17

Multiclass, Ranking, and Complex Prediction Problems

Multiclass categorization is the problem of classifying instances into one of several possible target classes. That is, we are aiming at learning a predictor $h : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y}$ is a finite set of categories. Applications include, for example, categorizing documents according to topic ($\mathcal{X}$ is the set of documents and $\mathcal{Y}$ is the set of possible topics) or determining which object appears in a given image ($\mathcal{X}$ is the set of images and $\mathcal{Y}$ is the set of possible objects).

The centrality of the multiclass learning problem has spurred the development of various approaches for tackling the task. Perhaps the most straightforward approach is a reduction from multiclass classification to binary classification. In Section 17.1 we discuss the most common two reductions as well as the main drawback of the reduction approach.

We then turn to describe a family of linear predictors for multiclass problems. Relying on the RLM and SGD frameworks from previous chapters, we describe several practical algorithms for multiclass prediction.

In Section 17.3 we show how to use the multiclass machinery for complex prediction problems in which $\mathcal{Y}$ can be extremely large but has some structure on it. This task is often called *structured output learning*. In particular, we demonstrate this approach for the task of recognizing handwritten words, in which $\mathcal{Y}$ is the set of all possible strings of some bounded length (hence, the size of $\mathcal{Y}$ is exponential in the maximal length of a word).

Finally, in Section 17.4 and Section 17.5 we discuss ranking problems in which the learner should order a set of instances according to their "relevance." A typical application is ordering results of a search engine according to their relevance to the query. We describe several performance measures that are adequate for assessing the performance of ranking predictors and describe how to learn linear predictors for ranking problems efficiently.

17.1 ONE-VERSUS-ALL AND ALL-PAIRS

The simplest approach to tackle multiclass prediction problems is by reduction to binary classification. Recall that in multiclass prediction we would like to learn a function $h : \mathcal{X} \to \mathcal{Y}$. Without loss of generality let us denote $\mathcal{Y} = \{1, \ldots, k\}$.

In the One-versus-All method (a.k.a. One-versus-Rest) we train k binary classifiers, each of which discriminates between one class and the rest of the classes. That is, given a training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$, where every y_i is in $\mathcal{Y}$, we construct k binary training sets, $S_1, \ldots, S_k$, where $S_i = (\mathbf{x}_1, (-1)^{\mathbb{1}_{[y_1 \neq i]}}), \ldots, (\mathbf{x}_m, (-1)^{\mathbb{1}_{[y_m \neq i]}})$. In words, S_i is the set of instances labeled 1 if their label in S was i, and -1 otherwise. For every $i \in [k]$ we train a binary predictor $h_i : \mathcal{X} \to \{\pm 1\}$ based on S_i, hoping that $h_i(\mathbf{x})$ should equal 1 if and only if $\mathbf{x}$ belongs to class i. Then, given $h_1, \ldots, h_k$, we construct a multiclass predictor using the rule

$$h(\mathbf{x}) \in \operatorname*{argmax}_{i \in [k]} h_i(\mathbf{x}). \tag{17.1}$$

When more than one binary hypothesis predicts "1" we should somehow decide which class to predict (e.g., we can arbitrarily decide to break ties by taking the minimal index in $\operatorname{argmax}_i h_i(\mathbf{x})$). A better approach can be applied whenever each h_i hides additional information, which can be interpreted as the confidence in the prediction $y = i$. For example, this is the case in halfspaces, where the actual prediction is $\operatorname{sign}(\langle \mathbf{w}, \mathbf{x} \rangle)$, but we can interpret $\langle \mathbf{w}, \mathbf{x} \rangle$ as the confidence in the prediction. In such cases, we can apply the multiclass rule given in Equation (17.1) on the real valued predictions. A pseudocode of the One-versus-All approach is given in the following.

One-versus-All

```
input:
  training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
  algorithm for binary classification $A$
foreach $i \in \mathcal{Y}$
  let $S_i = (\mathbf{x}_1, (-1)^{\mathbb{1}_{[y_1 \neq i]}}), \ldots, (\mathbf{x}_m, (-1)^{\mathbb{1}_{[y_m \neq i]}})$
  let $h_i = A(S_i)$
output:
  the multiclass hypothesis defined by $h(\mathbf{x}) \in \operatorname{argmax}_{i \in \mathcal{Y}} h_i(\mathbf{x})$
```

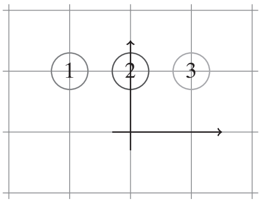
Another popular reduction is the *All-Pairs* approach, in which all pairs of classes are compared to each other. Formally, given a training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$, where every y_i is in $[k]$, for every $1 \leq i < j \leq k$ we construct a binary training sequence, $S_{i,j}$, containing all examples from S whose label is either i or j. For each such an example, we set the binary label in $S_{i,j}$ to be $+1$ if the multiclass label in S is i and -1 if the multiclass label in S is j. Next, we train a binary classification algorithm based on every $S_{i,j}$ to get $h_{i,j}$. Finally, we construct a multiclass classifier by predicting the class that had the highest number of "wins." A pseudocode of the All-Pairs approach is given in the following.

All-Pairs

```
input:
  training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
  algorithm for binary classification $A$
foreach $i, j \in \mathcal{Y}$ s.t. $i < j$
  initialize $S_{i,j}$ to be the empty sequence
  for $t = 1, \ldots, m$
    If $y_t = i$ add $(\mathbf{x}_t, 1)$ to $S_{i,j}$
    If $y_t = j$ add $(\mathbf{x}_t, -1)$ to $S_{i,j}$
  let $h_{i,j} = A(S_{i,j})$
output:
  the multiclass hypothesis defined by
  $h(\mathbf{x}) \in \operatorname{argmax}_{i \in \mathcal{Y}} \left( \sum_{j \in \mathcal{Y}} \operatorname{sign}(j - i)\, h_{i,j}(\mathbf{x}) \right)$
```

Although reduction methods such as the One-versus-All and All-Pairs are simple and easy to construct from existing algorithms, their simplicity has a price. The binary learner is not aware of the fact that we are going to use its output hypotheses for constructing a multiclass predictor, and this might lead to suboptimal results, as illustrated in the following example.

Example 17.1. Consider a multiclass categorization problem in which the instance space is $\mathcal{X} = \mathbb{R}^2$ and the label set is $\mathcal{Y} = \{1, 2, 3\}$. Suppose that instances of the different classes are located in nonintersecting balls as depicted in the following.



Suppose that the probability masses of classes $1, 2, 3$ are 40%, 20%, and 40%, respectively. Consider the application of One-versus-All to this problem, and assume that the binary classification algorithm used by One-versus-All is ERM with respect to the hypothesis class of halfspaces. Observe that for the problem of discriminating between class 2 and the rest of the classes, the optimal halfspace would be the all negative classifier. Therefore, the multiclass predictor constructed by One-versus-All might err on all the examples from class 2 (this will be the case if the tie in the definition of $h(\mathbf{x})$ is broken by the numerical value of the class label). In contrast, if we choose $h_i(\mathbf{x}) = \langle \mathbf{w}_i, \mathbf{x} \rangle$, where $\mathbf{w}_1 = \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$, $\mathbf{w}_2 = (0, 1)$, and $\mathbf{w}_3 = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$, then the classifier defined by $h(\mathbf{x}) = \operatorname{argmax}_i h_i(\mathbf{x})$ perfectly predicts all the examples. We see that even though the approximation error of the class of predictors of the form $h(\mathbf{x}) = \operatorname{argmax}_i \langle \mathbf{w}_i, \mathbf{x} \rangle$ is zero, the One-versus-All approach might fail to find a good predictor from this class.

17.2 LINEAR MULTICLASS PREDICTORS

In light of the inadequacy of reduction methods, in this section we study a more direct approach for learning multiclass predictors. We describe the family of linear multiclass predictors. To motivate the construction of this family, recall that a linear predictor for binary classification (i.e., a halfspace) takes the form

$$h(\mathbf{x}) = \operatorname{sign}(\langle \mathbf{w}, \mathbf{x} \rangle).$$

An equivalent way to express the prediction is as follows:

$$h(\mathbf{x}) = \operatorname*{argmax}_{y \in \{\pm 1\}} \langle \mathbf{w}, y\mathbf{x} \rangle,$$

where $y\mathbf{x}$ is the vector obtained by multiplying each element of $\mathbf{x}$ by y.

This representation leads to a natural generalization of halfspaces to multiclass problems as follows. Let $\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ be a *class-sensitive feature mapping*. That is, Ψ takes as input a pair $(\mathbf{x}, y)$ and maps it into a d dimensional feature vector. Intuitively, we can think of the elements of $\Psi(\mathbf{x}, y)$ as score functions that assess how well the label y fits the instance $\mathbf{x}$. We will elaborate on Ψ later on. Given Ψ and a vector $\mathbf{w} \in \mathbb{R}^d$, we can define a multiclass predictor, $h : \mathcal{X} \to \mathcal{Y}$, as follows:

$$h(\mathbf{x}) = \operatorname*{argmax}_{y \in \mathcal{Y}} \langle \mathbf{w}, \Psi(\mathbf{x}, y) \rangle.$$

That is, the prediction of h for the input $\mathbf{x}$ is the label that achieves the highest weighted score, where weighting is according to the vector $\mathbf{w}$.

