Foundation of Machine Learning: Part 1

N'dah Jean Kouagou

December 8, 2019

Contents

1 Linear regression

In Statistics, linear regression is a linear approach to modeling the relationship between a dependent (continuous) variable and one or more explanatory variables. In this section we consider a dataset $\mathcal{D} = \{(x^{(i)}, y_i)_{i=1,\dots,n}\} \subseteq \mathbb{R}^m \times \mathbb{R}$, where $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_m^{(i)})^T$, m is the number of features.

1.1 The hypothesis

The objective of a linear regression is to find a plane $\mathcal{H}_\theta \subset \mathbb{R}^m \times \mathbb{R}$ (a line if in dimension 1) of equation $h_\theta(x) = \theta^T x$, $\theta = (\theta_0, \theta_1, \dots, \theta_m)^T$ that is "as close as possible" to $\mathcal{D}$. We call $h_\theta(x)$, the hypothesis. Note that by "as close as possible" we mean the $\mathcal{H}_\theta$ that minimizes the quantity $\frac{1}{n}\sum_{i=1}^n (y_i - h_\theta(\bar{x}^{(i)}))^2$, where $\bar{x}^{(i)} = (1, x_1^{(i)}, x_2^{(i)}, \dots, x_m^{(i)})$.

1.2 The criterion or loss function

The quantity $l(\theta) = \frac{1}{n}\sum_{i=1}^n (y_i - h_\theta(\bar{x}^{(i)}))^2$ is called the loss or cost function or again the criterion.
Our next task is to minimize $l(\theta)$, that is, find $\theta^* = \underset{\theta}{Argmin}\ l(\theta)$.

1.3 Learning algorithm

Here, we present four learning algorithms that can be used to find θ^*.

1.3.1 Gradient descent

We compute the gradient $\nabla l(\theta)$.

$$\nabla l(\theta) = \begin{pmatrix} \frac{\partial l}{\partial \theta_0}(\theta) \\ \frac{\partial l}{\partial \theta_1}(\theta) \\ \vdots \\ \frac{\partial l}{\partial \theta_m}(\theta) \end{pmatrix}$$

$$= \begin{pmatrix} \frac{\partial}{\partial \theta_0}\frac{1}{n}\sum_{i=1}^n (y_i - h_\theta(\bar{x}^{(i)}))^2 \\ \frac{\partial}{\partial \theta_1}\frac{1}{n}\sum_{i=1}^n (y_i - h_\theta(\bar{x}^{(i)}))^2 \\ \vdots \\ \frac{\partial}{\partial \theta_m}\frac{1}{n}\sum_{i=1}^n (y_i - h_\theta(\bar{x}^{(i)}))^2 \end{pmatrix}$$

$$
\begin{aligned}
&= \begin{pmatrix} \frac{1}{n}\frac{\partial}{\partial\theta_0}\sum_{i=1}^{n}(y_i - \theta^T\bar{x}^{(i)})^2 \\ \frac{1}{n}\frac{\partial}{\partial\theta_1}\sum_{i=1}^{n}(y_i - \theta^T\bar{x}^{(i)})^2 \\ \vdots \\ \frac{1}{n}\frac{\partial}{\partial\theta_m}\sum_{i=1}^{n}(y_i - \theta^T\bar{x}^{(i)})^2 \end{pmatrix} \\
&= \begin{pmatrix} 2\frac{1}{n}\sum_{i=1}^{n}(y_i - \theta^T\bar{x}^{(i)}) \\ 2\frac{1}{n}\sum_{i=1}^{n}(y_i - \theta^T\bar{x}^{(i)})x_1^{(i)} \\ \vdots \\ 2\frac{1}{n}\sum_{i=1}^{n}(y_i - \theta^T\bar{x}^{(i)})x_m^{(i)} \end{pmatrix} \\
\nabla l(\theta) &= \frac{2}{n}\begin{pmatrix} \sum_{i=1}^{n}(y_i - \theta^T\bar{x}^{(i)}) \\ \sum_{i=1}^{n}(y_i - \theta^T\bar{x}^{(i)})x_1^{(i)} \\ \vdots \\ \sum_{i=1}^{n}(y_i - \theta^T\bar{x}^{(i)})x_m^{(i)} \end{pmatrix}
\end{aligned}
$$

