CS 337, Fall 2023
Introduction to Linear Regression

Scribes: Kartik S. Nair*, Ravindra Mohith*, Ravi Shankar Meena*,
Palle Bhavana, Vaibhav Vishal, Gowtham S
Edited by: Ashish Agrawal

3 August 2023

Disclaimer. Please note this document has not received the usual scrutiny that formal publications enjoy. This may be distributed outside this class only with the permission of the instructor.

1 Notation & Introduction

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.

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

*Larger credit to these scribes for the final notes.

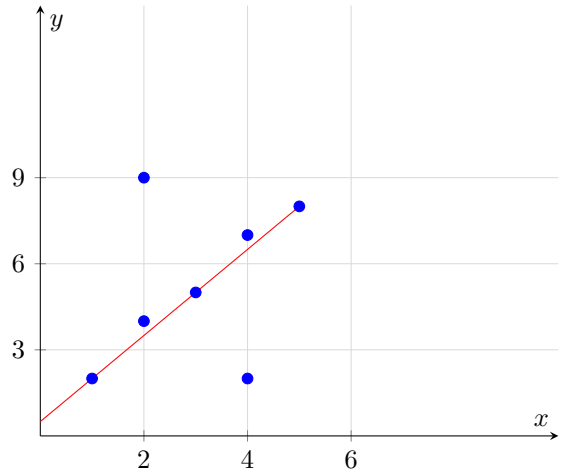


Figure 1: Fitting a line to the data

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 combination of d dimensional features.

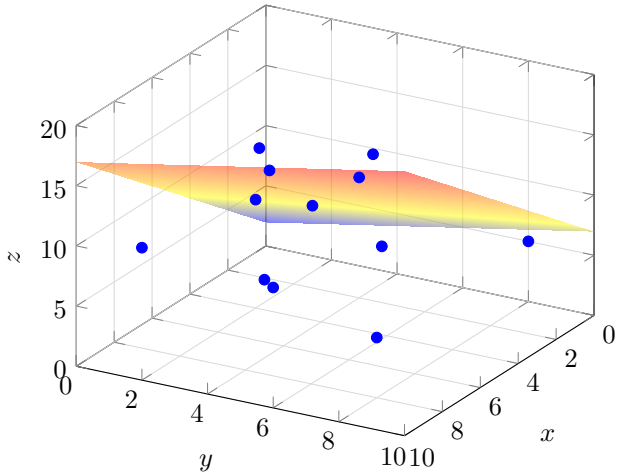


Figure 2: For higher dimensional feature spaces, linear regression is akin to hyperplane fitting

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 $\mathbf{x} = [1 \; x_1 \; \ldots \; x_d]^\top$. 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.

5 Homework

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

$$\mathbf{w}_1{}^* = \frac{\Sigma (x_i - \overline{x})(y_i - \overline{y})}{\Sigma (x_i - \overline{x})^2}$$

where $\overline{x} = \frac{\Sigma x_i}{N}$ and $\overline{y} = \frac{\Sigma y_i}{N}$.

[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.

CS 337, Fall 2023
Basis Functions for Linear Regression, Under/Overfitting

Scribes: Mridul Agarwal, Hruday Nandan Tudu, Soupati Sri Nithya
Arnav Aditya Singh, Chaitanya Garg, Rajkumar
Edited by: Ashwin Ramachandran

Monday 7th August, 2023

1 Recap

- The hypothesis $h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + \ldots + w_d x_d = \mathbf{w}^T\mathbf{x}$ represents a linearly weighted combination of features. Hypothesis class, $H = \{h_{\mathbf{w}} : \mathbf{w} \in \mathbb{R}^{d+1}\}$ where $d+1$ is the number of features in each datapoint.

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

where

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}_{(d+1)\times 1}, \quad \mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}_{(d+1)\times 1}$$

- Least squares loss: $L(\mathbf{w}, D_{\text{train}}) = \sum_i (y_i - \mathbf{w}^T\mathbf{x}_i)^2$ where $\mathbf{x}_i$ is a $(d+1)$ dimensional datapoint, y_i is the target value, $\mathbf{w}$ is the parameter or weight vector.
- Least squares solution: $W_{\text{LS}} = \arg\min_{\mathbf{w}} \sum_i (y_i - \mathbf{w}^T\mathbf{x}_i)^2, \qquad W_{\text{LS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$

where $\mathbf{X} = \begin{bmatrix} \longleftarrow & \mathbf{x}_1^T & \longrightarrow \\ & \vdots & \\ \longleftarrow & \mathbf{x}_n^T & \longleftarrow \end{bmatrix}_{n\times(d+1)}, \quad \mathbf{x}_i = \begin{bmatrix} 1 \\ x_{i1} \\ \vdots \\ x_{id} \end{bmatrix}_{(d+1)\times 1}, \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}_{(d+1)\times 1}$

2 Basis Functions

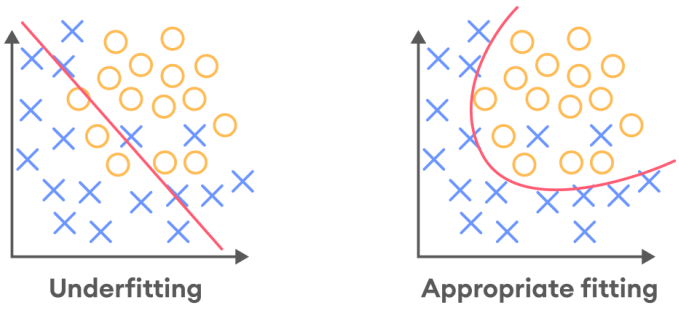


Figure 1: Basis Transformation

- 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

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

2.1 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.

3 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.

3.1 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.

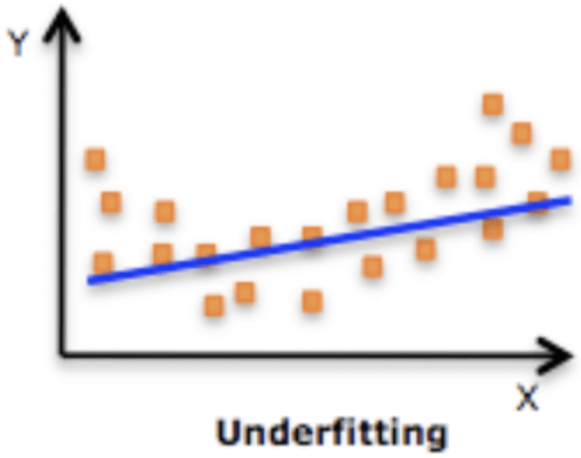


Figure 2: Model is too simple and underfits the data

3.2 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.

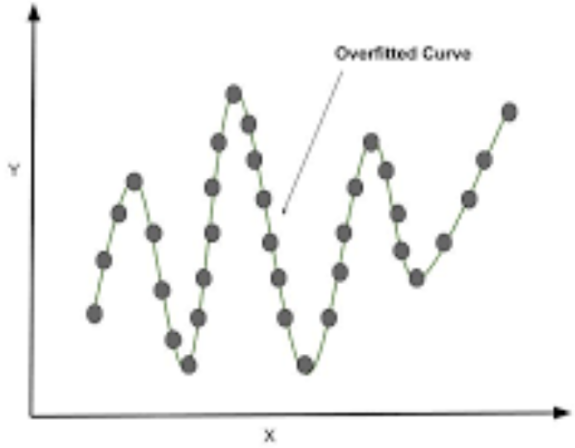


Figure 3: Model fits to the training data perfectly but is overly complex

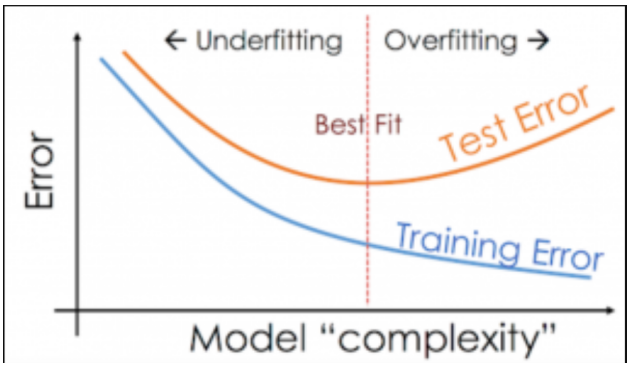


Figure 4: Hyperparameter tuning

CS 337, Fall 2023
Overfitting and Regularization

Scribes: Sabyasachi Samantaray*, Aditya Nemiwal*, Chilukuri Abhiram*,
Shah Param Kaushik, Kommineni Sai Lokesh, Mandala Praneeth Goud
Edited by: Ashwin Ramachandran

August 8, 2023

1 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.

2 Overfitting

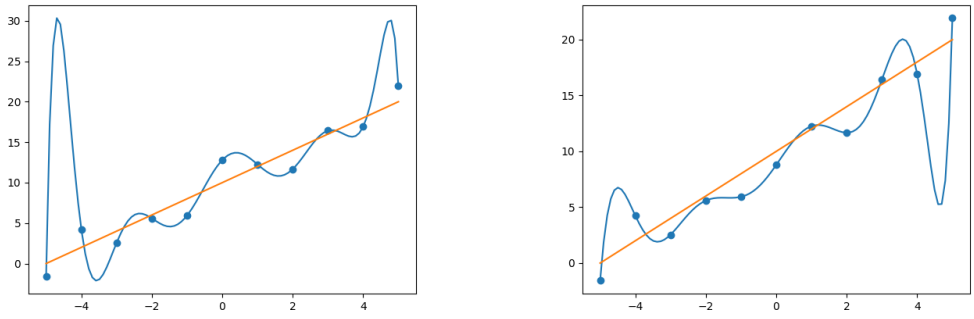
It is tempting to assume that having large number of parameters (making complex model) in our hypothesis class would fit the training data perfectly. But this would fail to predict new unseen data miserably. This is termed **overfitting**.
Consider the line y = 2x+10, let's say we sampled few points from this true line, and this process had some noise involved(Figure 1). In the figure 1a, 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), 1b), 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).

*Larger credit to these scribes for the final notes.



(a) Original (b) Perturbation

Figure 1: Overfitting due to complex modelling

2.1 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.

3 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!

3.1 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.

3.2 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)

3.3 Comparison

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

Referring to the figure 2, 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.

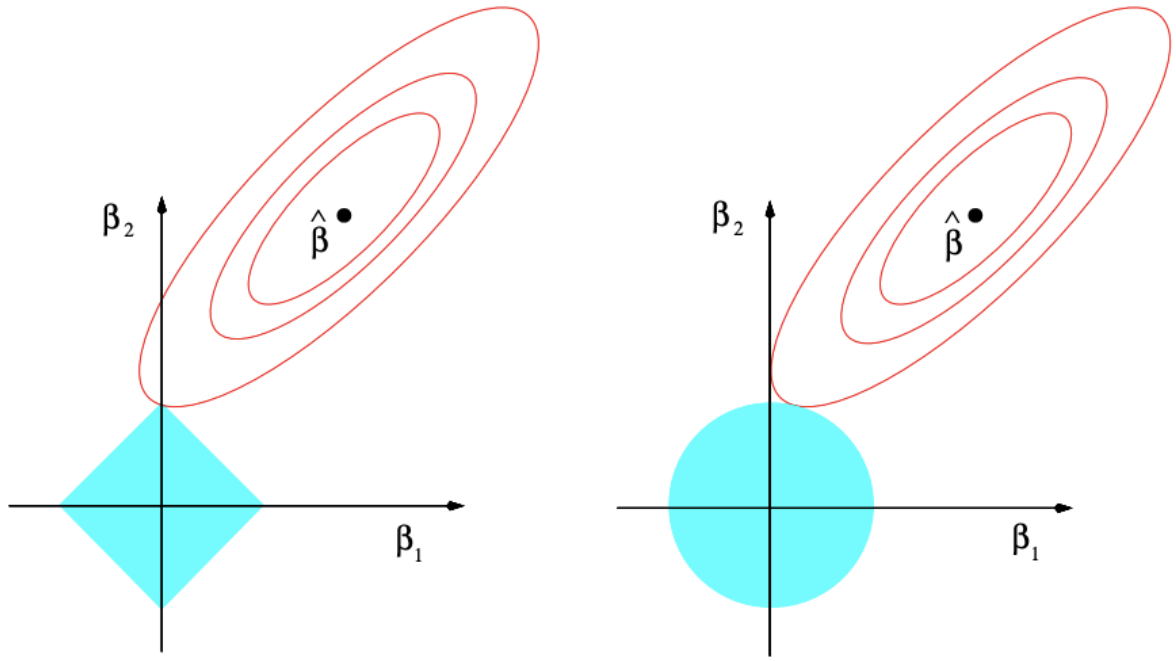


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are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

Figure 2: 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)

CS 337, Fall 2023
Gradient Descent

Scribes: Karan Godara, Akshat Kumar Gupta, Doddy Subhash
Sankalan Baidya, Omm Agrawal, Gangisetty Krishna Sai Kusalts*
Edited by: Ashwin Ramachandran

August 10, 2023

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.

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

*All scribes contributed towards the final notes.

