# Project 1 Due Date: February 3, 2020

**Note:** I setup Pycharm py3env virtual environment using conda package manager. The py3env virtual environment has python 3.8 along with machine learning tools like: scipy, numpy, matplotlib, pandas, statmodels, scikit-learn.

1. Read Chapters 1,2 and 3 Stephen Marsland’s M book

Ans: Done

1. **To Implement Naive Bayes From Scratch in Python**

<http://machinelearningmastery.com/naive-bayes-classifier-scratch-python/>

Ans:

iris.csv

5.1,3.5,1.4,0.2,Iris-setosa  
4.9,3.0,1.4,0.2,Iris-setosa  
4.7,3.2,1.3,0.2,Iris-setosa  
4.6,3.1,1.5,0.2,Iris-setosa  
5.0,3.6,1.4,0.2,Iris-setosa  
5.4,3.9,1.7,0.4,Iris-setosa  
4.6,3.4,1.4,0.3,Iris-setosa  
5.0,3.4,1.5,0.2,Iris-setosa  
4.4,2.9,1.4,0.2,Iris-setosa  
4.9,3.1,1.5,0.1,Iris-setosa  
5.4,3.7,1.5,0.2,Iris-setosa  
4.8,3.4,1.6,0.2,Iris-setosa  
4.8,3.0,1.4,0.1,Iris-setosa  
4.3,3.0,1.1,0.1,Iris-setosa  
5.8,4.0,1.2,0.2,Iris-setosa  
5.7,4.4,1.5,0.4,Iris-setosa  
5.4,3.9,1.3,0.4,Iris-setosa  
5.1,3.5,1.4,0.3,Iris-setosa  
5.7,3.8,1.7,0.3,Iris-setosa  
5.1,3.8,1.5,0.3,Iris-setosa  
5.4,3.4,1.7,0.2,Iris-setosa  
5.1,3.7,1.5,0.4,Iris-setosa  
4.6,3.6,1.0,0.2,Iris-setosa  
5.1,3.3,1.7,0.5,Iris-setosa  
4.8,3.4,1.9,0.2,Iris-setosa  
5.0,3.0,1.6,0.2,Iris-setosa  
5.0,3.4,1.6,0.4,Iris-setosa  
5.2,3.5,1.5,0.2,Iris-setosa  
5.2,3.4,1.4,0.2,Iris-setosa  
4.7,3.2,1.6,0.2,Iris-setosa  
4.8,3.1,1.6,0.2,Iris-setosa  
5.4,3.4,1.5,0.4,Iris-setosa  
5.2,4.1,1.5,0.1,Iris-setosa  
5.5,4.2,1.4,0.2,Iris-setosa  
4.9,3.1,1.5,0.1,Iris-setosa  
5.0,3.2,1.2,0.2,Iris-setosa  
5.5,3.5,1.3,0.2,Iris-setosa  
4.9,3.1,1.5,0.1,Iris-setosa  
4.4,3.0,1.3,0.2,Iris-setosa  
5.1,3.4,1.5,0.2,Iris-setosa  
5.0,3.5,1.3,0.3,Iris-setosa  
4.5,2.3,1.3,0.3,Iris-setosa  
4.4,3.2,1.3,0.2,Iris-setosa  
5.0,3.5,1.6,0.6,Iris-setosa  
5.1,3.8,1.9,0.4,Iris-setosa  
4.8,3.0,1.4,0.3,Iris-setosa  
5.1,3.8,1.6,0.2,Iris-setosa  
4.6,3.2,1.4,0.2,Iris-setosa  
5.3,3.7,1.5,0.2,Iris-setosa  
5.0,3.3,1.4,0.2,Iris-setosa  
7.0,3.2,4.7,1.4,Iris-versicolor  
6.4,3.2,4.5,1.5,Iris-versicolor  
6.9,3.1,4.9,1.5,Iris-versicolor  
5.5,2.3,4.0,1.3,Iris-versicolor  
6.5,2.8,4.6,1.5,Iris-versicolor  
5.7,2.8,4.5,1.3,Iris-versicolor  
6.3,3.3,4.7,1.6,Iris-versicolor  
4.9,2.4,3.3,1.0,Iris-versicolor  
6.6,2.9,4.6,1.3,Iris-versicolor  
5.2,2.7,3.9,1.4,Iris-versicolor  
5.0,2.0,3.5,1.0,Iris-versicolor  
5.9,3.0,4.2,1.5,Iris-versicolor  
6.0,2.2,4.0,1.0,Iris-versicolor  
6.1,2.9,4.7,1.4,Iris-versicolor  
5.6,2.9,3.6,1.3,Iris-versicolor  
6.7,3.1,4.4,1.4,Iris-versicolor  
5.6,3.0,4.5,1.5,Iris-versicolor  
5.8,2.7,4.1,1.0,Iris-versicolor  
6.2,2.2,4.5,1.5,Iris-versicolor  
5.6,2.5,3.9,1.1,Iris-versicolor  
