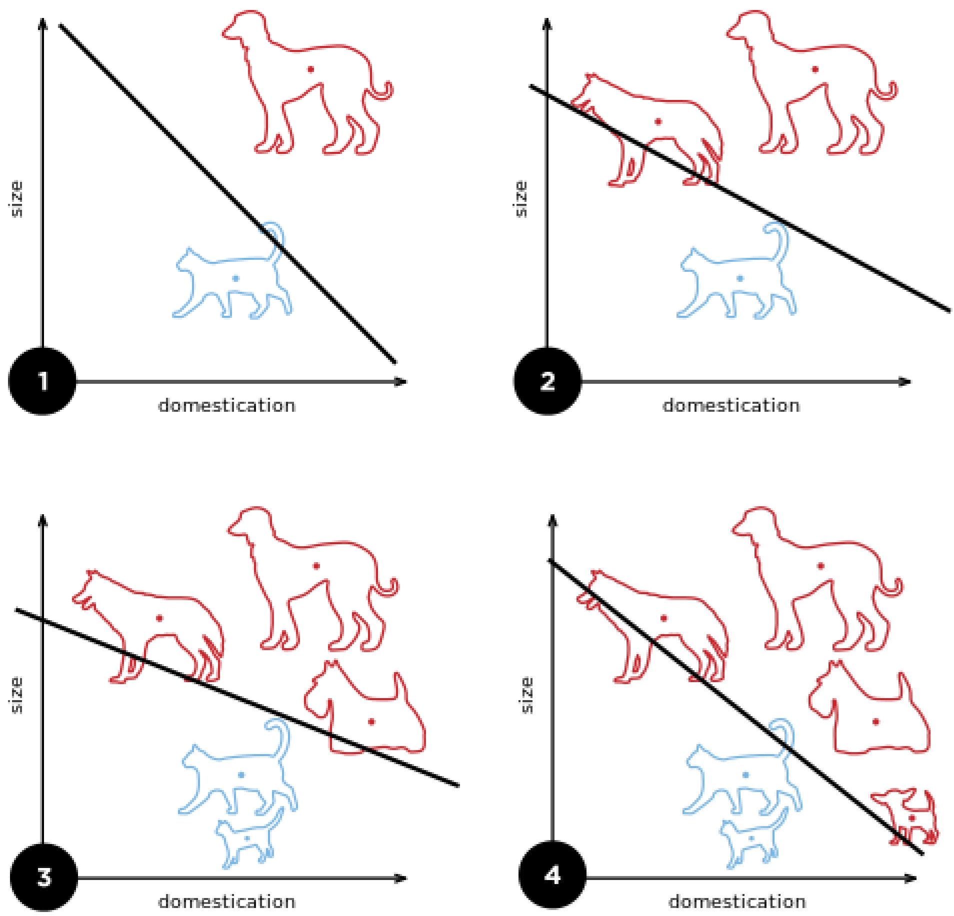
Perceptron



Optimization for classification

$$y = mx + c$$

$$e(y_i, x_i; m, c) = \begin{cases} 0 & \text{if } \operatorname{sign}(y_i - mx_i + c) = l_i \\ |y_i - mx_i + c| & \text{if } \operatorname{sign}(y_i - mx_i + c) \neq l_i \end{cases}$$

$$e(y_i, x_i; m, c) = \begin{cases} 0 & \text{if } \operatorname{sign}(y_i - mx_i + c) = l_i \\ |mx_i + c| & \text{if } \operatorname{sign}(y_i - mx_i + c) \neq l_i \end{cases}$$

$$\mathbf{m} = \begin{bmatrix} m \\ c \end{bmatrix}$$

$$e(y_i, x_i; \mathbf{m}) = \begin{cases} 0 & \text{if } \begin{bmatrix} x_i & 1 \end{bmatrix} \mathbf{m} = l_i \\ |y_i - \begin{bmatrix} x_i & 1 \end{bmatrix} \mathbf{m}| & \text{if } \begin{bmatrix} x_i & 1 \end{bmatrix} \mathbf{m} \neq l_i \end{cases}$$

$$\nabla_{\mathbf{m}} e(y_i, x_i; \mathbf{m}) = \begin{cases} 0 & \text{if } \begin{bmatrix} x_i & 1 \end{bmatrix} \mathbf{m} = l_i \\ |y_i - \begin{bmatrix} x_i & 1 \end{bmatrix}| & \text{if } \begin{bmatrix} x_i & 1 \end{bmatrix} \mathbf{m} \neq l_i \end{cases}$$

If $l_i \in \{-1, 1\}$, then we can write

$$e(y_i, x_i; \mathbf{m}) = \max\{0, -l_i(y_i - \begin{bmatrix} x_i & 1 \end{bmatrix} \mathbf{m})\}$$

$$\nabla_{\mathbf{m}} e(y_i, x_i; \mathbf{m}) = \max\{0, l_i(\begin{bmatrix} x_i & 1 \end{bmatrix})\}$$

For the entire dataset, we have $\mathbf{y} = [y_1; \dots; y_n]$ and $\mathbf{x} = [x_1; \dots; x_n]$, $\mathbf{l} = [l_1; \dots; l_n]$ the average error is:

$$e(\mathbf{x}, \mathbf{y}; \mathbf{m}) = \frac{1}{n} 1_n^\top \max\{0, -\mathbf{l} \odot (\mathbf{y} - \begin{bmatrix} \mathbf{x} & 1_n \end{bmatrix} \mathbf{m})\}$$

and the average gradient is:

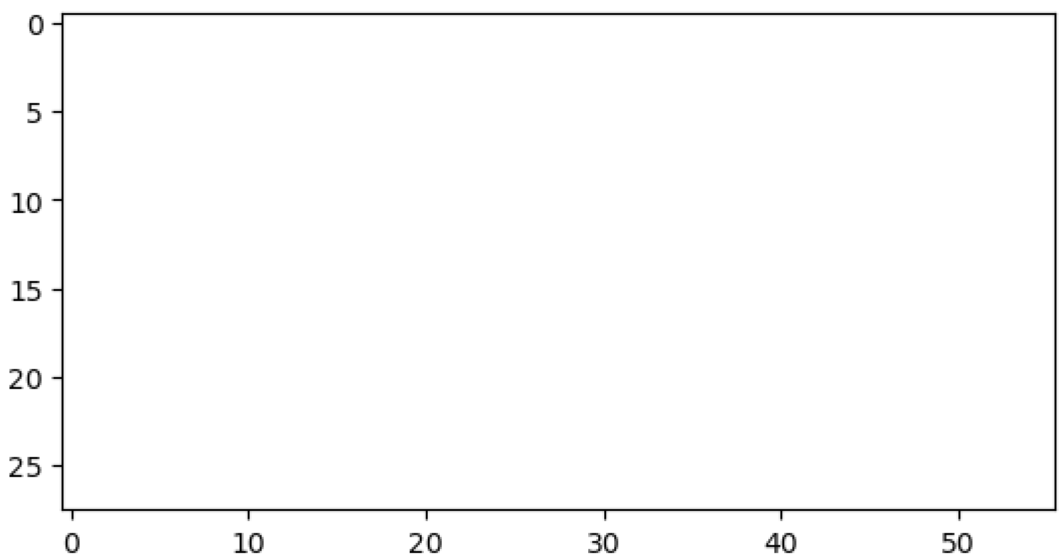
$$\nabla_{\mathbf{m}} e(\mathbf{x}, \mathbf{y}; \mathbf{m}) = \frac{1}{n} 1_n^\top \max\{0, \mathbf{l} \odot (\begin{bmatrix} \mathbf{x} & 1_n \end{bmatrix})\}$$

