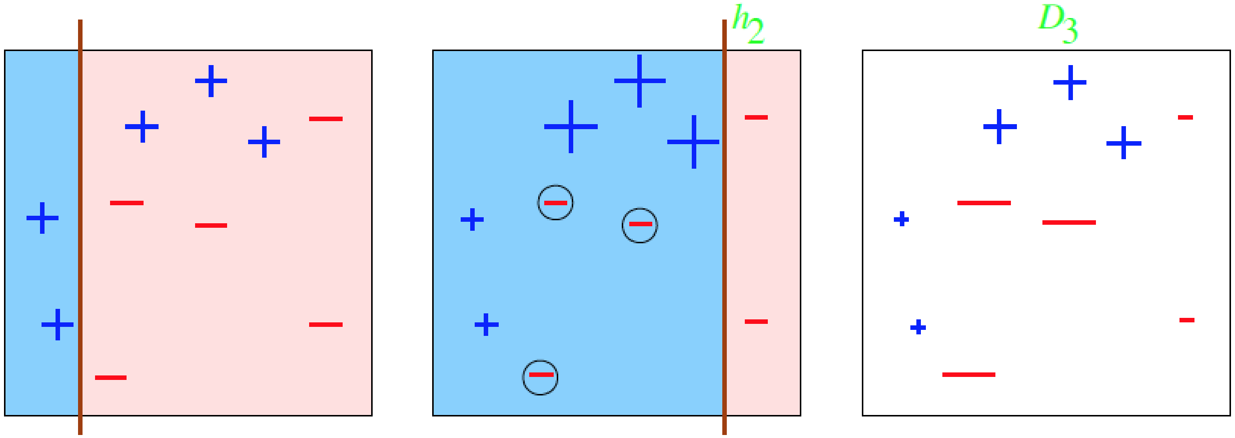
all right predict to be "−"

$$\beta_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$$

- 3 misclassified (with circles): $\epsilon_1 = 0.3 \rightarrow \beta_1 = 0.42$.
- Recompute the weights; the 3 misclassified data points receive larger weights

AdaBoost: Example

t=2



$$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$

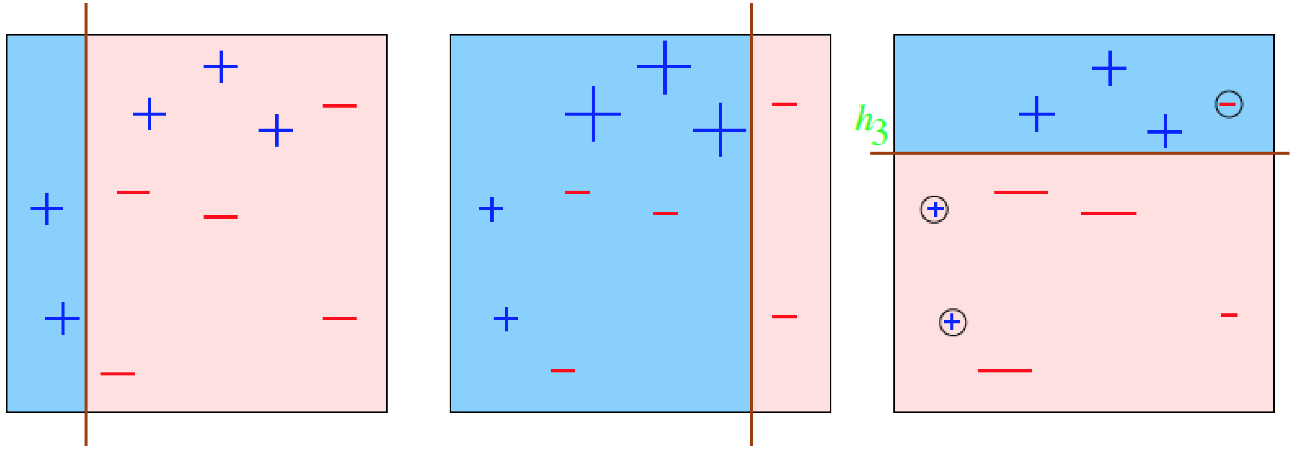
$$\beta_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$$

$$w_{t+1}(n) \propto w_t(n)e^{-\beta_t y_n h_t(\boldsymbol{x}_n)}$$

- 3 misclassified (with circles): $\epsilon_2 = 0.21 \rightarrow \beta_2 = 0.65$.
  $\epsilon_2 < 0.3$ as those 3 misclassified data points have weights < 0.1
- 3 newly misclassified data points get larger weights
- Data points classified correctly in both rounds have small weights

