

# CS-E4710 Machine Learning: Supervised Methods

## Lecture 11: Multi-class Classification

---

Juho Rousu

November 21, 2023

Department of Computer Science  
Aalto University

# Multi-class classification

- Given a training data set  $\{(\mathbf{x}_i, y_i)\}_{i=1}^m, (\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$
- Outputs belongs to a set of possible classes or labels:  
 $y_i \in \mathcal{Y} = \{1, 2, \dots, k\}$
- In **multi-class classification**, one of the labels is considered to be the correct label, the other ones incorrect<sup>1</sup>
- In **multi-label classification**, several of the labels can be correct for a given input  $x_i$ , the output space will be  $\mathcal{Y} = \{-1, +1\}^k$ , each output  $\mathbf{y}_i$  is a  $k$ -dimensional vector
- In both cases, we aim learning a function

$$f : X \mapsto \mathcal{Y},$$

for predicting the outputs

---

<sup>1</sup>Mohri et al. book calls this case **mono-label** multi-class classification, but that is not standard vocabulary

# Multi-class classification

Two basic strategies to solve the problem:

1. Aggregated methods using multiple binary classifiers:
  - One-versus-all approach (OVA) : Separate each class from all the others
  - One-versus-one or all-pairs approach (OVO): Separate each class pair from each other
  - Error-correcting output code approach (ECOC): Represent each class with a binary code vector and predict the bits of the vector
2. Standalone models: learning to predict multiple classes directly
  - Multiclass SVM
  - Multiclass Boosting

# One-versus-All Classification

---

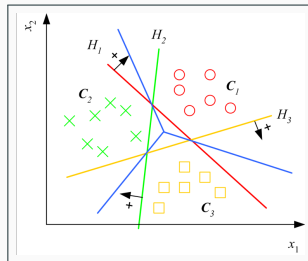
# One-versus-All Classification

- Given a training data set  $\{(\mathbf{x}_i, y_i)\}_{i=1}^m, (\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$
- If we have  $k > 2$  classes we will train  $k$  binary hypotheses  $h_1, \dots, h_k$ ,  
 $h_\ell : \mathcal{X} \mapsto \{+1, -1\}$
- For training the  $\ell$ th hypothesis, new binary labels, called the **surrogate labels** are computed  $\tilde{y}_i^{(\ell)} = \begin{cases} +1, & \text{if } y_i = \ell, \\ -1, & \text{if } y_i \neq \ell \end{cases}$
- A binary classifier is trained to predict the surrogate labels
- The hypothesis class for the binary classifiers is not restricted: we can use any that is deemed suitable

# Geometry of the linear OVA model: separable case

An example with three classes in two-dimensional space (green crosses, red circles and yellow boxes)

- Linear classifiers  $\mathbf{w}_\ell^T \mathbf{x} + w_{\ell 0}$  are used as the predictors (Note that the bias terms  $w_{\ell 0}$  are written out explicitly)
- In the linearly separable case, there is a hyperplane  $H_\ell : \mathbf{w}_\ell^T \mathbf{x} + w_{\ell 0} = 0$  so that all  $\mathbf{x} \in C_i$  lie in the positive halfspace and all other points lie in the negative halfspace
- $h_\ell(\mathbf{x}) = \text{sgn}(\mathbf{w}_\ell^T \mathbf{x} + w_{\ell 0}) = +1$  for a single class  $\ell$



- In general, there may be more than one class  $\ell$  for which  $h_\ell(\mathbf{x}) = +1$
- Some arbitrary tie-breaking could be used, e.g. predict the class with the smallest index  $\ell$
- Better results can be obtained if the hypotheses also provide some real-valued score  $f_\ell(\mathbf{x}) \in \mathbb{R}$  (confidence, margin, etc.) for the label to be  $\ell$ .
- In that case, we can choose the label with the highest score

$$h(\mathbf{x}) = \operatorname{argmax}_\ell f_\ell(\mathbf{x})$$

- With linear models  $h_\ell(\mathbf{x}) = \operatorname{sgn}(\mathbf{w}_\ell^T \mathbf{x})$  using the margin of the example is a natural choice  $f_\ell(\mathbf{x}) = \mathbf{w}_\ell^T \mathbf{x}$

