Machine Learning - Michaelmas Term 2017
Lecture 3 : Linear Regression

Lecturers: Christoph Haase & Varun Kanade

Linear regression is one of the central methods in supervised machine learning and statistics. This method goes back at least to Legendre and Gauss who applied it to astronomical observations. Incidentally, the word 'regression' in this context seems to have first appeared in the work of Galton (1886). Although the basic model assumes a linear relationship between the input and the output, by performing *basis function expansion* it can be used to model non-linear relationships as well, something that we will study in the next few lectures.

As the linear regression framework is relatively easy to describe and in the most basic setting closed form expressions can be obtained, it serves as a useful tool to introduce many key notions in machine learning.

1 Linear Model

Let us suppose that our datapoints are vectors $\mathbf{x} \in \mathbb{R}^D$ and the output (the value to be predicted) is $y \in \mathbb{R}$. For the purpose of training, we are given N observations, $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$. We will assume that all vectors are column vectors.

A linear model assumes that y can be expressed (approximately) as a linear (or affine) function of the input $\mathbf{x}$. Thus, we may express,

$$y = w_0 + x_1 w_1 + \cdots + x_D w_D + \epsilon. \tag{1}$$

Here w_0 is the constant term (also called bias or intercept) which does not depend on the input at all. The term ϵ models noise or uncertainty—the fact that we don't expect the observed data to be exactly captured by a linear relationship.

2 A Toy Example

Throughout this lecture, we will make use of a toy problem of predicting commute time using simple features. For a few different individuals, we are given the distance they need to travel and the day of the week, along with the observed time their commute took. This (fake) data is summarised in Table 1.

dist (km)	day	commute time (min)
2.7	fri	25
4.1	mon	33
1.0	sun	15
5.2	tue	45
2.8	sat	22

Table 1: Distance travelled, day of week and commute time.

The inputs in this instance are distance and day of the week. In order to apply a mathematical model, we first need to project them in $\mathbb{R}^D$ for some D. Distance already takes a numerical value, so this is straightforward. For days of the week, one might at first glance think of encoding them as integers $\{0, 1, 2, \ldots, 6\}$. However, this imposes an unnatural and untrue mathematical structure on the data which may only serve to confuse the model that we wish

to train. A better encoding would be to use seven binary attributes, one for each day, say x_{mon} for monday, which takes the value 1 if the day indeed were monday and 0 otherwise. This is referred to as *one-hot* encoding, a term that refers to the fact that exactly one of the seven variables will be 1 for a given datapoint.

For now, we will make the (reasonable?) simplifying assumption that the commute time is only affected by whether it is a weekday or the weekend, and use a single binary variable that takes value 1 for a weekday and 0 for the weekend. Thus, our data can be expressed as $\mathbf{x} \in \mathbb{R}^2$, where x_1 is the distance and x_2 denotes whether or not the day is a weekday.

Thus, the linear model for this problem is given by:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \epsilon$$

For technical convenience, we will sometimes assume that there is an extra input column in Table 1, where every entry is 1. Thus, $\mathbf{x} = (x_0, x_1, x_2) \in \mathbb{R}^3$, where $x_0 = 1$ for every possible input. This way we don't have to treat the constant term w_0 separately and the linear model can be more succinctly expressed as $y = \mathbf{w} \cdot \mathbf{x} + \epsilon$, where $\mathbf{w} \cdot \mathbf{x}$ is the scalar (dot) product between the vectors $\mathbf{w}$ and $\mathbf{x}$. This means that the data is now treated as being $(D+1)$-dimensional.

Note: Sometimes we'll just refer to $D+1$ as D. Whether the bias term is folded into the input or not should be clear from context.

3 Least Squares Estimate

When it comes to learning the linear model, the parameters that need to be estimated are $w_0, w_1, \ldots, w_D$. Thus, during the training (or learning, or estimation) phase, we process the input data using a *learning algorithm* that outputs an estimate of the parameters $\mathbf{w} = (w_0, \ldots, w_D)$. When actually using the model, we will only be given a new input, say $\mathbf{x}_{\text{new}}$, and we use the model to predict. The prediction is given by $\widehat{y}_{\text{new}} = \mathbf{w} \cdot \mathbf{x}_{\text{new}}$. In order to understand whether or not the learned model does well on new data, it is usually a good idea to keep aside some data that we don't supply to the training algorithm, and use this to test the performance of the model. We'll return to the issue of testing the quality of models in the coming weeks.

