9 Multi-Class Classification

The classification problems we examined in the previous chapters were all binary. However, in most real-world classification problems the number of classes is greater than two. The problem may consist of assigning a topic to a text document, a category to a speech utterance or a function to a biological sequence. In all of these tasks, the number of classes may be on the order of several hundred or more.

In this chapter, we analyze the problem of multi-class classification. We first introduce the multi-class classification learning problem and discuss its multiple settings, and then derive generalization bounds for it using the notion of Rademacher complexity. Next, we describe and analyze a series of algorithms for tackling the multi-class classification problem. We will distinguish between two broad classes of algorithms: *uncombined algorithms* that are specifically designed for the multi-class setting such as multi-class SVMs, decision trees, or multi-class boosting, and *aggregated algorithms* that are based on a reduction to binary classification and require training multiple binary classifiers. We will also briefly discuss the problem of structured prediction, which is a related problem arising in a variety of applications.

9.1 Multi-class classification problem

Let $\mathcal{X}$ denote the input space and $\mathcal{Y}$ denote the output space, and let $\mathcal{D}$ be an unknown distribution over $\mathcal{X}$ according to which input points are drawn. We will distinguish between two cases: the *mono-label case*, where $\mathcal{Y}$ is a finite set of classes that we mark with numbers for convenience, $\mathcal{Y} = \{1, \ldots, k\}$, and the *multi-label case* where $\mathcal{Y} = \{-1, +1\}^k$. In the mono-label case, each example is labeled with a single class, while in the multi-label case it can be labeled with several. The latter can be illustrated by the case of text documents, which can be labeled with several different relevant topics, e.g., *sports*, *business*, and *society*. The positive components of a vector in $\{-1, +1\}^k$ indicate the classes associated with an example.

In either case, the learner receives a labeled sample $S = \big((x_1, y_1), \ldots, (x_m, y_m)\big) \in (\mathcal{X} \times \mathcal{Y})^m$ with $x_1, \ldots, x_m$ drawn i.i.d. according to $\mathcal{D}$, and $y_i = f(x_i)$ for all $i \in [m]$, where $f \colon \mathcal{X} \to \mathcal{Y}$ is the target labeling function. Thus, we consider a deterministic scenario, which, as discussed in section 2.4.1, can be straightforwardly extended to a stochastic one that admits a distribution over $\mathcal{X} \times \mathcal{Y}$.

Given a hypothesis set $\mathcal{H}$ of functions mapping $\mathcal{X}$ to $\mathcal{Y}$, the multi-class classification problem consists of using the labeled sample S to find a hypothesis $h \in \mathcal{H}$ with small generalization error $R(h)$ with respect to the target f:

$$R(h) = \mathop{\mathbb{E}}_{x \sim \mathcal{D}}[1_{h(x) \neq f(x)}] \qquad \text{mono-label case} \tag{9.1}$$

$$R(h) = \mathop{\mathbb{E}}_{x \sim \mathcal{D}}\Big[\sum_{l=1}^{k} 1_{[h(x)]_l \neq [f(x)]_l}\Big] \qquad \text{multi-label case.} \tag{9.2}$$

The notion of *Hamming distance* d_H, that is, the number of corresponding components in two vectors that differ, can be used to give a common formulation for both errors:

$$R(h) = \mathop{\mathbb{E}}_{x \sim \mathcal{D}}\Big[d_H(h(x), f(x))\Big]. \tag{9.3}$$

The empirical error of $h \in \mathcal{H}$ is denoted by $\widehat{R}_S(h)$ and defined by

$$\widehat{R}_S(h) = \frac{1}{m} \sum_{i=1}^{m} d_H(h(x_i), y_i)\,. \tag{9.4}$$

Several issues, both computational and learning-related, often arise in the multi-class setting. Computationally, dealing with a large number of classes can be problematic. The number of classes k directly enters the time complexity of the algorithms we will present. Even for a relatively small number of classes such as $k = 100$ or $k = 1{,}000$, some techniques may become prohibitive to use in practice. This dependency is even more critical in the case where k is very large or even infinite as in the case of some structured prediction problems.

A learning-related issue that commonly appears in the multi-class setting is the existence of unbalanced classes. Some classes may be represented by less than 5 percent of the labeled sample, while others may dominate a very large fraction of the data. When separate binary classifiers are used to define the multi-class solution, we may need to train a classifier distinguishing between two classes with only a small representation in the training sample. This implies training on a small sample, with poor performance guarantees. Alternatively, when a large fraction of the training instances belong to one class, it may be tempting to propose a hypothesis always returning that class, since its generalization error as defined earlier is likely to be relatively low. However, this trivial solution is typically not the one intended. Instead, the loss function may need to be reformulated by assigning different misclassification weights to each pair of classes.

Another learning-related issue is the relationship between classes, which can be hierarchical. For example, in the case of document classification, the error of misclassifying a document dealing with *world politics* as one dealing with *real estate* should naturally be penalized more than the error of labeling a document with *sports* instead of the more specific label *baseball*. Thus, a more complex and more useful multi-class classification formulation would take into consideration the hierarchical relationships between classes and define the loss function in accordance with this hierarchy. More generally, there may be a graph relationship between classes as in the case of gene ontology in computational biology. The use of hierarchical relationships between classes leads to a richer and more complex multi-class classification problem.

9.2 Generalization bounds

In this section, we present margin-based generalization bounds for multi-class classification in the mono-label case. In the binary setting, classifiers are often defined based on the sign of a scoring function. In the multi-class setting, a hypothesis is defined based on a scoring function $h\colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$. The label associated to point x is the one resulting in the largest score $h(x, y)$, which defines the following mapping from $\mathcal{X}$ to $\mathcal{Y}$:

$$x \mapsto \operatorname*{argmax}_{y \in \mathcal{Y}} h(x, y).$$

This naturally leads to the following definition of the *margin $\rho_h(x, y)$ of the function h at a labeled example (x, y)*:

$$\rho_h(x, y) = h(x, y) - \max_{y' \neq y} h(x, y').$$

Thus, h misclassifies (x, y) iff $\rho_h(x, y) \leq 0$. For any $\rho > 0$, we can define the *empirical margin loss of a hypothesis h* for multi-class classification as

$$\widehat{R}_{S,\rho}(h) = \frac{1}{m} \sum_{i=1}^{m} \Phi_\rho(\rho_h(x_i, y_i)), \tag{9.5}$$

where Φ_ρ is the margin loss function (definition 5.5). Thus, the empirical margin loss for multi-class classification is upper bounded by the fraction of the training points misclassified by h or correctly classified but with confidence less than or equal to ρ:

$$\widehat{R}_{S,\rho}(h) \leq \frac{1}{m}\sum_{i=1}^{m} 1_{\rho_h(x_i,y_i)\leq\rho}. \tag{9.6}$$

The following lemma will be used in the proof of the main result of this section.

Lemma 9.1 *Let $\mathcal{F}_1, \ldots, \mathcal{F}_l$ be l hypothesis sets in $\mathbb{R}^{\mathcal{X}}$, $l \geq 1$, and let $\mathcal{G} = \{\max\{h_1, \ldots, h_l\} \colon h_i \in \mathcal{F}_i, i \in [l]\}$. Then, for any sample S of size m, the empirical Rademacher complexity of $\mathcal{G}$ can be upper bounded as follows:*

$$\widehat{\mathfrak{R}}_S(\mathcal{G}) \leq \sum_{j=1}^{l} \widehat{\mathfrak{R}}_S(\mathcal{F}_j). \tag{9.7}$$

Proof: Let $S = (x_1, \ldots, x_m)$ be a sample of size m. We first prove the result in the case $l = 2$. By definition of the max operator, for any $h_1 \in \mathcal{F}_1$ and $h_2 \in \mathcal{F}_2$,

$$\max\{h_1, h_2\} = \frac{1}{2}[h_1 + h_2 + |h_1 - h_2|].$$

Thus, we can write:

$$\begin{aligned}
\widehat{\mathfrak{R}}_S(\mathcal{G}) &= \frac{1}{m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\Big[\sup_{\substack{h_1\in\mathcal{F}_1\\ h_2\in\mathcal{F}_2}} \sum_{i=1}^{m} \sigma_i \max\{h_1(x_i), h_2(x_i)\}\Big] \\
&= \frac{1}{2m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\Big[\sup_{\substack{h_1\in\mathcal{F}_1\\ h_2\in\mathcal{F}_2}} \sum_{i=1}^{m} \sigma_i\big(h_1(x_i) + h_2(x_i) + |(h_1 - h_2)(x_i)|\big)\Big] \\
&\leq \frac{1}{2}\widehat{\mathfrak{R}}_S(\mathcal{F}_1) + \frac{1}{2}\widehat{\mathfrak{R}}_S(\mathcal{F}_2) + \frac{1}{2m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\Big[\sup_{\substack{h_1\in\mathcal{F}_1\\ h_2\in\mathcal{F}_2}} \sum_{i=1}^{m} \sigma_i|(h_1 - h_2)(x_i)|\Big], \qquad (9.8)
\end{aligned}$$

using the sub-additivity of sup. Since $x \mapsto |x|$ is 1-Lipschitz, by Talagrand's lemma (lemma 5.7), the last term can be bounded as follows

$$\begin{aligned}
\frac{1}{2m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\Big[\sup_{\substack{h_1\in\mathcal{F}_1\\ h_2\in\mathcal{F}_2}} \sum_{i=1}^{m} \sigma_i|(h_1 - h_2)(x_i)|\Big] &\leq \frac{1}{2m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\Big[\sup_{\substack{h_1\in\mathcal{F}_1\\ h_2\in\mathcal{F}_2}} \sum_{i=1}^{m} \sigma_i(h_1 - h_2)(x_i)\Big] \\
&\leq \frac{1}{2}\widehat{\mathfrak{R}}_S(\mathcal{F}_1) + \frac{1}{2m}\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\Big[\sup_{h_2\in\mathcal{F}_2} \sum_{i=1}^{m} -\sigma_i h_2(x_i)\Big] \\
&= \frac{1}{2}\widehat{\mathfrak{R}}_S(\mathcal{F}_1) + \frac{1}{2}\widehat{\mathfrak{R}}_S(\mathcal{F}_2), \qquad (9.9)
\end{aligned}$$

where we again use the sub-additivity of sup for the second inequality and the fact that σ_i and $-\sigma_i$ have the same distribution for any $i \in [m]$ for the last equality. Combining (9.8) and (9.9) yields $\widehat{\mathfrak{R}}_S(\mathcal{G}) \leq \widehat{\mathfrak{R}}_S(\mathcal{F}_1) + \widehat{\mathfrak{R}}_S(\mathcal{F}_2)$. The general case can be derived from the case $l = 2$ using $\max\{h_1, \ldots, h_l\} = \max\{h_1, \max\{h_2, \ldots, h_l\}\}$ and an immediate recurrence. □