# OVA training pseudo-code

**Input:** Dataset  $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m, \mathbf{x}_i \in X, y_i \in \mathcal{Y} = \{1, \dots, k\}$

**Output:** Multiclass hypothesis  $h : X \mapsto \mathcal{Y}$

**for**  $\ell \in \{1, \dots, k\}$  **do**

    Generate training dataset with surrogate labels:  $\{(\mathbf{x}_i, \tilde{y}_i^{(\ell)})\}_{i=1}^m$

    Train a binary hypothesis  $h_\ell : X \mapsto \{-1, +1\}$

    Let  $f_\ell(\mathbf{x})$  be the score for  $\mathbf{x}$  given by the model  $h_\ell$

**end for**

$h(\mathbf{x}) = \operatorname{argmax}_\ell f_\ell(\mathbf{x})$

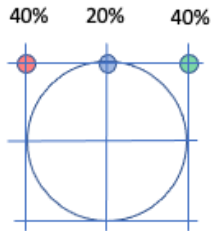


# Pros and cons of the OVA approach

- OVA classification is simple to implement and therefore popular
- Training is relatively efficient with  $O(kt)$  time where  $t$  is the time to train a single binary classifier, if  $k$  is not too large
- The method may suffer from the **class imbalance** of the training sets for a given class  $\ell$ : there may be a low number of positive examples and a high number of negative examples per class
- In general OVA approach suffers from a **calibration problem**: the scores  $f_\ell(\mathbf{x})$  returned by the individual classifiers may not be comparable
- It does not always produce the optimal empirical error rate for the dataset (example below)

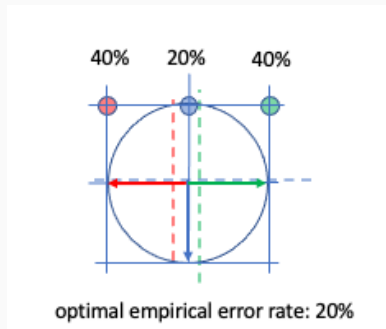
## Example: sub-optimality of OVA classification

- Consider a dataset with 3 classes (red, blue, green), with class frequencies 40%, 20%, 40%, respectively
- The classes are concentrated in distinct clusters centered at  $(-1, 1)$ ,  $(0, 1)$ , and  $(1, 1)$ , respectively



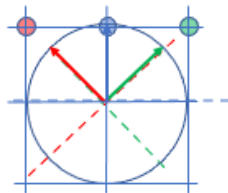
## Example: sub-optimality of OVA classification

- Optimal linear classifiers can separate red and green classes from the other two
- However, the optimal linear classifier for the blue class classifies all data as negative
- Combination of the three classifiers will predict (incorrectly) blue cluster to be either red or green, depending on tie breaking
- Empirical error rate is 20%



## Example: sub-optimality of OVA classification

- However, the three classes are separable by three hyperplanes  
 $f_\ell(\mathbf{x}) = \mathbf{w}_\ell^T \mathbf{x}$ ,  
 $\mathbf{w}_{red} = (-1/\sqrt{2}, 1/\sqrt{2})$ ,  $\mathbf{w}_{blue} = (0, 1)$   
and  $\mathbf{w}_{green} = (1/\sqrt{2}, 1/\sqrt{2})$  using the  
rule  $h(\mathbf{x}) = \operatorname{argmax}_\ell f_\ell(\mathbf{x})$
- Note that the hyperplane  $\mathbf{w}_{blue}$  is not  
a good classifier as a independent  
model, its empirical error rate is 80%!
  - Thus we see that independent training of the binary hypotheses  
loses information and may result in sub-optimal error rates.