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 ← w_0;
for t ← 1 to N_max do
    Compute gradient direction: ∇L(w);
    Update: w ← w − α∇L(w);
    if ||∇L(w)||_2^2 ≤ ε then
        break;
return w* = w
```

3.1 Why the name Gradient Descent

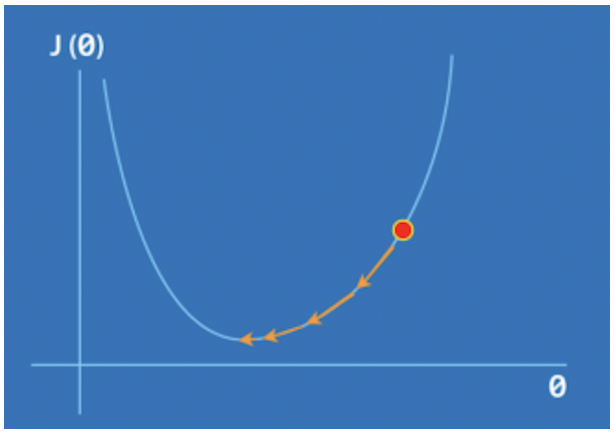
The name Gradient Descent is made of two parts:

- **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

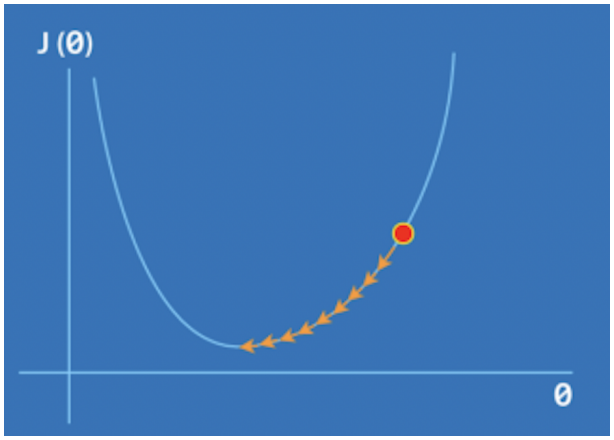
4 GD for the 1-D case and Linear Regression



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:

4.1 What if the step size is too small?

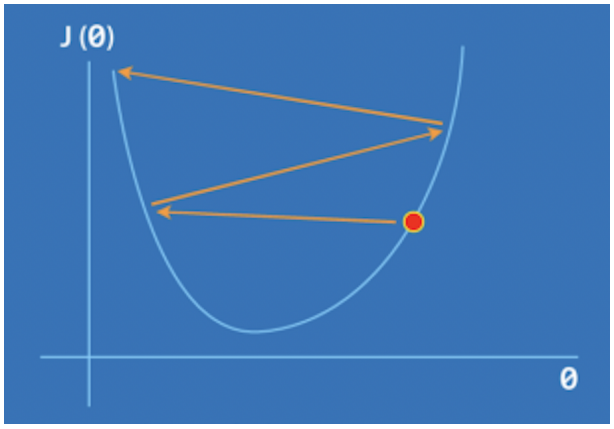
The plot obtained with very small step size would look something like the plot given below



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.

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.

5 Weight update rule in GD for linear regression

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

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

Unregularized Loss:

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

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

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

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

6 Different Variants of Gradient Descent

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.

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.

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.

7 Probabilistic View of Linear Regression

(Elaborated on further in the next lecture)

For training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{\mathrm{n}}$. 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.($\mathrm{Cov}(\epsilon_i, \epsilon_j) = 0$ for all $j \neq i$). So basically now we can get the following interpretation of data,

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

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

$\mathrm{P}(y_1, y_2, \ldots, y_n \mid x_1, x_2, \ldots, x_n, \mathbf{w}) = \prod_i \mathrm{P}(y_i \mid x_i, \mathbf{w})$ [y_i's are conditionally independent given x_i's]

$\implies \log \mathrm{P}(y_1, y_2, \ldots, y_n \mid x_1, x_2, \ldots, x_n, \mathbf{w}) = \sum_i \log \mathrm{P}(y_i \mid x_i, \mathbf{W})$ [Taking log on both sides]

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

CS 337, Fall 2023
Probabilistic Model of Linear Regression, MLE, MAP, Conjugate Priors

Scribes: Manasi Ingle, Sreelaxmi Gasada, Kanishk Garg,
Modulla Hrushikesh Reddy, Ankan Sarkar, Shantanu Welling*
Edited by: Bhavani Shankar

Monday 14th August, 2023

1 Probabilistic Model Of Linear Regression

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 $y_i \in \mathbb{R}$,

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

*All scribes get equal credit for the final notes.

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

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

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.

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.)

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

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

$$P(D \mid \theta) = \theta_H^N (1-\theta)_T^N$$

$$\begin{aligned}
\theta_{\text{MLE}} &= \arg\max_{\theta} \log P(\mathcal{D}|\theta) && (1)\\
&= \arg\max_{\theta} \log \left(\theta^{N_H}(1-\theta)^{N_T}\right) && (2)\\
&= \arg\max_{\theta} \left(N_H \log\theta + N_T \log(1-\theta)\right) && (3)
\end{aligned}$$

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:

$$\frac{N_H}{\theta_{\text{MLE}}} - \frac{N_T}{1-\theta_{\text{MLE}}} = 0$$

$$\theta_{\text{MLE}} = \frac{N_H}{N_H + N_T} = \frac{N_H}{N}$$

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:

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

2.3 MLE for linear regression

$$w_{\text{MLE}} = \arg\max \sum_i \log P(y_i \mid \mathbf{x}_i, \mathbf{w}) \tag{4}$$
$$= \arg\max \sum_i \log(\mathcal{N}(\mathbf{w}^T\mathbf{x}_i, \sigma^2)) \tag{5}$$
$$= \arg\max \sum_i \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_i - \mathbf{w}^T\mathbf{x}_i)^2}{2\sigma^2}} \tag{6}$$
$$= \arg\max C - \sum_i \frac{(y_i - \mathbf{w}^T\mathbf{x}_i)^2}{2\sigma^2} \tag{7}$$
$$w_{\text{MLE}} = \arg\min \sum_i (y_i - \mathbf{w}^T\mathbf{x}_i)^2 \tag{8}$$

- 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.

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 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 $P(\theta)$ for parameters θ (before observing the data D).

By Bayes' theorem, we have :

$$P(\theta|\mathcal{D}) = \frac{P(\mathcal{D}|\theta)P(\theta)}{P(\mathcal{D})}$$

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

$$\theta_{\text{MAP}} = \arg\max_{\theta} P(\theta|\mathcal{D}) \tag{9}$$

$$= \arg\max_{\theta} (\log P(\mathcal{D}|\theta) + \log P(\theta)) \tag{10}$$

Note:

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

$$P(\theta|\mathcal{D}) \propto P(\mathcal{D}|\theta)P(\theta)$$

$$\implies \log P(\theta|\mathcal{D}) \propto \log P(\mathcal{D}|\theta) + \log P(\theta)$$

4.1 Coin example (MAP estimate)

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

$$P(\mathcal{D}|\theta) = \theta_H^N (1-\theta)_T^N$$

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.

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

5 Conjugate Priors

For a likelihood $\mathrm{P}(\mathcal{D} \mid \theta)$ coming from a family of distributions d_1, a prior (from a family d_2) is said to be a conjugate prior if the posterior distribution also comes from the family d_2. (d_2 could also be from the same family as d_1).

5.1 Coin example

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

$$B(N_{\mathrm{H}}, N_{\mathrm{T}}, \theta) = \theta^{N_{\mathrm{H}}}(1-\theta)^{N_{\mathrm{T}}}$$

Also, the PDF for beta distributions is given as,

$$Beta(\theta, \alpha, \beta) = C.\theta^{\alpha-1}(1-\theta)^{\beta-1}$$

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

$$P(\theta|\mathcal{D}) \propto P(\mathcal{D}|\theta)P(\theta) \tag{11}$$
$$\propto \theta^{N_{\mathrm{H}}+\alpha-1}(1-\theta)^{N_{\mathrm{T}}+\beta-1} \tag{12}$$
$$\propto Beta(\theta, N_{\mathrm{H}}+\alpha, N_{\mathrm{T}}+\beta) \tag{13}$$

$$\implies P(\theta|\mathcal{D}) \propto Beta(\theta, N_{\mathrm{H}}+\alpha, N_{\mathrm{T}}+\beta)$$

From the above coin example, the Beta distribution is a conjugate prior for Bernoulli or Binomial likelihoods.

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.

CS 337, Fall 2023
MAP, Bias and Variance

Scribes: Aadhitya Narayanan A B (210070001)*, Kadivar Lisan Kumar (210050076)*, Ancha Pranavi (210050012)*, Ameya Vikrama Singh, Ramavath Sai Srithan, Pragnan Maharshi
Edited by: Bhavani Shankar

August 17, 2023

Recap

We are currently analyzing the probabilistic model of Linear Regression, where the data points are sampled according to

$$y = \mathbf{w}^T\mathbf{x} + \epsilon$$

where ϵ is a zero mean Gaussian noise in the observation with standard deviation σ. Using this, we get

$$\mathcal{P}(y \mid \mathbf{x}, \mathbf{w}) \sim \mathcal{N}\left(\mathbf{w}^T\mathbf{x}, \sigma^2\right)$$

Also, for a given training dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, the expression for log-likelihood of this data will be

$$\sum_{i=1}^{n} \log\big(\mathcal{P}(y_i \mid \mathbf{x}_i, \mathbf{w})\big)$$

Also we derived the Maximum Likelihood Estimate (MLE) as follows

$$\mathbf{w}' = \arg\max_{\mathbf{w}} \left(\frac{-1}{2\sigma^2} \sum_{i=1}^{n} \left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2 \right) \tag{1}$$

Here, we focus on the Maximum A Posteriori (MAP) estimate of the parameter $\mathbf{w}$

1 Maximum A Posteriori Estimate (MAP)

Unlike the Maximum Likelihood Estimate (MLE), which was only dependent on the data, Maximum A Posteriori Estimate (MAP) incorporates information beyond the data. Here, we consider the parameter that we want to estimate as a random variable, say Θ. Our goal will be to find the parameter that maximizes the probability of that parameter given the data, i.e. the **Posterior Distribution**.

$$\Theta_{\text{MAP}} = \arg\max_{\Theta} \big(\mathcal{P}\left(\Theta \mid \mathcal{D}\right)\big) \tag{2}$$

*Larger credit to these scribes for the final notes.

Using Bayes Theorem, we can write the Posterior Distribution as follows

$$\mathcal{P}(\Theta \mid \mathcal{D}) \propto \mathcal{P}(\mathcal{D} \mid \Theta)\ \mathcal{P}(\Theta)$$

the first term is the probability of observing given data for a particular parameter, i.e., the **Likelihood**. And the second term is called the **Prior Distribution** over the parameter. This is the extra information about the parameter which is being added to improve the estimate when the provided data is limited. Hence, we can rewrite equation 2 as

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

1.1 Coin Toss MAP Estimate

We have to estimate the probability of getting a head for a biased coin (say Θ). We are given data of N coin tosses, of which N_H are heads and N_T are tails. As we have derived earlier, the likelihood of this observation will be

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

For the prior distribution, we can choose a Beta distribution (which is the conjugate prior to Bernoulli Likelihood)

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

And hence, the Posterior will be

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

And,

$$\begin{aligned}\Theta_{\text{MAP}} &= \arg\max_{\Theta}\Big(\log\big(\mathcal{P}(\mathcal{D} \mid \Theta)\big) + \log\big(\mathcal{P}(\Theta)\big)\Big) \\ &= \arg\max_{\Theta}\Big(\big(N_H+\alpha-1\big)\log(\Theta) + \big(N_T+\beta-1\big)\log(1-\Theta)\Big)\end{aligned}$$

On maximizing it, we get

$$\begin{aligned}\frac{d}{d\Theta}\Big(\big(N_H+\alpha-1\big)\log(\Theta) + \big(N_T+\beta-1\big)\log(1-\Theta)\Big) &= 0 \\ \frac{N_H+\alpha-1}{\Theta} - \frac{N_T+\beta-1}{1-\Theta} &= 0 \\ \Theta_{\text{MAP}} = \frac{N_H+\alpha-1}{N+\alpha+\beta-2}&\end{aligned}$$

