ML-MDS - Machine Learning

Jose Antonio Lorencio Abril

Spring 2023

Professor: Marta Arias

Student e-mail: jose.antonio.lorencio@estudiantat..upc.edu

This is a summary of the course *Machine Learning* taught at the Universitat Politècnica de Catalunya by Professor Marta Arias in the academic year 22/23. Most of the content of this document is adapted from the course notes by Arias, [1], so I won't be citing it all the time. Other references will be provided when used.

Contents

List of Figures

List of Algorithms

1 Introduction

2 Linear regression

2.1 Introduction

In Figure 1, we can observe a dataset of the population of different states plotted against the number of fatal accidents in each of the states. Here, each blue circle corresponds to a row of our data, and the coordinates are the $(population, \#accidents)$ values in the row. The red line is the linear regression model of this data. This means it is the line that 'best' approximates the data, where best refers to minimizing some kind of error: the squared error between each point to its projection on the y axis of the line, in this case. This approach is called the **least squares method**.

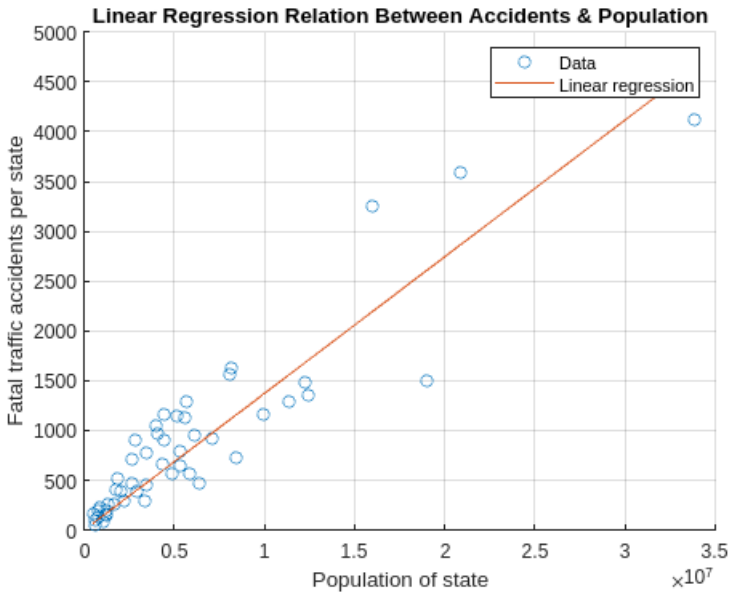


Figure 1: Matlab's accidents dataset and best linear fit.

2.2 Least squares method

2.2.1 Least squares in 2D

In 2D, we have a dataset $\{(x_i, y_i), i = 1, ..., n\}$ and we want to find the line that best approximates y as a function of x. As we want a line, we need to specify its slope, θ_1, and its intercept, θ_0. So, our estimations are:

$$\hat{y}(x_i) = \hat{y}_i = \theta_0 + \theta_1 x_i.$$

The least squares linear regression method chooses θ_0 and θ_1 in such a way that the **error function**

$$J(\theta_0, \theta_1) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - \theta_0 - \theta_1 x_i)^2$$

is minimized.
Note that this function only depends on the parameters θ_0 and θ_1, since the data is assumed to be fixed (they are observations).
To compute them, we just need to find the minimum of J, by taking partial derivatives and setting them to 0. Let's do this optimization. First, we can develop the square:

$$J(\theta_0, \theta_1) = \sum_{i=1}^{n} y_i^2 + \theta_0^2 + \theta_1^2 x_i^2 - 2\theta_0 y_i - 2\theta_1 x_i y_i + 2\theta_0 \theta_1 x_i.$$

