CS60010: Deep Learning

Spring 2021

Sudeshna Sarkar and Abir Das

Linear Models

Sudeshna Sarkar

12 Jan 2021

ML Background and Linear Models

Based on Slides by Abir Das

Machine Learning Background

$\mathcal{X}$: A space of “observations” (Instance space)

$\mathcal{Y}$: space of “targets” or “labels”

How the observations determine the targets?

Data: Pairs $\{(x^{(i)}, y^{(i)})\}$ with $x^{(i)} \in \mathcal{X}$ and $y^{(i)} \in \mathcal{Y}$.

Prediction: Given a new observation x, predict the corresponding y.

Prediction Problems

Observation Space $\mathcal{X}$:	Target Space $\mathcal{Y}$:
House attributes	Price of house
Car attributes, Route attributes, Driving behaviour	Battery energy consumption
Email	Spam or Non-spam
Images	Object: “cat”, “dog” etc.
Images	Caption
Face Images	User’s identity
Human Speech Waveform	Text transcript of the speech
Document	Topic of the Document
Scene Description in English	Sketch of the Scene
Video from an Automobile Camera	Steering, Accelerator, Braking
General Video Segment	Closed Caption Text

Prediction Functions

Assumption about the model $\hat{P}(X, Y)$, namely that $y = f(x)$, i.e. y ***takes a single value*** given x.

Inputs often referred to as predictors and features;

Outputs are known as targets and labels.

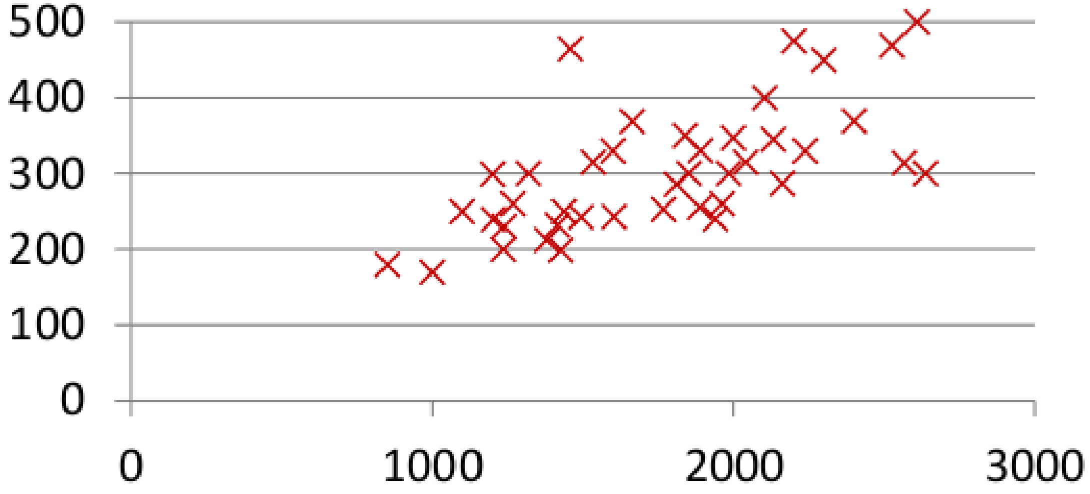
1. **Regression**: $y = f(x)$ is the predicted value of the output, and $y \in \mathcal{R}$ is a real value.
2. **Classifier**: $y = f(x)$ is the predicted class of x,
   and $y \in \{1, \ldots, k\}$ is the class number.

Prediction Functions

Linear regression, $y = f(x)$ is a linear function. Examples:

- (Outside temperature, People inside classroom, target room temperature | Energy requirement)
- (Size, Number of Bedrooms, Number of Floors, Age of the Home | Price)

A set of *N* observations of *y* as $\{y^{(1)}, \ldots, y^{(m)}\}$ and the corresponding inputs $\{x^{(1)}, \ldots, x^{(m)}\}$



Regression

- The input and output variables are assumed to be related via a relation, known as hypothesis, $\hat{y} = h_\theta(x)$

  θ is the parameter vector.

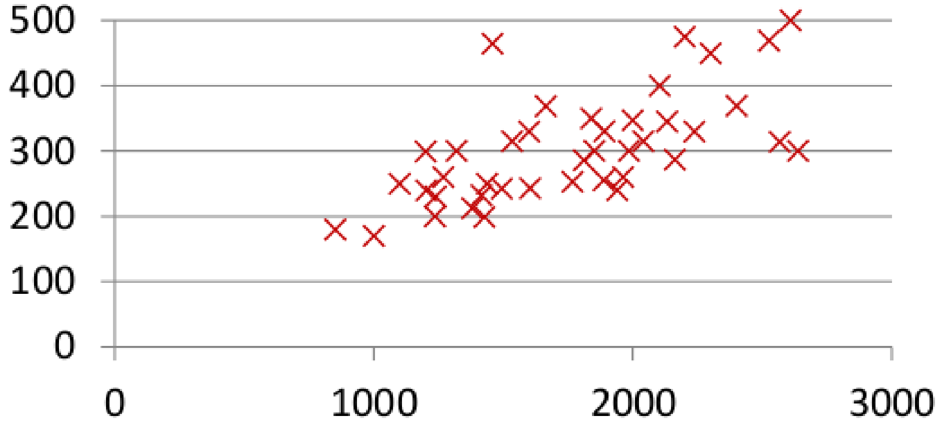
- The goal is to predict the output variable $y = f(x)$ for an arbitrary value of the input variable x.

