

CSCI567 Machine Learning (Fall 2025)

Haipeng Luo

University of Southern California

Sep 12, 2025

Administration

- HW 1 is due on Wed, Sep 17th.
- recall the late day policy: 3 in total, at most 1 for each homework

Outline

- 1 Review of Last Lecture
- 2 Linear Classifiers and Surrogate Losses
- 3 A Detour of Numerical Optimization Methods
- 4 Perceptron
- 5 Logistic Regression

Outline

- 1 Review of Last Lecture
- 2 Linear Classifiers and Surrogate Losses
- 3 A Detour of Numerical Optimization Methods
- 4 Perceptron
- 5 Logistic Regression

Regression

Predicting a continuous outcome variable using past observations

- temperature, amount of rainfall, house price, etc.

Key difference from classification

- continuous vs discrete
- measure *prediction errors* differently.
- lead to quite different learning algorithms.

Linear Regression: regression with linear models: $f(x) = w^T x$

Least square solution

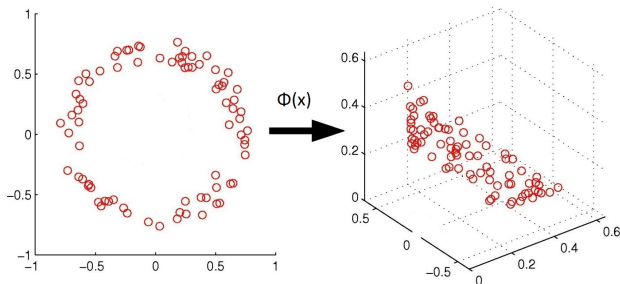
$$\begin{aligned}
 \mathbf{w}^* &= \underset{\mathbf{w}}{\operatorname{argmin}} \operatorname{RSS}(\mathbf{w}) \\
 &= \underset{\mathbf{w}}{\operatorname{argmin}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 \\
 &= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}
 \end{aligned}$$

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

Two approaches to find the minimum:

- find **stationary points** by setting gradient = 0
- “**complete the square**”

Regression with nonlinear basis



Model: $f(x) = w^T \phi(x)$ where $w \in \mathbb{R}^M$

Similar least square solution: $w^* = (\Phi^T \Phi)^{-1} \Phi^T y$

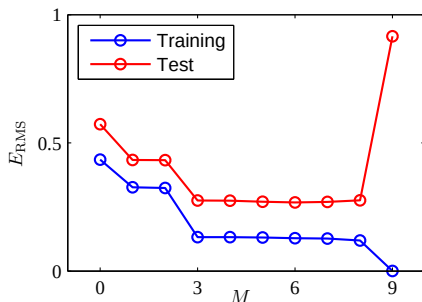
Underfitting and Overfitting

$M \leq 2$ is *underfitting* the data

- large training error
- large test error

$M \geq 9$ is *overfitting* the data

- small training error
- **large test error**



How to prevent overfitting? more data + regularization

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \left(\operatorname{RSS}(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2 \right) = (\Phi^T \Phi + \lambda \mathbf{I})^{-1} \Phi^T \mathbf{y}$$

General idea to derive ML algorithms

Step 1. Pick a set of **models** $\mathcal{F}$

- e.g. $\mathcal{F} = \{f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} \mid \mathbf{w} \in \mathbb{R}^D\}$
- e.g. $\mathcal{F} = \{f(\mathbf{x}) = \mathbf{w}^T \Phi(\mathbf{x}) \mid \mathbf{w} \in \mathbb{R}^M\}$

Step 2. Define **error/loss** $L(y', y)$

Step 3. Find **(regularized) empirical risk minimizer (ERM)**:

$$\mathbf{f}^* = \operatorname{argmin}_{f \in \mathcal{F}} \sum_{n=1}^N L(f(x_n), y_n) + \lambda R(f)$$

ML becomes optimization

Today: another exercise of this recipe + a closer look at Step 3

Outline

- 1 Review of Last Lecture
- 2 Linear Classifiers and Surrogate Losses**
- 3 A Detour of Numerical Optimization Methods
- 4 Perceptron
- 5 Logistic Regression

Classification

Recall the setup:

- input (feature vector): $x \in \mathbb{R}^D$
- output (label): $y \in [C] = \{1, 2, \dots, C\}$
- goal: learn a mapping $f : \mathbb{R}^D \rightarrow [C]$

This lecture: **binary classification**

- Number of classes: $C = 2$
- Labels: $\{-1, +1\}$ (cat or dog, fraud or not, price up or down...)

