# CSCI3230 Fundamentals of Artificial Intelligence

Ryan Chan

December 4, 2025

**Abstract**

This is a note for **CSCI3230 Fundamentals of Artificial Intelligence**.
Contents are adapted from the lecture notes of CSCI3230, prepared by Qi Dou, as well as some online resources.
This note is intended solely as a study aid. While I have done my best to ensure the accuracy of the content, I do not take responsibility for any errors or inaccuracies that may be present. Please use the material thoughtfully and at your own discretion.
If you believe any part of this content infringes on copyright, feel free to contact me, and I will address it promptly.
Mistakes might be found. So please feel free to point out any mistakes.

# Contents

# Chapter 1

# Basis of AI

## 1.1 Introduction

Artificial intelligence (AI) is intelligence demonstrated by machines, as opposed to the natural intelligence displayed by humans or animals. [1]

It can be distinguished into **Artificial Narrow Intelligence** and **Artificial General Intelligence**. The latter refers to the ability of an intelligent agent to understand or learn any intellectual task that a human being can. Machine learning, as often mentioned, is a branch of AI, while deep learning is a specific subset of machine learning.

Before delving into details, we consider how humans learn, since AI is basically simulating the way people learn.

For humans, we acquire or learn some skills by accumulating experience from observations. This is basically the same for computers, except that the observations are now in the form of data, which can vary from text documents to sound recordings.

The skill of a machine can be illustrated by prediction, which is basically what AI does—makes predictions. Then, how can data be learned? There are learning algorithms that train the machine on how to make predictions; we construct a model corresponding to such an algorithm and then use the data to train the model.

Think of it this way: a learning algorithm is the functionality of the brain, where the model is the container, i.e., the brain itself. We have observations (training data), and then, using these observations, we make predictions based on new information (test data) — for example, the answer to a math question, the answer to someone else's question, etc.

Then, you observe that there are two sets of data: one is the training set and the other is the test set. Their usage is just like their names suggest. For example, for labeled data, we might randomly split them into 80% training data and 20% test data. Notice that a larger test set will give a more accurate assessment of performance.

We sometimes also use a validation data set, which is a set of examples used to tune the *hyperparameters* of the model.

In short, we have an original labeled data set, and we either split it into two sets, i.e., training and test sets, or we split it into three sets with a validation set used to tune the model.

If there is a small-scale dataset, we can use **cross-validation**. We shuffle the dataset randomly and split it into $k$ groups. For each unique group, take that group as a hold-out or test dataset, and use the remaining groups as the training dataset. Then, summarize the performance of the model using all the evaluation scores from each fold.

With all these methods, the question remains: how do we evaluate the performance of a model given a dataset?

[1] Here are some funny facts I would like to share, which are not necessary. You can directly go to 1.2.

## 1.2 Model Performance

To evaluate the model performance, we can compare it with previous models if they exist. There are also some metrics that can be used, which are based on the confusion matrix.

Assume we have a binary classifier that can classify the data into positive and negative samples. Then we have

| | | **Actual** | |
|---|---|---|---|
| | | Positive | Negative |
| **Predicted** | Positive | True Positive (TP) | False Positive (FP) |
| | Negative | False Negative (FN) | True Negative (TN) |

Table 1.1: Standard Binary Confusion Matrix Layout

Here TP and TN show correct prediction, whereas FP and FN show incorrect prediction.

Below might provide a more intuitive illustration.[2]

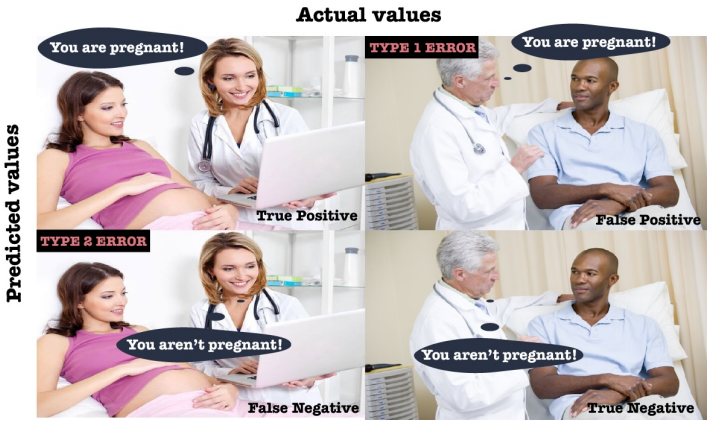


Figure 1.1: Pregnancy Confusion Matrix

We define **Accuracy** as the fraction of correct prediction,

$$\frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{FP}+\mathrm{TN}+\mathrm{FN}}.$$

**Precision** is the accuracy of the positive predictions:

$$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}.$$

Notice that high precision means low 'false alarm rate'.

Lastly, **recall** (or **sensitivity**) is the accuracy for the positive class,

$$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}.$$

High recall means low possibility of missing many positives.

We use a confusion matrix to visualize the performance of an algorithm. Each row of the matrix represents the instances in an actual class, while each column represents the instances in a predicted class, or vice versa.

[2] https://codefinity.com/courses/v2/b71ff7ac-3932-41d2-a4d8-060e24b00129/6b1d35ef-cd16-405c-8202-ef0f3d8e10c5/be441b8d-e533-4c4d-8e34-817ea61b6fa0

## 1.3 Types of Learning

There are three types of learning.

**Supervised Learning:** data is labeled such that the machine can learn from the characters. This is used to predict outcomes or the future. It can be seen as a function that maps an input to an output based on example input–output pairs.

**Unsupervised Learning:** data is in the form of unlabeled samples, where the machine needs to learn on its own and find hidden structure.

**Reinforcement Learning:** this can be considered as a reward system, where it deals with the way intelligent agents take actions in order to maximize their cumulative reward.

