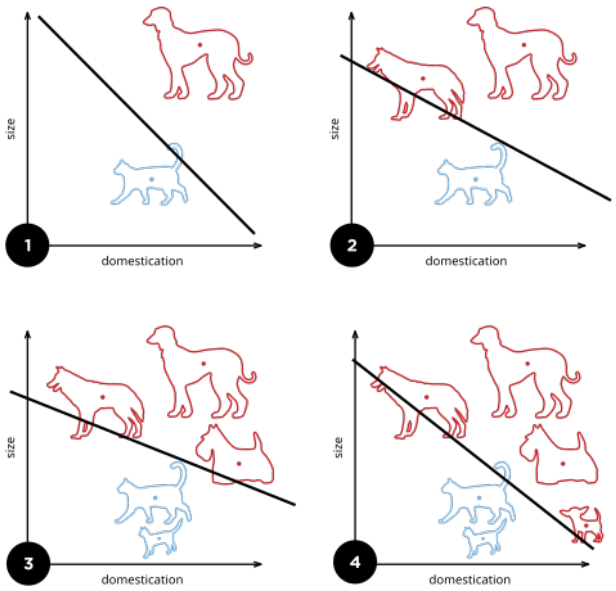
Perceptron

Vikas Dhiman

Thursday 25th September, 2025

---

1 Perceptron



1.1 Optimization for classification

1.1.1 Dataset

Let the dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i \in R^d$ is the feature vector and $y_i \in \{-1, +1\}$ is the binary class label.

1.1.2 Model

We encode the prediction model as

$$\hat{y}_i = f(\mathbf{x}_i; \mathbf{m}, c) = \mathbf{m}^\top \mathbf{x}_i + c, \tag{1}$$

where $\mathbf{m} \in R^d$ and $c \in R$.

We say that the prediction is of class -1, if $\hat{y}_i < 0$ and +1 if $\hat{y}_i > 0$.

1.1.3 Loss function

$$l(y_i, \hat{y}_i; \mathbf{m}, c) = \begin{cases} 0 & \text{if } y_i\hat{y}_i > 0 \text{ or the sign of } y_i \text{ and } \hat{y}_i \text{ is the same} \\ |\hat{y}_i| & \text{if } y_i\hat{y}_i \leq 0 \text{ or the sign of } y_i \text{ and } \hat{y}_i \text{ is different} \end{cases} \tag{2}$$

$$l(y_i, \hat{y}_i; \mathbf{m}, c) = \begin{cases} 0 & \text{if } y_i\hat{y}_i > 0 \\ -y_i\hat{y}_i & \text{if } y_i\hat{y}_i \leq 0 \end{cases} \tag{3}$$

Let

$$\mathbf{w} = \begin{bmatrix} \mathbf{m} \\ c \end{bmatrix} \in R^{d+1}. \tag{4}$$

Then

$$\hat{y}_i = f(\mathbf{x}_i; \mathbf{m}, c) = \mathbf{m}^\top \mathbf{x}_i + c = \begin{bmatrix} \mathbf{x}_i^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{m} \\ c \end{bmatrix} = \begin{bmatrix} \mathbf{x}_i^\top & 1 \end{bmatrix} \mathbf{w} \tag{5}$$

We can rewrite loss interms of $\mathbf{w}$,

$$l(y_i, \hat{y}_i; \mathbf{w}) = \begin{cases} 0 & \text{if } y_i\hat{y}_i > 0 \\ -y_i\mathbf{w}^\top \begin{bmatrix} \mathbf{x}_i \\ 1 \end{bmatrix} & \text{if } y_i\hat{y}_i \leq 0 \end{cases} \tag{6}$$

Optimization We want to find the parameters $\mathbf{w} \in R^{d+1}$ that minimize the loss over the entire dataset,

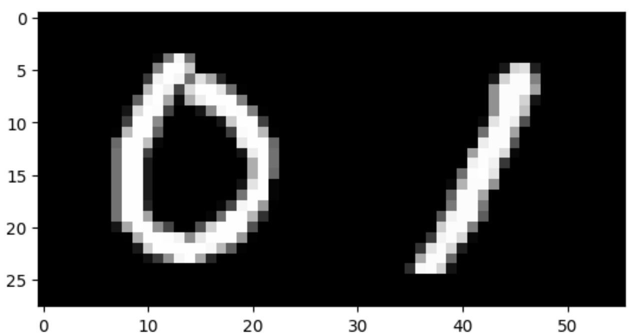
$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} l(y_i, \hat{y}_i; \mathbf{w}) \tag{7}$$