optimal empirical error rate: 0%

# One-versus-One Classification

---

# One-versus-one approach

- An alternative is one-versus-one (OVO) or all-pairs approach
- In OVO classification, we divide a multiclass problem into a set of  $k(k-1)/2$  binary classification problems, one for each pair of classes  $(\ell, \ell'), 1 \leq \ell < \ell' \leq k$
- This entails generating a new training set consisting of examples of the pair of classes  $(\ell, \ell')$  and generating a surrogate label

$$\tilde{y}^{\ell, \ell'} = \begin{cases} +1 & \text{if } y = \ell \\ -1 & \text{if } y = \ell' \end{cases}$$

- For each class pair, a binary hypothesis  $h_{\ell, \ell'}(\mathbf{x}) : X \mapsto \{-1, +1\}$  is trained using the generated training set

- In predicting, for each class  $\ell$  we have  $k - 1$  pairwise hypotheses, one for each class containing  $\ell$  ( $h_{\ell, \ell'}$  and  $h_{\ell', \ell}$ , for all  $\ell' \neq \ell$ )
- In the ideal case, all of the  $k - 1$  hypotheses involving class  $\ell$  would predict class  $\ell$
- In practice this may not happen, we might have for some classes  $\ell', \ell''$ 
  - $h_{\ell, \ell'}(\mathbf{x}) = +1$  - predicting class  $\ell$  for  $\mathbf{x}$
  - $h_{\ell, \ell''}(\mathbf{x}) = -1$  - predicting class  $\ell''$  for  $\mathbf{x}$
- We need to resolve these discrepancies

A voting approach can be used:

- We count for each input  $\mathbf{x}$ , how many pairwise hypotheses predict class  $\ell$  (the votes)

$$h(\mathbf{x}) = \operatorname{argmax}_{\ell} \sum_{\ell' < \ell} \mathbf{1}_{\{h_{\ell\ell'}(\mathbf{x})=+1\}} + \sum_{\ell' > \ell} \mathbf{1}_{\{h_{\ell'\ell}(\mathbf{x})=-1\}}$$

- Ties can occur with several classes receiving the same number of votes, we can break them arbitrarily (e.g. predicting the smallest index  $\ell$ )



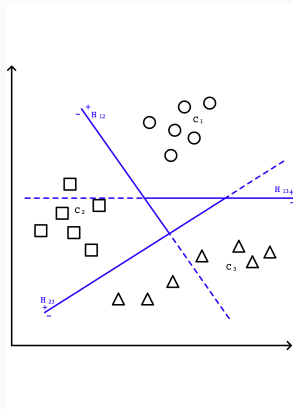
# Geometry of linear OVO classifier

An example with three classes and linear predictors  $\mathbf{w}_{\ell'\ell}^T \mathbf{x} + b_{\ell\ell'}$  for each class pair (Again the bias terms  $b_{\ell\ell'}$  written out explicitly)

- A class  $\ell$  is predicted within a region of the feature space where the number of votes for the class equal the maximum
- Geometrically, the region is defined by intersection of half-spaces

$$H_{\ell,\ell'} = \{\mathbf{x} | \mathbf{w}_{\ell\ell'}^T \mathbf{x} + b_{\ell\ell'} > 0\}, \text{ for all } \ell < \ell'$$

$$H_{\ell',\ell} = \{\mathbf{x} | \mathbf{w}_{\ell'\ell}^T \mathbf{x} + b_{\ell'\ell} < 0\}, \text{ for all } \ell > \ell'$$



- The triangle in the middle represents the region where all classes have one vote

## Pros and cons of the OVO model

- Compared to OVA, we are training many more binary classifiers:  $O(k^2)$  compared to  $O(k)$
- However, the training sets are smaller since they only contain examples of two classes at a time:
  - Faster to train
  - Increased chance of overfitting
- The OVO training sets are less likely to be imbalanced than in OVA
- Better theoretical justification through the voting approach

