blghhrvkh

December 2, 2023

1 Worksheet 22

Name: Ivanna M. UID: U69469925

1.0.1 Topics

- Gradient Descent

1.1 Gradient Descent

Recall in Linear Regression we are trying to find the line

$$y = X\beta$$

that minimizes the sum of square distances between the predicted `y` and the `y` we observed in our dataset:

$$\mathcal{L}(\ ) = \|\mathbf{y} - X\ \|^2$$

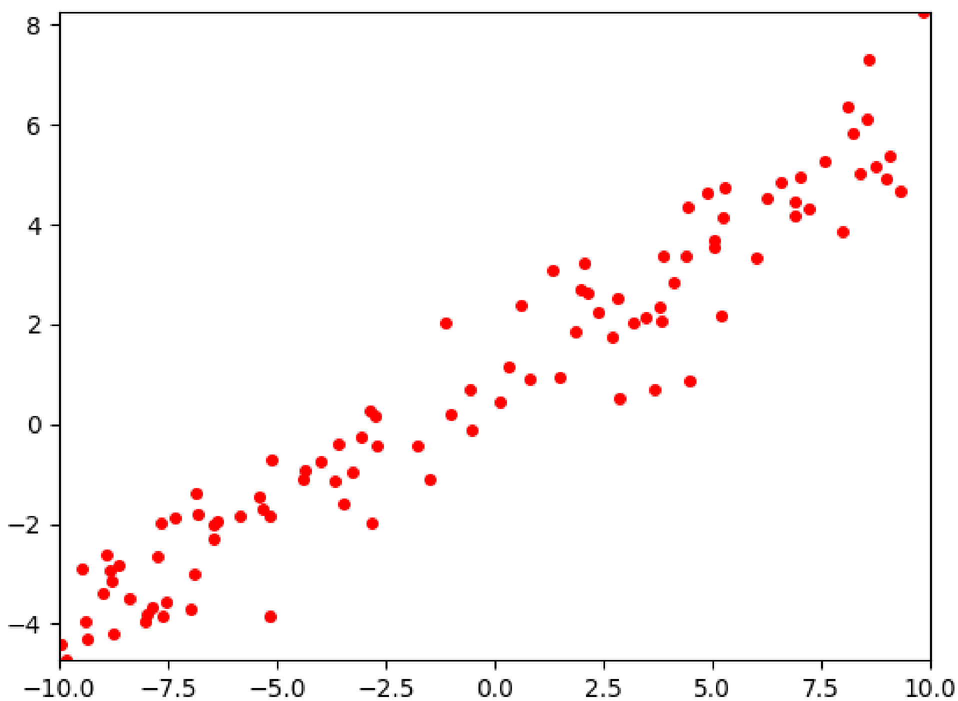
We were able to find a global minimum to this loss function but we will try to apply gradient descent to find that same solution.

a) Implement the `loss` function to complete the code and plot the loss as a function of beta.

[23]:
```
from google.colab import output
output.enable_custom_widget_manager()
```

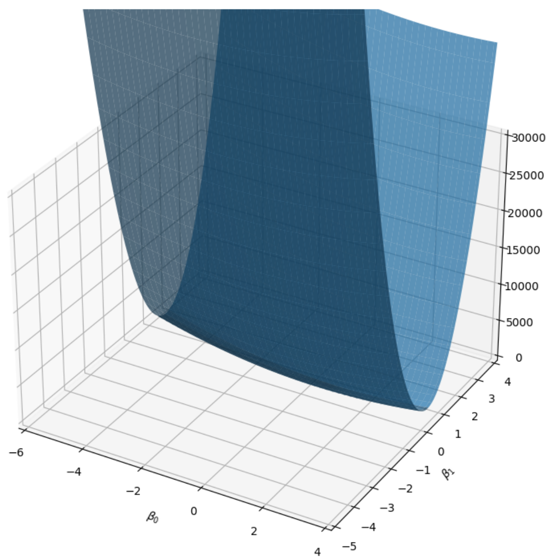
[24]:
```
%matplotlib widget
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt

beta = np.array([ 1 , .5 ])
xlin = -10.0 + 20.0 * np.random.random(100)
X = np.column_stack([np.ones((len(xlin), 1)), xlin])
y = beta[0]+(beta[1]*xlin)+np.random.randn(100)

fig, ax = plt.subplots()
ax.plot(xlin, y,'ro',markersize=4)
ax.set_xlim(-10, 10)
```

```
ax.set_ylim(min(y), max(y))
plt.show()
```



