Statistical Modeling with Spline Functions
Methodology and Theory

Mark H. Hansen
University of California at Los Angeles

Jianhua Z. Huang
University of Pennsylvania

Charles Kooperberg
Fred Hutchinson Cancer Research Center

Charles J. Stone
University of California at Berkeley

Young K. Truong
University of North Carolina at Chapel Hill

January 5, 2006

5

Polychotomous Regression and Multiple Classification

The previous two chapters discussed polynomial spline methodologies for linear models and generalized linear models. In the next several chapters we discuss such methodologies for several other models. While Chapters 3 and 4 provided ample methodological details, they were not focused on one particular implementation. This is in part because on the one hand there is no single publically available implementation on which we have worked ourselves, and on the other hand it is fairly straightforward to design one's own routine in a statistical language such as R or S-Plus.

That situation is somewhat different for the problems discussed in Chapters 5–9: the problems in these chapters (multiple classification, density estimation, hazard regression, spectral density estimation, and bivariate function estimation) are more specialized. Over the years we have developed a variety of methodologies for these problems, and consequently our discussion focuses on our implementations. As a result, these chapters provide somewhat more implementation details and somewhat less general discussion. It is possible to read any of these chapters independently of the other chapters.

5.1 An example

5.1.1 The vowel data

One of the goals in speech recognition is accurately to identify the text that is spoken, based on a processed set of acoustic signals. Typically, an

vowel	word	vowel	word
i	heed	O	hod
I	hid	C:	hoard
E	head	U	hood
A	had	u:	who'd
a:	hard	3:	heard
Y	hud		

TABLE 5.1. Vowels occurring in the vowel recognition data set.

intermediate step in this recognition process is to predict which phoneme is spoken at which time. Phonemes are basic sounds that, informally, correspond to something like a letter. The English language has approximately fifty phonemes (the exact number depends on which phonetic alphabet is used). Predicted phonemes can be combined into words and phrases, often using methods like hidden Markov chains. Bourlard and Morgan (1994) describes a speech recognition system that, for each time instance, uses for each phoneme the probability of its being spoken. Kooperberg, Bose, and Stone (1997) and Kooperberg and Stone (1999) analyze a large database from the area of speech recognition using the Polyclass and PolyMARS methodologies to estimate these probabilities. Both methodologies are described in this chapter; Polyclass is a polychotomous (multiclass logistic) regression methodology, and PolyMARS is a multi-response implementation of the multivariate regression approach discussed in Section 3.4.

While actual speech recognition data sets are often too large to be useful as examples, smaller data sets, where the goal is to predict which sounds is spoken at a particular instance, are good practice problems for many classification algorithms. We now focus on one data set from the area of speech recognition that has often been used as a practice problem. The data are due to Robinson (1989). They have also been analyzed by Hastie, Tibshirani, and Buja (1994). Fifteen different speakers (eight males, seven females) each spoke 11 different vowel sounds 6 times, for a total of 990 records. The 11 different vowels are summarized in Table 5.1. The speech signals were low pass filtered at 4.7kHz and then digitized to 12 bits with a 10kHz sampling rate. Twelfth-order linear predictive analysis was carried out on six 512 sample Hamming windowed segments from the steady part of the vowel. The reflection coefficients were used to calculate 10 log-area parameters, giving a 10 dimensional input space.

In the analysis of the data that is presented later in this chapter, we will use the data for four of the male speakers and four of the female speakers as training data and the data for the remaining four male and three female speakers as test data. For the initial data analysis presented here we combined all data. In Figure 5.1 we show boxplots for four of the features for each of the eleven vowels. From this plot it is immediate that

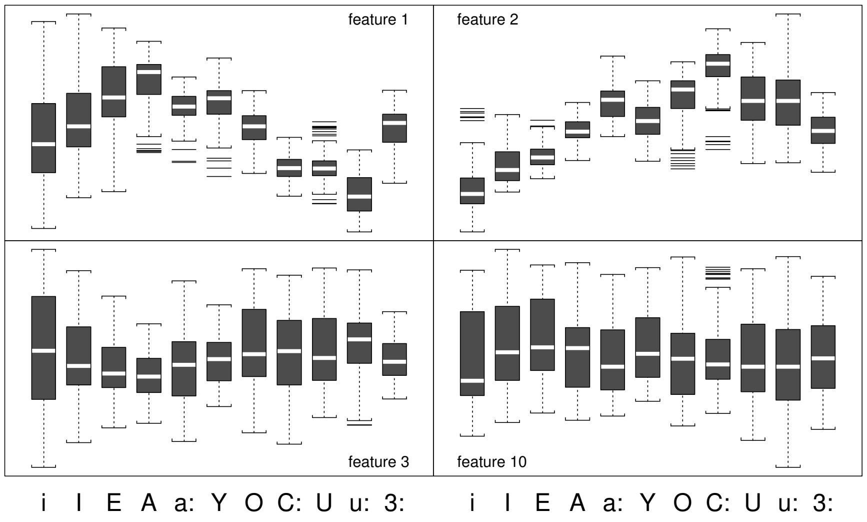

FIGURE 5.1. Boxplots for four of the features for the vowel data.

features 1 and 2 will probably be very useful in distinguishing the different phonemes as the values of these features differ considerably from vowel to vowel. The usefulness of features 3 and 10 is less clear. The boxplots for the remaining six features look much like those for features 3 and 10. For many of the boxplots there seems to be a group of “outliers”; see, for example, the feature 2 boxplots for “i”, “O” and “C:”. More often than not, such a group of outliers consists of the six repetitions of that vowel by one of the speakers.

Figure 5.2 shows the cases for five of the vowels as a function of features 1 and 2. From this figure it is already clear that a simple combination of these two features will likely not be sufficient for good discrimination, as “A” and “Y” seem very mixed up. It would appear that “i” and “u” are more easily distinguished from these other two vowels.

5.1.2 Background

The multiple classification problem is well studied in statistics. Typically, there is a qualitative random variable Y that takes on a finite number K of values, which we refer to as classes. We want to predict Y based on a random vector $\boldsymbol{X} \in \mathbb{R}^M$. Many methods have been proposed for this problem. See Mardia, Kent, and Bibby (1979) for a discussion of “classical” discriminant analysis methods. One of the popular modern multiple classification techniques is CART (Breiman, Friedman, Olshen, and Stone 1984), which approaches the multiple classification problem using recursive partitioning techniques that have strong links to nonparametric regression. Hastie, Tibshirani, and Buja (1994) introduce flexible discriminant analy-

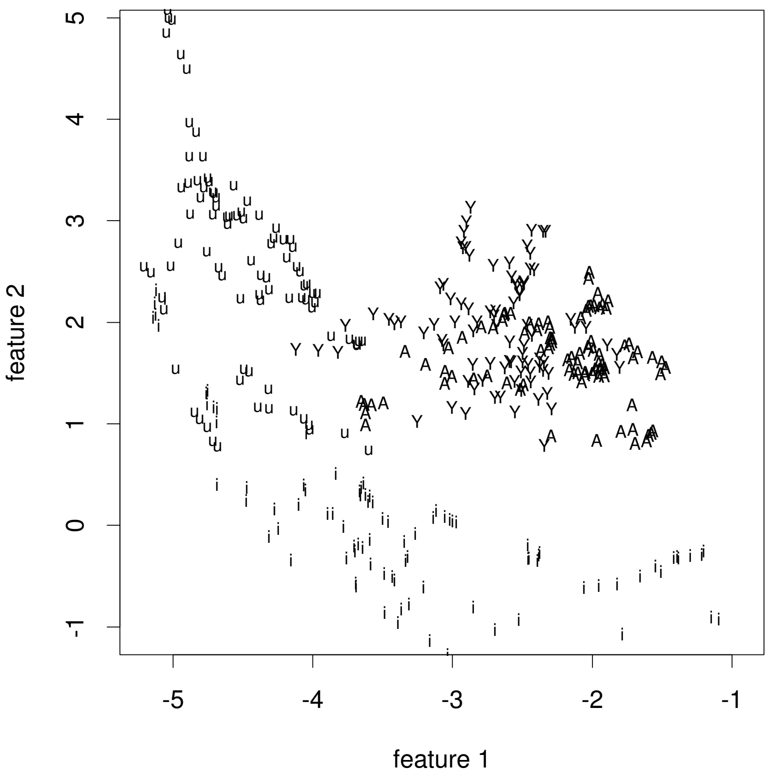

FIGURE 5.2. Scatterplot for five of the classes as a function of features 1 and 2.

sis, which combines nonparametric regression techniques with discriminant analysis. In computer science and engineering, neural networks (see Ripley 1996 and Cheng and Titterington 1994 for overviews). and Support Vector Machines (Vapnik 1995; Vapnik 1998) seem to be the methods of choice.

As is well known, the optimal classification rule predicts Y to be the class with the largest *posterior* probability: $\arg\max_k P(Y = k|\boldsymbol{X})$. Most of the popular classification methods try to find this class without precise estimation of the conditional class probabilities. However, there are many problems in which direct classification does not suffice. For example, Kooperberg, Bose, and Stone (1997) discuss the approach by Bourlard and Morgan (1994) to the phoneme recognition problem, which requires accurate estimation of the probability of a phoneme being in any particular class. Clearly, pure multiple classification methods are not useful in such applications.

On the other hand, multiple logistic regression (polychotomous regression) techniques have been used for a long time (see Hosmer and Lemeshow 1989). In a polychotomous regression model we do obtain an estimate of all the conditional class probabilities. In Chapter 4 we discussed the use of splines for logistic regression. In this chapter we will extend the logistic regression model to polychotomous regression problems with $K > 2$. In particular, the Polyclass model, which we discuss in this chapter, uses linear splines and their tensor products for fitting a regression model to data involving a polychotomous response variable. An advantage of this approach is that Polyclass yields not just a predicted class, but also estimates of the conditional class probabilities for each possible value of $\boldsymbol{X}$. Many of the issues that were relevant in the application of polynomial splines to generalized regression, as discussed in Chapter 4, are also relevant for Polyclass. In addition, we discuss a variety of issues that are unique to the multiple classification problem, and we also discuss computational issues involving polychotomous regression models with large data sets and very many parameters.

5.1.3 A Polyclass model for the vowel data

In this section we present a Polyclass model that was fit to the vowel data for eight of the speakers (training set). Validation of this model is obtained by evaluating the predictions of the Polyclass model on the data for the seven remaining speakers. In Figure 5.3 we show how test cases for which the values of features 3 through 10 are at the median of the value of these features for the training set would be classified. Figure 5.4 shows the estimated probabilities of selected classes for these values of the covariates.

In Table 5.2 we show how the cases in the test set are predicted. Overall, 222 out of 462 (48%) instances were predicted incorrectly, but the results differ dramatically over the classes: “i” and “O” are incorrectly classified 25% of the time, while “E”, “a:”, “Y”, and “3:” are misclassified 60% of the time. Interestingly, it appears to depend on the method of classification which vowels are hard and which are easy. We generated the same table as Table 5.2 for linear discriminant analysis (LDA). While this method had an overall misclassification error on the test set of 56% the second *worst* vowel for LDA was “O”, which was misclassified 74% of the time. Hastie, Tibshirani, and Buja (1994) lists test set error rates for a number of classification methods, the better of which have error rates of between 42% and 52%[1]

When a particular case was misclassified, often the class with the second highest estimated probability was the correct one. Table 5.3 shows that in 90% of the cases the correct class was among the three with the highest probability. From Table 5.4 it appears that there is a substantial difference in how Polyclass performs on the cases of particular individuals in the test set, suggesting that there may be a substantial speaker effect. In Section

[1]The appropriate comparison excludes the methods that were listed by Hastie, Tibshirani, and Buja (1994) as *Best reduced-dimension* We did not standardize features.

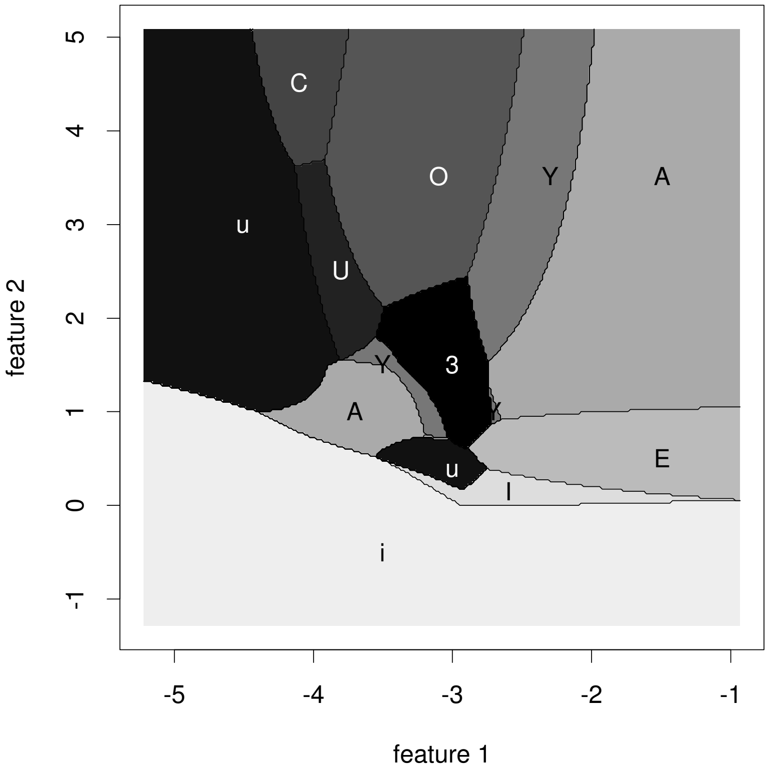

FIGURE 5.3. Classification map as a function of features 1 and 2 when all other features are equal to their median value.

5.3 we will see that this is a prime reason why the training set error rate (5%) is so much lower than the test set error rate.

