CS 178: Machine Learning: Winter 2019

Homework 3

Due Date: **Thursday, February 28th, 11:59pm on EEE/Canvas**

The submission for this homework should be `a PDF report and code`. Code submission is accepted in `.py` or `.ipynb` or `.zip` format. Use the .zip format if you are submitting multiple code files, but `do not` zip the pdf report. Your submission must include exactly two files: one (`.py` OR `.ipynb` OR .zip) file AND one `.pdf` file. For many questions you will be completing missing sections in a Python file that we provide. Be sure to include copies of any code you write as answers to the appropriate question, so that it may be graded.

Your solution may be submitted up to 4 days late (by 11:59pm, 96 hours after the deadline); Late homework will be penalized 10% per day late (or part thereof), up to 4 days late (96 hours after the deadline); after this point, solutions will be distributed and handins will no longer be accepted.

We will use predefined functions "shuffleData", "rescale", and "toIndex". For your convenience, there is a version of the code in a more convenient form for Jupyter notebooks available as `HW3_Template.ipynb`. There is also a version in PDF format `HW3_Template.pdf`. If you prefer to run python as a script, you can edit the template file directly and then import it (as in the code snippets here), or copy it into your notebook and update it directly. Either way, be sure to include the listings of the updated functions in your writeup.

Problem 1: Logistic Regression (75 points)

In this problem, we'll build a logistic regression classifier and train it on separable and non-separable data. It will be specialized to binary classification. We start by creating two binary classification datasets, 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  = shuffleData(X,Y)      # order randomly rather than by class label
X,_  = rescale(X)               # rescale to improve numerical stability, speed convergence

XA, YA = X[Y<2,:], Y[Y<2]       # Dataset A: class 0 vs class 1
XB, YB = X[Y>0,:], Y[Y>0]       # Dataset B: class 1 vs class 2
YA[YA==0] = -1  # turn into -1, 1 binary targets for further boundary
YB[YB==2] = -1  # turn into -1, 1 binary targets for further boundary
```

For this problem, we focus on the properties of the logistic regression learning algorithm, rather than classification performance. Thus we will not create a separate validation dataset, and simply use all available data for training.

1. For each of the two datasets, create a separate scatter plot in which the training data from the two classes is plotted in different colors. Which of the two datasets is linearly separable? *(5 points)*

2. Write the function `plotBoundary` to compute the points on the decision boundary. In particular, you only need to make sure `yy` is set correctly using `slop1, slop2, bias`. Plot the data and decision boundary corresponding to the logistic regression classifier
(`sklearn.linear_model.LogisticRegression`) on dataset A, and again on dataset B.

```
def plotBoundary(slop1, slop2, bias):
    raise NotImplementedError
    ## TODO: find points on decision boundary defined by
    ##             theta0 + theta1 X1 + theta2 X2 == 0
    xx = np.linspace(-1.2, 1.5, 200)
    yy = ...
    plt.plot(xx, yy, "c", label="boundary")
    plt.legend()
```

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

3. Complete `prediction` function to make predictions for the classifier. Verify that your function works by computing & reporting the error rate for the logistic regression classifier on both datasets A and B. Your solution pdf should include the `prediction` function implementation and the computed error rates. *(10 points)*

```
def prediction(slop1, slop2, bias, X, Y):
    """ Return the error rate and the prediction confident of each data point in X"""
    raise NotImplementedError
    ## TODO: compute linear response
    ##             r[i] = theta0 + theta1 X[i,1] + theta2 X[i,2]  + ... for each i
    ## TODO: if z[i] >= 0, predict class 1:  Yhat[i] = 1.0
    ##       else predict class -1:  Yhat[i] = -1.0
    ##

    # predY is float value corresponding to the confident level
    return err, predY
```

4. Verify that your `prediction` and `plotBoundary` implementations are consistent by using `plotClassify2D` with the logistic regression learner on each dataset. This will call `prediction` on a dense grid of points, and you should find that the resulting decision boundary matches the one you plotted previously. *(5 points)*