5.9,3.2,4.8,1.8,Iris-versicolor  
6.1,2.8,4.0,1.3,Iris-versicolor  
6.3,2.5,4.9,1.5,Iris-versicolor  
6.1,2.8,4.7,1.2,Iris-versicolor  
6.4,2.9,4.3,1.3,Iris-versicolor  
6.6,3.0,4.4,1.4,Iris-versicolor  
6.8,2.8,4.8,1.4,Iris-versicolor  
6.7,3.0,5.0,1.7,Iris-versicolor  
6.0,2.9,4.5,1.5,Iris-versicolor  
5.7,2.6,3.5,1.0,Iris-versicolor  
5.5,2.4,3.8,1.1,Iris-versicolor  
5.5,2.4,3.7,1.0,Iris-versicolor  
5.8,2.7,3.9,1.2,Iris-versicolor  
6.0,2.7,5.1,1.6,Iris-versicolor  
5.4,3.0,4.5,1.5,Iris-versicolor  
6.0,3.4,4.5,1.6,Iris-versicolor  
6.7,3.1,4.7,1.5,Iris-versicolor  
6.3,2.3,4.4,1.3,Iris-versicolor  
5.6,3.0,4.1,1.3,Iris-versicolor  
5.5,2.5,4.0,1.3,Iris-versicolor  
5.5,2.6,4.4,1.2,Iris-versicolor  
6.1,3.0,4.6,1.4,Iris-versicolor  
5.8,2.6,4.0,1.2,Iris-versicolor  
5.0,2.3,3.3,1.0,Iris-versicolor  
5.6,2.7,4.2,1.3,Iris-versicolor  
5.7,3.0,4.2,1.2,Iris-versicolor  
5.7,2.9,4.2,1.3,Iris-versicolor  
6.2,2.9,4.3,1.3,Iris-versicolor  
5.1,2.5,3.0,1.1,Iris-versicolor  
5.7,2.8,4.1,1.3,Iris-versicolor  
6.3,3.3,6.0,2.5,Iris-virginica  
5.8,2.7,5.1,1.9,Iris-virginica  
7.1,3.0,5.9,2.1,Iris-virginica  
6.3,2.9,5.6,1.8,Iris-virginica  
6.5,3.0,5.8,2.2,Iris-virginica  
7.6,3.0,6.6,2.1,Iris-virginica  
4.9,2.5,4.5,1.7,Iris-virginica  
7.3,2.9,6.3,1.8,Iris-virginica  
6.7,2.5,5.8,1.8,Iris-virginica  
7.2,3.6,6.1,2.5,Iris-virginica  
6.5,3.2,5.1,2.0,Iris-virginica  
6.4,2.7,5.3,1.9,Iris-virginica  
6.8,3.0,5.5,2.1,Iris-virginica  
5.7,2.5,5.0,2.0,Iris-virginica  
5.8,2.8,5.1,2.4,Iris-virginica  
6.4,3.2,5.3,2.3,Iris-virginica  
6.5,3.0,5.5,1.8,Iris-virginica  
7.7,3.8,6.7,2.2,Iris-virginica  
7.7,2.6,6.9,2.3,Iris-virginica  
6.0,2.2,5.0,1.5,Iris-virginica  
6.9,3.2,5.7,2.3,Iris-virginica  
5.6,2.8,4.9,2.0,Iris-virginica  
7.7,2.8,6.7,2.0,Iris-virginica  
6.3,2.7,4.9,1.8,Iris-virginica  
6.7,3.3,5.7,2.1,Iris-virginica  
7.2,3.2,6.0,1.8,Iris-virginica  
6.2,2.8,4.8,1.8,Iris-virginica  
6.1,3.0,4.9,1.8,Iris-virginica  
6.4,2.8,5.6,2.1,Iris-virginica  
7.2,3.0,5.8,1.6,Iris-virginica  
7.4,2.8,6.1,1.9,Iris-virginica  
7.9,3.8,6.4,2.0,Iris-virginica  
6.4,2.8,5.6,2.2,Iris-virginica  
6.3,2.8,5.1,1.5,Iris-virginica  
6.1,2.6,5.6,1.4,Iris-virginica  
7.7,3.0,6.1,2.3,Iris-virginica  
6.3,3.4,5.6,2.4,Iris-virginica  
6.4,3.1,5.5,1.8,Iris-virginica  
6.0,3.0,4.8,1.8,Iris-virginica  
6.9,3.1,5.4,2.1,Iris-virginica  
6.7,3.1,5.6,2.4,Iris-virginica  
6.9,3.1,5.1,2.3,Iris-virginica  
5.8,2.7,5.1,1.9,Iris-virginica  
6.8,3.2,5.9,2.3,Iris-virginica  
6.7,3.3,5.7,2.5,Iris-virginica  
6.7,3.0,5.2,2.3,Iris-virginica  
6.3,2.5,5.0,1.9,Iris-virginica  
6.5,3.0,5.2,2.0,Iris-virginica  
6.2,3.4,5.4,2.3,Iris-virginica  
5.9,3.0,5.1,1.8,Iris-virginica