5.2 The Polyclass methodology

5.2.1 The Polyclass model

Consider a qualitative random variable Y that takes on a finite number K of values. We may think of Y as ranging over $\mathcal{K} = \{1, \ldots, K\}$. Suppose the distribution of Y depends on features $x_1, \ldots, x_M$, where $\boldsymbol{x} = (x_1, \ldots, x_M)$ ranges over a subset $\mathcal{X}$ of $\mathbb{R}^M$. Let $\boldsymbol{x}$ now be distributed as a random vector; that is, consider the random pair $(\boldsymbol{X}, Y)$, where $\boldsymbol{X}$ is an $\mathcal{X}$-valued random vector and Y is a $\mathcal{K}$-valued random variable. Suppose that $P(Y = k|\boldsymbol{X} = \boldsymbol{x}) > 0$ for $\boldsymbol{x} \in \mathcal{X}$ and $k \in \mathcal{K}$. Let $\psi(\boldsymbol{x})$ be any function on $\mathcal{X}$ and set

$$\theta(k|\boldsymbol{x}) = \log P(Y = k|\boldsymbol{X} = \boldsymbol{x}) - \psi(x), \qquad \boldsymbol{x} \in \mathcal{X} \text{ and } k \in \mathcal{K}.$$

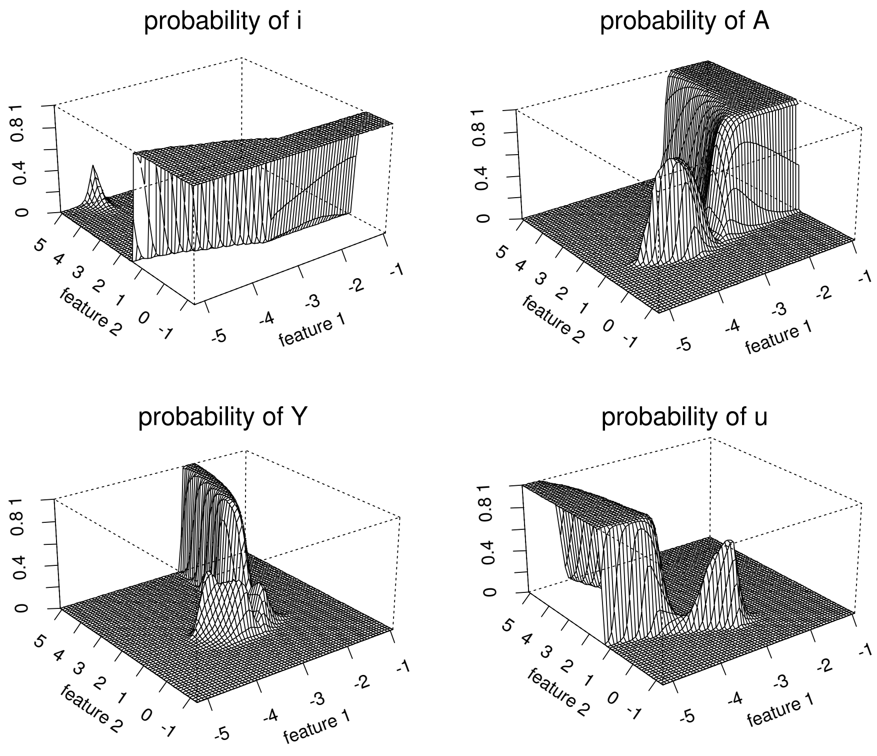

FIGURE 5.4. Probabilities as a function of features 1 and 2 when all other features are equal to their median value for four of the classes.

Then

$$P(Y = k|\boldsymbol{X} = \boldsymbol{x}) = \frac{\exp \theta(k|\boldsymbol{x})}{\sum_k \exp \theta(k|\boldsymbol{x})}, \qquad \boldsymbol{x} \in \mathcal{X} \text{ and } k \in \mathcal{K}. \tag{5.2.1}$$

Thinking of $\theta(1|\boldsymbol{x}), \dots, \theta(K|\boldsymbol{x})$ as unknown functions, we refer to (5.2.1) as the Polyclass model; when $K = 2$ it is referred to as the logistic regression model, which was discussed in Chapter 4. Observe that the model in (5.2.1) is nonidentifiable in that it does not involve the function ψ. In order to obtain an identifiable model, we add the restriction that

$$\theta(K|\boldsymbol{x}) = 0, \qquad \boldsymbol{x} \in \mathcal{X}. \tag{5.2.2}$$

With this restriction, (5.2.1) is now an identifiable model; indeed,

$$\theta(k|\boldsymbol{x}) = \log \frac{P(Y = k|\boldsymbol{X} = \boldsymbol{x})}{P(Y = K|\boldsymbol{X} = \boldsymbol{x})}, \qquad \boldsymbol{x} \in \mathcal{X} \text{ and } k \in \mathcal{K}.$$

Let p be a positive integer and let $\mathbb{G}$ be a p-dimensional linear space of functions on $\mathcal{X}$ with basis $B_1, \dots, B_p$.

true	predicted class										
class	i	I	E	A	a:	Y	O	C:	U	u:	3:
i	76	21	2	0	0	0	0	0	0	0	0
I	0	50	48	0	0	0	0	0	0	0	2
E	0	0	38	33	12	0	0	0	0	0	17
A	0	0	7	69	0	24	0	0	0	0	0
a:	0	0	0	0	38	36	21	0	0	0	5
Y	0	0	0	10	31	38	0	0	0	0	21
O	0	0	5	0	14	0	79	0	0	0	2
C:	0	0	0	0	0	0	48	52	0	0	0
U	2	0	2	2	0	0	10	14	50	12	7
u:	12	33	0	0	0	0	0	5	7	43	0
3:	0	0	5	5	5	24	5	0	17	2	38

TABLE 5.2. Classification results (in percents) for the initial Polyclass model on the test set of the vowel recognition data.

rank	1	2	3	4	5	6	7	8	9	10	11
percent	51.9	27.9	10.0	2.4	5.2	2.4	0.0	0.2	0.0	0.0	0.0

TABLE 5.3. Rank of the probability of the correct class for the initial Polyclass model on the test set of the vowel recognition data.

speaker	1	2	3	4	5	6	7
gender	male	male	male	male	female	female	female
percent wrong	39	39	33	58	53	61	53

TABLE 5.4. Percent of misclassified vowels per speaker in the test set.

Consider the model

$$\theta(k|\boldsymbol{x}) = \theta(k|\boldsymbol{x};\boldsymbol{\beta}) = \sum_{j=1}^{p} \beta_{jk} B_j(\boldsymbol{x}), \qquad \boldsymbol{x} \in \mathcal{X} \text{ and } k \in \mathcal{K}; \tag{5.2.3}$$

here $\boldsymbol{\beta}_k = (\beta_{k1}, \ldots, \beta_{kp})^{\mathrm{T}}$ for $1 \le k \le K-1$, $\boldsymbol{\beta}_K = 0$, and $\boldsymbol{\beta}$ is the $p(K-1)$-dimensional column vector consisting of the entries of $\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_{K-1}$, which ranges over $\mathcal{B} = \mathbb{R}^{p(K-1)}$. Setting $\boldsymbol{\beta}_K = 0$ enforces the identifiability restriction (5.2.2).[2] Correspondingly, we set

$$P(Y = k|\boldsymbol{X} = \boldsymbol{x};\boldsymbol{\beta}) = \frac{\exp\theta(k|\boldsymbol{x};\boldsymbol{\beta})}{\exp\theta(1|\boldsymbol{x};\boldsymbol{\beta}) + \cdots + \exp\theta(K|\boldsymbol{x};\boldsymbol{\beta})}$$
$$= \exp\big(\theta(k|\boldsymbol{x};\boldsymbol{\beta}) - c(\boldsymbol{x};\boldsymbol{\beta})\big) \tag{5.2.4}$$

[2]Except for floating point roundoff errors the Polyclass and PolyMARS methodologies discussed in this chapter are invariant under permutations of the values of Y.

for $\boldsymbol{\beta} \in \mathcal{B}$, $\boldsymbol{x} \in \mathcal{X}$, and $k \in \mathcal{K}$, where

$$c(\boldsymbol{x}; \boldsymbol{\beta}) = \log[\exp\theta(1|\boldsymbol{x}; \boldsymbol{\beta}) + \cdots + \exp\theta(K|\boldsymbol{x}; \boldsymbol{\beta})], \qquad \boldsymbol{\beta} \in \mathcal{B} \text{ and } \boldsymbol{x} \in \mathcal{X}.$$

Now

$$\log P(Y = k|\boldsymbol{X} = \boldsymbol{x}; \boldsymbol{\beta}) = \theta(k|\boldsymbol{x}; \boldsymbol{\beta}) - c(\boldsymbol{x}; \boldsymbol{\beta})$$

for $\boldsymbol{\beta} \in \mathcal{B}$, $\boldsymbol{x} \in \mathcal{X}$, and $k \in \mathcal{K}$. Thus

$$\frac{\partial}{\partial\beta_{k_1 j_1}} \log P(Y = k|\boldsymbol{X} = \boldsymbol{x}; \boldsymbol{\beta}) \\ = B_{j_1}(\boldsymbol{x})\big[\delta_{k_1 k} - \exp\big(\theta(k_1|\boldsymbol{x}; \boldsymbol{\beta}) - c(\boldsymbol{x}; \boldsymbol{\beta})\big)\big]$$

for $\boldsymbol{\beta} \in \mathcal{B}$, $\boldsymbol{x} \in \mathcal{X}$, $k \in \mathcal{K}$, $1 \le k_1 \le K - 1$, and $1 \le j_1 \le p$, where $\delta_{k_1 k} = 1$ for $k_1 = k$ and $\delta_{k_1 k} = 0$ for $k_1 \ne k$. Consequently

$$\begin{aligned} &\frac{\partial^2}{\partial\beta_{k_1 j_1}\partial\beta_{k_2 j_2}} \log P(Y = k|\boldsymbol{X} = \boldsymbol{x}; \boldsymbol{\beta}) \\ &\quad = B_{j_1}(\boldsymbol{x})B_{j_2}(\boldsymbol{x})\Big[- \delta_{k_1 k_2} \exp\big(\theta(k_1|\boldsymbol{x}; \boldsymbol{\beta}) - c(\boldsymbol{x}; \boldsymbol{\beta})\big) \\ &\qquad + \exp\big(\theta(k_1|\boldsymbol{x}; \boldsymbol{\beta}) - c(\boldsymbol{x}; \boldsymbol{\beta})\big) \exp\big(\theta(k_2|\boldsymbol{x}; \boldsymbol{\beta}) - c(\boldsymbol{x}; \boldsymbol{\beta})\big)\Big] \end{aligned} \tag{5.2.5}$$

for $\boldsymbol{\beta} \in \mathcal{B}$, $\boldsymbol{x} \in \mathcal{X}$, $1 \le k_1, k_2 \le K - 1$, and $1 \le j_1, j_2 \le p$. Since $\sum_k \exp\big(\theta(k|\boldsymbol{x}; \boldsymbol{\beta}) - c(\boldsymbol{x}; \boldsymbol{\beta})\big) = 1$, it follows easily from the Cauchy–Schwarz inequality that the Hessian matrix of $\log P(Y = k|\boldsymbol{X} = \boldsymbol{x})$ is negative semidefinite for $\boldsymbol{\beta} \in \mathcal{B}$, $\boldsymbol{x} \in \mathcal{X}$, and $k \in \mathcal{K}$.

5.2.2 Fitting Polyclass models

For fixed $\mathbb{G}$, $\boldsymbol{\beta}$ can be estimated using the method of maximum likelihood. In particular, let $(\boldsymbol{X}_1, Y_1), \ldots, (\boldsymbol{X}_n, Y_n)$ be independent random pairs, with each pair having the same joint distribution as $(\boldsymbol{X}, Y)$. The log-likelihood function corresponding to the Polyclass model (5.2.3) is given by

$$\ell(\boldsymbol{\beta}) = \sum_i [\theta(Y_i|\boldsymbol{X}_i; \boldsymbol{\beta}) - c(\boldsymbol{X}_i; \boldsymbol{\beta})], \qquad \boldsymbol{\beta} \in \mathcal{B}, \tag{5.2.6}$$

which is a concave function on $\mathcal{B}$.

For numerical reasons, we add a small penalty term to the log-likelihood function (5.2.6). Specifically, set

$$\ell_\epsilon(\boldsymbol{\beta}) = \ell(\boldsymbol{\beta}) - \epsilon \sum_i \sum_{k=1}^{K} u_{ik}^2, \tag{5.2.7}$$

where

$$u_{ik} = \theta(k|\boldsymbol{X}_i; \boldsymbol{\beta}) - \frac{1}{K} \sum_{k'=1}^{K} \theta(k'|\boldsymbol{X}_i; \boldsymbol{\beta}), \qquad k \in \mathcal{K}. \tag{5.2.8}$$

The penalized log-likelihood function, in which we have typically used $\epsilon = 10^{-6}$, is guaranteed to have a finite maximum. Without the penalty term, however, it is possible that, when the likelihood function is maximized, some $\hat{\beta}_{kj}$ equals $\pm\infty$. This can happen, for example, if $B_j(\boldsymbol{X}_i) = 0$ for all i such that $Y_i = k$.

The effect of this penalty term is usually negligible when $|\hat{\beta}_{kj}| < \infty$ for all j and k; that is, in our experience the estimates of the parameters with and without the penalty parameter are extremely close, while the estimates of the conditional class probabilities are indistinguishable.

The maximum (penalized) likelihood estimate $\widehat{\boldsymbol{\beta}}$ can be found using, for example, a Newton–Raphson algorithm, as described in Section 2.5.3.

5.2.3 Model selection

For Polyclass we use the same allowable spaces as in the Two-factor interaction models in Section 3.4; the basis functions are linear splines and depend on at most two variables. In particular the candidate basis functions are:

- x_k, $k = 1, \ldots, d$;
- $(x_k - t_{k,m})_+$ if x_k is already a basis function in the model;
- $x_{k_1}x_{k_2}$ if x_{k_1} and x_{k_2} are already basis functions in the model;
- $x_{k_1}(x_{k_2} - t_{k_2,m})_+$, if $x_{k_1}x_{k_2}$ and $(x_{k_2} - t_{k_2,m})_+$ are in the model;
- $(x_{k_1} - t_{k_1,m_1})_+(x_{k_2} - t_{k_2,m_2})_+$ if $x_{k_1}(x_{k_1} - t_{k_1,m_1})$ and $(x_{k_1} - t_{k_1,m_1})_+x_{k_2}$ are in model.

As in the multiple regression approach described in Section 3.4, we get a sequence of models using stepwise addition followed by stepwise deletion of basis functions. During the stepwise addition stage, for Polyclass, it is infeasible to compute Rao statistics for each possible location of a new knot, so we use a heuristic algorithm to find a good location for a new knot (see Section 5.5.2). In Section 5.5.1 we discuss a default rule for the maximum number $p_{\max}$ of basis functions in a model.

