Python Machine Learning Tutorial

Linear Regression

Import Numpy and matplotlib:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Define 100 random x points between a=10 and b=90:

https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.random.random.html

```
a=10
b=90
x = (b-a)* np.random.random((100, 1)) + a
```

Plot a linear curve y = slope*x + y_int + random_normal_noise:

```
noise = 10*np.random.normal(size=x.shape)
slope = 2.5
y_int = 3.25
y = slope*x + y_int + noise
```

Scatter plot y vs x; plot true for comparison:

```
plt.scatter(x,y)
plt.plot(np.linspace(0,100,100),3.25+2.5*np.linspace(0,100,100),'r')    #true y for compariso
plt.xlabel('x')
plt.ylabel('y')
plt.show
```

Perform GD: Define number of iterations, learning rate, cost function, gradient, weights

```
m = len(x)
w = 10*np.random.random((2,1))                  #random initialize: w0=y_int ; w1=slope
alpha = 0.0001
itera = 1000
dJdw0 = 1
dJdw1 = x

for i in range(itera):                  #be careful iter() is a function i.e reserved word
    y_hat = w[0] + w[1]*x
    error = y_hat-y
    J = np.sum(error**2)/(2*m)

    w[0] = w[0] - alpha/m*np.sum(error*dJdw0)
    w[1] = w[1] - alpha/m*np.sum(error*dJdw1)
    print("iteration: %4d cost: %10.2f alpha: %10.8f w0: %10.2f w1: %10.2f" %(i, J, alpha, w[0], w[1]))

print("cost: %10.2f   alpha: %10.8f  w0: %10.2f   w1:  %10.2f" %(J, alpha, w[0], w[1]))
```

Display the scatter plot, truth curve, GD curve

```
plt.scatter(x,y)        #data
plt.plot(x,y_hat)       #GD curve
plt.plot(np.linspace(0,100,100),3.25+2.5*np.linspace(0,100,100),'r--')  #truth
plt.show
```

Perform GD using the matrix method: augment x into X; then perform $W = (X^T X)^{-1} X^T Y$

```
X = np.ones((len(x),2))
X[:,1]=list(x)
Y = y
W = np.dot(np.dot(np.linalg.inv(np.dot(X.T,X)),X.T),Y)
print(W)
```

Perform GD using built-in function. Can use a number of functions, checkout:
https://medium.freecodecamp.org/data-science-with-python-8-ways-to-do-linear-regression-and-measure-their-speed-b5577d75f8b

```
X = x.reshape(100,)
Y = y.reshape(100,)
W=np.polyfit(X,Y,1)
print(W)
```

Create a 3rd order polynomial x:

```
order = 3
X=np.zeros((len(x),order+1))
for     i in range(order+1):
        X[:,i]=list(x**i)
```

Repeat GD for a univariate polynomial regression:

```
W = np.dot(np.dot(np.linalg.inv(np.dot(X.T,X)),X.T),Y)
print(W)
```

Plot original data and the polynomial fit:

```
plt.scatter(x,y)
xs = x
xs.sort(axis=0)
X=np.zeros((len(x),order+1))
for i in range(order+1):
    X[:,i]=list(xs**i)

W = W.reshape(4,1)
y_hat=np.dot(X,W)

plt.plot(x,y_hat)
plt.show
```

Logistic Regression

Create 2-class (binary) random normally distributed data

```
x11 = np.random.normal(10, 2, 20).reshape(20,1)     #mu = 10, sig=2, samples=20
x21 = np.random.normal(5, 2, 20).reshape(20,1)
x12 = np.random.normal(5, 3, 20).reshape(20,1)
x22 = np.random.normal(10, 3, 20).reshape(20,1)

X1 = np.hstack((np.ones((20,1)),x11,x21))               #20x3 for class 1
X2 = np.hstack((np.ones((20,1)),x12,x22))           #20x3 for class 2

X = np.vstack ((X1,X2))                 #combine all x values
Y = np.vstack ((np.ones((20,1)), np.zeros((20,1))))
```

Display the data

```
plt.plot(x11,x21,'ro',x12,x22,'bo')     #class1 is (x11,x21); class2 is (x12,x22)
plt.show
```

