Artificial Intelligence and Machine Learning

Abhijit Amrendra Kumar

August 2023

Chapter 1

Introduction

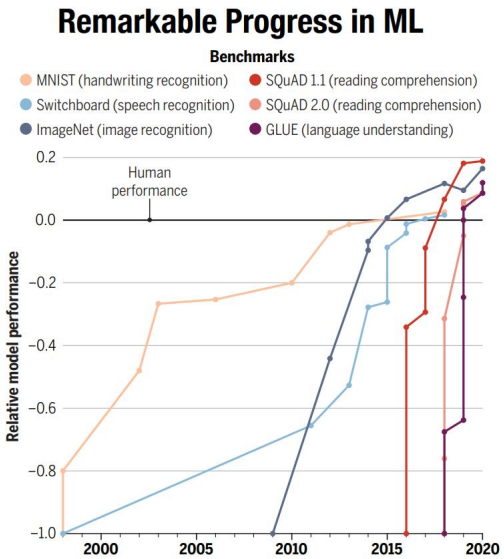
1.1 What is Machine Learning ?

Machine learning is the ability of machines to **learn** from data or past experiences, which come from various sources such as sensors, domain knowledge, experimental runs, etc. Mathematically, it involves making a theoretical function/model $f : X \rightarrow Y$ which predicts outcomes from given inputs, and optimizing it and training it to increase it's prediction accuracy.

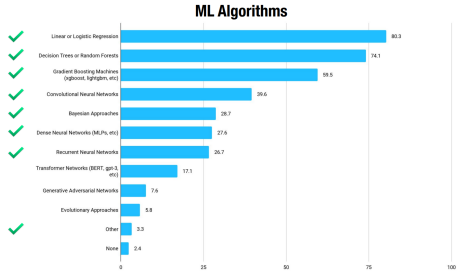
1.2 Why Machine Learning ?

We need machine learning in the following cases

- When we need to perform tasks that are easy for humans, but complex for computer systems to emulate (eg. distinguishing between a muffin and a Chihuahua)
- When we need to perform tasks which are beyond human capabilities, like the analysis of large and complex datasets



Here is a usage histogram for various ML algorithms, which indicates that linear regression and decision trees are very likely to be useful in a machine learning problem.



1.3 ML Pipeline

- **Data**: Collect data for your problem
  - Is the data labelled or unlabelled ?
- **Representation**: Choose features that represent your data
  - Is the data representation raw, expert-derived or learned ?
- **Modeling**: Choose a model for a task
  - Should the model be linear/non-linear ? Also, what will be the computational overheads ?
- **Training/Learning**: Model will (likely) be parameterized, and these parameters will be learned by training over the data. So, choose a loss function to optimize.
  - Which loss function to choose ? And how to best optimise this loss function ?
- **Prediction/Inference**: Given a model, assign labels to unseen test instances, then choose an evaluation metric.
  - Is the evaluation manual or automatic ?

1.4 Resources

Here are some links to find various datasets

- https://archive.ics.uci.edu/
- https://huggingface.co/datasets
- http://www.openslr.org/

Chapter 2

Linear Regression

2.1 Notation

In general we format variables as: x for scalars, $\mathbf{x}$ for vectors, and X for matrices.

- Input / Feature space / Attributes Space : $\mathcal{X} = \mathbb{R}^d$ for some $d \in \mathbb{N}$
- Output / *Label space* / Response space : $\mathcal{Y} = \mathbb{R}$
- Dataset : $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i \in \mathcal{X}, y_i \in \mathcal{Y}\ \forall\ i \in \{1 \dots n\}$

We consider a target function $f : \mathcal{X} \to \mathcal{Y}$ on the training dataset $\mathcal{D}$, i.e. $\mathbf{x} \overset{f}{\mapsto} y\ \forall\ (\mathbf{x}, y) \in \mathcal{D}$.

We focus on the task of finding a *hypothesis function* $h : \mathcal{X} \to \mathcal{Y}$ that ideally closely approximates f. We call the family of the hypothesis functions as $\mathcal{H}$, the *Hypothesis Class*. It follows that $h \in \mathcal{H}$ This now brings the following questions:

1. What are the possibilities for the predictor function h ? [**Hypothesis Class**]
2. How do you quantify the performance of the predictor? [**Loss/Error Function**]
3. How do we find the best predictor? [**Optimization**]

2.2 What are the possible predictor functions?

Let us first play with a simpler case with a one-dimensional feature space. We may consider the problem as a line fitting problem, taking our hypothesis class to be all linear functions.

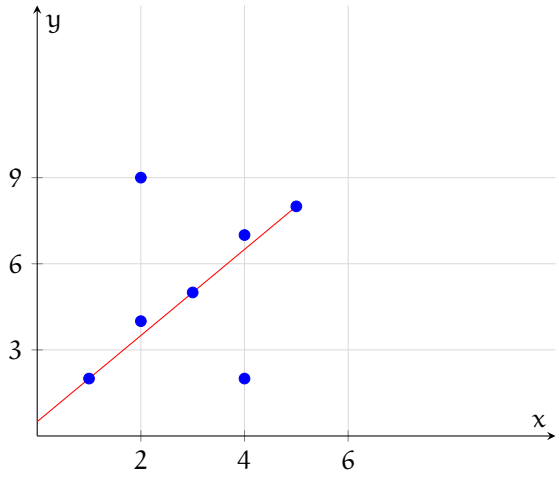


Figure 2.1: Fitting a line to the data

Let us parametrize the line with w_0, w_1 (intercept and slope). We can vectorize our parameters as $W = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$. Then we may write our hypothesis function $h_{\mathbf{w}}$ parametrised by $\mathbf{w}$ as.

$$h_w(x) = w_0 + w_1 x$$

Let us also vectorize our input as $\mathbf{x} = \begin{bmatrix} 1 \\ x \end{bmatrix}$, so that

$$h_w(x) = \mathrm{tr}\{\mathbf{w}\}\mathbf{x}$$

We can now extend this to the multidimensional case. We consider our function to return a linear combination of our d dimensional features.

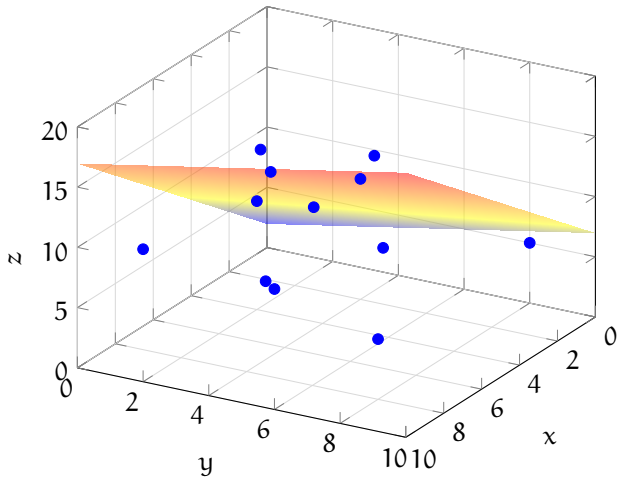


Figure 2.2: For higher dimensional feature spaces, linear regression is akin to *Hyperplane Fitting*

We now have our parameter $\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} \in \mathbb{R}^{d+1}$ and input vector $\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}$ to get a identical expression -

$$h_w(x) = \mathrm{tr}\{\mathbf{w}\}\mathbf{x}$$

Our hypothesis class is, then -

$$\mathcal{H} = \{h_w : \mathbf{w} \in \mathbb{R}^{d+1}\}$$

This is essentially the *linear* in linear regression. However, note that it does not mean we are restricted to linear functions of the features, we may transform the feature space to another space to regress.

2.3 How to quantify the performance of a predictor ?

We define a function that operates on the predictor function and dataset to quantify the "mismatch" between the two. Higher the loss function, lesser is the given predictor suitable for the dataset. Given that our function is parametrized, we may also define the loss function in terms of the parameter.

$$\text{Loss Function: } \mathcal{L}(h, \mathcal{D}) \text{ or } \mathcal{L}(\mathbf{w}, \mathcal{D})$$

Let us consider a singleton dataset $\mathcal{D}_{test} = \{(\mathbf{x}, y)\}$ where $\mathbf{x} = \mathrm{tr}\{[1\ x_1\ \ldots\ x_d]\}$. We define a loss for this dataset as

$$\mathcal{L}(\mathbf{w}, \mathcal{D}_{test}) = (y - h_w(\mathbf{x}))^2 = |y - \hat{y}|^2$$

We define $h_w(x) = \hat{y}$, and $|y - \hat{y}|$ is called a *residual*.

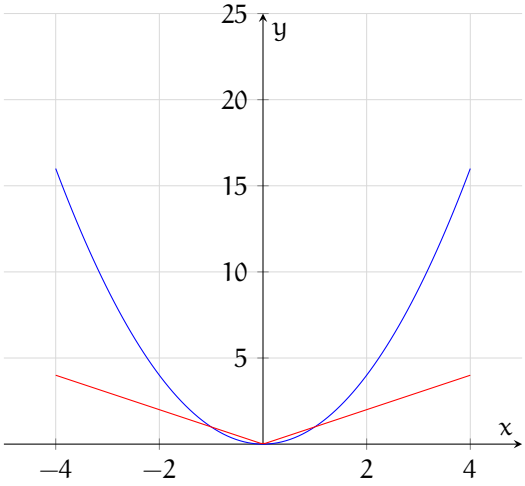
Now for a dataset containing n datapoints,

$$\textbf{Least Squares Loss}: \ \mathcal{L}(\mathbf{w}, \mathcal{D}_{train}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - h_w(\mathbf{x}_i))^2 = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|^2$$

Note that the least squares loss is the mean of the squares of the residuals. We can also define a loss equal to the mean residual value.

$$\text{Mean Absolute Error Loss: } \mathcal{L}(\mathbf{w}, \mathcal{D}_{train}) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

The Mean Absolute Error does not penalise high deviations as much as the Squares Loss, thus it may be more appropriate to use when the dataset contains many outliers.



Here we are interested in the least squares loss. We can vectorize the loss function as follows

$$\boxed{\mathcal{L}(\mathbf{w}, \mathcal{D}_{\text{train}}) = \frac{\mathbf{1}}{\mathbf{n}}\|\mathbf{y}-\mathbf{X}\mathbf{w}\|_2^2}$$

$$\text{, where } X = \begin{bmatrix} \longleftarrow \text{tr}\{\mathbf{x}_1\} \longrightarrow \\ \longleftarrow \text{tr}\{\mathbf{x}_2\} \longrightarrow \\ \vdots \\ \longleftarrow \text{tr}\{\mathbf{x}_n\} \longrightarrow \end{bmatrix}_{n\times(d+1)} \quad , \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}_{n\times 1}$$

2.4 How do we find the best predictor?

We use the loss function to find our optimum hypothesis h^*, where

$$h^* = \arg\min_{h\in\mathcal{H}} \mathcal{L}(h, \mathcal{D}_{train})$$

[Note that the optimisation is performed only over the training dataset]
We may also define the objective in terms of the parameter $\mathbf{w}$ corresponding to h.

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}, \mathcal{D}_{train})$$

$$\mathbf{w}_{LS} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} (y_i - \text{tr}\{\mathbf{w}\}x_i)^2$$

We set out to find a closed form solution for $\mathbf{w}_{LS}$ for d dimensional data:
To find the optimum, we set the derivative[1] to zero. We may drop the $\frac{1}{n}$ term.

$$\nabla_{\mathbf{w}}\mathcal{L} = \frac{\partial\mathcal{L}(\mathbf{w}, \mathcal{D}_{train})}{\partial\mathbf{w}} = \left[\frac{\partial\mathcal{L}(\mathbf{w}, \mathcal{D}_{train})}{\partial w_i}\right]_{i=1}^{n} = [-2(y_i - \text{tr}\{\mathbf{w}\}x_i)x_i]_{i=1}^{n} = 0$$
$$\implies 2(-\text{tr}\{X\}\mathbf{y} + \text{tr}\{X\}X\mathbf{w}) = 0$$
$$\implies \mathbf{w}_{LS} = (\text{tr}\{X\}X)^{-1}\,\text{tr}\{X\}Y$$

We can also derive this result with vector-derivative identities[1],

$$\begin{aligned}
&\nabla_{\mathbf{w}}\mathcal{L} = 0 \\
\implies &\frac{\partial}{\partial \mathbf{w}}\|\mathbf{y} - X\mathbf{w}\|_2^2 = \frac{\partial}{\partial \mathbf{w}}\left(\mathrm{tr}\{\mathbf{y}\}\mathbf{y} - 2\,\mathrm{tr}\{\mathbf{w}\}\,\mathrm{tr}\{X\}\mathbf{y} + \mathrm{tr}\{\mathbf{w}\}\,\mathrm{tr}\{X\}X\mathbf{w}\right) = 0 \\
\implies &-2\,\mathrm{tr}\{X\}\mathbf{y} + 2\,\mathrm{tr}\{X\}X\mathbf{w} = 0 \\
\implies &\mathrm{tr}\{X\}X\mathbf{w} = \mathrm{tr}\{X\}\mathbf{y} \\
&\boxed{\mathbf{w} = (\mathrm{tr}\{X\}X)^{-1}(\mathrm{tr}\{X\}\mathbf{y})}
\end{aligned}$$

Note that $\mathrm{tr}\{X\}X$ need not be invertible.

