Recall Linear Least square regression

$\sum_i (y_i - mx_i + c)^2$

$$
\begin{aligned}
\mathbf{0}^\top &= \frac{\partial}{\partial \mathbf{m}} (\mathbf{y}^\top \mathbf{y} + \mathbf{m}^\top \mathbf{X}^\top \mathbf{X} \mathbf{m} - 2\mathbf{y}^\top \mathbf{X} \mathbf{m}) && (1) \\
&= 2{\mathbf{m}^*}^\top \mathbf{X}^\top \mathbf{X} - 2\mathbf{y}^\top \mathbf{X} && (2)
\end{aligned}
$$

This gives us the solution

$$\mathbf{m}^* = \underbrace{(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top}\mathbf{y}$$

Psuedo inverse

The symbol $\mathbf{V}^{-1}$ is called inverse of matrix $\mathbf{V}$.

The term $(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top$ is also called the pseudo-inverse of a matrix $\mathbf{X}$, denoted as $\mathbf{X}^\dagger$.

In [1]:
```
%%writefile saltconcentration.tsv
#Observation    SaltConcentration       RoadwayArea
1       3.8     0.19
2       5.9     0.15
3       14.1    0.57
4       10.4    0.4
5       14.6    0.7
6       14.5    0.67
7       15.1    0.63
8       11.9    0.47
9       15.5    0.75
10      9.3     0.6
11      15.6    0.78
12      20.8    0.81
13      14.6    0.78
14      16.6    0.69
15      25.6    1.3
16      20.9    1.05
17      29.9    1.52
18      19.6    1.06
19      31.3    1.74
20      32.7    1.62
```

```
Overwriting saltconcentration.tsv
```

In [2]:
```
# numpy can import text files separated by seprator like tab or comma
import numpy as np
salt_concentration_data = np.loadtxt("saltconcentration.tsv")
```

In [3]:
```
n = salt_concentration_data.shape[0]
bfx = salt_concentration_data[:, 2:3]
bfy = salt_concentration_data[:, 1]
```

```python
bfX = np.hstack((bfx, np.ones((bfx.shape[0], 1))))
bfX
```

Out[3]:
```
array([[0.19, 1.  ],
       [0.15, 1.  ],
       [0.57, 1.  ],
       [0.4 , 1.  ],
       [0.7 , 1.  ],
       [0.67, 1.  ],
       [0.63, 1.  ],
       [0.47, 1.  ],
       [0.75, 1.  ],
       [0.6 , 1.  ],
       [0.78, 1.  ],
       [0.81, 1.  ],
       [0.78, 1.  ],
       [0.69, 1.  ],
       [1.3 , 1.  ],
       [1.05, 1.  ],
       [1.52, 1.  ],
       [1.06, 1.  ],
       [1.74, 1.  ],
       [1.62, 1.  ]])
```

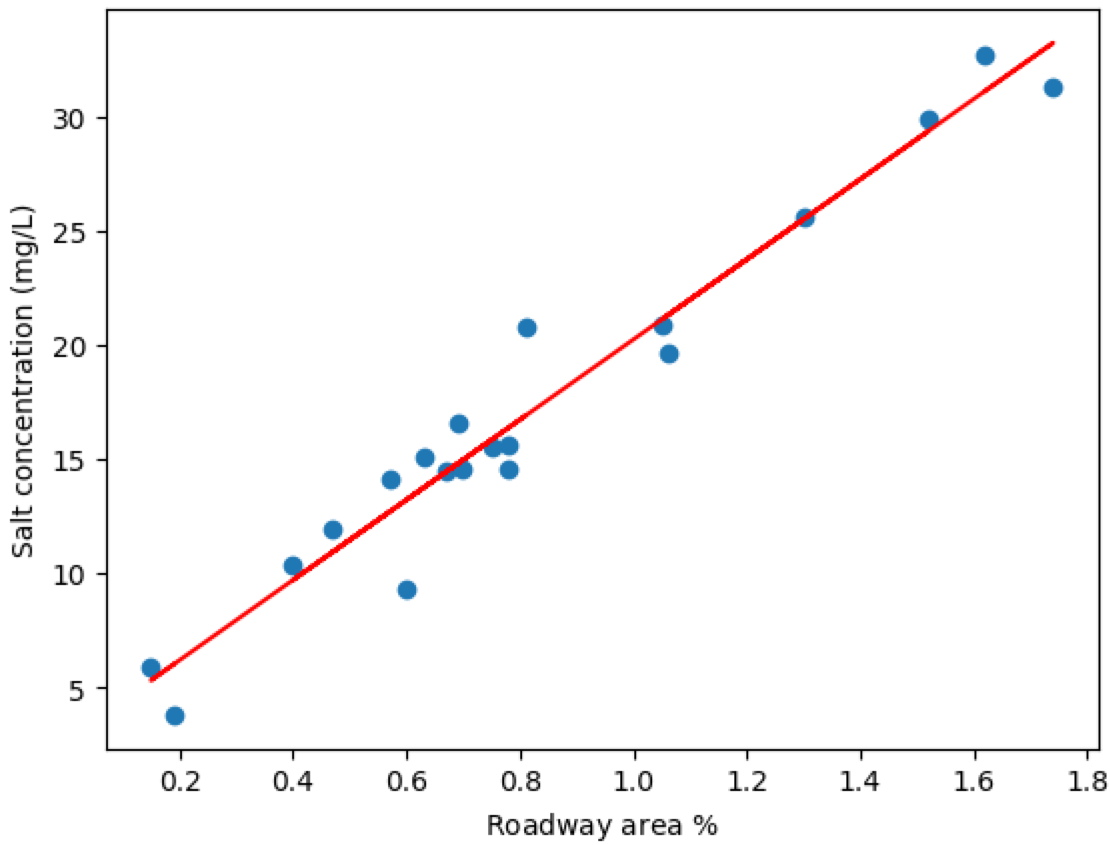
$$X = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ \vdots & \\ x_n & 1 \end{bmatrix}$$

In [4]:
```python
bfm = np.linalg.inv(bfX.T @ bfX) @ bfX.T @ bfy
print(bfm)
bfm, *_ = np.linalg.lstsq(bfX, bfy, rcond=None)
print(bfm)
```

```
[17.5466671   2.67654631]
[17.5466671   2.67654631]
```

In [5]:
```python
import matplotlib.pyplot as plt
m = bfm.flatten()[0]
c = bfm.flatten()[1]

# Plot the points
fig, ax = plt.subplots()
ax.scatter(salt_concentration_data[:, 2], salt_concentration_data[:, 1])
ax.set_xlabel(r"Roadway area $\%$")
ax.set_ylabel(r"Salt concentration (mg/L)")
x = salt_concentration_data[:, 2]
y = m * x + c
# Plot the points
ax.plot(x, y, 'r-') # the line
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0x7f86d7c4bdc0>]
```



Second derivative

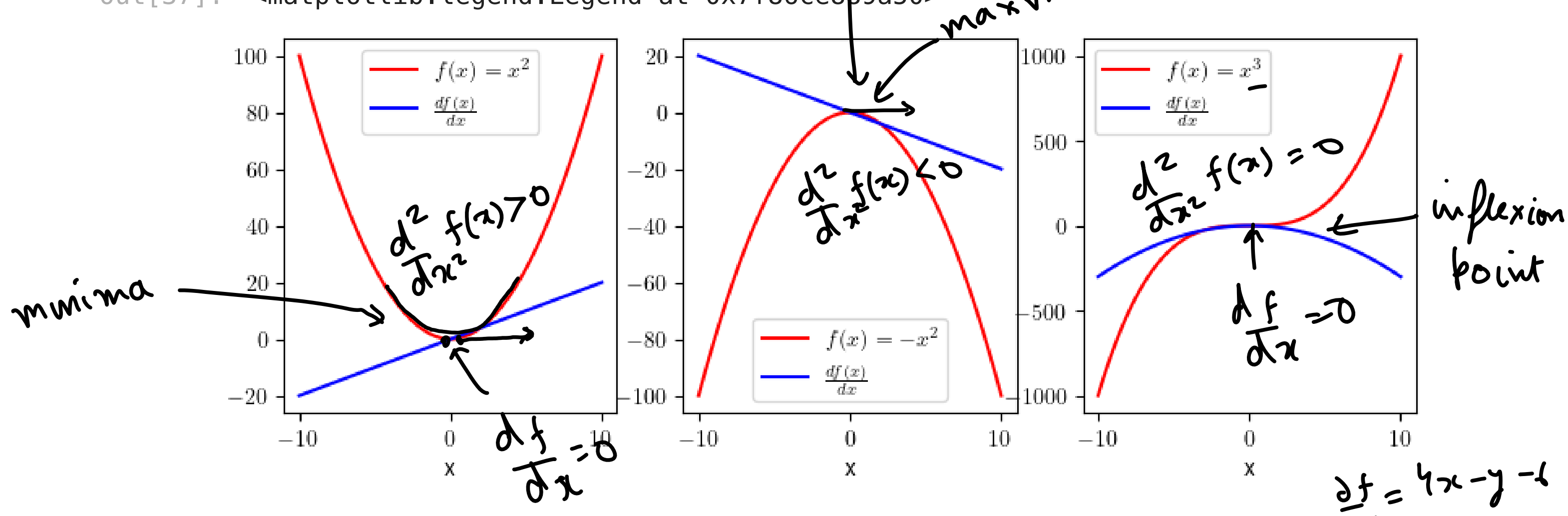
Geometry of second derivative

In [36]:
```
import matplotlib.pyplot as plt
plt.rcParams.update({
    "text.usetex": True # turns on math latex rendering in matplotlib
})
```

