Foundations of Machine Learning

redatto da Michele Valenzano

Anno Accademico 2024/2025

Contents

List of Figures

Chapter 1

Lesson 1 - 25/09/2024

1.1 AI definitions

Here some definitions of AI:

- It is the science and engineering of making intelligent machines, especially intelligent computer programs. It's related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biological observable. John McCarthy 1956.
  AI is science-based, so it has mathematical bases, but engineering build something over scientifically bases. So, AI is much more software than hardware. For example, intelligent robot are a type of AI application, but it's better to say that robots moves in an intelligent way due to AI software
- The goal of AI is solving problems and building applications that actually works. Peter Norvig, 2016

1.1.1 The Turing test

The Turing test it's used to determine if a machine has an intelligent behaviour. This test idea was presented by Alan Turing in a paper in 1950.
This test takes inspiration from *the imitation game* in which there are two person and a machine in three different rooms. The Turing test succeed, so the machine is considered intelligent, when a person is no more able to recognize if he's talking to a machine or a human.

The Turing test open to different scenarios and AI applications:

- **Machine Learning**: in the Turing test the machine **learns**, by examples, how to behave like an human
- **Computer Vision**: a machine is able to recognize objects, people and many other things starting from images
- **Natural Language Processing (NLP) - Chatbot**: AI is able to understand and analyze text content
- **Recommender Systems**: AI is able to recommend something to the user knowing his behaviour or his taste

- **Big Data Analytics**: there are a lot of examples and heterogeneous data which AI can use to learn
- **Knowledge Representation and Automated Reasoning**: machines doesn't know to interpret a word, so need to learn what a word means and how to represent it. For example, a machine doesn't know what the word cat means, so it has to learn what this word means and how to use depending on context

1.1.2 A bit of history and applications

AI was not born recently, but dates back in 50's. Before AI's employment in real applications, there was a large mathematical analysis to study them from a theoretical point of view. AI and Machine Learning, which is a part of AI, is a lot of *matrix processing*, which is an operation hugely performed by GPU. So, a fun fact is that playing video-games helped the improvement of AI. AI today is used in medicine, demotic, virtual assistants, translators, autonomous driving, games and so many other applications.

1.2 Machine Learning

Here some Machine Learning definitions:

- Learning is any process by which a system improves performance from experience. Herbert Alexander Simon
- Machine Learning is concerned with computer programs that automatically improve their performance through experience. Herbert Alexander Simon
- Machine Learning: field of study that gives computer the ability to learn without being explicitly programmed. Arthur Samuel
- A computer program is said to learn from experience (E) with some class of tasks (T) and a performance measure (P) if its performance at task in T as measured by P improves with E. Tom Mitchell

The goal of the explainable AI is to build an AI which can give explanations based on human requests.

Let's suppose we want to build a program able to watch our classifications of emails (spam or not spam) and learn how to better filter spam. In this case:

- (T) The task is classifying emails as spam or not spam
- (E) The experience is watching you label emails as spam or not spam
- (P) The performance is the number (or fraction) of emails correctly classified as spam/not

1.2.1 First Machine Learning Classification

Supervised Learning

The aim is to search particular relationship or pattern in data, to build a model which we can use to make predictions on unseen data. For example, let's consider an house price prediction tool. We know some living area and the corresponding house price. The goal of supervised learning is to find relationship between data and build a model which can be used to predict the price of a house knowing it's area.

Regression or Classification

When the output of the model is in a finite set of values we talk about **classification**, instead, if the set is infinite the output can be any real number and we talk about **regression**.

Unsupervised Learning

In unsupervised learning we haven't labels for any example, so the aim of this learning is to find similarities in data and create some clusters of data.

Reinforcement Learning

In reinforcement learning we give the AI the possibility to explore the environment. If the explorations goes in a correct way then the AI receives a reward, instead it receives a negative reward. Some projects like AI learn to play a video-game are applications of reinforcement learning.

Recommender Systems

This type of AI can predict what an user could like based on his previous interests. For example, these systems are used by Netflix to suggest films or TV-series or by Amazon suggesting something to buy.

1.2.2 Second Machine Learning Classification

Discriminative Models

These models are trained over a training set. When these models take in input, they estimate the most probable output. The purpose is to estimate the conditional probability $p(y|x)$.
Some applications of discriminative model might be house price prediction, image classification, text classification, ecc...

Generative Models

The purpose in this models is to estimate the joint probability $p(x, y)$. These are probabilistic models that produce both input and output. After the model is trained, the conditional probability can be inferred. These types of models learns and study the probability distribution of data and, since we're talking about probability and not certainty, with the same input we can get different outputs.

Chapter 2

Lesson 2 - 30/09/2024

2.1 Supervised Learning

2.1.1 Linear regression with one variable (Univariate Linear Regression)

We want to predict the price of a house knowing its area. We have lot of data points in the space and we want to find a linear approximation of them.
The set of data that we use to train our model (algorithm) is called **training set**.
In the rest of the course we'll use this notation:

- m = number of training examples
- x = input variable or *feature*
- y = output variable or *target* variable
- (x, y) = tuple representing one training example
- $(x^{(i)}, y^{(i)}) = i^{th}$ training example

What we want to realize is an algorithm able to find pattern in data, which is called **learning algorithm** and an hypothesis h which is able to estimate new values. We want to compute **best hypothesis** (model), that is the best function to represent input data. Since there are many types of functions, on the same data, we can have different learning algorithms based on different functions.

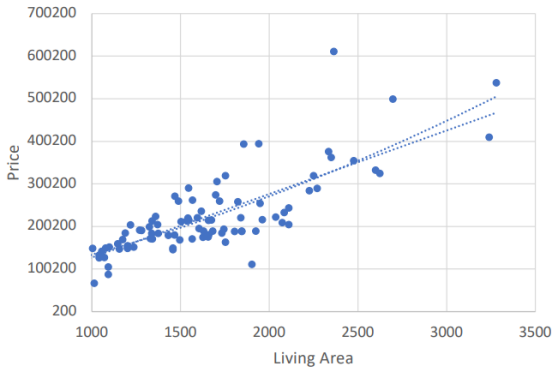


Figure 2.1: House Price Linear Regression

2.1.2 Cost Function

In the case of linear regression our hypothesis is in the form $h_\theta(x) = \theta_0 + \theta_1 x$.

In the learning algorithm, we are searching the best θ_0, θ_1, also called weights (or parameters) such that our model fits well data.
How θ_i can be chosen based on our data? Remember, the value of our parameters depends on our input data.
In order to choose parameters well, we need to minimize the average (not overall) error of our model on input data.

$$\text{argmin}_{\theta_0,\theta_1} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

where $h_\theta(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$.
The cost function is:

$$J(\theta_0, \theta_1) = J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

In the case of linear regression, cost function is a quadratic function, which has only a single minimum point.