AdaBoost: Example

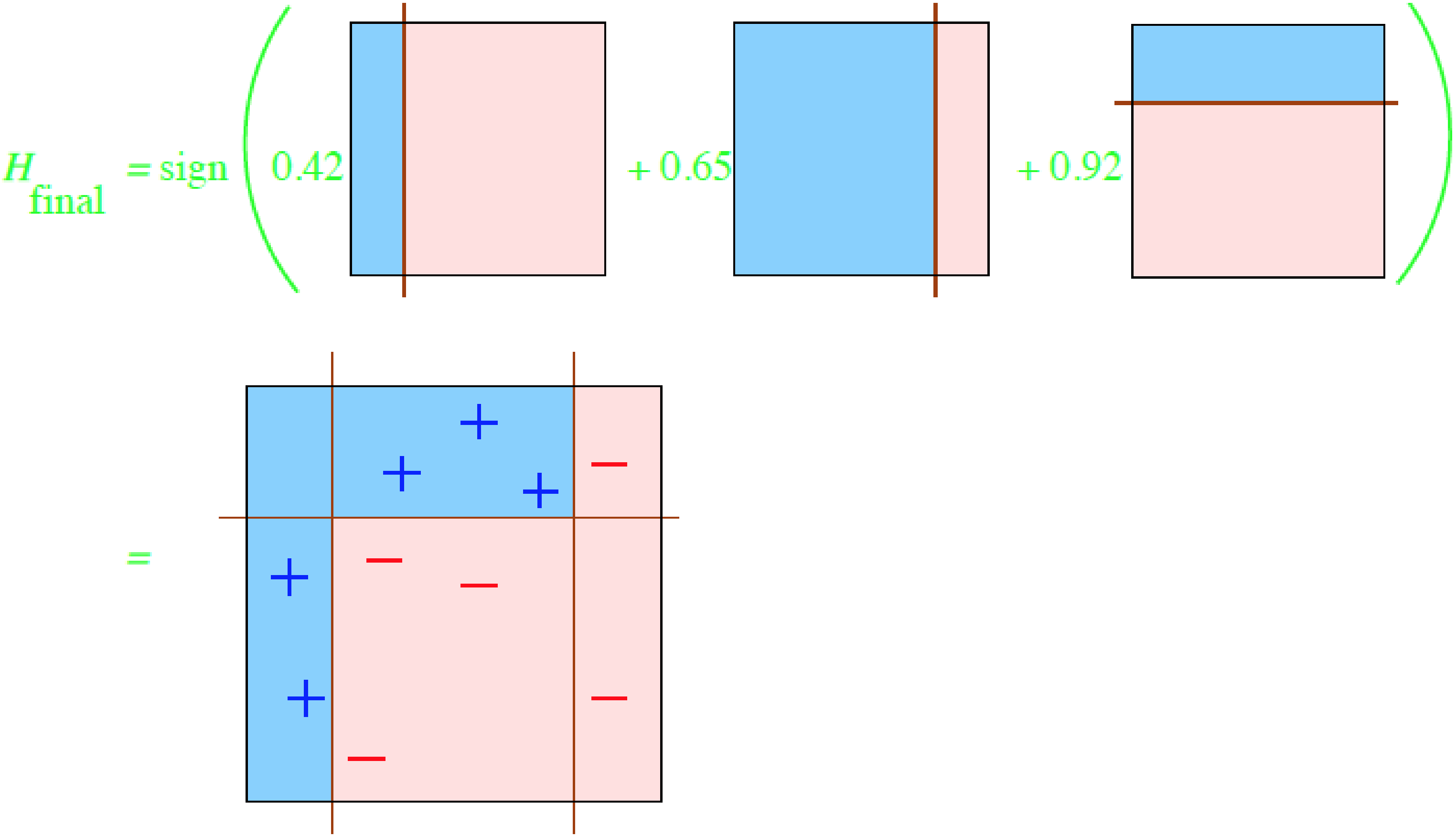
t=3



- 3 misclassified (with circles): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.
- Previously correctly classified data points are now misclassified, hence our error is low. Why?
  - Since they have been consistently classified correctly, this round's mistake will hopefully not have a huge impact on the overall prediction

AdaBoost: Example

Combine 3 Classifier



Why does AdaBoost work?

It minimizes a loss function related to classification error.

Classification loss

- Suppose we want to have a classifier

$$h(\boldsymbol{x}) = \text{sign}[f(\boldsymbol{x})] = \begin{cases} 1 & \text{if } f(\boldsymbol{x}) > 0 \\ -1 & \text{if } f(\boldsymbol{x}) < 0 \end{cases}$$

- One seemingly natural loss function is 0-1 loss:

$$\ell(h(\boldsymbol{x}), y) = \begin{cases} 0 & \text{if } yf(\boldsymbol{x}) > 0 \\ 1 & \text{if } yf(\boldsymbol{x}) < 0 \end{cases}$$

Namely, the function $f(\boldsymbol{x})$ and the target label y should have the same sign to avoid a loss of 1.