3.1 Least Squares Estimate in One Dimension

Let us start with the most simple of cases, where the input itself is one-dimensional. Thus, the data we have is $\langle (x_i, y_i) \rangle_{i=1}^N$. Let $\widehat{y}(x) = w_0 + w_1 \cdot x$ denote the prediction of the linear model (without the noise term). Then the least squares estimate looks for w_0, w_1 that minimise the following function.

$$\mathcal{L}(w_0, w_1) = \frac{1}{2N} \sum_{i=1}^{N} (\widehat{y}_i - y_i)^2 = \frac{1}{2N} \sum_{i=1}^{N} (w_0 + x_i \cdot w_1 - y_i)^2 \tag{2}$$

The function $\mathcal{L}$ is called the loss function; it is often also referred to as the cost function, objective function, or energy function. Geometrically, we can view this as looking for a line such that when the actual observation points are projected onto the line using vertical lines, the sum of the squares of these projected segments is minimised. The length of these line segements is the absolute value of the difference between the prediction according to the model (line) and the observation, referred to as the residual. For this reason, this objective is referred to as the *residual sum of squares*. The estimate (w_0, w_1) that minimises this objective is called the least squares estimate. (See Figure 1).

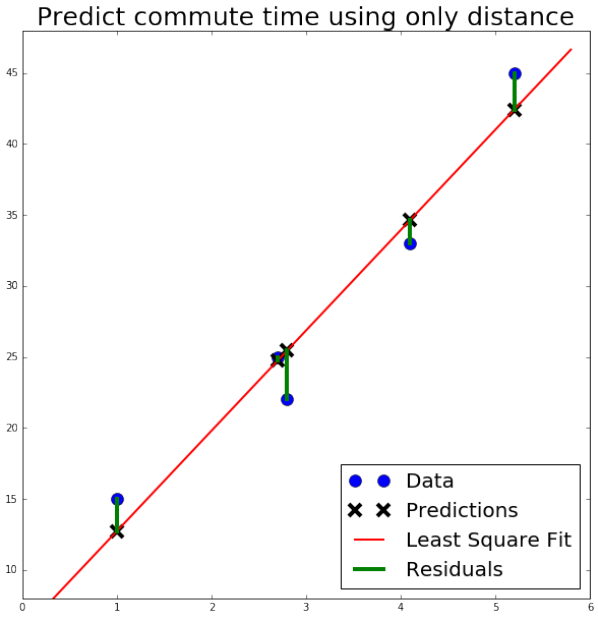

Figure 1: Figure showing least square fit to data in Table 1 along with residuals.

In order to find the least square estimate analytically, we can minimise $\mathcal{L}(w_0, w_1)$ given by (2). We obtain the partial derivatives as follows:

$$\frac{\partial \mathcal{L}}{\partial w_0} = \frac{1}{N}\sum_{i=1}^{N}(w_0 + w_1 \cdot x_i - y_i)$$
$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{1}{N}\sum_{i=1}^{N}(w_0 + w_1 \cdot x_i - y_i)x_i$$

We obtain the solution for (w_0, w_1) by setting the partial derivatives to 0 and solving the resulting system. The equations defining this sytem are called the *normal equations*.

$$w_0 + w_1 \cdot \frac{\sum_i x_i}{N} = \frac{\sum_i y_i}{N} \tag{3}$$
$$w_0 \cdot \frac{\sum_i x_i}{N} + w_1 \cdot \frac{\sum_i x_i^2}{N} = \frac{\sum_i x_i y_i}{N} \tag{4}$$

The system of equations given by (3) and (4) is fairly straightforward to solve. However, we can look at the empirical distribution on the data, *i.e.*, each point (x_i, y_i) appears with probability $\frac{1}{N}$, and define the following statistical quantities:

$$\bar{x} = \frac{\sum_i x_i}{N} \qquad \bar{y} = \frac{\sum_i y_i}{N}$$
$$\widehat{\text{var}}(x) = \frac{\sum_i x_i^2}{N} - \bar{x}^2 \qquad \widehat{\text{cov}}(x, y) = \frac{\sum_i x_i y_i}{N} - \bar{x} \cdot \bar{y}$$

Then, the least squares estimate (w_0, w_1) is given by:

$$w_1 = \frac{\widehat{\text{cov}}(x, y)}{\widehat{\text{var}}(x)} \qquad w_0 = \bar{y} - w_1 \cdot \bar{x} \tag{5}$$