Loss Functions

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

There may be no "true" target value y for an observation x

There may also be noise or unmodeled effects in the dataset



So we try to predict a value that is "close to" the observed target values.

A **loss function** measures the difference between a target prediction and target data value.

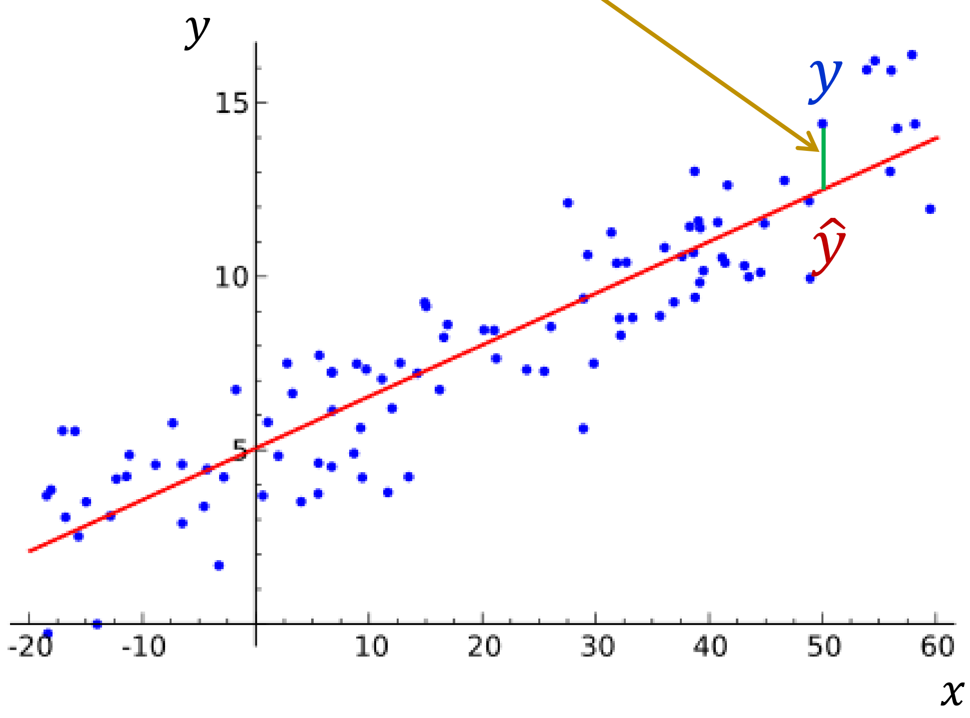
e.g. squared loss $L_2(\hat{y}, y) = (\hat{y} - y)^2$ where $\hat{y} = h_\theta(x)$ is the prediction,

Optimization objective: Find model parameters $\boldsymbol{\theta}$ that will minimize the loss.

Linear Regression

Simplest case, $\hat{y} = h(x) = \theta_0 + \theta_1 x$

The loss is the squared loss $L_2(\hat{y}, y) = (\hat{y} - y)^2$



Data (x, y) pairs are the blue points.
The model is the red line.

Linear Regression

The total loss across all points is

$$L = \sum_{i=1}^{m} \left(\widehat{y^{(i)}} - y^{(i)}\right)^2$$

$$= \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2$$

$$J(\theta_0, \theta_1) = \frac{1}{N} \sum_{i=1m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

We want the optimum values of θ_0, θ_1 that will minimize the sum of squared errors. Two approaches:

1. Analytical solution via mean squared error
2. Iterative solution via MLE and gradient ascent

Linear Regression

Since the loss is differentiable, we set

$$\frac{dL}{d\theta_0} = 0 \quad \text{and} \quad \frac{dL}{d\theta_1} = 0$$

We want the loss-minimizing values of $\boldsymbol{\theta}$, so we set

$$\frac{dL}{d\theta_1} = 0 = 2\theta_1 \sum_{i=1}^{N} \left(x^{(i)}\right)^2 + 2\theta_0 \sum_{i=1}^{N} x^{(i)} - 2 \sum_{i=1}^{N} x^{(i)} y^{(i)}$$

$$\frac{dL}{d\theta_0} = 0 = 2\theta_1 \sum_{i=1}^{N} x^{(i)} + 2\theta_0 N - 2 \sum_{i=1}^{N} y^{(i)}$$