Thus:

$$\frac{\partial J}{\partial \theta_0} = \sum_{i=1}^{n} 2\theta_0 - 2y_i + 2\theta_1 x_i = 2n\theta_0 - 2\sum_{i=1}^{n} y_i + 2\theta_1 \sum_{i=1}^{n} x_i$$

and

$$\frac{\partial J}{\partial \theta_1} = \sum_{i=1}^{n} 2\theta_1 x_i^2 - 2x_i y_i + 2\theta_0 x_i = 2\theta_1 \sum_{i=1}^{n} x_i^2 - 2\sum_{i=1}^{n} x_i y_i + 2\theta_0 \sum_{i=1}^{n} x_i.$$

We have now to solve the system given by

$$\begin{cases} 2n\theta_0 - 2\sum_{i=1}^{n} y_i + 2\theta_1 \sum_{i=1}^{n} x_i = 0 \\ 2\theta_1 \sum_{i=1}^{n} x_i^2 - 2\sum_{i=1}^{n} x_i y_i + 2\theta_0 \sum_{i=1}^{n} x_i = 0 \end{cases}$$

which is equivalent to

$$\begin{cases} n\theta_0 - \sum_{i=1}^{n} y_i + \theta_1 \sum_{i=1}^{n} x_i = 0 \\ \theta_1 \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i y_i + \theta_0 \sum_{i=1}^{n} x_i = 0 \end{cases}.$$

We can isolate θ_0 from the first equation:

$$\theta_0 = \frac{\sum_{i=1}^{n} y_i - \theta_1 \sum_{i=1}^{n} x_i}{n},$$

and substitute it in the second one

$$\theta_1 \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i y_i + \frac{\sum_{i=1}^{n} y_i - \theta_1 \sum_{i=1}^{n} x_i}{n} \sum_{i=1}^{n} x_i = 0,$$

which is equivalent to

$$\theta_1 \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i y_i + \frac{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i - \theta_1 \left[\sum_{i=1}^{n} x_i\right]^2}{n} = 0$$

or

$$\theta_1 \left[\sum_{i=1}^{n} x_i^2 - \frac{\left[\sum_{i=1}^{n} x_i\right]^2}{n}\right] - \sum_{i=1}^{n} x_i y_i + \frac{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i}{n} = 0.$$

At this point, we can divide everything by n, yielding:

$$\theta_1 \left[\frac{\sum_{i=1}^{n} x_i^2}{n} - \frac{\left[\sum_{i=1}^{n} x_i\right]^2}{n^2}\right] - \frac{\sum_{i=1}^{n} x_i y_i}{n} + \frac{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i}{n^2} = 0.$$

If we now assume that the observations are equiprobable, i.e., $P(x_i) = \frac{1}{n}$, and we call X the random variable from which observations x_i are obtained and the same for the observations y_i, obtained from Y, then:

$$\frac{\sum_{i=1}^{n} x_i^2}{n} = E\left[X^2\right], \ \frac{\left[\sum_{i=1}^{n} x_i\right]^2}{n^2} = E\left[X\right]^2, \ \frac{\sum_{i=1}^{n} x_i y_i}{n} = E\left[XY\right], \ \frac{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i}{n^2} = E\left[X\right] E\left[Y\right].$$

This means that the previous equation can be rewritten as:

$$\theta_1 \left(E\left[X^2\right] - E\left[X\right]^2\right) - \left(E\left[XY\right] - E\left[X\right] E\left[Y\right]\right) = 0 \iff \theta_1 Var\left[X\right] - Cov\left[X, Y\right] = 0$$

So

$$\theta_1 = \frac{Cov\left[X, Y\right]}{Var\left[X\right]},$$

$$\theta_0 = E\left[Y\right] - \theta_1 E\left[X\right].$$

2.2.2 Least squares regression: multivariate case

Now, we can assume that we have m independent variables $X_1, ..., X_m$ which we want to use to predict the dependent variable Y. Again, we have n observations of each variable. Now, we want to construct an hyperplane in $\mathbb{R}^{m+1}$, whose predictions would be obtained as

$$\hat{y}(X_i) = \theta_0 + \theta_1 x_{i1} + ... + \theta_m x_{im} = \theta_0 + \sum_{j=1}^{m} \theta_j x_{ij} = \sum_{j=0}^{m} \theta_j x_{ij},$$

where we define $x_{i0} = 1$, for all i. We can represent

$$X = (x_{ij})_{i=1,...,n;\ j=1,...,m}, \ Y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}, \ \theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_m \end{bmatrix},$$

so that we can write

$$\hat{Y} = X\theta.$$

The error function is defined as in the simple case

$$J(\theta) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2,$$

but now we can rewrite this as

$$J(\theta) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \left(Y - \hat{Y}\right)^T \left(Y - \hat{Y}\right) = (Y - X\theta)^T (Y - X\theta).$$

