INF8245E - Fall 2021

Machine Learning

- Sarath Chandar

2. Regression

2. Regression

Linear models for regression:

Simplest model: Linear regression

$$\hat{y}(x; w) = w_0 + w_1 x_1 + \ldots + w_p x_p$$

This is linear function of the parameters $w_0 \ldots w_p$.

Limitation: This is also a linear function of the input variables $x_1 \ldots x_p$.

Linear basis function model:-

$$\hat{y}(x; w) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(x) \quad \text{---} (1)$$

$\phi_j(x)$ - basis function [a fixed non-linear fn]

$$= \sum_{j=0}^{M-1} w_j \phi_j(x) \qquad [\phi_0(x) = 1]$$

$$= w^T \phi(x)$$

where $w = (w_0 \ldots w_{M-1})^T$

$\phi = (\phi_0 \ldots \phi_{M-1})^T$

function ① is a non-linear function of x.

still, it is linear w.r.t. the parameters and hence a linear model.

Ex: 1) Polynomial basis fn.

$$\phi_j(x) = x^j.$$

2) Gaussian basis fn.

$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$$

μ_j - governs the locations of the basis fns in input space.

s - governs the spatial scale.

Note: This basis fn. does not have a probabilistic interpretation. The normalization constant is unimportant here because the basis fns. will be multiplied by adaptive parameters w_j.

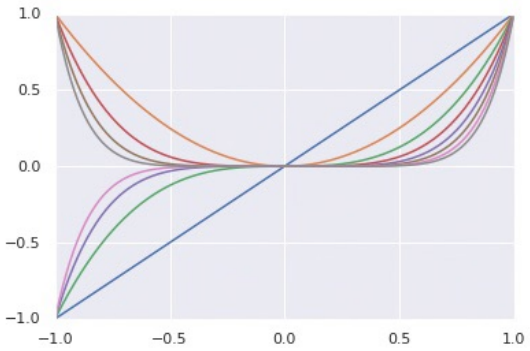
3) Sigmoidal basis fns :-

$$\phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right)$$

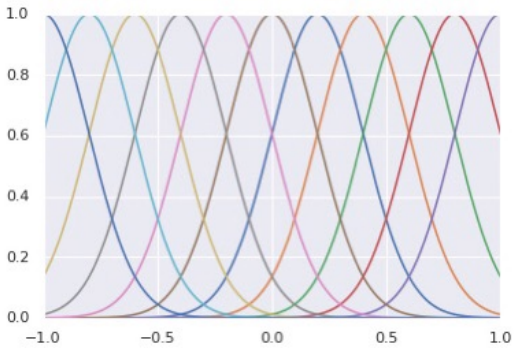
where $\sigma(a) = \frac{1}{1+e^{-a}}$ (logistic sigmoid fn)

4) Identity basis fn:

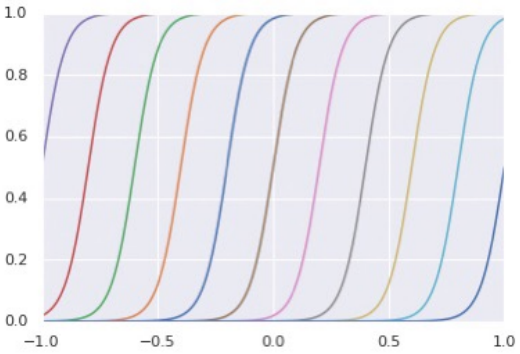
$$\phi(x) = x.$$



Polynomials



Gaussians



Sigmoidal.

Least Squares:-

$$RSS(w) = (y - \phi w)^T (y - \phi w)$$

$$w^* = \underset{w}{\operatorname{argmin}} \; RSS(w)$$

$$w^* = \boxed{(\phi^T \phi)^{-1} \phi^T} y$$

↳ Moore - Penrose pseudo inverse of matrix ϕ.

$$\phi^+ = (\phi^T \phi)^{-1} \phi^T$$

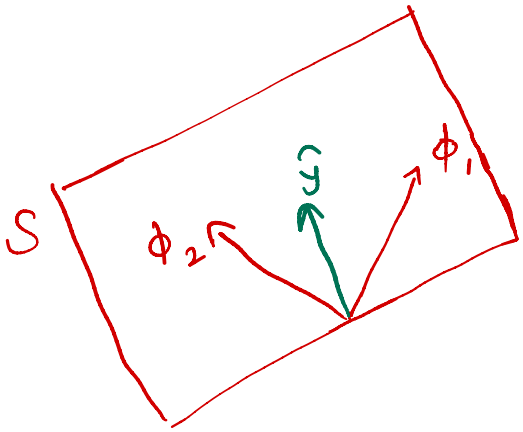
Note: If ϕ is square and invertible, then

$$\phi^+ = \phi^{-1}$$

Geometry of least squares:-

Consider 3 examples, 2 features.

