LSML19: Introduction to Large Scale Machine Learning

MINES ParisTech — March 2019

Chloé-Agathe Azencott
Centre for Computational Biology, Mines ParisTech
chloe-agathe.azencott@mines-paristech.fr
Practical information

- **Course website**
  
  [http://cazencott.info > Teaching > LSML 19](http://cazencott.info > Teaching > LSML 19)

- **Schedule:**
  - Morning sessions: 09:30 – 12:30
  - Afternoon sessions: 14:00 – 17:00

- **Lectures** are here in L.118

- **Practicals** are in L.117 L.119 L.120

- **Grade:**
  - 60% Exam (Friday, 14:00)
  - 40% RAMP
Today's goals

- Review what machine learning is
- Understand scalabilities issues
- Discover how to accelerate gradient descents with stochastic gradient descent
Acknowledgements

Slides inspired by Ala Al-Fuqaha, Ethem Alpaydi, Matthew Blaschko, Léon Bottou, Sanjiv Kumar, Trevor Hastie, Rob Tibshirani and Jean-Philippe Vert
Why machine learning?
Business Insider: 2017 is the year of machine learning

- Improving doctors
- Assisting lawyers
- Make cars drive themselves
Perception
Communication

Support Bot

Hello, what can I help you with?

May I please have your email address?

Customer

I would like to create a new ticket.

---

La souris est en dessous de la table. Le chat est sur la chaise. Le singe est sur la branche.

The mouse is below the table. The cat is on the chair. The monkey is on the branch.

Chatbot example from www.supportbots.ai
Text courtesy an Eddie Izzard show from 1999 called *Dressed to Kill.*
Reasoning
FDA News Release

FDA permits marketing of artificial intelligence-based device to detect certain diabetes-related eye problems

For Immediate Release  
April 11, 2018
Scientific discovery
A common thread: ML

- Using algorithms to build models from example (training) data.
- Statistics + optimization + computer science
Empirical risk minimization

**Ingredients:**

- **Data**
  \[ D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \]

- **Hypothesis class:**
  Shape of the decision function \( f \)

- **Loss function:**
  Cost/error of \( f \) on one data point

**Recipe:**

Find, among all functions of the hypothesis class, one that minimizes the loss on the training data (empirical risk).
(Un)supervised learning setting

- Iris dataset: n=150, p=4, t=1.
- Cancer drug sensitivity: n=10^3, p=10^6, t=100.
- Shopping, e-marketing...: n=O(10^6), p=O(10^9), t=O(10^8).
- Astronomy, GAFAs, web...: n=O(10^9), n=O(10^9), n=O(10^9).
Scaling ML algorithms

- **What is large scale?**
  - Data does not fit in RAM;
  - Data streams;
  - Algorithms do not run in a reasonable time on a single machine.

- **Important considerations:**
  - **Performance** increases with the number of samples.
  - **Likelihood to overfit** increases with the number of features.

- **Iris dataset:** n=150, p=4, t=1.
- **Cancer drug sensitivity:** n=10^3, p=10^6, t=100.
- **ImageNet:** n=14.10^6, p=6.10^3, t=22.10^3.
- **Shopping, e-marketing...:** n=O(10^6), p=O(10^9), t=O(10^8).
- **Astronomy, GAFAs, web...:** n=O(10^9), n=O(10^9), n=O(10^9).
A brief zoo of ML problems
Unsupervised learning

Learn a **new representation** of the data

\[ \mathcal{D} = \{ x_1, x_2, \ldots, x_n \} \]

Images, text, measurements, omics data...
Dimensionality reduction

Find a **lower-dimensional** representation

\[ \mathcal{D} = \{x_1, x_2, \ldots, x_n\} \]

Images, text, measurements, omics data...

Data \[ x^i \in \mathbb{R}^p \]

\[ m \ll p \]

ML algo

Data \[ x^i \in \mathbb{R}^m \]
Clustering

Group similar data points together

\[ D = \{ x_1, x_2, \ldots, x_n \} \]
Unsupervised learning

- **Dimensionality reduction**
  - PCA
- **Clustering**
  - k-means
- **Density estimation**
- **Feature learning**
Supervised learning

Make predictions

\[ \mathcal{D} = \{x_1, x_2, \ldots, x_n\} \]

\( X \) + \( y \)

\( \mathcal{D} \) is the data set with \( n \) samples, each with \( p \) features.

