Linear Models

Advanced Machine Learning

Yadder Aceituno

November 3, 2025

1 Content of Course

1. Linear or Logistic
2. Decision Trees
3. SVM
4. Neural Networks
5. Representational Issues
6. Deep Learning
7. Gaussian Processes

2 Week 1 - Lecture 1

2.1 Linear Regression

Given a set of data points $< x_i, y_i >$, the goal is to find the weights θ such that $X_i\theta = Y_i$.
Where:

$$X = \langle 1, x_1, x_2, \ldots, x_m \rangle$$

$$Y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m$$

2.2 Maximum Likelihood Estimation

Assuming that data points are independent and identically distributed, the goal is to find the parameters θ that maximize the likelihood of the data with the next equation:

$$\theta = argmax_{\theta_0,\theta_1,\ldots,\theta_m} \prod_{i=1}^{N} p(y_i|x_i;\theta)$$

Where

$$p(y_i|x_i;\theta) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i-\mu_i)^2}{2\sigma^2}}$$

$$\mu_i = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m$$

If we assume that the data points are independent and identically distributed, then the probability of observing all data points is the product of their individual probabilities. This is the joint likelihood. This method can derive into the Mean Squared Error (MSE). This can be proven as follows:

Because it's equivalent to maximize the likelihood, we can maximize the log-likelihood:

$$\theta = argmax_{\theta_0,\theta_1,\ldots,\theta_m} \log\left(\prod_{i=1}^{N} p(y_i|x_i;\theta)\right)$$

Because the logaritm of a product is the sum of the logaritms, we can rewrite the equation as:

$$\theta = argmax_{\theta_0,\theta_1,\ldots,\theta_m} \sum_{i=1}^{N} \log p(y_i|x_i;\theta)$$

Further we can transform the product in the logaritm into a sum of logaritms:

$$\theta = argmax_{\theta_0,\theta_1,\ldots,\theta_m} \sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \log\left(e^{-\frac{(y_i-\mu_i)^2}{2\sigma^2}}\right)$$

Here, the first term in the sum is a constant that doesn't depend on θ, so it can be ignored. In the second term, the logaritm of the exponential is the exponential itself, so we can rewrite the equation as:

$$\theta = argmax_{\theta_0,\theta_1,\ldots,\theta_m} \sum_{i=1}^{N} -\frac{(y_i-\mu_i)^2}{2\sigma^2}$$

Which is the same as minimizing the Mean Squared Error (MSE).

2.3 Normal Equation

Method to find the optimal parameters. The equation for finding the optimal parameters is:

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

How we got this equation?

First we need to define the cost function that we want to minimize, which is the Mean Squared Error (MSE):

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

Writing it in matrix form:

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

So, we need to minimize the cost function $J(\theta)$. We can do this by taking the derivative of $J(\theta)$ with respect to θ and setting it to zero:

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

So, our gradient is:

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

If we set the gradient to zero, we get:

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

Solving for θ:

$$X^T X\theta = X^T y$$

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

And that's the **normal equation**.

2.4 Gradient Descent

Iterative method to find the min/max optimal parameters of a function.

The steps are:

1. Start with random parameters
2. Iteratively update them to minimize the cost function.
3. Keep changing the parameters until the cost function is minimized.

The equation to repeat until convergence is:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) x_{ij}$$

Where α is the learning rate. And x_{ij} is the j-th feature of the i-th training example. There's an strategy where the learning rate can start with a high value and then decrease over time.

Some strategies:

- Batch Gradient Descent Updates the parameters using the entire dataset. Computationally expensive.
- Stochastic Gradient Descent (SGD) Updates the parameters using one training example at a time. Computationally efficient but less smooth convergence.
- Mini-batch Gradient Descent Updates the parameters using a small subset of the dataset. Computationally efficient and smooth convergence.

Learning rate

If the learning rate is too small, the algorithm will take a long time to converge. If the learning rate is too large, the algorithm may not converge.

Overfitting

High number of features means high number of heights

- Reduce the number of features
- Regularization

2.5 Regularization

Imagine you train a linear regression model, but your model fits too closely to your training data — it starts capturing noise instead of the real underlying pattern. That's called overfitting:

- Good on training data
- Poor on new (unseen) data

Overfitting often happens when:

- You have too many features relative to your number of examples.
- Some features are highly correlated or not informative.

Regularization is a technique to discourage complexity in the model by adding a penalty to the cost function. This penalty term shrinks the coefficients θ_j, keeping them smaller and preventing the model from depending too heavily on any single feature.

The cost function for linear regression with regularization is:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

In matrix form:

$$J(\theta) = \frac{1}{2m} \left[(X\theta - y)^T (X\theta - y) + \lambda \theta^T \theta \right]$$

Where λ is the regularization parameter.

3 Logistic Regression

Cast a binary classification problem as a regression problem. It uses the sigmoid function to map the output to a probability.

Sigmoid function: $\sigma(z) = \frac{1}{1+e^{-z}}$

Cost function:

$$Cost(h_\theta(x), y) = -y \log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))$$

4 Regression as a shallow network

Same math as modern

Summary: - Linear Regression: $h_\theta(x) = \theta_0 + \theta_1 x$ - Logistic Regression: $h_\theta(x) = \sigma(\theta_0 + \theta_1 x)$ - Softmax regression vs k-binary classifiers

5 Week 1 - Lecture 2

Neural networks have the same math as linear regression. There are different network architectures:

- Multiple hidden layers, also called multilayer perceptron
- Recurrent networks, which gives a memory to the network

For learning the weights, we use gradient descent similar to linear regression.

For calculating the error in each neuron, we use the same cost function as in linear regression. For linear regression with a sigmoid activation function, the cost function is:

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

For updating the weights, we use gradient descent:

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

Here, the error term is calculated with the next function:

$$\delta_j = (h_\theta(x_i) - y_i)$$

5.1 Hopfield Network

This is a network where the neurons interact with each other in a symmetric way. It means that weights are the same for both directions.

Every neuron updates its state based on the states of its neighbors. This is done iteratively, one neuron at a time. The function that updates the state of a neuron is:

$$s_i = \sum_{j=1}^{n} w_{ij}s_j - \theta_i$$