are given as

$$\begin{aligned}\nabla_{\tilde{\mathbf{w}}} g\left(\tilde{\mathbf{w}}\right) &= -2\sum_{p=1}^{P} y_p \tilde{\mathbf{f}}_p \max\left(0,\, 1 - y_p \tilde{\mathbf{f}}_p^T \tilde{\mathbf{w}}\right) \\ \nabla_{\tilde{\mathbf{w}}}^2 g\left(\tilde{\mathbf{w}}\right) &= 2\sum_{p \in \Omega_{\tilde{\mathbf{w}}}} \tilde{\mathbf{f}}_p \tilde{\mathbf{f}}_p^T,\end{aligned} \tag{6.29}$$

where $\Omega_{\tilde{\mathbf{w}}}$ is the index set $\Omega_{\tilde{\mathbf{w}}} = \left\{p | 1 - y_p \tilde{\mathbf{f}}_p^T \tilde{\mathbf{w}} > 0\right\}$. These are entirely similar to the calculations given in Example 4.2 except for using the feature map $\mathbf{f}_p$ in place of the input $\mathbf{x}_p$.

b) Show that the squared margin cost using M elements of any fixed feature basis is convex. *Hint: see Exercise 4.6.*

Exercises 6.3 Polynomial basis features and the softmax cost

In this exercise you will explore how various degree D polynomial basis features fit using the softmax cost and the dataset shown in the bottom panel of Fig. 6.6. For this exercise you will need the wrapper *poly_classification_hw* and the data located in *2eggs_data.csv*.

a) Use the description of the two-dimensional polynomial basis features given in footnote 5 to transform the input using a general degree D polynomial. Write this feature transformation in the module

$$\mathbf{F} = \text{poly\_features}\ (\mathbf{X}, D) \tag{6.30}$$

located in the wrapper. Here $\mathbf{X}$ is the input data, D the degree of the polynomial features, and $\mathbf{F}$ the corresponding degree D feature transformation of the input (note your code should be able to transform the input to any degree D desired).

b) With your module complete you may run the wrapper. Two figures will be generated: the first shows the data along with various degree D polynomial fits, and the second shows the average number of misclassifications of each fit to the data-set. Discuss the results shown in these two figures. In particular, describe the relationship between a model's average number of misclassifications and how well it seems to represent the phenomenon generating the data as D increases over the range shown.

Exercises 6.4 Calculate the gradient using a single hidden layer basis

When employing M single hidden layer basis features (using any activation $a(\cdot)$) the full gradient of a cost g (e.g., the softmax) is a vector of length $Q = M(N+2)+1$ containing the derivatives of the cost with respect to each variable,

$$\nabla g = \begin{bmatrix} \frac{\partial}{\partial b} g & \frac{\partial}{\partial w_1} g & \cdots & \frac{\partial}{\partial w_M} g & \frac{\partial}{\partial c_1} g \cdots & \frac{\partial}{\partial c_M} g & \nabla_{\mathbf{v}_1}^T g & \cdots & \nabla_{\mathbf{v}_M}^T g \end{bmatrix}^T, \tag{6.31}$$

where the derivatives are easily calculated using the chain rule.

a) Using the chain rule verify that the derivatives of this gradient (using the softmax cost) are given by

$$\begin{aligned}
\frac{\partial}{\partial b} g &= -\sum_{p=1}^{P} \sigma\left(-y_p\left(b + \sum_{m=1}^{M} w_m a\left(c_m + \mathbf{x}_p^T \mathbf{v}_m\right)\right)\right) y_p \\
\frac{\partial}{\partial w_n} g &= -\sum_{p=1}^{P} \sigma\left(-y_p\left(b + \sum_{m=1}^{M} w_m a\left(c_m + \mathbf{x}_p^T \mathbf{v}_m\right)\right)\right) a\left(c_n + \mathbf{x}_p^T \mathbf{v}_n\right) y_p \\
\frac{\partial}{\partial c_n} g &= -\sum_{p=1}^{P} \sigma\left(-y_p\left(b + \sum_{m=1}^{M} w_m a\left(c_m + \mathbf{x}_p^T \mathbf{v}_m\right)\right)\right) a'\left(c_n + \mathbf{x}_p^T \mathbf{v}_n\right) w_n y_p \\
\nabla_{\mathbf{v}_n} g &= -\sum_{p=1}^{P} \sigma\left(-y_p\left(b + \sum_{m=1}^{M} w_m a\left(c_m + \mathbf{x}_p^T \mathbf{v}_m\right)\right)\right) a'\left(c_n + \mathbf{x}_p^T \mathbf{v}_n\right) \mathbf{x}_p w_n y_p.
\end{aligned} \tag{6.32}$$