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

We can see that as $N \to \infty$, Θ_{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.

1.2 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.

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

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

$$\begin{aligned}
\mathbf{w}_{\text{MAP}} &= \arg\max_{\mathbf{w}} \big(\mathcal{P}(\mathcal{D}\,|\,\mathbf{w})\ \mathcal{P}(\mathbf{w})\big) \\
&= \arg\max_{\mathbf{w}} \Big(\log\big(\mathcal{P}(\mathcal{D}\,|\,\mathbf{w})\big) + \log\big(\mathcal{P}(\mathbf{w})\big)\Big)
\end{aligned}$$

Using equation 1 to rewrite the log-likelihood term and plugging in the prior term gives us,

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

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)

2 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).

2.1 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

$$\text{Bias} = \mathbb{E}\left[h_{\mathbf{w}}(\mathbf{x})\right] - f(\mathbf{x}) \tag{3}$$

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 (figure 1). Then the bias represents how close the average of these predictor functions is to the true distribution.

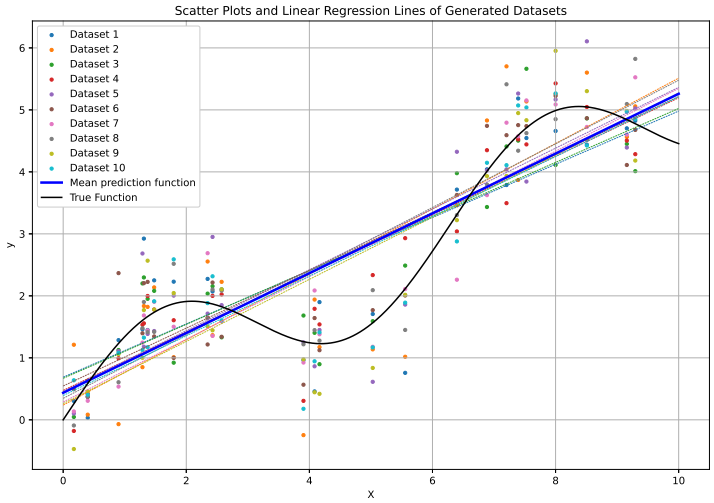


Figure 1: Showing various regression arising from various data-sets and the mean of them

2.2 Variance

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

$$\text{Variance} = \mathbb{E}\left[\left(h_{\mathbf{w}}(\mathbf{x}) - \mathbb{E}\left[h_{\mathbf{w}}(\mathbf{x})\right]\right)^2\right] \tag{4}$$

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 1.

2.3 Noise

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

$$\text{Noise} = \mathbb{E}\left[\left(y - f(\mathbf{x})\right)^2\right] \tag{5}$$

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

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

2.4 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

$$E_{\text{test error}} = \mathbb{E}_{\widetilde{\epsilon}, \mathcal{D}}\left[\left(\widetilde{y} - h_{\mathbf{w}}(\widetilde{\mathbf{x}})\right)^2\right] \tag{6}$$

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

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

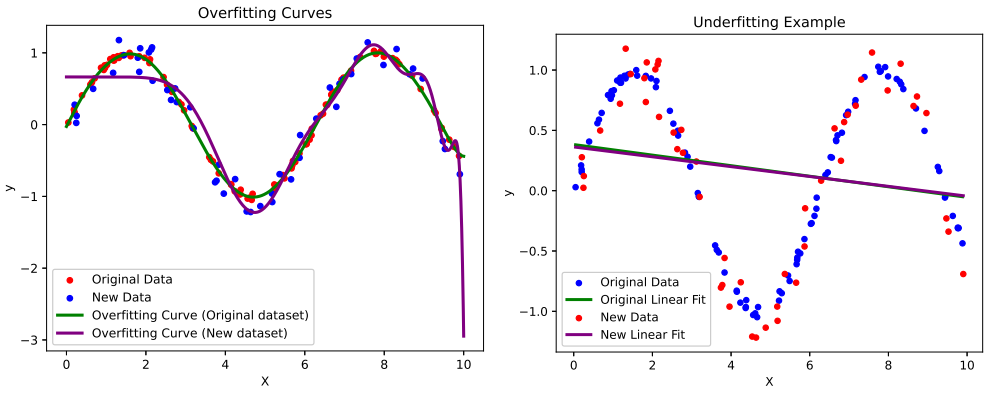
Explanations of steps :-

- The predicted value $h(\widetilde{\mathbf{x}})$ depends only on the data set and hence it is independent of the actual value $\widetilde{y}$ (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 $\mathbb{E}[\widetilde{y}] = \mathbb{E}[f(\widetilde{\mathbf{x}}) + \widetilde{\epsilon}] = \mathbb{E}[f(\widetilde{\mathbf{x}})] + \mathbb{E}[\widetilde{\epsilon}] = f(\widetilde{\mathbf{x}})$ as the noise is zero mean and $\widetilde{\mathbf{x}}$ is a constant (as mentioned before)

2.5 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 (as in Figure 2a).



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

Figure 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 (as in Figure 2b). 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.

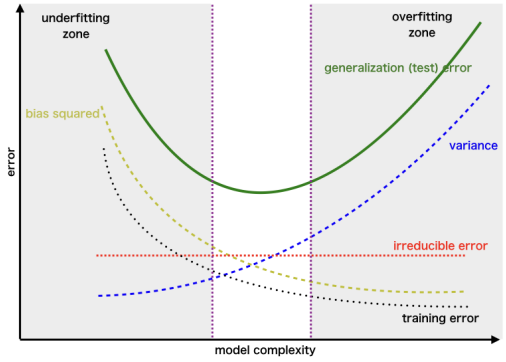


Figure 3: Variance Bias Trade-off with Expected Test Error/Generalisation Error

Hence, there is an inherent trade-off between variance and bias. We can't make both of them arbitrarily small at the same time. This is qualitatively shown in figure 3.

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.

CS 337, Fall 2023
Logistic Regression

Scribes: Arhaan Ahmad*, Prerak Contractor*, Venkatesh*
Tanmay Arun Patil, Harshit K Morj, Ayan Minham Khan
Edited by: Barah Fazili

September 3, 2023

Disclaimer. Please note this document has not received the usual scrutiny that formal publications enjoy. This may be distributed outside this class only with the permission of the instructor.

Remark: In the subsequent text, whenever the term $\mathbf{w}^T\mathbf{x}$ occurs, it is to be assumed that $\mathbf{x}$ is padded with the constant 1 as the first feature.

1 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 $\hat{y} = 1$.

*Most of the content was derived from their submission.

2 Logistic Regression

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:

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

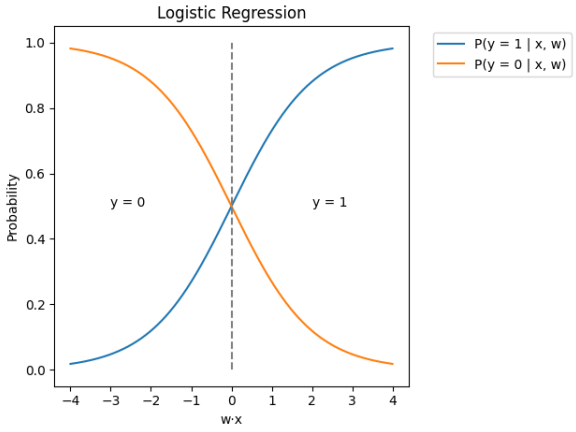
The derivative of the sigmoid function can hence be written in terms of the sigmoid function itself, a property we will use later.

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:

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

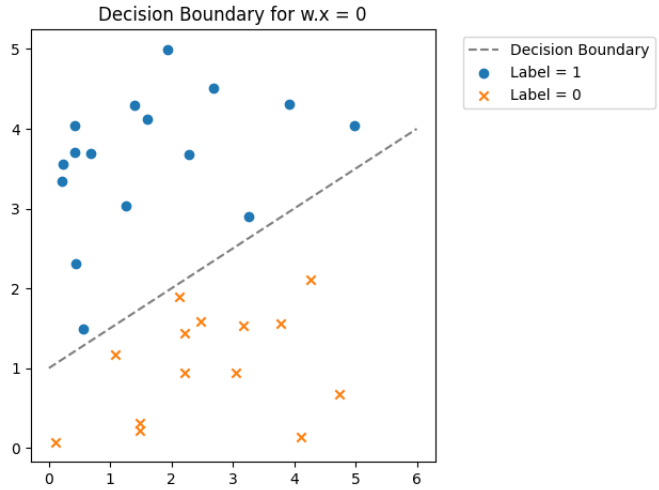
$$\boxed{P(y = 0|\mathbf{x}, \mathbf{w}) = 1 - P(y = 1|\mathbf{x}, \mathbf{w}) = 1 - \sigma(\mathbf{w}^T\mathbf{x})}$$



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

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

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.

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, $P(y_i|\mathbf{x}_i, \mathbf{w})$ is shorthand for $P(\hat{y}_i = y_i|\mathbf{x}_i, \mathbf{w})$, where $P(\hat{y}_i = 1|\mathbf{x}_i, \mathbf{w}) = \sigma\left(\mathbf{w}^T\mathbf{x}_i\right)$.
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

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

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

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.

CS337 , Fall 2023
Regularized Logistic Regression, Decision Tree Classification

Scribes: Isha Arora*, Harshit Agarwal*, Pampa Sow Mondal*
Bijili Prachothan Varma*, Gorle Krishna Chaitanya*, Madur Adarsh Reddy*
Aditya Singh, Parth Pujari, Nenavath Preetham
Yashwanth Reddy Challa, Aditya Yadav, Aditya Rao Gulbarga
Edited by: Barah Fazili

August 29 and August 31, 2023

1 Logistic Regression with Regularization

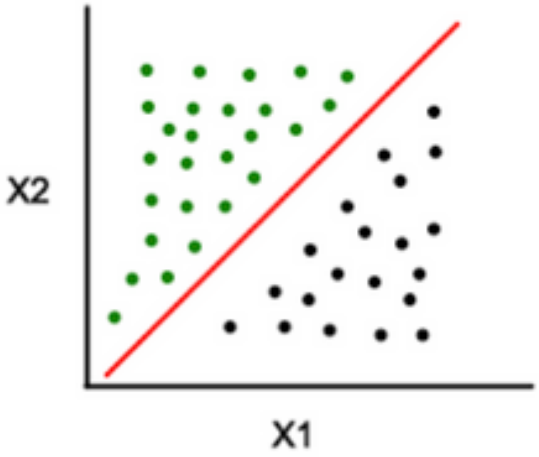


Figure 1: Linearly separable data

*Most of the content was derived from their submission.

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 the straight line:

$$x_1 - 2x_2 = 0$$

as shown in the above figure.
The logistic regression model while predicting the decision boundary, may predict it as

$$x_1 - 2x_2 = 0$$

and thus

$$w_1 = 1,\ w_2 = -2$$

or the values of w_1 and w_2 might be arbitrarily scaled up by the same amount (the decision boundary remains unchanged with arbitrary equal scaling) For instance, the weights predicted by the logistic regression model might be

$$w_1 = 10,\ w_2 = -20$$

or

$$w_1 = 10^5,\ w_2 = -2 * 10^5$$

Given this one might wonder what are the values of w that the logistic regression classifier is likely to converge at?
We know that:

$$P(y_i = 1|x_i, w) = \sigma(w^T x) = \frac{1}{1 + exp(-w^T x)}$$

The aim of the classifier is to maximize the above probability for the points in the training dataset and thus it is likely that it will scale up the values of the weights to decrease the value of

$$exp(-w^T x)$$

and thus increasing the value of

$$P(y_i = 1|x_i, w)$$

to about 1 for all of the positive samples. Hence it is likely that this (unregularized) classifier would choose high values of the weights w.
But is this ideal?

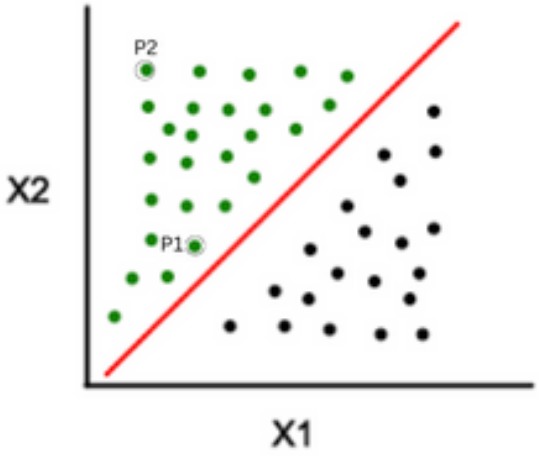


Figure 2: See points P1 and P2

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

$$P(y_i = 1|x_i, w) = \sigma(w^T x) = \frac{1}{1 + exp(-w^T x)}$$

for P2 than P1 as it is far away from the decision boundary and so prediction confidence should be higher for this point than P1. 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.

$$w_R^* = \arg\min_w \; L_{CE}(w, D) + \lambda ||w||_2^2$$

The regularization serves to constrain the values of 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 by preventing 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). Regularization also ensures the model 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 two desired features of classification models:

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

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

2 Learning Complex decision boundary: Decision Trees

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.

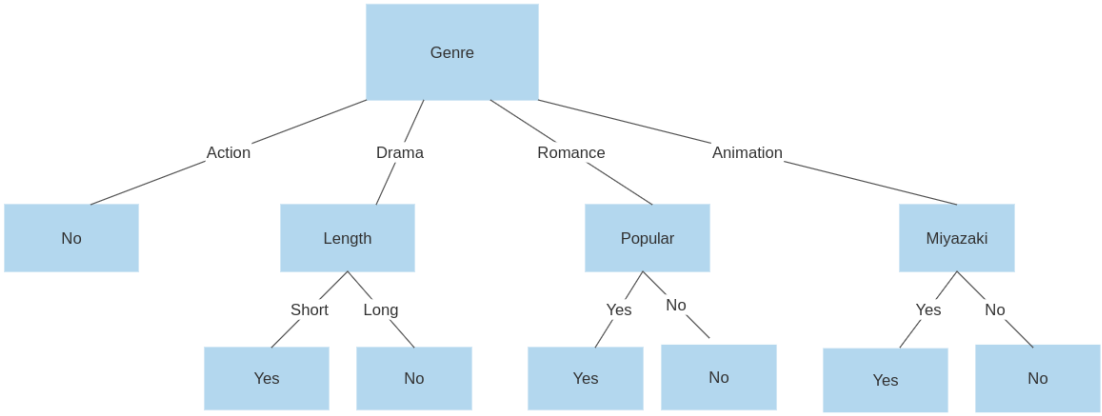


Figure 3: A decision tree to classify movies as liked (represented by 'Yes') or disliked (represented by 'No'). Each node of the tree represents an attribute and each path of the tree represents a conjunction of attributes and the final decision is a disjunction of these conjunctions. For instance for the above tree, the final prediction of the model can be written as $(Genre = Drama \wedge Length = Short) \vee (Genre = Romance \wedge Popular = Yes) \vee (Genre = Animation \wedge Mizayaki = Yes)$. The expression evaluates to True (yields the value 1) for a 'Yes' instance and evaluates to False (yields the value 0) for a 'No' instance.

2.1 Decision Boundaries of Decision Trees

Consider the following xor-like classified dataset in the above figure. The following might be a decision tree for the classification of this dataset.

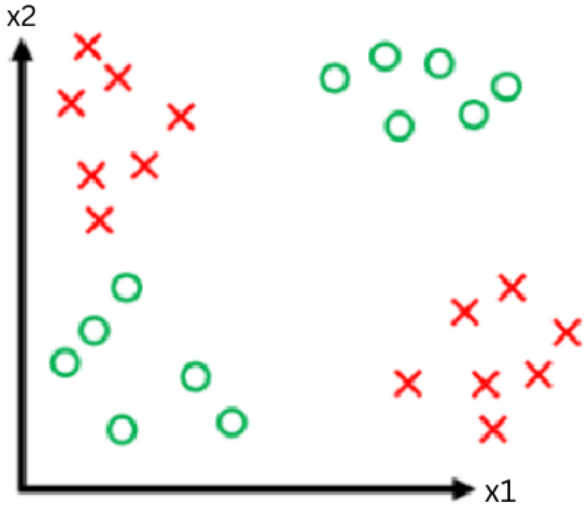


Figure 4: The x and y axis correspond to 2 attributes of the dataset, $x1$ and $x2$ respectively. The datapoints marked by green circles and the datapoints marked by red crosses represent 2 different categories (y_{labels}) say category 1 and category 2 respectively.

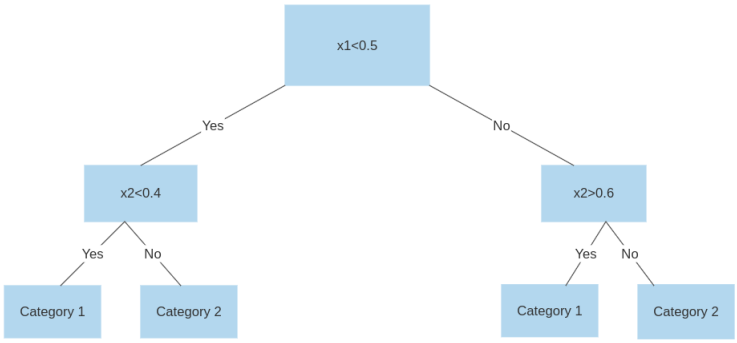


Figure 5: If the condition in the topmost decision node of the decision tree which is '$x_1 < 0.5$' evaluates to true, we move along the left edge of the tree (right otherwise). If the condition in the node at this level further evaluates to True, we move along the left edge again and predict category 1(represented by green circles).

Thus, on the basis of the above decision tree, the decision boundaries of the decision tree are shown in the figure below.
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.

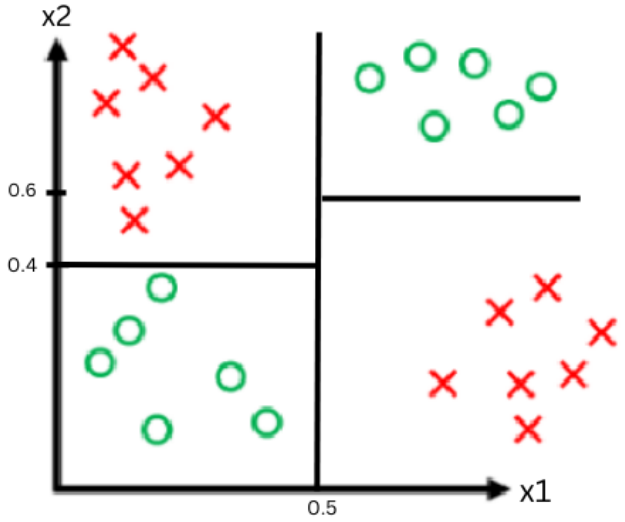


