SoC: Learning in a Library

Reading Material

Summer'24

1 Introduction to Linear Regression

Before we introduce the Linear Regression problem, let us introduce some of the basic notation and terminology used.

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 $\mathbf{x}_i \in \mathcal{X}, y_i \in \mathcal{Y} \;\; \forall i \in \{1 \dots n\}$

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

The focus is to find 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*. 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 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.

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 $h_{\mathbf{w}}$ parameterised by $\mathbf{w}$ as.

$$h_{\mathbf{w}}(x) = w_0 + w_1 x$$

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

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$$

We can now extend this to the multi-dimensional case. We consider our function to return a linear

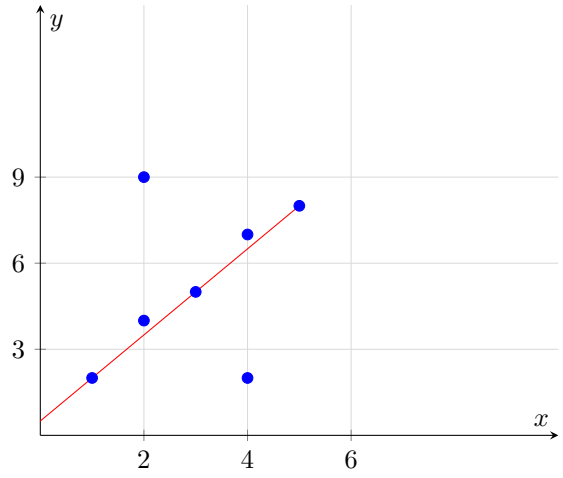


Figure 1: Fitting a line to the data

combination of d dimensional features.

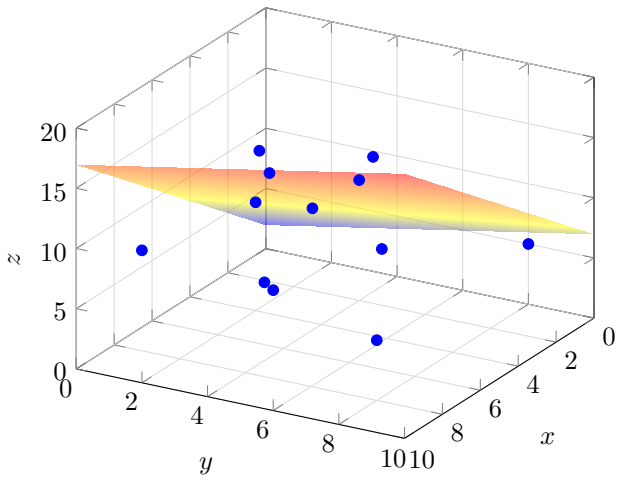


Figure 2: For higher dimensional feature spaces, linear regression is akin to hyperplane fitting

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 an identical expression:

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$$

Our hypothesis class is, then:

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

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.

3 Quantifying 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 parameterized, 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 $\mathbf{x} = [1 \ x_1 \ \dots \ x_d]^\top$. We define a loss for this dataset as

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

We define $h_{\mathbf{w}}(x) = \hat{y}$, and $|y - \hat{y}|$ is called a *residual*.
Now for a dataset containing n datapoints,

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

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 penalize high deviations as much as the mean squared loss.

Here we are interested in the least squared loss. We can vectorize the loss function as:

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

where,

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

4 Solving the optimization problem to find the best predictor

Minimizing 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.

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

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

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.

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

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

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

Note that $\mathbf{X}^\top \mathbf{X}$ need not be invertible.