Predictor

\[ f(x) \approx y \]

decision function

\( f \) is the decision function that maps the input \( x \) to the output \( y \).

\( x \) is the input data.

\( y \) is the output label.
Classification

Make discrete predictions

Data

\[ D = \{ x_1, x_2, \ldots, x_n \} \]

ML algo

\[ f(x) \approx y \]

Predictor

Labels

\[ \{ y_1, y_2, \ldots, y_n \} \]

Binary classification

\[ y_i \in \{ 0, 1 \} \]

Multi-class classification

\[ y_i \in \{ 0, 1, \ldots, k \} \]
Regression

Make **continuous** predictions

\[ D = \{x_1, x_2, \ldots, x_n\} \]

\[ \{y_1, y_2, \ldots, y_n\} \]

\[ f(x) \approx y \]

\[ y_i \in \mathbb{R} \]
Supervised learning

- **Regression**
  - Ordinary least squares, ridge regression

- **Classification**
  - Logistic regression, SVM

- **Structured output prediction**
Main ML paradigms

- **Unsupervised learning:**
  - Dimensionality reduction;
  - Clustering;
  - Density estimation;
  - Feature learning.

- **Supervised learning:**
  - Regression;
  - Classification;
  - Structured output prediction.

- **Semi-supervised learning.**

- **Reinforcement learning.**
Dimensionality reduction: Principal Components Analysis
Principal Components Analysis

• **Objective:**
  - Reduce the dimension **without losing the variability** in the data;
  - Find a low-dimensional space such as to **maximize the variance** of the data projected onto that space.

• **The k-th principal component:**
  - Is **orthogonal** to all previous components:
    \[
    \langle w_k, w_1 \rangle = \langle w_k, w_2 \rangle = \cdots = \langle w_k, w_{k-1} \rangle = 0.
    \]
  - Captures the **largest amount of variance.**
    \[
    \max_k w^\top X^\top X w.
    \]
  - **Solution:** \( w \) is the k-th **eigenvector** of \( X^\top X \).
PCA example: Population genetics

Genetic data of 1387 Europeans

Novembre et al, 2008
Algorithmic complexity of PCA

- **Memory:**
  
  store the data $(X)$ and the covariance matrix $(X^TX)$:
  \[
  \mathcal{O}(\max(np, p^2)).
  \]

- **Runtime:**
  
  - Computing $X^TX$: $\mathcal{O}(np^2)$.
  
  - Computing $K$ eigenvectors by Lanczos iterations: $\mathcal{O}(Kp^2)$.
  
  - Computing the covariance matrix is more expensive than computing the $K$ first principal components!

- **Example** $n=10^9$, $p=10^8$:
  
  - Computing the covariance matrix: $10^{25}$ FLOPS.

  *Fastest world computer (Nov 2018): 75 peta flops $\rightarrow$ 2+ years.*
  
  - Storing the covariance matrix: $10^{16}\mathrm{B} = \left(10^{16}/2^{50}\right)\mathrm{PB} = 8\mathrm{PB}$. 


Clustering:
k-means
K-means clustering

• Goal:
  – Find a \textbf{cluster assignment} \( c_i \in \{1, \ldots, k\} \) for all \( i = 1, \ldots, n \).
  – that minimizes the \textbf{intra-cluster variance}:

\[
V = \sum_{k=1}^{K} \sum_{i : c_i = k} \frac{1}{|\{i : c_i = k\}|} \|x^i - \mu_k\|^2.
\]

– \textbf{centroids}: \( \mu_k = \frac{1}{|\{i : c_i = k\}|} \sum_{i : c_i = k} x^i \).
K-means clustering

• Goal:
  – Find a cluster assignment \( c_i \in \{1, \ldots, k\} \) for all \( i = 1, \ldots, n \).
  – that minimizes the intra-cluster variance:
    \[
    V = \sum_{k=1}^{K} \sum_{i:c_i=k} \frac{1}{|\{i : c_i = k\}|} \|x^i - \mu_k\|^2.
    \]
  – centroids: \( \mu_k = \frac{1}{|\{i : c_i = k\}|} \sum_{i:c_i=k} x^i \).

