CSE 151: Machine learning **Fall 2019**

Homework 2

Submission instructions:

- If a problem asks for a numerical answer, you need only provide this answer. There is no need to show your work, unless you would like to.
- Please type up your solutions. We suggest using an online latex editor like `www.overleaf.com`, though this is not a requirement.
- Upload the PDF file for your homework to `gradescope` by midnight on Thursday, November 14 by 11:59pm.

1. We have a data set $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$, where $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$. We want to express y as a linear function of x, but the error penalty we have in mind is not the usual squared loss: if we predict $\widehat{y}$ and the true value is y, then the penalty should be the absolute difference, $|y - \widehat{y}|$. Write down the loss function that corresponds to the total penalty on the training set.

   Solution

   $L(w) = \sum_{i=1}^{d} |y^{(i)} - w \cdot x^{(i)}|$

2. Consider the following loss function on vectors $w \in \mathbb{R}^4$:

   $$L(w) = w_1^2 + 2w_2^2 + w_3^2 - 2w_3w_4 + w_4^2 + 2w_1 - 4w_2 + 4.$$

   (a) What is $\nabla L(w)$?

   (b) Suppose we use gradient descent to minimize this function, and that the current estimate is $w = (0, 0, 0, 0)$. If the step size is η, what is the next estimate?

   (c) What is the minimum value of $L(w)$?

   (d) Is there is a unique solution w at which this minimum is realized?

   Solution

   (a) $\nabla L(w) = \begin{pmatrix} 2w_1 + 2 \\ 4w_2 - 4 \\ 2w_3 - 2w_4 \\ 2w_4 - 2w_3 \end{pmatrix}$

(b) $\begin{pmatrix} 2\eta \\ -4\eta \\ 0 \\ 0 \end{pmatrix}$

(c) 1

(d) No, the function is minimized anywhere $w_1 = -1$, $w_2 = 1$, and $w_3 = w_4$

3. For each of the following functions of one variable, say whether it is convex, concave, both, or neither.

   (a) $f(x) = x^2$
   (b) $f(x) = -x^2$
   (c) $f(x) = x^2 - 2x + 1$
   (d) $f(x) = x$
   (e) $f(x) = x^3$
   (f) $f(x) = x^4$
   (g) $f(x) = \ln x$

Solution

(a) convex

(b) concave

(c) convex

(d) both

(e) neither

(f) convex

(g) concave

4. Show that the matrix $M = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ is not positive semidefinite.

Solution

For M to be positive semidefinite, $z^T M z \geq 0$ for all $z \in \mathbb{R}^2$
$\begin{pmatrix} a & b \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = ab + ab$
When $a = 1$, $b = -1$, this expression is -2, and therefore this matrix is not positive semidefinite.

5. Show that the matrix $M = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$ is positive semidefinite.

Solution

$$\begin{pmatrix} a & b \end{pmatrix} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = a^2 - 2ab + b^2 = (a-b)^2$$

Because of this squaring, there is no z in $\mathbb{R}^2$ that can make $z^T M z$ less than 0. M is positive semidefinite.

6. We are given a set of data points $x^{(1)}, \ldots, x^{(n)} \in \mathbb{R}^d$, and we want to find a single point $z \in \mathbb{R}^d$ that minimizes the loss function

$$L(z) = \sum_{i=1}^{n} \|x^{(i)} - z\|^2.$$

Use calculus to determine z, in terms of the $x^{(i)}$.

Solution

Simplify the loss function:

$$\begin{aligned} L(w) &= \sum_{i=1}^{d} \left(\sqrt{\left(x^{(i)} - z\right)^2} \right)^2 \\ &= \sum_{i=1}^{d} \left(x^{(i)} - z \right)^2 \end{aligned}$$

Take derivative with respect to z:

$$\begin{aligned} \frac{\delta}{\delta z} L(w) &= \frac{\delta}{\delta z} \sum_{i=1}^{d} \left(x^{(i)} - z \right)^2 \\ &= \sum_{i=1}^{d} \frac{\delta}{\delta z} \left(x^{(i)} - z \right)^2 \\ &= \sum_{i=1}^{d} \left(-2x^{(i)} + 2z \right) \\ 0 &= 2zd + \sum_{i=1}^{d} -2x^{(i)} \\ \sum_{i=1}^{d} 2x^{(i)} &= 2zd \\ \frac{\sum_{i=1}^{d} x^{(i)}}{d} &= z \end{aligned}$$

7. Consider the loss function for ridge regression (ignoring the intercept term):

$$L(w) = \sum_{i=1}^{n} (y^{(i)} - w \cdot x^{(i)})^2 \; + \; \lambda \|w\|^2$$

where $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathbb{R}^d \times \mathbb{R}$ are the data points and $w \in \mathbb{R}^d$. There is a closed-form equation for the optimal w (as we saw in class), but suppose that we decide instead to minimize the function using local search.

(a) What is $\nabla L(w)$?

(b) Write down the update step for gradient descent.

(c) Write down a stochastic gradient descent algorithm.

Solution

(a)

(b)

(c)

8. Draw the decision boundary in $\mathbb{R}^2$ that corresponds to the prediction rule $\text{sign}(2x_1 - x_2 - 6)$. Make sure to clearly indicate where this boundary intersects the axes. Show which side of the boundary is classified as positive and which side as negative.

Solution

9. The Perceptron algorithm is run on a data set, and converges after performing $p+q$ updates (i.e. makes $p + q$ incorrect predictions during training). Of these updates, p are on data points whose label is -1 and q are on data points whose label is $+1$. What is the final value of the parameter b? (Equivalently, what is the final value of the weight on the bias feature?)

