MLISP: Machine Learning in Signal Processing

Lecture 2

Prof. V. I. Morgenshtern

Scribe: M. Elminshawi

Illustrations: The elements of statistical learning, Hastie, Tibshirani, Friedman

Agenda:

1. Supervised learning
2. Linear regression
3. Gradient descent
4. The normal equations for linear regression

1 Supervised learning

Let's return to the example of house prices:

Living area m^2	Price(1000$)
195	400
149	330
223	369
132	232
...	...

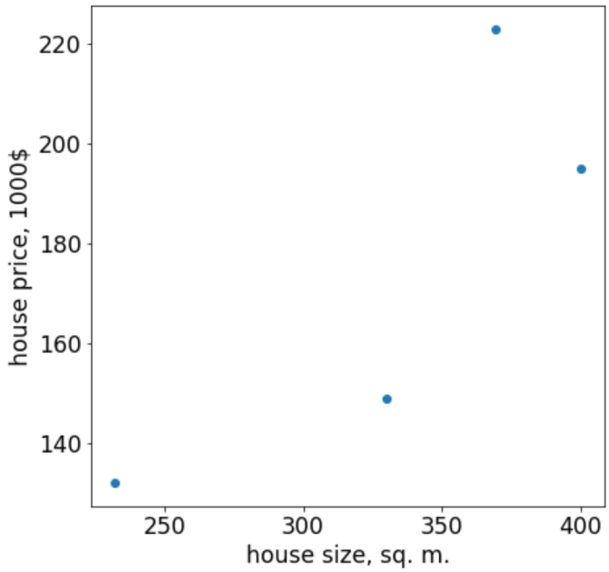


Figure 1: House price vs. living area

How can we learn to predict the house prices? This is an example of supervised learning problem. Here, there is one *input* variable or *feature*, the living area. It has some influence on the *output* variable (aka *target*), the house price. The goal is to use the input variable to predict the output.

If the output variable is continuous (price), the problem is called *regression*. If the *output* variable is discrete (for example house vs. apartment), the the problem is called *classification*.

Notation: We will use $x^{(i)}$ to denote the *input* variables and $y^{(i)}$ to denote the *output* variables. A pair $(x^{(i)}, y^{(i)})$ is called the *training example*. The dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ is called the *training set*.

Our goal is to learn the function $h : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ is the set of input values and $\mathcal{Y}$ is the set of output values, so that $h(x)$ is a "good" predictor of y.

Next, we will consider two simple approaches to prediction: *linear regression* and *nearest neighbors*.

2 Linear regression

Let's make our example more interesting:

Living area m^2	#bedrooms	Price(1000$)
195	3	400
149	3	330
223	3	369
132	2	232
...	...	...

Here the feature vector $\mathbf{x} = [x_1^{(i)}, x_2^{(i)}]^\mathsf{T}$ is a two-dimensional vector in $\mathbb{R}^2$. Specifically, $x_1^{(i)}$ is the living area of the ith house in the training set, and $x_2^{(i)}$ is its number of bedrooms in that house.

Notation: In this course we will use the boldface small letters $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ to denote vectors and the boldface capital letters $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ to denote matrices.

By looking at Figure 1 we might decide to model y as a *linear function* of $\mathbf{x}$:

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2.$$

This is a simple choice and many other reasonable functional forms for h are possible.

Above, θ_0, θ_1, θ_2 are the *parameters* of the model. There are $p = 3$ parameters in our example. It is convenient to set $x_0 = 1$ so that

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_{i=0}^{p-1} \theta_i x_i = \boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}. \tag{1}$$

Given the training set, how do we pick or learn the parameters $\boldsymbol{\theta}$? One reasonable method is to make $h_{\boldsymbol{\theta}}(x)$ to be close to y, at least for the training examples we have. Define the *cost function*:

$$J(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2. \tag{2}$$

As we will see, this cost function gives rise to the *ordinary least squares* estimator.

3 Gradient descent

We want to choose $\boldsymbol{\theta}$ to minimize $J(\boldsymbol{\theta})$. How?

Observe that $J(\boldsymbol{\theta})$ is a quadratic function of $\boldsymbol{\theta}$, so it should be "easy" to find its minimum. For example, we can start with an initial guess for $\boldsymbol{\theta}$ and then repeatedly change $\boldsymbol{\theta}$ to make $J(\boldsymbol{\theta})$ smaller. To make $J(\boldsymbol{\theta})$ smaller, we can, for example, make a small step in the direction of negative gradient of $J(\boldsymbol{\theta})$, which implies the following update rule:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) \text{ for all } j = 0, ..., p-1.$$

Here α is called the *learning rate*. In the matrix notation this can be written as:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha \, \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

where

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \vdots \\ \frac{\partial J}{\partial \theta_{p-1}} \end{bmatrix}$$

is the gradient vector.

