Exercises 4.3 Code up gradient descent for the softmax cost/logistic regression on a toy dataset

In this exercise you will code up gradient descent to minimize the softmax cost function on a toy dataset, reproducing the left panel of Fig. 4.3 in Example 4.1.

a) Verify the gradient of the softmax cost shown in Equation (4.12).

b) (optional) 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 in matrix-vector form as

$$\nabla g\left(\widetilde{\mathbf{w}}\right) = \widetilde{\mathbf{X}}\mathbf{r}, \tag{4.71}$$

where $\widetilde{\mathbf{X}}$ is the $(N+1) \times P$ matrix formed by stacking the P vectors $\widetilde{\mathbf{x}}_p$ column-wise, and where $\mathbf{r}$ is a $P \times 1$ vector based on the form of the gradient shown in Equation (4.12). Verify that this can be done and determine $\mathbf{r}$.

c) Code up gradient descent to minimize the softmax cost, reproducing the left panel of Fig. 4.3. This figure is generated via the wrapper *softmax_grad_demo_hw* using the dataset *imbalanced_2class.csv*. You must complete a short gradient descent function located within the wrapper which takes the form

$$\widetilde{\mathbf{w}} = \text{softmax\_grad}\left(\widetilde{\mathbf{X}}, \mathbf{y}, \widetilde{\mathbf{w}}^0, \text{alpha}\right). \tag{4.72}$$

Here $\widetilde{\mathbf{w}}$ is the optimal weights learned via gradient descent, $\widetilde{\mathbf{X}}$ is the input data matrix, $\mathbf{y}$ the output values, and $\widetilde{\mathbf{w}}^0$ the initial point.

Almost all of this function has already been constructed for you. For example, the step length is given and fixed for all iterations, etc., and you must 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.

$$\nabla g_2\left(\tilde{\mathbf{w}}\right) = -\sum_{p=1}^{P} \sigma\left(-y_p \tilde{\mathbf{x}}_p^T \tilde{\mathbf{w}}\right) y_p \tilde{\mathbf{x}}_p = \mathbf{0}_{(N+1)\times 1}. \tag{4.12}$$

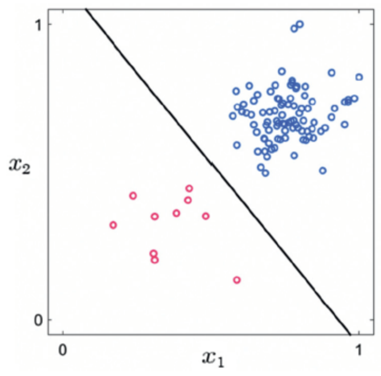

Fig 4.3 left panel

Exercises 4.5 The softmax cost and diverging weights with linearly separable data

Suppose that a two class dataset of P points is linearly separable, and that the pair of finite-valued parameters $(b, \mathbf{w})$ defines a separating hyperplane for the data.

a) Show while multiplying these weights by a positive constant $C > 1$, that as $(C \cdot b,\ C \cdot \mathbf{w})$ does not alter the equation of the separating hyperplane, the scaled parameters reduce the value of the softmax cost as $g\left(C \cdot b, C \cdot \mathbf{w}\right) < g\left(b, \mathbf{w}\right)$ where g is the softmax cost in (4.9). *Hint: remember from (4.3) that if the point $\mathbf{x}_p$ is classified correctly then* $-y_p\left(b + \mathbf{x}_p^T \mathbf{w}\right) < 0$.

b) Using part a) describe how, in minimizing the softmax cost over a linearly separable dataset, it is possible for the parameters to grow infinitely large. Why do you think this is a problem, practically speaking?

There are several simple ways to prevent this problem: one is to add a stopping condition that halts gradient descent/Newton's method if the parameters $(b,\ \mathbf{w})$ become larger than a preset maximum value. A second option is to add an ℓ_2 regularizer (see Section 3.3.2) to the softmax cost with a small penalty parameter λ, since adding the regularizer $\lambda \left\|\mathbf{w}\right\|_2^2$ will stop $\mathbf{w}$ from growing too large (since otherwise the value of the regularized softmax cost will grow to infinity).