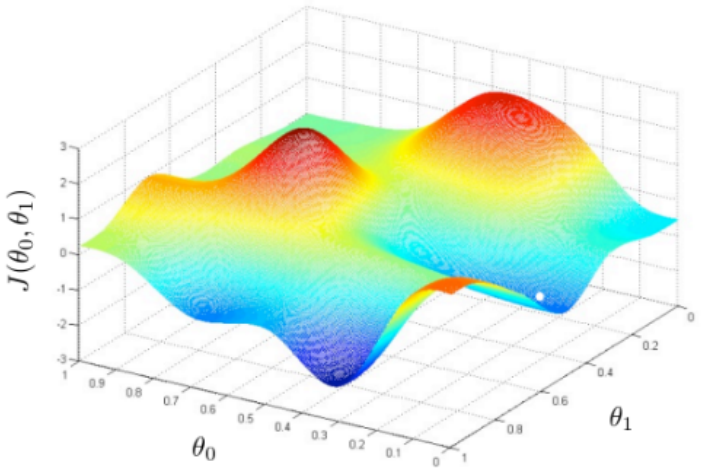


Figure 2.2: Gradient Descent Minimum Point with 2 Parameters

Sometimes is impossible to find the minimum value of the cost function. We want to find the closest parameters to the minimum pair of parameters of the cost function.

2.1.3 Gradient Descent

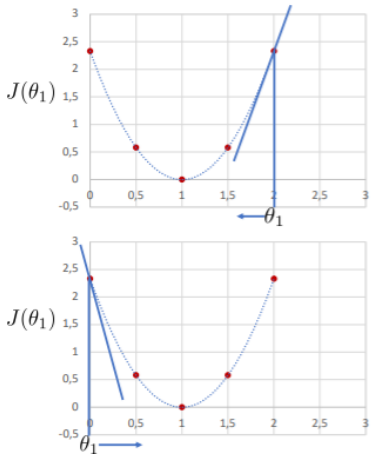
Let's see an algorithmic method to find the best possible hypothesis. The main idea is:

- Initialize θ_0, θ_1 to some values
- Keep changing θ_0, θ_1 to reduce cost function $J(\theta_0, \theta_1)$ to get closer to minimum value

The update below is repeated until convergence, for θ_0 and θ_1:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

In this phase we must pay attention to parameter's update: this should happen after new parameter computation. Otherwise we will calculate the second parameter based on the new first parameter and this is an error.



$$\theta_1 := \theta_1 - \alpha \underbrace{\frac{\partial}{\partial \theta_1} J(\theta_1)}_{\geq 0}$$

$$\theta_1 := \theta_1 - \alpha * (positive \quad number)$$

$$\theta_1 := \theta_1 - \alpha \underbrace{\frac{\partial}{\partial \theta_1} J(\theta_1)}_{\leq 0}$$

$$\theta_1 := \theta_1 - \alpha * (negative \quad number)$$

Figure 2.3: Gradient Descent Parameter Update

α is called **learning rate** and we have to fix manually this rate (is an hyperparameter). If α is too small gradient descent is slow, otherwise gradient can diverge.

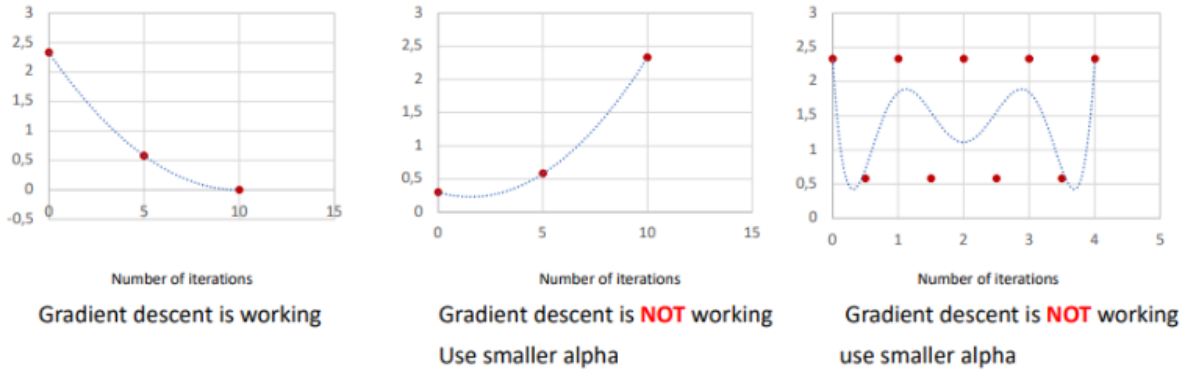


Figure 2.4: Gradient Descent Examples

$$j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$j = 1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

Chapter 3

Lesson 3 - 02/10/2024

All the examples we've seen until now works on only one feature. But, what happens when we have more than one features?

3.1 Multiple Feature

Let's consider out house pricing example, and suppose we have some other features like the number of bedrooms, or the number of floors. The notation changes a little bit:

- m = number of training examples
- n = number of features (of each example)
- $x^{(i)}$ = input features of the i^{th} training example
- $y^{(i)}$ = output/target variable of the i^{th} training example
- $x_j^{(i)}$ = value of feature j in i^{th} training example

Also, the hypothesis changes because we consider new features with, each one, a corresponding parameter. Since we have more variables, we work with an **hyperplane**.

Considering for example 3 features, our hypothesis become

$$h_\theta(x^{(i)}) = \theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \theta_3 x_3^{(i)}$$

In general, with n features our hypothesis become

$$h_\theta(x^{(i)}) = \theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + ... + \theta_n x_n^{(i)}$$

We can represent our features and our parameters with vectors and for convenience we set $x_0^{(i)} = 1$.

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

The overall hypothesis can be written multiplying θ_T by x: $h_\theta(x^{(i)}) = \theta^T x$.

3.1.1 Batch Gradient Descent

As for the linear gradient descent, we have the **batch gradient descent** to update parameters using all the dataset for each update of θ_j. So, until convergence we have to repeat

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad for\ j = 0, 1, ..., n$$

3.1.2 Stochastic Gradient Descent

There is another type of gradient descent, called **stochastic gradient descent**. The parameter are updated after each training sample (every update is based on one single training sample), so the gradient is a **stochastic approximation** of the "true" cost gradient.
The optimal solution is generally approximated faster, but the algorithm could swing around it without never getting convergence (zig-zag problem). Sometimes, a close solution to the optimal one is often acceptable.
The parameters are continuously updated: the h function will use different values at each iteration.

The main difference between these gradient descent algorithm is the position of the for loops:

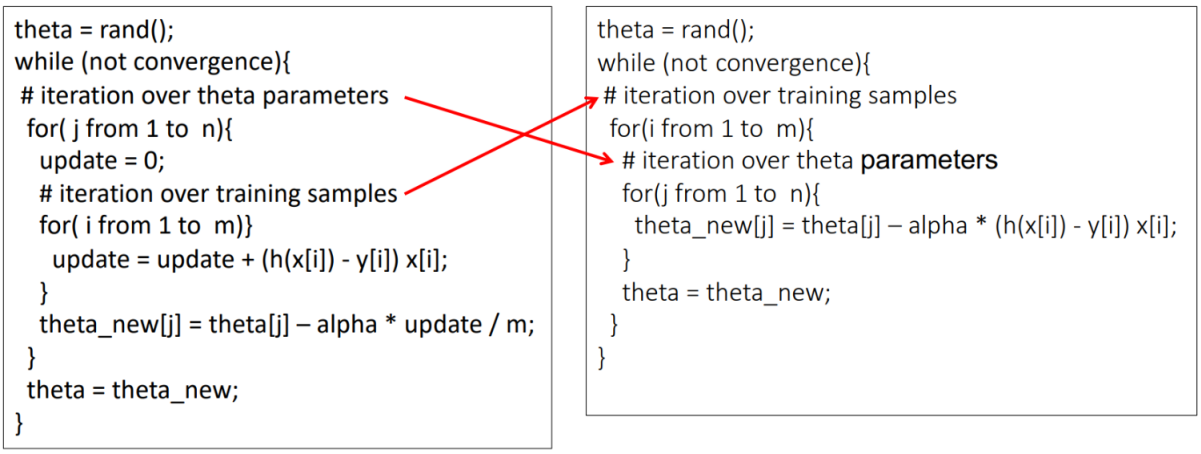


Figure 3.1: Batch Gradient Descent vs Stochastic Gradient Descent

In a visual point of view, each "jump" in Batch Gradient Descent is direct to the minimum, but the update step goes through all the examples. This method is slow, but convergence in sure.
Instead, in Stochastic Gradient Descent each "jump" may potentially go everywhere, but it's relative to just one training sample. This method is fast, but convergence is not sure.

