Hare
Krishna

Machine Learning

CHEAT SHEET

RAGHAVENDRA BAZARI

Table of Contents

Machine Learning is said as a subset of **artificial intelligence** that is mainly concerned with the development of algorithms which allow a computer to learn from the data and past experiences on their own.

1. Supervised

Problem Statement: Learn a function $f: D^m \to R^1$, which has m dimensional input $\vec{X} = (X_0, X_2, \ldots, X_{m-1}) \vee X_j \in D$ where D is Domain, called features & outputs one value $Y \in R$ where R is range, when given n samples (called *sample size*) from their distribution.

Input: Sample Set $s = n$ samples of $(\vec{X}, Y) = \{(\vec{x}_0, y_0),\ (\vec{x}_1, y_1), \ldots, (\vec{x}_{n-1}, y_{n-1})\}$, where each $\vec{x}_i$ can be thought of as one of the values of independent random vector $\vec{X} = (X_0, X_2, \ldots, X_{m-1})$ sampled from its distribution and corresponding mapped value y_i from dependent random variable Y.

Output: $f: D^m \to R^1$ i.e., $Y = f(\vec{X}) = f(X_0, X_2, \ldots, X)$

Assumptions on *sample set*:

1. n samples of $(\vec{x}_i, y_i)$ are representative of population $(\vec{X}, Y)$ at large.
2. Independent variables $\vec{x}_i$ are measured with no error.
3. No perfect multicollinearity $\rho(x_i, x_j) \neq \pm 1, i \neq j$. In other words, no two samples are exactly same.

a. Regression

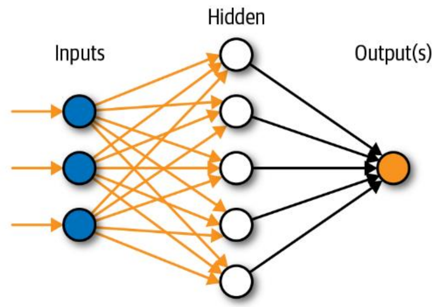
Supervised algorithm is called Regression when Range of Y is Real, i.e., $R = Real$. Domain might be Integral or Real.

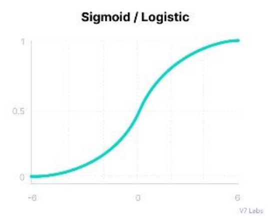
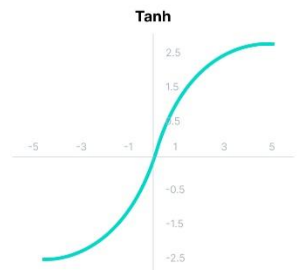
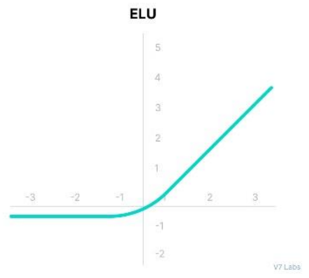
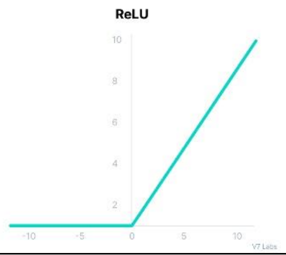
i. Components

1) Basis Function

We start by assuming form of function f, also called *Basis function*, which itself is parameterized by $\vec{\beta} = (\beta_0, \beta_1, \beta_2, \dots, \beta_{k-1})$, so prediction p_i for any sample $\vec{x}_i$ from this function can be represented as

$$p_i = f(\vec{x}_i, \vec{\beta})$$

Linear Functions	$\sum_{i=0}^{m-1} x_i \beta_i + \beta_m$	$k = m + 1$, i.e., number of β_i parameters are one more than number of features (or dimension of Domain).
Polynomial Function (degree d)	$\left(\sum_{i=0}^{m-1} x_i \gamma_i\right)^d$	$k = number\ of\ term\ in\ polynomial\ expansion$ γ_i is just an intermediate parameter, such that $\vec{\beta} = g(\vec{\gamma})$
Neural Network	Neural network can be considered as a composition of non-linear activation function. $$p_{i,j} = f_{i,j}\left(\sum_{k=0}^{n_{l-1}} p_{i-1,k} \times \beta_{i,j,k} + \beta_{i,j,n_{l-1}+1}\right)$$  Where $p_{i,j}$ is prediction from i^{th} layer and j^{th} node. $\beta_{i,j,k}$ is multiplier parameter for i^{th} layer and j^{th} node for input k^{th} node from previous layer. $\beta_{i,j,n_{l-1}+1}$ is additive parameter for i^{th} layer and j^{th} node. $f_{i,j}(.)$ Is activation function for i^{th} layer and j^{th} node. Few common activation Functions:	

	Activation Function	Sigmoid		$f(x) = \dfrac{1}{1+e^{-x}}$
		Tanh		$f(x) = \dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
		Exponential Linear Units(ELU)		$f(x) = \begin{cases} \alpha(e^{x}-1), & x < 0 \\ x, & x \geq 0 \end{cases}$
		Rectified Linear Unit (ReLU)		$f(x) = \max\,(x, 0)$
		More activation Functions can be found here.		

