Machine Learning Fall 2019 Homework 2

Homework must be submitted electronically on Canvas. Make sure to explain you reasoning or show your derivations. Except for answers that are especially straightforward, you will lose points for unjustified answers, even if they are correct.

General Instructions

You are allowed to work with at most one other student on the homework. With your partner, you will submit only one copy, and you will share the grade that your submission receives. You should set up your partnership on Canvas as a two-person group.

Submit your homework electronically on Canvas. We recommend using LaTeX, especially for the written problems. But you are welcome to use anything as long as it is neat and readable.

Since we may work on some of the homework in class, clearly indicate which parts of your submitted homework are work done in class and which are your own work.

Relatedly, cite all outside sources of information and ideas.

Written Problems

1. Multiclass logistic regression has the form

$$p(y|\boldsymbol{x}; W) := \frac{\exp(\boldsymbol{w}_y^\top \boldsymbol{x})}{\sum_{c=1}^{C} \exp(\boldsymbol{w}_c^\top \boldsymbol{x})} \,, \tag{1}$$

where W is a set of weight vectors $\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_C\}$ for each class.

   (a) (5 points) You have a batch of n data points and labels $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$. Write the conditional log-likelihood of the labels given the features and simplify it. Show and explain the steps you take to simplify the expression. You should end up with an expression very similar to Equation (7).

   (b) (5 points) Derive the gradient of the logistic regression likelihood with respect to any one of the $\boldsymbol{w}_c$ vectors. It should look very similar to Equation (8) or Equation (9) below (either form is acceptable for full credit). Show and explain the steps you take to derive the gradient.

   (c) (5 points) We often add a regularizer (as we do below for the programming portion) that penalizes the magnitude of the weight vectors. These regularizers "prefer" the weights to be the all-zeros vector. What is the estimated class probability $p(y|\boldsymbol{x}; W)$ of this all-zeros solution? Explain in one to three sentences why this probability is reasonable as the maximally regularized solution.

Programming Assignment

For this assignment, you will run an experiment comparing two different versions of linear classifiers for multiclass classification.

Models

The two models you will implement are perceptron and logistic regression. The multiclass forms of these models are summarized here.

Multiclass Perceptron The multiclass perceptron uses a weight vector for each class, which can conveniently be represented with a matrix $W = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_C\}$. The prediction function is

$$f_{\text{perc}}(\boldsymbol{x}) := \operatorname*{arg\,max}_{c \in \{1,\ldots,C\}} \boldsymbol{w}_c^\top \boldsymbol{x} = \operatorname*{arg\,max}_{c \in \{1,\ldots,C\}} \left[W^\top \boldsymbol{x}\right]_c . \tag{2}$$

The multiclass perceptron update rule when learning from example $\boldsymbol{x}_t$, ground-truth label y_t is.

$$\boldsymbol{w}_{y_t} \leftarrow \boldsymbol{w}_{y_t} + \boldsymbol{x}_t \tag{3}$$

$$\boldsymbol{w}_{(f_{\text{perc}}(\boldsymbol{x}))} \leftarrow \boldsymbol{w}_{(f_{\text{perc}}(\boldsymbol{x}))} - \boldsymbol{x}_t \tag{4}$$

Multiclass Logistic Regression Multiclass logistic regression also uses a weight vector for each class, and in fact has the same prediction formula as perceptron.

$$f_{\text{lr}}(\boldsymbol{x}) := \operatorname*{arg\,max}_{c \in \{1,\ldots,C\}} \boldsymbol{w}_c^\top \boldsymbol{x} = \operatorname*{arg\,max}_{c \in \{1,\ldots,C\}} \left[W^\top \boldsymbol{x}\right]_c . \tag{5}$$

The key difference from perceptron is that it is built around a probabilistic interpretation:

$$p_{\text{lr}}(y|\boldsymbol{x}; W) := \frac{\exp(\boldsymbol{w}_y^\top \boldsymbol{x})}{\sum_{c=1}^{C} \exp(\boldsymbol{w}_c^\top \boldsymbol{x})} . \tag{6}$$

For data set $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, the regularized negative log likelihood is

$$L(D) = \frac{\lambda}{2} ||W||_{\text{F}}^2 + \sum_{i=1}^{n} \log \left(\sum_c \exp(\boldsymbol{w}_c^\top \boldsymbol{x}_i) \right) - \sum_{i=1}^{n} \boldsymbol{w}_{y_i}^\top \boldsymbol{x}_i \tag{7}$$

where $||W||_{\text{F}}^2$ is the squared Frobenius norm $\sum_{ij} \boldsymbol{w}_i[j]^2$, and the gradient of the log likelihood is

$$\nabla_{\boldsymbol{w}_c} L = \lambda \boldsymbol{w}_c + \sum_{i=1}^{n} \boldsymbol{x}_i \left(\frac{\exp(\boldsymbol{w}_c^\top \boldsymbol{x}_i)}{\sum_{c'} \exp(\boldsymbol{w}_{c'}^\top \boldsymbol{x}_i)} - I(y_i = c) \right) \tag{8}$$

$$= \lambda \boldsymbol{w}_c + \sum_{i=1}^{n} \boldsymbol{x}_i \left(p_{\text{lr}}(y = c|\boldsymbol{x}_i; W) - I(y_i = c) \right) \tag{9}$$

