

Machine Learning with Python

Wenbin Guo
Bioinformatics IDP, UCLA
wbguo@ucla.edu
2022 Fall

Day 3: **Unsupervised** learning

Wenbin Guo
Bioinformatics IDP, UCLA
wbguo@ucla.edu
2022 Fall

Agenda

- Day 1: Introduction to machine learning
 - Some key concepts in machine learning
 - Jupyter notebook and some packages usage
- Day 2: Supervised learning
 - Classification
 - Regression
 - Regularization
- Day 3: **Unsupervised** learning
 - Dimension reduction
 - Clustering



Notations of the slides

- Code or Pseudo-Code chunk starts with "➤", e.g.
➤ `print("Hello world!")`
- [Link](#) is underlined

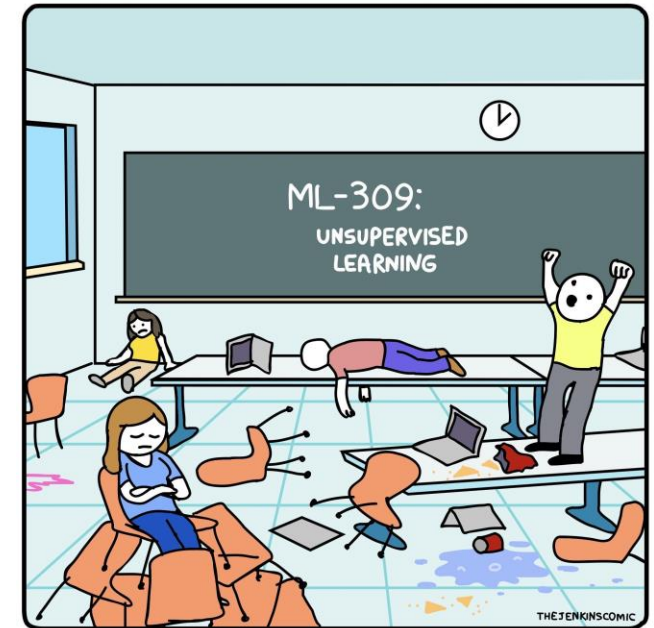
Overview

Time

- 3-hour workshop (45min + 45min + 30min + practice/Q&A)

Topics

- ☐ Regression
- ☐ Regularization
- ☐ Dimension reduction
- ☐ Clustering

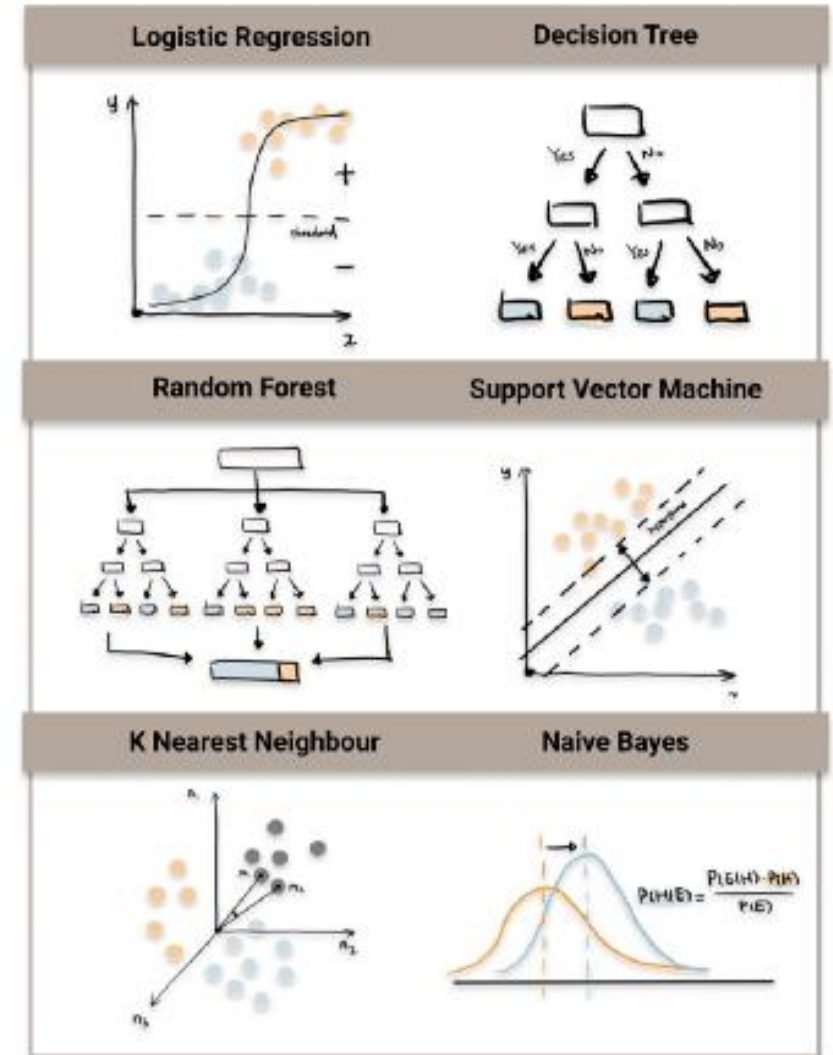


Summary –Day2

Classification:

❑ 9 different classification algorithms

- Logistic regression
- KNN
- Naïve bayes
- Support vector machine
- Decision tree
- Random forest
- Adaboost
- Gradient boosting
- Neural network



Summary –Day2

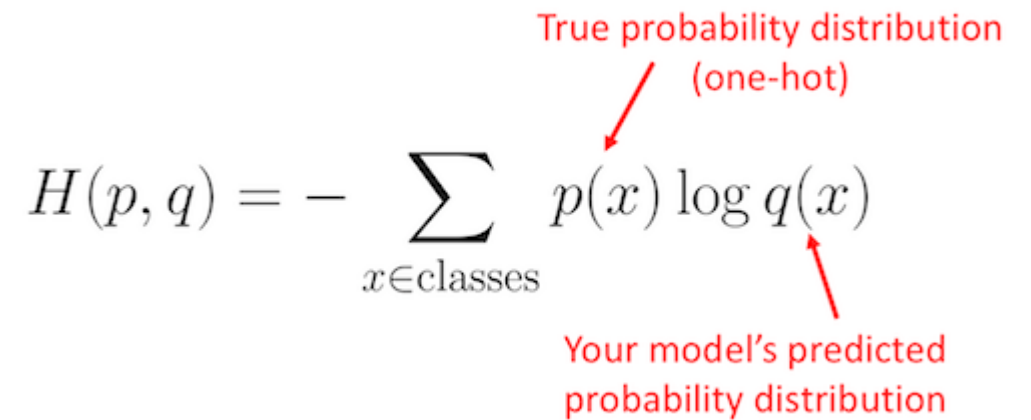
Classification:

- ❑ 9 different classification algorithms
- ❑ Cross entropy loss function

$$H(p, q) = - \sum_{x \in \text{classes}} p(x) \log q(x)$$

True probability distribution (one-hot)

Your model's predicted probability distribution



Summary –Day2

Classification:

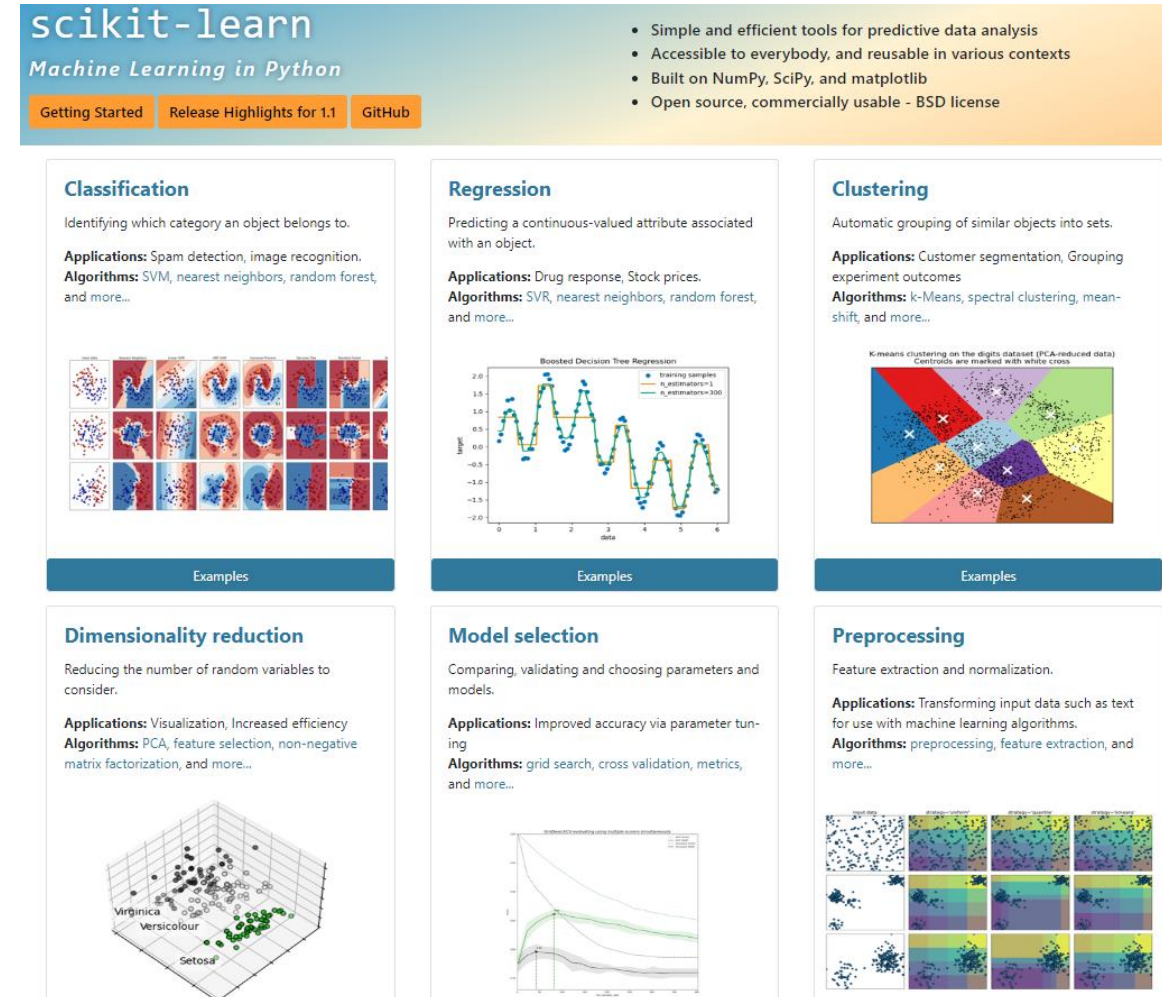
- ❑ 9 different classification algorithms
- ❑ Cross entropy loss function
- ❑ Performance measure
 - Accuracy
 - Confusion matrix
 - Precision & Recall
 - ROC, AUC, PR-ROC