# Chapter 2

# Regression

Regression is a statistical process for estimating the relation between dependent and independent variables. Consider it as a program, where the input is independent variables and the output is dependent variables.

## 2.1 Linear Regression

Linear regression is a kind of regression model, which hypothesizes that the relation between dependent variables and independent variables is linear. Mathematically, a function $f(x)$ is linear iff. $f(u+v) = f(u) + f(v)$ and $f(cu) = cf(u)$.

### 2.1.1 Formulation

Consider intelligent machines as functions with input and output. Then we have

$$f : X \to Y$$

where $X \subset \mathbb{R}^n$ is the domain of input and $Y \subset \mathbb{R}$ is the domain of output. The goal of the learning algorithm is to take the given data as training examples, then try to find a general mapping $\hat{f} : X \to Y$ such that it is close to the true $f$, i.e. $\hat{f} \approx f$.

As we plot the given data on a graph, if the data points are distributed along a straight line, we can assume the relation between input $X$ and output $Y$ is linear for simplification.

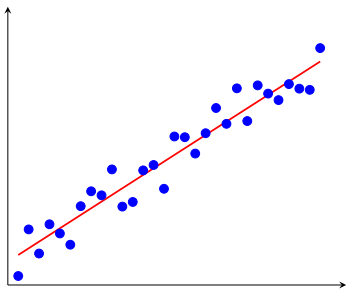

Figure 2.1: Linear Regression

For a univariate linear function in a 1-dimensional feature space, we have

$$\hat{y} = \hat{f}_{\theta_0,\theta_1}(x) = \theta_0 + \theta_1 x.$$

If we can find the optimal $\theta_0$ and $\theta_1$, i.e., the relation parameters, then we can predict the output with higher precision.

When there are more than one factors, we consider an $n$-dimensional feature space. The multivariate linear function is

$$\hat{y} = \hat{f}_{\theta_0,\theta_1,\ldots,\theta_n}(x_1,\ldots,x_n) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n.$$

We can also write it in vector form as

$$\hat{y} = \hat{f}_{\theta_0,\Theta}(X) = X^T\Theta + \theta_0,$$

where

$$\Theta = \begin{pmatrix} \theta_1 \\ \vdots \\ \theta_n \end{pmatrix}, \quad X = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}.$$

**Remark.** $\theta_0$ and $\Theta$ are parameters that should be learned from the training data.

There might be more than one sample being studied. Suppose we have $m$ samples, and

$$X^{(1)} = \begin{pmatrix} x_1^{(1)} \\ \vdots \\ x_n^{(1)} \end{pmatrix}, \quad X^{(2)} = \begin{pmatrix} x_1^{(2)} \\ \vdots \\ x_n^{(2)} \end{pmatrix}, \ldots, X^{(m)} = \begin{pmatrix} x_1^{(m)} \\ \vdots \\ x_n^{(m)} \end{pmatrix},$$

with respective labels $y^{(1)}, y^{(2)}, \ldots, y^{(m)}$.

We denote $\mathbf{X} \in \mathbb{R}^{m\times n}$ as the data matrix, where rows represent samples and columns represent features:

$$\mathbf{X} = \begin{pmatrix} {X^{(1)}}^T \\ \vdots \\ {X^{(m)}}^T \end{pmatrix} = \begin{pmatrix} x_1^{(1)} & \cdots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(m)} & \cdots & x_n^{(m)} \end{pmatrix}.$$

Then we have a more general function for linear regression with $n$ features and $m$ samples:

$$\hat{Y} = \hat{f}_{\theta_0,\Theta}(\mathbf{X}) = \mathbf{X}\Theta + \theta_0,$$

where

$$\Theta = \begin{pmatrix} \theta_1 \\ \vdots \\ \theta_n \end{pmatrix}, \quad \hat{Y} = \begin{pmatrix} \hat{y}^{(1)} \\ \vdots \\ \hat{y}^{(m)} \end{pmatrix}.$$

**Remark.** For $\mathbf{X}\Theta$, consider it as combining the features for each sample:

$$\mathbf{X}\Theta = \theta_1 \begin{pmatrix} x_1^{(1)} \\ \vdots \\ x_1^{(m)} \end{pmatrix} + \theta_2 \begin{pmatrix} x_2^{(1)} \\ \vdots \\ x_2^{(m)} \end{pmatrix} + \cdots + \theta_n \begin{pmatrix} x_n^{(1)} \\ \vdots \\ x_n^{(m)} \end{pmatrix}.$$

For simplification, we define

$$\mathbf{X} = \begin{pmatrix} 1 & x_1^{(1)} & \cdots & x_n^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & \cdots & x_n^{(m)} \end{pmatrix}, \quad \Theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{pmatrix},$$

then

$$\hat{Y} = \hat{f}_{\Theta}(\mathbf{X}) = \mathbf{X}\Theta.$$

**Note** (Geometry of Linear Regression). Data points $\{(x_1^{(1)}, \cdots, x_n^{(1)}, y^{(1)})\}, \ldots, \{(x_1^{(m)}, \cdots, x_n^{(m)}, y^{(m)})\}$ occupy a $(n+1)$-dimensional space. The fitted line is actually an $n$-dimensional hyperplane in this space.

Since all data points are supposed to be sampled from the same distribution, once we find the optimal $\Theta$, we obtain $\hat{f}_\Theta \approx f$. The next question is: how do we find the optimal $\Theta$? We first focus on fitting the training data.

### 2.1.2 Optimization

Intuitively, we need to measure and minimize the distance between the estimated values $\hat{f}_\Theta(X^{(i)})$ and the true training labels $y^{(i)}$.

We denote $J(\Theta)$ as the objective function (or cost function) to measure this distance. The goal is to find $\Theta$ such that $J(\Theta)$ is minimized.