• Voronoi tessellation:
K-means clustering

• **Goal:**
  - Find a cluster assignment \( c_i \in \{1, \ldots, k\} \) for all \( i = 1, \ldots, n \).
  - that minimizes the intra-cluster variance:
    \[
    V = \sum_{k=1}^{K} \sum_{i:c_i=k} \frac{1}{|\{i : c_i = k\}|} \|x^i - \mu_k\|^2.
    \]
  - **centroids:** \( \mu_k = \frac{1}{|\{i : c_i = k\}|} \sum_{i : c_i = k} x^i. \)

• **NP-hard → Iterative algorithm**
  - **Assignment step:** fix the centroids, update assignments
    \[
    c_i \leftarrow \arg \min_{k \in 1, \ldots, K} \|x^i - \mu_{C_k}\|^2
    \]
  - **Update step:** update the centroids
    \[
    \mu_k \leftarrow \frac{1}{|\{i : c_i = k\}|} \sum_{i : c_i = k} x^i.
    \]
K-means clustering

$K = 3$
K-means clustering

Pick 3 centroids at random.
Assign each observation to the nearest centroid.

\[ K = 3 \]
K-means clustering

Recompute centroids.
K-means clustering

$K = 3$

Re-assign each observation to the nearest centroid.
K-means clustering

$K = 3$

Recompute centroids, and iterate process until convergence.
k-means complexity

- **Assignment step:** fix the centroids, update assignments:
  \[ c_i \leftarrow \arg \min_{k \in 1, \ldots, K} \| x^i - \mu_{C_k} \|^2 \]

  Compute \( n \times K \) distances in \( p \) dimensions \( \rightarrow O(Knp) \).

- **Update step:** update the centroids:
  \[ \mu_k \leftarrow \frac{1}{|\{ i : c_i = k \}|} \sum_{i: c_i = k} x^i. \]

  Sum \( n \) values in \( p \) dimensions \( \rightarrow O(np) \).

  \( T \) iterations \( \rightarrow O(KnpT) \).

- **Storage:**
  Store \( n \) cluster assignments + \( K \) centroids \( \rightarrow O(n + Kp) \).
  Store \( X \) \( \rightarrow O(np) \).
Ridge regression
Linear regression

Least-squares fit (equivalent to MLE under the assumption of Gaussian noise):

$$\hat{\beta} = \arg \min_{\beta} (y - X\beta)^\top (y - X\beta) = (X^\top X)^{-1} X^\top y$$

Solution uniquely defined when $X^\top X$ invertible.
Ridge regression

- Hoerl & Kennard 1970
- Goodness-of-fit + ridge regularization

\[ \hat{\beta}_{\text{ridge}} = \arg \min_{\beta} \left[ \| y - X \beta \|_2^2 + \lambda \| \beta \|_2^2 \right] \]

- Solution unique and always exists

\[ \hat{\beta}_{\text{ridge}} = (X^\top X + \lambda I)^{-1} X^\top y \]

- Limit cases:
  - \( \lambda \to 0 \): OLS (non-regularized) solution (low bias, high variance);
  - \( \lambda \to \infty \): \( \beta=0 \) (high bias, low variance).
- Correlated features get similar weights.
Complexity of ridge regression

\[ \hat{\beta}_{\text{ridge}} = (X^\top X + \lambda I)^{-1} X^\top y \]

- **Computing** \( X^\top X + \lambda I \)
  
  \[ O(np^2) \]

- **Inverting** \( X^\top X + \lambda I \)
  
  \[ O(p^3) \]

- When \( n \gg p \), computing \( X^\top X + \lambda I \) is more expensive than inverting it!
Regularization path

**Graph 1:**
- Y-axis: error (MSE)
- X-axis: regularization parameter
- Graph shows the relationship between error (MSE) and regularization parameter for different values of alpha.

**Graph 2:**
- Y-axis: regression weights
- X-axis: regularization parameter
- Graph plots the change in regression weights for different variables (e.g., fixed acidity, volatile acidity, citric acid, residual sugar) as the regularization parameter changes.
Ridge regression

Hyperparameter setting
Setting $\lambda$

- Underfitting
- Overfitting

Prediction error vs. Model complexity

On training data
On new data
Setting $\lambda$

- Data splitting strategy: **cross-validation**:
  - Cut the training set in $k$ equally-sized chunks.
  - $K$ **folds**: one chunk to test, the ($K-1$) others for training.