		Predicted Class		
		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error	Sensitivity $\frac{TP}{(TP + FN)}$
	Negative	False Positive (FP) Type I Error	True Negative (TN)	Specificity $\frac{TN}{(TN + FP)}$
		Precision $\frac{TP}{(TP + FP)}$	Negative Predictive Value $\frac{TN}{(TN + FN)}$	Accuracy $\frac{TP + TN}{(TP + TN + FP + FN)}$

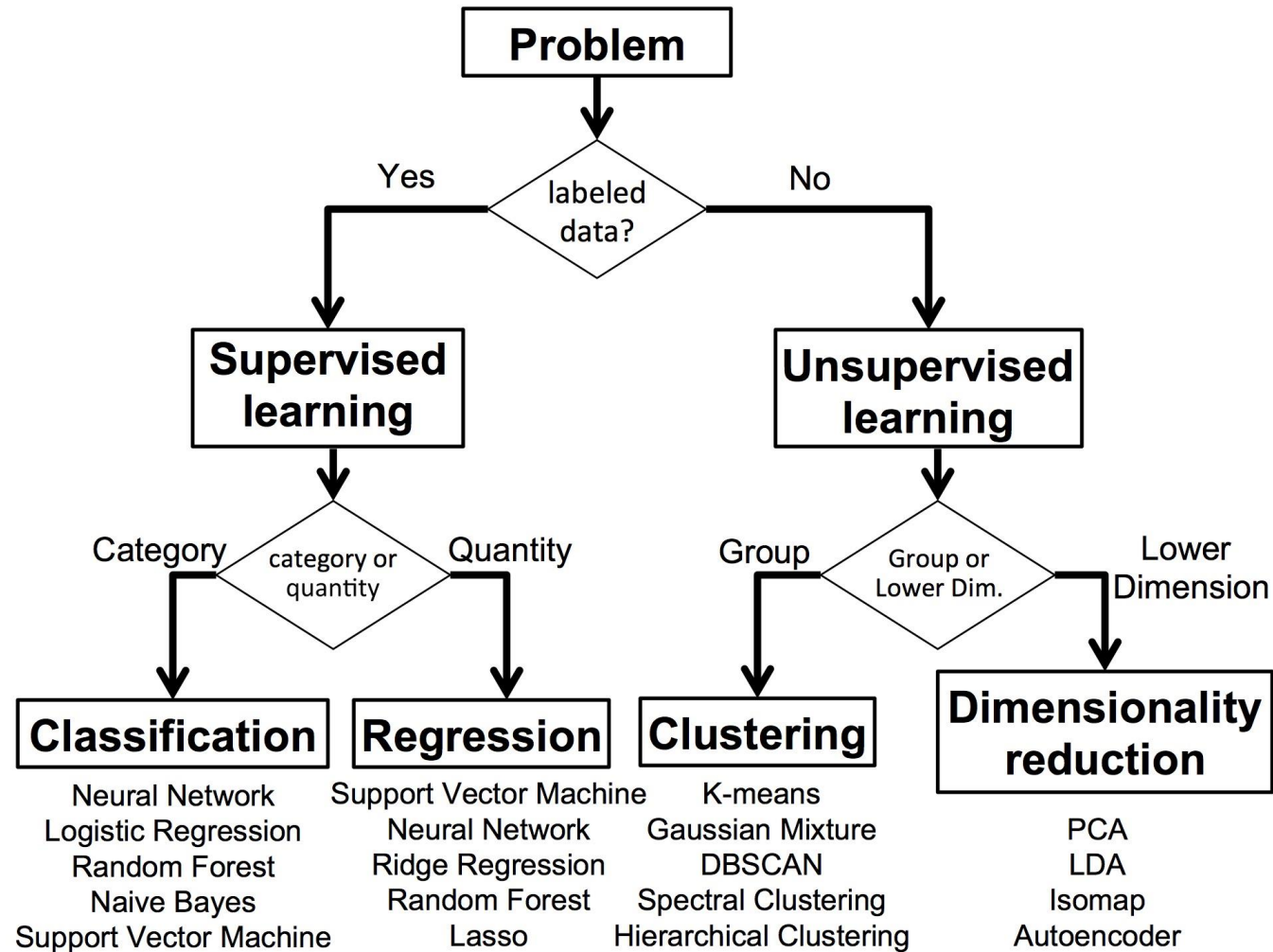
Summary –Day2

Classification:

- ❑ 9 different classification algorithms
- ❑ Cross entropy loss function
- ❑ Performance measure
- ❑ Practice
 - Scikit-learn API
 - Training-test-validation set construction
 - Performance measure calculation
 - ROC curve, decision boundary
 - Linearly non-separable example
 - Parameter tuning (overfitting/underfitting)

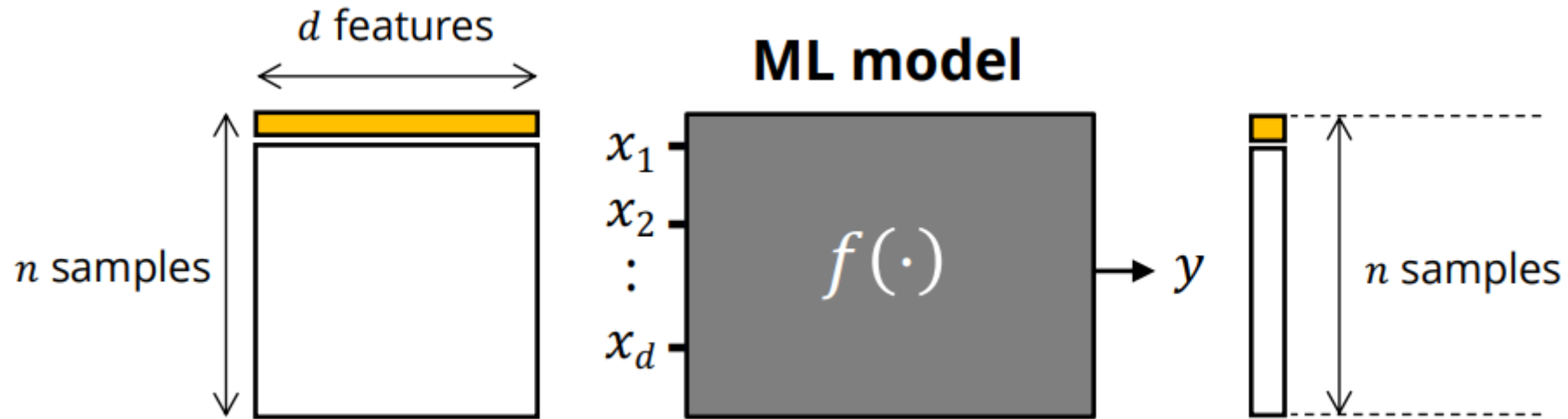


Types of machine learning



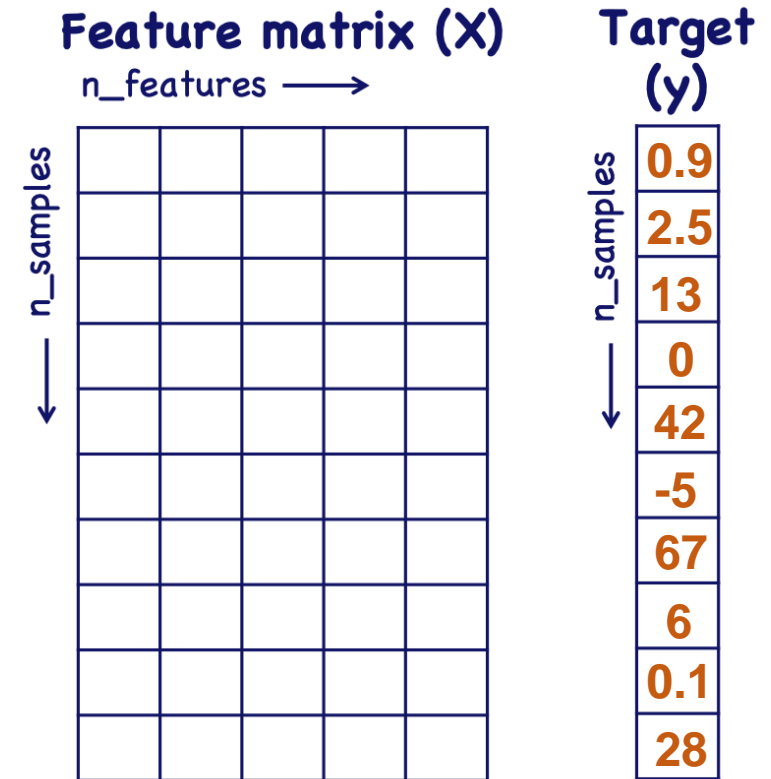
Supervised learning

- Training data with n ***samples*** of ***features*** x and ***label*** y
- Learn a function class $f(x)$ to describe y based on x



Different choice of $f()$ for regression tasks

- Linear regression
- Polynomial regression
- SVR
- Tree-based regression
- Boosting
- Neural Network
- ...