For any family of hypotheses mapping $\mathcal{X} \times \mathcal{Y}$ to $\mathbb{R}$, we define $\Pi_1(\mathcal{H})$ by

$$\Pi_1(\mathcal{H}) = \{x \mapsto h(x, y) \colon y \in \mathcal{Y}, h \in \mathcal{H}\}.$$

The following theorem gives a general margin bound for multi-class classification.

Theorem 9.2 (Margin bound for multi-class classification) *Let $\mathcal{H} \subseteq \mathbb{R}^{\mathcal{X}\times\mathcal{Y}}$ be a hypothesis set with $\mathcal{Y} = \{1, \ldots, k\}$. Fix $\rho > 0$. Then, for any $\delta > 0$, with probability at least $1 - \delta$, the following multi-class classification generalization bound holds for all $h \in \mathcal{H}$:*

$$R(h) \leq \widehat{R}_{S,\rho}(h) + \frac{4k}{\rho}\mathfrak{R}_m(\Pi_1(\mathcal{H})) + \sqrt{\frac{\log\frac{1}{\delta}}{2m}}\,. \tag{9.10}$$

Proof: We will need the following definition for this proof:

$$\rho_{\theta,h}(x, y) = \min_{y'}(h(x, y) - h(x, y') + \theta 1_{y'=y}),$$

where $\theta > 0$ is an arbitrary constant. Observe that $\mathbb{E}[1_{\rho_h(x,y)\leq 0}] \leq \mathbb{E}[1_{\rho_{\theta,h}(x,y)\leq 0}]$ since the inequality $\rho_{\theta,h}(x, y) \leq \rho_h(x, y)$ holds for all $(x, y) \in \mathcal{X} \times \mathcal{Y}$:

$$\begin{aligned}\rho_{\theta,h}(x, y) &= \min_{y'}\big(h(x, y) - h(x, y') + \theta 1_{y'=y}\big)\\ &\leq \min_{y'\neq y}\big(h(x, y) - h(x, y') + \theta 1_{y'=y}\big)\\ &= \min_{y'\neq y}\big(h(x, y) - h(x, y')\big) = \rho_h(x, y),\end{aligned}$$

where the inequality follows from taking the minimum over a smaller set.

Now, similar to the proof of theorem 5.8, let $\widetilde{\mathcal{H}} = \{(x, y) \mapsto \rho_{\theta,h}(x, y) \colon h \in \mathcal{H}\}$ and $\widetilde{\mathcal{H}} = \{\Phi_\rho \circ \widetilde{h} \colon \widetilde{h} \in \widetilde{\mathcal{H}}\}$. By theorem 3.3, with probability at least $1 - \delta$, for all $h \in \mathcal{H}$,

$$\mathbb{E}\left[\Phi_\rho(\rho_{\theta,h}(x, y))\right] \leq \frac{1}{m}\sum_{i=1}^{m}\Phi_\rho(\rho_{\theta,h}(x_i, y_i)) + 2\mathfrak{R}_m(\widetilde{\mathcal{H}}) + \sqrt{\frac{\log\frac{1}{\delta}}{2m}}.$$

Since $1_{u\leq 0} \leq \Phi_\rho(u)$ for all $u \in \mathbb{R}$, the generalization error $R(h)$ is a lower bound on the left-hand side, $R(h) = \mathbb{E}[1_{\rho_h(x,y)\leq 0}] \leq \mathbb{E}[1_{\rho_{\theta,h}(x,y)\leq 0}] \leq \mathbb{E}\left[\Phi_\rho(\rho_{\theta,h}(x, y))\right]$, and we can write:

$$R(h) \leq \frac{1}{m}\sum_{i=1}^{m}\Phi_\rho(\rho_{\theta,h}(x_i, y_i)) + 2\mathfrak{R}_m(\widetilde{\mathcal{H}}) + \sqrt{\frac{\log\frac{1}{\delta}}{2m}}.$$

Fixing $\theta = 2\rho$, we observe that $\Phi_\rho(\rho_{\theta,h}(x_i, y_i)) = \Phi_\rho(\rho_h(x_i, y_i))$. Indeed, either $\rho_{\theta,h}(x_i, y_i) = \rho_h(x_i, y_i)$ or $\rho_{\theta,h}(x_i, y_i) = 2\rho \leq \rho_h(x_i, y_i)$, which implies the desired result. Furthermore, Talagrand's lemma (lemma 5.7) yields $\mathfrak{R}_m(\widetilde{\mathcal{H}}) \leq \frac{1}{\rho}\mathfrak{R}_m(\widetilde{\mathcal{H}})$ since Φ_ρ is a $\frac{1}{\rho}$-Lipschitz function. Therefore, for any $\delta > 0$, with probability at least $1-\delta$, for all $h \in \mathcal{H}$:

$$R(h) \leq \widehat{R}_{S,\rho}(h) + \frac{2}{\rho}\mathfrak{R}_m(\widetilde{\mathcal{H}}) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}.$$

and to complete the proof it suffices to show that $\mathfrak{R}_m(\widetilde{\mathcal{H}}) \leq 2k\mathfrak{R}_m(\Pi_1(\mathcal{H}))$.

Here $\mathfrak{R}_m(\widetilde{\mathcal{H}})$ can be upper-bounded as follows:

$$\begin{aligned}
\mathfrak{R}_m(\widetilde{\mathcal{H}}) &= \frac{1}{m}\mathop{\mathbb{E}}_{S,\sigma}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\sigma_i(h(x_i,y_i) - \max_y(h(x_i,y) - 2\rho 1_{y=y_i}))\right] \\
&\leq \frac{1}{m}\mathop{\mathbb{E}}_{S,\sigma}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\sigma_i h(x_i,y_i)\right] + \frac{1}{m}\mathop{\mathbb{E}}_{S,\sigma}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\sigma_i \max_y(h(x_i,y) - 2\rho 1_{y=y_i})\right].
\end{aligned}$$

Now we bound the first term above. Observe that

$$\begin{aligned}
\frac{1}{m}\mathop{\mathbb{E}}_{\sigma}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\sigma_i h(x_i,y_i)\right] &= \frac{1}{m}\mathop{\mathbb{E}}_{\sigma}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\sum_{y\in\mathcal{Y}}\sigma_i h(x_i,y)1_{y_i=y}\right] \\
&\leq \frac{1}{m}\sum_{y\in\mathcal{Y}}\mathop{\mathbb{E}}_{\sigma}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\sigma_i h(x_i,y)1_{y_i=y}\right] \\
&= \sum_{y\in\mathcal{Y}}\frac{1}{m}\mathop{\mathbb{E}}_{\sigma}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\sigma_i h(x_i,y)\left(\frac{\epsilon_i}{2}+\frac{1}{2}\right)\right],
\end{aligned}$$

where $\epsilon_i = 2\cdot 1_{y_i=y} - 1$. Since $\epsilon_i \in \{-1,+1\}$, we have that σ_i and $\sigma_i\epsilon_i$ admit the same distribution and, for any $y \in \mathcal{Y}$, each of the terms of the right-hand side can be bounded as follows:

$$\begin{aligned}
&\frac{1}{m}\mathop{\mathbb{E}}_{\sigma}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\sigma_i h(x_i,y)\left(\frac{\epsilon_i}{2}+\frac{1}{2}\right)\right] \\
&\quad\leq \frac{1}{2m}\mathop{\mathbb{E}}_{\sigma}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\sigma_i\epsilon_i h(x_i,y)\right] + \frac{1}{2m}\mathop{\mathbb{E}}_{\sigma}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\sigma_i h(x_i,y)\right] \\
&\quad\leq \widehat{\mathfrak{R}}_m(\Pi_1(\mathcal{H})).
\end{aligned}$$

Thus, we can write $\frac{1}{m}\mathbb{E}_{S,\sigma}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\sigma_i h(x_i,y_i)\right] \leq k\,\mathfrak{R}_m(\Pi_1(\mathcal{H}))$. To bound the second term, we first apply lemma 9.1 which immediately yields that

$$\begin{aligned}
&\frac{1}{m}\mathop{\mathbb{E}}_{S,\sigma}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\sigma_i \max_y(h(x_i,y) - 2\rho 1_{y=y_i})\right] \\
&\qquad\qquad\qquad \leq \sum_{y\in\mathcal{Y}}\frac{1}{m}\mathop{\mathbb{E}}_{S,\sigma}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\sigma_i(h(x_i,y) - 2\rho 1_{y=y_i})\right]
\end{aligned}$$

and since Rademacher variables are mean zero, we observe that

$$\begin{aligned}\mathop{\mathbb{E}}_{S,\sigma}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\sigma_i(h(x_i,y)-2\rho 1_{y=y_i})\right] &= \mathop{\mathbb{E}}_{S,\sigma}\left[\sup_{h\in\mathcal{H}}\left(\sum_{i=1}^{m}\sigma_i h(x_i,y)\right)-2\rho\sum_{i=1}^{m}\sigma_i 1_{y=y_i}\right]\\ &= \mathop{\mathbb{E}}_{S,\sigma}\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{m}\sigma_i h(x_i,y)\right] \le \mathfrak{R}_m(\Pi_1(\mathcal{H}))\end{aligned}$$

which completes the proof. □

These bounds can be generalized to hold uniformly for all $\rho > 0$ at the cost of an additional term $\sqrt{(\log\log_2(2/\rho))/m}$, as in theorem 5.9 and exercise 5.2. As for other margin bounds presented in previous sections, they show the conflict between two terms: the larger the desired pairwise ranking margin ρ, the smaller the middle term, at the price of a larger empirical multi-class classification margin loss $\widehat{R}_{S,\rho}$. Note, however, that here there is additionally a dependency on the number of classes k. This suggests either weaker guarantees when learning with a large number of classes or the need for even larger margins ρ for which the empirical margin loss would be small.

