

Probability Basics and Linear Classification

Machine Learning – Basic Methods

Prof. Gerhard Neumann

Autonomous Learning Robots

KIT, Institut für Anthropomatik und Robotik

Learning Outcomes

- Understand probabilistic models and maximum likelihood
- Understand the classification problem
- What is a linear classifier?
- What is the loss function of linear classification?
- What is gradient descent ?

Today's Agenda!

Basics: Probability Theory

- Probabilistic Models
- Expectations and Monte Carlo Methods
- Maximum Likelihood

Basics: Gradient Descent

Classification:

- Generative vs. discriminative classification
- Linear Classification
- Logistic Regression

Many slides are based on slides from Shenlon Wang, Yingyu Jiang, Michail Michailidis and Patrick Maiden

Basics: Probability Theory

- *“Probability theory is nothing but common sense reduced to calculation”, Pierre Laplace, 1812*
- We will keep our discussion relatively informal and pick the things we need from probability theory

Notation

- A **random variable** X represents uncertain states or outcomes of the world
- We will write $p(x)$ to mean the probability that X takes the value x
- The sample space is the space of all possible outcomes
 - Might be discrete, continuous or mixed
- $p(x)$ is the **probability mass** (density) function
 - Assigns a number to each point of the sample space
 - Non-negative, sums (integrates) to 1
 - Intuitively: How often does x occur? How much do we believe in x ?

Distributions

- **Joint distribution**

$$p(x, y)$$

- Probability that $X=x$ and $Y=y$

- **Conditional distribution**

$$p(x|y)$$

- Probability that $X=x$ given $Y=y$

Conditional Distributions

$P(W T)$	$P(W T = hot)$	
	W	P
	sun	0.8
	rain	0.2
	$P(W T = cold)$	
	W	P
	sun	0.4
	rain	0.6

Joint Distribution

$P(T, W)$		
T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

Rules of Probability

- **Sum rule (marginalization / integrating out):**

$$p(x) = \sum_y p(x, y)$$

$$p(x_1) = \sum_{x_2} \sum_{x_3} \cdots \sum_{x_D} p(x_1, \dots, x_D)$$

- **Note:** For continuous distributions, the sums will be replaced by integrals

$P(T, W)$			$P(T)$	
T	W	P		
hot	sun	0.4	hot	0.5
hot	rain	0.1	cold	0.5
cold	sun	0.2	$P(W)$	
cold	rain	0.3	sun	0.6
			rain	0.4

$P(t) = \sum_w P(t, w)$

$P(w) = \sum_t P(t, w)$

Rules of Probability

- Chain / product rule

$$p(x, y) = p(x|y)p(y)$$

$$p(x_1, \dots, x_D) = p(x_1)p(x_2|x_1) \dots p(x_D|x_1, \dots, x_{D-1})$$

$P(W)$		$P(D W)$			$P(D, W)$		
W	P	D	W	P	D	W	P
sun	0.8	wet	sun	0.1	wet	sun	0.08
rain	0.2	dry	sun	0.9	dry	sun	0.72
		wet	rain	0.7	wet	rain	0.14
		dry	rain	0.3	dry	rain	0.06

Bayes Rule

Bayes rule is one of the most important equations in probability theory and in machine learning

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} = \frac{p(y|x)p(x)}{\sum_{x'} p(y|x')p(x')}$$

- Way of “reversing” the conditional probabilities
- Often one conditional is tricky but the other one is simple
- One of the **most important equations** for ML!



Expectations

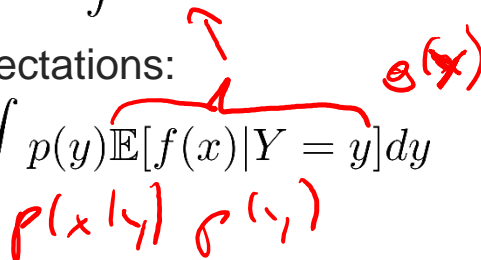
The expectation of a function $f(x)$ with respect to a distribution $p(x)$ is given by

$$\mathbb{E}_p[f(x)] = \int p(x)f(x)dx$$

A conditional expectation is given by

$$\mathbb{E}_p[f(x)|Y = y] = \int p(x|y)f(x)dx$$

Chain rule for expectations:

$$\mathbb{E}_p[f(x)] = \int p(y) \mathbb{E}[f(x)|Y = y] dy$$


Monte-carlo estimation

Expectations can always be **approximated by samples**:

$$\mathbb{E}_p[f(x)] = \int p(x)f(x)dx \approx \frac{1}{N} \sum_{x_i \sim p(x)} f(x_i)$$

- Necessary if no analytical solution exists to compute the integral (typical case)

Moments

Moments are expectations:

- 1st moment (mean): $\boldsymbol{\mu} = \mathbb{E}_p[\boldsymbol{x}]$
- 2nd moment: $\boldsymbol{M}_2 = \mathbb{E}_p[\boldsymbol{x}\boldsymbol{x}^T]$

Central moments are always computed relatively to the mean:

- 2nd central moment (covariance):
$$\boldsymbol{\Sigma} = \mathbb{E}_p[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T]$$
- Captures variability (diagonal entries) and correlation (off-diagonal)

Distributions

Bernoulli Distribution:

- Binary random variable $X \in \{0, 1\}$
- One parameter $p(X = 1) = \mu$
- Probability distribution $p(x) = \mu^x (1 - \mu)^{(1-x)}$
- Think of it as tossing a coin

Depending on x , selects either μ or $1-\mu$ as probability

Distributions

Multinomial / Categorical Distribution:

- K different events: $C \in \{1, \dots, K\}$
- Directly specifies probabilities: $p(C = k) = \mu_k, \quad \mu_k \geq 0, \quad \sum_{k=1}^K \mu_k = 1$
- Or written with 1-hot-encoding (without an “if” clause)

$$p(c) = \prod_{k=1}^K \mu_k^{h_{c,k}}$$

$h_c = [1, 0, 0, \dots, 0]$, $h_c = [0, 0, 1, 0, \dots]$
Depending on the class label of x, selects the correct μ_k

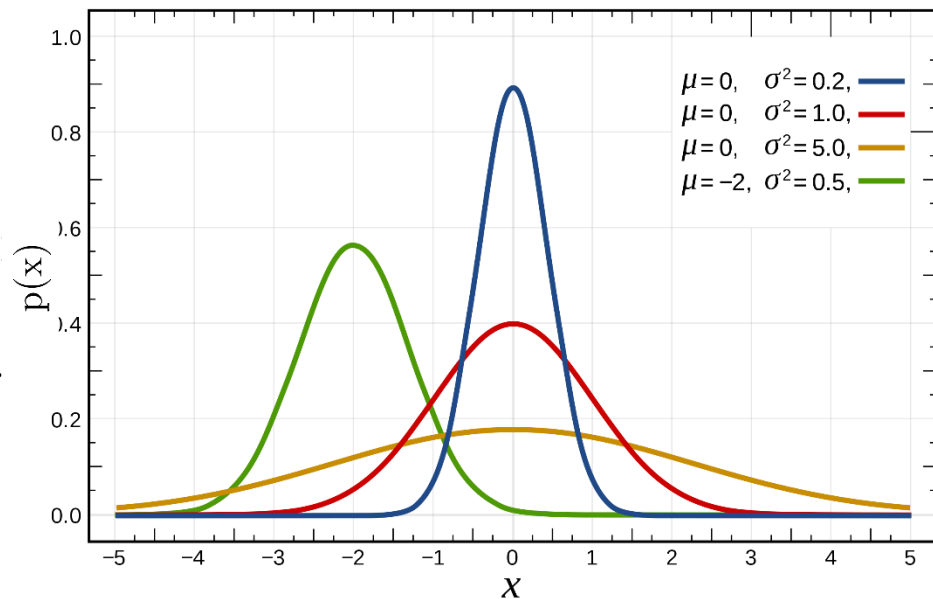
- where h_x is the K-dimensional 1-hot encoding vector, which is one for the dimension $c = k$ and 0 elsewhere. $h_{x,k}$ is the k-th element of this vector.
- Think of it as tossing a die

Distributions

Gaussian Distribution

- Continuous RV: $X \in \mathbb{R}$
- Distribution is completely specified by mean μ and variance σ^2