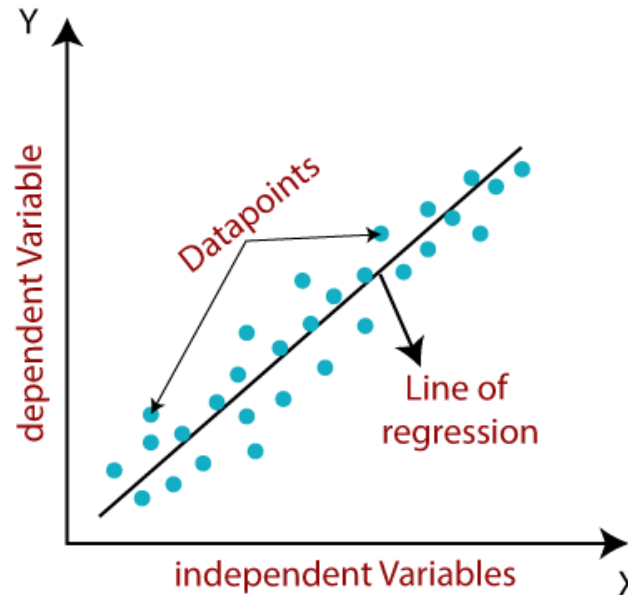
$$\hat{y} = f(x)$$

Linear regression

- Make prediction by computing the weighted sum of features

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

$$\hat{y} = h_{\theta}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$



Linear regression

- Make prediction by computing the weighted sum of features

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

$$\hat{y} = h_{\theta}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

- Train the model: find a parameter set to **best fit** the data

- Loss function

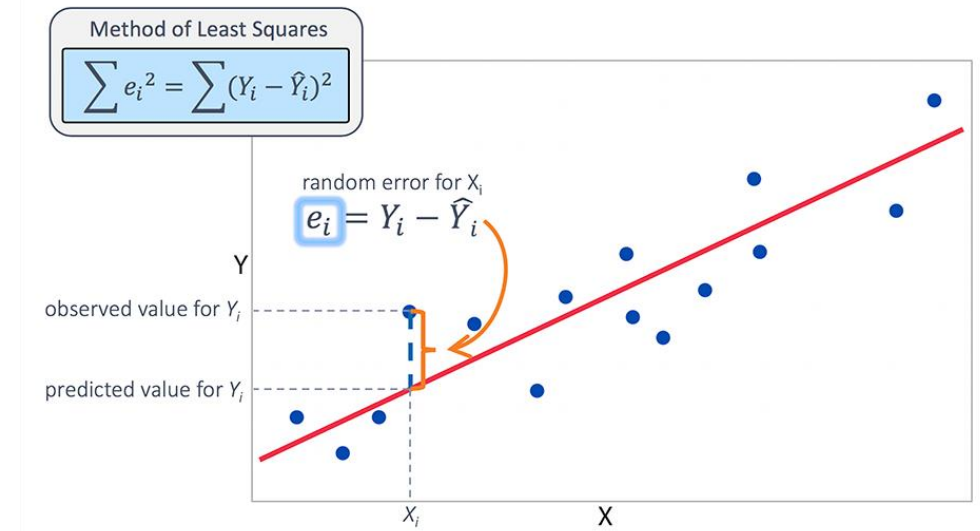
Equation 4-3. MSE cost function for a Linear Regression model

$$\text{MSE}(\mathbf{X}, h_{\theta}) = \frac{1}{m} \sum_{i=1}^m \left(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)} \right)^2$$

Sometimes MAE is also used (less sensitive to outliers)

Equation 2-2. Mean absolute error (MAE)

$$\text{MAE}(\mathbf{X}, h) = \frac{1}{m} \sum_{i=1}^m \left| h(\mathbf{x}^{(i)}) - y^{(i)} \right|$$



Linear regression

- Make prediction by computing the weighted sum of features

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

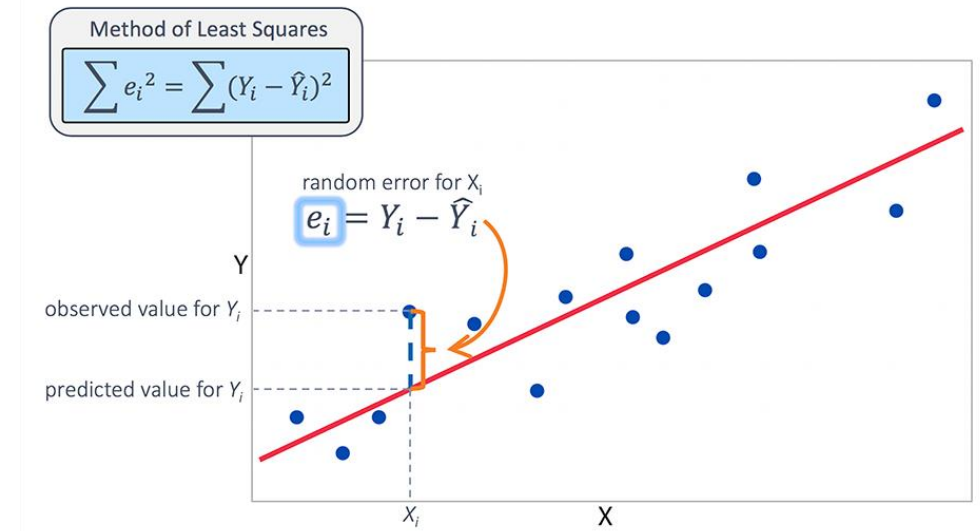
$$\hat{y} = h_{\theta}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

- Train the model: find a parameter set to **best fit** the data

- Normal equation

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

$$\boldsymbol{\theta} = E(\hat{\boldsymbol{\theta}})$$



Polynomial regression

- When your data is more complex than a straight line
 - Add powers of a feature as new features

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2 + \beta_5 x_1 x_2$$

Be aware of the combinatory explosion of features!

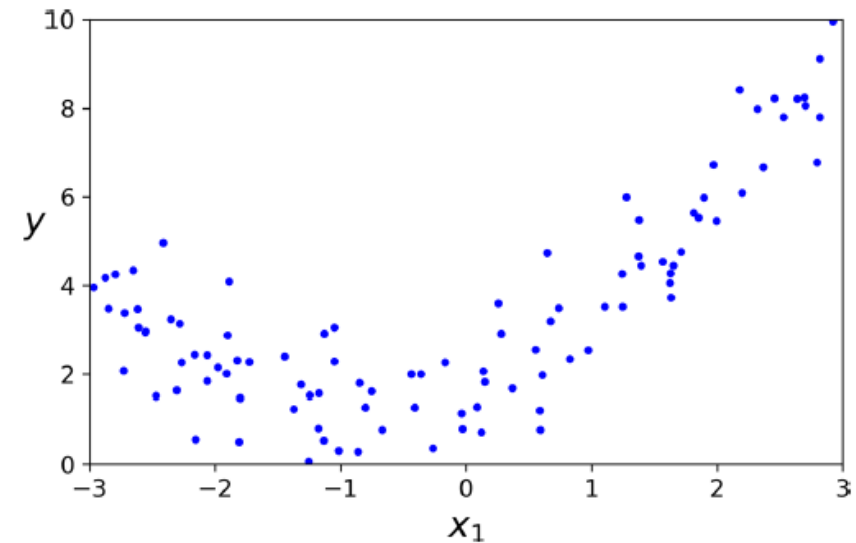
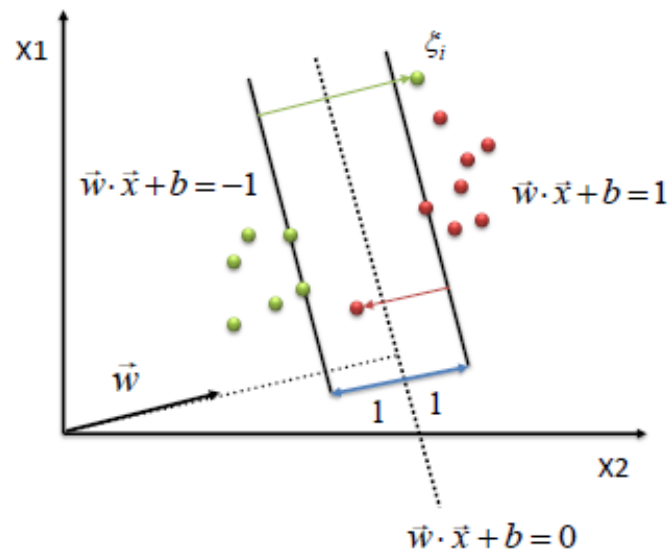


Figure 4-12. Generated nonlinear and noisy dataset

Support vector regression

SVM classification

fit the largest possible street between two classes
while limiting margin violations



Constraint becomes :