[25]:
```
b0 = np.arange(-5, 4, 0.1)
b1 = np.arange(-5, 4, 0.1)
b0, b1 = np.meshgrid(b0, b1)

def loss(X, y, beta):
    return np.sum((y - np.dot(X,beta))**2)

def get_cost(B0, B1):
    res = []
    for b0, b1 in zip(B0, B1):
        line = []
        for i in range(len(b0)):
            beta = np.array([b0[i], b1[i]])
            line.append(loss(X, y, beta))
        res.append(line)
    return np.array(res)
```

```python
cost = get_cost(b0, b1)

# Creating figure
fig = plt.figure(figsize =(14, 9))
ax = plt.axes(projection ='3d')
ax.set_xlim(-6, 4)
ax.set_xlabel(r'$\beta_0$')
ax.set_ylabel(r'$\beta_1$')
ax.set_ylim(-5, 4)
ax.set_zlim(0, 30000)

# Creating plot
ax.plot_surface(b0, b1, cost, alpha=.7)

# show plot
plt.show()
```



Since the loss is

$$\mathcal{L}( ) = \|\mathbf{y} - X \|^2 = \beta^T X^T X\beta - 2 \, {}^T X^T \mathbf{y} + \mathbf{y}^T \mathbf{y}$$

The gradient is

$$\nabla_\beta \mathcal{L}( ) = 2X^T X\beta - 2X^T \mathbf{y}$$

b) Implement the gradient function below and complete the gradient descent algorithm

[26]:
```
import numpy as np
from PIL import Image as im
import matplotlib.pyplot as plt

TEMPFILE = "temp.png"

def snap(betas, losses):
    # Creating figure
    fig = plt.figure(figsize =(14, 9))
    ax = plt.axes(projection ='3d')
    ax.view_init(20, -20)
    ax.set_xlim(-5, 4)
    ax.set_xlabel(r'$\beta_0$')
    ax.set_ylabel(r'$\beta_1$')
    ax.set_ylim(-5, 4)
    ax.set_zlim(0, 30000)

    # Creating plot
    ax.plot_surface(b0, b1, cost, color='b', alpha=.7)
    ax.plot(np.array(betas)[:,0], np.array(betas)[:,1], losses, 'o-', c='r',␣
↪markersize=10, zorder=10)
    fig.savefig(TEMPFILE)
    plt.close()
    return im.fromarray(np.asarray(im.open(TEMPFILE)))


def gradient(X, y, beta):
    return -2 * np.dot(X.T, y - np.dot(X, beta))



def gradient_descent(X, y, beta_hat, learning_rate, epochs, images):
    losses = [loss(X, y, beta_hat)]
    betas = [beta_hat]

    for _ in range(epochs):
        images.append(snap(betas, losses))
        beta_hat = betas[-1] - learning_rate * gradient(X, y, betas[-1])

        losses.append(loss(X, y, beta_hat))
        betas.append(beta_hat)

    return np.array(betas), np.array(losses)
```

```python
beta_start = np.array([-5, -2])
learning_rate = 0.0002 # try .0005
images = []
betas, losses = gradient_descent(X, y, beta_start, learning_rate, 10, images)

images[0].save(
    'gd.gif',
    optimize=False,
    save_all=True,
    append_images=images[1:],
    loop=0,
    duration=500
)
```

Support for third party widgets will remain active for the duration of the session. To disable support:

c) Use the code above to create an animation of the linear model learned at every epoch.

[27]:
```python
def snap_model(beta):
    xplot = np.linspace(-10,10,50)
    yestplot = beta[0] + beta[1] * xplot
    fig, ax = plt.subplots()
    ax.plot(xplot, yestplot,'b-',lw=2)
    ax.plot(xlin, y,'ro',markersize=4)
    ax.set_xlim(-10, 10)
    ax.set_ylim(min(y), max(y))
    fig.savefig(TEMPFILE)
    plt.close()
    return im.fromarray(np.asarray(im.open(TEMPFILE)))


def gradient_descent(X, y, beta_hat, learning_rate, epochs, images):
    losses = [loss(X, y, beta_hat)]
    betas = [beta_hat]

    for _ in range(epochs):
        images.append(snap_model(betas[-1]))
        beta_hat = beta_hat - learning_rate * gradient(X, y, beta_hat)

        losses.append(loss(X, y, beta_hat))
        betas.append(beta_hat)

    return np.array(betas), np.array(losses)


images = []
betas, losses = gradient_descent(X, y, beta_start, learning_rate, 100, images)
```

```
images[0].save(
    'model.gif',
    optimize=False,
    save_all=True,
    append_images=images[1:],
    loop=0,
    duration=200
)
```