Figure 6: The vertical line represents the topmost decision node of the decision tree that is '$x_1 < 0.5$' and the two horizontal lines represent $x_2 < 0.4$' and '$x_2 > 0.6$' respectively. We can see that the decision boundaries are just the depiction of the nodes (the attributes based on which decisions were taken) of the decision tree when plotted on a graph.

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.

2.2 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 mostly done **greedily** in which the tree is built recursively.

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 in contrast to logistic regression where we have 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.

2.3 Choosing node attributes

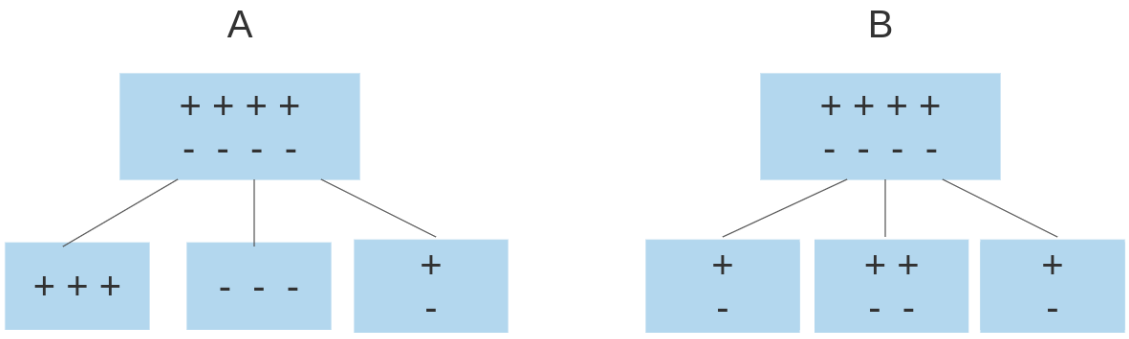


Figure 7: Unlike the previous trees we have seen, the nodes of the above tree represent the datapoints that would reach these nodes based on the values of some attribute. In both of the above images, the topmost node represents the entire dataset (as no division of the dataset on the basis of some attribute 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.

2.4 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

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

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:

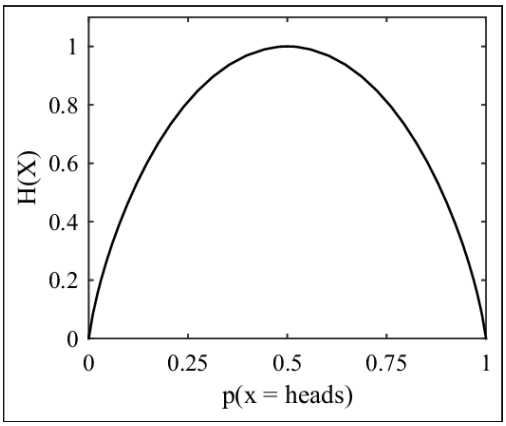


Figure 8: Entropy of coin toss

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.

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

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

where $P_{i,S}$ 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? $p_{i,s}$ is 1 which implies that entropy of the dataset is 0.

2.5 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 *Gain(s,a)* where $Gain(S, a)$ 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.
Recap: A decision tree (DT) is a supervised learning method mainly used for classification tasks. It has a tree structure, with input for classification given at the root and the classification label of the input is decided based on the leaf node it ends up at. An input reaching a node is sent to one of the child nodes based on a certain condition on the input. A label of a node is defined as the maximum occurring label of the instances that reach the node (in training set).
Lets take an example

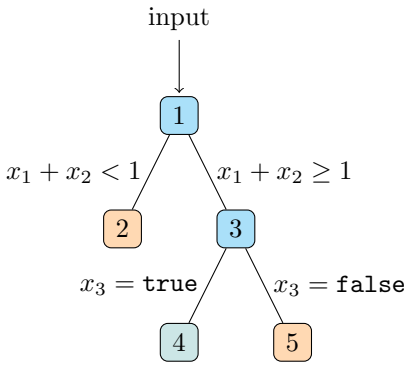


Figure 9: A simple DT, classifying the input based on 3 attributes. Nodes 2,4 and 5 are the leaf nodes.

Note Although node 1 of the DT in the figure above uses a combination of 2 attributes (x_1, x_2) in the condition, in practice, we usually split using condition on only one attribute. This is because it becomes very complex to consider all possible combinations (why only $x_1 + x_2$?, why not $x_1 x_2$, $\log x_1 + x_2$, etc...) of attributes while training. We can take combinations of input attributes if we know some priors on the dataset.

To train the DT we have to answer two main questions:

1. What is the best attribute to split the data?
2. When should we stop building the decision tree?

3 Information Gain

Finding the optimal way of splitting the data is NP-hard. So we resort to using heuristic greedy strategies to get a reasonably good split.

Heuristic is a technique designed for problem solving more quickly when classic methods are too slow for finding an exact or approximate solution, or when classic methods fail to find any exact solution. This is achieved by trading optimality, completeness, accuracy, or precision for speed. In a way, it can be considered a shortcut.

One common heuristic measure used in the context of decision trees is the **information gain**. Before we go to information gain, we have to define **entropy**. Consider a dataset S. Each element $s \in S$ has some attributes $\boldsymbol{x} = (x_1, \ldots, x_n)$ and a label ($l \in \{1, 2, \ldots, C\}$) associated with it. The entropy $H(S)$ is defined as

$$H(S) = -\sum_{l=1}^{C} p_{l,S} \log_2 p_{l,S}$$

$p_{l,S}$ denotes the fraction/probability of elements $s \in S$ that have label l.

Entropy The entropy defined above is conceptually very similar to that in physics and chemistry. Remember this (from JEE days)?

$$\Delta S_{\text{mix}} = -R(n_1 + n_2)\left(\frac{n_1}{n_1+n_2}\ln\frac{n_1}{n_1+n_2} + \frac{n_2}{n_1+n_2}\ln\frac{n_2}{n_1+n_2}\right)$$

Its the entropy of mixing 2 gases (n_1 and n_2 mols). Apart from the multiplicative constants, the expression is the same as $H(S)$.

Now we choose an attribute a from one of $x_1, \ldots, x_n$. Suppose a can attain values from Values(a). All the elements of $s \in S$ which have γ as the value of attribute a will be placed in S_γ. Now, the information gain is defined as

$$\text{Gain}(S, a) = H(S) - \sum_{\gamma \in \text{Values}(a)} \frac{|S_\gamma|}{|S|} H(S_\gamma)$$

Intuition : suppose we choose an ideal attribute that separates all the labels perfectly, then each of $H(S_\gamma)$ will be zero and information gain will be maximum. To gain information we need to reduce entropy by separating the elements. The factor $\frac{|S_\gamma|}{|S|}$ is needed to account for the fact that the set sizes are different. The attribute that gives the maximum information gain is the best attribute to split.

$$a^* = \arg\max_a \text{Gain}(S, a)$$

Apart from information gain there are other heuristics like **Gini impurity** that are commonly used in DTs.

Information Theory Basis The information gain that we defined above is called as mutual information in information theory.

$$I(X, Y) = H(Y) - H(Y|X) = H(X) - H(X|Y)$$

$H(X|Y)$ is the conditional entropy.

$$H(X|Y) = \sum_i p(X = x_i) H(Y|X = x_i)$$

Example Consider a dataset (Table 1) with two boolean attributes x_1 and x_2 and label y. Let us find the optimal attribute to split.

x_1	x_2	y
1	1	1
1	0	1
1	1	1
1	0	1
0	1	1
0	0	0

Table 1: Dataset for the example

$$H(S) = -\left(\frac{5}{6}\log_2\frac{5}{6} + \frac{1}{6}\log_2\frac{1}{6}\right) = 0.65$$

$$\text{Gain}(S, x_1) = H(S) - \left(\frac{4}{6} \times 0 + \frac{2}{6} \times 1\right) = 0.32$$

$$\text{Gain}(S, x_2) = H(S) - \left(\frac{3}{6} \times 0 + \frac{3}{6} \times 0.92\right) = 0.19$$

Since information gain is maximum when split using x_1, it is best to split using x_1.

Till now we have considered the attributes of dataset to be discrete variables. What if they are continuous? In case of discrete attribute we can easily split using the different values of the attribute. For **continuous** attributes, we need to identify thresholds (τ) for the attribute values that we can use to design binary questions of the kind, "Is attribute $a \leq \tau$?" with yes/no answers. Here is the procedure to identify these thresholds:

- Let $v_1, \ldots, v_n$ be the values of attribute a in the training set.
- Sort the values in increasing order : $v_{i_1}, \ldots, v_{i_n}$.
- Midpoint m_j is defined as
$$m_j = \frac{v_{i_j} + v_{i_{j+1}}}{2}$$
- Choose all such midpoints m_j whose underlying instances on either side have different labels. These midpoints will serve as thresholds for this continuous attribute.

For example, consider an attribute a that takes the values $\{10, 26, 40, 50, 100, 120\}$ across six training instances with corresponding labels $\{0, 0, 0, 1, 1, 0\}$. The midpoints are $\{18, 33, 45, 75, 110\}$, and the ones that will get picked up as thresholds are $\{45, 110\}$ since they are midpoints located between two instances with differing labels. Thus, the questions that can be asked relevant to attribute a are "Is $a \leq 45$?" and "Is $a > 110$?".

Example: Consider the dataset in Table 2 with two attributes x_1, x_2 which are real numbers. Each instance of the dataset has an associated binary label y (0 or 1).

x_1	x_2	y
0.1	0.53	1
0.2	0.86	1
0.25	0.36	0
0.36	0.91	1
0.47	0.87	1
0.65	0.13	0
0.71	0.82	0
0.85	0.55	0

Table 2: Dataset for the example

Let us start by splitting using x_1. (See the figure below)

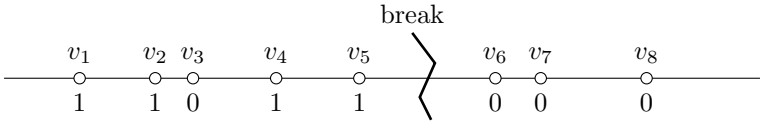


Figure 10: Visualisation of attribute x_1 of Table 2

To maximise information gain, we split between v_5 and v_6 using

$$m_5 = \frac{v_5 + v_6}{2} = 0.56$$

4 Stopping Criteria

Now we come to the second question, when do we stop splitting? These are some of the stopping criteria that help avoid overfitting to the data.

- Stop splitting if all the instances have the same label.
- Do not split if the number of instances at a node is less than k (k is some constant).
- Do not split if the number of leaf nodes exceed a certain threshold.
- Do not split if information gain at a node is below a certain threshold.

The above criteria act as some form of regularization for the DTs. Limiting the max depth of the tree is another possible regularization.

5 Pruning

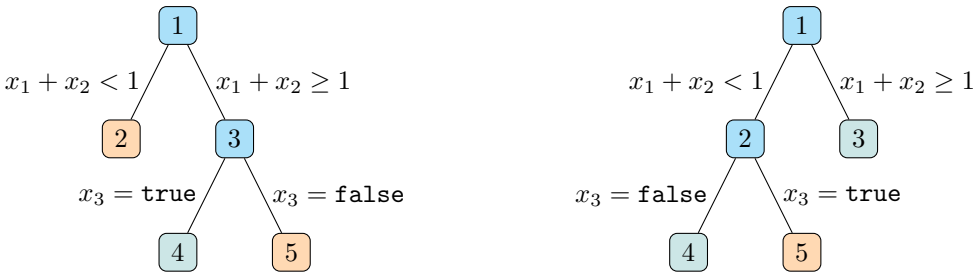
This is an alternate approach to tackle overfitting. Here we build the complete tree and then remove/prune some of the non-critical/redundant branches. To evaluate the performance after pruning, validation set is used.
Pruning a node means replacing the subtree rooted at the node with a leaf node whose label is the max label of the node that was pruned. We can use a greedy approach to pruning:

1. Pick a node which when pruned results in the highest gain in validation accuracy.
2. Repeat step 1 until there is no further improvement in validation accuracy.

This is called as **reduced error pruning**.

6 Random Forests (Brief Introduction)

Decision trees in the form that we studied till now, are very rarely used today. However, DTs are used in random forests (RTs), which is a very common method in **ensemble** learning. An ensemble method uses a combination of methods/models to predict the final output. A random forest is an ensemble model which uses **bootstrap aggregation** or **bagging**.