To perform gradient descent on the loss function we need the gradient,

$$\nabla_{\mathbf{w}} \frac{1}{n} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} l(y_i, \hat{y}_i; \mathbf{w}) = \frac{1}{n} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} \nabla_{\mathbf{w}} l(y_i, \hat{y}_i; \mathbf{w}) = \frac{1}{n} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} \begin{cases} 0 & \text{if } y_i\hat{y}_i > 0 \\ -y_i \begin{bmatrix} \mathbf{x}_i \\ 1 \end{bmatrix} & \text{if } y_i\hat{y}_i \leq 0 \end{cases} \tag{8}$$

```
# Download MNIST dataset
!mkdir -p data
!F=train -images -idx3 -ubyte && cd data && \
    [ ! -f $F ] && \
    wget http://yann.lecun.com/exdb/mnist/$F.gz  && \
    gunzip $F.gz
!F=train -labels -idx1 -ubyte && cd data && \
    [ ! -f $F ] && \
    wget http://yann.lecun.com/exdb/mnist/$F.gz  && \
    gunzip $F.gz

# Load MNIST dataset from uint8 byte files
import struct
import numpy as np

# Ref:https://github.com/sorki/python -mnist/blob/master/mnist/loader.py
def mnist_read_labels(fname='data/train -labels -idx1 -ubyte'):
    with open(fname, 'rb') as file:
        # The file starts with 4 byte 2 unsigned ints
        magic, size = struct.unpack('>II', file.read(8))
        assert magic == 2049
        labels = np.frombuffer(file.read(), dtype='u1')
        return labels

# Ref:https://github.com/sorki/python -mnist/blob/master/mnist/loader.py
def mnist_read_images(fname='data/train -images -idx3 -ubyte'):
    with open(fname, 'rb') as file:
        # The file starts with 4 byte 4 unsigned ints
        magic, size, rows, cols = struct.unpack('>IIII', file.read(16))
        assert magic == 2051
        image_data = np.frombuffer(file.read(), dtype='u1')
        images = image_data.reshape(size, rows, cols)
        return images

# Visualize the dataset
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import matplotlib as mpl
mpl.rc('animation', html='jshtml')
train_images = mnist_read_images('data/train -images -idx3 -ubyte')
labels = mnist_read_labels('data/train -labels -idx1 -ubyte')
zero_images = train_images[labels==0, ...] # Filter by label == 0
one_images = train_images[labels==1, ...] # Filter by label == 1

fig, ax = plt.subplots()
```

```
# ims is a list of lists, each row is a list of artists to draw in the
# current frame; here we are just animating one artist, the image, in
# each frame
ims = [[ax.imshow(np.hstack((zero_images[i], one_images[i])), animated=True, cmap='gray', vm
    for i in range(60)]
zero_images_anim = animation.ArtistAnimation(fig, ims, interval=50, blit=True,
                                repeat_delay=1000, repeat=False)
zero_images_anim
```



1.2 Images as arrays

```
train_images.shape
```

```
(60000, 28, 28)
```

```
img1 = train_images[0]
img1
```

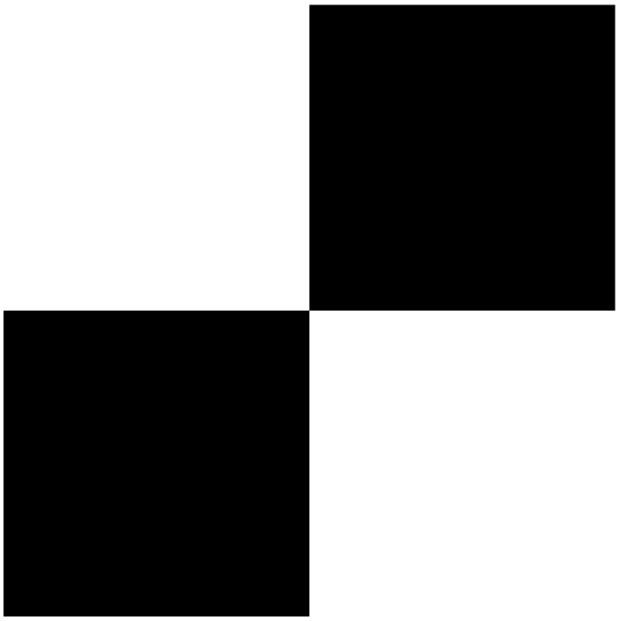
```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
```

```
  18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
 253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253,
 253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,  18, 219, 253, 253, 253, 253,
 253, 198, 182, 247, 241,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253, 253,
 205,  11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1, 154, 253,
  90,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 139, 253,
 190,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190,
 253,  70,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  35,
 241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,  45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,  16,  93, 252, 253, 187,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0, 249, 253, 249,  64,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,  46, 130, 183, 253, 253, 207,   2,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39,
 148, 229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114, 221,
 253, 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,   0,
```

```
        0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  23,  66, 213, 253, 253,
        253, 253, 198,  81,   2,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0],
       [  0,   0,   0,   0,   0,   0,  18, 171, 219, 253, 253, 253, 253,
        195,  80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0],
       [  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,
         11,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0],
       [  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   0,
        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        0,   0]], dtype=uint8)
```

```
## Visualizing matrices
fig, ax = plt.subplots()
ax.axis('off')
ax.imshow([[1, 0],
           [0, 1]], cmap='gray')
# ax.imshow(np.random.rand(28, 28), cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x78912f56ae60>
```



