Stanford CS229: Machine Learning

James Zhang*

June 18, 2023

*Email: jzhang72@terpmail.umd.com

Contents

§1 Linear Regression

§1.1 Notation

- $x^{(i)}$ denotes input variables of features, and $y^{(i)}$ denotes the output or target variable
- $x^{(i)} \in \mathbb{R}^{n+1}$ because we have a fake feature $x_0 = 1$
- $y^{(i)} \in \mathbb{R}$ always in the case of regression
- $x_j^{(i)}$ represents the j-th feature of the i-th training example
- m denotes the number of training examples
- n denotes the number of features
- $h(x)$ denotes the hypothesis function, which is a linear function of the features x and $x_0 = 1$
- $J(\vec{w})$ denotes the cost function you're trying to minimize by finding the optimal set of parameters $\vec{w}$ to straight line fit the data
- X denotes the space of input values, and Y denotes the space of output values

§1.2 Hypothesis Function

We represent our hypothesis function as a linear function of x, such that our values w_i are weights

$$h(x) = \sum_{i=0}^{n} w_i x_i = w \cdot x = w^T x$$

such that w and x are both vectors, and n is the number of input variables.
Now let us define the least squares cost function that gives rise to the ordinary least squares regression model.

§1.3 Cost Function

$$J(\vec{w}) = \frac{1}{2m} \sum_{i=1}^{m} (h(x^{(i)} - y^{(i)})^2$$

and we want to choose $\vec{w}$ so as to minimize the cost function, and this is known as the gradient descent algorithm.

$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} J(w)$$

and this is performed for all values of $j \in [0, n]$. Note that α is the learning rate.

§1.4 LMS (Widrow-Hoff) Learning Rule

Let us solve for $\frac{\partial}{\partial w_j} J(\vec{w})$ such that we can substitute it into our formula for gradient descent.

$$\begin{aligned}\frac{\partial}{\partial w_j} J(\vec{w}) &= \frac{\partial}{\partial w_j} \frac{1}{2}(h(x) - y)^2 \\ &= (h(x) - y) \cdot \frac{\partial}{\partial w_j}(h(x) - y) \\ &= (h(x) - y) \cdot \frac{\partial}{\partial w_j}(\sum_{i=0}^{n} w_i x_i - y \\ &= (h(x) - y) x_j\end{aligned}$$

For a single training example, the update rule is

$$w_j := w_j + \alpha(y^{(i)} - h(x^{(i)}))x_j^{(i)}$$

and this is known as the LMS (Least Mean Squares) and the Widrow-Hoff Learning Rule and stochastic gradient descent.

§1.5 Stochastic vs. Batch Gradient Descent

Note 1.1. Parameters may keep oscillating around the local extreme, but it will get there way faster than batch gradient descent.

Note 1.2. We can ensure that this algorithm converges if as we get close to the local extreme, let $\alpha \to 0$.

To modify this for more than one example and this is batch gradient descent

$$w_j := w_j + \alpha \sum_{i=1}^{m} (y^{(i)} - h(x^{(i)}))x_j^{(i)}$$

Note 1.3. When the training set m is large, stochastic gradient descent is preferred over batch gradient descent

§1.6 The Normal Equations

Instead of using the iterative algorithm of gradient descent, we can minimize our cost function $J(\vec{w})$ by explicitly taking derivatives with respect to the w_j and set them to 0.

§1.6.1 Matrix Derivatives

Suppose we have a function $f : \mathbb{R}^{m \times n} \to \mathbb{R}$ that maps from an $m \times n$ matrix to the real numbers, the derivative of f with respect to matrix A is

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{m1}} & \cdots & \frac{\partial f}{\partial A_{mn}} \end{bmatrix}$$

Example 1.4

Suppose

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

is a 2×2 matrix, and the function $f : \mathbb{R}^{2 \times 2} \to \mathbb{R}$ is given by

$$f(A) = \frac{3}{2} A_{11} + 5A_{12}^2 + A_{21} A_{22}$$

then

$$\nabla_A f(A) = \begin{bmatrix} \frac{3}{2} & 10A_{12} \\ A_{22} & A_{21} \end{bmatrix}$$