We have discussed **nearest neighbor classifier**:

- require carrying the training set
- intuitive but more like a heuristic

Deriving classification algorithms

Let's follow the recipe:

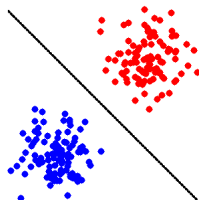
Step 1. Pick a set of models $\mathcal{F}$.

Again try linear models, but how to predict a label using $w^T x$?

Sign of $w^T x$ predicts the label:

$$\text{sign}(w^T x) = \begin{cases} +1 & \text{if } w^T x > 0 \\ -1 & \text{if } w^T x \leq 0 \end{cases}$$

(Sometimes use sgn for sign too.)



The models

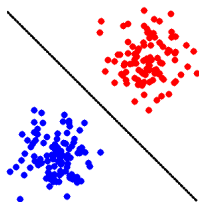
The set of **(separating) hyperplanes**:

$$\mathcal{F} = \{f(x) = \text{sgn}(w^T x) \mid w \in \mathbb{R}^D\}$$

Good choice for *linearly separable* data, i.e., $\exists w$ s.t.

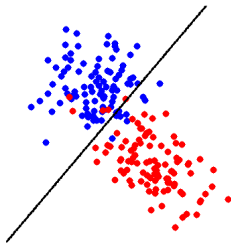
$$\text{sgn}(w^T x_n) = y_n \quad \text{or} \quad y_n w^T x_n > 0$$

for all $n \in [N]$.



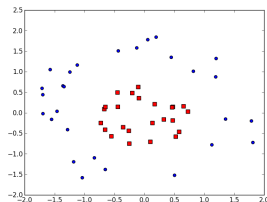
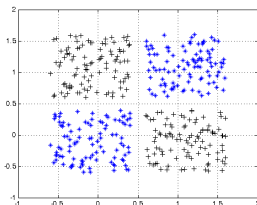
The models

Still makes sense for “almost” linearly separable data



The models

For clearly not linearly separable data,



Again can apply a **nonlinear mapping** Φ :

$$\mathcal{F} = \{f(x) = \text{sgn}(w^T \Phi(x)) \mid w \in \mathbb{R}^M\}$$

More discussions in future lectures.

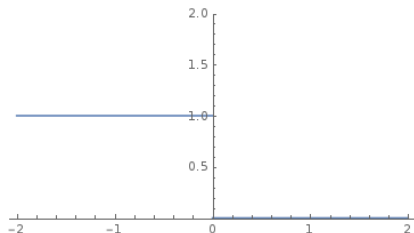
0-1 Loss

Step 2. Define error/loss $L(y', y)$.

Most natural one for classification: **0-1 loss** $L(y', y) = \mathbb{I}[y' \neq y]$

For classification, more convenient to look at the loss **as a function of** $yw^T x$. That is, with

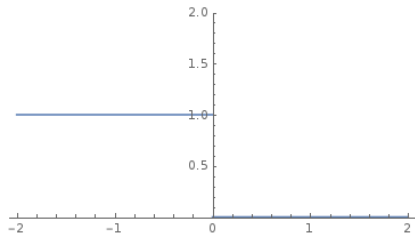
$$\ell_{0-1}(z) = \mathbb{I}[z \leq 0]$$



the loss for hyperplane w on example (x, y) is $\ell_{0-1}(yw^T x)$

Minimizing 0-1 loss is hard

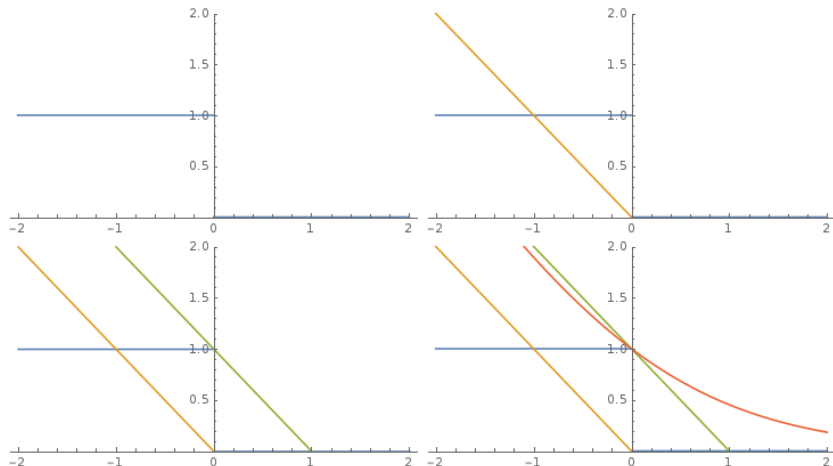
However, 0-1 loss is *not convex*.