# Generalization performance of OVO models

- OVO model has some theoretical justification through viewing it as a kind of **majority voting ensemble**
- Assume that the pairwise hypotheses have generalization error of at most  $r$
- Now if an example  $\mathbf{x}$  with true class  $\ell'$  is misclassified by the OVO model, there must be at least one pairwise hypothesis  $h_{\ell\ell'}$  or  $h_{\ell'\ell}$  that makes an error on  $\mathbf{x}$
- The probability of this event is at most

$$\sum_{\ell < \ell'} P("h_{\ell\ell'} \text{ makes an error}") + \sum_{\ell' < \ell} P("h_{\ell'\ell} \text{ makes an error}") \leq r(k-1)$$

- Thus if the pairwise classifiers are accurate enough, the risk of the multiclass classifier can be kept relatively low

# Error-correcting codes

---

# Error-correcting codes (ECOC)

- Error-correcting output codes (ECOC) is a general methods for reducing multi-class problems to binary classification
- In the ECOC approach, each class  $\ell$  is allocated a codeword  $m_\ell$  of length  $c > 1$
- In the simplest case a binary vector can be used  $m_\ell \in \{-1, +1\}^c$
- The code words of all  $k$  classes together form a matrix  $M \in \{-1, +1\}^{k \times c}$

		codes					
classes		1	2	3	4	5	6
	1	-1	-1	-1	+1	-1	-1
	2	+1	-1	-1	-1	-1	-1
	3	-1	+1	+1	-1	+1	-1
	4	+1	+1	-1	-1	-1	-1
	5	+1	+1	-1	-1	+1	-1
	6	-1	-1	+1	+1	-1	+1
	7	-1	-1	+1	-1	-1	-1
	8	-1	+1	-1	+1	-1	-1

$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$
-1	+1	+1	-1	+1	+1

new example  $x$

# Error-correcting codes

- Given the codeword matrix, a binary classifier  $f_j : X \mapsto \{-1, +1\}$  is learned for each column  $j = 1, \dots, c$  of the codeword matrix
- For the classifier of column  $j$ , the training input  $x_i$  is relabeled with surrogate label  $\tilde{y}_i^{(j)} = m_{y_i j}$ , read from the row of the codeword matrix corresponding to the codeword of class  $y_i$ .
- The prediction of the ECOC model is taken as the class  $\ell$  with the fewest wrongly predicted columns of the keyword:

$$h(\mathbf{x}) = \operatorname{argmin}_{\ell=1}^k \sum_{j=1}^c \mathbf{1}_{f_j(\mathbf{x}) \neq m_{\ell j}}$$

	1	2	3	4	5	6
1	-1	-1	-1	+1	-1	-1
2	+1	-1	-1	-1	-1	-1
3	-1	+1	+1	-1	+1	-1
4	+1	+1	-1	-1	-1	-1
5	+1	+1	-1	-1	+1	-1
6	-1	-1	+1	+1	-1	+1
7	-1	-1	+1	-1	-1	-1
8	-1	+1	-1	+1	-1	-1

$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$
-1	+1	+1	-1	+1	+1

new example  $x$

# How to generate the codewords?

How to generate the codewords

- Deterministic code: decide on the length  $c$  and choose binary vectors for each class so that the between class Hamming distance is as large as possible
- Random code: draw code words randomly
- Use domain knowledge: each column could be a feature describing the class

		codes					
classes		1	2	3	4	5	6
	1	-1	-1	-1	+1	-1	-1
	2	+1	-1	-1	-1	-1	-1
	3	-1	+1	+1	-1	+1	-1
	4	+1	+1	-1	-1	-1	-1
	5	+1	+1	-1	-1	+1	-1
	6	-1	-1	+1	+1	-1	+1
	7	-1	-1	+1	-1	-1	-1
	8	-1	+1	-1	+1	-1	-1