2) Loss or Error Function

Now we define *error* e_i in ith sample in prediction using *Loss function* $L(y_i, p_i)$ which tells us how far prediction is from truth value y_i for that sample $\vec{x_i}$:

$$e_i = L(y_i, p_i) = L\left(y_i, f(\vec{x}_i, \vec{\beta})\right)$$

Squared Loss/Error	$e_i = (y_i - p_i)^2 = \left(y_i - f(\vec{x}_i, \vec{\beta})\right)^2$ $hene, E[e_i] = E\left[\left(y_i - f(\vec{x}_i, \vec{\beta})\right)^2\right]$
Absolute Loss/Error	$e_i = \lvert y_i - p_i \rvert = \lvert y_i - f(\vec{x}_i, \vec{\beta}) \rvert$ $hence, E[e_i] = E[\lvert y_i - f(\vec{x}_i, \vec{\beta}) \rvert]$
Zero-One Loss/Error	$e_i = \begin{cases} 1, & y_i \neq p_i \\ 0, & y_i = p_i \end{cases}$
R^2 Loss /Error (coefficient of determination)	Almost never used while training, only used to test the quality of regression: $E[e_i] = R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$ Where $SS_{res} = E[e_i^2] = \sum_{i=0}^{n-1} \left(y_i - f(\vec{x}_i, \vec{\beta})\right)^2 = Varience\ of\ learner\ from\ ground\ truth$ and $SS_{tot} = E[y_i - \bar{y}\,] = \sum_{i=0}^{n-1} (y_i - \bar{y}) = Total\ Variance$ Where $\bar{y} = \sum_{i=0}^{n-1} y_i$ In the best case, the modelled values exactly match the observed values, which results in $R^2 = 1$

ii. Learning/Optimization

Our goal is to find values of $\vec{\beta}$ such that Expectation of *error* e_i is minimized, ie.

$$\vec{\beta} = \arg\min_{\vec{\beta}} E[e_i] = \arg\min_{\vec{\beta}} \sum_{i=0}^{n-1} e_i \; = \arg\min_{\vec{\beta}} \sum_{i=0}^{n-1} L\left(y_i, f(\vec{x}_i, \vec{\beta})\right)$$

It is popularly known as optimization problem and is used one way or another in almost every industry.

Many optimization techniques work even with one sample data $(\vec{x}, y)$, where goal is not to approximate $f(\vec{x}, \vec{\beta})$ (as its already known) rather finding the best values of $\vec{\beta}$ which will minimize the error in that single sample point. In other words, some of these will work even with $n = 1$.

Some popular methods:

1) Least Square Method

In addition to Assumptions on *sample set*:

, Least Square Method makes these assumptions:

1. Basis Function = Linear Functions, i.e., $f(\vec{x}_i, \vec{\beta}) = \sum_{i=0}^{m-1} x_i \beta_i$
2. Loss/Error Function = Squared Loss/Error, i.e.,

$$e_i = (y_i - p_i)^2 = \left(y_i - f(\vec{x}_i, \vec{\beta})\right)^2$$

3. As we train the model, different values of $\vec{\beta}$ will generate different vales of e_i. Hence each e_i itself is a random variable with following assumptions:
    a. $E[e_i] = 0 \; \forall \; i$
    b. $Var[e_i] = \sigma^2 \; \forall \; i$
    c. $Covar[e_i, e_j] = 0 \forall i, j, i \neq j$, i.e., Covariance Matrix is $\sigma^2 I_n$.
    d. e_i and x_i are independent.

It is also known as Linear Regression method.

a) Unregularized Linear Regression (Ordinary Least Square)

Here to compute $\vec{\beta}$

$$\begin{aligned}
\vec{\beta} &= \arg\min_{\vec{\beta}} E[e] \\
&= \arg\min_{\vec{\beta}} \sum_{i=0}^{n-1} e_i \\
&= \arg\min_{\vec{\beta}} \sum_{i=0}^{n-1} \left(y_i - f(\vec{x}_i, \vec{\beta})\right)^2 \\
&= \arg\min_{\vec{\beta}} \sum_{i=0}^{n-1} \left(y_i - \sum_{i=0}^{m-1} x_i \beta_i\right)^2 \\
&= \arg\min_{\vec{\beta}} \left(\left|Y - X\vec{\beta}\right|^2 \right) \\
&= \arg\min_{\vec{\beta}} \left(\left(Y - X\vec{\beta}\right)^T \left(Y - X\vec{\beta}\right) \right)
\end{aligned}$$

Let $Z = \left(Y - X\vec{\beta}\right)^T \left(Y - X\vec{\beta}\right)$ *and* diff wrt $\vec{\beta}$ & equate that to zero:

$$\frac{\partial Z}{\partial \vec{\beta}} = \left(Y - X\vec{\beta}\right)^T \times (-X) + (-X)^T \left(Y - X\vec{\beta}\right) = 0$$

since $\left(Y - X\vec{\beta}\right)^T \times (-X)\ is\ 1 \times 1\ matrix\ then\ it\ can\ be\ written\ as\ its\ transpose$

$$\begin{aligned}
&2(-X)^T \left(Y - X\vec{\beta}\right) = 0 \\
&X^T Y = X^T X \vec{\beta} \\
&\vec{\beta} = (X^T X)^{-1} (X^T Y)
\end{aligned}$$

b) Regularized Linear Regression

Plain OLS regression suffers from overfitting in case correlation between two features of $\vec{X}$, let's say two columns $\overrightarrow{X_i}$ & $\overrightarrow{X_j}$ are quite same. In extreme case when they are exactly same, i.e., perfectly correlated then X^TX will have two rows exactly same, i.e., its determinant will be zero hence we will not be able to solve, $\vec{\beta} = (X^TX)^{-1}(X^TY)$ as inverse of X^TX won't exist $\left(\frac{1}{0}\text{ case}\right)$ or in other words, solution for $\vec{\beta}$ will be infinite. To avoid this overfitting (with large variance), we add a regularization in Loss function.