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \forall x_i$$

$$\xi_i \geq 0$$

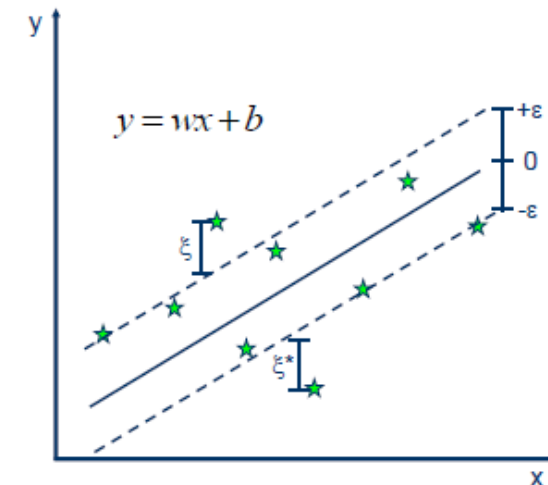
Objective function
penalizes for misclassified
instances and those within
the margin

$$\min \frac{1}{2} \|w\|^2 + C \sum_i \xi_i$$

C trades-off margin width
and misclassifications

SVM regression

fit as many instances as possible *on* the street
while limiting margin violations



• Minimize:

$$\frac{1}{2} \|w\|^2 + C \sum_{i=1}^N (\xi_i + \xi_i^*)$$

• Constraints:

$$y_i - wx_i - b \leq \epsilon + \xi_i$$

$$wx_i + b - y_i \leq \epsilon + \xi_i^*$$

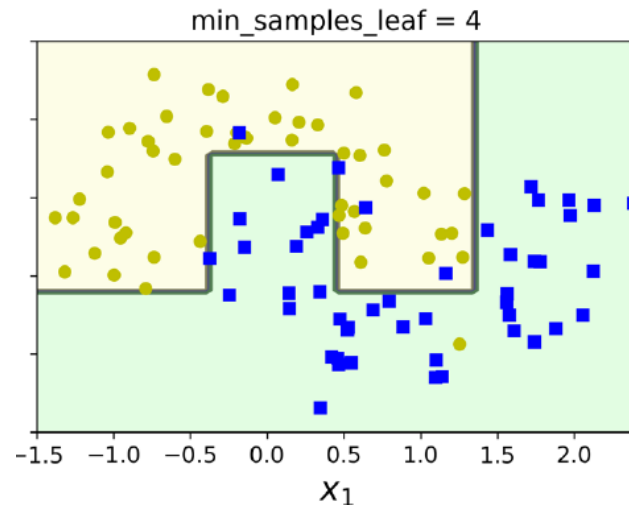
$$\xi_i, \xi_i^* \geq 0$$

The kernel tricks can be used to
solve nonlinear regression problem

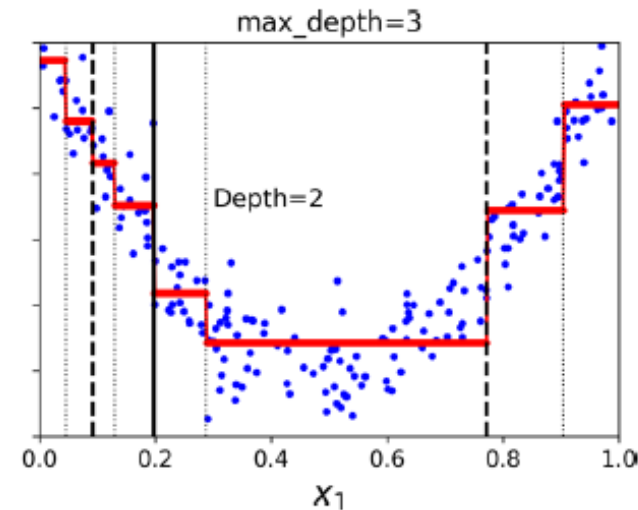
Tree-based regression

- The predicted value for each region is always the average target value of the instances in that region

Decision tree classification



Decision tree regression



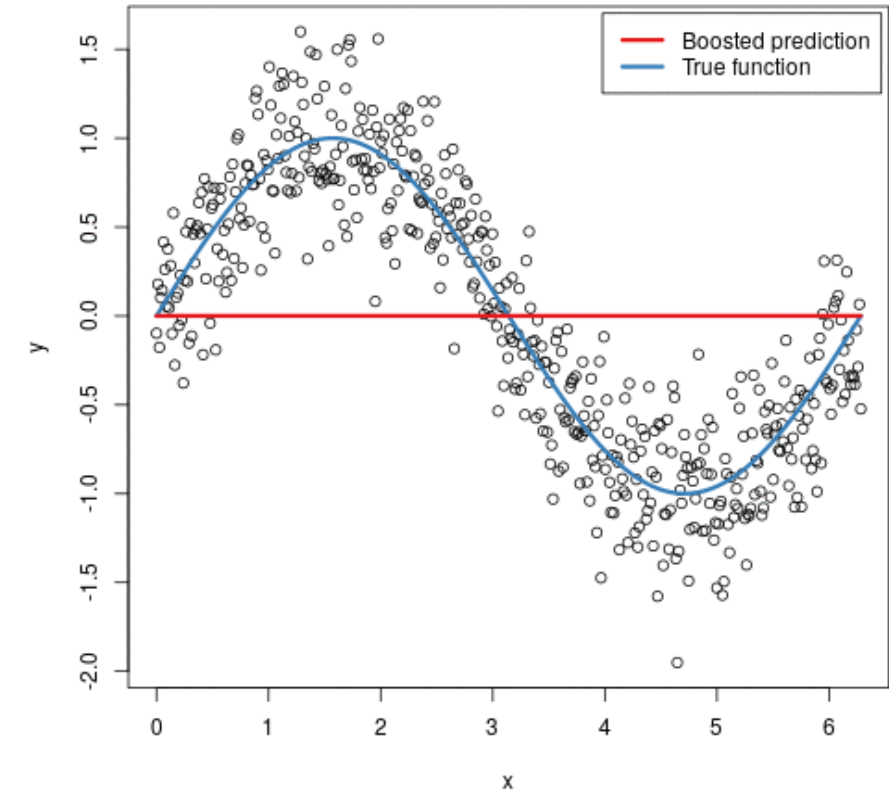
Equation 6-4. CART cost function for regression

$$J(k, t_k) = \frac{m_{\text{left}}}{m} \text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m} \text{MSE}_{\text{right}} \quad \text{where} \quad \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} (\hat{y}_{\text{node}} - y^{(i)})^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

Gradient Boosting Machines

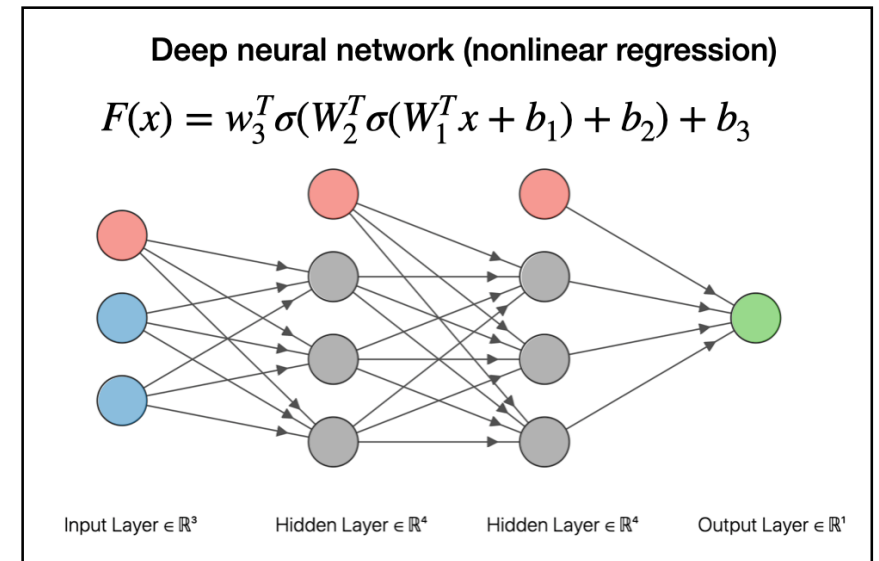
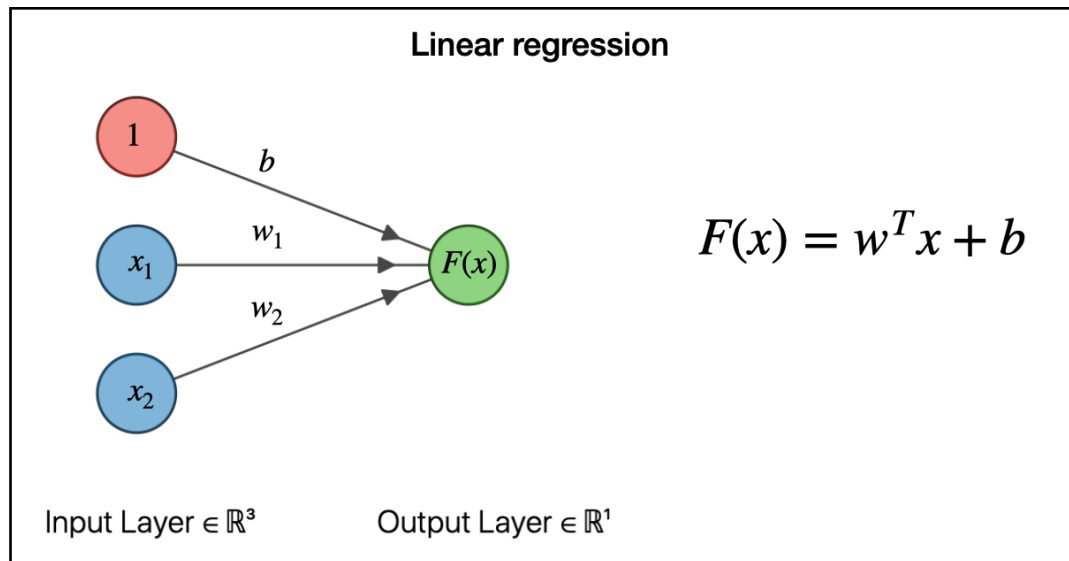
Use decision trees as base-learner, iteratively improves the weak learning model