Bagging involves training an ensemble on bootstrapped data sets. Consider a dataset $\mathcal{D}$ having n samples in it (that is $|\mathcal{D}| = n$. Now we generate m datasets $\mathcal{D}_1, \ldots, \mathcal{D}_m$ each of the same size n ($|\mathcal{D}_1| = \cdots = |\mathcal{D}_m| = n$) by randomly sampling (with replacement) from $\mathcal{D}$. As a result an instance $d \in \mathcal{D}$ might occur zero, once or more than once in $\mathcal{D}_i$. Each of the m datasets is used in training a separate model. The final prediction is made by **aggregating** the results of each of the models (example averaging or voting).
Random forests (Figure 3) are bagged DTs, each of the m datasets is made into a separate DT. In each of these DTs, we pick a random set S (not too small not too large) of attributes to split the data. Taking very small set of attributes will result in losing too much information and taking too many will result in all the m models being nearly the same.



(a) Tree 1 (b) Tree 2

Figure 11: A random forest with $m = 2$

CS 337, Fall 2023

The Perceptron Classifier

Scribes: Atharva Tambat, Balaji Siddardha, Nikhil Biradar,
Anand Narasimhan, B Prabandh, Adithya Gautam
(***Equal contribution by all scribes***)
Edited by: Dhiraj Kumar Sah

September 4, 2023

Recap

We aim to estimate decision boundaries (hyper-planes) that best separate the training data if they are linearly separable. We have already seen one log-linear model - Logistic Regression. We will look at Perceptron now. Although it isn't used in practice directly, it is the building block for SVM classifiers (with non-linearity introduced) and Artificial Neural Networks (although with different update rules).

1 Introduction

Perceptron: A linear model of classification that estimates a hyperplane that 'best' separates the training data.

Hyperplanes are represented by the equation $\mathbf{w}^T\mathbf{x} = 0$, where $\mathbf{w}$ is a column vector of weights (including the bias) and $\mathbf{x}$ being a column vector of inputs (with 1 prepended to the $\mathbf{x}_i$s).

Recall: Equation of hyperplane:

$$\mathbf{w}^T\mathbf{x} = 0$$

where, $\mathbf{w}$ is the orthogonal vector to the plane.
The Perceptron classifier, at test time, predicts a test instance to have the label $h(x)$

$$h(x) = sign(\mathbf{w}^T\mathbf{x})$$

where,

$$sign(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases} \tag{1}$$

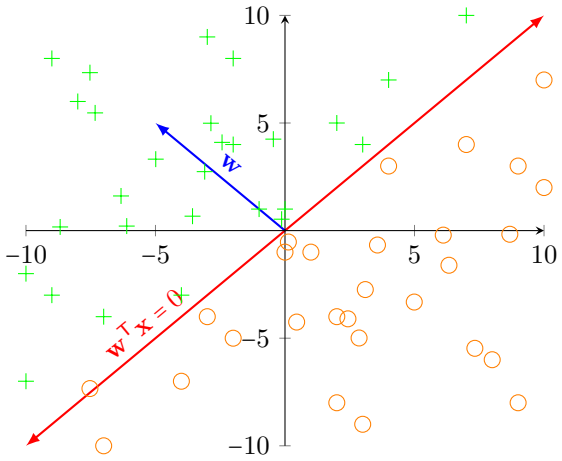


Figure 1: Linear model of classification

2 Properties of the Perceptron Model

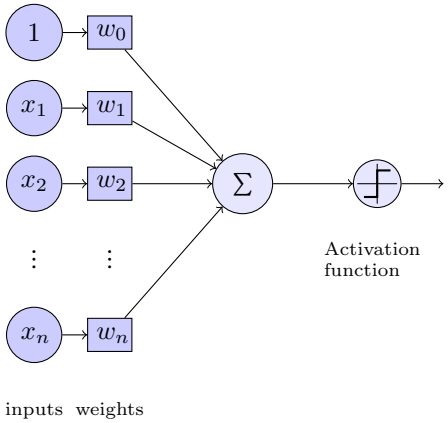


Figure 2: The Perceptron Model

2.1 Neuron-Like

A perceptron is a basic unit of the **Neural model of Learning**. It's very loosely based on how a neuron works. The junctions between the dendrite and axons consist of multiple electrical signals being modulated differently. This is quite similar to the inputs receiving different weights to be multiplied. This can be evolved into a basic feed-forward artificial neural network by adding many more such units and deciding on appropriate (mostly non-linear) activation functions.

A **multi-layered perceptron** (MLP) is a special case of the above but is often used interchangeably. The slight difference arises in the type of loss function used, which in the case of MLPs is the **Heavside** step function, instead of normal non-linear activation functions.
Also, this is mentioned because these generalise the notion of a perceptron to other types of data as well, not just linearly separable.

2.2 Online Algorithm

The perceptron algorithm uses its inputs (training examples) one at a time, using them to update weights. It does this in the same order in which those examples are supplied and does not revisit them again.

These types of algorithms are called **online** algorithms and are of interest to computer scientists from an algorithmic standpoint.

2.3 Error-driven Algorithm

The algorithm only takes action if one of the training examples is classified incorrectly by the hyperplane specified by the current weight vector. In other words, if the training example is classified correctly, change nothing. If not, update the weight vector. We'll see the updation rule shortly.

3 The Perceptron Update Algorithm

Weight update for perceptron model

Inputs: the set of training examples;
Algorithm hyperparameters: number of iterations T;
Initialize weight $\mathbf{w}$ to some $\mathbf{w_0}$
foreach *iteration* $\in \{1, 2...T\}$ **do**
  Pick a random training instance $(\mathbf{x}, y)$ where $\mathbf{x} \in \mathbb{R}^d$ and $y \in \{-1, 1\}$;
  Predict $\hat{y} = sign(\mathbf{w}^T\mathbf{x})$;
  if $\hat{y} \neq y$ **then**
    $\mathbf{w_{t+1}} \leftarrow \mathbf{w_t} + y\mathbf{x}$
  end if
end foreach
return $\mathbf{w_T}$

The update equation can be made more compact by saying:

Let $(\mathbf{x}', y')$ be one training instance at iteration t, where weight is $\mathbf{w_t}$
if $y' \cdot \mathbf{w_t}^T\mathbf{x}' \leq 0$ **then**
  $\mathbf{w_{t+1}} \leftarrow \mathbf{w_t} + y'\mathbf{x}'$;
else
end if

The above modification implies that if the signs of y' (label) and $\mathbf{w_t}^T\mathbf{x}$ (scaled predicted label) are not aligned, then the weights should be updated (Note that weights should also be updated on $y' \cdot \mathbf{w_t}^T\mathbf{x}' = 0$ because the weights may be initialized to 0, so they should definitely be updated in the first iteration).

- To ensure that all instances are gone over at least once without repetition, the training set is shuffled and sequentially gone through - avoiding picking the same data repeatedly.
- It is common to take an average of weights $\mathbf{w}$ over training iterations because updates in the latter part of the training may have changed the weight to miss-classify some samples in the initial training.

4 Intuition of the weight update rule

Weight update rule (for in instance $(x, y) \in \mathcal{D}_{train}$) of the perceptron:

$$\mathbf{w_{t+1}} \leftarrow \begin{cases} \mathbf{w_t} + \mathbf{x}, & \text{if } y = 1 \\ \mathbf{w_t} - \mathbf{x}, & \text{if } y = -1 \end{cases} \tag{2}$$

The intuition behind the update rule is that the weight vector ($\mathbf{w_{t+1}}$) is moved towards or away from the current weight ($\mathbf{w_t}$), depending on whether its label is positive or negative respectively. As shown below, the train instance $\mathbf{x}$ will be correctly classified (shifted to the correct side of the hyperplane) after the update.

Also, the condition can be replaced by checking whether $y_i \mathbf{w}_t^T x_i \leq 0$ and performing the update rule if it is. This quantity will also be used later as a metric to show algorithm improvement.

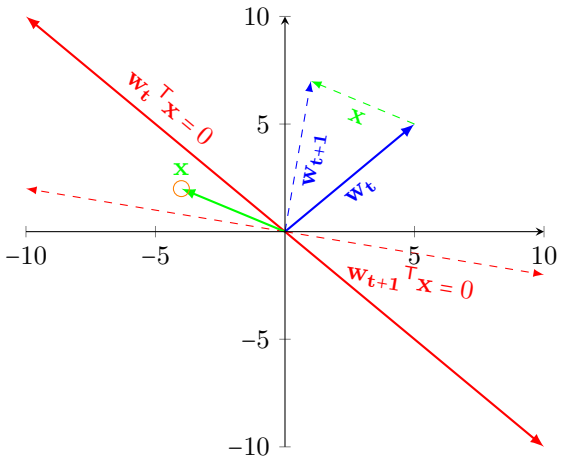


Figure 3: Visualization of the perceptron weight update rule

5 Does the perceptron weight update rule improve performance?

Is the perceptron update algorithm **guaranteed** to improve after a weight update on an erroneous training sample? **No**, it is **not guaranteed** to become correct on the erroneous example immediately after the update. It might work correctly at some later point but, it isn't assured. Let us consider a misclassified example $(\mathbf{x}, y) \in \mathcal{D}_{train}$ i.e. $sign(\mathbf{w_{old}}^T \mathbf{x}) \neq y$. According to the perceptron weight update rule,

$$\mathbf{w_{new}} \leftarrow \mathbf{w_{old}} + \mathbf{x}y$$

$$\begin{aligned} \Rightarrow \quad & y \cdot \mathbf{w_{new}^T x} = y \cdot (\mathbf{w_{old}} + y\mathbf{x})^T \mathbf{x} \\ \Rightarrow \quad & y\mathbf{w_{new}}^T \mathbf{x} = y(\mathbf{w_{old}}^T \mathbf{x}) + y^2 \|\mathbf{x}\|_2^2 > y\mathbf{w_{old}}^T \mathbf{x} \end{aligned} \tag{3}$$

We know that $y\mathbf{w_{old}}^T \mathbf{x} \leq 0$, and $y\mathbf{w_{new}}^T \mathbf{x} > y\mathbf{w_{old}}^T \mathbf{x}$. Our goal is to make $y\mathbf{w_{new}}^T \mathbf{x}$ positive (which implies the label is correctly predicted), or at least closer to that (less negative). As shown above, the weight update rule does exactly that. Note that the update does not guarantee that the label for that example will be correctly predicted. The new value of $y\mathbf{w_{new}}^T \mathbf{x}$ might still be negative (but lesser in magnitude, hence closer to being positive).

So by this, we would assume that the order of training examples is critical since there is no convergence guarantee.

It turns out though, that the perceptron is guaranteed to converge if the data is **linearly separable!**

Perhaps even more surprisingly, the number of iterations does not depend on things like n or d, it just depends on an upper bound of the norm of the points, call it D, and a possible margin of separation γ. The order of iterations that must be made is $\mathcal{O}(D^2/\gamma^2)$

A note on γ: If there exists a hyperplane that correctly classifies the training points such that for both the set of positive and negative examples separately, the minimum distance between the points and the hyperplane is $\geq \gamma$, then the hyperplane is said to have margin of separation γ.

Intuition says that during an update, it could also end up misclassifying previously correctly classified examples, but that's the beauty of the perceptron. The algorithm will find an appropriate classifier if the points are linearly separable.

Aside: In-Class Discussions

- During the update, the bias in $\mathbf{x}$ will only be updated by ± 1 with the above update rule, which is undesirable. To avoid this, the update is modified by a factor α, i.e.

$$\mathbf{w_{new}} \leftarrow \mathbf{w_{old}} + \alpha \mathbf{x} y$$

- While training on the training set, the perceptron can misclassify some previously correctly classified data due to updates while training on the remaining subset of training data. So, how are we guaranteeing that it is improving? Wait for the next class where convergence guarantees will be discussed.
- If one sees that the error is high after training, then the training data is reshuffled and trained over again. The same training sample is not to be iterated over and over again. One common scheme used in practice is to take an average of weights over all iterations.