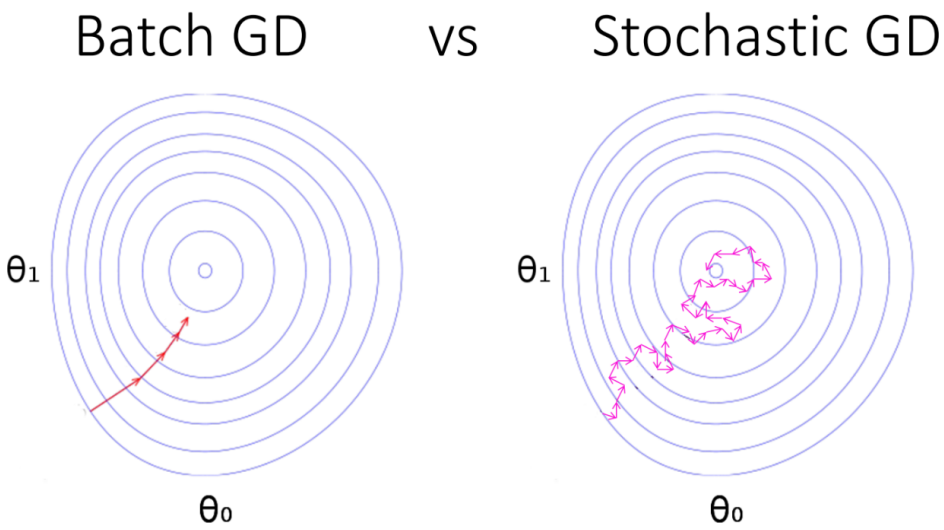


Figure 3.2: Visual Batch Gradient Descent vs Stochastic Gradient Descent

3.1.3 Mini-Batch Gradient Descent

Can we find an intermediate solution? A compromise between batch and stochastic gradient descent takes the advantages of both. With mini-batch gradient each update is performed in a subset (mini-batch) of the examples set. The size of the mini-batch is denoted with b and $1 < b << m$ (where m is the total number of training samples). For each update, we choose a different subset of the dataset.
With this algorithm we reduce the variance of the parameters update, having a more stable convergence. Furthermore, it allows the use vectorization libraries rather than computing each step separately.

```
b = 50;
while (not convergence){
 # iteration over training samples
  while(i  <  m){
    # iteration over theta parameters
    for(j from 1 to  n){
       update = 0;
       # iteration over the b samples
       for( z from i to i + b ){
          update = update + (h(x[z]) - y[z]) x[z];
        }
        theta[j] = theta[j] – alpha * update / b;
    }
  i = i + b;
  }
}
```

Figure 3.3: Mini-Batch Gradient Descent Algorithm

If $b = 1$ then we use stochastic gradient descent, instead if $b = m$ we use batch gradient descent.

3.1.4 Gradient Descent Algorithm Stopping Conditions

Here some stopping conditions of the gradient descent algorithm:

- **Max iteration**: the algorithm stops after a fixed number of iterations
- **Absolute tolerance**: the algorithm stops when a fixed value of the cost function is reached
- **Relative tolerance**: the algorithm stops when the difference between the cost function value of the previous step and the cost function value of the actual step is below a fixed rate
- **Gradient Norm tolerance**: the algorithm stops when the norm of the gradient is lower than a fixed value (usually very small)

3.2 Features and Polynomial Regression

Let's consider an example:

$$h_\theta(x^{(i)}) = \theta_0 + \theta_1 \cdot front^{(i)} + \theta_2 \cdot side^{(i)}$$

We can reduce the dimension of our problem by introducing a new feature area = front · side. Then, our hypothesis become $h_\theta(x^{(i)}) = \theta_0 + \theta_1 \cdot area^{(i)}$.

Let's consider another example:

$$h_\theta(x^{(i)}) = \theta_0 + \theta_1 \cdot x_1^{(i)} + \theta_2 \cdot x_2^{(i)} + \theta_3 \cdot x_1^{(i)} x_2^{(i)}$$

This is not a linear function, but we can introduce a new variable $x_3^{(i)} = x_1^{(i)} x_2^{(i)}$ to get again a linear function.

We can use polynomial regression when the relationship between our examples can't be well modeled by a linear function. This type of regression involves the use of a polynomial of grade t defined as:

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + ... + \theta_t x^t$$

In this case, we are considering examples composed only by one feature.

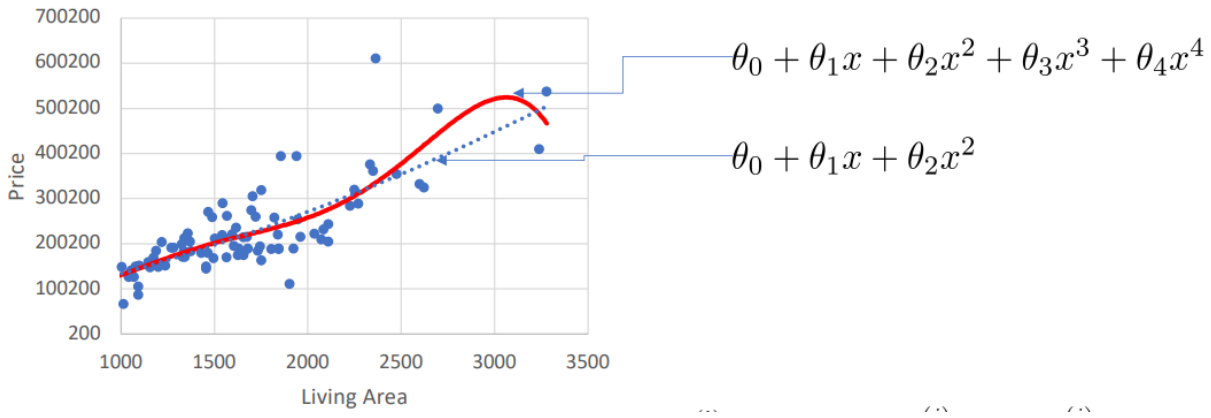


Figure 3.4: Linear Regression vs Polynomial Regression

3.3 Feature Scaling

Sometimes it's useful to scale our data in order to make them more uniform and potentially increase the computing performances of the cost function minimization.
For example, we can resize our features in a bounding region, like $[0, 1]$. Let's consider our example where x_1 is the living area of a house (with values between 0 and 2110) and x_2 is the number of bedrooms (with values between 1 and 5).
In order to get our data in a bounded range of $[0, 1]$ we can use the formula below:

$$x_1 = \frac{\text{area}(\text{feet}^2)}{2110} \qquad x_2 = \frac{\text{num of bedrooms}}{5}$$