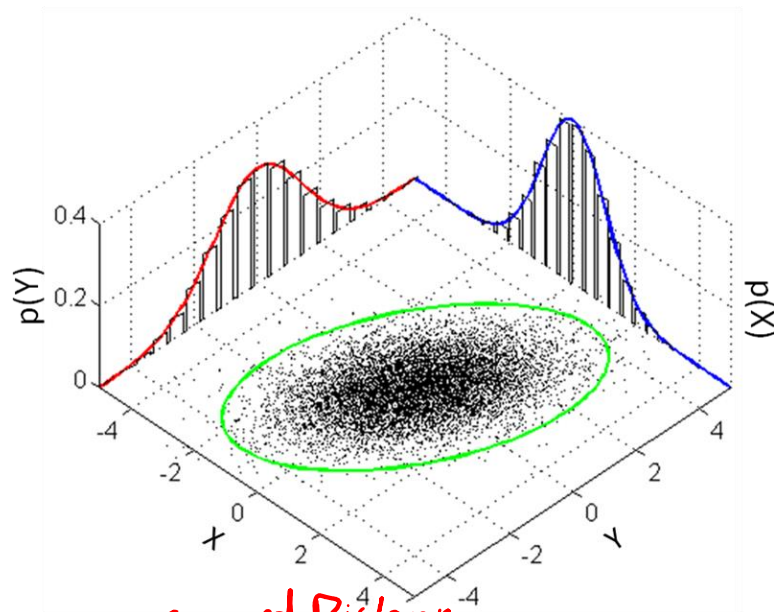
$$p(x) = \mathcal{N}(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left\{ -\frac{(x - \mu)^2}{2\sigma^2} \right\}$$



Distributions

Multivariate Gaussian Distribution

- Continuous RV: $X \in \mathbb{R}^d$
- Distribution is completely specified by mean vector μ and covariance matrix Σ



$$p(x) = \mathcal{N}(x|\mu, \Sigma) = \frac{1}{\sqrt{|2\pi\Sigma|}} \exp \left\{ -\frac{((x - \mu)^T \Sigma^{-1} (x - \mu))}{2} \right\}$$

Squared Distance

Normalization

Distributions

Important Properties of Gaussians:

- All marginals of a Gaussian are again Gaussian
- Every conditional is Gaussian
- The product of 2 Gaussians is again Gaussian
- Even the sum of 2 Gaussian RVs is again Gaussian

Maximum Likelihood Estimation (MLE)

- Given: the training data $D = \{(x_i, y_i)\}_{i=1 \dots N}$ **identically independently distributed (iid)** from the data distribution p_{data}
- Let $p_{\theta}(x, y)$ be a family of distributions parametrized by $\theta \in \Theta$
- We want to find θ such that p fits the data well

Fitness of θ for one single data point:

$$\text{lik}(\theta; x_i, y_i) = p_{\theta}(x_i, y_i)$$

Fitness of θ for whole dataset (iid. assumption):

$$\text{lik}(\theta; D) = \prod_i p_{\theta}(x_i, y_i)$$

Maximum Likelihood Estimation (MLE)

Log-likelihood is easier to optimize:

$$\text{loglik}(\boldsymbol{\theta}; D) = \sum_i \log \underline{p_{\boldsymbol{\theta}}}(x_i, y_i)$$

- Log is monotonous -> same optimum
- Sums are “nicer” to optimize than products
- Log cancels exponential form (most distributions are in the exponential family)

The MLE solution is given by:

$$\boldsymbol{\theta}_{\text{ML}} = \text{argmax}_{\boldsymbol{\theta}} \text{loglik}(\boldsymbol{\theta}; D)$$

Example: Gaussian

Gaussian density function:

$$\text{loglik}(\boldsymbol{\theta}; D) = -\cancel{N \log \sqrt{2\pi\sigma^2}} - \sum_i \frac{(x_i - \mu)^2}{2\sigma^2}$$

MLE solution for μ :

$$\frac{\partial \text{loglik}(\mu; D)}{\partial \mu} = \cancel{-\sum_i \frac{2(x_i - \mu)}{2\sigma^2}} = 0$$
$$\sum_i x_i - N\mu = 0$$
$$\mu = \frac{\sum_i x_i}{N}$$

MLE: conditional log-likelihood

- Given the training data $D = \{(x_i, y_i)\}_{i=1 \dots N}$ iid. from the data distribution p_{data}
- Let $p_{\theta}(y|x)$ be a family of distributions parametrized by $\theta \in \Theta$
- We only care about distribution of y , not of x
- Typical case in supervised learning

Log-likelihood:

$$\text{loglik}(\theta; D) = \sum_i \log p_{\theta}(y_i|x_i)$$

Example: Linear Gaussian model

We consider the following conditional Gaussian model:

$$p_{\theta}(y|\mathbf{x}) = \mathcal{N}(y|\underline{\mathbf{w}^T \tilde{\mathbf{x}}}, \sigma^2), \quad \theta = \{\mathbf{w}, \sigma^2\}$$

Linear function of \mathbf{x}

Log-likelihood:

$$\text{loglik}(\theta; D) = -N \log \sqrt{2\pi\sigma^2} - \sum_i \frac{(y_i - \mathbf{w}^T \tilde{\mathbf{x}}_i)^2}{2\sigma^2}$$

- For obtaining \mathbf{w} , only the squared errors matter, i.e.

$$\text{loglik}(\theta; D) = \text{const}_1 - \text{const}_2 \sum_i (y_i - \mathbf{w}^T \tilde{\mathbf{x}}_i)^2$$

- Hence, the **MLE solution is equivalent to the least squares solution!**
- **But:** we can also obtain the variance!

Takeaway messages

What have we learned so far?

- Basic rules of probabilities ... nothing new so far
- Expectations can be evaluated by samples
- How to compute the ML estimator
- Maximum likelihood is equivalent to minimizing the squared loss for:
 - Conditional Gaussian models
 - With constant noise



Today's Agenda!

Basics: Probability Theory

- Probabilistic Models
- Expectations and Monte Carlo Methods
- Maximum Likelihood

Linear Classification:

- Linear Classifiers
- Logistic Regression

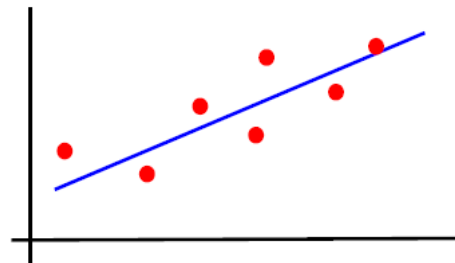
Basics: Gradient Descent

Supervised Learning

Training data includes targets

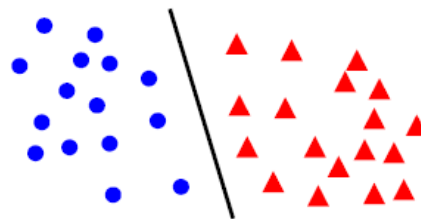
- **Regression:**

- Learn continuous function
- Example: line

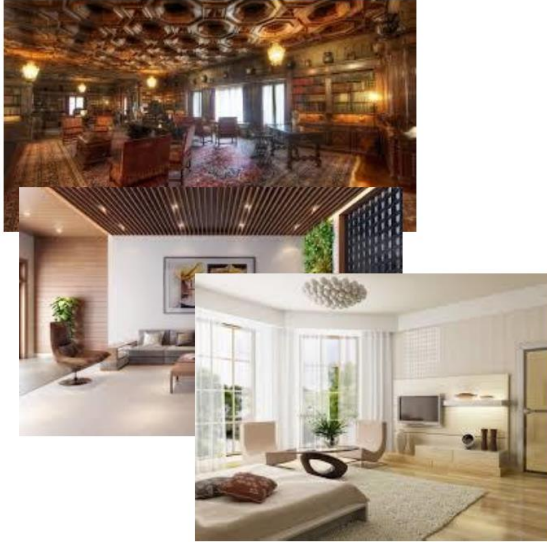


- **Classification:**

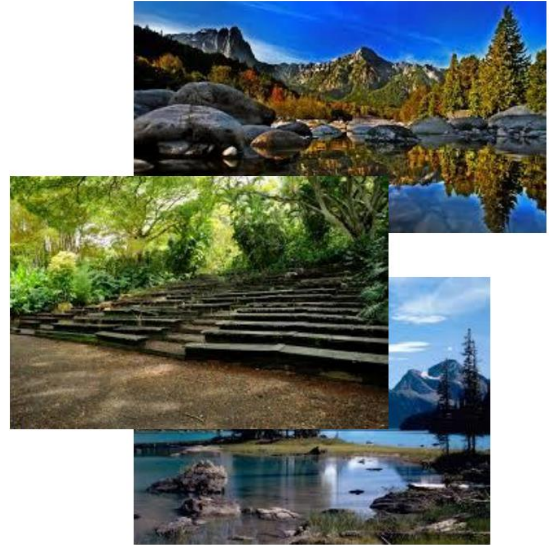
- Learn class labels
- Example: Digit recognition



Example 1: Image classification



Indoor



outdoor

Example 2: Spam Classification

	#"\$"	#"Mr."	#"sale"	...	Spam?
Email 1	2	1	1		Yes
Email 2	0	1	0		No
Email 3	1	1	1		Yes
...					
Email n	0	0	0		No
New email	0	0	1		??