2.5 Homework

For 1D data, $h_{\mathbf{w}}(x) = w_0 + w_1 x$, prove that –

$$\mathbf{w_1}^* = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

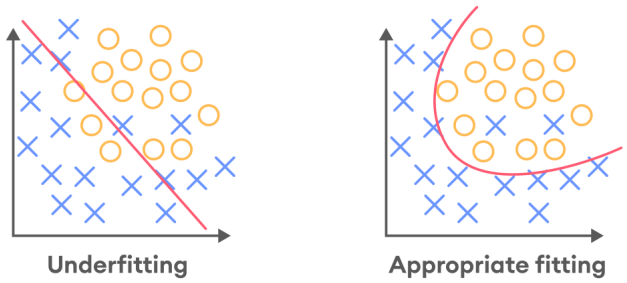
where $\bar{x} = \frac{\sum x_i}{N}$ and $\bar{y} = \frac{\sum y_i}{N}$.

Solution.

$$\begin{aligned}
\mathcal{L} &= \sum_{i=1}^{N}(y_i - w_0 - w_1 x_i)^2 \\
\frac{\partial \mathcal{L}}{\partial w_0} &= -\sum_{i=1}^{N}(y_i - w_0 - w_1 x_i) = 0 \\
\implies w_0^* &= \bar{y} - w_1\bar{x} \\
\frac{\partial \mathcal{L}}{\partial w_1} &= -\sum_{i=1}^{N} x_i(y_i - w_0 - w_1 x_i) = 0 \\
\implies w_1^* &= \frac{\sum_{i=1}^{N} x_i y_i - N w_0^* \bar{x}}{\sum_{i=1}^{N} x_i^2} \\
&= \frac{\sum_{i=1}^{N} x_i y_i - N\bar{x}\bar{y} + w_1^* N\bar{x}^2}{\sum_{i=1}^{N} x_i^2} \\
&= \frac{\sum_{i=1}^{N}(x_i y_i - \bar{x}\bar{y})}{\sum_{i=1}^{N}(x_i - \bar{x})^2} \\
&= \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{N}(x_i - \bar{x})^2}
\end{aligned}$$

[1]There are two conventions for the derivative of a scalar by a vector, i.e., whether the result is a row or column. Here we will stick with the latter.

2.6 Basis Functions



- In the above figure, the left-side model represent $h_w(x) = w_0 + w_1 x$ for some w_0, w_1, but it doesn't fit the given dataset well.
- Right-side model fits the dataset well, and represents $h_w(x) = w_0 + w_1 x + w_2 x^2$ for some $w_0, w_1, w2$.
- This suggests we may want to change/transform the feature space we are working with for improved model fits. We detail the procedure below using basis functions.

Given a basis function $\phi : \mathcal{X} \to \mathbb{R}^{m+1}$, where

$$\phi(\mathbf{x}) = \begin{bmatrix} 1 \\ \phi_1(\mathbf{x}) \\ \phi_2(\mathbf{x}) \\ \vdots \\ \phi_m(\mathbf{x}) \end{bmatrix}_{(m+1)\times 1}$$

The hypothesis space becomes

$$\mathcal{H}_\phi = \left\{ h_w \in \mathcal{X}^{\mathbb{R}} \middle| \, \mathbf{w} \in \mathbb{R}^{m+1}, h_w(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) \text{ for } \mathbf{x} \in \mathcal{X} \right\}$$

The design matrix with basis function-based transformations can be written as:

$$\Phi_{\mathcal{D}} = \begin{bmatrix} \phi(\mathbf{x}_1)^\top \\ \phi(\mathbf{x}_2)^\top \\ \vdots \\ \phi(\mathbf{x}_n)^\top \end{bmatrix}_{n\times(m+1)}$$

Note that $\Phi_{\mathcal{D}}$ yields m-dimensional feature vectors, that could be smaller or larger than the original d-dimensional input feature space.
The least-squares objective using basis functions can be written as:

$$\begin{aligned} \mathcal{L}_{\text{LS}}(h_w, \mathcal{D}) &= \frac{1}{n} \sum_{i=1}^{n} (y_i - h_w(\mathbf{x}_i))^2 \\ &= \frac{1}{n} \sum_{i=1}^{n} (y_i - \mathbf{w}^\top \phi(\mathbf{x}_i))^2 \\ &= \frac{1}{n} \|\mathbf{y} - \Phi_{\mathcal{D}} \mathbf{w}\|_2^2 \end{aligned}$$

and as before

$$\mathbf{w}^*_{\text{LS}} = \left(\Phi_{\mathcal{D}}^\top \Phi_{\mathcal{D}}\right)^{-1} \Phi_{\mathcal{D}}^\top \mathbf{y}$$

2.6.1 Examples of Basis Functions

- **Polynomial Basis**
  For 1-D data, $\Phi_j(x) = x^j$ for $1 \leq j \leq d$.
- **Radial Basis Function(RBF)**
  For d-dimensional data, $\Phi_j(\mathbf{x}) = e^{-\frac{\|x-\mu_j\|_2^2}{\sigma_j^2}}$ for $\mathbf{x}, \mu_j \in \mathbb{R}^d, \sigma_j \in \mathbb{R}$.
- **Fourier Basis**
- **Piecewise Linear Basis**
- **Periodic Basis** ($\sin(x)$, $\cos(x)$ **etc.**)

Data Splits (Preliminaries) : The original data in a machine learning model is typically taken and split into three sets.

- Training data (Training set): Dataset used to train the model and learn the parameters. (Below, n is the number of data points in the training set)

$$D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$$

$$\text{Training Error} : \frac{\Sigma_{i=1}^n (y_i - h_w(\mathbf{x}_i))^2}{n}$$

- Development Set (Validation Set) : Mainly used to tune *hyperparameters* of the model. Hyperparameters are not learnable parameters, and are instead predetermined quantities that are used with the model. For example, σ in an RBF basis function is a hyperparameter.
- Test Set (Evaluation Set) : This is the unseen/test set used for a final evaluation of the model, after choosing the best hyperparameters determined via tuning on the development set. The error on the test set is also referred to as the generalization error.

Errors on the development/test sets are a measure of the generalization ability of the trained machine learning model.

2.7 Hyperparameter tuning

Hyperparameter tuning involves two general methods :

- Grid Search: Define a search space as a grid of hyperparameter values and evaluate every position in the grid.
- Random Search: Define a search space as a bounded domain of hyperparameter values and randomly sample points in that domain.

2.7.1 Underfitting

Underfitting is a scenario where a data model is overly simple and unable to capture the relationship between the input and output variables accurately.

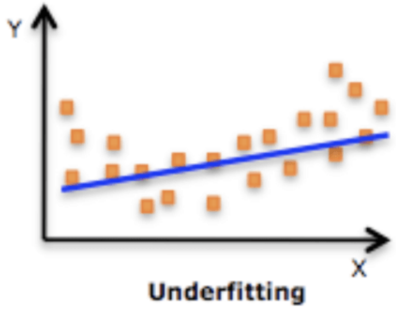
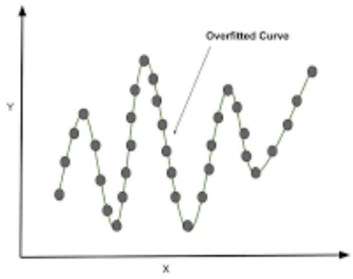
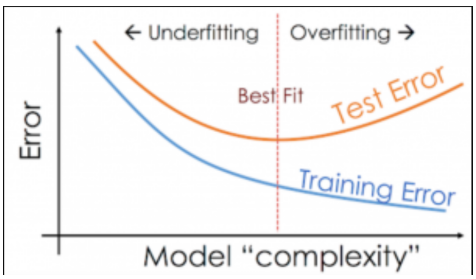


Figure 2.3: Model is too simple and underfits the data



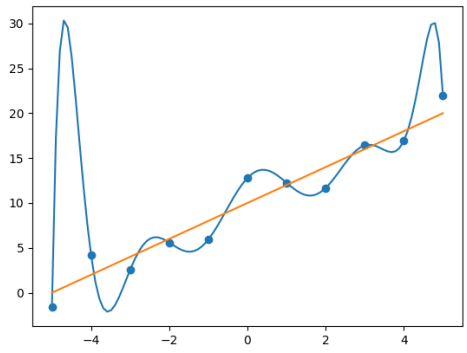
(a) Model is overly complex



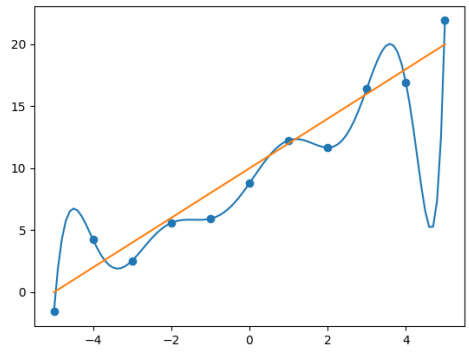
(b) Hyperparameter tuning

2.7.2 Overfitting

Overfitting is an undesirable machine learning behavior that occurs when the machine learning model gives accurate predictions for training data but not for new data (i.e., does not generalise well). Model fits the training data perfectly but is overly complex.
It is tempting to assume that having large number of parameters (making complex model) in our hypothesis class would fit the training data perfectly. But this would fail to predict new unseen data miserably. This is termed **overfitting**.



(a) Original



(b) Perturbation

Figure 2.5: Overfitting due to complex modelling

Consider the line y = 2x+10, let's say we sampled few points from this true line, and this process had some noise involved. In the above figure, we try to fit a polynomial regression model with degree 10. Our predictor function $h_{\mathbf{w}}(\mathbf{x})$ is given as:

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_{10} x^{10}$$

It is clearly evident from the image, that the training error is 0, the model perfectly fits the training data, but the test error is fairly high due to the huge variance of the predictor function. Additionally, on small perturbation(here, changing the the data point(0,13) to (0,8) in the second image), has vastly changed the shape of the curve fit, because of several degrees of freedom.

Ideally we would want to hit the sweet spot between a simple fit (Underfitting, suffering from high bias) and a complex fit (Overfitting, suffering from high variance), and thus we need to tune the hyperparameter $\mathbf{d}$.

2.8 Regularization

Instead of tuning, regularization modifies the loss function to explicitly constrain the model complexity. The general regularised optimisation equation looks like:

$$L_{reg}(\mathbf{w}, D_{train}) = L_{MSE}(\mathbf{w}, D_{train}) + \lambda R(\mathbf{w})$$

where the first term, $L_{MSE}(\mathbf{w}, D_{train})$ is the measure of fit to the training data set and the second term $\lambda R(\mathbf{w})$ is the regulariser term, with $\lambda \geq 0$. The $R(\mathbf{w})$ is a measure of the model's complexity. This can help alleviate

overfitting caused by the first term. It does this by penalising/shrinking weights in the weight vector making some of them equal to near zero, Thus even if the model is a high-degree polynomial, minimizing the L_{reg} yields many (near) zero weights, thereby shrinking the model complexity. Two examples of these shrinkage-based regularizations are discussed in the sections below.

Why this works? When the coefficients are very large and the model is highly complex, the errors on dev set or test set are large due to large variances in the predictor function. This can be alleviated by penalising the weight terms(weight norm), thereby making values of $\mathbf{w}$ small, and hence the errors are not every large!