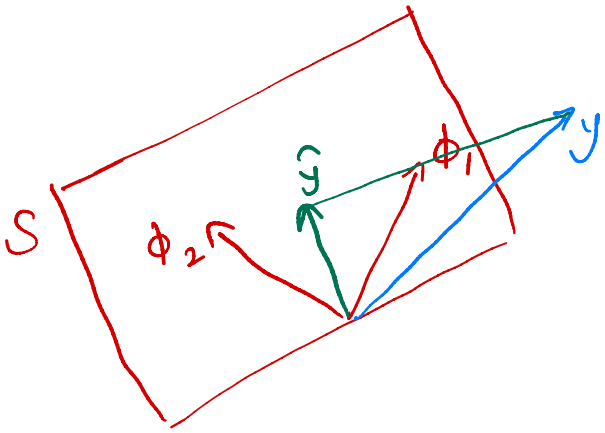
We know that $\hat{y}$ is a linear combination of the vectors ϕ_1 and ϕ_2. So it lives anywhere in the M-dimensional subspace S.



$$\hat{y} = (\hat{y}^{(1)}, \ldots \hat{y}^{(n)})$$

SSE is the squarred euclidean distance between y and $\hat{y}$. Thus least squares solution for w corresponds to that choice of $\hat{y}$ that lies in the subspace S and that is closest to y.

This solution corresponds to the orthogonal projection of y onto the subspace S.



Consider an N-dimensional space whose axis are given by $y^{(n)}$, so that $y = (y^{(1)} \ldots y^{(n)})^T$ is a vector in this space.

Each basis fn. $\phi_j(x^{(n)})$ evaluated at N data points, can also be represented as a vector in the same space, denoted by ϕ_j.

ϕ_j = j^{th} column of ϕ.

$\phi(x^{(n)})$ = n^{th} row of ϕ.

If the number M of basis functions is smaller than the number N of data points, then the M vectors $\phi_j(x^{(n)})$ will span a linear subspace S of dimensionality M.

Issues :-

① If Columns of X are not linearly independant, then X is not of full rank.

ex: two of the input features are perfectly correlated.

ex: $X_2 = 3X_1$.

Then X^TX is singular and w^* is not uniquely defined.

Soln: Filter the redundant features.

② Rank deficiencies can also occur when the number of input features 'p' exceeds the number of training cases 'N'.

Soln: reduce the number of features

(or)

add regularization.

Multiple outputs :-

So far we have considered only single target variable y.

However, y could be a vector. $y \in \mathbb{R}^k$.

How to modify the linear regression model?

Soln 1 : Different set of basis fns. for each component of $y \longrightarrow$ independent regression problems.

Better solution :- Use same set of basis fns. to model all the components of the target vector.

$$\hat{y}(x; w) = W^T \phi(x)$$

where W is an $M \times k$ matrix of parameters.

$$W^* = (\Phi^T \Phi)^{-1} \Phi^T Y$$

where $y = N \times k$ target matrix.

For individual target variable,

$$w_k = (\Phi^T \Phi)^{-1} \Phi^T y_k = \Phi^\dagger y_k.$$

→ The solution to the regression problem decouples between the different target variables.

→ We only need to compute a single pseudo-inverse matrix which is shared by all of the vectors w_k.

Learning by gradient descent :-

Consider a simple regression problem where $x \in \mathbb{R}$ and $y \in \mathbb{R}$.

Linear regression: $\hat{y}(x; w) = w_0 + w_1 x$

Parameters : $w = (w_0, w_1)$

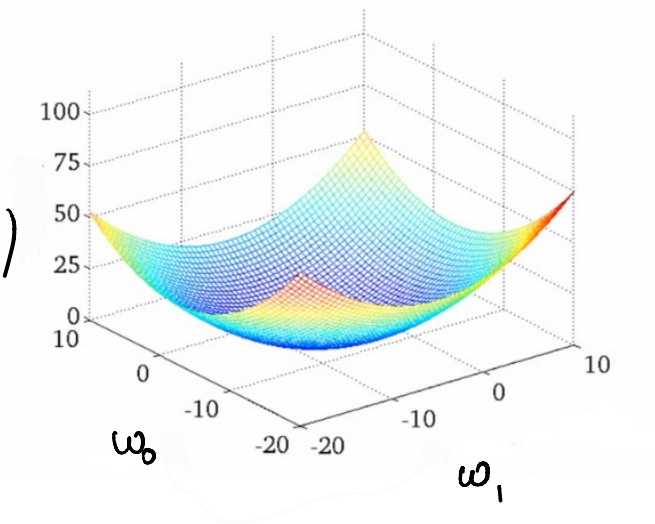
Loss function: $L(w_0, w_1) = \frac{1}{2N} \sum_{i=1}^{N} \left(\hat{y}(x^{(i)}; w) - y^{(i)} \right)^2$