Definition

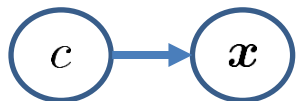
Given the dataset $\mathcal{D} = \{(\mathbf{x}_i, c_i)\}_{i=1\dots N}$, where $\mathbf{x}_i \in \mathbb{R}^d$ are the input samples and $c \in \{1 \dots K\}$ are the class labels, we want to learn a classifier $f(\mathbf{x})$ that predicts the class label for unseen samples.

- $K = 2$: Binary classification
- $K > 2$: Multi-class classification

In difference to regression, the **output is now discrete!**

Generative vs. discriminative modelling

Generative Models:

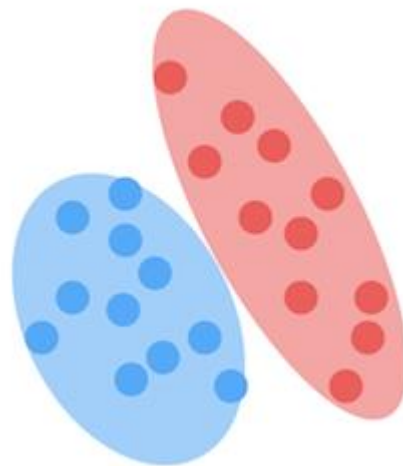


- Assume some functional form for class prior $p(c)$ and class densities $p(\mathbf{x}|c)$
- Learn prior and densities from data
 - This is a “generative” model, as we can create new datapoints \mathbf{x} using $p(\mathbf{x}|c)$
- Predict class label by **computing posterior** $p(c|\mathbf{x}) = \frac{p(\mathbf{x}|c)p(c)}{p(\mathbf{x})}$

Learn full **joint distribution** of the data (typically very **hard**)

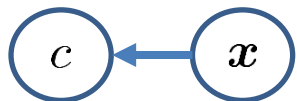
- Our modelling assumptions, e.g. that $p(\mathbf{x}|c)$ is Gaussian, might introduce big errors

Generative



Generative vs. discriminative modelling

Discriminative Models:



- Directly assume some functional form for $p(c|x)$ (or any other predictor $f(x)$ that returns the class label).
- This is a ‘discriminative’ model of the data!
- Estimate parameters of $p(c|x)$ directly from training data

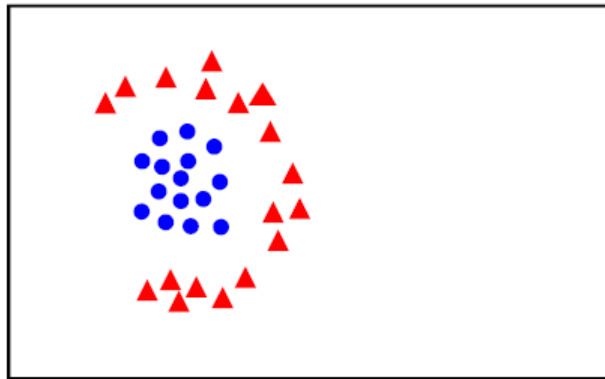
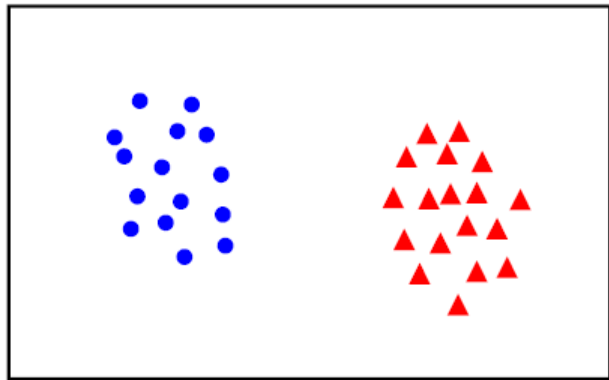
Modelling needs to **consider only points on the border**

- Typically much simpler than generative modelling
- We therefore concentrate on discriminative models

Discriminative



(Discriminative) Binary Classification



Given the training data (\mathbf{x}_i, y_i) , $i = 1 \dots N$, with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$, learn a classifier $f(\mathbf{x})$ such that:

$$f(\mathbf{x}_i) = \begin{cases} > 0, & \text{if } y_i = 1 \\ < 0, & \text{if } y_i = 0 \end{cases}$$

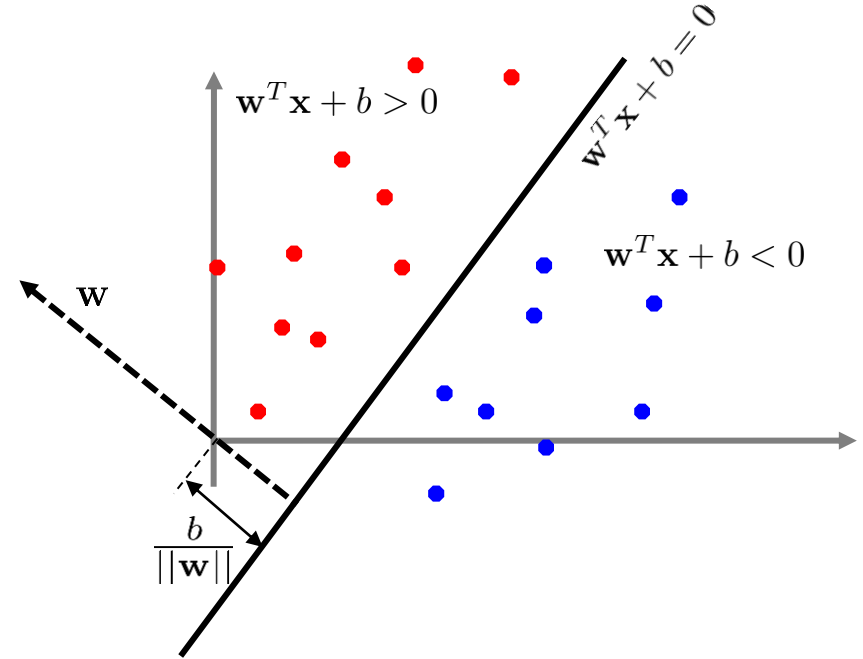
Linear Classifiers

A linear classifier is given in the form:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

In 2D, the classifier is a line

- \mathbf{w} is the normal to the line
- b is the bias



Linear Discriminators

A linear discriminator is given in the form:

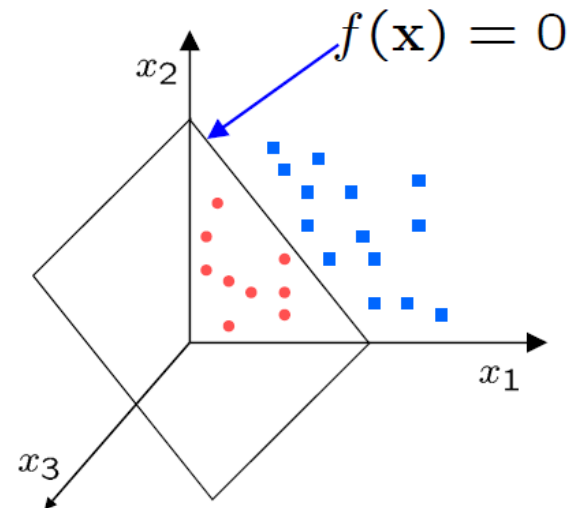
$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