Again, to obtain θ we need to optimize this function using matrix calculus.

Lemma 2.1. *If* $A = \begin{bmatrix} a_{11} & ... & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & ... & a_{nm} \end{bmatrix} \in \mathcal{M}_{n\times m}(\mathbb{R})$, $\theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix} \in \mathbb{R}^m$ *and* $B = \begin{bmatrix} b_{11} & ... & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & ... & b_{nn} \end{bmatrix} \in \mathcal{M}_{m\times m}(\mathbb{R})$ *is a symmetric matrix, it holds:*

$$\frac{\partial A\theta}{\partial \theta} = A,$$

$$\frac{\partial \theta^T A^T}{\partial \theta} = A,$$

and

$$\frac{\partial \theta^T B\theta}{\partial \theta} = 2\theta^T B^T.$$

Proof. First, notice that $A\theta = \begin{bmatrix} \sum_{j=1}^{m} a_{1j}\theta_j \\ \vdots \\ \sum_{j=1}^{m} a_{nj}\theta_j \end{bmatrix} \in \mathbb{R}^n$,so it is

$$\frac{\partial A\theta}{\partial \theta} = \begin{bmatrix} \frac{\partial \sum_{j=1}^{m} a_{1j}\theta_j}{\partial \theta_1} & ... & \frac{\partial \sum_{j=1}^{m} a_{1j}\theta_j}{\partial \theta_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial \sum_{j=1}^{m} a_{nj}\theta_j}{\partial \theta_1} & ... & \frac{\partial \sum_{j=1}^{m} a_{nj}\theta_j}{\partial \theta_m} \end{bmatrix} = \begin{bmatrix} a_{11} & ... & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & ... & a_{nm} \end{bmatrix} = A.$$

For the second result, the procedure is the same.
Lastly, notice that $\theta^T B\theta = \sum_{k=1}^{m}\sum_{j=1}^{m} \theta_k b_{kj} \theta_j \in \mathbb{R}$, so

$$\frac{\partial \theta^T B\theta}{\partial \theta} = \begin{bmatrix} \frac{\partial \sum_{k=1}^{m}\sum_{j=1}^{m} \theta_k b_{kj}\theta_j}{\partial \theta_1} & ... & \frac{\partial \sum_{k=1}^{m}\sum_{j=1}^{m} \theta_k b_{kj}\theta_j}{\partial \theta_m} \end{bmatrix} = \begin{bmatrix} 2\sum_{j=1}^{m} b_{1j}\theta_j & ... & 2\sum_{j=1}^{m} b_{mj}\theta_j \end{bmatrix} = 2[B\theta]^T = 2\theta^T B^T.$$

□

Now, we can proceed and minimize J:

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta} =& \frac{\partial (Y - X\theta)^T (Y - X\theta)}{\theta} \\ =& \frac{\partial}{\partial \theta}\left[Y^T Y - Y^T X\theta - \theta^T X^T Y + \theta^T X^T X\theta\right] \\ =& 0 - Y^T X - Y^T X + 2X^T X\theta \\ =& -2Y^T X + 2\theta^T X^T X,\end{aligned}$$

setting this to be 0, we get

$$\theta^T X^T X = Y^T X \iff X^T X\theta = X^T Y \iff \theta = \left(X^T X\right)^{-1} X^T Y.$$

Thus, the 'best' linear model is given by

$$\theta_{lse} = \left(X^T X\right)^{-1} X^T Y.$$

Once we have this model, if we have an observation of X, $x' = (x'_1, ..., x'_m)$ and we want to make a prediction, we compute

$$y' = x'\theta_{lse}.$$

The approach that we have followed here is the **optimization** view of learning, which basically consists of the steps:

1. Set up an error function as a function of some parameters.
2. Optimize this function to find the suitable values for this parameters, assuming the data as given.
3. Use incoming values to make predictions.