Lasso Regression (L1 Norm)

Lasso stands for Least Absolute Shrinkage Selector Operator. Lasso assigns a penalty to the coefficients in the linear model using the formula below and eliminates variables with coefficients that zero. This is called shrinkage or the process where data values are shrunk to a central point such as a mean.

Here to compute $\vec{\beta}$

$$\vec{\beta} = \arg\min_{\vec{\beta}} E[e + \lambda^2|\overrightarrow{\beta}|]$$

$$= \arg\min_{\vec{\beta}} \left(\sum_{i=0}^{n-1} e_i + \lambda^2 \sum_{i=0}^{m-1} |\beta_i| \right)$$

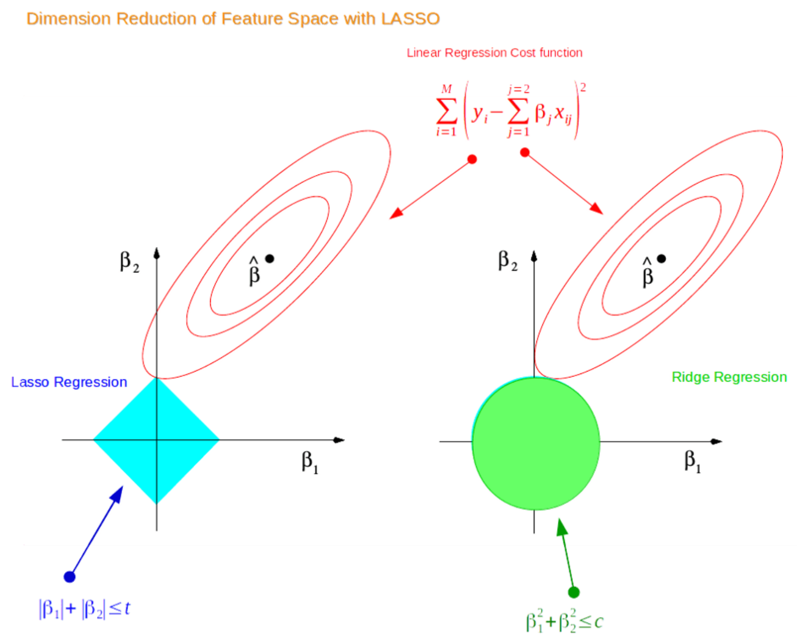
In other words,

$$= \arg\min_{\vec{\beta}} \left(\sum_{i=0}^{n-1} e_i \right) \mathit{conditional\ on} \sum_{i=0}^{m-1} |\beta_i| < c \mathit{\ for\ some\ positive\ } c$$

Properties:

1. Some of the parameters β_i can go to zero. i.e., provide sparse solution.
2. Useful when we have large number of features in model.
3. Lasso does not work well with multicollinearity. If you are unfamiliar, multicollinearity occurs when some of the dependent variables are correlated with each other. Why? Lasso might randomly choose one of the multicollinear variables without understanding the context. Such an action might eliminate relevant independent variables.

Ridge Regression (L2 Norm)	Here to compute $\vec{\beta}$ $\vec{\beta} = \arg\min_{\vec{\beta}} E[e + \lambda^2 \vec{\beta}^2]$ $= \arg\min_{\vec{\beta}} \left(\sum_{i=0}^{n-1} e_i + \lambda^2 \sum_{i=0}^{m-1} \beta_i^2 \right)$ In other words, $= \arg\min_{\vec{\beta}} \left(\sum_{i=0}^{n-1} e_i \right) \mathit{conditional\ on} \sum_{i=0}^{m-1} \beta_i^2 < c \mathit{\ for\ some\ positive\ } c$ Solving above, we get: $\vec{\beta} = (X^T X + \lambda^2 I)^{-1}(X^T Y)$ Properties: 1. β_i are minimized but does not tend to zero. i.e., provide dense solution. 2. $Inti$utation, Ridge regression can also be thought of as augmenting $X_{new} = \binom{X}{\lambda I}$ and corresponding $Y_{new} = \binom{Y}{O}$ i.e. we added m more data points in original samples, hence increasing the dimensions (*sample size*) of $\vec{X}$ from n to $n + m$. (If we solve now for X_{new} & Y_{new} like how we derived for linear solver, we will get above error function to minimize. Basically, we removed the possibility for any two vectors $\overrightarrow{X_i}$ & $\overrightarrow{X_j}$ to be highly correlated, hence avoided overfitting by paying the price for increasing the Bias a bit.
Elastic Net Regression	Elastic net combines ridge (L2 norm) and lasso (L1 norm) to train the model. The objective is to minimize $\vec{\beta} = \arg\min_{\vec{\beta}} E[e + \lambda^2 \vec{\beta}^2]$ $= \arg\min_{\vec{\beta}} \left(\sum_{i=0}^{n-1} e_i + \lambda^2 \sum_{i=0}^{m-1} \beta_i^2 + \lambda^2 \sum_{i=0}^{m-1} \lvert\beta_i\rvert \right)$ Properties: • Elastic Net combines characteristics of both lasso and ridge. Elastic Net reduces the impact of different features while not eliminating all the features.

