CSCI567 Machine Learning (Fall 2018)

Prof. Haipeng Luo

U of Southern California

Sep 5, 2018

Administration

- HW 1 has been released.
- *Complete the GitHub survey ASAP* if you haven't.
- Follow Piazza for clarifications/typos of HW 1.
- DO NOT post your programming assignment outputs on Piazza.
- DEN problems should have been resolved.

Outline

1. Review of Last Lecture
2. Linear Classifier and Surrogate Losses
3. Perceptron
4. Logistic regression

Outline

1. Review of Last Lecture
2. Linear Classifier and Surrogate Losses
3. Perceptron
4. 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: $f(\boldsymbol{w}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$

Least square solution

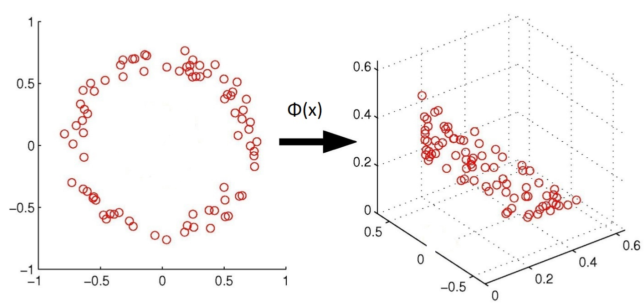
$$\begin{aligned}\boldsymbol{w}^* &= \operatorname*{argmin}_{\boldsymbol{w}} \mathrm{RSS}(\boldsymbol{w}) \\ &= \operatorname*{argmin}_{\boldsymbol{w}} \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 \\ &= \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}\end{aligned}$$

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1^{\mathrm{T}} \\ \boldsymbol{x}_2^{\mathrm{T}} \\ \vdots \\ \boldsymbol{x}_{\mathrm{N}}^{\mathrm{T}} \end{pmatrix}, \quad \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{\mathrm{N}} \end{pmatrix}$$

Two approaches to find the minimum:

- find **stationary points** by setting gradient $= \mathbf{0}$
- "**complete the square**"

Regression with nonlinear basis



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

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

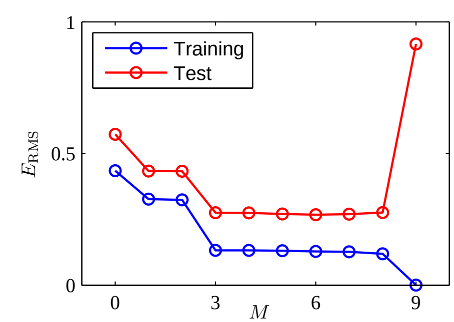
Underfitting and Overfitting

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

- large training error
- large test error

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

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



How to prevent overfitting? more data + regularization

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

General idea to derive ML algorithms

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

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

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

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

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

or **regularized empirical risk minimizer**:

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

ML becomes optimization

Outline

Classification

Recall the setup:

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

This lecture: **binary classification**

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

We have discussed **nearest neighbor classifier**:

- require carrying the training set
- more like a heuristic

Deriving classification algorithms

Let's follow the steps:

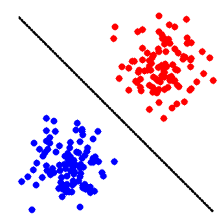
Step 1. Pick a set of models $\mathcal{F}$.

Again try linear models, but how to predict a label using $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$?

Sign of $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$ predicts the label:

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

(Sometimes use sgn for sign too.)



The models

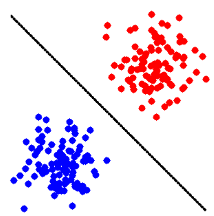
The set of **(separating) hyperplanes**:

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

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

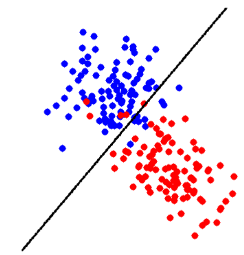
$$\mathrm{sgn}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x_n}) = y_n \quad \text{or} \quad y_n\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x_n} > 0$$

for all $n \in [N]$.



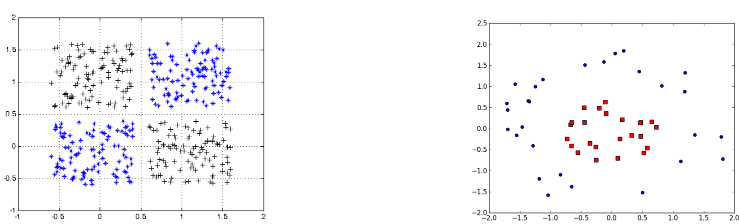
The models

Still makes sense for "almost" linearly separable data



The models

For clearly not linearly separable data,



Again can apply a **nonlinear mapping** $\boldsymbol{\Phi}$:

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

More discussions in the next two lectures.