These being linear equations of θ, have a unique closed form solution

Univariate Linear Regression Closed Form Solution

$$\theta_1 = \frac{m\sum_{i=1}^{m} y^{(i)}x^{(i)} - \left(\sum_{i=1}^{m} x^{(i)}\right)\left(\sum_{i=1}^{m} y^{(i)}\right)}{m\sum_{i=1}^{m}(x^{(i)})^2 - \left(\sum_{i=1}^{m} x^{(i)}\right)^2}$$

$$\theta_0 = \frac{1}{m}\left(\sum_{i=1}^{m} y^{(i)} - \theta_1 \sum_{i=1}^{m} x^{(i)}\right)$$

Risk Minimization

We found θ_0, θ_1 which minimize the squared loss on data ***we already have***. What we actually minimized was an averaged loss across a finite number of data points. This averaged loss is called **empirical risk.**

What we really want to do is predict the y values for points x ***we haven't seen yet***. i.e. minimize the expected loss on some new data:

$$E[(\hat{y} - y)^2]$$

The expected loss is called **risk**.

Machine learning approximates risk-minimizing models with empirical-risk minimizing ones.

Risk Minimization

Generally minimizing empirical risk (loss on the data) instead of true risk works fine, but it can fail if:

- The **data sample is biased**. e.g. you cant build a (good) classifier with observations of only one class.
- There is **not enough data** to accurately estimate the parameters of the model. Depends on the complexity (number of parameters, variation in gradients, complexity of the loss function, generative vs. discriminative etc.).

Multivariate Linear Regression

$x \in \mathcal{R}^d$

$$y = h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ \dots + \theta_d x_d$$

Define $x_0 = 1$

$$h_\theta(\mathbf{x}) = \theta^T \mathbf{x}$$

Cost Function:

$$J(\boldsymbol{\theta}) = J(\theta_0, \theta_1, \dots, \theta_d) = \frac{1}{m}\left(\boldsymbol{\theta}^T \mathrm{x}^{(i)} - y^{(i)}\right)^2$$

Multivariate Linear Regression

$$\begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(m)} \end{bmatrix} = \begin{bmatrix} x_0{}^{(1)} & x_1{}^{(1)} & x_2{}^{(1)} & \cdots & x_d{}^{(1)} \\ x_0{}^{(2)} & x_1{}^{(2)} & x_2{}^{(2)} & \cdots & x_d{}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_0{}^{(m)} & x_1{}^{(m)} & x_2{}^{(m)} & \cdots & x_d{}^{(m)} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_d \end{bmatrix}$$

$$\hat{\mathrm{y}} = \mathbf{X}\boldsymbol{\theta}$$

Multivariate Linear Regression

$$J(\boldsymbol{\theta}) = \frac{1}{m}\left(\boldsymbol{\theta}^T \mathrm{x}^{(i)} - y^{(i)}\right)^2 = \frac{1}{m}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

$$= \frac{1}{m}\|\hat{y} - y\|_2^2 = \frac{1}{m}(\hat{y} - y)^T(\hat{y} - y)$$

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

$$= \frac{1}{m}\{\theta^T(X^T X)\theta - \theta^T X^T y - y^T X\,\theta + y^T Y\}$$

$$= \frac{1}{m}\{\theta^T(X^T X)\theta - (X^T y)^T\theta - (X^T y)^T\,\theta + y^T Y\}$$

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

Multivariate Linear Regression

- Equating the gradient of the cost function to 0,

$$\nabla_\theta J(\boldsymbol{\theta}) = \frac{1}{m}\{2\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} - 2\mathbf{X}^T\mathbf{y} + 0\} = 0$$

$$\nabla_\theta J(\boldsymbol{\theta}) = \frac{2}{m}\{\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} - \mathbf{X}^T\mathbf{y}\} = 0$$

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} - \mathbf{X}^T\mathbf{y} = 0$$

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} = \mathbf{X}^T\mathbf{y}$$

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

Multivariate Linear Regression

- Equating the gradient of the cost function to 0,

$$\nabla_\theta J(\boldsymbol{\theta}) = \frac{1}{m}\{2\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} - 2\mathbf{X}^T\mathbf{y} + 0\} = 0$$

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\theta} - \mathbf{X}^T\mathbf{y} = 0$$

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

This gives a closed form solution, but another option is to use iterative solution