Let W be some set of vectors in $\mathbb{R}^d$, for example, $W = \{\mathbf{w} \in \mathbb{R}^d : \|\mathbf{w}\| \le B\}$, for some scalar $B > 0$. Each pair (Ψ, W) defines a hypothesis class of multiclass predictors:

$$\mathcal{H}_{\Psi,W} = \{\mathbf{x} \mapsto \operatorname*{argmax}_{y \in \mathcal{Y}} \langle \mathbf{w}, \Psi(\mathbf{x}, y) \rangle \ : \ \mathbf{w} \in W\}.$$

Of course, the immediate question, which we discuss in the sequel, is how to construct a good Ψ. Note that if $\mathcal{Y} = \{\pm 1\}$ and we set $\Psi(\mathbf{x}, y) = y\mathbf{x}$ and $W = \mathbb{R}^d$, then $\mathcal{H}_{\Psi,W}$ becomes the hypothesis class of homogeneous halfspace predictors for binary classification.

17.2.1 How to Construct Ψ

As mentioned before, we can think of the elements of $\Psi(\mathbf{x}, y)$ as score functions that assess how well the label y fits the instance $\mathbf{x}$. Naturally, designing a good Ψ is similar to the problem of designing a good feature mapping (as we discussed in Chapter 16 and as we will discuss in more detail in Chapter 25). Two examples of useful constructions are given in the following.

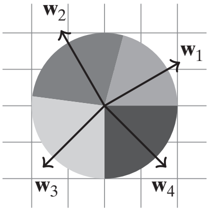
The Multivector Construction:
Let $\mathcal{Y} = \{1, \ldots, k\}$ and let $\mathcal{X} = \mathbb{R}^n$. We define $\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$, where $d = nk$, as follows

$$\Psi(\mathbf{x},\ y) = [\ \underbrace{0, \ldots, 0}_{\in \mathbb{R}^{(y-1)n}},\ \underbrace{x_1, \ldots, x_n}_{\in \mathbb{R}^n},\ \underbrace{0, \ldots, 0}_{\in \mathbb{R}^{(k-y)n}}\ ]. \tag{17.2}$$

That is, $\Psi(\mathbf{x}, y)$ is composed of k vectors, each of which is of dimension n, where we set all the vectors to be the all zeros vector except the y'th vector, which is set to be $\mathbf{x}$. It follows that we can think of $\mathbf{w} \in \mathbb{R}^{nk}$ as being composed of k weight vectors in $\mathbb{R}^n$, that is, $\mathbf{w} = [\mathbf{w}_1; \; \ldots \; ; \mathbf{w}_k]$, hence the name *multivector construction*. By the construction we have that $\langle \mathbf{w}, \Psi(\mathbf{x}, y)\rangle = \langle \mathbf{w}_y, \mathbf{x}\rangle$, and therefore the multiclass prediction becomes

$$h(\mathbf{x}) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \; \langle \mathbf{w}_y, \mathbf{x}\rangle.$$

A geometric illustration of the multiclass prediction over $\mathcal{X} = \mathbb{R}^2$ is given in the following.



TF-IDF:

The previous definition of $\Psi(\mathbf{x}, \; y)$ does not incorporate any prior knowledge about the problem. We next describe an example of a feature function Ψ that does incorporate prior knowledge. Let $\mathcal{X}$ be a set of text documents and $\mathcal{Y}$ be a set of possible topics. Let d be a size of a dictionary of words. For each word in the dictionary, whose corresponding index is j, let $TF(j, \mathbf{x})$ be the number of times the word corresponding to j appears in the document $\mathbf{x}$. This quantity is called Term-Frequency. Additionally, let $DF(j, y)$ be the number of times the word corresponding to j appears in documents in our training set that are not about topic y. This quantity is called Document-Frequency and measures whether word j is frequent in other topics. Now, define $\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ to be such that

$$\Psi_j(\mathbf{x}, y) = TF(j, \mathbf{x}) \log\left(\frac{m}{DF(j, y)}\right),$$

where m is the total number of documents in our training set. The preceding quantity is called term-frequency-inverse-document-frequency or TF-IDF for short. Intuitively, $\Psi_j(\mathbf{x}, \; y)$ should be large if the word corresponding to j appears a lot in the document $\mathbf{x}$ but does not appear at all in documents that are not on topic y. If this is the case, we tend to believe that the document $\mathbf{x}$ is on topic y. Note that unlike the multivector construction described previously, in the current construction the dimension of Ψ does not depend on the number of topics (i.e., the size of $\mathcal{Y}$).

17.2.2 Cost-Sensitive Classification

So far we used the zero-one loss as our performance measure of the quality of $h(\mathbf{x})$. That is, the loss of a hypothesis h on an example $(\mathbf{x}, y)$ is 1 if $h(\mathbf{x}) \neq y$ and 0 otherwise. In some situations it makes more sense to penalize different levels of loss for different mistakes. For example, in object recognition tasks, it is less severe to predict that an image of a tiger contains a cat than predicting that the image contains a whale. This can be modeled by specifying a loss function, $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$, where for every pair of labels, y', y, the loss of predicting the label y' when the correct label is y is defined to be $\Delta(y', y)$. We assume that $\Delta(y, y) = 0$. Note that the zero-one loss can be easily modeled by setting $\Delta(y', y) = \mathbb{1}_{[y' \neq y]}$.

17.2.3 ERM

We have defined the hypothesis class $\mathcal{H}_{\Psi,W}$ and specified a loss function Δ. To learn the class with respect to the loss function, we can apply the ERM rule with respect to this class. That is, we search for a multiclass hypothesis $h \in \mathcal{H}_{\Psi,W}$, parameterized by a vector $\mathbf{w}$, that minimizes the empirical risk with respect to Δ,

$$L_S(h) = \frac{1}{m} \sum_{i=1}^{m} \Delta(h(\mathbf{x}_i), y_i).$$

We now show that when $W = \mathbb{R}^d$ and we are in the realizable case, then it is possible to solve the ERM problem efficiently using linear programming. Indeed, in the realizable case, we need to find a vector $\mathbf{w} \in \mathbb{R}^d$ that satisfies

$$\forall i \in [m], \quad y_i = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \langle \mathbf{w}, \Psi(\mathbf{x}_i, y) \rangle.$$

Equivalently, we need that $\mathbf{w}$ will satisfy the following set of linear inequalities

$$\forall i \in [m],\ \forall y \in \mathcal{Y} \setminus \{y_i\}, \quad \langle \mathbf{w}, \Psi(\mathbf{x}_i, y_i) \rangle > \langle \mathbf{w}, \Psi(\mathbf{x}_i, y) \rangle.$$

Finding $\mathbf{w}$ that satisfies the preceding set of linear equations amounts to solving a linear program.

As in the case of binary classification, it is also possible to use a generalization of the Perceptron algorithm for solving the ERM problem. See Exercise 17.2.

In the nonrealizable case, solving the ERM problem is in general computationally hard. We tackle this difficulty using the method of convex surrogate loss functions (see Section 12.3). In particular, we generalize the hinge loss to multiclass problems.

17.2.4 Generalized Hinge Loss

Recall that in binary classification, the hinge loss is defined to be $\max\{0, 1 - y\langle \mathbf{w}, \mathbf{x} \rangle\}$. We now generalize the hinge loss to multiclass predictors of the form

$$h_{\mathbf{w}}(\mathbf{x}) = \underset{y' \in \mathcal{Y}}{\operatorname{argmax}} \langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{y}') \rangle.$$

Recall that a surrogate convex loss should upper bound the original nonconvex loss, which in our case is $\Delta(h_{\mathbf{w}}(\mathbf{x}), y)$. To derive an upper bound on $\Delta(h_{\mathbf{w}}(\mathbf{x}), y)$ we first note that the definition of $h_{\mathbf{w}}(\mathbf{x})$ implies that

$$\langle \mathbf{w}, \Psi(\mathbf{x}, y) \rangle \leq \langle \mathbf{w}, \Psi(\mathbf{x}, h_{\mathbf{w}}(\mathbf{x})) \rangle.$$

Therefore,

$$\Delta(h_{\mathbf{w}}(\mathbf{x}), y) \ \leq\ \Delta(h_{\mathbf{w}}(\mathbf{x}), y) + \langle \mathbf{w}, \Psi(\mathbf{x}, h_{\mathbf{w}}(\mathbf{x})) - \Psi(\mathbf{x}, y) \rangle.$$