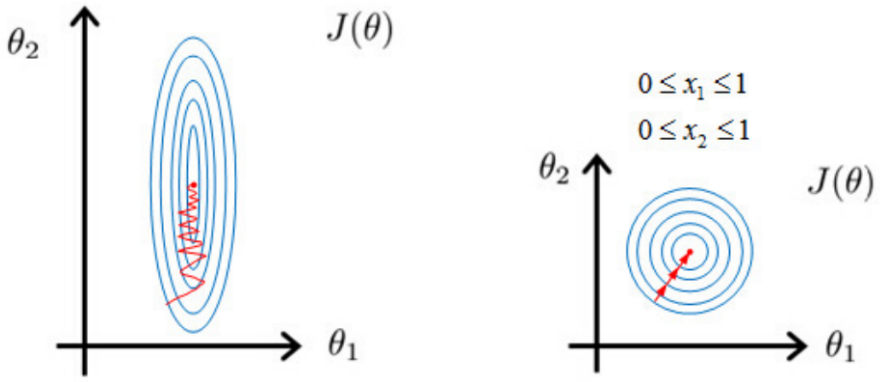


Figure 3.5: Cost Function variation with Feature Scaling

3.3.1 Min-Max Normalization

In general we can scale our data in a given range of $[a, b]$ using this formula:

$$x' = \frac{x - \min}{\max - \min} \cdot (b - a) + a$$

Where max is the maximum value of our data and min is the minimum value of our data. The main advantage of this technique is that it shifts the distribution to a smaller scale preserving the relations with the original data. In the other hand, this technique is negative influenced by outliers (since the formula consider the max and the min). Furthermore, there can be a new input which is greater (or smaller) than the max (or the min) and this affects our scaling process.

3.3.2 Z-Score Normalization

This type of normalization produces a zero-mean distribution using the mean of our values and the standard deviation devstd. The formula is:

$$x' = \frac{x - \text{mean}(x)}{\text{devstd}(x)}$$

This approach is less influenced by outliers because we consider the standard variation of our data (the distance from the mean) and doesn't need a min-max value setup.

3.4 Normal Equations

Using the vectorial notation

$$\mathbf{X} = \begin{bmatrix} x^{(1)T} \\ x^{(2)T} \\ \vdots \\ x^{(i)T} \\ \vdots \\ x^{(m)T} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(i)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

The cost function can be written as

$$J(\theta) = \frac{1}{2}(X\theta - y)^T(X\theta - y)$$

We want to minimize the cost function so we need to compute the gradient of $J(\theta)$ and set it equal to 0:

$$\nabla J(\theta) = X^T X\theta - X^T y = 0$$

From this equation we can compute the value of θ:

$$\hat{\theta} = (X^T X)^{-1} X^T y$$

3.4.1 Proof of Normal Equations Derivations

Starting from the original cost function

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

we can write the difference $X\theta - y$ as

$$X\theta - y = \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} = \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ h_\theta(x^{(2)}) - y^{(2)} \\ \vdots \\ h_\theta(x^{(m)}) - y^{(m)} \end{bmatrix}$$

It follows that the product $(X\theta - y)^T(X\theta - y)$ is equal to

$$[(h_\theta(x^{(1)}) - y^{(1)}) \quad \cdots \quad (h_\theta(x^{(m)}) - y^{(m)})] \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ \vdots \\ h_\theta(x^{(m)}) - y^{(m)} \end{bmatrix} = \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

which is the original cost function (without $\frac{1}{2}$).

We want to find the minimum, which is the point in which the gradient has null value:

$$\nabla J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_k} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix} = 0$$

With

$$\frac{\partial J(\theta)}{\partial \theta_k} = \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_k^{(i)} = [(h_\theta(x^{(1)}) - y^{(1)}) \quad \cdots \quad (h_\theta(x^{(m)}) - y^{(m)})] \begin{bmatrix} x_k^{(i)} \\ \vdots \\ x_k^{(m)} \end{bmatrix}$$

We can complete the gradient equation and re-write it as a matrix product

$$\nabla J(\theta) = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \cdots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ h_\theta(x^{(2)}) - y^{(2)} \\ \vdots \\ h_\theta(x^{(m)}) - y^{(m)} \end{bmatrix} = X^T(X\theta - y) = X^T X\theta - X^T y$$

$$\nabla J(\theta) = X^T X\theta - X^T y = 0 \quad \rightarrow \quad \hat{\theta} = (X^T X)^{-1} X^T y$$

3.4.2 Drawbacks

This process is useful, but the matrix inverse computing it's computationally expensive (in the order of $O(n^3)$, where n is the number of feature times the number of examples). Furthermore, there are some non-invertible matrix, due to linear dependent features, not enough training examples or too many features.

3.5 Probabilistic Interpretation of Linear Regression

Output can be computed as a predicted value plus an error.

$$y^{(i)} = \theta^T x^{(i)} + e^{(i)}$$

where $e^{(i)}$ is the error between the real value $y^{(i)}$ and the predicted value $h(x)$.
Assuming **independent samples** we can model the error as an exponential random variable, which we assume follows a normal distribution with mean 0 and variance σ^2.

$$p(e^{(i)}) = N(0, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{(e^{(i)})^2}{2\sigma^2}} \quad \text{for } i = 1, ..., m$$

We know that $e^{(i)} = y^{(i)} - \theta^T x^{(i)}$, so the probability $p(e^{(i)})$ is the same as considering **how likely we will obtain the output $y^{(i)}$ given the parameter $x^{(i)}$ and θ in input**.
By substitution in the previous equation, we obtain:

$$p(y^{(i)}|x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}} \quad \text{for } i = 1, ..., m$$

The closer the prediction gets to the actual value, the bigger is the value of this probability. It has the maximum value when $y^{(i)} = \theta^T x^{(i)}$.

We want to find a function that, the input given, returns the output for each training case. This probability is called **likelihood**.

$$L(\theta) = L(\theta; X; y) = p(y|X; \theta)$$

With the independence assumption we did before the probability of the joint event is

$$L(\theta) = p(y|X;\theta) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{(y^{(i)}-\theta^T x^{(i)})^2}{2\sigma^2}}$$

Since we know that the likelihood function has the maximum value when $y^{(i)} = \theta^T x^{(i)}$ we want to find the parameters vector that maximizes the likelihood function, that is the thetas that has the maximum probability to return the output given the input. This operation is called **maximum likelihood criterion**.

$$\hat{\theta} = \arg\max_{\theta} L(\theta)$$

Maximize $L(\theta)$ is equal to maximize $\log L(\theta)$ because the logarithm is monotonic function and it doesn't change the monotonicity of a function. So

$$\hat{\theta} = \arg\max_{\theta} L(\theta) = \arg\max_{\theta} \log L(\theta) = \arg\max_{\theta} l(\theta)$$

