COMS 4771 Machine Learning *Fall 2023*

Homework 5

Due FRIDAY December 1, 2023 at 11:59 PM

Instructions for preparing and submitting your homework write-up are available here:
`https://www.cs.columbia.edu/~djhsu/coms4771-f23/policies-homework.html`

Please submit your Python code for Problems 1, 5, 6 and 8 on Gradescope.

Multi-class logistic regression and gradient descent

Multi-class logistic regression is a statistical model for classification data where the number of classes is possibly more than two. Every distribution in the model is specified by an entire matrix of parameters

$$W = \begin{bmatrix} \uparrow & & \uparrow \\ w^{(0)} & \cdots & w^{(K-1)} \\ \downarrow & & \downarrow \end{bmatrix} \in \mathbb{R}^{d \times K},$$

where d is the dimension of the input space and K is the number of classes. The k-th column of W is a weight vector $w^{(k)} \in \mathbb{R}^d$ corresponding to class $k \in \{0, \dots, K-1\}$. Under the distribution with parameter matrix W, for any $x \in \mathbb{R}^d$, the conditional probability that the label Y is equal to k given the feature vector $X = x$ is

$$\Pr_W(Y = k \mid X = x) = \frac{\exp(\langle w^{(k)}, x \rangle)}{\sum_{j=0}^{K-1} \exp(\langle w^{(j)}, x \rangle)}.$$

(Please check for yourself that this is a valid conditional distribution for Y given $X = x$.)
Notice that under the distribution with parameter matrix W, the classifier with smallest error rate is given by

$$f_W(x) = \operatorname*{arg\,max}_{k \in \{0,\dots,K-1\}} \Pr_W(Y = k \mid X = x) = \operatorname*{arg\,max}_{k \in \{0,\dots,K-1\}} \langle w^{(k)}, x \rangle. \tag{1}$$

Here is a common approach to learning a multi-class classifier using this model. Treat a training dataset $(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})$ from $\mathbb{R}^d \times \{0, \dots, K-1\}$ as an i.i.d. sample from some distribution in this model, and attempt to estimate the parameter matrix W from the training data by maximizing the log-likelihood objective:

$$\ln L(W) = \sum_{i=1}^{n} \ln\big(\Pr_W(Y = y^{(i)} \mid X = x^{(i)})\big) = \sum_{i=1}^{n} \ln\left(\frac{\exp(\langle w^{(y^{(i)})}, x^{(i)} \rangle)}{\sum_{j=0}^{K-1} \exp(\langle w^{(j)}, x^{(i)} \rangle)}\right).$$

Maximizing the log-likelihood objective is equivalent to minimizing the negative log-likelihood objective, which can be attempted using gradient descent.[1] With a high-likelihood parameter matrix W in hand, the multi-class classifier f_W as defined in (1) is returned.

[1] The negative log-likelihood of W can also be interpreted as a sum of log losses on the n training examples incurred by the function p_W defined by $p_W(x) = (\Pr_W(Y = 1 \mid X = x), \dots, \Pr_W(Y = K-1 \mid X = x))$.

Problem 1. Write a Python function `logreg_nll_gd` that takes as input:

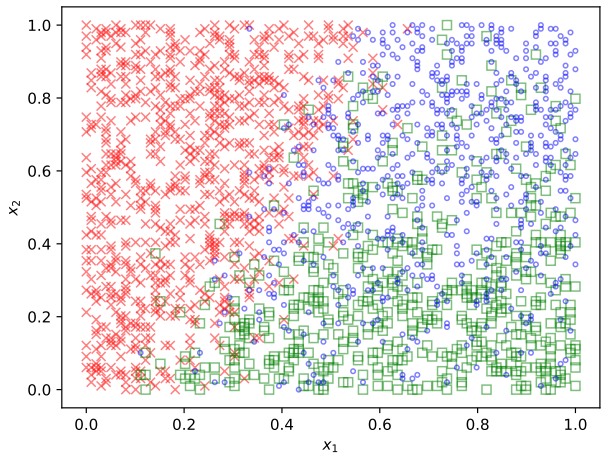
- number of classes K, (constant) step size $\eta > 0$, positive integer T;
- a Numpy array of n feature vectors from $\mathbb{R}^d$; and
- a Numpy array of n labels from $\{0, \ldots, K-1\}$;

and returns the parameter matrix obtained by running T steps of gradient descent for the negative log-likelihood function starting from the $d \times K$ all-zeros parameter matrix. You can use the Numpy functions `numpy.exp`, `numpy.log`, `numpy.sum`, `numpy.mean`, `numpy.matmul`, `numpy.max`, `numpy.argmax`, `numpy.dot`, `numpy.arange`, `numpy.zeros`, and the Scipy function `scipy.special.softmax`. (You do not necessarily need all of these.)