In [37]:
```
x = np.linspace(-10, 10, 100)
fig, ax = plt.subplots(1, 3, figsize=(9, 3))
ax[0].plot(x, x**2, 'r', label=r'$f(x)=x^2$')
ax[0].plot(x, 2*x, 'b', label=r'$\frac{df(x)}{dx}$')
ax[0].set_xlabel('x')
ax[0].legend()

ax[1].plot(x, -x**2, 'r', label=r'$f(x)=-x^2$')
ax[1].plot(x, -2*x, 'b', label=r'$\frac{df(x)}{dx}$')
ax[1].set_xlabel('x')
ax[1].legend()


ax[2].plot(x, x**3, 'r', label=r'$f(x)=x^3$')
ax[2].plot(x, -3*x**2, 'b', label=r'$\frac{df(x)}{dx}$')
ax[2].set_xlabel('x')
ax[2].legend()
```

$\frac{df}{dx} = 0$

Out[37]: <matplotlib.legend.Legend at 0x7f86ce839a50>



$\frac{\partial f}{\partial x} = 4x - y - 6$

Second derivatives in 2 dimension

$\frac{\partial^2}{\partial x^2} f = 4$

$\frac{\partial^2}{\partial y^2} f = 8$

$\frac{\partial^2}{\partial x^2} f = -4$

$\frac{\partial^2}{\partial y^2} f = -8$

1. $f(x, y) = 2x^2 + 4y^2 - xy - 6x - 8y + 6$
2. $f(x, y) = -2x^2 - 4y^2 + xy + 6x + 8y + 6$
3. $f(x, y) = 2x^2 - 4y^2 - xy - 6x + 8y + 6$

$\frac{\partial^2}{\partial x^2} f = 4$

$\frac{\partial^2}{\partial y^2} f = -8$

Example 1:

$$f(x, y) = 2x^2 + 4y^2 - xy - 6x - 8y + 6$$

$$f([x, y]) = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 2 & -1/2 \\ -1/2 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} -6 & -8 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + 6$$

$x^T A x \quad + \quad b^T x \quad + \quad c$

Example 2:

$$f(x, y) = -2x^2 - 4y^2 + xy + 6x + 8y + 6$$

$$f([x, y]) = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} -2 & 1/2 \\ 1/2 & -4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 6 & 8 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + 6$$

Example 3:

$$f(x, y) = 2x^2 - 4y^2 - xy - 6x + 8y + 6$$

$$f([x, y]) = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 2 & -1/2 \\ -1/2 & -4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} -6 & 8 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + 6$$

In [38]:

```
import plotly.graph_objects as go
import numpy as np
import matplotlib.pyplot as plt
def plot_surface(func):
    x, y = np.mgrid[-20:20:21j,
```

```
                    -20:20:21j]
    f =  func(x, y)

    print(f.shape, x.shape, y.shape)
    fig = go.Figure(data=[go.Surface(z=f, x=x, y=y)])
    fig.update_traces(contours_z=dict(show=True, usecolormap=True,
                                      highlightcolor="limegreen", project_z=
    fig.show()
```

Example 1:

$$f(x,y) = 2x^2 + 4y^2 - xy - 6x - 8y + 6$$

$$f([x,y]) = \underbrace{\begin{bmatrix} x & y \end{bmatrix}}_{x^T} \underbrace{\begin{bmatrix} 2 & -1/2 \\ -1/2 & 4 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x \\ y \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} -6 & -8 \end{bmatrix}}_{b^T} \underbrace{\begin{bmatrix} x \\ y \end{bmatrix}}_{x} + 6$$

In [39]:

```
def f(x, y):
    return 2*x**2 + 4*y**2 - x*y - 6*x - 8*y  + 6

def f_vec(x, y):
    # x is n x n and y is n x n
    xn = x[..., None] # n x n x 1
    yn = y[..., None] # n x n x 1
    vecx = np.concatenate([xn, yn], axis=-1) # n x n x 2
    vecx_col_vec = vecx[..., None] # n x n x 2 x 1
    vecx_row_vec = vecx[..., None, :] # n x n x 1 x 2
    A = np.array([[2, -0.5],
                  [-0.5, 4]]) # 2 x 2
    b = np.array([-6, -8]) # 2
    c = 6
    print("Minima at, ", -np.linalg.inv(A + A.T) @ b)
    quad  = (vecx_row_vec @ A @ vecx_col_vec).squeeze(-1).squeeze(-1)
    return quad + vecx @ b + c

plot_surface(f_np)
```

```
Minima at,  [1.80645161 1.22580645]
(21, 21) (21, 21) (21, 21)
```

$$f(x,y) = -2x^2 - 4y^2 + xy - 6x - 8y + 6$$

In [18]:

```
def f(x, y): return - 2*x**2 - 4*y**2 + x*y + 6*x + 8*y  + 6

def f_vec(x, y):
    # x is n x n and y is n x n
    xn = x[..., None] # n x n x 1
    yn = y[..., None] # n x n x 1
    vecx = np.concatenate([xn, yn], axis=-1) # n x n x 2
    vecx_col_vec = vecx[..., None] # n x n x 2 x 1
    vecx_row_vec = vecx[..., None, :] # n x n x 1 x 2
    A = np.array([[-2, 0.5],
                  [0.5, -4]]) # 2 x 2
    b = np.array([6, 8]) # 2
    c = 6
    print("Maxima at, ", -np.linalg.inv(A + A.T) @ b)
    quad  = (vecx_row_vec @ A @ vecx_col_vec).squeeze(-1).squeeze(-1)
    return quad + vecx @ b + c
plot_surface(f_vec)
```

```
Maxima at,  [1.80645161 1.22580645]
(21, 21) (21, 21) (21, 21)
```

$$f(x,) = 2x^2 - 4y^2 - xy - 6x - 8y + 6$$

In [19]:
```
def f(x, y): return  2*x**2 - 4*y**2 - x*y - 6*x + 8*y  + 6
plot_surface(f)
```

```
(21, 21) (21, 21) (21, 21)
```

Second derivative in n-D : Hessian matrix

Hessian matrix of a scalar-valued vector function $f : \mathbb{R}^n \to \mathbb{R}$ is defined as the following arrangement of second derivatives,

$$\mathcal{H}f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{bmatrix}$$

It is sometimes also written as $\nabla^2 f(\mathbf{x})$, and hessian can be computed by taking the Jacobian of the gradient,

$$\mathcal{H}f(\mathbf{x}) = \mathcal{J}^\top(\nabla f(\mathbf{x}))$$

Hessian matrix

$f: \mathbb{R}^n \longrightarrow \mathbb{R}$

scalar-valued vector function

$$H_{\underline{x}} f(\underline{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \\ \vdots & & & \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & & & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}_{n \times n} \qquad \underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i}$$ when the partial derivatives are smooth

$\Rightarrow H_x f(x)$ is symmetric $\quad H_x^T f(x) = H_x f(x)$

$$H_x f(x) = J_{\underline{x}}^T \left[\nabla_{\underline{x}} f(\underline{x}) \right]$$

Another notation

$\nabla_{\underline{x}}^2 f(\underline{x})$

Two rules of vector calculus

$$\boxed{\nabla_x^T (\underline{b}^T \underline{x}) = \frac{\partial}{\partial \underline{x}} \underline{b}^T \underline{x} = ? = \underline{b}^T}$$

$$\nabla_x^T \underline{x}^T A \underline{x} = \frac{\partial}{\partial \underline{x}} \underline{x}^T A \underline{x} = ? = \underline{x}^T (A + A^T)$$

$H_x f(x) = ?$ when $f(\underline{x}) = \underline{x}^T A \underline{x} + \underline{b}^T \underline{x} + c$

$$\nabla^T_{\underline{x}} f(x) = x^T(A + A^T) + \underline{b}^T + \underline{0}^T$$

$$\nabla_x f(x) = (A^T + A)\underline{x} + \underline{b}$$

$$H_{\underline{x}} f(\underline{x}) = J^T_{\underline{x}}\left[\nabla_x f(\underline{x})\right]$$

$$J_x g(\underline{x}) = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \frac{\partial g_2}{\partial x_1} & & \\ \vdots & & \vdots \\ \frac{\partial g_m}{\partial x_1} & & \frac{\partial g_m}{\partial x_n} \end{bmatrix}$$

$$J_{\underline{x}}[\underline{b}] = ? = 0_{n \times n}$$

$\underline{b} \in \mathbb{R}^n$

$\underline{x} \in \mathbb{R}^n$

$$J_{\underline{x}}[G\underline{x}] = J_x \underbrace{\begin{bmatrix} \underline{g}_1^T \\ \underline{g}_2^T \\ \vdots \\ \underline{g}_m^T \end{bmatrix}}_{G} \underline{x}$$

g_1^T is the first row

$$= J_x \begin{bmatrix} \underline{g}_1^T \underline{x} \\ \underline{g}_2^T \underline{x} \\ \vdots \\ \underline{g}_m^T \underline{x} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial \underline{x}} \underline{g}_1^T \underline{x} \\ \frac{\partial}{\partial \underline{x}} \underline{g}_2^T \underline{x} \\ \vdots \\ \frac{\partial}{\partial \underline{x}} \underline{g}_m^T \underline{x} \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} \underline{g}_1^T \\ \underline{g}_2^T \\ \vdots \\ \underline{g}_m^T \end{bmatrix}}_{\underline{G}}$$

$$\boxed{J_x\left[G\,\underline{x}\right] = \underline{G}}$$

$$\frac{\partial\, ax}{\partial x} = a$$

3rd rule of vector calculus

$$f(x) = \underline{x}^T A\underline{x} + \underline{b}^T\underline{x} + c$$

$$H_{\underline{x}}\, f(\underline{x}) = J_{\underline{x}}^T\left[\nabla_x f(\underline{x})\right]$$

$$\nabla_x f(x) = (A^T + A)\,\underline{x} + \underline{b}$$

$$= J_{\underline{x}}^T\left[(A^T+A)\underline{x} + \underline{b}\right]$$

$$= \left(\left(A^T+A\right) + O_{n\times n}\right)^T = A + A^T$$

Ex1

$$A_1 = \begin{bmatrix} 2 & -1/2 \\ -1/2 & 4 \end{bmatrix} \Rightarrow H_{\underline{x}} f_1(x) = \begin{bmatrix} 4 & -1 \\ -1 & 8 \end{bmatrix}$$