We introduced the logarithm operator to use its properties to transform the productory in a summatory

$$\begin{aligned}
l(\theta) &= \log \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) \\
&= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{(y^{(i)}-\theta^T x^{(i)})^2}{2\sigma^2}} \\
&= \sum_{i=1}^{m} \left(\log \frac{1}{\sqrt{2\pi}\sigma} + \log \mathrm{e}^{-\frac{(y^{(i)}-\theta^T x^{(i)})^2}{2\sigma^2}} \right) \\
&= m \log \frac{1}{\sqrt{2\pi}\sigma} - \sum_{i=1}^{m} \frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2} \\
&= m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2
\end{aligned}$$

Let's move back to the maximization problem:

$$\begin{aligned}
\max_{\theta} l(\theta) &= \max_{\theta} \left(m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2 \right) \\
&= \max_{\theta} \left(-\frac{1}{2\sigma^2} \sum_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2 \right) \\
&= \max_{\theta} \left(-\frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2 \right) \\
&= \min_{\theta} \left(\frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2 \right)
\end{aligned}$$

Now we can compare the least squares problem in linear regression and the minimization problem above:

$$\min_{\theta}\left(\frac{1}{2m}\sum_{i=1}^{m}(y^{(i)}-\theta^{T}x^{(i)})^{2}\right)=\min_{\theta}\left(\frac{1}{2}\sum_{i=1}^{m}(y^{(i)}-\theta^{T}x^{(i)})^{2}\right)$$

These two problems are identical. The only difference is a regularization constant $\frac{1}{m}$ that **has no influence** in the minimization problem.
So, solving the least square problem in the linear regression means solving the problem of finding the best parameters that produces the output given the input.

3.6 Logistic Regression: Classification

Let's consider another class of problems: classification problem. Instead of a real number, the output of a classification problem is a class. For example, some classifications problems are spam-not spam email classification, malignant-benignant tumor classification, ecc...
If the problem has only two possible output than $y \in \{0,1\}$, otherwise we talk about multiple classes (for example $y \in \{0,1,2,3,4,5\}$).

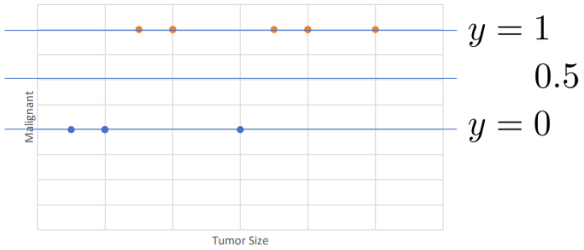


Figure 3.6: Classifier Threshold

Let's consider this example: we have a classifier with a threshold set to 0.5. If the hypothesis $h_\theta(x) \geq 0.5$ our classifier predicts 1, if the hypothesis $h_\theta(x) \leq 0.5$ our classifier predicts 0.
What if the hypothesis is exactly 0.5? In this case, the predicted class it's indifferent (it's like tossing a coin).

In logistic regression we want the hypothesis output to be bound $0 \leq h_\theta(x) \leq 1$.

3.6.1 Hypothesis Representation

The hypothesis can be represented with a sigmoid logistic function:

$$h_\theta(x)=\theta^{T}x=g(\theta^{T}x)=g(z)=\frac{1}{1+e^{-z}}=\frac{1}{1+e^{-\theta^{T}x}}$$

Each input will produce a bounded output $y \in [0,1]$ since the minimum value is 0 and the maximum value is 1.

Given our hypothesis, which is an hyperplane, we can predict the output by seeing where a point is in the space. We can predict 0 on one side of the boundary and 1 on the other side.

3.6.2 Probabilistic Interpretation

Our prediction $h_\theta(x)$ is the probability that y is 1 (or 0) on input x. We can express this probability as $h_\theta(x) = p(y = 1|x; \theta)$. Since we know there are only two possible values of y we can say that $p(y = 1|x; \theta) + p(y = 0|x; \theta) = 1$.

3.6.3 Cost Function

Our goal is to demonstrate that solving the cost function minimization problem would be equal to solve the determine hypothesis using the maximum likelihood criterion.
Given the same assumptions (independent training examples with identical distribution), we know that there are only two possible outputs (0 and 1), so we are facing a Bernoulli distribution.

$$L(\theta) = L(\theta; X; y) = p(y|X; \theta) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}; \theta)$$

Now we can write the formulas for the two probabilities:

$$\begin{cases} p(y^{(i)} = 1|x^{(i)}; \theta) = h_\theta(x^{(i)}) \\ p(y^{(i)} = 0|x^{(i)}; \theta) = 1 - h_\theta(x^{(i)}) \end{cases}$$

And we can express it in a more compact form using the general formula of Bernoulli distribution:

$$p(y^{(i)}|x^{(i)}; \theta) = h_\theta(x^{(i)})^{y^{(i)}} \cdot (1 - h_\theta(x^{(i)}))^{(1-y^{(i)})}$$

We can substitute the equation above in the likelihood function and we get:

$$L(\theta) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}; \theta) = \prod_{i=1}^{m} h_\theta(x^{(i)})^{y^{(i)}} \cdot (1 - h_\theta(x^{(i)}))^{(1-y^{(i)})}$$

Now we can use the logarithm to simplify $L(\theta)$ as we did before:

$$l(\theta) = \log \prod_{i=1}^{m} h_\theta(x^{(i)})^{y^{(i)}} \cdot (1 - h_\theta(x^{(i)}))^{(1-y^{(i)})}$$

$$= \sum_{i=1}^{m} \left(y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right)$$

Cross Entropy Function

We can move from maximization problem to the minimization one:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right)$$

And than we can set the whole problem as a parameter fitting problem $\hat{\theta} = \arg\min_\theta J(\theta)$

3.6.4 Errors

[H] Since the logistic function cannot be easily studied analytically, we can get an intuition using the error contribution:

$$e^{(i)} = -y^{(i)} \log h_\theta(x^{(i)}) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

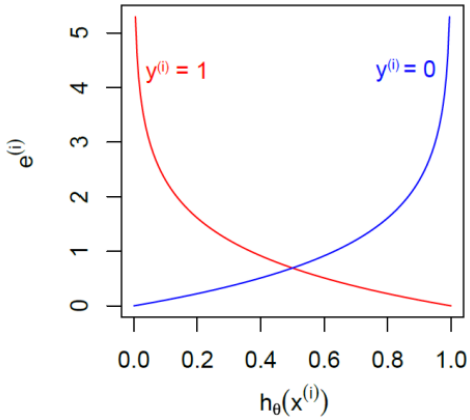


Figure 3.7: Logistic function error analysis

Now we can analyze the two cases:

- if $y^{(i)} = 1$ the error has value $e^{(i)} = -\log h_\theta(x^{(i)})$.
  If $h_\theta(x)$ is low (wrong prediction) then the error is high ($e^{(i)} \to \infty$), otherwise if $h_\theta(x)$ is high (right prediction) then the error is low ($e^{(i)} \to 0$)
- if $y^{(i)} = 0$ the error has value $e^{(i)} = -\log(1 - h_\theta(x^{(i)}))$.
  If $h_theta(x)$ is low (right prediction) then the error is low, otherwise if $h_\theta(x)$ is high (wrong prediction) the error is high

