## 1 Supervised classification and regression

The goal of classification and regression is to find a function $f \in \mathcal{F}$, such that given and input $\mathbf{x}$, $f(\mathbf{x})$, "does what we want it to do". We can define this as finding an $f$ solving the follwing optimization

$$\min_{f \in \mathcal{F}} \mathbb{E} \left( l(f\left(\mathbf{x}\right), y \right)$$

where $l$ is some loss function
**Classification**: $\mathbf{x} \in \mathbb{R}^p, y \in \{1, 2, \ldots, k\}$
**Regression**: $\mathbf{x} \in \mathbb{R}^p, y \in \mathbb{R}$

### 1.1 Empirical risk minimization

In supervised learning, we have a training dataset, $(\mathbf{x}_i, y_i)_{i=1}^n$. A learning procedure is a mapping from the training dataset to some $f \in \mathcal{F}$. The standard paradigm is to try to minimize the empirical risk:

$$\hat{f} = \min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} l\left(f\left(x_i\right), y_i\right)$$

What are some exceptions to this? Tree based methods, CART learning procedure

## 2 Loss functions

### 2.1 Loss functions for binary classification

Let's define the output as $y \in \{-1, 1\}$

**Zero-one loss**

$$l\left(\hat{y}, y\right) = \mathbf{1}\left\{\hat{y} = y\right\}$$

Another way to write this:

$$l\left(\hat{y}, y\right) = \mathbf{1}\left\{\hat{y}y \leq 0\right\}$$

What's the problem with using this loss function? It is hard to optimize: one must use combinatorial search, and there is no gradient.

**Surrogate losses**

Let's say your function $f\left(x\right)$ actually outputs a number representing 'degree of certainty' (e.g. distance from a margin, or a probability). Then we can define a function like $f\left(u\right)$

$$l\left(\hat{y}, y\right) = f\left\{yf\left(\mathbf{x}\right)\right\}$$

Examples:

$$\begin{aligned} f(u) &= \max(0, -u) \\ f(u) &= \max(0, 1-u) \\ f(u) &= \log\left(1 + e^{-u}\right) \end{aligned}$$

### 2.2 Loss function for regression

Output is some $y \in \mathbb{R}$. Define

$$l(\hat{y}, y) = (\hat{y} - y)^2$$

## 3 Statistical models

Principle of maximum likelihood is a way to derive common loss functions.

### 3.1 Logistic regression

A case of a generalized linear model:

$$g(P(y = 1 \mid \mathbf{x})) = \boldsymbol{\beta}^T \mathbf{x}$$

In the case of logistic regression, let

$$\begin{aligned} g(u) &= \log\left(\frac{u}{1-u}\right) \\ g^{-1}(u) &= \frac{e^u}{1+e^u} \end{aligned}$$

Therefore the modeling assumption is:

$$\log \frac{P(y = 1 \mid \mathbf{x})}{P(y = -1 \mid \mathbf{x})} = \boldsymbol{\beta}^T \mathbf{x}$$

Let g($\mathbf{x}$) $= \frac{e^{\boldsymbol{\beta}^T \mathbf{x}}}{1+e^{\boldsymbol{\beta}^T \mathbf{x}}}$. So $1 - g(\mathbf{x}) = \frac{1}{1+e^{\boldsymbol{\beta}^T \mathbf{x}}}$.

If $y = 1$, we have

$$R(\boldsymbol{\beta}) = \log\left(\frac{e^{\boldsymbol{\beta}^T \mathbf{x}}}{1+e^{\boldsymbol{\beta}^T \mathbf{x}}}\right) = -\log\left(1 + e^{-\boldsymbol{\beta}^T \mathbf{x}}\right) = -\log\left(1 + e^{-y\boldsymbol{\beta}^T \mathbf{x}}\right)$$

If $y = -1$, we have

$$R(\boldsymbol{\beta}) = \log\left(\frac{1}{1+e^{\boldsymbol{\beta}^T \mathbf{x}}}\right) = -\log\left(1 + e^{\boldsymbol{\beta}^T \mathbf{x}}\right) = \log\left(1 + e^{-y\boldsymbol{\beta}^T \mathbf{x}}\right)$$

Note this corresponds to the logistic loss function above.