Ex2

$$A_2 = \begin{bmatrix} -2 & 1/2 \\ 1/2 & -4 \end{bmatrix} \Rightarrow H_x f_2(\underline{x}) = \begin{bmatrix} -4 & 1 \\ 1 & -8 \end{bmatrix}$$

Ex3

$$A_3 = \begin{bmatrix} 2 & -1/2 \\ -1/2 & -4 \end{bmatrix} \Rightarrow H_x f_3(\underline{x}) = \begin{bmatrix} 4 & -1 \\ -1 & -8 \end{bmatrix}$$

Positive definite $(A \succ 0)$

A square matrix $A \in \mathbb{R}^{n \times n}$ is called +ve definite iff $\underline{x}^T A \underline{x} > 0$ for all $\underline{x} \in \mathbb{R}^n$

Negative definite $(A \prec 0)$

$\underline{x}^T A \underline{x} < 0$ for all x

Indefinite if neither of the above is true

Eigen vectors and eigen values of a square matrix A

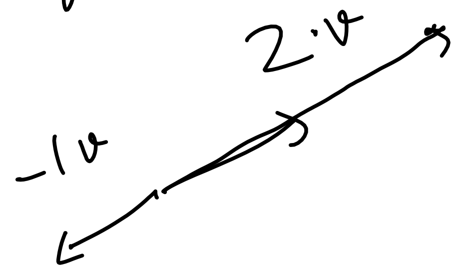
$\underline{v} \in \mathbb{R}^n$ $\quad \lambda \in \mathbb{R}$

unit vector

$A \in \mathbb{R}^{n \times n}$

are the solutions to the equation

$$A\underline{v} = \lambda \underline{v}$$



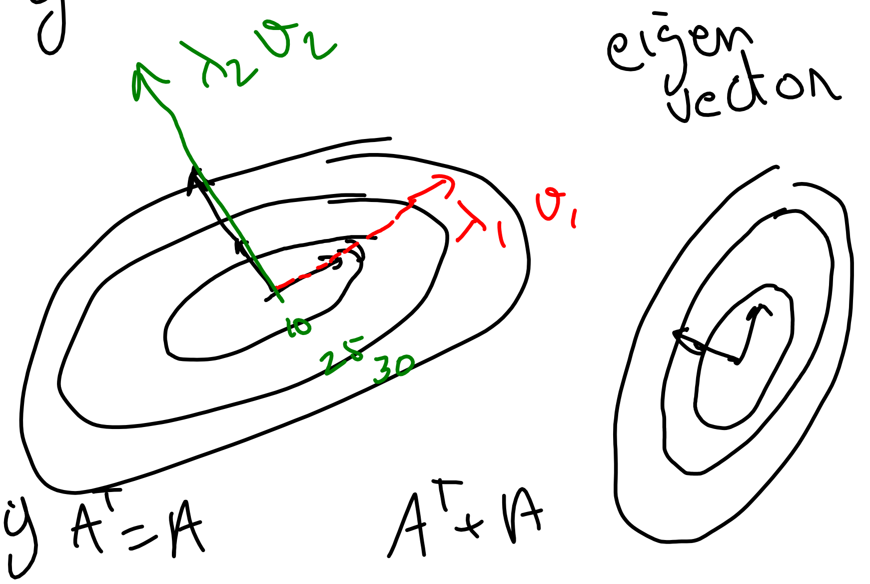
$$\Rightarrow A\underline{v} - \lambda \underline{v} = 0$$

$$\Rightarrow \underbrace{(A - \lambda I_n)\underline{v} = 0}_{\text{This system of eqns have a solution if } \det(A - \lambda I_n) = 0}$$

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & & \\ 0 & & \ddots & \\ 0 & & & 1 \end{bmatrix}_{n \times n}$$

$\lambda_1 > 0$

$\lambda_2 > 0$



$\underline{H_x f(x) = 2A}$ if $A^T = A$ $\quad A^T + A$

$$\left.\begin{matrix} \underline{v_1}, \underline{v_2} \\ \lambda_1, \lambda_2 \end{matrix}\right] = \text{eig}(2A)$$

λ_1, λ_2 curvatures of f along the direction $\underline{v_1}$ and $\underline{v_2}$

Proof of this next time

$A \in \mathbb{R}^{n \times n}$ is P.D. (+ve definite)

if all the eigen values are +ve

$A \succ 0$ if $\lambda_i > 0$ for all eigen values λ_i

$A \prec 0$ if $\lambda_i < 0$

A is N.D.

---

if some $\lambda_i < 0$ and other $\lambda_j > 0$

the A is indefinite

If $H_{\underline{x}} f(\underline{x}) \succ 0$ (Hessian is P.D)

then the extremum is a minima

If $H_x f(\underline{x}) \prec 0$ (Hessian is N.D)

then the extremum is a maxima

If $H_x f(x)$ is indefinite then the extremum is a saddle point

If the second partial derivatives are continuous then the Hessian matrix is symmetric.

Find the Hessian of the general quadratic form,

$$f(\mathbf{x}) = \mathbf{x}^\top A\mathbf{x} + \backslash\mathrm{bfb}^\top \mathbf{x} + c$$

Find the gradient of $f(\mathbf{x})$

$$\nabla^\top f(\mathbf{x}) = \mathbf{x}^\top (A + A^\top) + \backslash\mathrm{bfb}^\top$$

Take transpose

$$\nabla f(\mathbf{x}) = (A + A^\top)\mathbf{x} + \backslash\mathrm{bfb}$$

Find the Jacobian of the gradient

$$\mathcal{J}^\top \nabla f(\mathbf{x}) = (A + A^\top)$$

Homework 4: Problem 1

Find the Hessian of the quadratic function that we got as the objective function in linear regression,

$$R(\mathbf{m}) = \mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top \mathbf{X}\mathbf{m} + \mathbf{m}^\top \mathbf{m}$$

Find $\mathcal{H}_{\mathbf{m}} R(\mathbf{m})$

Positive definite, Negative definite and Indefinite

Positiive definite

A square matrix $A \in \mathbb{R}^{n \times n}$ is called positive definite if for all $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{x}^\top A\mathbf{x} \succ 0$.

Negative definite

A square matrix $A \in \mathbb{R}^{n \times n}$ is called negative definite if for all $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{x}^\top A\mathbf{x} \prec 0$.

Indefinite

A square matrix $A \in \mathbb{R}^{n \times n}$ is called indefinite if it is neither positive definite nor negative definite.

Eigenvalues and Eigen vectors

Eigen values $\lambda \in \mathbb{R}$ and eigen vector $\textcolor{red}{\backslash\text{bfv}} \in \mathbb{R}^n$ of a given matrix $\textcolor{red}{\backslash\text{bfA}}$ are the solutions of the equation,

$$A\textcolor{red}{\backslash\text{bfv}} = \lambda\textcolor{red}{\backslash\text{bfv}}$$