We define $J(\Theta)$ as the **Residual Sum of Squares (RSS)**:

$$J(\Theta) = \sum_{i=1}^{m} (\hat{f}_\Theta(X^{(i)}) - y^{(i)})^2 = \|\hat{f}_\Theta(\mathbf{X}) - Y\|_2^2.$$

The method that approximates the solution by minimizing the RSS is called **ordinary least squares**.

**Remark.** The *residual* is the difference between the estimated value and the corresponding training label.

Consider the univariate linear regression problem. Its residual sum of squares (RSS) is

$$J(\theta_0, \theta_1) = \sum_{i=1}^{m} (\hat{f}_{\theta_0,\theta_1}(x^{(i)}) - y^{(i)})^2 = \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2.$$

Then we can rewrite $J(\theta_0, \theta_1)$ as

$$\begin{aligned} J(\theta_0, \theta_1) = \sum_{i=1}^{m} (x^{(i)} - \overline{x})^2 \left( \theta_1 - \frac{\sum_{i=1}^{m} (x^{(i)} - \overline{x})(y^{(i)} - \overline{y})}{\sum_{i=1}^{m} (x^{(i)} - \overline{x})^2} \right)^2 + m\big(\theta_0 - (\overline{y} - \theta_1 \overline{x})\big)^2 \\ + \sum_{i=1}^{m} (y^{(i)} - \overline{y})^2 - \frac{\left( \sum_{i=1}^{m} (x^{(i)} - \overline{x})(y^{(i)} - \overline{y}) \right)^2}{\sum_{i=1}^{m} (x^{(i)} - \overline{x})^2}. \end{aligned}$$

**Remark.** Only the first two terms depend on $\theta_0$ and $\theta_1$. Therefore, to minimize $J(\theta_0, \theta_1)$, it suffices to set these two terms to zero.

Thus, we solve

$$\begin{cases} 0 &= \sum_{i=1}^{m} (x^{(i)} - \overline{x})^2 \left( \theta_1 - \frac{\sum_{i=1}^{m} (x^{(i)} - \overline{x})(y^{(i)} - \overline{y})}{\sum_{i=1}^{m} (x^{(i)} - \overline{x})^2} \right)^2, \\ 0 &= m\big(\theta_0 - (\overline{y} - \theta_1 \overline{x})\big)^2, \end{cases}$$

which gives the closed-form solution

$$\begin{cases} \theta_1 &= \frac{\sum_{i=1}^{m} (x^{(i)} - \overline{x})(y^{(i)} - \overline{y})}{\sum_{i=1}^{m} (x^{(i)} - \overline{x})^2}, \\ \theta_0 &= \overline{y} - \theta_1 \overline{x}. \end{cases}$$

Usually, we use optimization methods to find $\Theta^\star$ that minimizes the objective function. Optimization is the process of selecting the best element with regard to some criterion. We can write it as

$$\arg\min_x g(x) \quad \text{s.t. some conditions here,}$$

where $g(x)$ is the target to minimize and $x$ is the element to search.

For ordinary linear regression, we can formulate the optimization problem as

$$\hat{\Theta} = \Theta^\star = \arg\min_{\Theta} J(\Theta).$$

Since the RSS function is convex, i.e., the global minimum is also a local minimum, we can find the global minimum by solving

$$J'(\Theta^\star) = 0 \quad \text{and} \quad J''(\Theta^\star) > 0.$$

We have

$$\begin{aligned} J(\Theta) &= \|\hat{f}_\Theta(\mathbf{X}) - Y\|_2^2 = (\mathbf{X}\Theta - Y)^T(\mathbf{X}\Theta - Y) \\ &= \Theta^T\mathbf{X}^T\mathbf{X}\Theta - Y^T\mathbf{X}\Theta - \Theta^T\mathbf{X}^TY - Y^TY. \end{aligned}$$

Then, the derivatives are

$$\begin{aligned} \frac{\partial J(\Theta)}{\partial \Theta} &= 2\mathbf{X}^T\mathbf{X}\Theta - \mathbf{X}^TY - \mathbf{X}^TY \\ &= 2\mathbf{X}^T(\mathbf{X}\Theta - Y) = 0, \\ \frac{\partial^2 J(\Theta)}{\partial \Theta^2} &= 2\mathbf{X}^T\mathbf{X} > 0. \end{aligned}$$

Finally, we obtain the closed-form solution

$$\mathbf{X}^T\mathbf{X}\Theta = \mathbf{X}^TY \quad \Longrightarrow \quad \hat{\Theta} = \Theta^\star = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^TY.$$

> **Intuition.** All in all, the goal is to find the best $\Theta$, i.e., the parameters, since we want the model to capture most of the data and provide more precise predictions. For the case with only one feature, the derived formula is sufficient. However, for multiple features and samples, the calculation becomes cumbersome.
>
> Thus, we consider the RSS function in another way. Since it is convex by nature, we can use its derivative to find the optimal $\Theta$ efficiently.

### 2.1.3 Shrinkage Methods

There are noises from observation. Therefore, we have $y = f(X) + \epsilon$, where $\mathbb{E}(\epsilon) = 0$ and $\text{Var}(\epsilon) = \sigma^2$. Noise could lead to error, so for any fixed $X$ with label $y$ the expected prediction error at $X$ is

$$\text{EPE}(X) = \mathbb{E}((y - \hat{f}(X))^2) = \text{Bias}(\hat{f}(X))^2 + \text{Var}(\hat{f}(X)) + \sigma^2$$

where

$$\text{Bias}(\hat{f}(X)) = \mathbb{E}(\hat{f}(X)) - f(X) \quad \text{Var}(\hat{f}(X)) = \mathbb{E}((\hat{f}(X) - \mathbb{E}[\hat{f}(X)])^2)$$

The expected prediction error is composed of bias, variance, and irreducible error. We can achieve a considerable EPE if both bias and variance are minimized simultaneously (see Figure 2.2[1]).

