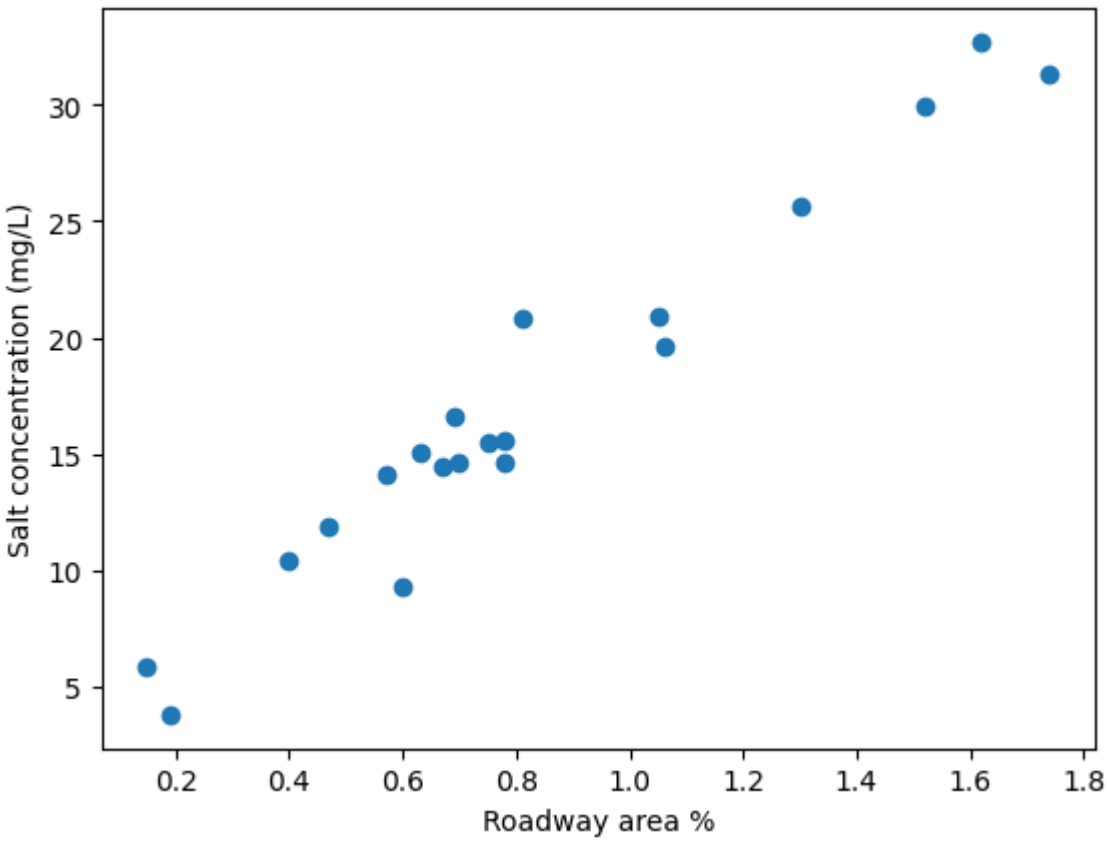


Least squares regression

The problem of linear regression is to find a line that "best fits" the given data. That is we want all the points $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ to satisfy the equation of the line $y = mx + c$. Since we know that there exists no such line, so we will try to make $y \approxeq mx + c$, by minimizing some error/distance/cost/loss function between y and $mx + c$ for every point (x_i, y_i) in the dataset. The simplest error function that results in nice answers is squared distance:

$$e(x_i, y_i) = (y_i - (mx_i + c))^2$$

Then we can minimize the total error to find the line:

$$m^*, c^* = \arg \min_{m,c} \sum_{i=1}^{n} e(x_i, y_i)$$

Geometrically, this error minimization corresponds to minimizing the stubs in the following figure:

Vectorization of Least square regression

Recall that the magnitude of a vector $\|\mathbf{v}\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$ has a similar form to the error function. This suggests that we can define an error vector with the signed error for each data point as it's elements

$$\mathbf{e} = \begin{bmatrix} y_1 - (mx_1 + c) \\ y_2 - (mx_2 + c) \\ \vdots \\ y_n - (mx_n + c) \end{bmatrix}$$