The gradient descent algorithm is then the following:

- Choose a learning rate lr and a tolerance eps
- Initialize $\theta = \theta^0$
- Repeat until $Norm(\theta^{j-1} - \theta^j) < eps$:
    - $\theta^j = \theta^{j-1} - lr\nabla l(\theta^{j-1})$

1.3.2 Mini-batch Gradient Descent

This algorithm is quite similar to the previous one. The only difference is that instead of considering all the examples in the gradient, we only consider few of them chosen randomly.
The algorithm is as follows:

- Choose a learning rate lr and a tolerance eps
- Initialize $\theta = \theta^0$
- Repeat until $Norm(\theta^{j-1} - \theta^j) < eps$:
    - Randomly split the training data $\mathcal{D}$ into p mini-batches $\mathcal{B}_i, i = 1, \ldots, p$ of size $\frac{n}{p}$.
    - For each mini-batch $\mathcal{B}_i$, compute the corresponding gradient

$$\nabla_{\mathcal{B}_i} = \frac{2p}{n} \begin{pmatrix} \sum_{(x,y)\in\mathcal{B}_i}(y - \theta^T\bar{x}) \\ \sum_{(x,y)\in\mathcal{B}_i}(y - \theta^T\bar{x})x_1 \\ \vdots \\ \sum_{(x,y)\in\mathcal{B}_i}(y - \theta^T\bar{x})x_m \end{pmatrix}$$

$\star\ \theta^j = \theta^{j-1} - lr\nabla_{\mathcal{B}_i}(\theta^{j-1})$

1.3.3 Stochastic Gradient Descent

This algorithm is the most used in machine learning especially when we deal with a large training data. The algorithm is the following:

- Choose a learning rate lr and a tolerance eps
- Initialize $\theta = \theta^0$
- Repeat until $Norm(\theta^{j-1} - \theta^j) < eps$:
  - $\star$ Shuffle the training dataset $\mathcal{D}$.
  - $\star$ For each data point (x, y), compute the corresponding gradient

$$\nabla_{(x,y)} = \begin{pmatrix} (y - \theta^T\bar{x}) \\ (y - \theta^T\bar{x})x_1 \\ \vdots \\ (y - \theta^T\bar{x})x_m \end{pmatrix}$$

$\star\ \theta^j = \theta^{j-1} - lr\nabla_{(x,y)}(\theta^{j-1})$

2 Logistic regression

Logistic regression is used for classification. Here, we investigate a binary classification and consider a dataset $\mathcal{C} = \{(x^{(i)}, y_i)_{i=1,\dots,n}\} \subseteq \mathbb{R}^m \times \{0, 1\}$, where $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_m^{(i)})^T$, m is the number of features or explanatory variables.

2.1 The hypothesis

We first define a function called sigmoid function which we denote by σ: $\sigma(x) = \frac{1}{1+\exp(-x)}$. The hypothesis for the logistic regression model is then: $h_\theta(x) = \frac{1}{1+\exp(-\theta^T x)}$, where θ is the vector (of parameters) to learn and fit to the training data. Note that to include the bias (intercept) term, we add a column of ones to the features $x^{(i)}_{i=1,\dots,n}$ and we denote by $\bar{x}^{(i)}_{i=1,\dots,n}$ the new features.

2.2 The criterion

We assume that the random variables $\{y/\bar{x}^{(i)};\theta\}_{i=1,\dots,n}$ are independent and follow Bernoulli distribution of parameter $h_\theta(\bar{x}^{(i)})$, that is
$P(y/\bar{x}^{(i)};\theta) = h_\theta(\bar{x}^{(i)})^y(1-h_\theta(\bar{x}^{(i)}))^{1-y}$.
Let $\mathcal{L}(\theta) = \prod_{i=1}^n P(y_i/\bar{x}^{(i)};\theta)$.
The objective is to find the parameter θ that maximize $\mathcal{L}(\theta)$.
Maximizing $\mathcal{L}(\theta)$ is equivalent to maximizing $l(\theta) = \frac{1}{n}\log(\mathcal{L}(\theta))$.
We have: $l(\theta) = \frac{1}{n}\sum_{i=1}^n y_i \log(h_\theta(\bar{x}^{(i)})) + (1-y_i)\log(1-h_\theta(\bar{x}^{(i)}))$.