In [1]:
```
# Download MNIST dataset
!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
```

In [1]:
```
# 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
```

In [2]:
```
# 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, axes = plt.subplots(2, 10)
# for axrow, imgs in zip(axes, (zero_images, one_images)):
#     for ax, img in zip(axrow, imgs):
#         ax.imshow(img, cmap='gray', vmin=0, vmax=255)
#         ax.axis('off')


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,
    for i in range(60)]
zero_images_anim = animation.ArtistAnimation(fig, ims, interval=50, blit=Tru
                                repeat_delay=1000, repeat=False)
```



Images as arrays

In [3]:
```
train_images.shape
```

Out[3]:
```
(60000, 28, 28)
```

In [4]:
```
img1 = train_images[0]
img1
```

Out[4]:

```
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,
```

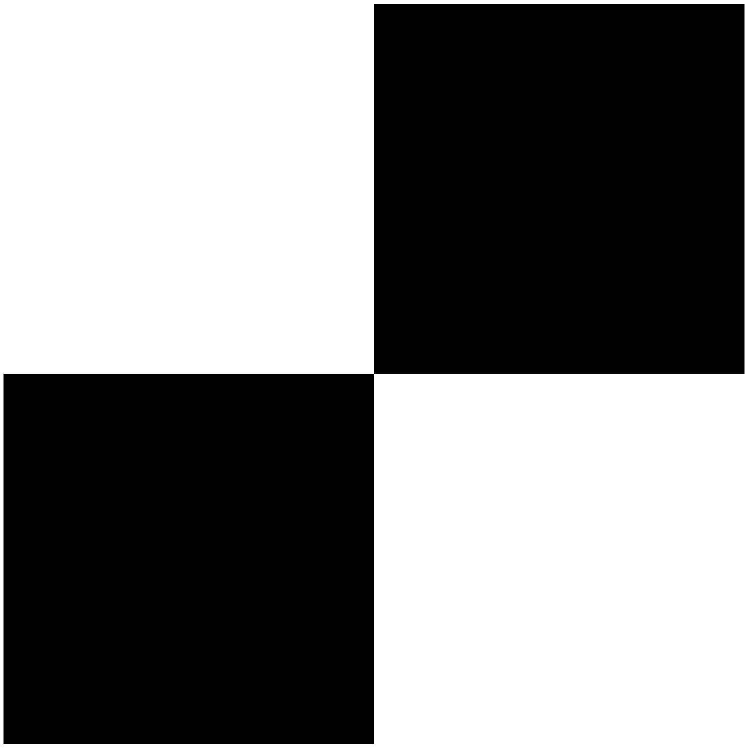
```
         0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39,
        148, 229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114, 221,
        253, 253, 253, 253, 201,  78,   0,   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)
```

In [5]:

```python
## 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')
```

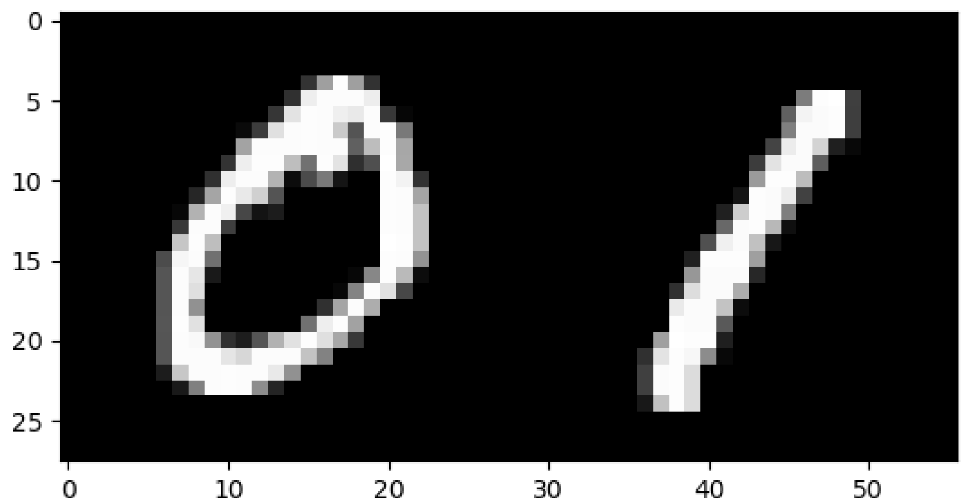
Out[5]:

```
<matplotlib.image.AxesImage at 0x7f675bbd5b10>
```



In [6]:
```
zero_images_anim
```

Out[6]:



Once Loop Reflect

What is a feature

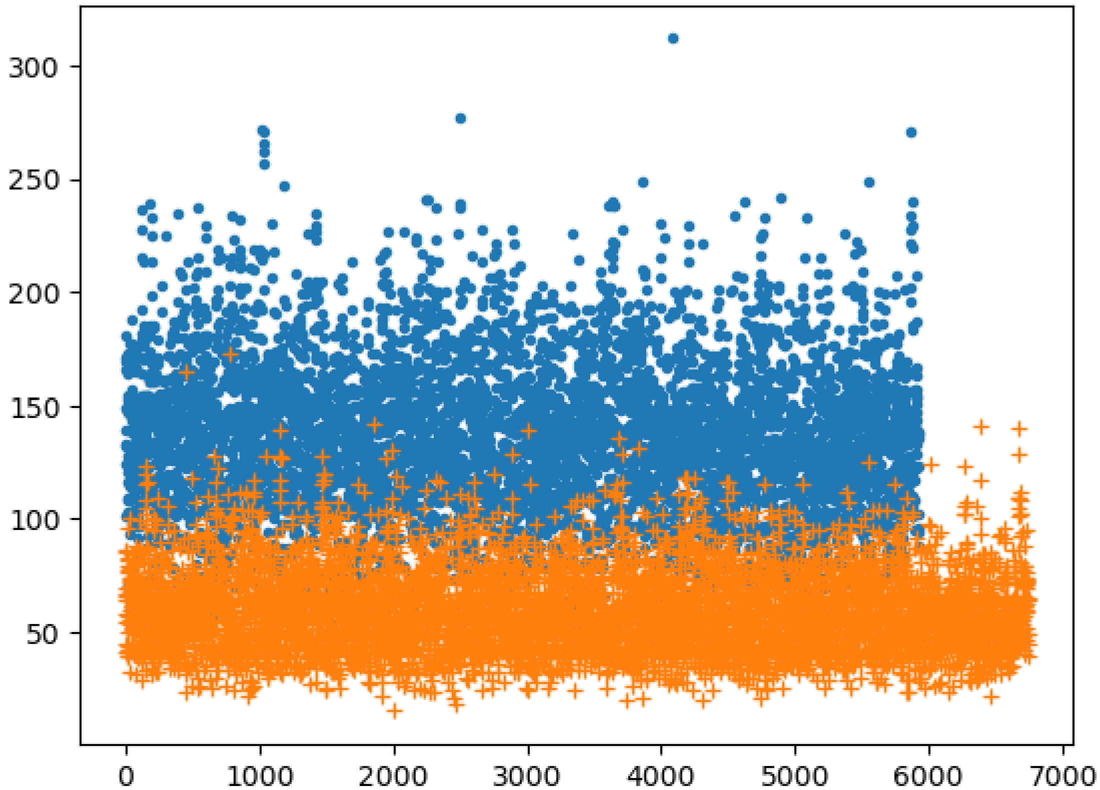
Any property of data sample that helps with the task.

