HOMEWORK 3:
LINEAR REGRESSION AND LOGISTIC REGRESSION

CMU 10601: MACHINE LEARNING (SPRING 2017)
https://piazza.com/cmu/spring2017/10601
OUT: Feb 13, 2017
DUE: Feb 22, 2017 11:59 pm
TAs: Daniel Bird, Sree Harsha, Abhinav Maurya, Ye Yuan

START HERE: Instructions

- **Collaboration Policy**: Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators fully and completely (e.g., "Jane explained to me what is asked in Question 3.4"). Second, write your solution *independently*: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only. See the collaboration policy on the website for more information: http://www.cs.cmu.edu/~mgormley/courses/10601-s17/about.html
- **Late Submission Policy:** See the late submission policy here: http://www.cs.cmu.edu/~mgormley/courses/10601-s17/about.html
- **Submitting your work:** You will use Gradescope to submit answers to all theory questions, and Autolab to submit only your code required in sections 1.2 and 2.2.
  - **Gradescope:** For written problems such as derivations, proofs, or plots we will be using Gradescope. You can access the site here: https://gradescope.com/. Each derivation/proof should be completed on a separate page. Submissions can be handwritten, but should be labeled and clearly legible. If your writing is not legible, you will not be awarded marks. Alternatively, submissions can be written in LaTeX. Upon submission, label each question using the template provided. Regrade requests can be made, however this gives the TA to regrade your entire paper, meaning if additional mistakes are found then points will be deducted.
  - **Autolab:** You can access the 10601 course on autolab by going to https://autolab.andrew.cmu.edu/ All programming assignments will be graded automatically on Autolab using Octave 3.8.2 and Python 2.7. You may develop your code in your favorite IDE, but please make sure that it runs as expected on Octave 3.8.2 or Python 2.7 before submitting you should use the same language for both linear regression and logistic regression implementations. The code which you write will be executed remotely against a suite of tests, and the results are used to automatically assign you a grade. To make sure your code executes correctly on our servers, you should avoid using libraries which are not present in the basic Octave install. For Python users, you are encouraged to use the `numpy` package. The deadline displayed on Autolab may not correspond to the actual deadline for this homework, since we are allowing late submissions (as discussed in the late submission policy on the course site) Any attempt to "hack" Autolab or any other kind of code cheating will be dealt with according to university policy on student cheating.

In this assignment, capital bold notation such as $\mathbf{X}$ refers to the feature matrix whose rows correspond to datapoints and columns to features. Small letter, bold notation refers to a vector e.g. $\mathbf{x}_i$ denotes the i^{th} datapoint and $\mathbf{y}$ denotes the vector containing output values. Subscripts denote appropriate indices of a matrix or a vector. e.g. y_i refers to i^{th} element of the vector $\mathbf{y}$. Scalars are italicized e.g. eta.

1 Linear Regression [45 pts]

1.1 Theoretical Derivation [15 pts]

In this problem, you will derive the linear regression formula from a statistical point of view. Consider the model:

$$p(y|\mathbf{x}) = \mathcal{N}(y|\mathbf{w}^\top\mathbf{x}, \sigma^2)$$

where $\mathbf{x}$ is an example of your data (a vector of feature values), and y is the target value. $\mathcal{N}(y|\mu, \sigma^2)$ is the Gaussian distribution with mean μ and variance σ^2.
Throughout this question, we assume σ is known. Suppose we have the following data independently generated from this model:

$$\mathcal{D} = (\mathbf{X}, \mathbf{y})$$

where $\mathbf{X} \in \mathbb{R}^{N\times K}$, the i^{th} row $\mathbf{x_i}$ has the features of the i^{th} training sample and $\mathbf{y} \in \mathbb{R}^N$, y_i is the target value of the i^{th} training sample.