Even worse, minimizing 0-1 loss is *NP-hard in general*.

Surrogate Losses

Solution: find a **convex surrogate loss**



- **perceptron loss** $\ell_{\text{perceptron}}(z) = \max\{0, -z\}$ (used in Perceptron)

ML becomes convex optimization

Step 3. Find ERM:

$$\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w} \in \mathbb{R}^D} \sum_{n=1}^N \ell(y_n \mathbf{w}^T \mathbf{x}_n) = \operatorname{argmin}_{\mathbf{w} \in \mathbb{R}^D} \frac{1}{N} \sum_{n=1}^N \ell(y_n \mathbf{w}^T \mathbf{x}_n)$$

where $\ell(\cdot)$ can be perceptron/hinge/logistic loss

- *no closed-form* in general (unlike linear regression)
- can apply general convex optimization methods

Note: minimizing perceptron loss *does not really make sense* (try $\mathbf{w} = \mathbf{0}$), but the algorithm derived from this perspective does.

Outline

- 1 Review of Last Lecture
- 2 Linear Classifiers and Surrogate Losses
- 3 A Detour of Numerical Optimization Methods**
 - First-order methods
 - Second-order methods
- 4 Perceptron
- 5 Logistic Regression

Numerical optimization

Problem setup

- Given: a function $F(\boldsymbol{w})$
- Goal: minimize $F(\boldsymbol{w})$ (approximately)

First-order optimization methods

Two simple yet extremely popular methods

- **Gradient Descent (GD)**: simple and fundamental
- **Stochastic Gradient Descent (SGD)**: faster, effective for large-scale problems

Gradient is sometimes referred to as *first-order* information of a function. Therefore, these methods are called *first-order methods*.

Gradient Descent (GD)

GD: keep moving in the *negative gradient direction*

Start from some (random) $\mathbf{w}^{(0)}$. For $t = 0, 1, 2, \dots$

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \nabla F(\mathbf{w}^{(t)})$$

where $\eta > 0$ is called step size or learning rate

- in theory η should be set in terms of some parameters of F
- in practice we often try different small values

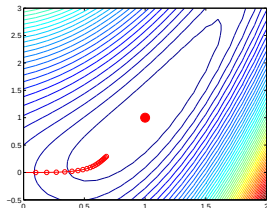
Stop when $F(\mathbf{w}^{(t)})$ **does not change much** or t **reaches a fixed number**

Why GD?

Intuition: by first-order **Taylor approximation**

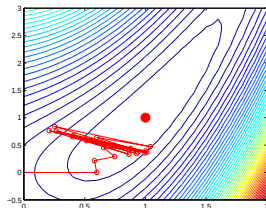
$$F(\mathbf{w}) \approx F(\mathbf{w}^{(t)}) + \nabla F(\mathbf{w}^{(t)})^T (\mathbf{w} - \mathbf{w}^{(t)})$$

$$\begin{aligned} \text{GD ensures } F(\mathbf{w}^{(t+1)}) &\approx F(\mathbf{w}^{(t)}) + \nabla F(\mathbf{w}^{(t)})^T (\mathbf{w}^{(t+1)} - \mathbf{w}^{(t)}) \\ &= F(\mathbf{w}^{(t)}) - \eta \|\nabla F(\mathbf{w}^{(t)})\|_2^2 \leq F(\mathbf{w}^{(t)}) \end{aligned}$$



reasonable η decreases function value

See Colab Example 1



but large η is unstable

More on learning rate

Learning rate η might need to be **changing** over iterations

- often **decreasing**, according to some schedule (e.g., $\eta \approx \frac{1}{t}$ or $\frac{1}{\sqrt{t}}$)
- think $F(w) = |w|$

Adaptive and automatic step size tuning is an active research area

- notable examples: AdaGrad, Adam, etc.
- ideas: tune η based on past gradient information

Stochastic Gradient Descent (SGD)

GD: keep moving in the negative gradient direction

SGD: keep moving in some *noisy* negative gradient direction

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \tilde{\nabla} F(\mathbf{w}^{(t)})$$

where $\tilde{\nabla} F(\mathbf{w}^{(t)})$ is a random variable (called **stochastic gradient**) s.t.

$$\mathbb{E} \left[\tilde{\nabla} F(\mathbf{w}^{(t)}) \right] = \nabla F(\mathbf{w}^{(t)}) \quad (\text{unbiasedness})$$

See Colab Example 1.