6 What is the perceptron algorithm minimizing?

A natural objective is to **minimize** in classification is the **number of miss-classified examples**, or equivalently, maximizing $y\mathbf{w}^\mathbf{T}\mathbf{x}$ for all $(\mathbf{x}, y) \in \mathcal{D}_{train}$, with the intuition of making this quantity as positive as possible for as many test instances as possible. Let $\mathcal{D}'$ be the set of miss classified instances $(\mathbf{x}, y) \in \mathcal{D}_{train}$

$$min \sum_{(\mathbf{x},y)\in\mathcal{D}'} -yw^T x$$

The perceptron loss for an instance $(\mathbf{x}, y)$ (whether or not it is miss-classified or not):

$$max\{0, -y\mathbf{w}^T\mathbf{x}\}$$

So, the loss for all training data is:

$$L_{perc}(\mathbf{w}, \mathcal{D}_{train}) = \sum_{(\mathbf{x},y)\in\mathcal{D}_{train}} max\{0, -y\mathbf{w}^T\mathbf{x}\}$$

Figure 4 shows the perceptron loss specified above:

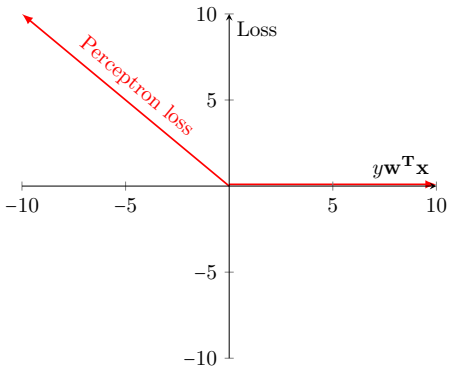


Figure 4: Graph of the perceptron loss function for a single training instance

6.1 Optimizing the loss function

The loss is convex hence, Stochastic Gradient Descent should work for optimizing this loss function. Except for where $y\mathbf{w}^\mathbf{T}\mathbf{x} = 0$, where the loss function is non-differentiable. At these points, subgradients are calculated instead of gradients.

Aside: Subgradients

A subgradient of a convex function $f(\mathbf{w_0})$ is all the vectors $\mathbf{g}$ s.t. for any other point $\mathbf{w}$, we have $f(\mathbf{w}) - f(\mathbf{w_0}) \geq \mathbf{g}^T(\mathbf{w} - \mathbf{w_0})$. Subgradient reduces to gradient when the function is differentiable.

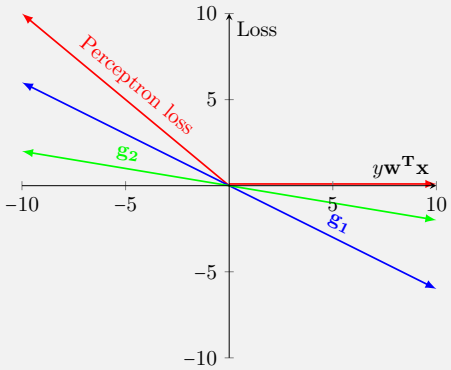


Figure 5: Graph of sub-gradients of the perceptron loss at $y\mathbf{w}^\mathbf{T}\mathbf{x} = 0$

It can be proved that if f is convex and is differentiable at w_0, then its gradient is its only subgradient. The converse can also be proved. Using these subgradients instead of normal gradients, we can run SGD to minimise the loss and obtain a good linear classifier for the data.

CS 337, Fall 2023

Mistake Bounds of The Perceptron Classifier

Scribes: Ashutosh Singh, Avadhoot Gorakh Jadhav, Gohil Megh Hiteshkumar
Govind Kumar, Aman Sharma, Vir Wankhede
(***Equal contribution by all scribes***)
Edited by: Dhiraj Kumar Sah

September 12, 2023

Recap

Perceptron is a linear model of classification. The model aims to estimate the best hyperplane that separates the training data. Equation of hyperplane: $\mathbf{w^T x} = \mathbf{0}$. It makes weight updates by seeing train instances one at a time. Weight updates are triggered **only when** Perceptron makes a mistake. For a given $\mathbf{x}$, perceptron predicts label as $\text{sgn}(\mathbf{w}^\mathsf{T}\mathbf{x})$. If the predicted label does not match the actual label, then the weight vector is moved either towards or away from $\mathbf{x}$ as follows:

$$\mathbf{w_{t+1}} = \mathbf{w_t} + \mathbf{x} \quad \text{if } y = 1$$
$$\mathbf{w_{t+1}} = \mathbf{w_t} - \mathbf{x} \quad \text{if } y = -1$$

Perceptron Algorithm

$\mathbf{w} \leftarrow \mathbf{w_0}$
for iter $= 1, \ldots,$ MaxIter **do**
  for all $(\mathbf{x}, y) \in D$ **do**
    $\hat{y} \leftarrow \text{sgn}(\mathbf{w}^\mathsf{T}\mathbf{x})$
    if $\hat{y} \neq y$ **then**
      $\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}$
    end if
  end for
end for
return $\mathbf{w}$

Perceptron Loss

For an example $(\mathbf{x_i}, y_i)$ the perceptron loss is: $\max(0, -y_i \mathbf{w^T x_i})$
So, the loss for the entire dataset will be:

$$L_{per}(\mathbf{w}, \mathcal{D}) = \sum_i \max(0, -y_i \mathbf{w^T x_i})$$

In stochastic Gradient Descent, the weight update will be:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_w L_{per}(\mathbf{w}, (x_i, y_i))$$

The gradient of above loss function does not exist when $\mathbf{w^T x_i} = \mathbf{0}$.
So, we use subgradient functions.

> For a convex function $f(x)$, a vector $\mathbf{g}$ is a subgradient of f at $\mathbf{w_0}$ if the following condition holds for all $\mathbf{w}$:
>
> $$\forall w, f(\mathbf{w}) - f(\mathbf{w_0}) \geq g^T(\mathbf{w} - \mathbf{w_0})$$
>
> If $\mathbf{f}$ is convex and differentiable, $\nabla f(x)$ is a subgradient of $\mathbf{f}$ at $\mathbf{x}$. Also, subgradients are not necessarily strictly decreasing.

The perceptron loss function isn't differentiable at all points so we can use a subgradient in place of the gradient. The subgradient of the perceptron loss, $\mathbf{g}$ can be written as

$$g = \begin{cases} 0, & \text{if} \quad y\mathbf{w}^T\mathbf{x} > 0 \\ -y\mathbf{x}, & \text{if} \quad y\mathbf{w}^T\mathbf{x} < 0 \\ ky\mathbf{x}, & \text{if} \quad y\mathbf{w}^T\mathbf{x} = 0, k \in [-1, 0] \end{cases}$$

If we pick $k = -1$, then

$$g = \begin{cases} 0, & \text{if} \quad y\mathbf{w}^T\mathbf{x} > 0 \\ -y\mathbf{x}, & \text{if} \quad y\mathbf{w}^T\mathbf{x} \leq 0 \end{cases}$$

Thus, the SGD weight update rule would become

$$w \leftarrow w + y\mathbf{x} \text{ if } y\mathbf{w}^T\mathbf{x} \leq 0$$

Convergence bound of perceptron algorithm

Consider a linearly separable dataset $\mathcal{D}$, i.e., $\exists \mathbf{u}$ such that,

$$y = sgn(\mathbf{u^T x}) \quad \forall (\mathbf{x}, y) \in D$$

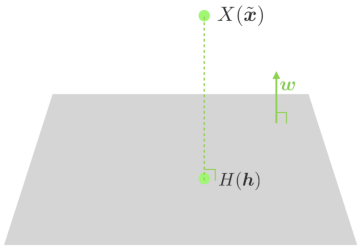
Without loss of generality, Let's assume:

- $\mathbf{u}$ is a unit vector, $\|\mathbf{u}\|_2 = 1$
- All $\mathbf{x_i} \in \mathcal{D}$ are scaled to lie within a Euclidean ball of unit radius, i.e. $\|\mathbf{x_i}\|_2 \leq 1$.

Def^{n} : **margin of separation** (γ) is defined as,

$$\gamma = \min_{x \in \mathcal{D}} |u^T x| \tag{1}$$

Distance between point and hyperplane



The equation of the hyperplane in the figure is $\mathbf{w^T x} + \mathbf{b} = 0$. We have to find the distance of point X from the hyperplane. Now consider a point H on the hyperplane such that vector $\tilde{\mathbf{x}} - \mathbf{h}$ is orthogonal to the plane. that is $\mathbf{w}$ is parallel to $\tilde{\mathbf{x}} - \mathbf{h}$. So,

$$\begin{aligned}
\tilde{\mathbf{x}} - \mathbf{h} &= k\mathbf{w} \\
\therefore \mathbf{h} &= \tilde{\mathbf{x}} - k\mathbf{w} \\
\therefore \mathbf{w^T}(\tilde{\mathbf{x}} - k\mathbf{w}) + \mathbf{b} &= 0 \\
\therefore \mathbf{w^T}\tilde{\mathbf{x}} + \mathbf{b} &= k||\mathbf{w}||_2^2 \\
\therefore k &= \frac{\mathbf{w^T}\tilde{\mathbf{x}} + \mathbf{b}}{||\mathbf{w}||_2^2}
\end{aligned}$$

Now, the distance between the hyperplane and point X is the same as the distance between point X and H. So, the distance between hyperplane $\mathbf{w^T x} + \mathbf{b}$ and point $\tilde{\mathbf{x}}$ is:

$$\begin{aligned}
||\tilde{\mathbf{x}} - \mathbf{h}||_2 &= ||k\mathbf{w}||_2^2 \\
&= |k| \cdot ||\mathbf{w}||_2^2 \\
&= \frac{|\mathbf{w^T}\tilde{\mathbf{x}} + \mathbf{b}|}{||\mathbf{w}||_2}
\end{aligned}$$

So, the distance between hyperplane $\mathbf{w^T x} + \mathbf{b}$ and point $\tilde{\mathbf{x}}$ is $\frac{|\mathbf{w^T}\tilde{\mathbf{x}}+\mathbf{b}|}{||\mathbf{w}||_2}$

So, the distance of any point $\tilde{x}$ from the given hyperplane $\mathbf{u^T x} = 0$ will be:

$$\frac{|\mathbf{u^T}\tilde{\mathbf{x}}|}{||\mathbf{u}||_2}$$

Here, $\mathbf{u}$ is a unit vector. So, the distance is $|\mathbf{u^T}\tilde{\mathbf{x}}|$.
So, γ is the minimum distance of a point $\mathbf{x} \in D$ from the hyperplane $\mathbf{u^T x} = 0$

Theorem 1. ***Novikoff's Theorem:*** *If there exist a unit vector* $\mathbf{u}$ *s.t.* $y\mathbf{u}^T\mathbf{x} \geq \gamma \quad \forall (x, y) \in D$ *then perceptron algorithm will make no more than* $\frac{1}{\gamma^2}$ *mistake on training dataset* $\mathcal{D}$ *when initialisation is* $\mathbf{w_0} = \mathbf{0}$.

That is, the number of mistakes made by the perceptron algorithm -

- **Does not** depend on the size of feature space
- **Does not** depend on the number of examples
- **Only** depends on the margin of separation

Proof. We will monitor the growth of the following quantities across weight updates

- $\mathbf{w}^T\mathbf{u}$
- $||\mathbf{w}||_2^2$

Why these quantities?

The desired weight vector is $\mathbf{u}$. We want our trained weight vector $\mathbf{w}$ to be more and more like $\mathbf{u}$. We want $\mathbf{w}$ to be more and more parallel to $\mathbf{u}$. So, we want the minimum angle between $\mathbf{w}$ and $\mathbf{u}$. So, we have to maximize the $\mathbf{w}^T\mathbf{u}$ value. Now value of $\mathbf{w}^T\mathbf{u}$ can increase due to two reasons,

- Value of $||\mathbf{w}||_2$ increases
- The angle between them decreases, i.e. $\mathbf{w}$ comes closer to $\mathbf{u}$

But, we want to minimize the angle, so we keep track of $||\mathbf{w}||_2$ to check that the angle between the vectors is decreasing.

First, we will see what happens to $\mathbf{w}^T\mathbf{u}$ after each iteration. Let the weight at i^{th} iteration be $\mathbf{w}_i$ and after update be $\mathbf{w}_{i+1}$. The difference between $\mathbf{w}_{i+1}^T\mathbf{u}$ and $\mathbf{w}_i^T\mathbf{u}$ is,

$$\begin{aligned} \mathbf{w}_{i+1}^T\mathbf{u} - \mathbf{w}_i^T\mathbf{u} &= (\mathbf{w}_i + yx)^T\mathbf{u} - \mathbf{w}_i^T\mathbf{u} \\ \mathbf{w}_{i+1}^T\mathbf{u} &= \mathbf{w}_i^T\mathbf{u} + yx^T\mathbf{u} \\ \mathbf{w}_{i+1}^T\mathbf{u} &\geq \mathbf{w}_i^T\mathbf{u} + \gamma \end{aligned}$$

Since, $\mathbf{w_0}$ is initialised to 0.

$$\boxed{\mathbf{w}_k^T\mathbf{u} \geq k\gamma} \tag{2}$$

Now, $||\mathbf{w}||_2^2$ at each weight update increases by at most 1.

$$
\begin{aligned}
\|\mathbf{w} + y\mathbf{x}\|^2 &= (\mathbf{w} + y\mathbf{x})^T(\mathbf{w} + y\mathbf{x}) \\
&= \|\mathbf{w}\|^2 + 2y\mathbf{w}^T\mathbf{x} + y^2\|\mathbf{x}\|^2 \\
&= \|\mathbf{w}\|^2 + 2y\mathbf{w}^T\mathbf{x} + \|\mathbf{x}\|^2 \\
&\leq \|\mathbf{w}\|^2 + \|\mathbf{x}\|^2 && \because y\mathbf{w}^T\mathbf{x} \leq 0 \\
&\leq \|\mathbf{w}\|^2 + 1 && \because \text{assumption that } \|\mathbf{x}\| \leq 1