Relation between L1 and L2 Norma	Both methods determine coefficients by finding the first point where the elliptical contours hit the region of constraints. The diamond (Lasso) has corners on the axes, unlike the disk, and whenever the elliptical region hits such point, one of the features completely vanishes! For higher dimensional feature space there can be many solutions on the axis with Lasso regression and thus we get only the important features selected.  Dimension Reduction of Feature Space with LASSO Linear Regression Cost function $\sum_{i=1}^{M}\left(y_i-\sum_{j=1}^{j=2}\beta_j x_{ij}\right)^2$ β_2 $\hat{\beta}$ β_1 Lasso Regression Ridge Regression $\|\beta_1\|+\|\beta_2\|\leq t$ $\beta_1^2+\beta_2^2\leq c$

2) Gradient Descent

Gradient descent (GD) is an iterative first-order optimisation algorithm used to find a local minimum/maximum of a given function. Reiterating the problem statement:

$$\vec{\beta} = \arg\min_{\vec{\beta}} E[e_i] = \arg\min_{\vec{\beta}} \sum_{i=0}^{n-1} e_i \; = \arg\min_{\vec{\beta}} \sum_{i=0}^{n-1} L\left(y_i, f(\vec{x}_i, \vec{\beta})\right)$$

Assumptions:

1. Loss Function = Squared Error, i.e.,

$$L\left(y_i, f(\vec{x}_i, \vec{\beta})\right) = \left(y_i - f(\vec{x}_i, \vec{\beta})\right)^2$$

2. $f(\vec{x}_i, \vec{\beta})$ is differentiable for all $\vec{\beta}$
3. $f(\vec{x}_i, \vec{\beta})$is convex, i.e., $\frac{\partial^2 f(\vec{x}_i, \vec{\beta})}{\partial \vec{\beta^2}} > 0$ for all $\vec{\beta}$
   a. If it's not convex for all $\vec{\beta}$ then Gradient Descent does not promise to return global minimum, optimization can get stuck in local minimum.
4. $f(\vec{x}_i, \vec{\beta})$ follows Euler Approximation:

$$f(\vec{x}_i\, \vec{\beta} + \vec{\delta}) = f(\vec{x}_i, \vec{\beta}) + J_i(\vec{\beta})\, \vec{\delta}$$

Where, $J_i(\vec{\beta})$ is Jacobian or Gradient of $f(\vec{x}_i, \vec{\beta})$ at $\vec{\beta}$

$$J_i(\vec{\beta}) = \begin{bmatrix} \frac{\partial f(\vec{x}_i, \vec{\beta})}{\partial \beta_0} & \frac{\partial f(\vec{x}_i, \vec{\beta})}{\partial \beta_1} & \dots & \frac{\partial f(\vec{x}_i, \vec{\beta})}{\partial \beta_{k-1}} & \frac{\partial f(\vec{x}_i, \vec{\beta})}{\partial \beta_k} \end{bmatrix}^T$$

And $\vec{\delta}$ is step size also called *learning rate*

$$\vec{\delta} = [\delta_0 \quad \delta_1 \quad \dots \quad \delta_{k-1} \quad \delta_k]$$

a) Unregularized Gradient Descent (Newton/Newton-Raphson/Newton-Fourier/Secant):

1. We start with some specialized/random guess for $\vec{\beta} = (\beta_0, \beta_1, \beta_2, \dots, \beta_{k-1})$.
2. Next, we need to find $\vec{\delta}$ move in opposite direction $\vec{\beta}$ such that $\sum_{i=0}^{n-1}\left(y_i - f(\vec{x}_i, \vec{\beta} + \vec{\delta})\right)^2$ can be minimized, from assumptions we know

$$\begin{aligned}
S(\vec{\beta} + \vec{\delta}) &= \sum_{i=0}^{n-1}\left(y_i - f(\vec{x}_i, \vec{\beta} + \vec{\delta})\right)^2 \\
&= \sum_{i=0}^{n-1}\left(y_i - f(\vec{x}_i, \vec{\beta}) - J_i(\vec{\beta})\vec{\delta}\right)^2 \\
&= \left| Y - F(\vec{\beta}) - J(\vec{\beta})\vec{\delta}\right|^2 \\
&= \left| Y - F(\vec{\beta}) - J(\vec{\beta})\vec{\delta}\right|^T \left| Y - F(\vec{\beta}) - J(\vec{\beta})\vec{\delta}\right| \\
&= \left| Y - F(\vec{\beta})\right|^T \left| Y - F(\vec{\beta})\right| - \left| Y - F(\vec{\beta})\right|^T J(\vec{\beta})\vec{\delta} - J(\vec{\beta})\vec{\delta}\left| Y - F(\vec{\beta})\right| + \vec{\delta}^T J(\vec{\beta})^T J(\vec{\beta})\vec{\delta} \\
&= \left| Y - F(\vec{\beta})\right|^T \left| Y - F(\vec{\beta})\right| - 2\left| Y - F(\vec{\beta})\right|^T J(\vec{\beta})\vec{\delta} + \vec{\delta}^T J(\vec{\beta})^T J(\vec{\beta})\vec{\delta}
\end{aligned}$$