During the combination of stepwise addition and stepwise deletion, we get a sequence of models indexed by ν, with the νth model having $p_\nu(K-1)$ parameters. For Polyclass the methods of selecting one model from this sequence that we consider are the (generalized) Akaike information criterion (AIC), an independent test set, and cross-validation.

AIC

Let $\hat{l}_\nu$ denote the fitted log-likelihood for the νth model, and let $\text{AIC}_{\alpha,\nu} = -2\hat{l}_\nu + \alpha p_\nu(K-1)$ be the Akaike information criterion with penalty parameter α for this model. We select the model corresponding to the value

$\hat{\nu}$ of ν that minimizes $\mathrm{AIC}_{\alpha,\nu}$ (see Chapter 4). As before, we recommend choosing $\alpha = \log n$ as in the Bayesian information criterion (BIC) due to Schwarz (1978).

Test set

Consider an independent test set $(\boldsymbol{X}_i^{\mathrm{TS}}, Y_i^{\mathrm{TS}})$, $1 \leq i \leq n^{\mathrm{TS}}$. For the νth model let $\widehat{\boldsymbol{\beta}}_\nu$ be the maximum likelihood estimate of $\boldsymbol{\beta}$ and set $\widehat{Y}_{i,\nu}^{\mathrm{TS}} = \arg\max_k P(Y = k|\boldsymbol{X} = \boldsymbol{X}_i^{\mathrm{TS}}; \widehat{\boldsymbol{\beta}}_\nu)$ (see 5.2.4) as the most likely class for case i out of the test set. We can now estimate the risk (probability of misclassification) by $\widehat{R}_\nu^{\mathrm{TS}} = \sum_i \mathrm{ind}(\widehat{Y}_i^{\mathrm{TS}} \neq Y_i^{\mathrm{TS}})/n^{\mathrm{TS}}$.

Given a finite number of estimates of the optimal classifier, we choose the model having the smallest estimated risk. The minimum value of $\widehat{R}_\nu^{\mathrm{TS}}$ is an estimate of the risk for classifying a new object using the final Polyclass model. This estimate is slightly biased downwards, since the test set is used to minimize the risk.

Cross-validation

Alternatively, cross-validation can be used to estimate the risk. Here we first randomly divide the cases into $c \geq 2$ approximately equally-sized subsets. Then the following procedure is carried out for $j = 1, \ldots, c$ (see Breiman, Friedman, Olshen, and Stone 1984):

- Fit a sequence of Polyclass models, as described in Section 5.2.3, to all cases not in the jth subset.
- For each $\alpha > 0$ select the model $\hat{\nu}_{j\alpha}$ that minimizes $\mathrm{AIC}_{\alpha,\nu}$.
- For each α compute the loss $r_j(\alpha) = \sum \mathrm{ind}(\widehat{Y}_i \neq Y_i)$, where the sum is over the cases in the jth subset (which were not used to fit these models).

For every α we compute the cross-validated loss $R(\alpha) = n^{-1} \sum_{j=1}^{c} r_j(\alpha)$. Let $\tilde{\alpha}$ be the geometric mean of the endpoints of the interval of values of α that minimizes $R(\alpha)$. We proceed by fitting a sequence of Polyclass models to all data, using AIC with penalty parameter $\tilde{\alpha}$ to select the model.

Note that $\min R(\alpha)$ is a slightly optimistic (downward biased) estimate of the risk for classifying a new object using the final Polyclass model.

5.3 Further analysis of the vowel data

The Polyclass model for the vowel data that was discussed in Section 5.1.3 was selected using AIC based on the data for the eight training set speakers. The model involved 12 basis functions, which are summarized in Table 5.5.

$B_1(\boldsymbol{x}) = 1$	$B_7(\boldsymbol{x}) = (x_1 + 2.930)_+$
$B_2(\boldsymbol{x}) = x_1$	$B_8(\boldsymbol{x}) = (x_2 - 1.492)_+$
$B_3(\boldsymbol{x}) = x_2$	$B_9(\boldsymbol{x}) = (x_4 - 0.574)_+$
$B_4(\boldsymbol{x}) = x_4$	$B_{10}(\boldsymbol{x}) = (x_8 - 0.676)_+$
$B_5(\boldsymbol{x}) = x_5$	$B_{11}(\boldsymbol{x}) = x_1 x_2$
$B_6(\boldsymbol{x}) = x_8$	$B_{12}(\boldsymbol{x}) = x_5 x_8$

TABLE 5.5. Basis functions for the Polyclass model for the vowel data that was selected using AIC.

This model was optimal for α between 4.98 and 6.45; since $n = 528$, the default value of α was $\log 528 = 6.27$.

When we use ten-fold cross validation, $R(\alpha)$ is minimized for α between 0.25 and 0.29, for which $R(\alpha) = 42$, a cross-validation loss estimate of about 8%. With α this small Polyclass selected the largest model that was fit. This model had 20 basis functions and a test set loss of 221 or 48%. The fitted average test set log-likelihood for the cross validated model is -4.49, which is much worse than the fitted average test set log-likelihood for the AIC model of -2.88.

Given the data structure, a more sensible way to do cross-validation for this example would be each time to leave out one of the eight speakers in the training set and fit a Polyclass model to the remaining seven speakers, otherwise proceeding as if this is regular eight-vold cross validation. Now $R(\alpha)$ is minimized for α between 2.82 and 2.87, for which $R(\alpha) = 246$, a cross-validation loss estimate of about 47%. With this value of α, Polyclass selected a model with 17 basis functions. This model performs marginally better than the other two in test set misclassification (46%), and it is halfway between the other two in average test set log-likelihood (-3.87). It is more remarkable that for this model the cross-validated estimate of the loss is very close to the actual test set loss.

Finally, we can "cheat" a little and select the model that has the smallest test set loss. This turns out to be the model with 18 basis functions. However, the average test set log-likelihood is not very good (-4.34). Actually, if we used the test set to select the model with the best test set log-likelihood we would pick a model with only four basis functions, a misclassification rate of 52% and an average test set log-likelihood of -1.39. All models are summarized in Table 5.6.

As we mentioned several times, when analyzing the vowel data we are left with the impression that the enormous discrepancy between the training set and the test set results, as well as the very low average test set log-likelihood, is caused by the large speaker to speaker variation. To verify this, we randomly redivided the 990 cases into a training set of 528 cases and a test set of 462 cases, allowing some records from a speaker to end up in the training set and the others to end up in the test set. We then refit the Polyclass models using the four different selection mechanism used

selection mechanism	number of basis functions	misclassification error training	misclassification error CV	misclassification error test	average log-likelihood training	average log-likelihood test
AIC	12	4.9%	–	48.0%	−0.17	−2.88
10-fold CV	20	0.0%	8.0%	47.8%	−0.02	−4.49
8-fold speaker CV	17	0.2%	46.8%	46.1%	−0.04	−3.87
test set misclass.	18	0.0%	–	44.8%	−0.03	−4.34
test set log-lik.	4	46.8%	–	51.8%	−1.34	−1.39

TABLE 5.6. Summary statistics for Polyclass models that were selected for the vowel data.

selection mechanism	number of basis functions	misclassification error training	misclassification error CV	misclassification error test	average log-likelihood training	average log-likelihood test
AIC	10	16.7%	–	21.2%	−0.38	−0.60
10-fold CV	16	3.2%	15.9%	17.7%	−0.13	−0.55
test set misclass.	16	4.7%	–	11.9%	−0.13	−0.38
test set log-lik.	16	4.7%	–	11.9%	−0.13	−0.38

TABLE 5.7. Summary statistics for Polyclass models that were selected for the rerandomized vowel data. Note that the 16 basis function model for the 10-fold CV selection was fit during the stepwise deletion, while those for the test set selection were fit during the stepwise addition.

before: AIC, 10-fold CV, using the test set to minimize misclassification rate, and using the test set to maximize log-likelihood (the 8-fold speaker based cross validation no longer makes sense). The results are summarized in Table 5.7. As can be seen when comparing this table with Table 5.6, the speaker to speaker variation indeed influences the results considerably. The differences between training and test set are now much more reasonable, as is the average test set log-likelihood.

A feature that sets Polyclass apart from many other classification methods is that it gives not only an estimate of the class, but also estimates of the conditional class probabilities that are positive and add up to 1. Figure 5.5 plots the estimated probability that a case is a particular vowel grouped in bins of size 0.1 on the horizontal axis and the fraction of cases with that probability that correspond to the correct vowel on the vertical axis. Note that every case contributes 11 observations to this graph: one per candidate vowel. We note that the graph for the rerandomized data on the right-hand side is close to the ideal straight line (fraction true class) = (estimated probability) for both the test and the training set. However, for the original data the estimates of probabilities that are larger than, say, 0.8, seem to be upwards biased, as the fraction in the true class is considerably lower than the estimated probability.

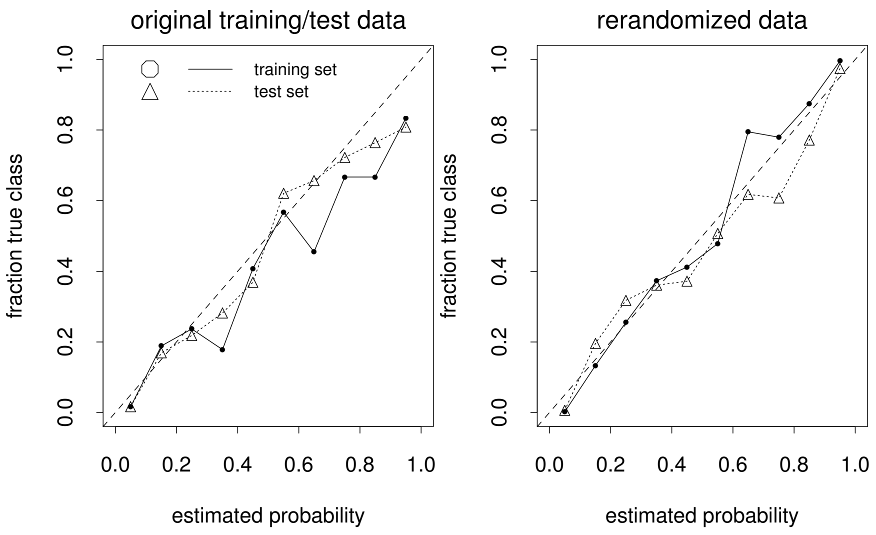

FIGURE 5.5. Fraction of vowels that correspond to the true class versus the estimated probability. The data have been binned in bins of size 0.1

5.4 Applying Polyclass to large data sets

5.4.1 The fruit data

At the check-out counters of grocery stores, clerks typically have to type in a code to indicate which types of fruit or vegetables are being bought. NCR, a leading manufacturer of check-out machines, is developing tools to assist clerks by suggesting which type of produce is lying on the scale. A technique they explored makes use of the color reflection of the produce. In brief, a light source is installed under the scale in the check-out stand, and the power-spectrum of the reflected light is captured. Based on this spectrum (as well as possible additional information regarding the shape of the produce), the goal is to indicate which produce is likely to be currently laying on the scale.

We received power spectra from NCR on approximately 100 different types of produce. Spectra were computed on produce that were bought during three different shopping trips at grocery stores in Atlanta, GA. The spectra were computed using two different devices. For our analysis we use only the data that was collected on one of the two devices, as the two devices differed considerably in their characteristics. Spectra were computed at 51 different wavelengths from 450 to 700nm, wavelengths that are all in the visible range. Each power spectrum was normalized, as the overall level of the spectrum primarily reflects the amount of light present.

The number of spectra that we have for each type of produce ranges from 16 to 144, with about two thirds of the classes having over 100 spectra. We

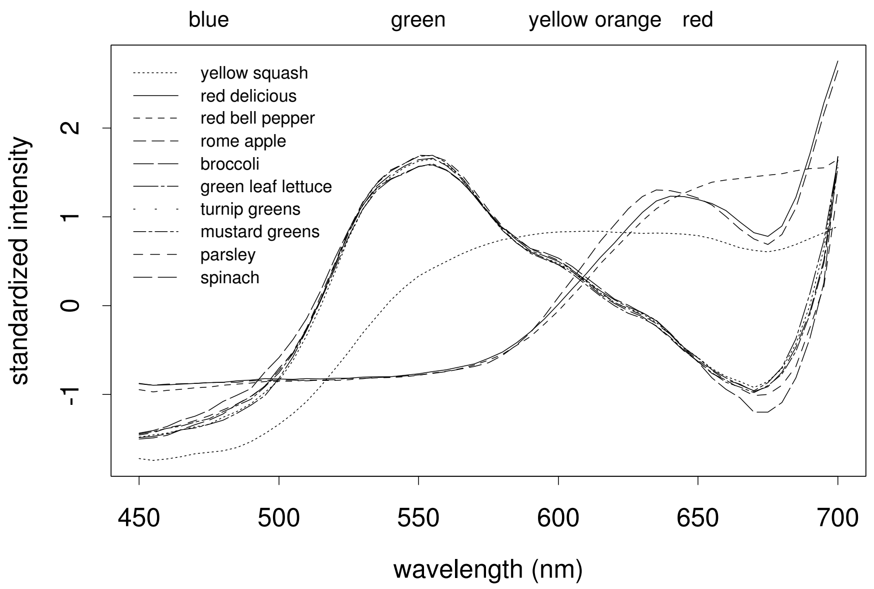

FIGURE 5.6. Median power spectra for ten different types of produce.

randomly divided the data into a test set containing one third of the data for each shopping trip and each produce type and a training set containing the remainder. In Figure 5.6 we show the pointwise medians over the training set of the spectrum for ten different types of produce, six of which are primarily green, three of which are primarily red, and one which is primarily yellow. In Figure 5.7 we show the individual spectra for the 96 samples of red delicious and 96 samples of mustard greens in the training set. Based on these graphs it seems clear that it will not be hard to distinguish between primarily red, primarily yellow, and primarily green types of produces, but that it may be much harder to distinguish among the types of the same color. We also notice that at least one power spectrum of a red delicious went astray. Such cases have been removed from the data set.