2.3 Learning algorithm

Using $\sigma'(x) = \sigma(x)(1-\sigma(x))$, we can compute the gradient of $l(\theta)$ as follows:

$$\nabla l(\theta) = \begin{pmatrix} \frac{\partial l}{\partial \theta_0}(\theta) \\ \frac{\partial l}{\partial \theta_1}(\theta) \\ \vdots \\ \frac{\partial l}{\partial \theta_m}(\theta) \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{n}\frac{\partial}{\partial \theta_0}\sum_{i=1}^n y_i \log(h_\theta(\bar{x}^{(i)})) + (1-y_i)\log(1-h_\theta(\bar{x}^{(i)})) \\ \frac{1}{n}\frac{\partial}{\partial \theta_1}\sum_{i=1}^n y_i \log(h_\theta(\bar{x}^{(i)})) + (1-y_i)\log(1-h_\theta(\bar{x}^{(i)})) \\ \vdots \\ \frac{1}{n}\frac{\partial}{\partial \theta_m}\sum_{i=1}^n y_i \log(h_\theta(\bar{x}^{(i)})) + (1-y_i)\log(1-h_\theta(\bar{x}^{(i)})) \end{pmatrix}$$

$$= \frac{1}{n}\begin{pmatrix} \sum_{i=1}^n \frac{y_i}{h_\theta(\bar{x}^{(i)})}\frac{\partial}{\partial \theta_0}h_\theta(\bar{x}^{(i)}) + \frac{1-y_i}{1-h_\theta(\bar{x}^{(i)})}\frac{\partial}{\partial \theta_0}(-h_\theta(\bar{x}^{(i)})) \\ \sum_{i=1}^n \frac{y_i}{h_\theta(\bar{x}^{(i)})}\frac{\partial}{\partial \theta_1}h_\theta(\bar{x}^{(i)}) + \frac{1-y_i}{1-h_\theta(\bar{x}^{(i)})}\frac{\partial}{\partial \theta_1}(-h_\theta(\bar{x}^{(i)})) \\ \vdots \\ \sum_{i=1}^n \frac{y_i}{h_\theta(\bar{x}^{(i)})}\frac{\partial}{\partial \theta_m}h_\theta(\bar{x}^{(i)}) + \frac{1-y_i}{1-h_\theta(\bar{x}^{(i)})}\frac{\partial}{\partial \theta_m}(-h_\theta(\bar{x}^{(i)})) \end{pmatrix}$$

$$= \frac{1}{n}\begin{pmatrix} \sum_{i=1}^n \frac{y_i}{\sigma(\theta^T\bar{x}^{(i)})}\frac{\partial}{\partial \theta_0}\sigma(\theta^T\bar{x}^{(i)}) + \frac{1-y_i}{1-\sigma(\theta^T\bar{x}^{(i)})}\frac{\partial}{\partial \theta_0}(-\sigma(\theta^T\bar{x}^{(i)})) \\ \sum_{i=1}^n \frac{y_i}{\sigma(\theta^T\bar{x}^{(i)})}\frac{\partial}{\partial \theta_1}\sigma(\theta^T\bar{x}^{(i)}) + \frac{1-y_i}{1-\sigma(\theta^T\bar{x}^{(i)})}\frac{\partial}{\partial \theta_1}(-\sigma(\theta^T\bar{x}^{(i)})) \\ \vdots \\ \sum_{i=1}^n \frac{y_i}{\sigma(\theta^T\bar{x}^{(i)})}\frac{\partial}{\partial \theta_m}\sigma(\theta^T\bar{x}^{(i)}) + \frac{1-y_i}{1-\sigma(\theta^T\bar{x}^{(i)})}\frac{\partial}{\partial \theta_m}(-\sigma(\theta^T\bar{x}^{(i)})) \end{pmatrix}$$

$$\nabla l(\theta) = \frac{1}{n} \begin{pmatrix} \sum_{i=1}^{n} y_i - \sigma(\theta^T \bar{x}^{(i)}) \\ \sum_{i=1}^{n} (y_i - \sigma(\theta^T \bar{x}^{(i)})) x_1^{(i)} \\ \vdots \\ \sum_{i=1}^{n} (y_i - \sigma(\theta^T \bar{x}^{(i)})) x_m^{(i)} \end{pmatrix}$$