- Cross-validation score: perf averaged over the $K$ folds.

For a grid of values for $\lambda$.

| $\lambda_1$ | $\lambda_2$ | ... | $\lambda_M$ |
Setting $\lambda$

- Data splitting strategy: **cross-validation:**
  - Cut the training set into $k$ equally-sized chunks.
  - $K$ **folds:** one chunk to test, the $(K-1)$ others for training.
    - Cross-validation score: perf averaged over the $K$ folds.
    - Choose the $\lambda$ with the best cross-validation score.
- Multiplies **time complexity** by $KM$. 
Ridge regression

$\ell_2$-regularized learning
**\(\ell_2\)-regularized learning**

\[
\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} \ell(f_{\beta}(x^i), y^i) + \lambda \|\beta\|^2.
\]

- Generalization of the ridge regression to any **loss**.
- If the loss is **convex**, then the problem is **strictly convex** and has a **unique global solution**, which can be found **numerically**.
**ℓ2-regularized learning**

\[
\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} \ell(f_{\beta}(x^i), y^i) + \lambda \|\beta\|^2.
\]

**Empirical risk**

**ℓ2 regularization**

- Generalization of the ridge regression to any **loss**.
- If the loss is **convex**, then the problem is **strictly convex** and has a **unique global solution**, which can be found **numerically**.

- **Absolute loss**: $|u - y|$  
- **Quadratic loss**: $(u - y)^2$  
- **ε-insensitive loss**: $(|u - y| - \epsilon)_+$  
- **Huber loss**: mix quadratic & linear

\[
\begin{cases}
\frac{1}{2} (u - y)^2 & \text{if } |u - y| < \epsilon \\
\epsilon |u - y| - \frac{1}{2} \epsilon^2 & \text{otherwise}.
\end{cases}
\]
Gradient descent
Gradient descent

If the loss is **convex**, then the problem is **strictly convex** and has a **unique global solution**, which can be found **numerically**.

- **Suppose** the function to minimize is **derivable**: 

  \[ J(v) \geq J(u) + \nabla J(u)^\top (v - u). \]

  First-order Taylor expansion of \( f \) in \( u \)

  \[ \nabla J(u) = 0 \iff u \text{ minimizes } J. \]

\[(v, J(v)) \]
\[(v, J(u) + J'(u)(v-u)) \]
\[(u, J(u)) \]
Gradient descent

- Minimize a derivable, strictly convex function $J$ by finding where its gradient is 0
  - Set $a_0$ randomly
  - Update: $a_1 = a_0 - \alpha \nabla J(a_0)$
  - Repeat
  - Stop when $|\nabla J(a_k)| < \varepsilon$. 

$J'(a_0) < 0$
Classification
Logistic regression

\[ y_i \in \{-1, 1\} \]

- Model \( y \) as a linear function of \( x \)?

\[ y = \beta_0 + \sum_{j=1}^{p} \beta_j x_j. \]
Logistic regression

\[ y_i \in \{-1, 1\} \]

- Model \( y \) as a linear function of \( x \)?
  \[ y = \beta_0 + \sum_{j=1}^{p} \beta_j x_j. \]

Model the **log-odds ratio** as a linear function of \( x \).

\[
\log \frac{\mathbb{P}(y = 1 | x)}{1 - \mathbb{P}(y = 1 | x)} = \beta_0 + \sum_{j=1}^{p} \beta_j x_j. 
\]
Logistic regression

\[ y_i \in \{-1, 1\} \]

- Model \( y \) as a linear function of \( x \)?
  \[ y = \beta_0 + \sum_{j=1}^{p} \beta_j x_j. \]

  Model the **log-odds ratio** as a linear function of \( x \).

  \[
  \log \frac{\mathbb{P}(y = 1|x)}{1 - \mathbb{P}(y = 1|x)} = \beta_0 + \sum_{j=1}^{p} \beta_j x_j. \]

  \[
  \mathbb{P}(y = 1|x) = \frac{1}{1 + \exp(-y f_\beta(x))} = \sigma(y f_\beta(x))
  \]

\[
\text{Logit}(p) = \log \left( \frac{p}{1-p} \right)
\]

\[
\text{Logistic}(u) = \left( \frac{1}{1 + e^{-u}} \right)
\]
Ridge logistic regression

- Le Cessie and van Houwelingen, 1992
- Goodness-of-fit + ridge regularization

\[
\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} \log (1 + \exp(-y^i \cdot \beta^\top \mathbf{x}^i)) + \lambda \|\beta\|_2^2
\]

\text{Empirical risk} \\
\text{Logistic loss} \\
\text{Ridge regularization}

- **Logistic loss**: negative conditional likelihood

\[
\ell_{\text{logistic}}(y, f(\mathbf{x})) = -\log \mathbb{P}(y \mid f(\mathbf{x})) = \log (1 + \exp(-y f(\mathbf{x}))).
\]

- No explicit solution.