Now let us introduce the trace operator such that the trace of a matrix A is the sum of its diagonal entries.

$$\operatorname{Tr} A = \sum_{i=1}^{n} A_{ii}$$

Lemma 1.5

$$\operatorname{Tr} AB = \operatorname{Tr} BA$$
$$\operatorname{Tr} ABC = \operatorname{Tr} CAB = \operatorname{Tr} BCA$$
$$\operatorname{Tr} A = \operatorname{Tr} A^T$$
$$\operatorname{Tr}(A + B) = \operatorname{Tr} A + \operatorname{Tr} B$$
$$\operatorname{Tr} aA = a \operatorname{Tr} A$$

Note that the fourth equation only applies to non-singular square matrices.

Lemma 1.6

$$\nabla_A \operatorname{Tr} AB = B^T$$
$$\nabla_{A^T} f(A) = (\nabla_A f(A))^T$$
$$nabla_A \operatorname{Tr}(ABA)^T C = CAB + C^T AB^T$$
$$\nabla_A \det(A) = \det(A)(A^{-1})^T$$

§1.6.2 Least Squares Revisited

We seek a closed form of $\vec{w}$ that minimizes $J(\vec{w})$. Let us rewrite $J(\vec{w})$ in matrix-vectorial notation.

Proof. Let X be an $m \times n$ matrix such that each column represents a training example.

$$X = \begin{bmatrix} (x^{(1)})^T & \cdots & (x^{(m)})^T \end{bmatrix}$$

and let

$$\vec{y} = \begin{bmatrix} y^{(i)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

be an $m \times 1$ vector. Recall that $h(x^{(i)}) = (x^{(i)})^T w$ and therefore

$$X\vec{w} - \vec{y} = \begin{bmatrix} h(x^{(1)}) - y^{(1)} \\ \vdots \\ h(x^{(m)}) - y^{(m)} \end{bmatrix}$$

Therefore,

$$\frac{1}{2}(X\vec{w} - \vec{y})^T(X\vec{w} - \vec{y}) - \frac{1}{2}\sum_{i=1}^{m}(h(x^{(i)} = y^{(i)})^2 = J(\vec{w})$$

Thus we've represented our cost function in terms of matrices, but now let us minimize it using the trace equations above.

$$\nabla_{\vec{w}J(\vec{w}} = \nabla_{\vec{w}}(X\vec{w} - \vec{w})^T(X\vec{w} - \vec{y})$$

$$= \frac{1}{2}\nabla_{\vec{w}}(\vec{w}^T X^T X\vec{w} - \vec{w}^T X^T\vec{y} - \vec{y}^T X\vec{w} + \vec{y}^T\vec{y})$$

by homogeneity, tranpose of a matrix, and distributive property.

$$= \frac{1}{2}\nabla_{\vec{w}} \operatorname{Tr}(\vec{w}^T X^T X\vec{w} - \vec{w}^T X^T\vec{y} - \vec{y}^T X\vec{w} + \vec{y}^T\vec{y})$$

because the trace of a real numher is just the real number

$$= \frac{1}{2}\nabla_{\vec{w}}(\operatorname{Tr}\vec{w}^T X^T X\vec{w} - 2\operatorname{Tr}\vec{y}^T X\vec{w})$$

using the property $\operatorname{Tr} A = \operatorname{Tr} A^T$.

$$= \frac{1}{2}(X^T X\vec{w} + X^T X\vec{w} - 2X^T\vec{y})$$

$$= X^T X - X^T\vec{y} \implies X^T X\vec{w} = X^T\vec{y} \implies \vec{w} = (X^T X)^{-1}X^T\vec{y}$$

□

§1.7 Probabilistic Interpretation: Why Squared Error

Suppose in our linear regression model

$$y^{(i)} = \vec{w}^T x^{(i)} + \epsilon^{(i)}$$

where $\epsilon^{(i)}$ is an error term. Let's now assume that $\epsilon^{(i)}$ follows a Gaussian distribution of mean $\mu = 0$ and variance σ^2, which is reasonable.