Goal : $\min_{w_0, w_1} L(w_0, w_1)$

We were analytically able to solve this minimization problem. However, there will be situations when it is not tractable to compute the analytical solution. We can use the idea of gradient descent in such situations.

Let us visualize the loss surface for our loss function.

loss surface for our loss function



For every possible (w_0, w_1) this plot shows the corresponding loss.

There is only one minima for this function.

If we can start with a random w_0, w_1 and slowly change our w_0, w_1 such that we move in this loss surface towards the minimum, we can converge at minimum after several steps.

Gradient descent outline :

1. Start with some w_0, w_1.
2. Keep changing w_0, w_1 to reduce $L(w_0, w_1)$ until we hopefully end up at a minimum.

Algorithm:

repeat until Convergence

{

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(w_0, w_1)$$

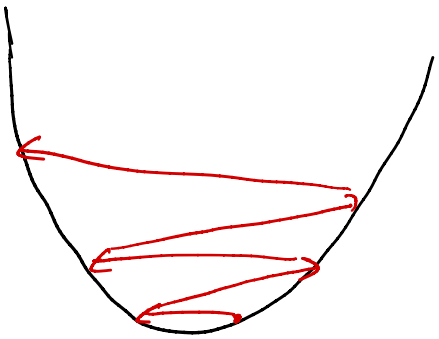
}

step size or learning rate.

[simultaneously update $j=0$ and $j=1$]

Note 1: If α is too small, gradient descent can be slow.

Note 2: If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



Note 3: Gradient descent can converge to a local minimum, even with the learning rate

α fixed. As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time.

Linear regression example:-

$$L(w_0, w_1) = \frac{1}{2N} \sum_{i=1}^{N} \left(w_0 + w_1 x^{(i)} - y^{(i)} \right)^2$$

$$\frac{\partial L}{\partial w_0} = \frac{2}{2N} \sum_{i=1}^{N} \left(\underbrace{w_0 + w_1 x^{(i)}}_{\hat{y}(x^{(i)}; w)} - y^{(i)} \right)$$

$$\frac{\partial L}{\partial w_1} = \frac{2}{2N} \sum_{i=1}^{N} \left(w_0 + w_1 x^{(i)} - y^{(i)} \right) \cdot x^{(i)}$$

Gradient descent algorithm:-

repeat until convergence {

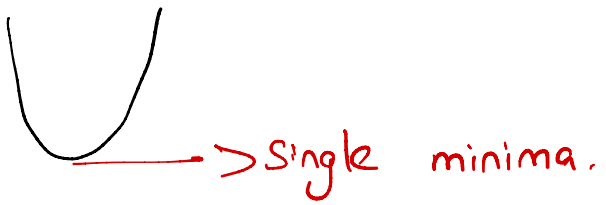
$$w_0 = w_0 - \alpha \frac{1}{N} \sum_{i=1}^{N} \left(\hat{y}(x^{(i)}; w) - y^{(i)} \right)$$

$$w_1 = w_1 - \alpha \frac{1}{N} \sum_{i=1}^{N} \left(\hat{y}(x^{(i)}; w) - y^{(i)} \right) \cdot x^{(i)}$$

}

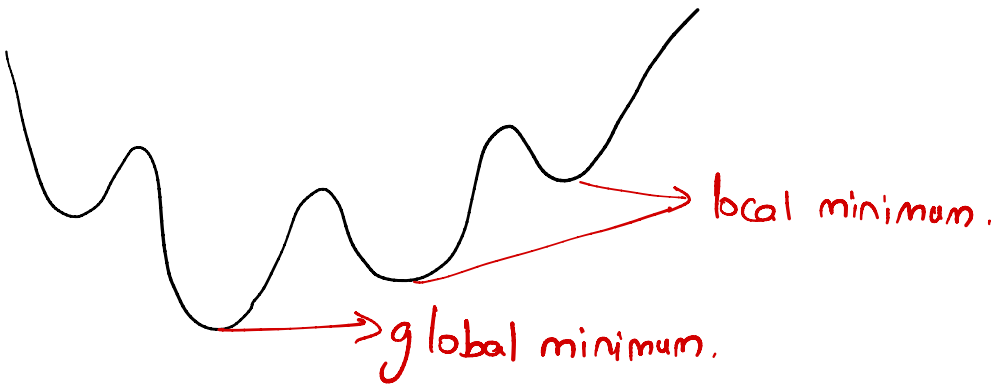
Gradient descent animation.