More examples coming soon. Key point: it could be *much faster to obtain a stochastic gradient!*

Convergence guarantees — convex objectives

Many for both GD and SGD on **convex objectives**.

They tell you how many iterations t (in terms of ϵ) needed to achieve

$$F(\mathbf{w}^{(t)}) - F(\mathbf{w}^*) \leq \epsilon$$

- usually SGD needs more iterations
- but again each iteration takes less time

Convergence guarantees — nonconvex objectives

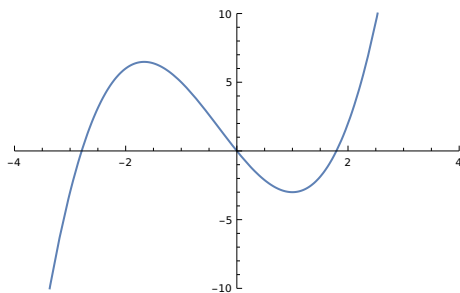
Even for *nonconvex objectives*, some guarantees exist: e.g. how many iterations t (in terms of ϵ) needed to achieve

$$\|\nabla F(\mathbf{w}^{(t)})\| \leq \epsilon$$

- that is, how close $\mathbf{w}^{(t)}$ is as an **approximate stationary point**
- for convex objectives, stationary point $\Rightarrow$ global minimizer
- for nonconvex objectives, *what does it mean?*

Convergence guarantees — nonconvex objectives

A stationary point can be a **local minimizer** or even a **local/global maximizer** (but the latter is not an issue for GD/SGD).

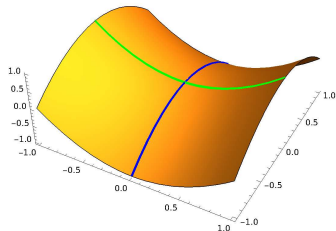


$$f(w) = w^3 + w^2 - 5w$$

Convergence guarantees — nonconvex objectives

A stationary point can also be *neither a local minimizer nor a local maximizer!* This is called a **saddle point**.

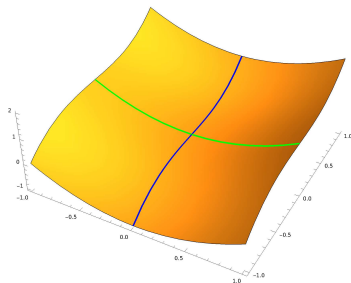
- $f(\mathbf{w}) = w_1^2 - w_2^2$
- $\nabla f(\mathbf{w}) = (2w_1, -2w_2)$
- so $\mathbf{w} = (0, 0)$ is stationary
- local max for **blue direction** ($w_1 = 0$)
- local min for **green direction** ($w_2 = 0$)
- but GD gets stuck at $(0, 0)$ only if initialized along the **green direction**
- so not a real issue especially *when initialized randomly* (Colab Example 2)



Convergence guarantees — nonconvex objectives

But not all saddle points look like a “saddle” ...

- $f(\mathbf{w}) = w_1^2 + w_2^3$
- $\nabla f(\mathbf{w}) = (2w_1, 3w_2^2)$
- so $\mathbf{w} = (0, 0)$ is stationary
- not local min/max for **blue direction** ($w_1 = 0$)
- GD gets stuck at $(0, 0)$ for *any initial point with $w_2 \geq 0$ and small η* (Colab Example 3)



Even worse, distinguishing local min and saddle point is generally **NP-hard**.

Convergence guarantees

Summary:

- GD/SGD converges to a stationary point
- for convex objectives, this is all we need
- for nonconvex objectives, can get stuck at local minimizers or “bad” saddle points (random initialization escapes “good” saddle points)
- recent research shows that *many problems have no “bad” saddle points or even “bad” local minimizers*
- justify the practical effectiveness of GD/SGD (default method to try)

Second-order methods

Recall the intuition of GD: we look at first-order **Taylor approximation**

$$F(\mathbf{w}) \approx F(\mathbf{w}^{(t)}) + \nabla F(\mathbf{w}^{(t)})^T (\mathbf{w} - \mathbf{w}^{(t)})$$

What if we look at *second-order* Taylor approximation?