2.8.1 Ridge Regression

Ridge regression (also called L2-Normalised Regression) is a regularization technique to combat overfitting. The penalty term $R(\mathbf{w})$ here is a Euclidean norm, given as $R(\mathbf{w}) = \|\mathbf{w}\|^2$. The optimisation objective function and the optimal weights for L2-regularized regression can be written as:

$$\mathbf{w}_{L_2} = \arg\min_{\mathbf{w}}(\|\mathbf{y} - \phi\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_2^2)$$

The above equation is equivalent to the following constrained optimization problem.

$$\mathbf{w}_{L_2} = \arg\min_{\mathbf{w}} \|\mathbf{y} - \phi\mathbf{w}\|_2^2 \text{ s.t } \|\mathbf{w}\|_2^2 \leq t^2$$

Note: Usually, we don't include w_0 in the penalty term $R(\mathbf{w})$. This goes well with the intuition that the intercept doesn't depend on the feature vectors and hence penalising w_0 would lead to a poor fit.

Infact, this optimisation problem has a closed form solution just as the non regularised objective function.

$$\nabla L_{ridge}(\mathbf{w}) = 0$$

$$-2\phi^T(\mathbf{y} - \phi\mathbf{w}) + 2\lambda\mathbf{w} = 0$$

$$\mathbf{w}_{ridge} = (\phi^T\phi + \lambda I)^{-1}\phi^T\mathbf{y}$$

Note: The best thing about the closed form solution is that such a closed form solution always exists, unlike the unregularized loss function solution, because here the term $\phi^T\phi + \lambda I$ is always invertible(provided the regularization parameter $\lambda > 0$).

A matrix $\mathbf{X} \in \mathbb{R}_{n\times n}$ is positive definite if for any $\mathbf{v} \neq 0$, $\mathbf{v}^T\mathbf{X}\mathbf{v} > 0$. And a positive definite matrix is always invertible. Since $\mathbf{v}^T\mathbf{X}\mathbf{v} > 0$ for all $\mathbf{v} \neq 0$, this implies $\mathbf{X}\mathbf{v} \neq 0$ for all $\mathbf{v} \neq 0$, which is the definition of an invertible matrix.

In our case, consider a non zero vector $\mathbf{v}$,

$$\mathbf{v}^T(\phi^T\phi + \lambda I)\mathbf{v} = \mathbf{v}^T\phi^T\phi\mathbf{v} + \lambda\mathbf{v}^T\mathbf{v} = (\phi\mathbf{v})^T\phi\mathbf{v} + \lambda\mathbf{v}^T\mathbf{v} = \|\phi\mathbf{v}\|_2^2 + \lambda\|\mathbf{v}\|_2^2$$

The above expression is strictly positive for positive values of λ.

2.8.2 Lasso Regression

Also known as L1-normalised regression, Lasso (Least Absolute Shrinkage & Selection Operation) Regression is a similar technique to Ridge regression, but instead of using Euclidean L2 norm for the penalty, we use the L1 norm. So $R(\mathbf{w}) = \|\mathbf{w}\|_1$, and the total loss function can be expressed as:

$$\mathbf{w}_{L_1} = \arg\min_{\mathbf{w}}(\|\mathbf{y} - \phi\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_1)$$

The equivalent constrained form for the above is:

$$\mathbf{w}_{L_1} = \arg\min_{\mathbf{w}}(\|\mathbf{y} - \phi\mathbf{w}\|_2^2) \text{ s.t } \|\mathbf{w}\|_1 \leq t$$

As the L1 norm isn't differentiable, there is no closed form solution for the above optimisation problem. Other methods which can be used to solve for the optimal weights:

- Quadratic Programming
- Iterative optimisation algorithms (e.g. Gradient Descent)

2.8.3 Comparison

L1 Regularization yields sparse weight vectors compared to L2 Regularization. This can be intuitively understood from an example.

Referring to the figure below diagram, assume the constrained version of the regularization methods, and the predictor with only two weights β_1 and β_2. Say, at $\hat{\beta}$, the MSE is minimised. So around that we draw the contours(ellipses) of the loss function, and the points at which a contour touches the boundary of enclosed area by the constraints (rhombus in Lasso, and circle in Ridge) gives the final optimal weights.

There will be a large number of cases, where the contour will touch the square at one of its corners (one weight resulting in zero) as compared to touching the circle on the axes. By extrapolation, we can see that in higher dimensions, the cases where multiple weights are zero will occur much more frequently in Lasso regression due to the sharp boundaries. So Lasso regression will prove to be useful in cases when there are outliers present in the training data and using Ridge regression could result in non-zero weights of undesired features present in the set of basis functions. Lasso can effectively ignore these features by setting their weights to zero. Ridge, with its more gradual constraints, might still assign non-zero weights to these features.

Note: It is not that always sparsity helps. Infact, for neural networks L2 regularization is most widely used (due to its smoothness). But for many benchmark tasks (feature selection tasks), L1 regularization is preferred and encourages models that utilize a smaller subset of features, which can be valuable for interpretability and reduces computational complexity.

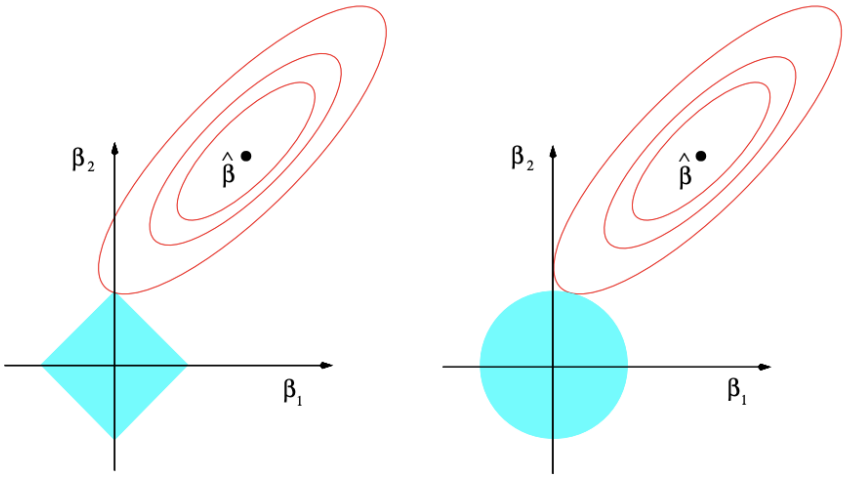


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

Figure 2.6: Comparison of L1 and L2 Regression methods. Figures reproduced from The Elements of Statistical Learning (Trevor Hastie, Robert Tibshirani and Jerome Friedman. Second Edition. 2009)

Chapter 3

Gradient Descent

3.1 Introduction

Gradient Descent is a **first-order iterative optimization algorithm** for finding local minimum points of a differentiable function. It's commonly used in machine learning and optimization problems (e.g., matrix factorization, neural networks), especially when dealing with non-convex cost functions that don't have a closed-form solution.

3.2 General Template of GD-Style Algorithms

Let the function being optimized be dependent on the weight vector $\mathbf{w}$. We now wish to find the optimal value of $\mathbf{w}$ so that the function in consideration is minimized (since its the loss function which we generally are trying to optimise in ML). GD-style algorithms helps us in finding that. The general flow of such algorithms is as follows:

- Initialize $\mathbf{w}$ (e.g. $\mathbf{w} = \vec{0}$)
- Repeat
  - Choose a descent direction (directions of fastest decrease)
  - Choose a step size
  - Update $\mathbf{w}$
- Exit repeat loop when certain stopping criterion is met. Some common stopping criterion include (where t represents a time step)
  - $||\nabla L(w_t)||_2 < \epsilon$
  - $||w_{t+1} - w_t||_2 < \epsilon$

3.3 Algorithm of Gradient Descent

The flow for the Gradient Descent algorithm keeping in the mind the template above is:

- $\mathbf{w} \leftarrow \mathbf{w}_0$: Initialisation can be done in various ways such as zero initialisation ($w_0 = \vec{0}$), randomly sampled Gaussian etc.
- For the iterative part, we use the following values :
  - Direction : $-\nabla L(\mathbf{w}_t)$
  - Step size (Learning Rate) : $\alpha > 0$ is a hyperparameter
  - Update : $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \alpha \nabla L(\mathbf{w}_t)$
- Some common stopping criteria used among others are:
  - $||\nabla L(w_t)||_2 < \epsilon$
  - $||w_{t+1} - w_t||_2 < \epsilon$

Algorithm 1: Gradient Descent Algorithm with Epochs

Input : Initial weight vector w_0, step size $\alpha > 0$, tolerance $\epsilon > 0$, maximum number of epochs N_{max}
Output: Optimal weight vector w^*

$w \leftarrow w_0$;
for $t \leftarrow 1$ **to** N_{max} **do**
    Compute gradient direction: $\nabla L(w)$;
    Update: $w \leftarrow w - \alpha \nabla L(w)$;
    if $\|\nabla L(w)\|_2^2 \leq \epsilon$ **then**
        break;
return $w^* = w$

3.3.1 Why the name Gradient Descent ?

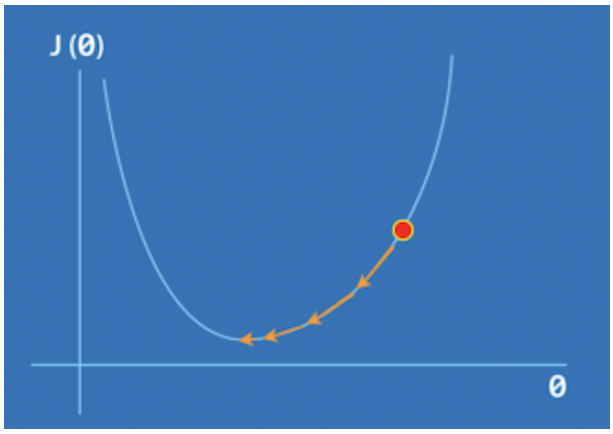
The name Gradient Descent is made of two parts:

- **Gradient** : Gives us the direction of fastest increase in function L

$$\nabla L(w) = \begin{bmatrix} \frac{\partial L(w)}{\partial w_1} \\ \frac{\partial L(w)}{\partial w_2} \\ \vdots \\ \frac{\partial L(w)}{\partial w_d} \end{bmatrix}$$

- **Descent** : Since we update in the direction of fastest decrease in L

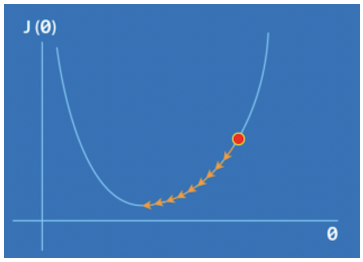
3.4 GD for the 1-D case and Linear Regression



The overall representation of Gradient Descent in linear regression would take on a similar form as given in the plot above, and given that the step size governs the speed and feasibility of convergence, it leads us to inquire about the following matters:

3.4.1 What if the step size is too small?

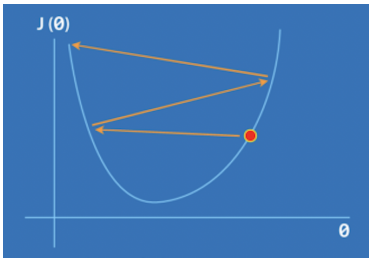
The plot obtained with very small step size would look something like the plot given below



From the plot, we can conclude that the loss will take very long to converge since we only update by a small value each time. This may also result in never reaching the optimal point if we train using a (fixed) maximum number of epochs.

3.4.2 What if the step size is very large?

The plot obtained with very large step size would look something like the plot given below



Clearly in this case the curve will take longer to converge or in the worst case may diverge as seen above.