As we repeat the training on randomly sampled data, if the model perfectly fits the training data, the prediction bias is low while variance is very high, since the model varies among different training datasets. Yet, if the model is constant among different training datasets, the prediction variance will be zero but with high bias. This means that in general, low variance causes high bias, and vice versa.

Thus, we need to make a trade-off between minimizing bias and variance.

[1] http://scott.fortmann-roe.com/docs/BiasVariance.html

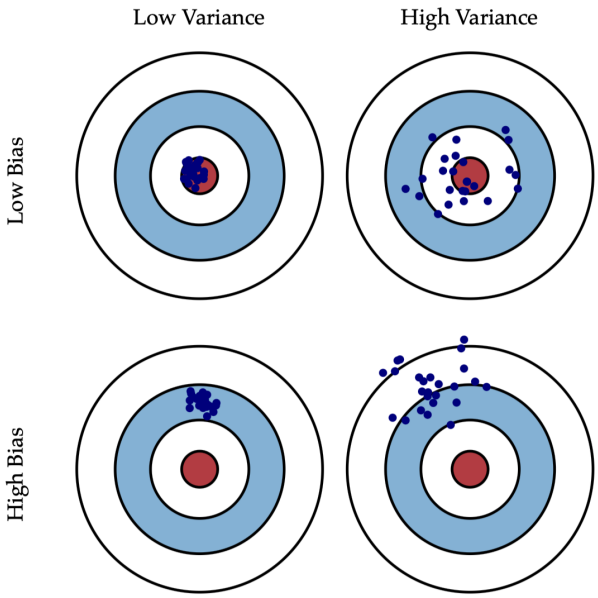


Figure 2.2: Graphical illustration of bias and variance

Over-fitting refers to the case where we have low bias but high variance, which usually occurs when the model has high complexity, i.e., many features used in training and prediction.

Under-fitting refers to the case where we have high bias but low variance, which occurs when the model complexity is too low.

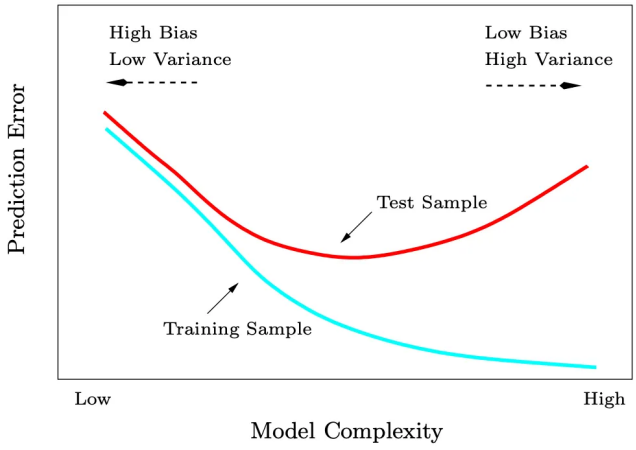


Figure 2.3: Model Complexity

Ordinary least squares is prone to producing a model with low bias but high variance. To improve the EPE, we can tune the model complexity. Shrinkage methods are a set of techniques for automatically controlling and reducing model complexity.

### Ridge Regression

When there are many correlated features in a linear regression model, the estimated coefficients can become poorly determined and exhibit high variance. For example, a very large positive coefficient on one variable can be offset by a similarly large negative coefficient on a correlated variable. The ridge penalty regularizes the coefficients by adding a cost for large values, so that the optimization function discourages excessively large coefficients. As a result, the problem of unstable estimates due to multicollinearity is alleviated.

As we have

$$\hat{Y} = \hat{f}_{\theta_0,\Theta}(\mathbf{X}) = \mathbf{X}\Theta + \theta_0,$$

the ordinary least squares problem can be formulated as the following optimization:

$$\hat{\Theta} = \arg\min_{\Theta} \|\mathbf{X}\Theta + \theta_0 - Y\|_2^2.$$

Ridge regression aims to shrink the parameters by imposing a penalty on the squared magnitude of the coefficients:

$$\hat{\Theta}^{ridge} = \arg\min_{\Theta} \|\mathbf{X}\Theta + \theta_0 - Y\|_2^2 + \lambda\|\Theta\|_2^2.$$

The penalty term (also known as the L2 norm) is defined as

$$\|\Theta\|_2^2 = \sum_{i=1}^{n} \theta_i^2.$$

Note that $\theta_0$ is not included in the penalty term, and $\lambda$ is a user-specified hyperparameter.

In the figures below, the red '+' markers represent points on the true relationship $f(X)$ (shown as blue lines). Ordinary least squares fits the training data very well but exhibits high variability across different training sets. In contrast, ridge regression reduces the variability of the estimated coefficients, leading to more stable models, but may not fit the training data perfectly.

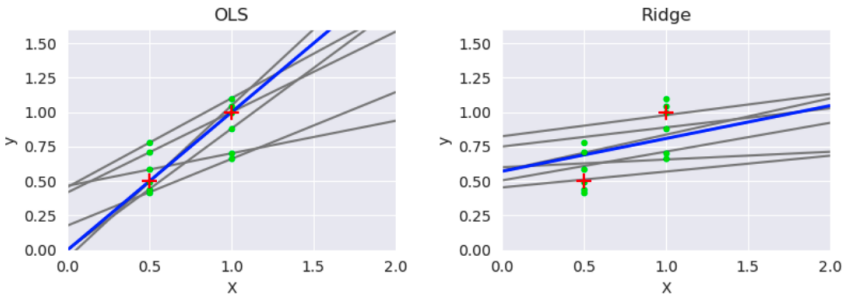


Figure 2.4: OLS v.s. Ridge

**Intuition.** In a test, each feature represents a factor affecting your score: hours studied, sleep, coffee, etc. In OLS, you try to perfectly fit the past test scores. Some factors may receive very large weights to match the training data exactly.