### 3.2 Linear regression

Assumption: $y = \boldsymbol{\beta}^T \mathbf{x} + \epsilon$, where $\epsilon \sim \mathcal{N}\left(0, \sigma^2\right)$.

$$\begin{aligned} R(\boldsymbol{\beta}) &= \log \mathcal{N}\left(\boldsymbol{\beta}^T \mathbf{x}, \sigma^2\right)(y) \\ &= \log\left[\left(2\pi\sigma^2\right)^{-1/2} \exp\left\{-\frac{1}{2}\left(y - \boldsymbol{\beta}^T \mathbf{x}\right)^2\right\}\right] \\ &= K - \frac{1}{2}\left(y - \boldsymbol{\beta}^T \mathbf{x}\right)^2 \end{aligned}$$

## 4 How to solve these things?

We want to solve

$$\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} l\left(f\left(x_i\right), y_i\right)$$

There have been decades of research on solving these kinds of problems. There is very rarely a closed form solution. Linear regression is a case where there is. The problem is

$$\min_{\boldsymbol{\beta}} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \boldsymbol{\beta}^T \mathbf{x}_i\right)^2$$

Let's switch to vector - matrix notation. Define

$$\begin{aligned} \mathbf{y} &= \left(y_1, \ldots, y_n\right)^T \\ \mathbf{X} &= \text{design matrix} \end{aligned}$$

Then we rewrite

$$l(\boldsymbol{\beta}) = \frac{1}{n} \left\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\right\|^2$$

Let's solve it! We ignore the $\frac{1}{n}$ term, because it is just a scaling.

$$\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 = \mathbf{y}^{\mathbf{T}}\mathbf{y} - 2\left(\mathbf{X}\boldsymbol{\beta}\right)^T \mathbf{y} + \boldsymbol{\beta}^T \mathbf{X}^T \mathbf{X} \boldsymbol{\beta}$$

Let's take the gradient w.r.t. $\boldsymbol{\beta}$:

$$\nabla l = -2\mathbf{X}^T \mathbf{y} + 2\mathbf{X}^T \mathbf{X} \boldsymbol{\beta}$$

Set it to zero

$$\begin{aligned} 0 &= -2\mathbf{X}^T \mathbf{y} + 2\mathbf{X}^T \mathbf{X} \boldsymbol{\beta} \\ \mathbf{X}^T \mathbf{X} \boldsymbol{\beta} &= \mathbf{X}^T \mathbf{y} \\ \boldsymbol{\beta} &= \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{y} \end{aligned}$$

This is a closed form solution, but it involves matrix multipication. Also it requires $\left(\mathbf{X}^T \mathbf{X}\right)^{-1}$ to exist.

## 4.1 Gradient descent

In general the function class $\mathcal{F}$is composed of some parametrized functions.

$$R(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} l\left(f_{\boldsymbol{\beta}}(x_i), y_i\right)$$

To minimize $R$, we can run gradient descent

$$\boldsymbol{\beta}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta \nabla R(\boldsymbol{\beta}_t)$$

In higher dimensions, the directional derivative is

$$\begin{aligned}\nabla_{\mathbf{v}} R(\boldsymbol{\beta}) &= \frac{d}{dt} R(\boldsymbol{\beta} + t\mathbf{v})\,|_{t=0} \\ &= \mathbf{v}^T \nabla_{\mathbf{v}} R(\boldsymbol{\beta})\end{aligned}$$

Heuristic justification of gradient descent

$$\begin{aligned}R\left(\boldsymbol{\beta}_{t+1}\right) &= R\left(\boldsymbol{\beta}_t - \eta \nabla R(\boldsymbol{\beta}_t)\right) \\ &\approx R(\boldsymbol{\beta}_t) - \eta \nabla R(\boldsymbol{\beta}_t)^T \nabla R(\boldsymbol{\beta}_t) \\ &= R(\boldsymbol{\beta}_t) - \eta \|\nabla R(\boldsymbol{\beta}_t)\|^2\end{aligned}$$