NaivesBayes.py

# Example of calculating class probabilities  
from math import sqrt  
from math import pi  
from math import exp  
  
# Split the dataset by class values, returns a dictionary  
def separate\_by\_class(dataset):  
 separated = dict()  
 for i in range(len(dataset)):  
 vector = dataset[i]  
 class\_value = vector[-1]  
 if (class\_value not in separated):  
 separated[class\_value] = list()  
 separated[class\_value].append(vector)  
 return separated  
  
# Calculate the mean of a list of numbers  
def mean(numbers):  
 return sum(numbers)/float(len(numbers))  
  
# Calculate the standard deviation of a list of numbers  
def stdev(numbers):  
 avg = mean(numbers)  
 variance = sum([(x-avg)\*\*2 for x in numbers]) / float(len(numbers)-1)  
 return sqrt(variance)  
  
# Calculate the mean, stdev and count for each column in a dataset  
def summarize\_dataset(dataset):  
 summaries = [(mean(column), stdev(column), len(column)) for column in zip(\*dataset)]  
 del(summaries[-1])  
 return summaries  
  
# Split dataset by class then calculate statistics for each row  
def summarize\_by\_class(dataset):  
 separated = separate\_by\_class(dataset)  
 summaries = dict()  
 for class\_value, rows in separated.items():  
 summaries[class\_value] = summarize\_dataset(rows)  
 return summaries  
  
# Calculate the Gaussian probability distribution function for x  
def calculate\_probability(x, mean, stdev):  
 exponent = exp(-((x-mean)\*\*2 / (2 \* stdev\*\*2 )))  
 return (1 / (sqrt(2 \* pi) \* stdev)) \* exponent  
  
# Calculate the probabilities of predicting each class for a given row  
def calculate\_class\_probabilities(summaries, row):  
 total\_rows = sum([summaries[label][0][2] for label in summaries])  
 probabilities = dict()  
 for class\_value, class\_summaries in summaries.items():  
 probabilities[class\_value] = summaries[class\_value][0][2]/float(total\_rows)  
 for i in range(len(class\_summaries)):  
 mean, stdev, \_ = class\_summaries[i]  
 probabilities[class\_value] \*= calculate\_probability(row[i], mean, stdev)  
 return probabilities  
  
# Test calculating class probabilities  
dataset = [[3.393533211,2.331273381,0],  
 [3.110073483,1.781539638,0],  
 [1.343808831,3.368360954,0],  
 [3.582294042,4.67917911,0],  
 [2.280362439,2.866990263,0],  
 [7.423436942,4.696522875,1],  
 [5.745051997,3.533989803,1],  
 [9.172168622,2.511101045,1],  
 [7.792783481,3.424088941,1],  
 [7.939820817,0.791637231,1]]  
summaries = summarize\_by\_class(dataset)  
probabilities = calculate\_class\_probabilities(summaries, dataset[0])  
print(probabilities)

On run of NaivesBayes.py

/Users/upgautam/anaconda3/envs/py3env/bin/python /Users/upgautam/PycharmProjects/NaivesBayesQ2/NaivesBayes.py