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

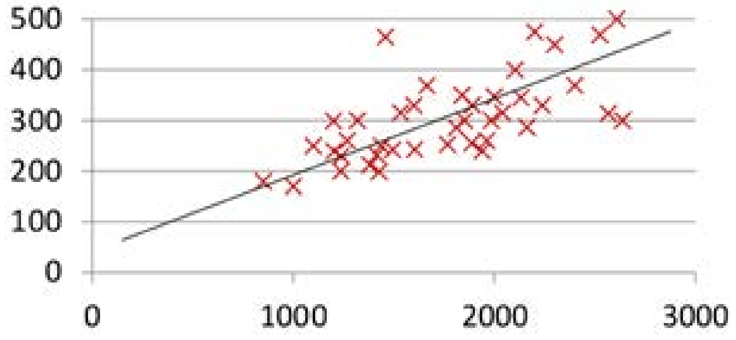
Iterative Gradient Descent

- Iterative Gradient Descent needs to perform many iterations and need to choose a stepsize parameter judiciously. But it works equally well even if the number of features (d) is large.
- For the least square solution, there is no need to choose the step size parameter or no need to iterate. But, evaluating $(\mathbf{X}^T\mathbf{X})^{-1}$ can be slow if d is large.

Linear Regression as Maximum Likelihood Estimation

Considers the following

- $y^{(i)}$ are generated from the $x^{(i)}$ following a underlying hyperplane.
- But we don't get to "see" the generated data. Instead we "see" a noisy version of the $y^{(i)}$'s.
- Maximum likelihood models this uncertainty in determining the data generating function.



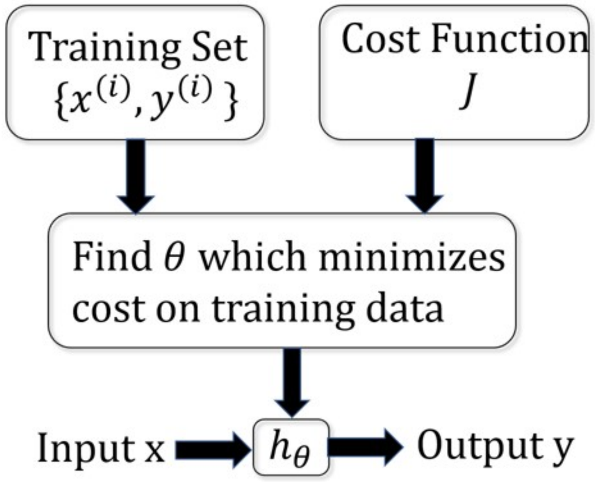
Data assumed to be generated as

$$y^{(i)} = h_\theta\left(x^{(i)}\right) + \epsilon^{(i)}$$

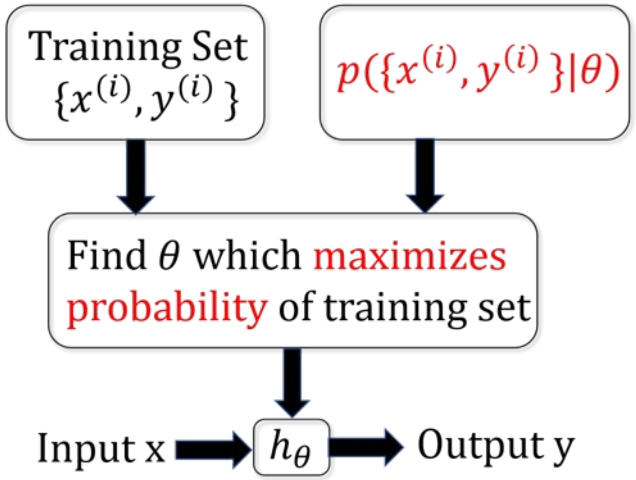
where $\epsilon^{(i)}$ is an additive noise following some probability distribution.

- Assume a parameterized probability distribution on the noise (e.g., Gaussian with 0 mean and covariance σ^2)
- Then find the parameters (both θ and σ^2) that is "most likely" to generate the data.

Loss Function Optimization



Maximum Likelihood



Maximum Likelihood for Linear Regression

- Assume that the noise is Gaussian distributed with mean 0 and variance σ^2

$$y^{(i)} = h_\theta\left(x^{(i)}\right) + \epsilon^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$$

- Noise $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$
- Thus $y^{(i)} \sim \mathcal{N}\left(\theta^T x^{(i)}, \sigma^2\right)$

Maximum Likelihood for Linear Regression

$$y^{(i)} \sim \mathcal{N}\left(\theta^T x^{(i)}, \sigma^2\right)$$

Compute the likelihood.

$$\begin{aligned}
p(\mathbf{y}|\mathbf{X};\boldsymbol{\theta},\sigma^2) &= \prod_{i=1}^{N} p(y^{(i)}|\mathbf{x}^{(i)};\boldsymbol{\theta},\sigma^2) \\
&= \prod_{i=1}^{N} (2\pi\sigma^2)^{-\frac{1}{2}}\, e^{-\frac{1}{2\sigma^2}\left(y^{(i)}-\boldsymbol{\theta}^T\mathbf{x}^{(i)}\right)^2} \\
&= (2\pi\sigma^2)^{-\frac{N}{2}}\, e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{N}\left(y^{(i)}-\boldsymbol{\theta}^T\mathbf{x}^{(i)}\right)^2} \\
&= (2\pi\sigma^2)^{-\frac{N}{2}}\, e^{-\frac{1}{2\sigma^2}\left(\mathbf{y}-\mathbf{X}\boldsymbol{\theta}\right)^T\left(\mathbf{y}-\mathbf{X}\boldsymbol{\theta}\right)}
\end{aligned}$$