```
def plotClassify2D(slop1, slop2, bias):
    x0 = np.linspace(-2, 2, 100)
    x1 = np.linspace(-2, 2, 100)
    gridData = np.array(np.meshgrid(x0, x1)).T.reshape(-1, 2)
    _, predGridData = prediction(slop1, slop2, bias, gridData, \
                                 np.ones((gridData.shape[0], 1)))
    predGridData = np.absolute(predGridData.reshape((-1,)))
    plt.plot(gridData[predGridData<0.01, 0], gridData[predGridData<0.01, 1], "r", \
             label="plotClassify2D_boundary")
```

5. In the provided training 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). Let $r^{(j)} = x^{(j)} \cdot \theta = \sum_i x_i^{(j)} \theta_i$ denote the linear response of the classifier, and $\sigma(r)$ equal 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

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

where $y^{(j)}$ is 0 or 1.

(a) Make a proof that the logistic function has the following derivative:*(5 points)*

$$\sigma'(r) = \sigma(r)(1 - \sigma(r))$$

(b) Derive the gradient of the negative log likelihood $J_j(\theta)$ for logistic regression (this will be needed to implement stochastic gradient descent in the next part). Include the gradient equation in your solutions.

(10 points)

6. Complete the `train` function to perform stochastic gradient descent on the logistic regression loss function. This will require that you fill in:
(a) computing the response $r^{(j)}$ and gradient $\nabla J_j(\theta)$ associated with each data point $x^{(j)}, y^{(j)}$;
(b) computing the overall loss function, $J = \frac{1}{m} \sum_j J_j$, after each pass through the full dataset (or epoch);
(c) a stopping criterion that checks two conditions: stop when either you have reached `stopEpochs` epochs, or J has changed by less than `stopTol` since the last epoch.

Include the complete implementation of `train` in your solutions. *(20 points)*

7. Run the logistic regression `train` algorithm on both datasets. Describe the parameter choices (step sizes and stopping criteria) you use for each dataset. Include two plots: one shows the convergence of the surrogate loss and error rate as a function of the number of training epochs; one shows the classification boundary after the final training iteration with the dataset. *(10 points)*

```
theta, J01, Jnll = train(...)
print("Training error rate: ", prediction(theta[1], theta[2], theta[0], XA,YA)[0])
plt.plot(J01, "r", label="Error rate")
plt.plot(Jnll, "b", label="NLL")
plt.legend()
...
```

Debugging hints: Debugging machine learning algorithms can be quite challenging, since the results of the algorithm are highly data-dependent, and often somewhat randomized (from the initialization, as well as the order points are visited by stochastic gradient descent). We suggest starting with a small step size and verifying both that the learner's prediction evolves slowly in the correct direction, and that the objective function J decreases. If that works, explore the convergence of the algorithm with larger step sizes. It is often useful to manually step through the code, for example by pausing after each parameter update using `raw_input()` (Python 2.7) or `input()` (Python 3). Of course, you may also use a more sophisticated debugger.

Problem 2: Maximum Margin Classifiers (20 points)

In this question, we examine the properties of max-margin separating hyperplanes on toy, two-dimensional data. (Note: this problem is intended to be done by hand, and should require very little numerical calculation.)

1. First consider a dataset with $m = 4$ points. There are 2 examples of class $+1$, located at $x = (-1,-1)$ and $x = (+1,+1)$. There are 2 examples of class -1, located at $x = (-1,+1)$ and $x = (+1,-1)$.

   Define the non-linear feature mapping $\phi(x) = [x_1, x_1 x_2]$. Plot the four input points in this space, and the maximum margin separating hyperplane. What max-margin weight vector w would produce predictions $w \cdot \phi(x)$ that, when thresholded, perfectly classify the training data? What is the corresponding margin? *(5 points)*

2. Plot the max-margin separator from part 1 as a curve in the original input space. *(5 points)*

3. Now consider a different dataset with $m = 6$ points. There are 3 examples of class $+1$, located at $x = (1,1)$, $x = (2,2)$, and $x = (2,0)$. There are 3 examples of class -1, located at $x = (0,0)$, $x = (1,0)$, and $x = (0,1)$.

   Define the feature mapping $\phi(x) = [1, x_1, x_2]$, which adds a bias feature to the raw inputs. Plot the data and a maximum margin separating hyperplane in the original input space. What max-margin weight vector w would produce predictions $w \cdot \phi(x)$ that, when thresholded, perfectly classify the training data? What is the corresponding margin? *(5 points)*

4. For part 3, which data points are support vectors? If you remove one of these support vectors from the training data does the size of the optimal margin decrease, stay the same, or increase? Justify your answer for each of the support vectors. *(5 points)*

Problem 3: Statement of Collaboration (5 points)

It is **mandatory** to include a *Statement of Collaboration* in each submission, that follows 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 students to organize (perhaps using Piazza) to discuss the task descriptions, requirements, possible bugs in the support 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 (no photographs of the blackboard, written notes, referring to Piazza, etc.). Especially *after* you have started working on the assignment, try to restrict the discussion to Piazza as much as possible, so that there is no doubt as to the extent of your collaboration.