$$F(\mathbf{w}) \approx F(\mathbf{w}^{(t)}) + \nabla F(\mathbf{w}^{(t)})^T (\mathbf{w} - \mathbf{w}^{(t)}) + \frac{1}{2} (\mathbf{w} - \mathbf{w}^{(t)})^T \mathbf{H}_t (\mathbf{w} - \mathbf{w}^{(t)})$$

where $\mathbf{H}_t = \nabla^2 F(\mathbf{w}^{(t)}) \in \mathbb{R}^{D \times D}$ is the *Hessian* of F at $\mathbf{w}^{(t)}$, i.e.,

$$H_{t,ij} = \left. \frac{\partial^2 F(\mathbf{w})}{\partial w_i \partial w_j} \right|_{\mathbf{w}=\mathbf{w}^{(t)}}$$

(think “second derivative” when $D = 1$)

Newton method

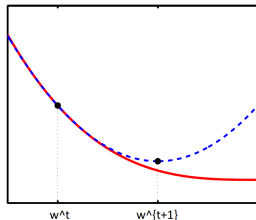
If we minimize the second-order approximation (via “complete the square”)

$$F(\mathbf{w})$$

$$\begin{aligned} &\approx F(\mathbf{w}^{(t)}) + \nabla F(\mathbf{w}^{(t)})^T (\mathbf{w} - \mathbf{w}^{(t)}) + \frac{1}{2} (\mathbf{w} - \mathbf{w}^{(t)})^T \mathbf{H}_t (\mathbf{w} - \mathbf{w}^{(t)}) \\ &= \frac{1}{2} \left(\mathbf{w} - \mathbf{w}^{(t)} + \mathbf{H}_t^{-1} \nabla F(\mathbf{w}^{(t)}) \right)^T \mathbf{H}_t \left(\mathbf{w} - \mathbf{w}^{(t)} + \mathbf{H}_t^{-1} \nabla F(\mathbf{w}^{(t)}) \right) + \text{const.} \end{aligned}$$

for strictly convex F (so H_t is **positive definite**), we obtain **Newton method**:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \mathbf{H}_t^{-1} \nabla F(\mathbf{w}^{(t)})$$



Comparing GD and Newton

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \nabla F(\mathbf{w}^{(t)}) \quad (\text{GD})$$

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \mathbf{H}_t^{-1} \nabla F(\mathbf{w}^{(t)}) \quad (\text{Newton})$$

Both are iterative optimization procedures, but Newton method

- has no learning rate η (so **no tuning needed!**)
- converges **super fast** in terms of #iterations (for convex objectives)
 - e.g. how many iterations needed when applied to a quadratic?
- computing Hessian in each iteration is *very slow* though
- does not really make sense for *nonconvex objectives*

Outline

- 1 Review of Last Lecture
- 2 Linear Classifiers and Surrogate Losses
- 3 A Detour of Numerical Optimization Methods
- 4 Perceptron**
- 5 Logistic Regression

Recall the perceptron loss

$$\begin{aligned} F(\mathbf{w}) &= \frac{1}{N} \sum_{n=1}^N \ell_{\text{perceptron}}(y_n \mathbf{w}^T \mathbf{x}_n) \\ &= \frac{1}{N} \sum_{n=1}^N \max\{0, -y_n \mathbf{w}^T \mathbf{x}_n\} \end{aligned}$$

Let's approximately minimize it with GD/SGD.

Applying GD to perceptron loss

Objective

$$F(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N \max\{0, -y_n \mathbf{w}^T \mathbf{x}_n\}$$

Gradient (or really *sub-gradient*) is

$$\nabla F(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N -\mathbb{I}[y_n \mathbf{w}^T \mathbf{x}_n \leq 0] y_n \mathbf{x}_n$$

(only misclassified examples contribute to the gradient)

GD update

$$\mathbf{w} \leftarrow \mathbf{w} + \frac{\eta}{N} \sum_{n=1}^N \mathbb{I}[y_n \mathbf{w}^T \mathbf{x}_n \leq 0] y_n \mathbf{x}_n$$

Slow: each update makes one pass of the entire training set!

Applying SGD to perceptron loss

How to construct a stochastic gradient?

One common trick: pick one example $n \in [N]$ uniformly at random, let

$$\tilde{\nabla} F(\mathbf{w}^{(t)}) = -\mathbb{I}[y_n \mathbf{w}^T \mathbf{x}_n \leq 0] y_n \mathbf{x}_n$$

clearly unbiased (convince yourself).

SGD update:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta \mathbb{I}[y_n \mathbf{w}^T \mathbf{x}_n \leq 0] y_n \mathbf{x}_n$$

Fast: each update touches only one data point!

Conveniently, objective of most ML tasks is a *finite sum* (over each training point) and the above trick applies!