Where Y is $(n \times 1)$ matrix : $[y_0 \quad y_1 \quad \dots \quad y_{n-1} \quad y_n]^T$

$F(\vec{\beta})$ is also $(n \times 1)$ matrix : $\left[f(\vec{x}_0, \vec{\beta}) \quad f(\vec{x}_1, \vec{\beta}) \quad \dots \quad f(\vec{x}_{n-1}, \vec{\beta}) \quad f(\vec{x}_n, \vec{\beta})\right]^T$

$J(\vec{\beta})$ is $(n \times k)$ matrix : $\left[J_i(\vec{\beta})^T \quad J_1(\vec{\beta})^T \quad \dots \quad J_{n-1}(\vec{\beta})^T \quad J_n(\vec{\beta})^T\right]^T$

$\vec{\delta}$ is $(n \times 1)$ matrix

$S(\vec{\beta} + \vec{\delta})$ derivative of w.r.t. $\vec{\delta}$ should be zero for finding best $\vec{\delta}$, i.e.,

$$\frac{\partial S(\vec{\beta} + \vec{\delta})}{\partial \vec{\delta}} = \frac{\partial\left(\left| Y - F(\vec{\beta})\right|^T \left| Y - F(\vec{\beta})\right| - 2\left| Y - F(\vec{\beta})\right|^T J(\vec{\beta})\vec{\delta} + \vec{\delta}^T J(\vec{\beta})^T J(\vec{\beta})\vec{\delta}\right)}{\partial \vec{\delta}}$$

$$= \frac{\partial\left(\left|Y - F(\vec{\beta})\right|^T \left|Y - F(\vec{\beta})\right|\right)}{\partial\vec{\delta}} + \frac{\partial\left(-2\left|Y - F(\vec{\beta})\right|^T J(\vec{\beta})\vec{\delta}\right)}{\partial\vec{\delta}} + \frac{\partial\left(\vec{\delta}^T J(\vec{\beta})^T J(\vec{\beta})\vec{\delta}\right)^T}{\partial\vec{\delta}} \left(using\ \frac{\partial A}{\partial B} = \frac{\partial(A^T)}{\partial B}\ for\ n = 1\right)$$

$$= -2\left|Y - F(\vec{\beta})\right|^T J(\vec{\beta})\frac{\partial\vec{\delta}}{\partial\vec{\delta}} + \frac{\partial\left(J(\vec{\beta})\vec{\delta}\right)^T\left(J(\vec{\beta})\vec{\delta}\right)}{\partial\vec{\delta}} \left(using\ \frac{\partial(CA)}{\partial B} = C\frac{\partial A}{\partial B}\right)$$

$$= -2\left|Y - F(\vec{\beta})\right|^T J(\vec{\beta}) + 2\vec{\delta}^T J(\vec{\beta})^T J(\vec{\beta}) \left(using\ \frac{\partial(DA)}{\partial B} = D\frac{\partial A}{\partial B} + A^T\frac{\partial D^T}{\partial B}\right)$$

Equating it to zero, we get:

$$\vec{\delta} = \frac{J(\vec{\beta})^T \left|Y - F(\vec{\beta})\right|}{J(\vec{\beta})^T J(\vec{\beta})} = \frac{\left|Y - F(\vec{\beta})\right|}{J(\vec{\beta})}$$

i.e., we would get new $\vec{\beta}_{new}$ using

$$\vec{\beta}_{new} = \vec{\beta} + \frac{\left|Y - F(\vec{\beta})\right|}{J(\vec{\beta})}$$

3. Keep repeating 2nd step until error $\left|Y - F(\vec{\beta})\right|^T \left|Y - F(\vec{\beta})\right|$ is under specified tolerance.

Special Cases:

1. When derivative of $f(\vec{x}_i, \vec{\beta})$ is not known, we can use finite difference to compute its Jacobian, then this optimization technique is called Secant method.
2. Single dimension function i.e.,

$$f(\vec{x}_i, \vec{\beta}) = f(x, \beta)$$

- Number of sample set points are one.
- Input dimension of function is one
- Number of parameters to optimize is one.

then $\beta_{new} = \beta + \frac{y - f(x,\beta)}{f'(x,\beta)}$

Example: Solve $\beta^2 = 37$

Let's start with assigning $f(x,\beta) = \beta^2$ & $y = 37$.