For some hypothesis sets, a simple upper bound can be derived for the Rademacher complexity of $\Pi_1(\mathcal{H})$, thereby making theorem 9.2 more explicit. We will show this for kernel-based hypotheses. Let $K\colon \mathcal{X}\times\mathcal{X}\to\mathbb{R}$ be a PDS kernel and let $\mathbf{\Phi}\colon\mathcal{X}\to\mathbb{H}$ be a feature mapping associated to K. In multi-class classification, a kernel-based hypothesis is based on k weight vectors $\mathbf{w}_1,\ldots,\mathbf{w}_k\in\mathbb{H}$. Each weight vector $\mathbf{w}_l$, $l\in[k]$, defines a scoring function $x\mapsto\mathbf{w}_l\cdot\mathbf{\Phi}(x)$ and the class associated to point $x\in\mathcal{X}$ is given by

$$\operatorname*{argmax}_{y\in\mathcal{Y}}\mathbf{w}_y\cdot\mathbf{\Phi}(x).$$

We denote by $\mathbf{W}$ the matrix formed by these weight vectors: $\mathbf{W}=(\mathbf{w}_1,\ldots,\mathbf{w}_k)^\top$ and for any $p\ge 1$ denote by $\|\mathbf{W}\|_{\mathbb{H},p}$ the $L_{\mathbb{H},p}$ group norm of $\mathbf{W}$ defined by

$$\|\mathbf{W}\|_{\mathbb{H},p}=\big(\sum_{l=1}^{k}\|\mathbf{w}_l\|_{\mathbb{H}}^p\big)^{1/p}.$$

For any $p\ge 1$, the family of kernel-based hypotheses we will consider is[13]

$$\mathcal{H}_{K,p}=\{(x,y)\in\mathcal{X}\times\{1,\ldots,k\}\mapsto\mathbf{w}_y\cdot\mathbf{\Phi}(x)\colon\mathbf{W}=(\mathbf{w}_1,\ldots,\mathbf{w}_k)^\top,\|\mathbf{W}\|_{\mathbb{H},p}\le\Lambda\}.$$

Proposition 9.3 (Rademacher complexity of multi-class kernel-based hypotheses) *Let $K\colon\mathcal{X}\times\mathcal{X}\to\mathbb{R}$ be a PDS kernel and let $\mathbf{\Phi}\colon\mathcal{X}\to\mathbb{H}$ be a feature mapping associated to*

[13] The hypothesis set $\mathcal{H}$ can also be defined via $\mathcal{H}=\{h\in\mathbb{R}^{\mathcal{X}\times\mathcal{Y}}\colon h(\cdot,y)\in\mathbb{H}\wedge\|h\|_{K,p}\le\Lambda\}$, where $\|h\|_{K,p}=\big(\sum_{y=1}^{k}\|h(\cdot,y)\|_{\mathbb{H}}^p\big)^{1/p}$, without referring to a feature mapping for K.

K. Assume that there exists $r > 0$ such that $K(x, x) \leq r^2$ for all $x \in \mathcal{X}$. Then, for any $m \geq 1$, $\mathfrak{R}_m(\Pi_1(\mathcal{H}_{K,p}))$ can be bounded as follows:

$$\mathfrak{R}_m(\Pi_1(\mathcal{H}_{K,p})) \leq \sqrt{\frac{r^2\Lambda^2}{m}}.$$

Proof: Let $S = (x_1, \ldots, x_m)$ denote a sample of size m. Observe that for all $l \in [k]$, the inequality $\|\mathbf{w}_l\|_{\mathbb{H}} \leq \big(\sum_{l=1}^k \|\mathbf{w}_l\|_{\mathbb{H}}^p\big)^{1/p} = \|\mathbf{W}\|_{\mathbb{H},p}$ holds. Thus, the condition $\|\mathbf{W}\|_{\mathbb{H},p} \leq \Lambda$ implies that $\|\mathbf{w}_l\|_{\mathbb{H}} \leq \Lambda$ for all $l \in [k]$. In view of that, the Rademacher complexity of the hypothesis set $\Pi_1(\mathcal{H}_{K,p})$ can be expressed and bounded as follows:

$$\begin{aligned}
\mathfrak{R}_m(\Pi_1(\mathcal{H}_{K,p})) &= \frac{1}{m} \mathop{\mathbb{E}}_{S,\boldsymbol{\sigma}}\Big[\sup_{\substack{y\in\mathcal{Y}\\ \|\mathbf{W}\|\leq\Lambda}} \Big\langle \mathbf{w}_y, \sum_{i=1}^m \sigma_i \mathbf{\Phi}(x_i)\Big\rangle\Big] \\
&\leq \frac{1}{m} \mathop{\mathbb{E}}_{S,\boldsymbol{\sigma}}\Big[\sup_{\substack{y\in\mathcal{Y}\\ \|\mathbf{W}\|\leq\Lambda}} \|\mathbf{w}_y\|_{\mathbb{H}} \Big\|\sum_{i=1}^m \sigma_i \Phi(x_i)\Big\|_{\mathbb{H}}\Big] && \text{(Cauchy-Schwarz ineq.)} \\
&\leq \frac{\Lambda}{m} \mathop{\mathbb{E}}_{S,\boldsymbol{\sigma}}\Big[\Big\|\sum_{i=1}^m \sigma_i \Phi(x_i)\Big\|_{\mathbb{H}}\Big] \\
&\leq \frac{\Lambda}{m}\Big[\mathop{\mathbb{E}}_{S,\boldsymbol{\sigma}}\Big[\Big\|\sum_{i=1}^m \sigma_i \Phi(x_i)\Big\|_{\mathbb{H}}^2\Big]\Big]^{1/2} && \text{(Jensen's inequality)} \\
&= \frac{\Lambda}{m}\Big[\mathop{\mathbb{E}}_{S,\boldsymbol{\sigma}}\Big[\sum_{i=1}^m \|\Phi(x_i)\|_{\mathbb{H}}^2\Big]\Big]^{1/2} && (i \neq j \Rightarrow \mathop{\mathbb{E}}_{\boldsymbol{\sigma}}[\sigma_i\sigma_j] = 0) \\
&= \frac{\Lambda}{m}\Big[\mathop{\mathbb{E}}_{S,\boldsymbol{\sigma}}\Big[\sum_{i=1}^m K(x_i, x_i)\Big]\Big]^{1/2} \\
&\leq \frac{\Lambda\sqrt{mr^2}}{m} = \sqrt{\frac{r^2\Lambda^2}{m}},
\end{aligned}$$

which concludes the proof. □

Combining theorem 9.2 and proposition 9.3 yields directly the following result.

Corollary 9.4 (Margin bound for multi-class classification with kernel-based hypotheses) *Let $K\colon \mathcal{X}\times\mathcal{X} \to \mathbb{R}$ be a PDS kernel and let $\mathbf{\Phi}\colon \mathcal{X} \to \mathbb{H}$ be a feature mapping associated to K. Assume that there exists $r > 0$ such that $K(x, x) \leq r^2$ for all $x \in \mathcal{X}$. Fix $\rho > 0$. Then, for any $\delta > 0$, with probability at least $1-\delta$, the following multi-class classification generalization bound holds for all $h \in \mathcal{H}_{K,p}$:*

$$R(h) \leq \widehat{R}_{S,\rho}(h) + 4k\sqrt{\frac{r^2\Lambda^2/\rho^2}{m}} + \sqrt{\frac{\log\frac{1}{\delta}}{2m}}. \tag{9.11}$$

In the next two sections, we describe multi-class classification algorithms that belong to two distinct families: *uncombined algorithms*, which are defined by a single optimization problem, and *aggregated algorithms*, which are obtained by training multiple binary classifications and by combining their outputs.

9.3 Uncombined multi-class algorithms

In this section, we describe three algorithms designed specifically for multi-class classification. We start with a multi-class version of SVMs, then describe a boosting-type multi-class algorithm, and conclude with *decision trees*, which are often used as base learners in boosting.

