{

"cells": [

{

"cell\_type": "code",

"execution\_count": 2,

"metadata": {

"collapsed": true

},

"outputs": [],

"source": [

"import csv\n",

"import numpy as np\n",

"import scipy as sp\n",

"import matplotlib\n",

"from matplotlib import pyplot as plt"

]

},

{

"cell\_type": "code",

"execution\_count": 3,

"metadata": {

"collapsed": true

},

"outputs": [],

"source": [

"#############\n",

"## DATA IO ##\n",

"#############\n",

"\n",

"def get\_data(filepath):\n",

" # Opens the file handler for the dataset file. Using variable 'f' we can access and manipulate our file anywhere in our code\n",

" # after the next code line.\n",

" f = open(\"hardware\_v2\",\"r\")\n",

"\n",

" # Predictors Collection (or your input variable) (which in this case is just the duration of eruption)\n",

" X1 = []\n",

" X2 = []\n",

" X3 = []\n",

" X4 = []\n",

" X5 = []\n",

" X6 = []\n",

" \n",

" # Output Response (or your output variable) (which in this case is the duration after which next eruption will occur.)\n",

" Y = []\n",

"\n",

" # Initializing a reader generator using reader method from csv module. A reader generator takes each line from the file\n",

" # and converts it into list of columns.\n",

" reader = csv.reader(f)\n",

"\n",

" # Using for loop, we are able to read one row at a time.\n",

" for row in reader:\n",

" if row[1]!=\"MYCT\":\n",

" X1.append(float(row[1]))\n",

" X2.append(float(row[2]))\n",

" X3.append(float(row[3]))\n",

" X4.append(float(row[4]))\n",

" X5.append(float(row[5]))\n",

" X6.append(float(row[6]))\n",

" Y.append(float(row[7]))\n",

"\n",

" # Close the file once we have succesffuly stored all data into our X and Y variables.hgg\n",

" f.close()\n",

"\n",

" return [[np.array(X1),np.array(X2),np.array(X3),np.array(X4),np.array(X5),np.array(X6)],np.array(Y)]\n"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {

"collapsed": true

},

"outputs": [],

"source": [

"#####################\n",

"## RSS Calculation ##\n",

"#####################\n",

"\n",

"def RSS(x, y, beta\_0,betas):\n",

" rss = 0\n",

" for i in range(x[0].shape[0]):\n",

" predicted\_value = (betas[0] + (betas[1] \* x[1][i])+(betas[2] \* x[2])[i]+(betas[3] \* x[3])[i]+(betas[4] \* x[4][i])+(betas[5] \*x[5] x[i])+(betas[6] \*x[6] x[i]))\n",

" actual\_value = y[i]\n",

" rss = rss + ((predicted\_value - actual\_value)\*\*2)\n",

" return rss"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {

"collapsed": true

},

"outputs": [],

"source": [

"def predicted\_value\_final(x,i,betas):\n",

" return (betas[0] + (betas[1]\*X[0][i]) + (betas[2]\*X[1][i]) + (betas[3]\*X[2][i]) + (betas[4]\*X[3][i]) \n",

" + (betas[5]\*X[4][i]) + (betas[6]\*X[5][i]))"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {

"collapsed": true

},

"outputs": [],

"source": [

"#Gradient Descent\n",

"\n",

"def gradientDescentAlgorithm(x, y, learning\_rate):\n",

" \n",

" print (\"Training Linear Regression Model using Gradient Descent\")\n",

" \n",

" maximum\_iterations = 2500\n",

" \n",

" # This flag lets the program know wether the gradient descent algorithm has reached it's converged state which means wether \n",

" # the algorithm was able to find the local minima (where the slope of RSS wrt your parameters beta\_0 and beta\_1 is zero)\n",

" converge\_status = False\n",

" \n",

" # num\_rows stores the number of datapoints in the current dataset provided for training.\n",

" num\_rows = x[0].shape[0]\n",

"\n",

" # Initial Value of parameters ((beta\_0, beta\_1) - for our simple linear regression case)\n",

" betas = [0,0,0,0,0,0]\n",

"\n",

" # Initial Error or RSS(beta\_0,beta\_1) based on the initial parameter values\n",

" error = RSS(x, y, betas)\n",

" print('Initial Value of RSS (Cost Function)=', error);\n",

" \n",

" # Iterate Loop\n",

" num\_iter = 0\n",

" while not converge\_status:\n",

" # for each training sample, compute the gradient\n",

" gradient\_0 = 1.0/num\_rows \* sum([(predicted\_value\_final(x,i,betas) - y[i]) for i in range(num\_rows)]) \n",

" gradient\_1 = 1.0/num\_rows \* sum([(predicted\_value\_final(x,i,betas) - y[i])\*x[1][i] for i in range(num\_rows)])\n",