Perform GD: Define number of iterations, learning rate, cost function, gradient, weights

```
alpha = 0.01
itera = 10000
m = Y.shape[0]                       #or use m = Y.size

W = np.random.random((3,1))                       #3x1. (in the 3 class example it has been corrected)

for i in range(itera):
    Z = np.dot(X, W)
    H = 1 / (1 + np.exp(-Z))
    L = -np.sum(Y*np.log(H)+ (1-Y)*np.log(1-H)) #Log(H) for Y = 1 and log(1-H) for Y=0
    dW = np.dot(X.T, (H - Y)) / m
    W = W - alpha*dW
```

Display the boundary: Boundary exist where sigmoid = 1/2, i.e. WX= 0 $\Rightarrow w_0 + w_1x_1 + w_2x_2 = 0 \Rightarrow x_2 = -\frac{w_0+w_1}{w_2}$

```
y1 = np.array([np.min(X),np.max(X)])              #plot varies between 0 and 15
y2 = -((W[0,0] + W[1,0]*y1)/W[2,0])
plt.plot(x11,x21,'ro',x12,x22,'bo')
plt.plot(y1,y2,'--')
plt.show
```

Add a 3rd class to the data and display data

```
x13 = np.random.normal(10, 2, 20).reshape(20,1) #center around 10,15
x23 = np.random.normal(15, 3, 20).reshape(20,1)
X3 =np.hstack([np.ones((20,1)),x13,x23])        #20x3
X = np.vstack((X1,X2,X3))
plt.plot(x11,x21,'ro',x12,x22,'bo',x13,x23,'go')
plt.show
```

Apply k-binary logistic regression: Pick a class as y=1 and others as y=0 then regress. Repeat.

```
classes = 3
alpha = 0.01
itera = 10000

for c in range(classes):
    Y = np.zeros((60,1))
    a = 20*c
    b = 20*(c+1)
    Y[a:b,:]=np.ones((20,1))            #pick class c for y = 1, other classes = 0

    W = np.random.random((3,1))                #1x3
    m = Y.shape[0]

    for i in range(itera):
        Z = np.dot(X, W)
```

```
        H = 1 / (1 + np.exp(-Z))
        L = -np.sum(Y*np.log(H)+(1-Y)*np.log(1-H))  #dont need to calculate in a loop
        dW = np.dot(X.T, (H - Y)) / m
        W = W - alpha*dW

    y1 = np.array([np.min(X[:,1]),np.max(X[:,1])])          #plot varies between 0 and 15
    y2 = -((W[0,0] + W[1,0]*y1)/W[2,0])
    plt.plot(X[:,1],X[:,2],'go',X[a:b,1],X[a:b,2],'ro')
    plt.plot(y1,y2,'--')
    plt.show
    plt.figure()
```

Softmax Regression

Define 3 classes:

```
x11 = np.random.normal(10, 2, 20).reshape(20,1)     #mu = 10, sig=2, samples=20
x21 = np.random.normal(5, 2, 20).reshape(20,1)
x12 = np.random.normal(5, 3, 20).reshape(20,1)
x22 = np.random.normal(10, 3, 20).reshape(20,1)
x13 = np.random.normal(10, 2, 20).reshape(20,1) #center around 10,15
x23 = np.random.normal(15, 3, 20).reshape(20,1)

X1 = np.hstack((x11,x21))           #20x2 for class 1
X2 = np.hstack((x12,x22))           #20x2 for class 2
X3 = np.hstack((x13,x23))           #20x2 for class 2

X = np.vstack ((X1,X2,X3))                      #combine all x values
Y = np.vstack ((np.zeros((20,1)),np.ones((20,1)),2*np.ones((20,1))))    #classes 1,2,3
```

Use scikit-learn to apply softmax regression:

```
from sklearn.linear_model import LogisticRegression

softmax_reg = LogisticRegression(multi_class="multinomial",solver="lbfgs", C=10)
logreg = softmax_reg.fit(X, Y.reshape(60,))

logreg.fit(X, Y.reshape(60,))

# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
h = .02  # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.plot(X[0:20, 0], X[0:20, 1],'ro',X[20:40, 0], X[20:40, 1],'bo',X[40:60, 0], X[40:60, 1],'go')
plt.show()
```