9.3.1 Multi-class SVMs

We describe an algorithm that can be derived directly from the theoretical guarantees presented in the previous section. Proceeding as in section 5.4 for classification, the guarantee of corollary 9.4 can be expressed as follows: for any $\delta > 0$, with probability at least $1 - \delta$, for all $h \in \mathcal{H}_{K,2} = \{(x, y) \to \mathbf{w}_y \cdot \mathbf{\Phi}(x) \colon \mathbf{W} = (\mathbf{w}_1, \ldots, \mathbf{w}_k)^\top, \sum_{l=1}^{k} \|\mathbf{w}_l\|^2 \leq \Lambda^2\}$,

$$R(h) \leq \frac{1}{m} \sum_{i=1}^{m} \xi_i + 4k \sqrt{\frac{r^2 \Lambda^2}{m}} + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}\,, \tag{9.12}$$

where $\xi_i = \max\big(1 - [\mathbf{w}_{y_i} \cdot \mathbf{\Phi}(x_i) - \max_{y' \neq y_i} \mathbf{w}_{y'} \cdot \mathbf{\Phi}(x_i)], 0\big)$ for all $i \in [m]$.

An algorithm based on this theoretical guarantee consists of minimizing the right-hand side of (9.12), that is, minimizing an objective function with a term corresponding to the sum of the slack variables ξ_i, and another one minimizing $\|\mathbf{W}\|_{\mathbb{H},2}$ or equivalently $\sum_{l=1}^{k} \|\mathbf{w}_l\|^2$. This is precisely the optimization problem defining the *multi-class SVM* algorithm:

$$\begin{aligned} &\min_{\mathbf{W}, \boldsymbol{\xi}} \frac{1}{2} \sum_{l=1}^{k} \|\mathbf{w}_l\|^2 + C \sum_{i=1}^{m} \xi_i \\ &\text{subject to: } \forall i \in [m], \forall l \in \mathcal{Y} - \{y_i\}, \\ &\qquad \mathbf{w}_{y_i} \cdot \mathbf{\Phi}(x_i) \geq \mathbf{w}_l \cdot \mathbf{\Phi}(x_i) + 1 - \xi_i, \\ &\qquad \xi_i \geq 0. \end{aligned}$$

The decision function learned is of the form $x \mapsto \operatorname{argmax}_{l \in \mathcal{Y}} \mathbf{w}_l \cdot \mathbf{\Phi}(x)$. As with the primal problem of SVMs, this is a convex optimization problem: the objective function is convex, since it is a sum of convex functions, and the constraints are affine and thus qualified. The objective and constraint functions are differentiable, and the KKT conditions hold at the optimum. Defining the Lagrangian and applying these conditions leads to the equivalent dual optimization problem, which can be expressed in terms of the kernel function K alone:

$$\max_{\boldsymbol{\alpha} \in \mathbb{R}^{m \times k}} \sum_{i=1}^{m} \boldsymbol{\alpha}_i \cdot \mathbf{e}_{y_i} - \frac{1}{2} \sum_{i=1}^{m} (\boldsymbol{\alpha}_i \cdot \boldsymbol{\alpha}_j) K(x_i, x_j)$$
$$\text{subject to: } \forall i \in [m], (0 \leq \alpha_{iy_i} \leq C) \wedge (\forall j \neq y_i, \alpha_{ij} \leq 0) \wedge (\boldsymbol{\alpha}_i \cdot \mathbf{1} = 0).$$

Here, $\boldsymbol{\alpha} \in \mathbb{R}^{m \times k}$ is a matrix, $\boldsymbol{\alpha}_i$ denotes the ith row of $\boldsymbol{\alpha}$, and $\mathbf{e}_l$ the lth unit vector in $\mathbb{R}^k$, $l \in [k]$. Both the primal and dual problems are simple QPs generalizing those of the standard SVM algorithm. However, the size of the solution and the number of constraints for both problems is in $\Omega(mk)$, which, for a large number of classes k, can make it difficult to solve. However, there exist specific optimization solutions designed for this problem based on a decomposition of the problem into m disjoint sets of constraints.

9.3.2 Multi-class boosting algorithms

We describe a boosting algorithm for multi-class classification called *AdaBoost.MH*, which in fact coincides with a special instance of AdaBoost. An alternative multi-class classification algorithm based on similar boosting ideas, AdaBoost.MR, is described and analyzed in exercise 9.4. AdaBoost.MH applies to the multi-label setting where $\mathcal{Y} = \{-1, +1\}^k$. As in the binary case, it returns a convex combination of base classifiers selected from a hypothesis set $\mathcal{H} = \{h_1, \ldots, h_N\}$. Let F be the following objective function defined for all samples $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$ and $\bar{\boldsymbol{\alpha}} = (\bar{\alpha}_1, \ldots, \bar{\alpha}_N) \in \mathbb{R}^N$, $N \geq 1$, by

$$F(\bar{\boldsymbol{\alpha}}) = \sum_{i=1}^{m} \sum_{l=1}^{k} e^{-y_i[l] f_N(x_i, l)} = \sum_{i=1}^{m} \sum_{l=1}^{k} e^{-y_i[l] \sum_{j=1}^{N} \bar{\alpha}_j h_j(x_i, l)}, \tag{9.13}$$

where $f_N = \sum_{j=1}^{N} \bar{\alpha}_j h_j$ and where $y_i[l]$ denotes the lth coordinate of y_i for any $i \in [m]$ and $l \in [k]$. F is a convex and differentiable upper bound on the multi-class multi-label loss:

$$\sum_{i=1}^{m} \sum_{l=1}^{k} 1_{y_i[l] \neq f_N(x_i, l)} \leq \sum_{i=1}^{m} \sum_{l=1}^{k} e^{-y_i[l] f_N(x_i, l)}, \tag{9.14}$$

since for any $x \in \mathcal{X}$ with label $y = f(x)$ and any $l \in [k]$, the inequality $1_{y[l] \neq f_N(x, l)} \leq e^{-y[l] f_N(x, l)}$ holds. Using the same arguments as in section 7.2.2, we see that AdaBoost.MH coincides exactly with the application of coordinate descent to the objective function F. Figure 9.1 gives the pseudocode of the algorithm in the case where the base classifiers are functions mapping from $\mathcal{X} \times \mathcal{Y}$ to $\{-1, +1\}$. The algorithm takes as input a labeled sample $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$ and maintains a distribution $\mathcal{D}_t$ over $\{1, \ldots, m\} \times \mathcal{Y}$. The remaining details of the

```
AdaBoost.MH($S = ((x_1, y_1), \ldots, (x_m, y_m))$)
 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      $Z_t \leftarrow 2[\epsilon_j(1-\epsilon_j)]^{\frac{1}{2}}$    ▷ 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 $h = \mathrm{sgn}(f_N)$
```

Figure 9.1
AdaBoost.MH algorithm, for $\mathcal{H} \subseteq (\{-1, +1\}^k)^{\mathcal{X}\times\mathcal{Y}}$.

algorithm are similar to AdaBoost. In fact, AdaBoost.MH exactly coincides with AdaBoost applied to the training sample derived from S by splitting each labeled point (x_i, y_i) into k labeled examples $((x_i, l), y_i[l])$, with each example (x_i, l) in $\mathcal{X} \times \mathcal{Y}$ and its label in $\{-1, +1\}$:

$$(x_i, y_i) \to ((x_i, 1), y_i[1]), \ldots, ((x_i, k), y_i[k]), i \in [m].$$

Let S' denote the resulting sample, then $S' = ((x_1, 1), y_1[1]), \ldots, (x_m, k), y_m[k]))$. S' contains mk examples and the expression of the objective function F in (9.13) coincides exactly with that of the objective function of AdaBoost for the sample S'. In view of this connection, the theoretical analysis along with the other observations we presented for AdaBoost in chapter 7 also apply here. Hence, we will focus on aspects related to the computational efficiency and to the weak learning condition that are specific to the multi-class scenario.

The complexity of the algorithm is that of AdaBoost applied to a sample of size mk. For $\mathcal{X} \subseteq \mathbb{R}^d$, using boosting stumps as base classifiers, the complexity of the algorithm is therefore in $O((mk)\log(mk) + mkdN)$. Thus, for a large number of classes k, the algorithm may become impractical using a single processor. The

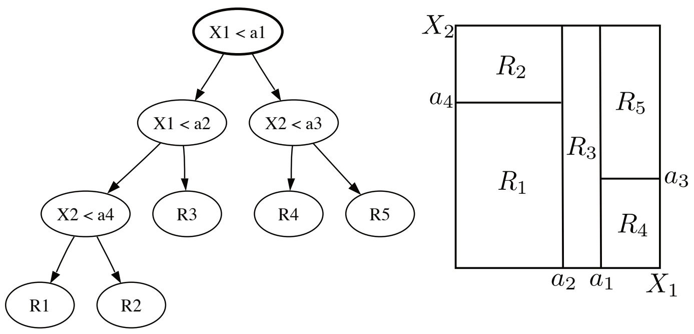

Figure 9.2
Left: example of a decision tree with numerical questions based on two variables X_1 and X_2. Here, each leaf is marked with the region it defines. The class labeling for a leaf is obtained via majority vote based on the training points falling in the region it defines. Right: Partition of the two-dimensional space induced by that decision tree.

weak learning condition for the application of AdaBoost in this scenario requires that at each round there exists a base classifier $h_j : \mathcal{X} \times \mathcal{Y} \to \{-1, +1\}$ such that $\mathbb{P}_{(i,l)\sim\mathcal{D}_j}[h_j(x_i, l) \neq y_i[l]] < 1/2$. This may be hard to achieve if some classes difficult to distinguish between. It is also more difficult in this context to come up with "rules of thumb" h_j defined over $\mathcal{X} \times \mathcal{Y}$.