2.2.3 Computation of least squares solution via the singular values decomposition (SVD)

Inverting $X^T X$ can entail numerical problems, so the SVD can be used instead.

Theorem 2.1. *Any matrix $A \in \mathbb{R}^{m\times n}$, $m > n$, can be expressed as*

$$A = U\Sigma V^T,$$

where $U \in \mathbb{R}^{m\times n}$ has orthonormal columns $\left(U^T U = I\right)$, $\Sigma \in \mathbb{R}^{n\times n}$ is diagonal and contains the singular values in its diagonal and $V \in \mathbb{R}^{n\times n}$ is orthonormal $\left(V^{-1} = V^T\right)$.

Proof. Let $C = A^T A \in \mathbb{R}^{n\times n}$. C is square, symmetric and positive semidefinite. Therefore, C is diagonalizable, so it can be written as

$$C = V\Lambda V^T,$$

where $V = (v_i)_{i=1,...,n}$ is orthogonal and $\Lambda = diag(\lambda_1, ..., \lambda_d)$ with $\lambda_1 \geq ... \geq \lambda_r > 0 = \lambda_{r+1} = ... = \lambda_n$ and r is $rank(A) \leq n$.
Now, define $\sigma_i = \sqrt{\lambda_i}$, and form the matrix

$$\Sigma = \begin{bmatrix} diag(\sigma_1, ..., \sigma_r) & 0_{r\times(n-r)} \\ 0_{(m-r)\times r} & 0_{(m-r)\times(n-r)} \end{bmatrix}.$$

Define also

$$u_i = \frac{1}{\sigma_i} X v_i \in \mathbb{R}^m, i = 1, ..., r.$$

Then, this vectors are orthonormal:

$$u_i^T u_j = \left(\frac{1}{\sigma_i} X v_i\right)^T \left(\frac{1}{\sigma_j} X v_j\right) = \frac{1}{\sigma_i\sigma_j} v_i^T X^T X v_j = \frac{1}{\sigma_i\sigma_j} v_i^T C v_j \overset{*}{=} \frac{1}{\sigma_i\sigma_j} v_i^T (\lambda_j v_j) \overset{(\lambda_j=\sigma_j^2)}{=} \frac{\sigma_j}{\sigma_i} v_i^T v_j \overset{**}{=} 0,$$

where $(*)$ is because V is formed with the eigenvectors of C, and $(**)$ is because V is orthonormal.
Now, we can complete the base with $u_{r+1}, ..., u_n$ (using Gram-Schmidt) in such a way that

$$U = [u_1, ..., u_r, u_{r+1}, ..., u_n] \in \mathbb{R}^{n \times n}$$

is column orthonormal.
Now, if it is the case that $XV = U\Sigma$, then

$$X = XVV^T = U\Sigma V^T,$$

so it is only left to see that indeed this holds. Consider two cases:

- $1 \leq i \leq r$: $Xv_i = u_i\Sigma$ by the definition of u_i.
- $r+1 \leq i \leq n$: It is $Xv_i = 0$, because $X^TXv_i = Cv_i = \lambda_i v_i \overset{i \geq r}{=} 0$. As $X, v_i \neq 0$, it must be $Xv_i = 0$. On the other side of the equation we also have 0 because $u_i\Sigma = u_i\sigma_i = 0$ as $i > r$.