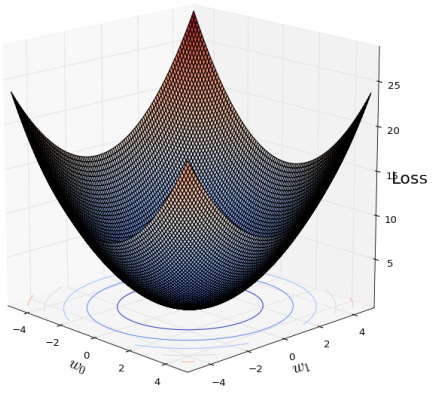

Figure 2: Loss Function for Least Squares

3.2 Least Squares Estimate in the General Case

We now consider the case when the data may be in D dimensions. Consider the prediction made by the linear model as governed by:

$$\widehat{y}_i = \sum_{j=0}^{D} x_{ij} w_j,$$

where we use the convention that x_{ij} denotes the j^{th} feature of the datapoint $\mathbf{x}_i$ and that $x_{i0} = 1$ for all examples, so that the constant term w_0 does not have to be treated separately.

It will be convenient to express everything in matrix notation. Let $\mathbf{X}$ denote an $N \times (D+1)$ matrix whose i^{th} row is the datapoint $\mathbf{x}_i^\mathsf{T}$ (transposed), $\widehat{\mathbf{y}}$ is a column vector whose i^{th} entry is $\widehat{y}_i$ and $\mathbf{w}$ is the vector of parameters to be learnt. We can express these in matrix form as,

$$\underset{\widehat{\mathbf{y}}_{N\times 1}}{\begin{bmatrix} \widehat{y}_1 \\ \widehat{y}_2 \\ \vdots \\ \widehat{y}_N \end{bmatrix}} = \underset{\mathbf{X}_{N\times(D+1)}}{\begin{bmatrix} \mathbf{x}_1^\mathsf{T} \\ \mathbf{x}_2^\mathsf{T} \\ \vdots \\ \mathbf{x}_N^\mathsf{T} \end{bmatrix}} \underset{\mathbf{w}_{(D+1)\times 1}}{\begin{bmatrix} w_0 \\ \vdots \\ w_D \end{bmatrix}} = \underset{\mathbf{X}_{N\times(D+1)}}{\begin{bmatrix} x_{10} & \cdots & x_{1D} \\ x_{20} & \cdots & x_{2D} \\ \vdots & \ddots & \vdots \\ x_{N0} & \cdots & x_{ND} \end{bmatrix}} \underset{\mathbf{w}_{(D+1)\times 1}}{\begin{bmatrix} w_0 \\ \vdots \\ w_D \end{bmatrix}}$$

More succinctly, we write $\widehat{\mathbf{y}} = \mathbf{X}\mathbf{w}$. The loss function $\mathcal{L}(\mathbf{w})$ can also be expressed in matrix form. We have,

$$\begin{aligned} \mathcal{L}(\mathbf{w}) &= \frac{1}{2N}\sum_{i=1}^{N}(\mathbf{x}_i^\mathsf{T}\mathbf{w} - y_i)^2 = \frac{1}{2N}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^\mathsf{T}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) \\ &= \frac{1}{2N}\left(\mathbf{w}^\mathsf{T}\left(\mathbf{X}^\mathsf{T}\mathbf{X}\right)\mathbf{w} - \mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{y} - \mathbf{y}^\mathsf{T}\mathbf{X}\mathbf{w} + \mathbf{y}^\mathsf{T}\mathbf{y}\right) \\ &= \frac{1}{2N}\left(\mathbf{w}^\mathsf{T}\left(\mathbf{X}^\mathsf{T}\mathbf{X}\right)\mathbf{w} - 2\cdot\mathbf{y}^\mathsf{T}\mathbf{X}\mathbf{w} + \mathbf{y}^\mathsf{T}\mathbf{y}\right) \end{aligned} \tag{6}$$

In the last step above, we used the fact that $\mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{y} = \mathbf{y}^\mathsf{T}\mathbf{X}\mathbf{w}$ since they are both scalars and transposes of each other. We could expand out the loss function, which results in a quadratic

function in $w_0, \ldots, w_D$ (see Fig. 2) and meticulously calculate every partial derivative. However, since we'll have to do this often in machine learning, it is beneficial to develop tricks to differentiate matrix expressions directly. This is not a new kind of calculus, just tricks to simplify calculations in the standard multivariate calculus.