9.3.3 Decision trees

We present and discuss the general learning method of *decision trees* that can be used in multi-class classification, but also in other learning problems such as regression (chapter 11) and clustering. Although the empirical performance of decision trees often is not state-of-the-art, decision trees can be used as weak learners with boosting to define effective learning algorithms. Decision trees are also typically fast to train and evaluate and relatively easy to interpret.

Definition 9.5 (Binary decision tree) *A* binary decision tree *is a tree representation of a partition of the feature space. Figure 9.2 shows a simple example in the case of a two-dimensional space based on two features X_1 and X_2, as well as the partition it represents. Each interior node of a decision tree corresponds to a question related to the features. It can be a* numerical question *of the form $X_i \leq a$ for a feature variable X_i, $i \in [N]$, and some threshold $a \in \mathbb{R}$, as in the example of figure 9.2, or a* categorical question *such as $X_i \in \{\text{blue}, \text{white}, \text{red}\}$, when feature X_i takes a categorical value such as a color. Each leaf is labeled with a label $l \in \mathcal{Y}$.*

Decision trees can be defined using more complex node questions, resulting in partitions based on more complex decision surfaces. For example, *binary space*

```
GreedyDecisionTrees($S = ((x_1, y_1), \ldots, (x_m, y_m))$)
1  tree ← {$n_0$}  ▷ root node.
2  for $t \leftarrow 1$ to $T$ do
3      $(\mathsf{n}_t, \mathsf{q}_t) \leftarrow \operatorname{argmax}_{(\mathsf{n},\mathsf{q})} \widetilde{F}(\mathsf{n}, \mathsf{q})$
4      Split(tree, $\mathsf{n}_t$, $\mathsf{q}_t$)
5  return tree
```

Figure 9.3
Greedy algorithm for building a decision tree from a labeled sample S. The procedure Split(tree, n_t, q_t) splits node n_t by making it an internal node with question q_t and leaf children $\mathsf{n}_-(\mathsf{n}, \mathsf{q})$ and $\mathsf{n}_+(\mathsf{n}, \mathsf{q})$, each labeled with the dominating class of the region it defines, with ties broken arbitrarily.

partition (BSP) trees partition the space with convex polyhedral regions, based on questions of the form $\sum_{i=1}^{n} \alpha_i X_i \leq a$, and *sphere trees* partition with pieces of spheres based on questions of the form $\|X - a_0\| \leq a$, where X is a feature vector, a_0 a fixed vector, and a is a fixed positive real number. More complex tree questions lead to richer partitions and thus hypothesis sets, which can cause overfitting in the absence of a sufficiently large training sample. They also increase the computational complexity of prediction and training. Decision trees can also be generalized to branching factors greater than two, but binary trees are most commonly used due their more limited computational cost.

Prediction/partitioning: To predict the label of any point $x \in \mathcal{X}$ we start at the root node of the decision tree and go down the tree until a leaf is found, by moving to the right child of a node when the response to the node question is positive, and to the left child otherwise. When we reach a leaf, we associate x with the label of this leaf.

Thus, each leaf defines a *region* of $\mathcal{X}$ formed by the set of points corresponding exactly to the same node responses and thus the same traversal of the tree. By definition, no two regions intersect and all points belong to exactly one region. Thus, leaf regions define a partition of $\mathcal{X}$, as shown in the example of figure 9.2. In multi-class classification, the label of a leaf is determined using the training sample: the class with the majority representation among the training points falling in a leaf region defines the label of that leaf, with ties broken arbitrarily.

Learning: We will discuss two different methods for learning a decision tree using a labeled sample. The first method is a greedy technique. This is motivated by the fact that the general problem of finding a decision tree with the smallest error is NP-hard. The method consists of starting with a tree reduced to a single (root) node, which is a leaf whose label is the class that has majority over the entire sample. Next, at each round, a node n_t is split based on some question q_t. The pair $(\mathsf{n}_t, \mathsf{q}_t)$ is chosen so that the *node impurity* is maximally decreased according to some measure of impurity F. We denote by $F(\mathsf{n})$ the impurity of n. The decrease in node impurity after a split of node n based on question q is defined as follows. Let $\mathsf{n}_+(\mathsf{n}, \mathsf{q})$ denote the right child of n after the split, $\mathsf{n}_-(\mathsf{n}, \mathsf{q})$ the left child, and $\eta(\mathsf{n}, \mathsf{q})$ the fraction of the points in the region defined by n that are moved to $\mathsf{n}_-(\mathsf{n}, \mathsf{q})$. The total impurity of the leaves $\mathsf{n}_-(\mathsf{n}, \mathsf{q})$ and $\mathsf{n}_+(\mathsf{n}, \mathsf{q})$ is therefore $\eta(\mathsf{n}, \mathsf{q})F(\mathsf{n}_-(\mathsf{n}, \mathsf{q})) + (1-\eta(\mathsf{n}, \mathsf{q}))F(\mathsf{n}_+(\mathsf{n}, \mathsf{q}))$. Thus, the decrease in impurity $\widetilde{F}(\mathsf{n}, \mathsf{q})$ by that split is given by

$$\widetilde{F}(\mathsf{n}, \mathsf{q}) = F(\mathsf{n}) - [\eta(\mathsf{n}, \mathsf{q})F(\mathsf{n}_-(\mathsf{n}, \mathsf{q})) + (1 - \eta(\mathsf{n}, \mathsf{q}))F(\mathsf{n}_+(\mathsf{n}, \mathsf{q}))].$$

Figure 9.3 shows the pseudocode of this greedy construction based on $\widetilde{F}$. In practice, the algorithm is stopped once all nodes have reached a sufficient level of purity, when the number of points per leaf has become too small for further splitting or based on some other similar heuristic.

For any node n and class $l \in [k]$, let $p_l(\mathsf{n})$ denote the fraction of points at n that belong to class l. Then, the three most commonly used measures of node impurity F are defined as follows:

$$F(\mathsf{n}) = \begin{cases} 1 - \max_{l \in [k]} p_l(\mathsf{n}) & \textit{misclassification}; \\ -\sum_{l=1}^{k} p_l(\mathsf{n}) \log_2 p_l(\mathsf{n}) & \textit{entropy}; \\ \sum_{l=1}^{k} p_l(\mathsf{n})(1 - p_l(\mathsf{n})) & \textit{Gini index}. \end{cases}$$

Figure 9.4 illustrates these definitions in the special cases of two classes ($k = 2$). The entropy and Gini index impurity functions are upper bounds on the misclassification impurity function. All three functions are concave, which ensures that

$$F(\mathsf{n}) - [\eta(\mathsf{n}, \mathsf{q})F(\mathsf{n}_-(\mathsf{n}, \mathsf{q})) + (1 - \eta(\mathsf{n}, \mathsf{q}))F(\mathsf{n}_+(\mathsf{n}, \mathsf{q}))] \geq 0.$$

However, the misclassification function is piecewise linear, so $\widetilde{F}(\mathsf{n}, \mathsf{q})$ is zero if the fraction of positive points remains less than (or more than) half after a split. In some cases, the impurity cannot be decreased by any split using that criterion. In contrast, the entropy and Gini functions are strictly concave, which guarantees a strict decrease in impurity. Furthermore, they are differentiable which is a useful feature for numerical optimization. Thus, the Gini index and the entropy criteria are typically preferred in practice.

The greedy method just described faces some issues. One issue relates to the greedy nature of the algorithm: a seemingly bad split may dominate subsequent useful splits, which could lead to trees with less impurity overall. This can be

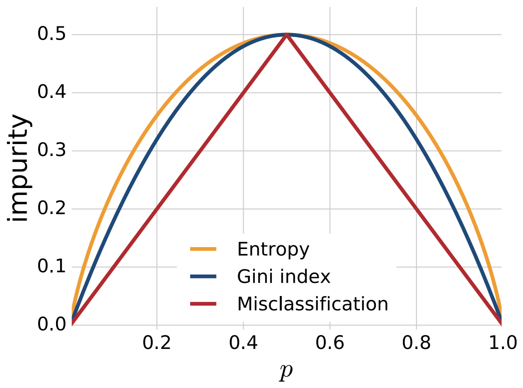

Figure 9.4
Three node impurity definitions plotted as a function of the fraction of positive examples in the binary case: misclassification, entropy (scaled by 0.5 to set the maximum to the same value for all three functions), and the Gini index.

addressed to a certain extent by using a look-ahead of some depth d to determine the splitting decisions, but such look-aheads can be computationally very costly. Another issue relates to the size of the resulting tree. To achieve some desired level of impurity, trees of relatively large sizes may be needed. However, larger trees define overly complex hypotheses with high VC-dimensions (see exercise 9.5) and thus could overfit.