Since $h_{\mathbf{w}}(\mathbf{x}) \in \mathcal{Y}$ we can upper bound the right-hand side of the preceding by

$$\max_{y' \in \mathcal{Y}} \left(\Delta(y', y) + \langle \mathbf{w}, \Psi(\mathbf{x}, y') - \Psi(\mathbf{x}, y) \rangle \right) \overset{\text{def}}{=} \ell(\mathbf{w}, (\mathbf{x}, y)). \tag{17.3}$$

We use the term "generalized hinge loss" to denote the preceding expression. As we have shown, $\ell(\mathbf{w}, (\mathbf{x}, y)) \geq \Delta(h_{\mathbf{w}}(\mathbf{x}), y)$. Furthermore, equality holds whenever the score of the correct label is larger than the score of any other label, y', by at least $\Delta(y', y)$, namely,

$$\forall y' \in \mathcal{Y} \setminus \{y\}, \quad \langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{y}) \rangle \geq \langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{y}') \rangle + \Delta(y', y).$$

It is also immediate to see that $\ell(\mathbf{w}, (\mathbf{x}, y))$ is a convex function with respect to $\mathbf{w}$ since it is a maximum over linear functions of $\mathbf{w}$ (see Claim 12.5 in Chapter 12), and that $\ell(\mathbf{w}, (\mathbf{x}, y))$ is ρ-Lipschitz with $\rho = \max_{y' \in \mathcal{Y}} \|\Psi(\mathbf{x}, y') - \Psi(\mathbf{x}, y)\|$.

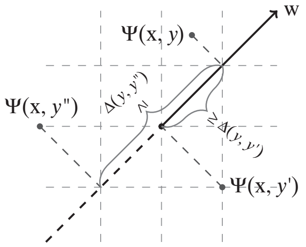
Remark 17.2. We use the name "generalized hinge loss" since in the binary case, when $\mathcal{Y} = \{\pm 1\}$, if we set $\Psi(\mathbf{x}, y) = \frac{y\mathbf{x}}{2}$, then the generalized hinge loss becomes the vanilla hinge loss for binary classification,

$$\ell(\mathbf{w}, (\mathbf{x}, y)) = \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x} \rangle\}.$$

Geometric Intuition:
The feature function $\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ maps each $\mathbf{x}$ into $|\mathcal{Y}|$ vectors in $\mathbb{R}^d$. The value of $\ell(\mathbf{w}, (\mathbf{x}, y))$ will be zero if there exists a direction $\mathbf{w}$ such that when projecting the $|\mathcal{Y}|$ vectors onto this direction we obtain that each vector is represented by the scalar $\langle \mathbf{w}, \Psi(\mathbf{x}, y) \rangle$, and we can rank the different points on the basis of these scalars so that

- The point corresponding to the correct y is top-ranked
- For each $y' \neq y$, the difference between $\langle \mathbf{w}, \Psi(\mathbf{x}, y) \rangle$ and $\langle \mathbf{w}, \Psi(\mathbf{x}, y') \rangle$ is larger than the loss of predicting y' instead of y. The difference $\langle \mathbf{w}, \Psi(\mathbf{x}, y) \rangle - \langle \mathbf{w}, \Psi(\mathbf{x}, y') \rangle$ is also referred to as the "margin" (see Section 15.1).

This is illustrated in the following figure:



17.2.5 Multiclass SVM and SGD

Once we have defined the generalized hinge loss, we obtain a convex-Lipschitz learning problem and we can apply our general techniques for solving such problems. In particular, the RLM technique we have studied in Chapter 13 yields the multiclass SVM rule:

```
Multiclass SVM

input: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
parameters:
  regularization parameter $\lambda > 0$
  loss function $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$
  class-sensitive feature mapping $\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$
solve:
  $\min_{\mathbf{w} \in \mathbb{R}^d} \left( \lambda \|\mathbf{w}\|^2 + \frac{1}{m} \sum_{i=1}^{m} \max_{y' \in \mathcal{Y}} \left( \Delta(y', y_i) + \langle \mathbf{w}, \Psi(\mathbf{x}_i, y') - \Psi(\mathbf{x}_i, y_i) \rangle \right) \right)$
output the predictor $h_{\mathbf{w}}(\mathbf{x}) = \mathrm{argmax}_{y \in \mathcal{Y}} \langle \mathbf{w}, \Psi(\mathbf{x}, y) \rangle$
```

We can solve the optimization problem associated with multiclass SVM using generic convex optimization algorithms (or using the method described in Section 15.5). Let us analyze the risk of the resulting hypothesis. The analysis seamlessly follows from our general analysis for convex-Lipschitz problems given in Chapter 13. In particular, applying Corollary 13.8 and using the fact that the generalized hinge loss upper bounds the Δ loss, we immediately obtain an analog of Corollary 15.7:

Corollary 17.1. *Let $\mathcal{D}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$, let $\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$, and assume that for all $\mathbf{x} \in \mathcal{X}$ and $y \in \mathcal{Y}$ we have $\|\Psi(\mathbf{x}, y)\| \le \rho/2$. Let $B > 0$. Consider running Multiclass SVM with $\lambda = \sqrt{\frac{2\rho^2}{B^2 m}}$ on a training set $S \sim \mathcal{D}^m$ and let $h_{\mathbf{w}}$ be the output of Multiclass SVM. Then,*

$$\mathop{\mathbb{E}}_{S \sim \mathcal{D}^m} [L_{\mathcal{D}}^{\Delta}(h_{\mathbf{w}})] \le \mathop{\mathbb{E}}_{S \sim \mathcal{D}^m} [L_{\mathcal{D}}^{\text{g-hinge}}(\mathbf{w})] \le \min_{\mathbf{u} : \|\mathbf{u}\| \le B} L_{\mathcal{D}}^{\text{g-hinge}}(\mathbf{u}) + \sqrt{\frac{8\rho^2 B^2}{m}},$$

where $L_{\mathcal{D}}^{\Delta}(h) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}} [\Delta(h(\mathbf{x}), y)]$ and $L_{\mathcal{D}}^{\text{g-hinge}}(\mathbf{w}) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}} [\ell(\mathbf{w}, (\mathbf{x}, y))]$ with ℓ being the generalized hinge-loss as defined in Equation (17.3).

We can also apply the SGD learning framework for minimizing $L_{\mathcal{D}}^{\text{g-hinge}}(\mathbf{w})$ as described in Chapter 14. Recall Claim 14.6, which dealt with subgradients of max functions. In light of this claim, in order to find a subgradient of the generalized hinge loss all we need to do is to find $y \in \mathcal{Y}$ that achieves the maximum in the definition of the generalized hinge loss. This yields the following algorithm:

```
SGD for Multiclass Learning

parameters:
  Scalar $\eta > 0$, integer $T > 0$
  loss function $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$
  class-sensitive feature mapping $\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$
initialize: $\mathbf{w}^{(1)} = \mathbf{0} \in \mathbb{R}^d$
for $t = 1,\ 2, \ldots, T$
  sample $(\mathbf{x},\ y) \sim \mathcal{D}$
  find $\hat{y} \in \operatorname{argmax}_{y' \in \mathcal{Y}} \left(\Delta(y', y) + \langle \mathbf{w}^{(t)}, \Psi(\mathbf{x}, y') - \Psi(\mathbf{x}, y)\rangle\right)$
  set $\mathbf{v}_t = \Psi(\mathbf{x}, \hat{y}) - \Psi(\mathbf{x}, y)$
  update $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{v}_t$
output $\bar{\mathbf{w}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{w}^{(t)}$
```

Our general analysis of SGD given in Corollary 14.12 immediately implies:

Corollary 17.2. *Let $\mathcal{D}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$, let $\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$, and assume that for all $\mathbf{x} \in \mathcal{X}$ and $y \in \mathcal{Y}$ we have $\|\Psi(\mathbf{x}, y)\| \le \rho/2$. Let $B > 0$. Then, for every $\epsilon > 0$, if we run SGD for multiclass learning with a number of iterations (i.e., number of examples)*

$$T \ge \frac{B^2 \rho^2}{\epsilon^2}$$

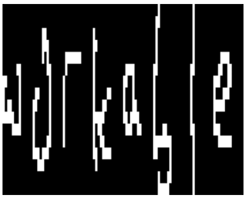
and with $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$, then the output of SGD satisfies

$$\mathop{\mathbb{E}}_{S \sim \mathcal{D}^m}[L_{\mathcal{D}}^{\Delta}(h_{\bar{\mathbf{w}}})] \le \mathop{\mathbb{E}}_{S \sim \mathcal{D}^m}[L_{\mathcal{D}}^{\text{g-hinge}}(\bar{\mathbf{w}})] \le \min_{\mathbf{u}:\|\mathbf{u}\| \le B} L_{\mathcal{D}}^{\text{g-hinge}}(\mathbf{u}) + \epsilon.$$

Remark 17.3. It is interesting to note that the risk bounds given in Corollary 17.1 and Corollary 17.2 do not depend explicitly on the size of the label set $\mathcal{Y}$, a fact we will rely on in the next section. However, the bounds may depend implicitly on the size of $\mathcal{Y}$ via the norm of $\Psi(\mathbf{x}, y)$ and the fact that the bounds are meaningful only when there exists some vector $\mathbf{u}$, $\|\mathbf{u}\| \le B$, for which $L_{\mathcal{D}}^{\text{g-hinge}}(\mathbf{u})$ is not excessively large.