3.7 Logistic Regression: Gradient Descent

Our goal is find $\hat{\theta} = \arg\min_\theta J(\theta)$ and we can use the gradient descent to reach a minimum. The weight update is:

$$\theta_k = \theta_k - \alpha \frac{\partial J(\theta)}{\partial \theta_k}$$

We can derive the Logistic function

$$g'(z) = \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right) = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right) = g(z)(1-g(z))$$

From $g'(z) = g(z)(1-g(z))$ we can compute the derivative of $h_\theta(x)$

$$\frac{\partial h_\theta(x)}{\partial \theta_k} = h_\theta(x)(1 - h_\theta(x)) \frac{\partial(\theta^T x)}{\partial \theta_k}$$

In linear regression gradient descent we saw that

$$\frac{\partial(\theta^T x)}{\partial \theta_k} = x_k$$

So, by substitution we obtain

$$\frac{\partial h_\theta(x)}{\partial \theta_k} = h_\theta(x)(1 - h_\theta(x))x_k$$

Now, we can compute the partial derivative of

$$\frac{\partial J(\theta)}{\partial \theta_k} = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\frac{\partial}{\partial \theta_k}(\log h_\theta(x^{(i)})) + (1 - y^{(i)})\frac{\partial}{\partial \theta_k}(\log(1 - h_\theta(x^{(i)})))\right)$$

Let's consider the two derivative separately

$$\begin{cases} \dfrac{\partial}{\partial \theta_k}\log h_\theta(x^{(i)}) = \dfrac{1}{h_\theta(x^{(i)}))}h_\theta(x^{(i)})(1 - h_\theta(x^{(i)}))x_k^{(i)} = (1 - h_\theta(x^{(i)}))x_k^{(i)} \\ \dfrac{\partial}{\partial \theta_k}\log(1 - h_\theta(x^{(i)})) = \dfrac{1}{1 - h_\theta(x^{(i)})}(-h_\theta(x^{(i)}))(1 - h_\theta(x^{(i)}))x_k^{(i)} = -h_\theta(x^{(i)})x_k^{(i)} \end{cases}$$

We can substitute in the previous equation

$$\frac{\partial J(\theta)}{\partial \theta_k} = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}(1 - h_\theta(x^{(i)}))x_k^{(i)} + (1 - y^{(i)})(-h_\theta(x^{(i)}))x_k^{(i)}\right)$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}(1 - h_\theta(x^{(i)})) + (1 - y^{(i)})(-h_\theta(x^{(i)}))\right)x_k^{(i)}$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_k^{(i)}$$

Chapter 4

Lesson 4 - 07/10/2024

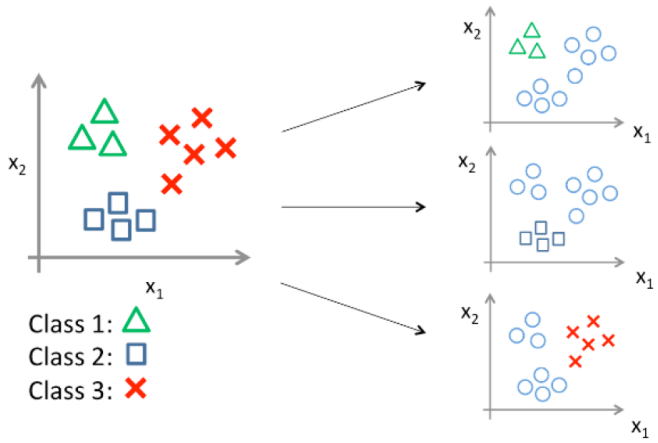
4.1 First Hands-On Lesson (LAB. 1)

Chapter 5

Lesson 5 - 09/10/2024

5.1 Multi-class Classification

Suppose we're dealing with a classification problem with more than 2 classes. With only 2 classes, we can divide our data in 2 parts. With more than 2 classes we can bring us back to a binary problem. For example, we can create different problems in witch we fix a class and we distinguish between objects belonging to that class and object that doesn't belong to that class.



$$h_\theta^{(i)}(x) = p(y = i|x; \theta) \quad (i = 1, 2, 3)$$

Figure 5.1: Multi-Class Classification Problem

5.2 Fitting

How we can design the model and its characteristics? We want to find a good approximation that fit well our data, for both regression and classification, and have a model able to perform well (generalize) with unseen data.

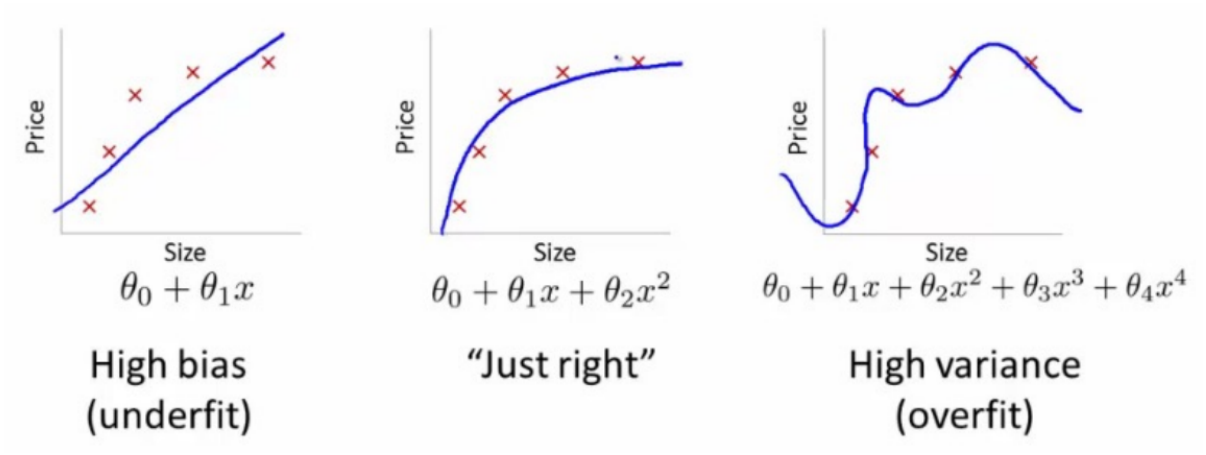


Figure 5.2: Fitting Regression

5.2.1 Overfitting and Underfitting

What we want is a model able to generalize from its experience (training examples), but the aim of Machine Learning is creating a model that perform well also on new (unseen) data, after having trained on data set.

- **Underfitting**: the model doesn't fit well the training data, because it's too simple and not able to learn. It performs bad both on training and test data.
- **Overfitting**: the model performs well on training data but not on unseen data, because it's complex and not able to generalize. The model memorize training data rather than learning how to generalize them

5.2.2 Expected Squared Error and its relationship with expected value

Expected Value

The expected value of a random variable X

$$E[X] = \sum_{i=1}^{m} p_i \cdot x_i$$

As we know, the output Y is given by our model evaluated with the example x plus an error of the model

$$Y = f(x) + e$$

Mean Squared Error - MSE

We assume our hypothesis $h(X)$ to approximate $f(X)$. The expected squared error of our hypothesis is

$$Err(f(X)) = E[(Y - h(X))^2]$$

5.2.3 Bias and Variance

Bias

Bias is the systematic deviation of the estimator. It represents the mean of the error in the predicted values:

$$Bias(h(X), f(X)) = (E[h(X)] - f(X))^2$$

Variance

Variance represents the mean of the variations of the predicted values with respect to the mean of the predicted values. It gives us an idea of the mean of the variations of the errors of the predicted values with reference to Y.
With the variance we can measure the distribution of the error relative to the expected value.
Variance is related to the stability of the model when we give it new examples.