In 2D, the discriminator is a **line**

- \mathbf{w} is the normal to the line
- b is the bias

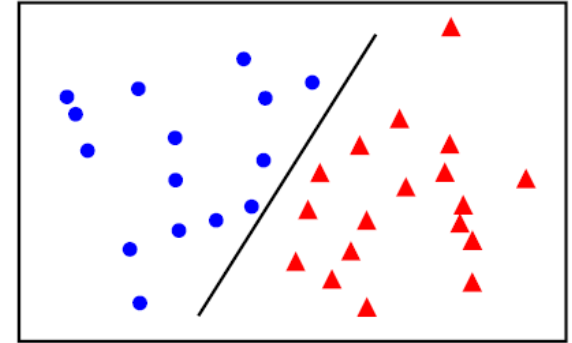
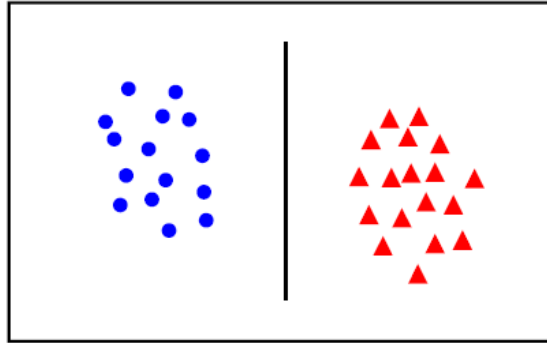
In 3D, it's a **plane**

In N-D, it's a **hyper-plane**

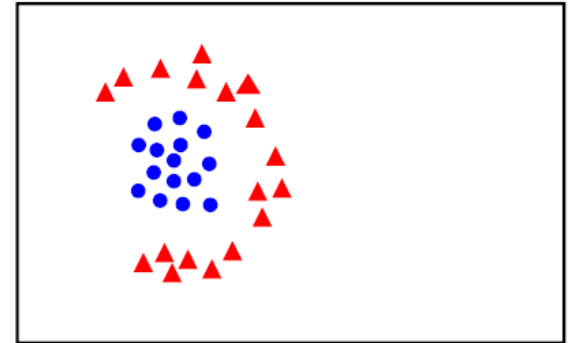
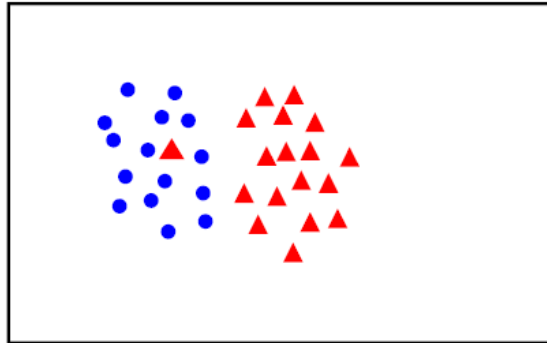


Linear Separability

Linear Separable



Non-Linear Separable



Linear Classification: 0-1 loss (1st attempt)

Prediction: $y = \text{step}(f(\mathbf{x})) = \text{step}(\mathbf{w}^T \mathbf{x} + b)$

- Predict class 1 for $f(\mathbf{x}) > 0$
- else predict class 0

Optimization: Find \mathbf{w} such that

$$L_0(\mathbf{w}) = \sum_i \mathbb{I}(\text{step}(\mathbf{w}^T \mathbf{x} + b) \neq y_i)$$

- where \mathbb{I} returns 1 if the argument is true
- ... counts the number of misclassifications

✗ Very **difficult to optimize!!!** (NP-hard)

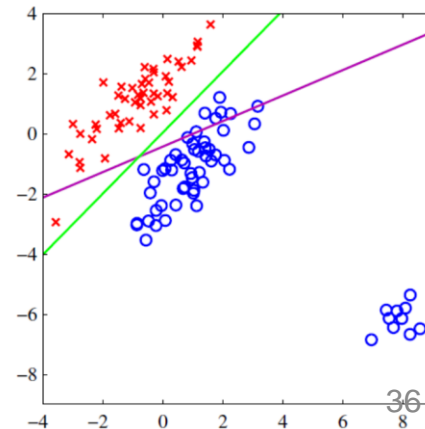
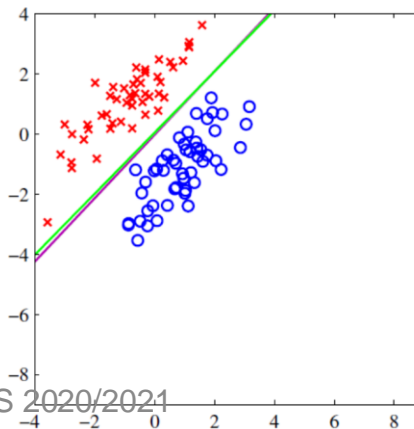
Linear Classification: regression loss (2nd attempt)

We can use same loss as in regression

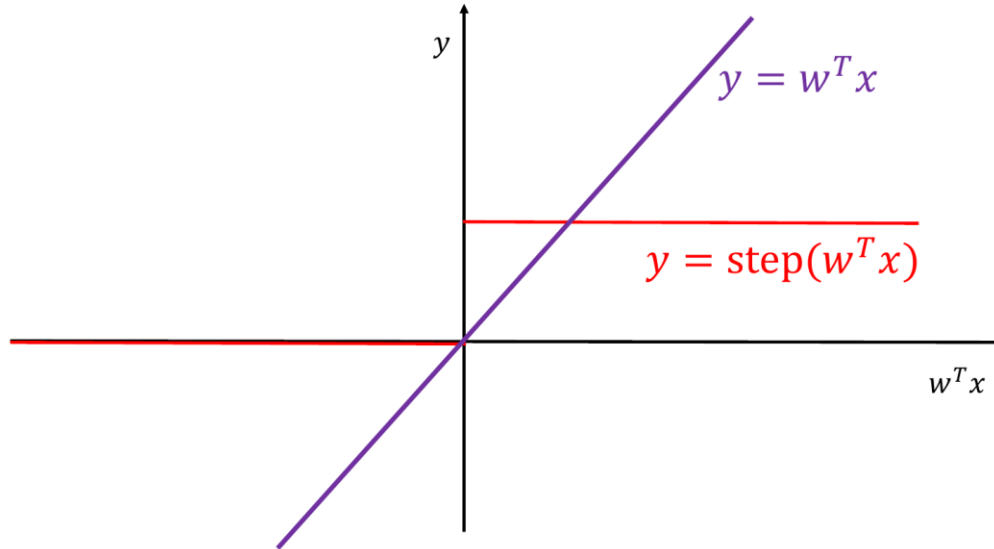
$$L_{\text{reg}}(\mathbf{w}) = \sum_i (f(\mathbf{x}_i) - y_i)^2$$

- Minimize the squared error: Easy!
- However: we ignored the fact that y_i is restricted to $\{0,1\}$

× **Not robust** to outliers



Compare the two



- The output of a linear function is unbounded!
- However, useful output values are only 0 or 1

Logistic sigmoid function

Sigmoid function:

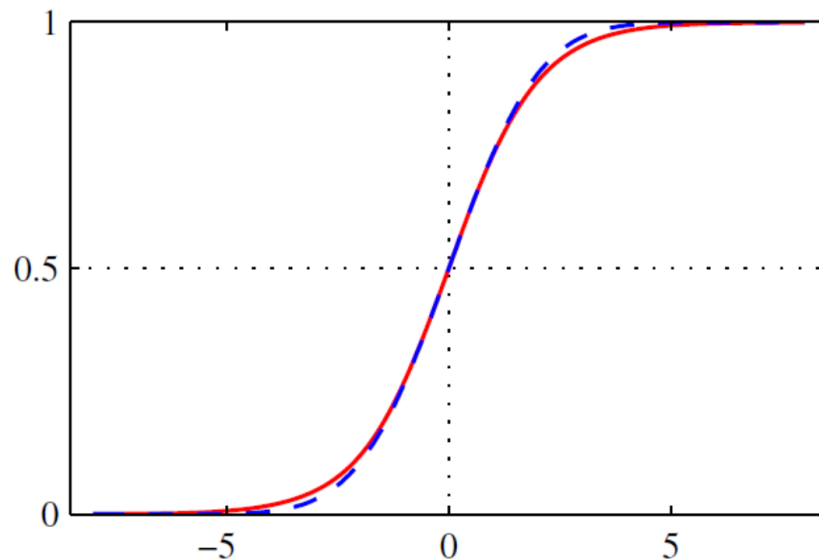
$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

- Output is bounded between 0 and 1
- Smooth

For linear classification:

- Squash the output of the linear function
- Minimize the loss

$$L(\mathbf{w}) = \sum_i (\sigma(f(\mathbf{x}_i)) - y_i)^2 = \sum_i (\sigma(\mathbf{w}^T \mathbf{x} + b) - y_i)^2$$



Better: Probabilistic View

Define conditional probability distribution of the class label

$$p(c = 1|\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + b), \quad p(c = 0|\mathbf{x}) = 1 - \sigma(\mathbf{w}^T \mathbf{x} + b)$$

- This is now a **conditional Bernoulli** distribution. I.e. the **outcome of the event c depends on \mathbf{x}**
- We can use the same “**exponential trick**” to select the correct probability depending on the value of c , i.e.