17.3 STRUCTURED OUTPUT PREDICTION

Structured output prediction problems are multiclass problems in which $\mathcal{Y}$ is very large but is endowed with a predefined structure. The structure plays a key role in constructing efficient algorithms. To motivate structured learning problems, consider the problem of optical character recognition (OCR). Suppose we receive an image of some handwritten word and would like to predict which word is written in the image. To simplify the setting, suppose we know how to segment the image into a sequence of images, each of which contains a patch of the image corresponding to a single letter. Therefore, $\mathcal{X}$ is the set of sequences of images and $\mathcal{Y}$ is the set of sequences of letters. Note that the size of $\mathcal{Y}$ grows exponentially with the maximal length of a word. An example of an image $\mathbf{x}$ corresponding to the label $\mathbf{y} =$ "workable" is given in the following.



To tackle structure prediction we can rely on the family of linear predictors described in the previous section. In particular, we need to define a reasonable loss function for the problem, Δ, as well as a good class-sensitive feature mapping, Ψ. By "good" we mean a feature mapping that will lead to a low approximation error for the class of linear predictors with respect to Ψ and Δ. Once we do this, we can rely, for example, on the SGD learning algorithm defined in the previous section.

However, the huge size of $\mathcal{Y}$ poses several challenges:

1. To apply the multiclass prediction we need to solve a maximization problem over $\mathcal{Y}$. How can we predict efficiently when $\mathcal{Y}$ is so large?
2. How do we train $\mathbf{w}$ efficiently? In particular, to apply the SGD rule we again need to solve a maximization problem over $\mathcal{Y}$.
3. How can we avoid overfitting?

In the previous section we have already shown that the sample complexity of learning a linear multiclass predictor does not depend explicitly on the number of classes. We just need to make sure that the norm of the range of Ψ is not too large. This will take care of the overfitting problem. To tackle the computational challenges we rely on the structure of the problem, and define the functions Ψ and Δ so that calculating the maximization problems in the definition of $h_{\mathbf{w}}$ and in the SGD algorithm can be performed efficiently. In the following we demonstrate one way to achieve these goals for the OCR task mentioned previously.

To simplify the presentation, let us assume that all the words in $\mathcal{Y}$ are of length r and that the number of different letters in our alphabet is q. Let $\mathbf{y}$ and $\mathbf{y}'$ be two words (i.e., sequences of letters) in $\mathcal{Y}$. We define the function $\Delta(\mathbf{y}', \mathbf{y})$ to be the average number of letters that are different in y' and y, namely, $\frac{1}{r}\sum_{i=1}^{r} \mathbb{1}_{[y_i \neq y'_i]}$.

Next, let us define a class-sensitive feature mapping $\Psi(\mathbf{x}, \mathbf{y})$. It will be convenient to think about $\mathbf{x}$ as a matrix of size $n \times r$, where n is the number of pixels in each image, and r is the number of images in the sequence. The j'th column of $\mathbf{x}$ corresponds to the j'th image in the sequence (encoded as a vector of gray level values of pixels). The dimension of the range of Ψ is set to be $d = nq + q^2$.

The first nq feature functions are "type 1" features and take the form:

$$\Psi_{i,j,1}(\mathbf{x}, \mathbf{y}) = \frac{1}{r}\sum_{t=1}^{r} x_{i,t}\, \mathbb{1}_{[y_t = j]}.$$

That is, we sum the value of the i'th pixel only over the images for which $\mathbf{y}$ assigns the letter j. The triple index $(i, j, 1)$ indicates that we are dealing with feature (i, j) of type 1. Intuitively, such features can capture pixels in the image whose gray level values are indicative of a certain letter. The second type of features take the form

$$\Psi_{i,j,2}(\mathbf{x},\mathbf{y}) = \frac{1}{r}\sum_{t=2}^{r} \mathbb{1}_{[y_t=i]}\,\mathbb{1}_{[y_{t-1}=j]}.$$

That is, we sum the number of times the letter i follows the letter j. Intuitively, these features can capture rules like "It is likely to see the pair 'qu' in a word" or "It is unlikely to see the pair 'rz' in a word." Of course, some of these features will not be very useful, so the goal of the learning process is to assign weights to features by learning the vector $\mathbf{w}$, so that the weighted score will give us a good prediction via

$$h_{\mathbf{w}}(\mathbf{x}) = \underset{\mathbf{y}\in\mathcal{Y}}{\operatorname{argmax}}\ \langle \mathbf{w}, \Psi(\mathbf{x},\mathbf{y})\rangle.$$

It is left to show how to solve the optimization problem in the definition of $h_{\mathbf{w}}(\mathbf{x})$ efficiently, as well as how to solve the optimization problem in the definition of $\hat{y}$ in the SGD algorithm. We can do this by applying a dynamic programming procedure. We describe the procedure for solving the maximization in the definition of $h_{\mathbf{w}}$ and leave as an exercise the maximization problem in the definition of $\hat{y}$ in the SGD algorithm.

To derive the dynamic programming procedure, let us first observe that we can write

$$\Psi(\mathbf{x},\mathbf{y}) = \sum_{t=1}^{r} \boldsymbol{\phi}(\mathbf{x}, y_t, y_{t-1}),$$

for an appropriate $\boldsymbol{\phi} : \mathcal{X} \times [q] \times [q] \cup \{0\} \to \mathbb{R}^d$, and for simplicity we assume that y_0 is always equal to 0. Indeed, each feature function $\Psi_{i,j,1}$ can be written in terms of

$$\phi_{i,j,1}(\mathbf{x}, y_t, y_{t-1}) = x_{i,t}\,\mathbb{1}_{[y_t=j]},$$

while the feature function $\Psi_{i,j,2}$ can be written in terms of

$$\phi_{i,j,2}(\mathbf{x}, y_t, y_{t-1}) = \mathbb{1}_{[y_t=i]}\,\mathbb{1}_{[y_{t-1}=j]}.$$

Therefore, the prediction can be written as

$$h_{\mathbf{w}}(\mathbf{x}) = \underset{\mathbf{y}\in\mathcal{Y}}{\operatorname{argmax}} \sum_{t=1}^{r} \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}, y_t, y_{t-1})\rangle. \tag{17.4}$$

In the following we derive a dynamic programming procedure that solves every problem of the form given in Equation (17.4). The procedure will maintain a matrix $M \in \mathbb{R}^{q,r}$ such that

$$M_{s,\tau} = \max_{(y_1,\ldots,y_\tau):y_\tau=s} \sum_{t=1}^{\tau} \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}, y_t, y_{t-1})\rangle.$$

Clearly, the maximum of $\langle \mathbf{w}, \Psi(\mathbf{x},\mathbf{y})\rangle$ equals $\max_s M_{s,r}$. Furthermore, we can calculate M in a recursive manner:

$$M_{s,\tau} = \max_{s'} \left(M_{s',\tau-1} + \left\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}, s, s')\right\rangle\right). \tag{17.5}$$

This yields the following procedure:

Dynamic Programming for Calculating $h_{\mathbf{w}}(\mathbf{x})$ as Given in Equation (17.4)

```
input: a matrix $\mathbf{x} \in \mathbb{R}^{n,r}$ and a vector $\mathbf{w}$
initialize:
  foreach $s \in [q]$
    $M_{s,1} = \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}, s, -1) \rangle$
for $\tau = 2, \ldots, r$
  foreach $s \in [q]$
    set $M_{s,\tau}$ as in Equation (17.5)
    set $I_{s,\tau}$ to be the $s'$ that maximizes Equation (17.5)
set $y_t = \mathrm{argmax}_s\, M_{s,r}$
for $\tau = r,\ r-1, \ldots, 2$
  set $y_{\tau-1} = I_{y_\tau,\tau}$
output: $\mathbf{y} = (y_1, \ldots, y_r)$
```

17.4 RANKING

Ranking is the problem of ordering a set of instances according to their "relevance." A typical application is ordering results of a search engine according to their relevance to the query. Another example is a system that monitors electronic transactions and should alert for possible fraudulent transactions. Such a system should order transactions according to how suspicious they are.