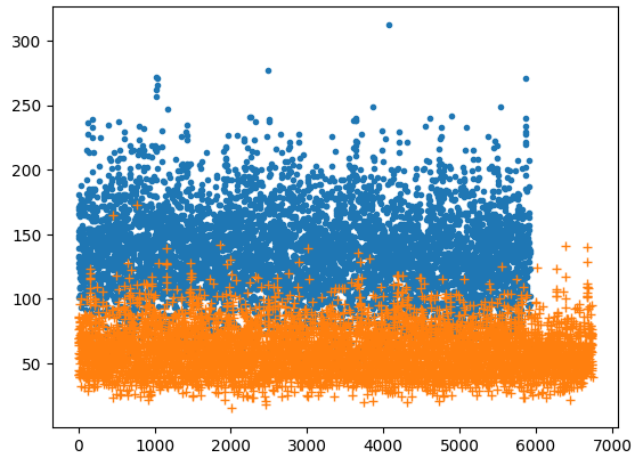
zero_images_anim

2 What is a feature

Any property of data sample that helps with the task.

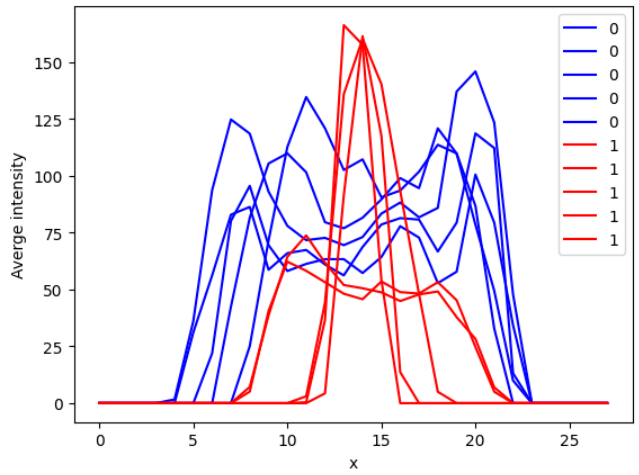
```
def feature_n_pxls(imgs):
    n, *shape = imgs.shape
    return np.sum(imgs[:, :, :].reshape(n, -1) > 128, axis=1)

n_pxls_zero_images = feature_n_pxls(zero_images)
n_pxls_one_images = feature_n_pxls(one_images)
fig, ax = plt.subplots()
ax.plot(n_pxls_zero_images, '.')
ax.plot(n_pxls_one_images, '+')
```

```
[<matplotlib.lines.Line2D at 0x789123bba680>]
```




```
fig, ax = plt.subplots()
for i in range(5):
    ax.plot(zero_images[i].mean(axis=0), 'b -', label='0')
for i in range(5):
    ax.plot(one_images[i].mean(axis=0), 'r -', label='1')
ax.legend()
ax.set_xlabel('x')
ax.set_ylabel('Averge intensity')
```

```
Text(0, 0.5, 'Averge intensity')
```



We will compute the intensity weighted variance of image along the image rows so that we can take the variance along the rows as a measure fo the spread along the x axis.

Let the `img` be denoted as an array $I \in R^{H \times W}$ where H is the height and W is the width of the array.