$$p(c|\mathbf{x}) = p(c = 1|\mathbf{x})^c p(c = 0|\mathbf{x})^{1-c} = \sigma(\mathbf{w}^T \mathbf{x} + b)^c (1 - \sigma(\mathbf{w}^T \mathbf{x} + b))^{1-c}$$

Log-Likelihood

We can now directly optimize the conditional Bernoulli log-likelihood

$$\begin{aligned}\text{loglik}(\tilde{\mathbf{w}}, D) &= \sum_i \log p(c_i | \mathbf{x}_i) = \sum_i \log (p(c = 1 | \mathbf{x}_i)^{c_i} p(c = 0 | \mathbf{x}_i)^{1-c_i}) \\ &= \sum_i c_i \log p(c = 1 | \mathbf{x}_i) + (1 - c_i) \log p(c = 0 | \mathbf{x}_i) \\ &= \sum_i c_i \log \sigma(\tilde{\mathbf{w}}^T \tilde{\mathbf{x}}_i) + (1 - c_i) \log (1 - \sigma(\tilde{\mathbf{w}}^T \tilde{\mathbf{x}}_i))\end{aligned}$$

- Negative likelihood is also often referred to as **cross-entropy loss**

Logistic Regression

Optimizing the log-likelihood of a sigmoid is called **logistic regression**

$$\operatorname{argmax}_{\tilde{\mathbf{w}}} \operatorname{loglik}(\tilde{\mathbf{w}}, D) = \operatorname{argmax}_{\tilde{\mathbf{w}}} \sum_i c_i \log \sigma(\tilde{\mathbf{w}}^T \tilde{\mathbf{x}}_i) + (1 - c_i) \log (1 - \sigma(\tilde{\mathbf{w}}^T \tilde{\mathbf{x}}_i))$$

- ... even though we solve a classification problem
- One can show that the function is still convex (only one maximum exists)
- However, there is **no closed form solution** as in linear regression

How can we find the maximum? -> **Gradient Descent!**

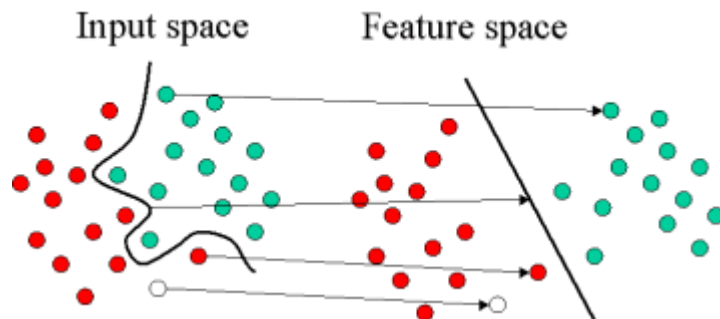
Generalized logistic models

We can fit a **linear discriminator** in a **non-linear feature space**

- Similar to generalized linear regression models

$$\operatorname{argmax}_{\mathbf{w}} \operatorname{loglik}(\mathbf{w}, D) = \operatorname{argmax}_{\mathbf{w}} \sum_i c_i \log \sigma(\mathbf{w}^T \phi(\mathbf{x}_i)) + (1 - c_i) \log (1 - \sigma(\mathbf{w}^T \phi(\mathbf{x}_i)))$$

- Problems that are not linear separable in input space can be linear separable in feature space



Regularization

Similar as in linear regression, we can again add a **regularization penalty**

$$L(\tilde{\mathbf{w}}, D) = \text{loglik}(\tilde{\mathbf{w}}, D) - \lambda \text{penalty}(\tilde{\mathbf{w}})$$

Most common: L2 regularization loss

$$\text{penalty}(\tilde{\mathbf{w}}) = ||\tilde{\mathbf{w}}||^2$$

- L is still convex for most penalty terms

Today's Agenda!

Basics: Probability Theory

- Probabilistic Models
- Expectations and Monte Carlo Methods
- Maximum Likelihood

Linear Classification:

- Linear Classifiers
- Logistic Regression

Basics: Gradient Descent

Optimization

For most ML algorithms, we want to find the best model to fit the data.

Two examples we already know:

- **Least squares solution:**

$$\operatorname{argmin}_{\boldsymbol{w}} \operatorname{SSE}(\boldsymbol{w}, D)$$

- **Maximum likelihood solution:**

$$\operatorname{argmax}_{\boldsymbol{w}} \operatorname{loglik}(\boldsymbol{w}, D)$$

Optimization

For most ML algorithms, we want to find the best model to fit the data.

Two examples we already know:

- **Least squares solution:**

$$\operatorname{argmin}_{\boldsymbol{w}} \operatorname{SSE}(\boldsymbol{w}, D) + \lambda \operatorname{penalty}(\boldsymbol{w})$$

- **Maximum likelihood solution:**

$$\operatorname{argmax}_{\boldsymbol{w}} \operatorname{loglik}(\boldsymbol{w}, D) - \lambda \operatorname{penalty}(\boldsymbol{w})$$

... plus regularization penalty

Note that:

$$\operatorname{argmin}_{\boldsymbol{x}} f(\boldsymbol{x}) = \operatorname{argmax}_{\boldsymbol{x}} -f(\boldsymbol{x})$$

Hence, the role of the penalty is the same

Optimization

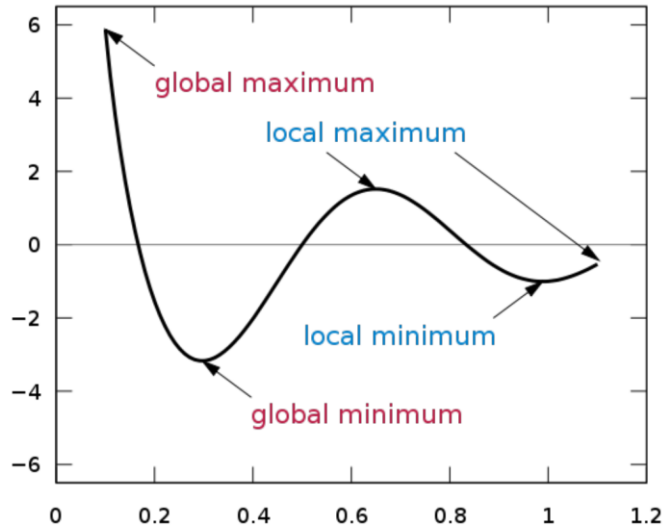
General form of optimization for ML: loss + penalty

$$\arg \min_{\text{parameters } \boldsymbol{\theta}} \sum_{i=1}^N l(\boldsymbol{x}_i, \boldsymbol{\theta}) + \lambda \text{ penalty}(\boldsymbol{\theta})$$

- Summed sample-loss plus regularization penalty

How to do that? **Optimization**

When can we do that?



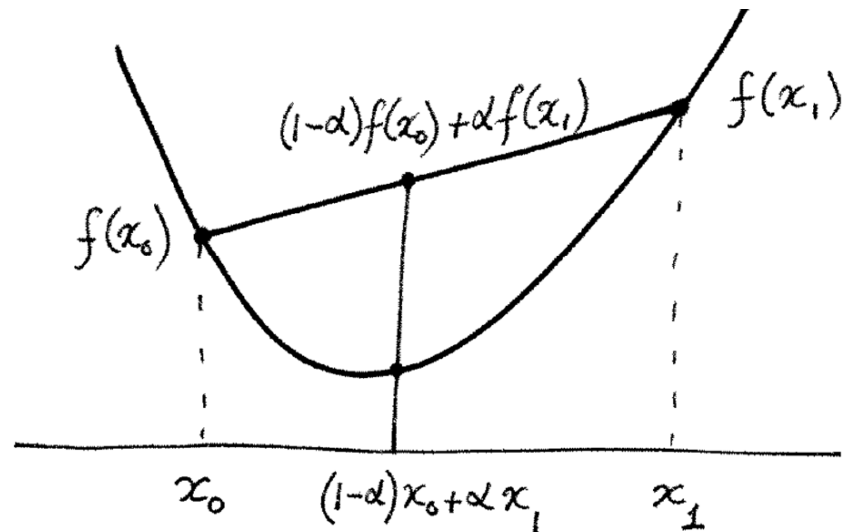
- The global minimum/maximum can only be found for convex functions!
- For non-convex functions we are limited to finding a local minimum / maximum

Convex functions

A convex function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ satisfies for any $\mathbf{x}_0, \mathbf{x}_1 \in \mathbb{R}^d$

$$f((1 - \alpha)\mathbf{x}_0 + \alpha\mathbf{x}_1) \leq (1 - \alpha)f(\mathbf{x}_0) + \alpha f(\mathbf{x}_1), \quad \alpha \in [0, 1]$$

- Line joining $(\mathbf{x}_0, f(\mathbf{x}_0))$ and $(\mathbf{x}_1, f(\mathbf{x}_1))$ is always above the function value
- There is only one minimum!