- Fit a decision tree to the data $\hat{y} = F_1(x)$
- Fit the next decision tree to the residuals from previous tree $h_1(x) = y - F_1(x)$
- Add the new tree to the algorithm $F_2(x) = F_1(x) + h_1(x)$
- Repeat this process until some mechanism tells us to stop (e.g. cross validation)

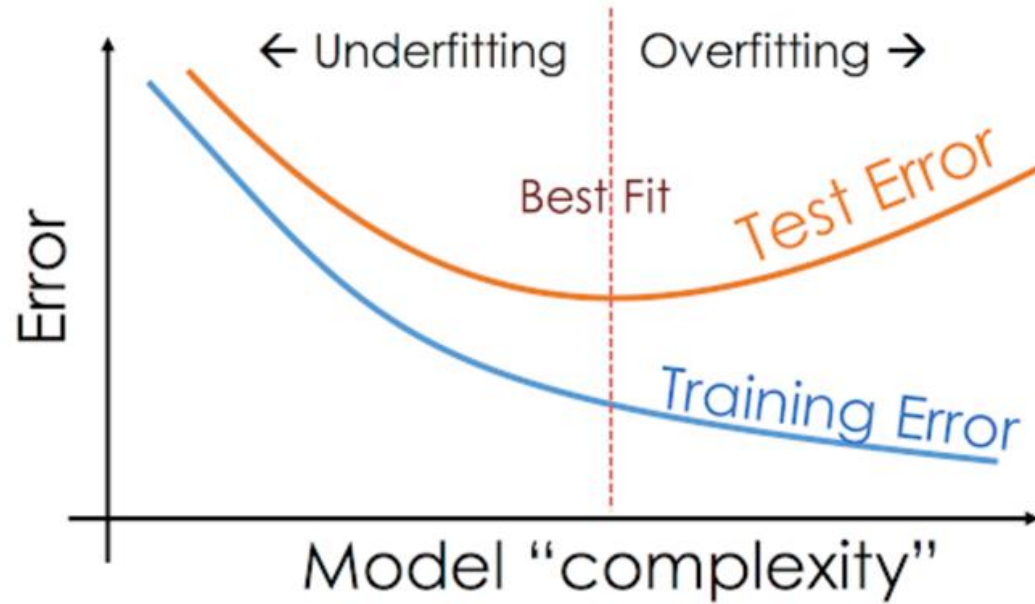


Neural Network

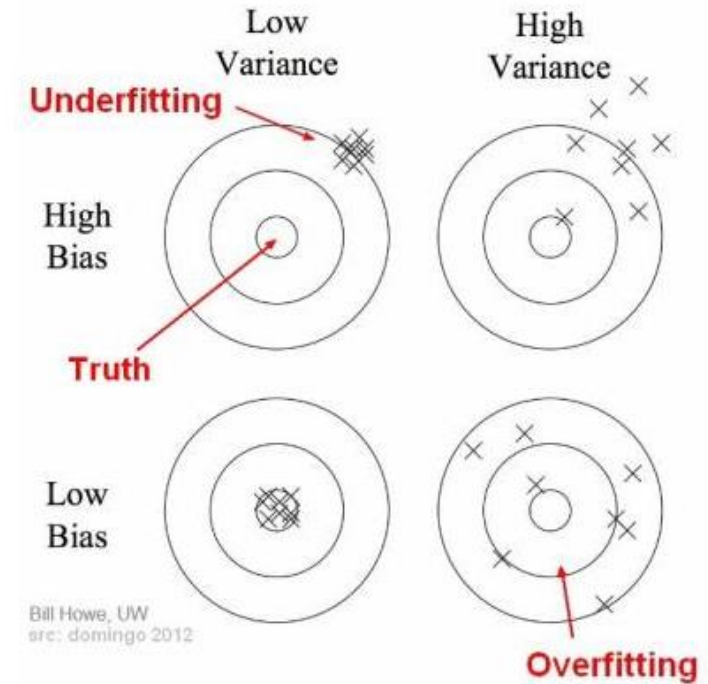
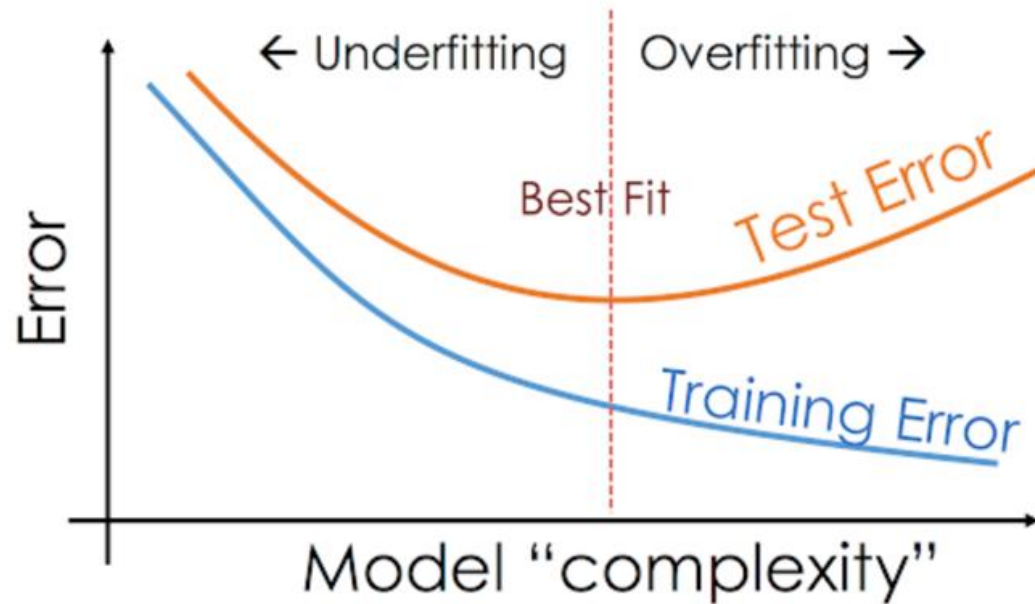
- A universal function approximator
 - Loss function: MSE
 - No sigmoid activation function at the output node



Over/underfit and bias-variance tradeoff



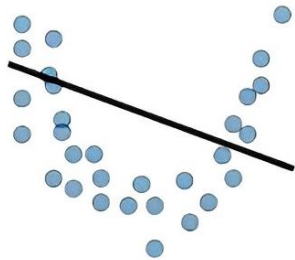
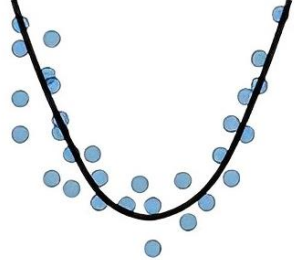
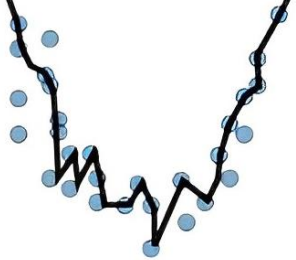
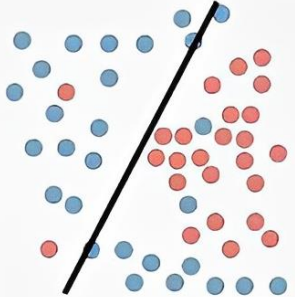
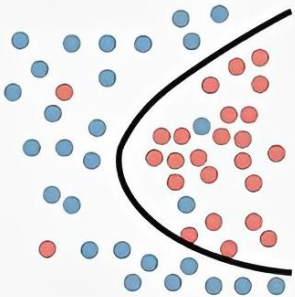
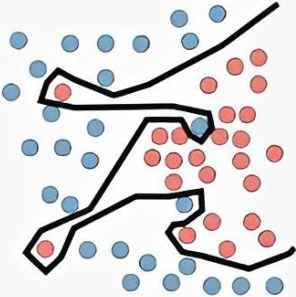
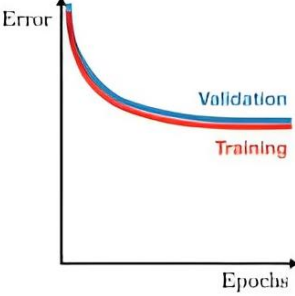
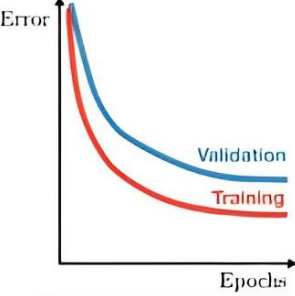
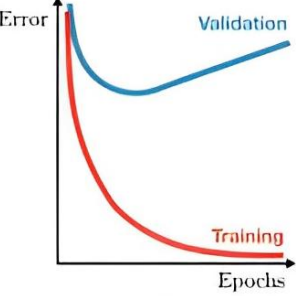
Over/underfit and bias-variance tradeoff