Exercises 4.5 The softmax cost and diverging weights with linearly separable data

Suppose that a two class dataset of P points is linearly separable, and that the pair of finite-valued parameters $(b, \mathbf{w})$ defines a separating hyperplane for the data.

a) Show while multiplying these weights by a positive constant $C > 1$, that as $(C \cdot b,\ C \cdot \mathbf{w})$ does not alter the equation of the separating hyperplane, the scaled parameters reduce the value of the softmax cost as $g\left(C \cdot b, C \cdot \mathbf{w}\right) < g\left(b, \mathbf{w}\right)$ where g is the softmax cost in (4.9). *Hint: remember from (4.3) that if the point $\mathbf{x}_p$ is classified correctly then* $-y_p\left(b + \mathbf{x}_p^T \mathbf{w}\right) < 0$.

b) Using part a) describe how, in minimizing the softmax cost over a linearly separable dataset, it is possible for the parameters to grow infinitely large. Why do you think this is a problem, practically speaking?

There are several simple ways to prevent this problem: one is to add a stopping condition that halts gradient descent/Newton's method if the parameters $(b,\ \mathbf{w})$ become larger than a preset maximum value. A second option is to add an ℓ_2 regularizer (see Section 3.3.2) to the softmax cost with a small penalty parameter λ, since adding the regularizer $\lambda \left\|\mathbf{w}\right\|_2^2$ will stop $\mathbf{w}$ from growing too large (since otherwise the value of the regularized softmax cost will grow to infinity).

Exercises 4.9 **Perform classification on the breast cancer dataset**

Compare the efficacy of the softmax and squared margin costs in distinguishing healthy from cancerous tissue using the entire breast cancer dataset as training data, located in

breast_cancer_dataset.csv, first discussed in Example 4.3. This dataset consists of $P = 699$ data points, with each data point having nine medically valuable features (i.e., $N = 9$) which you may read about by reviewing the readme file *breast_cancer_readme.txt*. Note that for simplicity we have removed the sixth feature from the original version of this data, taken from [47], due to its absence from many of the data points.

To compare the two cost functions create a plot like the one shown in Fig. 4.8, which compares the number of misclassifications per iteration of Newton's method as applied to minimize each cost function over the data (note: depending on your initialization it could take between 10–20 iterations to achieve the results shown in this figure). As mentioned in footnote 6, you need to be careful not to overflow the exponential function used with the softmax cost here. In particular make sure to choose a small initial point for your Newton's method algorithm with the softmax cost.

Exercises 4.12 **Probabilistic perspective of logistic regression**

In the previous chapter (in Section 3.3.1) we first introduced logistic regression in the context of its original application: modeling population growth. We then followed this geometric perspective to re-derive the softmax cost function in the instance of classification.

In the classification setting, logistic regression may also be derived from a probabilistic perspective.[22] Doing so one comes to the following cost function for logistic regression:

$$h\left(b,\mathbf{w}\right) = -\sum_{p=1}^{P} \bar{y}_p \log \sigma\left(b + \mathbf{x}_p^T\mathbf{w}\right) + \left(1 - \bar{y}_p\right) \log\left(1 - \sigma\left(b + \mathbf{x}_p^T\mathbf{w}\right)\right), \qquad (4.81)$$

where the modified labels $\bar{y}_p$ are defined as

$$\bar{y}_p = \begin{cases} 0 & \text{if } y_p = -1 \\ 1 & \text{if } y_p = +1. \end{cases} \qquad (4.82)$$

Show that the cost function $h\left(b,\mathbf{w}\right)$, also referred to as the *cross-entropy* cost for logistic regression, is equivalent to the softmax cost function $g\left(b,\mathbf{w}\right) = \sum_{p=1}^{P} \log\left(1 + e^{-y_p\left(b + \mathbf{x}_p^T\mathbf{w}\right)}\right)$.

Hint: this can be done in cases, i.e., suppose $y_p = +1$, show that the corresponding summand of the softmax cost becomes that of the cross-entropy cost when substituting $\bar{y}_p = 1$.

Exercises 4.16 **Handwritten digit recognition**

In this exercise you will perform $C = 10$ multiclass classification for handwritten digit recognition, as described in Example 4.4 , employing the OvA multiclass classification framework. Employ the softmax cost with gradient descent or Newton's method to solve each of the two-class subproblems.

a) Train your classifier on the training set located in *MNIST_training_data.csv*, that contains $P = 60\,000$ examples of handwritten digits $0 - 9$ (all examples are vectorized grayscale images of size 28×28 pixels). Report the accuracy of your trained model on this training set.

b) Using the weights learned from part a) report the accuracy of your model on a new test dataset of handwritten digits located in *MNIST_testing_data.csv*. This contains $P = 10\,000$ new examples of handwritten digits that were not used in the training of your model.