An alternative method for learning decision trees using a labeled training sample is based on the so-called *grow-then-prune strategy*. First a very large tree is grown until it fully fits the training sample or until no more than a very small number of points are left at each leaf. Then, the resulting tree, denoted as tree, is pruned back to minimize an objective function defined (based on generalization bounds) as the sum of an empirical error and a complexity term. The complexity can be expressed in terms of the size of $\widetilde{\mathsf{tree}}$, the set of leaves of tree. The resulting objective is

$$G_\lambda(\mathsf{tree}) = \sum_{\mathsf{n} \in \widetilde{\mathsf{tree}}} |\mathsf{n}| F(\mathsf{n}) + \lambda |\widetilde{\mathsf{tree}}| \,, \tag{9.15}$$

where $\lambda \geq 0$ is a regularization parameter determining the trade-off between misclassification, or more generally impurity, versus tree complexity. For any tree tree', we denote by $\widehat{R}(\mathsf{tree}')$ the total empirical error $\sum_{\mathsf{n} \in \widetilde{\mathsf{tree}}'} |\mathsf{n}| F(\mathsf{n})$. We seek a sub-tree tree_λ of tree that minimizes G_λ and that has the smallest size. tree_λ can be shown to be unique. To determine tree_λ, the following pruning method is used, which defines a finite sequence of nested sub-trees $\mathsf{tree}^{(0)}, \ldots, \mathsf{tree}^{(n)}$. We start with the full tree $\mathsf{tree}^{(0)} = \mathsf{tree}$ and for any $i \in \{0, \ldots, n-1\}$, define $\mathsf{tree}^{(i+1)}$ from $\mathsf{tree}^{(i)}$ by collapsing an internal node n' of $\mathsf{tree}^{(i)}$, that is by replacing the sub-tree rooted at n' with a leaf, or equivalently by combining the regions of all the leaves dominated by n'. n' is chosen so that collapsing it causes the smallest per node increase in $\widehat{R}(\mathsf{tree}^{(i)})$, that is the smallest $r(\mathsf{tree}^{(i)}, \mathsf{n}')$ defined by

$$r(\mathsf{tree}^{(i)}, \mathsf{n}') = \frac{|\mathsf{n}'|F(\mathsf{n}') - \widehat{R}(\mathsf{tree}')}{|\widetilde{\mathsf{tree}}'| - 1},$$

where n' is an internal node of $\mathsf{tree}^{(i)}$. If several nodes n' in $\mathsf{tree}^{(i)}$ cause the same smallest increase per node $r(\mathsf{tree}^{(i)}, \mathsf{n}')$, then all of them are pruned to define $\mathsf{tree}^{(i+1)}$ from $\mathsf{tree}^{(i)}$. This procedure continues until the tree $\mathsf{tree}^{(n)}$ obtained has a single node. The sub-tree tree_λ can be shown to be among the elements of the sequence $\mathsf{tree}^{(0)}, \ldots, \mathsf{tree}^{(n)}$. The parameter λ is determined via n-fold cross-validation.

Decision trees seem relatively easy to interpret, and this is often underlined as one of their most useful features. However, such interpretations should be carried out with care since decision trees are *unstable*: small changes in the training data may lead to very different splits and thus entirely different trees, as a result of their hierarchical nature. Decision trees can also be used in a natural manner to deal with the problem of *missing features*, which often appears in learning applications; in practice, some features values may be missing because the proper measurements were not taken or because of some noise source causing their systematic absence. In such cases, only those variables available at a node can be used in prediction. Finally, decision trees can be used and learned from data in a similar way in the *regression* setting (see chapter 11).[14]

9.4 Aggregated multi-class algorithms

In this section, we discuss a different approach to multi-class classification that reduces the problem to that of multiple binary classification tasks. A binary classification algorithm is then trained for each of these tasks independently, and the multi-class predictor is defined as a combination of the hypotheses returned by each of these algorithms. We first discuss two standard techniques for the reduction of multi-class classification to binary classification, and then show that they are both special instances of a more general framework.

[14] The only changes to the description for classification are the following. For prediction, the label of a leaf is defined as the mean squared average of the labels of the points falling in that region. For learning, the impurity function is the mean squared error.

9.4.1 One-versus-all

Let $S = ((x_1, y_1), \ldots, x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$ be a labeled training sample. A straightforward reduction of the multi-class classification to binary classification is based on the so-called *one-versus-all (OVA) or one-versus-the-rest technique*. This technique consists of learning k binary classifiers $h_l \colon \mathcal{X} \to \{-1, +1\}$, $l \in \mathcal{Y}$, each seeking to discriminate one class $l \in \mathcal{Y}$ from all the others. For any $l \in \mathcal{Y}$, h_l is obtained by training a binary classification algorithm on the full sample S after relabeling points in class l with 1 and all others with -1. For $l \in \mathcal{Y}$, assume that h_l is derived from the sign of a scoring function $f_l \colon \mathcal{X} \to \mathbb{R}$, that is $h_l = \operatorname{sgn}(f_l)$, as in the case of many of the binary classification algorithms discussed in the previous chapters. Then, the multi-class hypothesis $h \colon \mathcal{X} \to \mathcal{Y}$ defined by the OVA technique is given by:

$$\forall x \in \mathcal{X}, \quad h(x) = \operatorname*{argmax}_{l \in \mathcal{Y}} f_l(x). \tag{9.16}$$

This formula may seem similar to those defining a multi-class classification hypothesis in the case of uncombined algorithms. Note, however, that for uncombined algorithms the functions f_l are learned together, while here they are learned independently. Formula (9.16) is well-founded when the scores given by functions f_l can be interpreted as confidence scores, that is when $f_l(x)$ is learned as an estimate of the probability of x conditioned on class l. However, in general, the scores given by functions f_l, $l \in \mathcal{Y}$, are not comparable and the OVA technique based on (9.16) admits *no principled justification.* This is sometimes referred to as a *calibration problem*. Clearly, this problem cannot be corrected by simply normalizing the scores of each function to make their magnitudes uniform, or by applying other similar heuristics. When it is justifiable, the OVA technique is simple and its computational cost is k times that of training a binary classification algorithm, which is similar to the computation costs for many uncombined algorithms.

9.4.2 One-versus-one

An alternative technique, known as the *one-versus-one (OVO) technique*, consists of using the training data to learn (independently), for each pair of distinct classes $(l, l') \in \mathcal{Y}^2$, $l \neq l'$, a binary classifier $h_{ll'} \colon \mathcal{X} \to \{-1, 1\}$ discriminating between classes l and l'. For any $(l, l') \in \mathcal{Y}^2$, $h_{ll'}$ is obtained by training a binary classification algorithm on the sub-sample containing exactly the points labeled with l or l', with the value $+1$ returned for class l' and -1 for class l. This requires training $\binom{k}{2} = k(k-1)/2$ classifiers, which are combined to define a multi-class classification hypothesis h via majority vote:

$$\forall x \in \mathcal{X}, \quad h(x) = \operatorname*{argmax}_{l' \in \mathcal{Y}} \big|\{l \colon h_{ll'}(x) = 1\}\big|. \tag{9.17}$$

Table 9.1

Comparison of the time complexity the OVA and OVO techniques for both training and testing. The table assumes a full training sample of size m with each class represented by m/k points. The time for training a binary classification algorithm on a sample of size n is assumed to be in $O(n^\alpha)$. Thus, the training time for the OVO technique is in $O(k^2(m/k)^\alpha) = O(k^{2-\alpha}m^\alpha)$. c_t denotes the cost of testing a single classifier.

	Training	Testing
OVA	$O(km^\alpha)$	$O(kc_t)$
OVO	$O(k^{2-\alpha}m^\alpha)$	$O(k^2c_t)$

Thus, for a fixed point $x \in \mathcal{X}$, if we describe the prediction values $h_{ll'}(x)$ as the results of the matches in a tournament between two players l and l', with $h_{ll'}(x) = 1$ indicating l' winning over l, then the class predicted by h can be interpreted as the one with the largest number of wins in that tournament.

Let $x \in \mathcal{X}$ be a point belonging to class l'. By definition of the OVO technique, if $h_{ll'}(x) = 1$ for all $l \neq l'$, then the class associated to x by OVO is the correct class l' since $\big|\{l \colon h_{ll'}(x) = 1\}\big| = k - 1$ and no other class can reach $(k-1)$ wins. By contraposition, if the OVO hypothesis misclassifies x, then at least one of the $(k-1)$ binary classifiers $h_{ll'}$, $l \neq l'$, incorrectly classifies x. Assume that the generalization error of all binary classifiers $h_{ll'}$ used by OVO is at most r, then, in view of this discussion, the generalization error of the hypothesis returned by OVO is at most $(k-1)r$.

The OVO technique is not subject to the calibration problem pointed out in the case of the OVA technique. However, when the size of the sub-sample containing members of the classes l and l' is relatively small, $h_{ll'}$ may be learned without sufficient data or with increased risk of overfitting. Another concern often raised for the use of this technique is the computational cost of training $k(k-1)/2$ binary classifiers versus that of the OVA technique.

Taking a closer look at the computational requirements of these two methods reveals, however, that the disparity may not be so great and that in fact under some assumptions the time complexity of training for OVO could be less than that of OVA. Table 9.1 compares the computational complexity of these methods both for training and testing assuming that the complexity of training a binary classifier on a sample of size m is in $O(m^\alpha)$ and that each class is equally represented in the training set, that is by m/k points. Under these assumptions, if $\alpha \in [2, 3)$ as in the case of some algorithms solving a QP problem, such as SVMs, then the time complexity of training for the OVO technique is in fact more favorable than that of OVA. For $\alpha = 1$, the two are comparable and it is only for sub-linear algorithms that the OVA technique would benefit from a better complexity. In all cases, at test time, OVO requires $k(k-1)/2$ classifier evaluations, which is $(k-1)$

times more than OVA. However, for some algorithms the evaluation time for each classifier could be much smaller for OVO. For example, in the case of SVMs, the average number of support vectors may be significantly smaller for OVO, since each classifier is trained on a significantly smaller sample. If the number of support vectors is k times smaller and if sparse feature representations are used, then the time complexities of both techniques for testing are comparable.