Formally, let $\mathcal{X}^* = \bigcup_{n=1}^{\infty} \mathcal{X}^n$ be the set of all sequences of instances from $\mathcal{X}$ of arbitrary length. A ranking hypothesis, h, is a function that receives a sequence of instances $\bar{\mathbf{x}} = (\mathbf{x}_1, \ldots, \mathbf{x}_r) \in \mathcal{X}^*$, and returns a permutation of $[r]$. It is more convenient to let the output of h be a vector $\mathbf{y} \in \mathbb{R}^r$, where by sorting the elements of $\mathbf{y}$ we obtain the permutation over $[r]$. We denote by $\pi(\mathbf{y})$ the permutation over $[r]$ induced by $\mathbf{y}$. For example, for $r = 5$, the vector $\mathbf{y} = (2,\ 1,\ 6, -1,\ 0.5)$ induces the permutation $\pi(\mathbf{y}) = (4,\ 3,\ 5,\ 1,\ 2)$. That is, if we sort $\mathbf{y}$ in an ascending order, then we obtain the vector $(-1,\ 0.5,\ 1,\ 2,\ 6)$. Now, $\pi(\mathbf{y})_i$ is the position of y_i in the sorted vector $(-1,\ 0.5,\ 1,\ 2,\ 6)$. This notation reflects that the top-ranked instances are those that achieve the highest values in $\pi(\mathbf{y})$.

In the notation of our PAC learning model, the examples domain is $Z = \bigcup_{r=1}^{\infty} (\mathcal{X}^r \times \mathbb{R}^r)$, and the hypothesis class, $\mathcal{H}$, is some set of ranking hypotheses. We next turn to describe loss functions for ranking. There are many possible ways to define such loss functions, and here we list a few examples. In all the examples we define $\ell(h, (\bar{\mathbf{x}}, \mathbf{y})) = \Delta(h(\bar{\mathbf{x}}), \mathbf{y})$, for some function $\Delta : \bigcup_{r=1}^{\infty} (\mathbb{R}^r \times \mathbb{R}^r) \to \mathbb{R}_+$.

- **0–1 Ranking loss:** $\Delta(\mathbf{y}', \mathbf{y})$ is zero if $\mathbf{y}$ and $\mathbf{y}'$ induce exactly the same ranking and $\Delta(\mathbf{y}', \mathbf{y}) = 1$ otherwise. That is, $\Delta(\mathbf{y}', \mathbf{y}) = \mathbb{1}_{[\pi(\mathbf{y}') \neq \pi(\mathbf{y})]}$. Such a loss function is almost never used in practice as it does not distinguish between the case in which $\pi(\mathbf{y}')$ is almost equal to $\pi(\mathbf{y})$ and the case in which $\pi(\mathbf{y}')$ is completely different from $\pi(\mathbf{y})$.

- **Kendall-Tau Loss:** We count the number of pairs (i, j) that are in different order in the two permutations. This can be written as

$$\Delta(\mathbf{y}', \mathbf{y}) = \frac{2}{r(r-1)} \sum_{i=1}^{r-1} \sum_{j=i+1}^{r} \mathbb{1}_{[\mathrm{sign}(y'_i - y'_j) \neq \mathrm{sign}(y_i - y_j)]}.$$

  This loss function is more useful than the 0–1 loss as it reflects the level of similarity between the two rankings.
- **Normalized Discounted Cumulative Gain (NDCG):** This measure emphasizes the correctness at the top of the list by using a monotonically nondecreasing discount function $D : \mathbb{N} \to \mathbb{R}_+$. We first define a discounted cumulative gain measure:

$$G(\mathbf{y}', \mathbf{y}) = \sum_{i=1}^{r} D(\pi(\mathbf{y}')_i)\, y_i.$$

  In words, if we interpret y_i as a score of the "true relevance" of item i, then we take a weighted sum of the relevance of the elements, while the weight of y_i is determined on the basis of the position of i in $\pi(\mathbf{y}')$. Assuming that all elements of $\mathbf{y}$ are nonnegative, it is easy to verify that $0 \leq G(\mathbf{y}', \mathbf{y}) \leq G(\mathbf{y}, \mathbf{y})$. We can therefore define a normalized discounted cumulative gain by the ratio $G(\mathbf{y}', \mathbf{y})/G(\mathbf{y}, \mathbf{y})$, and the corresponding loss function would be

$$\Delta(\mathbf{y}', \mathbf{y}) = 1 - \frac{G(\mathbf{y}', \mathbf{y})}{G(\mathbf{y}, \mathbf{y})} = \frac{1}{G(\mathbf{y}, \mathbf{y})} \sum_{i=1}^{r} \left(D(\pi(\mathbf{y})_i) - D(\pi(\mathbf{y}')_i) \right) y_i.$$

  We can easily see that $\Delta(\mathbf{y}', \mathbf{y}) \in [0, 1]$ and that $\Delta(\mathbf{y}', \mathbf{y}) = 0$ whenever $\pi(\mathbf{y}') = \pi(\mathbf{y})$.

  A typical way to define the discount function is by

$$D(i) = \begin{cases} \frac{1}{\log_2(r-i+2)} & \text{if } i \in \{r-k+1, \ldots, r\} \\ 0 & \text{otherwise} \end{cases}$$

  where $k < r$ is a parameter. This means that we care more about elements that are ranked higher, and we completely ignore elements that are not at the top-k ranked elements. The NDCG measure is often used to evaluate the performance of search engines since in such applications it makes sense completely to ignore elements that are not at the top of the ranking.

Once we have a hypothesis class and a ranking loss function, we can learn a ranking function using the ERM rule. However, from the computational point of view, the resulting optimization problem might be hard to solve. We next discuss how to learn linear predictors for ranking.

17.4.1 Linear Predictors for Ranking

A natural way to define a ranking function is by projecting the instances onto some vector $\mathbf{w}$ and then outputting the resulting scalars as our representation of the ranking function. That is, assuming that $\mathcal{X} \subset \mathbb{R}^d$, for every $\mathbf{w} \in \mathbb{R}^d$ we define a ranking function

$$h_{\mathbf{w}}((\mathbf{x}_1,\ldots,\mathbf{x}_r)) = (\langle \mathbf{w},\mathbf{x}_1\rangle,\ldots,\langle \mathbf{w},\mathbf{x}_r\rangle). \tag{17.6}$$

As we discussed in Chapter 16, we can also apply a feature mapping that maps instances into some feature space and then takes the inner products with $\mathbf{w}$ in the feature space. For simplicity, we focus on the simpler form as in Equation (17.6).

Given some $W \subset \mathbb{R}^d$, we can now define the hypothesis class $\mathcal{H}_W = \{h_{\mathbf{w}} : \mathbf{w} \in W\}$. Once we have defined this hypothesis class, and have chosen a ranking loss function, we can apply the ERM rule as follows: Given a training set, $S = (\bar{\mathbf{x}}_1,\mathbf{y}_1),\ldots,(\bar{\mathbf{x}}_m,\mathbf{y}_m)$, where each $(\bar{\mathbf{x}}_i,\mathbf{y}_i)$ is in $(\mathcal{X}\times\mathbb{R})^{r_i}$, for some $r_i \in \mathbb{N}$, we should search $\mathbf{w} \in W$ that minimizes the empirical loss, $\sum_{i=1}^m \Delta(h_{\mathbf{w}}(\bar{\mathbf{x}}_i),\mathbf{y}_i)$. As in the case of binary classification, for many loss functions this problem is computationally hard, and we therefore turn to describe convex surrogate loss functions. We describe the surrogates for the Kendall tau loss and for the NDCG loss.

A Hinge Loss for the Kendall Tau Loss Function:
We can think of the Kendall tau loss as an average of 0–1 losses for each pair. In particular, for every (i,j) we can rewrite

$$\mathbb{1}_{[\mathrm{sign}(y'_i-y'_j)\neq\mathrm{sign}(y_i-y_j)]} = \mathbb{1}_{[\mathrm{sign}(y_i-y_j)(y'_i-y'_j)\leq 0]}.$$

In our case, $y'_i - y'_j = \langle \mathbf{w},\mathbf{x}_i - \mathbf{x}_j\rangle$. It follows that we can use the hinge loss upper bound as follows:

$$\mathbb{1}_{[\mathrm{sign}(y_i-y_j)(y'_i-y'_j)\leq 0]} \leq \max\left\{0, 1-\mathrm{sign}\left(y_i-y_j\right)\left\langle \mathbf{w},\mathbf{x}_i-\mathbf{x}_j\right\rangle\right\}.$$

Taking the average over the pairs we obtain the following surrogate convex loss for the Kendall tau loss function:

$$\Delta(h_{\mathbf{w}}(\bar{\mathbf{x}}),\mathbf{y}) \leq \frac{2}{r(r-1)}\sum_{i=1}^{r-1}\sum_{j=i+1}^{r} \max\left\{0, 1-\mathrm{sign}(y_i-y_j)\left\langle \mathbf{w},\mathbf{x}_i-\mathbf{x}_j\right\rangle\right\}.$$

The right-hand side is convex with respect to $\mathbf{w}$ and upper bounds the Kendall tau loss. It is also a ρ-Lipschitz function with parameter $\rho \leq \max_{i,j}\|\mathbf{x}_i - \mathbf{x}_j\|$.

