Introduction to Machine Learning

Linear Regression

Ashley Gao

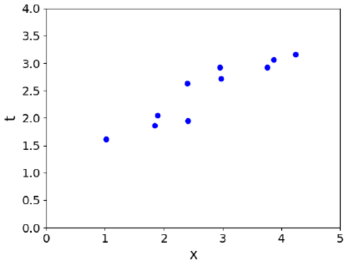
College of William & Mary

September 11, 2023

Overview

- Second learning algorithm of the course: linear regression.
  - Task: predict scalar-valued targets (e.g. stock prices)
  - Architecture: linear function of the inputs
- While KNN was a complete algorithm, linear regression exemplifies a modular approach that will be used throughout this course:
  - Choose a model describing the relationships between variables of interest
  - Define a loss function quantifying how bad the fit to the data is
  - Choose a regularizer saying how much we prefer different candidate models (or explanations of data)
  - Fit a model that minimizes the loss function and satisfies the constraint/penalty imposed by the regularizer, possibly using an optimization algorithm
- Mixing and matching these modular components gives us a lot of new ML methods.

Supervised Learning Setup



- In supervised learning:
  - There is input $x \in \mathcal{X}$, typically a vector of features (or covariates)
  - There is target $t \in \mathcal{T}$ (also called response, outcome, output, class)
  - Objective is to learn a function $f : \mathcal{X} \to \mathcal{T}$ such that $t \approx y = f(x)$ based on the dataset $\mathcal{D} = \{(x^{(i)}, t^{(i)})\}$ for $i = 1, 2, ..., N$.

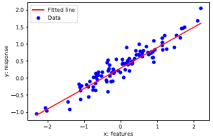
Linear Regression - Model

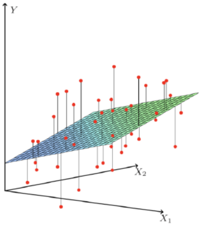
- Model: In linear regression, we use a linear function of the features $x = (x_1, ..., x_D) \in \mathbb{R}^D$ to make prediction y of the target value $t \in \mathbb{R}$:

$$y = f(x) = \sum_j w_j x_j + b \tag{1}$$

  - y is the prediction
  - w is the weights
  - b is the bias (or intercept)
- w and b together are the parameters
- We hope that our prediction is close to the target: $y \approx t$.

What is Linear? 1 feature vs D features





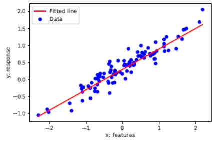
- If we have only 1 feature: $y = wx + b$ where $w, x, b \in \mathbb{R}$
- y is linear in x

- If we have only D feature: $y = \boldsymbol{w}^\top \boldsymbol{x} + b$ where $\boldsymbol{w}, \boldsymbol{x} \in \mathbb{R}^D$ and $b \in \mathbb{R}$
- y is linear in x

- Relation between the prediction y and inputs x is linear in both cases.

Linear Regression

- We have a dataset $\mathcal{D} = \{(x^{(i)}, t^{(i)})\}$ for $i = 1, 2, ..., N$, where
  - $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_D^{(i)})^\top \in \mathbb{R}^D$ are the inputs (i.e. age, height)
  - $t^{(i)} \in \mathbb{R}$ is the target or response (i.e. income)
  - Predict $t^{(i)}$ with a linear function of $x^{(i)}$



- $t^{(i)} \approx y^{(i)} = \boldsymbol{w}^\top \boldsymbol{x} + b$
- Different $(\boldsymbol{w}, b)$ combinations define different lines
- We want the best line $(\boldsymbol{w}, b)$
- How to quantify "best"?

- Relation between the prediction y and inputs x is linear in both cases.

Linear Regression - Loss Function

- A loss function $\mathcal{L}(y, t)$ defines how bad it is if, for some example $\boldsymbol{x}$, the algorithm predicts y, but the target is actually t.
- Squared error loss function:

$$\mathcal{L}(y, t) = \frac{1}{2}(y - t)^2 \tag{2}$$

- $y - t$ is the residual, and we want to make this small in magnitude
- $\frac{1}{2}$ factor is just to make the calculations convenient
- Cost function: loss function averaged over all training examples

$$\mathcal{J}(\boldsymbol{w}, b) = \frac{1}{2N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)})^2 = \frac{1}{2N}\sum_{i=1}^{N}(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b - t^{(i)})^2 \tag{3}$$

- Terminology varies. Some call "cost" empirical or average loss.