9.4.3 Error-correcting output codes

A more general method for the reduction of multi-class to binary classification is based on the idea of *error-correcting output codes (ECOC)*. This technique consists of assigning to each class $l \in \mathcal{Y}$ a *code word* of length $c \geq 1$, which in the simplest case is a binary vector $\mathbf{M}_l \in \{-1,+1\}^c$. $\mathbf{M}_l$ serves as a signature for class l, and together these vectors define a matrix $\mathbf{M} \in \{-1,+1\}^{k\times c}$ whose lth row is $\mathbf{M}_l$, as illustrated by figure 9.5. Next, for each column $j \in [c]$, a binary classifier $h_j \colon \mathcal{X} \to \{-1,+1\}$ is learned using the full training sample S, after all points that belong to a class represented by $+1$ in column j are labeled with $+1$, while all other points are labeled with -1. For any $x \in \mathcal{X}$, let $\mathbf{h}(x)$ denote the vector $\mathbf{h}(x) = (h_1(x), \ldots, h_c(x))^\top$. Then, the multi-class hypothesis $h \colon \mathcal{X} \to \mathcal{Y}$ is defined by

$$\forall x \in \mathcal{X}, \quad h(x) = \operatorname*{argmin}_{l \in \mathcal{Y}} d_H\big(\mathbf{M}_l, \mathbf{h}(x)\big). \tag{9.18}$$

Thus, the class predicted is the one whose signatures is the closest to $\mathbf{h}(x)$ in Hamming distance. Figure 9.5 illustrates this definition: no row of matrix $\mathbf{M}$ matches the vector of predictions $\mathbf{h}(x)$ in that case, but the third row shares the largest number of components with $\mathbf{h}(x)$.

The success of the ECOC technique depends on the minimal Hamming distance between the class code words. Let d denote that distance, then up to $r_0 = \left\lfloor \frac{d-1}{2} \right\rfloor$ binary classification errors can be corrected by this technique: by definition of d, even if $r < r_0$ binary classifiers h_l misclassify $x \in \mathcal{X}$, $\mathbf{h}(x)$ is closest to the code word of the correct class of x. For a fixed c, the design of error-correction matrix $\mathbf{M}$ is subject to a trade-off, since larger d values may imply substantially more difficult binary classification tasks. In practice, each column may correspond to a class feature determined based on domain knowledge.

The ECOC technique just described can be extended in two ways. First, instead of using only the label predicted by each classifier h_j the magnitude of the scores defining h_j is used. Thus, if $h_j = \operatorname{sgn}(f_j)$ for some function f_j whose values can be interpreted as confidence scores, then the multi-class hypothesis $h \colon \mathcal{X} \to \mathcal{Y}$ is 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

Figure 9.5
Illustration of error-correcting output codes for multi-class classification. Left: binary code matrix $\mathbf{M}$, with each row representing the code word of length $c = 6$ of a class $l \in [8]$. Right: vector of predictions $\mathbf{h}(x)$ for a test point x. The ECOC classifier assigns label 3 to x, since the binary code for the third class yields the minimal Hamming distance with $\mathbf{h}(x)$ (distance of 1).

defined by

$$\forall x \in \mathcal{X}, \quad h(x) = \operatorname*{argmin}_{l \in \mathcal{Y}} \sum_{j=1}^{c} L(m_{lj} f_j(x)), \tag{9.19}$$

where (m_{lj}) are the entries of $\mathbf{M}$ and where $L\colon \mathbb{R} \to \mathbb{R}_+$ is a loss function. When L is defined by $L(x) = \frac{1-\operatorname{sgn}(x)}{2}$ for all $x \in \mathcal{X}$ and $h_l = f_l$, we can write:

$$\sum_{j=1}^{c} L(m_{lj} f_j(x)) = \sum_{j=1}^{c} \frac{1 - \operatorname{sgn}(m_{lj} h_j(x))}{2} = d_h(\mathbf{M}_l, \mathbf{h}(x)),$$

and (9.19) coincides with (9.18). Furthermore, ternary codes can be used with matrix entries in $\{-1, 0, +1\}$ so that examples in classes labeled with 0 are disregarded when training a binary classifier for each column. With these extensions, both OVA and OVO become special instances of the ECOC technique. The matrix $\mathbf{M}$ for OVA is a square matrix, that is $c = k$, with all terms equal to -1 except from the diagonal ones which are all equal to $+1$. The matrix $\mathbf{M}$ for OVO has $c = k(k-1)/2$ columns. Each column corresponds to a pair of distinct classes (l, l'), $l \neq l'$, with all entries equal to 0 except from the one with row l, which is -1, and the one with row l', which is $+1$.

Since the values of the scoring functions are assumed to be confidence scores, $m_{lj} f_j(x)$ can be interpreted as the margin of classifier j on point x and (9.19) is thus based on some loss L defined with respect to the binary classifier's margin.

A further extension of ECOC consists of extending discrete codes to continuous ones by letting the matrix entries take arbitrary real values and by using the training sample to *learn* matrix $\mathbf{M}$. Starting with a discrete version of $\mathbf{M}$, c binary classifiers with scoring functions f_l, $l \in [c]$, are first learned as described previously. We will denote by $\mathbf{F}(x)$ the vector $(f_1(x), \ldots, f_c(x))^\top$ for any $x \in \mathcal{X}$. Next, the entries of $\mathbf{M}$ are relaxed to take real values and learned from the training sample with the objective of making the row of $\mathbf{M}$ corresponding to the class of any point $x \in \mathcal{X}$ more similar to $\mathbf{F}(x)$ than other rows. The similarity can be measured using any PDS kernel K. An example of an algorithm for learning $\mathbf{M}$ using a PDS kernel K and the idea just discussed is in fact multi-class SVMs, which, in this context, can be formulated as follows:

$$\begin{aligned}
&\min_{\mathbf{M}, \boldsymbol{\xi}} \|\mathbf{M}\|_F^2 + C \sum_{i=1}^{m} \xi_i \\
&\text{subject to: } \forall (i, l) \in [m] \times \mathcal{Y}, \\
&\qquad K(\mathbf{f}(x_i), \mathbf{M}_{y_i}) \geq K(\mathbf{f}(x_i), \mathbf{M}_l) + 1 - \xi_i.
\end{aligned}$$

Similar algorithms can be defined using other matrix norms. The resulting multi-class classification decision function has the following form:

$$h \colon x \mapsto \operatorname*{argmax}_{l \in \{1, \ldots, k\}} K(\mathbf{f}(x), \mathbf{M}_l).$$

9.5 Structured prediction algorithms

In this section, we briefly discuss an important class of problems related to multi-class classification that frequently arises in computer vision, computational biology, and natural language processing. These include all sequence labeling problems and complex problems such as parsing, machine translation, and speech recognition.

In these applications, the output labels have a rich internal structure. For example, in *part-of-speech tagging* the problem consists of assigning a part-of-speech tag such as N (noun), V (verb), or A (adjective), to every word of a sentence. Thus, the label of the sentence $\omega_1 \ldots \omega_n$ made of the words ω_i is a sequence of part-of-speech tags $t_1 \ldots t_n$. This can be viewed as a multi-class classification problem where each sequence of tags is a possible label. However, several critical aspects common to such *structured output* problems make them distinct from the standard multi-class classification.

First, the label set is exponentially large as a function of the size of the output. For example, if Σ denotes the alphabet of part-of-speech tags, for a sentence of length n there are $|\Sigma|^n$ possible tag sequences. Second, there are dependencies between the substructures of a label that are important to take into account for an accurate prediction. For example, in part-of-speech tagging, some tag sequences may be ungrammatical or unlikely. Finally, the loss function used is typically not a zero-one loss, but one that depends on the substructures. Let $L \colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ denote a loss function such that $L(y', y)$ measures the penalty of predicting the label $y' \in \mathcal{Y}$ instead of the correct label $y \in \mathcal{Y}$.[15] In part-of-speech tagging, $L(y', y)$ could be for example the Hamming distance between y' and y.

The relevant features in structured output problems often depend on both the input and the output. Thus, we will denote by $\mathbf{\Phi}(x, y) \in \mathbb{R}^N$ the feature vector associated to a pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$.

To model the label structures and their dependency, the label set $\mathcal{Y}$ is typically assumed to be endowed with a *graphical model* structure, that is, a graph giving a probabilistic model of the conditional dependence between the substructures. It is also assumed that both the feature vector $\mathbf{\Phi}(x, y)$ associated to an input $x \in \mathcal{X}$ and output $y \in \mathcal{Y}$ and the loss $L(y', y)$ factorize according to the cliques of that graphical model.[16] A detailed treatment of this topic would require a further background in graphical models, and is thus beyond the scope of this section.

The hypothesis set used by most structured prediction algorithms is then defined as the set of functions $h: \mathcal{X} \to \mathcal{Y}$ such that

$$\forall x \in \mathcal{X}, \quad h(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} \mathbf{w} \cdot \mathbf{\Phi}(x, y), \tag{9.20}$$

for some vector $\mathbf{w} \in \mathbb{R}^N$. Let $S = ((x_1, y_1), \ldots, x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$ be an i.i.d. labeled sample. Since the hypothesis set is linear, we can seek to define an algorithm similar to multi-class SVMs. The optimization problem for multi-class SVMs can be rewritten equivalently as follows:

$$\min_{\mathbf{w}} \ \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \max_{y \neq y_i} \max\Big(0, 1 - \mathbf{w} \cdot [\mathbf{\Phi}(x_i, y_i) - \mathbf{\Phi}(x_i, y)]\Big), \tag{9.21}$$

However, here we need to take into account the loss function L, that is $L(y, y_i)$ for each $i \in [m]$ and $y \in \mathcal{Y}$, and there are multiple ways to proceed. One possible way is to let the margin violation be penalized additively with $L(y, y_i)$. Thus, in that case $L(y, y_i)$ is added to the margin violation. Another natural method consists of penalizing the margin violation by multiplying it with $L(y, y_i)$. A margin violation with a larger loss is then penalized more than one with a smaller loss.

[15] More generally, in some applications, the loss function could also depend on the input. Thus, L is then a function mapping $L: \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, with $L(x, y', y)$ measuring the penalty of predicting the label y' instead of y given the input x.