The file `hw5p1.pkl` contains data from $\mathbb{R}^3 \times \{0, 1, 2\}$ for a 3-class classification problem.

```
import pickle
hw5p1 = pickle.load(open('hw5p1.pkl', 'rb'))
```

Training data (features and labels) are in `hw5p1['data']` and `hw5p1['labels']`, and test data are in `hw5p1['testdata']` and `hw5p1['testlabels']`. The feature vectors are actually 2-dimensional, but we have appended an "always-1" feature to each feature vector. Here is a plot of the training data (the shape/color of each point indicates its label):



Use `logreg_nll_gd` to learn a multi-class classifier (of the form in (1)) for this problem. In practice, you will need to experiment with η and T, but for this assignment, just use $\eta = 2/n$ (for all iterations) and $T = 10000$, where n is the number of training examples.

(a) Report the training error rate and test error rate of the classifier corresponding to the final parameter matrix.

(b) Modify `logreg_nll_gd` so that, at the start of every twentieth iteration of gradient descent (i.e., at the start of iterations 20, 40, ..., 10000), it records (i) the negative log-likelihood of the parameter matrix, and (ii) the training error rate of the corresponding classifier. Plot each of these quantities (on separate plots) as a function of the number of steps of gradient descent. Include these two plots with your write-up. What do you notice about the difference between these plots?

Convexity and gradients

Problem 2. Consider the function $J\colon \mathbb{R}^2 \to \mathbb{R}$ given by

$$J(w) = \frac{1}{2}w_1^2 + \ln(1 + \exp(2w_1 + w_2)) - (2w_1 + w_2) + \ln(1 + \exp(-2w_1 + w_2)).$$

(a) Is the function J a convex function? Please justify your answer.

(b) The function J is differentiable. Write a formula for the gradient of J.

Problem 3. Consider the function $J\colon \mathbb{R}^2 \to \mathbb{R}$ given by

$$J(w) = (\text{logistic}(2w_1 + w_2) - 1)^2 + (\text{logistic}(-2w_1 + w_2) - 0)^2.$$

(Recall: $\text{logistic}(t) = 1/(1 + \exp(-t))$.)

(a) Is the function J a convex function? Please justify your answer.

(b) The function J is differentiable. Write a formula for the gradient of J.

Problem 4. Consider the function $J\colon \mathbb{R}^2 \to \mathbb{R}$ given by

$$J(w) = \max\{0, 1 - (2w_1 + w_2)\}.$$

(a) Is the function J a convex function? Please justify your answer.

It turns out J is not differentiable, which means there are some $u \in \mathbb{R}^2$ at which $J(w)$ is not well-approximated by any affine function $A(w)$ for w close to u. Nevertheless, J does have a good affine *minorant* at every $u \in \mathbb{R}^2$. By this, we mean that for every $u \in \mathbb{R}^2$, there exists an affine function $A\colon \mathbb{R}^2 \to \mathbb{R}$ such that

$$J(u) = A(u) \quad \text{and} \quad J(w) \geq A(w) \quad \text{for all } w \in \mathbb{R}^2. \tag{2}$$

(b) Write an affine function $A\colon \mathbb{R}^2 \to \mathbb{R}$ that satisfies (2) at $u = (0, 1/2)$.

(c) Write an affine function $A\colon \mathbb{R}^2 \to \mathbb{R}$ that satisfies (2) at $u = (1/4, 1/2)$.