Maximum Likelihood for Linear Regression

- So we have got the likelihood as

$$p(\mathbf{y}|\mathbf{X};\boldsymbol{\theta},\sigma^2) = (2\pi\sigma^2)^{-\frac{m}{2}}(\mathbf{y}-\mathbf{X}\boldsymbol{\theta})^T(\mathbf{y}-\mathbf{X}\boldsymbol{\theta})$$

- The log likelihood is

$$l(\boldsymbol{\theta},\sigma^2) = -\frac{m}{2}\log(2\pi\sigma^2)(\mathbf{y}-\mathbf{X}\boldsymbol{\theta})^T(\mathbf{y}-\mathbf{X}\boldsymbol{\theta})$$

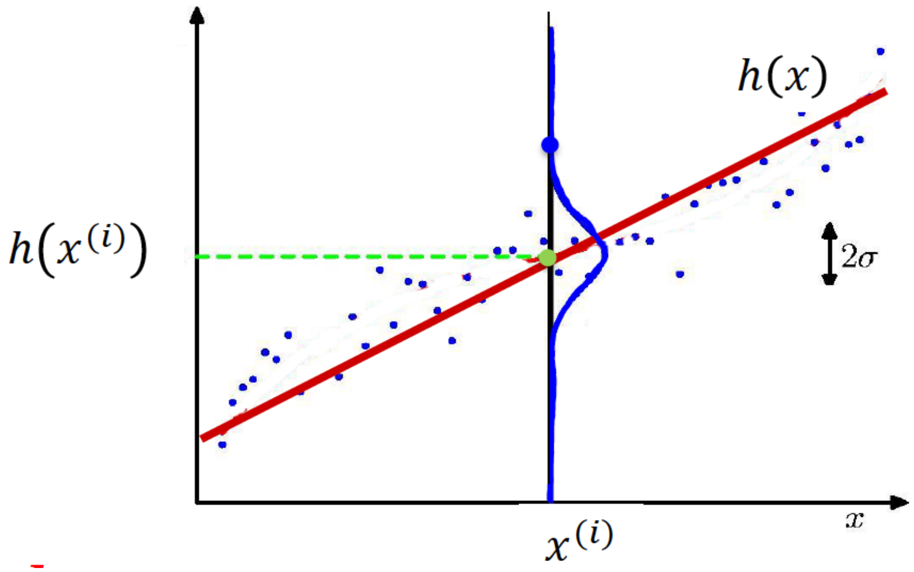
Maximum Likelihood for Linear Regression

- Likelihood: $p(\mathbf{y}|\mathbf{X};\boldsymbol{\theta},\sigma^2) = (2\pi\sigma^2)^{-\frac{m}{2}}(\mathbf{y}-\mathbf{X}\boldsymbol{\theta})^T(\mathbf{y}-\mathbf{X}\boldsymbol{\theta})$
- The log likelihood: $l(\boldsymbol{\theta},\sigma^2) = -\frac{m}{2}\log(2\pi\sigma^2)(\mathbf{y}-\mathbf{X}\boldsymbol{\theta})^T(\mathbf{y}-\mathbf{X}\boldsymbol{\theta})$
- Maximizing the likelihood w.r.t. θ means *maximizing* $-(\mathbf{y}-\mathbf{X}\boldsymbol{\theta})^T(\mathbf{y}-\mathbf{X}\boldsymbol{\theta})$

which in turn means *minimizing* $(\mathbf{y}-\mathbf{X}\boldsymbol{\theta})^T(\mathbf{y}-\mathbf{X}\boldsymbol{\theta})$

- Note the similarity with what we did earlier.
- Thus linear regression can be equivalently viewed as minimizing error sum of squares as well as maximum likelihood estimation under zero mean Gaussian noise assumption.

In a similar manner, the maximum likelihood estimate of σ^2 can also be calculated.



