converge extremely rapidly to a reasonable solution of a problem, even after only a single pass through the dataset.

8.7 Exercises

Section 8.1 exercises

Exercises 8.1 Code up backtracking line search for the squared margin cost

Code up an adaptive step length sub-function for the minimization of the squared margin perceptron and install it into the wrapper detailed in Exercise 4.7, replacing the fixed step length given there. Test your code by running the wrapper, and produce a plot of the objective value decrease at each iteration.

Exercises 8.2 Code up stochastic gradient descent

Reproduce a part of the experiment shown in Example 8.4 by comparing standard and stochastic gradient descent methods on a large two class classification dataset of $P = 10000$ points. Employ any cost function (e.g., softmax) to fit to this data and plot the cost value at each iteration from runs of each method, along with the average value of these runs (as in Fig. 8.8).

Exercises 8.3 Code up backtracking line search for the multiclass softmax cost

Code up an adaptive step length sub-function for the minimization of the multiclass softmax perceptron and install it into the wrapper detailed in Exercise 4.15, replacing the fixed step length given there. Test your code by running the wrapper, and produce a plot of the objective value decrease at each iteration.

Exercises 8.4 Alternative formal definition of Lipschitz gradient

An alternative to defining the Lipschitz constant by Equation (8.11) for functions f with Lipschitz continuous gradient is given by

$$\left\|\nabla f\left(\mathbf{x}\right) - \nabla f\left(\mathbf{y}\right)\right\|_2 \leq L \left\|\mathbf{x} - \mathbf{y}\right\|_2 , \tag{8.60}$$

which follows from the limit definition of a derivative (in defining the Hessian of f). This

9.4 Summary

In this chapter we have described several methods for dimension reduction, beginning in Section 9.1 describing commonly used data dimension reduction techniques (or ways of properly reducing the size of a dataset). Random subsampling, described first, is the most commonly used method for slimming down a regression/classification dataset that simply involves keeping a random selection of points from an original dataset. Here one typically keeps as much of the original data as computational resources permit. A popular alternative to random subsampling, also used for various "data analysis" tasks beyond predictive modeling, is the K-means clustering method. This approach involves the computation of cluster "centroids" of a dataset that best describe its overall structure.

In Section 9.2 we described a classic technique for feature dimension reduction (that is shrinking the dimension of each data point) referred to as principal component analysis (PCA). While not particularly useful for predictive modeling (see Example 9.2 for how it can in fact be destructive when applied to classification data), PCA is the fundamental matrix factorization problem which can help frame our understanding of a wide array of models/problems including: Least Squares for linear regression, K-means, and more. For example in the final section of the chapter we detailed a common matrix factorization approach to recommender systems, or algorithms that recommend products/services to a common base of users.

9.5 Exercises

Section 9.1 exercises

Exercises 9.1 Code up K-means

In this exercise you will reproduce the results shown in Fig. 9.4 by coding up the K-means algorithm (shown in Algorithm 9.1).

a) Place your K-means code in the function

$$[\mathbf{C}, \mathbf{W}] = \text{your\_}K\text{\_means}\left(\mathbf{X}, K\right), \tag{9.25}$$

located inside the wrapper *kmeans_demo* (this wrapper together with the assocated dataset *kmeans_demo_data.csv* may be downloaded from the book website). All of the additional code necessary to generate the associated plots is already provided in the wrapper. Here **C** and **W** are the centroid and assignment matrices output by the algorithm, while **X** and K are the data matrix and number of desired centroids, respectively.

b) Run the wrapper with $K = 2$ centroids using the initialization $\mathbf{C} = \begin{bmatrix} 0 & 0 \\ -0.5 & 0.5 \end{bmatrix}$. This should reproduce the successful run of *K*-means shown in the bottom panels of the figure.

c) Run the wrapper with $K = 2$ centroids using the initialization $\mathbf{C} = \begin{bmatrix} 0 & 0 \\ 0 & 0.5 \end{bmatrix}$. This should reproduce the unsuccessful run of *K*-means shown in the top panels of the figure.

Section 9.2 exercises

Exercises 9.2 Code up PCA

In this exercise you will reproduce the results shown in Fig. 9.6 by coding up PCA.

a) Implement a singular value decomposition approach to PCA described in Section 9.2.1, placing the resulting code in the function

$$[\mathbf{C}, \mathbf{W}] = \text{your\_PCA}\left(\mathbf{X}, K\right), \tag{9.26}$$

located inside the wrapper *PCA_demo* (this wrapper together with the assocated dataset *PCA_demo_data.csv* may be downloaded from the book website). Here **C** and **W** are the spanning set and weight matrices output by the algorithm, while **X** and *K* are the data matrix and number of desired basis elements, respectively.