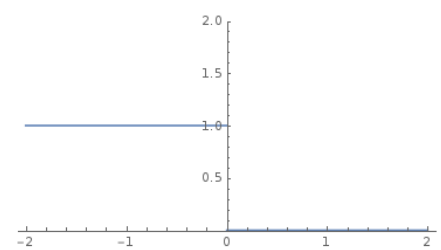
0-1 Loss

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 $y\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$**. That is, with

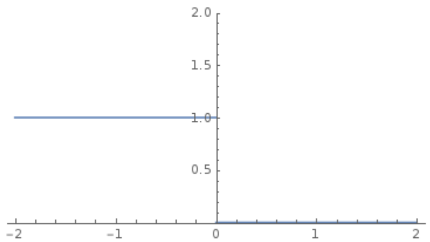
$$\ell_{\text{0-1}}(z) = \mathbb{I}[z \leq 0]$$



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

Minimizing 0-1 loss is hard

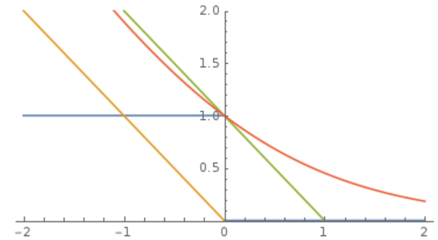
However, 0-1 loss is *not convex*.



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)
- 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; the base of $\log$ doesn't matter)

ML becomes convex optimization

Step 3. Find ERM:

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

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 $\boldsymbol{w} = \boldsymbol{0}$), but the algorithm derived from this perspective does.

Outline

The Perceptron Algorithm

In one sentence: Stochastic Gradient Descent applied to perceptron loss

i.e. find the minimizer of

$$F(\boldsymbol{w}) = \sum_{n=1}^{N} \ell_{\text{perceptron}}(y_n \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)$$
$$= \sum_{n=1}^{N} \max\{0, -y_n \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n\}$$

using SGD

A detour of numerical optimization methods

We describe 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)

Goal: minimize $F(\boldsymbol{w})$

Algorithm: move a bit in the *negative gradient direction*

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

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 just try several small values

An example

Example: $F(\boldsymbol{w}) = 0.5(w_1^2 - w_2)^2 + 0.5(w_1 - 1)^2$. Gradient is

$$\frac{\partial F}{\partial w_1} = 2(w_1^2 - w_2)w_1 + w_1 - 1 \qquad \frac{\partial F}{\partial w_2} = -(w_1^2 - w_2)$$

GD:

- Initialize $w_1^{(0)}$ and $w_2^{(0)}$ (to be 0 or *randomly*), $t = 0$
- do

$$w_1^{(t+1)} \leftarrow w_1^{(t)} - \eta \left[2({w_1^{(t)}}^2 - w_2^{(t)})w_1^{(t)} + w_1^{(t)} - 1 \right]$$
$$w_2^{(t+1)} \leftarrow w_2^{(t)} - \eta \left[-({w_1^{(t)}}^2 - w_2^{(t)}) \right]$$
$$t \leftarrow t + 1$$

- until $F(\boldsymbol{w}^{(t)})$ **does not change much**

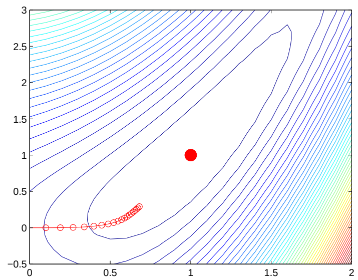
Why GD?

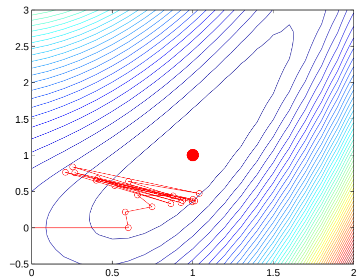
Intuition: by first-order **Taylor approximation**

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

GD ensures

$$F(\boldsymbol{w}^{(t+1)}) \approx F(\boldsymbol{w}^{(t)}) - \eta\|\nabla F(\boldsymbol{w}^{(t)})\|_2^2 \leq F(\boldsymbol{w}^{(t)})$$





reasonable η decreases function value   but large η is unstable

Stochastic Gradient Descent (SGD)

GD: move a bit in the negative gradient direction

SGD: move a bit in a *noisy* negative gradient direction

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

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

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

Key point: it could be *much faster to obtain a stochastic gradient*!