$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$
-1	+1	+1	-1	+1	+1

new example  $x$

# Why does ECOC work?

- The prediction of the ECOC model can be seen as correcting incorrectly predicted bits of the codeword
- The corrected codeword is then the one in the codebook (matrix  $M$ ) that has the smallest Hamming distance to the predicted codeword
- If the between class Hamming distance of the codewords is at least  $d$ , then upto  $\lfloor \frac{d-1}{2} \rfloor$  one bit errors can be corrected
- Another explanation comes from ensemble learning: model averaging between diverse classifiers  $f_j$  happens by minimizing the Hamming distance between codewords

		codes					
classes		1	2	3	4	5	6
	1	-1	-1	-1	+1	-1	-1
	2	+1	-1	-1	-1	-1	-1
	3	-1	+1	+1	-1	+1	-1
	4	+1	+1	-1	-1	-1	-1
	5	+1	+1	-1	-1	+1	-1
	6	-1	-1	+1	+1	-1	+1
	7	-1	-1	+1	-1	-1	-1
	8	-1	+1	-1	+1	-1	-1

$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$
-1	+1	+1	-1	+1	+1

new example  $x$



## **Standalone multi-class classifiers**

---

- Models that directly aim to minimize a multi-class loss function may give better predictive performance than the approaches based on aggregating binary classifiers
- Defining a combined model may be more efficient to train
- Multiclass models
  - Multiclass SVM
  - Multiclass boosting

# Multi-class SVM

---

# Multi-class SVM

- Multi-class SVM learns  $k$  hyperplanes  $f_\ell(\mathbf{x}) = \mathbf{w}_\ell^T \mathbf{x} = 0$  simultaneously
- The predicted class is the class with the highest score

$$h(\mathbf{x}) = \operatorname{argmax}_\ell f_\ell(\mathbf{x})$$

- The ideal objective would be to minimize the zero-one loss

$$L(h(\mathbf{x}), y_i) = \frac{1}{m} \sum_{i=1}^m \mathbf{1}_{h(\mathbf{x}) \neq y_i}$$

but like in binary classification, this is non-convex and NP-hard to optimize

# Multi-class SVM

- Instead, multi-class SVM focuses on the score differences between pairs of classes

$$f_{\ell}(\mathbf{x}) - f_{\ell'}(\mathbf{x}) = \mathbf{w}_{\ell}^T \mathbf{x}_i - \mathbf{w}_{\ell'}^T \mathbf{x}_i$$

- In particular, the margins between the correct class  $y_i$  and all the incorrect classes  $\ell \neq y_i$  are optimized
- We aim the score of the correct class to be higher than all the other classes by a margin (of 1)

$$\mathbf{w}_{y_i}^T \mathbf{x}_i - \mathbf{w}_{\ell}^T \mathbf{x}_i \geq 1 - \xi_i, \text{ for all } \ell \neq y_i$$

- Above, slack  $\xi_i \geq 0$  is used in the analogous way to binary SVMs to allow some examples to not have the required margin

- Multi-class SVM has  $k$  weight vectors  $\mathbf{w}_1, \dots, \mathbf{w}_k$  to control
- This is achieved by a regularizer that computes the sum of norms:  
 $\sum_{\ell=1}^k \|\mathbf{w}_\ell\|_2^2$
- The regularizer is motivated by controlling the empirical Rademacher complexity  $\hat{\mathcal{R}}(H)$  of the hypothesis class  $H$  of multi-class SVMs:

$$\hat{\mathcal{R}}(H) \leq \sqrt{\frac{r^2 \Lambda^2}{m}},$$

where  $\sum_{\ell=1}^k \|\mathbf{w}_\ell\|_2^2 \leq \Lambda^2$  and  $\|\mathbf{x}_i\|_2^2 \leq r^2$  for all  $i = 1, \dots, m$

- Thus, minimizing the sum of norms aids achieving good generalization

# Multi-class SVM

The Multi-class SVM optimization problem can be written as follows:

$$\begin{aligned} \min_{\mathbf{W}, \xi} \quad & \frac{1}{2} \sum_{\ell=1}^k \|\mathbf{w}_{\ell}\|^2 + C \sum_{i=1}^m \xi_i \\ \text{s.t.} \quad & \mathbf{w}_{y_i}^T \mathbf{x}_i - \mathbf{w}_{\ell}^T \mathbf{x}_i \geq 1 - \xi_i, \\ & \text{for all } \ell \neq y_i \\ & \text{and for all } i = 1, \dots, m \\ & \xi_i \geq 0, i = 1, \dots, m \end{aligned}$$