## 4.2 Stochastic gradient descent

The gradient of $R$ is a sum over all training examples

$$\begin{aligned}\nabla R(\boldsymbol{\beta}) &= \nabla\left(\frac{1}{n}\sum_i R_i(\boldsymbol{\beta})\right) \\ &= \frac{1}{n}\sum_{i=1}^{n} \nabla l\left(f_{\boldsymbol{\beta}}(x_i), y_i\right)\end{aligned}$$

To evaluate one step of gradient descent, we have to look at all n examples and compute the loss. This is expensive, potentially very much so if the gradient is expensive to compute, as in a deep network. In stochastic gradient descent, we sample a single i, and calculate the gradient

$$\nabla R_i(\boldsymbol{\beta}) = \nabla l\left(f_{\boldsymbol{\beta}}(x_i), y_i\right)$$

The SGD update rule is:

$$\boldsymbol{\beta}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta \nabla R_i(\boldsymbol{\beta}_t)$$

Why is it called stochastic gradient descent? Because of the random choice of i. By random sampling:

$$\mathbb{E}\left(\nabla R_i(\boldsymbol{\beta}_t)\right) = \nabla R(\boldsymbol{\beta})$$

In general, SGD refers to a case where the update step is taken with a noisy gradient - i.e. one whose expectation is equal to the true gradient. The vast majority of the time, it refers this algorithm, in which an update is made to the parameters after visiting a single datapoint.

## 4.3 SGD for linear regression

$$R(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \boldsymbol{\beta}^T\mathbf{x}_i\right)^2$$

So $R_i = \left(y_i - \boldsymbol{\beta}^T\mathbf{x}_i\right)^2$. Let's calculate:

$$\nabla R_i(\boldsymbol{\beta}) = -2\left(y_i - \boldsymbol{\beta}^T\mathbf{x}_i\right)\mathbf{x}_i$$

So what is the SGD update rule?

$$\boldsymbol{\beta}_{t+1} \leftarrow \boldsymbol{\beta}_t + \eta\left(y_i - \boldsymbol{\beta}^T\mathbf{x}_i\right)\mathbf{x}_i$$

This is the LMS algorithm! The predictor vector is added to the parameter, scaled by the residual.

## 4.4 SGD for classification

Let's use a linear classifier. Let's use the loss function $l(u) = \max(0, -u)$. Then

$$R(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} l\left(y_i\boldsymbol{\beta}^T\mathbf{x}_i\right)$$

So $R_i = \max\left(0, -y_i\boldsymbol{\beta}^T\mathbf{x}_i\right)$. What is

$$l'(u) = \begin{cases} -1 & \text{if } u < 0 \\ 0 & \text{if } u \geq 0 \end{cases}$$

Let's use the chain rule.

$$\nabla R_i(\boldsymbol{\beta}) = \begin{cases} -y_i\mathbf{x}_i & \text{if } y_i\boldsymbol{\beta}^T\mathbf{x}_i < 0 \\ 0 & \text{if } y_i\boldsymbol{\beta}^T\mathbf{x}_i \geq 0 \end{cases}$$

Does this look familiar? Let's say $y_i = 1$. If $\boldsymbol{\beta}^T\mathbf{x}_i < 0$, then this means the classifier put the example on the wrong side of the separating hyperplane. Then our update is

$$\boldsymbol{\beta}_{t+1} \leftarrow \boldsymbol{\beta}_t + \eta\mathbf{x}_i$$

This is the Perceptron update rule!

## 4.5 SGD for classification - logistic regression

Recall the logistic loss function:

$$R(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n}\log\left(1 + e^{-y\boldsymbol{\beta}^T\mathbf{x}}\right)$$

Then calculate the gradient

$$\begin{aligned}\nabla R_i\left(\boldsymbol{\beta}\right) &= \frac{-y\mathbf{x}e^{-y\boldsymbol{\beta}^T\mathbf{x}}}{1+e^{-y\boldsymbol{\beta}^T\mathbf{x}}} \\ &= -y\mathbf{x}\left(1-\text{logistic}\left(y\boldsymbol{\beta}^T\mathbf{x}\right)\right)\end{aligned}$$

This depends on the fact that logistic $(-u) = 1 -$ logistic $(u)$

Note that this is like a smoothed version of the perceptron rule: as the classifier is more confidently correct, the update becomes close to 0. As it is more confidently wrong, the update is close to $-y\mathbf{x}$, the perceptron rule.