Let's first compute the average intensity along the columns and call that `wts` in the python program and $\mathbf{w}$ in maths,

$$w(c) = \frac{1}{H} \sum_{r=1}^{H} I(r, c) \tag{9}$$

$$\mathbf{w} = \begin{bmatrix} w(c=1) & w(c=2) & \dots & w(c=W) \end{bmatrix} \in R^{1 \times W} \tag{10}$$

For the first image in `zero_images`, computing the `wts` vector is one line of code in numpy,

```
wts = zero_images[0].mean(axis=0)
wts
```

```
array([  0.        ,   0.        ,   0.        ,   0.        ,
         0.        ,   0.        ,  21.96428571,  80.17857143,
        95.60714286,  69.5       ,  58.14285714,  61.25      ,
        63.35714286,  63.35714286,  57.25      ,  64.28571429,
        77.85714286,  72.82142857,  52.82142857,  57.82142857,
       100.57142857,  79.42857143,  34.32142857,   0.        ,
         0.        ,   0.        ,   0.        ,   0.        ])
```

Now we want to compute the mean and variance of the column variable c weighted by the weight vector $\mathbf{w}$

$$\mu_c(\mathbf{w}) = \frac{\sum_{c=1}^{W} cw(c)}{\sum_{c=1}^{W} w(c)} \tag{11}$$

```
cs = np.arange(wts.shape[0]) # cs = [1, ..., W]
mean = (cs * wts).sum() / wts.sum()
mean
```

```
np.float64(14.097571956906256)
```

Now we do the same with variance,

$$\sigma_c^2(\mathbf{w}) = \sum_{c=1}^{W} \frac{(c - \mu_c)^2 w(c)}{\sum_{x=1}^{W} w(c)} \tag{12}$$

```
var = ((cs - mean)**2 * wts).sum() / wts.sum()
var
```

```
np.float64(22.811061800377757)
```

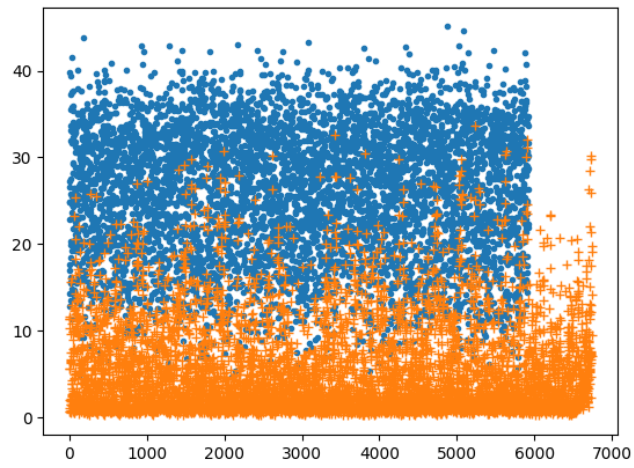
Put it all together in a function so that we can repeatedly run in on all the images,

```
def feature_y_var(img):
    wts = img.mean(axis=0)
    mean = (np.arange(wts.shape[0]) * wts).sum() / np.sum(wts)
    var = ((np.arange(wts.shape[0]) - mean)**2 * wts).sum() / np.sum(wts)
    return var

feature_y_var(zero_images[0]), feature_y_var(one_images[0])
```

```
(np.float64(22.811061800377757), np.float64(11.384958735403274))
```

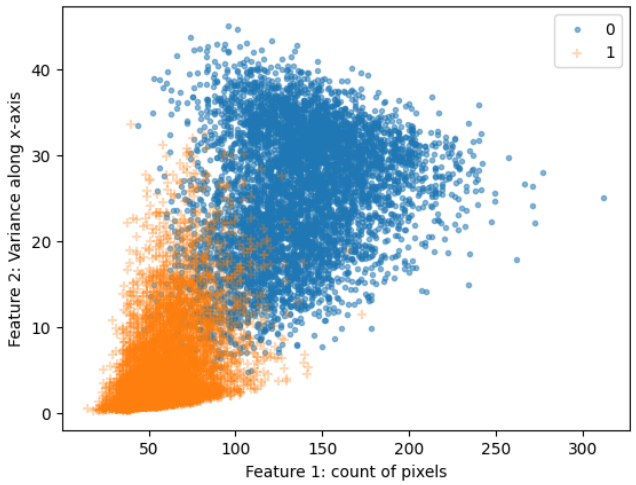
```
def feature_y_var(imgs):
    wts = imgs.mean(axis= -2)
    arange = np.arange(wts.shape[ -1])
    mean = (arange * wts).sum(axis= -1) / wts.sum(axis= -1)
    mean = mean[:, np.newaxis]
    var = ((arange - mean)**2 * wts).sum(axis= -1) / wts.sum(axis= -1)
    return var

fig, ax = plt.subplots()
ax.plot(feature_y_var(zero_images), '.')
ax.plot(feature_y_var(one_images), '+')
```

```
[<matplotlib.lines.Line2D at 0x789122120760>]
```