A Hinge Loss for the NDCG Loss Function:
The NDCG loss function depends on the predicted ranking vector $\mathbf{y}' \in \mathbb{R}^r$ via the permutation it induces. To derive a surrogate loss function we first make the following observation. Let V be the set of all permutations of $[r]$ encoded as vectors; namely, each $\mathbf{v} \in V$ is a vector in $[r]^r$ such that for all $i \neq j$ we have $v_i \neq v_j$. Then (see Exercise 17.4),

$$\pi(\mathbf{y}') = \operatorname*{argmax}_{\mathbf{v}\in V} \sum_{i=1}^{r} v_i\, y'_i. \tag{17.7}$$

Let us denote $\Psi(\bar{\mathbf{x}}, \mathbf{v}) = \sum_{i=1}^{r} v_i \mathbf{x}_i$; it follows that

$$\begin{aligned}
\pi(h_{\mathbf{w}}(\bar{\mathbf{x}})) &= \operatorname*{argmax}_{\mathbf{v}\in V} \sum_{i=1}^{r} v_i \langle \mathbf{w}, \mathbf{x}_i \rangle \\
&= \operatorname*{argmax}_{\mathbf{v}\in V} \left\langle \mathbf{w}, \sum_{i=1}^{r} v_i \mathbf{x}_i \right\rangle \\
&= \operatorname*{argmax}_{\mathbf{v}\in V} \langle \mathbf{w}, \Psi(\bar{\mathbf{x}}, \mathbf{v}) \rangle.
\end{aligned}$$

On the basis of this observation, we can use the generalized hinge loss for cost-sensitive multiclass classification as a surrogate loss function for the NDCG loss as follows:

$$\begin{aligned}
\Delta(h_{\mathbf{w}}(\bar{\mathbf{x}}), \mathbf{y}) &\le \Delta(h_{\mathbf{w}}(\bar{\mathbf{x}}), \mathbf{y}) + \langle \mathbf{w}, \Psi(\bar{\mathbf{x}}, \pi(h_{\mathbf{w}}(\bar{\mathbf{x}}))) \rangle - \langle \mathbf{w}, \Psi(\bar{\mathbf{x}}, \pi(\mathbf{y})) \rangle \\
&\le \max_{\mathbf{v}\in V} \left[\Delta(\mathbf{v}, \mathbf{y}) + \langle \mathbf{w}, \Psi(\bar{\mathbf{x}}, \mathbf{v}) \rangle - \langle \mathbf{w}, \Psi(\bar{\mathbf{x}}, \pi(\mathbf{y})) \rangle \right] \\
&= \max_{\mathbf{v}\in V} \left[\Delta(\mathbf{v}, \mathbf{y}) + \sum_{i=1}^{r} (v_i - \pi(\mathbf{y})_i) \langle \mathbf{w}, \mathbf{x}_i \rangle \right]. \qquad (17.8)
\end{aligned}$$

The right-hand side is a convex function with respect to $\mathbf{w}$.

We can now solve the learning problem using SGD as described in Section 17.2.5. The main computational bottleneck is calculating a subgradient of the loss function, which is equivalent to finding $\mathbf{v}$ that achieves the maximum in Equation (17.8) (see Claim 14.6). Using the definition of the NDCG loss, this is equivalent to solving the problem

$$\operatorname*{argmin}_{\mathbf{v}\in V} \sum_{i=1}^{r} (\alpha_i v_i + \beta_i D(v_i)),$$

where $\alpha_i = -\langle \mathbf{w}, \mathbf{x}_i \rangle$ and $\beta_i = y_i / G(\mathbf{y}, \mathbf{y})$. We can think of this problem a little bit differently by defining a matrix $A \in \mathbb{R}^{r,r}$ where

$$A_{i,j} = j\alpha_i + D(j)\beta_i.$$

Now, let us think about each j as a "worker," each i as a "task," and $A_{i,j}$ as the cost of assigning task i to worker j. With this view, the problem of finding $\mathbf{v}$ becomes the problem of finding an assignment of the tasks to workers of minimal cost. This problem is called "the assignment problem" and can be solved efficiently. One particular algorithm is the "Hungarian method" (Kuhn 1955). Another way to solve the assignment problem is using linear programming. To do so, let us first write the assignment problem as

$$\begin{aligned}
&\underset{B \in \mathbb{R}_+^{r,r}}{\operatorname{argmin}} \sum_{i,j=1}^{r} A_{i,j} B_{i,j} \qquad (17.9)\\
&\text{s.t.} \quad \forall i \in [r], \quad \sum_{j=1}^{r} B_{i,j} = 1\\
&\qquad \forall j \in [r], \quad \sum_{i=1}^{r} B_{i,j} = 1\\
&\qquad \forall i,j, \quad B_{i,j} \in \{0,1\}
\end{aligned}$$

A matrix B that satisfies the constraints in the preceding optimization problem is called a permutation matrix. This is because the constraints guarantee that there is at most a single entry of each row that equals 1 and a single entry of each column that equals 1. Therefore, the matrix B corresponds to the permutation $\mathbf{v} \in V$ defined by $v_i = j$ for the single index j that satisfies $B_{i,j} = 1$.

The preceding optimization is still not a linear program because of the combinatorial constraint $B_{i,j} \in \{0,1\}$. However, as it turns out, this constraint is redundant – if we solve the optimization problem while simply omitting the combinatorial constraint, then we are still guaranteed that there is an optimal solution that will satisfy this constraint. This is formalized later.

Denote $\langle A, B\rangle = \sum_{i,j} A_{i,j} B_{i,j}$. Then, Equation (17.9) is the problem of minimizing $\langle A, B\rangle$ such that B is a permutation matrix.

A matrix $B \in \mathbb{R}^{r,r}$ is called *doubly stochastic* if all elements of B are nonnegative, the sum of each row of B is 1, and the sum of each column of B is 1. Therefore, solving Equation (17.9) without the constraints $B_{i,j} \in \{0,1\}$ is the problem

$$\underset{B \in \mathbb{R}^{r,r}}{\operatorname{argmin}} \langle A, B\rangle \quad \text{s.t.} \quad B \text{ is a doubly stochastic matrix.} \qquad (17.10)$$

The following claim states that every doubly stochastic matrix is a convex combination of permutation matrices.

Claim 17.3 (Birkhoff 1946, Von Neumann 1953). *The set of doubly stochastic matrices in $\mathbb{R}^{r,r}$ is the convex hull of the set of permutation matrices in $\mathbb{R}^{r,r}$.*

On the basis of the claim, we easily obtain the following:

Lemma 17.4. *There exists an optimal solution of Equation (17.10) that is also an optimal solution of Equation (17.9).*

Proof. Let B be a solution of Equation (17.10). Then, by Claim 17.3, we can write $B = \sum_i \gamma_i C_i$, where each C_i is a permutation matrix, each $\gamma_i > 0$, and $\sum_i \gamma_i = 1$. Since all the C_i are also doubly stochastic, we clearly have that $\langle A, B\rangle \le \langle A, C_i\rangle$ for every i. We claim that there is some i for which $\langle A, B\rangle = \langle A, C_i\rangle$. This must be true since otherwise, if for every i $\langle A, B\rangle < \langle A, C_i\rangle$, we would have that

$$\langle A, B\rangle = \left\langle A, \sum_i \gamma_i C_i \right\rangle = \sum_i \gamma_i \langle A, C_i\rangle > \sum_i \gamma_i \langle A, B\rangle = \langle A, B\rangle,$$

which cannot hold. We have thus shown that some permutation matrix, C_i, satisfies $\langle A,B\rangle = \langle A, C_i\rangle$. But, since for every other permutation matrix C we have $\langle A,B\rangle \leq \langle A,C\rangle$ we conclude that C_i is an optimal solution of both Equation (17.9) and Equation (17.10). □

17.5 BIPARTITE RANKING AND MULTIVARIATE PERFORMANCE MEASURES

In the previous section we described the problem of ranking. We used a vector $\mathbf{y} \in \mathbb{R}^r$ for representing an order over the elements $\mathbf{x}_1,\ldots,\mathbf{x}_r$. If all elements in $\mathbf{y}$ are different from each other, then $\mathbf{y}$ specifies a full order over $[r]$. However, if two elements of $\mathbf{y}$ attain the same value, $y_i = y_j$ for $i \neq j$, then $\mathbf{y}$ can only specify a partial order over $[r]$. In such a case, we say that $\mathbf{x}_i$ and $\mathbf{x}_j$ are of equal relevance according to $\mathbf{y}$. In the extreme case, $\mathbf{y} \in \{\pm 1\}^r$, which means that each $\mathbf{x}_i$ is either relevant or nonrelevant. This setting is often called "bipartite ranking." For example, in the fraud detection application mentioned in the previous section, each transaction is labeled as either fraudulent ($y_i = 1$) or benign ($y_i = -1$).