Let's derive this another way (more akin to backpropagation). Consider the neural network

$$\begin{bmatrix}x_1\\ \vdots \\ x_p\end{bmatrix} \to a \to p \to l$$

Where

$$\begin{aligned}a &= \boldsymbol{\beta}^T\mathbf{x} \\ p &= \frac{e^a}{1+e^a} \\ l &= -\mathbf{1}\left\{y=1\right\}\log p - \mathbf{1}\left\{y=-1\right\}\log\left(1-p\right)\end{aligned}$$

Let's calculate the gradient w.r.t. $\boldsymbol{\beta}$.

$$\nabla l = \frac{dl}{dp}\frac{dp}{da}\left(\nabla a\right)$$

Let's say $y = 1$:

$$\frac{dl}{dp} = -\frac{1}{p}$$

Then:

$$\frac{dp}{da} = \frac{d}{da}\left(\frac{e^a}{1+e^a}\right) = \frac{e^a+e^{2a}-e^{2a}}{\left(1+e^a\right)^2} = \frac{e^a}{\left(1+e^a\right)^2} = \left(\frac{e^a}{1+e^a}\right)\left(\frac{1}{1+e^a}\right) = p\left(1-p\right)$$

Also, $\nabla a = \mathbf{x}$.

So the full gradient becomes (for $y = 1$):

$$\nabla l = -\frac{1}{p}\left(p\left(1-p\right)\right)\mathbf{x} = -\left(1-p\right)\mathbf{x}$$

Now let's say $y = -1$. Then

$$\frac{dl}{dp} = \frac{1}{1-p}$$

So

$$\nabla l = \frac{1}{1-p}\left(p\left(1-p\right)\right)\mathbf{x} = p\mathbf{x}$$

## 5 Multiclass classification

How to generalize a binary classifier to a multi-class classifier? There are many ways. In deep learning, by far the most common is to use multinomial logistic regression / softmax activations with cross-entropy loss.

The assumption of multinomial logistic regression is that

$$P\left(y = j \mid \mathbf{x}\right) \propto \exp\left(\boldsymbol{\beta}_j^T \mathbf{x}\right)$$

Or in other words

$$P\left(y = j \mid \mathbf{x}\right) = p_j = \frac{e^{\boldsymbol{\beta}_j^T \mathbf{x}}}{\sum_{l=1}^{k} e^{\boldsymbol{\beta}_l^T \mathbf{x}}}$$

This is known as the softmax function. The denominator ensures that the probabilities are normalized, i.e. that they sum to 1. This is the standard way to convert a vector of real-valued numbers into a probability distribution. We can think of this as a neural network:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \rightarrow \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \end{bmatrix} \rightarrow \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_k \end{bmatrix}$$

Where $a_j = \boldsymbol{\beta}_j^T \mathbf{x}$. Note that it is invariant to constant shifts:

$$\frac{e^{C + \boldsymbol{\beta}_k^T \mathbf{x}}}{\sum_{j=1}^{k} e^{C + \boldsymbol{\beta}_k^T \mathbf{x}}} = \frac{e^{C} e^{\boldsymbol{\beta}_k^T \mathbf{x}}}{e^{C} \sum_{j=1}^{k} e^{\boldsymbol{\beta}_k^T \mathbf{x}}} = \frac{e^{\boldsymbol{\beta}_k^T \mathbf{x}}}{\sum_{j=1}^{k} e^{\boldsymbol{\beta}_k^T \mathbf{x}}}$$

Because of this, layer is overparametrized: we can arbitrarily set one of the activations to zero. We can set $a_k = 0$, and $a_j = \boldsymbol{\beta}_j^T \mathbf{x}$ for $j \in \{1, \ldots, k-1\}$. Then we have a total of $p \times (k-1)$ parameters given by the vectors $\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_{k-1}$.