(a) **[3 Points]** Conditional Log-likelihood Function: Please write out the conditional log-likelihood function $\log p\left(\mathbf{y}|\mathbf{X}, \mathbf{w}, \sigma\right)$ in terms of $\mathbf{X}$, $\mathbf{y}$, $\mathbf{w}$, and σ.

$$p\left(\mathbf{y}|\mathbf{X}, \mathbf{w}\right) = \prod_{i=1}^{n} p(y_i|\mathbf{x}_i, \mathbf{w}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{(y_i - \mathbf{w}^\top\mathbf{x}_i)^2}{2\sigma^2}) = (\frac{1}{\sqrt{2\pi}\sigma})^n \exp(-\frac{\sum_{i=1}^{n}(y_i - \mathbf{w}^\top\mathbf{x}_i)^2}{2\sigma^2}) \tag{1}$$

$$\log p\left(\mathbf{y}|\mathbf{X}, \mathbf{w}\right) = -\frac{\sum_{i=1}^{n}(y_i - \mathbf{w}^\top\mathbf{x}_i)^2}{2\sigma^2} + constant \tag{2}$$

$$\log p\left(\mathbf{y}|\mathbf{X}, \mathbf{w}\right) = -\frac{||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2}{2\sigma^2} + constant \tag{3}$$

(b) **[4 Points]** Maximum Conditional Likelihood Estimation: Please derive a formula for the MCLE estimate $\hat{\mathbf{w}}_{mle}$ that maximizes $\log p\left(\mathbf{y}|\mathbf{X}, \mathbf{w}, \sigma\right)$ in terms of $\mathbf{X}$, $\mathbf{y}$.

Hint: Write the conditional log-likelihood in terms of $\mathbf{X}$, $\mathbf{w}$, and $\mathbf{y}$. Take its derivative, set it to zero, and solve to obtain $\hat{\mathbf{w}}_{mle}$. The following identities might help: $\nabla_{\mathbf{w}} tr(\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}) = 2\mathbf{X}^T\mathbf{X}\mathbf{w}$ and $\nabla_{\mathbf{w}} tr(\mathbf{y}^T\mathbf{X}\mathbf{w}) = \mathbf{X}^T\mathbf{y}$.

Taking the gradient of $\log p\left(\mathbf{y}|\mathbf{X}, \mathbf{w}\right)$ with respect to $\mathbf{w}$ and set it to 0, we get

$$\nabla_{\mathbf{w}} \log p\left(\mathbf{y}|\mathbf{X}, \mathbf{w}\right) \propto \nabla_{\mathbf{w}}(\mathbf{X}\mathbf{w} - \mathbf{y})^\top(\mathbf{X}\mathbf{w} - \mathbf{y}) \tag{4}$$

$$\nabla_{\mathbf{w}} \log p\left(\mathbf{y}|\mathbf{X}, \mathbf{w}\right) \propto \nabla_{\mathbf{w}}(\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} + \mathbf{y}^T\mathbf{y} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{y}) \tag{5}$$

$$\nabla_{\mathbf{w}} \log p\left(\mathbf{y}|\mathbf{X}, \mathbf{w}\right) \propto 2\mathbf{X}^\top\mathbf{X}\mathbf{w} - 2\mathbf{X}^\top\mathbf{y} = 0 \tag{6}$$

$$\mathbf{w}_{\text{MLE}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y} \tag{7}$$

(c) **[3 Points]** While the analytical MCLE estimate for $\hat{\mathbf{w}}_{mle}$ that we just derived is theoretically appealing, its implementation involves an expensive linear algebraic operation related to calculating a Moore-Penrose pseudo-inverse. An alternative approach is to formulate an optimization objective corresponding to the linear regression problem and minimize it using gradient descent. To that end, show that maximizing the conditional log-likelihood $\log p\left(\mathbf{y}|\mathbf{X}, \mathbf{w}, \sigma\right)$ is equivalent to minimizing the loss objective corresponding to mean squared error $\mathcal{L}(\mathbf{w}) = \frac{1}{n}||\mathbf{X}\cdot\mathbf{w} - \mathbf{y}||_2{}^2$.

$$p\left(\mathbf{y}|\mathbf{X}, \mathbf{w}\right) = (\frac{1}{\sqrt{2\pi}\sigma})^n \exp(-\frac{\sum_{i=1}^{n}(y_i - \mathbf{w}^\top\mathbf{x}_i)^2}{2\sigma^2}) \tag{8}$$

$$\log p\left(\mathbf{y}|\mathbf{X}, \mathbf{w}\right) = -\frac{\sum_{i=1}^{n}(y_i - \mathbf{w}^\top\mathbf{x}_i)^2}{2\sigma^2} + const \tag{9}$$

$$\log p\left(\mathbf{y}|\mathbf{X}, \mathbf{w}\right) = -\frac{||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2}{2\sigma^2} + const \tag{10}$$

Hence maximizing $\log p\left(\mathbf{y}|\mathbf{X}, \mathbf{w}\right)$ is the same as minimizing $\frac{1}{n}||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2$.