Example: Linear Regression Objective

$$L_{\text{ridge}} = (\mathbf{y} - \Phi \mathbf{w})^T (\mathbf{y} - \Phi \mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w}$$

- Convex
- Quadratic function in \mathbf{w}
- Minimum can be obtained analytically
- One of the very rare cases!

In most other cases, we have to resort to incremental methods: **Gradient descent**

Gradient Descent

- Is good for finding **global minima** if function is **convex**
- Is good for finding **local minima** if function is **non-convex**
- Has many applications in ML:
 - Logistic Regression
 - Linear Regression (for large input dimensions)
 - Neural Networks
 - Mixture Models
 - ...

Gradient Descent

Start at some point, follow the gradient towards (a) minimum

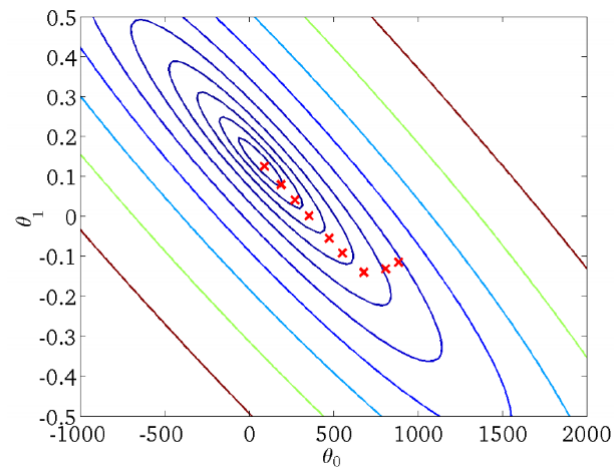
$\mathbf{x}_0 \leftarrow \text{init}, t = 0$

while termination condition does not hold **do**

$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \nabla f(\mathbf{x}_t), \quad t = t + 1$

end while

- $\eta \dots$ learning rate or step size
- Gradient always points in the direction of **steepest ascen.**



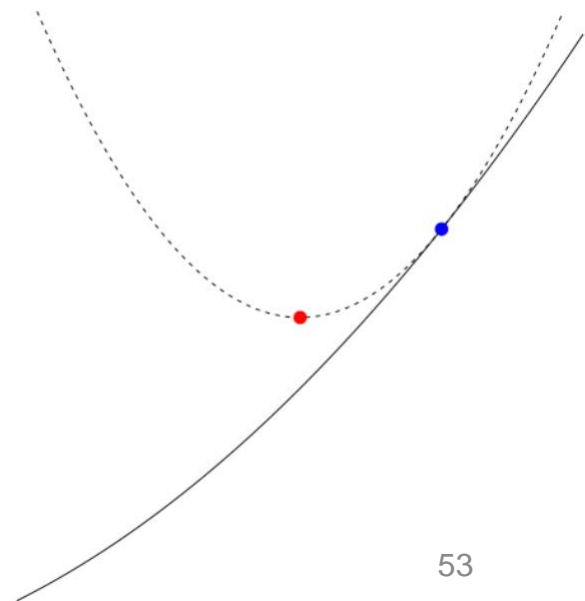
Gradient Descent Interpretation

Approximate the function as quadratic function:

$$\hat{f}(\mathbf{x}) = \underbrace{f(\mathbf{x}_t) + \nabla f(\mathbf{x}_t)^T (\mathbf{x} - \mathbf{x}_t)}_{\text{linear approximation}} + \underbrace{\frac{1}{2\eta} \|\mathbf{x}_t - \mathbf{x}\|^2}_{\text{proximity of } \mathbf{x}_t} \approx f(\mathbf{x})$$

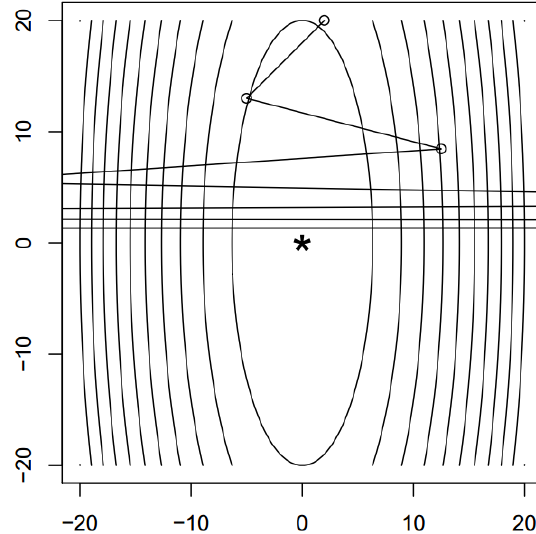
- Finding the minimum of $\hat{f}(\mathbf{x})$ yields the gradient descent rule

$$\mathbf{x}_{t+1} = \arg \min_{\mathbf{x}} \hat{f}(\mathbf{x}), \quad \mathbf{x}_{t+1} = \mathbf{x}_t - \eta \nabla f(\mathbf{x}_t)$$

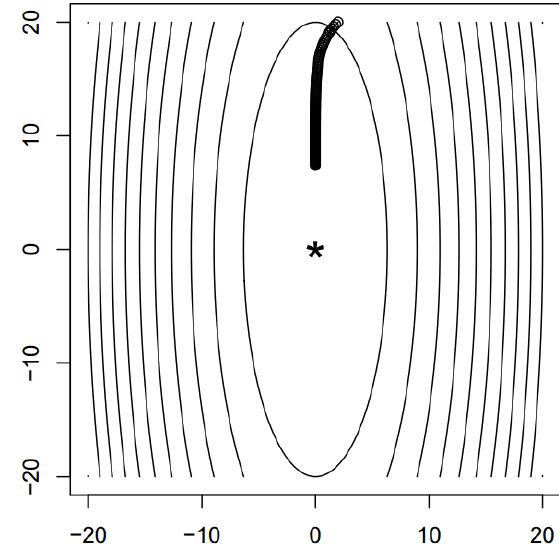


Choosing the step-size

$\eta_t = t$, it is too big



too small η_t , after 100 iterations



How to terminate

When change in iterates is small

- When gradient is small
- When change in function value is small

Or after a fixed time step or budget

Stochastic Gradient Descent

- Usually we are minimizing the empirical loss (**batch gradient descent**)

$$\frac{1}{n} \sum_i l(\mathbf{x}_i; \boldsymbol{\theta}) \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{n} \sum_i \nabla_{\boldsymbol{\theta}} l(\mathbf{x}_i; \boldsymbol{\theta}_t)$$

- We do this to approximate the expected loss

$$\mathbb{E}_{\mathbf{x}} [l(\mathbf{x}; \boldsymbol{\theta})] \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \mathbb{E}_{\mathbf{x}} [\nabla_{\boldsymbol{\theta}} l(\mathbf{x}; \boldsymbol{\theta}_t)]$$

- Use a rougher, cheaper approximation: **stochastic gradient descent**

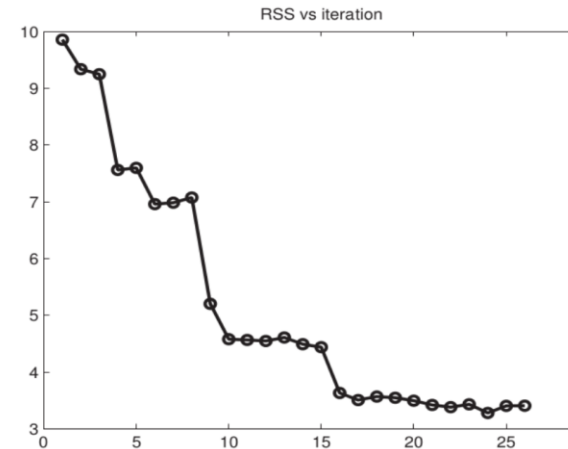
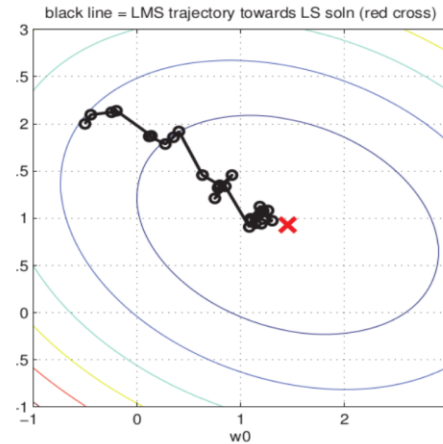
$$l(\mathbf{x}_i; \boldsymbol{\theta}) \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} l(\mathbf{x}_i; \boldsymbol{\theta}_t)$$