- Smooth convex optimization problem that can be solved numerically.
Newton-Raphson iterations

- Minimise $J$ convex, differentiable.

- **Gradient descent:**
  
  $$u^{\text{new}} \leftarrow u^{\text{old}} - \alpha \nabla J(u^{\text{old}}).$$

- Suppose $f$ is **twice differentiable**.
  
  - **Second-order Taylor’s expansion:**
    
    $$J(v) \approx J(u) + \nabla J(u)^\top (v - u) + \frac{1}{2} (v - u)^\top \nabla^2 J(u)^\top (v - u).$$

  - Minimize in $v$ instead of in $u$.
    
    $$\nabla_v g(v) = \nabla J(u) + \nabla^2 J(u)(v - u).$$

    $$\nabla_v g(v) = 0 \Rightarrow v = u - (\nabla^2 J(u))^{-1} \nabla J(u).$$

  - Hence use $\alpha = (\nabla^2 J(u^{\text{old}}))^{-1}$. 

Solving the logistic regression

$$\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} \log \left( 1 + \exp(-y^i \cdot \beta^\top x^i) \right) + \lambda \lVert \beta \rVert_2^2$$

$$\nabla_{\beta} J(\beta) = -\frac{1}{n} \sum_{i=1}^{n} \frac{y_i x_i}{1 + e^{y_i \beta^\top x_i}} + 2\lambda \beta$$

$$\nabla^2_{\beta} J(\beta) = \frac{1}{n} \sum_{i=1}^{n} \frac{x_i x_i^\top e^{y_i \beta^\top x_i}}{(1 + e^{y_i \beta^\top x_i})^2} + 2\lambda I$$

- Can be solved with Newton-Raphson iterations.
- Each step is equivalent to solving a weighted ridge regression.
- \( \rightarrow \) IRLS: Iteratively Reweighted Least Squares.
- Complexity: \( O(T(np^2 + p^3)) \).

Number of iterations
Large margin classifiers

- **Margin:**
  - classify $\mathbf{x}$ as positive if $f(\mathbf{x}) > 0$ and negative otherwise.
  - ideally, $yf(\mathbf{x}) > 0$.

- **Large margin classifier:** maximize $yf(\mathbf{x})$:
  \[
  \min_{\beta} \sum_{i=1}^{n} \varphi(y^i f(\mathbf{x}^i)) + \lambda ||\beta||^2_2
  \]
  for a **convex, non-increasing** function $\varphi : \mathbb{R} \to \mathbb{R}_+$.

- **Logistic regression:**
  \[
  \min_{\beta} \sum_{i=1}^{n} \log \left(1 + \exp(-y^i . \beta^\top \mathbf{x}^i)\right) + \lambda ||\beta||^2_2
  \]
  \[
  \varphi : u \mapsto \log(1 + e^{-u}).
  \]
Large margin classifiers

- **Linear Support Vector Machine (SVM):**

\[
\min_{\beta} \sum_{i=1}^{n} \max(0, 1 - y^i \cdot \beta^\top x^i) + \lambda \|\beta\|_2^2.
\]

Hinge loss
\[
\varphi : u \mapsto \max(0, 1 - u).
\]
Linear SVM

\[
\min_{\beta} \sum_{i=1}^{n} \max \left( 0, 1 - y^i \beta^\top x^i \right) + \lambda \|\beta\|_2^2.
\]

- Non-smooth, convex optimization problem; Quadratic Program.
- Equivalent to the dual problem

\[
\max_{\alpha \in \mathbb{R}^n} 2\alpha^\top 1 - \alpha^\top XX^\top \alpha \quad \text{s. t.} \quad 0 \leq y^i \alpha_i \leq \frac{1}{2\lambda} \quad \text{for } i = 1, \ldots, n.
\]

\[
\beta^* = X^\top \alpha^*
\]

\[
f_{\beta^*}(x) = \beta^*^\top x = \alpha^*^\top X x.
\]

- Complexity (training):
  - Storing \(XX^\top\) \(\mathcal{O}(n^2)\).
  - Optimization: \(\mathcal{O}(n^3)\).