$$\epsilon^{(i)} \sim N(0, \sigma^2) \implies \mathbb{P}(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{(e^{(i)})^2}{2\sigma^2})$$

Now let us assume that each $\epsilon^{(i)}$ is IID (independently and identically distributed) from each other. This may not be entirely true, but it's good enough to get a good approximation for a model. Therefore, we can express the following

$$\mathbb{P}(y^{(i)} \mid x^{(i)}; \vec{w}) = \frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{(y^{(i)} - \vec{w}x^{(i)})^2}{2\sigma^2})$$

by substitution. Essentially, given $\vec{x}, \vec{w}$, what's the probability density of a particular house's price?

$$y^{(i)} \mid x^{(i)}; \vec{w}) \sim N(\vec{w}^T x^{(i)}, \sigma^2)$$

The random variable $y^{(i)}$ given $x^{(i)}$ parameterized by $\vec{w}$ is that Gaussian. We're essentially modeling the price of a house using the random variable y

§2 Locally Weighted Linear Regression

§2.1 Fitting Your Data

You can either fit

- Linear of the form $w_0 + w_1 x_1$
- Quadratic of the form $w_0 + w_1 x_1 + w_2 x_2^2$
- Some custom fit?

Whatever way, the linear regression naturally fits these functions of input features in your dataset.

§2.2 Locally Weighted Linear Regression

Locally Weighted Linear Regression is an algorithm that modifies linear regression to fit non-linear functions.

Note 2.1. The more features we add does not always correspond to better performance because it could lead to overfitting.

Recall that the original cost function was

$$J(\vec{w}) = \frac{1}{2m}\sum_{i=1}^{m}(y^{(i)} - \vec{w}^T x^{(i)})^2$$

. In prediction, to make a prediction at an input x usuing locally weighted linear regression

- Fit a straight line through the local neighborhood of training examples close to x.
- Fit a line passing through the outputs y in that neighborhood, and make a prediction at x.
- Note that by focusing on the local neighborhood of input points, we attribute these points to influence the output prediction the most, but the other points can also be weighted lesser and utilized for our prediction.

We can represent this in a tweaked cost function such that

$$J(\vec{w}) = \sum_{i=1}^{m} w'^{(i)}(y^{(i)} - \vec{w}^T x^{(i)})^2$$

where w' is a weighting function with values $\in [0, 1]$. A common choice for w' is

$$w'^{(i)} = \exp(-\frac{x^{(i)} - x)^2}{2\tau^2})$$

Note 2.2. The closer the neighbor ie. $|x^{(i)} - x| \to 0$ the more heavily weighted as in $w' \approx e^0 = 1$. The counter logic applies as well. The further a neighbor, the smaller the weight.

Note 2.3. The weight function is not a normal distribution, and the weights are not random variables.

§2.3 Bandwidth Parameter

- The bandwidth parameter τ decides the width of the neighborhood.
- τ controls how quickly the weight of a training example $x^{(i)}$ falls off as distance from the query point x increases, and τ can have a big impact on potentially underfitting and overfitting
- τ is a hyperparameter of this algorithm, and it doesn't have to be bell-shaped.

§2.4 Applications

The main benefit of this algorithm is that fitting a non-linear dataset doesn't require you to manually fiddle with features.

Note 2.4. This works best when the number of features isn't too large, say 2 or 3.

In terms of computation and space complexity, you need to solve a linear system of equations of dimension equal to m so we may need to resort to scaling algorithms.

§3 Newton's Method

Gradient ascent is very effective, but it needs many iterations before convergence. Newton's method takes bigger steps and converges earlier when maximizing the likelihood function $l(\vec{w})$. Although there are less iterations, each iteration tends to be more expensive.

Lemma 3.1

Suppose we have a function $f : \mathbb{R} \to \mathbb{R}$ and a scalar value w. The update rule to find the zero of a function is the following

$$w := w - \frac{f(w)}{f'(w)}$$

The nautral interpretation of this formula is we take our function f and approximate it with a linear function tangent to f at the current value of w, and solving for where this linear function equals 0 (the x-intercept) gives us our next guess for w.