We need to remember two simple rules, one for linear form expressions and one for quadratic form expressions. We derive both of them here

(i) Linear Form Expressions: $\nabla_{\mathbf{w}} \left(\mathbf{c}^\mathsf{T} \mathbf{w} \right) = \mathbf{c}$

$$\mathbf{c}^\mathsf{T} \mathbf{w} = \sum_{j=0}^{D} c_j w_j$$

$$\frac{\partial \left(\mathbf{c}^\mathsf{T} \mathbf{w} \right)}{\partial w_j} = c_j, \tag{7}$$

$$\nabla_{\mathbf{w}} \left(\mathbf{c}^\mathsf{T} \mathbf{w} \right) = \mathbf{c} \tag{8}$$

(ii) Quadratic Form Expressions: $\nabla_{\mathbf{w}} \left(\mathbf{w}^\mathsf{T} \mathbf{A} \mathbf{w} \right) = \mathbf{A}\mathbf{w} + \mathbf{A}^\mathsf{T}\mathbf{w} \quad (= 2\mathbf{A}\mathbf{w}$ for symmetric $\mathbf{A})$

$$\mathbf{w}^\mathsf{T} \mathbf{A} \mathbf{w} = \sum_{i=0}^{D} \sum_{j=0}^{D} w_i w_j A_{ij}$$

$$\frac{\partial \left(\mathbf{w}^\mathsf{T} \mathbf{A} \mathbf{w} \right)}{\partial w_k} = \sum_{i=0}^{D} w_i A_{ik} + \sum_{j=0}^{D} A_{kj} w_j = \mathbf{A}^\mathsf{T}_{[:,k]} \mathbf{w} + \mathbf{A}_{[k,:]} \mathbf{w}$$

$$\nabla_{\mathbf{w}} \left(\mathbf{w}^\mathsf{T} \mathbf{A} \mathbf{w} \right) = \mathbf{A}^\mathsf{T}\mathbf{w} + \mathbf{A}\mathbf{w} \tag{9}$$

Above we use `python` notation, where $\mathbf{A}_{[:,k]}$ refers to the k^{th} column of $\mathbf{A}$ and $\mathbf{A}_{[k,:]}$ refers to the k^{th} row of $\mathbf{A}$. We can now apply rules (8) and (9) to the loss function in (6) to obtain the gradient as follows:

$$\nabla_{\mathbf{w}} \mathcal{L} = \frac{1}{N} \left(\left(\mathbf{X}^\mathsf{T} \mathbf{X} \right) \mathbf{w} - \mathbf{X}^\mathsf{T} \mathbf{y} \right)$$

By setting $\nabla_{\mathbf{w}} \mathcal{L} = \mathbf{0}$ and solving we get,

$$\left(\mathbf{X}^\mathsf{T} \mathbf{X} \right) \mathbf{w} = \mathbf{X}^\mathsf{T} \mathbf{y}$$

$$\mathbf{w} = \left(\mathbf{X}^\mathsf{T} \mathbf{X} \right)^{-1} \mathbf{X}^\mathsf{T} \mathbf{y} \qquad \text{(Assuming inverse exists)} \tag{10}$$

The predictions made by the model on the data $\mathbf{X}$ are given by

$$\widehat{\mathbf{y}} = \mathbf{X}\mathbf{w} = \mathbf{X} \left(\mathbf{X}^\mathsf{T} \mathbf{X} \right)^{-1} \mathbf{X}^\mathsf{T} \mathbf{y} \tag{11}$$

For this reason the matrix $\mathbf{X} \left(\mathbf{X}^\mathsf{T} \mathbf{X} \right)^{-1} \mathbf{X}^\mathsf{T}$ is called the "hat" matrix—it puts a hat on $\mathbf{y}$.

Discussion on the Least Squares Estimate

In order to derive the least square estimate, we required that $\mathbf{X}^\mathsf{T}\mathbf{X}$ be invertible. It is worth pondering when one might expect this to be the case. Observe that $\mathbf{X}^\mathsf{T}\mathbf{X}$ is a $(D+1) \times (D+1)$

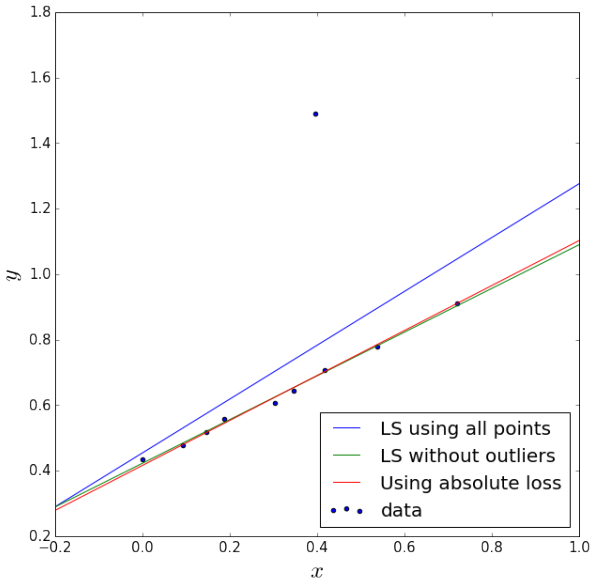

Figure 3: Least Squares Estimate is not robust to outliers.