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

5 Basis Functions

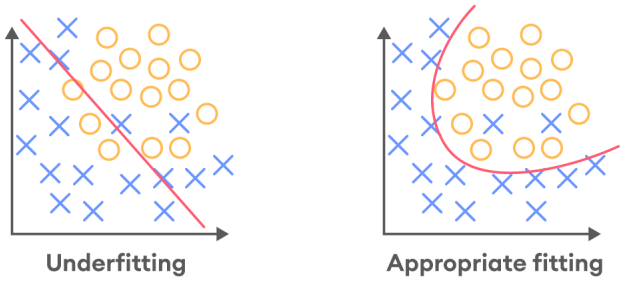


Figure 3: Basis Transformation

- In the above figure, the left-side model represent $h_{\mathbf{w}}(x) = w_0 + w_1 x$ for some w_0, w_1, but it doesn't fit the given dataset well.
- Right-side model fits the dataset well, and represents $h_{\mathbf{w}}(x) = w_0 + w_1 x + w_2 x^2$ 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

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

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

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

Note that $\mathbf{\Phi}_{\mathcal{D}}$ 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:

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

and as before

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

5.1 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, $\Phi_j(\mathbf{x}) = e^{-\frac{\|\mathbf{x}-\mu_j\|_2^2}{\sigma_j^2}}$ 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)

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

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

- 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.

6 Hyperparameter tuning

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.

6.1 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.

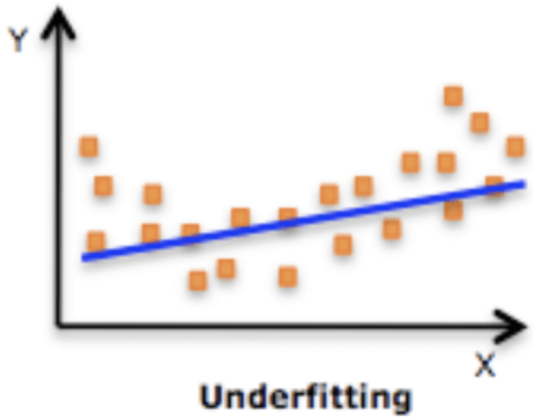


Figure 4: Model is too simple and underfits the data

6.2 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.

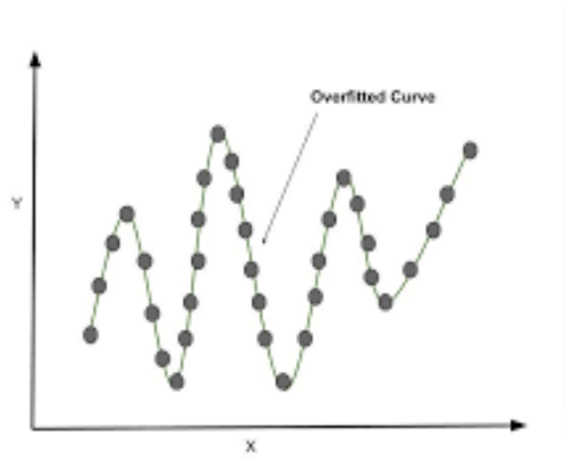


Figure 5: Model fits to the training data perfectly but is overly complex

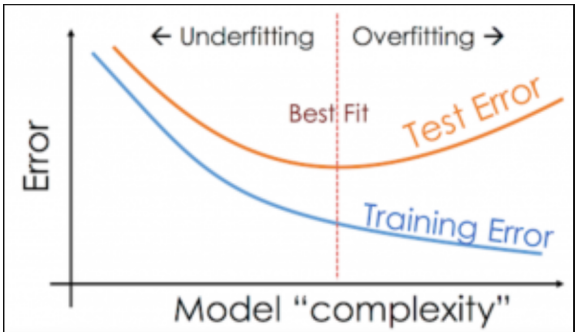


Figure 6: Hyperparameter tuning

7 Notation

The following are the definitions of symbols used in the scribe:

- d is the number of features or dimensions of the data
- n is the total number of data points
- $\mathbf{w} \in \mathbb{R}^{d+1}$ is the vector of parameters that the model aims to learn.
- $\mathbf{y} \in \mathbb{R}^n$ is the vector of target or output values in the training data.
- $\phi \in \mathbb{R}^{n \times (d+1)}$ is the feature matrix; each row in the matrix corresponds to a data point, and each column corresponds to a specific feature.

8 Overfitting

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**.

