Machine Learning

Lectured by Andrew Ng on Coursera

2017

1 Introduction

Definition (Machine Learning - Tom Mitchell, 1998)**.** A computer program is said to *learn* from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

2 Linear Regression

Notation (Training data)**.** We have a set of training examples, denoted by $x \in \mathbb{R}^n \times \mathbb{R}^m$ (m examples of n features). Concretely, $x^{(i)}$ denotes the features of the i^{th} training example, with $x_j^{(i)}$ being the value of feature j in the i^{th} training example.

Conventionally, we take an extra row of ones as the 0^{th} feature ($x_0 := 1$) and denote the corresponding $(m+1) \times n$ matrix X the *design matrix.*

The output variable is $y \in \mathbb{R}^m$.

Definition (Linear hypothesis)**.** Our hypothesis is

$$h_\theta(x) = \theta^T X$$

where $\theta \in \mathbb{R}^{n+1}$ is a vector of parameters to be estimated.

Definition (Squared Loss Cost Function)**.** The *cost function*

$$\begin{aligned} J(\theta) &= \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^i) - y^i\right)^2 \\ &= \frac{1}{2m} (\theta^T X - y)^T (\theta^T X - y) \end{aligned}$$

represents a measure of the fit of a particular choice of parameters θ.

We estimate θ as the value minimising $J(\theta)$.

2.1 Gradient Descent

Gradient descent involves following the gradient of this cost function to find its minimum. In practice, taking an initial estimate θ_0 and *learning rate* α we iteratively compute

$$\theta := \theta - \alpha \boldsymbol{\nabla} J$$

that is

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

for $j \in \{0, \dots, n+1\}$.

For linear regression,

$$\begin{aligned}\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^i) - y^i \right) x_j^{(i)} \\ &= \frac{1}{m} X(\theta^T X - y)\end{aligned}$$

We declare convergence when $J(\theta)$ decreases by less than ϵ in a single iteration.

The learning rate α must be chosen carefully. Too large and convergence may not occur, too small and convergence will be slow.

2.1.1 Feature scaling

We can speed up gradient descent via *feature scaling* and *mean normalization*. Intuitively, this involves scaling each feature so that the steps taken along the gradient are roughly uniform across the dimensions of the feature space. Concretely, we set

$$x = \frac{x - \mu}{\sigma}$$

where μ is the (row-wise) mean of x and σ is the (row-wise) standard deviation of x.

2.2 Analytic solution

The minimum of the cost function can also be found analytically (solve the system of equations $\boldsymbol{\nabla} J = 0$ to obtain the exact solution

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

in the cast where $(X^T X)$ is invertible. Even in the singular case, we can take a numerical solution with the pseudo-inverse, e.g. via the Octave function `pinv`. The singular case can occur when there are redundant (linearly dependent) features, or too many features ($m \leq n$).

3 Logistic Regression - Classification

We now restrict $y \in \{0,1\}^m$ so that we now have the problem of *classifying* an observation x.

Definition (Classification hypothesis)**.** Our hypothesis is

$$\begin{aligned}h_\theta(x) &= g(\theta^T x) \\ &= \frac{1}{1 + e^{-\theta^T x}}\end{aligned}$$

where $g(z)$ is the *sigmoid* function, so that $0 \leq h_\theta(x) \leq 1$. We interpret

$$\begin{aligned} h_\theta(x) &= \mathbb{P}_\theta(y = 1 \mid x) \\ &= \mathbb{P}(y = 1 \mid x, \theta) \end{aligned}$$

and 'predict' that $y = 1$ if $h_\theta(x) >= 0.5$, i.e. if $\theta^T x >= 0$. The surface $h_\theta(x) = 0.5$ is known as the *decision boundary.*

The squared loss cost function is not convex in the case of logistic regression, we instead use the logistic cost function.

Definition (Logistic Cost function)**.** The *logistic* cost function is

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

where

$$\begin{aligned} \mathrm{Cost}(h_\theta(x), y) &= \begin{cases} -\log(h_\theta(x)) & \text{for } y = 1 \\ -(1 - \log(h_\theta(x))) & \text{for } y = 0 \end{cases} \\ &= -y \log(h_\theta(x)) - (1 - y)(1 - \log(h_\theta(x))) \end{aligned}$$

This form of cost function can be derived from *maximum likelihood estimation* for the binomial distribution.

Gradient descent applies in the same way, and moreover we again find that

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

though of course with the new hypothesis function.

3.0.1 Advanced optimization

There exist other, more complex algorithms to minimum the cost function, such as

(i) Conjugate gradient

(ii) BFGS

(iii) L-BFGS

which don't involve picking a learning rate and are often faster.

3.1 Multiclass classification

For the case of classifying $y \in \{1, \ldots, k\}$ we can apply the principle of *one-vs-all* to train k classifiers $h_\theta^{(i)}$ each of which is predicting the probability $\mathbb{P}_\theta(y = i \mid x)$ (against all other classes). The final prediction is then taken as the class i maximising $h_\theta^{(i)}$.

3.2 Addressing overfitting

With too many features, the learned hypothesis may fit the training set very well ($J(\theta) \approx 0$ but fail to generalize to new examples. This is known as *overfitting*.

This can be addressed by reducing the number of features (manually or alorithmically), or via regularization.

3.2.1 Regularization

The idea is too keep all features, but reduce the magnitude of the parameters θ_j. This works well when there are lots of features, each contributing a bit to predicting y.

To achieve this, we *penalize* the parameters inside the cost function.

For *linear regression*, we have

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

N.B. we conventionally do not penalize θ_0, so it is excluded from the sum.

The analytic solution is then

$$\theta = \left(X^TX + \lambda\begin{bmatrix}0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1\end{bmatrix}\right)^{-1} X^Ty$$

which does not suffer from the problem of non-invertibility.

For *logistic regression*, we have

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

As with the learning rate, λ must be appropriately chosen: too small and the regulation will have little effect, but too large will lead to *underfitting*!