In [7]:

```
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, '+')
```

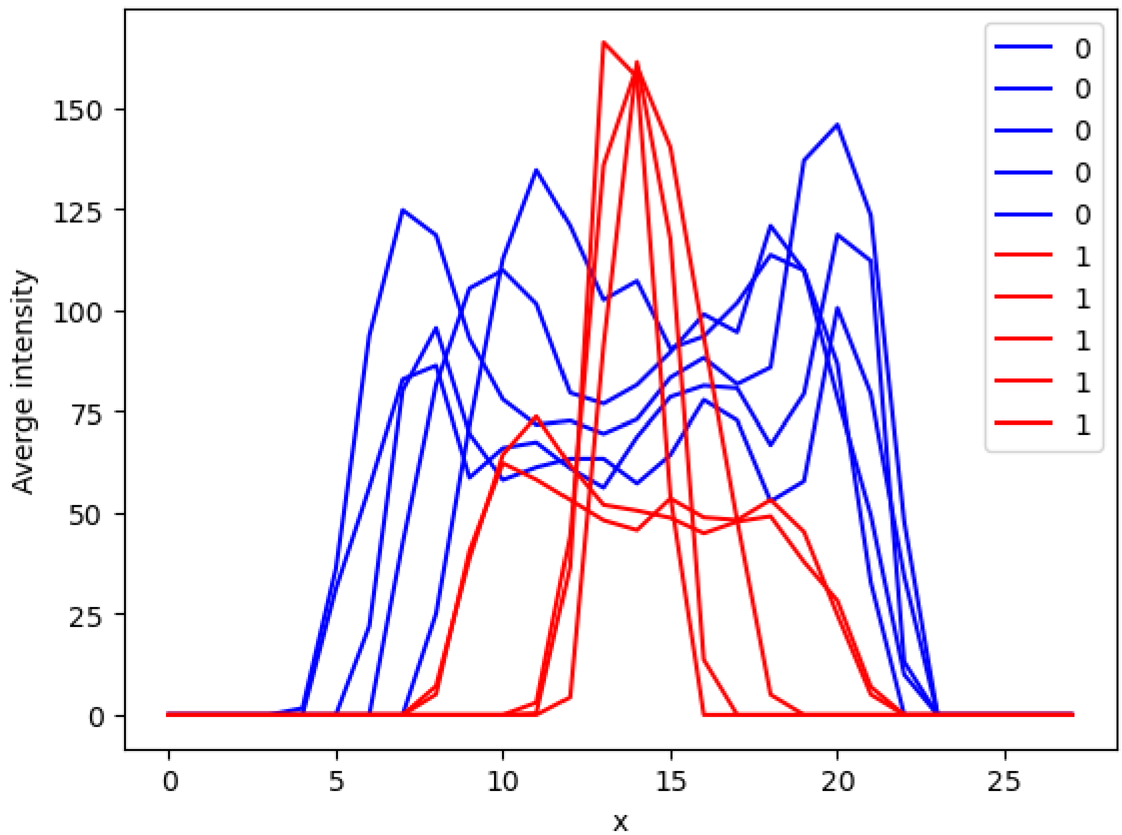
Out[7]:

```
[<matplotlib.lines.Line2D at 0x7f675baaedd0>]
```



In [8]:
```
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')
```

Out[8]:
```
Text(0, 0.5, 'Averge intensity')
```



$$\mu_x(I) = \sum_{x=1}^{W} \frac{xI(x,y)}{\sum_{x=1}^{W} I(x,y)}$$

$$\sigma_x^2(I) = \sum_{x=1}^{W} \frac{(x-\mu_x)^2 I(x,y)}{\sum_{x=1}^{W} I(x,y)}$$

In [9]:
```
wts = zero_images[0].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)
var
```

Out[9]: 22.811061800377757

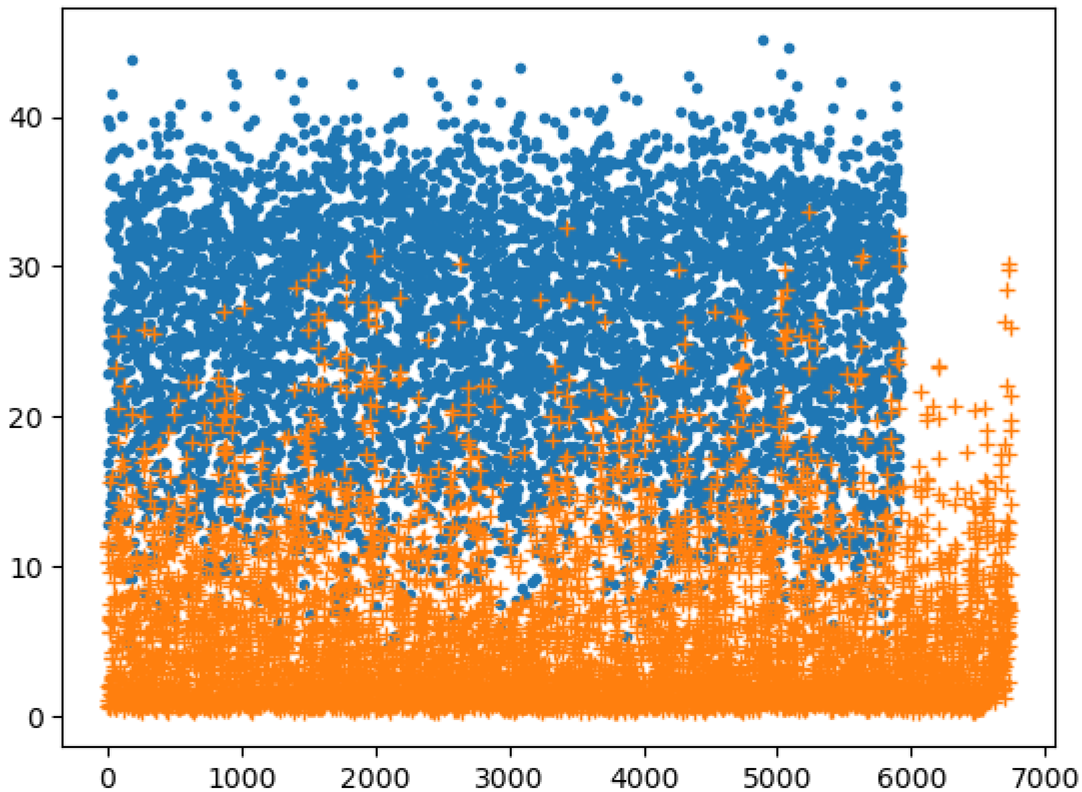
In [10]:
```
def feature_y_var(img):
    wts = img.mean(axis=0)
    mean = (np.arange(wts.shape[0]) * wts).sum() / np.sum(wts)
    var = ((np.arange(wts.shape[0]) - mean)**2 * wts).sum() / np.sum(wts)
    return var
feature_y_var(zero_images[0]), feature_y_var(one_images[0])
```

Out[10]: (22.811061800377757, 11.384958735403274)

In [11]:
```
def feature_y_var(imgs):
    wts = imgs.mean(axis=-2)
    arange = np.arange(wts.shape[-1])
```