You might have solved for eigen values and eigen vectors using the equation

$$(A - \lambda I_n)\textcolor{red}{\backslash\text{bfv}} = 0$$

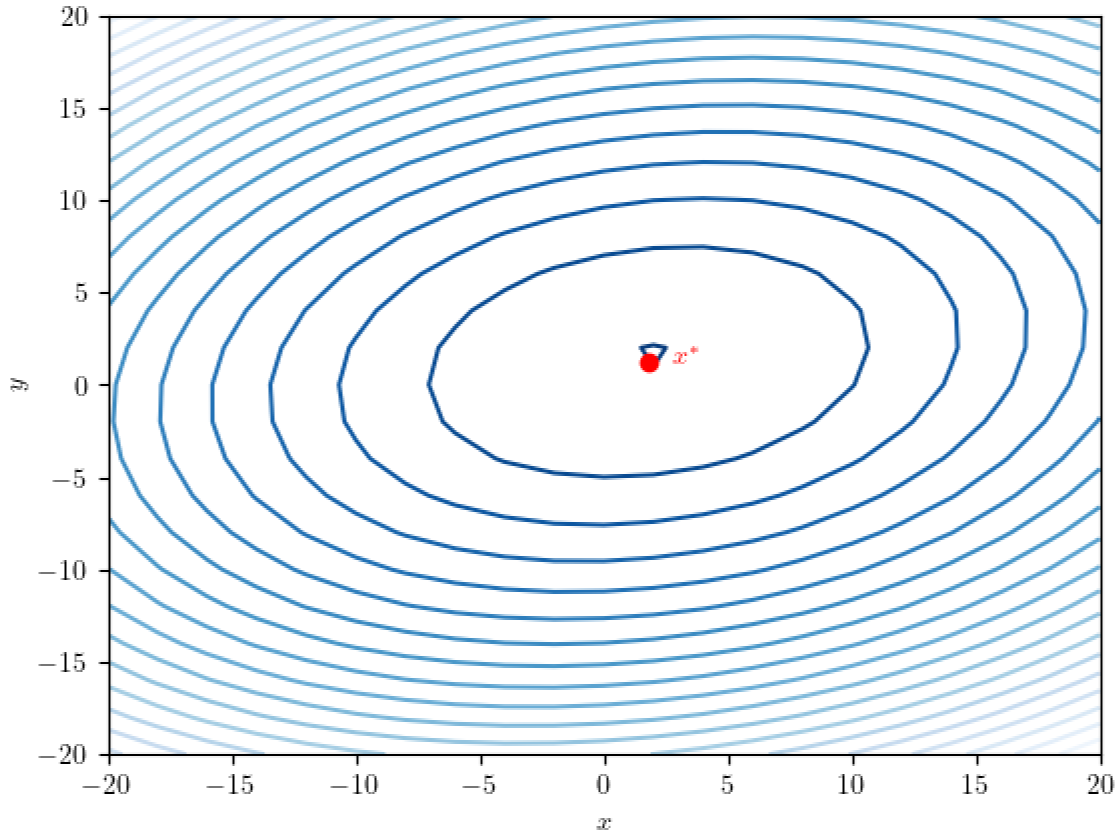
whose solution is given by,

$$\det(A - \lambda I_n) = 0$$

Contour Plots

In [41]:
```
def plot_contour(func):
    x, y = np.mgrid[-20:20:21j,
                    -20:20:21j]
    bfx = np.array([x, y])
    f = func(x,y)

    plt.contour(x, y, f, 20, cmap='Blues_r')
    plt.plot([1.8], [1.2], 'ro')
    plt.text(1.8+1, 1.2, '$x^*$', color='r')
    plt.xlabel('$x$')
    plt.ylabel('$y$')
    plt.show()
```

In [42]:
```
def f(x, y): return  2*x**2 + 4*y**2 - x*y - 6*x - 8*y  + 6
plot_contour(f)
```

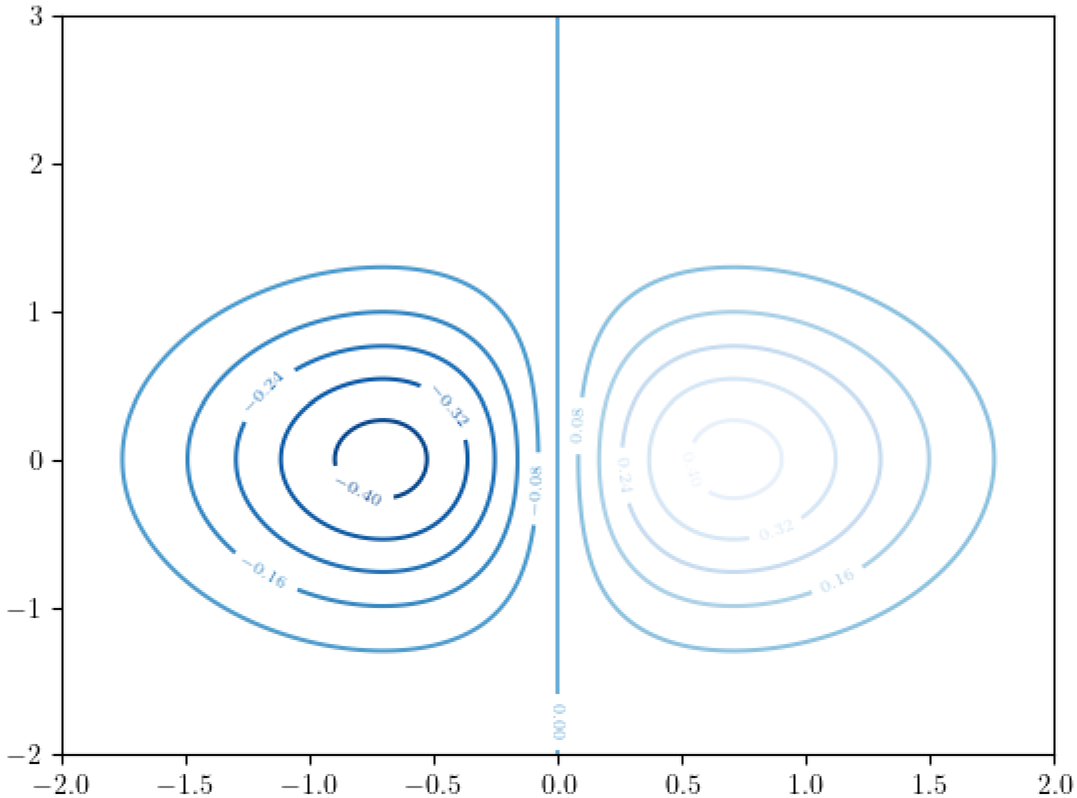


But how about other kinds of functions say:

$$\arg \min_{x} f(x) = x \exp(-(x^2 + y^2))$$

In [43]:
```
def plot_contour(func):
    x, y = np.mgrid[-2:2:201j,
                    -2:3:201j]
    f = func(x,y)

    ctr = plt.contour(x, y, f, 10, cmap='Blues_r')
    plt.clabel(ctr, ctr.levels, inline=True, fontsize=6)
    plt.show()
```

In [44]:
```
def f(x,y): return  x * np.exp(-(x**2 + y**2))
plot_contour(f)
```



In [45]:
```
def plot_surface_3d(func):
    x, y = np.mgrid[-2:2:201j,
                    -2:3:201j]
    f = func(x,y)
    fig = go.Figure(data=[go.Surface(z=f, x=x, y=y,
                                     contours = {
                                         "x": {"start": -2, "end": 2, "size":
                                         "z": {"start": -2, "end": 2, "size":
                                     },
                                     )])
    fig.update_traces(contours_z=dict(show=True, usecolormap=True, project_z
    fig.show()
```

In [46]:
```
plot_surface_3d(f)
```

Geometry of eigen vectors and eigen values

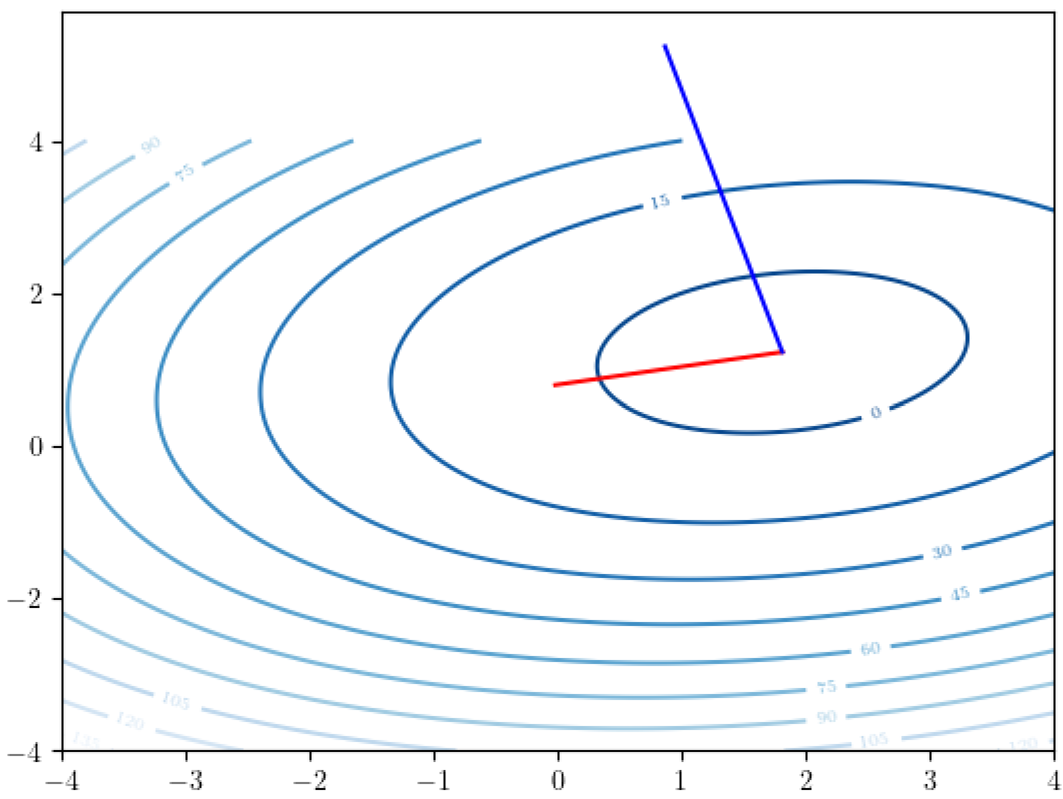
Example 1:

$$f(x, y) = 2x^2 + 4y^2 - xy - 6x - 8y + 6$$

$$f([x, y]) = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 2 & -1/2 \\ -1/2 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} -6 & -8 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + 6$$

In [57]:
```
def f(x, y):
    return 2*x**2 + 4*y**2 - x*y - 6*x - 8*y  + 6

def f_vec(x, y):
    # x is n x n and y is n x n
    xn = x[..., None] # n x n x 1
    yn = y[..., None] # n x n x 1
    vecx = np.concatenate([xn, yn], axis=-1) # n x n x 2
    vecx_col_vec = vecx[..., None] # n x n x 2 x 1
    vecx_row_vec = vecx[..., None, :] # n x n x 1 x 2
    A = np.array([[2, -0.5],
                  [-0.5, 4]]) # 2 x 2
```

```
    b = np.array([-6, -8]) # 2
    c = 6
    print("Minima at, ", -np.linalg.inv(A + A.T) @ b)
    quad  = (vecx_row_vec @ A @ vecx_col_vec).squeeze(-1).squeeze(-1)
    return quad + vecx @ b + c

x, y = np.mgrid[-4:4:201j,
                -4:4:201j]
fvals = f(x,y)


A = np.array([[2, -0.5],
              [-0.5, 4]]) # 2 x 2

b = np.array([-6, -8]) # 2
minpt = -np.linalg.inv(A + A.T) @ b
ctr = plt.contour(x, y, fvals, 10, cmap='Blues_r')
lambdas, vs = np.linalg.eigh(A)
plt.plot([minpt[0], minpt[0]+lambdas[0] * vs[0, 0]], [minpt[1], minpt[1]+ la
plt.plot([minpt[0], minpt[0]+ lambdas[1] * vs[0, 1]], [minpt[1], minpt[1]+ l
plt.clabel(ctr, ctr.levels, inline=True, fontsize=6)
plt.show()
```



