Unit 02 - Nonlinear Classification, Linear regression, Collaborative Filtering

Lecture 5. Linear Regression

1. Unit 2 Overview

Building up from the previous unit, in this unit we will introduce:

- linear regression (output a number in R)
- non-linear classification methods
- recommender problems (sometime called collaborative filtering problems)

2. Objectives

At the end of this lecture, you will be able to

- write the training error as least squares criterion for linear regression
- use stochastic gradient descent for fitting linear regression models
- solve closed-form linear regression solution
- identify regularization term and how it changes the solution, generalization

3. Introduction

Today we will see Linear Classification In the last unit we saw linear *classification*, where we was trying to land the mapping between the feature vectors of our data ($x^{(t)} \in \mathbb{R}^d$) and the corresponding (binary) label ($y^{(t)} \in \{-1, +1\}$).

This relatively simple set up can be already used to answer pretty complex questions, like making recommendations to buy or not some stocks, where the feature vector is given by the stock prices in the last d-days.

We can extend such problem to return, instead of just a binary output (price will increase - buy, vs price will drop - sell) a more informative output on the extent of the expected variation in price.

With regression we want to predict things that are continuous in nature.

The set up is very similar to before with $x^{(t)} \in \mathbb{R}^d$. The only differences is that now we consider $y^{(t)} \in \mathbb{R}$).

The goal of our model will be to map-- to learn how to map-- feature vectors into these continuous values.

We will consider for now only linear regression:

$$f(\mathbf{x}; \theta, \theta_0) = \sum_{i=1}^{d} \theta_i x_i + \theta_0 = \theta \cdot \mathbf{x} + \theta_0$$

For compactness of the equations we will consider $\theta_0 = 0$ (we can always think of θ_0 of being just a $d+1$ dimension of θ).

Are we limiting ourself to just linear relations? No, we can still use linear classifier by doing an appropriate representation of the feature vector, by mapping into some kind of complex space and from that mapping then apply linear regression (note that at this point we will not yet talk about how we can construct this feature vector position. We will assume instead that somebody already gave us an appropriate feature vector).

3 questions to address:

1. Which would be an appropriate objective that can quantify the extent of our mistake?. How do we address the correctness of our output? In classification we had a binary error term, here it must be a range, our prediction could be "almost there" or very far.
2. How do we set up the learning algorithm. We will see two today, a numerical, gradient based ones, and a analytical, closed-form algorithm where we do not need to approximate.
3. How we do regularisation. How we perform a better generalization to be more robust when we don't have enough training data or when the data is noisy

4. Empirical Risk

Let's deal with the first question, the objective. We want to measure how much our prediction deviates from the know values of y, how far from y they are. We call our objective **empirical risk** (here denoted with R) and will be a sort of average loss of all our data points, depending from the parameter to estimate.

$$R_n(\theta) = \frac{1}{n} \sum_{t=1}^{n} \text{Loss}_h(y^{(t)} - \theta \cdot \mathbf{x}^{(t)}) = \frac{1}{n} \sum_{t=1}^{n} \frac{(y^{(t)} - \theta \cdot \mathbf{x}^{(t)})^2}{2}$$

Why squaring the deviation? Intuitively, since our training data may be noisy and the values that we record may be noisy, if it is a small deviation between our prediction and the true value, it's OK. However, if the deviation is large, we want really, truly penalize. And this is the behaviour we are getting from the squared function, that the bigger difference would actually result in much higher loss.

Note that the above equation use the squared error as loss function, but other loss functions could also be used, e.g. the hinge loss we already saw in unit 1:

$$\text{Hinge Loss}_h(z) = \begin{cases} 0 & \text{if } z \geq 1 \\ 1 - z & \text{oth.} \end{cases}$$

I will minimise this risk for the known data, but what I really want to do it so get it minimised for the unknown data I don't already see.

2 mistakes are possible:

1. **structural mistakes** Maybe the linear function is not sufficient for you to model your training data. Maybe the mapping between your training vectors and y's is actually highly nonlinear. Instead of just considering linear mappings, you should consider a much broader set of function. This is one class of mistakes.
2. **estimation mistakes** The mapping itself is indeed linear, but we don't have enough training data to estimate the parameters correctly.