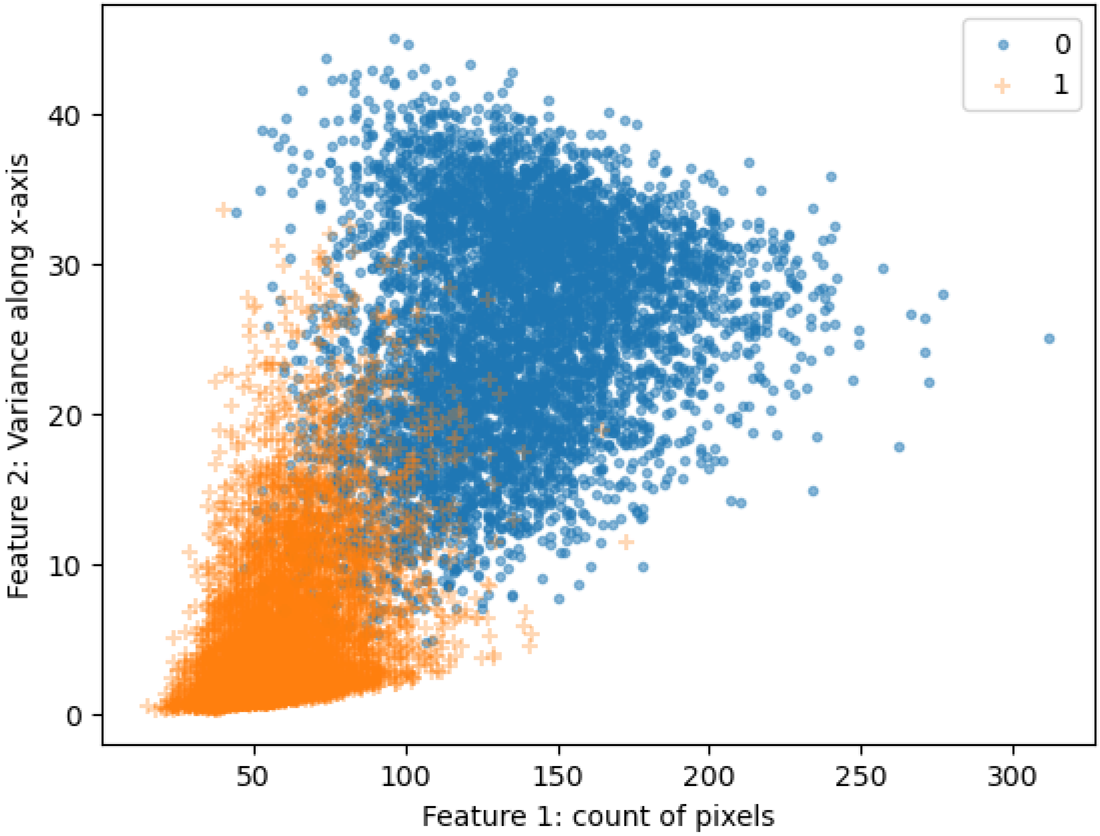
```
    mean = (arange * wts).sum(axis=-1) / wts.sum(axis=-1)
    mean = mean[:, np.newaxis]
    var = ((arange - mean)**2 * wts).sum(axis=-1) / wts.sum(axis=-1)
    return var

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

Out[11]: [<matplotlib.lines.Line2D at 0x7f675a1fa4a0>]



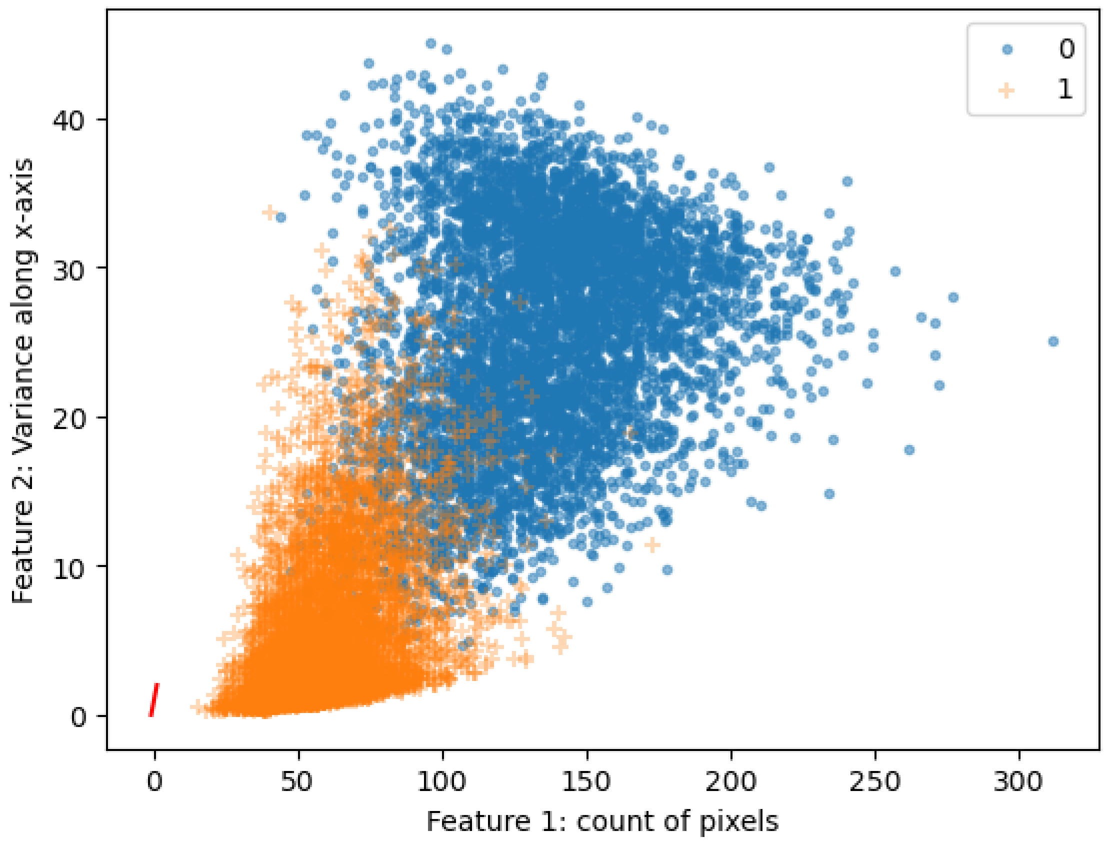
In [13]:
```
def features_extract(images):
    return np.vstack((feature_n_pxls(images),
                      feature_y_var(images))).T
zero_features = features_extract(zero_images)
one_features = features_extract(one_images)