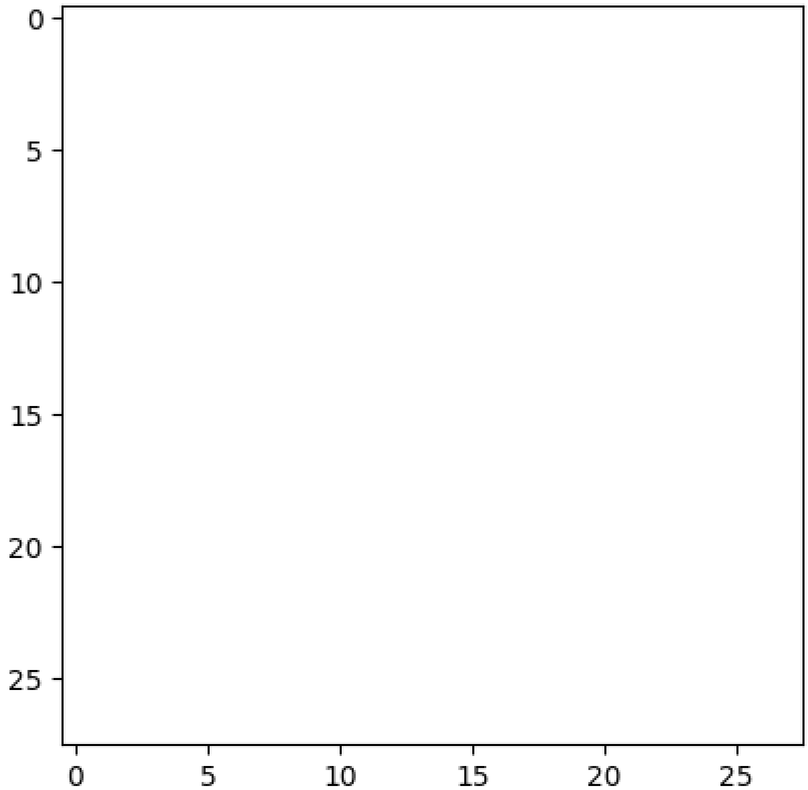
In [ ]:

In [16]:
```
!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
```

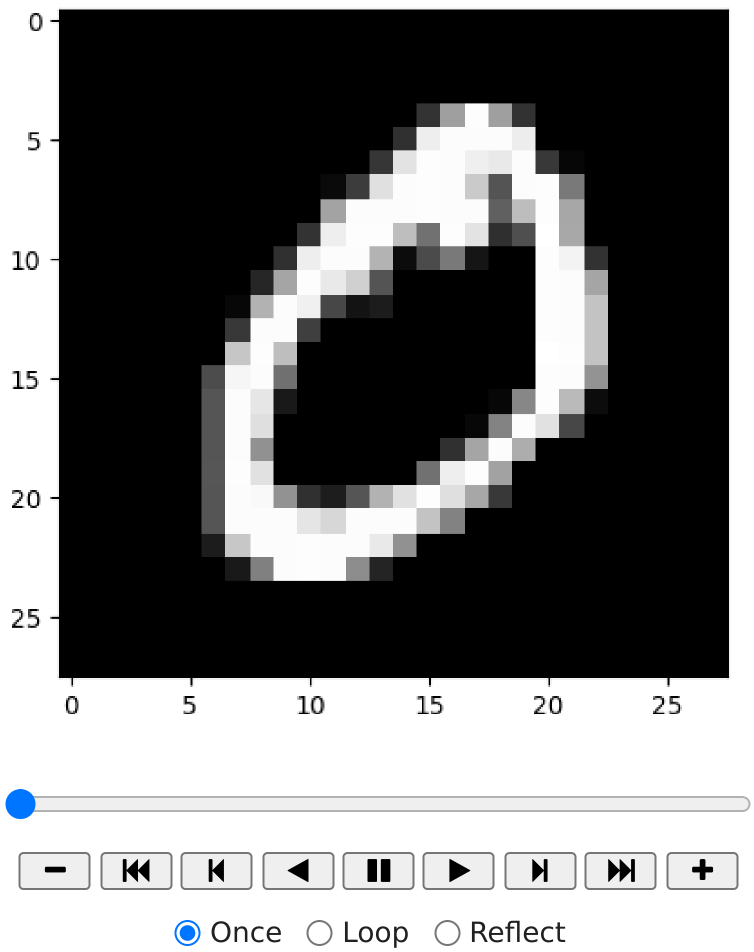
In [17]:
```python
import struct
import numpy as np

# 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 [18]:
```python
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(zero_images[i], animated=True, cmap='gray', vmin=0, vmax=2
    for i in range(60)]
zero_images_anim = animation.ArtistAnimation(fig, ims, interval=50, blit=Tru
                                repeat_delay=1000, repeat=False)
```



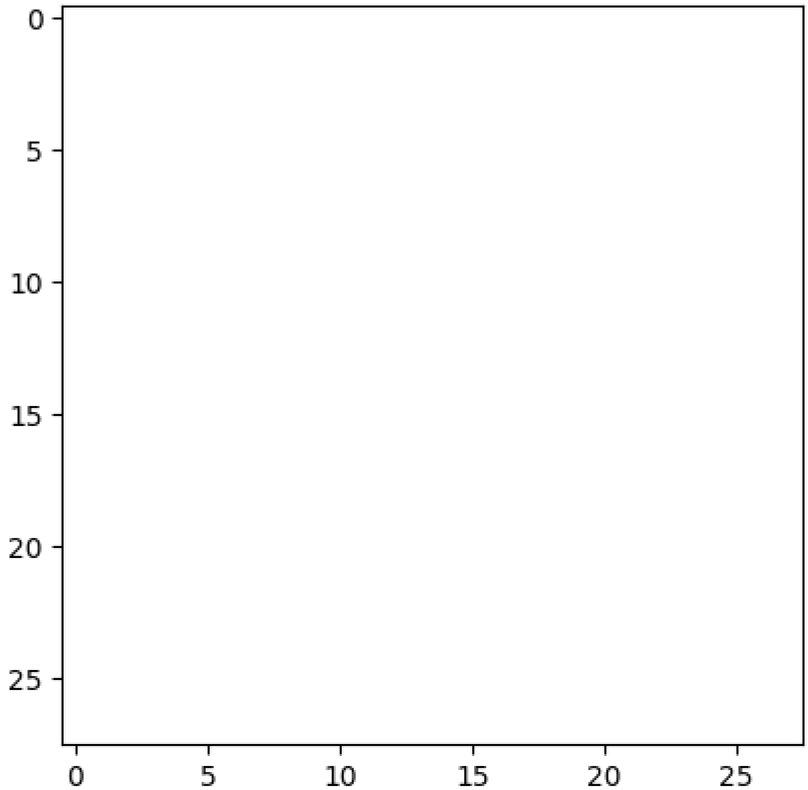
In [19]:
```
zero_images_anim
```

Out[19]:



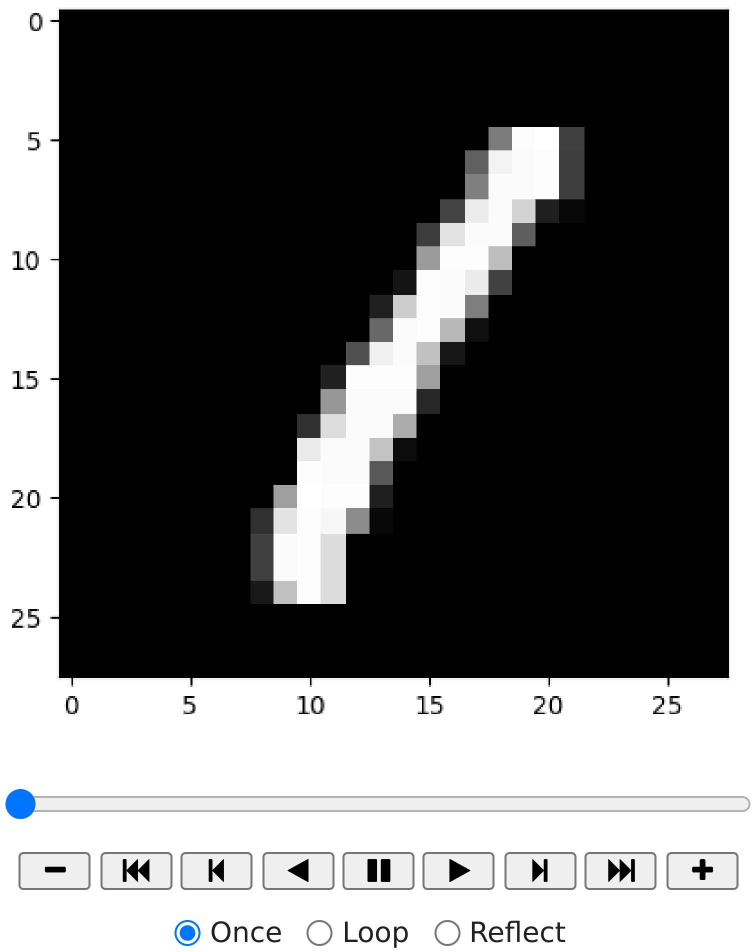
In [20]:

```
fig, ax = plt.subplots()
oneims = [[ax.imshow(one_images[i], animated=True, cmap='gray', vmin=0, vmax
    for i in range(60)]
one_images_anim = animation.ArtistAnimation(fig, oneims, interval=50, blit=T
                                            repeat_delay=1000, repeat=False)
```



In [21]:
```
one_images_anim
```

Out[21]:

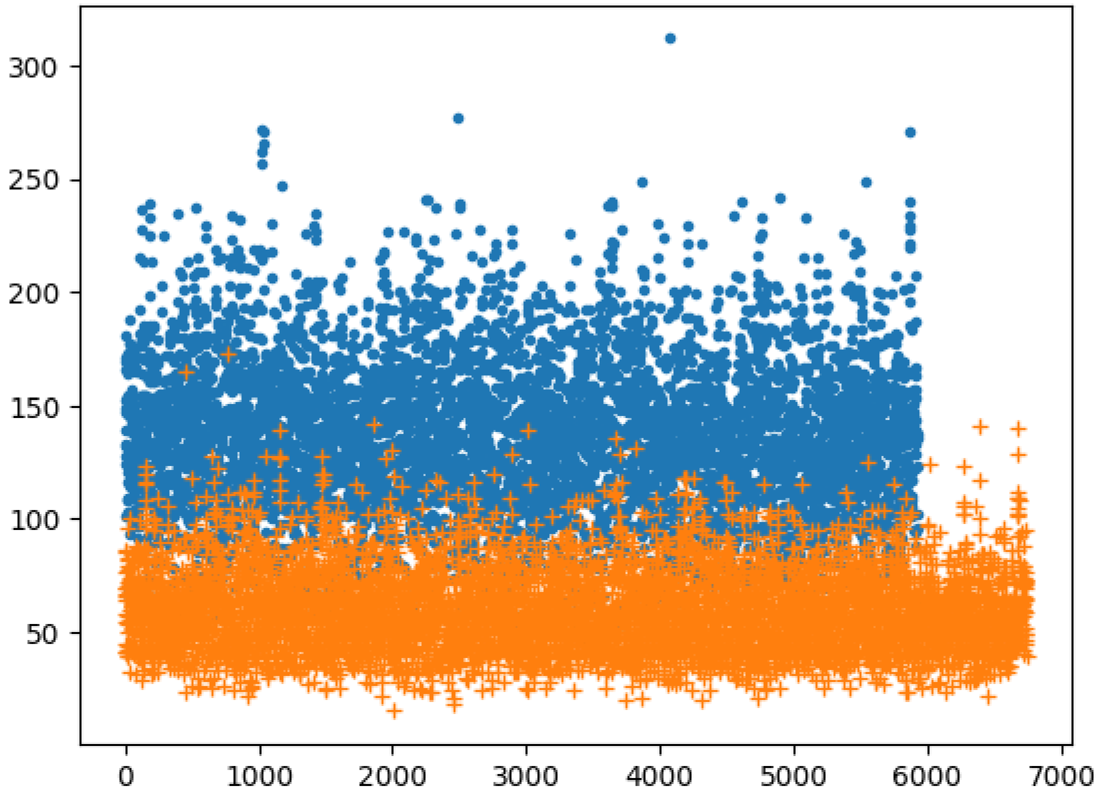


What is a feature

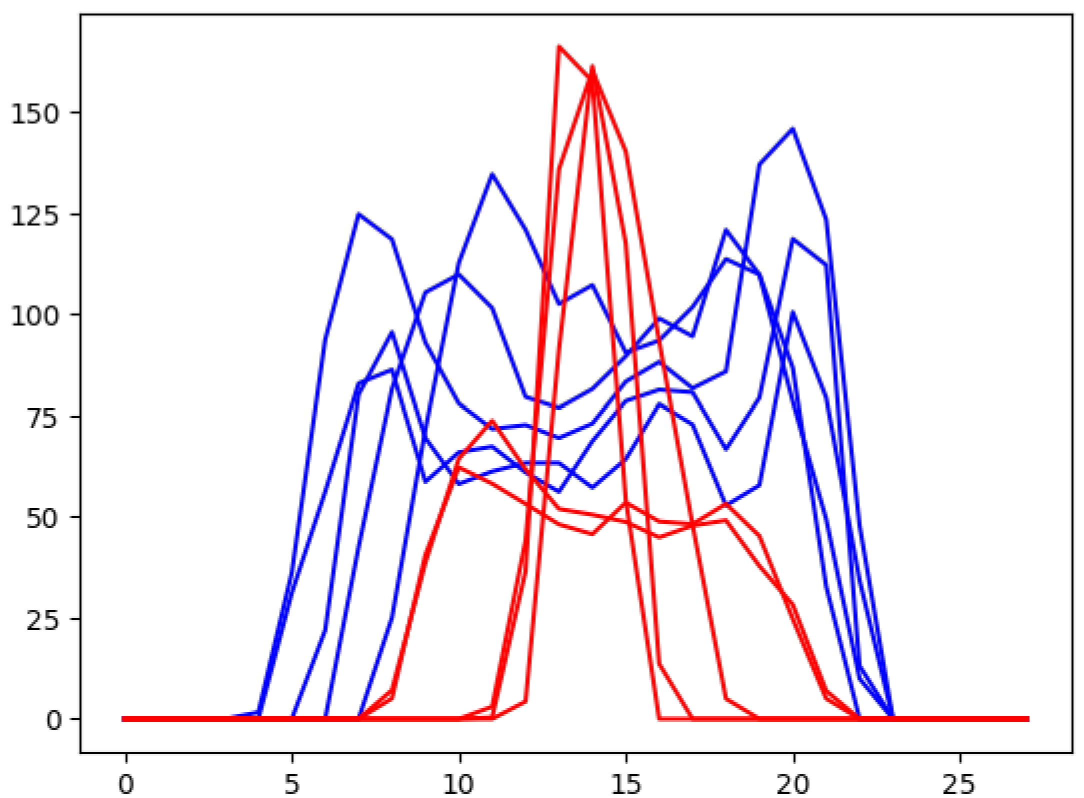
Any property of data sample that helps with the task.

In [22]:
```python
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[22]:
```
[<matplotlib.lines.Line2D at 0x7f3afa8327d0>]
```



In [23]:
```
fig, ax = plt.subplots()
for i in range(5):
    ax.plot(zero_images[i].mean(axis=0), 'b-')
for i in range(5):
    plt.plot(one_images[i].mean(axis=0), 'r-')
```