Minimizing the total error is same as minimizing the square of error vector magnitude

$$m^*, c^* = \arg \min_{m,c} \|\mathbf{e}\|^2$$

While we are at at it let us define $\mathbf{x} = [x_1; \ldots; x_n]$ to denote the vector of all x coordinates of the dataset and $\mathbf{y} = [y_1; \ldots; y_n]$ to denote y coordinates. Then the error vector is:

$$\mathbf{e} = \mathbf{y} - (\mathbf{x}m + \mathbf{1}_n c)$$

where $\mathbf{1}_n$ is a n-D vector of all ones. Finally, we vectorize parameters of the line $\mathbf{m} = [m; c]$. We will also need to horizontally concatenate $\mathbf{x}$ and $\mathbf{1}_n$. Let's call the result $\mathbf{X} = [\mathbf{x}, \mathbf{1}_n] \in \mathbb{R}^{n \times 2}$. Now, the error vector looks like this:

$$\mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{m}$$

Expanding the error magnitude:

$$\begin{aligned} \|\mathbf{e}\|^2 &= (\mathbf{y} - \mathbf{X}\mathbf{m})^\top (\mathbf{y} - \mathbf{X}\mathbf{m}) \\ &= \mathbf{y}^\top \mathbf{y} + \mathbf{m}^\top \mathbf{X}^\top \mathbf{X}\mathbf{m} - 2\mathbf{y}^\top \mathbf{X}\mathbf{m} \end{aligned}$$

Homework 3: Problem 4

Expand

$$(\mathbf{y} - \mathbf{X}\mathbf{m})^\top (\mathbf{y} - \mathbf{X}\mathbf{m})$$

and show that it is equal to

$$\mathbf{y}^\top \mathbf{y} + \mathbf{m}^\top \mathbf{X}^\top \mathbf{X}\mathbf{m} - 2\mathbf{y}^\top \mathbf{X}\mathbf{m}$$

Our minimization problem in vectorized form is:

$$\mathbf{m}^* = \arg \min_{\mathbf{m}} \mathbf{y}^\top \mathbf{y} + \mathbf{m}^\top \mathbf{X}^\top \mathbf{X} \mathbf{m} - 2\mathbf{y}^\top \mathbf{X} \mathbf{m}$$

This is a quadratic equation in $\mathbf{m}$ that can be minimized by equating the derivate to zero.

Two rules of vector derivatives

There are two conventions in vector derivatives:

1. Gradient convention
2. Jacobian convention

Gradient convention

Under gradient convention the derivative of scalar-valued vector function function $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ is defined as vertical stacking of element-wise derivatives

$$\frac{\partial}{\partial \mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_n} \end{bmatrix} \in \mathbb{R}^n$$

Jacobian convention

Under gradient convention the derivative of scalar-valued vector function function $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ is defined as horizontal stacking of element-wise derivatives

$$\frac{\partial}{\partial \mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} & \dots & \frac{\partial f(\mathbf{x})}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{1 \times n}$$

For a vector-value vector function $\mathbf{f}(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^m$, Jacobian of $\mathbf{f}(\mathbf{x})$ is the vertical concatentation of gradients transposed, resulting in $m \times n$ matrix

$$\mathbf{J}_{\mathbf{x}}(\mathbf{f}(\mathbf{x})) = \frac{\partial}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial \mathbf{x}} \\ \dots \\ \frac{\partial f_m(\mathbf{x})}{\partial \mathbf{x}} \end{bmatrix}$$

We will use Jacobian convention in this course, because it works nicely with chain rule.

Derivative of a linear function

All scalar-valued linear functions of $\mathbf{x}$ can be written in the form $f(\mathbf{x}) = \bfc^\top \mathbf{x}$.

$$\frac{\partial}{\partial \mathbf{x}} \mathbf{c}^\top \mathbf{x} = \mathbf{c}^\top \tag{10}$$

Derivative of a quadratic function

All scalar-valued homogeneous quadratic functions of $\mathbf{x}$ can be written in the form $f(\mathbf{x}) = \mathbf{x}^\top \mathbf{A} \mathbf{x}$.

$$\frac{\partial}{\partial \mathbf{x}} \mathbf{x}^\top \mathbf{A} \mathbf{x} = \mathbf{x}^\top (\mathbf{A} + \mathbf{A}^\top) \tag{11}$$

Homework 3: Problem 5

Proof of above two derivatives is left as an exercises.