Analysis of the data on ten types of produce

The subset of ten types of produce summarized in Figures 5.6 and 5.7 is a particularly hard subset of types of produce; distinguishing between the six green types of produce is almost impossible, while many of the types of produce that are not among the ten selected are fairly easily distinguished from each other.

Before applying Polyclass to a larger part of the fruit data, we studied its behavior on the ten types of produce for which the median curves were shown in Figure 5.6. As the 51 predictors for this data are extremely highly correlated and there is no particular interest in the predictors themselves, it makes sense to consider transformations of the predictor space

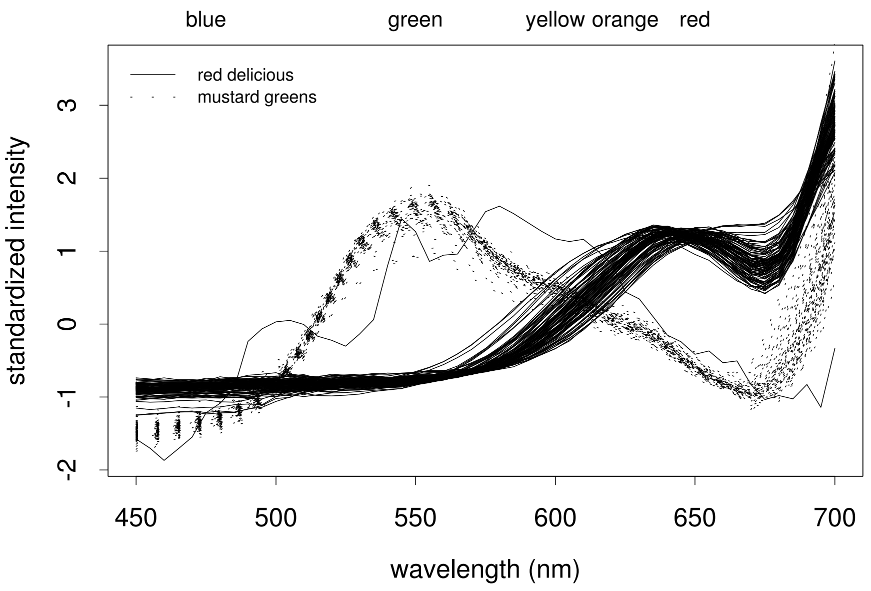

FIGURE 5.7. Power spectra for the red delicious and the mustard greens in the training set.

before applying Polyclass (or, for that matter, any classification algorithm). Transformations that we considered were:

- using as predictors a smaller number of averages of the spectrum over consecutive wavelengths, and estimates of the derivative of the power spectrum based on differences of untransformed predictors;
- projecting the 51 predictors on a smaller number of B-spline functions in wavelength;
- projecting the predictors on the principal components of the predictors of the training set.

We found that the first transformation improved the Polyclass results slightly over using the untransformed predictors while the second possibility did not improve results at all. The largest improvement over using the raw predictors was achieved when the predictors were projected on the principal components. Fig 5.8 shows the first four principal components; clearly these are related to different primary colors.

We applied Polyclass with 10-vold cross-validation to the training data. The selected model had ten basis functions. Only the first six principal components were involved in the Polyclass model. Overall 360 out of 478 (75.3%) pieces of produce in the test set were correctly classified. Table 5.8 shows the classification results on the test set. It is obvious from this data that distinguishing between the various green vegetables is indeed the

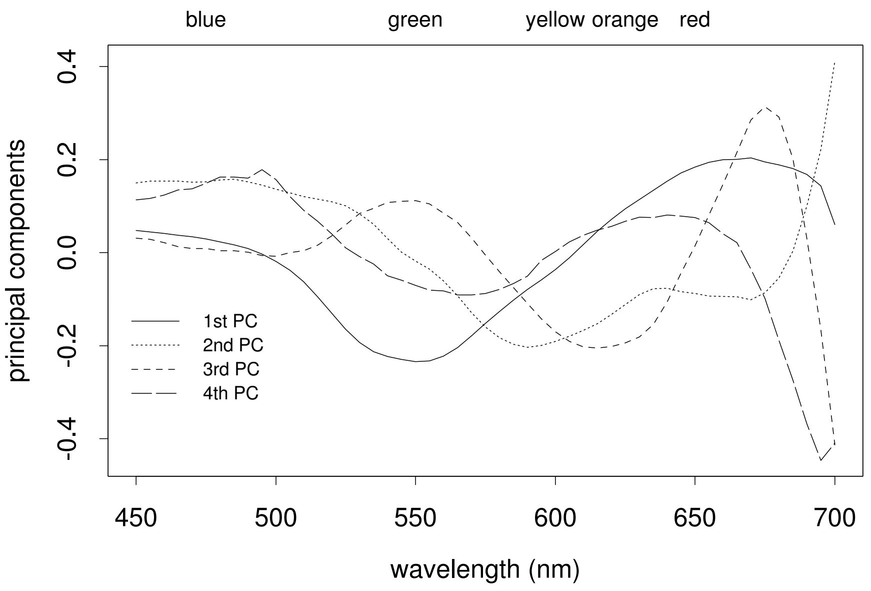

FIGURE 5.8. The first four principal components for the training set on ten types of produce.

challenge in this data set. As the intended application of this technology is to alert clerks when they mistyped a produce code, it may be sufficient if the correct class is not the most likely posterior class, but if it is one of the most likely classes, For our model, the correct class is in 92.5% of the cases among the three classes with the highest fitted probabilities, and in 93.1% of the cases, the fitted probability of the correct class is above 0.1.

A larger subset of the fruit data

In the remainder of this section, we will focus on the subset of the 66 types of produce that have 125 samples or more. The resulting training set has $n_{\text{tr}} = 6188$ cases, the resulting test set has $n_{\text{ts}} = 3092$ cases. This is *not* a particularly large data set, but still it poses considerable computational difficulties as there are $K = 66$ classes, so that for every basis function we need to estimate 65 parameters. For a model with, say, $p = 40$ basis functions this yields a coefficient vector $\boldsymbol{\beta}$ of length 2600, and a Hessian with 3.4 million unique elements. To evaluate the Hessian once, we need $O(p^2K^2n_{\text{tr}})$ operations, which is approximately 10^{10} for the problem size mentioned.

As mentioned before, this is *not* a particularly large data set. In Kooperberg and Stone (1999) we discuss a data set from the area of speech recognition with 153,426 cases, 81 predictors and 45 classes (further referred to as the "phoneme data"). For this data set models with as many as 1000 basis functions were considered. In this situation, the Hessian would have

	true class									
predicted class	yellow squash	red delicious	rEd bell pepper	rome apple	broccoli	green leaf lettuce	turnip greens	mustard greens	parsley	spinach
yellow squash	48	0	0	0	0	0	0	0	0	0
red delicious	0	45	0	1	0	0	0	0	0	0
red bell pepper	0	0	47	0	0	0	0	0	1	0
rome apple	0	3	0	47	0	0	0	0	0	0
broccoli	0	0	0	0	46	0	1	3	0	0
green leaf lettuce	0	0	0	0	0	28	5	2	2	6
turnip greens	0	0	0	0	0	6	17	9	8	2
mustard greens	0	0	0	0	1	5	8	24	5	3
parsley	0	0	0	0	1	5	9	8	27	6
spinach	0	0	0	0	0	4	8	1	5	31

TABLE 5.8. Test set classification results of applying Polyclass to a subset of ten types of produce for the fruit data.

approximately 1 billion unique elements, and computing the Hessian would require 10^{14} operations. The fifth international conference on data mining and knowledge discovery in 1999 (KDD99), had a data mining competition for which the data set (related to computer security) had approximately 5,000,000 training records and 3,000,000 test records, 40 predictors and five classes. In general, modern data bases make it possible to generate huge data sets. In this section we will focus on the fruit data, as the relatively modest size of that data set allows us to compare various approaches to dealing with larger data sets.

The larger fruit data set is still sufficiently small that we can apply Polyclass to it. The resulting model, selected using AIC, contained 18 basis functions, while the largest model that was fit had 20 basis functions. This required 30 hours of CPU time on the computer on which we carried out the computations[3] Selection of the model using cross-validation is clearly out of the question since this would require us to run the basic Polyclass algorithm 11 times (for ten cross-validation runs and a final run). The fact that the largest model that was considered contained only two more basis functions than the selected model clearly suggests that we should have considered models with more basis functions. Clearly, this was not really feasible.

[3]A Sun ULTRA-II with a 250 mhz processor and 1.8 gigabyte RAM.

We postpone discussing the results on the fruit data until Section 5.4.5, as we will be able to compare different approaches to dealing with this data set at that time.

5.4.2 Analysis of cpu-time required for large data sets

The two activities in the "standard" Polyclass algorithm that require the most cpu time are parameter estimation using maximum likelihood, implemented via a Newton–Raphson algorithm, and computation of the Rao statistics during the stepwise addition stage of the algorithm. There are several factors that influence the number of floating point operations, and thus the cpu time, that are required to apply the Polyclass algorithm to large data sets:

n_{tr}	the number of cases in the training set;
K	the number of classes;
M	the number of predictors;
$p_{\max}$	the maximum number of basis functions fit.

The computations that, for large problems, require the largest number of operations are the computation of the Hessian during the Newton–Raphson iterations and the computation of the K extra columns of the Hessian for the Rao statistics. In particular, the number of operations that are required per element of a Hessian is approximately $2n_{\mathrm{tr}}$: one multiplication to compute the contribution of one case (5.2.5) based on earlier computed terms, and one addition over all cases in the training data set.

A full Hessian during the Newton–Raphson iterations has approximately $p^2K^2/2$ unique elements, and typically the number of Newton–Raphson iterations that is required to fit a model is approximately 4[4]. Thus, we require about

$$4 \times \frac{p^2K^2}{2} \times 2n_{\mathrm{tr}} = 4p^2K^2n_{\mathrm{tr}} \tag{5.4.1}$$

operations to fit a model with p basis functions. During the process of stepwise addition and deletion we need to fit each model of size $p = 1, \ldots, p_{\max} - 1$ twice, and we need to fit the model of size $p_{\max}$ once. Thus, in total we require approximately

$$\sum_{p=1}^{p_{\max}-1} 8p^2K^2n_{\mathrm{tr}} + 4p_{\max}^2K^2n_{\mathrm{tr}} \approx \frac{8}{3}p_{\max}^3K^2n_{\mathrm{tr}}$$

floating point operations to fit all models.

The additional columns of a Hessian for Rao statistics have approximately pK^2 unique elements. In our experience, the number of Rao statistics that is computed when the current model has p basis functions is $O(p)$.

[4]This is the number of iterations required to get fairly close to the MLE. To get really close quite a few more iterations would be needed.

In particular, over a range of problems $M + 4p$ appears to be a reasonable approximation to the number of Rao statistics that is being computed. Thus, for computing all Rao statistics for the addition of a basis function to a model with p basis functions, we require approximately

$$(M + 4p) \times pK^2 \times 2n_{\mathrm{tr}}$$

operations. As we need to compute Rao statistics for the addition of a basis function to models with $p = 1, \ldots, p_{\max} - 1$ basis functions, the computation of all Rao statistics requires approximately

$$\sum_{p=1}^{p_{\max}-1} (Mp + 4p^2)K^2 n_{\mathrm{tr}} \approx \left(\frac{M}{2} + \frac{4p_{\max}}{3}\right) p_{\max}^2 K^2 n_{\mathrm{tr}}$$

floating point operations.

In Table 5.9 we summarize how long this would take on the same single processor computer on which we ran the large fruit data. As can be seen from this table, for problems like the phoneme data or the KDD99 data using Polyclass directly is infeasible.

Quasi-Newton optimization

Clearly, one possibility for reducing the amount of cpu time required is to use another optimization algorithm. The two other algorithms that immediately come to mind are quasi-Newton methods and conjugate gradient methods. Conjugate gradient methods, discussed further in Section 5.4.4, are *not* appropriate when Rao statistics need to be computed, since the conjugate gradient method does not yield a Hessian matrix, which is needed for the computation of Rao statistics.

Quasi-Newton methods do yield an approximate Hessian matrix. For the quasi-Newton method second derivatives are not computed, but are approximated by differences between the first derivatives at consecutive iterations of the algorithm. Essentially, during the iterations a working Hessian matrix is maintained, which at each iteration is altered by adding a rank two matrix to the inverse of the working Hessian. The two most commonly used algorithms are the Davidon-Fletcher-Powell (DFP) and the Boyden-Fletcher-Goldfarb-Shanno (BFGS) updating formulas (Kennedy and Gentle 1980). A disadvantage of quasi-Newton algorithms is that they still require storage of the $O(p^2K^2)$ working Hessian matrix, which can become substantial, especially when K is large, as is the case for the fruit data and the larger phoneme data.

Using a quasi-Newton method would be attractive if we could also use a simple updating formula to compute the extra columns of the Hessian needed for the Rao statistics. Unfortunately, in our experience, this does not work, probably because the extra columns can be updated only once, which is not enough to attain a high enough precision. An alternative approach

					number of operations for			approximate
data set	n_{tr}	M	K	$p_{\max}$	Rao statistics	NR iterations	total	cpu time
fruit data	6188	51	66	20	6×10^{11}	6×10^{11}	1×10^{12}	8 hours
				40	5×10^{12}	3×10^{12}	8×10^{12}	3 days
phoneme data	153426	81	45	350	4×10^{16}	2×10^{16}	5×10^{16}	50 years
				1000	8×10^{17}	4×10^{17}	1×10^{18}	1000 years
KDD99 data	5000000	40	5	100	3×10^{14}	2×10^{14}	5×10^{14}	6 months
				500	4×10^{16}	2×10^{16}	6×10^{16}	60 years

TABLE 5.9. Required cpu time for applying Polyclass to some large problems (cpu time reflects approximately a Linux machine with a 2.4gHz processor).

is to use the working Hessian part of the larger Hessian matrix for the computation of Rao statistics, but to compute the additional columns of the Hessian using exact second derivatives. The disadvantage of this approach is that there is no reduction in the amount of cpu time required for the Rao statistics. This means that while a reduction of the cpu time is possible, this reduction will not be an order of magnitude, since about 50% of the cpu time is spent on computing the Rao statistics. In addition, the number of iterations required for a quasi-Newton algorithm is much larger than that required for a full Newton–Raphson algorithm. Thus, this approach would not allow us to use Polyclass on substantially larger problems. In addition, even this smaller reduction of cpu time comes at the price of less accurate Rao statistics.