□

This, added to the fact that if X has full rank, X^TX is invertible and all its eigenvalues non null, gives us:

$$\begin{aligned}
\theta_{lse} &= \left(X^TX\right)^{-1} X^Ty \\
&= \left(\left(U\Sigma V^T\right) U\Sigma V^T\right)^{-1} \left(U\Sigma V^T\right)^T y \\
&= \left(V\Sigma U^T U\Sigma V^T\right)^{-1} V\Sigma U^T y \\
&= \Sigma^{-2} V\Sigma U^T y = V\Sigma^{-1}U^T y \\
&= V \cdot diag\left(\frac{1}{\sigma_1}, ..., \frac{1}{\sigma_n}\right) \cdot U^T y.
\end{aligned}$$

Intuitive interpretation

The intuition behind the SVD is summarized in Figure 2. Basically, every linear transformation can be decomposed into a rotation, a scaling and a simpler transformation (column orthogonal).

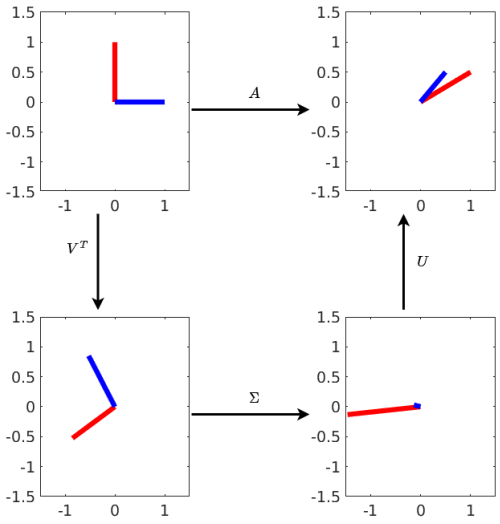


Figure 2: SVD visual.

The intuition behind SVD lies in the idea of finding a low-rank approximation of a given matrix. The rank of a matrix is the number of linearly independent rows or columns it contains. A high-rank matrix has many linearly independent rows or columns, which makes it complex and difficult to analyze. On the other hand, a low-rank matrix has fewer linearly independent rows or columns, which makes it simpler and easier to analyze.

SVD provides a way to find the best possible low-rank approximation of a given matrix by decomposing it into three components. The left singular vectors represent the **direction of maximum variance** in the data, while the right singular vectors represent the **direction of maximum correlation** between the variables. The singular values represent the **magnitude of the variance or correlation** in each direction.
By truncating the diagonal matrix of singular values to keep only the top-k values, we can obtain a low-rank approximation of the original matrix that retains most of the important information. This is useful for reducing the dimensionality of data, compressing images, and solving linear equations, among other applications.

Example 2.1. How to use SVD in Python and Matlab.

```python
import numpy as np

U, d, Vt = np.linalg.svd(X, full_matrices=False)
D = np.diag(1/d)
theta = Vt.T @ D @ U.T @ y
```

Algorithm 1: SVD in Python.

```matlab
[U, d, V] = svd(X)
D = diag(diag(1./d))
theta = V'*D*U'*y
```

Algorithm 2: SVD in Matlab.

2.3 Things that could go wrong when using linear regression

2.3.1 Our independent variable is not enough

It is possible that our variable X does not provide enough information to predict Y.

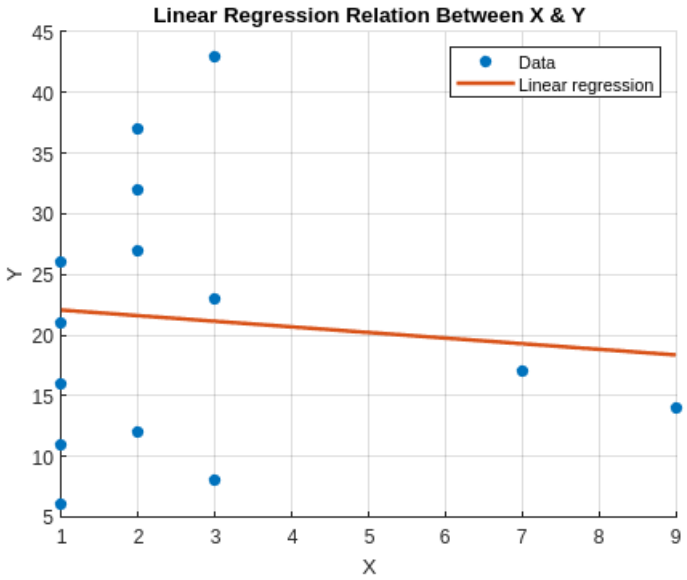


Figure 3: The independent variable does not provide enough information.

