CS452, Spring 2018, Problem Set # 2
Ashesi University

February 13, 2018

Due: March 1st 2018, 11:59pm

This problem set requires you to provide written answers to a few technical questions. In addition, you will need to implement some learning algorithms in Python and apply them to the enclosed data sets. The questions requiring a written answer are denoted with **W**, while the Python programming questions are marked with **P**.

Both the written answers as well as the code must be submitted electronically via Courseware.

For the written answers, you must provide a single PDF containing all your answers. Upload the PDF files trough the appropriate link in Courseware. The PDF file can be generated using editing software (e.g., Latex) or can be a scanned copy of your hand-written answers. If you opt for scanned copies, please make sure to use a scanner, *not* your phone camera. Familiarize yourself with the process of scanning documents a few days before the due date. Late submissions due to scanning problems will be treated and penalized as any other form of late submission, as per the rules stated on the Syllabus. Also, make sure your scanned submission is readable. If we cannot read it clearly, you will receive 0 points for that problem.

Your Python code must also be submitted via Courseware. We have provided Python function files with an empty body for all the functions that you need to implement. Each file specifies in detail the input and the output arguments of the function. A few of the functions include some commands to help you get started with the implementation. The function stubs include also *import* commands for all the libraries that are needed for your implementation (e.g., numpy, math). Do not add any other import command, i.e., do not use any other libraries except those already specified in the function stubs. If in doubt, please ask. We have also provided scripts that check the sizes of the inputs and outputs of your functions. Do not modify these scripts. In order to allow us to test your code, it is imperative that you do not rename the files or modify the syntax of the functions. Make sure to include all the files necessary to run your code. **If we are unable to run one of your functions, you will receive 0 points for that question.** Please place all your files (data, .png files automatically generated by the provided scripts, and function files) in a single folder (**without subfolders**). Zip up the entire folder, and upload the compressed file to Courseware. Note that we provided **two** separate links for your homework submission: one for your written answers, one for your code.

1. **[30 points]** For this problem you need to implement locally weighted linear regression (LWLR) and to apply it to the MPG regression problem already introduced in HW1. LWLR predicts the output of test example x as $f_\theta(x) = \theta(x)^\top b^l(x)$, where $\theta(x)$ is given by:

$$\theta(x) = \arg\min_\theta E^{LWLR}(\theta) \tag{1}$$

with $E^{LWLR}(\theta) = \frac{1}{2}\sum_{i=1}^m w^i(x)\left[y^{(i)} - \theta^\top b^l(x^{(i)})\right]^2$ and $w^i(x) = \exp\left(-\frac{||x^{(i)}-x||^2}{2\tau^2}\right)$.

(a) [**W**, 8 points] Show that the locally weighted linear regression error $E^{LWLR}(\theta)$ can be written in matrix notation as

$$E^{LWLR}(\theta) = (B\theta - \mathbf{y})^\top W(B\theta - \mathbf{y}) \tag{2}$$

where B and $\mathbf{y}$ are defined as in the slides used in class, and W is a matrix whose entries are written as functions of the weights $w^i(x)$. Specify W in full detail and remember to include the scaling factor 1/2.

(b) [**P**, 12 points] It is easy to show (you may want to do it as an exercise) that the minimizer $\theta(x)$ of Eq. 2 can be computed via the following closed-form solution:

$$\theta(x) = (B^\top W B)^{-1} B^\top W \mathbf{y} \tag{3}$$

Use this closed-form solution to implement LWLR. Your task is to fill the body of the following functions that we have provided to you:

- [4 points] `q1_W.py` constructs the matrix W as specified in your answer to question 1(a). In order to avoid numerical problems, before computing W, scale the weights $w^i(x)$ so that they sum up to 1, i.e., normalize them so that $\sum_{i=1}^m w^i(x) = 1$.
- [4 points] `q1_train.py` trains the model for a given test example x via the closed-form solution of Eq. 3 (use the function `'numpy.linalg.solve'` to compute the solution). This function should call `q1_W.py` from the previous point to construct W. Note that the matrix B can be computed using the function `q1_features.py` (which we have provided to you and was already used in HW1), with argument `mode` set to `'linear'`. The function `q1_features.py` takes care of adding the constant feature 1 to each input example.
- [2 points] `q1_predict.py` predicts the output value for an input test example x. This function should invoke `q1_train.py` to learn the parameter $\theta(x)$ specific to the input test example.
- [2 points] `q1_test_error.py` performs full evaluation of LWLR on the entire test set and returns the test Mean Squared Error. For this purpose, you can use the provided function `q1_mse.py` which we already introduced in HW1.

After you are done with all these implementations, you can run the enclosed script `q1b.py`, which uses your functions to plot the test set error of LWLR for $\tau \in \{10^2, 10^3, 10^5, 10^6\}$.

(c) [**P**, 4 points] Now implement `q1_cross_validation_error.py` to perform N-fold cross-validation given a training set of examples and a set of values for hyperparameter τ. For each value of τ, this function should return the cross-validation error, i.e., the average of

the mean squared errors over the validation sets. Again, reuse the function `q1_mse.py` to calculate the mean squared error. Note that in each validation run, you will need to evaluate m distinct functions $f_{\theta(\hat{x}^{(i)})}$, as the function itself varies with the validation input $\hat{x}^{(i)}$. Run the script `q1c.py` to plot the cross validation error for $\tau \in \{10^2, 10^3, 10^5, 10^6\}$.

(d) [**W**, 2 points] Using the plot of the cross validation error, please identify the values of τ (if any) causing underfitting.

(e) [**W**, 2 points] Which values of τ yield overfitting?