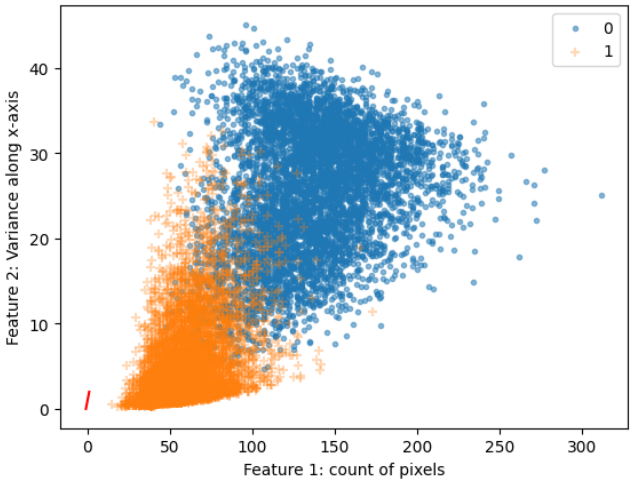



```
def features_extract(images):
    return np.vstack((feature_n_pxls(images),
                      feature_y_var(images))).T
zero_features = features_extract(zero_images)
one_features = features_extract(one_images)
```

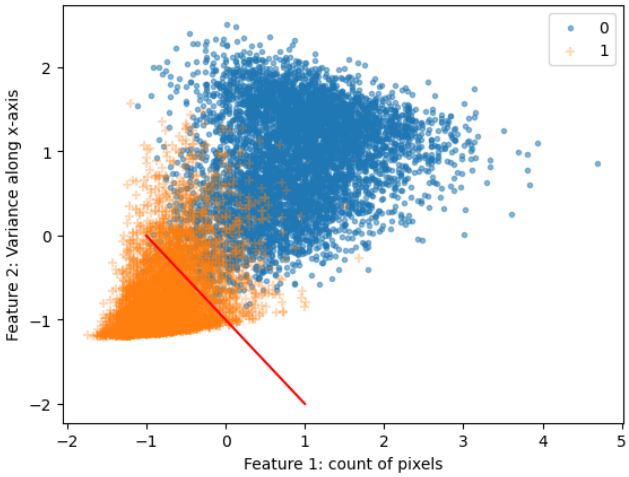
```
def draw_features(ax, zero_features, one_features):
    zf = ax.scatter(zero_features[:, 0], zero_features[:, 1], marker='.', label='0', alpha=0
    of = ax.scatter(one_features[:, 0], one_features[:, 1], marker='+', label='1', alpha=0.3
    ax.legend()
    ax.set_xlabel('Feature 1: count of pixels')
    ax.set_ylabel('Feature 2: Variance along x -axis')
    return [zf, of] # return list of artists

fig, ax = plt.subplots()
draw_features(ax, zero_features, one_features)
plt.show()
```




```
bfw = np.ones(3)
fig, ax = plt.subplots()
draw_features(ax, zero_features, one_features)
x = np.linspace( -1, 1, 21)
ax.plot(x, (x*bfw[0] + bfw[2])/bfw[1], 'r -')

[<matplotlib.lines.Line2D at 0x789123b2bbe0>]
```




```
bfw = np.ones(3)

Y = np.hstack((np.ones(zero_features.shape[0]), np.full(one_features.shape[0], -1.0)))
features = np.vstack((zero_features, one_features))
FEATURES_MEAN = features.mean(axis=0, keepdims=1)
FEATURES_STD = features.std(axis=0, keepdims=1)

def norm_features(features):
    return (features - FEATURES_MEAN) / FEATURES_STD

X = norm_features(features)

np.savez('zero_one_train_features.npz',
         mean=FEATURES_MEAN, std=FEATURES_STD,
         normed_features=X,
         labels=Y)

fig, ax = plt.subplots()
draw_features(ax, X[Y > 0, :], X[Y < 0, :])
x = np.linspace( -1, 1, 21)
ax.plot(x, -(x*bfw[0] + bfw[2])/bfw[1], 'r -')

[<matplotlib.lines.Line2D at 0x789121f2b010>]
```




```
X.shape, Y.shape
```

```
((12665, 2), (12665,))
```

Concatenate 1 to all X

$$\bar{X} = \begin{bmatrix} X_{n\times 2}, & \mathbf{1}_{n\times 1} \end{bmatrix} \in R^{n\times 3} \tag{13}$$

```
X_and_1 = np.hstack((X, np.ones((X.shape[0], 1))))
X_and_1.shape
```

```
(12665, 3)
```

Homework (Perceptron): Problem 1 (10 marks)