(d) **[3 Points]** Derive the gradient of the above loss objective with respect to $\mathbf{w}$: $\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w})$. Also, write down the gradient descent update rule.

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = \nabla_{\mathbf{w}}(Xw - y)^T(Xw - y) \tag{11}$$
$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = \nabla_{\mathbf{w}}(w^T X^T Xw - 2y^T Xw + y^T y) \tag{12}$$
$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = 2X^T Xw - 2X^T y \tag{13}$$
$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = 2X^T(Xw - y) \tag{14}$$

(e) **[2 Points]** Each batch gradient descent iteration involves calculation of the gradient involving all datapoints. To improve convergence, stochastic gradient descent is often used in practice instead of batch gradient descent. In stochastic gradient descent, each gradient update step uses gradient calculated from a single datapoint. Write the stochastic gradient update rule using the i^{th} datapoint $(\mathbf{x}_i, y_i)$.

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = 2(\mathbf{x}_i^T\mathbf{w} - y_i)\mathbf{x}_i \tag{15}$$

1.2 Implementation [30 pts]

We will now implement linear regression using the stochastic gradient descent update rule we derived. You should submit all questions in this implementation section to autolab with the exception of question (1.2.g) which you should submit via gradescope. We will use the Boston housing prices dataset to predict median housing prices in Boston using your implementation. The original dataset can be found at https://vincentarelbundock.github.io/Rdatasets/csv/MASS/Boston.csv and the data dictionary describing the dataset can be found at https://vincentarelbundock.github.io/Rdatasets/doc/MASS/Boston.html.
You may implement your code either in Python or Octave using function signatures in the starter code provided in the handout. The following requirements must be present in the code:

(a) **[3 Points]** Implement the function `LinReg_ReadInputs(filepath)` function that reads the following four CSV files for Linear Regression:

- `LinReg_XTrain.csv` should be read in as a $N \times K$ dimensional matrix describing N training examples with K features. Name this matrix as **XTrain**.
- `LinReg_yTrain.csv` should be read in as a $N \times 1$ dimensional vector which describes the output values for training samples. Name this vector as **yTrain**.
- `LinReg_XTest.csv` should be read in as a $n \times K$ dimensional matrix describing n test examples each with K features. Name this matrix as **XTest**.
- `LinReg_yTest.csv` should be read in as a $n \times 1$ dimensional vector which describes the output values for test samples. Name this vector as **yTest**.

The function should also standardize each feature in the feature matrix to lie in the range $[0, 1]$ by using the following transformation:

$$\frac{x_k - \min(x_k)}{\max(x_k) - \min(x_k)}$$

Here $\min(x_k)$ denotes the minimum value of k^{th} feature and $\max(x_k)$ denotes the maximum value of that feature. You need to use the same standardization formula for each feature in both the train and test dataset. Do not standardize features in train dataset and test dataset separately. Target values y_i are never standardized.

(b) **[2 Points]** In your previous function, insert a bias column (i.e. a column of ones) to your **XTrain** and **XTest** matrices as the first column. You should not standardize this column.

(c) **[4 Points]** Implement the function `LinReg_CalcObj(X, y, w)` that takes X = **XTrain/XTest**, y = **yTrain/yTest** and your weight vector $\mathbf{w}$ as inputs, and outputs the value of the loss function $\mathcal{L}(\mathbf{w}) = \frac{1}{n}||\mathbf{X} \cdot \mathbf{w} - \mathbf{y}||_2^2$ we want to minimize.

(d) **[5 Points]** Implement the function `LinReg_CalcSG(x, y, w)` that calculates and returns the stochastic gradient using a particular data point $(\mathbf{x}, y)$.

(e) **[5 Points]** Implement the function `LinReg_UpdateParams(w, g, eta)` which takes in your weight vector $\mathbf{w}$, the stochastic gradient $\mathbf{g}$ and a learning constant eta, and returns an updated weight vector $\mathbf{w}$.