CS60010: Deep Learning

Logistic Regression

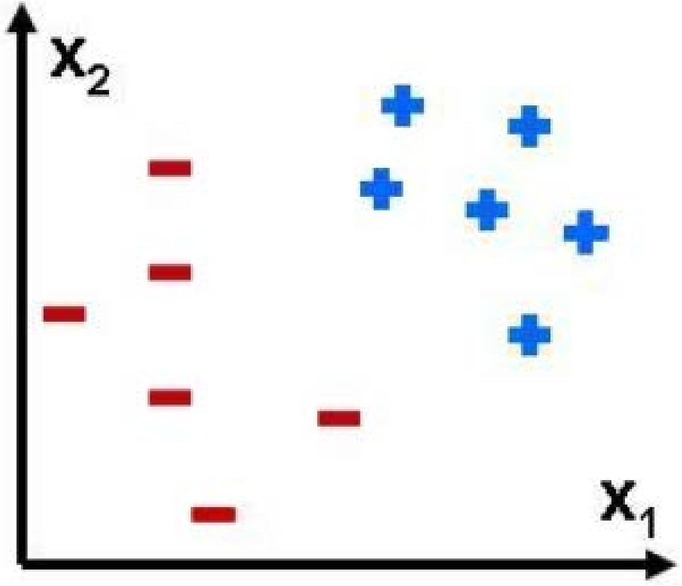
Sudeshna Sarkar

Spring 2021

Classification

A binary classifier is a mapping from $R^d \to \{-1, +1\}$

$$x \to \boxed{h} \to y$$



Training data set

$$\mathcal{D}_m = \{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$$

- Assume that each $x^{(i)}$ is a $d \times 1$ column vector
- Given a training set $\mathcal{D}_N$ and a classifier h, we can define the training error of h to be

$$\varepsilon_N(h) = \frac{1}{m} \sum_{i=1}^{m} \begin{cases} 1 & h(x^{(i)}) \neq y^{(i)} \\ 0 & \text{otherwise} \end{cases}$$

- For now, we will try to find a classifier with small training error (and hope it generalizes well to new data, and has a small test error

Learning algorithm

- A hypothesis class $\mathcal{H}$ is a set (finite or infinite) of possible classifiers, each of which represents a mapping from $R^d \to \{-1,+1\}$
- A learning algorithm is a procedure that takes a data set $\mathcal{D}_n$ as input and returns an element $h \in \mathcal{H}$

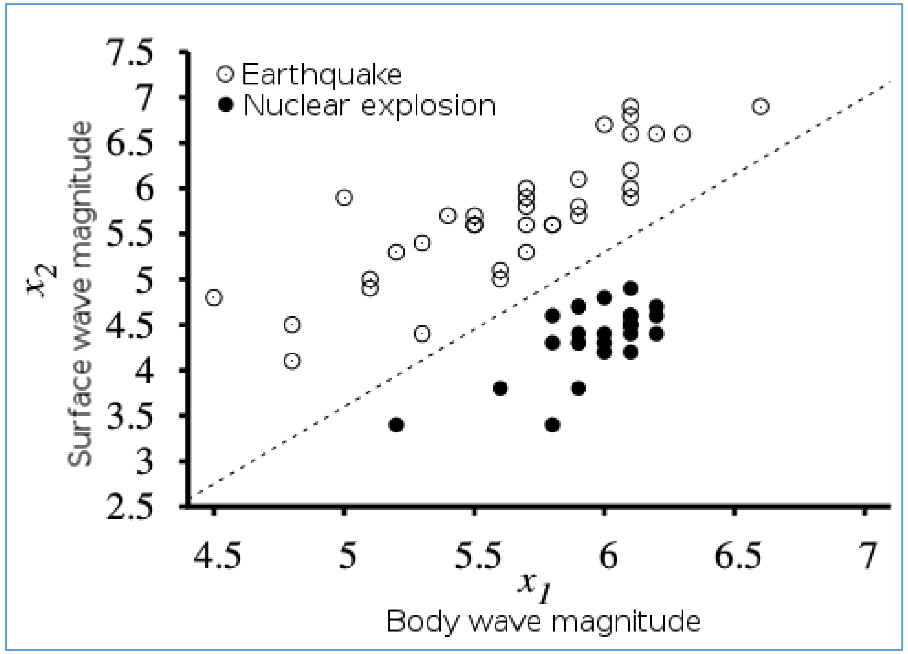
$$x \to \boxed{\text{learning alg } (\mathcal{H})} \to y$$

- Choice of $\mathcal{H}$ so as to get low test error

Hypothesis class :Linear classifiers

- A linear classifier in d dimensions is defined by
  - A vector of parameters $\theta \epsilon R^d$ and scalar $\theta_0 \epsilon \mathcal{R}$
  - We'll assume a d × 1 column vector

$$h(x;\theta,\theta_0) = sign(\theta^T x + \theta_0) = \begin{cases} +1 & if\, \theta^T x + \theta_0 > 0 \\ -1 & \text{otherwise} \end{cases}$$



θ, θ_0 specifies a hyperplane (decision boundary) that divides the instance space into two half-spaces.