Postscript: The slope vector of an affine minorant of J at a point u (satisfying (2)) is called a *subgradient* of J at u. There is a variant of gradient descent for non-differentiable objective functions (called *subgradient descent*) that only requires a subroutine that computes a subgradient of the objective at any given point. The Online Perceptron algorithm can be viewed as variant of subgradient descent (analogous to the way SGD is a variant of gradient descent) for the objective function

$$J_{\text{Perceptron}}(w) = \sum_{i=1}^{n} \max\{0, -y^{(i)}\langle w, x^{(i)}\rangle\},$$

where $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$ are the training examples from $\mathbb{R}^d \times \{-1, 1\}$.

Automatic differentiation

Please read the PyTorch "Learn the basics" tutorial, specifically the part on "Tensors" (i.e., multi-dimensional arrays) and the part on "Autograd". (The other parts are not essential for this assignment, but recommended anyway!)

Problem 5. Use the automatic differentiation facilities in PyTorch (called "autograd") to re-implement `logreg_nll_gd` from Problem 1. You can use the following template, which gives most of the code that is not specific to the objective function:

```
import torch

def logreg_nll_gd_ad(K, eta, T, X, y):
  X = torch.tensor(X, dtype=torch.double)
  y = torch.tensor(y, dtype=torch.int)
  W = torch.zeros(X.shape[1], K, dtype=torch.double)
  W.requires_grad = True
  for _ in range(T):
      J = # FILL IN

      J.backward() # this computes all partial derivatives of variables involved
      ↪  (with "requires_grad = True") in computation of J, and stores the
      ↪  results in an appropriate place (see below)
      with torch.no_grad():
          W -= eta * W.grad # W.grad is where the invokation of J.backward() has
          ↪  put the gradient with respect to W
          W.grad.zero_() # this zeros out the space used for storing the
          ↪  gradient with respect to W
  return W.detach()
```

You just need to fill-in the part that computes the objective value of the current `W`. You can use the PyTorch functions `torch.exp`, `torch.log`, `torch.sum`, `torch.mean`, `torch.matmul`, `torch.max`, `torch.argmax`, `torch.dot`, `torch.arange`, `torch.zeros`, and `torch.softmax`. (You do not necessarily need all of these.)[a] Use `logreg_nll_gd_ad` with the training data from `hw5p1.pkl` (again with $\eta = 2/n$ and $T = 10000$), and compare the parameter matrix W you obtain using `logreg_nll_gd_ad` to what you obtain using `logreg_nll_gd` from Problem 1 by computing their Frobenius norm difference:

$$\|A - B\|_{\mathrm{F}} = \sqrt{\sum_{i=1}^{d}\sum_{j=1}^{K}(A_{ij} - B_{ij})^2}.$$

Report this Frobenius norm difference in your write-up; it should be quite small. (You should also check that you obtain the same training/test error rates as you got for Problem 1, but you don't need to report this in your write-up.)

[a] There are some other PyTorch functions that could make this problem even easier, but you should solve this problem without them.

Class imbalance

The file `hw5click.pkl` contains data for a binary classification problem related internet advertising.

```
import pickle
click = pickle.load(open('hw5click.pkl', 'rb'))
```

Training data (features and labels) are in `click['data']` and `click['labels']`, and test data are in `click['testdata']` and `click['testlabels']`. Each example (x, y) corresponds to a visit of a particular user to an internet website on which a particular advertisement is displayed. The label $y \in \{0, 1\}$ indicates if the user clicked on the advertisement. The feature vector $x = (x_1, \ldots, x_{54}) \in \mathbb{R}^{54}$ gives information about the user and the advertisement.

Also provided are estimates of the weight vector $w \in \mathbb{R}^{54}$ and intercept $b \in \mathbb{R}$ parameters from a logistic regression model for this data, where under the distribution with parameters (w, b), the conditional probability that the label Y is equal to 1 given the feature vector $X = x$ is

$$\Pr_{w,b}(Y = 1 \mid X = x) = \text{logistic}(\langle w, x \rangle + b).$$

These parameter estimates, w_{mle} and b_{mle}, are given in `click['w_mle']` and `click['b_mle']`, respectively.