However, if a future test changes slightly (due to noise), your predictions fluctuate a lot, resulting in high variance. Ridge regression addresses this by discouraging any factor from having too much influence. If a parameter becomes too large, it incurs a penalty.

As a result, the optimization shrinks $\Theta$ toward zero.

### Lasso Regression

Lasso regression is also a shrinkage method, similar to ridge regression, but differs in the penalty term:

$$\hat{\Theta}^{lasso} = \arg\min_{\Theta} \|\mathbf{X}\Theta + \theta_0 - Y\|_2^2 + \lambda\|\Theta\|_1.$$

The penalty term (also known as the L1 norm) is defined as

$$\|\Theta\|_1 = \sum_{i=1}^{n} |\theta_i|,$$

and is designed to encourage sparsity in the parameters. Sparsity means that many elements in $\Theta$ are exactly zero, effectively reducing the number of features used and therefore decreasing model complexity.

Similarly to ridge regression, Lasso yields models with less variance, but it is less likely to perfectly fit every data point.

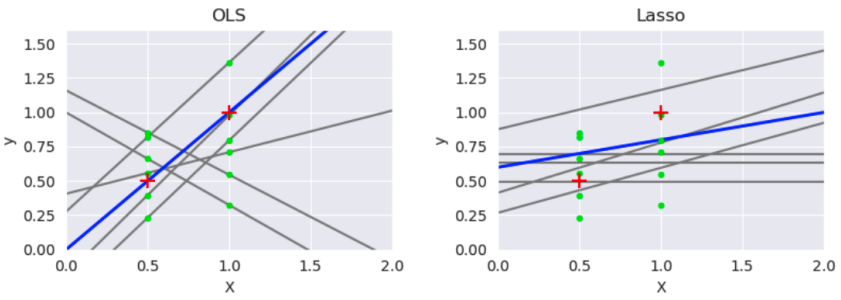


Figure 2.5: OLS v.s. Ridge

## 2.2 Logistic Regression

Before looking at regression, we take a look at classification.

### 2.2.1 Classification

Classification is a process related to categorizing things. The goal of a classification problem is to learn a classifier for a given set of samples with different categories such that it can automatically recognize new samples within the set of given categories.

The output of classification is a discrete value, while for regression it is a continuous value. Think of it as: classification gives a class (category) and is therefore discrete, while regression gives a number without a category.

In classification, samples are represented as feature vectors. Suppose we have $m$ training samples, each with $n$-dimensional features. We can write them as

$$\begin{aligned} X^{(1)} &= (x_1^{(1)}, \cdots, x_n^{(1)}) \\ X^{(2)} &= (x_1^{(2)}, \cdots, x_n^{(2)}) \\ &\vdots \\ X^{(m)} &= (x_1^{(m)}, \cdots, x_n^{(m)}) \end{aligned}$$

with their corresponding category labels $y^{(1)}, y^{(2)}, \cdots, y^{(m)}$.

There are two types of classification. One is **binary classification**, where there are only two categories. We use 0 (negative) and 1 (positive) to label the categories. Another is **multi-class classification**, where we use 1, 2, 3, ... to label the categories.

Then, how do we classify the samples? We partition the entire feature space into multiple parts, one for each category (aka class). Data points associated with the same category lie in the same partition. A **decision boundary** is a union of curves or surfaces that partition the feature space.

A linear classifier is a kind of linear model that classifies data based on a linear combination of input features. The decision boundary for a linear classifier is a union of straight lines or hyperplanes.

For a linear binary classifier, its decision boundary is a straight line or hyperplane that partitions the feature space into two half-spaces.

Then, the problem that arises is: how do we classify the samples, i.e., which class should a sample belong to? We consider logistic regression.

### 2.2.2 Formulation

Logistic regression models the probabilities for classification problems with two possible outcomes.

Consider linear binary classification. Given an input $n$-dimensional feature vector $X \in \mathbb{R}^n$ and its possible category $y \in \{0, 1\}$, we would like to construct a model to estimate

$$\mathbb{P}(\hat{y} = 1 \mid X).$$

As we have only two classes, we have

$$\mathbb{P}(\hat{y} = 0 \mid X) + \mathbb{P}(\hat{y} = 1 \mid X) = 1.$$

If $\mathbb{P}(\hat{y} = 1 \mid X) \geq \mathbb{P}(\hat{y} = 0 \mid X)$ or equivalently $\mathbb{P}(\hat{y} = 1 \mid X) \geq 0.5$, we say that $y = 1$ is more likely to occur, and it is predicted to be label 1; otherwise, it is predicted to be label 0.

**Remark** (Is linear regression applicable for binary classification?).

We can use a linear regression model to fit data points. Can we use the same model for binary classification and predict the probability?

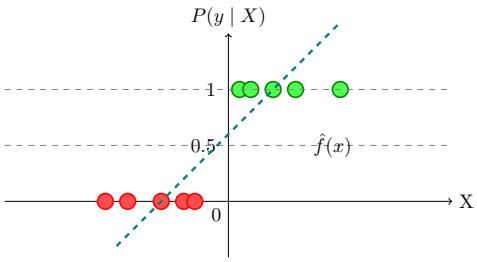


Figure 2.6: Linear Regression

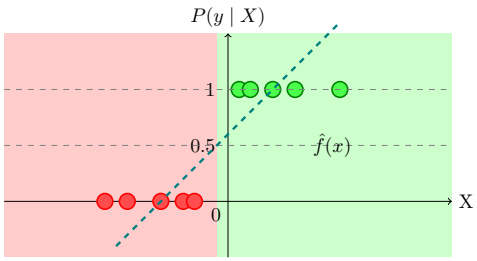


Figure 2.7: Linear Regression

Figure 2.6 shows that it is possible, but the range of linear functions is $(-\infty, +\infty)$, which is not suitable for outputting probabilities.