3.5 Weight update rule in GD for linear regression

The formula for update of $\mathbf{w}$ is,

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla L(\mathbf{w}) \tag{3.1}$$

Unregularized Loss:

$$L(\mathbf{w}) = \frac{1}{N} \sum_{i}^{N} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 \tag{3.2}$$

$$L(\mathbf{w}) = \frac{1}{N} \sum_{i}^{N} (y_i - \hat{y_i})^2 \tag{3.3}$$

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = -\frac{2}{N} \sum_{i}^{N} (y_i - \hat{y_i}) \mathbf{x}_i = \frac{2}{N} \sum_{i}^{N} (\hat{y_i} - y_i) \mathbf{x}_i \tag{3.4}$$

$$\boxed{\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \alpha \frac{2}{N} \sum_{i}^{N} ([\mathbf{w}^t]^T \mathbf{x}_i - y_i) \mathbf{x}_i}$$

3.6 Different Variants of Gradient Descent

3.6.1 (Full) Gradient Descent

Here, we find the gradient of loss function over the entire training dataset. Our gradient update formula in this case is :

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla L(\mathbf{w}, \mathcal{D}_{train})$$

The training data can be very large in many applications of ML with millions of data points. Hence, it seems wasteful to compute the full loss function over the entire training set in order to perform only a single parameter update. Hence, we usually use other variations of GD which helps us in avoiding this computationally expensive step of finding gradient of loss over entire data every time.

3.6.2 Stochastic Gradient Descent(SGD)

In this modification of GD, rather than finding gradient of loss over entire training set, we find loss over only single instance of training data that is picked randomly and we update $\mathbf{w}$ based on the gradient for this loss,

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla L(\mathbf{w}, \mathcal{D}_{random})$$

where $\mathcal{D}_{random} = \{(x_i, y_i)\}$, $(x_i, y_i) \rightarrow$ randomly sampled point in $\mathcal{D}_{train}$
Now the issue with this version is that there is a lot of noise in the convergence path of the loss function, as a single data point is not representative of entire data set and this leads to lot of fluctuations in computed gradient. Outliers can lead to training instability in this case. For example, training may stop prematurely, if for a certain point gradient becomes zero.

3.6.3 Mini Batch Gradient Descent

In this version of GD, we try to find the best of both the worlds of GD and SGD by using the gradient of loss computed over only a batch (small subset) of original data set. This helps not only in faster gradient calculation but also is less noisier as batch is still a better representative than a single instance used in SGD.

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla L(\mathbf{w}, \mathcal{D}_{batch})$$

where $\mathcal{D}_{batch} = \{(x_i, y_i)\}_{i=1}^{\mathbb{B}}$, $\mathbb{B}$ =batch size
Here $\mathbb{B}$ is a hyper parameter. We should know that larger $\mathbb{B}$ is more stable as we approach close to GD and smaller $\mathbb{B}$ is less stable as we move closer to SGD, so the onus lies on us to find that optimal $\mathbb{B}$ that can help us gain the benefit of both the worlds. Also, note that it is a common convention to use $\mathbb{B} = 16, 32, 64...$ usually a power of 2. This method is an improvement over SGD, as it enhances training stability. Batches are randomly sampled during each epoch.

3.7 Probabilistic View of Linear Regression

For training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$. Let the target y_i have some noise defined as:

$$y_i = f(x_i) + \epsilon_i, \qquad \epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

$\epsilon_{i's}$ are independent and identically distributed (**i.i.d.**), 0 mean Gaussians with the some variance σ^2.($\text{Cov}(\epsilon_i, \epsilon_j) = 0$ for all $j \neq i$). So basically now we can get the following interpretation of data,

$$\begin{aligned}
& y_i = f(x_i) + \epsilon_i \\
\implies & y_i = \mathbf{w}^T x_i + \epsilon_i \\
\implies & y_i \sim \mathcal{N}(\mathbf{w}^T x_i, \sigma^2) \\
\implies & P(y_i \mid x_i, \mathbf{w}) \sim \mathcal{N}(\mathbf{w}^T x_i \sigma^2), \qquad \text{[where } P(y_i \mid x_i, \mathbf{w}) \text{ is likelihood function]}
\end{aligned}$$

For training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$,

$$P(y_1, y_2, \ldots, y_n \mid x_1, x_2, \ldots, x_n, \mathbf{w}) = \prod_i P(y_i \mid x_i, \mathbf{w})$$

where y_i's are conditionally independent given x_i's.

$$\implies \log P(y_1, y_2, \ldots, y_n \mid x_1, x_2, \ldots, x_n, \mathbf{w}) = \sum_i \log P(y_i \mid x_i, \mathbf{W})$$

Note, here we can use log likelihood instead of likelihood because of following reasons,

- Does not change the maxima or minima of function as log is monotonic function
- Mathematical convenience
- Computational convenience as it is easier to add many small values than to multiply them as they might underflow if multiplied

Chapter 4

Probabilistic Model of Linear Regression

4.1 Introduction

The probabilistic model of linear regression provides a way to understand and interpret linear regression models from a probabilistic view.

Consider training data $\mathcal{D}$:

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$$

Assumption: The relation between the input features $\mathbf{x}$ and the target y can be represented by a linear equation with some added noise ϵ:

$$y_i = \mathbf{w}^T \mathbf{x}_i + \epsilon_i$$

Noise: The noise component ϵ signifies the fluctuation within the data that contributes towards uncertainty in the target values and cannot be explained by the linear relationship. Assume that it follows a Gaussian distribution with mean 0 and variance σ^2

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

In other words, ϵ_i's are independent and identically distributed (i.i.d.) Gaussian random variables with zero-mean and the same variance σ^2. Now, we can represent target y_i as a Gaussian distribution with mean $\mathbf{w}^T \mathbf{x}_i$ and variance σ^2

$$y_i \sim \mathcal{N}(\mathbf{w}^T \mathbf{x}_i, \sigma^2)$$

Given training data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ where feature vector $\mathbf{x}_i \in \mathbb{R}^d$, target $y_i \in \mathbb{R}$,

$$P(y_i \mid \mathbf{x}_i, \mathbf{w}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_i - \mathbf{w}^T \mathbf{x}_i)^2}{2\sigma^2}}$$

$$\text{Likelihood :} \quad P(y_1, y_2, \ldots, y_n \mid \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) = \pi_i P(y_i \mid \mathbf{x}_i, \mathbf{w})$$

$$\text{Log Likelihood :} \quad \log P(y_1, y_2, \ldots, y_n \mid \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) = \sum_i \log P(y_i \mid \mathbf{x}_i, \mathbf{w})$$

The likelihood function models how well the observed data fits the assumed model. In case of linear regression, it measures the probability of observing the actual target values given the parameter vector w and feature vectors.

4.2 Maximum Likelihood Estimation

The process of finding the best-fitting model often involves estimating the model's parameters in a way that maximizes the likelihood of observing the actual data. This estimation process is known as Maximum Likelihood Estimation (MLE). It is a *Frequentist* view in machine learning.

For the model, y_i(target value) represented as a Gaussian distribution $\mathcal{N}(\mathbf{w}^T \mathbf{x}_i, \sigma^2)$, the parameter of interest is the weight vector $\mathbf{w}$. We need to find $\mathbf{w}$ that maximizes the likelihood function. Maximizing likelihood function is equivalent to maximizing log likelihood function. (The log function is strictly increasing, and thus maximizing the log likelihood gives the same solution as maximizing the likelihood function.)

$$\mathbf{w}_{\text{MLE}} = \arg\max_{\mathbf{w}} \sum_i \log P(y_i \mid \mathbf{x}_i, \mathbf{w})$$

4.2.1 Motivating Example

You want to estimate the probability of a biased coin landing on heads. The probability that the outcome is head for a coin toss is θ. Let us say your data contains N coin tosses with N_H heads and N_T tails. What is the maximum likelihood estimate of θ ?

Likelihood function: probability of data, given parameter θ

$$\begin{aligned} P(D \mid \theta) &= \theta_H^N (1-\theta)_T^N \\ \implies \theta_{MLE} &= \arg\max_\theta \log P(\mathcal{D} \mid \theta) \\ &= \arg\max_\theta \log \left(\theta^{N_H}(1-\theta)^{N_T}\right) \\ &= \arg\max_\theta \left(N_H \log\theta + N_T \log(1-\theta)\right) \end{aligned}$$

To find θ_{MLE}, we need to find the derivative of $N_H \log\theta + N_T \log(1-\theta)$ w.r.t. θ, set to zero and solve for θ:

$$\begin{aligned} \frac{N_H}{\theta_{MLE}} - \frac{N_T}{1-\theta_{MLE}} &= 0 \\ \implies \theta_{MLE} = \frac{N_H}{N_H + N_T} &= \frac{N_H}{N} \end{aligned}$$

4.2.2 Question

Say you have a coin with P $\in$ {0.4,0.6}
Data: 3 coin tosses; 2 heads, 1 tail
What is MLE of P ?

Solution:

$$\begin{aligned} P(D \mid \theta)_{\theta=0.4} &= (0.4)^2(0.6) \\ P(D \mid \theta)_{\theta=0.6} &= (0.6)^2(0.4) \\ P(D \mid \theta)_{\theta=0.6} &> P(D \mid \theta)_{\theta=0.4} \\ \textbf{Answer} &= 0.6 \end{aligned}$$

4.2.3 MLE for linear regression

$$\begin{aligned} w_{MLE} &= \arg\max \sum_i \log P(y_i \mid \mathbf{x}_i, \mathbf{w}) \\ &= \arg\max \sum_i \log\left(\mathcal{N}(\mathbf{w}^T\mathbf{x}_i, \sigma^2)\right) \\ &= \arg\max \sum_i \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_i - \mathbf{w}^T\mathbf{x}_i)^2}{2\sigma^2}} \\ &= \arg\max C - \sum_i \frac{(y_i - \mathbf{w}^T\mathbf{x}_i)^2}{2\sigma^2} \\ w_{MLE} &= \arg\min \sum_i (y_i - \mathbf{w}^T\mathbf{x}_i)^2 \end{aligned}$$

- MLE under Gaussian noise distribution is equivalent to the least squares solution.
- MLE under Laplacian noise distribution is equivalent to Least Absolute Deviation solution.
- A significant drawback of Maximum Likelihood Estimation (MLE) arises when working with limited data, as it can lead to overfitting due to its heavy reliance on the available dataset.

4.3 Bayesian Parameter Estimation

In MLE, observations are random variables while parameters are not. On the contrary, in the Bayesian framework, parameters are also treated as random variables alongside observations. Parameters have an underlying

prior distribution which encode beliefs prior to observing the data. As MLE has a high tendency to overfit, we switch to a different estimation, namely the **Maximum Aposteriori Estimation** with the hope of alleviating overfitting using prior information.

4.4 Maximum Aposteriori Estimate (MAP)

Maximum A Posteriori (MAP) estimation focuses on incorporation of prior knowledge into estimating the underlying model parameters which are denoted by Θ. We assume a prior distribution $\mathcal{P}(\Theta)$ for parameters Θ (before observing the data $\mathcal{D}$).

By Bayes' theorem, we have :

$$\mathcal{P}(\Theta \mid \mathcal{D}) = \frac{\mathcal{P}(\mathcal{D} \mid \Theta)\mathcal{P}(\Theta)}{\mathcal{P}(\mathcal{D})}$$

$\boldsymbol{\mathcal{P}(\Theta \mid \mathcal{D})}$ is the **posterior** probability for the model parameters Θ given data $\mathcal{D}$.
$\boldsymbol{\mathcal{P}(\mathcal{D} \mid \Theta)}$ is the **likelihood** of observing the data given model parameters & $\boldsymbol{\mathcal{P}(\Theta)}$ is the **prior**.

$$\begin{aligned}\Theta_{\text{MAP}} &= \arg\max_{\Theta} \mathcal{P}(\Theta \mid \mathcal{D}) \\ &= \arg\max_{\Theta} (\log \mathcal{P}(\mathcal{D} \mid \Theta) + \log \mathcal{P}(\Theta))\end{aligned}$$

Note: The posterior is directly proportional to the product of likelihood and the prior functions as shown by the following equation,

$$\begin{aligned}\mathcal{P}(\Theta \mid \mathcal{D}) &\propto \mathcal{P}(\mathcal{D} \mid \Theta)\mathcal{P}(\Theta) \\ &\propto \log \mathcal{P}(\mathcal{D} \mid \Theta) + \log \mathcal{P}(\Theta)\end{aligned}$$

4.4.1 Coin example (MAP estimate)

Suppose, we have a biased coin and N_H and N_T are the number of heads and tails obtained.
Likelihood $(\mathcal{PD} \mid \Theta)$, i.e, the probability of data given the parameter Θ is given by the following equation,

$$\mathcal{P}(\mathcal{D} \mid \Theta) = \Theta^{N_H}(1-\Theta)^{N_T}$$

What is a good prior on Θ ?
Beta distribution is a good candidate for prior because the functional form of prior matches with that of the likelihood.

$$\mathcal{B}(\Theta, \alpha, \beta) = c\,\Theta^{\alpha-1}(1-\Theta)^{\beta-1} \qquad [c = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)},\ \Gamma(.) \text{ is the gamma function}]$$

Hence, the posterior will be

$$\begin{aligned}\mathcal{P}(\Theta, \mathcal{D}) &\propto \mathcal{P}(\mathcal{D} \mid \Theta)\mathcal{P}(\Theta) \\ &\propto c\,\Theta^{N_H+\alpha-1}(1-\Theta)^{N_T+\beta-1} \\ &\propto \mathcal{B}(\Theta, N_H+\alpha, N_T+\beta)\end{aligned}$$

4.5 Conjugate Priors

For a likelihood $P(\mathcal{D} \mid \Theta)$ coming from a family of distributions $\mathtt{d}_1$, a prior (from a family $\mathtt{d}_2$) is said to be a conjugate prior if the posterior distribution also comes from the family $\mathtt{d}_2$. ($\mathtt{d}_2$ could also be from the same family as $\mathtt{d}_1$).