The Perceptron Algorithm

Perceptron algorithm is SGD with $\eta = 1$ applied to perceptron loss:

Repeat:

- Pick a data point $\mathbf{x}_n$ uniformly at random
- If $\text{sgn}(\mathbf{w}^T \mathbf{x}_n) \neq y_n$

$$\mathbf{w} \leftarrow \mathbf{w} + y_n \mathbf{x}_n$$

Note:

- $\mathbf{w}$ is always a *linear combination* of the training examples
- why $\eta = 1$? Does not really matter in terms of prediction of $\mathbf{w}$

Why does it make sense?

If the current weight $\mathbf{w}$ makes a mistake

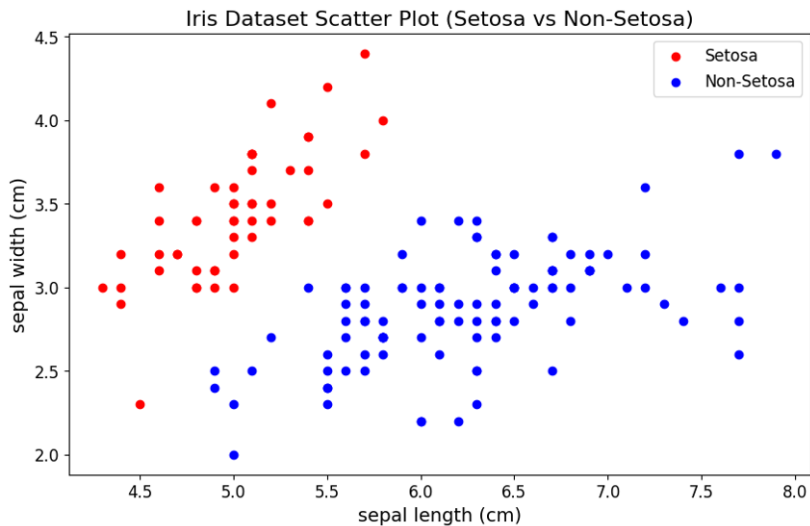
$$y_n \mathbf{w}^T \mathbf{x}_n < 0$$

then after the update $\mathbf{w}' = \mathbf{w} + y_n \mathbf{x}_n$ we have

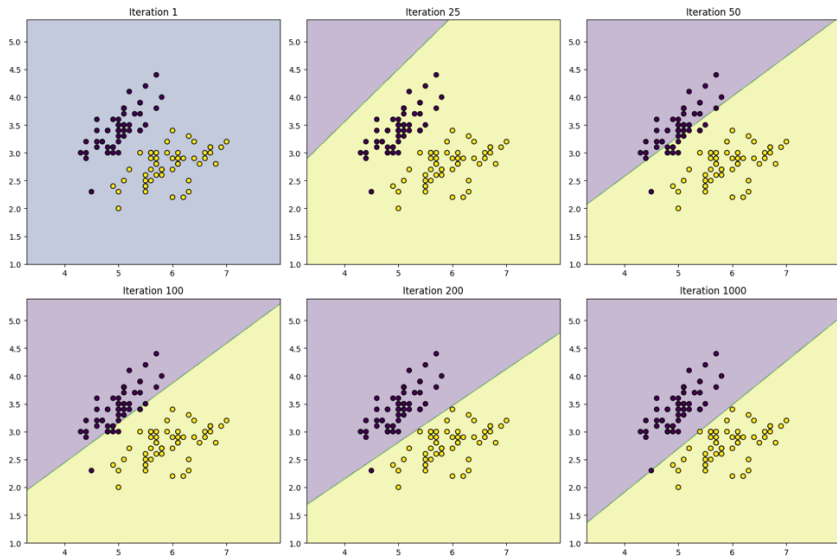
$$y_n \mathbf{w}'^T \mathbf{x}_n = y_n \mathbf{w}^T \mathbf{x}_n + y_n^2 \mathbf{x}_n^T \mathbf{x}_n \geq y_n \mathbf{w}^T \mathbf{x}_n$$

Thus it is more likely to get it right after the update.

Example: Iris Dataset



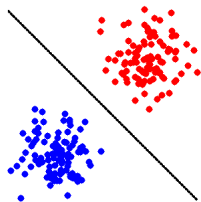
Example: Perceptron for Iris Dataset



Any theory?

If training set is linearly separable

- Perceptron *converges in a finite number of steps*
- training error is 0



There are also guarantees when the data are not linearly separable.