Seemingly, we can solve the bipartite ranking problem by learning a binary classifier, applying it on each instance, and putting the positive ones at the top of the ranked list. However, this may lead to poor results as the goal of a binary learner is usually to minimize the zero-one loss (or some surrogate of it), while the goal of a ranker might be significantly different. To illustrate this, consider again the problem of fraud detection. Usually, most of the transactions are benign (say 99.9%). Therefore, a binary classifier that predicts "benign" on all transactions will have a zero-one error of 0.1%. While this is a very small number, the resulting predictor is meaningless for the fraud detection application. The crux of the problem stems from the inadequacy of the zero-one loss for what we are really interested in. A more adequate performance measure should take into account the predictions over the entire set of instances. For example, in the previous section we have defined the NDCG loss, which emphasizes the correctness of the top-ranked items. In this section we describe additional loss functions that are specifically adequate for bipartite ranking problems.

As in the previous section, we are given a sequence of instances, $\bar{\mathbf{x}} = (\mathbf{x}_1,\ldots,\mathbf{x}_r)$, and we predict a ranking vector $\mathbf{y}' \in \mathbb{R}^r$. The feedback vector is $\mathbf{y} \in \{\pm 1\}^r$. We define a loss that depends on $\mathbf{y}'$ and $\mathbf{y}$ and depends on a threshold $\theta \in \mathbb{R}$. This threshold transforms the vector $\mathbf{y}' \in \mathbb{R}^r$ into the vector $(\mathrm{sign}(y'_i - \theta),\ldots,\mathrm{sign}(y'_r - \theta)) \in \{\pm 1\}^r$. Usually, the value of θ is set to be 0. However, as we will see, we sometimes set θ while taking into account additional constraints on the problem.

The loss functions we define in the following depend on the following 4 numbers:

$$\begin{aligned}
\text{True positives: } & a = |\{i : y_i = +1 \wedge \mathrm{sign}(y'_i - \theta) = +1\}| \\
\text{False positives: } & b = |\{i : y_i = -1 \wedge \mathrm{sign}(y'_i - \theta) = +1\}| \\
\text{False negatives: } & c = |\{i : y_i = +1 \wedge \mathrm{sign}(y'_i - \theta) = -1\}| \\
\text{True negatives: } & d = |\{i : y_i = -1 \wedge \mathrm{sign}(y'_i - \theta) = -1\}|
\end{aligned} \tag{17.11}$$

The **recall** (a.k.a. **sensitivity**) of a prediction vector is the fraction of true positives $\mathbf{y}'$ "catches," namely, $\frac{a}{a+c}$. The **precision** is the fraction of correct predictions among the positive labels we predict, namely, $\frac{a}{a+b}$. The **specificity** is the fraction of true negatives that our predictor "catches," namely, $\frac{d}{d+b}$.

Note that as we decrease θ the recall increases (attaining the value 1 when $\theta = -\infty$). On the other hand, the precision and the specificity usually decrease as we decrease θ. Therefore, there is a tradeoff between precision and recall, and we can control it by changing θ. The loss functions defined in the following use various techniques for combining both the precision and recall.

- **Averaging sensitivity and specificity**: This measure is the average of the sensitivity and specificity, namely, $\frac{1}{2}\left(\frac{a}{a+c} + \frac{d}{d+b}\right)$. This is also the accuracy on positive examples averaged with the accuracy on negative examples. Here, we set $\theta = 0$ and the corresponding loss function is $\Delta(\mathbf{y}', \mathbf{y}) = 1 - \frac{1}{2}\left(\frac{a}{a+c} + \frac{d}{d+b}\right)$.
- F_1**-score**: The F_1 score is the harmonic mean of the precision and recall: $\frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}}$. Its maximal value (of 1) is obtained when both precision and recall are 1, and its minimal value (of 0) is obtained whenever one of them is 0 (even if the other one is 1). The F_1 score can be written using the numbers a, b, c as follows; $F_1 = \frac{2a}{2a+b+c}$. Again, we set $\theta = 0$, and the loss function becomes $\Delta(\mathbf{y}', \mathbf{y}) = 1 - F_1$.
- F_β**-score**: It is like F_1 score, but we attach β^2 times more importance to recall than to precision, that is, $\frac{1+\beta^2}{\frac{1}{\text{Precision}} + \beta^2 \frac{1}{\text{Recall}}}$. It can also be written as $F_\beta = \frac{(1+\beta^2)a}{(1+\beta^2)a + b + \beta^2 c}$. Again, we set $\theta = 0$, and the loss function becomes $\Delta(\mathbf{y}', \mathbf{y}) = 1 - F_\beta$.
- **Recall at** k: We measure the recall while the prediction must contain at most k positive labels. That is, we should set θ so that $a + b \le k$. This is convenient, for example, in the application of a fraud detection system, where a bank employee can only handle a small number of suspicious transactions.
- **Precision at** k: We measure the precision while the prediction must contain at least k positive labels. That is, we should set θ so that $a + b \ge k$.

The measures defined previously are often referred to as *multivariate performance measures*. Note that these measures are highly different from the average zero-one loss, which in the preceding notation equals $\frac{b+d}{a+b+c+d}$. In the aforementioned example of fraud detection, when 99.9% of the examples are negatively labeled, the zero-one loss of predicting that all the examples are negatives is 0.1%. In contrast, the recall of such prediction is 0 and hence the F_1 score is also 0, which means that the corresponding loss will be 1.

17.5.1 Linear Predictors for Bipartite Ranking

We next describe how to train linear predictors for bipartite ranking. As in the previous section, a linear predictor for ranking is defined to be

$$h_{\mathbf{w}}(\bar{\mathbf{x}}) = (\langle \mathbf{w}, \mathbf{x}_1 \rangle, \ldots, \langle \mathbf{w}, \mathbf{x}_r \rangle).$$

The corresponding loss function is one of the multivariate performance measures described before. The loss function depends on $\mathbf{y}' = h_{\mathbf{w}}(\bar{\mathbf{x}})$ via the binary vector it induces, which we denote by

$$\mathbf{b}(\mathbf{y}') = (\mathrm{sign}(y'_1 - \theta), \ldots, \ \mathrm{sign}(y'_r - \theta)) \ \in \ \{\pm 1\}^r. \tag{17.12}$$

As in the previous section, to facilitate an efficient algorithm we derive a convex surrogate loss function on Δ. The derivation is similar to the derivation of the generalized hinge loss for the NDCG ranking loss, as described in the previous section.

Our first observation is that for all the values of θ defined before, there is some $V \subseteq \{\pm 1\}^r$ such that $\mathbf{b}(\mathbf{y}')$ can be rewritten as

$$\mathbf{b}(\mathbf{y}') = \underset{\mathbf{v}\in V}{\mathrm{argmax}} \sum_{i=1}^{r} v_i y'_i. \tag{17.13}$$

This is clearly true for the case $\theta = 0$ if we choose $V = \{\pm 1\}^r$. The two measures for which θ is not taken to be 0 are precision at k and recall at k. For precision at k we can take V to be the set $V_{\geq k}$, containing all vectors in $\{\pm 1\}^r$ whose number of ones is at least k. For recall at k, we can take V to be $V_{\leq k}$, which is defined analogously. See Exercise 17.5.

Once we have defined $\mathbf{b}$ as in Equation (17.13), we can easily derive a convex surrogate loss as follows. Assuming that $\mathbf{y} \in V$, we have that

$$\begin{aligned}
\Delta(h_{\mathbf{w}}(\bar{\mathbf{x}}), \mathbf{y}) &= \Delta(\mathbf{b}(h_{\mathbf{w}}(\bar{\mathbf{x}})), \mathbf{y}) \\
&\leq \Delta(\mathbf{b}(h_{\mathbf{w}}(\bar{\mathbf{x}})), \mathbf{y}) + \sum_{i=1}^{r} (b_i(h_{\mathbf{w}}(\bar{\mathbf{x}})) - y_i)\langle \mathbf{w}, \mathbf{x}_i \rangle \\
&\leq \max_{\mathbf{v}\in V} \left[\Delta(\mathbf{v}, \mathbf{y}) + \sum_{i=1}^{r} (v_i - y_i) \langle \mathbf{w}, \mathbf{x}_i \rangle \right].
\end{aligned} \tag{17.14}$$

The right-hand side is a convex function with respect to $\mathbf{w}$.

We can now solve the learning problem using SGD as described in Section 17.2.5. The main computational bottleneck is calculating a subgradient of the loss function, which is equivalent to finding $\mathbf{v}$ that achieves the maximum in Equation (17.14) (see Claim 14.6).

In the following we describe how to find this maximizer efficiently for any performance measure that can be written as a function of the numbers $a, \ b, \ c, \ d$ given in Equation (17.11), and for which the set V contains all elements in $\{\pm 1\}^r$ for which the values of a, b satisfy some constraints. For example, for "recall at k" the set V is all vectors for which $a + b \leq k$.