{0: 0.05032427673372075, 1: 0.00011557718379945765}

Process finished with exit code 0

NaivesBayesFromData.py

# This is from iris.csv data

# Naive Bayes On The Iris Dataset  
from csv import reader  
from random import seed  
from random import randrange  
from math import sqrt  
from math import exp  
from math import pi  
  
# Test Naive Bayes on Iris Dataset  
seed(1)  
filename = 'iris.csv'  
  
# Load a CSV file  
def load\_csv(filename):  
 dataset = list()  
 with open(filename, 'r') as file:  
 csv\_reader = reader(file)  
 for row in csv\_reader:  
 if not row:  
 continue  
 dataset.append(row)  
 return dataset  
  
dataset = load\_csv(filename)  
  
# Convert string column to float  
def str\_column\_to\_float(dataset, column):  
 for row in dataset:  
 row[column] = float(row[column].strip())  
  
# Convert string column to integer  
def str\_column\_to\_int(dataset, column):  
 class\_values = [row[column] for row in dataset]  
 unique = set(class\_values)  
 lookup = dict()  
 for i, value in enumerate(unique):  
 lookup[value] = i  
 for row in dataset:  
 row[column] = lookup[row[column]]  
 return lookup  
  
for i in range(len(dataset[0])-1):  
 str\_column\_to\_float(dataset, i)  
# convert class column to integers  
str\_column\_to\_int(dataset, len(dataset[0])-1)  
# evaluate algorithm  
n\_folds = 5  
  
  
  
# Split a dataset into k folds  
def cross\_validation\_split(dataset, n\_folds):  
 dataset\_split = list()  
 dataset\_copy = list(dataset)  
 fold\_size = int(len(dataset) / n\_folds)  
 for \_ in range(n\_folds):  
 fold = list()  
 while len(fold) < fold\_size:  
 index = randrange(len(dataset\_copy))  
 fold.append(dataset\_copy.pop(index))  
 dataset\_split.append(fold)  
 return dataset\_split  
  
# Calculate accuracy percentage  
def accuracy\_metric(actual, predicted):  
 correct = 0  
 for i in range(len(actual)):  
 if actual[i] == predicted[i]:  
 correct += 1  
 return correct / float(len(actual)) \* 100.0  
  
# Evaluate an algorithm using a cross validation split  
def evaluate\_algorithm(dataset, algorithm, n\_folds, \*args):  
 folds = cross\_validation\_split(dataset, n\_folds)  
 scores = list()  
 for fold in folds:  
 train\_set = list(folds)  
 train\_set.remove(fold)  
 train\_set = sum(train\_set, [])  
 test\_set = list()  
 for row in fold:  
 row\_copy = list(row)  
 test\_set.append(row\_copy)  
 row\_copy[-1] = None  
 predicted = algorithm(train\_set, test\_set, \*args)  
 actual = [row[-1] for row in fold]  
 accuracy = accuracy\_metric(actual, predicted)  
 scores.append(accuracy)  
 return scores  
  
# Split the dataset by class values, returns a dictionary  
def separate\_by\_class(dataset):  
 separated = dict()  
 for i in range(len(dataset)):  
 vector = dataset[i]  
 class\_value = vector[-1]  
 if (class\_value not in separated):  
 separated[class\_value] = list()  
 separated[class\_value].append(vector)  
 return separated  
  
# Calculate the mean of a list of numbers  
def mean(numbers):  
 return sum(numbers)/float(len(numbers))  
  
# Calculate the standard deviation of a list of numbers  
def stdev(numbers):  
 avg = mean(numbers)  
 variance = sum([(x-avg)\*\*2 for x in numbers]) / float(len(numbers)-1)  
 return sqrt(variance)  
  
# Calculate the mean, stdev and count for each column in a dataset  
def summarize\_dataset(dataset):  
 summaries = [(mean(column), stdev(column), len(column)) for column in zip(\*dataset)]  
 del(summaries[-1])  
 return summaries  
  
# Split dataset by class then calculate statistics for each row  
def summarize\_by\_class(dataset):  
 separated = separate\_by\_class(dataset)  
 summaries = dict()  
 for class\_value, rows in separated.items():  
 summaries[class\_value] = summarize\_dataset(rows)  
 return summaries  
  
# Calculate the Gaussian probability distribution function for x  
def calculate\_probability(x, mean, stdev):  
 exponent = exp(-((x-mean)\*\*2 / (2 \* stdev\*\*2 )))  
 return (1 / (sqrt(2 \* pi) \* stdev)) \* exponent  
  