4.5.1 Coin example

Continuing with the coin example mentioned above, the probability of getting N_H heads and N_T tails upon tossing a biased coin with the probability of getting head as Θ (Bernoulli distribution), is given by,

$$\mathcal{B}(N_H, N_T, \Theta) = \Theta^{N_H}(1-\Theta)^{N_T}$$

Also, the PDF for beta distributions is given as,

$$\mathcal{B}(\Theta, \alpha, \beta) = c.\Theta^{\alpha-1}(1-\Theta)^{\beta-1}$$

The posterior is proportional to the product of likelihood and prior and putting their values we have,

$$\begin{aligned} P(\Theta \mid \mathcal{D}) &\propto P(\mathcal{D} \mid \Theta) P(\Theta) \\ &\propto \Theta^{N_H+\alpha-1}(1-\Theta)^{N_T+\beta-1} \\ &\propto \mathcal{B}(\Theta, N_H+\alpha, N_T+\beta) \end{aligned}$$

Hence,

$$\Theta_{MAP} = \arg\max((N_H+\alpha-1)\log(\Theta) + (N_T+\beta-1)\log(1-\Theta))$$

On maximizing it, we get

$$\begin{aligned} &\frac{N_H+\alpha-1}{\Theta} - \frac{N_T+\beta-1}{1-\Theta} = 0 \\ \implies &\Theta_{MAP} = \frac{N_H+\alpha-1}{N_H+N_T+\alpha+\beta-2} \end{aligned}$$

Compared to the MLE estimate $\Theta_{MLE} = \frac{N_H}{N}$, MAP estimate has extra counts denoted by α and β.

We can see that as $N \to \infty \Theta_{MAP}$ converges to Θ_{MLE}. This represents the fact that prior beliefs become less representative over more data. This is encapsulated in the Bernstein - von Mises Theorem. Further, we also see that in essence, the prior data is enforcing a belief of a "pre-existing" $\alpha-1$ heads out of $\alpha+\beta-2$ tosses. For this reason, these numbers can also be thought of as pseudo coin counts.

Additional Note: There are a large number of exponential families that serve as conjugate priors for different distributions due to their exponential mathematical form.

1. Beta distribution serves as conjugate prior for Bernoulli & Binomial likelihoods.
2. Normal distribution is a conjugate prior for itself, i.e. a Gaussian distribution is a conjugate prior for other Gaussian likelihoods, however, inverse gamma distribution also is a conjugate prior for Gaussian likelihood.
3. Dirichlet prior is used as conjugate for Multinomial distributions.

4.5.2 Linear Regression MAP Estimate

Given the probabilistic setup of Linear Regression, we try to estimate the weights $\mathbf{w}$.
Let's consider zero mean Gaussian Prior on the weights $\mathbf{w}$ with covariance matrix as scaled identity matrix, i.e.

$$\begin{aligned} \mathcal{P}(\mathbf{w}) &= \mathcal{N}\left(0, \frac{\mathbf{I}}{\lambda}\right) \\ &= \frac{1}{(2\pi)^{d/2}\sqrt{\det(\mathbf{I}/\lambda)}} \exp\left\{\left(-\frac{1}{2}\mathbf{w}^T\left(\frac{\mathbf{I}}{\lambda}\right)^{-1}\mathbf{w}\right)\right\} \\ &= \left(\frac{\lambda}{2\pi}\right)^{d/2} \exp\left\{\left(-\frac{\lambda}{2}\mathbf{w}^T\mathbf{w}\right)\right\} \\ &= \left(\frac{\lambda}{2\pi}\right)^{d/2} \exp\left\{\left(-\frac{\lambda}{2}\|\mathbf{w}\|_2^2\right)\right\} \end{aligned}$$

Now, if we try to find the MAP estimate for $\mathbf{w}$,

$$\begin{aligned} \mathbf{w}_{MAP} &= \arg\max_{\mathbf{w}} \left(\mathcal{P}(\mathcal{D} \mid \mathbf{w})\ \mathcal{P}(\mathbf{w})\right) \\ &= \arg\max_{\mathbf{w}} \Big(\log\left(\mathcal{P}(\mathcal{D} \mid \mathbf{w})\right) + \log\left(\mathcal{P}(\mathbf{w})\right)\Big) \end{aligned}$$

Using the MLE equation to rewrite the log-likelihood term and plugging in the prior term gives us,

$$\begin{aligned} \mathbf{w}_{MAP} &= \arg\max_{\mathbf{w}} \left(\frac{-1}{2\sigma^2}\sum_{i=1}^{n}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2 + \frac{d}{2}\log\left(\frac{\lambda}{2\pi}\right) - \frac{\lambda}{2}\|\mathbf{w}\|_2^2\right) \\ &= \arg\min_{\mathbf{w}} \left(\frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2\right) \end{aligned}$$

Recall that this is same as the L_2-regularised linear regression (or Ridge Regression).

NOTE: By changing the prior, we can get various types of regularisations. For example, using a Laplace Prior will give L_1 Regularisation (or Lasso Regression)

4.6 Bias and Variance of Estimates

Natural question that arises after creating different types of Linear Regression models is that "How do we evaluate whether the predictor is good or not?".

One of the metric that we use to generate the model is Training Loss, but this does not tell whether the model will be able to generalize or not. Hence, to define whether a model is good or not, we look at the Test Loss.

So, we try to decompose the Expected Test Loss and it turns out that it has 3 components

1. Variance
2. Bias
3. Noise (or Irrecoverable error/ Unavoidable error)

Out of these, Variance and Bias are inherent to the model and Noise is inherent to the data (due to inaccuracies in measurements).

4.6.1 Bias

Let us assume that we are sampling data from a true distribution

$$y = f(\mathbf{x}) + \epsilon$$

where ϵ is a zero mean Gaussian Noise with standard deviation σ. Let $h_{\mathbf{w}}(.)$ be the predictor for a given data $\mathcal{D}$. Then the bias of the given model for a particular $\mathbf{x}$ can be defined as follows

$$\text{Bias} = \mathbb{E}\left[h_{\mathbf{w}}(\mathbf{x})\right] - f(\mathbf{x})$$

Note that the expectation is over various choice of training data set $\mathcal{D}$ which will give rise to various $h_{\mathbf{w}}(.)$. Also, $\mathbf{x}$ is fixed value and the expectation does not depend on it. To explain the meaning of bias, let us sample multiple different data sets from the given true distribution and find the predictor function for each of them. Then the bias represents how close the average of these predictor functions is to the true distribution.

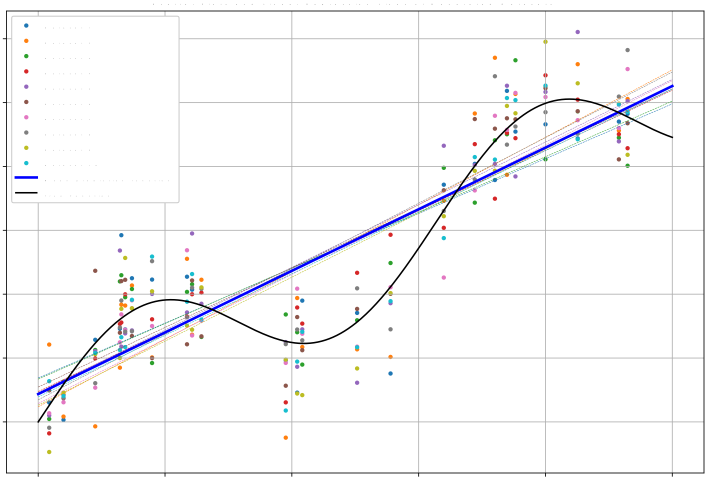


Figure 4.1: Showing various regression arising from various data-sets and the mean of them

4.6.2 Variance

Considering the same setup as before, the variance of the given model for a particular $\mathbf{x}$ can be defined as follows

$$\text{Variance} = \mathbb{E}\left[\left(h_{\mathbf{w}}(\mathbf{x}) - \mathbb{E}\left[h_{\mathbf{w}}(\mathbf{x})\right]\right)^2\right] \tag{4.1}$$

Note that the expectation is over various choices of training data set $\mathcal{D}$ which will give rise to various $h_{\mathbf{w}}(.)$. Intuitively, variance of given model is basically the variance of different predictor functions that we get by varying the data set $\mathcal{D}$. Qualitatively, it measures the spread of predictor functions in figure 4.1.

4.6.3 Noise

Noise is the unavoidable error in the data caused by the errors in measurement. The expression for this error is as follows

$$\text{Noise} = \mathbb{E}\left[\left(y - f(\mathbf{x})\right)^2\right] \tag{4.2}$$

Note that this expectation is over the random noise (i.e. ϵ in this case) and not dependent on the data set. This reduces to

$$\begin{aligned}\text{Noise} &= \mathbb{E}\left[(\epsilon)^2\right] \\ &= \text{Var}(\epsilon) \\ &= \sigma^2\end{aligned}$$

4.6.4 Analysing Unregularized Linear Regression

Let us assume that we have the same model as before, i.e.

$$y = f(\mathbf{x}) + \epsilon \quad \text{and} \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

As mentioned before, we are trying to find the expected test error. Let us denote the test data point as $(\widetilde{\mathbf{x}}, \widetilde{y})$ i.e. $\widetilde{y} = f(\widetilde{\mathbf{x}}) + \widetilde{\epsilon}$. Then we have to find

$$E_{\text{test error}} = \mathbb{E}_{\widetilde{\epsilon},\mathcal{D}}\left[\left(\widetilde{y} - h_{\mathbf{w}}(\widetilde{\mathbf{x}})\right)^2\right]$$

NOTE: Here, we are assuming that $\tilde{x}$ is a constant while taking the expectation, i.e. LHS should ideally be parameterized as $E_{\text{test error}}(\tilde{x})$. This would also mean that $\tilde{y}$ (which is a random variable) is only dependent on $\widetilde{\epsilon}$.

$$\begin{aligned}E_{\text{test error}} &= \mathbb{E}\left[\left(\widetilde{y} - h(\widetilde{\mathbf{x}})\right)^2\right] \\ &= \mathbb{E}\left[\tilde{y}^2\right] + \mathbb{E}\left[h(\widetilde{\mathbf{x}})^2\right] - 2\,\mathbb{E}[\tilde{y}]\;\mathbb{E}[h(\widetilde{\mathbf{x}})] \\ &= \mathbb{E}\left[\left(\tilde{y} - \mathbb{E}[\tilde{y}]\right)^2\right] + \mathbb{E}[\tilde{y}]^2 + \mathbb{E}\left[\left(h(\widetilde{\mathbf{x}}) - \mathbb{E}[h(\widetilde{\mathbf{x}})]\right)^2\right] + \mathbb{E}[h(\widetilde{\mathbf{x}})]^2 - 2\,\mathbb{E}[\tilde{y}]\;\mathbb{E}[h(\widetilde{\mathbf{x}})] \\ &= \mathbb{E}\left[\left(\tilde{y} - f(\tilde{x})\right)^2\right] + \mathbb{E}\left[\left(h(\widetilde{\mathbf{x}}) - \mathbb{E}[h(\widetilde{\mathbf{x}})]\right)^2\right] + \mathbb{E}[h(\widetilde{\mathbf{x}})]^2 + \mathbb{E}[\tilde{y}]^2 - 2\,\mathbb{E}[\tilde{y}]\;\mathbb{E}[h(\widetilde{\mathbf{x}})] \\ &= \mathbb{E}\left[\left(\tilde{y} - f(\tilde{x})\right)^2\right] + \mathbb{E}\left[\left(h(\widetilde{\mathbf{x}}) - \mathbb{E}[h(\widetilde{\mathbf{x}})]\right)^2\right] + \left(\mathbb{E}[h(\widetilde{\mathbf{x}})] - \mathbb{E}[\tilde{y}]\right)^2 \\ &= \mathbb{E}\left[\left(\tilde{y} - f(\tilde{x})\right)^2\right] + \mathbb{E}\left[\left(h(\widetilde{\mathbf{x}}) - \mathbb{E}[h(\widetilde{\mathbf{x}})]\right)^2\right] + \left(\mathbb{E}[h(\widetilde{\mathbf{x}})] - f(\tilde{x})\right)^2 \\ &= \text{Noise} + \text{Variance} + \text{Bias}^2\end{aligned}$$