\end{aligned}
$$

Hence, after $\mathbf{k}$ weight updates,

$$\|\mathbf{w}_k\|^2 \leq \|\mathbf{w}_0\|^2 + k$$

since $\mathbf{w}_0$ is initialised to 0.

$$\boxed{\|\mathbf{w}_k\|^2 \leq k} \tag{3}$$

so from (2) and (3) we get,

$$\sqrt{k} \geq \|\mathbf{w_k}\| \geq \mathbf{w_k^T u} \geq k\gamma$$

$$\boxed{k \leq \frac{1}{\gamma^2}}$$

□

Hence, the number of mistakes made by the perceptron algorithm is bounded. As the perceptron algorithm updates weight only when there is a mistake, the mistake bound implies that the convergence is guaranteed.

Why is k independent of the number of examples?

Suppose the mistake in predicting an example $(\mathbf{x}, y)$ caused a weight update. But the change in hyperplane won't just correctly classify this point $(\mathbf{x}, y)$. It can classify multiple points correctly. Hence, the total number of iterations for the perceptron to converge does not depend on the number of examples in the dataset.

Limitation of Perceptron Algorithm

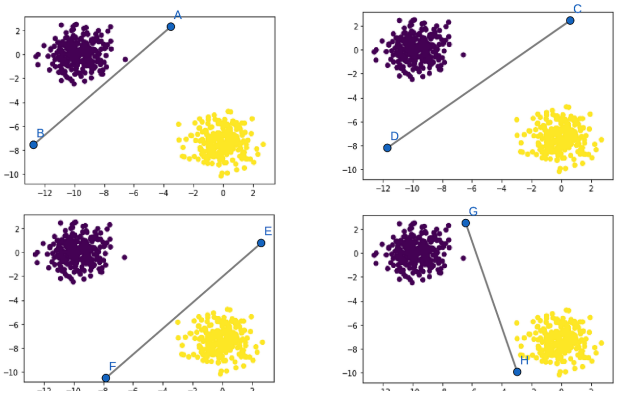


Figure 1: Possible Separation Hyperplanes

Consider Figure 1. All the hyperplanes AB, CD, EF, and GH classify data correctly. But, the hyperplanes AB, EF and GH on slight perturbation to the dataset will give incorrect predictions for examples that are closer to the hyperplane.
Hyperplane CD will give correct prediction even if there are perturbations because the closest point to the hyperplane is at the maximum distance for both groups. The problem with the perceptron algorithm is that it does not account for the margin of separation while choosing the hyperplane.
Objective: Maximize the margin of separation while correctly predicting all data points. Hence, the desirable hyperplane is the one whose margin of separation is higher for both groups while predicting all the training points correctly.
One of the possible solutions to this problem is a **Support Vector Machine**, which maximizes the margin while predicting correct values.

CS 337, Fall 2023
Support Vector Machine

Scribes: Arijit Saha, Atharv Kshirsagar, Vanapalli Raja Gopal
Ashwin Goyal, Patil Vipul Sudhir, Hrishikesh Jedhe Deshmukh
(***Equal contribution by all scribes***)
Edited by: Dhiraj Kumar Sah

September 7, 2023

Disclaimer. Please note this document has not received the usual scrutiny that formal publications enjoy. This may be distributed outside this class only with the permission of the instructor.

Recap

For linearly separable data, the perceptron finds a hyperplane which separates the given data into 2 classes. Note that the perceptron algorithm finds a hyperplane among all possible hyperplanes which separate the data. Different algorithm runs will not lead to the same hyperplane as output.

The figure below shows an example of hyperplanes that separate the linear separable data

different initial weight vectors won't lead to the same hyperplane classifying the data points.

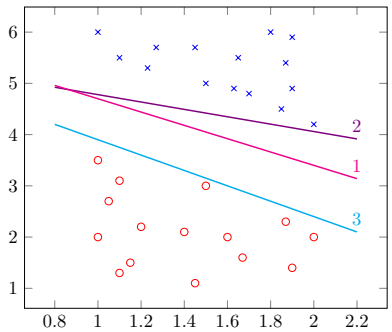


Figure 1: Hyperplanes found by perceptron algorithm

Question: Which is the **best** hyperplane? What do we mean by **best**? How do we find it?
Ans: In Figure 1, we would like our hyperplane to be 1 instead of 2 or 3. The reason is hyperplanes 2 and 3 are extremely close to either of the classes creating a possibility of mis-classification for new test samples. So we define **best** hyperplane to be the hyperplane which **maximizes** the **margin**, that is the distance of the closest point in either class from this hyperplane should be maximized while correctly classifying/predicting all the training points. We can find that hyperplane using **Support Vector Machines** (SVM).

finding the classifier while simultaneously maximizing the margin.

Binary Classification using Support Vector Machine (SVM)

At its essence, an SVM functions as a linear classifier, but it distinguishes itself by going beyond merely finding a separating hyperplane for data. It takes the additional step of seeking the optimal hyperplane, which doesn't just divide the data accurately but also maximizes the **margin***.

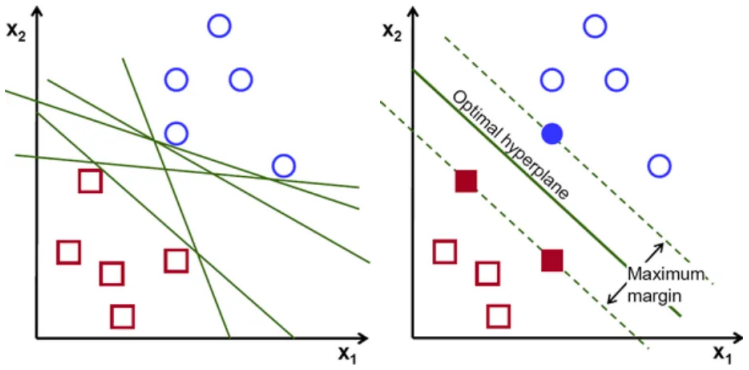


Figure 2: Different Hyperplane Seperators

In Figure 2, while all the lines in the left image successfully classify the data points, the line in the right image stands out as the optimal classifier because it boasts the widest margin, ensuring better robustness and generalization.

We know that,

$$\text{Margin } \gamma(w,b) = \min_i \frac{|w^T x_i + b|}{\|w\|}$$ distance of any point from any plane.

So, our optimization problem becomes,

$$\max_{(w,b)} \min_i \frac{2|w^T x_i + b|}{\|w\|}$$ 2 side (1)

$$\text{such that, } \forall i \; y_i(w^T x_i + b) > 0$$

We can scale w and b to make,

$$\min_i |w^T X_i + b| = 1$$ the point which is minimum, has the distance of 1 (2)

From equations (1) and (2), We can write,

$$\max_{(w,b)} \frac{2}{\|w\|}$$

$$\text{or, } \min_{(w,b)} \frac{\|w\|^2}{2}$$ Final optimization problem.

$$\text{such that, } \forall i \; y_i(w^T x_i + b) \geq 1 \quad (3)$$

Equation (3) is our Final Optimization Problem.

Hard-Margin SVM

A Hard Margin Support Vector Machine (SVM) imposes a stringent condition where no data point is allowed to exist within the margin, and it must accurately classify all training data. This approach is most effective when dealing with linearly separable data, with a distinct and unfaltering boundary between the different classes.

You might wonder why our final optimisation hasn't included the constraint $\min_i |w^T X_i + b| = 1$. We don't need this constraint separately because it's already accounted for by the requirement $\forall i \; y_i(w^T x_i + b) \geq 1$. This latter condition ensures that the margin between the data points and the decision boundary is maximized and, as a result, incorporates the concept of $\min_i |w^T X_i + b| = 1$.

The blue line is the decision boundary in Figure 3. The orange and green lines pass through support vectors. Here, we are trying to maximize the margin, i.e. the distance between the orange and green lines.

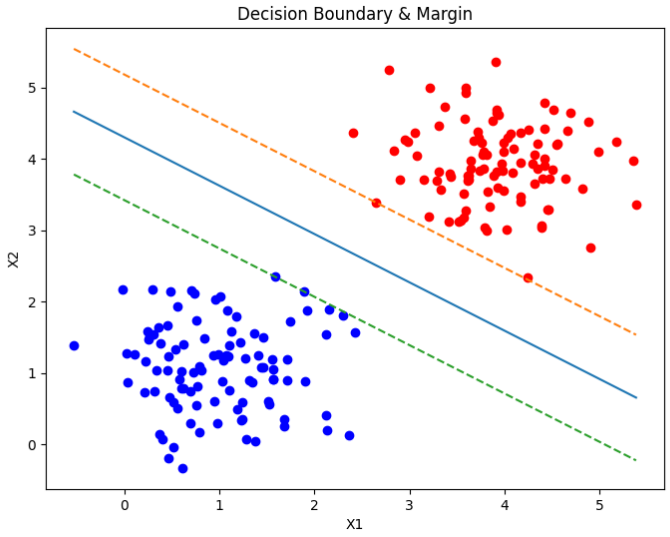


Figure 3: Example of Hard Margin SVM

Limitations

- Hard Margin SVMs exhibit high sensitivity to outliers or incorrectly labelled data points. Even a solitary outlier situated on the incorrect side of the decision boundary or within the margin can severely disrupt the model's ability to identify a valid hyperplane.
- Furthermore, Hard Margin SVMs face limitations when the data is not linearly separable, rendering them incapable of determining appropriate decision boundaries.
- Gradient descent is used to find the optimal value as there is no closed-form solution for (w, b).

Soft-Margin SVM

A Soft Margin Support Vector Machine (SVM) represents an advancement over the conventional "hard margin" SVM. Its purpose is to accommodate datasets lacking perfect linear separability and may include

some noise or data point overlap. In scenarios like these, imposing a stringent separation criterion, as in the case of the hard margin SVM, can result in overfitting or a failure to identify a suitable hyperplane.

Let there be some points which are misclassified.

$$\text{i.e. } \exists i \; y_i(w^T x_i + b) < 1$$

$$\text{or, } \exists i \; (1 - y_i(w^T x_i + b) > 0)$$

So, we will use $\max(1 - y_i(w^T x_i + b), 0)$ as penalty to track ***how badly this point is mislabelled***. This is also known as **Hinge Loss**.

So, our new optimization problem is to minimize

$$\min_{(w,b)} \frac{1}{2}\|w\|^2 + C\sum_i \max(1 - y_i(w^T x_i + b), 0)$$

Here C is the regularization factor and $\sum_i \max(1 - y_i(w^T x_i + b), 0)$ is regularization loss. C helps strike a balance between large margins and small training errors. To make this more formal, we introduce new variables $\mathcal{E}_i$ (referred to as **slack variables**), which will denote the penalty corresponding to each training instance.

So, Our Final Optimization Problem for Soft-Margin SVM is,

$$\min_{(w,b)} \left(\frac{\|w\|^2}{2} + C\sum_i \mathcal{E}_i\right)$$

$$\text{such that, } \forall i \; y_i(w^T x_i + b) \geq 1 - \mathcal{E}_i \text{ and } \forall i \; \mathcal{E}_i \geq 0$$

SVM Characteristics

- Soft margin SVMs offer increased robustness to outliers and noisy data, making them suitable for datasets with imperfect linear separability or class overlap. Consequently, they tend to provide improved generalization to new, unseen data compared to Hard-Margin SVMs.
- SVM can be used to classify non-linear data using a kernel. This will be elaborated further in the next lecture.
- A smaller value of C encourages a wider margin, allowing for more classification errors but potentially better generalization to unseen data.
- A larger C places a higher penalty on misclassifications, resulting in a narrower margin and potentially overfitting the training data.
- All the points for which $y_i(w^T x_i + b) = 1$ are called **support vectors** because these points determine the decision boundary in SVMs. Removing any of these points will alter the learned SVM decision boundary.

CS 337, Fall 2023
SVMs and Kernels

Scribes: Divakar Sai Savaram, Kanakala Dittu Akanksh, Shikhar Parmar
(***Equal contribution by all scribes***)
Edited by: Govind Saju

September 11, 2023

Disclaimer. Please note this document has not received the usual scrutiny that formal publications enjoy. This may be distributed outside this class only with the permission of the instructor.

Recap

We are currently studying about the support vector machines. In this, the goal is to find the best decision boundary. The aim is to maximize the margin while classifying the points. Here margin refers to the distance of the closest points across both classes. The following formula represents the margin.

$$r(w, b) = \min_{i=1 to n} \frac{(\mathbf{w}^T \mathbf{x}_i + b)}{\|w\|}$$

Now the SVM optimization problem to solve is:

$$\max_{w \in R^n} 2r(w, b)$$