Now suppose we wish to maximize the log likelihood function $l(w)$. The maxima of $l(w)$ corresponds to points where its first derivative is equal to 0. Thus, let

$$f(w) = l'(w) \implies w := w - \frac{l'(w)}{l''(w)}$$

In terms of indices,

$$w^{(t+1)} = w^{(t)} - \frac{l'(w)}{l''(w)}$$

§3.1 Quadratic Convergence

Note that when you apply Newton's method, the iterations represent quadratic steps such that the number of siginifcant digits that you have converged to the minimum double on a single iteration.

§3.2 Newton's Method for Vectors

If w is a vector and not a scalar, we have to generalize Newton's method to a multi-dimensional setting, and this is known as the Newton-Raphson method

$$\vec{w} := \vec{w} - H^{-1}\nabla_0 l(\vec{w})$$

) where

- $\nabla_0 l(\vec{w})$ is the vector of partial derivatives with respect to the $w_i's$
- H is the Hessian square matrix defined as

$$H_{ij} = \frac{\partial^2 l(\vec{w})}{\partial w_i \partial w_j}$$

§4 The Classification Problem

Let us take this framework that we have developed and apply it to a classification problem. This differs from regression because the y values can only take on a discrete number of values.
Let us simplify this even further by beginning with the binary classification problem such that $y \in 0, 1$ where 0 is the negative class and 1 is the positive class.

Note 4.1. The problem with applying linear regreesion to a classification problem and applying a threshold of 0.5 is that if there is a single outlier, the regression algorithm will adjust its straight line, which could drastically alter the decision boundary. Furthermore, the decision boundary for classification problems may not be clear cut.

§4.1 Logistic Regression

If we know output values will be between 0 and 1, it makes sense to choose our hypothesis function $h(x)$ as the following

$$h(x) = g(w^T x) = \frac{1}{1 + e^{-w^T x}}$$

where $g(z) = \frac{1}{1+e^{-z}}$ is known as the logisitic function or the sigmoid function.

Lemma 4.2

When we will later try to adjust w such as to minimize our cost function, we must be able to compute the derivative of the cost function, and thereby we must be able to compute the derivative of the sigmoid function.

$$g'(z) = \frac{d}{dz}\frac{1}{1 + e^{-z}} = g(z)(1 - g(z))$$

A rigorous derivation can be found online.

§4.2 Update Rule

Let's use a similar technique of making some probabilistic assumptions and then fitting the parameters via a a maximum likelihood. Note that because y only takes 2 values, 0, and 1

$$(y^{(i)} = 1|x^{(i);w}) = h(x^{(i)})$$
$$(y^{(i)} = 0|x^{(i)}; w) = 1 - h(x^{(i)})$$

Assuming that the m training examples were generated independently, we have the likelihood function as follows

$$L(w) = \mathbb{P}(\vec{y}|X; w) = \prod_{i=1}^{m} \mathbb{P}(y^{(i)}|x^{(i)}; w) = \prod_{i=1}^{m} h(x^{(i)})^{y^{(i)}}(1 - h(x^{(i)}))^{1-y^{(i)}}$$

Note that it's easier instead to maximumze the log likelihood rather than the actual likelihood, as local extreme will occur in the same points in both.

$$l(w) = \log L(w) = \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))$$

Now we will use batch gradient ascent to obtain the update rule

$$w := w + \alpha \nabla_w l(w)$$

Note 4.3. Note that there's a plus sign because we are using gradient ascent, not descent, and here we are maximizing the log-likelihood function instead of minimizng the squared cost function like in linear regression.

Now if we plug in our sigmoid function value of $h(x)$, and then solve for $\frac{\partial l(w)}{\partial w_j}$ we get the following

$$\frac{\partial l(w)}{\partial w_j} = (y - h(x^{(i)}))x_j^{(i)}$$

which yields the update rule for an entire training set m of examples

$$w_j = w_j + \alpha \sum_{i=1}^{m} (y^{(i)} = h(x^{(i)})x_j^{(i)})$$

While this appears to be the same as the LMS update rule, it is not because now our hypothesis function is now not a linear function of $w^T x$. However, this is not a coincedence, and this update rule is a property of a bigger class of algorithms called generalized linear models.