Note, this is a very natural algorithm, because the direction of negative gradient is the direction of steepest local descent of $J(\cdot)$.

Let's calculate $\frac{\partial J}{\partial \theta_j}$ first for the case when we have only one training example:

$$\begin{aligned}\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \left(h_\theta(\mathbf{x}) - y\right)^2 \\ &= 2 \times \frac{1}{2} (h_{\boldsymbol{\theta}}(\mathbf{x}) - y) \times \frac{\partial}{\partial \theta_j} \left(h_{\boldsymbol{\theta}}(\mathbf{x}) - y\right) \\ &= (h_{\boldsymbol{\theta}}(\mathbf{x}) - y) \times \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{p-1} \theta_i x_i - y \right) \\ &= (h_{\boldsymbol{\theta}}(\mathbf{x}) - y) x_j.\end{aligned}$$

For a single training example $(x^{(i)}, y^{(i)})$ this gives the update rule:

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(\mathbf{x}^{(i)}) \right) x_j^{(i)} \text{ for every } j.$$

Note that the magnitude of the update is proportional to the error term $y^{(i)} - h_\theta(\mathbf{x}^{(i)})$.

For n training examples our update rule becomes:

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{n} \left(y^{(i)} - h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) \right) x_j^{(i)} \text{ for every } j.$$

This is called *batch gradient descent* because this method looks at the entire training set at each step: Note the summation, $\sum_{i=1}^{n}$.

Here is one run of the algorithm:

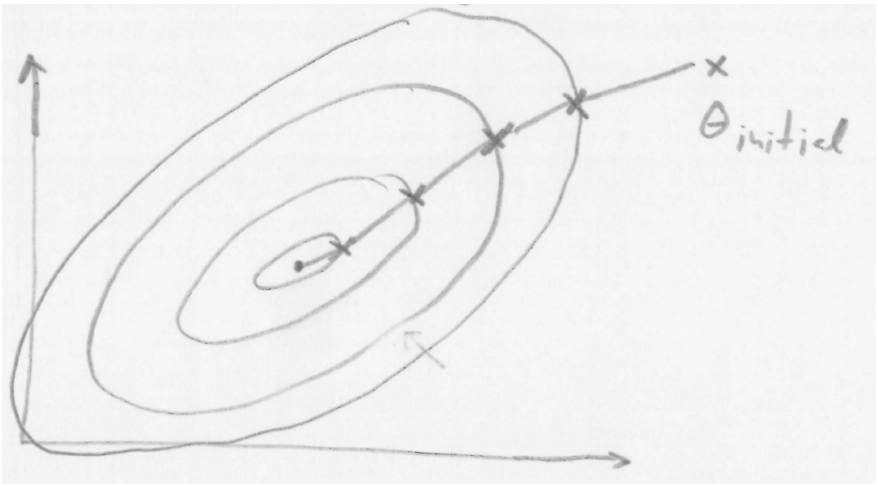
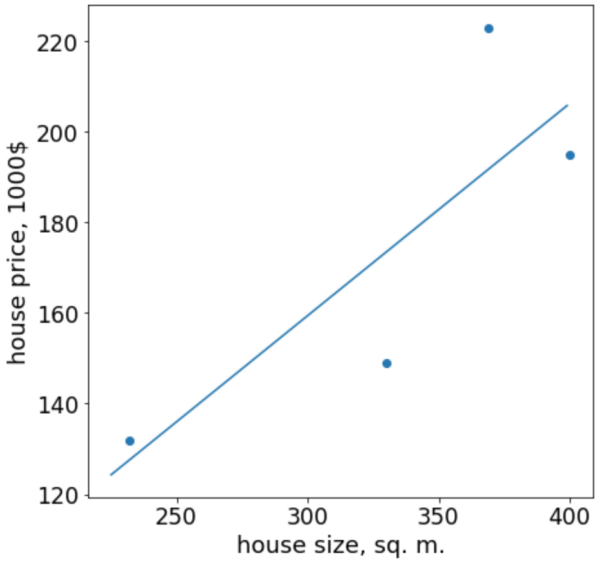


Figure 2: Contour plot of $J(\cdot)$ and steps of gradient descent.

Ellipses are contours of the quadratic function $J(\boldsymbol{\theta})$.

For the dataset with area only, $h_{\boldsymbol{\theta}}(\mathbf{x})$ looks like the straight line:



An alternative to batched gradient descent is *stochastic gradient descent*, where at each update step we only look at one data point.