There is a trade-off between these two kind of error: on one side, minimising the structural mistakes ask for a broader set of functions with more parameters, but this, at equal training set size, would increase the estimation mistakes. On the other side, minimising the estimation mistakes call for simpler set of functions, with less parameters, where however I become susceptible for structural mistakes.

In this lesson we remain commit to linear regression, and we want to minimise the empirical risk.

5. Gradient Based Approach

In this segment we will study the first of the two algorithms to implement the learning phase, the gradient based approach.

The advantage of the Empirical Risk function with the squared error as loss is that it is differentiable everywhere. Its gradient with respect to the parameter is:

$$\nabla_\theta\left(\frac{(y^{(t)}-\theta\cdot\mathbf{x}^{(t)})^2}{2}\right) = -(y^{(t)} - \theta \cdot \mathbf{x}^{(t)}) * \mathbf{x}^{(t)}$$

We will implement its stochastic variant: we start by randomly select one sample in the training set, look at its gradient, and update our parameter in the opposite direction (as we want to minimise the empirical risk).

The algorithm will then be as follow:

1. We initialise the thetas to zero
2. We randomly pick up a data pair
3. We compute the gradient and update θ as $\theta = \theta - \eta * \nabla_\theta = \theta + \eta * (y^{(t)} - \theta \cdot \mathbf{x}^{(t)}) * \mathbf{x}^{(t)}$ where η is the learning rate, influencing the size of the movement of the parameter at each iteration. We can have η constant or making it depends from the number of k iteration we already have done, e.g. $\eta = 1/(1+k)$, so to minimise the steps are we get closer to our minimum.

Note that the parameter updates at each step, not only on some "mistake" like in classification (i.e. we treat all deviations as "mistake"), and that the amount depends on the deviation itself (i.e. not of a fixed amount like in classification). Going against the gradient assure that the algorithm self-correct itself, i.e. we obtain parameters that lead to predictions closer and closer to the actual true y.

6. Closed Form Solution

The second learning algorithm we study is the closed form (analytical) solution. This is quite an exception in the machine learning field, as typically closed form solutions do not typically exist. But here the empirical risk happens to be a convex function that we can solve it exactly.

Let's compute the gradient with respect of θ, but this time of the whole empirical risk, not just the loss function:

$$R_n(\hat{\theta}) = \frac{1}{n} \sum_{t=1}^{n} \frac{(y^{(t)} - \hat{\theta} \cdot \mathbf{x}^{(t)})^2}{2}$$

$$\nabla_\theta R_n(\hat{\theta}) = \frac{1}{n} \sum_{t=1}^{n} \nabla_\theta (\frac{(y^{(t)} - \hat{\theta} \cdot \mathbf{x}^{(t)})^2}{2}) = -\frac{1}{n} \sum_{t=1}^{n} (y^{(t)} - \hat{\theta} \cdot \mathbf{x}^{(t)}) * \mathbf{x}^{(t)}$$

$$\nabla_\theta R_n(\hat{\theta}) = \frac{1}{n} \sum_{t=1}^{n} y^{(t)} * \mathbf{x}^{(t)} + \frac{1}{n} \sum_{t=1}^{n} \mathbf{x}^{(t)} * (\mathbf{x}^{(t)})^T * \hat{\theta}$$

$$\nabla_\theta R_n(\hat{\theta}) = -b + \mathbf{A}\hat{\theta} = 0$$

With $b = \frac{1}{n} \sum_{t=1}^{n} y^{(t)} x^{(t)}$ is a scalar and $\mathbf{A} = \frac{1}{n} \sum_{t=1}^{n} x^{(t)} (x^{(t)})^T$ is an (dxd) matrix. If $\mathbf{A}$ is invertible we can finally write $\hat{\theta} = \mathbf{A}^{-1}b$.