In [24]:
```python
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
```

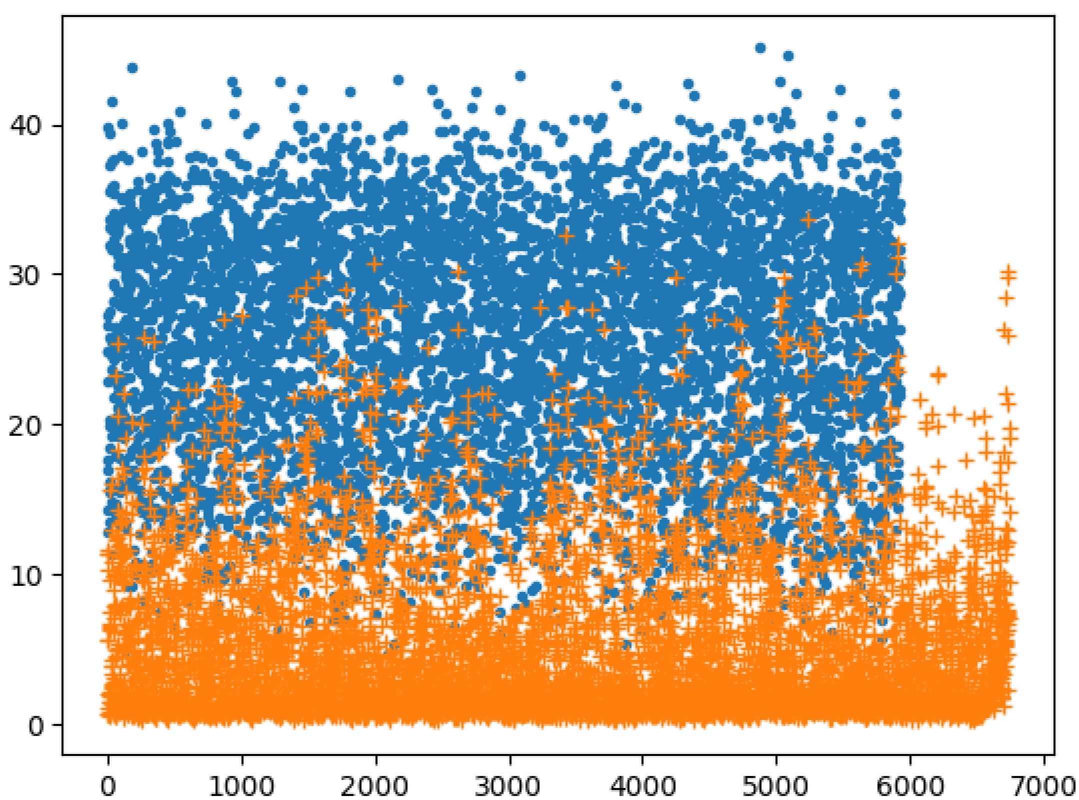
Out[24]: 22.811061800377757

In [25]:
```python
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[25]: (22.811061800377757, 11.384958735403274)

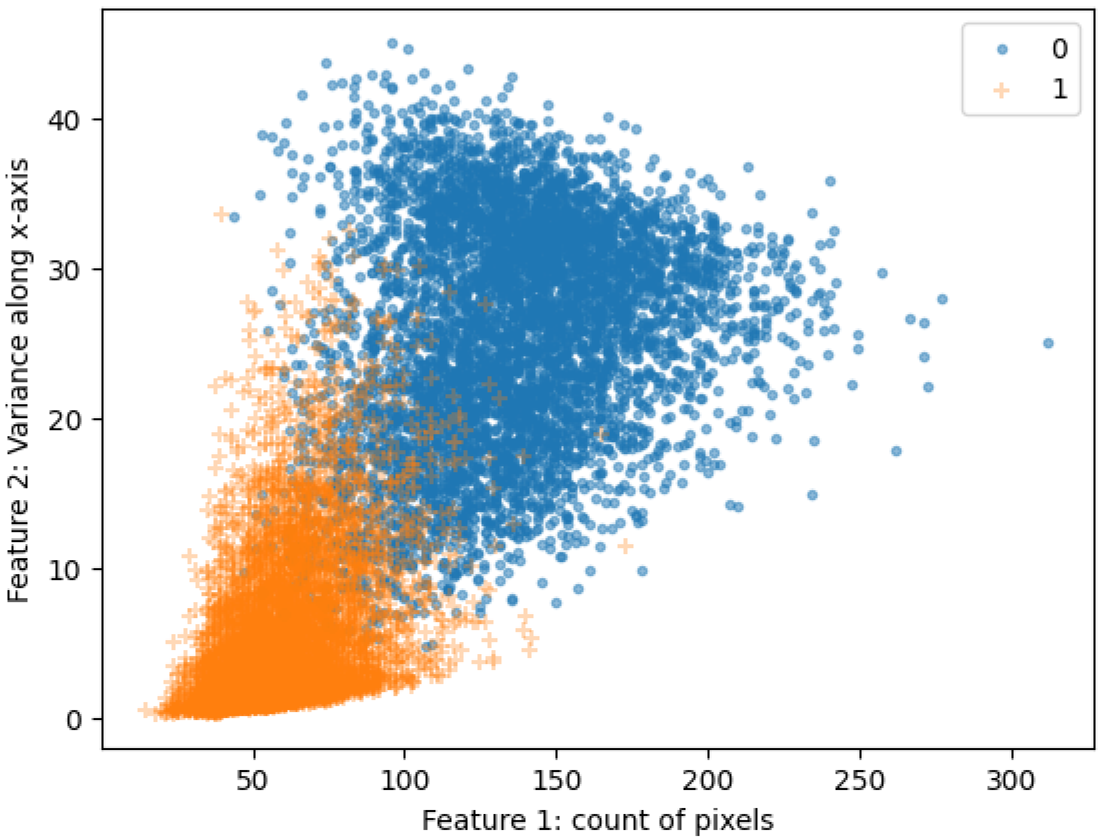
In [26]:
```python
def feature_y_var(imgs):
    wts = imgs.mean(axis=-2)
    arange = np.arange(wts.shape[-1])
    mean = (arange * wts).sum(axis=-1) / wts.sum(axis=-1)
    mean = mean[:, np.newaxis]
    var = ((arange - mean)**2 * wts).sum(axis=-1) / wts.sum(axis=-1)
    return var

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

Out[26]: [<matplotlib.lines.Line2D at 0x7f3afa64cac0>]



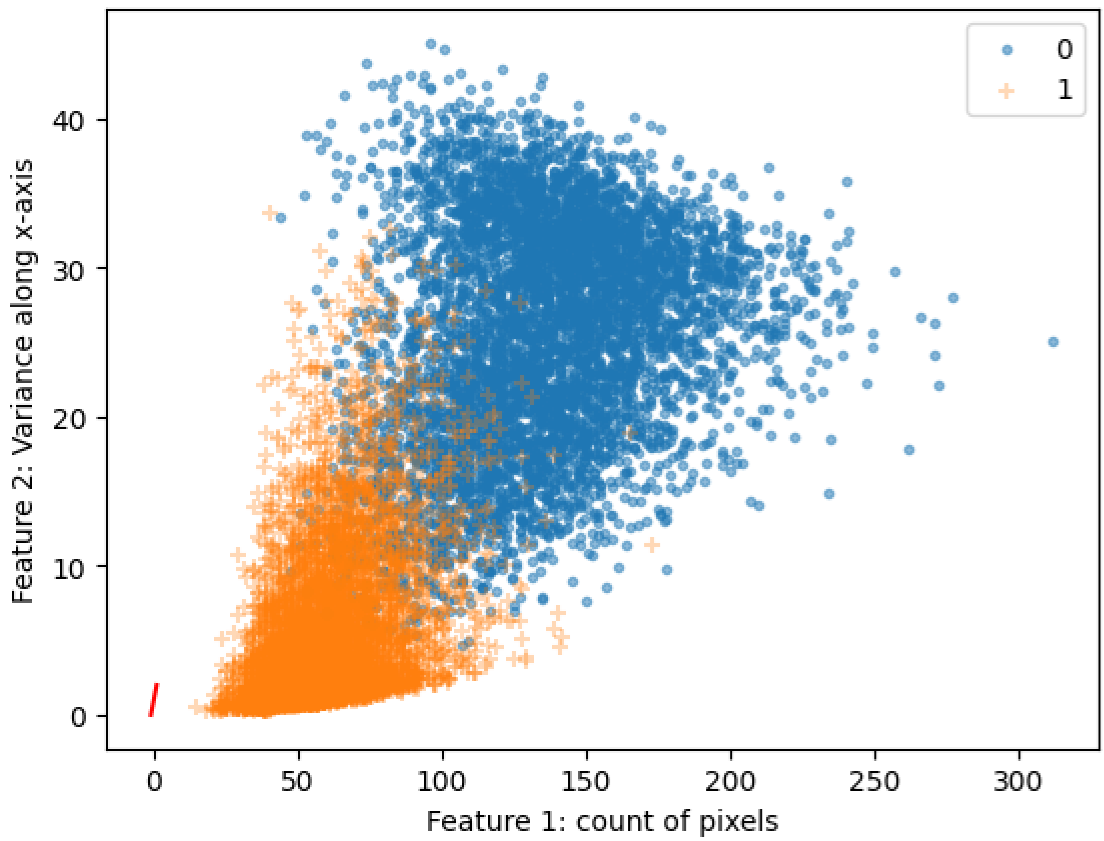
In [27]:
```python
def features_extract(images):
    return np.vstack((feature_n_pxls(images),
                      feature_y_var(images))).T
zero_features = features_extract(zero_images)
one_features = features_extract(one_images)


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 [28]:
```python
fig, ax = plt.subplots()
draw_features(ax, zero_features, one_features)
plt.show()
```



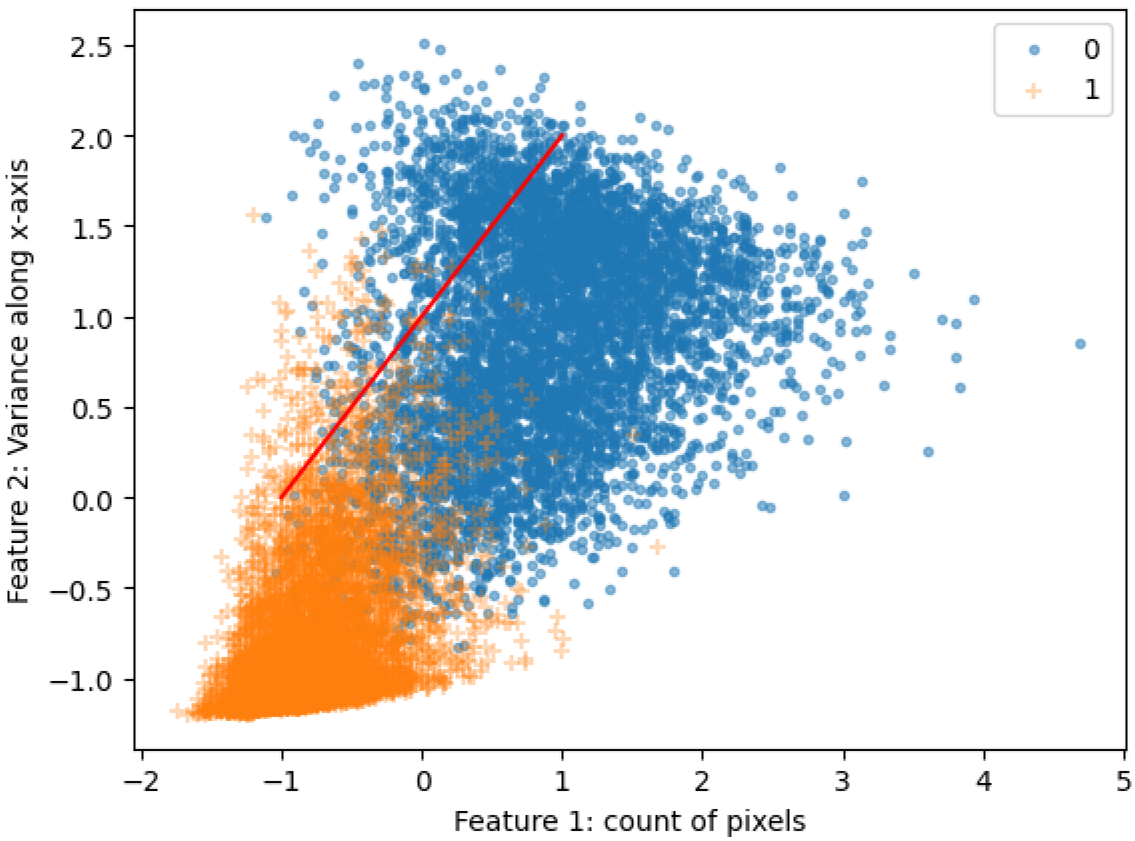
In [29]:
```python
bfm = np.ones(2)
fig, ax = plt.subplots()
draw_features(ax, zero_features, one_features)
x = np.linspace(-1, 1, 21)
ax.plot(x, x*bfm[0] + bfm[1], 'r-')
```

Out[29]: [<matplotlib.lines.Line2D at 0x7f3afae3d180>]



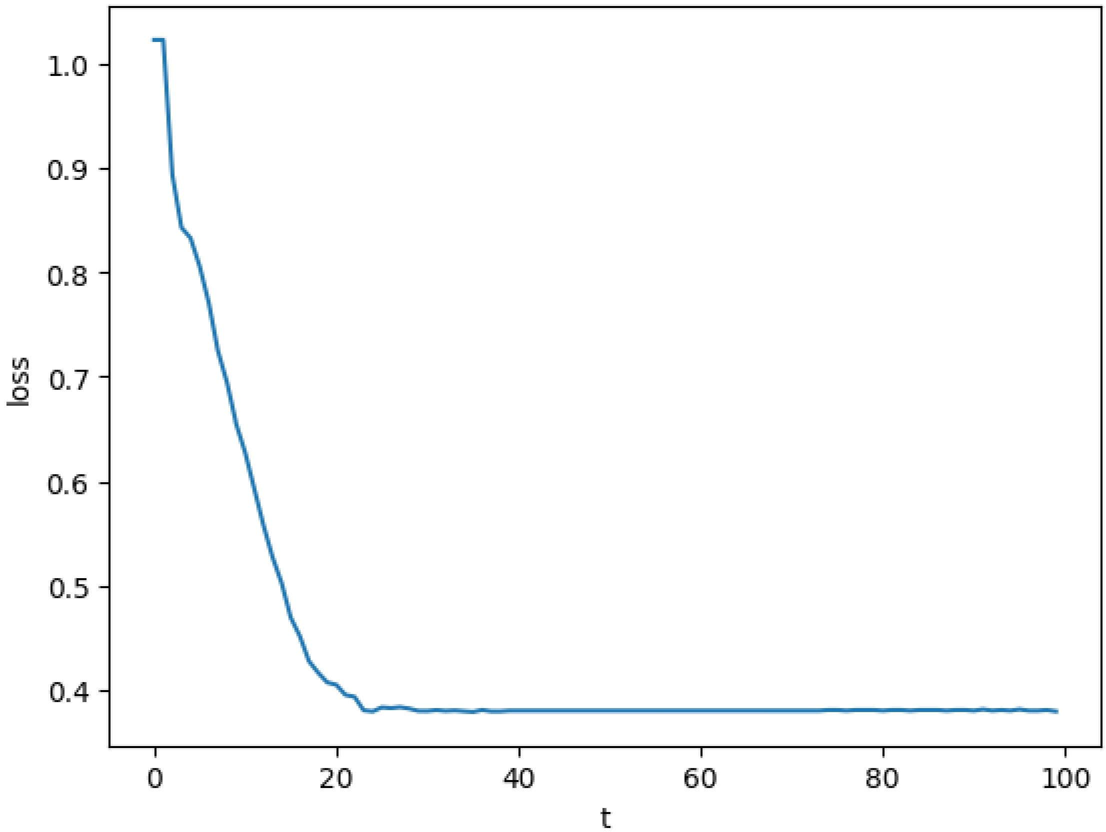
In [30]:
```
bfm = np.ones(2)