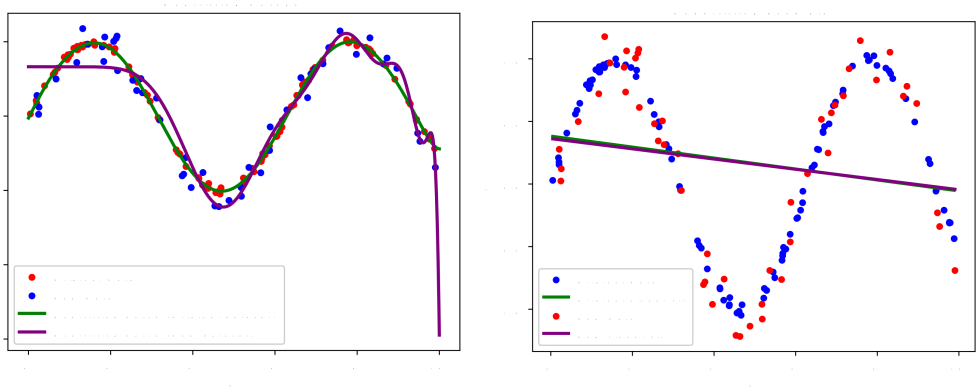
Explanations of steps:

- The predicted value $h(\tilde{x})$ depends only on the data set and hence it is independent of the actual value $\tilde{y}$ (which is only dependent on $\widetilde{\epsilon}$)
- Using the definition of variance we can get $\mathbb{E}\left[Y^2\right] = \mathbb{E}\left[\left(Y - \mathbb{E}[Y]\right)^2\right] + \mathbb{E}[Y]^2$
- Also, note that $\mathbb{E}[\tilde{y}] = \mathbb{E}[f(\tilde{x}) + \widetilde{\epsilon}] = \mathbb{E}[f(\tilde{x})] + \mathbb{E}[\widetilde{\epsilon}] = f(\tilde{x})$ as the noise is zero mean and $\tilde{x}$ is a constant (as mentioned before)

4.6.5 Variance-Bias Trade-Off

Let us look at the use of the above-mentioned formula.

Consider the following case where we have used a very high complexity model for linear regression.



(a) Over-fitting - High Variance, Low Bias (b) Under-fitting - Low Variance, High Bias

Figure 4.2: Analysing models using bias and variance

It is evident that the bias of this high-complexity model is very low. But, as we change the dataset slightly, we can see that there is too much variation in the newly fitted curve. This means that this high-complexity model also has high variance. Similarly, we can consider a low-complexity model such as linear. Here, the predicted values deviate significantly from the actual function, and hence the bias is high. But, changing the dataset has very less effect on the predictor function, suggesting that there is low variance.

The ideal model is one that strikes a balance between bias and variance. Highly complex models, like high-order polynomials, may overfit, resulting in **low bias but high variance**. In contrast, lower complexity models might not capture all data complexities, leading to **high bias and low variance**. Both can yield high test errors.

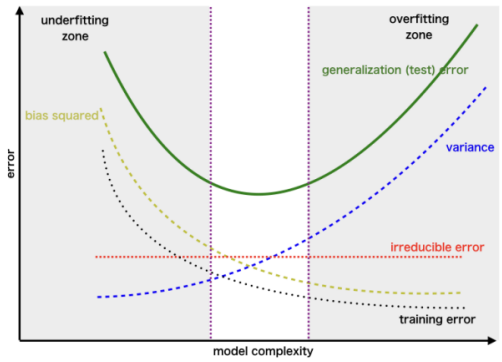


Figure 4.3: Variance Bias Trade-off with Expected Test Error/Generalisation Error

Hence, there is an inherent trade-off between variance and bias. We can't make both of them arbitrarily small at the same time.

Recall L_2-Regularised model where, $\mathbf{w}_{\text{ridge}} = \arg\min_{\mathbf{w}} \left(\|y - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 \right)$. As λ increases, the loss penalises high variation of $\mathbf{w}$ and hence the variance decreases. However, this is also reducing the flexibility of model and hence leading to increased bias.

Chapter 5

Logistic Regression

5.1 Introduction

Logistic regression involves a classification task, we instead of predicting a real value, we try to predict the label class for each input.
A natural way to re-purpose linear regression for classification tasks is to predict the class label as follows:

$$\hat{y} = \begin{cases} 1 \text{ if } \mathbf{w}^T\mathbf{x} \geq \tau \\ 0 \text{ otherwise} \end{cases}$$

where τ acts as the threshold, and is a hyper-parameter for the model.
However, this approach has certain limitations:

- It is not easy to pick a good value for τ.
- It is difficult to calibrate the prediction quality, that is estimating the confidence the model has for a certain prediction.

To overcome these issues, we modify the range of score that the model assigns from $(-\infty, \infty)$ to $(0, 1)$ which can then be interpreted as the "probability" of the input $\mathbf{x}$ being in class $\hat{y} = 1$.

5.2 Logistic Regression

5.2.1 Sigmoid Function

The first task is to collapse the score from (∞, ∞) to $(0, 1)$. To do so, we use the **Sigmoid Function**, which is as follows:

$$\boxed{\sigma(x) = \frac{1}{1 + \exp(-x)}}$$

Observe that sigmoid is a monotonically increasing function with the range being $(0, 1)$. Another property which makes it useful for our task is the form it's derivative takes:

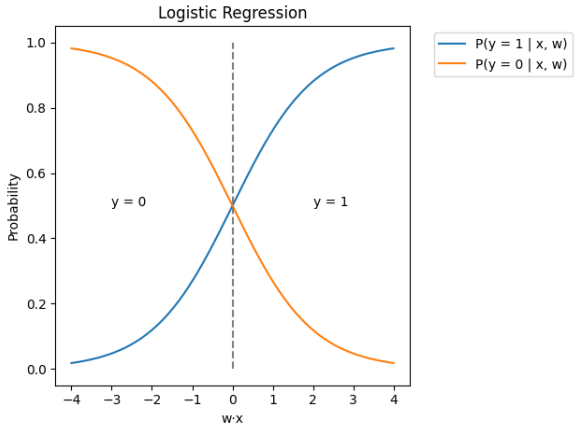
$$\begin{aligned} \sigma'(x) &= \frac{-1}{(1 + e^{-x})^2} \cdot (-e^{-x}) \\ &= \frac{1}{1 + e^{-x}} \cdot \frac{e^{-x}}{1 + e^{-x}} \\ &= \sigma(x) \cdot (1 - \sigma(x)) \end{aligned}$$

The derivative of the sigmoid function can hence be written in terms of the sigmoid function itself, a property we will use later.

5.2.2 Model

The model used in logistic regression outputs the probability of the input vector $\mathbf{x}$ having class label $y = 1$ as follows:

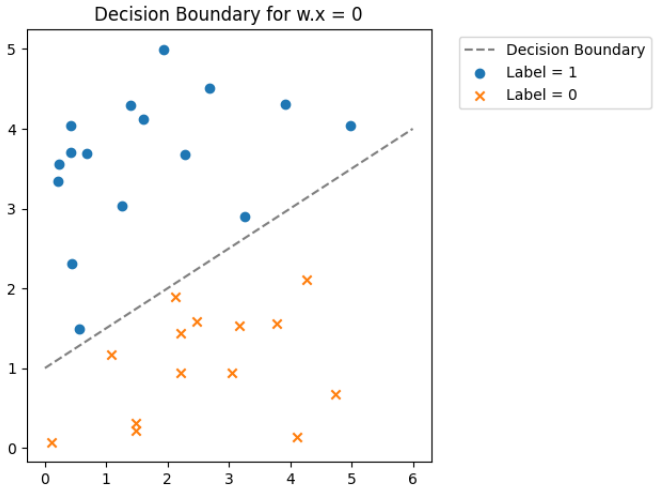
$$P(y = 1 \mid \mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x})$$
$$P(y = 0 \mid \mathbf{x}, \mathbf{w}) = 1 - P(y = 1 \mid \mathbf{x}, \mathbf{w}) = 1 - \sigma(\mathbf{w}^T\mathbf{x})$$



We then predict the label $\hat{y}$ as the label which has higher probability. Hence $\hat{y} = 1$ if:

$$\frac{P(y = 1 \mid \mathbf{x}, \mathbf{w})}{P(y = 0 \mid \mathbf{x}, \mathbf{w})} > 1$$
$$\implies \frac{\sigma(\mathbf{w}^T\mathbf{x})}{1 - \sigma(\mathbf{w}^T\mathbf{x})} > 1$$
$$\implies \exp(\mathbf{w}^T\mathbf{x}) > 1$$
$$\implies \mathbf{w}^T\mathbf{x} > 0$$

Hence, the **hyperplane** $\mathbf{w}^T\mathbf{x} = 0$ acts as a **decision boundary** in the sense that all input vectors $\mathbf{x}$ lying "above" the boundary (that is $\mathbf{w}^T\mathbf{x} > 0$) are given label 1 and rest are given label 0.

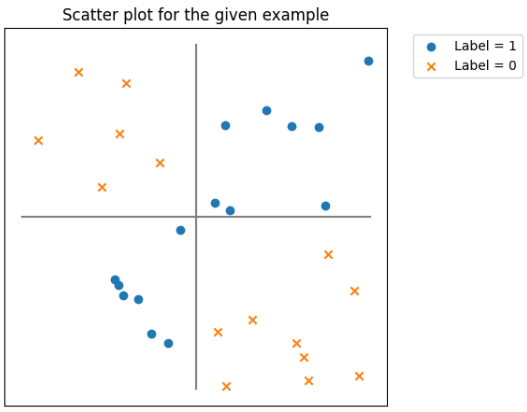


Observe that logistic regression yields a linear decision boundary. Hence it can perfectly classify the training points for **linearly separable data**.

Linearly Separable Data: A data-set $\mathcal{D}$ is linearly separable if $\exists \mathbf{w}$ such that $\forall$ positive training points $(y = 1)$, $\mathbf{w}^T\mathbf{x} > 0$ and $\forall$ negative training points $(y = 0)$, $\mathbf{w}^T\mathbf{x} < 0$.

This puts a severe restriction on the data-sets which can be classified using logistic regression. Consider a simple example, where $\mathbf{x} = [x_1, x_2]^T \in \mathbb{R}^2$:

$$y = \begin{cases} 1 & \text{if} \quad x_1 \cdot x_2 > 0 \\ 0 & \text{otherwise} \end{cases}$$



Observe that a Logistic Regression Classifier will not be able to find a good linear decision boundary for the above data. However, adding a feature to get $\phi(\mathbf{x}) = [1, x_1, x_2, x_1x_2]^T$ makes the data linearly separable and enables the model to find a perfect weight vector $\mathbf{w} = [0, 0, 0, 1]^T$ to classify the data.
Hence, to get better results, we may have to add new features to the input vector before training the model.