(f) [**W**, 2 points] Now look at the plot of the test error. Is the performance similar to the one measured via cross validation (cross validation error)? Why or why not?

2. **[35 points]** For this problem you will need to implement logistic regression in Python. You will evaluate your algorithm on the Parkinsons dataset contained in *parkinsons.mat*, which we obtained from the UCI Machine Learning Repository (the data is formatted as in the previous homework). Look at *parkinsons.names* for a description of the features. The objective is to recognize healthy people from those with Parkinson's disease using a series of biomedical voice measurements.

(a) **[P, 23 points]** Implement a gradient ascent algorithm maximizing the logistic regression log-likelihood function (you are not allowed to use Python built-in functionalities for gradient ascent optimization). The method requires selecting a value for the learning rate α: for now, use a fixed small value (e.g., $\alpha = 10^{-6}$). Your task is to fill the body of the following functions that we have provided to you:

- [**P**, 3 points] `q2_initialize.py` initializes the weights for maximum log likelihood training. The function provides two options: *random* and *heuristic* initialization. We have implemented for you the *random* option. Your task is to code the *heuristic* option that we have discussed in class, i.e., the one involving the solution of the least squares system originating by setting $g(\theta^\top x^{(i)}) = w^{(i)}$ for $i = 1, ..., m$, where g is the logistic function, and $w^{(i)} = 0.95$ if $y^{(i)} = 1, w^{(i)} = 0.05$ otherwise. Use `numpy.linalg.lstsq` to solve the least squares system.
- [**P**, 2 point] `q2_predict.py` computes the label predictions and the class posterior probabilities for the input examples in matrix X using the parameter vector θ.
- [**P**, 6 points] `q2_loglik.py` computes the log likelihood of the training data given the model θ. Add the Python constant `numpy.spacing(1)` (i.e., a tiny positive number) to the probabilities before taking the log in order to avoid numerical issues. [Hint: you can/should call `q2_predict.py` inside this function].
- [**P**, 4 points] `q2_gradient.py` computes the gradient of the log likelihood at the current θ. [Hint: use `q2_predict.py` inside this function].
- [**P**, 6 points] `q2_train.py` trains logistic regression with gradient ascent using a fixed step size α. [Hint: you should call `q2_loglik.py` and `q2_gradient.py` inside this function].
- [**P**, 2 point] `q2_error.py` calculate the misclassification rate by comparing the predicted labels to the true labels Y. The misclassification rate is given by the number of misclassified examples over the total number of examples.

After implementing the above functions, you should be able to run `q2a.py`. This script will train logistic regression via gradient ascent using a small fixed step size $\alpha = 10^{-6}$. It will report both the training and the test error, the number of iterations needed to reach convergence, and will plot a curve of the log likelihood as a function of the number of iterations. [Hint: if you have implemented the functions correctly, the log likelihood curve should be monotonically increasing].

(b) **[P, 8 points]** Implement the function `q2_train_line_search.py` which trains logistic regression using line search to refine the step size α adaptively at each iteration (see lecture slides for the pseudocode). The line search method requires an initial value α_0. As discussed in class this should be chosen fairly large, e.g., $\alpha_0 = 10^{-4}$. Again, you are not allowed to use Python built-in functionalities for gradient ascent optimization. After finishing this function, you can run the `q2b.py` to test your gradient ascent algorithm with line search. The script will report, for both the version using the fixed α (as coded for the previous question) and the one using line search, the following information: the training and the test errors, the number of iterations needed to reach convergence, and the log likelihood as a function of the number of iterations. Based on these results, answer the questions below:

(c) **[W, 2 points]** Compare the training and test errors of the two variants of gradient ascent. Are the errors different in the two cases? Explain why or why not.

(d) **[W, 2 points]** Now compare the two log likelihood curves. Does the method using line search converge faster or slower than the version using a fixed step size? Explain the result.

3. **[W, 15 points]** You are on a game show. The game host asks you to choose one of three doors, #1, #2, or #3. Behind one door there is a car. Behind the other two doors there is a zonk (i.e., a worthless consolation prize). You obviously want to win the car. You choose a door, say #1 (without loss of generality). Then the host (who knows what's behind each door), opens one of the other two doors **not** containing the car, say #3. At this point the host asks you whether you want to stick with your initial choice (i.e., #1), or choose the other unopened door (i.e., #2). What do you decide to do and why?

   You should provide a formal justification of your answer using random variable $C \in \{\#1, \#2, \#3\}$ to indicate the actual position of the car, so that $P(C = \#1)$ denotes the probability that the car is behind door #1. Similarly, use random variable $H \in \{\#1, \#2, \#3\}$ to denote the door opened by the host.

4. **[W, 20 points]** Suppose we want to perform binary classification of real-valued vectors $x \in \mathbb{R}^n$ using a generative classifier.

   (a) [**W**, 8 points] Show that the decision rule to classify a text example x can be written as:

$$y = \begin{cases} 1 & \text{if } \frac{1}{1+\exp(-a(x))} > \frac{1}{2} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

   where $a(x) = \log\left[\frac{p(x|y=1)p(y=1)}{p(x|y=0)p(y=0)}\right]$.

(b) [**W**, 12 points] Assume that we decide to use as generative classifier a Linear Gaussian Discriminant Analysis with maximum likelihood parameters $\{\mu_0, \mu_1, \Sigma\}$ learned from a given training set. Prove that for this choice of classifier the resulting decision boundary is *linear* by showing that $a(x) = \theta^T x + b$. To prove this you must derive the analytical expressions of $\theta \in \mathbb{R}^n$ and $b \in \mathbb{R}$ in terms of the maximum likelihood parameters μ_0, μ_1, Σ.