CS 8673 Programming Assignment 1: Basic Machine Learning

August 30, 2017

This programming assignment is due on Wednesday, September 13, 2017, at 11:59pm.

Honor code note: You are welcome to discuss the assignment with other students, but you must individually and independently write the code and other deliverables that you submit for the assignment. Any copying of code from outside sources or other students will be considered an honor code violation.

---

1 Introduction

The folder you are provided contains:

- `PA1_Handout.pdf` - This file that you are reading
- `pa1data` - This is a folder containing the data files you will use for this assignment.

This first assignment will give you practice with some simple machine learning algorithms. These exercises have been tested with Python 2.7. They also rely on the numpy, scipy, and matplotlib packages. The easiest way to get all of that installed on your machine is to use Anaconda, which will install everything you need all together through a single installation. (https://www.continuum.io/downloads)

If you have not used numpy for matrix/data operations before, there will be some links from the assignment on MyCourses that will point you to some help.

If you are more comfortable with matlab or octave, you are welcome to do the assignments using those languages. Details that will help are included in next.

1.1 Credit

These assignments are based off of assignments from Andrew Ng's Stanford class, which you can find at:

http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=MachineLearning

Instructions there are for matlab/octave, so please reference them if you want to work in matlab. Solutions are available online for matlab, but please make sure that any code you submit is your own. The data for this assignment has been modified from the data used in that course, so the answers are different.

1.2 Deliverables

Please turn in your submission as a *single .pdf file* with answers to all of the relevant questions as well as the plots that you need and a code appendix with your solutions. This will make the grading process much easier.

1.3 Data

Download `pa1data.zip`, and extract the files from the zip file. They should end up in a folder called `pa1data`.

1.4 Code Imports

All of the code examples assume that the following import statements have been executed (put at the top of your python file).

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
```

2 Part A

The data files `ax.dat` and `ay.dat` contain some example measurements of heights for various boys between the ages of two and eights. The y-values are the heights measured in meters, and the x-values are the ages of the boys corresponding to the heights.

Each height and age tuple constitutes one training example $(x^{(i)}, y^{(i)})$ in our dataset. There are $m = 50$ training examples, and you will use them to develop a linear regression model.

2.1 Supervised learning problem

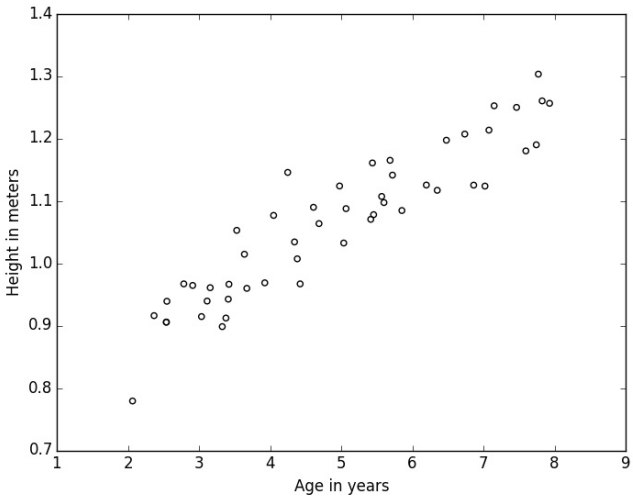
In this problem, you'll implement linear regression using gradient descent. In Python 2.7, you can load the training set using the commands

```
x = np.loadtxt('ax.dat')
y = np.loadtxt('ay.dat')
```

This will be our training set for a supervised learning problem with $n = 1$ features (in addition to the usual $x_0 = 1$, so $x \in \mathbb{R}^2$). In Python 2.7 run the following commands to plot your training set (and label the axes):

```
#Plot original data out
plt.scatter(x, y, facecolors='none')
plt.xlabel('Age in years')
plt.ylabel('Height in meters')
plt.show()
```

You should see a series of data points similar to the figure below.



Before starting gradient descent, we need to add the $x_0 = 1$ intercept term to every example. Vector data is loaded as a single-dimensional numpy array. Before we treat it as a two-dimensional matrix, we need to change its shape. To do both of these steps in Python 2.7, the commands are:

```
#Get the number of examples
m = x.shape[0]

#Reshape x to be a 2D column vector
x.shape = (m,1)