(f) **[6 Points]** Implement the stochastic gradient descent algorithm function `LinReg_SGD(XTrain, yTrain, XTest, yTest)` for linear regression by making use of the functions `LinReg_CalcObj`, `LinReg_CalcSG`, `LinReg_UpdateParams` implemented so far. Your function should have inputs **XTrain**, **yTrain**, **XTest** and **yTest** only. You should initialize your weight vector $\mathbf{w}$ as $w_i = 0.5\ \forall i$. Your function should output `w`: the final weight vector, `trainLoss`: a vector of loss values on your training data calculated at every epoch, and `testLoss`: a vector of loss values on your test data calculated at every epoch.

In SGD, each update using a single datapoint counts as an iteration and an SGD pass through the entire dataset is called an epoch. Your function should iterate through your entire dataset 100 times. Autolab will score your code for both accuracy and runtime efficiency. Hence, you should vectorize your code for runtime efficiency as much as possible.

Set the gradient descent learning constant to $eta = \frac{0.5}{\sqrt{(iter)}}$ where $iter$ is the current gradient descent iteration. (Note: Although this is supposed to be stochastic, **please do not randomize your training instances for the purpose of this assignment**, since we will be evaluating your code on Autolab.)

(g) **[5 Points]** Using your functions, plot the training and test losses versus the epoch. Make sure to include axis labels and a title for your plot. Report the number of epochs that are required for the algorithm to converge. Submit your answer and your graph via gradescope.

2 Logistic Regression [55 pts]

2.1 Theoretical Derivation [20 pts]

(a) **[5 Points]** In logistic regression, our goal is to learn a set of parameters by maximizing the conditional log likelihood of the data. Assuming you are given a dataset with N training examples and K features, write down a formula for the conditional log likelihood of the training data using the feature matrix $\mathbf{X}$, the class labels $\mathbf{y}$, and the weight vector $\mathbf{w}$. This will be your objective function for gradient ascent.

Taking $\mathbf{x}^{(i)}$ to be a $(p+1)$-dimensional vector where $x_0^{(i)} = 1$, the likelihood $p\left(\mathbf{y}|\mathbf{X}, \mathbf{w}\right)$ is:

$$p\left(\mathbf{y}|\mathbf{X}, \mathbf{w}\right) = L(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \prod_{i=1}^{n} p(y_i|\mathbf{x}_i, \mathbf{w}) = \prod_{i=1}^{n} \left(\frac{e^{\mathbf{w}^T\mathbf{x}^{(i)}}}{1+e^{\mathbf{w}^T\mathbf{x}^{(i)}}}\right)^{y_i} \left(\frac{1}{1+e^{\mathbf{w}^T\mathbf{x}^{(i)}}}\right)^{(1-y_i)} \tag{16}$$

$$= \prod_{i=1}^{n} \frac{\left(e^{\mathbf{w}^T\mathbf{x}^{(i)}}\right)^{y_i}}{1+e^{\mathbf{w}^T\mathbf{x}^{(i)}}} \tag{17}$$

Hence, the log-likelihood is:

$$\log p\left(\mathbf{y}|\mathbf{X}, \mathbf{w}\right) = \sum_{i=1}^{n} y_i \left(\mathbf{w}^T\mathbf{x}^{(i)}\right) - \log\left(1 + e^{\mathbf{w}^T\mathbf{x}^{(i)}}\right) \tag{18}$$

(b) **[7 Points]** Compute the partial derivative of the objective function with respect to an arbitrary w_j, i.e. derive $\partial f/\partial w_j$, where f is the objective that you provided above. Please show all derivatives can be written in a finite sum form.

The partial derivate of the log-likelihood wrt $\mathbf{w}_j$, $j \in \{0, ..., p\}$ is:

$$\frac{\partial \log p\left(\mathbf{y}|\mathbf{X},\mathbf{w}\right)}{\partial \mathbf{w}_j} = \sum_{i=1}^{n} \mathbf{x}_j^{(i)} \left[y_i - \frac{e^{\mathbf{w}^T \mathbf{x}^{(i)}}}{1 + e^{\mathbf{w}^T \mathbf{x}^{(i)}}} \right] \tag{19}$$

(c) **[3 Points]** Write gradient ascent update rules for logistic regression for arbitrary w_j.

$$w_j \leftarrow w_j + \eta \frac{\partial \log p\left(\mathbf{y}|\mathbf{X},\mathbf{w}\right)}{\partial \mathbf{w}_j} \tag{20}$$

(d) **[2 Points]** Write down the stochastic gradient ascent update using the i^{th} datapoint with features $\mathbf{x_i}$ and output label y_i.