" gradient\_2 = 1.0/num\_rows \* sum([(predicted\_value\_final(x,i,betas) - y[i])\*x[2][i] for i in range(num\_rows)])\n",

" gradient\_3 = 1.0/num\_rows \* sum([(predicted\_value\_final(x,i,betas) - y[i])\*x[3][i] for i in range(num\_rows)])\n",

" gradient\_4 = 1.0/num\_rows \* sum([(predicted\_value\_final(x,i,betas) - y[i])\*x[4][i] for i in range(num\_rows)])\n",

" gradient\_5 = 1.0/num\_rows \* sum([(predicted\_value\_final(x,i,betas) - y[i])\*x[5][i] for i in range(num\_rows)])\n",

" gradient\_6 = 1.0/num\_rows \* sum([(predicted\_value\_final(x,i,betas) - y[i])\*x[6][i] for i in range(num\_rows)])\n",

" # Computation of new parameters according to the current gradient.\n",

" temp0 = betas[0] - learning\_rate \* gradient\_0\n",

" temp1 = betas[1] - learning\_rate \* gradient\_1\n",

" temp2 = betas[2] - learning\_rate \* gradient\_2\n",

" temp3 = betas[3] - learning\_rate \* gradient\_3\n",

" temp4 = betas[4] - learning\_rate \* gradient\_4\n",

" temp5 = betas[5] - learning\_rate \* gradient\_5\n",

" temp6 = betas[6] - learning\_rate \* gradient\_6\n",

" \n",

" # Simultaneous Update of Parameters Beta\_0 and Beta\_1.\n",

" beta\_0 = temp0\n",

" beta\_1 = temp1\n",

" beta\_2 = temp2\n",

" beta\_3 = temp3\n",

" beta\_4 = temp4\n",

" beta\_5 = temp5\n",

" beta\_6 = temp6\n",

"\n",

" \n",

" current\_error = RSS(x, y,betas)\n",

" \n",

" if num\_iter % 250 == 0:\n",

" print ('Current Value of RSS (Cost Function) based on updated values of beta\_0 and beta\_1 = ', current\_error)\n",

" \n",

" error = current\_error # update error \n",

" num\_iter = num\_iter + 1 # update iter\n",

" \n",

" if num\_iter == maximum\_iterations:\n",

" print (\"Training Interrupted as Maximum number of iterations were crossed.\\n\\n\")\n",

" converge\_status = True\n",

"\n",

" return [betas]\n"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {

"collapsed": true

},

"outputs": [],

"source": [

"# Method to predict response variable Y (in this case interval before the next erruption) for new values of X (in this case\n",

"# duration of eruption) using the estimated coefficientsself.\n",

"# This method can predict Response variable (Y) for single as well as multiple values of X. If only a single numerical Value\n",

"# input variable (X) which in this case is Duration is passed. It will return the prediction for only that single numerical\n",

"# value. If a collection of different values for input variable (list) is passed, it will return a list of predictions\n",

"# for each input value.\n",

"# \"if\" statement on line number 72 takes care of understanding if the input value is singular or a list.\n",

"def predict(coef,X):\n",

" beta\_0 = coef[0]\n",

" beta\_1 = coef[1]\n",

" beta\_2 = coef[2]\n",

" beta\_3 = coef[3]\n",

" beta\_4 = coef[4]\n",

" beta\_5 = coef[5]\n",

" beta\_6 = coef[1]\n",

" fy = []\n",

" if type(X) == list:\n",

" for x in X:\n",

" fy.append(beta\_0 + (beta\_1 \* x[1])+(beta\_2 \* x[2])+(beta\_3 \* x[3])+(beta\_4 \* x[4])+(beta\_5 \* x[5])+(beta\_6 \* x[6]))\n",

" return fy\n",

"\n",

" # Our Regression Model defined using the coefficients from slr function\n",

" x = X[0]\n",

" Y = beta\_0 + (beta\_1 \* x[1]) + (beta\_2 \* x[2]) + (beta\_3 \* x[3]) + (beta\_4 \* x[4]) + (beta\_5 \* x[5]) + (beta\_6 \* x[6])\n",

"\n",

" return Y"

]

}

],

"metadata": {

"kernelspec": {

"display\_name": "Python 3",

"language": "python",

"name": "python3"

},

"language\_info": {

"codemirror\_mode": {

"name": "ipython",

"version": 3

},

"file\_extension": ".py",

"mimetype": "text/x-python",

"name": "python",

"nbconvert\_exporter": "python",

"pygments\_lexer": "ipython3",

"version": "3.6.3"

}

},

"nbformat": 4,

"nbformat\_minor": 2

}