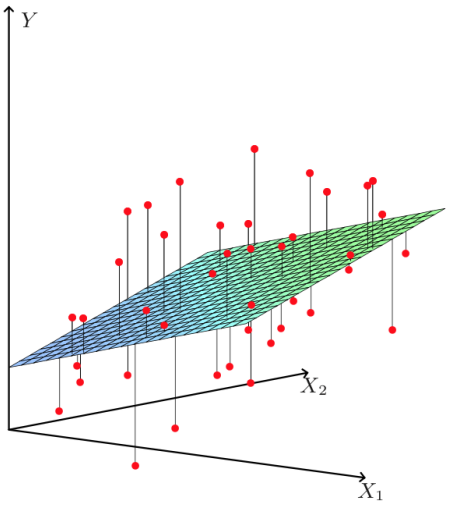
Repeat until convergence:

$$\begin{aligned}&\text{for } i \text{ in } [1:n]:\\ &\quad \theta_j := \theta_j + \alpha\left(y^{(i)} - h_\theta(\mathbf{x}^{(i)})\right) x_j^{(i)} \text{ for every } j.\end{aligned}$$

This is a good alternative to batched gradient descent in practice, because this algorithm is much faster.

4 The normal equations for linear regression

Recall that we are trying to minimize the quadratic cost function, $J(\boldsymbol{\theta})$ defined in (2), on the training set. We model our output variables as a linear function of the input variables, as specified by $h_{\boldsymbol{\theta}}(\mathbf{x})$ in (1). Here is a picture describing the situation:



Above, the red dots are the points from the training set and black lines denote the distances from these points to the linear function. As we have seen above, one way to minimize $J(\cdot)$ is to use the gradient descent algorithm.

Since $J(\cdot)$ is a quadratic function, it can also be minimized in closed form. To make the derivation simpler, it is convenient to work with the matrix derivatives.

Matrix derivative: For a function $f : \mathbb{R}^{n\times p} \longmapsto \mathbb{R}$ we define the derivative of f with respect to $\mathbf{A}$ to be:

$$\nabla_{\mathbf{A}} f(\mathbf{A}) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1p}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{n1}} & \cdots & \frac{\partial f}{\partial A_{np}} \end{bmatrix}.$$

Example:

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \\ f(\mathbf{A}) &= \frac{3}{2}A_{11} + 5A_{12}^2 + A_{21}A_{22} \\ \nabla_{\mathbf{A}} f(\mathbf{A}) &= \begin{bmatrix} \frac{3}{2} & 10A_{12} \\ A_{22} & A_{21} \end{bmatrix}. \end{aligned}$$

We will also need the *trace* of a matrix:

$$\operatorname{tr} \mathbf{A} = \sum_{i=1}^{n} A_{ii}.$$

Properties of the trace:

$$\begin{aligned}
&\operatorname{tr}\mathbf{AB} = \operatorname{tr}\mathbf{BA}, \quad \text{if } \mathbf{AB} \text{ is square} \\
&\operatorname{tr}\mathbf{ABC} = \operatorname{tr}\mathbf{CAB} = \operatorname{tr}\mathbf{BCA}, \quad \text{if } \mathbf{ABC} \text{ is square} \\
&\operatorname{tr}\mathbf{ABCD} = \operatorname{tr}\mathbf{DABC} = \operatorname{tr}\mathbf{CDAB} = \operatorname{tr}\mathbf{BCDA} \\
&\operatorname{tr}\mathbf{A} = \operatorname{tr}\mathbf{A}^\mathsf{T} \\
&\operatorname{tr}(\mathbf{A}+\mathbf{B}) = \operatorname{tr}\mathbf{A} + \operatorname{tr}\mathbf{B} \\
&\operatorname{tr}(a\mathbf{A}) = a \operatorname{tr}\mathbf{A}.
\end{aligned}$$

Properties of matrix derivatives: (without proof, more in review session):

$$\nabla_{\mathbf{A}} \operatorname{tr}\mathbf{AB} = \mathbf{B}^\mathsf{T} \tag{3}$$

$$\nabla_{\mathbf{A}^\mathsf{T}} f(\mathbf{A}) = (\nabla_{\mathbf{A}} f(\mathbf{A}))^\mathsf{T} \tag{4}$$

$$\nabla_{\mathbf{x}} \mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x} = 2\mathbf{A}\mathbf{x} \text{ (if } \mathbf{A} \text{ is symmetric)} \tag{5}$$

$$\nabla_{\mathbf{A}} |\mathbf{A}| = |\mathbf{A}|(\mathbf{A}^{-1})^\mathsf{T} \tag{6}$$

where $|\mathbf{A}|$ is the determinant of the matrix $\mathbf{A}$.

Now we are ready to derive the normal equations.

