Lecture 6: Linear Models

CSE 427: Machine Learning

Md Zahidul Hasan

Lecturer, Computer Science
BRAC University

Spring 2023

Linearity and Affinity

Linear Functions and Affine Functions

Let x be a d-dimensional vector, w be a vector of d weights and b be a real number, then the first one is a linear function and the second one is an affine function:

$$h_w(x) = \langle w, x \rangle = \sum_{i=1}^{d} w_i x_i$$
$$h_{w,b}(x) = \langle w, x \rangle + b = \left(\sum_{i=1}^{d} w_i x_i \right) + b$$

If we append 1 to the end of the vector x and b to the end of the weight vector w, then the affine function becomes a linear function. Since $x_{d+1} = 1$ and $w_{d+1} = b$,

$$h_{w,b}(x) = \langle w, x \rangle = \sum_{i=1}^{d+1} w_i x_i .$$

When we extend x this way, it becomes a vector of homogeneous $d + 1$ dimensional coordinate system. That which is a linear function in a homogeneous coordinate system is an affine function in a non-homogeneous coordinate system. b is called the bias.

Hyperplanes and Halfspaces

Hyperplanes and Halfspaces

In d-dimension, hyperplanes are defined by the following equation:

$$w_1x_1 + w_2x_2 + \cdots + w_dx_d + b = 0$$

A d-dimensional hyperplane divides $\mathbb{R}^d$ into two halfspaces depending on:

$$sign(\langle w, x \rangle + b)$$

We can easily use hyperplanes to classify a feature vector x. We have already seen that in the case of non-homogeneous affine hyperplanes the VC-dimension is $d + 1$. In the homogeneous case, it's d. We can learn the hyperplanes/halfspaces using ERM. In the following slides, we discuss methods of finding solutions in the realizable case i.e. linearly separable case. The non-realizable case is computationally intractable.

Linear Programming

LP

In Linear Programming, we try to maximize a linear objective function subject to a set of linear inequalities represented by a matrix multiplication:

$$\max_{w \in \mathbb{R}^d} \langle w, u \rangle \text{ subject to } Aw \geq v$$

Linear programming problems can be solved efficiently with commonplace implementations. In the linearly separable case, we can convert our problem of finding a hyperplane that leads to zero error on the training set to a linear programming problem. For simplicity, let's assume that we are using homogeneous coordinates for feature vector x. So, a hyperplane can be $h_w(x) = \langle w, x \rangle$. We are looking for a w so that,

$$sign(\langle w, x_i \rangle) = y_i \text{ for all } i \in [m]$$
$$\text{in other words, } y_i \langle w, x_i \rangle > 0 \text{ for all } i \in [m]$$

Linear Programming to solve ERM

Since the points are separable using hyperplanes, such a w definitely exists. Let's say $w*$ is one such vector. Let's define, $\gamma = \min_{i \in [m]} y_i \langle w, x_i \rangle$ and let $\bar{w} = \frac{w*}{\gamma}$. So, we have,

$$y_i \langle \bar{w}, x_i \rangle = \frac{1}{\gamma} y_i \langle w*, x_i \rangle \geq \frac{1}{\gamma} \gamma = 1 \text{ for all } i \in [m]$$

So, we have just arrived at the necessary conditions of our LP problem. We can express all of the above inequalities as $Aw \geq v$ where $A_{i,j} = y_i x_{i,j}$ where $x_{i,j}$ is the j'th component of the i'th vector x_i. v is the vector of all ones i.e. $(1, 1, \cdots, 1)$.
Notice that, we are only looking for one such weight vector w. Any one of them is fine. So, we don't want to maximize any linear objective function. So, we can pick any dummy vector such as $u = (0, 0, \cdots, 0)$.

Perceptron for Halfspaces

- The perceptron algorithm is due to Rosenblatt. We start with a weight vector $w = (0, 0, \cdots, 0)$.
- Our goal is to reach a state where $y_i \langle w, x_i \rangle > 0$ for all $i \in [m]$.
- At iteration t, if there is still an i so that $y_i \langle w, x_i \rangle \leq 0$, then we update $w_{t+1} = w_t + y_i x_i$ like this.