Using the datapoint $(\mathbf{x}_i, y_i)$ for SGA:

$$w_j \leftarrow w_j + \eta \mathbf{x}_j^{(i)} \left[y_i - \frac{e^{\mathbf{w}^T \mathbf{x}^{(i)}}}{1 + e^{\mathbf{w}^T \mathbf{x}^{(i)}}} \right] \tag{21}$$

(e) **[3 Points]** If you train logistic regression for infinite iterations without l_1 or l_2 regularization, the weights can go to infinity. What is an intuitive explanation for this phenomenon? How does regularization help correct the problem?

The weights in unregularized logistic regression can be driven to infinity because it is possible to keep maximizing the conditional log-likelihood by multiplying weights with a positive scalar and making the predicted training probabilities arbitrarily close to 0 or 1. Regularization such as l_1 or l_2 has the effect of preventing weights from going to infinity by appropriately penalizing the magnitude of the weight vector elements.

2.2 Implementation [35 pts]

We will now implement logistic regression for binary classification using the stochastic gradient ascent update rule we derived. You should submit all questions in this implementation section to autolab with the exception of question (2.2.g) which you should submit via gradescope. We will use an ad filtering dataset to predict if an image posted online is part of an advertisement using your implementation. The original dataset can be found at https://www.andrew.cmu.edu/user/amaurya/docs/10601/

You should implement your code in the same language you used for the linear regression implementation and use the corresponding starter code provided in the handout. The following requirements must be present in the code:

(a) **[4 Points]** Implement the function `LogReg_ReadInputs(filepath)` that reads the following four CSV files for Logistic Regression:

- `LogReg_XTrain.csv` should be read in as a $N \times K$ dimensional matrix describing N training examples with K features. Name this matrix as **XTrain**.
- `LogReg_yTrain.csv` should be read in as a $N \times 1$ dimensional vector which describes the binary output labels for training samples. Name this vector as **yTrain**.
- `LogReg_XTest.csv` should be read in as a $n \times K$ dimensional matrix describing n test examples with K features. Name this matrix as **XTest**.
- `LogReg_yTest.csv` should be read in as a $n \times 1$ dimensional vector which describes the binary output labels for test samples. Name this vector as **yTest**.

Add a bias column at the beginning of your feature matrices like you did with Linear Regression.

(b) **[4 Points]** Implement the function `LogReg_CalcObj(X, y, w)` that takes X = **XTrain/XTest**, y = **yTrain/yTest**, your weight vector **w** as inputs, and calculates the conditional log-likelihood function $\mathcal{L}(\mathbf{w})$ we want to maximize.

(c) **[5 Points]** Implement the function `LogReg_CalcSG(x, y, w)` that calculates the stochastic gradient using a particular data point $(\mathbf{x}, y)$

(d) **[5 Points]** Implement the function `LogReg_UpdateParams(w, g, eta)` which takes in your weight vector $\mathbf{w}$, the stochastic gradient $\mathbf{g}$ and a learning constant eta and returns an updated weight vector $\mathbf{w}$.

(e) **[4 Points]** Complete the function `LogReg_PredictLabels(X, y, w)` which takes inputs X = **XTest/XTrain**, y = **yTest/yTrain** and your trained weight vector **w** and outputs `yPred` and `errorRate`, where `yPred` is a vector of predictions for input `X` and `errorRate` is percentage of misclassifications that your algorithm makes when comparing its predictions to the supplied output labels `y`.