The bias-variance trade off

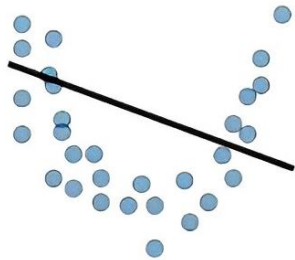
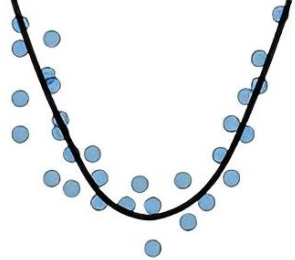
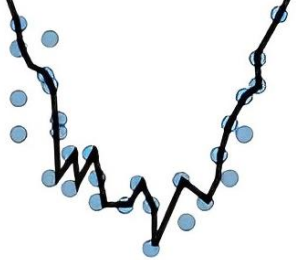
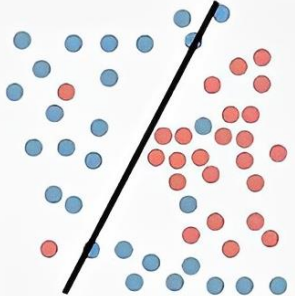
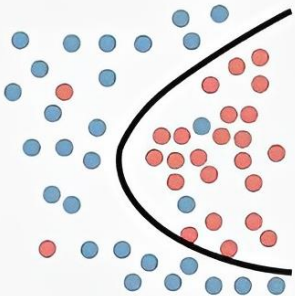
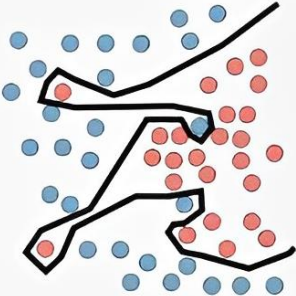
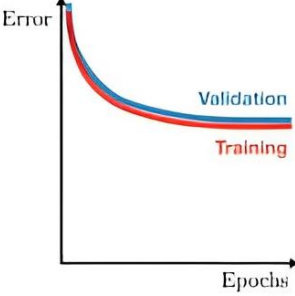
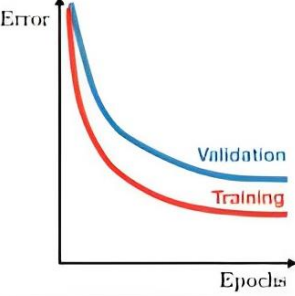
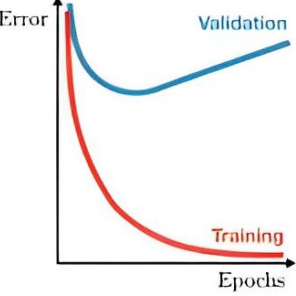
Underfit & Overfit

Symptom and solutions

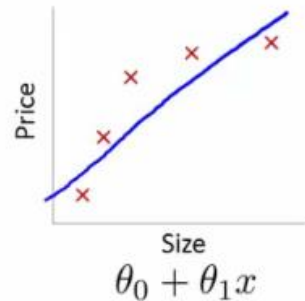
	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none">• High training error• Training error close to test error• High bias	<ul style="list-style-type: none">• Training error slightly lower than test error	<ul style="list-style-type: none">• Very low training error• Training error much lower than test error• High variance
Regression illustration			
Classification illustration			
Deep learning illustration	 <p>(too simple to explain the variance)</p>		 <p>(forcefitting -- too good to be true)</p>

Underfit & Overfit

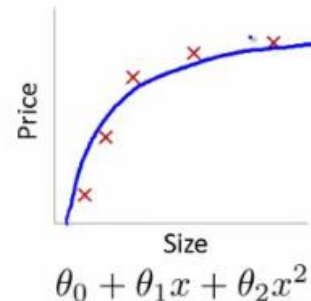
Symptom and solutions

	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none">• High training error• Training error close to test error• High bias	<ul style="list-style-type: none">• Training error slightly lower than test error	<ul style="list-style-type: none">• Very low training error• Training error much lower than test error• High variance
Regression illustration			
Classification illustration			
Deep learning illustration			
Possible remedies	<ul style="list-style-type: none">• Complexify model• Add more features• Train longer		<ul style="list-style-type: none">• Perform regularization• Get more data

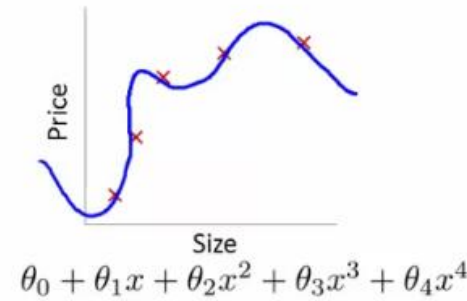
Regularization techniques in regression



High bias
(underfit)



"Just right"



High variance
(overfit)

- Regularization: to put constraints on the model
 - The fewer degrees of freedom, the harder it overfits
 - In reality, it's common to have $d \gg n$, regularization is usually needed

3 typical constraints in regression

- Lasso (L1 penalty)

$$J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + \alpha \sum_{i=1}^n |\theta_i|$$

- Ridge (L2 penalty)

$$J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + \alpha \frac{1}{2} \sum_{i=1}^n \theta_i^2$$

This forces the learning algorithm to not only fit the data but also keep the model weights as small as possible!

- Elastic-net

$$J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + r\alpha \sum_{i=1}^n |\theta_i| + \frac{1-r}{2}\alpha \sum_{i=1}^n \theta_i^2$$

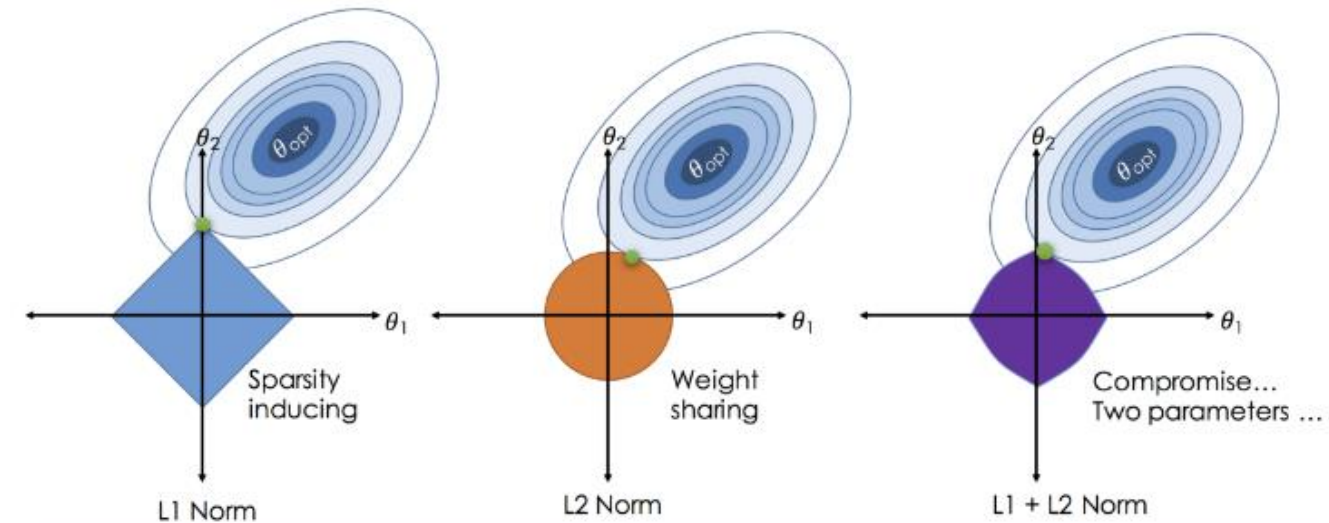
3 typical constraints in regression

- Lasso (L1 penalty)

$$J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + \alpha \sum_{i=1}^n |\theta_i|$$

- Ridge (L2 penalty)

$$J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + \alpha \frac{1}{2} \sum_{i=1}^n \theta_i^2$$



- Elastic-net

$$J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + r\alpha \sum_{i=1}^n |\theta_i| + \frac{1-r}{2}\alpha \sum_{i=1}^n \theta_i^2$$

Summary

- 6 regression algorithms
- MSE/MAE loss function
- Over/Underfit and Bias-variance tradeoff
- Regularization
 - LASSO
 - Ridge
 - Elastic net

Unsupervised learning

- Dimension reduction

- PCA
- t-SNE
- Autoencoder
- ...

- Clustering

- K-means
- Hierarchical
- DBSCAN
- ...



Dimension reduction

- High dimensional: a blessing and a curse

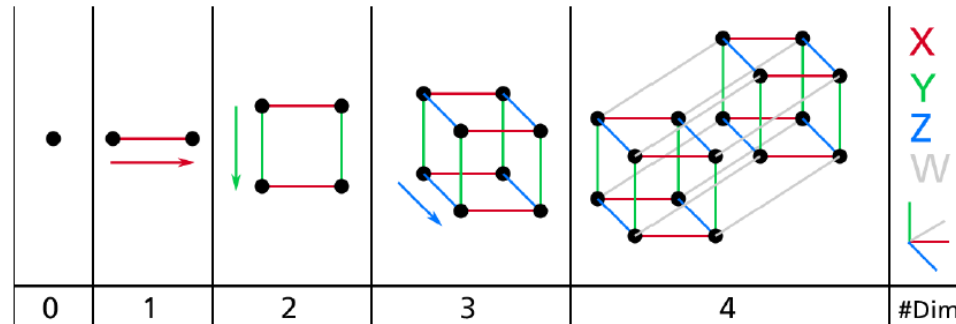


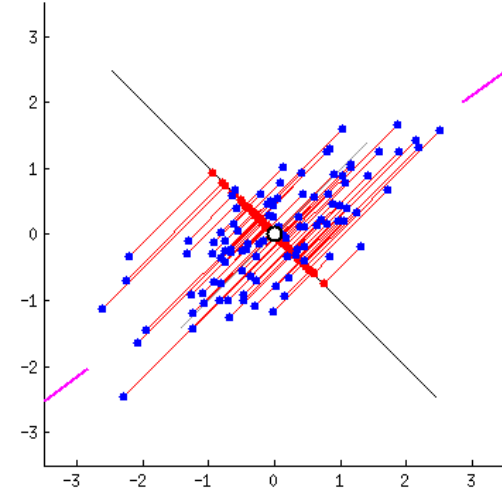
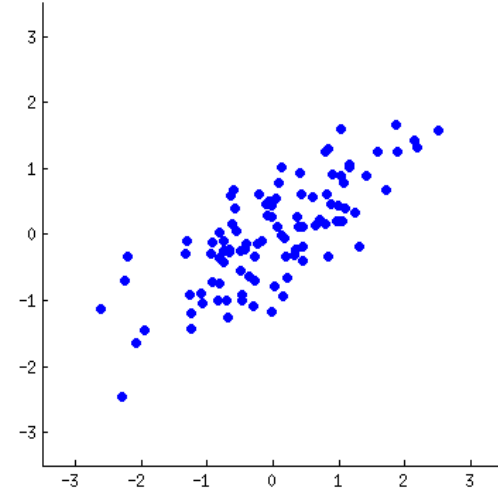
Figure 8-1. Point, segment, square, cube, and tesseract (0D to 4D hypercubes)²