#Add a column of ones to x
x = np.hstack([np.ones((m,1)), x])
```

From this point on, you will need to remember that the age values from your training data are actually in the second column of x. This will be important when plotting your results later.

2.1.1 Linear regression

Now, we will implement linear regression for this problem. Recall that the linear regression model is

$$h_\theta(x) = \theta^T x = \sum_{i=0}^{n} \theta_i x_i,$$

and the batch gradient descent update rule is

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \qquad \text{(for all } j\text{)}$$

1. Implement gradient descent using a learning rate of $\alpha = 0.07$. Initialize the parameters to $\theta = \vec{0}$ (i.e., $\theta_0 = \theta_1 = 0$), and run one iteration of gradient descent from this initial starting point. Record the value of of θ_0 and θ_1 that you get after this first iteration. To verify that your implementation is correct, you should get the following values for θ:

θ_0	0.4942
θ_1	0.1280

2. Continue running gradient descent for more iterations until θ converges. (this will take a total of about 1300 iterations). After convergence, record the final values of θ_0 and θ_1 that you get.

   When you have found θ, plot the straight line fit from your algorithm on the same graph as your training data. The plotting commands will look something like this:

```
plt.plot(x[:,1],np.dot(x,theta))
plt.legend(['Linear Regression', 'Training Data'])
plt.show()
```

   Note that for most machine learning problems, x is very high dimensional, so we don't be able to plot $h_\theta(x)$. But since in this example we have only one feature, being able to plot this gives a nice sanity-check on our result.

3. Finally, we'd like to make some predictions using the learned hypothesis. Use your model to predict the height for a two boys of age 3.5 and age 7.

2.2 Understanding $J(\theta)$

We'd like to understand better what gradient descent has done, and visualize the relationship between the parameters $\theta \in \mathbb{R}^2$ and $J(\theta)$. To do this, we'll plot $J(\theta)$ as a 3D surface plot. (When applying learning algorithms, we don't usually try to plot $J(\theta)$ since usually $\theta \in \mathbb{R}^n$ is very high-dimensional so that we don't have any simple way to plot or visualize $J(\theta)$. But because the example here uses a very low dimensional $\theta \in \mathbb{R}^2$, we'll plot $J(\theta)$ to gain more intuition about linear regression.) Recall that the formula for $J(\theta)$ is

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

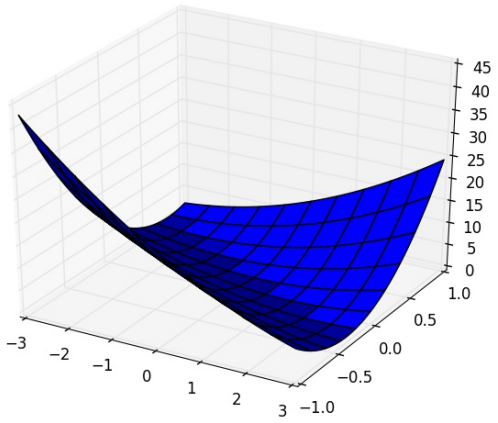
To get the best viewing results on your surface plot, use the range of theta values that we suggest in the code skeleton below.

In addition to the 3D surface plot, it can also be helpful to plot a contour plot of $J(\theta)$. This can make it easier to see where the minimum of the function is.

```
#Display Surface Plot of J
t0 = np.linspace(-3,3,100)
t1 = np.linspace(-1,1,100)

t0.shape = (len(t0),1)
t1.shape = (len(t1),1)

T0, T1 = np.meshgrid(t0,t1)

J_vals = np.zeros((len(t0),len(t1)))
for i in range(len(t0)):
    for j in range(len(t1)):
        t = np.hstack([t0[i], t1[j]])
        J_vals[i,j] = #YOUR CODE HERE#

#Because of the way meshgrids work with plotting surfaces
#we need to transpose J to show it correctly
J_vals = J_vals.T

fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(T0,T1,J_vals)
plt.show()

plt.close()