Special nature of linear regression loss function :-



This is an example of Convex function. Convex functions have unique global minimum. This is because of the use of least square loss with a linear model.

For non-linear models, we might get non-convex loss functions.



Depending on where we start from, gradient descent might get stuck in any of the local minimum. This is the issue of gradient descent with non-convex loss functions. More on this later.

When the dataset is so huge that it takes a lot of time to compute gradients for all the examples and make a single gradient update, we can use stochastic gradient descent.

Stochastic gradient descent : (SGD)

Loss function is a sum of losses for every datapoint.

$$\text{Loss} = \sum_n \text{loss}_n .$$

We can approximate the gradient of this total loss by the gradient of individual datapoint loss.

$$w = w - \alpha \frac{\partial}{\partial w} \text{loss}_k$$ (for a random datapoint k)

This stochastic step only tries to approximate the true gradient. The approximation error can help in escaping the local minimum.

SGD for linear regression :

Consider N datapoints.

repeat until Convergence:

Epoch:

for i in 1 to N:

{

$$w_0 = w_0 - \alpha \left(\hat{y}(x^{(i)}; w) - y^{(i)} \right)$$

$$w_1 = w_1 - \alpha \left(\hat{y}(x^{(i)}; w) - y^{(i)} \right) x^{(i)}$$

}

epoch: One sweep of all the examples in the training set.

Note 1: You should shuffle the order in which you are visiting the data points in the beginning of every epoch. Otherwise you can get stuck with a sequence of gradient updates that might almost cancel each other and hinder learning.

Note 2: Consider a situation where the data observations are arriving in a continuous stream, and predictions must be made before all data points are seen. This in known

as online learning. We can only use SGD in such scenarios.

Note 3: We can get the best of GD and SGD by doing mini-batch SGD. In mini-batch SGD, we sample a mini-batch of 'k' examples and do gradient update using it.

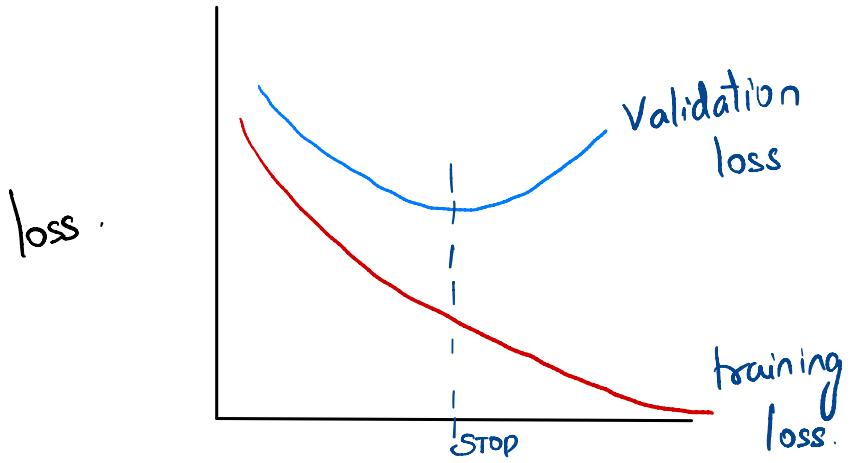
GD → N examples.

SGD → 1 example.

mini-batch GD → 'k' examples $k << N$.

Early Stopping:-

When to stop gradient descent? Training upto zero error in the training set can lead to overfitting. Hence, we can monitor the validation loss after each epoch.



Always save the best model (model with best val loss) so far. Stop training if the validation loss does not decrease for more than 'k' epochs. 'k' - patience.

Regularization:

Adding regularization term to the error function.

↳ Controls overfitting.

total error function : $E_D(w) + \lambda E_w(w)$

λ — regularization coefficient.

Controls the relative importance of $E_w(w)$.