In logistic regression, the `loss` is the negative log-likelihood

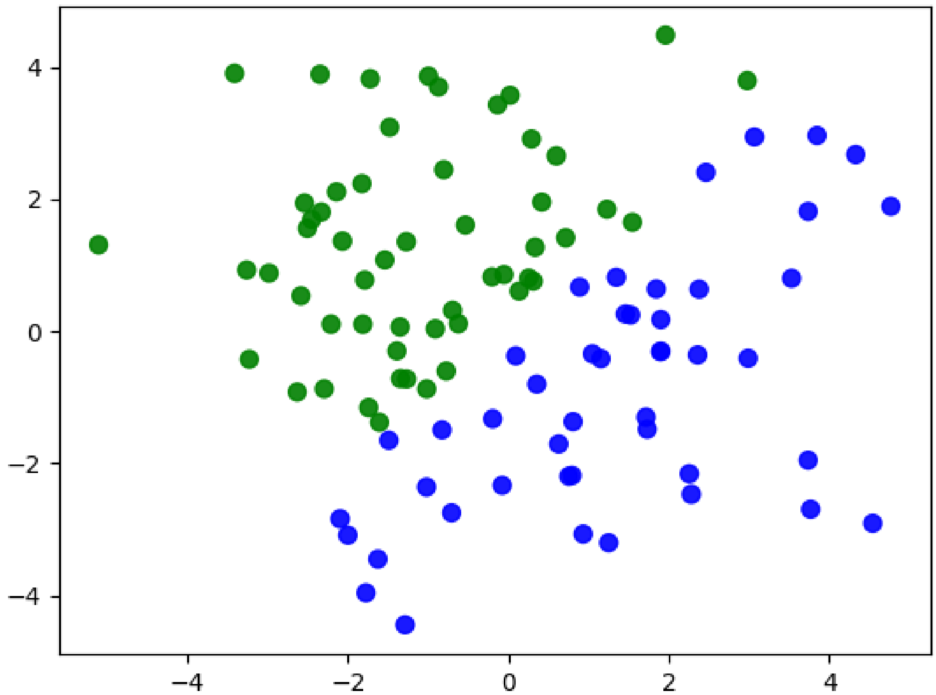
$$l(\ ) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\sigma(x_i\beta)) + (1-y_i)\log(1-\sigma(x_i\beta))$$

the gradient of which is:

$$\nabla_\beta l(\ ) = \frac{1}{N}\sum_{i=1}^{N} x_i(y_i - \sigma(x_i\beta))$$

d) Plot the loss as a function of b.

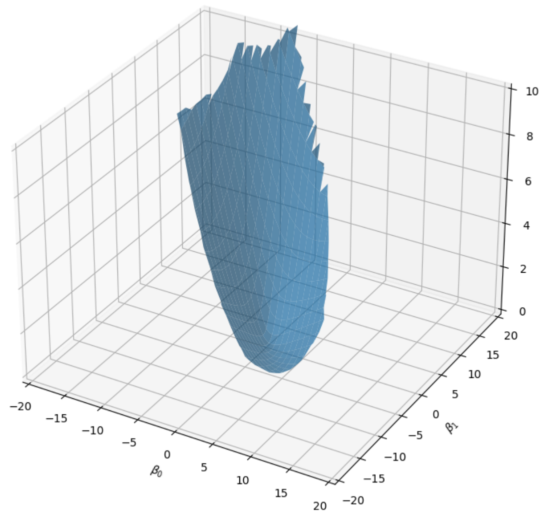
[28]:
```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
import sklearn.datasets as datasets

centers = [[0, 0]]
t, _ = datasets.make_blobs(n_samples=100, centers=centers, cluster_std=2, random_state=0)

# LINE
def generate_line_data():
    # create some space between the classes
    X = t
    Y = np.array([1 if x[0] - x[1] >= 0 else 0 for x in X])
    return X, Y

X, y = generate_line_data()

cs = np.array([x for x in 'gb'])
fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], color=cs[y].tolist(), s=50, alpha=0.9)
plt.show()
```




```
[29]: b0 = np.arange(-20, 20, 0.1)
b1 = np.arange(-20, 20, 0.1)
b0, b1 = np.meshgrid(b0, b1)


def sigmoid(x):
    e = np.exp(x)
    return e / (1 + e)


def loss(X, y, beta):
    yhat = sigmoid(np.dot(X, beta))
    return -np.mean(y * np.log(yhat) + (1 - y) * np.log(1 - yhat))

def get_cost(B0, B1):
    res = []
    for b0, b1 in zip(B0, B1):
        line = []
        for i in range(len(b0)):
            beta = np.array([b0[i], b1[i]])
            line.append(loss(X, y, beta))
```

```python
        res.append(line)
    return np.array(res)

cost = get_cost(b0, b1)

# Creating figure
fig = plt.figure(figsize =(14, 9))
ax = plt.axes(projection ='3d')
ax.set_xlim(-20, 20)
ax.set_xlabel(r'$\beta_0$')
ax.set_ylabel(r'$\beta_1$')
ax.set_ylim(-20, 20)
ax.set_zlim(0, 10)

# Creating plot
ax.plot_surface(b0, b1, cost, alpha=.7)

# show plot
plt.show()
```

```
<ipython-input-29-e7a785ac5076>:13: RuntimeWarning: divide by zero encountered
in log
  return -np.mean(y * np.log(yhat) + (1 - y) * np.log(1 - yhat))
<ipython-input-29-e7a785ac5076>:13: RuntimeWarning: invalid value encountered in
multiply
  return -np.mean(y * np.log(yhat) + (1 - y) * np.log(1 - yhat))
/usr/local/lib/python3.10/dist-packages/mpl_toolkits/mplot3d/art3d.py:1180:
RuntimeWarning: invalid value encountered in subtract
  v1[poly_i, :] = ps[i1, :] - ps[i2, :]
/usr/local/lib/python3.10/dist-packages/mpl_toolkits/mplot3d/art3d.py:1181:
RuntimeWarning: invalid value encountered in subtract
  v2[poly_i, :] = ps[i2, :] - ps[i3, :]
/usr/local/lib/python3.10/dist-packages/numpy/core/numeric.py:1665:
RuntimeWarning: invalid value encountered in subtract
  cp1 -= tmp
/usr/local/lib/python3.10/dist-packages/mpl_toolkits/mplot3d/proj3d.py:180:
RuntimeWarning: invalid value encountered in divide
  txs, tys, tzs = vecw[0]/w, vecw[1]/w, vecw[2]/w
```