Linear classifiers

$$h(x;\theta,\theta_0) = sign(\theta^T x + \theta_0) = \begin{cases} +1 & \text{if } \theta^T x + \theta_0 > 0 \\ -1 & \text{otherwise} \end{cases}$$

- θ, θ_0 specifies a hyperplane that divides the instance space into two half-spaces.
- The one that is on the same side as the normal vector is the positive half-space, and we classify all points in that space as positive.
- The half-space on the other side is negative and all points in it are classified as negative.

Linear Classifier with Hard Threshold

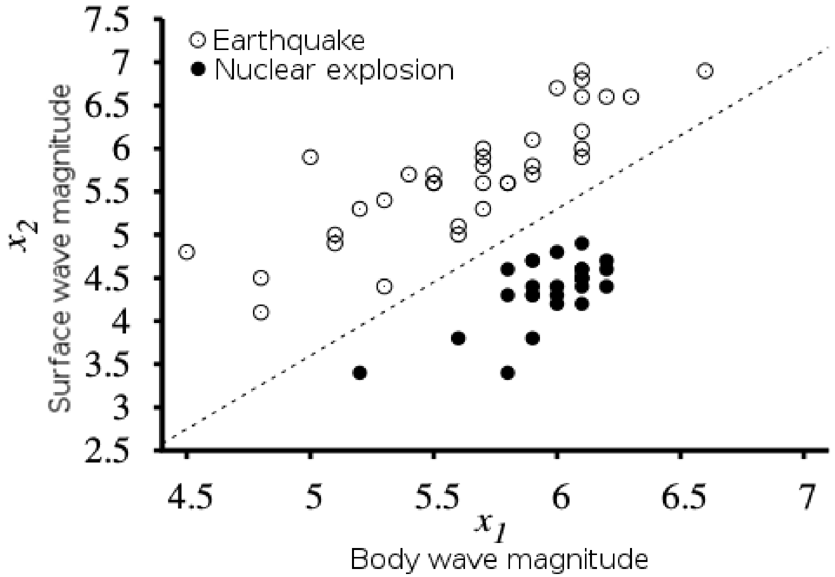
- The linear separator in the associated fig is given by

$$x_2 = 1.7x_1 - 4.9$$

$$\rightarrow -4.9 + 1.7x_1 - x_2 = 0$$

$$\rightarrow [-4.9 \quad 1.7 \quad -1]\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = 0$$

$$\boldsymbol{\theta}^T\mathbf{x} = 0$$



Classification Rule:

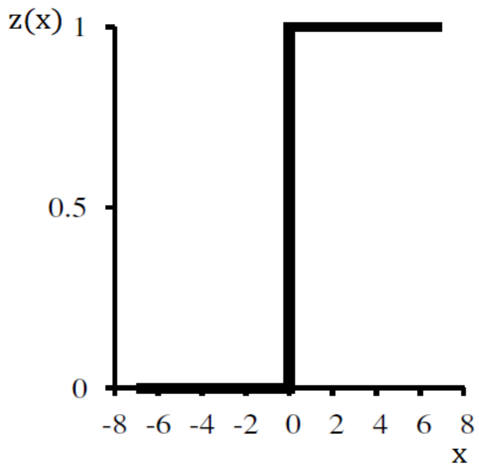
$$y(x) = \begin{cases} +1 & if\theta^T x > 0 \\ -1 & \text{otherwise} \end{cases}$$

Linear Classifier with Hard Threshold

Classification Rule:

$$y(x) = \begin{cases} +1 & \text{if } \theta^T x > 0 \\ -1 & \text{otherwise} \end{cases}$$

Alternatively, we can think y as the result of passing the linear function $\theta^T x$ through a threshold function.



- To get the linear separator we have to find the θ which minimizes classification error on the training set.
- We cannot use gradient descent at all points for the above threshold function

Perceptron Rule

- Perceptron Learning Rule can find a linear separator given the data is *linearly separable*.
- For data that are not linearly separable, the Perceptron algorithm fails.

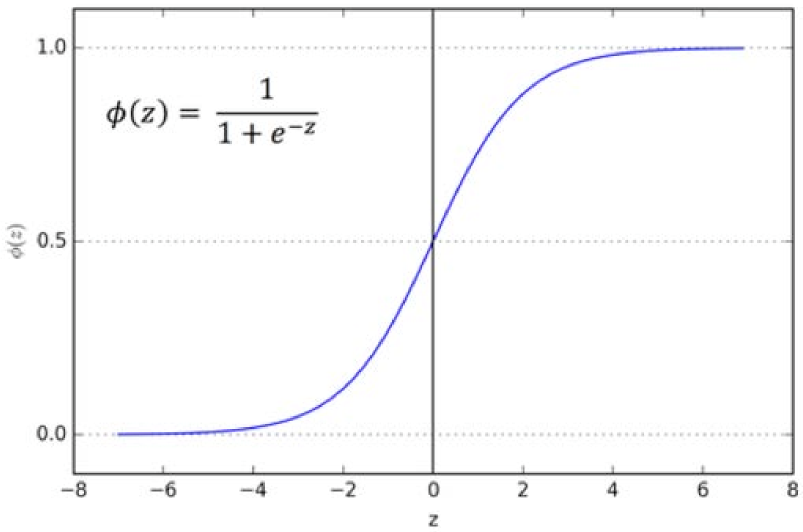
Linear Classifiers by Gradient Descent

For a gradient based optimization approach, we need to approximate hard threshold function with something smooth.