[16] In an undirected graph, a *clique* is a set of fully connected vertices.

The additive penalization leads to the following algorithm known as *Maximum Margin Markov Networks (M^{3}N)*:

$$\min_{\mathbf{w}} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \max_{y \neq y_i} \max\Big(0, L(y_i, y) - \mathbf{w}\cdot[\mathbf{\Phi}(x_i, y_i) - \mathbf{\Phi}(x_i, y)]\Big). \tag{9.22}$$

An advantage of this algorithm is that, as in the case of SVMs, it admits a natural use of PDS kernels. As already indicated, the label set $\mathcal{Y}$ is assumed to be endowed with a graph structure with a Markov property, typically a chain or a tree, and the loss function is assumed to be decomposable in the same way. Under these assumptions, by exploiting the graphical model structure of the labels, a polynomial-time algorithm can be given to determine its solution.

A multiplicative combination of the loss with the margin leads to the following algorithm known as *SVMStruct*:

$$\min_{\mathbf{w}} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \max_{y \neq y_i} L(y_i, y) \max\Big(0, 1 - \mathbf{w}\cdot[\mathbf{\Phi}(x_i, y_i) - \mathbf{\Phi}(x_i, y)]\Big). \tag{9.23}$$

This problem can be equivalently written as a QP with an infinite number of constraints. In practice, it is solved iteratively by augmenting at each round the finite set of constraints of the previous round with the most violating constraint. This method can be applied in fact under very general assumptions and for arbitrary loss definitions. As in the case of M^{3}N, SVMStruct naturally admits the use of PDS kernels and thus an extension to non-linear models for the solution.

Another standard algorithm for structured prediction problems is *Conditional Random Fields (CRFs)*. We will not describe this algorithm in detail, but point out its similarity with the algorithms just described, in particular M^{3}N. The optimization problem for CRFs can be written as

$$\min_{\mathbf{w}} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \log \sum_{y \in \mathcal{Y}} \exp\Big(L(y_i, y) - \mathbf{w}\cdot[\mathbf{\Phi}(x_i, y_i) - \mathbf{\Phi}(x_i, y)]\Big). \tag{9.24}$$

Assume for simplicity that $\mathcal{Y}$ is finite and has cardinality k and let f denote the function $(x_1, \ldots, x_k) \mapsto \log(\sum_{j=1}^{k} e^{x_j})$. f is a convex function known as the *soft-max*, since it provides a smooth approximation of $(x_1, \ldots, x_k) \mapsto \max(x_1, \ldots, x_k)$. Then, problem (9.24) is similar to (9.22) modulo the replacement of the max operator with the soft-max function just described.

9.6 Chapter notes

The margin-based generalization bound for multi-class classification presented in theorem 9.2 is due to Kuznetsov, Mohri, and Syed [2014]. It admits only a linear dependency on the number of classes. This improves over a similar result by Koltchinskii and Panchenko [2002], which admits a quadratic dependency on the number of classes. Proposition 9.3 bounding the Rademacher complexity of multi-class kernel-based hypotheses and corollary 9.4 are new.

An algorithm generalizing SVMs to the multi-class classification setting was first introduced by Weston and Watkins [1999]. The optimization problem for that algorithm was based on $k(k-1)/2$ slack variables for a problem with k classes and thus could be inefficient for a relatively large number of classes. A simplification of that algorithm by replacing the sum of the slack variables $\sum_{j \neq i} \xi_{ij}$ related to point x_i by its maximum $\xi_i = \max_{j \neq i} \xi_{ij}$ considerably reduces the number of variables and leads to the multi-class SVM algorithm presented in this chapter [Crammer and Singer, 2001, 2002].

The AdaBoost.MH algorithm is presented and discussed by Schapire and Singer [1999, 2000]. As we showed in this chapter, the algorithm is a special instance of AdaBoost. Another boosting-type algorithm for multi-class classification, AdaBoost.MR, is presented by Schapire and Singer [1999, 2000]. That algorithm is also a special instance of the RankBoost algorithm presented in chapter 10. See exercise 10.5 for a detailed analysis of this algorithm, including generalization bounds.

The most commonly used tools for learning decision trees are CART (classification and regression tree) [Breiman et al., 1984] and C4.5 [Quinlan, 1986, 1993]. The greedy technique we described for learning decision trees benefits in fact from an interesting analysis: remarkably, it has been shown by Kearns and Mansour [1999], Mansour and McAllester [1999] that, under a weak learner hypothesis assumption, such decision tree algorithms produce a strong hypothesis. The grow-then-prune method is from CART. It has been analyzed by a variety of different studies, in particular by Kearns and Mansour [1998] and Mansour and McAllester [2000], who give generalization bounds for the resulting decision trees with respect to the error and size of the best sub-tree of the original tree pruned. Hardness of ERM for decision trees of a fixed size was shown by Grigni et al. [2000].

The idea of the ECOC framework for multi-class classification is due to Dietterich and Bakiri [1995]. Allwein et al. [2000] further extended and analyzed this method to margin-based losses, for which they presented a bound on the empirical error and a generalization bound in the more specific case of boosting. While the OVA technique is in general subject to a calibration issue and does not have any justification, it is very commonly used in practice. Rifkin [2002] reports the results of extensive experiments with several multi-class classification algorithms that are rather favorable to the OVA technique, with performances often very close or better than for those of several uncombined algorithms, unlike what has been claimed by some authors (see also Rifkin and Klautau [2004]).

The CRFs algorithm was introduced by Lafferty, McCallum, and Pereira [2001]. M^{3}N is due to Taskar, Guestrin, and Koller [2003] and StructSVM was presented by Tsochantaridis, Joachims, Hofmann, and Altun [2005]. An alternative technique for tackling structured prediction as a regression problem was presented and analyzed by Cortes, Mohri, and Weston [2007c].

9.7 Exercises

9.1 Generalization bounds for multi-label case. Use similar techniques to those used in the proof of theorem 9.2 to derive a margin-based learning bound in the multi-label case.

9.2 Multi-class classification with kernel-based hypotheses constrained by an L_p norm. Use corollary 9.4 to define alternative multi-class classification algorithms with kernel-based hypotheses constrained by an L_p norm with $p \neq 2$. For which value of $p \geq 1$ is the bound of proposition 9.3 tightest? Derive the dual optimization of the multi-class classification algorithm defined with $p = \infty$.

9.3 Alternative multi-class boosting algorithm. Consider the objective function G defined for any sample $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$ and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{R}^n$, $n \geq 1$, by

$$G(\boldsymbol{\alpha}) = \sum_{i=1}^{m} e^{-\frac{1}{k}\sum_{l=1}^{k} y_i[l] f_n(x_i, l)} = \sum_{i=1}^{m} e^{-\frac{1}{k}\sum_{l=1}^{k} y_i[l] \sum_{t=1}^{n} \alpha_t h_t(x_i, l)}. \tag{9.25}$$

Use the convexity of the exponential function to compare G with the objective function F defining AdaBoost.MH. Show that G is a convex function upper bounding the multi-label multi-class error. Discuss the properties of G and derive an algorithm defined by the application of coordinate descent to G. Give theoretical guarantees for the performance of the algorithm and analyze its running-time complexity when using boosting stumps.

9.4 Multi-class algorithm based on RankBoost. This problem requires familiarity with the material presented both in this chapter and in chapter 10. An alternative boosting-type multi-class classification algorithm is one based on a ranking criterion. We will define and examine that algorithm in the mono-label setting. Let $\mathcal{H}$ be a family of base hypotheses mapping $\mathcal{X} \times \mathcal{Y}$ to $\{-1, +1\}$. Let F be the following objective function defined for all samples $S = ((x_1, y_1), \dots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$ and $\bar{\boldsymbol{\alpha}} = (\bar{\alpha}_1, \dots, \bar{\alpha}_N) \in \mathbb{R}^N$, $N \geq 1$, by

$$F(\bar{\boldsymbol{\alpha}}) = \sum_{i=1}^{m} \sum_{l \neq y_i} e^{-(f_N(x_i, y_i) - f_N(x_i, l))} = \sum_{i=1}^{m} \sum_{l \neq y_i} e^{-\sum_{j=1}^{N} \bar{\alpha}_j (h_j(x_i, y_i) - h_j(x_i, l))}. \tag{9.26}$$

where $f_N = \sum_{j=1}^{N} \bar{\alpha}_j h_j$.

(a) Show that F is convex and differentiable.

(b) Show that $\frac{1}{m} \sum_{i=1}^{m} 1_{\rho_{f_N}(x_i, y_i)} \leq \frac{1}{k-1} F(\bar{\boldsymbol{\alpha}})$, where $f_N = \sum_{j=1}^{N} \bar{\alpha}_j h_j$.

(c) Give the pseudocode of the algorithm obtained by applying coordinate descent to F. The resulting algorithm is known as AdaBoost.MR. Show that AdaBoost.MR exactly coincides with the RankBoost algorithm applied to the problem of ranking pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$. Describe exactly the ranking target for these pairs.

(d) Use question (9.4b) and the learning bounds of this chapter to derive margin-based generalization bounds for this algorithm.

(e) Use the connection of the algorithm with RankBoost and the learning bounds of chapter 10 to derive alternative generalization bounds for this algorithm. Compare these bounds with those of the previous question.

9.5 Decision trees. Show that VC-dimension of a binary decision tree with n nodes in dimension N is in $O(n \log N)$.

9.6 Give an example where the generalization error of each of the $k(k-1)/2$ binary classifiers $h_{ll'}$, $l \neq l'$, used in the definition of the OVO technique is r and that of the OVO hypothesis $(k-1)r$.