Bias and Variance graphically

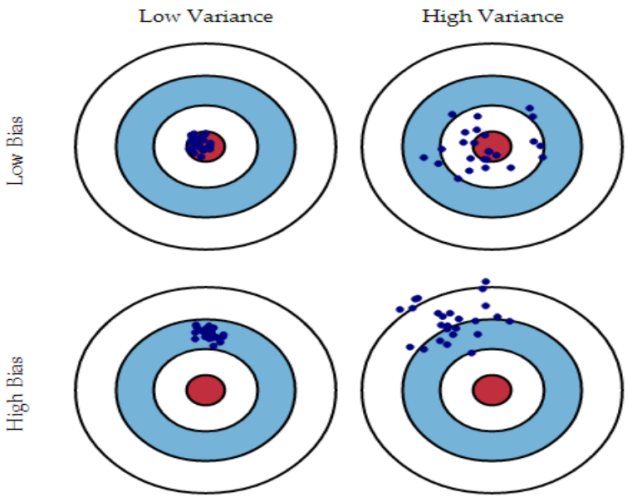


Figure 5.3: Bias and Variance Graphical Representation

The data we have to predict are in the red circle. If our predictions are grouped then the variance is low. If our predictions are far away from right values then the bias is high.

For example, in the example on the top-left we can see that our predictions are almost all in the red circle (low bias) and they're grouped (low variance).
Instead, in the example on bottom-right our predictions are far away from the red circle (high bias) and they're scattered (high variance).

5.2.4 Bias and Variance trade-off

We assume to have in infinite number of Dataset D, that represents all the possible combinations of the training samples that could feed our model. They are all possible subsets of the corresponding random variable.

We fix the model and train it each time with a different D_i. We get an infinite number of hypothesis $h^{D_i}(x)$.
Each training sample is in the form $< x^{(i)}, y^{(i)} >$ where y is the sum of $f(x^{(i)})$ and a Gaussian error with mean 0 and variance σ_e.
Each hypothesis $h^{(D_i)}(x)$ will be affected by an error in predicting values compared to the ground truth. We can model this error in the form of a Mean Squared Error (MSE):

$$MSE\left(h^{(D_i)}(x)\right) = E_x[(h^{(D_i)}(x) - f(x))^2]$$

We want to estimate the expectation of the error of all possible D_i. We want to compute the MSE of each hypothesis and then we compute the overall mean.
Performing this operation over all the infinite datasets is not feasible, but we can compute the expected value of the mean as the expectation of the MSE over all the datasets.
This expectation is named **Generalization Error (GER)**:

$$GER = E_D[MSE] = E_D[E_x[(h^{(D_i)}(x) - f(x))^2]]$$

Now, due to the linearity of expectation operator we can swap the expectations

$$= E_x[E_D[(h^{(D)}(x^{(i)}) - f(x^{(i)}))^2]]$$

Let's focus on the inner term $E_D[(h^{(D)}(x^{(i)}) - f(x^{(i)}))^2]$.
We can define a new quantity, **the best estimation** of $f(x^{(i)})$, evaluating each hypothesis with the same training sample $x^{(i)}$ (we're not partitioning our dataset so a single training example can appear in more than one dataset), and then we compute the mean of all the values

$$\overline{h}(x^{(i)}) = E_D[h^{(D)}(x^{(i)})]$$

Now, we can perform a sum zero operation by adding and subtracting $\overline{h}(x^{(i)})$ from the inner term we focused before.

$$E_D\left[\left(h^{(D)}(x) - \overline{h}(x^{(i)}) + \overline{h}(x^{(i)}) - f(x^{(i)})\right)^2\right]$$

We can solve the power (considering the first two members and the second two members) and exploit the linearity of E

$$E_D\left[\left(h^{(D)}(x) - \overline{h}(x^{(i)})\right)^2\right] + E_D\left[\left(\overline{h}(x^{(i)}) - f(x^{(i)})\right)^2\right] + E_D\left[2\left(h^{(D)}(x^{(i)}) - \overline{h}(x^{(i)})\right)\left(\overline{h}(x^{(i)}) - f(x^{(i)})\right)\right]$$

Let's focus on the first term:

$$E_D\left[\left(h^{(D)}(x^{(i)}) - \overline{h}(x^{(i)})\right)^2\right] = Var(h^{(D)}(x^{(i)}))$$

Now let's focus on the second term:

$$E_D\left[\left(\overline{h}(x^{(i)}) - f(x^{(i)})\right)^2\right] = (\overline{h}(x^{(i)}) - f(x^{(i)}))^2$$

We can ignore the expected values because we don't care about the dataset.

And let's consider the third term:

$$\begin{aligned} &2(\overline{h}(x^{(i)}) - f(x^{(i)}))E_D[h^{(D)}(x^{(i)}) - \overline{h}(x^{(i)})] \\ &= 2(\overline{h}(x^{(i)}) - f(x^{(i)}))(E_D[h^{(D)}(x^{(i)})] - E_D[\overline{h}(x^{(i)})]) \\ &= 2(\overline{h}(x^{(i)}) - f(x^{(i)}))(\overline{h}(x^{(i)}) - \overline{h}(x^{(i)})) = 0 \end{aligned}$$

Finally, we can move back to the MSE definition we gave before:

$$MSE(h^{(D)}(x^{(i)})) = Bias^2(h^{(D)}(x^{(i)})) + Var(h^{(D)}(x^{(i)}))$$

In order to reduce Generalization Error, we should reduce the bias or the variance:

- **Bias** represents how much the best estimation is able to get close to the ground truth. An high bias for a specific model denotes it's too simple and it's not able to predict, whatever dataset you are training on
- **Variance** represents how much a single hypothesis can be different from the best estimation. The high variance means that the same model, trained with different datasets, generates very different hypothesis. The hypothesis are very complex, the may overfit and they're not able to generalize

The optimal solution would be to maintain low bias and variance, but given a model, the bias and the variance behave at the opposite.

5.2.5 Bias and Variance Complexity

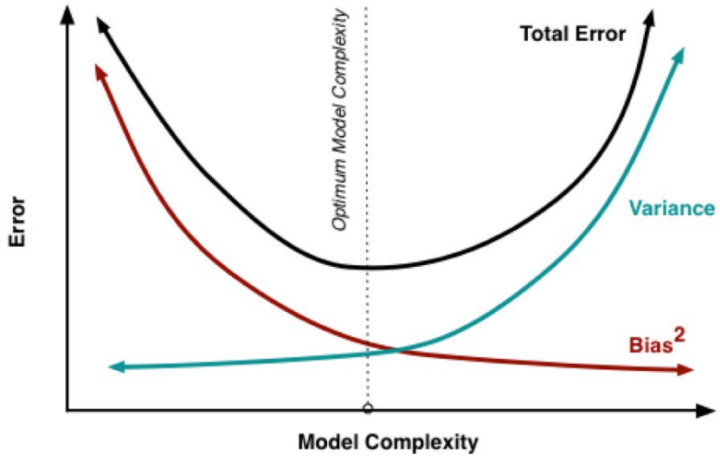


Figure 5.4: Bias and Variance with different Model Complexities

As we can see from the graphic with a simple model in terms of complexity the variance is low but the bias is high. Moving forward through the complexity of our model, the bias decrease, because we have a complex model that fits better our data, but the variance increase because the model become more sensible to smaller variations of the training data.
The total error is given by the sum of bias squared and the variance. The best trade-off is choosing the model complexity that show the minimum sum between bias and variance (in the graphic is denoted by the dotted line).