def draw_features(ax, zero_features, one_features):
    zf = ax.scatter(zero_features[:, 0], zero_features[:, 1], marker='.', la
    of = ax.scatter(one_features[:, 0], one_features[:, 1], marker='+', labe
    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
```

In [14]:
```
fig, ax = plt.subplots()
draw_features(ax, zero_features, one_features)
plt.show()
```



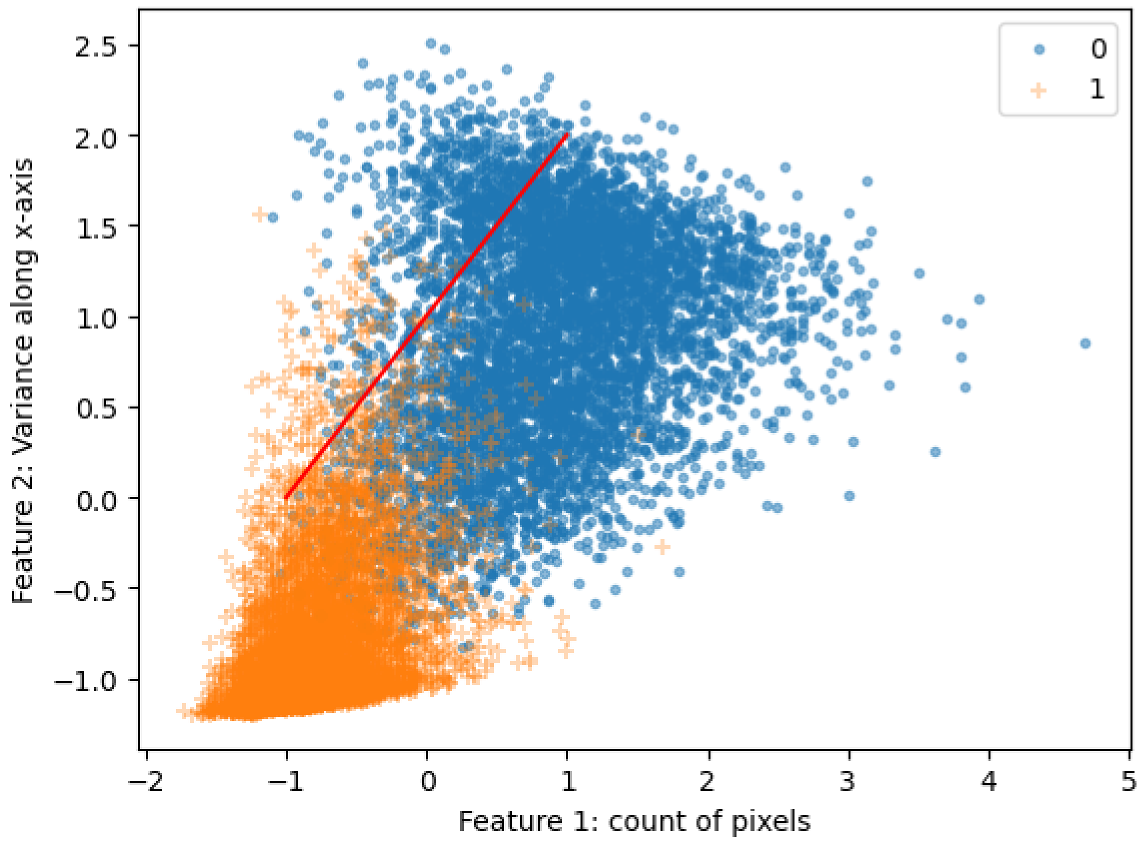
In [15]:
```
bfm = np.ones(2)
fig, ax = plt.subplots()
draw_features(ax, zero_features, one_features)
x = np.linspace(-1, 1, 21)
ax.plot(x, x*bfm[0] + bfm[1], 'r-')
```

Out[15]: `[<matplotlib.lines.Line2D at 0x7f6759fc66e0>]`



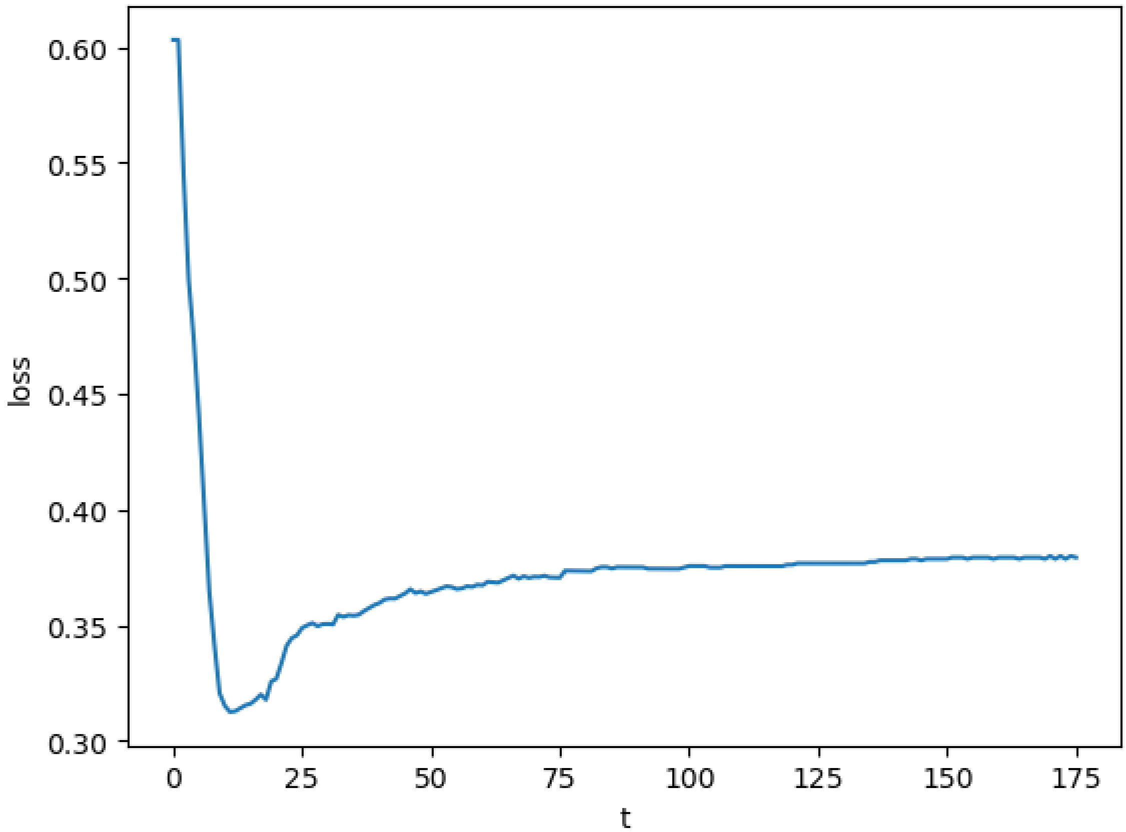
In [16]:
```
bfm = np.ones(2)

Y = np.hstack((np.ones(zero_features.shape[0]), np.full(one_features.shape[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*bfm[0] + bfm[1], 'r-')
```

Out[16]: [<matplotlib.lines.Line2D at 0x7f675662ece0>]



In [17]:
```
def error(X, Y, bfm):
    ### BEGIN SOLUTION
    return - (X[:,1] - X[:, 0] * bfm[0] - bfm[1]) * Y
    ### END SOLUTION

def grad_error(Xw, Yw, bfm):
    ### BEGIN SOLUTION
    return np.array([(Xw[:, 0]*Yw).mean(), Yw.mean()])
    ### END SOLUTION

def train(X, Y, lr = 0.1):
    ### BEGIN SOLUTION
    bfm = np.random.rand(2)*4-2
    bfm_prev = bfm + 1
    list_of_bfms = [bfm]
    list_of_errors = []


    err = error(X, Y, bfm)
    grad_err = grad_error(X[err > 0, :], Y[err > 0], bfm)
    list_of_errors.append(err[err > 0].mean())
    for _ in range(400):
        if np.linalg.norm(grad_err) < 0.001:
            break
        err = error(X, Y, bfm)
        grad_err = grad_error(X[err > 0, :], Y[err > 0], bfm)
        bfm_prev = bfm
        bfm = bfm - lr * grad_err
        list_of_bfms.append(bfm)
```

```
        list_of_errors.append(err[err > 0].mean())
    return bfm, list_of_bfms, list_of_errors
    ### END SOLUTION