If we apply a cut-off rule (Figure 2.7), such that $\hat{f}(X) \geq 0.5 \Rightarrow \hat{y} = 1$, it is valid. However, linear regression with a cut-off rule would be sensitive to outliers. This is because outliers (points with very large or small $y$ values) have a disproportionate effect due to squaring large errors, which can shift the regression line. This, in turn, changes the threshold crossing point and may change the predicted class for many samples.

Thus, it is hard to use a cut-off with linear regression for classification.

Consider the following:

For an event $A$, we say that the probability it happens is $\mathbb{P}(A) = p$ and $\mathbb{P}(A^C) = 1 - p$. Then, the odds (the ratio of the probability of the event to its complement) of event $A$ is defined as

$$\text{odds}(p) = \frac{p}{1-p}.$$

The range of the odds is $(0, +\infty)$. We define the logit (log-odds) of an event $A$ as

$$\text{logit}(p) = \ln \text{odds}(p) = \ln\left(\frac{p}{1-p}\right),$$

whose range is $(-\infty, +\infty)$.

This is called the logit transformation, which maps probabilities in the range $(0, 1)$ to $(-\infty, +\infty)$.

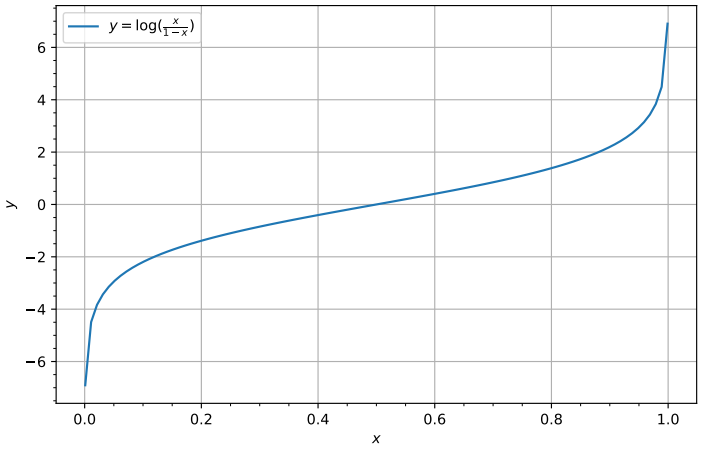


Figure 2.8: Logit

Then, in order to adopt a linear function for classification, we can apply $\text{logit}^{-1}(\cdot)$, which transforms any value in $(-\infty, +\infty)$ to a probability in $(0, 1)$.

Note that $\text{logit}(p)$ is strictly monotonically increasing, so $\text{logit}^{-1}(\cdot)$ must exist. To find it, for all $z \in (-\infty, +\infty)$, there exists $p \in (0, 1)$ such that

$$\begin{aligned} z &= \text{logit}(p) = \ln\left(\frac{p}{1-p}\right) \\ -z &= \ln\left(\frac{1-p}{p}\right) \\ e^{-z} &= \frac{1}{p} - 1 \\ p &= \frac{1}{1+e^{-z}} \end{aligned} \qquad \Longrightarrow \qquad \text{logit}^{-1}(z) = p = \frac{1}{1+e^{-z}}$$

This is also known as the logistic sigmoid function:

$$\sigma(x) = \text{logit}^{-1}(x) = \frac{1}{1+e^{-x}}.$$

The graph of the sigmoid function is S-shaped. $\sigma(x)$ is monotonic, and its first derivative is bell-shaped.

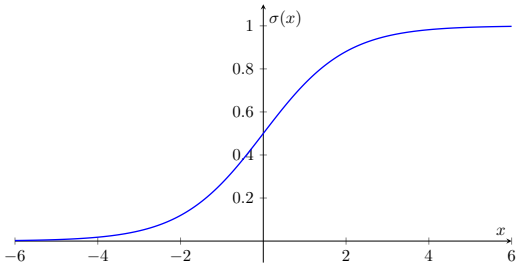


Figure 2.9: Logistic Sigmoid Function

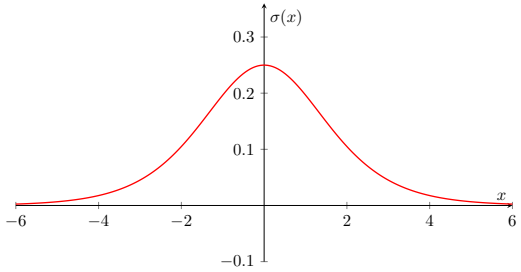


Figure 2.10: Logistic Sigmoid Function - Derivative

Recall the linear function

$$\hat{f}_\Theta(X) = X^T\Theta + \theta_0.$$

As our goal is to find $\mathbb{P}(\hat{y} = 1 \mid X)$, we apply the logit transformation on the linear function to fit the probability for classification prediction:

$$\mathbb{P}(\hat{y} = 1 \mid X) = \sigma(X^T\Theta + \theta_0) = \text{logit}^{-1}(X^T\Theta + \theta_0) = \frac{1}{1+e^{-(X^T\Theta+\theta_0)}}.$$

Here, $\Theta$ and $\theta_0$ are parameters learned from the training data. For simplification, we absorb $\theta_0$ into $\Theta$, then we have

$$\mathbb{P}(\hat{y} = 1 \mid X) = \sigma(X^T\Theta) = \frac{1}{1 + e^{-(X^T\Theta)}}.$$

This is the basic formulation of logistic regression.