We thus note that both the computation of Rao statistics and the estimation of the parameters. require a lot of cpu time. In the rest of this section we will consider alternatives both to the selection of basis functions using Rao statistics and to the fitting of models using Newton–Raphson that require less cpu-time.

5.4.3 PolyMARS: A least squares approximation of the addition process

Using least squares regression with a 0-1 response variable is often seen as a poor man's version of logistic regression. Clearly the nonlinear binomial error structure is not taken into account properly and the fitted "probabilities" may not be between 0 and 1. But it is easy and fast. PolyMARS, a least squares approximation to the addition process of Polyclass, based on the Two-factor interactions model described in Section 3.4 is similarly quick and dirty. We have two computational advantages: the nonlinear problem becomes a linear problem, and, the design matrix is not of order p^2K^2, but only of order p^2.

To be able to apply Polyclass to large data sets we considered the following least squares approximation to the stepwise addition process when dealing with such data sets: Let $\mathbf{Z}_i$, $1 \leq i \leq n$, be the column vector of length K, whose kth element is $\text{ind}(Y_i = k)$. The estimate $\widehat{\boldsymbol{\beta}}$ of $\boldsymbol{\beta}$ is obtained by minimizing

$$V(\boldsymbol{\beta}) = \sum_i \sum_k [Z_{ik} - \theta(k|\boldsymbol{X}_i;\boldsymbol{\beta})]^2, \tag{5.4.2}$$

where $\theta(k|\boldsymbol{X}_i;\boldsymbol{\beta}) = \sum_{j=1}^{p} \beta_{jk} B_j(\boldsymbol{X}_i)$. The selection of the new basis function is carried out by finding a function B_p that minimizes $V(\widehat{\boldsymbol{\beta}})$ given the existing basis functions $B_1, \ldots, B_{p-1}$. This approximation of the stepwise addition part of Polyclass model selection reduces the cpu time required by several orders of magnitude for large problems. See below for details. Note that if in (5.4.2) $K = 1$ and the Z_{i1} are not binary we are exactly left with the Two-factor interactions algorithm in Section 3.4.

To speed up the calculations for large problems, we modified the stepwise addition algorithm by, at each stage, considering only a limited number of candidate knot locations in each variable. The resulting least squares version of the stepwise addition and deletion algorithm, whether or not $K = 1$ and whether or not the Z_{ik} are binary, is referred to as PolyMARS. This algorithm is similar to the MARS algorithm in Friedman (1991), but it is substantially faster. MARS is discussed in Section 3.4.3.

As part of the least squares approximation to Polyclass, we need to solve many equations of the form $\widehat{\boldsymbol{\beta}}_k = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y}_k$ for $1 \le k \le K$. Here $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ is a $p \times p$ matrix having a previously inverted $(p-1)\times(p-1)$ submatrix. Inverting $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ now requires only $O(p^2)$ flops. Assuming that all necessary inner products among predictors and between predictors and responses are known, computing all $\widehat{\boldsymbol{\beta}}_k$ requires $O(p^2K)$ flops.

In the context of deciding which basis function to enter next, we need to compute numerous quantities of the form $Q_k(\boldsymbol{\beta}_k) = -||\boldsymbol{Y}_k - \boldsymbol{X}\boldsymbol{\beta}_k||^2$. To evaluate the corresponding Rao statistics, we need to compute

$$[\nabla Q_k(\widehat{\boldsymbol{\beta}}_{k0})]^{\mathrm{T}}\boldsymbol{I}^{-1}\nabla Q_k(\widehat{\boldsymbol{\beta}}_{k0}).$$

Here $\boldsymbol{I} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ and $\nabla Q_k(\boldsymbol{\beta}_k) = 2\boldsymbol{X}^{\mathrm{T}}(\boldsymbol{Y}_k - \boldsymbol{X}\boldsymbol{\beta}_k)$. Only one entry of $\nabla Q_k(\widehat{\boldsymbol{\beta}}_{k0})$ is nonzero, corresponding to the candidate basis function. Since $\boldsymbol{X}\widehat{\boldsymbol{\beta}}_{k0}$ does not depend on the new basis function under consideration, it can be assumed known. Thus to compute $\nabla Q_k(\widehat{\boldsymbol{\beta}}_{k0})$ we need to compute the component of $\boldsymbol{X}^{\mathrm{T}}(\boldsymbol{Y}_k - \boldsymbol{X}\boldsymbol{\beta}_k)$ corresponding to the candidate basis function.

We also need to compute the lower-right entry of $\boldsymbol{I}^{-1}$, having already computed the inverse of the $(p-1)\times(p-1)$ submatrix corresponding to the existing basis functions. For each k this requires $O(p^2)$ flops once the p entries (inner products) corresponding to the new basis functions are determined. Thus the number of flops required for each candidate basis function is $O(p^2K)$.

If $p_{\max}$ is the largest number of basis functions that we consider, there are $p_{\max}K$ inner products between basis functions in the model and the responses and $\frac{1}{2}p_{\max}^2$ between basis functions in the model. Note that the candidate basis functions that are *not* selected for addition at one particular step of the addition process are still candidates at the next step of the addition process. If we thus limit the number of candidate basis functions that we consider (without compromising the quality of the fit) we can considerably reduce the amount of cpu time required by storing all inner products between candidate basis functions and the basis functions that are in the model and the responses. We can further reduce the amount of cpu time required when we limit the number of candidate knots in each variable. If we fix the number of candidate knots in each variable at N_0,

data set	n_{tr}	M	K	$p_{\max}$	number of operations	approximate cpu time
fruit data	6188	51	66	20	1×10^8	4 seconds
				40	3×10^8	10 seconds
phoneme data	153426	81	45	350	2×10^{11}	1.5 hours
				1000	2×10^{12}	16 hours
KDD99 data	5000000	40	5	100	5×10^{11}	4 hours
				500	1×10^{13}	3 days

TABLE 5.10. Required cpu time for applying PolyMARS with $N_0 = 10$ to some large problems (cpu time reflects approximately a Linux machine with a 2.4gHz processor).

the number of candidate basis functions (knots and interactions) remains limited to at most $N_0 M + p_{\max}^2/2$. Typically, we take N_0 between 5 and 20. In our experience, the total number of candidates is in practice only about $M + N_0 p_{\max}$. Thus approximately $(N_0 p_{\max} + M) \times (p_{\max} + K)$ inner products need to be computed between candidate basis functions and basis functions in the model and responses. Note that each inner product requires n_{tr} operations. In Table 5.10 we summarize the required cpu time for applying the PolyMARS algorithm to a number of large data sets. Note that for the smaller data sets the cpu time involved with overhead may be larger than the amount reported in Table 5.10!

PolyMARS is now much faster than the standard version (Friedman 1991). As an illustration, we generated a subset of the phoneme data with 10000 cases, 2 classes and 63 predictors, and applied both PolyMARS with $N_0 = 50$ and Friedman's program. With $p_{\max} = 40$ in both programs, our program took 177 seconds of cpu time, while Friedman's program took 2196 seconds on the same machine. With 80 basis functions the corresponding cpu times were 474 seconds and 12636 seconds. We save considerable cpu time by storing old inner products, which MARS does not and must recompute. Note that the standard version of MARS takes $O(M p_{\max}^3 n_{\text{tr}})$ flops (Friedman 1991, p. 127), while PolyMARS requires $O(N_0 p_{\max}^2 n_{\text{tr}})$ operations. Our illustrative cpu results agree with this order-of-magnitude comparison.

There are other differences between PolyMARS and standard MARS: the stepwise addition schemes are different: we add first a linear term and perhaps later a knot, while in MARS two basis functions, essentially corresponding to a linear function and a knot, are added at the same time; in MARS, but not in PolyMARS, a piecewise cubic approximation to the piecewise linear function is applied after a basis function is added.

5.4.4 Fitting Polyclass models with large data sets and many basis functions

After selecting the basis functions, we need to obtain a reasonably accurate approximation to the maximum likelihood estimate $\widehat{\boldsymbol{\beta}}$ of the coefficient vector $\boldsymbol{\beta}$[5]. For a given collection of basis functions the corresponding log-likelihood function is concave, so this numerical approximation problem is conceptually straightforward. There is a large literature about optimization of concave functions. A reference that we found particularly useful is Kennedy and Gentle (1980). When basis functions are selected with Polyclass, the approximation to $\widehat{\boldsymbol{\beta}}$ could be obtained using a Newton–Raphson algorithm. While fitting a whole sequence of Polyclass models takes about the same amount of cpu time as computing the Rao statistics, fitting just one model for a set of basis functions that have been selected otherwise (e.g. by PolyMARS) clearly is an order of magnitude faster. However, when we fit one single Polyclass model we do not have the benefit of inheriting very good starting values from the fitting of the previous model, so we would need many more iterations of the Newton–Raphson algorithm. While we have no experience in fitting such large models using this approach, we think that it is not unreasonable to expect that we may need as many as 20 iterations, rather than the four that usually suffice when we take the stepwise approach. Thus, fitting a Polyclass model this way would require approximately $20p^2K^2n_{\mathrm{tr}}$ operations. Among the six examples in Tables 5.9 and 5.10 this would be feasible for the two fruit data examples (approximately 6 hours cpu time for the smaller model and 1 day for the larger model), barely feasible for the smaller model on the KDD99 data (one month cpu time) and infeasible for all other models (years of cpu time).

An alternative to using Newton–Raphson would be to use quasi-Newton optimization to estimate the parameters. The quasi-Newton method (see Section 5.4.2) and the conjugate gradient method (discussed below) both require $O(pKn_{\mathrm{tr}})$ operations per iteration since for each method the cpu intensive activity is the computation of the gradient. In our experience a quasi-Newton algorithm requires somewhat fewer iterations than a conjugate gradient algorithm to fit the same large Polyclass model, though both may need as many as a hundred iterations, many more than a Newton–Raphson algorithm. In practice, the fact that for a quasi-Newton algorithm we need to store a quasi-Hessian with p^2K^2 elements, while for a conjugate gradient method we do not need to do this, will typically be the deciding factor in choosing between these two approaches.

[5]Note also that all methods discussed in this section separately fit each model (collection of basis functions) under consideration. We proceed as if we only consider one single model, the basis functions of which have been selected using, for example, PolyMARS.

Before we go into the details of the various optimization methods, we summarize in Table 5.11 the amount of storage needed for the Hessian for quasi-Newton and Newton–Raphson methods and the amount of cpu time needed for one pass through the data for the Newton–Raphson method and one pass through the data for the gradient based optimization methods (quasi-Newton, conjugate gradient, stochastic gradient, and stochastic conjugate gradient). Note, however, that while the cpu time per pass through the data differs among the various methods, the number of passes that are needed for reaching the same accuracy of the parameter estimates differs dramatically. We will address this later.

The conjugate gradient method

The conjugate gradient method for numerically approximating the maximum likelihood estimate $\widehat{\boldsymbol{\beta}} = \arg\max \ell(\boldsymbol{\beta})$ is an iterative method. At the beginning of the $(\nu+1)$th iteration, where ν is a nonnegative integer, we know the approximation $\boldsymbol{\beta}^{(\nu)}$ determined during the νth iteration, the value $\boldsymbol{g}^{(\nu+1)}$ of the gradient of the log-likelihood function at $\boldsymbol{\beta}^{(\nu)}$, and the search direction $\boldsymbol{\gamma}^{(\nu)}$ used during the νth iteration ($\boldsymbol{\gamma}^{(0)} = \mathbf{0}$). The search direction used during the $(\nu+1)$th iteration is given by $\boldsymbol{\gamma}^{(\nu+1)} = \boldsymbol{g}^{(\nu+1)} + b\boldsymbol{\gamma}^{(\nu)}$ for some number b, and the corresponding approximation to $\widehat{\boldsymbol{\beta}}$ is given by $\boldsymbol{\beta}^{(\nu+1)} = \boldsymbol{\beta}^{(\nu)} + a\boldsymbol{\gamma}^{(\nu+1)}$, where a is determined by a line search. (In a minimization problem, $\boldsymbol{\gamma}^{(\nu+1)} = -\boldsymbol{g}^{(\nu+1)} + b\boldsymbol{\gamma}^{(\nu)}$.)

Dixon (1975, eq. 6–9) lists four choices of b. We experimented with all of these choices, but found that, for our current problem, the choice of b has little influence on the results. We eventually settled on

$$b = \frac{{\boldsymbol{g}^{(\nu+1)}}^{\mathrm{T}}(\boldsymbol{g}^{(\nu+1)} - \boldsymbol{g}^{(\nu)})}{{\boldsymbol{g}^{(\nu)}}^{\mathrm{T}}\boldsymbol{g}^{(\nu)}},$$

which Dixon (1975) attributes to Polak and Ribiere (1969). Some authors advocate resetting the search direction to the steepest ascent direction (that is, setting $b = 0$) after every so many iterations. In our experiments, however, such a modification did not yield a significant improvement in performance.

Briefly, the motivation for the conjugate gradient method is that for a convex (concave) quadratic optimization problem with p parameters, the minimum (maximum) of the objective function is found in exactly p iterations, when exact line searches are used. The p search directions $\boldsymbol{\gamma}^{(\nu)}$, $\nu = 1, \ldots, p$ are all conjugate with respect to the Hessian matrix H of the objective function; that is,

$$[\boldsymbol{\gamma}^{(\nu_1)}]^{\mathrm{T}} H \boldsymbol{\gamma}^{(\nu_2)} = 0, \qquad \text{for all } \nu_1 \neq \nu_2. \tag{5.4.3}$$

For exact quadratic problems all four proposals of Dixon (1975) yield conjugate directions satisfying 5.4.3. See Kennedy and Gentle (1980) for more details about conjugate gradient methods.

				# elements	one pass NR		one pass gradient	
data set	n_{tr}	K	p	of Hessian	operations	cpu time	operations	cpu time
fruit data	6188	66	20	9×10^5	1×10^{10}	15 minutes	2×10^7	2 seconds
			40	3×10^6	4×10^{10}	1 hour	3×10^7	4 seconds
phoneme data	153426	45	350	1×10^8	4×10^{13}	6 weeks	5×10^9	7 minutes
			1000	1×10^9	3×10^{14}	1 year	1×10^{10}	20 minutes
KDD99 data	5000000	5	100	1×10^5	1×10^{12}	1 day	5×10^8	1 minute
			500	3×10^6	3×10^{13}	4 weeks	2×10^9	3 minutes

TABLE 5.11. Storage requirements and cpu requirements per pass for several optimization methods and some large problems.