Implement a function `model` that takes the dataset inputs `X_and_1` $\bar{X} \in R^{n\times 3}$, the dataset labels, and the weight vector `bfw` $\mathbf{w} \in R^3$ and returns the $\hat{y}_i$ for all elements in the dataset,

$$\hat{y}_i = f(\mathbf{x}_i; \mathbf{w}) = \mathbf{w}^\top \begin{bmatrix} \mathbf{x}_i \\ 1 \end{bmatrix} \tag{14}$$

```
def model(X_and_1, bfw):
    """
    X_and_1.shape = (n, 3)
    bfw.shape = (3,)
    """
    # YOUR CODE HERE
    raise NotImplementedError()
    return Yhat # Yhat.shape = (n,)
```

You are given the implementation of the function `loss` that takes the dataset inputs $\bar{X} \in R^{n\times 3}$, the dataset labels, $Y \in \{0,1\}^n$ and the weight vector $\mathbf{w} \in R^3$ and returns the total loss for all elements in the dataset,

$$\frac{1}{n}\sum_{(x_i,y_i)\in\mathcal{D}} l(y_i,\hat{y}_i;\mathbf{w}) = \frac{1}{n}\sum_{(x_i,y_i)\in\mathcal{D}} \begin{cases} 0 & \text{if } y_i\hat{y}_i > 0 \\ -y_i\mathbf{w}^\top \begin{bmatrix}\mathbf{x}_i \\ 1\end{bmatrix} & \text{if } y_i\hat{y}_i \leq 0 \end{cases} \tag{15}$$

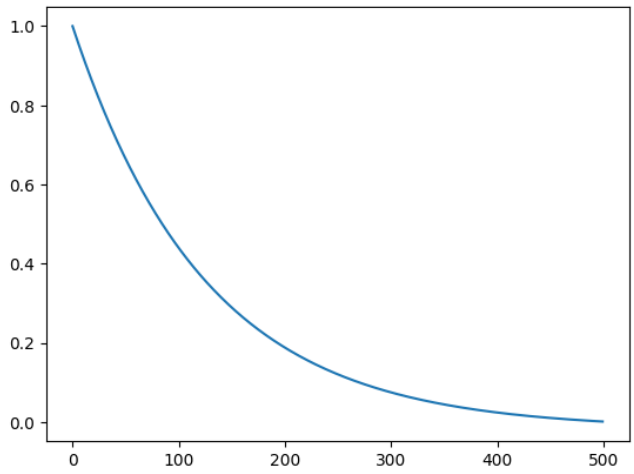
```
def loss(X_and_1, Y, bfw):
    """
    X_and_1.shape = (n, 3)
    Y.shape = (n,)
    bfw.shape = (3,)
    """
    Yhat = model(X_and_1, bfw)
    YYhat = Y * Yhat  # YYhat.shape = (n,)
    l_per_sample = np.where(YYhat > 0, 0, -YYhat) # l_per_sample.shape = (n,)
    l_mean = l_per_sample.mean(axis= -1) # l_per_sample.shape = (n,)
    return l_mean # l_mean.shape = (1,)
```

Homework (Perceptron): Problem 2 (10 marks)

Implement a function `grad_loss` that takes the dataset inputs `X_and_1` $\bar{X} \in R^{n\times 3}$, the dataset labels, $Y \in \{0,1\}^n$ and the weight vector $\mathbf{w} \in R^3$ and returns the total loss for all elements in the dataset,

$$\nabla_{\mathbf{w}}\left(\frac{1}{n}\sum_{(\mathbf{x}_i,y_i)\in\mathcal{D}} l(y_i,\hat{y}_i;\mathbf{w})\right) = \frac{1}{n}\sum_{(\mathbf{x}_i,y_i)\in\mathcal{D}} \begin{cases} 0 & \text{if } y_i\hat{y}_i > 0 \\ -y_i \begin{bmatrix}\mathbf{x}_i \\ 1\end{bmatrix} & \text{if } y_i\hat{y}_i \leq 0 \end{cases} \tag{16}$$

```
def grad_loss(X_and_1, Y, bfw):
    # YOUR CODE HERE
    raise NotImplementedError()
    return grad_L_total # grad_L_total.shape = (3,)

def lr_scheduler(t, lr0=1, lr_end=0.001, t0=0, t_end=500, curv=4):
    normalized = (np.exp( -curv*(t -t0)/(t_end -t0)) -1) / (np.exp( -curv) - 1)
    return (lr_end - lr0) * normalized + lr0

fig, ax = plt.subplots()
ax.plot(lr_scheduler(np.arange(500)))
lr_scheduler(500)

np.float64(0.0010000000000000009)
```



Homework (Perceptron): Problem 3 (10 marks)

Implement gradient descent to find $\mathbf{w}$ that minimizes $\nabla_{\mathbf{w}} \frac{1}{n} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} l(y_i, \hat{y}_i; \mathbf{w})$