OPTIMAL_BFM, list_of_bfms, list_of_errors = train(X, Y)
fig, ax = plt.subplots()
ax.plot(list_of_errors)
ax.set_xlabel('t')
ax.set_ylabel('loss')
plt.show()
```

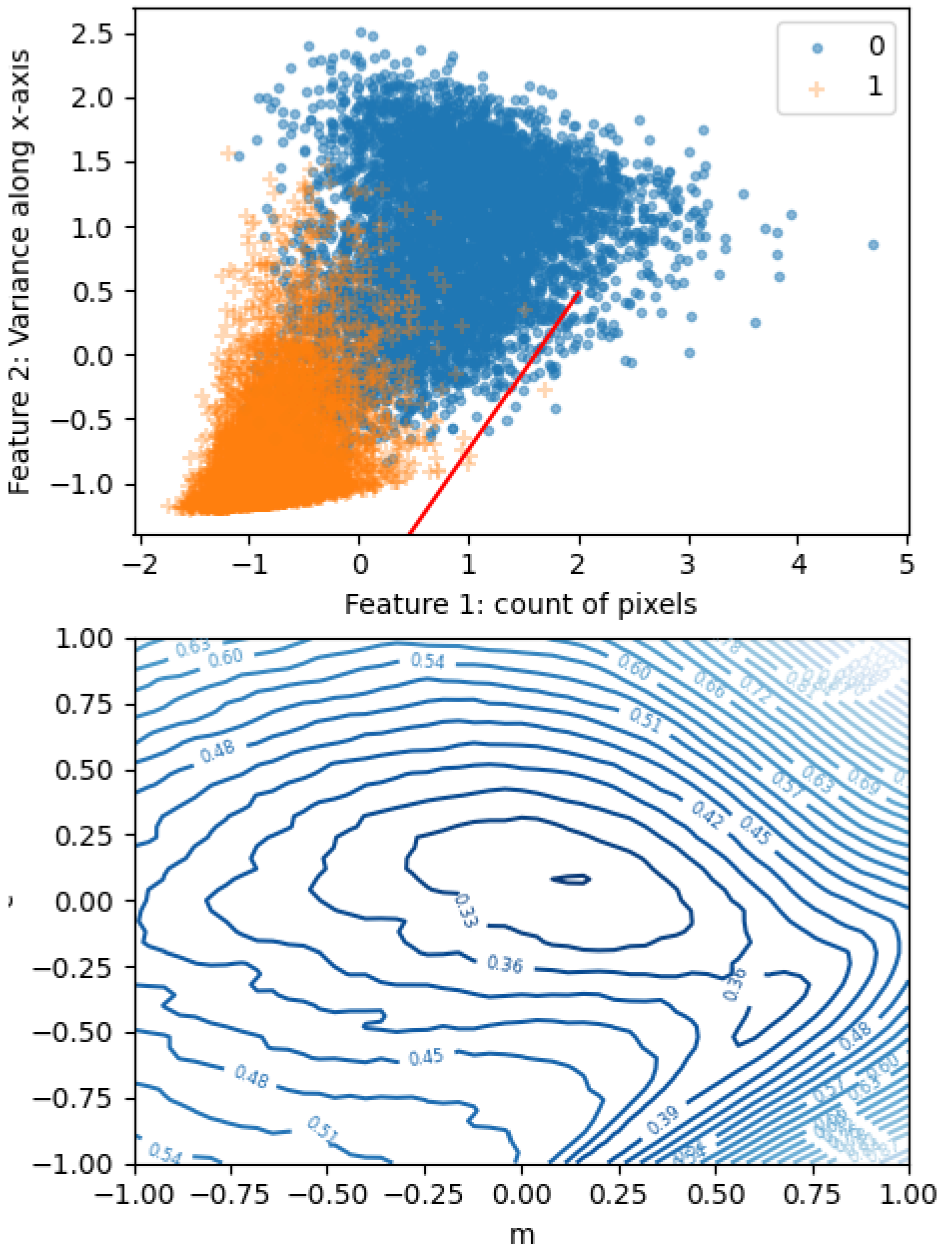


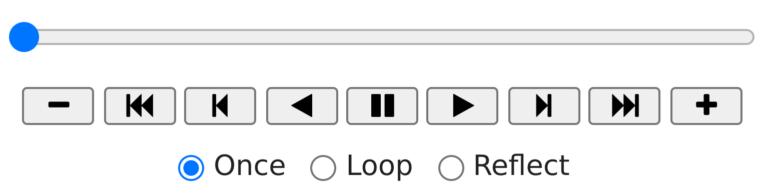
In [18]:

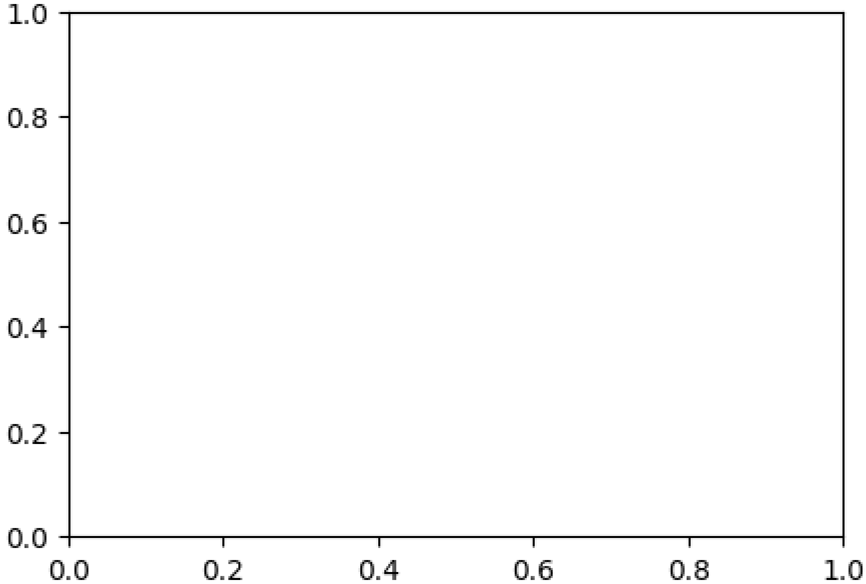
```
fig, axes = plt.subplots(2, 1, figsize=(5, 7.5))
class Anim:
    def __init__(self, fig, axes, X, Y):
        self.fig = fig
        self.ax = axes[0]
        self.ax1 = axes[1]
        self.fts = draw_features(self.ax, X[Y > 0, :], X[Y < 0, :])
        self.line, = self.ax.plot([], [], 'r-')

        m, c = np.meshgrid(np.linspace(-1, 1, 51), np.linspace(-1, 1, 51))
        totalerr = np.empty_like(m)
        for i in range(m.shape[0]):
            for j in range(m.shape[1]):
                err = error(X, Y, [m[i, j], c[i,j]])
                totalerr[i, j] = err[err > 0].mean()

        self.ctr = self.ax1.contour(m, c, totalerr, 30, cmap='Blues_r')
        self.ax1.set_xlabel('m')
        self.ax1.set_ylabel('c')
```

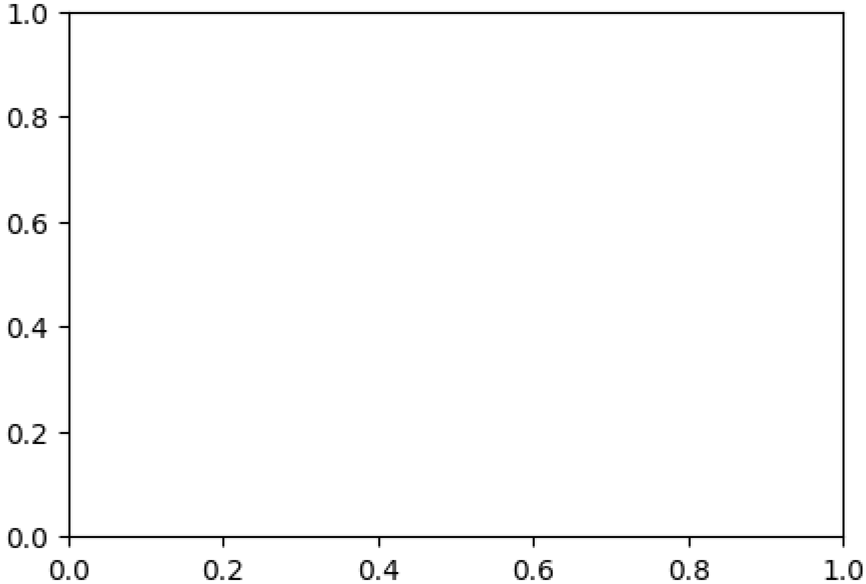
```
        self.ax1.clabel(self.ctr, self.ctr.levels, inline=True, fontsize=6)
        self.m_hist = []
        self.c_hist = []
        self.line2, = self.ax1.plot([], [], 'r*-')