Vectorization

- Notion-wise, $\frac{1}{2N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)})^2$ gets messy if we expand $y^{(i)}$:

$$\frac{1}{2N}\sum_{i=1}^{N}(\sum_{j=1}^{D}(w_j x_j^{(i)} + b) - t^{(i)})^2 \tag{4}$$

- The code equivalent is to compute the prediction using a for loop:

```
y = b
for j in range(M):
    y += w[j] * x[j]
```

- Excessive super/sub scripts are hard to work with, and Python loops are slow, so we vectorize algorithms by expressing them in terms of vectors and matrices.

$$\boldsymbol{w} = (w_1, ..., w_D)^\top; \boldsymbol{x} = (x_1, ..., x_D)^\top; y = \boldsymbol{w}^\top \boldsymbol{x} + b \tag{5}$$

- This is simpler and executes much faster:

$$y = np.dot(\boldsymbol{w}, \boldsymbol{x}) + b \tag{6}$$

Vectorization

- Why vectorize?
  - The equations, and the code, will be simpler and more readable. Gets rid of dummy variables and indices!
  - Vectorized code is much faster
    - Cut down on Python interpreter overhead
    - Use highly optimized linear algebra libraries (hardware support)
    - Matrix multiplication very fast on GPU (Graphics Processing Unit)
- Switching in and out of vectorized form is a skill you gain with practice
  - Some algorithms are easier to write/understand using for-loops and vectorize later for performance

Vectorization

- We can organize all the training examples into a design matrix $\boldsymbol{X}$ with one row per training example, and all the targets into the target vector $\boldsymbol{t}$.

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}^{(1)\top} \\ \mathbf{x}^{(2)\top} \\ \mathbf{x}^{(3)\top} \end{pmatrix} = \begin{array}{ccccc} & \text{one feature across} & & & \\ & \text{all training examples} & & & \\ 8 & \boxed{0} & 3 & 0 & \\ \boxed{6} & \boxed{-1} & \boxed{5} & \boxed{3} & \text{one training example (vector)} \\ 2 & \boxed{5} & -2 & 8 & \end{array}$$

- Computing the predictions for the whole dataset:

$$\mathbf{X}\mathbf{w} + b\mathbf{1} = \begin{pmatrix} \mathbf{w}^T\mathbf{x}^{(1)} + b \\ \vdots \\ \mathbf{w}^T\mathbf{x}^{(N)} + b \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{pmatrix} = \mathbf{y}$$

Vectorization

- Computing the squared error cost across the whole dataset:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w} + b\mathbf{1}; \mathcal{J} = \frac{1}{2N}||y - t||^2 \tag{7}$$

- Sometimes we use $\mathcal{J} = \frac{1}{2}||y - t||^2$ without a normalizer. This would correspond to the sum of losses, and not the averaged loss. The minimizer does not depend on N (but optimization might!).
- We can also add a column of 1's to design matrix, combine the bias and the weights, and conveniently write

$$\mathbf{X} = \begin{bmatrix} 1 & [\mathbf{x}^{(1)}]^\top \\ 1 & [\mathbf{x}^{(2)}]^\top \\ 1 & \vdots \end{bmatrix} \in \mathbb{R}^{N\times(D+1)} \text{ and } \mathbf{w} = \begin{bmatrix} b \\ w_1 \\ w_2 \\ \vdots \end{bmatrix} \in \mathbb{R}^{D+1}$$

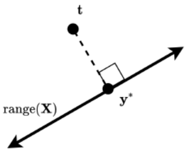
- Then, our predictions reduce to $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w}$.

Vectorization

- We have defined a cost function. This is what we'd like to minimize.
- Two commonly applied mathematical approaches:
  - Algebraic, e.g., using inequalities:
    - To show that z^* minimizes $f(z)$, show that $\forall z, f(z) \geq f(z^*)$
  - Calculus: minimum of a smooth function (if it exists) occurs at a critical point, i.e. point where the derivative is zero.
    - multivariate generalization: set the partial derivatives to zero (or equivalently the gradient).
- Solutions may be direct or iterative
  - Sometimes we can directly find provably optimal parameters (e.g. set the gradient to zero and solve in closed form). We call this a direct solution.
  - We may also use optimization techniques that iteratively get us closer to the solution. We will get back to this soon.

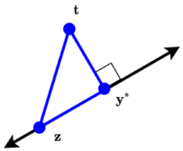
Direct Solution I: Linear Algebra

- We seek $\boldsymbol{w}$ to minimize $||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t}||^2$, or equivalently $||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t}||^2$.
- $\text{Range}(\boldsymbol{X}) = \{\boldsymbol{X}\boldsymbol{w} | \boldsymbol{w} \in \mathbb{R}^D\}$ is a D-dimensional subspace of $\mathbb{R}^N$
- Recall that the closest point $\boldsymbol{y}^* = \boldsymbol{X}\boldsymbol{w}^*$ in subspace Range($\boldsymbol{X}$) of $\mathbb{R}^N$ to any arbitrary point $\boldsymbol{t} \in \mathbb{R}^N$ is found by orthogonal projection.