2.3.2 The relationship between the variables is not linear (underfitting)

It is also possible that the variables are related in non-linear ways.

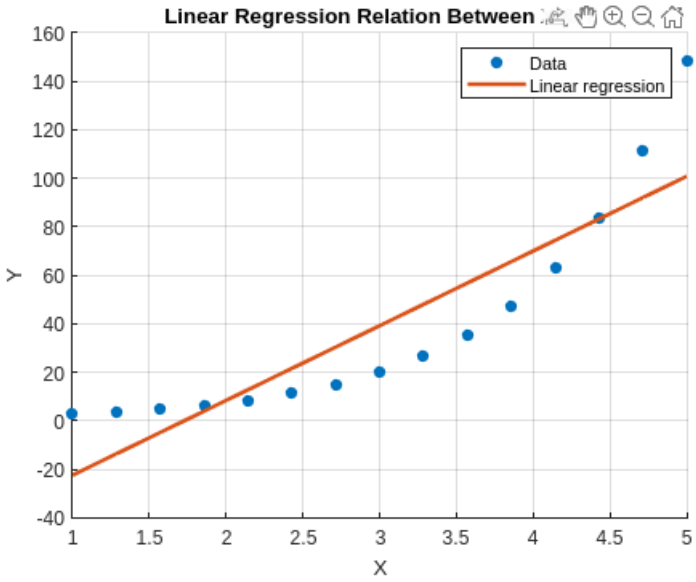


Figure 4: The variables are not linearly related.

2.3.3 Outliers affect the fit

In the presence of outliers, the model obtained can be distorted, leading to bad results.

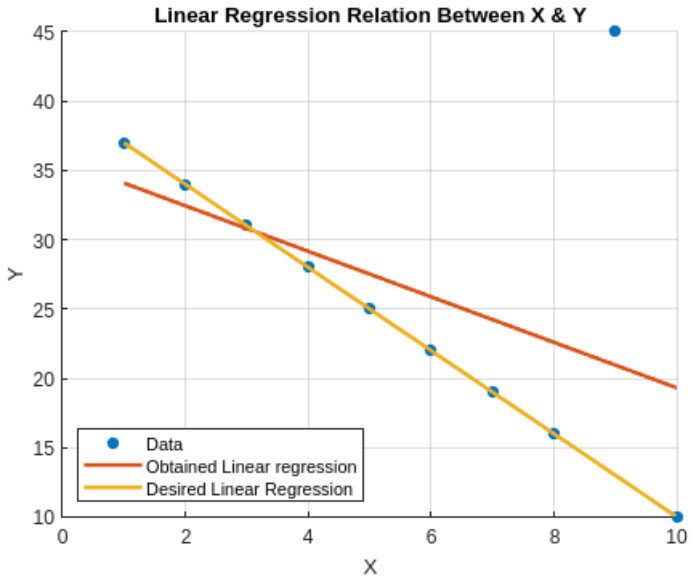


Figure 5: The outlier distort the fit.

2.4 Basis Functions

In order to fix the second problem (Subsection 2.3.2), we can make use of basis functions. The idea is to apply different transformations to the data, so that we can extend the expressive power of our model.

Definition 2.1. A **feature mapping** is a non-linear transformation of the inputs $\phi : \mathbb{R}^m \to \mathbb{R}^k$. The resulting **predictive function** or model is $y = \phi(x)\theta$.

Example 2.2. For example, we can consider the **polynomial expansion of degree** k, which is a commonly used feature mapping that approximates the relationship between the independent variable x and the dependent variable y to be polynomial of degree k, i.e.:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + ... + \theta_k x^k.$$

The feature mapping is $\phi(x) = \begin{pmatrix} 1 & x & x^2 & ... & x^k \end{pmatrix}$.