All of the additional code necessary to generate the associated plots is already provided in the wrapper. Run the wrapper to ensure that you have coded up PCA correctly.

b) Using the wrapper and data from part a) implement the alternating directions solution to PCA described in footnote 3, once again placing this code inside the function *your_PCA* described previously.

Exercises 9.3 Deriving principal components as orthogonal directions of variance

In this exercise you will show how to derive principal component analysis as the orthogonal directions of largest variance of a dataset. Given *P* points $\left\{\mathbf{x}_p\right\}_{p=1}^{P}$ of dimension *N* we may calculate the variance in a unit direction **d** (i.e., how much the dataset spreads out in the direction $\mathbf{d}$) with respect to the data as the average squared inner product of the data against $\mathbf{d}$,

$$\frac{1}{P}\sum_{p=1}^{P}\left\langle \mathbf{x}_p, \mathbf{d}\right\rangle^2. \tag{9.27}$$

This can be written more compactly as

$$\frac{1}{P}\left\|\mathbf{X}^T\mathbf{d}\right\|_2^2 = \frac{1}{P}\mathbf{d}^T\mathbf{X}\mathbf{X}^T\mathbf{d}. \tag{9.28}$$

Note that the outer product $\mathbf{X}\mathbf{X}^T$ is a symmetric positive semi-definite matrix.

a) Compute the largest direction of variance of the data, i.e., the unit vector $\mathbf{d}$ that maximizes the value $\mathbf{d}^T\mathbf{X}\mathbf{X}^T\mathbf{d}$. *Hint: use the eigen-decomposition of* $\mathbf{X}\mathbf{X}^T$.

b) Compute the second largest direction of variance of the matrix $\mathbf{X}\mathbf{X}^T$, i.e., the unit vector $\mathbf{d}$ that maximizes the value of $\mathbf{d}^T\mathbf{X}\mathbf{X}^T\mathbf{d}$ but where $\mathbf{d}$ is also orthogonal to the first largest direction of variance. *Hint: use the eigen-decomposition of* $\mathbf{X}\mathbf{X}^T$.

c) Conclude from part a) and b) that the orthogonal directions of variance of the data are precisely the singular value solution given in Equation (9.17).

Section 9.3 exercises

Exercises 9.4 Code up the matrix completion recommender system

In this exercise you will reproduce the matrix completion recommender system results shown in Fig. 9.12.

Code up the alternating minimization algorithm described in Section 9.3.2, placing the resulting code in the function

$$[\mathbf{C}, \mathbf{W}] = \text{matrix\_complete}\left(\mathbf{X}, K\right), \tag{9.29}$$

located inside the wrapper *recommender_demo* (this wrapper and the assocated dataset *recommender_demo_data.csv* may be downloaded from the book website). Here $\mathbf{C}$ and $\mathbf{W}$ are the spanning set and weight matrices output by the algorithm, while $\mathbf{X}$ and K are the data matrix and number of desired basis elements, respectively.

All of the additional code necessary to generate the associated plots is already provided in the wrapper. Run the wrapper to ensure that you have coded up the matrix completion algorithm correctly.

---

1 Although it is possible to adopt a variety of different ways to define similarity between data points in the feature space (e.g., spectral clustering and subspace clustering), proximity in the Euclidean sense is the most popular measure for clustering data.

2 Pictures of natural subjects such as cities, meadows, people, and animals, etc., as opposed to synthetic images.

3 Beginning at an initial value for both $\left(\mathbf{C}^{(0)},\mathbf{W}^{(0)}\right)$ this produces a sequence of iterates $\left(\mathbf{C}^{(k)},\mathbf{W}^{(k)}\right)$ where

$$\begin{aligned}\mathbf{C}^{(k)} &= \underset{\mathbf{C}}{\operatorname{argmin}} \left\|\mathbf{C}\mathbf{W}^{(k-1)}-\mathbf{X}\right\|_F^2 \\ \mathbf{W}^{(k)} &= \underset{\mathbf{W}}{\operatorname{argmin}} \left\|\mathbf{C}^{(k)}\mathbf{W}-\mathbf{X}\right\|_F^2,\end{aligned} \tag{9.16}$$

and where each may be expressed in closed form. Setting the gradient in each case to zero and solving gives $\mathbf{C}^{(k)} = \mathbf{X}\left(\mathbf{W}^{(k-1)}\right)^T\left(\mathbf{W}^{(k-1)}\left(\mathbf{W}^{(k-1)}\right)^T\right)^\dagger$ and $\mathbf{W}^{(k)} = \left(\left(\mathbf{C}^{(k)}\right)^T\mathbf{C}^{(k)}\right)^\dagger\left(\mathbf{C}^{(k)}\right)^T\mathbf{X}$ respectively, where $(\cdot)^\dagger$ denotes the pseudo-inverse. The procedure is stopped when the subsequent iterations do not change significantly (see exercises).