Convergence Guarantees

Many for both GD and SGD on convex objectives.

They tell you at most how many iterations you need to achieve

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

Even for *nonconvex objectives*, many recent works show effectiveness of GD/SGD.

Applying GD to perceptron loss

Objective

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

Gradient (or really *sub-gradient*) is

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

(only misclassified examples contribute to the gradient)

GD update

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

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

$$\tilde{\nabla} F(\boldsymbol{w}^{(t)}) = -N\mathbb{I}[y_n \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n \le 0] y_n \boldsymbol{x}_n$$

clearly unbiased.

SGD update (with η absorbing the constant N)

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta\mathbb{I}[y_n \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n \le 0] y_n \boldsymbol{x}_n$$

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!

Exercise: try SGD to minimize RSS for linear regression.

The Perceptron Algorithm

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

Repeat:

- Pick a data point $\boldsymbol{x}_n$ uniformly at random
- If sgn$(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) \ne y_n$

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

Note:

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

Why does it make sense?

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

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

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

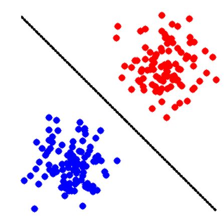
$$y_n \boldsymbol{w}'^{\mathrm{T}} \boldsymbol{x}_n = y_n \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n + y_n^2 \boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{x}_n \ge y_n \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n$$

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

Any theory?

(HW 1) 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 is not linearly separable.

Outline

A simple view

In one sentence: find the minimizer of

$$F(\boldsymbol{w}) = \sum_{n=1}^{N} \ell_{\text{logistic}}(y_n \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) = \sum_{n=1}^{N} \ln(1 + e^{-y_n \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n})$$

But why logistic loss? and why "regression"?

Predicting probability

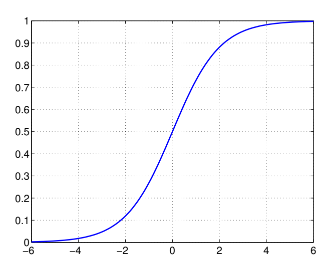
Instead of predicting a discrete label, can we *predict the probability of each label?* i.e. regress the probabilities

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

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

where σ is the sigmoid function:

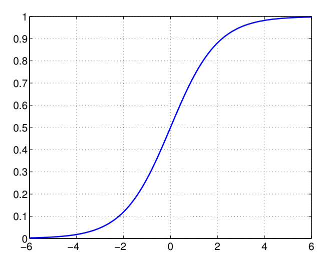
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Properties

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

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



The probability of label -1 is naturally

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

and thus

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

How to regress with discrete labels?

What we observe are labels, not probabilities.

Take a **probabilistic view**

- assume data is generated in this way by some $\boldsymbol{w}$
- perform Maximum Likelihood Estimation (MLE)

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

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

MLE: find $\boldsymbol{w}^*$ that **maximizes the probability** $P(\boldsymbol{w})$

The MLE solution

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

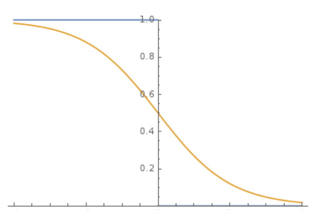
i.e. *minimizing logistic loss is exactly doing MLE for the sigmoid model!*

Let's apply SGD again

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

This is a *soft version of Perceptron!*

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



A second-order method: Newton method

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

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

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

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

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

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

(think "second derivative" when $D = 1$)

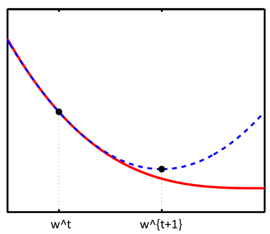
Deriving Newton method

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

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

for convex F (so H_t is *positive semidefinite*)
we obtain **Newton method**:

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



Comparing GD and Newton

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

Both are iterative optimization procedures, but Newton method

- has no learning rate η (*so no tuning needed!*)
- converges *super fast* in terms of #iterations needed
  - e.g. how many iterations needed when applied to a quadratic?
- requires **second-order** information and is *slow* each iteration (there are many ways to improve it though)

Applying Newton to logistic loss

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

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

Exercises:

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

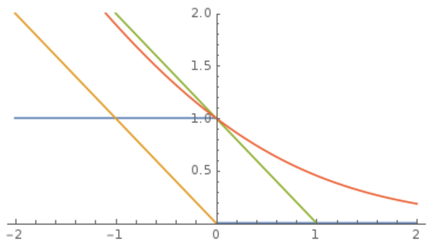
Summary

Linear models for classification:

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

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

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):

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

using **GD/SGD/Newton**.