- Curse of dimensionality:
 - Many machine learning algorithm have hard time to find good solutions in high dimensional setting
 - The training can be extremely slow when dimension (number of features) is high

Principal component analysis (PCA)

Preserving the maximum variance

- Data compression
- Noise removal
- Data visualization



Principal component analysis (PCA)

Preserving the maximum variance

- Data compression
- Noise removal
- Data visualization

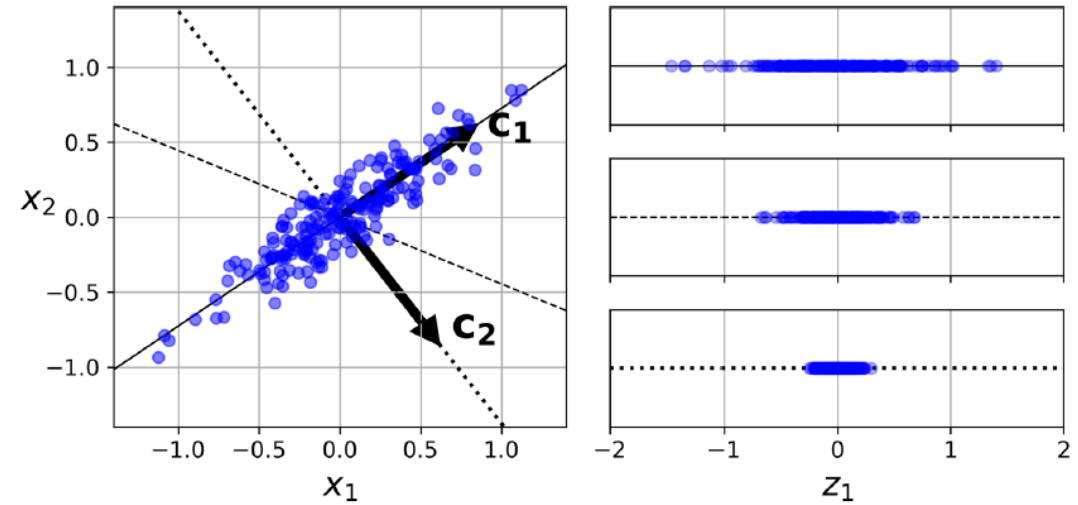


Figure 8-7. Selecting the subspace to project on

Principal component analysis (PCA)

Preserving the maximum variance

- Data compression
- Noise removal
- Data visualization

Goal:

- find an orthogonal set of r linear basis vectors $w_j \in R^d$ and the corresponding score $z_i \in R^r$, such that we minimize the average **reconstruction error**

$$J(\mathbf{W}, \mathbf{Z}) = \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2$$

$$\text{where } \hat{\mathbf{x}}_i = \mathbf{W}\mathbf{z}_i$$

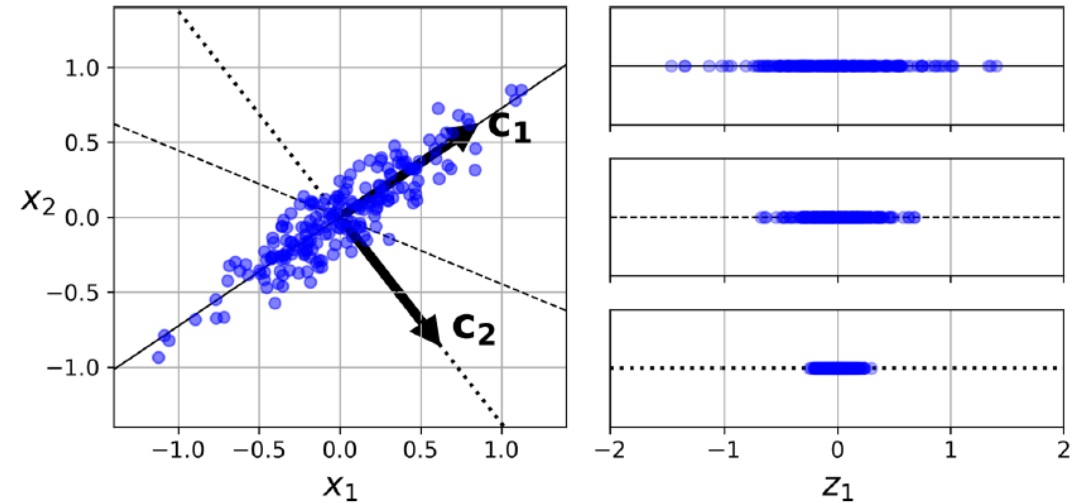
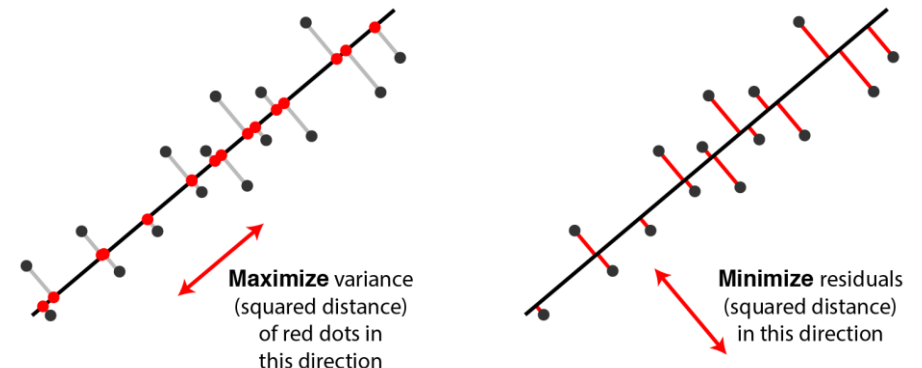
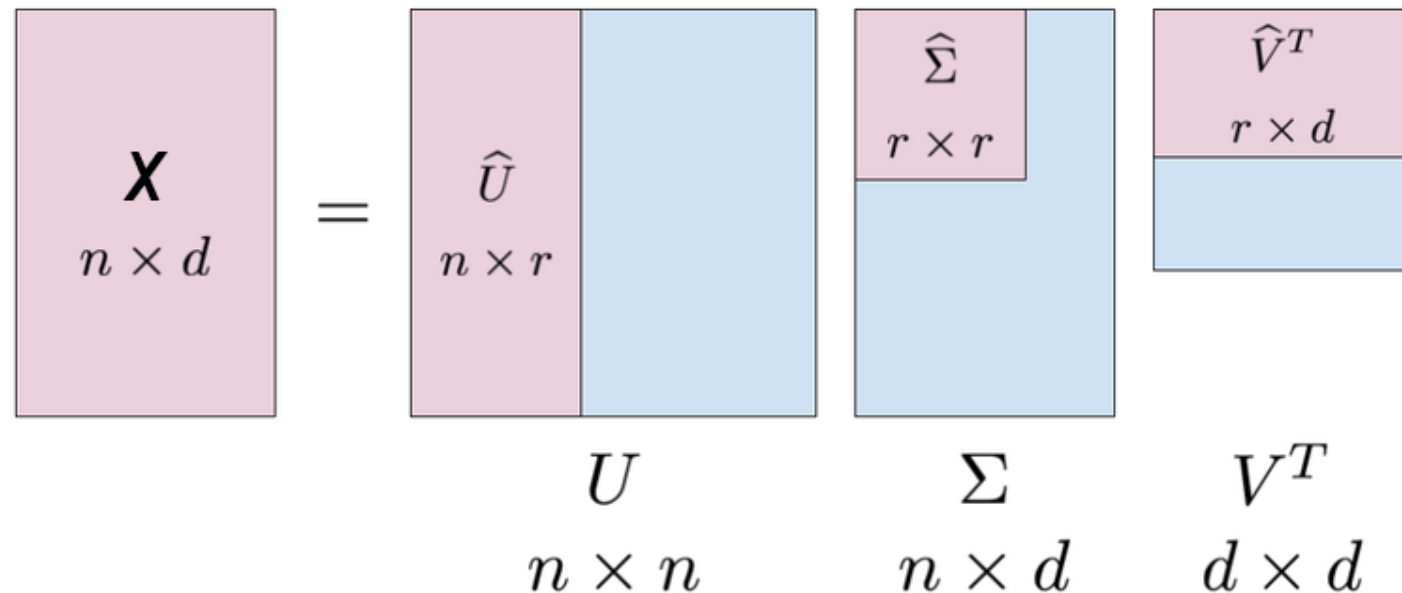


Figure 8-7. Selecting the subspace to project on



Principal component analysis (PCA)

- PCA achieved by Singular value decomposition (SVD)



The diagram illustrates the Singular Value Decomposition (SVD) of a matrix X of size $n \times d$. The matrix X is shown as a pink rectangle on the left. It is equal to the product of three matrices: \hat{U} (pink rectangle, $n \times r$), $\hat{\Sigma}$ (pink rectangle, $r \times r$), and \hat{V}^T (pink rectangle, $r \times d$). The matrix \hat{U} is labeled U and $n \times n$ below it. The matrix $\hat{\Sigma}$ is labeled Σ and $n \times d$ below it. The matrix \hat{V}^T is labeled V^T and $d \times d$ below it. The matrices \hat{U} and \hat{V}^T are shown as pink rectangles, while $\hat{\Sigma}$ is shown as a pink rectangle with a blue rectangle below it, representing the zero singular values.