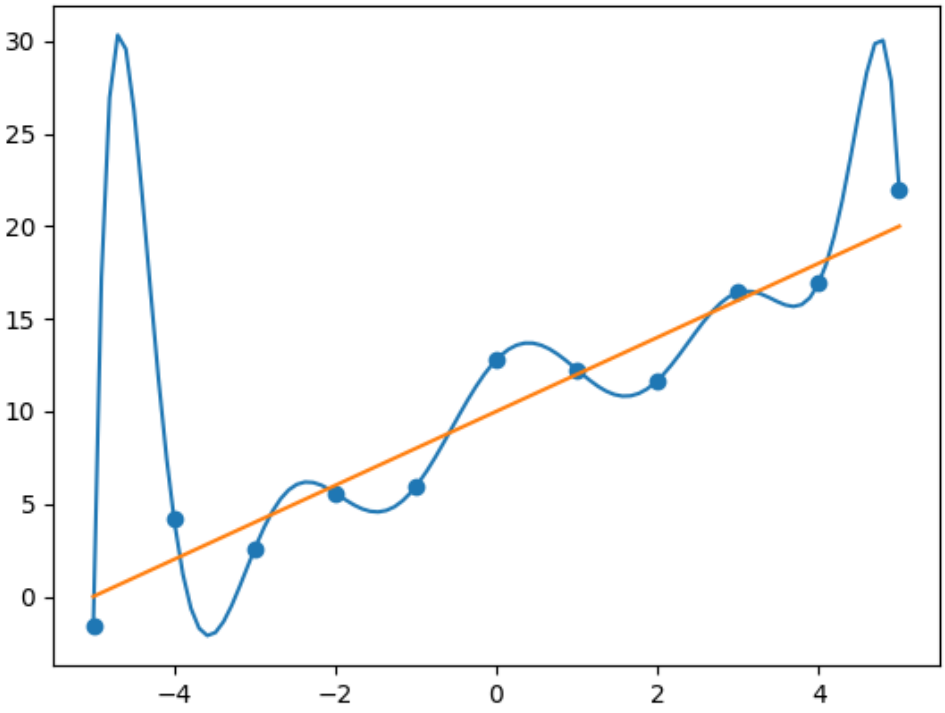


Figure 7: Original

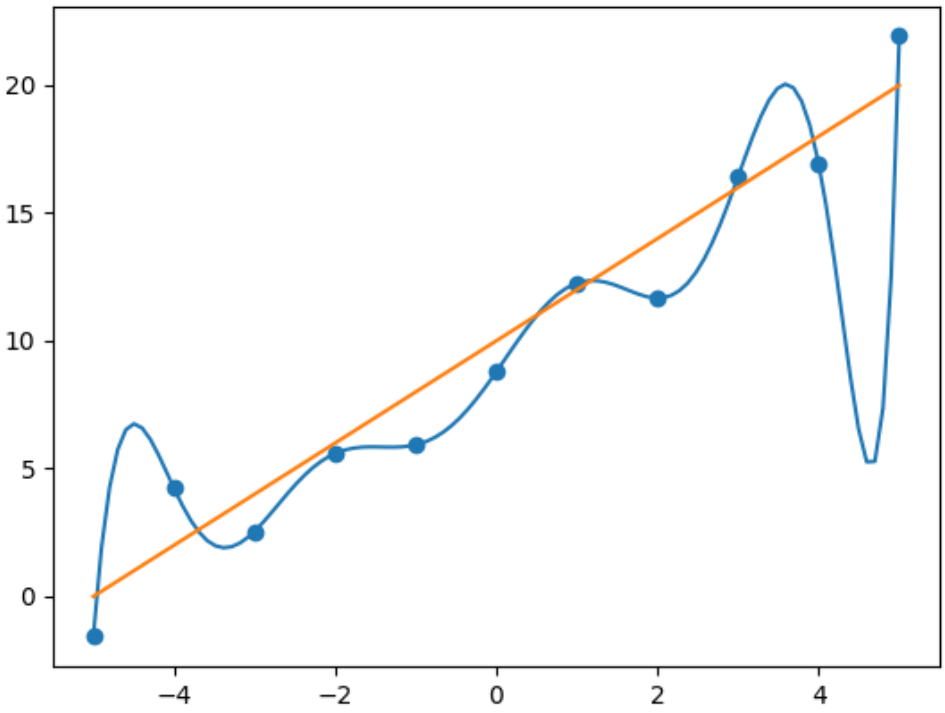


Figure 8: Perturbation

Consider the line y = 2x+10, let's say we sampled few points from this true line, and this process had some noise involved(Figure **??**). In the figure 7, 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), 8), 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).