We can invert \mathbf{A} only if the feature vectors $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}$ span over R^d, that is if $n >> d$.

Also, we should put attention that inverting $\mathbf{A}$ is an operation of order $O(d^3)$, so we should be carefully when d is very large, like in bag of words approaches used in sentiment analysis where d can be easily be in the tens of thousands magnitude.

7. Generalization and Regularization

We now focus the discussion in how do we assure that the algorithm we found, the parameters we estimated, will be good also for the unknown data, will be robust and not too much negatively impacted by the noise that it is in our training data?

This question is more and more important as we have less data to train the algorithm.

The way to solve this problem is to use a mechanism called regularisation that try to push us away to fit the training data "perfectly" (where we "fit" also the errors, the noises embedded in our data) and try to instead generalise to possibly unknown data.

The idea is to introduce something that push the thetas to zero, so that it would be only worth for us to move our parameters if there is really a very strong pattern that justify the move.

8. Regularization

The implementation of the regularisation we see in this lesson is called **ridge regression** and it is the same we used in the classification problem:

The new objective function to minimise becomes:

$$J_{n,\lambda} = R_n(\theta) + \frac{\lambda}{2}||\theta||^2$$

The first term is our empirical risk, and it catches how well we are fitting the data. The second term, being the square norm of thetas, tries to push the thetas to remain zero, to not move unless there is a significant advantage in doing so. And λ is the parameter that determine the trade off, the relative contribution, between these two terms. Note that being the norm it doesn't influence any specific dimension of theta. Its role is is actually to determine how much do I care to fit my training examples versus how much do I care to be staying close to zero. In other words, we don't want any weak piece of evidence to pull our thetas very strongly. We want to keep them grounded in some area and only pulls them when we have enough evidence that it would really, in substantial way, impact the empirical loss.

What's very nice about using the squared norm as regularisation term is that actually everything that we discussed before, both the gradient and closed form solution, can be very easily adjusted to this new loss function.

Gradient based approach with regularisation

With respect to a single point the gradient of J is :

$$\nabla_\theta(\frac{(y^{(t)} - \theta \cdot \mathbf{x}^{(t)})^2}{2} + \frac{\lambda}{2}||\theta||^2) = -(y^{(t)} - \theta \cdot \mathbf{x}^{(t)}) * \mathbf{x}^{(t)} + \lambda\theta$$

We can modify the gradient descent algorithm where the update rule becomes:

$$\theta = \theta - \eta * \nabla_\theta = \theta - \eta * (\lambda\theta - (y^{(t)} - \theta \cdot \mathbf{x}^{(t)}) * \mathbf{x}^{(t)}) = (1 - \eta\lambda)\theta + \eta * (y^{(t)} - \theta \cdot \mathbf{x}^{(t)}) * \mathbf{x}^{(t)})$$

The difference with the empirical risk without the regularisation term is the $(1 - \eta\lambda)$ term that multiply θ trying to put it down at each update.

The closed form approach with regularisation

In the homework ?

9. Closing Comment

By using regularisation, by requiring much more evidence to push the parameters into the right direction we will increase the mistakes of our prediction within the training set, but we will reduce the test error when the fitted thetas are used with respect to data that has not been used for the fitting step.

However if we continue to increase λ, if we continue to give weight to the regularisation term, then also the testing error will start to increase as well.

Our objective is to find the "sweet spot" of lambda where the test error is those that is minimised, and we can do that using the validation set to calibrate the value that lambda should have.

We saw regularization in the context of linear regression, but we will see regularization across many different machine learning tasks. This is just one way to implement it. We will see some other mechanism to implement regularisation, for example in neural networks.

Lecture 6. Nonlinear Classification

1. Objectives

At the end of this lecture, you will be able to

- derive non-linear classifiers from feature maps
- move from coordinate parameterization to weighting examples
- compute kernel functions induced from feature maps
- use kernel perceptron, kernel linear regression
- understand the properties of kernel functions

Lecture 7. Recommender Systems

Homework 3

Project 2: Digit recognition (Part 1)

[MITx 6.86x Notes Index]