# Calculate the probabilities of predicting each class for a given row  
def calculate\_class\_probabilities(summaries, row):  
 total\_rows = sum([summaries[label][0][2] for label in summaries])  
 probabilities = dict()  
 for class\_value, class\_summaries in summaries.items():  
 probabilities[class\_value] = summaries[class\_value][0][2]/float(total\_rows)  
 for i in range(len(class\_summaries)):  
 mean, stdev, \_ = class\_summaries[i]  
 probabilities[class\_value] \*= calculate\_probability(row[i], mean, stdev)  
 return probabilities  
  
# Predict the class for a given row  
def predict(summaries, row):  
 probabilities = calculate\_class\_probabilities(summaries, row)  
 best\_label, best\_prob = None, -1  
 for class\_value, probability in probabilities.items():  
 if best\_label is None or probability > best\_prob:  
 best\_prob = probability  
 best\_label = class\_value  
 return best\_label  
  
# Naive Bayes Algorithm  
def naive\_bayes(train, test):  
 summarize = summarize\_by\_class(train)  
 predictions = list()  
 for row in test:  
 output = predict(summarize, row)  
 predictions.append(output)  
 return(predictions)  
  
  
scores = evaluate\_algorithm(dataset, naive\_bayes, n\_folds)  
print('Scores: %s' % scores)  
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))

On run of NaivesBayesFromData.py

/Users/upgautam/anaconda3/envs/py3env/bin/python /Users/upgautam/PycharmProjects/NaivesBayesQ2/NaivesBayesFromData.py

Scores: [93.33333333333333, 96.66666666666667, 100.0, 93.33333333333333, 93.33333333333333]

Mean Accuracy: 95.333% = all\_sum/5

Process finished with exit code 0

# Linear Regression

## Files included in this exercise

ex1data1.txt – Dataset for linear regression with one variable

## Linear regression with one variable

Task: Predict profits for a food truck, suppose you’re the CEO of a restaurant franchise and are considering different cities for opening a new outlet. The chain already has trucks in various cities and you have data for profits and populations from the cities.

You would like to use this data to help you select which city to expand to next. The file ex1data1.txt contains the dataset for our linear regression problem. The first column is the population of a city and the second column is the profit of a food truck in that city. A negative value for profit indicates a loss.

Ans:

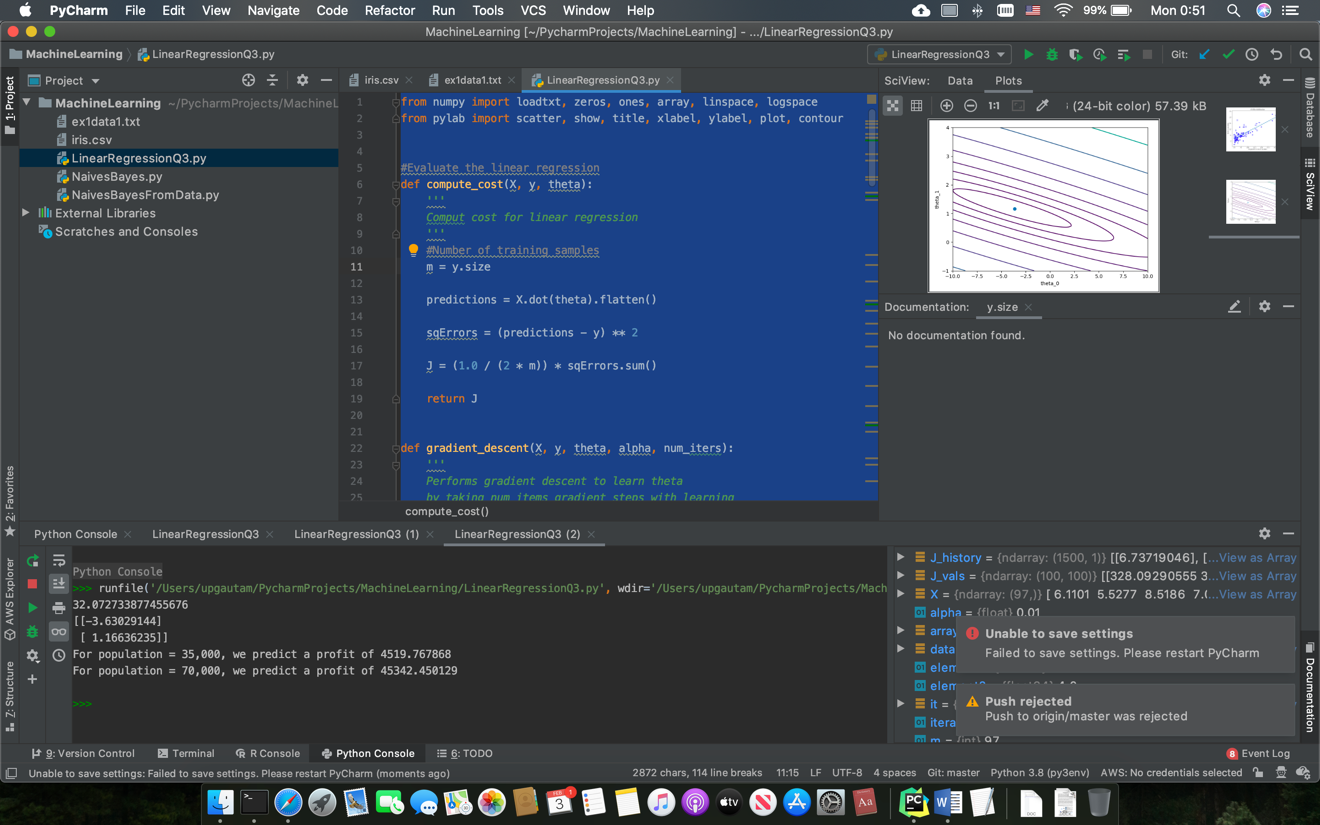
ex1data1.txt

6.1101,17.592  
5.5277,9.1302  
8.5186,13.662  
7.0032,11.854  
5.8598,6.8233  
8.3829,11.886  
7.4764,4.3483  
8.5781,12  
6.4862,6.5987  
5.0546,3.8166  
5.7107,3.2522  
14.164,15.505  
5.734,3.1551  
8.4084,7.2258  
5.6407,0.71618  
5.3794,3.5129  
6.3654,5.3048  
5.1301,0.56077  
6.4296,3.6518  
7.0708,5.3893  
6.1891,3.1386  
20.27,21.767  
5.4901,4.263  
6.3261,5.1875  
5.5649,3.0825  
18.945,22.638  
12.828,13.501  
10.957,7.0467  
13.176,14.692  
22.203,24.147  
5.2524,-1.22  
6.5894,5.9966  
9.2482,12.134  
5.8918,1.8495  
8.2111,6.5426  
7.9334,4.5623  
8.0959,4.1164  
5.6063,3.3928  
12.836,10.117  
6.3534,5.4974  
5.4069,0.55657  
6.8825,3.9115  
11.708,5.3854  
5.7737,2.4406  
7.8247,6.7318  
7.0931,1.0463  
5.0702,5.1337  
5.8014,1.844  
11.7,8.0043  
5.5416,1.0179  
7.5402,6.7504  
5.3077,1.8396  
7.4239,4.2885  
7.6031,4.9981  
6.3328,1.4233  
6.3589,-1.4211  
6.2742,2.4756  
5.6397,4.6042  
9.3102,3.9624  
9.4536,5.4141  
8.8254,5.1694  
5.1793,-0.74279  
21.279,17.929  
14.908,12.054  
18.959,17.054  
7.2182,4.8852  
8.2951,5.7442  
10.236,7.7754  
5.4994,1.0173  
20.341,20.992  
10.136,6.6799  
7.3345,4.0259  
6.0062,1.2784  
7.2259,3.3411  
5.0269,-2.6807  
6.5479,0.29678  
7.5386,3.8845  
5.0365,5.7014  
10.274,6.7526  
5.1077,2.0576  
5.7292,0.47953  
5.1884,0.20421  
6.3557,0.67861  
9.7687,7.5435  
6.5159,5.3436  
8.5172,4.2415  
9.1802,6.7981  
6.002,0.92695  
5.5204,0.152  
5.0594,2.8214  
5.7077,1.8451  
7.6366,4.2959  
5.8707,7.2029  
5.3054,1.9869  
8.2934,0.14454  
13.394,9.0551  
5.4369,0.61705

LinearRegressionQ3.py

from numpy import loadtxt, zeros, ones, array, linspace, logspace  
from pylab import scatter, show, title, xlabel, ylabel, plot, contour  
  