Solution

10. In this problem, you are going to build a *logistic regression* digit classifier using the same MNIST dataset you used in past homeworks. If you don't already have them, first go and download the train, validation, and test sets from the class website. (For a description of the data format, please see the last problem of Homework 0.) **Please append your final code for this problem to the end of your PDF submission.**

Since the label set for this task consists of ten possible digits, you will be building a *multi-class* logistic regression classifier. The conditional probability of the digit label, y, given the image feature vector, x, when using this model class can be written as:

$$P_w(y|x) = \frac{\exp(w_y \cdot x)}{\sum_{y'} \exp(w_{y'} \cdot x)}$$

If the image feature vector is d-dimensional (i.e. $x \in \mathbb{R}^d$), then, for each digit type $y \in \{0, 1, 2, \ldots, 9\}$, the model has a d-dimension parameter vector, $w_y \in \mathbb{R}^d$. The role of each w_y in this *multi-class* model is to determine the linear score, $(w_y \cdot x)$, of the corresponding digit, y, paired with feature vector, x. Ideally, once trained, the score of the correct digit will be higher than the scores of the rest of the digits, given the input, x. We can write the full set of parameters as $w = (w_0, w_1, \ldots, w_9)$, which we can view as a $(10 \cdot d)$-dimensional vector (i.e. all digit-specific vectors appended together).

(a) Before actually coding up this model, we need write down the objective function. Let our training data be denoted as a sequence of pairs: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})$. Here, $x^{(i)}$ is the image feature vector of the ith training datum, and $y^{(i)}$ is the correct digit label. Write maximum likelihood estimation for the multi-class logistic regression model on this training set in the form of a minimization problem, including a term for L2 regularization. Use λ to denote the hyperparameter that is the weight of the regularization term. Use log-likelihood rather than likelihood. This will help numerical stability when you implement your system. [Hint: In other words, if training consists of finding w to minimize the function $L(w)$, what should $L(w)$ be?]

Solution

(b) Write the equation for $\nabla L(w)$. Remember to include the regularization term. [Hint: you can always just differentiate $L(w)$ w.r.t. each dimension of w separately and then collect the results in a vector. However, it may be faster if you try to use vector calculus directly, computing the gradient w.r.t. each class-specific vector w_y in turn.]

Solution

(c) Now it's time to begin coding. You are going to build and train a logistic regression classifier. We have provided skeleton PyTorch code in "logreg.py" for you to complete. Please complete your code in PyTorch without calling external logistic regression libraries (e.g. in numpy).

First, we have to define a feature representation of the input images, i.e. you have to define the dimensions of x. For this part, we will use the following simple feature representation: Let x be 57-dimensional. The first 28 dimensions of x will contain the number of dark pixels in each row of the input image (i.e. x_i will be the number of pixels that have a value above or below some threshold, ϵ, in row i of the image). Similarly, let the next 28 dimensions of x contain the number of dark pixels in each column. Finally, the last dimension of x will be the intercept term, which will always have value 1. The data loader that maps each image to a vector x using this feature representation is provided for you in in the skeleton.

Next, you need to implement gradient descent to learn the model parameters, w, from the training set. In PyTorch, it is possible to simply implement $L(w)$ and let PyTorch compute the gradients automatically. The skeleton code currently does this for a dummy loss function. The skeleton code also implements a simple training loop that iteratively updates w based on the current gradient of the loss and a step-size hyperparameter, "learning_rate". You may need to experiment with several settings of "learning_rate" in order to find a step size that works on this dataset and objective. In PyTorch, it is also possible to use built-in gradient optimization methods instead of doing gradient updates yourself. Feel free to explore this approach as an alternative.

Feel free to get advice on your implementation from other students and from TAs, but make sure that you write your own code. You may need to do some debugging to get gradient descent to work properly. We recommend printing out the value of the training objective after each iteration, as well as the accuracy of the classifier on the validation set. This will help you find appropriate hyperparameters. You may encounter issues with numerical stability depending on how you implement your gradient computation. Remember, directly exponentiating large values (or large negative values) will lead to unstable results. Use alternative formulas involving logs when possible.

Finally, for this part, we want you to plot (or simply list the values of) the objective function as it changes over 20 iterations of gradient descent for the best hyperparameter settings you found. Which values of λ, "learning_rate', and ϵ lead to the best validation accuracy? What validation accuracy were you able to achieve? [Note: using our simple feature representation, your classification accuracy won't be great but should be better than chance.]

Get started by having a look at "logreg.py" and trying to understand what each line does. The code will run as given, but instead of logistic regression we've implemented a dummy loss function. See the relevant "TODO" sections.

Solution

(d) Now, implement stochastic gradient descent. Your step size, "learning_rate", will likely need to be smaller than it was for standard gradient descent. Plot (or list the values of) the objective after each of 10 epochs of stochastic gradient descent for the best hyperparameter settings you can find. Does stochastic gradient descent lead to faster training?

Solution

(e) Finally, we're going to turn you loose to define your own feature representations. Feel free to define any new features you like. It's also fine to discard the old features, and design better ones from scratch. Find new features that improve your best validation accuracy from part (d) by at least 0.5%. This should not be difficult. Much better gains are possible. [Hint: You may need to try new hyperparameters or use a larger number of gradient descent steps depending on the features you built. Feel free to use more advanced optimization methods, e.g. Adagrad or LBFGS. Also, in general, features that consider 'edges' in the image are typically quite helpful.]

Describe the new features you implemented and report the best validation accuracy you achieved. Also include a plot (or list of values) of your training objective over the first 20 iterations. If you used a more advanced optimization method, tell us about it.

Solution