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

    def update(self, bfm):
        x = np.linspace(-2, 2, 21)
        self.line.set_data(x, x * bfm[0] + bfm[1])
        self.m_hist.append(bfm[0])
        self.c_hist.append(bfm[1])
        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, Y)
animation.FuncAnimation(fig, a.update, frames=list_of_bfms[::3],
                        init_func=a.anim_init, blit=True, repeat=False)
```

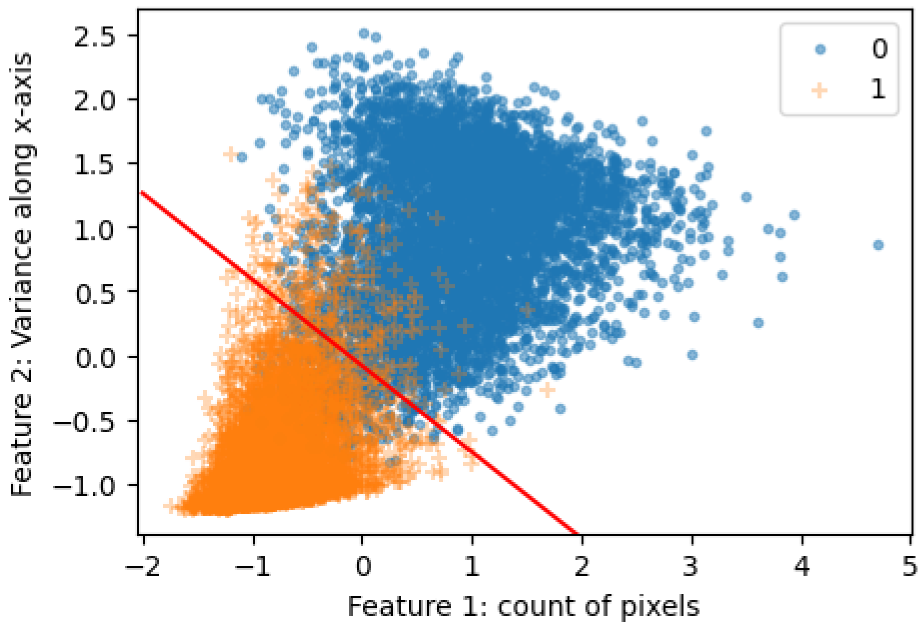
Out[18]:

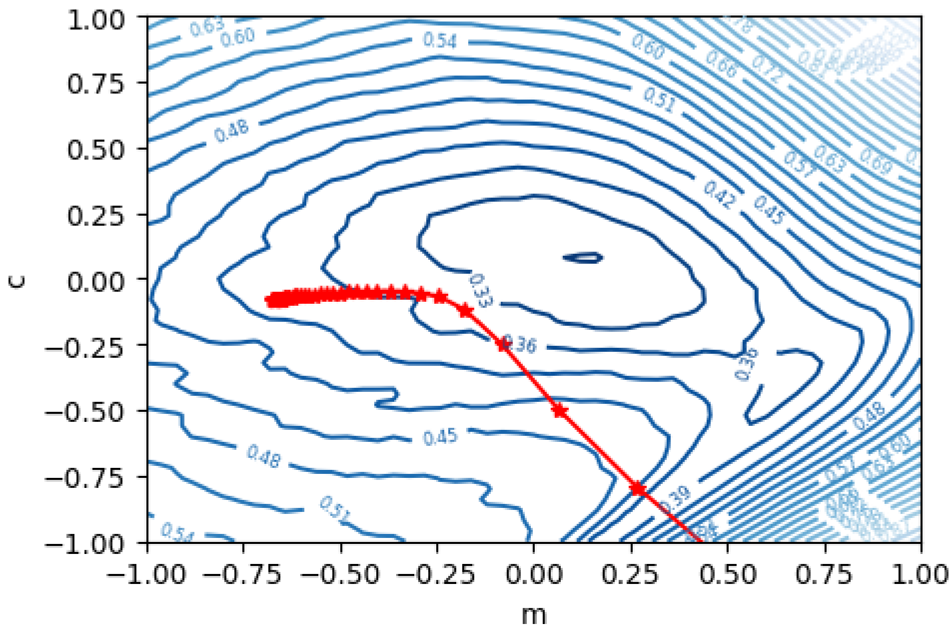

Feature 2: Variance along x-axis
Feature 1: count of pixels
0
1
m


Once
Loop
Reflect


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0





In [19]:
```
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, ...]