Let $J(w, b)$ be the negative log-likelihood of $(w, b) \in \mathbb{R}^d \times \mathbb{R}$ based on the training data, and let $f_{w,b} \colon \mathbb{R}^d \to \{0, 1\}$ be the linear classifier defined by

$$f_{w,b}(x) = \arg\max_{y \in \{0,1\}} \Pr_{w,b}(Y = y \mid X = x) = \mathbb{1}\{x^\mathsf{T} w + b > 0\}.$$

Problem 6.

(a) Compute the gradient of J evaluated at $(w_{\text{mle}}, b_{\text{mle}})$, and report its Euclidean norm $\|\nabla J(w_{\text{mle}}, b_{\text{mle}})\|$. It should be quite close to zero. Since J is a convex function, the parameters $(w_{\text{mle}}, b_{\text{mle}})$ nearly maximize the log-likelihood.

(b) What are the training and test error rates of $f_{w_{\text{mle}}, b_{\text{mle}}}$?

You should find that the training error rate is the same as the fraction of training examples with a label of 1, and you should make an analogous finding regarding the test error rate. Indeed, $f_{w_{\text{mle}}, b_{\text{mle}}}$ predicts 0 on all training and test examples.

The severe class imbalance in this problem is due to the fact that the vast majority of visits to the website do not result in the user clicking on the advertisement.

In many applications, the error rate $\Pr(f(X) \neq Y)$ of a binary classifier $f \colon \mathbb{R}^d \to \{0, 1\}$ may not be the desired performance metric, but rather one may prefer a metric that balances the two possible classes. The *balanced error rate* of a binary classifier f is

$$\frac{1}{2} \Pr(f(X) = 1 \mid Y = 0) + \frac{1}{2} \Pr(f(X) = 0 \mid Y = 1).$$

We analogously define the *balanced training error rate* and *balanced test error rate* as above simply by letting the distribution of (X, Y) be uniform over the training dataset and test dataset, respectively.

Problem 7. What are the balanced training and test error rates of the classifier $f_{w_{\mathrm{mle}},b_{\mathrm{mle}}}$? (This problem doesn't really require any code . . .)

Suppose we have a classifier learning procedure that was designed with (unbalanced) error rate in mind as the primary objective, but now we want a procedure that is adapted for balanced error rate. A natural approach is to "re-balance" the training data—effectively, replicate the examples from the under-represented class until the two classes are equally represented in the training data—and then apply the original classifier learning procedure. If the learning procedure is based on minimizing an objective function that involves a sum of losses over training examples, then the re-balancing can be achieved by changing the sum to an appropriately weighted sum.

Consider the following objective function $J_{\mathrm{bal}} \colon \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}$ based on the training data:

$$J_{\mathrm{bal}}(w,b) = \frac{\lambda}{2}\big(\|w - w_{\mathrm{mle}}\|^2 + (b - b_{\mathrm{mle}})^2\big) - \frac{1}{2n_0}\sum_{(x,y)\in\mathcal{S}_0} \ln \Pr_{w,b}(Y = 0 \mid X = x) - \frac{1}{2n_1}\sum_{(x,y)\in\mathcal{S}_1} \ln \Pr_{w,b}(Y = 1 \mid X = x).$$

Here,

- $\mathcal{S}_0$ is the subset of training examples with label 0, and $n_0 = |\mathcal{S}_0|$;
- $\mathcal{S}_1$ is the subset of training examples with label 1, and $n_1 = |\mathcal{S}_1|$;
- and $\lambda \geq 0$ is a hyperparameter.

We can interpret this objective as a trade-off between the following concerns:

- we want parameters (w, b) that are not too far from $(w_{\mathrm{mle}}, b_{\mathrm{mle}})$;
- and at the same time, we also want parameters (w, b) with small average logarithmic loss on the re-balanced training data.

  (Here, we regard the logarithmic loss as a surrogate for the zero-one loss.)

Problem 8. Implement gradient descent for minimizing J_{bal}, and apply it to obtain new parameter estimates $(w_{\mathrm{bal}}, b_{\mathrm{bal}})$. You should initialize gradient descent with $(w, b) = (w_{\mathrm{mle}}, b_{\mathrm{mle}})$, and use (constant) step size $\eta = 0.001$, number of iterations $T = 50000$, and $\lambda = 0.01$. What are the balanced training and test error rates of the classifier $f_{w_{\mathrm{bal}},b_{\mathrm{bal}}}$?