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)
np.savez('features_stats.npz', mean=FEATURES_MEAN, std=FEATURES_STD)

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

X = norm_features(features)

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[30]:
```
[<matplotlib.lines.Line2D at 0x7f3afae3f460>]
```



In [31]:
```
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 [33]:

```
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 = []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

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)
        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

a = Anim(fig, axes, X, Y)
```

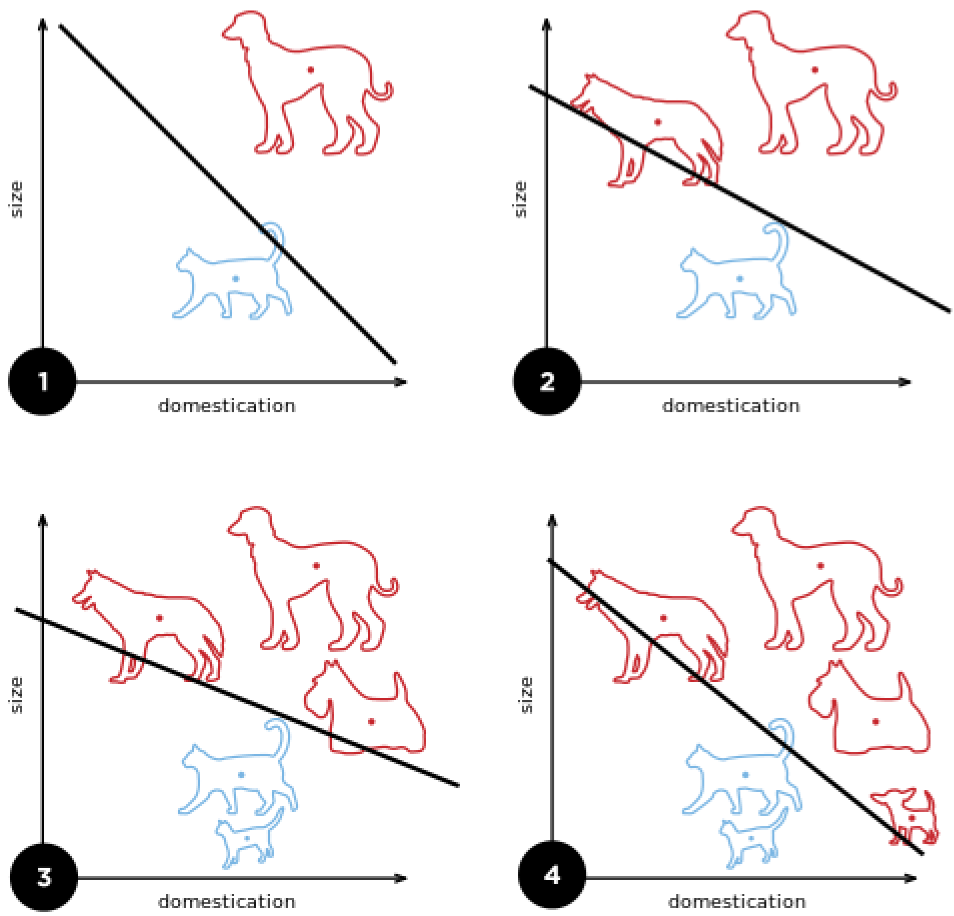
```
animation.FuncAnimation(fig, a.update, frames=list_of_bfms[::3],
                        init_func=a.anim_init, blit=True, repeat=False)
```

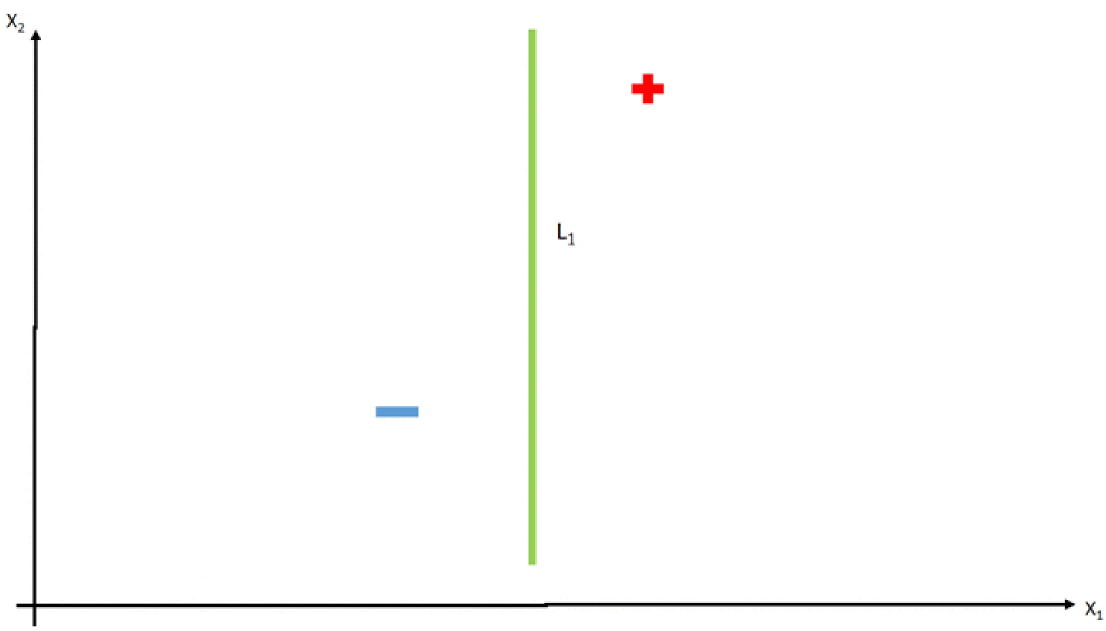
```
  File <tokenize>:64
    def anim_init(self):
    ^
IndentationError: unindent does not match any outer indentation level
```

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

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 [ ]:
```
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)
```

Perceptron


size
domestication
1
2
3
4


X_2
L_1
X_1