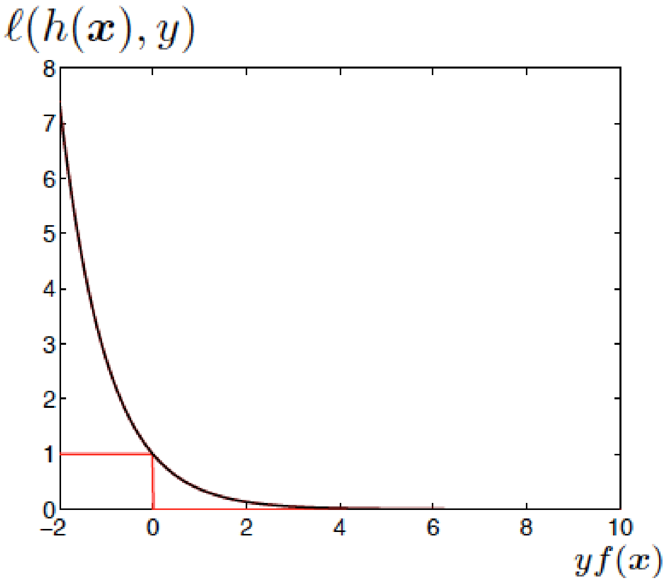
Surrogate function of 0-1 loss?

$0 - 1$ loss function $\ell(h(\mathbf{x}), y)$ is non-convex and difficult to optimize.

We can instead use a surrogate loss – what are examples?

Exponential Loss

$$\ell^{\mathrm{EXP}}(h(\mathbf{x}), y) = e^{-yf(\mathbf{x})}$$



How to choose the *t*-classifier ?

Suppose a classifier $f_{t-1}(\boldsymbol{x})$, and want to add a weak learner $h_t(\boldsymbol{x})$

$$f(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t h_t(\boldsymbol{x})$$

Note: $h_t(\cdot)$ outputs -1 or 1, as does $\text{sign}\left[f_{t-1}(\cdot)\right]$

How can we 'optimally' choose $h_t(\boldsymbol{x})$ and combination coefficient β_t?

Adaboost greedily *minimizes the exponential loss function*!

$$\begin{aligned}(h_t^*(\boldsymbol{x}), \beta_t^*) &= \text{argmin}_{(h_t(\boldsymbol{x}),\beta_t)} \sum_n e^{-y_n f(\boldsymbol{x}_n)} \\ &= \text{argmin}_{(h_t(\boldsymbol{x}),\beta_t)} \sum_n e^{-y_n[f_{t-1}(\boldsymbol{x}_n)+\beta_t h_t(\boldsymbol{x}_n)]} \\ &= \text{argmin}_{(h_t(\boldsymbol{x}),\beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)}\end{aligned}$$

where we have used $w_t(n)$ as a shorthand for $e^{-y_n f_{t-1}(\boldsymbol{x}_n)}$

The New Classifier

We can decompose the *weighted* loss function into two parts

$$
\begin{aligned}
&\sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\
&= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + \sum_n w_t(n) e^{-\beta_t} \mathbb{I}[y_n = h_t(\boldsymbol{x}_n)] \\
&= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + \sum_n w_t(n) e^{-\beta_t} (1 - \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]) \\
&= (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + e^{-\beta_t} \sum_n w_t(n)
\end{aligned}
$$

We have used the following properties to derive the above

- $y_n h_t(\boldsymbol{x}_n)$ is either 1 or -1 as $h_t(\boldsymbol{x}_n)$ is the output of a binary classifier
- The indicator function $\mathbb{I}[y_n = h_t(\boldsymbol{x}_n)]$ is either 0 or 1, so it equals $1 - \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$

Finding the Optimal Weak Learner

Summary

$$(h_t^*(\mathbf{x}), \beta_t^*) = \operatorname{argmin}_{(h_t(\mathbf{x}),\beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(\mathbf{x}_n)}$$

$$= \operatorname{argmin}_{(h_t(\mathbf{x}),\beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\mathbf{x}_n)]$$

$$+ e^{-\beta_t} \sum_n w_t(n)$$

What term(s) must we optimize to choose $h_t(\mathbf{x}_n)$?

$$h_t^*(\mathbf{x}) = \operatorname{argmin}_{h_t(\mathbf{x})} \epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\mathbf{x}_n)]$$

Minimize weighted classification error as noted in step 1 of Adaboost!

Finding the Optimal Weak Learner

Summary

$$\begin{aligned}(h_t^*(\boldsymbol{x}), \beta_t^*) &= \text{argmin}_{(h_t(\boldsymbol{x}),\beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\ &= \text{argmin}_{(h_t(\boldsymbol{x}),\beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \\ &\qquad + e^{-\beta_t} \sum_n w_t(n)\end{aligned}$$

What term(s) must we optimize?