The idea is as follows. For any $a, b \in [r]$, let

$$\bar{\mathcal{Y}}_{a,b} = \{\mathbf{v} \ : \ |\{i : v_i = 1 \wedge y_i = 1\}| = a \ \wedge \ |\{i : v_i = 1 \wedge y_i = -1\}| = b\}.$$

Any vector $\mathbf{v} \in V$ falls into $\bar{\mathcal{Y}}_{a,b}$ for some $a, b \in [r]$. Furthermore, if $\bar{\mathcal{Y}}_{a,b} \cap V$ is not empty for some $a, b \in [r]$ then $\bar{\mathcal{Y}}_{a,b} \cap V = \bar{\mathcal{Y}}_{a,b}$. Therefore, we can search within each $\bar{\mathcal{Y}}_{a,b}$ that has a nonempty intersection with V separately, and then take the optimal value. The key observation is that once we are searching only within $\bar{\mathcal{Y}}_{a,b}$, the value of Δ is fixed so we only need to maximize the expression

$$\max_{\mathbf{v}\in\bar{\mathcal{Y}}_{a,b}} \sum_{i=1}^{r} v_i \langle \mathbf{w}, \mathbf{x}_i \rangle .$$

Suppose the examples are sorted so that $\langle \mathbf{w}, \mathbf{x}_1 \rangle \geq \cdots \geq \langle \mathbf{w}, \mathbf{x}_r \rangle$. Then, it is easy to verify that we would like to set v_i to be positive for the smallest indices i. Doing this, with the constraint on a, b, amounts to setting $v_i = 1$ for the a top ranked positive examples and for the b top-ranked negative examples. This yields the following procedure.

Solving Equation (17.14)

```
input:
  $(\mathbf{x}_1,\ldots,\mathbf{x}_r), (y_1,\ldots,y_r), \mathbf{w}, V, \Delta$
assumptions:
  $\Delta$ is a function of $a,\ b,\ c,\ d$
  $V$ contains all vectors for which $f(a,b) = 1$ for some function $f$
initialize:
  $P = |\{i : y_i = 1\}|, N = |\{i : y_i = -1\}|$
  $\boldsymbol{\mu} = (\langle \mathbf{w}, \mathbf{x}_1 \rangle, \ldots, \langle \mathbf{w}, \mathbf{x}_r \rangle), \alpha^\star = -\infty$
  sort examples so that $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_r$
  let $i_1, \ldots, i_P$ be the (sorted) indices of the positive examples
  let $j_1, \ldots, j_N$ be the (sorted) indices of the negative examples
for $a = 0, 1, \ldots, P$
  $c = P - a$
  for $b = 0, 1, \ldots, N$ such that $f(a,b) = 1$
    $d = N - b$
    calculate $\Delta$ using $a,\ b,\ c,\ d$
    set $v_1, \ldots, v_r$ s.t. $v_{i_1} = \cdots = v_{i_a} = v_{j_1} = \cdots = v_{j_b} = 1$
      and the rest of the elements of $\mathbf{v}$ equal $-1$
    set $\alpha = \Delta + \sum_{i=1}^{r} v_i \mu_i$
    if $\alpha \geq \alpha^\star$
      $\alpha^\star = \alpha, \mathbf{v}^\star = \mathbf{v}$
output $\mathbf{v}^\star$
```

17.6 SUMMARY

Many real world supervised learning problems can be cast as learning a multiclass predictor. We started the chapter by introducing reductions of multiclass learning to binary learning. We then described and analyzed the family of linear predictors for multiclass learning. We have shown how this family can be used even if the number of classes is extremely large, as long as we have an adequate structure on the problem. Finally, we have described ranking problems. In Chapter 29 we study the sample complexity of multiclass learning in more detail.

17.7 BIBLIOGRAPHIC REMARKS

The One-versus-All and All-Pairs approach reductions have been unified under the framework of Error Correction Output Codes (ECOC) (Dietterich & Bakiri 1995, Allwein, Schapire & Singer 2000). There are also other types of reductions such as tree-based classifiers (see, for example, Beygelzimer, Langford & Ravikumar (2007)). The limitations of reduction techniques have been studied in (Daniely et al. 2011, Daniely et al. 2012). See also Chapter 29, in which we analyze the sample complexity of multiclass learning.

Direct approaches to multiclass learning with linear predictors have been studied in (Vapnik 1998, Weston & Watkins 1999, Crammer & Singer 2001). In particular, the multivector construction is due to Crammer and Singer (2001).

Collins (2000) has shown how to apply the Perceptron algorithm for structured output problems. See also Collins (2002). A related approach is discriminative learning of conditional random fields; see Lafferty et al. (2001). Structured output SVM has been studied in (Weston et al. 2002, Collins 2002, Taskar et al. 2003, Tsochantaridis et al. 2004).

The dynamic procedure we have presented for calculating the prediction $h_{\mathbf{w}}(\mathbf{x})$ in the structured output section is similar to the forward-backward variables calculated by the Viterbi procedure in HMMs (see, for instance, (Rabiner & Juang 1986)). More generally, solving the maximization problem in structured output is closely related to the problem of inference in graphical models (see, for example, Koller & Friedman (2009*a*)).

Chapelle, Le, and Smola (2007) proposed to learn a ranking function with respect to the NDCG loss using ideas from structured output learning. They also observed that the maximization problem in the definition of the generalized hinge loss is equivalent to the assignment problem.

Agarwal and Roth (2005) analyzed the sample complexity of bipartite ranking. Joachims (2005) studied the applicability of structured output SVM to bipartite ranking with multivariate performance measures.

17.8 EXERCISES

17.1 Consider a set S of examples in $\mathbb{R}^n \times [k]$ for which there exist vectors $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ such that every example $(\mathbf{x}, y) \in S$ falls within a ball centered at $\boldsymbol{\mu}_y$ whose radius is $r \geq 1$. Assume also that for every $i \neq j$, $\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\| \geq 4r$. Consider concatenating each instance by the constant 1 and then applying the multivector construction, namely,

$$\Psi(\mathbf{x}, y) = [\ \underbrace{0, \ldots, 0}_{\in \mathbb{R}^{(y-1)(n+1)}}\ , \underbrace{x_1, \ldots, x_n, 1}_{\in \mathbb{R}^{n+1}},\ \underbrace{0, \ldots, 0}_{\in \mathbb{R}^{(k-y)(n+1)}}\ ].$$

Show that there exists a vector $\mathbf{w} \in \mathbb{R}^{k(n+1)}$ such that $\ell(\mathbf{w}, (\mathbf{x}, y)) = 0$ for every $(\mathbf{x}, y) \in S$.

Hint: Observe that for every example $(\mathbf{x}, y) \in S$ we can write $\mathbf{x} = \boldsymbol{\mu}_y + \mathbf{v}$ for some $\|\mathbf{v}\| \leq r$. Now, take $\mathbf{w} = [\mathbf{w}_1, \ldots, \mathbf{w}_k]$, where $\mathbf{w}_i = [\boldsymbol{\mu}_i, -\|\boldsymbol{\mu}_i\|^2/2]$.

17.2 **Multiclass Perceptron:** Consider the following algorithm:

```
Multiclass Batch Perceptron

Input:
  A training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
  A class-sensitive feature mapping $\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$
Initialize: $\mathbf{w}^{(1)} = (0, \ldots, 0) \in \mathbb{R}^d$
  For $t = 1, 2, \ldots$
    If $(\exists\, i$ and $y \neq y_i$ s.t. $\langle \mathbf{w}^{(t)}, \Psi(\mathbf{x}_i, y_i) \rangle \leq \langle \mathbf{w}^{(t)}, \Psi(\mathbf{x}_i, y) \rangle)$ then
      $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \Psi(\mathbf{x}_i, y_i) - \Psi(\mathbf{x}_i, y)$
    else
      output $\mathbf{w}^{(t)}$
```

Prove the following:

Theorem 17.5. *Assume that there exists* $\mathbf{w}^\star$ *such that for all* i *and for all* $y \neq y_i$ *it holds that* $\langle \mathbf{w}^\star, \Psi(\mathbf{x}_i, y_i) \rangle \geq \langle \mathbf{w}^\star, \Psi(\mathbf{x}_i, y) \rangle + 1$. *Let* $R = \max_{i,y} \|\Psi(\mathbf{x}_i, y_i) - \Psi(\mathbf{x}_i, y)\|$. *Then, the multiclass Perceptron algorithm stops after at most* $(R\|\mathbf{w}^\star\|)^2$ *iterations, and when it stops it holds that* $\forall i \in [m],\ \ y_i = \operatorname{argmax}_y \langle \mathbf{w}^{(t)}, \Psi(\mathbf{x}_i, y) \rangle$.

17.3 Generalize the dynamic programming procedure given in Section 17.3 for solving the maximization problem given in the definition of $\hat{h}$ in the SGD procedure for multiclass prediction. You can assume that $\Delta(\mathbf{y}', \mathbf{y}) = \sum_{t=1}^{r} \delta(y'_t, y_t)$ for some arbitrary function δ.

17.4 Prove that Equation (17.7) holds.

17.5 Show that the two definitions of π as defined in Equation (17.12) and Equation (17.13) are indeed equivalent for all the multivariate performance measures.