Back to Least square regression

$$\mathbf{0}^\top = \frac{\partial}{\partial \mathbf{m}} (\mathbf{y}^\top \mathbf{y} + \mathbf{m}^\top \mathbf{X}^\top \mathbf{X} \mathbf{m} - 2\mathbf{y}^\top \mathbf{X} \mathbf{m}) \tag{12}$$

$$= 2\mathbf{m}^{*\top} \mathbf{X}^\top \mathbf{X} - 2\mathbf{y}^\top \mathbf{X} \tag{13}$$

This gives us the solution

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

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 [26]:
```
n = salt_concentration_data.shape[0]
bfx = salt_concentration_data[:, 2:3]
bfy = salt_concentration_data[:, 1]
bfX = np.hstack((bfx, np.ones((bfx.shape[0], 1))))
bfX
```

Out[26]:
```
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.  ]])
```

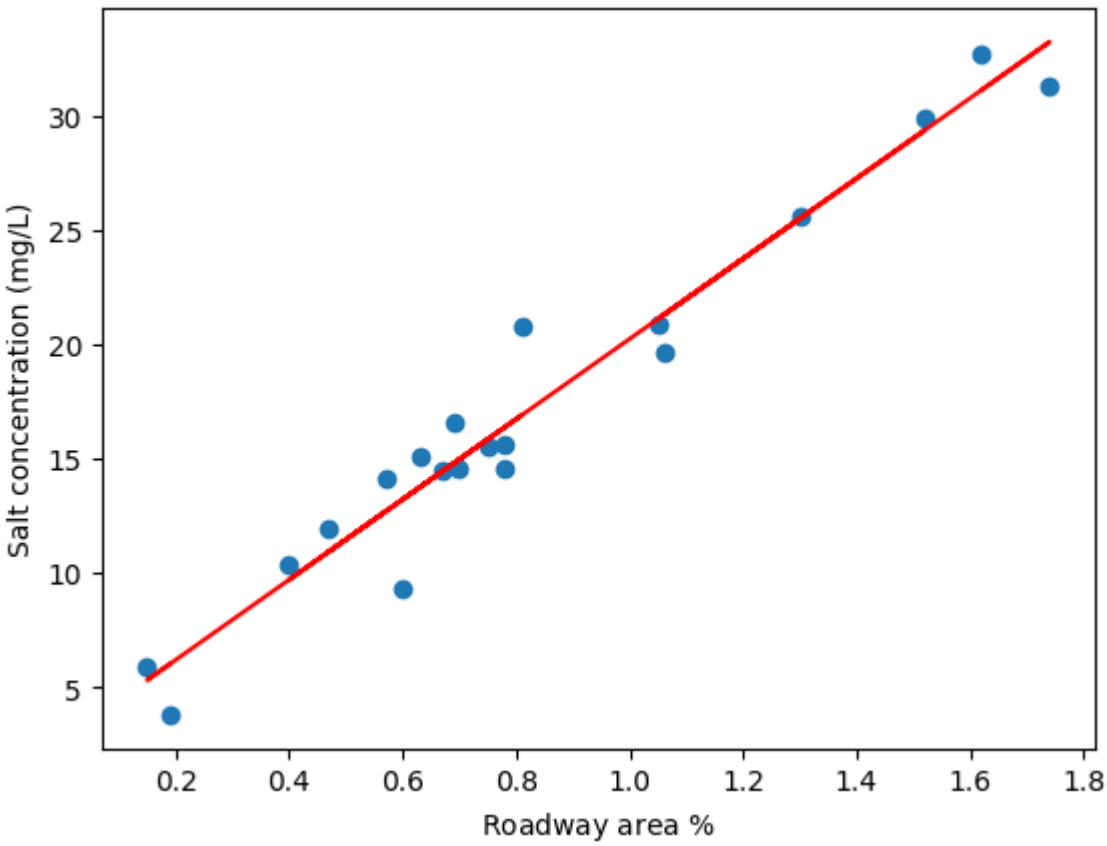
In [27]:
```
bfm = np.linalg.inv(bfX.T @ bfX) @ bfX.T @ bfy
print(bfm)
bfm, *_ = np.linalg.lstsq(bfX, bfy, rcond=None)
print(bfm)
```

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

In [28]:
```
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[28]:
```
[<matplotlib.lines.Line2D at 0x7fd3ed57a4a0>]
```


Salt concentration (mg/L)
30
25
20
15
10
5
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
Roadway area %