1. Let's start with some initial guess for $\beta_0 = 6$
2. $\beta_{new} = \beta + \frac{y-f(x,\beta)}{f\prime(x,\beta)} = 6 + \frac{37-36}{2*6} = 6 + \frac{1}{12} = 6.083$

b) Regularized Gradient Descent (Levenberg–Marquardt):

Like unregularized Square Loss Root Finding method, unregularized Gradient Descent also suffers from overfitting if data is highly collinear. We use $L1$ regularization while optimizing for best $\vec{\delta}$, which give:

$$\vec{\delta} = \frac{J(\vec{\beta})^T \left| Y - F(\vec{\beta}) \right|}{\left(J(\vec{\beta})^T J(\vec{\beta}) + \lambda I \right)}$$

3) Brent Root /Bisection Method

$$\vec{\beta} = \arg\min_{\vec{\beta}} E[e_i] = \arg\min_{\vec{\beta}} \sum_{i=0}^{n-1} e_i \; = \arg\min_{\vec{\beta}} \sum_{i=0}^{n-1} L\left(y_i, f(\vec{x}_i, \vec{\beta})\right)$$

$$Let\ f_{error}(\vec{\beta}) = \sum_{i=0}^{n-1} L\left(y_i, f(\vec{x}_i, \vec{\beta})\right)$$

Assumption

1. $f(\vec{x}_i, \vec{\beta}) = f(x, \beta)$ i.e.,
    a. Number of sample set points are one.
    b. Input dimension of function is one
    c. Number of parameters to optimize is one.

$$f_{error}(\beta) = \; L\big(y, f(x, \beta)\big)$$

2. $f(x, \beta)$ is continuous.

Algorithm:

1. Start with two initial values of β, a and b such that $f_{error}(a)$ & $f_{error}(b)$ have opposite sign.
2. Check $f_{error}(\beta), for\ \beta_{temp} = \frac{a+b}{2}$ and then move boundaries accordingly.

4) Lagrange Multiplier

The method of Lagrange multipliers is a common technique used to find local maximums/minimums of a multivariate function with one or more constraints

Assumption

1. $f(\vec{x}_i, \vec{\beta}) = f(\vec{x}, \vec{\beta})$ i.e.,
   d. Number of sample set points are one.

$$f_{error}(\vec{\beta}) = L(y, f(x, \beta))$$

2. $f(\vec{x}, \vec{\beta})$ is differentiable, with

$$J(\vec{\beta}) = \left[\frac{\partial f(\vec{x}, \vec{\beta})}{\partial \beta_0} \quad \frac{\partial f(\vec{x}, \vec{\beta})}{\partial \beta_1} \quad \cdots \quad \frac{\partial f(\vec{x}, \vec{\beta})}{\partial \beta_{k-1}} \quad \frac{\partial f(\vec{x}, \vec{\beta})}{\partial \beta_k}\right]^T$$

3. There are additional m constraints on $\vec{\beta}$, i.e.,

$$g_0(\beta_0, \beta_1, \beta_2, \dots, \beta_{k-1}) = 0$$
$$g_1(\beta_0, \beta_1, \beta_2, \dots, \beta_{k-1}) = 0$$
$$g_m(\beta_0, \beta_1, \beta_2, \dots, \beta_{k-1}) = 0$$

Algorithm:

1. then necessary and sufficient condition for $f(\vec{x}, \vec{\beta})$ to possess a minima/maximum is:

$$J(\vec{\beta}) + \sum_{i=0}^{m-1} \lambda_i J_{g_i}(\vec{\beta}) = 0 \ \ (K\ equations, one\ for\ each\ \beta_i)$$

   Where λ_i are called Lagrange multipliers.
2. so now we have $(k+m)$ equations and $(k+m)$ variables: $k\ \beta_i$ & m λ_i to solve for.

Example:

What is the minimum distance from origin to plane $2x + 3y + 4z = 12$

Let there be a point $(\beta_0, \beta_1, \beta_2)$ on above plane, which has minimal distance from origin, then we can start with

$$f(\vec{x}, \vec{\beta}) = \beta_0^2 + \beta_1^2 + \beta_2^2$$

With only one constraint,

$$g_0(\beta_0, \beta_1, \beta_2) = 2\beta_0 + 3\beta_1 + 4\beta_2 - 12 = 0$$

Then using Lagrange's,

$$2\beta_0 + 2\lambda_0 = 0$$

$$2\beta_1 + 3\lambda_1 = 0$$

$$2\beta_2 + 4\lambda_3 = 0$$

Solving all these 4 equations for $\beta_0, \beta_1, \beta_2$ and λ, we get: $\beta_0 = \frac{24}{29}, \beta_1 = \frac{36}{29}, \beta_2 = \frac{48}{29}$ and $\lambda = -\frac{24}{29}$

iii. Analysis

1) Main Prediction, p_m

Main Prediction p_m represent the prediction whose Expected distance (as defined by *loss function*) is least from all the possible predictions in P. Therefore, given a learner, with infinite training sets, one can assume that the main prediction would converge to a constant value that is linked with the nature of the learner. It can also be called **general tendency of learner**.