- Complexity (prediction):
  - Primal: \(\mathcal{O}(p)\).
  - Dual: \(\mathcal{O}(np)\).
Kernel methods
Motivation
Non-linear mapping to a feature space

\[ \Phi \left( \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right) = \begin{pmatrix} x_1^2 \\ x_2^2 \end{pmatrix} \]

\[ x_1^2 + x_2^2 - R^2 = 0 \]

\[ \Phi(x)_1 + \Phi(x)_2 - R^2 = 0 \]
SVM in the feature space

\[ \Phi : \mathcal{X} \to \mathcal{H}. \]

- **Train:**

\[
\max_{\alpha \in \mathbb{R}^n} 2 \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \sum_{l=1}^{n} \alpha_i \alpha_j \langle x^i, x^l \rangle \quad \text{s. t.} \quad 0 \leq y^i \alpha_i \leq \frac{1}{2\lambda} \quad \text{for } i = 1, \ldots, n.
\]

\[
\max_{\alpha \in \mathbb{R}^n} 2 \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \sum_{l=1}^{n} \alpha_i \alpha_j \langle \Phi(x^i), \Phi(x^l) \rangle_{\mathcal{H}} \quad \text{s. t.} \quad 0 \leq y^i \alpha_i \leq \frac{1}{2\lambda} \quad \text{for } i = 1, \ldots, n.
\]

- **Predict** with the decision function \( f_{\beta^*}(x) = \alpha^* \top X x \).

\[
f_{\alpha^*}(x) = \sum_{i=1}^{n} \alpha^*_i \langle x^i, x \rangle.
\]

\[
f_{\alpha^*}(x) = \sum_{i=1}^{n} \alpha^*_i \langle \Phi(x^i), \Phi(x) \rangle_{\mathcal{H}}.
\]
Kernel SVM

\[ k : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \]

- **Train:**

\[
\max_{\alpha \in \mathbb{R}^n} 2 \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \sum_{l=1}^{n} \alpha_i \alpha_j \langle \Phi(x^i), \Phi(x^l) \rangle_{\mathcal{H}} \quad \text{s. t. } 0 \leq y^i \alpha_i \leq \frac{1}{2\lambda} \text{ for } i = 1, \ldots, n.
\]

\[
\max_{\alpha \in \mathbb{R}^n} 2 \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \sum_{l=1}^{n} \alpha_i \alpha_j \quad k(x^i, x^l) \quad \text{s. t. } 0 \leq y^i \alpha_i \leq \frac{1}{2\lambda} \text{ for } i = 1, \ldots, n.
\]

- **Predict** with the decision function

\[
f_{\alpha^*}(x) = \sum_{i=1}^{n} \alpha^*_i \langle \Phi(x^i), \Phi(x) \rangle_{\mathcal{H}}.
\]

\[
f_{\alpha^*}(x) = \sum_{i=1}^{n} \alpha^*_i \quad k(x^i, x).
\]
Kernel trick

- $k$ may be quite **efficient** to compute, even if $\mathcal{H}$ is a **very high-dimensional** or even infinite-dimensional space.
- For any positive semi-definite function $k$, there exists a feature space $\mathcal{H}$ and a feature map $\phi$ such that
  $$ k(x, x') = \langle \Phi(x), \Phi(x') \rangle_\mathcal{H}. $$
- Hence you can define mappings **implicitly**.
- **Kernel trick**: algorithms that only involve the samples through their dot products can be rewritten using kernels in such a way that they can be applied in the initial space without ever computing the mapping.
Non-linear mapping to a feature space

\[ \Phi \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1^2 \\ x_2^2 \end{pmatrix} \]

\[ x_1^2 + x_2^2 - R^2 = 0 \quad \Phi(x)_1 + \Phi(x)_2 - R^2 = 0 \]

\[ K(x, x') = \left\langle \begin{pmatrix} x_1^2 \\ x_2^2 \end{pmatrix}, \begin{pmatrix} x'_1^2 \\ x'_2^2 \end{pmatrix} \right\rangle = x_1^2 x'_1^2 + x_2^2 x'_2^2 \]
Polynomial kernels

\[ \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^2 \quad \Phi(\mathbf{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix} \in \mathbb{R}^3 \]

\[ K(\mathbf{x}, \mathbf{x}') = x_1^2x_1'^2 + 2x_1x_2x_1'x_2' + x_2^2x_2'^2 = \langle \mathbf{x}, \mathbf{x}' \rangle^2 \]