5.2.3 Parameter Estimation

Since the output of model is a probability distribution on the labels, it is natural to estimate the parameters $\mathbf{w}$ as the **Maximum Likelihood Estimate** for the observed data (training data-set). Hence,

$$\begin{aligned} \mathbf{w}^* &= \underset{\mathbf{w}}{\text{argmax}} \prod_{i=1}^{|\mathcal{D}_{\text{train}}|} P(y_i|\mathbf{x}_i, \mathbf{w}) \\ &= \underset{\mathbf{w}}{\text{argmax}} \sum_{i=1}^{|\mathcal{D}_{\text{train}}|} \log P(y_i|\mathbf{x}_i, \mathbf{w}) \quad \text{(Maximum Conditional Likelihood)} \\ &= \underset{\mathbf{w}}{\text{argmin}} \sum_{i=1}^{|\mathcal{D}_{\text{train}}|} -\log P(y_i|\mathbf{x}_i, \mathbf{w}) \quad \text{(Corresponding Loss Function)} \end{aligned}$$

Here, $P(y_i \mid \mathbf{x}_i, \mathbf{w})$ is shorthand for $P(\hat{y}_i = y_i \mid \mathbf{x}_i, \mathbf{w})$, where $P(\hat{y}_i = 1 \mid \mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x}_i)$.
Hence, the loss associated with a single data point is:

$$L_{\mathbf{w}}(\mathbf{x}_i, y_i) = \begin{cases} -\log P(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}) & \text{if} \quad y_i = 1 \\ -\log(1 - P(y_i = 1 \mid \mathbf{x}_i, \mathbf{w})) & \text{if} \quad y_i = 0 \end{cases}$$

Combining them,

$$\boxed{L(\mathbf{w}, \mathbf{x}_i, y_i) = -y_i \log P(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}) - (1 - y_i) \log(1 - P(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}))}$$

Total loss across the dataset then would be,

$$L(\mathbf{w}, \mathcal{D}) = \sum_{i=1}^{|\mathcal{D}|} \underbrace{-y_i \log P(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}) - (1 - y_i) \log(1 - P(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}))}_{\text{(Binary) Cross Entropy Loss}}$$

This loss is called the binary cross entropy loss because of the similar structure as the formula for cross-entropy loss in information theory. Some properties about the above defined cross entropy loss:

- It is a convex loss function
- It is differentiable

- There is no closed form solution for the optimal parameter $\mathbf{w}^*$. Hence techniques such as gradient descent are used to optimise the model.

To calculate the gradient, note that

$$\begin{aligned}\nabla_{\mathbf{w}}\sigma(\mathbf{w}^T\mathbf{x}_i) &= \sigma(\mathbf{w}^T\mathbf{x}_i)(1-\sigma(\mathbf{w}^T\mathbf{x}_i))\nabla_{\mathbf{w}}\mathbf{w}^T\mathbf{x}_i \\ &= \sigma(\mathbf{w}^T\mathbf{x}_i)(1-\sigma(\mathbf{w}^T\mathbf{x}_i))\mathbf{x}_i \\ \implies \nabla_{\mathbf{w}}\log\sigma(\mathbf{w}^T\mathbf{x}_i) &= (1-\sigma(\mathbf{w}^T\mathbf{x}_i))\mathbf{x}_i \\ \nabla_{\mathbf{w}}\log\big(1-\sigma(\mathbf{w}^T\mathbf{x}_i)\big) &= -\sigma(\mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i\end{aligned}$$

$$\begin{aligned}\implies \nabla_{\mathbf{w}}L(\mathbf{w},\mathcal{D}) &= \sum_{i=1}^{|\mathcal{D}|} -y_i(1-\sigma(\mathbf{w}^T\mathbf{x}_i))\mathbf{x}_i + (1-y_i)\sigma(\mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i \\ &= \sum_{i=1}^{|\mathcal{D}|}(\sigma(\mathbf{w}^T\mathbf{x}_i)-y_i)\mathbf{x}_i \\ &= \sum_{i=1}^{|\mathcal{D}|}(\hat{y}_i-y_i)\mathbf{x}_i\end{aligned}$$

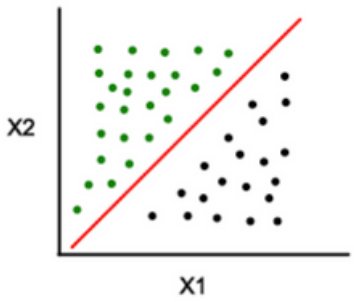
The form of the gradient is very similar to that in linear regression, with $\hat{y} = \sigma(\mathbf{w}^T\mathbf{x}_i)$
This also shows convexity, since

$$\begin{aligned}\nabla^2_{\mathbf{w}}L(\mathbf{w},\mathcal{D}) &= \nabla_{\mathbf{w}}\cdot\nabla_{\mathbf{w}}L(\mathbf{w},\mathcal{D}) \\ &= \sum_{i=1}^{|\mathcal{D}|}\nabla_{\mathbf{w}}\cdot(\sigma(\mathbf{w}^T\mathbf{x}_i)-y_i)\mathbf{x}_i \\ &= \sum_{i=1}^{|\mathcal{D}|}\sigma(\mathbf{w}^T\mathbf{x}_i)(1-\sigma(\mathbf{w}^T\mathbf{x}_i))\ \|\mathbf{x}_i\|_2^2 \\ &> 0\end{aligned}$$

and hence, gradient descent would be good option to minimise loss and find optimal paramters.

5.3 Logistic Regression with Regularization

Logistic regression is capable of perfectly classifying linearly separable data. But there may be mulitple weight vectors that can separate the data. Let us consider the case in which the dataset is linearly separable and the true decision boundary is represented by straight lines.



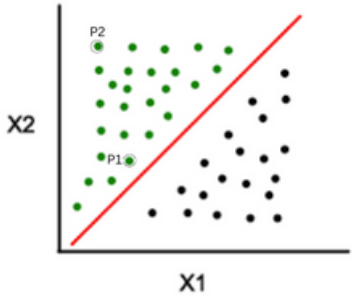
For the data given in the above diagram, the true decision boundary is represented by the straight line $x_1 - 2x_2 = 0$. However, the problem with the supposed values of $w_1 = 1, w_2 = -2$ is that they can be scaled up arbitrarily because the constant term is 0. Hence, it raises the question as to what are the values of $\mathbf{w}$ that the logistic regression classifier is likely to converge at ?

We know that

$$\mathcal{P}(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{w}^\top\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^\top\mathbf{x})}$$

The aim of the classifier is to maximise the above probability for the points in the training dataset and thus it is likely that it will pump up the values of the weights to decrease the value of $\exp(-\mathbf{w}^\top w\mathbf{x})$ and thus increasing the value of $\mathcal{P}(y_i = 1 \mid x_i, w)$ to about 1 for all of the positive samples. Thus, in short it is likely that our classifier would choose high values of the weights w.

But is this ideal?



Consider two positive sample points as shown in the above figure, one very close to the decision boundary P1, and the other further away from the decision boundary P2. We would ideally want a higher value of

$$P(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x})}$$

for P2 than P1 as it is far away from the decision boundary and thus confidence of prediction should be higher for this point than P1. Ideally, we want data points far from the decision boundary like P2 to have a higher probability of being correctly classified by the classifier, while data points near the decision boundary like P1 should not be excessively influenced and receive artificially higher probabilities of being classified into one of the regions. But when the values of the coefficients w in logistic regression are arbitrarily high, it diminishes the distinction between data points located near the decision boundary and those far from it as the probability for all the positive samples nearly tends to 1. This occurs because the exponential term in the logistic function causes the probabilities assigned to both types of points to become substantially higher.

To address this issue, we introduce a **regularization term** along with the loss function.

$$\mathbf{w}_R^* = \arg\min_{\mathbf{w}} \; L_{CE}(\mathbf{w}, \mathcal{D}) + \lambda \|\mathbf{w}\|_2^2$$

The regularization term's purpose is to impose constraints on the values of $\mathbf{w}$, preventing them from becoming arbitrarily high. By applying regularization, we effectively limit the coefficients' magnitudes, which helps maintain the proper distinction between data points and prevents the model from assigning overly confident probabilities to points near the decision boundary.

The regularization penalty term forces the model to find a balance between minimizing the cross entropy loss (fitting the data) and minimizing the regularization term (keeping coefficients small). This means that the model's predictions are less sensitive to minor fluctuations or noise in the training data. Consequently, the above model becomes more robust and less likely to overfit, leading to a **reduction in variance**.

In logistic regression, when the decision boundary is linear, it is relatively straightforward to interpret the model. The feature interpretability of logistic regression models with linear decision boundaries is high which means we can easily assess the importance and relevance of each feature or attribute in making predictions. However, when dealing with non-linear decision boundaries, especially in models with numerous features, interpreting the model and selecting the most informative features becomes much more challenging.

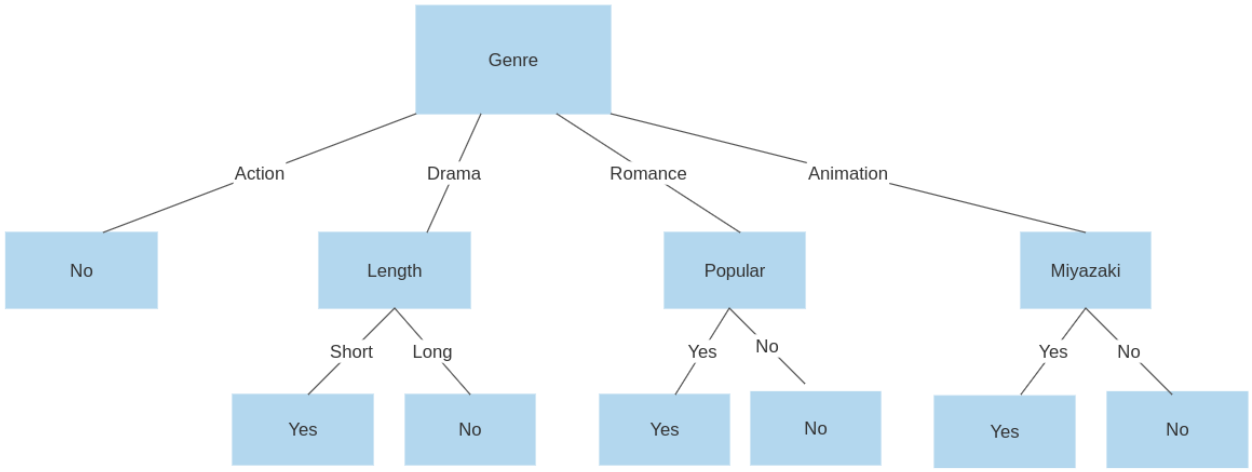
There are 2 desired features of classification models:

1. The ability to learn complex decision boundaries.
2. Interpretability of the model i.e. how useful are each of the features/attributes.

Non-linear logistic regression models aren't easily interpretable and so now, we move onto another kind of classifiers which are Decision Tree Classifiers.

5.4 Decision Trees: Learning complex decision boundary

Decision tree is an interpretable model whose final prediction can be written as a **disjunction of conjunctions** based on the attribute values over training instances.



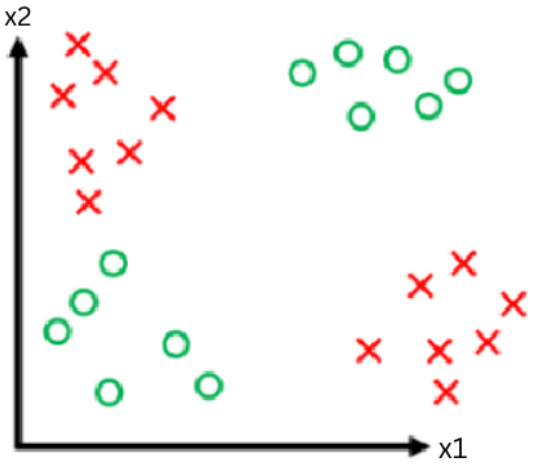
The above figure is an example of a decision tree. It represets how decisions might be made by a decision tree to classify movies that are liked (represented by 'Yes') or disliked (represented by 'No') by a person. Each node of the tree represents an attribute and each path of the tree represents a conjuction of attributes and the final decision is a disjunction of these conjuctions. For instance for the above tree, the final prediction of the model can be written as

$$(\mathtt{Genre} = \mathtt{Drama} \ \wedge \ \mathtt{Length} = \mathtt{Short}) \vee (\mathtt{Genre} = \mathtt{Romance} \ \wedge \ \mathtt{Popular} = \mathtt{Yes})$$
$$\vee (\mathtt{Genre} = \mathtt{Animation} \ \wedge \ \mathtt{Mizayaki} = \mathtt{Yes})$$

The above expression evaluates to true (yields the value 1) for a 'Yes' instance and evaluates to false (yields the value 0) for a 'No' instance.