Consider SSE function:

$$E_D(\omega) = (y - X\omega)^T (y - X\omega)$$

L_2 regularization :-

$$E_\omega(\omega) = \omega^T \omega .$$

$$\min_{\omega} \; (y - X\omega)^T (y - X\omega) + \lambda \, \omega^T \omega \quad \text{—— (*)}$$

$$-2 X^T (y - X\omega) + 2\lambda \omega = 0$$

$$(X^T X + \lambda I)\, \omega = X^T y$$

$$\boxed{\omega = (X^T X + \lambda I)^{-1} X^T y}$$

→ The solution adds a positive constant to the diagonal of $X^T X$ before inversion.

→ This makes the problem non-singular even if $X^T X$ is not of full rank!

This is also known as ridge regression / weight decay.

The unconstrained optimization problem (*) can be converted into a constrained

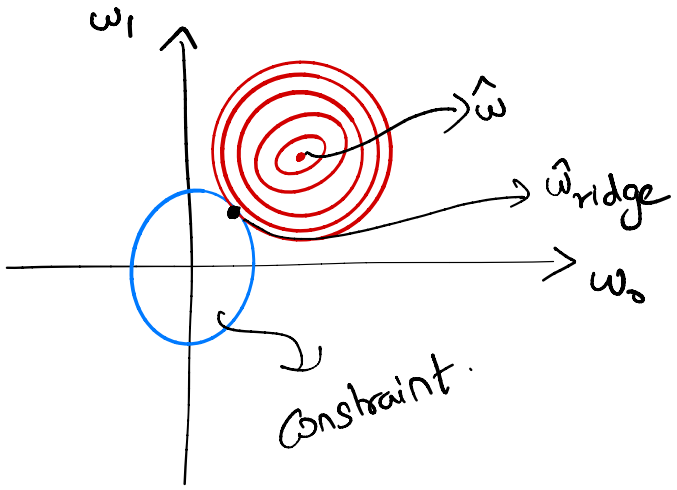
optimization problem :

$$\min_{w} \; (Y - Xw)^T (Y - Xw)$$

$$\text{Subject to} \quad w^T w \leq \eta.$$

η: Circle with radius η over origin in 2D.

One can show that $\eta \propto \frac{1}{\lambda}$.



As we increase the value of λ, radius of circle decreases. So $\hat{w}_{ridge}$ will have smaller magnitude.

L_1 regularization :-

$$E_w(w) = |w|$$

$$\min_{w} (y - xw)^T (y - xw) + \lambda |w|$$

→ There is no closed form solution.

→ This function is not differentiable at $w = 0$.

→ Also known as Lasso.

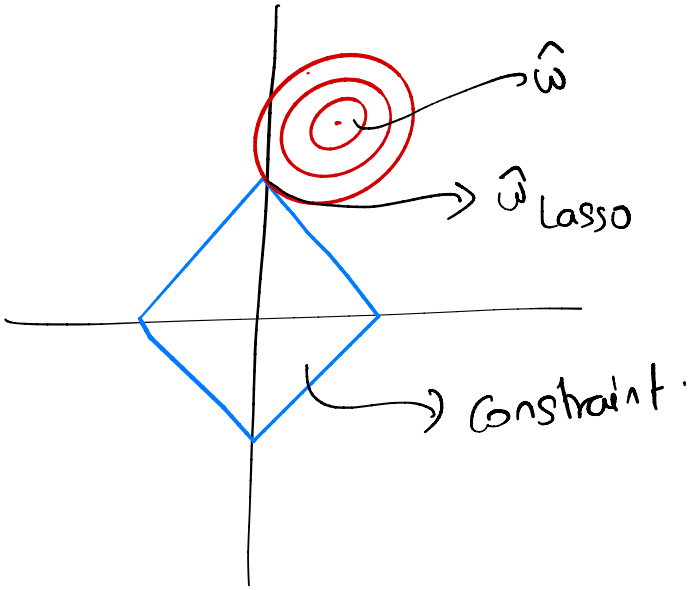
Corresponding Constrained optimization :-

$$\min_{w} (y - xw)^T (y - xw)$$

$$\text{subject to } |w| \leq \eta$$

→ simpler.

where $\eta \propto \frac{1}{\lambda}$



As we increase the values of λ, η decreases and hence more w_i will go to zero.

Note: L_1 regularization does feature selection by setting weights of irrelevant features to zero.

L_1 regularization prefers sparse models.

You should know!

1. Non-linear basis functions
   - Polynomial
   - Gaussian
   - Sigmoidal
2. Geometry of least squares.
3. Regression with multiple outputs.
4. Gradient descent
5. loss surface.
6. Step size or learning rate
7. Convex functions vs. Non-convex functions.
8. Stochastic gradient descent (SGD)
9. Mini-batch gradient descent.
10. Early stopping
11. Constrained optimization formulation for regularization.