More generally, for \( \mathcal{X} = \mathbb{R}^p \)

\[ K(x, x') = (\langle x, x' \rangle + 1)^d \]

is an inner product in a feature space of all monomials of degree up to \( d \).
Gaussian kernel

\[ K(x, x') = \exp \left( -\frac{||x - x'||^2}{2\sigma^2} \right) \]

The feature space has infinite dimension.

\[
K(x, x') = \exp \left( \frac{-1}{2\sigma^2} ||x||^2 \right) \exp \left( \frac{1}{\sigma^2} \langle x, x' \rangle \right) \exp \left( \frac{-1}{2\sigma^2} ||x'||^2 \right) = f(x) \sum_{r=0}^{+\infty} \frac{\langle x, x' \rangle^r}{\sigma^{2r} r!} f(x')
\]
Kernel ridge regression

- Ridge regression in input space: $\mathbb{R}^p$

$$\hat{\beta}_{\text{ridge}} = \arg\min_{\beta} \|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2$$

$$\hat{\beta}_{\text{ridge}} = (X^\top X + \lambda I)^{-1} X^\top y \quad f(x) = x\hat{\beta}_{\text{ridge}}.$$ 

- In a feature space of dimension $d$:

$$f(x) = \Phi(x) \left( \Phi(X)^\top \Phi(X) + \lambda I \right)^{-1} \Phi(X)^\top y.$$ 

$$f(x) = \Phi(x)\Phi(X)^\top \left( \lambda I + \Phi(X)\Phi(X)^\top \right)^{-1} y.$$ 

- Ridge regression in sample space: $\mathbb{R}^n$

$$f(x) = \kappa (\lambda I + K)^{-1} y.$$ 

$$\kappa_i = k(x, x^i) \quad K_{il} = k(x^i, x^l)$$
Complexity of KRR

\[ f(x) = \kappa (\lambda I + K)^{-1} y. \]

\[ \kappa_i = k(x, x^i) \]
\[ K_{il} = k(x^i, x^l) \]

- Computing K: \( \mathcal{O}(n^2p) \).
- Storing K: \( \mathcal{O}(n^2) \).
- Inverting K + λI: \( \mathcal{O}(n^3) \).
- Computing a prediction for one sample:
  - computing \( \kappa \): \( \mathcal{O}(np) \).
  - computing the products: \( \mathcal{O}(n) \).
Algorithmic complexity recap
## Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Memory</th>
<th>Train time</th>
<th>Predict time</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCA</td>
<td>$O(p^2)$</td>
<td>$O(np^2)$</td>
<td>$O(p)$</td>
</tr>
<tr>
<td>k-means</td>
<td>$O(np)$</td>
<td>$O(Knp)$</td>
<td>$O(Kp)$</td>
</tr>
<tr>
<td>Ridge regression</td>
<td>$O(p^2)$</td>
<td>$O(np^2)$</td>
<td>$O(p)$</td>
</tr>
<tr>
<td>Logistic regression</td>
<td>$O(np)$</td>
<td>$O(np^2)$</td>
<td>$O(p)$</td>
</tr>
<tr>
<td>SVM, kernel methods</td>
<td>$O(np)$</td>
<td>$O(n^3)$</td>
<td>$O(np)$</td>
</tr>
</tbody>
</table>

- **Training** can take place offline
- ... unless data is streaming
- **Prediction** should be fast!
Techniques for large-scale ML

- Use the **deep learning tricks**.
  - Deep learning (March 26, F. Moutarde).
  - Natural Language Processing (March 27, E. Grave).
- **Distribute** data & computation on modern archs.
  - Systems for large-scale ML (March 28, C.-A. Azencott).
- **Trade optimization accuracy for speed.**
Gradient descents
convergence rates
Flavours of convexity

- J differentiable is **convex** iff for all $u, v \in \mathbb{R}^p$, 
  \[ J(u) \geq J(v) + \nabla J(v)^\top (u - v). \]

- J is **L-smooth**, or **L-Lipschitz gradient** iff
  - it is twice differentiable and for all $u, v \in \mathbb{R}^p$, 
    \[ |\nabla J(u) - \nabla J(v)| \leq L \|u - v\|. \]
    J doesn’t vary “sharply”.