(f) **[8 Points]** Implement the gradient ascent algorithm `LogReg_SGA(XTrain, yTrain, XTest, yTest)` for logistic regression by making use of the functions `LogReg_CalcObj, LogReg_CalcSG, LogReg_UpdateParams, LogReg_PredictLabels` implemented so far. Your function should have inputs **XTrain**, **yTrain**, **XTest** and **yTest** only. You should initialize your weight vector **w** as $w_i = 0.5$ $\forall i$ and should output `w`: your final weight vector, `trainErrorRate`: a vector of the percentage of misclassificatons on your training data at every 200 iterations, `testErrorRate`: a vector of the percentage of misclassifications on your test data at every 200 iterations, and `yPred`: a vector of your predictions for **XTest**.

Your function should iterate through your entire data set 5 times (5 epochs). Autolab will score your code for both accuracy and runtime efficiency. Hence, you should vectorize your code for runtime efficiency as much as possible.

Set the gradient descent step size as $eta = \frac{0.5}{\sqrt{(iter)}}$ where $iter$ is the current gradient ascent iteration. (Note: Although this is supposed to be stochastic, **please do not randomize your training instances for the purpose of this assignment**, since we will be evaluating your code on Autolab.)

(g) **[5 Points]** Using your functions, plot the training and test percentages of misclassified points versus the stochastic gradient ascent per 200 iterations i.e. `trainErrorRate` and `testErrorRate` obtained as `LogReg_SGA` output before. Make sure to include axis labels and a title for your plot. Report the number of iterations that are required for the algorithm to converge. Submit your answer and your graph via gradescope.

3 Collaboration Questions [2 pts]

Please complete the following questions and submit your answers in Gradescope:

3.1 Collaboration [1 Point]

- Did you receive any help whatsoever from anyone in solving this assignment? Yes / No.
- If you answered *yes*, give full details: ________________________________
  (e.g. *Jane explained to me what is asked in Question 3.4*)
- Did you give any help whatsoever to anyone in solving this assignment? Yes / No.
- If you answered *yes*, give full details: ________________________________
  (e.g. *I pointed Joe to section 2.3 to help him with Question 2*).

3.2 Time Spent [1 Point]

- How many hours did this assignment take: ________________________________

4 Submission Instructions

You will submit your code online through the CMU autolab system, which will execute it remotely against a suite of tests. Your grade will be automatically determined from the testing results. Since you get immediate feedback after submitting your code and you are allowed to submit as many different versions as you like (without any penalty), it is easy for you to check your code as you go.

To get started, you can log into the autolab website (https://autolab.andrew.cmu.edu). From there you should see 10-601 in your list of courses. Download the handout for Homework 3 (Options → Download handout) and extract the contents (i.e., by executing `tar -xvf hw3.tar` at the command line). In the archive you will find three folders. The `data` folder contains the data files for this problem. The `python` folder contains `LinReg.py` and `LogReg.py` files which contain empty function templates for each of the functions you are asked to implement. Similarly, the `octave` folder contains separate `.m` files for each of the functions that you are asked to implement.

To finish each programming part of this problem, open the `LinReg.py`, `LogReg.py` or the function-specific `.m` template files and complete the function(s) defined inside. When you are ready to submit your solutions, you will create a new tar archive of the files you are submitting. Please create the tar archive exactly as detailed below.

If you are submitting Python code:
```
tar -cvf hw3.tar LogReg.py LinReg.py
```

If you are submitting Octave code:
```
tar -cvf hw3.tar LinReg_ReadInputs.m LinReg_CalcObj.m LinReg_CalcSG.m
LinReg_UpdateParams.m LinReg_SGD.m LogReg_ReadInputs.m LogReg_CalcObj.m
LogReg_CalcSG.m LogReg_UpdateParams.m LogReg_PredictLabels.m LogReg_SGA.m
```

Copying the above commands from PDF file directly might not work. You may have to type out these commands yourself as they are shown here. If you are using any other tool for creating the tar submission (e.g. on Windows OS), make sure you tar all the code files directly and not the folder which contains your code files.

If you are working in Octave and are missing **any** of the function-specific `.m` files in your tar archive, you will receive zero points.

We have provided all of the data for this assignment as CSV files in the `data` folder in your handout. You can load the data using the `numpy.genfromtext` function in Python or the `csvread` and `csv2cell` (`io` package) functions in Octave. However, you should not upload any data files as part of your Autolab submission. Your Autolab submission should consist only of the code files listed in the python- or octave-specific tar command above.