b) This gradient can be written more efficiently for programming languages like Python and MATLAB/OCTAVE that have especially good implementations of matrix/vector operations by writing it more compactly. Supposing that $a = \tanh(\cdot)$ is the activation function (meaning $a' = \text{sech}^2(\cdot)$ is the hyperbolic secant function squared), verify that the derivatives from part a) may be written more compactly as

$$\begin{aligned}
\frac{\partial}{\partial b} g &= -\mathbf{1}_{P\times 1}^T \mathbf{q} \odot \mathbf{y} \\
\frac{\partial}{\partial w_n} g &= -\mathbf{1}_{P\times 1}^T (\mathbf{q} \odot \mathbf{t}_n \odot \mathbf{y}) \\
\frac{\partial}{\partial c_n} g &= -1_{P\times 1}^T (\mathbf{q} \odot \mathbf{s}_n \odot \mathbf{y}) w_n \\
\nabla_{\mathbf{v}_n} g &= -\mathbf{X} \cdot \mathbf{q} \odot \mathbf{s}_n \odot \mathbf{y} w_n,
\end{aligned} \tag{6.33}$$

where $\odot$ denotes the component-wise product and denoting $q_p = \sigma\left(-y_p\left(b + \sum_{m=1}^{M} w_m \tanh\left(c_m + \mathbf{x}_p^T \mathbf{v}_m\right)\right)\right)$, $t_{np} = \tanh\left(c_n + \mathbf{x}_p^T \mathbf{v}_n\right)$, $s_{np} = \text{sech}^2\left(c_n + \mathbf{x}_p^T \mathbf{v}_n\right)$, and $\mathbf{q}$, $\mathbf{t}_n$, and $\mathbf{s}_n$ the P length vectors containing these entries.

Exercises 6.5 Code up gradient descent using single hidden layer bases

In this exercise you will reproduce the classification result using a single hidden layer feature basis with tanh activation shown in the middle panel of Fig. 6.9.

a) Plug the gradient from Exercise 6.4 into the gradient descent function

$$\mathbf{T} = \text{tanh\_softmax}\,(\mathbf{X}, \mathbf{y}, M) \tag{6.34}$$

located in the wrapper *single_layer_classification_hw* and the dataset *genreg_data.csv*, both of which may be downloaded from the book website. Here $\mathbf{T}$ is the set of optimal weights learned via gradient descent, $\mathbf{X}$ is the input data matrix, $\mathbf{y}$ contains the associated labels, and M is the number of basis features to employ.

Almost all of this function has already been constructed for you, e.g., various initializations, step length, etc., and you need only enter the gradient of the associated cost function. All of the additional code necessary to generate the associated plot is already provided in the wrapper. Due to the non-convexity of the associated cost function when using neural network features, the wrapper will run gradient descent several times and plot the result of each run.

b) Try adjusting the number of basis features M in the wrapper and run it several times. Is there a value of M other than $M = 4$ that seems to produce a good fit to the underlying function?

Exercises 6.6 Code up the *k*-nearest neighbors (*k*-NN) classifier

The *k-nearest neighbors* (*k*-NN) is a local classification scheme that, while differing from the more global feature basis approach described in this chapter, can produce nonlinear boundaries in the original feature space as illustrated for some particular examples in Fig. 6.18.

With the *k*-NN approach there is no training phase to the classification scheme. We simply use the training data directly to classify any new point $\mathbf{x}_{\text{new}}$ by taking the average of the labels of its *k*-nearest neighbors. That is, we create the label y_{new} for a point $\mathbf{x}_{\text{new}}$ by simply calculating

$$y_{\text{new}} = \text{sign}\left(\sum_{i \in \Omega} y_i\right), \tag{6.35}$$

where Ω is the set of indices of the k closest training points to $\mathbf{x}_{\text{new}}$. To avoid tie votes (i.e., a value of zero above) typically the number of neighbors k is chosen to be odd (however, in practice the value of k is typically set via cross-validation).

Code up the k-NN algorithm and reproduce the results shown in Fig. 6.18 using the dataset located in *knn_data.csv*.