This makes sense because
$y_i \langle w_{t+1}, x_i \rangle = y_i \langle w_t + y_i x_i, x_i \rangle = y_i \langle w_t, x_i \rangle + ||x_i||^2 \geq y_i \langle w_t, x_i \rangle$.
So, after each iteration, the constraints are becoming more correct for a mistaken i. It can also be proven that the algorithm stops after at most $(RB)^2$ iterations where $R = \max_i ||x_i||$ and $B = \min\{||w|| : \forall i \in [m], y_i \langle w, x_i \rangle \geq 1\}$ and when it stops it holds that $\forall i \in [m], y_i \langle w, x_i \rangle \geq 1$. The proofs are left to the readers for exercise.

Linear Regression

Regression is the task of estimating the value of one variable that's dependent on other variables. For example: determining the rent of a house is a function of the area of the house by square feet, the number of bedrooms, location and other facilities or estimating the salary of a person as a function of his educational background and skill-set etc. There are lots of regression models. We will discuss only the linear model here. But the best in practice are Kernel Ridge regression, Support Vector regression, Lasso etc. Imagine that the features of a house can be numerically expressed as the following vector $x = (x_1, x_2, \cdots, x_d)$ where x_1 can mean the number of kitchens or x_2 can mean whether there is a gym in the building or not. In problems where a change in the independent variables leads to a linear change in the dependent variables, linear regression is a perfect solution. Our linear hypothesis is that if y is the house rent, then,
$y = h_w(x) = w_1x_1 + w_2x_2 + \cdots + w_dx_d + w_{d+1}$ where $w_{d+1} = b$ and we can extend x by setting $x_{d+1} = 1$.

Multivariate Linear Regression

Imagine that the features of a house can be numerically expressed as the following vector $x = (x_1, x_2, \cdots, x_d)$ where x_1 can mean the number of kitchens or x_2 can mean whether there is a gym in the building or not. Our linear hypothesis is that if y is the house rent, then, $y = h_w(x) = w_1x_1 + w_2x_2 + \cdots + w_dx_d + w_{d+1}$
So, if we extend the vector x by one dimension by appending a dummy 1 at the end $x = (x_1, x_2, \cdots, x_d, 1)$, we can express y as follows, $y = w^Tx = \langle x, w \rangle$ where $w = (w_1, w_2, \cdots, w_d, w_{d+1})$ So, if we are given m data points $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots,$ $(x^{(m)}, y^{(m)})$, the squared error should be:
$E = \sum_{i=1}^{m} (y^{(i)} - \langle w, x^{(i)} \rangle)^2$
We can express the above equation in the following matrix form:

$$E(W) = ||Y - X^TW||^2.$$

Multivariate Linear Regression contd.

where, $Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ . \\ y^{(m)} \end{bmatrix}, X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & . & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & . & x_2^{(m)} \\ . & . & . & . \\ x_{d+1}^{(1)} & x_{d+1}^{(2)} & . & x_{d+1}^{(m)} \end{bmatrix}, W = \begin{bmatrix} w_1 \\ w_2 \\ . \\ w_{d+1} \end{bmatrix}$

We want to solve for a W so that the error is minimized. Since E is convex and differentiable, it admits a global minimum at a certain W where $\nabla E(W) = 0$.
In other words, $2X(X^TW - Y) = 0$ or $XX^TW = XY$. If XX^T is invertible, then there is a unique solution, $W = (XX^T)^{-1}XY$. Otherwise, we can get a family of solutions,

$$W = (XX^T)^{+}XY + (\mathcal{I} - (XX^T)^{+}XX^T)A$$

where A is an arbitrary matrix and $(XX^T)^{+}$ is the Moore-Penrose Pseudoinverse of XX^T. The solution with the smallest norm is achieved when $A = 0$. The overall complexity of the solution is $\mathcal{O}((m+d)d^2)$

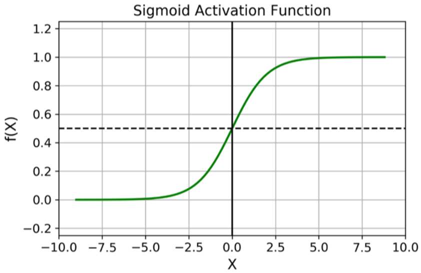
Univariate Linear Regression

Let's say, we only have one feature. So, our linear hypothesis is very simple. $h_w(x) = w_1 x + w_2$
So, the loss over the whole data set can be formulated as,
$E(W) = \sum_{i=1}^{m} (y_i - (w_1 x_i + w_2))^2$
We want to find values of w_1 and w_2 so that E is minimized. That will happen when $\frac{\partial E}{\partial w_1} = 0$ and $\frac{\partial E}{\partial w_2} = 0$. These two equations lead to,

$$2\sum_{i=1}^{m} (y_i - (w_1 x_i + w_2))x_i = 0 \text{ and } 2\sum_{i=1}^{m} (y_i - (w_1 x_i + w_2)) = 0$$