e) Plot the loss at each iteration of the gradient descent algorithm.

[30]:
```
import numpy as np
from PIL import Image as im
import matplotlib.pyplot as plt

TEMPFILE = "temp.png"

def snap(betas, losses):
    # Creating figure
    fig = plt.figure(figsize =(14, 9))
    ax = plt.axes(projection ='3d')
    ax.view_init(10, 10)
    ax.set_xlabel(r'$\beta_0$')
    ax.set_ylabel(r'$\beta_1$')
    ax.set_ylim(-20, 20)
    ax.set_zlim(0, 10)

    # Creating plot
    ax.plot_surface(b0, b1, cost, color='b', alpha=.7)
    ax.plot(np.array(betas)[:,0], np.array(betas)[:,1], losses, 'o-', c='r', markersize=10, zorder=10)
    fig.savefig(TEMPFILE)
    plt.close()
```

```
    return im.fromarray(np.asarray(im.open(TEMPFILE)))


def gradient(X, y, beta):
    return -np.dot(X.T, y - sigmoid(np.dot(X, beta))) / X.shape[0]


def gradient_descent(X, y, beta_hat, learning_rate, epochs, images):
    losses = [loss(X, y, beta_hat)]
    betas = [beta_hat]

    for _ in range(epochs):
        images.append(snap(betas, losses))
        beta_hat = beta_hat - learning_rate * gradient(X, y, beta_hat)

        losses.append(loss(X, y, beta_hat))
        betas.append(beta_hat)

    return np.array(betas), np.array(losses)


beta_start = np.array([-5, -2])
learning_rate = 0.1
images = []
betas, losses = gradient_descent(X, y, beta_start, learning_rate, 10, images)

images[0].save(
    'gd_logit.gif',
    optimize=False,
    save_all=True,
    append_images=images[1:],
    loop=0,
    duration=500
)
```

f) Create an animation of the logistic regression fit at every epoch.

[31]:
```
def snap_model(beta, X, y):
    xplot = np.linspace(-10, 10, 50)
    yestplot = sigmoid(beta[0] + beta[1] * xplot)

    fig, ax = plt.subplots()
    ax.plot(xplot, yestplot, 'b-', lw=2)
    ax.plot(X[:, 1], y, 'ro', markersize=4)
    ax.set_xlim(-10, 10)
    ax.set_ylim(-0.1, 1.1)
```

```
    fig.savefig(TEMPFILE)
    plt.close()
    return im.fromarray(np.asarray(im.open(TEMPFILE)))


def gradient_descent(X, y, beta_hat, learning_rate, epochs, images):
    losses = [loss(X, y, beta_hat)]
    betas = [beta_hat]

    for _ in range(epochs):
        images.append(snap_model(betas[-1], X, y))
        beta_hat = beta_hat - learning_rate * gradient(X, y, beta_hat)

        losses.append(loss(X, y, beta_hat))
        betas.append(beta_hat)

    return np.array(betas), np.array(losses)

images = []

betas, losses = gradient_descent(X, y, beta_start, learning_rate, 100, images)

images[0].save(
    'model_logit.gif',
    optimize=False,
    save_all=True,
    append_images=images[1:],
    loop=0,
    duration=200
)
```

g) Modify the above code to evaluate the gradient on a random batch of the data. Overlay the true loss curve and the approximation of the loss in your animation.