In many references about optimization it is suggested that good line searches are particularly important for conjugate gradient methods. In our setting this does not appear to be the case. While more accurate line searches do reduce the number of required passes through the data somewhat, the cpu time spent on each individual pass increases sufficiently to completely offset any gains. A reason for this may be that our problems are not quadratic and that sometimes it is not necessary to get very close to the exact parameter vector that maximizes the log-likelihood function.

We ended up by using the following algorithm to approximate the value $\tilde{a}$ of a that maximizes $\ell(\boldsymbol{\beta} + a\boldsymbol{\gamma})$, where $\boldsymbol{\gamma}$ is the search direction found, for example, by the conjugate gradient method described above:

1. Rescale $\boldsymbol{\gamma}$ to have the same norm as $a\boldsymbol{\gamma}$ at the end of the previous step (so that $a = 1$ may be a reasonable choice for a).
2. Find three numbers a_0, a_1, a_2 in the set
$$\{0\} \cup \{\pm 2^i : i \text{ is a nonnegative integer}\}$$
such that $a_0 < a_1 < a_2$ and $\ell(\boldsymbol{\beta}+a_1\boldsymbol{\gamma}) > \max\big(\ell(\boldsymbol{\beta}+a_0\boldsymbol{\gamma}), \ell(\boldsymbol{\beta}+a_2\boldsymbol{\gamma})\big)$.
3. Find $\tilde{a}$ using quadratic interpolation.

For Polyclass, computing $\ell(\boldsymbol{\beta} + a\boldsymbol{\gamma})$ requires computing $P(Y = k|\boldsymbol{X} = \boldsymbol{x}; \boldsymbol{\beta} + a\boldsymbol{\gamma})$. This can be done very rapidly for many values of a. To see this, note that

$$\exp\theta(k|\boldsymbol{x}; \boldsymbol{\beta} + a\boldsymbol{\gamma}) = \exp\left(\sum_{j=1}^{p} \beta_{jk} B_j(\boldsymbol{x})\right)\left[\exp\sum_{j=1}^{p} \gamma_{jk} B_j(\boldsymbol{x})\right]^a.$$

Thus, if we store $\exp\theta(k|\boldsymbol{x}_i; \boldsymbol{\beta})$ and $\exp\theta(k|\boldsymbol{x}_i; \boldsymbol{\gamma})$ for all i and k, we can compute $\ell(\boldsymbol{\beta} + a\boldsymbol{\gamma})$ for every integer a in $O(Kn_{\text{tr}})$ flops without additional exponentiations.

For exact maximization of a quadratic function, a conjugate gradient method would need as many computations of the gradient as there are variables, which in our situation would mean pK iterations. As each of these iterations requires $O(pKn_{\text{tr}})$ flops this would yield an algorithm that takes the same order of magnitude cpu time as a Newton–Raphson algorithm (no free lunch!). In practice, far fewer iterations are needed to obtain a reasonable solution, but, as in the quasi-Newton method, the initial convergence may be very slow. An important advantage of the conjugate gradient method over the quasi-Newton method when $p_{\max}K$ is large is that the former method does not require the storage of a Hessian matrix.

Stochastic optimization

During any iterative optimization algorithm, when the current estimates for the parameters are already fairly close to the values of the parameters that maximize (minimize) the objective function, it is important that all calculations, such as the computations of the gradient and the Hessian, be done very precisely. However, during the initial steps of the optimization algorithm, an approximation to the gradient may be good enough to move the parameter estimate in the right direction while requiring much less cpu time. Below we discuss two stochastic optimization methods that use approximations to the gradient that are based on a random sample of the data. The *stochastic conjugate gradient method* is a modification of the conjugate gradient method where initially the gradient is computed on a subset of the data. During the iterations this subset is increased, so that later on the stochastic conjugate gradient method behaves like the regular conjugate gradient method. We first discuss a more radical approach: in the *stochastic gradient method* the gradient is computed based only on a single observation, and line searches are completely abandoned. The stochastic gradient method has been popularized in the neural network literature.

The stochastic gradient method

Recall that

$$\theta_k = \sum_{j=1}^{p} \beta_{jk} B_j(k), \qquad \text{for } k = 1, \dots, K.$$

Set $\pi_k = (\exp\theta_k)/(1+\exp\theta_k)$ for $k = 1, \dots, K$. The coefficients β_{jk}, $k = 1, \dots, K$ and $j = 1, \dots, p$, are to be determined. Given the data $(\boldsymbol{x}, y)$ for a single case in the training set, set $\delta_y = 1$ and $\delta_k = 0$ for $k \neq y$. The corresponding (contribution to the) log-likelihood is given by

$$\ell = \sum_k \delta_k\theta_k - \log(1 + \sum_k \exp\theta_k).$$

Think of the log-likelihood as a function of the coefficients. Since

$$\frac{\partial\ell}{\partial\theta_k} = \delta_k - \pi_k,$$

the gradient of the log-likelihood function is given by

$$\frac{\partial\ell}{\partial\beta_{jk}} = (\delta_k - \pi_k)B_j(\boldsymbol{x}).$$

In the stochastic gradient method, we successively update the coefficients on a case-by-case basis according to the formula

$$\beta_{jk}^{(i+1)} = \beta_{jk}^{(i)} + r_i \frac{\partial\ell}{\partial\beta_{jk}}, \tag{5.4.4}$$

where ℓ is the log-likelihood at $\boldsymbol{\beta}^{(i)} = (\beta_{jk}^{(i)})$ based on a single case. We go through the cases in the training set in random order and make a number of passes through the data, choosing a fresh random order on each pass. Note that each pass requires $O(pKn_{\mathrm{tr}})$ flops.

The stochastic gradient method has been popularized in the neural network literature, although the neural network world seems to be divided about the benefits of the method. See Ripley (1996) for one opinion. It is possible to think of Polyclass models as corresponding to a single-layer network (no hidden layers) having inputs $B_j(\boldsymbol{x})$, $j = 1, \ldots, p$, and outputs θ_k, $k = 1, \ldots, K$.

In order to apply the stochastic gradient method we need to address several issues:

- How should we adjust the *learning rate* r_i?
- Do the parameter estimates based on the stochastic gradient method "converge" and, if so, do they get close to the maximum likelihood estimates $\widehat{\boldsymbol{\beta}}$?
- How do we assess the convergence?
- How many passes through the data do we need?

These issues are simpler for Polyclass than for neural networks (Kooperberg and Stone 1999), since for Polyclass the log-likelihood function is strictly concave and hence there is a unique maximum of the log-likelihood function.

For selected sets of basis functions we fit a Polyclass model to the corresponding linear spaces using the stochastic gradient method. Initially, we set $\beta_{jk} = 0$ for all j and k. (Since the log-likelihood for Polyclass is concave, the initial values used in the stochastic gradient method are largely irrelevant.)

We tried a number of schemes for adjusting the learning rate on the phoneme data. If this rate is reduced too slowly, the algorithm converges too slowly; although it appears that the misclassification error and log-likelihood have stabilized, the parameter estimates remain unstable and they do not get close to the corresponding maximum likelihood estimates. If the learning rate is reduced too rapidly, the change in the parameters may become too small, so that the log-likelihood does not get close to its maximum.

Eventually, we settled on starting with an initial rate r_0 and dividing this rate by two after every ten full passes through the data. We found a reasonable, simple rule for determining the initial rate in our examples (see Kooperberg and Stone 1999), but we also noticed that the accuracy of the approximations after a few passes through the data is very indicative of the accuracy after a much larger number of passes.

The stochastic conjugate gradient method

Let us refer to any subset of the training set as a *block*. Given such a block, we can write the corresponding normalized log-likelihood as

$$\bar{\ell}_{\text{block}}(\boldsymbol{\beta}) = \frac{1}{\text{blocksize}} \sum_{i \in \text{block}} \ell_i(\boldsymbol{\beta}).$$

We can think of $\bar{\ell}_{\text{block}}$ as an estimate of $\Lambda(\boldsymbol{\beta}) = E[\ell(\boldsymbol{\beta})]$, where $\ell(\boldsymbol{\beta})$ is the log-likelihood based on a single random case. The maximum likelihood estimate $\widehat{\boldsymbol{\beta}} = \arg\max \bar{\ell}_{\text{training set}}(\boldsymbol{\beta})$ can thereby be viewed as a Monte Carlo estimate of $\boldsymbol{\beta}^* = \arg\max \Lambda(\boldsymbol{\beta})$. If n is large, it may be computationally attractive to use $\widehat{\boldsymbol{\beta}}_{\text{block}} = \arg\max \bar{\ell}_{\text{block}}(\boldsymbol{\beta})$ as an estimate of $\boldsymbol{\beta}^*$.

Consider now the conjugate gradient method for finding the MLE. It may be worthwhile to use a different block at each iteration. This leads to a modification of the conjugate gradient method in which, at the $(\nu+1)$th iteration, the gradient $\boldsymbol{g}^{(\nu+1)}$ and the line search are based on $\bar{\ell}_{\text{block}^{(\nu+1)}}$.

Suppose we want to make a single pass through the data. A natural approach would be to partition the data randomly into S blocks of prespecified size and then iterate, starting with a prespecified $\boldsymbol{\beta}^{(0)}$. In practice, however, we need to make repeated passes through the data. With this in mind, it is reasonable to let each block size in a given pass be approximately n/S and to let S increase from pass to pass, as more accuracy is needed at later passes. On each pass, we use a fresh random partition of the training set. In practice it is beneficial to organize each partition such that the number of cases of any particular class k is approximately the same in each block. When some classes occur much less often than other classes in the training data, it may be beneficial to have each case of one of the rarer classes in several partitions but with a smaller weight, as this will reduce the variance of $\boldsymbol{g}^{(\nu)}$ considerably.

The problem remains of choosing the initial value $S^{(0)}$ of S and of coming up with a rule for increasing S. Based on some experimenting with the phoneme data and the fruit data, we propose using $S^{(i)} = 2^{L_0}$ for $i = 0, 1, 2$ and $S^{(i)} = 2^{\min(0, L_0+2-i)}$ for $i > 2$. If L_0 is too small, the initial block size is very large and the method is not much faster than a (nonstochastic) conjugate gradient method. If L_0 is too large, the initial passes could have an adverse effect on accuracy; in particular, $\boldsymbol{\beta}^{(1)}$ could be further away from $\boldsymbol{\beta}$ than is $\boldsymbol{\beta}^{(0)}$. Fortunately, we have found that even a single pass through the data is almost always indicative of the best choice of L_0. When i increases $S^{(i)}$ eventually equals 1, after which the method essentially coincides with a nonstochastic conjugate gradient method.

We refer to this method as the *stochastic conjugate gradient method*, as does Møller (1993), where a similar method is proposed in the context of fitting neural networks (see also Ripley 1996, p. 154). The motivation for using such a method is that it can perform numerous iterations and get close to convergence with only a moderate number of passes through the data. The larger the sample size n, the more redundancy there is in the data and hence the more attractive is this approach. Still, even for a moderately small data set as the fruit data (with only 96 cases per class in the training data) we have found that the stochastic conjugate gradient method can reduce the number of iterations required by the conjugate gradient method by as many as 30 to 40. See below for a detailed analysis of the fruit data.

5.4.5 Further analysis of the fruit data

As it is (just) possible to analyze the complete fruit data using the Polyclass algorithm, we will present results coming from a direct application of this algorithm, and we will compare these results with those obtained from the least-squares approximation of PolyMARS to Polyclass, and the Polyclass fits of the basis functions selected by PolyMARS. The amounts of computing required for these two approaches differs dramatically. However, before turning to the computational issues, we first discuss the fits.

As in the analysis of the smaller subset of the fruit data, we did not use the original predictors, but projected both the training data and the test data on the principal components of the training data. We recall the sizes $n_{\text{tr}} = 6188$ of the training data and $n_{\text{ts}} = 3092$ of the test data. There are $K = 66$ classes and $M = 51$ predictors.

We used Polyclass with a maximum number $p_{\max} = 20$ of basis functions. The stepwise addition and deletion algorithm and AIC with the default parameter $\log n_{\text{tr}} = \log 6188 \approx 8.73$ yielded a model with 18 basis functions. For PolyMARS the maximum number of basis functions $p_{\max}$ was set at 150. A PolyMARS model of that size for a problem with $K = 66$ classes has $150 \times (66 - 1) = 9750$ parameters. For models that had more basis functions, we not unexpectedly encountered numerical problems caused by singularities in the design matrix. Surprisingly, the various model selection criteria suggested in Chapter 3, as well as model selection using the test-set to minimize the residual sum of squares or the test-set misclassification, all yielded model sizes of between 140 and 150. For simplicity we decided to fit a Polyclass model to the 150 basis functions selected by PolyMARS. In addition, to facilitate the comparisons with model selection carried out by Polyclass, we also fitted a Polyclass model to the first 20 basis functions selected by PolyMARS.

Table 5.12 summarizes the types of basis functions selected by Polyclass and PolyMARS. As can be seen, both algorithms select principal component (predictor) number 14 before selecting number 12 and 13. In general, there seems to be some similarity between the sequence in which predictors enter the model. Both methods start with predictors 1 and 2, and while they differ after that, both enter predictors 4 and 5 before 3, and predictor 8 before 6. That is, however, where the similarities stop. Clearly the PolyMARS algorithm enters more knots and interactions early on.

Selected by	Polyclass	PolyMARS	PolyMARS
Number of basis functions			
Total	18(20)	20	150
Intercept	1	1	1
Linear (x_i)	11	6	12
Knot ($(x_i - t_{ik})$)	4(5)	8	25
Linear $\times$ linear	2(3)	4	26
Linear $\times$ knot	0	1	61
Knot $\times$ knot	0	0	25
Predictors involved	X_1–X_9, X_{11}, X_{14}	X_1–X_5, X_8	X_1–X_{11}, X_{14}

TABLE 5.12. Types of basis functions selected for the fruit data. The two additional basis functions between parenthesis for Polyclass were removed by the stepwise algorithm.