- for random sample i

Stochastic Gradient Descent (SGD)

Use only one sample to compute the update

- Does NOT always “descent”
- Iterations are much cheaper
- Requires more iterations
- ... and smaller step sizes

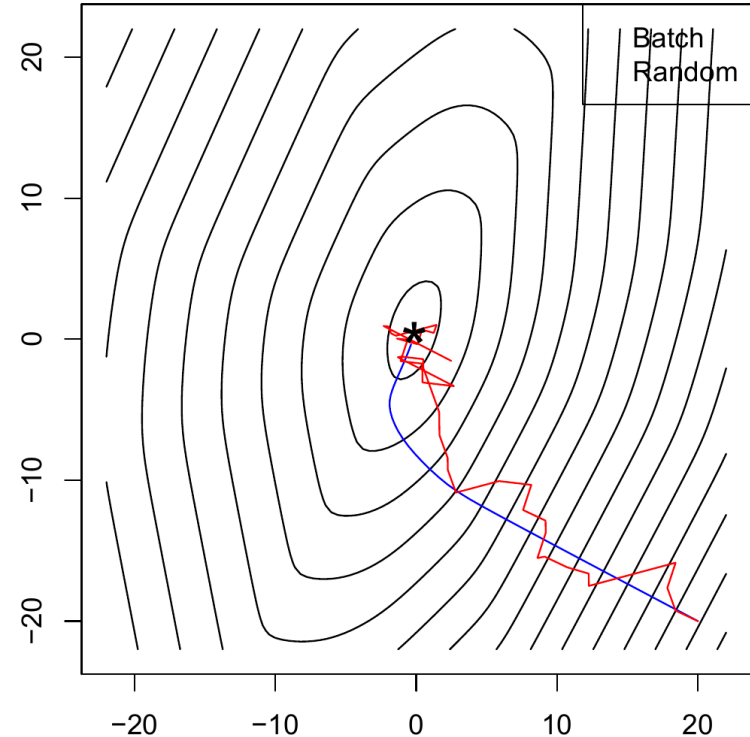


Stochastic vs. Batch Gradients

- **Blue:** Batch Gradients
- **Red:** Stochastic Gradients

Rule of thumb:

- Stochastic methods work well far away from optimum
- But struggle to find the exact optimum



Step-sizes

Standard in SGD is to use diminishing step sizes, e.g., $\eta_t = \frac{1}{t}$

- Asymptotically approach the optimum
- instead of “wiggling” around optimum

In general, it can be shown that SGD **converges to the optimum** for strictly convex functions if (**stochastic approximation theory**)

$$\sum_t \eta_t = \infty \quad \text{and} \quad \sum_t \eta_t^2 < \infty$$

Stochastic vs. Batch Gradients

Why are stochastic gradients often better than batch?

- Typically, our data-set will contain redundancy
- Hence, some computation in the batch gradients are redundant
 - compute the gradients for similar samples
 - using the same parameter vector
- This does not happen if we update immediately after one sample

As a consequence, **SGD requires less computation** (in most cases)

Mini-Batches

Take **subset of samples** $I_t \subset \{1, \dots, n\}$, $|I_t| = b$, $b \ll n$ to approximate real gradient:

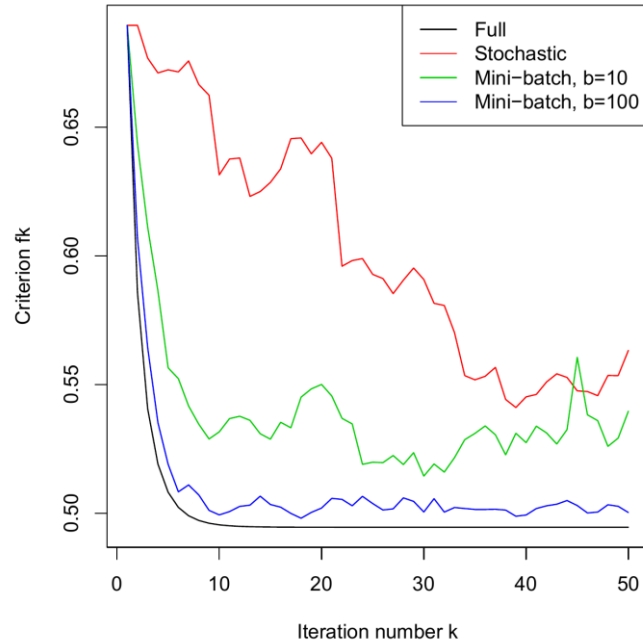
$$\frac{1}{b} \sum_{i \in I_t} l(\mathbf{x}_i; \boldsymbol{\theta})$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{b} \sum_{i \in I_t} \nabla_{\boldsymbol{\theta}} l(\mathbf{x}_i; \boldsymbol{\theta}_t)$$

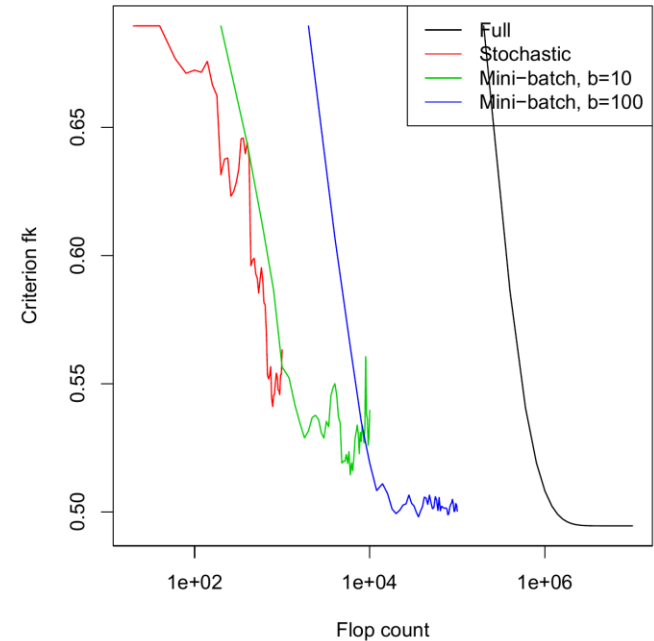
- Intermediate version of stochastic and batch gradient descent
- Less noisy estimates
- Achieves “descent” more often
- Preferable for GPU implementations

Example

10000 samples, loglikelihood logistic regression:



per iteration



per flop

Gradient Descent for Logistic Regression

Properties of the sigmoid function:

- Bounded: $\sigma(a) = \frac{1}{1 + \exp(-a)} \in (0, 1)$
- Symmetric: $1 - \sigma(a) = \frac{\exp(-a)}{1 + \exp(-a)} = \frac{1}{1 + \exp(a)} = \sigma(-a)$
- Gradient: $\sigma'(a) = \frac{\exp(-a)}{(1 + \exp(-a))^2} = \sigma(a)(1 - \sigma(a))$

Classification loss

Data log-likelihood:

$$\text{loglik}(\mathcal{D}, \mathbf{w}) = \sum_{i=1}^N p(c_i | \phi(\mathbf{x}_i), \mathbf{w}) = \sum_{i=1}^N \underbrace{c_i \log \sigma(\mathbf{w}^T \phi(\mathbf{x}_i)) + (1 - c_i) \log (1 - \sigma(\mathbf{w}^T \phi(\mathbf{x}_i)))}_{\text{loss}_i \dots \text{loss of the } i\text{th sample}}$$

$$\begin{aligned} \frac{\partial \text{loss}_i}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}} (c_i \log \sigma(\mathbf{w}^T \phi(\mathbf{x}_i)) + (1 - c_i) \log (1 - \sigma(\mathbf{w}^T \phi(\mathbf{x}_i)))) \\ &=? \end{aligned}$$