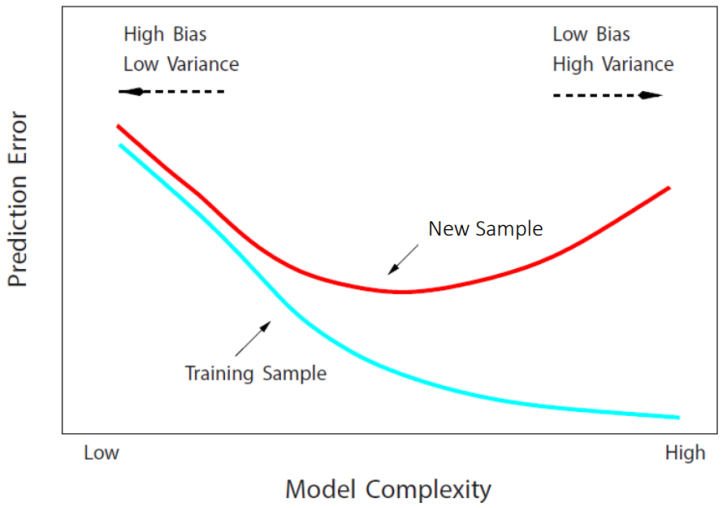


Figure 5.5: Training and Test Bias and Variance with different Model Complexities

Now let's analyze what happens to bias and variance when we consider our training samples and test samples, changing the model complexity: with a simple model for both training and test data the error is high. This is because our model isn't complex enough to fit well our data.
Increasing the complexity of our model, we can see also with the graphic, that the prediction error starts to decrease for both training and test sample: for training samples the error continue to decrease until reaches 0. But in this situation we're surely dealing with overfitting problem. For test samples instead, the error decreases until it reaches the minimum value, then it starts to increase. This is due to the fact that our model starts to memorize data instead of making a generalization which can perform well also with unseen data. **Remember**: the intersection between the training samples and the test samples **must be an empty set**.

5.3 Regularization

When we choose the complexity of the model we have to deal with overfitting and underfitting. How we can we reduce overfitting? Basically, we can reduce the number of the features (model complexity) selecting manually which feature to keep or let an algorithm decide.

Another technique is called regularization: basically we keep all the features but we reduce the magnitude of the parameters. This technique works well when we have a lot of features, each of which contributes a bit to predicting y.

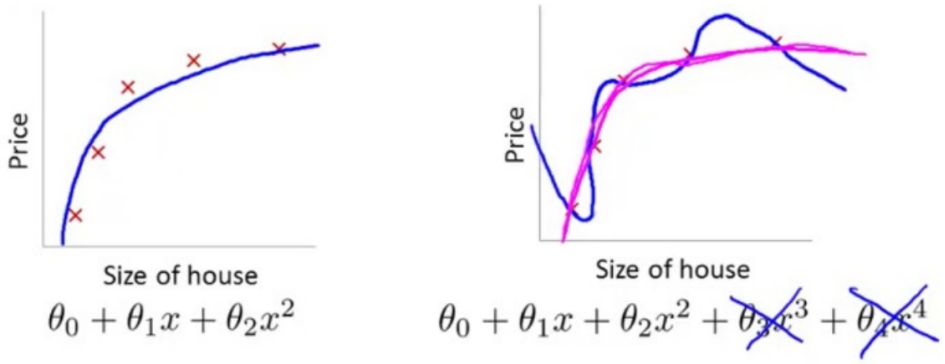


Figure 5.6: Regularization Intuition

In the image above we can see that, with a second degree polynomial we have a good approximation of our data. By increasing the degree of the polynomial we can see that new grades seems to take control of the function, changing completely its behaviour.
What we can do is to reduce the magnitude of the feature parameters with high degree terms and, at the same time, reduce the model error, in order to make the curve smoother.
If we add an heavy term to the high degree parameters, we are imposing the minimization function to make them really small:

$$\min_\theta J(\theta) = \min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})^2 + 1000\theta_3 + 1000\theta_4$$

How do we know which are the parameters we need to reduce? We cannot know it a priori, therefore we need to **reduce the magnitude/value of all the parameters** of our model.

How do we can keep the magnitude of parameters as low as possible? We can use an hyper-parameter lambda, called **regularization function** and multiply it by the squared L2-norm (which is the sum of the squares of the parameters, without considering the first one). By minimizing the cost function we also reduce the magnitude of all the parameters in order to make them have the same magnitude.
In fact, if we have small values for the parameters then we have simple hypothesis, which are less prone to overfitting.

$$J(\theta) = \frac{1}{2m} \left(\sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})^2 \right)$$

$$J(\theta) = \frac{1}{2m} \left(\sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})^2 + \lambda ||\theta||_2^2 \right)$$

$$J(\theta) = \frac{1}{2m} \left(\sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{i=1}^{n} \theta_j^2 \right)$$

The bigger the regularization factor is, the smaller will be the theta and smoother will be the curve.
But, if theta is too big, the L2-norm term "takes control" of the minimization problem which become the minimization of our parameters. However, this doesn't allow us to have a model which fits well our training data, running into an underfitting problem.

5.3.1 Regularized Linear Regression

Let's see what happens if we consider regularization in linear regression. We saw that our new cost function is

$$J(\theta) = \frac{1}{2m}\left(\sum_{i=1}^{m}(h(^{(i)}) - y^{(i)})^2 + \lambda\sum_{i=1}^{n}\theta_j^2\right)$$

We want to derive the cost function to modify the update rule in the gradient descent: until convergence we compute the updates below

$$\begin{cases} \theta_0 = \theta_0 - \alpha\dfrac{1}{m}\displaystyle\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)} \\ \theta_j = \theta_j - \alpha\left[\dfrac{1}{m}\displaystyle\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \dfrac{\lambda}{m}\theta_j\right] \quad j = 1, ..., n \end{cases}$$

The update function can be rewritten as

$$\theta_j = \theta_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

The second part is the same as before the regularization, but we update theta multiplying it by a quantity which is smaller than 1 (the first part). We need this term to be close to one, so we need to work on hyper-parameters (learning rate and regularization function). λ is high (not too much as we said before) and m is large. The only hyper-parameter that can be adjusted is α which should have a value much smaller than 1.

5.3.2 Regularization for Logistic Regression

In logistic regression we have something similar to linear regression. The new cost function is

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log h_\theta(x^{(i)}) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))\right) + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

We want to derive the cost function to modify the update rule in the gradient descent: until convergence we compute the updates below

$$\begin{cases} \theta_0 = \theta_0 - \alpha\dfrac{1}{m}\displaystyle\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)} \\ \theta_j = \theta_j - \alpha\left[\dfrac{1}{m}\displaystyle\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \dfrac{\lambda}{m}\theta_j\right] \quad j = 1, ..., n \end{cases}$$

Chapter 6

Lesson 6 - 14/10/2024

6.1 Second Hands-On Lesson (LAB. 2)