Outline

- 1 Review of Last Lecture
- 2 Linear Classifiers and Surrogate Losses
- 3 A Detour of Numerical Optimization Methods
- 4 Perceptron
- 5 Logistic Regression**
 - A probabilistic view
 - Algorithms

A simple view

In one sentence: find the minimizer of

$$\begin{aligned} F(\mathbf{w}) &= \frac{1}{N} \sum_{n=1}^N \ell_{\text{logistic}}(y_n \mathbf{w}^T \mathbf{x}_n) \\ &= \frac{1}{N} \sum_{n=1}^N \ln(1 + e^{-y_n \mathbf{w}^T \mathbf{x}_n}) \end{aligned}$$

Before optimizing it: *why logistic loss? and why "regression"?*

Predicting probability

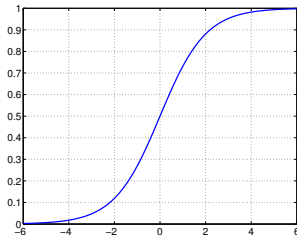
Instead of predicting a discrete label, can we *predict the probability of each label?* i.e. regress the probabilities

One way: **sigmoid function + linear model**

$$\mathbb{P}(y = +1 \mid \mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^T \mathbf{x})$$

where σ is the sigmoid function:

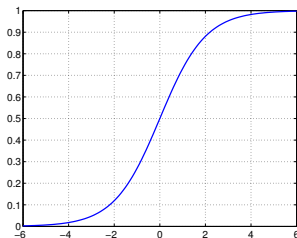
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Properties

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

- between 0 and 1 (good as probability)
- $\sigma(\mathbf{w}^T \mathbf{x}) \geq 0.5 \Leftrightarrow \mathbf{w}^T \mathbf{x} \geq 0$, consistent with predicting the label with $\text{sgn}(\mathbf{w}^T \mathbf{x})$
- larger $\mathbf{w}^T \mathbf{x} \Rightarrow$ larger $\sigma(\mathbf{w}^T \mathbf{x}) \Rightarrow$ higher *confidence* in label 1
- $\sigma(z) + \sigma(-z) = 1$ for all z



The probability of label -1 is naturally

$$1 - \mathbb{P}(y = +1 \mid \mathbf{x}; \mathbf{w}) = 1 - \sigma(\mathbf{w}^T \mathbf{x}) = \sigma(-\mathbf{w}^T \mathbf{x})$$

and thus

$$\mathbb{P}(y \mid \mathbf{x}; \mathbf{w}) = \sigma(y\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + e^{-y\mathbf{w}^T \mathbf{x}}}$$

How to regress with discrete labels?

What we observe are labels, not probabilities.

Take a **probabilistic view**

- assume data is independently generated in this way by some w
- perform Maximum Likelihood Estimation (MLE)

Specifically, what is the probability of seeing label $y_1, \dots, y_n$ given $x_1, \dots, x_n$, as a function of some w ?

$$P(w) = \prod_{n=1}^N \mathbb{P}(y_n \mid x_n; w)$$

MLE: find w^* that **maximizes the probability** $P(w)$

The MLE solution

$$\begin{aligned}\mathbf{w}^* &= \operatorname{argmax}_{\mathbf{w}} P(\mathbf{w}) = \operatorname{argmax}_{\mathbf{w}} \prod_{n=1}^N \mathbb{P}(y_n \mid \mathbf{x}_n; \mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{w}} \sum_{n=1}^N \ln \mathbb{P}(y_n \mid \mathbf{x}_n; \mathbf{w}) = \operatorname{argmin}_{\mathbf{w}} \sum_{n=1}^N -\ln \mathbb{P}(y_n \mid \mathbf{x}_n; \mathbf{w}) \\ &= \operatorname{argmin}_{\mathbf{w}} \sum_{n=1}^N \ln(1 + e^{-y_n \mathbf{w}^\top \mathbf{x}_n}) = \operatorname{argmin}_{\mathbf{w}} \sum_{n=1}^N \ell_{\text{logistic}}(y_n \mathbf{w}^\top \mathbf{x}_n) \\ &= \operatorname{argmin}_{\mathbf{w}} F(\mathbf{w})\end{aligned}$$

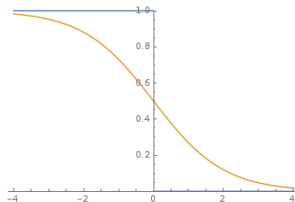
i.e. *minimizing logistic loss is exactly doing MLE for the sigmoid model!*