> **Intuition.** In logistic regression, each sample's features are combined linearly using $\Theta^T X$ to produce a score reflecting its tendency toward a class. Unlike linear regression, this score is not the prediction itself, because it can take any real value. Instead, we apply the sigmoid (inverse logit) function to transform the linear score into a valid probability between 0 and 1. This allows us to interpret the output as the probability of belonging to a class while keeping a linear decision boundary. Logistic regression is preferred over linear regression for classification because it produces stable probabilities, handles binary labels correctly, and is less sensitive to outliers than a simple cutoff rule.

Logistic regression exactly outputs the probability of $y = 1$ conditioned on the known input $X$. $\mathbb{P}(\hat{y} = 1 \mid X)$ changes significantly around $X^T\Theta = 0$, which is a good characteristic for distinguishing samples with different labels, and it is more robust to outliers.

Since $\sigma(X^T\Theta) \geq 0.5$ iff $X^T\Theta \geq 0$, we know that

$$X^T\Theta \geq 0 \Rightarrow \hat{y} = 1 \quad \text{and} \quad X^T\Theta < 0 \Rightarrow \hat{y} = 0.$$

The geometry of $X^T\Theta = 0$ is a hyperplane, which also serves as the decision boundary. The region where $X^T\Theta \geq 0$ lies on one side of the hyperplane, while the region where $X^T\Theta < 0$ lies on the other side.

### 2.2.3 Optimization

The problem now lies in finding the optimal $\Theta$ through the training data such that we maximize classification performance.

Given training data with labels, for each sample $X^{(i)} = (x_1^{(i)}, x_2^{(i)}, \cdots, x_n^{(i)})$, we have

$$\mathbb{P}(\hat{y}^{(i)} = 1 \mid X^{(i)}) = \sigma\left({X^{(i)}}^T\Theta\right) = \frac{1}{1 + e^{-({X^{(i)}}^T\Theta)}}.$$

We then follow the supervised criterion. If $y^{(i)} = 1$, we expect $\mathbb{P}(\hat{y}^{(i)} = 1 \mid X^{(i)})$ to approach 1 as closely as possible, and similarly for $y^{(i)} = 0$. The probability that the label is correctly predicted can be written as

$$p_i \overset{\text{def}}{=} \mathbb{P}\left(\hat{y}^{(i)} = 1 \mid X^{(i)}\right)^{y^{(i)}} \left(1 - \mathbb{P}\left(\hat{y}^{(i)} = 1 \mid X^{(i)}\right)\right)^{1-y^{(i)}}.$$

This means that when $\hat{y}^{(i)} = 0$, $p_i = \mathbb{P}(\hat{y}^{(i)} = 0 \mid X^{(i)})$, and when $\hat{y}^{(i)} = 1$, $p_i = \mathbb{P}(\hat{y}^{(i)} = 1 \mid X^{(i)})$.

So no matter which category the sample belongs to, $p_i$ represents the probability that the prediction $\hat{y}^{(i)}$ matches the true label $y^{(i)}$.

Then we can try to maximize the joint probability of all training samples to learn $\Theta$. Since all samples are independent, we have

$$\begin{aligned}
p(\Theta) &\overset{\text{def}}{=} \mathbb{P}\left(\hat{y}^{(1)} = y^{(1)}, \cdots, \hat{y}^{(m)} = y^{(m)} \mid X^{(1)}, \cdots, X^{(m)}\right) \\
&= \mathbb{P}\left(\hat{y}^{(1)} = y^{(1)} \mid X^{(1)}\right) \cdots \mathbb{P}\left(\hat{y}^{(m)} = y^{(m)} \mid X^{(m)}\right) \\
&= \prod_{i=1}^{m} \mathbb{P}\left(\hat{y}^{(i)} = 1 \mid X^{(i)}\right)^{y^{(i)}} \left(1 - \mathbb{P}\left(\hat{y}^{(i)} = 1 \mid X^{(i)}\right)\right)^{1-y^{(i)}} \\
&= \prod_{i=1}^{m} p_i.
\end{aligned}$$

Here, $p(\Theta)$ is also known as the likelihood function. The method that estimates the parameters of a probabilistic model by maximizing the likelihood function is called **maximum likelihood estimation (MLE)**.

Maximizing $p(\Theta)$ is equivalent to minimizing $-\ln p(\Theta)$. Then we consider

$$\begin{aligned} E(\Theta) &= -\ln p(\Theta) = -\ln\left(\prod_{i=1}^{m} p_i\right) = -\sum_{i=1}^{m} \ln p_i \\ &= -\sum_{i=1}^{m} \ln\left(\mathbb{P}\left(\hat{y}^{(i)} = 1 \mid X^{(i)}\right)^{y^{(i)}} \left(1 - \mathbb{P}\left(\hat{y}^{(i)} = 1 \mid X^{(i)}\right)\right)^{1-y^{(i)}}\right) \\ &= -\sum_{i=1}^{m} \left[y^{(i)} \ln \mathbb{P}\left(\hat{y}^{(i)} = 1 \mid X^{(i)}\right) + (1 - y^{(i)}) \ln\left(1 - \mathbb{P}\left(\hat{y}^{(i)} = 1 \mid X^{(i)}\right)\right)\right]. \end{aligned}$$

For each single sample, we define

$$E_i(\Theta) = -y^{(i)} \ln \mathbb{P}\left(\hat{y}^{(i)} = 1 \mid X^{(i)}\right) - (1 - y^{(i)}) \ln\left(1 - \mathbb{P}\left(\hat{y}^{(i)} = 1 \mid X^{(i)}\right)\right),$$

so that we can write

$$E(\Theta) = \sum_{i=1}^{m} E_i(\Theta).$$

Consider the cross-entropy error function

$$\text{cross-entropy error} = -y \ln \mathbb{P}(y) - (1 - y) \ln(1 - \mathbb{P}(y)), \quad y \in \{0, 1\},$$

where in our case we have $\mathbb{P}(y) = \mathbb{P}\left(\hat{y}^{(i)} = 1 \mid X^{(i)}\right)$.