Here, we have two equations with two variables.
$w_1 \sum_{i=1}^{m} x_i^2 + w_2 \sum_{i=1}^{m} x_i = \sum_{i=1}^{m} x_i y_i$
$w_1 \sum_{i=1}^{m} x_i + w_2 \sum_{i=1}^{m} 1 = \sum_{i=1}^{m} y_i$
Solving for w_1 and w_2 yields, $w_1 = \frac{m(\sum_{i=1}^{m} x_i y_i) - (\sum_{i=1}^{m} x_i)(\sum_{i=1}^{m} y_i)}{m(\sum_{i=1}^{m} x_i^2) - (\sum_{i=1}^{m} x_i)^2}$
$w_2 = \frac{(\sum_{i=1}^{m} x_i^2)(\sum_{i=1}^{m} y_i) - (\sum_{i=1}^{m} x_i)(\sum_{i=1}^{m} x_i y_i)}{m(\sum_{i=1}^{m} x_i^2) - (\sum_{i=1}^{m} x_i)^2}$

The Sigmoid Function



The hard threshold function $sign(\langle w, x \rangle)$ is not continuous and differentiable. To make learning a more predictable endeavour, we can use the logistic function to determine the probability of our output being 1 given that we are trying to classify points into the class $\{0, 1\}$ which is as follows:

$$L(h_w(x)) = \frac{1}{1+e^{-h_w(x)}}$$

The value of this function approaches 1 as $h_w(x)$ approaches $+\infty$ and as its value approaches $-\infty$, the function's value approaches 0.

Logistic Regression

Instead of the hard threshold function, we can use the logistic/sigmoid function for classification. If $L(h_w(x)) \geq \frac{1}{2}$, then the output is $+1$, otherwise it's -1. Some suitable loss functions would be:

$$\ell(h_w, (x_i, y_i)) = \log(1 + e^{-y_i\langle w, x_i \rangle}) \text{ if } y_i, h_w(x_i) \in \{-1, +1\}$$
$$\ell(h_w, (x_i, y_i)) = -y_i \log h_w(x_i) - (1 - y_i) \log(1 - h_w(x_i)) \text{ if } y_i, h_w(x_i) \in \{0, 1\}$$

So, our ERM solution will be the following:

$$\arg\min_{w \in \mathbb{R}^d} \frac{1}{m} \sum_{i=1}^{m} \log(1 + e^{-y_i\langle w, x_i \rangle})$$

The advantage of the above loss function is that it's convex, continuous and differentiable. But how did we come up with these objective functions?

Softmax Regression

Logistic regression is a binary classifier. But what if, we have k classes. If our input has n features, then we could learn a linear function for each output class as follows:

$o_1 = w_{1,1}x_1 + w_{1,2}x_2 + \cdots + w_{1,n}x_n + b_1$
$o_2 = w_{2,1}x_1 + w_{2,2}x_2 + \cdots + w_{2,n}x_n + b_2$
$\ldots$
$o_k = w_{k,1}x_1 + w_{k,2}x_2 + \cdots + w_{k,n}x_n + b_k$

Then, we could output a distribution over the k output classes using the softmax function. The softmax function looks like the following:

$softmax(x) = [\frac{e^{x_1}}{\sum_{i=1}^{n} e^{x_i}}, \frac{e^{x_2}}{\sum_{i=1}^{n} e^{x_i}}, \cdots, \frac{e^{x_n}}{\sum_{i=1}^{n} e^{x_i}}]$

So, our output distribution will be:

$softmax(o) = [\frac{e^{o_1}}{\sum_{i=1}^{k} e^{o_i}}, \frac{e^{o_2}}{\sum_{i=1}^{k} e^{o_i}}, \cdots, \frac{e^{o_k}}{\sum_{i=1}^{k} e^{o_i}}]$

But how do we measure the error? How can we come up with an objective function to minimize? That's what we will see in the next lecture.