Back to algorithms: apply SGD again

$$\begin{aligned}
 \mathbf{w} &\leftarrow \mathbf{w} - \eta \tilde{\nabla} F(\mathbf{w}) \\
 &= \mathbf{w} - \eta \nabla_{\mathbf{w}} \ell_{\text{logistic}}(y_n \mathbf{w}^T \mathbf{x}_n) \quad (n \in [N] \text{ is drawn u.a.r.}) \\
 &= \mathbf{w} - \eta \left(\frac{\partial \ell_{\text{logistic}}(z)}{\partial z} \Big|_{z=y_n \mathbf{w}^T \mathbf{x}_n} \right) y_n \mathbf{x}_n \\
 &= \mathbf{w} - \eta \left(\frac{-e^{-z}}{1 + e^{-z}} \Big|_{z=y_n \mathbf{w}^T \mathbf{x}_n} \right) y_n \mathbf{x}_n \\
 &= \mathbf{w} + \eta \sigma(-y_n \mathbf{w}^T \mathbf{x}_n) y_n \mathbf{x}_n \\
 &= \mathbf{w} + \eta \mathbb{P}(-y_n \mid \mathbf{x}_n; \mathbf{w}) y_n \mathbf{x}_n
 \end{aligned}$$

This is a *soft version of Perceptron!*

$\mathbb{P}(-y_n \mid \mathbf{x}_n; \mathbf{w})$ versus $\mathbb{I}[y_n \neq \text{sgn}(\mathbf{w}^T \mathbf{x}_n)]$



Applying Newton to logistic loss

$$\nabla_{\mathbf{w}} \ell_{\text{logistic}}(y_n \mathbf{w}^T \mathbf{x}_n) = -\sigma(-y_n \mathbf{w}^T \mathbf{x}_n) y_n \mathbf{x}_n$$

$$\begin{aligned} \nabla_{\mathbf{w}}^2 \ell_{\text{logistic}}(y_n \mathbf{w}^T \mathbf{x}_n) &= \left(\frac{\partial \sigma(z)}{\partial z} \Big|_{z=-y_n \mathbf{w}^T \mathbf{x}_n} \right) y_n^2 \mathbf{x}_n \mathbf{x}_n^T \\ &= \left(\frac{e^{-z}}{(1 + e^{-z})^2} \Big|_{z=-y_n \mathbf{w}^T \mathbf{x}_n} \right) \mathbf{x}_n \mathbf{x}_n^T \\ &= \sigma(y_n \mathbf{w}^T \mathbf{x}_n) (1 - \sigma(y_n \mathbf{w}^T \mathbf{x}_n)) \mathbf{x}_n \mathbf{x}_n^T \end{aligned}$$

Exercises:

- why is the Hessian of logistic loss positive semidefinite?
- can we apply Newton method to perceptron/hinge loss?

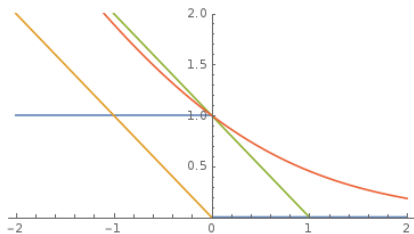
Summary

Linear models for classification:

Step 1. Model is the set of **separating hyperplanes**

$$\mathcal{F} = \{f(\mathbf{x}) = \text{sgn}(\mathbf{w}^T \mathbf{x}) \mid \mathbf{w} \in \mathbb{R}^D\}$$

Step 2. Pick the **surrogate loss**



- **perceptron loss** $\ell_{\text{perceptron}}(z) = \max\{0, -z\}$ (used in Perceptron)
- **hinge loss** $\ell_{\text{hinge}}(z) = \max\{0, 1 - z\}$ (used in SVM and many others)
- **logistic loss** $\ell_{\text{logistic}}(z) = \log(1 + \exp(-z))$ (used in logistic regression)

Step 3. Find empirical risk minimizer (ERM):

$$\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w} \in \mathbb{R}^D} \frac{1}{N} \sum_{n=1}^N \ell(y_n \mathbf{w}^T \mathbf{x}_n)$$

using

- **GD:** $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla F(\mathbf{w})$
- **SGD:** $\mathbf{w} \leftarrow \mathbf{w} - \eta \tilde{\nabla} F(\mathbf{w})$ $(\mathbb{E}[\tilde{\nabla} F(\mathbf{w})] = \nabla F(\mathbf{w}))$
- **Newton:** $\mathbf{w} \leftarrow \mathbf{w} - (\nabla^2 F(\mathbf{w}))^{-1} \nabla F(\mathbf{w})$