Machine Learning and Data Mining I: Lecture 2
Linear Regression

Morgan McCarty

05 July 2023

1 Problem Setup

- We are going to predict the market value of a house.
- Our Task: predict the value of a house given the number of rooms.
- The x-axis: will be the input feature values (in this case the number of rooms).
- The y-axis: will be the output/target values (in this case the price of the house).
- A domain expert says this model should be linear.

2 The Model

- We want to find a function that can map our input (a house and it's size) to our output (the house's price).
- To do this we use a linear model.
  For example:

$$y = w_0 + w_1 x \tag{1}$$

- w_0 is the bias term (the y-intercept).
- w_1 is the weight of the feature (the slope). This feature is the number of rooms.
- We need to find the best values for w_0 and w_1 so that our model fits the data.
- Our dataset contains N examples, as such we can write N equations:

$$\begin{aligned} y_1 &= w_0 + w_1 x_1 \\ y_2 &= w_0 + w_1 x_2 \\ &\vdots \\ y_N &= w_0 + w_1 x_N \end{aligned}$$

- This system of equations is overdetermined (we have more equations than unknowns). As such we cannot find a solution that satisfies all of the equations, but we can find a good approximation.

- How can we characterize what approximation is better than another?
- Assume we have a line l_1 and a line l_2 that both fit the data. Which line is better?
- Both lines will have an error value for each point. We can sum these errors to get a total error for each line.
- However we can't just sum the errors because some errors will be positive and some will be negative.
- There are two common ways to deal with this:
  - Sum the absolute values of the errors (L1 norm)
  - Sum the squares of the errors (L2 norm)
- The L2 norm is more common because it is easier to work with.

2.1 The Loss Function

- We will write an equation which represents the ith observation in our dataset.

$$\hat{y}_i = w_0 + w_1 x_i \tag{2}$$

- The error for the ith observation is:

$$e_i = (\hat{y}_i - y_i)^2 = (w_0 + w_1 x_i - y_i)^2 \tag{3}$$

- The total error is (the sum of the errors for each observation / the sum of the squared errors):

$$E(w) = \sum_{i=1}^{N} e_i = \sum_{i=1}^{N} (w_0 + w_1 x_i - y_i)^2 \tag{4}$$

2.1.1 Minimizing the Loss

- We are looking for the values of w_0 and w_1 that minimize the loss function:

$$\underset{w_0, w_1}{\operatorname{argmin}} E(w) \tag{5}$$

- We can find the minimum of a function by taking the derivative and setting it equal to zero.
- Consider the example:

$$f(x) = 3x^2 + 4x + 1 \tag{6}$$

- We want to find x^* such that $f(x^*) \leq f(x)$ for all x.
- In this case we can analytically find this value.

$$\begin{aligned} f'(x) &= 6x + 4 \\ 0 &= 6x + 4 \\ x^* &= -\frac{2}{3} \end{aligned}$$

- We can verify that this is a minimum by taking the second derivative and verifying that it is positive.

$$f''(x) = 6$$
$$f''(x^*) = 6 > 0$$

- Alternatively we can work iteratively to find the minimum.
  - Start with a random value for x. Let's say $x_0 = 1$.
  - Find the derivative at x_0.

$$f'(x_0) = 6(1) + 4 = 10$$

  - Move a small amount in the direction of the derivative. Let's say we move 0.1. (This value is called the learning rate or step size and is represented by η.)

$$x_1 = x_0 - \eta f'(x_0)$$
$$x_1 = 1 - 0.1(10)$$
$$x_1 = 0$$

  - Repeat this process until the derivative is close to zero.
  - This process is called gradient descent or the method of steepest ascent. (In this case the gradient is the derivative with respect to x as there is only one variable.)
  - Here is an example with code:

```
def f(x):
    return 3 * x ** 2 + 4 * x + 1

def df(x):
    return 6 * x + 4

def gradient_descent(x, eta, iterations):
    for i in range(iterations):
        x = x - eta * df(x)
    return x

x = gradient_descent(1, 0.1, 100)
print(x)
```

  - This code outputs -0.6666666666666667 which is close to the value we found analytically.
  - We can now write the gradient descent algorithm for our loss function:

```
import numpy as np

def E(w, x, y): # loss function
    return np.sum((w[0] + w[1] * x - y) ** 2)

def gradient_descent(w, x, y, eta, iterations):
    for i in range(iterations):
        w = w - eta * np.array([
            np.sum(2 * (w[0] + w[1] * x - y)),
```

```
            np.sum(2 * (w[0] + w[1] * x - y) * x)
        ])
    return w

x = np.array([1, 2, 3, 4, 5]) # number of rooms
y = np.array([1, 3, 2, 3, 5]) # price of house
w = gradient_descent(np.array([1, 1]), x, y, 0.1, 100)
print(w)
```