If we generate l different samples set (s_i) from population $(\vec{X}, Y)$ each sample set having n samples, i.e.:

$$\text{Sample sets } S = \{s_0, s_1, \dots, s_l\} \text{ where each sample } s_i = \{(\vec{x}_0, y_0),\ (\vec{x}_1, y_1), \dots, (\vec{x}_{n-1}, y_{n-1})\}_i$$

for a given loss function $L(y_i, p_i)$, we would have learnt l different $\vec{\beta}_i$ one for each set, hence we would have l different versions of f each predicting l different values of any test input $(\vec{x}_k, y_k)$:

$$P = \{p_0, p_1, \dots, p_l\} = \left\{f\left(\vec{x}_k, \overrightarrow{\beta_0}\right), f\left(\vec{x}_k, \overrightarrow{\beta_1}\right), \dots, f\left(\vec{x}_k, \overrightarrow{\beta_{l-1}}\right)\right\}$$

Then *Main Prediction* for that test input $(\vec{x}_k, y_k)$ for all l samples are defined as,

$$p_m = \arg\min_{p'} E_s[L(p', p_i)] = \arg\min_{p'} E_s\left[L\left(p', f\left(\vec{x}_k, \overrightarrow{\beta_\iota}\right)\right)\right]$$

where $E[.]\ is\ Expetation\ w.r.t. to\ all\ sample\ sets$, if all *sample sets* are equiprobable, we can simplify:

$$p_m = \arg\min_{p'} \sum_{i=0}^{l-1} L(p', p_i) = \arg\min_{p'} \sum_{i=0}^{l-1} L\left(p', f\left(\vec{x}_k, \overrightarrow{\beta_\iota}\right)\right)$$

Computation	To compute p_m, re-write above equation as $$p_m = \arg\min_{p'} h(p')$$ where $h(p') = \sum_{i=0}^{l-1} L(p', p_i)$ Now differentiate $h(p')$ w.r.t. p' and equate to zero, to get max/min and confirm using boundaries For few Loss function, we can specialize *Main Prediction*	
	Squared Loss/Error	$$h(p') = \sum_{i=0}^{l-1} (p' - p_i)^2$$ diff & equating to zero $$h'(p') = \sum_{i=0}^{l-1} 2(p' - p_i)^1 = 0$$ Hence, $$p' = \frac{\sum_{i=0}^{l-1} p_i}{l} = Mean\ of\ all\ the\ predictions$$
	Absolute Loss/Error	$$h(p') = \sum_{i=0}^{l-1} \lvert p' - p_i \rvert$$ Assuming p' lis between p_m & p_{m+1}(if we had sort all p_i) then: $$h(p') = \sum_{i=0}^{m} (p' - p_i) + \sum_{i=m+1}^{l-1} (p_i - p')$$ diff & equating to zero $$h'(p') = (m+1) - (l-1-m-1) = 0$$ Hence, $$m = \frac{l-3}{2}$$ Basically $p' = p_m$ where $p_m is\ median\ of\ P$

	Zero-One Loss/Error	$h(p') = \sum_{i=0}^{l-1} \begin{cases} 1, & p' \neq p_i \\ 0, & p' = p_i \end{cases}$ obviously for p' equal to most frequently occurring $p_i$above will be minimum, i.e., p_m will b mode of P.

2) Bias, $B(\vec{x}_k, y_k)$

Bias for that test input $(\vec{x}_k, y_k)$ for all l samples is error in general tendency of learner, i.e., **general tendency for error**. It is defined as,

$$B(\vec{x}_k, y_k) = L(y_k, p_m)$$

In a word, the bias of a learner, with regards to an example, is the loss incurred by the difference between the main prediction (the general tendency) and the actual value of the target attribute (target value).

The closer the main prediction to the target value, the less the loss. The more likely a learner produces a model that gives the right prediction, the less bias the learner has.

Properties:

1. Bias is independent of training sets S.
2. $B(\vec{x}_k, y_k)$ is zero then learner always makes optimal prediction.
3. $B(\vec{x}_k, y_k)$ is high then the model resulted from the learner has a high tendency to produce an erroneous prediction.

3) Variance $V(\vec{x}_k, y_k)$

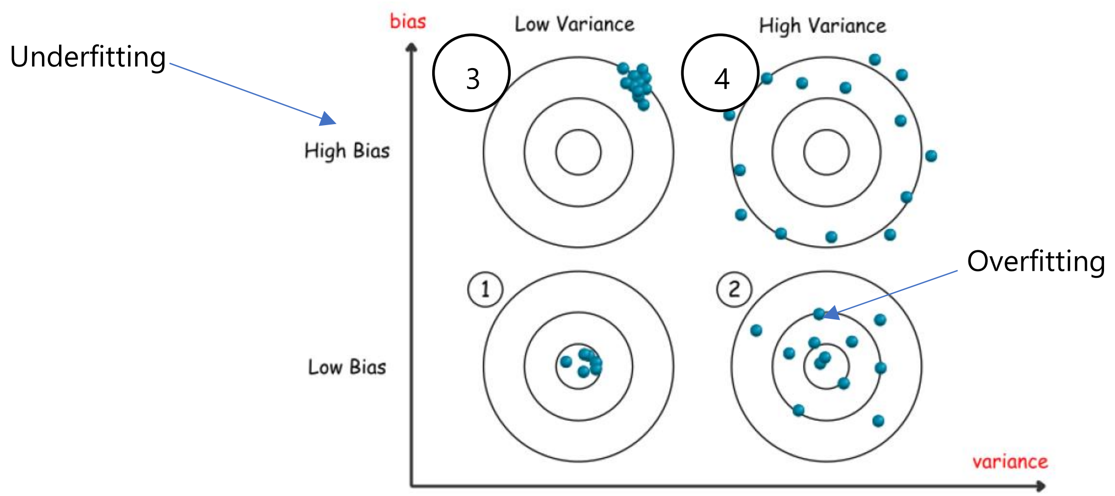
Variance is the average loss incurred by predictions relative to the main prediction. **It measures the loss incurred by its fluctuation around the main prediction in response to different training sets**. It is defined as, (same as what we minimized to get y_k)

$$V(\vec{x}_k, y_k) = E_s[L(p_m, p_i)] = E_s\left[L\left(p_m, f\left(\vec{x}_k, \overrightarrow{\beta_i}\right)\right)\right]$$