- We have $(y^* - t) \perp \boldsymbol{X}\boldsymbol{w}, \forall \boldsymbol{w} \in \mathbb{R}^D$
- why is y^* the closest to point $\boldsymbol{t}$?
  - Consider any $z = \boldsymbol{X}\boldsymbol{w}$
  - By Pythagorean theorem and the trivial inequality $(x^2 \geq 0)$

$$||z - \boldsymbol{t}||^2 = ||\boldsymbol{y}^* - \boldsymbol{t}||^2 + ||\boldsymbol{y}^* - z||^2 \geq ||\boldsymbol{y}^* - \boldsymbol{t}||^2 \tag{8}$$



Direct Solution I: Linear Algebra

- From the previous slide, we have $(y^* - t) \perp \boldsymbol{X}\boldsymbol{w}, \forall \boldsymbol{w} \in \mathbb{R}^D$
- Equivalently, the columns of the design matrix $\boldsymbol{X}$ are all orthogonal to $(\boldsymbol{y}^* - \boldsymbol{t})$, and we have that:

$$\boldsymbol{X}^\top(\boldsymbol{y}^* - \boldsymbol{t}) = 0 \tag{9}$$

$$\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{w}^* - \boldsymbol{X}^\top \boldsymbol{t} = 0 \tag{10}$$

$$\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{w}^* = \boldsymbol{X}^\top \boldsymbol{t} \tag{11}$$

$$\boldsymbol{w}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{t} \tag{12}$$

- While this solution is clean and the derivation easy to remember, like many algebraic solutions, it is somewhat ad hoc.
- On the hand, the tools of calculus are broadly applicable to differentiable loss functions...

Direct Solution II: Calculus

- Partial derivative: derivative of a multivariate function with respect to one of its arguments.

$$\frac{\partial}{\partial x_1} f(x_1, x_2) = \lim_{h \to 0} \frac{f(x_1 + h, x_2) - f(x_1, x_2)}{h} \tag{13}$$

- To compute, take the single variable derivative, pretending the other arguments are constant.
- Example: partial derivatives of the prediction y

$$\frac{\partial y}{\partial w_j} = \frac{\partial}{\partial w_j} \left(\sum_{j'} w_{j'} x_{j'} + b \right) = x_j \tag{14}$$

$$\frac{\partial y}{\partial b} = \frac{\partial}{\partial b} \left(\sum_{j'} w_{j'} x_{j'} + b \right) = 1 \tag{15}$$

Direct Solution II: Calculus

- For loss derivatives, apply the chain rule:

$$\frac{\partial(L)}{\partial w_j} = \frac{d(L)}{dy}\frac{\partial(y)}{\partial w_j} = \frac{d}{dy}\left(\frac{1}{2}(y-t)^2\right)x_j = (y-t)x^j \tag{16}$$

$$\frac{\partial(L)}{\partial b} = \frac{d(L)}{dy}\frac{\partial(y)}{\partial b} = y - t \tag{17}$$

- For cost derivatives, use linearity and average over data points.
- Minimum must occur at a point where partial derivatives are zero.

$$\frac{\partial(J)}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)}t^{(i)})x_j^{(i)} = 0 \tag{18}$$

$$\frac{\partial(J)}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N} y^{(i)} - t^{(i)} = 0 \tag{19}$$

- if $\frac{\partial(J)}{\partial w_j} \neq 0$, you could reduce the cost by changing w_j

Direct Solution II: Calculus

- The derivation on the previous slide gives a system of linear equations, which we can solve efficiently.
- As is often the case for models and code, however, the solution is easier to characterize if we vectorize our calculus.
- We call the vector of partial derivatives the gradient
- Thus, the gradient of $f : \mathbb{R}^D \to \mathbb{R}$, denoted $\nabla f(\boldsymbol{w})$, is:

$$\left(\frac{\partial}{\partial w_1} f(\boldsymbol{w}), \ldots, \frac{\partial}{\partial w_D} f(\boldsymbol{w})\right)^\top \tag{20}$$

- The gradient points in the direction of the greatest rate of increase.
- Analogue of the second derivative (the Hessian matrix): $\nabla^2 f(\boldsymbol{w}) \in \mathbb{R}^{D\times D}$ is a matrix with $[\nabla^2 f(\boldsymbol{w})]_{i,j} = \frac{\partial^2}{\partial w_i \partial w_j} f(\boldsymbol{w})$

Direct Solution II: Calculus

- We seek $\boldsymbol{w}$ to minimize $\mathcal{J}(\boldsymbol{w}) = \frac{1}{2}||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t}||^2$
- Taking gradient with respect to $\boldsymbol{w}$ we get

$$\nabla_{\boldsymbol{w}} \mathcal{J}_{\boldsymbol{w}} = \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^\top \boldsymbol{t} = 0 \tag{21}$$

- We get the same optimal weights as before:

$$\boldsymbol{w}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{t} \tag{22}$$

- Linear regression is one of only a handful of models in this course that permit a direct solution.