CS 273P: Machine Learning and Data Mining

Homework 3

Due date: **Wednesday, February 15, 2023**

Instructor: Xiaohui Xie

The submission for this homework, as for others so far, should be **one stand-alone PDF file** containing all of the relevant code, figures, and any text explaining your results. Although you will be primarily filling missing sections in a Python file, include all the relevant sections you have written as answers to the appropriate question.

Points: This homework adds up to a total of **110 points**, as follows:

Problem 1: Logistic Regression	105 points
Statement of Collaboration	5 points

Be sure to re-download the data and replace the Python code; even though it is mostly the same, we may make changes to the data or fix bugs for each homework.

Problem 1: Logistic Regression (105 points)

In this problem, we'll build a logistic regression classifier and train it on separable and non-separable data. Since it will be specialized to binary classification, we've named the class `logisticClassify2`.

We'll start by building two binary classification problems, one separable and the other not:

```
iris = np.genfromtxt("data/iris.txt",delimiter=None)
X, Y = iris[:,0:2], iris[:,-1]   # get first two features & target
X,Y  = ml.shuffleData(X,Y)       # reorder randomly (important later)
X,_  = rescale(X)                # works much better on rescaled data

XA, YA = X[Y<2,:], Y[Y<2]        # get class 0 vs 1
XB, YB = X[Y>0,:], Y[Y>0]        # get class 1 vs 2
```

For this problem, we are focused on the learning algorithm, rather than performance — so, we will not bother creating training and validation splits; just use all your data for training.

Note: The code uses numpy's `permute` to iterate over data randomly; should avoid issues due to the default order of the data (by class). Similarly, rescaling and centering the data may help speed up convergence as well.

1. Show the two classes in a scatter plot (one for each data set) and verify that one data set is linearly separable while the other is not. `(5 points)`

2. Write (fill in) the function `plotBoundary` in `logisticClassify2.py` to compute the points on the decision boundary. In particular, you only need to make sure `x2b` is set correctly using `self.theta`. This will plot the data & boundary quickly, which is useful for visualizing the model during training. To demo your function plot the decision boundary corresponding to the classifier

$$sign(\ .5 - .25x_1 + 1x_2\ )$$

along with the A data, and again with the B data; these fixed parameters will look like an OK classifier on one data set, but a poor classifier on the other.

You can create a "blank" learner and set the weights by:

```
import mltools as ml
from logisticClassify2 import *

learner = logisticClassify2();          # create "blank" learner
learner.classes = np.unique(YA)         # define class labels using YA or YB
wts = np.array([theta0,theta1,theta2]); # TODO: fill in values
learner.theta = wts;                    # set the learner's parameters
```

Include the lines of code you added to the function, and the two generated plots. `(10 points)`

3. Complete the `logisticClassify2.predict` function to make predictions for your classifier. Verify that your function works by computing & reporting the error rate of the classifier in the previous part on both data sets A and B. (The error rate on one should be ≈ 0.0505, and higher on the other.) Note that, in the code, the two classes are stored in the variable `self.classes`, with the first entry being the "negative" class (or class 0), and the second entry being the "positive" class, so you want to have different learner objects for each dataset, and you use `learner.err` directly.

   Include the function definition and the two computed errors. `(10 points)`

4. Verify that your predict code matches your boundary plot by using `plotClassify2D` with your manually constructed learner on the two data sets. This will call `predict` on a dense grid of points, and you should find that the resulting decision boundary matches the one you computed analytically. `(5 points)`

5. In the provided code, we first transform the classes in the data Y into YY, with canonical labels for the two classes: "class 0" (negative) and "class 1" (positive). In our notation, let $r^{(j)} = x^{(j)} \cdot \theta^T$ be the linear response of the classifier, and σ is the standard logistic function

$$\sigma(r) = \big(1 + \exp(-r)\big)^{-1}.$$

   The logistic negative log likelihood loss for a single data point j is then

$$J_j(\theta) = -y^{(j)} \log \sigma(x^{(j)}\theta^T) \; - \; (1 - y^{(j)}) \log(1 - \sigma(x^{(j)}\theta^T))$$

   where $y^{(j)}$ is either 0 or 1. Derive the gradient of the negative log likelihood J_j for logistic regression, and give it in your report. (You will need this in your gradient descent code for the next part.)

   Provide the gradient equations for $\frac{\partial}{\partial\theta_0}J_j$, $\frac{\partial}{\partial\theta_1}J_j$, and $\frac{\partial}{\partial\theta_2}J_j$. `(10 points)`

6. Complete `train` function to perform stochastic gradient descent on the logistic loss function. This will require that you fill in:
   (1) computing the surrogate loss function at each epoch ($J = \frac{1}{m}\sum J_j$, from the previous part);
   (2) computing the response ($r^{(j)}$ and gradient associated with each data point $x^{(j)}, y^{(j)}$;
   (3) a stopping criterion consisting of two conditions (stop when either you have reached `stopEpochs` epochs or J has not changed by more than `stopTol` since the last epoch).

   Include the complete implementation of `train`. `(25 points)`

7. Run `train` for your logistic regression classifier on both data sets (A and B). Describe your parameter choices for each dataset (stepsize, etc.) and include plots showing the convergence of the surrogate loss and error rate (e.g., the loss values as a function of epoch during gradient descent), and the final converged classifier with the data (the included `train` function does that for you already). `(10 points)`

   Note: Debugging machine learning algorithms can be quite challenging, since the results of the algorithm are highly data-dependent, and often somewhat randomized (initialization, etc.). I suggest starting with an extremely small step size and verifying both that the learner's prediction evolves slowly in the correct direction, and that the objective function J decreases monotonically. If that works, go to larger step sizes to observe the behavior. I often manually step through the code — for example by pausing after each parameter update using `raw_input()` (Python 2.7) or `input()` (Python 3) – so that I can examine its behavior. You can also (of course) use a more sophisticated debugger.

   Note on plotting: The code generates plots as the algorithm runs, so you can see its behavior over time; this is done with `pyplot.draw()`. Run your code either interactively or as a script to see these display over time; refer to discussion notebooks on how to plot the loss over time in Jupyter.

8. Add an L1 regularization term ($+\alpha \sum_i |\theta_i|$) to your surrogate loss function, and update the gradient and your code to reflect this addition. Try re-running your learner with some regularization (e.g. $\alpha = 2$) and see how different the resulting parameters are. Find a value of α that gives noticeably different results & explain them. `(10 points)`

9. Add an L2 regularization term ($+\alpha \sum_i \theta_i^2$) to your surrogate loss function, and update the gradient and your code to reflect this addition. Try re-running your learner with some regularization (e.g. $\alpha = 2$) and see how different the resulting parameters are. Find a value of α that gives noticeably different results & explain them. `(10 points)`

10. What are the major differences between L1 and L2 regularization? Which regularization method do you think fits this problem in a better way? `(10 points)`

Statement of Collaboration (5 points)

It is **mandatory** to include a `Statement of Collaboration` in each submission, with respect to the guidelines below. Include the names of everyone involved in the discussions (especially in-person ones), and what was discussed.

All students are required to follow the academic honesty guidelines posted on the course website. For programming assignments, in particular, I encourage the students to organize (perhaps using ed) to discuss the task descriptions, requirements, bugs in my code, and the relevant technical content `before` they start working on it. However, you should not discuss the specific solutions, and, as a guiding principle, you are not allowed to take anything written or drawn away from these discussions (i.e. no photographs of the blackboard, written notes, referring to ed, etc.). Especially `after` you have started working on the assignment, try to restrict the discussion to ed as much as possible, so that there is no doubt as to the extent of your collaboration.