We can now use the stochastic gradient descent (SGD) algorithm to minimize $l(\theta)$. For one example $(\bar{x}^{(i)}, y_i)$, the gradient is :

$$\nabla_i(\theta) = \begin{pmatrix} y_i - \sigma(\theta^T \bar{x}^{(i)}) \\ (y_i - \sigma(\theta^T \bar{x}^{(i)})) x_1^{(i)} \\ \vdots \\ (y_i - \sigma(\theta^T \bar{x}^{(i)})) x_m^{(i)} \end{pmatrix}$$

The stochastic gradient algorithm to maximize $l(\theta)$ is as follows:

- Choose a learning rate lr and a tolerance eps
- Initialize $\theta = \theta^0$
- Repeat until $Norm(\theta^{j-1} - \theta^j) < eps$:
    - ⋆ Shuffle the training dataset $\mathcal{C}$.
    - ⋆ For each data point $(x^{(i)}, y_i)$, update θ:
        - ⋆ $\theta^j = \theta^{j-1} + lr \nabla_i(\theta)$

3 Empirical Risk Minimization

Given a dataset $\mathcal{D} = \{(\mathbf{x}, \mathbf{y})\} = \{(x^{(i)}, y_i)_{i=1,\dots,n}\}$ and a hypothesis $h_\theta(x)$, we want to find θ that minimizes $\mathbb{E}_{(x,y)\sim\mathcal{P}}[(h_\theta(x) - y)^2]$ (this is the **True Risk Minimization**). But since we do not know $\mathcal{P}$, the best thing we can do is to minimize $\mathcal{R}_{h_\theta} = \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y)^2 = \mathbb{E}_{(x,y)\in\mathcal{D}}[(h_\theta(x) - y)^2]$. The optimization problem $\min_\theta \mathcal{R}_{h_\theta}$ is called **Empirical Risk Minimization**. If θ^* is the exact solution to the True Risk Minimization problem, then we can write $y = h_{\theta^*}(x) + \epsilon$, where ϵ follows the normal distribution $\mathcal{N}(0, \sigma^2)$. Therefore, we have the following:

$$
\begin{aligned}
\mathcal{R}_{h_\theta} &= \mathbb{E}[(\mathbf{y} - h_\theta(\mathbf{x}))^2] \\
&= \mathbb{E}[(h_{\theta^*}(\mathbf{x}) + \epsilon - h_\theta(\mathbf{x}))^2] \\
&= \mathbb{E}[\epsilon^2 + 2\epsilon(h_{\theta^*}(\mathbf{x}) - h_\theta(\mathbf{x})) + (h_{\theta^*}(\mathbf{x}) - h_\theta(\mathbf{x}))^2] \\
&= \mathbb{E}(\epsilon^2) + \mathbb{E}[(h_{\theta^*}(\mathbf{x}) - h_\theta(\mathbf{x}))^2] + 2\mathbb{E}(\epsilon)\mathbb{E}[h_{\theta^*}(\mathbf{x}) - h_\theta(\mathbf{x})], \text{ because } \epsilon \text{ and } \mathbf{x} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{are independently distributed} \\
&= \sigma^2 + \mathbb{E}[(h_{\theta^*}(\mathbf{x}) - h_\theta(\mathbf{x}))^2], \text{ beacuse } \mathbb{E}(\epsilon) = 0 \\
&= \underbrace{\sigma^2}_{Noise} + \underbrace{(\mathbb{E}[(h_{\theta^*}(\mathbf{x}) - h_\theta(\mathbf{x}))])^2}_{Bias} + \underbrace{Var(h_{\theta^*}(\mathbf{x}) - h_\theta(\mathbf{x}))}_{Variance} \\
\mathcal{R}_{h_\theta} &= \underbrace{\sigma^2}_{Noise} + \underbrace{(\mathbb{E}[(h_{\theta^*}(\mathbf{x}) - h_\theta(\mathbf{x}))])^2}_{Bias} + \underbrace{Var(h_{\theta^*}(\mathbf{x}) - h_\theta(\mathbf{x}))}_{Variance}
\end{aligned}
$$

The empirical risk can be divided into three terms: the noise, the bias and the variance.
A model with high variance is said to be over-fitting and a model with high bias is under-fitting.