#Display Contour Plot of J
```

```
plt.contour(T0,T1,J_vals, np.logspace(-2,2,15))
plt.show()
```

You should get a figure similar to the following. You should be able to use your mouse and view the plot from different viewpoints.



What is the relationship between these plots (3D surface and contour) and the value of θ_0 and θ_1 that your implementation of gradient descent had found?

2.3 Task A Deliverables

Please include in your project write up the following plots and answers to questions

1. Plot of raw data
2. Converged value of θ
3. Plot of line that you fit
4. Prediction of your model for the height of two boys, age 3.5 and 7
5. Plot of 3D surface of J
6. Plot of contour of J
7. Answer to question about relationship between these plots and θ value found by your algorithm.
8. Include your code solution to Task A in your pdf submission

3 Part B: Linear Regression with Multiple Variables

In this exercise, you will investigate multivariate linear regression using gradient descent and the normal equations. You will also examine the relationship between the cost function $J(\theta)$, the convergence of gradient descent, and the learning rate α.

3.1 Data

The data for this part of the assignment is a training set of housing prices in Portland, Oregon, where the outputs $y^{(i)}$ are the prices and the inputs $x^{(i)}$ are the living area and the number of bedrooms. There are $m = 47$ training examples. The data is in the files `bx.data` and `by.dat`.

3.2 Preprocessing your data

Load the data for the training examples into your program and add the $x_0 = 1$ intercept term into your x matrix.

Take a look at the values of the inputs $x^{(i)}$ and note that the living areas are about 1000 times the number of bedrooms. This difference means that preprocessing the inputs will significantly increase gradient descent's efficiency.

In your program, scale both types of inputs by their standard deviations and set their means to zero. In Python 2.7, this can be executed with:

```
#Preprocess data to give std of 1 and mean of 0
sigma = np.std(x,axis=0) #std
mu = np.mean(x,axis=0)   #mean
x = (x-mu) / sigma #adjustment
```

3.3 Gradient descent

In part A, you implemented gradient descent on a univariate regression problem. The only difference now is that there is one more feature in the matrix x.

Once again, initialize your parameters to $\theta = \vec{0}$.

3.3.1 Selecting a learning rate using $J(\theta)$

Now it's time to select a learning rate α. Our current goal is to pick a good learning rate in the range of

$$0.001 \leq \alpha \leq 10$$

You will do this by making an initial selection, running gradient descent and observing the cost function, and adjusting the learning rate accordingly. Recall that the cost function is defined as

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 .$$

The cost function can also be written in the following vectorized form,

$$J(\theta) = \frac{1}{2m} \left(X\theta - \vec{y} \right)^\top \left(X\theta - \vec{y} \right)$$

where

$$\vec{y} = \begin{bmatrix} (y^{(1)}) \\ (y^{(2)}) \\ \vdots \\ (y^{(m)}) \end{bmatrix} \quad X = \begin{bmatrix} — & (x^{(1)})^\top & — \\ — & (x^{(2)})^\top & — \\ & \vdots & \\ — & (x^{(m)})^\top & — \end{bmatrix}$$

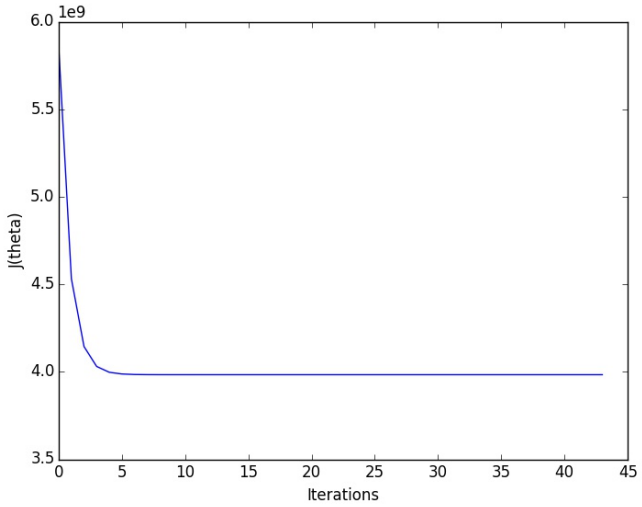
The vectorized version is useful and efficient when you're working with numerical computing tools like Python 2.7 / numpy. If you are familiar with matrices, you can prove to yourself that the two forms are equivalent.

While in part A you calculated $J(\theta)$ over a grid of θ_0 and θ_1 values, you will now calculate $J(\theta)$ using the θ of the current stage of gradient descent. After stepping through many stages, you will see how $J(\theta)$ changes as the iterations advance.

Now, run gradient descent for about 50 iterations at your initial learning rate. In each iteration, calculate $J(\theta)$ and store the result in a vector J. After the last iteration, plot the J values against the number of the iteration. In Python 2.7, the steps would look something like this:

```
theta = np.zeros(x.shape[1]) #Initialize theta
alpha = #Your learning rate#
J = []

iterations = 50

for i in range(iterations):
    J.append( #Calculate your cost function here# )
    theta = #Result of gradient descent update

#Now plot J
plt.plot(range(iterations), J)
plt.xlabel('Number of iterations')
plt.ylabel('Cost J')
```

If you picked a learning rate within a good range, your plot should appear like the figure below.



If your graph looks very different, especially if your value of $J(\theta)$ increases or even blows up, adjust your learning rate and try again. We recommend testing alphas at a rate of 3 times the next smallest value (i.e. 0.01, 0.03, 0.1, 0.3 and so on). You may also want to adjust the number of iterations you are running if that will help you see the overall trend in the curve.

To compare how different learning learning rates affect convergence, it's helpful to plot J for several learning rates on the same graph. In Python 2.7 this can be done by performing gradient descent multiple times and plotting the results each time (without the `plt.show()`), followed by finally calling `plt.show()` to plot them all).

Observe the changes in the cost function happens as the learning rate changes. What happens when the learning rate is too small? Too large?

Using the best learning rate that you found, run gradient descent until convergence to find:

1. The final values of θ
2. The predicted price of a house with 1650 square feet and 3 bedrooms. Don't forget to scale your features when you make this prediction!

3.4 Normal Equations

In lecture we saw that the closed-form solution to a least squares fit is:

$$\theta = \left(X^\top X\right)^{-1} X^\top \vec{y}.$$

Using this formula does not require any feature scaling, and you will get an exact solution in one calculation: there is no 'loop until convergence' like in gradient descent.

1. In your program, use the formula above to calculate θ. Remember that while you don't need to scale your features, you still need to add an intercept term.

2. Once you have found θ from this method, use it to make a price prediction for a 1650-square-foot house with 3 bedrooms. Did you get the same price that you found through gradient descent?

3.5 Part B Deliverables

Please include in your project write up the following plots and answers to questions

1. Plot(s) comparing different learning rates

2. Converged value of θ for best learning rate

3. Prediction of your model for the price of a house with 1650 square feet and 3 bedrooms

4. Value of θ computed using normal equations

5. Prediction of your normal equations θ on a house with 1650 square feet and 3 bedrooms.

6. Include your code solution to Task b in your pdf submission

4 Part C: Logistic Regression

In this part, you will use Newton's Method to implement logistic regression on a classification problem.

4.1 Data

For part C, suppose that a high school has a dataset representing 40 students who were admitted to college and 40 students who were not admitted. Each $(x^{(i)}, y^{(i)})$ training example contains a student's score on two standardized exams and a label of whether the student was admitted. The data is in the files `cx.data` and `cy.dat`.

Your task is to build a binary classification model that estimates college admission chances based on a student's scores on two exams. In your training data

- The first column of your x array represents all Test 1 scores, and the second column represents all Test 2 scores.

- The y vector uses '1' to label a student who was admitted and '0' to label a student who was not admitted.

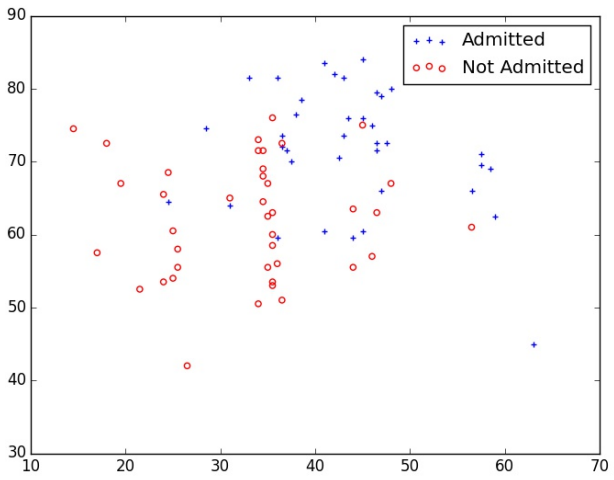
4.2 Plot the data

Load the data for the training examples into your program and add the $x_0 = 1$ intercept term into your x matrix.

Before beginning Newton's Method, we will first plot the data using different symbols to represent the two classes. In Python 2.7, you can separate the positive class and the negative class using the `nonzero` and/or `where` command:

```
#Get positive and negative indices
pos = np.nonzero(y)
neg = np.where(y==0)[0]

#Plot the different classes
plt.scatter(x[pos,0],x[pos,1],marker='+')
plt.scatter(x[neg,0],x[neg,1],facecolors='none',marker='o',
 . . . color='r')
plt.show()
```

Your plot should look like the following:



4.3 Newton's Method

Recall that in logistic regression, the hypothesis function is

$$h_\theta(x) = \sigma(\theta^\top x) = \frac{1}{1 + e^{-\theta^\top x}}$$

$$= P(y = 1|x; \theta)$$

In our example, the hypothesis is interpreted as the probability that a driver will be accident-free, given the values of the features in x.

In Python 2.7 the sigmoid function $\sigma()$ is called `expit()`, and must be imported as:

```
from scipy.special import expit
```

The cost function $J(\theta)$ is defined as

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

Our goal is to use Newton's method to minimize this function. Recall that the update rule for Newton's method is

$$\theta^{(t+1)} = \theta^{(t)} - H^{-1}\nabla_\theta J$$

In logistic regression, the gradient and the Hessian are

$$\nabla_\theta J = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

$$H = \frac{1}{m}\sum_{i=1}^{m}\left[h_\theta(x^{(i)})\left(1-h_\theta(x^{(i)})\right)x^{(i)}\left(x^{(i)}\right)^\top\right]$$

Note that the formulas presented above are the vectorized versions. Specifically, this means that $x^{(i)} \in \mathbb{R}^{n+1}$, $x^{(i)}\left(x^{(i)}\right)^\top \in \mathbb{R}^{(n+1)\times(n+1)}$, while $h_\theta(x^{(i)})$ and $y^{(i)}$ are scalars.

4.4 Implementation

Now, implement Newton's Method in your program, starting with the initial value of $\theta = \vec{0}$. To determine how many iterations to use, calculate $J(\theta)$ for each iteration and plot your results as you did in Part B. As mentioned in the lecture, Newton's method often converges in 5-15 iterations. If you find yourself using far more iterations, you should check for errors in your implementation.

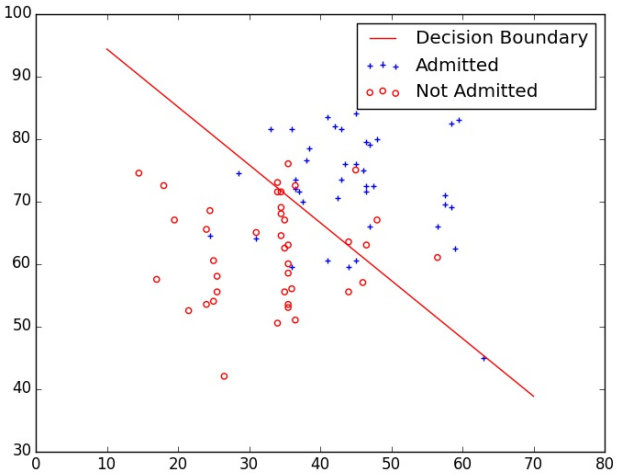
After convergence, use your values of theta to find the decision boundary in the classification problem. The decision boundary is defined as the line where

$$P(y = 1|x;\theta) = \sigma(\theta^T x) = 0.5$$

which corresponds to

$$\theta^T x = 0$$

Plotting the decision boundary is equivalent to plotting the $\theta^T x = 0$ line. When you are finished, your plot should appear like the figure below.



4.5 Part C Deliverables

Please include in your project write up the following plots and answers to questions

1. Plot of raw data
2. What values of θ did you get? How many iterations were required for convergence?
3. What is the probability that a student with a score of 20 on Exam 1 and a score of 80 on Exam 2 will not be admitted?
4. Plot of final decision boundary found