matrix, whereas $\mathbf{X}$ is $N \times (D+1)$. Assuming that the number of features D is significantly smaller than the number of datapoints N, we could expect $\mathbf{X}$ to have rank $D+1$, thus making $\mathbf{X}^\top\mathbf{X}$ invertible. Of course, if there is any linear dependence among the columns of $\mathbf{X}$ this will not be the case. In general, we don't expect perfect linear dependence between observed features. However, note that this may happen as an artifact of our constructions. For example, if we did use one-hot encoding for the days of the week using seven binary variables, $x_{\text{mon}}, \ldots, x_{\text{sun}}$. Then, we know that exactly one of them must be 1 and so $x_{\text{mon}} + \cdots + x_{\text{sun}} = 1$. This inadvertently introduces linear dependence in the columns of $\mathbf{X}$. One solution, would be to just drop one of the seven days, say sunday, and assume sunday corresponds to the case when the variables corresponding to all of the other days are 0.

Of course, the problem becomes much more pronounced when the number of datapoints is significantly smaller than the number of features. We'll discuss what to do in this case in the next couple of lectures.

Finally, if there are outliers in the data then the least squares estimate can be quite poor. This is because a penatly is placed on the square of the residual, $|\widehat{y}_i - y_i|$. Figure 3 shows the difference between the least square fit with and without the outlier (blue and green lines). Of course, when the data is not so simple, it is hard to figure out which points may be outliers. Rather than removing outliers, an alternative approach is to use only the sum of the residues, *i.e.*, $\sum_{i=1}^{N} |\widehat{y}_i - y_i|$ as the loss. This reduces the penalty placed on the outliers. The resulting line is shown in Fig 3 using the red line. Unfortunately, there is no simple closed form solution for this problem. We will discuss how to solve this later in the course.

4 Goal, Model and Algorithm

Goal. Having gone through the mathematics of deriving the least squares estimate, let's revisit the toy problem of predicting commute time. Throughout, we've been assuming that our goal is simply to predict the commute time. However, in reality we may also wish to make suggestions to users to change their behaviour. If there were many more features available, we may want to make suggestions such as use the bus instead of driving, or travel at a specific time, *etc.* The choice of model will typically depend on the intended goal of the machine learning application. If one only wishes to make predictions, it might be worth using an extremely complicated model

that is highly accurate. If on the other hand, there is some interpretation required, it may make more sense to compromise a bit on the accuracy in order to obtain a model that can be understood by human experts. This may be quite common in domains such as biology, finance, *etc.* In fact, for some financial applications, regulations require being able to explain how the decision was arrived at even if automatic methods were used![1]

Model. In this lecture, we used was the linear model (1), which assumes a linear relationship between inputs and output. The choice of loss function is also made as a modelling assumption. We focused on deriving the least squares estimate that minimises the residual sum of squares, where the residual is the difference between the prediction made by the model and the actual observation. We briefly mentioned how other choices of loss function may be applicable if the data has outliers.

The approach studied in this lecture can be viewed as a purely "optimisation approach" to machine learning. We try to fit a model according to a (somewhat sensible) loss function and want the resulting optimisation problem to be tractable. In particular, we assumed there was a noise term, as we don't expect the data to satisfy a perfect linear relationship, but we did not attempt to model the noise. A probabilistic view of machine learning will try to model the noise explicitly as uncertainty introduced while making the observations; this is a topic for the next few lectures.

Algorithm. For the least squares estimate, the algorithm is particularly simple as a closed form expression for the estimator can be derived. The algorithm simply involves elementary matrix operations. In particular, it is easy to see that the algorithm can be implemented in time $O(D^2N)$ assuming that $D < N$. When closed form solutions are not available, more sophisticated algorithms will have to be used.

References

Francis Galton. Regression towards mediocrity in hereditary stature. *The Journal of the Anthropological Institute of Great Britain and Ireland*, 15:246–263, 1886.

[1] https://www.linkedin.com/pulse/machine-learning-finance-25-years-robert-hillman