4 Newton's Method for finding the zeros of a function

To minimize a cost function $l(\theta)$ we usually seek for θ^* such that $\nabla l(\theta^*) = \mathbf{0}$. So we can use Newton's method to find the zero (s) of the gradient ∇l.
Newton's algorithm for finding the zero (s) of ∇l is the following:

- Choose a tolerance *eps*
- Initialize $\theta = \theta^0$
- Repeat until $Norm(\theta^{j-1} - \theta^j) < eps$:
    - $\theta^j = \theta^{j-1} - (\mathbf{H}l(\theta^{j-1}))^{-1}\nabla l(\theta^{j-1})$, $\mathbf{H}l$ is the Hessian matrix of l.

5 K-Fold cross-validation

K-Fold cross-validation is a procedure used to estimate the skill of models. It allows selecting the best model for a given data.
Given the dataset $\mathcal{D}$ of size n, the method operates as follows:

- Randomly split $\mathcal{D}$ into k disjoint subsets $\mathcal{D}_{i,\ i=1,\dots,k}$ of size $\frac{n}{k}$ each

• For each model $\mathcal{M}_i$:

⋆ For each $j = 1, \ldots, k$

⋆ Train $\mathcal{M}_i$ on $\mathcal{D} - \mathcal{D}_j$, obtain the model $\mathcal{M}_i^{(j)}$

⋆ Test $\mathcal{M}_i^{(j)}$ on $\mathcal{D}_j$, obtain the error $err_i^{(j)}$

• The generalization error of $\mathcal{M}_i$ is the average of $err_i^{(j)},\ j = 1, \ldots, k$.

The best model to choose is the one with the lowest generalization error.

6 Feature selection

This is a very important concept in Machine Learning. A model can only perform very well when it is using the best features.
We have several ways to select the best features given a data. Here,1 we present two methods: Wrapper methods and Filter methods.

6.1 Wrapper methods

Let $\mathcal{D} = \{(x^{(i)}, y_i)_{i=1,\ldots,n}\}$ be a dataset with d features.

6.1.1 Forward Wrapper method

The forward Wrapper method for selecting the k best features in $\mathcal{D}$ is the following:

• Initialize $\mathcal{F} = \emptyset$

• Repeat Until $|\mathcal{F}| = k$:

(a) ⋆ For i in $\{1, 2, \ldots, d\}$:

⋆ If i not in $\mathcal{F}$, let $\mathcal{F}_i = \mathcal{F} \cup \{i\}$

⋆ Use cross-validation to evaluate $\mathcal{F}_i$

(b) ⋆ Set $\mathcal{F}$ to be the best subset $\mathcal{F}_i$ found at (a)

• Select the best subset that was found during the entire procedure

6.1.2 Backward Wrapper method

The backward Wrapper method for selecting the k best features in $\mathcal{D}$ is the following:

• Initialize $\mathcal{F} = \{1, 2, \ldots, d\}$

• Repeat Until $|\mathcal{F}| = k$:

(a) ⋆ For i in $\{1, 2, \ldots, d\}$:

⋆ If i in $\mathcal{F}$, let $\mathcal{F}_i = \mathcal{F} - \{i\}$

⋆ Use cross-validation to evaluate $\mathcal{F}_i$

(b) ⋆ Set $\mathcal{F}$ to be the best subset $\mathcal{F}_i$ found at (a)

• Select the best subset that was found during the entire procedure

6.2 Filter method

Filter method uses the Kulback Liebler divergence to select the features that are more correlated with the target. $\mathcal{D} = \{(x^{(i)}, y_i)_{i=1,\ldots,n}\}$ is a dataset with d features. The method computes for each j in $\{1, 2, \ldots, d\}$, the mutual information MI:

$$\begin{aligned} MI(x^{(j)}, y) &= KL(P(x^{(j)}, y)||P(x^{(j)})P(y)) \\ &= \sum_{i=1}^{n} P(x_i^{(j)}, y_i) \log(\frac{P(x_i^{(j)}, y_i)}{P(x_i^{(j)})P(y_i)}) \end{aligned}$$

Features with the highest mutual information are the best features.