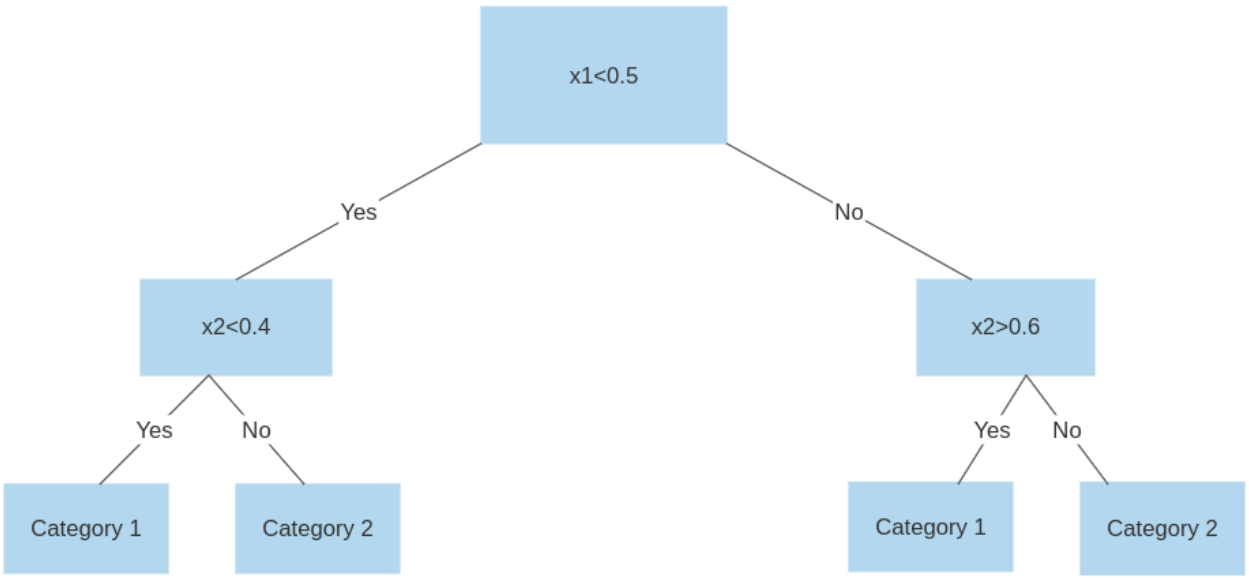
5.4.1 Decision Boundaries of Decision Trees

Consider the following xor-like classified dataset



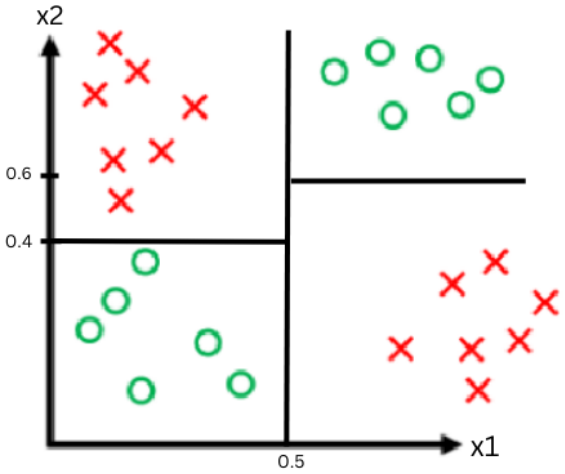
The x and y axis correspond to 2 attributes of the dataset, x1 and x2 respectively. The datapoints marked by green circles and the datapoints marked by red crosses represent 2 different categories (y_{labels}) say category 1 and category 2 respectively.

The following might be the decision tree for the classification of the above dataset.



If the condition in the topmost decision node of the decision tree that is $x_1 < 0.5$ evaluates to true, we move along the left edge of the tree and right otherwise. After traversing an edge, we either land on the condition(node) $x_2 < 0.4$ or $x_2 > 0.6$ respectively. If the condition in the node evaluates to true, we move along the left edge and we move along the right edge in case the condition evaluates to false. If we have moved along the left edge, in this example decision tree, we predict category 1(represented by green circles) or else we predict category 2(represented by red crosses)

Thus, on the basis of the above decision tree, the decision boundaries of the decision tree are as follows:



The vertical line represents the topmost decision node of the decision tree that is $x_1 < 0.5$ and the two horizontal lines represent $x_2 < 0.4$ and $x_2 > 0.6$ respectively. We can see that the decision boundaries are just the depiction of the nodes (the attributes based on which decisions were taken) of the decision tree when plotted on a graph.

Thus, as seen above if nodes of the decision tree depend on a single attribute only, then decision trees divide the feature space into **hyper-rectangles** or equivalently we can say that the resulting decision boundaries are axis-parallel hyper-planes.

NOTE: Instead of axis-parallel hyperplanes, we can even have linear hyper-planes. In the later case, instead of the decision nodes being a linear function of a single attribute, they can be a linear function of 2 or more attributes, say for instance $x_1 + x_2 > 0$. Thus, in such cases instead of axis-parrallel decision boundaries, we would have locally-linear decision boundaries. Typically, axis-parallel hyperplanes are used for decision tree modelling.

5.4.2 Finding the optimal Decision Tree

Unlike the case for logistic regression, finding the smallest Decision Tree that is optimal with respect to some metric (like cross entropy loss etc.) involving all of the attributes is an **NP-hard problem**.
Decision tree estimation is done rather largely **greedily** in which the tree is built recursively.

Here's a simple decision tree template:

- Start from an empty node with all instances.
- Pick the **best** attribute to split on. For instance in figure 1, genre was chosen as the first attribute
- Repeat step 2 recursively for each new node until a **stopping criterion** is met

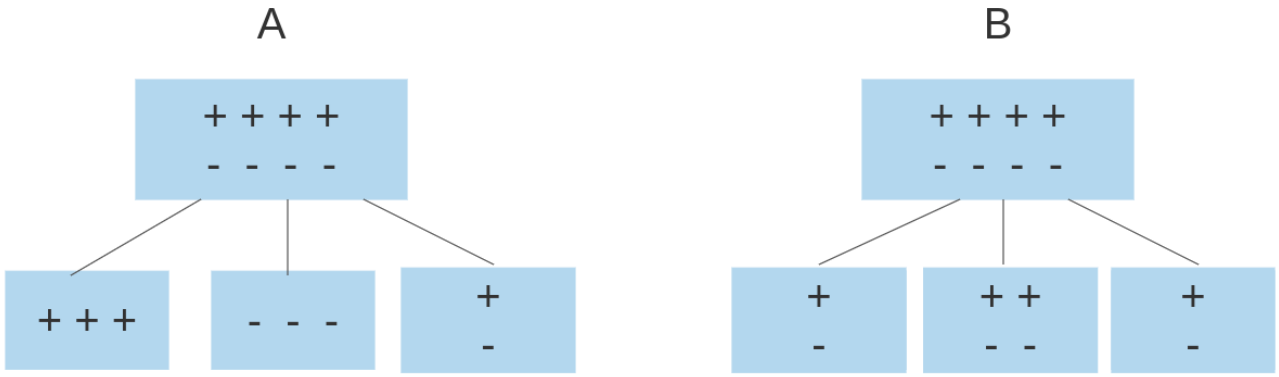
Now, two important questions arise:

- What is the notion of best attribute and how do we find it?
- What is a good stopping criterion?

It is to be noted that a lot of heuristics are involved in decision tree construction which is different from logistic regression we have studied in which we had a clearly formulated optimisation problem and a final solution to the optimisation problem.

We will start by reflecting on how to choose the best attribute.

5.4.3 Choosing node attributes



Unlike the previous trees we have seen, the nodes of the above tree represent the datapoints that would reach these nodes based on the values of some attribute. In both of the above images, the topmost node represents the entire dataset (as no division of the dataset on the basis of some attribute has been done yet). Then on the basis of some attribute which is clearly different for A and B, the dataset is divided into 3 datasets which are present in the three child nodes of each parent.

Seeing the above classification, it seems intuitive that the attribute on which decision has been taken in A is better than the one in B. This is because, A's attribute is leading to nodes that are already largely **homogeneous** and thus the attribute in A immediately effectively reduces the length of the decision tree.

On extending the tree, B might later on have better accuracy but the top attribute is just overly building out the tree which is not preferable as overly large trees might lead to overfitting. Thus, our goal somewhat broadly is find simple models that generate good subtrees that get added on and we want the size of subtrees to be as small as possible.

Intuition for a good split (attribute): A good split for an attribute results in subsets that are (mostly) entirely homogeneous that is the dataset in each split is (mostly) all of the same category.

Now, we need a quantitative way to suit our intuitions for a good attribute and thus we introduce the following concepts.

5.4.4 Entropy

Entropy of a random variable is a measure of **uncertainty** in the value of that random variable. Let X be a random variable and x represent a particular value of the random variable. Entropy of X is H(X) where H(X) is defined as

$$H(X) = -\sum_{x} \mathcal{P}(X = x) \log_2(\mathcal{P}(X = x))$$

To understand what H(X) signifies, let's pick up our familiar coin toss example in which the random variable can take the value 1 or 0 on the basis of the result of the coin toss (heads or tails respectively). H(X) for this random variable is plotted below:

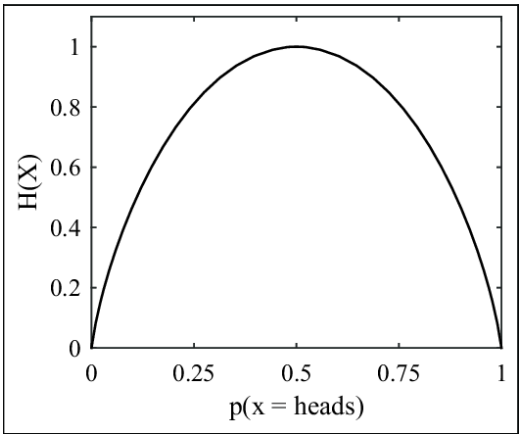


Figure 5.1: Entropy of a coin toss

If the coin is fair (P(Head) = P(Tail) = 0.5) then there is maximum uncertainty in toss outcome and thus entropy has a maxima at this point and if the coin is heavily biased (e.g P(Head)= 0.9, P(Tail)= 0.1) then there is some certainty about whether it would be head or tail and thus entropy is low. Thus, higher the uncertainty in the outcome of the random variable, higher is the entropy. Thus it can be said that,

- A high value of entropy asserts a nearly uniform distribution . We do not know the next outcome(e.g coin toss).
- A lower value of Entropy asserts that the distribution of the random variable has well-defined modes and thus there is some certainty in the outcome of the random variable.

5.4.5 Entropy of a Dataset

Entropy of a dataset measures the uncertainty in the group of observations. Consider a dataset S with k classes/labels. Entropy of the dataset S is H(S) where H(S) is defined as

$$H(S) = -\sum_{i=1}^{k} \mathcal{P}_{i,S} \log(\mathcal{P}_{i,S})$$

where $\mathcal{P}_{i,S}$ is the relative count of instances in S with label i (the probability of randomly selecting an example of class i in S).
What happens when all instances belong to the same class? $\mathcal{P}_{i,s}$ is 1 which implies that entropy of the dataset is 0.

5.4.6 Splitting Criterion: Information Gain

We are building towards deciding on how to choose the best attributes to build our decision tree such that when the data is split on their basis, we achieve maximum possible homogeneity in other words the maximum drop in the entropy within two tree levels.

Information gain, is simply the reduction in entropy caused by partitioning the dataset S according to a particular attribute. It determines the quality of splitting and helps to determine the order of attributes in the nodes of a decision tree.

Gain of a dataset S for a attribute a is defined as

$$Gain(S, a) = H(S) - \sum_{v \in V(a)} \frac{|S_v|}{|S|} H(s_v)$$

where $H(S)$ is the entropy of the dataset before spliting on the basis of the attribute a, S_v is the subset of dataset S whose instances all have the attribute a taking the value v, and thus the term subtracted from $H(S)$ is nothing but the weighted sum of the entropies of the dataset into which the dataset S is split on the basis of attribute a.

Thus for each node of the decision tree, we find the attribute with the maximum information gain and split on its basis in our decision tree.