- This code outputs $[0.2, 0.8]$.
- In this case we can also find the minimum analytically.
- We can write the loss function as a matrix equation:

$$\begin{aligned} E(w) &= \sum_{i=1}^{N} (w_0 + w_1 x_i - y_i)^2 \\ &= (w_0 + w_1 x_1 - y_1)^2 + (w_0 + w_1 x_2 - y_2)^2 + \cdots + (w_0 + w_1 x_N - y_N)^2 \\ &= \begin{bmatrix} w_0 + w_1 x_1 - y_1 \\ w_0 + w_1 x_2 - y_2 \\ \vdots \\ w_0 + w_1 x_N - y_N \end{bmatrix}^T \begin{bmatrix} w_0 + w_1 x_1 - y_1 \\ w_0 + w_1 x_2 - y_2 \\ \vdots \\ w_0 + w_1 x_N - y_N \end{bmatrix} \\ &= (Xw - y)^T (Xw - y) \end{aligned}$$

- Where X is the design matrix (the matrix of input values, in this case the number of rooms. It is also called the feature matrix or the Regressor.) The first column is all ones to account for the bias term:

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix} \tag{7}$$

- For our loss function we have two variables (w_0 and w_1) so we need to take the partial derivative with respect to each variable.
- We can calculate the partial derivative with respect to w_0:

$$\begin{aligned} \frac{\partial E(w)}{\partial w_0} &= \frac{\partial}{\partial w_0} \sum_{i=1}^{N} (w_0 + w_1 x_i - y_i)^2 \\ &= \sum_{i=1}^{N} \frac{\partial}{\partial w_0} (w_0 + w_1 x_i - y_i)^2 \\ &= \sum_{i=1}^{N} 2(w_0 + w_1 x_i - y_i) \\ &= 2 \sum_{i=1}^{N} (w_0 + w_1 x_i - y_i) \end{aligned}$$

– We can calculate the partial derivative with respect to w_1:

$$\begin{aligned}
\frac{\partial E(w)}{\partial w_1} &= \frac{\partial}{\partial w_1} \sum_{i=1}^{N} (w_0 + w_1 x_i - y_i)^2 \\
&= \sum_{i=1}^{N} \frac{\partial}{\partial w_1} (w_0 + w_1 x_i - y_i)^2 \\
&= \sum_{i=1}^{N} 2(w_0 + w_1 x_i - y_i) x_i \\
&= 2 \sum_{i=1}^{N} (w_0 + w_1 x_i - y_i) x_i
\end{aligned}$$

– We can now write the gradient of the loss function:

$$\begin{aligned}
\nabla E(w) &= \begin{bmatrix} \frac{\partial E(w)}{\partial w_0} \\ \frac{\partial E(w)}{\partial w_1} \end{bmatrix} \\
&= \begin{bmatrix} 2 \sum_{i=1}^{N} (w_0 + w_1 x_i - y_i) \\ 2 \sum_{i=1}^{N} (w_0 + w_1 x_i - y_i) x_i \end{bmatrix}
\end{aligned}$$

– Now to find the minimum we can set the gradient equal to zero and solve for w:

$$\begin{aligned}
\nabla E(w) &= \begin{bmatrix} 2 \sum_{i=1}^{N} (w_0 + w_1 x_i - y_i) \\ 2 \sum_{i=1}^{N} (w_0 + w_1 x_i - y_i) x_i \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
\begin{bmatrix} 2 \sum_{i=1}^{N} (w_0 + w_1 x_i - y_i) \\ 2 \sum_{i=1}^{N} (w_0 + w_1 x_i - y_i) x_i \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
\begin{bmatrix} \sum_{i=1}^{N} (w_0 + w_1 x_i - y_i) \\ \sum_{i=1}^{N} (w_0 + w_1 x_i - y_i) x_i \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
\begin{bmatrix} \sum_{i=1}^{N} w_0 + \sum_{i=1}^{N} w_1 x_i - \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} w_0 x_i + \sum_{i=1}^{N} w_1 x_i^2 - \sum_{i=1}^{N} x_i y_i \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
\begin{bmatrix} N w_0 + w_1 \sum_{i=1}^{N} x_i - \sum_{i=1}^{N} y_i \\ w_0 \sum_{i=1}^{N} x_i + w_1 \sum_{i=1}^{N} x_i^2 - \sum_{i=1}^{N} x_i y_i \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
\begin{bmatrix} N w_0 + w_1 \sum_{i=1}^{N} x_i \\ w_0 \sum_{i=1}^{N} x_i + w_1 \sum_{i=1}^{N} x_i^2 \end{bmatrix} &= \begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_i y_i \end{bmatrix} \\
\begin{bmatrix} N & \sum_{i=1}^{N} x_i \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} &= \begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_i y_i \end{bmatrix} \\
\begin{bmatrix} w_0 \\ w_1 \end{bmatrix} &= \begin{bmatrix} N & \sum_{i=1}^{N} x_i \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_i y_i \end{bmatrix}
\end{aligned}$$

– These equations are called the normal equations.