Let's stack out data points as rows of a matrix:

$$\mathbf{X} = \begin{bmatrix} - & (\mathbf{x}^{(1)})^\mathsf{T} & - \\ - & (\mathbf{x}^{(2)})^\mathsf{T} & - \\ & \vdots & \\ - & (\mathbf{x}^{(n)})^\mathsf{T} & - \end{bmatrix}.$$

Also, let $\mathbf{y}$ be the n-dimensional vector containing all the target values from the training set:

$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}.$$

Now,

$$\mathbf{X}\boldsymbol{\theta} - \mathbf{y} = \begin{bmatrix} (\mathbf{x}^{(1)})^\mathsf{T}\boldsymbol{\theta} \\ \vdots \\ (\mathbf{x}^{(n)})^\mathsf{T}\boldsymbol{\theta} \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix} = \begin{bmatrix} h_{\boldsymbol{\theta}}(\mathbf{x}^{(1)}) - y^{(1)} \\ \vdots \\ h_{\boldsymbol{\theta}}(\mathbf{x}^{(n)}) - y^{(n)} \end{bmatrix}$$

where we used $h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = (\mathbf{x}^{(i)})^\mathsf{T}\boldsymbol{\theta}$.

Therefore,

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{n}(h_{\boldsymbol{\theta}}(x^{(i)}) - y^{(i)})^2 = \frac{1}{2}(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^\mathsf{T}(\mathbf{X}\boldsymbol{\theta} - \mathbf{y}).$$

Since $J(\boldsymbol{\theta})$ is a quadratic function of $\boldsymbol{\theta}$ it has a unique global minimum. To find this minimum it is sufficient to compute the gradient of $J(\boldsymbol{\theta})$ and set the gradient to zero:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = 0. \tag{7}$$

Lets calculate $\nabla_\theta J(\theta)$:

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \frac{1}{2}(\mathbf{X}\theta - \mathbf{y})^\mathsf{T}(\mathbf{X}\theta - \mathbf{y}) \\ &= \frac{1}{2}\nabla_\theta(\theta^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X}\theta - \theta^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{y} - \mathbf{y}^\mathsf{T}\mathbf{X}\theta + \mathbf{y}^\mathsf{T}\mathbf{y}) \\ &= \frac{1}{2}\nabla_\theta(\theta^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X}\theta - 2\theta^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{y} + \mathbf{y}^\mathsf{T}\mathbf{y}) \\ &= \frac{1}{2}(2\mathbf{X}^\mathsf{T}\mathbf{X}\theta - 2\mathbf{X}^\mathsf{T}\mathbf{y}).\end{aligned}$$

where in the last step we used (5) for the first term and (3) for the second term.

With this, (7) yields:

$$\mathbf{X}^\mathsf{T}(\mathbf{X}\hat{\theta} - \mathbf{y}) = \mathbf{0}. \tag{8}$$

Above, $\hat{\theta}$ denotes the optimal value for θ, i.e. the value that minimizes $J(\theta)$. Rearranging the terms we obtain the normal equations:

$$\mathbf{X}^\mathsf{T}\mathbf{X}\hat{\theta} = \mathbf{X}^\mathsf{T}\mathbf{y}.$$

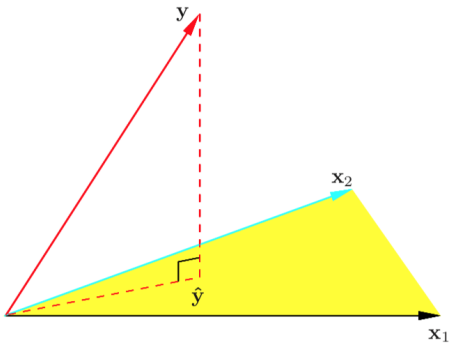
We conclude that the value of θ that minimizes $J(\theta)$ is given in closed form by the equation:

$$\hat{\theta} = (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y}.$$

Of corse, this formula is true only if the matrix $\mathbf{X}^\mathsf{T}\mathbf{X}$ is invertible.

Let's provide a geometric interpretation for these equations. First, recall that if $\mathbf{x}_0$ is an input vector, then the predicted output for this input is $\mathbf{x}_0^\mathsf{T}\hat{\theta}$. Therefore $\hat{\mathbf{y}} = \mathbf{X}\hat{\theta}$ in (8) is the vector of predicted outputs for the training set.

It turns out that the predicted output vector is the projection of the training output vector $\mathbf{y}$ onto the plane spanned by column vectors of $\mathbf{X}$ as depicted below:



To see this note first that

$$\hat{\mathbf{y}} = \hat{\theta}_0\mathbf{x}_0 + \ldots \hat{\theta}_{p-1}\mathbf{x}_{p-1}$$

is the linear combination of columns of $\mathbf{X} = [\mathbf{x}_0, \ldots, \mathbf{x}_{p-1}]$, i.e. $\hat{\mathbf{y}}$ is in the span of $\mathbf{x}_0, \ldots, \mathbf{x}_{p-1}$. Second, it follows from equation (8) that the error vector $\hat{\mathbf{y}} - \mathbf{y}$ is orthogonal to all columns of $\mathbf{X}$.