Algorithm 9.3 Gradient descent method.

given a starting point $\mathbf{x} \in \mathrm{dom} f$.

repeat

1. Choose step size α_t
2. Update. $\mathbf{x} := \mathbf{x} - \alpha_t \nabla f(\mathbf{x})$

until stopping criterion is satisfied.

```
def train(X_and_1, Y, lr_scheduler = lr_scheduler, MAX_ITER=500, SMALLEST_GRAD_L_NORM=0.001)
    """
    X_and_1.shape is (n, 3)
    Y.shape is (n, )

    Write a function that returns

    bfw : the solution for bfw that you get after gradient descent
    list_of_bfws : the list of bfw at each iteration of gradient descent
    list_of_losses : the list of loss() values that you get at each iteration of gradient de
    """
    bfw = np.random.rand(3)*4 -2
    list_of_bfws = [bfw]
    list_of_losses = []

    grad_l = grad_loss(X_and_1, Y, bfw)
    list_of_losses.append(loss(X_and_1, Y, bfw))
    for t in range(MAX_ITER):
```

```
        if np.linalg.norm(grad_l) < SMALLEST_GRAD_L_NORM:
            break
        learning_rate = lr_scheduler(t)

        # 1. Compute grad_loss for current bfw
        # 2. Update bfw using gradient descent
        # YOUR CODE HERE
        raise NotImplementedError()
        list_of_bfws.append(bfw)
        list_of_losses.append(loss(X_and_1, Y, bfw))
    return bfw, list_of_bfws, list_of_losses

OPTIMAL_BFW, list_of_bfws, list_of_losses = train(X_and_1, Y)
OPTIMAL_BFW /= np.linalg.norm(OPTIMAL_BFW)
print("optimal loss", list_of_losses[ -2:])
print("optimal bfw", OPTIMAL_BFW)
fig, ax = plt.subplots()
ax.plot(list_of_losses)
ax.set_xlabel('t')
ax.set_ylabel('loss')
plt.show()
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
NotImplementedError                       Traceback (most recent call last)
Cell In[27], line 31
     28         list_of_losses.append(loss(X_and_1, Y, bfw))
     29     return bfw, list_of_bfws, list_of_losses
 - - -> 31 OPTIMAL_BFW, list_of_bfws, list_of_losses = train(X_and_1, Y)
     32 OPTIMAL_BFW /= np.linalg.norm(OPTIMAL_BFW)
     33 print("optimal loss", list_of_losses[ -2:])

Cell In[27], line 16, in train(X_and_1, Y, lr_scheduler, MAX_ITER, SMALLEST_GRAD_L_NORM)
     13 list_of_bfws = [bfw]
     14 list_of_losses = []
 - - -> 16 grad_l = grad_loss(X_and_1, Y, bfw)
     17 list_of_losses.append(loss(X_and_1, Y, bfw))
     18 for t in range(MAX_ITER):

Cell In[25], line 3, in grad_loss(X_and_1, Y, bfw)
      1 def grad_loss(X_and_1, Y, bfw):
      2     # YOUR CODE HERE
 - - - -> 3     raise NotImplementedError()
      4     return grad_L_total

NotImplementedError:
```

```
def project_bfw_to_mc(bfw):
```

```
    """
    bfw = [w1, w2, w3]

    Converts equation of line
        w1 x + w2 y + w3 = 0
    to
        - m x + y - c = 0

    return [m, c]
    """
    bfw_normalized = bfw / np.linalg.norm(bfw)
    assert np.abs(bfw_normalized[..., 1]) > 1e -4
    m = -bfw_normalized[0] / bfw_normalized[1]
    c = -bfw_normalized[2] / bfw_normalized[1]
    return m, c

def lift_mc_to_bfw(m, c):
    """
    Converts equation of line
        - m x + y - c = 0
    to
        w1 x + w2 y + w3 = 0

    returns bfw = [w1, w2, w3]
    """
    bfw_norm_sq = 1 +  m**2 + c**2
    bfw_norm = np.sqrt(bfw_norm_sq)
    w1 = - m / bfw_norm
    w2 = np.sqrt(1 - (m**2 + c**2)/bfw_norm_sq)
    w3 = - c / bfw_norm
    return np.concatenate((w1[..., None], w2[..., None], w3[..., None]),
                          axis= -1)

fig, axes = plt.subplots(2, 1, figsize=(5, 7.5))
class Anim:
    def __init__(self, fig, axes, X_and_1, Y):
        self.fig = fig
        self.ax = axes[0]
        self.ax1 = axes[1]
        self.fts = draw_features(self.ax, X_and_1[Y > 0, :2], X_and_1[Y < 0, :2])