$$s.t \forall i,\ y_i(w^T x_i + b) \geq 0$$

the above optimization problem can be further reduced to the following problem:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

$$s.t \forall i,\ y_i(w^T x_i + b) \geq 1$$

The given problem represents Hard Margin SVM optimization. In cases where margin constraints are violated, we aim to minimize the hinge loss while maximizing the margin, resulting in a modified optimization problem which is as follows:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C.\Sigma_i \max\{0, 1 - y_i(w^T x_i + b)\}$$

The above problem indicates the soft margin SVM optimization. Here hyper parameter C indicates the penalty or regularization factor.

1 SVM Optimization Problem

The goal of the SVM optimization is to find the best decision boundary for classification as seen above.The generic contained optimization problem is as follows:

$$min_w f(w)$$

$$\text{s.t. } g_1(w) \leq 0, g_2(w) \leq 0, \ldots, g_n(w) \leq 0$$

where the function f(w)=$\frac{1}{2}$ $\|w\|^2$ and the function $g_i(w) = 1 - y_i(w^T x_i + b)$. Now the above form can be written using the Lagrangian multipliers in the following way:

$$L(w, \alpha) = f(w) + \sum_{i=1}^{n} \alpha_i g_i(w)$$

where α_i are the Lagrangian multipliers. An equivalent formulation can be expressed using the above formula provided, such that it represents the SVM optimization problem. The equivalent form is as follows:

$$min_w max_{\alpha \geq 0} L(w, \alpha)$$

Here if any of the $g_i(w) \geq 0$, then the corresponding $\alpha_i \rightarrow \infty$ which makes the above expression tending to ∞. Whereas if all the $g_i(w)$ satisfy the given conditions, then all the corresponding α_i will be 0 making L(w,α) maximum which in turn reduces to min_wf(w) (equivalent to SVM optimization).
Let's consider the **Dual form** of the above optimization problem.

$$max_{\alpha \geq 0} min_w L(w, \alpha)$$

The above is the dual form of the primal. In general, the following inequality always holds:

$$max_{\alpha \geq 0} min_w L(w, \alpha) \leq min_w max_{\alpha \geq 0} L(w, \alpha)$$

but in the case of SVM the equality is maintained.

$$max_{\alpha \geq 0} min_w L(w, \alpha) = min_w max_{\alpha \geq 0} L(w, \alpha)$$

Now from the above equation we can find the solution for the dual by solving the primal. The Lagrangian for our optimization problem from above is:

$$L(w, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(w^T x_i + b) - 1] \tag{1.1}$$

Let's find the dual form of the problem. To do this we need to first minimize L(w,b,α) with respect to w and b (for fixed α), which we will do by setting the derivatives of L with respect to w and b to zero. We have:

$$\nabla_w L(w, \alpha) = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0$$

This implies that

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{1.2}$$

As for the derivative with respect to b, we obtain

$$\frac{dL(w, \alpha)}{db} = \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{1.3}$$

If we take the definition of w in Equation 1.2 and put that back into the Lagrangian (Equation 1.1) and simplify, we get

$$L(w, \alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (x_i)^T x_j - b \sum_{i}^{n} \alpha_i y_i$$

But from the Equation 1.3 the last term must be zero, so we get

$$L(w, \alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (x_i)^T x_j.$$

We got the above equation by minimizing L with respect to w and b. Putting this together with the constraints $\alpha_i \geq 0$, we obtain the following dual optimization problem:

$$max_{\alpha_i \geq 0} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (x_i)^T x_j.$$

$$\text{s.t. } \alpha_i \geq 0, i = 1, 2, ..., n$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

The formulation provided represents the dual form of the hard margin SVM. In the dual form of the soft margin SVM, the only change occurs is on the constraints of α_i, which are modified to meet the condition $0 \leq \alpha_i \leq C$. All other aspects of the problem remain unchanged.

2 Observations based on the Dual form

1) By constructing the dual form we found that:

$$w^* = \sum_{i=1}^{n} \alpha_i y_i x_i$$

2) Under our above assumptions,there must exist w^*,α^* , so that w^* is the solution to the primal problem,α^* is the solution to the dual problem. Moreover w^* and α^* satisfy the **KKT(Karush-Kuhn-Tucker)** constraints which are as follows:

$$\nabla_w L(w^*, \alpha^*) = 0$$

$$\alpha_i^* g_i(w^*) = 0, i = 1, 2, ..., n \tag{2.1}$$

$$\alpha_i^* \geq 0, i = 1, 2, ..., n$$

The constraint of interest is Equation 2.1 which is also called as **KKT dual complimentarity condition**. This condition asserts that for all those points which have $g_i(w^*)$ less than 0, the corresponding α_i^* will be 0. Conversely, for the points where the corresponding $g_i(w^*)$ equals 0, the corresponding α_i^* will be greater than 0. This implies that the value of w^* only depends solely on the values of these α_i^* values. For the points of interest($\alpha_i^* > 0$) the equation is as follows:

$$g_i(w^*) = 0$$

$$y_i(w^T x_i + b) = 1$$

These are the points which lie on the margin.Therefore these are Support vectors for the SVM. This basically shows that the value of w^* depends only on the **support vectors**.

3) The dual form expression operates only on the inner product of training instance pairs. For predicting the test data point:

$$y_{predict} = sign(w^{*T}x + b^*)$$

By substituting the value of w^* which we got in the Equation 1.2. We get:

$$y_{predict} = sign(\sum_{i}^{n} \alpha_i^* y_i x_i^T x + b^*)$$

The above prediction of test data solely depends on the inner products of support vectors and test data. This will be useful when we talk about kernels next.

3 Non-linear decision boundary and Kernels

In the case of estimating non-linear decision boundaries, we increase the dimensions of the data points by using a basis function $\phi(x)$ which replaces the above equation as:

$$y_{predict} = sign(\sum_{i}^{n} \alpha_i^* y_i \phi(x_i)^T \phi(x) + b^*)$$

The above operation often encounters significant computational complexity challenges. This occurs when we select a high dimensional function $\phi(x)$ to make predictions. In such cases, we need to compute $\phi(x_i)$ for each data point x_i in our dataset. Furthermore, whenever α_i is not equal to 0(support vectors), we must calculate the dot product between $\phi(x_i)$ and $\phi(x)$ which becomes computationally very expensive. To address this issue, we can employ a technique called **"Kernels"**.

3.1 Kernel Trick

Consider a function $K : R^d \times R^d \rightarrow R$ such that $K(x, y) = \phi(x)^T \phi(y)$. This is called a Kernel function. What this kernel function allows us to do is that by using them, we can directly calculate the inner product $\phi(x)^T \phi(y)$ without having to individually compute $\phi(x)$. Kernel functions also provide an added advantage of computing this inner product for basis functions which increase the dimensions of feature space to infinity.

Consider the following example for instances $x = (x_1, x_2)$ and $y = (y_1, y_2)$. We define $\phi(x)$ as

$$\phi(x) = \phi((x_1, x_2)) = \begin{bmatrix} x_1^2 & \sqrt{2}x_1x_2 & x_2^2 \end{bmatrix}^T$$

and the kernel function as

$$K(x, y) = (X^T Y)^2$$

Our claim is that $K(x, y) = \phi(x)^T \phi(y)$. The proof is as follows:

$$\begin{aligned} K(x, y) &= (X^T Y)^2 \\ &= (\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} y_1 & y_2 \end{bmatrix}^T)^2 \\ &= (x_1x_2 + y_1y_2)^2 \\ \phi(x)^T \phi(y) &= \begin{bmatrix} x_1^2 & \sqrt{2}x_1x_2 & x_2^2 \end{bmatrix} \begin{bmatrix} y_1^2 & \sqrt{2}y_1y_2 & y_2^2 \end{bmatrix}^T \\ &= x_1^2y_1^2 + 2x_1x_2y_1y_2 + x_2^2y_2^2 \\ &= (x_1x_2 + y_1y_2)^2 \end{aligned}$$

Hence, we see that by choosing an appropriate kernel function, we can avoid the expensive computation for $\phi(x)$.

3.2 Choosing a Kernel function

A kernel $K(x, y)$ is a valid kernel function if it corresponds to an underlying $\phi(x)$ (projecting x to a high, possibly infinite dimensional space) i.e. $K(x, y) = \phi(x)^T \phi(y)$. One can use the following methods to create a kernel function:

1. **Mercers's Theorem** : Let $K : R^d \times R^d \rightarrow R$. For K to be a valid kernel, the necessary and sufficient condition is that for any finite set $\{x_1, x_2 ....x_n\}$, the kernel matrix is symmetric and positive semi-definite(i.e. $v^T M v > 0 \; \forall v \in R^n$). The kernel matrix is defined as an $n \times n$ matrix M such that

$$M[i][j] = K(x_i, x_j)$$

2. Using properties of kernel over existing kernel functions. If $K_1(x, y)$ and $K_2(x, y)$ are two valid kernel functions then

$$K(x, y) = K_1(x, y) + K_2(x, y)$$
$$K(x, y) = K_1(x, y) * K_2(x, y)$$
$$K(x, y) = c * K_1(x, y) \;\; where \;\; c > 0$$

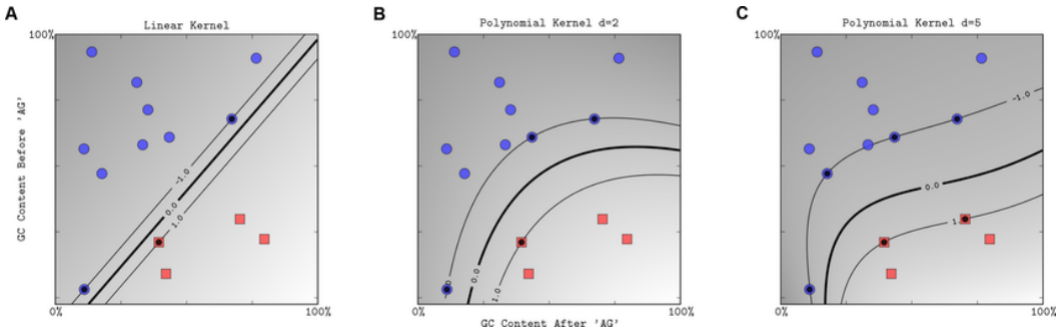
are also valid Kernel functions.

3.3 Examples of Valid Kernel Functions

1. **Polynomial Kernel**
The general form of the Polynomial Kernel Function is

$$K(x, y) = (X^T Y + c)$$

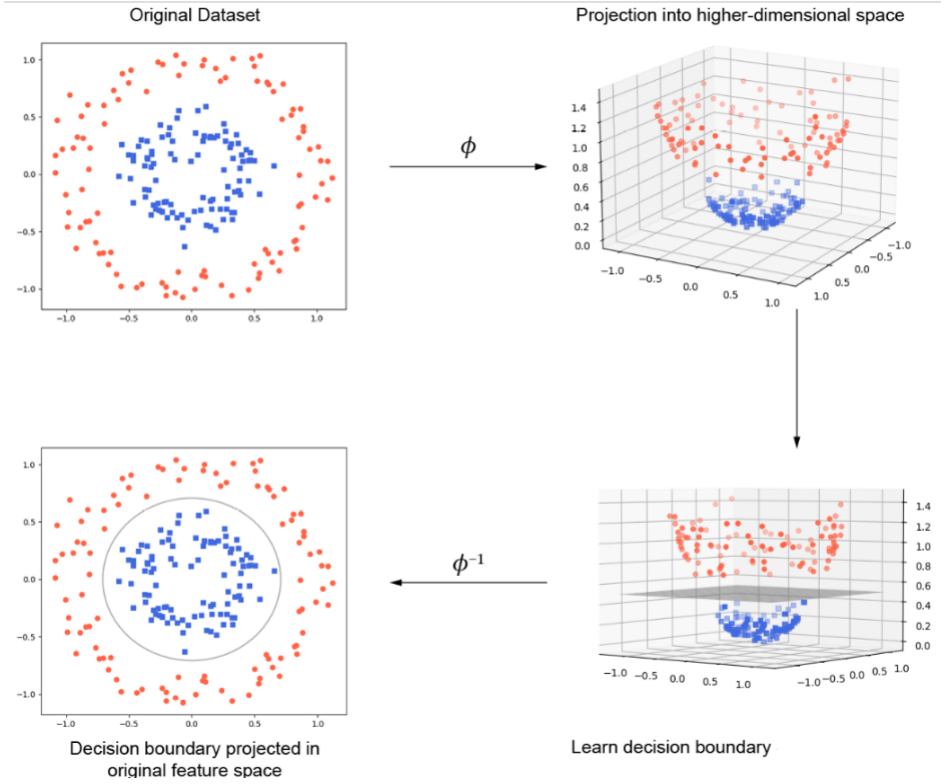
Here, all the terms will have order $\leq$ d. If we set $c = 0$, then the function becomes $K(x, y) = (X^T Y)$ and each term will now have order d.



2. **Radial Basis function(RBF) Kernel (Gaussian kernel)**
The radial basis function is given by-

$$K(x, y) = exp(\frac{-\|x - y\|_2}{\sigma^2})$$

Here, if $\sigma << 1$ then $K(x_i, x_j) \rightarrow 0 \;\; \forall i \neq j$ and $K(x_i, x_i) = 1 \;\; \forall i$. Therefore, our kernel matrix tends to an Identity matrix. The other extreme case is as $\sigma \rightarrow \infty$, $K(x_i, x_j) \rightarrow 1 \;\; \forall i, j$ and therefore, our kernel matrix tends to a unit matrix. The following is a visualisation of the RBF in action-


Original Dataset
Projection into higher-dimensional space
ϕ
ϕ^{-1}
Decision boundary projected in original feature space
Learn decision boundary