- J is **m-strongly convex** iff for all $u, v \in \mathbb{R}^p$, 
  \[ J(u) \geq J(v) + \nabla J(v)^\top (u - v) + \frac{m}{2} \|u - v\|^2. \]
  J is “not too flat”, lower-bounded by a quadratic.
Gradient descent convergence rates

- **Gradient descent** with fixed step size:
  \[ u^{\text{new}} \leftarrow u^{\text{old}} - \alpha \nabla J(u^{\text{old}}). \]
  - \( J \) convex, \( L \)-smooth:
    converges in \( O(1/\varepsilon) \) iterations \( \rightarrow O(np/\varepsilon). \)
  - \( J \) \( m \)-strongly convex, \( L \)-smooth:
    converges in \( O(\kappa \log 1/\varepsilon) \) iterations \( \rightarrow O(np \kappa \log(1/\varepsilon)). \)

- **Newton-Raphson** iterations
  \( \alpha = \frac{1}{L} \)
  \( \kappa = \frac{L}{m} \)
  \[ \alpha = \left( \nabla^2 J(u^{\text{old}}) \right)^{-1}. \]
  - \( J \) convex, \( L \)-smooth:
    converges in \( O(\log \log 1/\varepsilon) \) iterations \( \rightarrow O((np^2+p^3) \log \log(1/\varepsilon)). \)
Gradient descent convergence rates

- **Gradient descent** with fixed step size:
  \[ \mathbf{u}^{\text{new}} \leftarrow \mathbf{u}^{\text{old}} - \alpha \nabla J(\mathbf{u}^{\text{old}}). \]
  - \( J \) convex, \( L \)-smooth:
    converges in \( O(1/\epsilon) \) iterations \( \rightarrow O(np/\epsilon) \).
  - \( J \) \( m \)-strongly convex, \( L \)-smooth:
    converges in \( O(\kappa \log 1/\epsilon) \) iterations \( \rightarrow O(np \kappa \log(1/\epsilon)) \).

- **Newton-Raphson** iterations
  \[ \alpha = \left( \nabla^2 J(\mathbf{u}^{\text{old}}) \right)^{-1}. \]
  - \( J \) convex, \( L \)-smooth:
    converges in \( O(\log \log 1/\epsilon) \) iterations
    \( \rightarrow O((np^2+p^3) \log \log(1/\epsilon)) \).

\[ \alpha = \frac{1}{L} \]
\[ \kappa = \frac{L}{m} \]
Stochastic gradient descent

\[ J : u \mapsto \sum_{i=1}^{n} h_i(u). \]

- **Gradient descent:**
  \[ u^{\text{new}} \leftarrow u^{\text{old}} - \alpha \sum_{i=1}^{n} \nabla h_i(u^{\text{old}}). \]

- **Stochastic gradient descent:**
  \[ u^{\text{new}} \leftarrow u^{\text{old}} - \alpha \nabla h_{l_i}(u^{\text{old}}). \]

**Sampling with replacement:** chose \( l_i \) uniformly at random in \( \{1, 2, \ldots, m\} \).

- \( J \) **convex, L-smooth:**
  converges in \( O(\kappa/\varepsilon^2) \) iterations \( \rightarrow O(\kappa \rho/\varepsilon^2). \)

- \( J \) **m-strongly convex, L-smooth:**
  converges in \( O(\kappa/\varepsilon) \) iterations \( \rightarrow O(\kappa \rho/\varepsilon). \)

We got rid of \( n \)!
Convex optimization

- A number of ML algorithms can be formulated as **convex optimization** problems.
- They can be solved **numerically** thanks to variants of the **gradient descent**.
- Trading **optimization accuracy** for **speed**:
  - **Necessary** when reaching accuracy is too resource-consuming.
  - Does not necessarily have a large **impact on performance**: test error is more important than training error.