Gradient for Logistic Regression

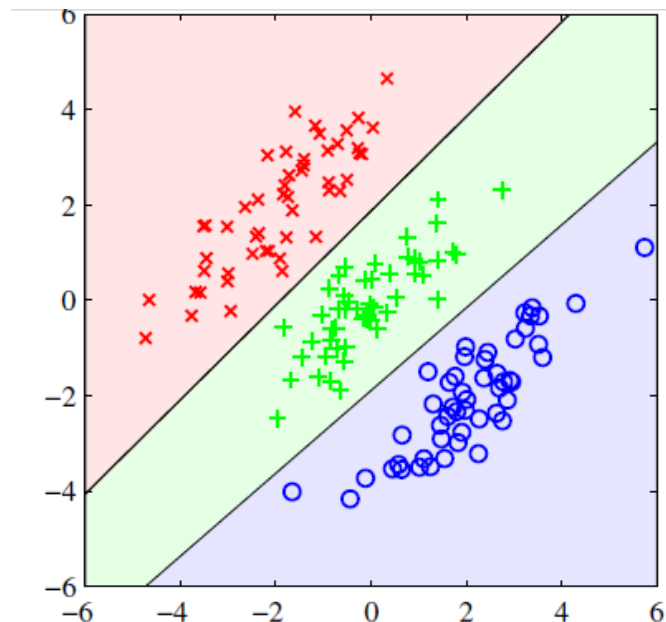
$$\begin{aligned}\frac{\partial \text{loss}_i}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}} \left(c_i \log \sigma(\mathbf{w}^T \phi(\mathbf{x}_i)) + (1 - c_i) \log (1 - \sigma(\mathbf{w}^T \phi(\mathbf{x}_i))) \right) \\ &= c_i \frac{1}{\sigma(\mathbf{w}^T \phi(\mathbf{x}_i))} \sigma(\mathbf{w}^T \phi(\mathbf{x}_i))(1 - \sigma(\mathbf{w}^T \phi(\mathbf{x}_i))) \phi(\mathbf{x}_i) \\ &\quad + (1 - c_i) \frac{1}{1 - \sigma(\mathbf{w}^T \phi(\mathbf{x}_i))} (-) \sigma(\mathbf{w}^T \phi(\mathbf{x}_i))(1 - \sigma(\mathbf{w}^T \phi(\mathbf{x}_i))) \phi(\mathbf{x}_i) \\ &= c_i (1 - \sigma(\mathbf{w}^T \phi(\mathbf{x}_i))) \phi(\mathbf{x}_i) - (1 - c_i) \sigma(\mathbf{w}^T \phi(\mathbf{x}_i)) \phi(\mathbf{x}_i) \\ &= (c_i - \sigma(\mathbf{w}^T \phi(\mathbf{x}_i))) \phi(\mathbf{x}_i)\end{aligned}$$

Multiclass Classification

Softmax Likelihood function:

$$p(c = i | \mathbf{x}) = \frac{\exp(\mathbf{w}_i^T \phi(\mathbf{x}))}{\sum_{k=1}^K \exp(\mathbf{w}_k^T \phi(\mathbf{x}))}$$

- Each class gets a weight vector
- Higher probability for class i if $\mathbf{w}_i^T \phi(\mathbf{x})$ is high
- For $K = 2$, \mathbf{w}_2 is redundant -> better to use sigmoid



Recap: Multinomial distribution

Multinomial / Categorical Distribution:

- K different events: $C \in \{1, \dots, K\}$
- Directly specifies probabilities: $p(C = k) = \mu_k, \quad \mu_k \geq 0, \quad \sum_{k=1}^K \mu_k = 1$
- Or written with 1-hot-encoding (without an “if” clause)

$$p(c) = \prod_{k=1}^K \mu_k^{h_{c,k}}$$

Depending on the class label of x, selects the correct μ_k

- where \mathbf{h}_x is the K-dimensional 1-hot encoding vector, which is one for the dimension $c = k$ and 0 elsewhere. $h_{x,k}$ is the k-th element of this vector.
- Think of it as tossing a die

Multiclass Classification

The multi-class classification problem can be expressed as a **conditional multinomial distribution**:

- I.e. the probability of the event c depends on the input \mathbf{x}
- We can again use the “exponential trick” to select the correct probability depending on c

$$\begin{aligned} p(c|\mathbf{x}) &= \prod_{k=1}^K p(c = k|\mathbf{x})^{h_{c,k}} \\ &= \prod_{k=1}^K \left(\frac{\exp(\mathbf{w}_k^T \phi(\mathbf{x}_i))}{\sum_{k'=1}^K \exp(\mathbf{w}_{k'}^T \phi(\mathbf{x}_i))} \right)^{h_{c,k}} \end{aligned}$$

Multiclass Classification

Data log-likelihood: $\text{loglik}(\mathcal{D}, \mathbf{w}_{1:K}) = \sum_{i=1}^N \log p(c_i | \mathbf{x}_i) = \sum_{i=1}^N \underbrace{\sum_{k=1}^K \mathbf{h}_{c_i,k} \log p(k | \mathbf{x}_i)}_{\text{loss}_i \dots \text{loss of the } i\text{th sample}}$

$$= \sum_{i=1}^N \sum_{k=1}^K \mathbf{h}_{c_i,k} \left[\mathbf{w}_k^T \phi(\mathbf{x}_i) - \log \left(\sum_{j=1}^K \exp(\mathbf{w}_j^T \phi(\mathbf{x}_i)) \right) \right]$$
$$= \sum_{i=1}^N \sum_{k=1}^K \mathbf{h}_{c_i,k} \underbrace{\mathbf{w}_k^T \phi(\mathbf{x}_i) - \log \left(\sum_{j=1}^K \exp(\mathbf{w}_j^T \phi(\mathbf{x}_i)) \right)}_{\text{independent from } k} \underbrace{\sum_k \mathbf{h}_{c_i,k}}_{=1}$$

- Can again be optimized by gradient ascent

Multiclass Classification

Gradient:

$$\frac{\partial \text{loss}_i}{\partial \mathbf{w}_k} = \frac{\partial}{\partial \mathbf{w}_k} \left(\sum_{k=1}^K \mathbf{h}_{c_i, k} \mathbf{w}_k^T \phi(\mathbf{x}_i) - \log \left(\sum_{j=1}^K \exp(\mathbf{w}_j^T \phi(\mathbf{x}_i)) \right) \right)$$

=?

Takeaway messages

What have we learned today?

- Refresher on probability theory and maximum likelihood
- Relation between maximum likelihood and least squares
- What is a linear classification problem ...
- ... and how to formalize it as likelihood maximization problem
 - Sigmoid likelihood for binary classification
 - Soft-max likelihood for multi-class
- What is gradient descent, stochastic gradient descent and mini-batches?
- How to apply gradient descent to logistic regression



Linear Regression:

$$SSE = \sum_{i=1}^N \left(y_i - \sum_{j=1}^D w_{ij} x_{ij} - w_0 \right)^2$$

$[w_0, \dots, w_D]^T$

\Rightarrow Matrix Form: $SSE = e^T e = (y - Xw)^T (y - Xw)$

(

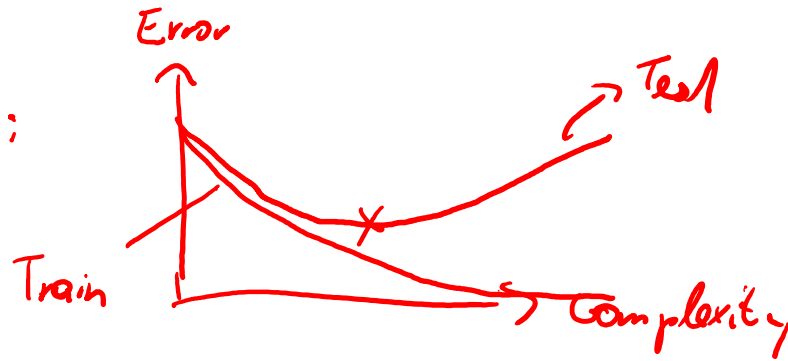
$$\begin{bmatrix} 1 & \vec{x}_1^T \\ 1 & \vec{x}_2^T \end{bmatrix}$$

\Rightarrow Least Squares Solution: $\frac{\partial SSE}{\partial w} \stackrel{!}{=} 0$

$$\Rightarrow w = \underbrace{(X^T X)^{-1}}_{\text{Pseudo Inverse}} X^T y$$

Regularization:

Overfitting:



⇒ Limit Complexity by using penalty on weight vector

$$(y - Xw)^T (y - Xw) + \lambda \underbrace{w^T w}_{L_2 \text{ penalty}} \rightarrow \text{would not include } w_0$$

$$\Rightarrow \text{Ridge Regression: } w = (X^T X + \underbrace{\lambda I})^{-1} X^T y$$
$$(X^T X + \text{diag}[0, \lambda, \dots, \lambda]) X^T y$$