- Above, we have denoted by  $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_k]$  a matrix that contains the weight vectors as columns
- The problem has quadratic objective and linear constraints
- Thus with small to medium sized data, it can be solved by Quadratic Programming (QP) solvers
- For large data, stochastic gradient approaches can be used

# Multi-class Hinge loss

- Rewrite the constraint

$$\mathbf{w}_{y_i}^T \mathbf{x}_i - \mathbf{w}_\ell^T \mathbf{x}_i \geq 1 - \xi_i, \text{ for all } \ell \neq y_i, \xi_i \geq 0 \Leftrightarrow$$

$$\mathbf{w}_{y_i}^T \mathbf{x}_i - \max_{\ell \neq y_i} \mathbf{w}_\ell^T \mathbf{x}_i \geq 1 - \xi_i, \xi_i \geq 0 \Leftrightarrow$$

$$\xi_i \geq 1 - [\mathbf{w}_{y_i}^T \mathbf{x}_i - \max_{\ell \neq y_i} \mathbf{w}_\ell^T \mathbf{x}_i], \xi_i \geq 0$$

- Minimizing  $\xi_i$  corresponds to minimizing the **multi-class Hinge loss**

$$L_{MCHinge}(\mathbf{W}\mathbf{x}_i, y_i) = \max\{0, 1 - [\mathbf{w}_{y_i}^T \mathbf{x}_i - \max_{\ell \neq y_i} \mathbf{w}_\ell^T \mathbf{x}_i]\}$$

- Intuitively, it measures by how much the score difference between the correct class  $y_i$  and all the other classes  $\ell$  fails to have the desired margin 1 (margin violation)



# Multi-class SVM as a regularized loss minimization problem

- We can write the Multi-class SVM as regularized loss minimization problem:

$$\min_{\mathbf{w}, \xi} \frac{\lambda}{2} \sum_{\ell=1}^k \|\mathbf{w}_{\ell}\|_2^2 + \sum_{i=1}^m \max\{0, 1 - [\mathbf{w}_{y_i}^T \mathbf{x}_i - \max_{\ell \neq y_i} \mathbf{w}_{\ell}^T \mathbf{x}_i]\}$$

- This problem corresponds to the QP formulation by setting  $\lambda = 1/C$
- The problem is convex but the loss is piecewise linear, thus not differentiable everywhere
- A stochastic gradient descent algorithm can be defined through computing the subgradients of the objective function (skipped here)

# Multi-class SVM with kernels

- We can perform non-linear multi-class classification by using a kernel  $\kappa(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$  over the data
- The kernelized version of the multi-class SVM optimizes dual variables  $\alpha = (\alpha_{i,\ell}), i = 1 \dots, m, \ell = 1, \dots, k$  (one dual variable for each training example  $i$  and possible class  $\ell$ )
- The optimization problem is given by

$$\begin{aligned} \max \quad & \sum_{i=1}^m \alpha_{i,y_i} - \frac{1}{2} \sum_{\ell=1}^k \sum_{i,i'=1}^m \alpha_{i,\ell} \alpha_{i',\ell} \kappa(\mathbf{x}_i, \mathbf{x}_{i'}) \\ \text{s.t.} \quad & \sum_{\ell} \alpha_{i,\ell} = 0 \\ & \alpha_{i,\ell} \leq 0, \text{ for } \ell \neq y_i, 0 \leq \alpha_{i,y_i} \leq C, \end{aligned}$$

- Model's prediction in dual form:

$$\hat{y}(\mathbf{x}) = \operatorname{argmax}_{\ell=1,\dots,k} \sum_{i=1}^m \alpha_{i,\ell} \kappa(\mathbf{x}_i, \mathbf{x})$$