        self.line, = self.ax.plot([], [], 'r -')

        m, c = np.meshgrid(np.linspace( -10, 0, 51), np.linspace( -1, 1, 51))
        bfw = lift_mc_to_bfw(m, c)
        totalloss = np.empty_like(m)
        for i in range(m.shape[0]):
```

```
            for j in range(m.shape[1]):
                totalloss[i, j] = loss(X_and_1, Y, bfw[i, j, :])

        self.ctr = self.ax1.contour(m, c, totalloss, 30, cmap='Blues_r')
        self.ax1.set_xlabel('m')
        self.ax1.set_ylabel('c')
        self.ax1.clabel(self.ctr, self.ctr.levels, inline=True, fontsize=6)
        self.m_hist = []
        self.c_hist = []
        self.line2, = self.ax1.plot([], [], 'r* -')


    def anim_init(self):
        return (self.line, self.line2)

    def update(self, bfw):
        x = np.linspace( -2, 2, 21)
        m, c = project_bfw_to_mc(bfw)
        self.line.set_data(x, x * m + c)
        self.m_hist.append(m)
        self.c_hist.append(c)
        self.line2.set_data(self.m_hist, self.c_hist)
        return self.line, self.line2

fig, axes = plt.subplots(2, 1, figsize=(5, 7.5))
a = Anim(fig, axes, X_and_1, Y)
animation.FuncAnimation(fig, a.update, frames=list_of_bfws[::5],
                        init_func=a.anim_init, blit=True, repeat=False)
```

```
# Download MNIST dataset
!mkdir -p data
!F=t10k -images -idx3 -ubyte && cd data && \
    [ ! -f $F ] && \
    wget http://yann.lecun.com/exdb/mnist/$F.gz  && \
    gunzip $F.gz
!F=t10k -labels -idx1 -ubyte && cd data && \
    [ ! -f $F ] && \
    wget http://yann.lecun.com/exdb/mnist/$F.gz  && \
    gunzip $F.gz

test_images = mnist_read_images('data/t10k -images -idx3 -ubyte')
test_labels = mnist_read_labels('data/t10k -labels -idx1 -ubyte')
zero_one_filter = (test_labels == 0) | (test_labels == 1)
zero_one_test_images = test_images[zero_one_filter, ...]
zero_one_test_labels = test_labels[zero_one_filter, ...]
```

```
np.savez('zero_one_PERCEPTRON_optmial_bfw.npz',
         optimal_bfw=OPTIMAL_BFW)

def returnclasslabel(test_imgs):
    Xtest = norm_features(features_extract(test_imgs))
    Xtest_and_1 = np.hstack((Xtest, np.ones((*Xtest.shape[: -1], 1))))
    bfw = OPTIMAL_BFW
    return np.where(
        model(Xtest_and_1, bfw) > 0,
        0,
        1)
zero_one_predicted_labels = returnclasslabel(zero_one_test_images)

accuracy = np.sum(zero_one_test_labels == zero_one_predicted_labels) / len(zero_one_test_lab
accuracy

positive_label = 1
negative_label = 0
TP = np.sum((zero_one_test_labels == positive_label) & (zero_one_predicted_labels == positiv
TP

TN =  np.sum((zero_one_test_labels == negative_label) & (zero_one_predicted_labels == negati
TN

FP = np.sum((zero_one_test_labels != positive_label) & (zero_one_predicted_labels == positiv
FP

FN = np.sum((zero_one_test_labels != negative_label) & (zero_one_predicted_labels == negativ
FN

# Confusion matrix
fig, ax = plt.subplots()
ax.imshow([[TN, FN],
           [FP, TP]])
ax.set_xlabel('predicted')
ax.set_ylabel('true')
ax.set_xticks([0, 1])
ax.set_yticks([0, 1])
ax.text(0, 0, 'TN = %.3f' % TN)
ax.text(1, 0, 'FN = %.3f' % FN, color='w')
ax.text(0, 1, 'FP = %.3f' % FP, color='w')
ax.text(1, 1, 'TP = %.3f' % TP)

PRECISION = TP / (TP + FP)
PRECISION

RECALL = TP / (TP + FN)
RECALL
```

```
fig, ax = plt.subplots()
artists = []
for i in range(60):
    artists.append(
        [ax.imshow(zero_one_test_images[i], animated=True, cmap='gray', vmin=0, vmax=255),
        ax.text(0, 2, 'The number is %d' % zero_one_predicted_labels[i], animated=True, colo
animation.ArtistAnimation(fig, artists, interval=50, blit=True,
                                repeat_delay=1000, repeat=False)
```