8.1 Combatting Overfitting

Consider the problem of polynomial regression. Our predictor function $h_{\mathbf{w}}(\mathbf{x})$ is given as:

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

The hyperparameter we need to tune here is d (the degree of the polynomial). To find the best d, a possible **coarse strategy** would be to tune d and evaluate the model on a development set to pick a good value of d. This is not a very efficient approach as it leads to higher training times and hence increased computational cost.

9 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!

9.1 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.

9.2 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:

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

The equivalent constrained form for the above is:

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

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)

9.3 Comparison

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

Referring to the figure 9, 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.

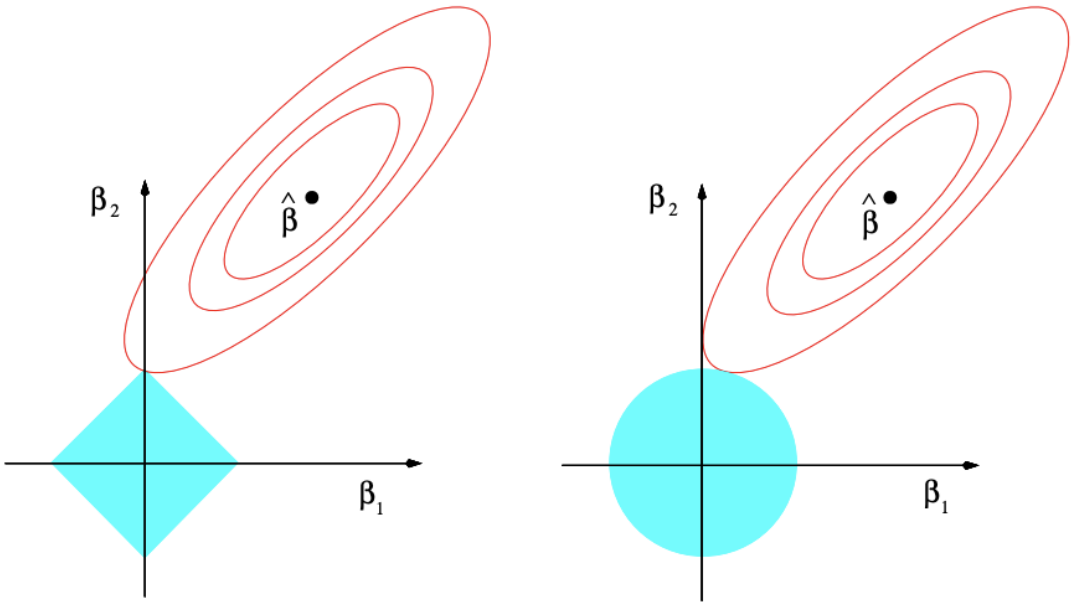


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are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

Figure 9: 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)

10 Introduction to Gradient Descent

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.

11 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)
  - $||\nabla L(w_t)||_2 < \epsilon$
  - $||w_{t+1} - w_t||_2 < \epsilon$

12 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 $w^* = w$

12.1 Why the name Gradient Descent

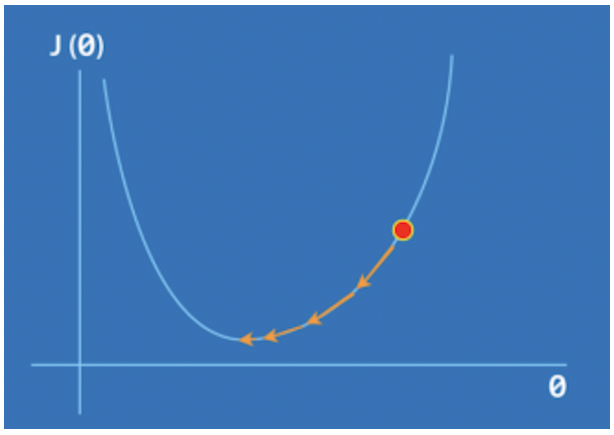
The name Gradient Descent is made of two parts:

- **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

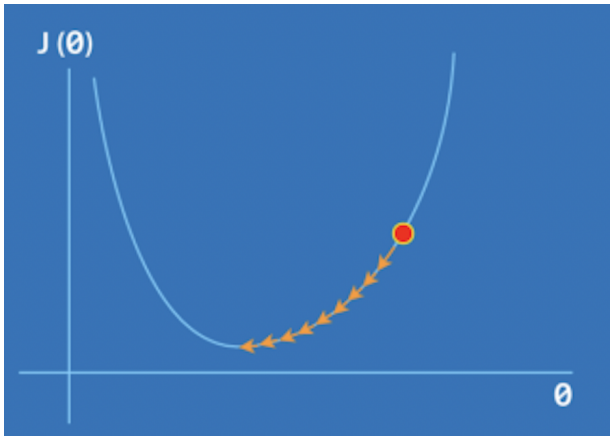
13 GD for the 1-D case and Linear Regression



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:

13.1 What if the step size is too small?

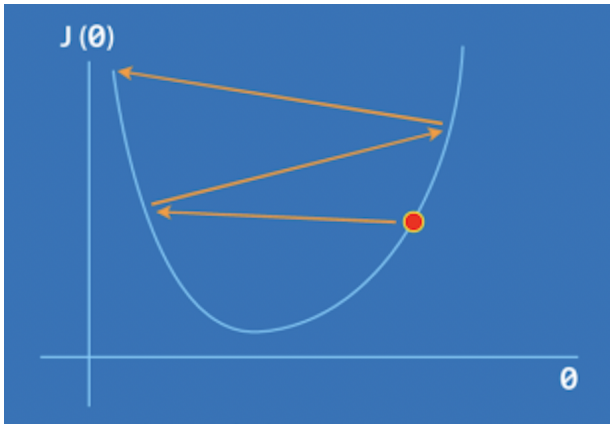
The plot obtained with very small step size would look something like the plot given below



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.

13.2 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.

14 Weight update rule in GD for linear regression

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

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla L(\mathbf{w}) \tag{1}$$

Unregularized Loss:

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

$$L(\mathbf{w}) = \frac{1}{N} \sum_{i}^{N} (y_i - \hat{y}_i)^2 \tag{3}$$

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = -\frac{2}{N} \sum_{i}^{N} (y_i - \hat{y}_i)\mathbf{x}_i = \frac{2}{N} \sum_{i}^{N} (\hat{y}_i - y_i)\mathbf{x}_i \tag{4}$$

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

15 Different Variants of Gradient Descent

15.1 (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.

15.2 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.

15.3 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.

16 Classification using Linear Regression

Recall: In Linear Regression, the output $\hat{y} \in \mathbb{R}$ for an input $\mathbf{x} \in \mathbb{R}^d$ is estimated by a linear function $\hat{y} = \mathbf{w}^T\mathbf{x}$ using a least square objective.

Classification Task

The classification task is to categorize a given input $\mathbf{x}$ into a class label $\hat{y}$ from the set of class labels $\mathcal{Y}$.
The task is referred to as **binary classification** if the set of class labels is $\mathcal{Y} = \{0, 1\}$ (or $\{-1, 1\}$)

A natural way to re-purpose linear regression for binary classification is to predict the class label as follows:

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

Where τ acts as a threshold value 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 $\hat{y} = 1$.