Since the cross-entropy function is convex, and the sum of convex functions is also convex, we know that $E_i(\Theta)$ is convex. Thus, $E(\Theta)$ is convex, and its global minimum is also a local minimum.

We can then use **gradient descent** to minimize $E(\Theta)$.

**Intuition** (Gradient Descent). Suppose the geometry of the objective function $z = f(x, y)$ is a surface resembling a mountain. Iterative optimization is similar to searching for a path to the foot of the mountain step by step. At each step, we only move downhill. To descend faster, we move along the steepest slope.

Gradient descent is motivated by this greedy scheme: at each optimization step, we adjust parameters in the direction where the objective function decreases fastest.

As the function we have is convex, gradient descent can find the steepest descent direction, i.e. global minimum.

**Proposition 2.2.1.** For a twice-differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, the gradient $\nabla f(\Theta)$ points in the direction of the steepest ascent or descent at $\Theta$, where

$$\nabla f(\Theta) = \frac{\partial f(\Theta)}{\partial \Theta} = \left(\frac{\partial f(\Theta)}{\partial \theta_1}, \cdots, \frac{\partial f(\Theta)}{\partial \theta_n}\right),$$

and $\theta_i$ is the $i$-th element of $\Theta$.

**$\nabla f(\Theta)$ maximizes the directional derivative.** Fix any $\Theta \in \mathbb{R}^n$ and $\mathbf{v} \in \mathbb{R}^n$ with $\|\mathbf{v}\|_2 = 1$. Define

$$g(h) = f(\Theta + h\mathbf{v}).$$

The directional derivative along $\mathbf{v}$ at $\Theta$ is

$$D_{\mathbf{v}} f(\Theta) = \lim_{h \to 0} \frac{f(\Theta + h\mathbf{v}) - f(\Theta)}{h} = \lim_{h \to 0} \frac{g(h) - g(0)}{h} = g'(0).$$

Also, we have

$$g'(h) = \frac{\partial f(\Theta + h\mathbf{v})}{\partial(\Theta + h\mathbf{v})} \cdot \frac{\partial(\Theta + h\mathbf{v})}{\partial h} = \nabla f(\Theta + h\mathbf{v}) \cdot \mathbf{v},$$

which gives

$$D_{\mathbf{v}} f(\Theta) = g'(0) = \nabla f(\Theta) \cdot \mathbf{v}.$$

By the Cauchy-Schwarz inequality:

$$D_{\mathbf{v}} f(\Theta) = \nabla f(\Theta) \cdot \mathbf{v} \leq \|\nabla f(\Theta)\|_2 \, \|\mathbf{v}\|_2.$$

Equality is achieved if and only if $\mathbf{v}$ is a scalar multiple of $\nabla f(\Theta)$, which means it points in the optimal direction $\mathbf{v}^\star$ that maximizes the directional derivative $D_{\mathbf{v}} f(\Theta)$. ■

**Algorithm 2.1:** Gradient Descent

**Ensure** $\alpha > 0$
Initialize $\Theta \leftarrow \Theta_0$ randomly
**while** *not converged* **do**
    $Theta \leftarrow \Theta - \alpha \nabla f(\Theta)$

In gradient descent, we introduce a hyperparameter called the **learning rate**, which can be adjusted at each iteration. A learning rate that is too small leads to slow convergence, while a learning rate that is too large may cause divergence or unstable updates.

**Remark.** Gradient descent also works for non-convex objective functions. However, for most complex and non-convex functions, gradient descent usually converges to a local minimum and may not reach the global minimum.

Then we return to logistic regression. The analytic expression of the gradient of $E(\Theta)$ is

$$\nabla E(\Theta) = \frac{\partial E(\Theta)}{\partial \Theta} = \sum_{i=1}^{m} \left( \frac{1}{1 + e^{-X^{(i)T} \Theta}} - y^{(i)} \right) X^{(i)}.$$

**Algorithm 2.2:** Gradient Descent for Logistic Regression

**Ensure:** $\alpha > 0$
Initialize $\Theta$ randomly
**while** *not converged* **do**
    $\Theta \leftarrow \Theta - \alpha \sum_{i=1}^{m} \left( \frac{1}{1+e^{-X^{(i)T} \Theta}} - y^{(i)} \right) X^{(i)}$

**Intuition.** Logistic regression is a probability model for binary classification, predicting the probability that a given sample belongs to class 1. The input features $X$ are combined linearly using $\theta^T X + \theta_0$, and the logit (sigmoid) transformation converts this linear score into a probability in $(0, 1)$. However, the parameters $\theta$ are unknown, and they directly affect the predicted probabilities. To find the optimal $\theta$, we use the training data to construct the likelihood function, which measures how well the predicted probabilities match the observed labels. Maximizing this likelihood is equivalent to minimizing the negative log-likelihood, which is a convex function. Because it is convex, we can efficiently find the minimum using gradient descent, which iteratively moves in the direction of the steepest descent of the loss function. This process gives the $\theta$ that maximizes the likelihood and produces the most accurate predicted probabilities for classification.

# Chapter 3

# Support Vector Machine

# Chapter 4

# Clustering Algorithms

# Chapter 5

# Neural Network Basics

# Chapter 6

# Convolutional Neural Networks

# Chapter 7

# Transformers

# Chapter 8

# Uniformed Search

# Chapter 9

# Informed Search

# Chapter 10

# AI Applications