  
#Evaluate the linear regression  
def compute\_cost(X, y, theta):  
 *'''  
 Comput cost for linear regression  
 '''* #Number of training samples  
 m = y.size  
  
 predictions = X.dot(theta).flatten()  
  
 sqErrors = (predictions - y) \*\* 2  
  
 J = (1.0 / (2 \* m)) \* sqErrors.sum()  
  
 return J  
  
  
def gradient\_descent(X, y, theta, alpha, num\_iters):  
 *'''  
 Performs gradient descent to learn theta  
 by taking num\_items gradient steps with learning  
 rate alpha  
 '''* m = y.size  
 J\_history = zeros(shape=(num\_iters, 1))  
  
 for i in range(num\_iters):  
  
 predictions = X.dot(theta).flatten()  
  
 errors\_x1 = (predictions - y) \* X[:, 0]  
 errors\_x2 = (predictions - y) \* X[:, 1]  
  
 theta[0][0] = theta[0][0] - alpha \* (1.0 / m) \* errors\_x1.sum()  
 theta[1][0] = theta[1][0] - alpha \* (1.0 / m) \* errors\_x2.sum()  
  
 J\_history[i, 0] = compute\_cost(X, y, theta)  
  
 return theta, J\_history  
  
  
#Load the dataset  
data = loadtxt('ex1data1.txt', delimiter=',')  
  
#Plot the data  
scatter(data[:, 0], data[:, 1], marker='o', c='b')  
title('Profits distribution')  
xlabel('Population of City in 10,000s')  
ylabel('Profit in $10,000s')  
#show()  
  
X = data[:, 0]  
y = data[:, 1]  
  
  
#number of training samples  
m = y.size  
  
#Add a column of ones to X (interception data)  
it = ones(shape=(m, 2))  
it[:, 1] = X  
  
#Initialize theta parameters  
theta = zeros(shape=(2, 1))  
  
#Some gradient descent settings  
iterations = 1500  
alpha = 0.01  
  
#compute and display initial cost  
print(compute\_cost(it, y, theta))  
  
theta, J\_history = gradient\_descent(it, y, theta, alpha, iterations)  
  
print(theta)  
#Predict values for population sizes of 35,000 and 70,000  
predict1 = array([1, 3.5]).dot(theta).flatten()  
print('For population = 35,000, we predict a profit of %f' % (predict1 \* 10000))  
predict2 = array([1, 7.0]).dot(theta).flatten()  
print('For population = 70,000, we predict a profit of %f' % (predict2 \* 10000))  
  
#Plot the results  
result = it.dot(theta).flatten()  
plot(data[:, 0], result)  
show()  
  
  
#Grid over which we will calculate J  
theta0\_vals = linspace(-10, 10, 100)  
theta1\_vals = linspace(-1, 4, 100)  
  
  
#initialize J\_vals to a matrix of 0's  
J\_vals = zeros(shape=(theta0\_vals.size, theta1\_vals.size))  
  
#Fill out J\_vals  
for t1, element in enumerate(theta0\_vals):  
 for t2, element2 in enumerate(theta1\_vals):  
 thetaT = zeros(shape=(2, 1))  
 thetaT[0][0] = element  
 thetaT[1][0] = element2  
 J\_vals[t1, t2] = compute\_cost(it, y, thetaT)  
  
#Contour plot  
J\_vals = J\_vals.T  
#Plot J\_vals as 15 contours spaced logarithmically between 0.01 and 100  
contour(theta0\_vals, theta1\_vals, J\_vals, logspace(-2, 3, 20))  
xlabel('theta\_0')  
ylabel('theta\_1')  
scatter(theta[0][0], theta[1][0])  
show()

On run of LinearRegressionQ3.py



References

tutorials used:

<http://aimotion.blogspot.com/2011/10/machine-learning-with-python-linear.html>

<https://gist.github.com/marcelcaraciolo/1321575>

<https://medium.com/@jdwittenauer/machine-learning-exercises-in-python-part-1-60db0df846a4>

derivatives

<https://www.symbolab.com/practice/derivatives-practice>

<https://www.symbolab.com/solver/derivative-calculator/%5Cfrac%7Bd%7D%7Bdx%7D%5Cleft(ax%2Bb%5Cright)>

dataset used:

[https://searchcode.com/codesearch/view/5404318/](https://searchcode.com/codesearch/view/5404318/" \t "_blank)