[36]:
```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image as im
import io

def snap_model(beta, X, y, losses, epoch):
    xplot = np.linspace(-10, 10, 50)
    yestplot = sigmoid(beta[0] + beta[1] * xplot)

    fig, ax = plt.subplots()
    ax.plot(xplot, yestplot, 'b-', lw=2)
    ax.plot(X[:, 1], y, 'ro', markersize=4)
    ax.set_xlim(-10, 10)
```

```
    ax.set_ylim(-0.1, 1.1)

    # Plotting the loss curve
    epochs = np.arange(epoch + 1)
    ax2 = ax.twinx()
    ax2.plot(epochs, losses, 'g-', lw=1)
    ax2.set_ylabel('Loss', color='g')

    buf = io.BytesIO()
    plt.savefig(buf, format='png')
    plt.close(fig)
    buf.seek(0)
    return im.open(buf)

def gradient_descent(X, y, beta_hat, learning_rate, epochs, images, batch_size):
    losses = [loss(X, y, beta_hat)]
    betas = [beta_hat]

    for epoch in range(epochs):
        # Random batch selection
        idx = np.random.choice(X.shape[0], batch_size, replace=False)
        X_batch = X[idx]
        y_batch = y[idx]

        images.append(snap_model(betas[-1], X, y, losses, epoch))
        beta_hat = beta_hat - learning_rate * gradient(X_batch, y_batch,␣
 ↪beta_hat)

        losses.append(loss(X, y, beta_hat))
        betas.append(beta_hat)

    return np.array(betas), np.array(losses)

images = []
beta_start = np.random.randn(2)
learning_rate = 0.01
batch_size = 32

betas, losses = gradient_descent(X, y, beta_start, learning_rate, 100, images,␣
 ↪batch_size)

images[0].save(
    'model_logit.gif',
    optimize=False,
    save_all=True,
    append_images=images[1:],
    loop=0,
```

```
    duration=200
)
```

h) Below is a sandox where you can get intuition about how to tune gradient descent parameters:

[37]:
```python
import numpy as np
from PIL import Image as im
import matplotlib.pyplot as plt

TEMPFILE = "temp.png"

def snap(x, y, pts, losses, grad):
    fig = plt.figure(figsize =(14, 9))
    ax = plt.axes(projection ='3d')
    ax.view_init(20, -20)
    ax.plot_surface(x, y, loss(np.array([x, y])), color='r', alpha=.4)
    ax.plot(np.array(pts)[:,0], np.array(pts)[:,1], losses, 'o-', c='b',␣
 ↪markersize=10, zorder=10)
    ax.plot(np.array(pts)[-1,0], np.array(pts)[-1,1], -1, 'o-', c='b', alpha=.
 ↪5, markersize=7, zorder=10)

    # Plot Gradient Vector
    X, Y, Z = [pts[-1][0]], [pts[-1][1]], [-1]
    U, V, W = [-grad[0]], [-grad[1]], [0]
    ax.quiver(X, Y, Z, U, V, W, color='g')
    fig.savefig(TEMPFILE)
    plt.close()
    return im.fromarray(np.asarray(im.open(TEMPFILE)))

def loss(x):
    return np.sin(sum(x**2)) # change this

def gradient(x):
    return 2 * x * np.cos(sum(x**2)) # change this

def gradient_descent(x, y, init, learning_rate, epochs):
    images, losses, pts = [], [loss(init)], [init]
    for _ in range(epochs):
        grad = gradient(init)
        images.append(snap(x, y, pts, losses, grad))
        init = init - learning_rate * grad
        losses.append(loss(init))
        pts.append(init)
    return images

init = np.array([-.5, -.5]) # change this
learning_rate = 1.394 # change this
```

```python
x, y = np.meshgrid(np.arange(-2, 2, 0.1), np.arange(-2, 2, 0.1)) # change this
images = gradient_descent(x, y, init, learning_rate, 12)

images[0].save(
    'gradient_descent.gif',
    optimize=False,
    save_all=True,
    append_images=images[1:],
    loop=0,
    duration=500
)
```