Note that the idea is to transform the data so that the fit is still linear, even if the relationship is not. Of course, this requires to apply the same transformation whenever we receive an input for which we want to make predictions. Also, the resulting model is more complex, so **complexity control** is necessary to avoid overfitting.
When we apply ϕ to the input matrix X, we get a new input matrix, given by

$$\Phi = \begin{pmatrix} \phi(x_1) \\ \phi(x_2) \\ \vdots \\ \phi(x_n) \end{pmatrix} = \begin{pmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_m(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_m(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_n) & \phi_2(x_n) & \dots & \phi_m(x_n) \end{pmatrix},$$

and we obtain the optimal solution as before:

$$\theta_{min} = \arg\min_{\theta} (y - \Phi\theta)^T (y - \Phi\theta) = \left(\Phi^T \Phi\right)^{-1} \Phi^T y.$$

Example 2.3. A MATLAB example

Example 2.3

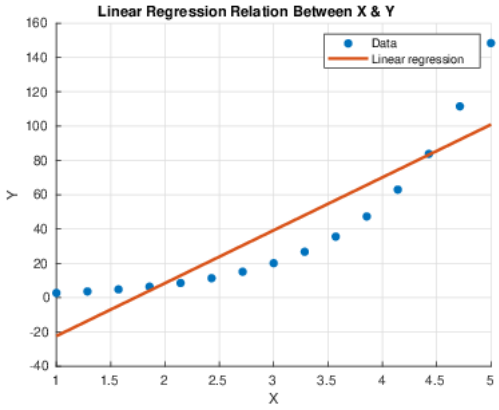
First, we define the dataset. In this case $y = e^x$.

```
x=linspace(1,5,15)';
y=exp(x);
```

Now, we first see what happens with linear regression:

```
b1 = X\y;
yCalc1 = X*b1;
figure;

scatter(x,y,'filled')
hold on
plot(x,yCalc1,'LineWidth',2)
xlabel('X')
ylabel('Y')
title('Linear Regression Relation Between X & Y')
legend('Data','Linear regression')
grid on
hold off
```



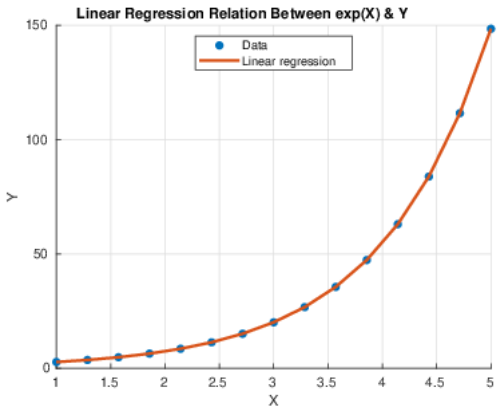
As we can see, the model does not fit the data addecuately. We can use a feature mapping and repeat the process with the transformed input:

```
X = feat_map(x)
```

```
X = 15x3
    1.0000    1.0000    2.7183
    1.0000    1.2857    3.6173
    1.0000    1.5714    4.8135
    1.0000    1.8571    6.4054
    1.0000    2.1429    8.5238
    1.0000    2.4286   11.3427
    1.0000    2.7143   15.0938
    1.0000    3.0000   20.0855
    1.0000    3.2857   26.7281
    1.0000    3.5714   35.5674
```

```
b2 = X\y;
yCalc2 = X*b2;

scatter(x,y,'filled')
hold on
plot(x,yCalc2,'LineWidth',2)
xlabel('X')
ylabel('Y')
title('Linear Regression Relation Between exp(X) & Y')
legend('Data','Linear regression','Location','north')
grid on
hold off
```



As we can see, the model is now perfectly fitting the data!

```
function X=feat_map(x)
    X = [ones(size(x)) x exp(x)];
end
```

References

[1] Marta Arias. Machine learning. Lecture Notes.