We need to minimize the entire objective function with respect to β_t!

We can take the derivative with respect to β_t, set it to zero, and solve for β_t. After some calculation and using $\sum_n w_t(n) = 1$...

$$\beta_t^* = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

which is precisely step 2 of Adaboost! (*Exercise – verify the solution*)

Updating the Weights

Once we find the optimal weak learner we can update our classifier:

$$f(\mathbf{x}) = f_{t-1}(\mathbf{x}) + \beta_t^* h_t^*(\mathbf{x})$$

We then need to compute the weights for the above classifier as:

$$w_{t+1}(n) \propto e^{-y_n f(\mathbf{x}_n)} = e^{-y_n[f_{t-1}(\mathbf{x})+\beta_t^* h_t^*(\mathbf{x}_n)]}$$

$$= w_t(n)e^{-y_n\beta_t^* h_t^*(\mathbf{x}_n)} = \begin{cases} w_t(n)e^{\beta_t^*} & \text{if } y_n \neq h_t^*(\mathbf{x}_n) \\ w_t(n)e^{-\beta_t^*} & \text{if } y_n = h_t^*(\mathbf{x}_n) \end{cases}$$

Intuition Misclassified data points will get their weights increased, while correctly classified data points will get their weight decreased

Meta Algorithm

Note that the AdaBoost algorithm itself never specifies how we would get $h_t^*(\mathbf{x})$ as long as it minimizes the weighted classification error

$$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t^*(\mathbf{x}_n)]$$

In this aspect, the AdaBoost algorithm is a meta-algorithm and can be used with any type of classifier

Other Boosting Algorithms

- AdaBoost is by far the most popular boosting algorithm.
- Gradient boosting generalizes AdaBoost by substituting another (smooth) loss function for exponential loss.
    - Squared loss, logistic loss, ...
    - Choose the candidate learner that greedily minimizes the error (mathematically, it should maximize the gradient of the residuals calculated with this loss function).
    - Reduce overfitting by constraining the candidate learner (e.g., computing the residuals only over a sample of points).
- LogitBoost minimizes the logistic loss instead of exponential loss.
- Gentle AdaBoost bounds the step size (β) of each learner update.

Bagging and Boosting - comparison

- Similarities
  - Use voting (for classification) and averaging (for prediction) to combine the outputs of the individual learners
  - Combine classifiers of the same type, typically trees – e.g. decision stumps or decision trees
- Differences
  - Creating base classifiers:
    - Bagging – separately
    - Boosting – iteratively – the new ones are encouraged to become experts for the misclassified examples by the previous base learners (complementary expertise)
  - Combination method
    - Bagging – equal weighs to all base learners
    - Boosting (AdaBoost) – different weights based on the performance on training data

Random Forest

Creating ensembles by manipulating the attributes

- Each base classifier uses only a subset of the features
- E.g. training data with K features, create an ensemble of M classifiers each using a smaller number of features L (L<K)

  1) Create feature subsets by random selection from the original feature set => creating multiple versions of the training data, each containing only the selected features

- 2) Build a classifier for each version of the training data
- 4) Combine predictions with majority vote
- Example: Random Forest
  - Combines decision trees
  - Uses 1) bagging + 2) subset of features (during decision tree building, when selecting the most important attribute)

Random Forest algorithm

n - number of training examples, *m* – number of all features, *k* – number of features to be used by each ensemble member (k<m), *M* – number of ensemble members

Model generation:

For each of *M* iteration

1. Bagging – generate a bootstrap sample

   Sample *n* instances with replacement from training data

2. Random feature selection for selecting the best attribute

   Grow decision tree without pruning. At each step select the best feature to split on by considering only *k* randomly selected features and calculating information gain

Classification:

Apply the new example to each of the *t* decision trees starting from the root. Assign it to the class corresponding to the leaf. Combine the decisions of the individual trees by majority voting.

Comments on Random Forests

- Performance depends on
  - Accuracy of the individual trees (strength of the trees)
  - Correlation between the trees
- Ideally: accurate individual trees but less correlated
- Bagging and random feature selection are used to generate diversity and reduce the correlation between the trees
- As the number of features k increases, both the strength and correlation increase
- Random Forest typically outperforms a single decision tree
- Robust to overfitting
- Fast as only a subset of the features are considered

Ensembles - summary

- Ensembles combine the predictions of several classifiers
- They work when the individual classifiers are accurate and diverse
- Diversity is generated by manipulating the
  - training data (Bagging, Boosting)
  - attributes (Random Forest = bagging + random selection of attributes)
  - learning algorithm
- Some ensembles combine classifiers of the same type, some not
- Most ensembles use a majority vote to make predictions on new data, others used a weighted vote
- Ensembles have shown excellent performance – often the winning solution in ML competitions is an ensemble method