In Table 5.13 we display the results after 40 and after 2000 passes through the data using a stochastic conjugate gradient algorithm. When fitting Polyclass models to basis functions selected by PolyMARS, we notice that for this data set after about 40 passes through the data using the stochastic conjugate gradient method, the test-set log likelihood reaches a maximum and decreases during further passes, while the training-set log-likelihood and the training data misclassification error continue to improve. The test data misclassification error is minimized at approximately 40 passes through the data, and it remains constant after that. The reason is that, because of the large number of parameters in these models, Polyclass is able to improve the fitted probability to nearly 1 for some cases. If, among the test data, the fitted probability is nearly 1 for a wrong class for some cases, then the fitted probability for the right class for these cases will be extremely small, thus considerably reducing the test data log-likelihood.

Note that when we use the PolyMARS fit for classification, "probabilities" (fitted values) are not guaranteed to be between 0 and 1. As such, we cannot provided a log-likelihood. As it turns out, the median fitted value is between 0 and 1. This is the number reported between parenthesis for the PolyMARS fit.

We note from Table 5.13 that refitting basis functions selected by PolyMARS using Polyclass substantially improves the percentage of correctly classified cases in both the training and the test set. Basis functions selected using Polyclass are substantially better than those selected by PolyMARS, as the numbers for the Polyclass fit to the model with 20 basis functions selected by PolyMARS are considerably worse than those for the pure Polyclass model. The model with 150 basis functions selected by PolyMARS after 40 passes of fitting using the stochastic conjugate gradient method yields uniformly better summary statistics than the straight Polyclass model. In addition, the combined PolyMARS/Polyclass method takes considerably less cpu time, as we will see below. This would there-

Selection method	Polyclass	PolyMARS	PolyMARS	PolyMARS	PolyMARS	PolyMARS	PolyMARS
Number of basis functions	18	20	20	20	150	150	150
Fitting method	Polyclass	PolyMARS	Polyclass	Polyclass	PolyMARS	Polyclass	Polyclass
Stoc. conj. gradient passes	–	–	40	2000	–	40	2000
Training data							
Percentage correct class	75.7	42.4	64.2	69.7	70.0	79.8	91.0
Percent correct class among the six classes with the highest fitted probability	96.5	86.1	93.1	95.4	95.3	97.6	99.6
Average exp(log-likelihood)	0.445	–	0.294	0.384	–	0.513	0.761
Median exp(log-likelihood)	0.722	(0.133)	0.448	0.644	(0.391)	0.810	1.000
Test data							
Percentage correct class	70.9	41.8	61.6	64.3	67.3	74.1	73.1
Percent correct class among the six classes with the highest fitted probability	95.2	84.8	91.9	93.2	93.4	95.6	95.8
Exp(mean log-likelihood)	0.339	–	0.252	0.182	–	0.374	0.014
Exp(median log-likelihood)	0.684	(0.132)	0.419	0.573	(0.366)	0.774	0.998

TABLE 5.13. Performance of Polyclass and PolyMARS on the 66 class fruit data with $n_{\mathrm{tr}} = 6188$ and $n_{\mathrm{ts}} = 3092$.

selection	fitting	# of basis functions	# of passes	cpu time
Polyclass	Polyclass	18(20)	NA	20 hours
PolyMARS	PolyMARS	20	NA	14 seconds
PolyMARS	Polyclass	20	40	2 minutes
PolyMARS	Polyclass	20	2000	1.3 hour
PolyMARS	PolyMARS	150	NA	9 minutes
PolyMARS	Polyclass	150	40	17 minutes
PolyMARS	Polyclass	150	2000	7 hours

TABLE 5.14. Amount of cpu time involved with obtaining fits for the fruit data.

fore be our model of choice. If no test data set is available, this approach would still outperform Polyclass if a cross-validation approach is used to determine when to stop the optimization algorithm.

We now further examine the phenomenon of the early stopping in the fitting algorithm. In Figure 5.9 we show histograms of the fitted probabilities for the correct class for the model with 150 basis functions selected by PolyMARS after 40 and after 2000 passes through the data using a stochastic conjugate gradient algorithm. We note that, for the training data, increasing the number of passes through the data for the fitting algorithm leads to many more fitted probabilities close to 1. For example, after 40 passes 8.7% of the probabilities are larger than 0.999 while after 2000 passes this percentage is 52.8. For the test data the probabilities also increase, and actually the fraction of large probabilities is extremely similar to that for the training data: after 40 passes 9.7% are larger than 0.999, and after 2000 passes this percentage is 48.5. At the same time the percentage of cases with probabilities smaller than 0.01 decreases for the training data from 0.2% to 0.1%, but it increases for the test data from 1.5% to 8.6%.

As mentioned earlier, the amounts of computing involved for the various approaches to fitting Polyclass or PolyMARS models differ considerably. In Table 5.14 we summarize how much cpu time was involved for each of the seven fits presented in Table 5.13. As can be seen, the differences are substantial.

Since for each of the methods the cpu time and the performance (in log-likelihood or classification) both depend considerably on the number of basis functions, we show in Figure 5.10 how for each of the methods the performance as a function of the number of basis functions and in Figure 5.11 the required cpu time as a function of the number of basis functions. For the fits in which basis functions were selected using PolyMARS and fitted using Polyclass we used 40 passes through the data of a stochastic conjugate gradient algorithm for the fitting. As can be seen from Figure 5.10, fitting coefficients of basis functions selected with PolyMARS using Polyclass substantially improves the performance, in particular when

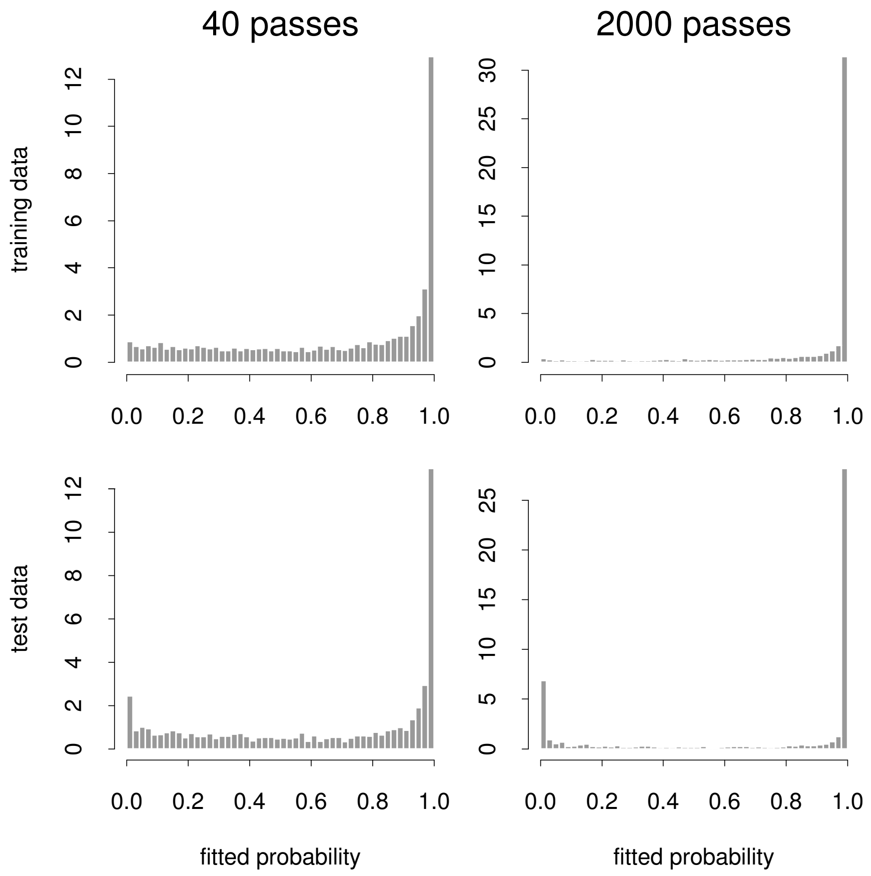

FIGURE 5.9. Histograms of the fitted probability for the correct class for the fruit data with 66 classes and the model with 150 basis functions selected by PolyMARS.

the number of basis functions is larger than about 10. For that many basis functions, using Polyclass directly gives even better results. On the other hand, Figure 5.11 shows that the cpu requirements for the better approaches are considerably larger. (Note that in Table 5.14 and Figure 5.11 the time involved with selecting the PolyMARS basis functions is included for the combined procedure.) To accommodate for the differences, a fairer comparison is one where we compare the log-likelihood or misclassification rate as a function of the required cpu time, as is done in Figure 5.12. As can be seen from this figure, for any performance level using Polyclass is less efficient than selecting basis functions using PolyMARS and fitting models using Polyclass. If only classification results are needed, and fast, but not super-accurate, results suffice, PolyMARS yields acceptable results using less cpu time than the alternative approaches. However, when more accurate results are needed, the results of the combined procedure are better.

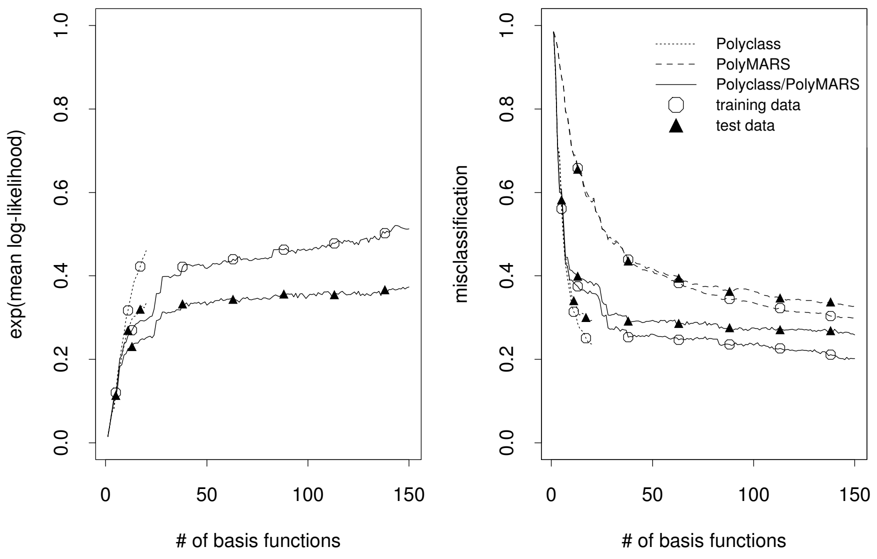

FIGURE 5.10. Misclassification and exponent of the mean log-likelihood as a function of the number of basis functions for three approaches of the Polyclass and PolyMARS methods to the fruit data.

In addition, using PolyMARS by itself does not yield valid probabilities, as estimated probabilities are regularly smaller than 0 or larger than 1.

We will now focus more on the different approaches to optimization of the log-likelihood function after selection of basis functions using PolyMARS. We present results for the model with 150 basis functions. Figure 5.13 shows the exponent of the average log-likelihood for the training data, on the left side as a function of the required cpu time, and on the right side as a function of the number of passes through the data. While on the right side a quasi-Newton algorithm appears competitive, we notice on the left side that it is not. The reason is that for the larger fruit data with 150 basis functions the size of the quasi-Hessian is almost 10000×10000, so that, although no matrix inversions are required, the other matrix operations for the quasi-Newton algorithm require substantial cpu time. On some computers the size of this matrix may also require a lot of "swapping" in and out of memory, further slowing down the program. Conceivably, for problems with large data sets but a smaller number of parameters (e.g. the KDD99 data), quasi-Newton algorithms perform better. We did not include a stochastic quasi-Newton algorithm in the comparison, as such an algorithm would involve a considerably increased number of matrix operations of the quasi-Hessian per pass through the data compared to the regular quasi-Newton algorithm. However, for problems with fewer parameters such an algorithm may be competitive. We note that, for the stochastic conjugate gradient method, the likelihood increases somewhat faster than for the regular conjugate

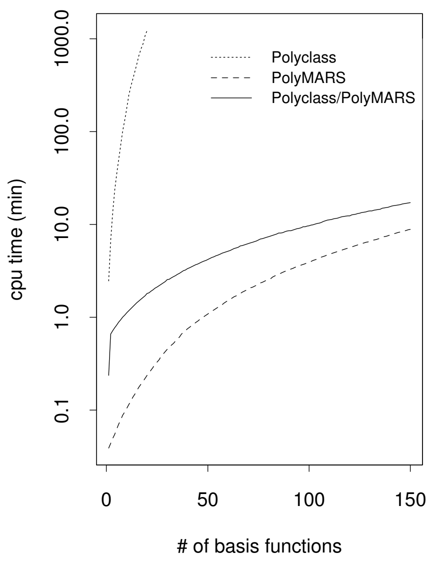

FIGURE 5.11. Required cpu time as a function of the number of basis functions for three approaches of the Polyclass and PolyMARS methods to the fruit data.

gradient method. These methods both require somewhat more cpu time per pass through the data than the stochastic gradient method, as both conjugate gradient methods involve a line-search. However, after the first few iterations it is very hard to determine a good step-size for a stochastic gradient algorithm. The algorithm that we present, with a slowly decreasing step-size, seemed to work best on the current problem.