– Not all matrices are invertible, and inverting a matrix is computationally expensive. So not all problems can be solved this way.

– With this loss function we can generalize to more than one feature.

– For example, if we have two features (number of rooms and square footage) we can write the loss function as:

$$E(w) = \sum_{i=1}^{N} (w_0 + w_1 x_{i1} + w_2 x_{i2} - y_i)^2 \tag{8}$$

– The normal equations become:

$$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} N & \sum_{i=1}^{N} x_{i1} & \sum_{i=1}^{N} x_{i2} \\ \sum_{i=1}^{N} x_{i1} & \sum_{i=1}^{N} x_{i1}^2 & \sum_{i=1}^{N} x_{i1}x_{i2} \\ \sum_{i=1}^{N} x_{i2} & \sum_{i=1}^{N} x_{i1}x_{i2} & \sum_{i=1}^{N} x_{i2}^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_{i1}y_i \\ \sum_{i=1}^{N} x_{i2}y_i \end{bmatrix}$$

– We can therefore write the normal equations in general as:

$$w = (X^T X)^{-1} X^T y \tag{9}$$

– Where X is the design matrix and y is the vector of target values.

– This is called the closed form solution.

– A note: often $\frac{1}{2}$ is included in the loss function. This is done to simplify the derivative. It does not change the minimum.

2.1.2 Why Use a Dummy/Bias Term?

– The dummy term will always be multiplied by one.

– It is essentially a constant that can be added to the model to shift it up or down.

– Without the bias term the model will always go through the origin.

– This is problematic because the origin is not always in the data.

2.1.3 Gradient Descent Explained

– Gradient descent is an iterative method for finding the minimum of a function.

– Assume we have a Loss Function $J(w)$ (e.g. our Sum of Squared Errors).

– Let's say this function is:

$$J(w) = \frac{1}{2}\sigma_{i=1}^{N}(w^T x_i - y_i)^2 \tag{10}$$

– We want to find the value of w (which is a vector) that minimizes this function. Say w^*.

– So this is:

$$w^* = \underset{w}{\operatorname{argmin}}\, J(w) \tag{11}$$

– We now need to find the gradient of $J(w)$.

– As such we take the partial derivative of $J(w)$ with respect to each element of w.

– We can write this as:

$$\frac{\partial}{\partial w_k} J = \frac{1}{2}\frac{\partial}{\partial w_k}\Sigma_{i=1}^{N}(w^T x_i - y_i)^2 \tag{12}$$

– We can now use the chain rule to expand this equation:

$$\frac{\partial}{\partial w_k} J = \frac{1}{2} \Sigma_{i=1}^{N} 2(w^T x_i - y_i) \frac{\partial}{\partial w_k} (w^T x_i - y_i) \tag{13}$$

– We can now take the partial derivative of $(w^T x_i - y_i)$ with respect to w_k:

$$\frac{\partial}{\partial w_k} (w^T x_i - y_i) = x_{ik} \tag{14}$$

– We can now substitute this back into our equation:

$$\frac{\partial}{\partial w_k} J = \frac{1}{2} \Sigma_{i=1}^{N} 2(w^T x_i - y_i) x_{ik} \tag{15}$$

– We can now simplify this equation:

$$\frac{\partial}{\partial w_k} J = \Sigma_{i=1}^{N} (w^T x_i - y_i) x_{ik} \tag{16}$$

– Now we have our partial derivative with respect to w_k.

– Next we need to create our update rule.

– This is an equation that will update the value of w (and hence all of the weights in our model) along the gradient.

– We can write a version which updates one weight at a time:

$$w_{k+1}^{i+1} = w_k^i - \eta \Sigma_{i=1}^{N} (w^T x_i - y_i) x_{ik} \tag{17}$$

– Where η is the learning rate.

– Or a version which updates all of the weights at once:

$$w^{i+1} = w^i - \eta \Sigma_{i=1}^{N} (w^T x_i - y_i) x_i \tag{18}$$

– In these cases x_i is the ith row of the design matrix, x_{ik} is the kth element of the ith row of the design matrix, and y_i is the ith element of the target vector.

– Now we have our update rule and we can start with a random value for w and update it until it converges or we reach a maximum number of iterations.