# Multi-class boosting

---

# Adaboost for multi-class problems

- AdaBoost.MH is a variant of AdaBoost designed for multi-class and multi-label problems

- Like Adaboost, it learns a linear combination of base classifiers

$$f_N(\mathbf{x}) = \sum_{j=1}^N \alpha_j h_j(\mathbf{x})$$

- The labels are represented as vectors

$\mathbf{y}_i = (y_{i1}, \dots, y_{ik})^T \in \{-1, +1\}^k$ , where in the multi-class setting  $y_{i\ell} = +1$  for the correct class and  $y_{i\ell'} = -1$  for all incorrect classes  $\ell'$

- The base classifiers also return vectors  $h_j(\mathbf{x}) \in \{-1, +1\}^k$ ,

$$h_j(\mathbf{x}, \ell) \in \{-1, +1\}$$

- Prediction either by taking the sign component-wise

$(h(\mathbf{x}) = \text{sgn}(f_N(\mathbf{x})))$  in multi-label setting or taking  $\arg\max_{\ell} f_N(\mathbf{x}, \ell)$  in multi-class setting where  $f_N(\mathbf{x}, \ell)$  is the  $\ell$ 'th component of the predicted vector.

# Adaboost for multi-class problems

- A distribution over the training examples and the possible classes is maintained:  $D_t(i, \ell)$  is the weight of example  $\mathbf{x}_i$  and class  $\ell$  at iteration  $t$
- The updates to the example weights is given by the formula:

$$D_{t+1}(i, \ell) = \frac{D_t(i, \ell) \exp(-\alpha y_{i\ell} h_t(\mathbf{x}_i, \ell))}{Z_t}, \ell = 1, \dots, k$$

- $Z_t$  is a normalization factor
- All weights  $D_t(i, \ell)$  where  $y_{i\ell} \neq h_t(\mathbf{x}_i, \ell)$  are exponentially upweighted
- AdaBoost.MH can be seen to minimize an exponential loss which upper bounds zero-one loss in a multi-class setting

$$\sum_{i=1}^m \sum_{\ell=1}^k \mathbf{1}_{y_{i\ell} \neq h(\mathbf{x}_i, \ell)} \leq \sum_{i=1}^m \sum_{\ell=1}^k \exp(-y_{i\ell} h(\mathbf{x}_i, \ell))$$

---

ADABOOST.MH( $S = ((x_1, y_1), \dots, (x_m, y_m))$ )

```
1  for  $i \leftarrow 1$  to  $m$  do
2      for  $l \leftarrow 1$  to  $k$  do
3           $\mathcal{D}_1(i, l) \leftarrow \frac{1}{mk}$ 
4  for  $j \leftarrow 1$  to  $N$  do
5       $h_j \leftarrow$  base classifier in  $\mathcal{H}$  with small error  $\epsilon_j = \mathbb{P}_{(i,l) \sim \mathcal{D}_j}[h_j(x_i, l) \neq y_i[l]]$ 
6       $\bar{\alpha}_j \leftarrow \frac{1}{2} \log \frac{1-\epsilon_j}{\epsilon_j}$ 
7       $Z_t \leftarrow 2[\epsilon_j(1-\epsilon_j)]^{\frac{1}{2}}$   $\triangleright$  normalization factor
8      for  $i \leftarrow 1$  to  $m$  do
9          for  $l \leftarrow 1$  to  $k$  do
10              $\mathcal{D}_{j+1}(i, l) \leftarrow \frac{\mathcal{D}_j(i, l) \exp(-\bar{\alpha}_j y_i[l] h_j(x_i, l))}{Z_j}$ 
11   $f_N \leftarrow \sum_{j=1}^N \bar{\alpha}_j h_j$ 
12  return  $h = \text{sgn}(f_N)$ 
```

---

- Multi-class classification can be approached as an aggregation of binary classification problems
  - One-versus-All, One-versus-One, and Error-correcting codes
- Standalone models aim to directly minimize a multiclass loss function
  - SVM and Boosting models
- Also other models exist: Multi-class neural networks, Decision trees