The stochastic conjugate gradient method that we used employed 16 blocks during the first three passes through the data, after which the number of blocks was reduced by half after each pass, until only one block was left. The improvement of the stochastic conjugate gradient method over the regular conjugate gradient method is fairly modest in this case. This is due to the fact that there is relatively little redundancy in the fruit data. To illustrate what happens in a more redundant data set, we created a new fruit data set by repeating the original fruit data 16 times. We now used a stochastic conjugate gradient method with 128 initial blocks. The comparison of the stochastic and the regular conjugate gradient method is shown on the left side of Figure 5.14. For both this problem and the original problem we can compute how many passes through the data we gain using a stochastic algorithm, by figuring out how many passes later that for the nonstochastic conjugate gradient method the log-likelihood reaches the same level as the stochastic version reached. This is effectively the horizontal distance between the two curves in the left side of Figures 5.13 and 5.14. From the right side of Figure 5.14 we thus see that a stochastic version

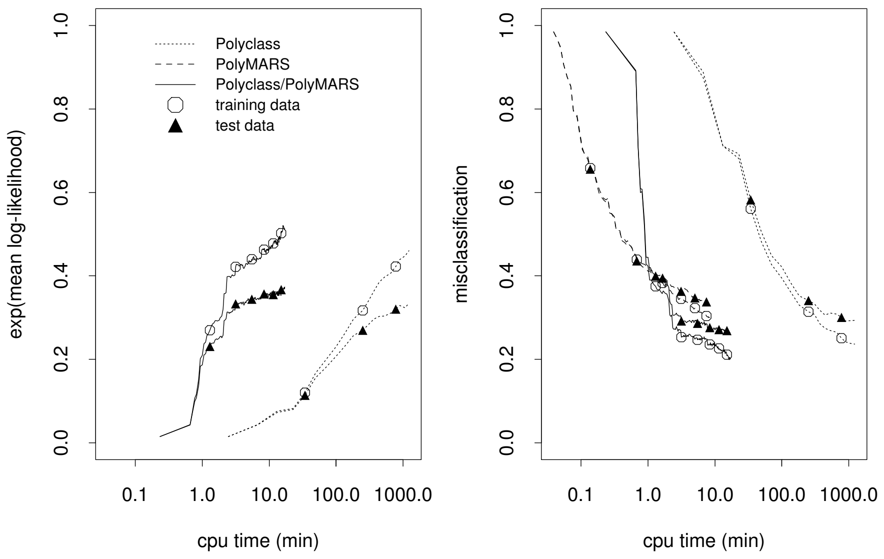

FIGURE 5.12. Misclassification and exponent of the mean log-likelihood as a function of the required cpu time for three approaches of the Polyclass and PolyMARS methods to the fruit data.

saves us about 60 passes through the data for the replicated data set and about 15 passes for the original data.

5.5 Technical details of the Polyclass algorithm

A number of the technical details about the algorithms that are discussed in this chapter and some of the later chapters have already been discussed in Chapters 3 and 4. In particular, these chapters contain details about the stepwise algorithm, AIC, Rao and Wald statistics and the Newton–Raphson algorithm. In this section we give a few details about the Polyclass, Logspline, Heft, and Hare methodologies, that are discussed in this chapter and the next two chapters.

5.5.1 Maximum number of basis functions

The theoretical results (Chapter 11) suggest that the maximum number of basis functions should increase with a low power of the sample size of the (training) data set. Default rules for the implementations that we discuss have typically been set by trial and error. There are a number of other considerations in setting the maximum number of basis functions, such as:

- in particular, if more than one parameter is involved per basis function (as is the case for Polyclass) or when there are many repeat obser-

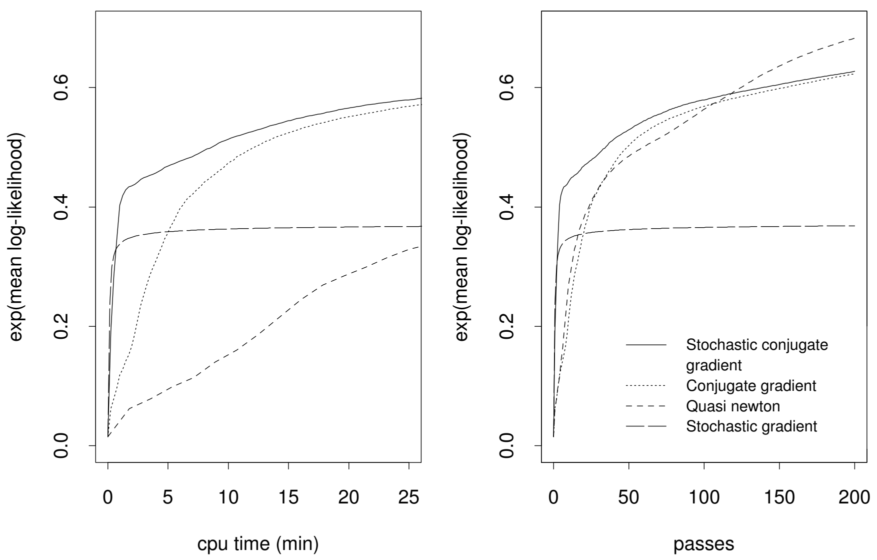

FIGURE 5.13. Exponent of the mean log-likelihood on the training data as a function of the number of passes through the data and as a function of the required cpu time for a variety of optimization methods.

vations, it is wise to reduce the maximum number of basis functions to assure a minimum number of unique observations per parameter;

- the computational complexity of stepwise polynomial spline algorithms typically increases more than linearly with the number of basis functions, so that it may make sense to have an overall maximum model size for the default;

- when basis functions are entered one at a time, if for several additions in a row there is very little improvement in the fit (e.g. there is very little increase in the fitted log-likelihood), it may make sense to stop adding basis functions before the original planned maximum number of basis functions is reached.

The default maximum number of basis functions for Polyclass, Logspline (Chapter 6), Hare and Heft (Chapter 7), and Lspec (Chapter 8) are all based on these principles. In particular, we stop the addition of basis functions for Polyclass when one of the following conditions is satisfied:

- the number p of basis functions equals $p_{\max}$, whose default value is $\min(4n^{1/3}, n/(2K), 50)$;
- $\hat{l}_p - \hat{l}_q < \frac{1}{2}(p-q) - 0.5$ for some q with $q \le p-3$, where $\hat{l}_q$ is the log-likelihood for the model with q parameters (so the addition of more basis functions is not likely to improve the fit);
- the search algorithm yields no possible new basis function.

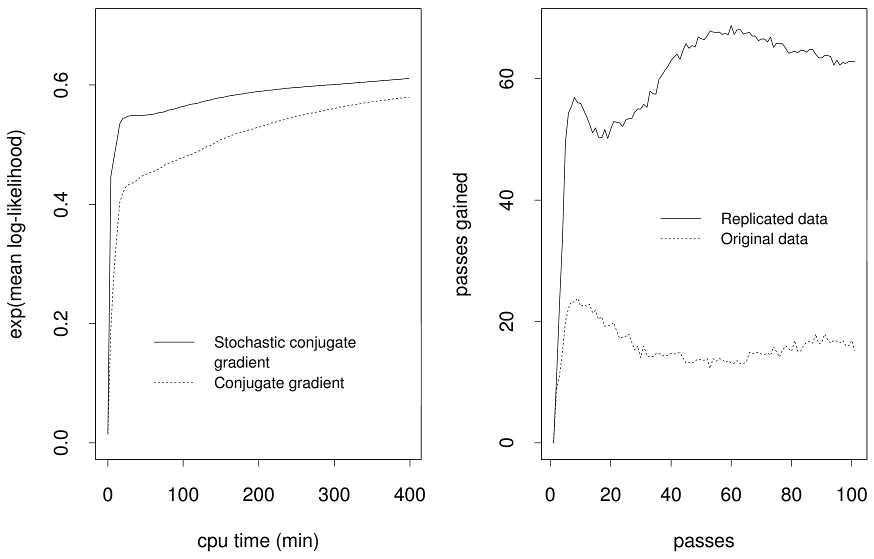

FIGURE 5.14. Exponent of the mean log-likelihood as a function of the number of passes through the data for the replicated data and the number of passes through the data gained by using a stochastic conjugate gradient method.

Clearly, in practice there are other issues that may be reasons for users to override these defaults. For example, for large data sets on fast computers the maximum of 50 may be unnecessary. In addition, when the final selected model using AIC has a number of basis functions p larger than, say, $0.8p_{\max}$, a larger value of $p_{\max}$ may be advisable.

5.5.2 Optimizing the location of a new knot.

One problem that repeatedly comes up in stepwise polynomial spline algorithms is finding a location for a new knot. Typically, during the process of stepwise addition of basis functions, at each step we find the basis function that has the largest Rao statistic among the candidate basis function whose addition to an allowable space would still render the larger space to be allowable. While the exact details differ among the different multidimensional procedures (Polyclass, PolyMARS, Hare) this typically means finding the basis function with the largest Rao statistic among

1. linear functions in variables that are not yet in the model;
2. functions that involve a new knot in a variable that is already in the model;
3. tensor products of two basis functions that depend only on one variable and keep the new space allowable.

See Section 3.4 for the details about multivariate regression. For the univariate procedures (Logspline, Heft, and Lspec), where no tensor product basis functions exist, and linear functions (if applicable) are in the minimal model, this means finding the best knot.

Since it is possible to come up with simple updating formulas to compute score functions and Hessians for PolyMARS(MARS) and Lspec, finding a new knot location is not a major issue. However, for Polyclass, Logspline, Heft, and Hare this is not possible, so we need to limit the number of knots for which we compute the Rao statistic.

In this section we describe the algorithm for finding the location of a potential new knot in a particular variable. For the multivariate procedures (Polyclass and Hare) we would typically carry out this algorithm for each variable that is in the model.

To find a new knot let $t_1 < t_2 < \cdots < t_K$ be the corresponding knots presently in the model, to which we want to add one more knot, and let $X_{(1)}, \ldots, X_{(m)}$ be the data written in nondecreasing order. Define l_i and u_i by $l_0 = 1$, $u_K = n$,

$$l_i = d_{\min} + \max\{j\colon 1 \le j \le n \text{ and } X_{(j)} \le t_i\}, \qquad i = 1, \ldots, K, \tag{5.5.1}$$

and

$$u_i = -d_{\min} + \min\{j\colon 1 \le j \le n \text{ and } X_{(j)} \ge t_{i+1}\}, \qquad i = 0, \ldots, K-1. \tag{5.5.2}$$

Here $d_{\min}$ is the minimum distance between consecutive knots in order statistics. Typically we take $d_{\min} = 3$ in procedures employing cubic splines, and a slightly larger value for linear splines. For $i = 0, \ldots, K$ we compute the Rao statistic r_i for the model with $X_{(j_i)}$ as a new knot, where $j_i = [(l_i + u_i)/2]$. (If any index is not an integer, we interpolate between data points.) Because of the $d_{\min}$ in (5.5.1) and (5.5.2) it is possible that $u_i < l_i$ for some i; if so, then no knot can be added between t_i and t_{i+1}. This forces knots to be at least $d_{\min}$ order statistics apart, which improves the numerical and statistical stability. If there is no i for which $u_i \ge l_i$, then no knots can be added to the model.

We place the potential new knot in the interval $[X_{(l_{i^*})}, X_{(u_{i^*})}]$, where $i^* = \arg\max r_i$. Within this interval we further optimize the location of the new knot. To this end, we proceed by computing the Rao statistic r_l for the model with $X_{(l)}$ as the knot with $l = [(l_{i^*} + j_{i^*})/2]$ and r_u for the model with $X_{(u)}$ as the knot with $u = [(j_{i^*} + u_{i^*})/2]$. If $r_{i^*} \ge r_l$ and $r_{i^*} \ge r_u$, we place the new knot at $X_{(i^*)}$; if $r_{i^*} < r_l$ and $r_| \ge r_u$, we continue searching for a knot location in the interval $[X_{(l_{i^*})}, X_{(j_{i^*})}]$; and if $r_{i^*} < r_u$ and $r_l < r_u$, we continue searching for a knot location on the interval $[X_{(j_{i^*})}, X_{(u_{i^*})}]$.

This procedure needs some modifications for variables with many repeated observations and for variables that involve censoring. We omit these (ad hoc) details.

5.6 Notes

Literature

There is an enormous literature on classification (discrimination, supervised learning) methods. See Mardia, Kent, and Bibby (1979) for an overview of "classical" discriminant analysis. Rather than remain incomplete, we only provide references on approaches to classification that involve splines.

Anderson and Blair (1982) and Villalobos and Wahba (1983) are two early papers that uses smoothing splines in logistic regression and classification. Because of the limited computing available at the time of these papers, Anderson and Blair (1982) gives no examples, and Villalobos and Wahba (1983) only presents small two class examples, and even for those examples many approximations need to be made.

Classification trees, e.g. CART (Breiman, Friedman, Olshen, and Stone 1984), can be seen as piecewise constant splines. While the adaptive model selection strategy of CART has similarities with Polyclass, the type of models that are selected by CART, which involve high order interactions, are quite different from those considered by Polyclass, which favors simpler additive models. Tibshirani and LeBlanc (1992) proposes a procedure for classification that can be seen as a special form of MARS, using stepfunctions as basis functions. This method also borrows much from the CART methodology.

Hastie, Tibshirani, and Buja (1994) proposes flexible discriminant analysis, which combines nonparametric regression techniques, such as smoothing splines and MARS, and discriminant analysis. Bose (1996) uses additive B-Splines and least squares regression to develop a classification rule.

There is a rich and voluminous literature on stochastic approximation, starting with Robbins and Monro (1951), which includes the one-pass version of the stochastic gradient method as a special case. In particular, in the context of fitting Polyclass models with a fixed collection of basis functions (or neural network models with a fixed number of hidden units) White (1989) applied results of Ljung (1977) to obtain conditions under which convergence should occur as $n \to \infty$. In these conditions the random pairs $(\boldsymbol{X}_1, Y_1)$, $(\boldsymbol{X}_2, Y_2), \ldots$ should be independent and have a common distribution with compact support and the learning rate r_n should be (say) of the form $A/(B+n)$. With probability one, the successive iterates $\boldsymbol{\beta}^{(n)}$ for $\widehat{\boldsymbol{\beta}}$ then either converge to a local maximum of the log-likelihood function or diverge ($|\boldsymbol{\beta}^{(n)}| \to \infty$). In particular, when the log-likelihood function is strictly concave, as in the fitting of a Polyclass model, the successive iterates either converge to $\widehat{\boldsymbol{\beta}}$ or diverge. We are unaware of similar theoretical results that deal with repeated passes through the data or with models whose size (e.g., the number of basis functions of Polyclass or the number of hidden units of a neural network model) depends on n.

Software

Programs for implementing Polyclass and PolyMARS as described in this chapter have been written in C and an interface based on R and S-Plus has been developed. These programs are currently available as part of the `polspline` package from `CRAN`

`http://cran.r-project.org/src/contrib/PACKAGES.html`

and from Kooperberg's website

`http://bear.fhcrc.org/~clk/soft.html`

The PolyMARS interface through S-Plus works well for data sets with several tens of thousands of cases if there are not too many classes. For data sets with many classes, memory use increases. Several others have ported this S-Plus and C code for easy installation on other platforms and under the R language. See Kooperberg's website for current links. The `agml()` function, developed by Insightful Corp., implements an extended linear modeling approach to logistic regression, a two-class version of Polyclass. A link to the appropriate Insightful website is on Kooperberg's website.

Programs implementing the various optimization approaches were never written as general purpose programs.