17 Logistic Regression

17.1 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:

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

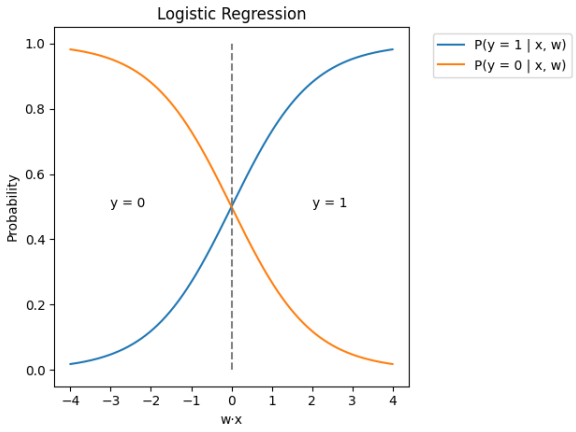
The derivative of the sigmoid function can hence be written in terms of the sigmoid function itself, a property we will use later.

17.2 Model

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

$$\boxed{P(y = 1|\mathbf{x}, \mathbf{w}) = \sigma\big(\mathbf{w}^T\mathbf{x}\big)}$$

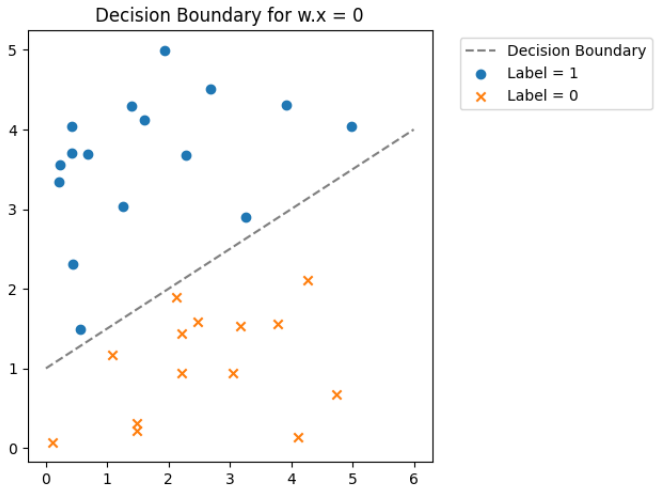
$$\boxed{P(y = 0|\mathbf{x}, \mathbf{w}) = 1 - P(y = 1|\mathbf{x}, \mathbf{w}) = 1 - \sigma(\mathbf{w}^T\mathbf{x})}$$



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

$$\begin{aligned}
&\frac{P(y = 1|\mathbf{x}, \mathbf{w})}{P(y = 0|\mathbf{x}, \mathbf{w})} > 1 \\
\implies &\frac{\sigma\big(\mathbf{w}^T\mathbf{x}\big)}{1 - \sigma(\mathbf{w}^T\mathbf{x})} > 1 \\
\implies &\exp\big(\mathbf{w}^T\mathbf{x}\big) > 1 \\
\implies &\mathbf{w}^T\mathbf{x} > 0
\end{aligned}$$

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.

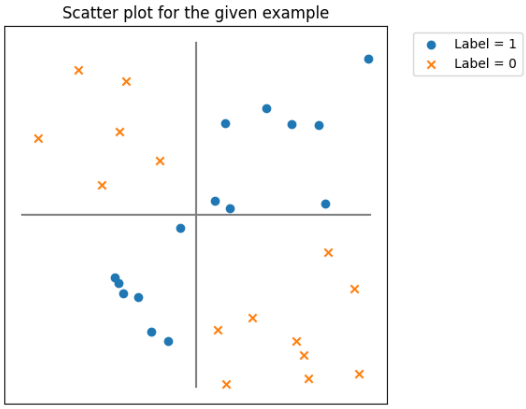


Observe that logistic regression yields a linear decision boundary. Hence it can perfectly classify the training points for **linearly separable data**.

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.

17.3 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,

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

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

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

Combining them,

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

Total loss across the dataset then would be,

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

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

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

$$\nabla_{\mathbf{w}} \log\big(1 - \sigma(\mathbf{w}^T \mathbf{x}_i)\big) = -\sigma(\mathbf{w}^T \mathbf{x}_i)\mathbf{x}_i$$

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

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.