Properties:

1. Independent of true value of sample's true label y_k.
2. $V(\vec{x}_k, y_k)$ is zero for a learner that always makes the same prediction regardless of the training set.
3. If a learner has high variance, then we would know that the learner is highly sensitive to the training data set, *i.e.* a minor noise in the training data could lead to an ill-formed model that produces wrong predictions.

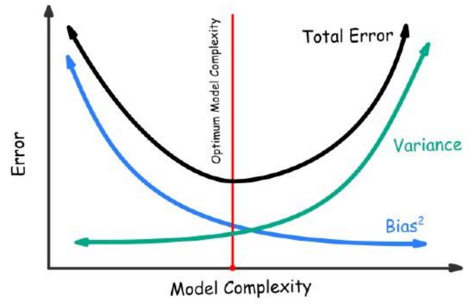
4) Relationship between Bias and Variance



1. An ***ideal learner*** would situate at the low-left part of the plot, where we find the low bias as well as low variance. This is the decent dart player that rarely misses the bullseye. A good player never makes mistakes, *i.e.,* a good learner always makes good predictions.

2. Now, moving to the right-hand side of the ideal learner, we have a ***fair learner*** who manages to score some points (*i.e.* low bias), yet the darts are all over the places (*i.e.* high variance). The learners that are situated in this area are usually complex algorithms that could manage to train some decent models sometimes, but in general, the performance of the model is not too promising. The case where we do not obtain a good model, is also called **overfitting**, *i.e.,* the model pays too much attention to the irrelevant noise.'
3. We then move to the up-left part of the plot, right next to the terrible learner, we find the ***naive learner***, who has high bias yet low variance. The naive learner often adopts some straightforward strategies, which could explain why it produces stable outputs (low variance). But the strategy is too simple to capture the essential information from the data, which results in producing a **underfitted** model.
4. Right next to the naive learner we find the ***terrible learner***, who has both high bias and high variance. The terrible learner is not able to extract information from the data and basically learns nothing. The prediction that learner produces is not relevant (high bias), and what's even worse, the learner is not consistent with its strategy but making a random guess (high variance).

Bias is reduced and variance is increased in relation to the complexity of the models produced by a learner. The correlation between the model complexity and bias-variance can be described in the following graph:



One can draw a few information from the above graph:

- When the model becomes more complex, it could potentially fit better the training data set, as a result, the bias is reduced.
- Meanwhile, when the model becomes more complex, the variance increases, since the model becomes more sensitive to the noise in the data.

Given a learner, often one can adjust its bias and variance, by tuning the parameters involved in the learner.

For example, if we construct a decision tree for a classification problem, without any constraint the decision tree could overgrow itself to fit every bit of the training data set, including the noisy data. As a result, we might obtain a decision tree model with a low bias for the given training data set. However, it could end up with high bias as well as high variance for the unseen data sets, since it ***overfits*** the training data set. To mitigate the problem, one can impose some constraints to limit the growth of a decision tree, *e.g.,* setting the max depth a tree can grow, which might result in higher bias in the training data set. In exchange, we could obtain a model with lower variance in the unseen data set, as well as lower bias, since the model is trained to be more generalized.

Relation between Expected error, Bias, and variance for Square Loss function:

- Expected Error: $E_s[(y_k - p_i)^2]$
- Bias: $(y_k - E_s[p_i])^2$
- Variance: $E_s[(p_i - E_s[p_i])^2]$

$$Expected\ Error = Bias + Variance$$

Note, here we have assumed that Bias is directly computed w.r.t. true value y_k, however some places it is computed w.r.t. $y*$ which is an optimal prediction from learner to minimize $E_s[L(y_k, y*)]$, and a new component is introduced, Noise= $E_s[L(y_k, y*)]$, then

$$Expected\ Error = Bias + Variance + Noise$$

b. Classification

Supervised algorithm is called Classification when Range of Y is Integral, i.e., $R = I$. Domain might be Integral or Real.

Types	• Random Forest • Decision Trees • Logistic Regression • Support vector Machines

2. Unsupervised

The training is provided to the machine with the set of data that has not been labelled, classified, or categorized, and the algorithm needs to act on that data without any supervision.

Types	• Clustering Clustering is a method of grouping the objects into clusters such that objects with most similarities remains into a group and has less or no similarities with the objects of another group. Cluster analysis finds the commonalities between the data objects and categorizes them as per the presence and absence of those commonalities. • Association An association rule is an unsupervised learning method which is used for finding the relationships between variables in the large database. It determines the set of items that occurs together in the dataset. Association rule makes marketing strategy more effective.

3. Reinforcement

Reinforcement learning is a feedback-based learning method, in which a learning agent gets a reward for each right action and gets a penalty for each wrong action.