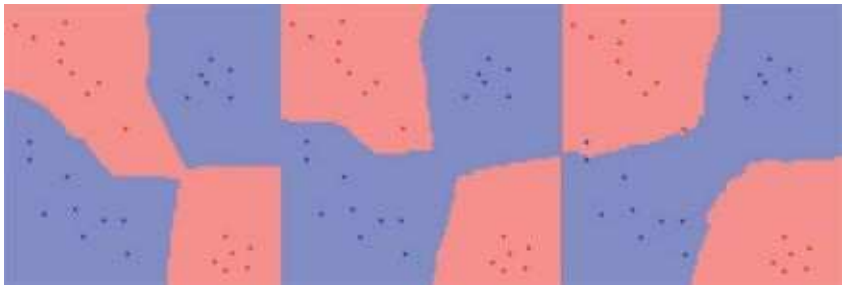

Fig. 6.18 The k-NN classifier applied to a two class dataset (the blue and red points) where (left panel) $k = 1$, (middle panel) $k = 5$, and (right panel) $k = 10$. All points in the space have been colored according to the rule given in Equation (6.35) where the red and blue classes have labels $+1$ and -1 respectively.

Section 6.2 exercises

Exercises 6.7 One-versus-all using a polynomial basis

In this exercise you will reproduce the one-versus-all classification on the $C = 3$ class dataset shown in the bottom panels of Fig. 6.10 using polynomial features. Note that for this exercise you will need to have completed the poly_features module for polynomial features described in Exercise 6.3.

a) Place the *poly_features* module in the wrapper *ova_fixed_basis* and use the dataset *bullseye_data.csv*, both of which may be downloaded from the book website. After installing the module try running the wrapper to reproduce the results shown in Fig. 6.10.

b) Try adjusting the degree D in the wrapper and run it several times. Is there a value of D other than $D = 2$ that seems to produce a good fit to the data?

Section 6.3 exercises

Exercises 6.8 Code up hold out cross-validation

In this exercise you will perform hold out cross-validation on the dataset shown in Fig. 6.12 as described in Example 6.5 of the text. Start by randomly splitting the dataset, located in *2eggs_data.csv*, into $k = 3$ equal sized folds (keeping 2 folds as training, and 1 fold as testing data). Use the polynomial basis features with M in the range $M = 2, 5, 9, 14, 20, 27, 35, 44$ (or likewise D in the range $D = 1, 2, \ldots, 8$) and produce a graph showing the training and testing error for each D like the one shown in Fig. 6.12, as well as the best (i.e., the lowest test error) model fit to the data.

Note: your results may appear slightly different than those of the figure, given that you will likely use a different random partition of the data. Note: you may find it very useful here to re-use code from previous exercises e.g., functions that compute polynomial features, plot curves, etc.

Exercises 6.9 Code up *k*-fold cross-validation

In this exercise you will perform k-fold cross-validation on the dataset shown in Fig. 6.13 as described in Example 6.6 of the text. Start by randomly splitting the dataset, located in *2eggs_data.csv*, into $k = 3$ equal sized folds (keeping 2 folds as training, and 1 fold as testing data). Use the polynomial basis features with M in the range $M = 2, 5, 9, 14, 20, 27, 35, 44$ (or likewise D in the range $D = 1, 2, \ldots, 8$) and produce a graph showing the training and testing error for each D like the one shown in Fig. 6.13, as well as the best (i.e., the lowest average test error) model fit to the data.

Note: your results may appear slightly different than those of the figure given that you will likely use a different random partition of the data. Note: you may find it very useful here to re-use code from previous exercises e.g., functions that compute polynomial features, plot curves, etc.

---

[1] The piecewise continuous functions in the top and bottom panels are defined over the unit interval, respectively as

$$y(x) = \begin{cases} +1 & 0.33 \leq x \leq 0.67 \\ -1 & \text{else,} \end{cases} \qquad y(x) = \begin{cases} 0.25\left(\left(1 - 10x^2\right)\cos(10\pi x) + 1\right) & 0 \leq x < 0.33 \\ 0.8 & 0.33 \leq x \leq 0.67 \\ 4\left(x^2 - 0.75\right) + 0.4 & 0.67 < x \leq 1. \end{cases} \tag{6.1}$$

[2] As with continuous function approximation, the details of this statement are quite technical, and we do not dwell on them here. Our goal is to provide an intuitive high level understanding of this sort of function approximation. See Section 5.7 for further information and reading.