def returnclasslabel(test_imgs):
    Xtest = norm_features(features_extract(test_imgs))
    bfm = OPTIMAL_BFM
    return np.where(
        Xtest[:, 1] - Xtest[:, 0] * bfm[0] - bfm[1] > 0,
        0,
        1)
zero_one_predicted_labels = returnclasslabel(zero_one_test_images)
```

In [20]:
```
accuracy = np.sum(zero_one_test_labels == zero_one_predicted_labels) / len(z
accuracy
```

Out[20]: 0.9569739952718677

In [21]:
```
positive_label = 1
negative_label = 0
TP = np.sum((zero_one_test_labels == positive_label) & (zero_one_predicted_l
TP
```

Out[21]: 1085

In [22]:
```
TN =  np.sum((zero_one_test_labels == negative_label) & (zero_one_predicted_
TN
```

Out[22]: 939

In [23]:
```
FP = np.sum((zero_one_test_labels != positive_label) & (zero_one_predicted_l
FP
```

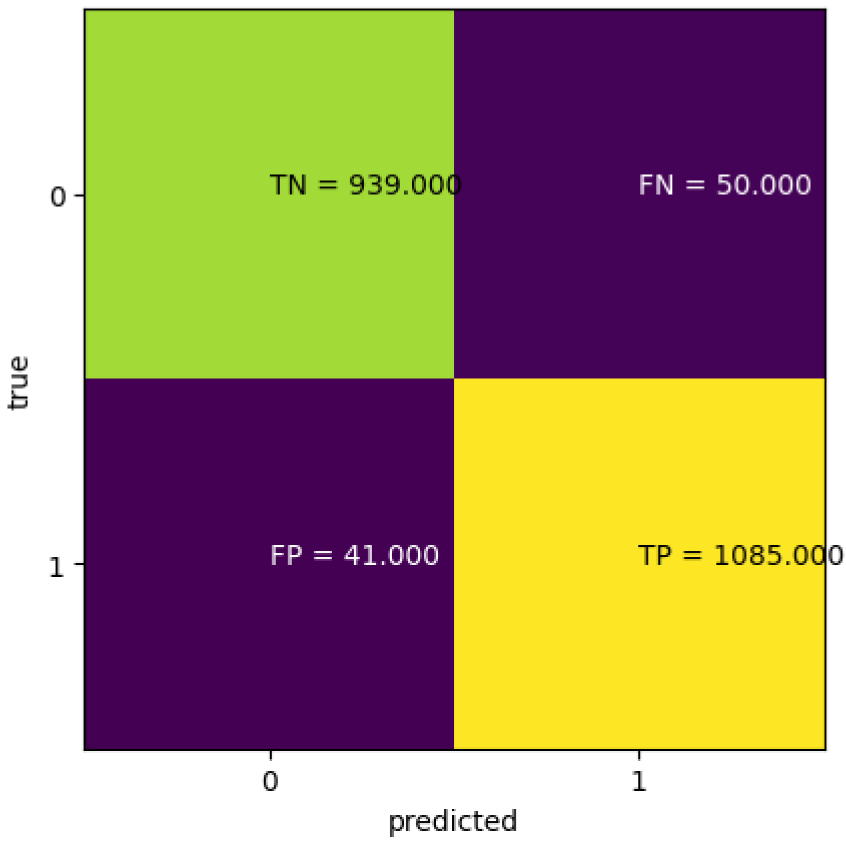
Out[23]: 41

In [24]:
```
FN = np.sum((zero_one_test_labels != negative_label) & (zero_one_predicted_l
FN
```

Out[24]: 50

In [25]:
```
# 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)
```

Out[25]: Text(1, 1, 'TP = 1085.000')



In [26]:
```
PRECISION = TP / (TP + FP)
PRECISION
```

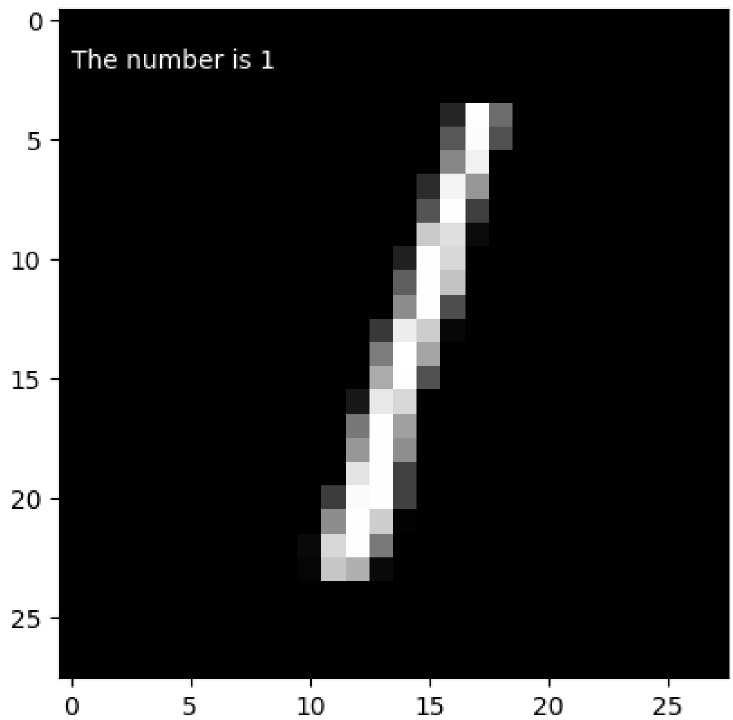
Out[26]: 0.9635879218472468

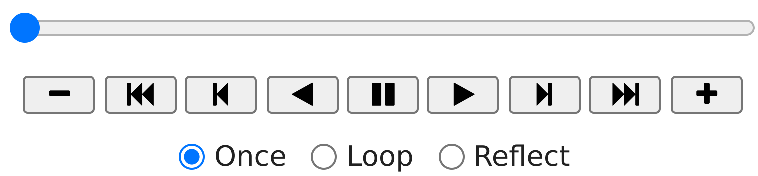
In [27]:
```
RECALL = TP / (TP + FN)
RECALL
```

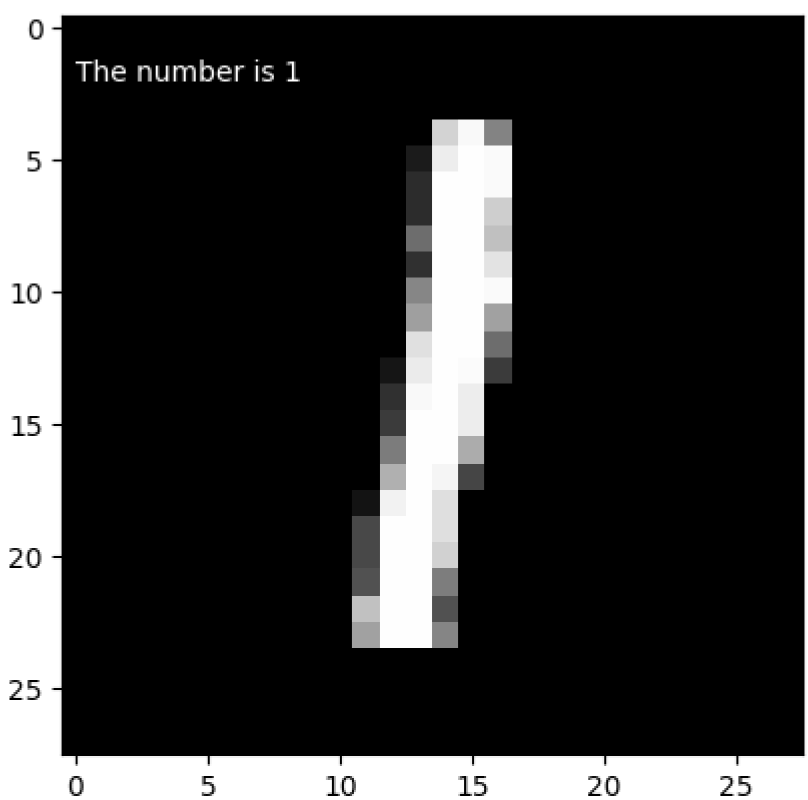
Out[27]: 0.9559471365638766

In [28]:
```
fig, ax = plt.subplots()
artists = []
for i in range(60):
    artists.append(
        [ax.imshow(zero_one_test_images[i], animated=True, cmap='gray', vmir
        ax.text(0, 2, 'The number is %d' % zero_one_predicted_labels[i], ani
animation.ArtistAnimation(fig, artists, interval=50, blit=True,
                                repeat_delay=1000, repeat=False)
```

Out[28]:


The number is 1
0
5
10
15
20
25


Once
Loop
Reflect


The number is 1
0
5
10
15
20
25

Single Layer Neural Networks

Read Chapter 2 and 3 of UDL Book

Notes Single Layer Neural Networks are simplest kind of neural networks. But before we dive into single layer neural networks, may be we should focus on the name *neural* networks. The name neural networks comes from biological neurons.

Similarities between Artificial neuron and Biological neuron

No description has been provided for this image

No description has been provided for this image

1. The excitation or firing of a biological neuron can be equated to a high positive value of units (x_1, x_2, x_3) in artificial neurons.

2. The synapse in biological neuron determines which input excitations will have excitatory or inhibitary impact on output excitations. Synapses can strengthen, weaken, disconnect or form new connections during biological learning. Similarly to excitatory synapes, positive weights can cause positive input values to contribute to positive output values.

3. Usually multiple excitatory inputs are required excite the output neuron.

References:

1. https://openstax.org/books/biology/pages/35-2-how-neurons-communicate

Differences

1. Biological neuron is all or None
2. Biological neuron has a time component
   No description has been provided for this image

notes

1. The activity of the biological neuron is an "all-or-none" process. Articial activations are typically continuous range. Even when sigmoid or softmax nonlinearities.

2. Biological neuron has time dynamics. The input activations are integrated over time.

Next

2. Show visualization of 1D optimization and loss functions.

3. Build to visualizations in the UDL book. Connect to KD tree and nearest neighbor classification.

4. Show the tensflow js visualization.

References

1. http://playground.tensorflow.org
2. https://knowyourdata-tfds.withgoogle.com/#tab=STATS&dataset=tf_flowers
3. "Flowers", The TensorFlow Team. Jan 2019. Online http://download.tensorflow.org/example_images/flower_photos.tgz