- logistic regression classifier

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

$$y = \sigma(h_\theta(\mathbf{x})) = \sigma(\boldsymbol{\theta}^T\mathbf{x})$$



Maximum Likelihood Estimation of Logistic Regression

The probability that an example belongs to class 1 is $P\left(y^{(i)} = 1|x^{(i)}; \boldsymbol{\theta}\right) = \sigma\left(\boldsymbol{\theta}^T\mathbf{x}^{(i)}\right)$

Thus $P\left(y^{(i)} = 0|x^{(i)}; \boldsymbol{\theta}\right) = 1 - \sigma\left(\boldsymbol{\theta}^T\mathbf{x}^{(i)}\right)$

Thus $P\left(y^{(i)}|x^{(i)}; \boldsymbol{\theta}\right) = \left(\sigma\left(\boldsymbol{\theta}^T\mathbf{x}^{(i)}\right)\right)^{y^{(i)}}\left(1 - \sigma\left(\boldsymbol{\theta}^T\mathbf{x}^{(i)}\right)\right)^{1-y^{(i)}}$

The joint probability of all the labels

$$\prod_{i=1}^{m}\left(\sigma\left(\boldsymbol{\theta}^T\mathbf{x}^{(i)}\right)\right)^{y^{(i)}}\left(1 - \sigma\left(\boldsymbol{\theta}^T\mathbf{x}^{(i)}\right)\right)^{1-y^{(i)}}$$

So the log likelihood for logistic regression is given by

$$l(\theta) = \sum_{i=1}^{m} y^{(i)} log\left(\sigma\left(\boldsymbol{\theta}^T\mathbf{x}^{(i)}\right)\right) + \left(1 - y^{(i)}\right) log\left(1 - \sigma\left(\boldsymbol{\theta}^T\mathbf{x}^{(i)}\right)\right)$$

Maximum Likelihood Estimation of Logistic Regression

Derivative of log likelihood w.r.t. one component of $\boldsymbol{\theta}$

$$
\begin{aligned}
\frac{\partial l(\boldsymbol{\theta})}{\partial \theta_j} &= \frac{\partial}{\partial \theta_j} \sum_{i=1}^{m} y^{(i)} \log \sigma\left(\boldsymbol{\theta}^T \mathbf{x}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \sigma\left(\boldsymbol{\theta}^T \mathbf{x}^{(i)}\right)\right) \\
&= \sum_{i-1}^{m} \left[\frac{y^{(i)}}{\sigma\left(\boldsymbol{\theta}^T \mathbf{x}^{(i)}\right)} - \frac{1 - y^{(i)}}{1 - \sigma\left(\boldsymbol{\theta}^T \mathbf{x}^{(i)}\right)} \right] \frac{\partial}{\partial \theta_j} \sigma\left(\boldsymbol{\theta}^T \mathbf{x}^{(i)}\right) \\
&= \sum_{i=1}^{m} \left[\frac{y^{(i)}}{\sigma\left(\boldsymbol{\theta}^T \mathbf{x}^{(i)}\right)} - \frac{1 - y^{(i)}}{1 - \sigma\left(\boldsymbol{\theta}^T \mathbf{x}^{(i)}\right)} \right] \sigma\left(\boldsymbol{\theta}^T \mathbf{x}^{(i)}\right) \left(1 - \sigma\left(\boldsymbol{\theta}^T \mathbf{x}^{(i)}\right)\right) \mathbf{x}_j^{(i)} \\
&= \sum_{i=1}^{m} \left[\frac{y^{(i)} - \sigma\left(\boldsymbol{\theta}^T \mathbf{x}^{(i)}\right)}{\sigma\left(\boldsymbol{\theta}^T \mathbf{x}^{(i)}\right)\left(1 - \sigma\left(\boldsymbol{\theta}^T \mathbf{x}^{(i)}\right)\right)} \right] \sigma\left(\boldsymbol{\theta}^T \mathbf{x}^{(i)}\right) \left(1 - \sigma\left(\boldsymbol{\theta}^T \mathbf{x}^{(i)}\right)\right) \mathbf{x}_j^{(i)} \\
&= \sum_{i=1}^{m} \left[y^{(i)} - \sigma\left(\boldsymbol{\theta}^T \mathbf{x}^{(i)}\right) \right] \mathbf{x}_j^{(i)} \qquad (12)
\end{aligned}
$$