Experiments

You'll build learning and prediction functions for perceptron and linear regression. You should be able to run an experiment on synthetic data, which will be possible to visualize on the screen, and on real medical data. These experiments are already set up for you in the two iPython notebooks: `synthetic_experiments.ipynb` and `cardio_experiments.ipynb`. The cardiotocography data includes measurements from fetal cardiotocograms that medical experts diagnosed into ten possible diagnosis classes.[1] Both experiments will measure how the perceptron model behavior changes as it sees more examples and how logistic regression behaves as you vary its regularization parameter.

You are required to use Python 3.6 or higher for this assignment. We will grade your code in a Python 3.7 environment (most likely 3.7.3).

Tasks

1. (4 points) You should only need to modify `linearclassifier.py`, which has three unfinished functions. The first function you should finish is `linear_predict`, which takes a set of data and a model object as input and outputs the predicted labels for each example. This linear prediction should be done with matrix operations and should take approximately two to five lines of code. This function will be used as the predictor function for both learning algorithms.

   The `model` is a `dict` that contains only one entry. The key is `'weights'`, and the value is the weight matrix for the multi-class linear classifier.

[1] `https://archive.ics.uci.edu/ml/datasets/Cardiotocography`

2. (4 points) Implement `perceptron_update`, which should run one perceptron learning step. It receives as input **one** example and its label and modifies the `model` object in place. In other words, the function should change the weights of the `model` dictionary to the new weights after applying the perceptron equations. Finally, the function should return whether the perceptron was correct on the given example.

   Your `perceptron_update` function should update the numpy matrix stored in `model['weights']` *in-place*. This in-place update can be done by directly setting the values of certain columns of the matrix. For example, if you want to set the values of the `i`th column of the matrix equal to the values in a vector `new_vector`, the command is

   ```
   model['weights'][:, i] = new_vector
   ```

   Once you complete this task, you should be able to run the perceptron experiments. The notebooks should generate plots of the training and testing accuracy as you run multiple epochs of online learning.

3. (7 points) Implement `log_reg_nll` (short for *logistic regression negative log-likelihood*), which is a function defined inside `log_reg_train` (short for *logistic regression train*). The function `log_reg_nll` returns a length-two tuple containing (1) the value of the negative log-likelihood—i.e., Equation (7)—and (2) the gradient of the negative log-likelihood with respect to the model weights—i.e., Equation (9).

   The first cell of the logistic regression portion in each notebook tests whether your objective function and gradient agree with each other by comparing your provided gradient with a numerically estimated gradient of your objective function. Use that to validate that you implemented the function correctly.

   Make sure to read and understand the other code in `log_reg_train`, which uses the function and gradient you compute and passes it into a general purpose numerical optimization tool to perform gradient-based minimization of the learning objective. You technically won't be graded on your understanding of this part for this homework, but it's an important concept in machine learning that will come up again.

   Once you complete this task, you should be able to run the logistic regression experiment. The notebooks should plot the training and testing accuracy as the regularization parameter λ changes.

Note on preventing numerical overflow In some cases, when you try to calculate $\frac{\exp(s_i)}{\sum_j \exp(s_j)}$, one or more of the exp operations may overflow. Here are two ways to handle this:

(a) You can subtract a constant from each of the scores. Let a be some constant. Then mathematically, $\frac{\exp(s_i)}{\sum_j \exp(s_j)} \equiv \frac{\exp(s_i - a)}{\sum_j \exp(s_j - a)}$. If you use this method, a good choice for a is $a = \max_i s_i$ because then the maximum value of all the $\exp(s_i - a) = \exp(0) = 1$, and you can be sure that none of these overflow. (Some may underflow, but that's okay.)

(b) Another approach is to take advantage of the fact that I gave you the logsumexp function (which basically does something very similar to option (a). You can calculate the whole fraction in log space. Taking the log of the expression, you get

$$\log\left(\frac{\exp(s_i)}{\sum_j \exp(s_j)}\right) = \log \exp(s_i) - \log\left(\sum_j \exp(s_j)\right) = s_i - \text{logsumexp}(\boldsymbol{s}),$$

where the last expression is the logsumexp operation run on the full vector $\boldsymbol{s} = [s_1, \ldots, s_C]$.

Overflow should only really happen if the weights or features are too large in magnitude. Sometimes the optimization may explore weight values that are too large, so having these tricks to prevent overflow can be helpful, but other times the optimization never tries these large values, so adding them just overcomplicates your code.