$$\begin{matrix} \boxed{\begin{matrix} X \\ n \times d \end{matrix}} = \boxed{\begin{matrix} \hat{U} \\ n \times r \end{matrix}} \boxed{\begin{matrix} \hat{\Sigma} \\ r \times r \end{matrix}} \boxed{\begin{matrix} \hat{V}^T \\ r \times d \end{matrix}} \\ \begin{matrix} U \\ n \times n \end{matrix} \quad \begin{matrix} \Sigma \\ n \times d \end{matrix} \quad \begin{matrix} V^T \\ d \times d \end{matrix} \end{matrix}$$

$\hat{U} \hat{\Sigma}$: Principal component scores
 \hat{V} : Principal directions

Principal component analysis (PCA)

- PCA achieved by Singular value decomposition (SVD)

The diagram illustrates the Singular Value Decomposition (SVD) of a matrix X . On the left is a pink box representing X with dimensions $n \times d$. This is followed by an equals sign. To the right of the equals sign are three matrices: U , Σ , and V^T . U is a pink box with dimensions $n \times r$ and a label U and $n \times n$ below it. Σ is a light blue box with dimensions $n \times d$ and a label Σ and $n \times d$ below it. V^T is a pink box with dimensions $d \times d$ and a label V^T and $d \times d$ below it. Inside the Σ box, there is a smaller pink box labeled $\hat{\Sigma}$ with dimensions $r \times r$. Inside the V^T box, there is a smaller pink box labeled \hat{V}^T with dimensions $r \times d$.

$$\begin{matrix} \boxed{\begin{matrix} X \\ n \times d \end{matrix}} = \boxed{\begin{matrix} \hat{U} \\ n \times r \end{matrix}} \begin{matrix} \boxed{\begin{matrix} \hat{\Sigma} \\ r \times r \end{matrix}} \\ \text{light blue box} \end{matrix} \begin{matrix} \boxed{\begin{matrix} \hat{V}^T \\ r \times d \end{matrix}} \\ \text{light blue box} \end{matrix} \\ \begin{matrix} U \\ n \times n \end{matrix} \quad \begin{matrix} \Sigma \\ n \times d \end{matrix} \quad \begin{matrix} V^T \\ d \times d \end{matrix} \end{matrix}$$

- Project data on to the reduced dimension space

$$X_{d-proj} = X\hat{V}$$

t-Distributed Stochastic Neighbor Embedding (t-SNE)

- Nonlinear dimensionality reduction
 - PCA uses the global covariance matrix
 - t-SNE focus more on the local structure

- Core algorithm

- In high dimensional space

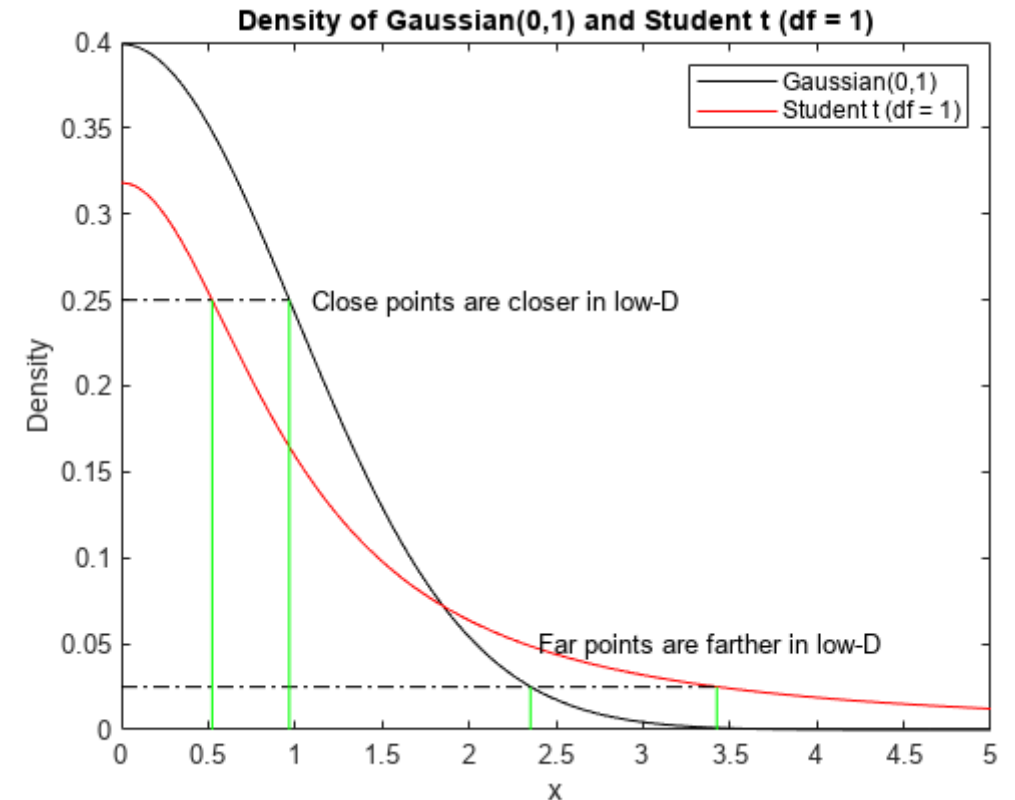
$$p_{ij} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq l} \exp(-\|x_k - x_l\|^2 / 2\sigma_i^2)}$$

- In lower dimensional space

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}}$$

- Try to minimize the difference of 2 distributions

$$C = D_{\text{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$



t-Distributed Stochastic Neighbor Embedding (t-SNE)

- Nonlinear dimensionality reduction
 - PCA uses the global covariance matrix
 - t-SNE focus more on the local structure

- Core algorithm

- In high dimensional space

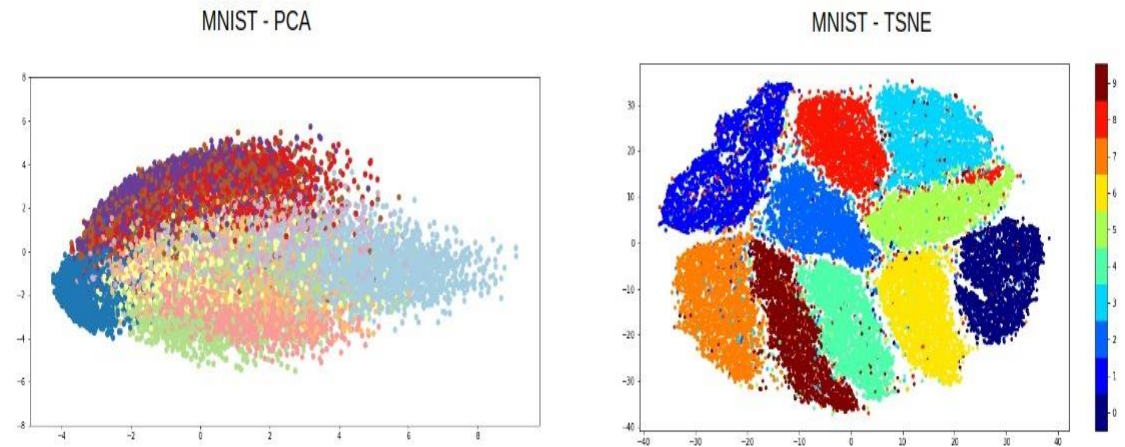
$$p_{ij} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq l} \exp(-\|x_k - x_l\|^2 / 2\sigma_i^2)}$$

- In lower dimensional space

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}}$$

- Try to minimize the difference of 2 distributions

$$C = D_{\text{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$



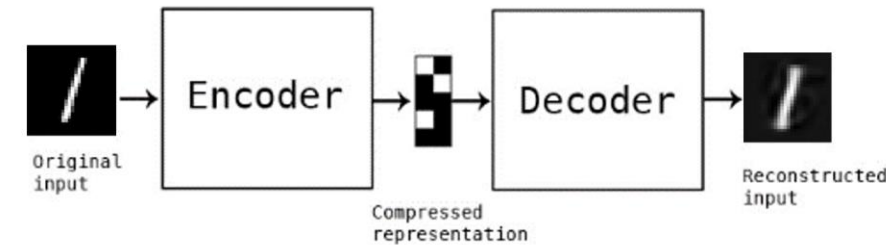
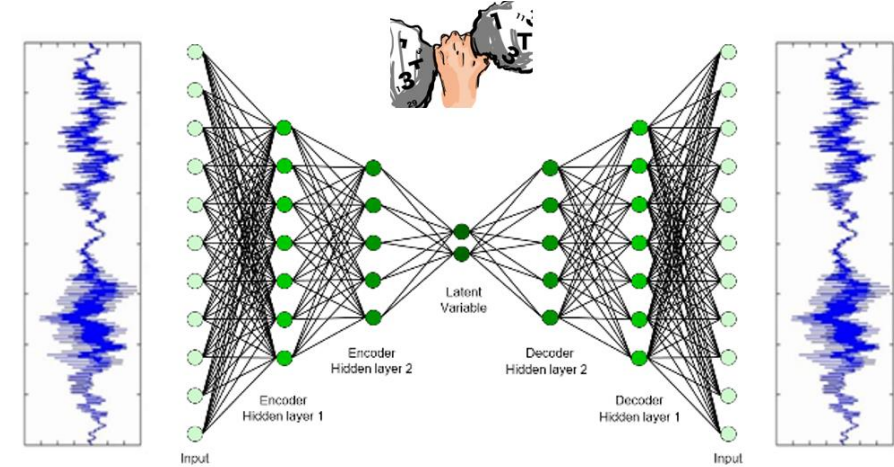
Autoencoder

- Nonlinear dimension reduction with NN



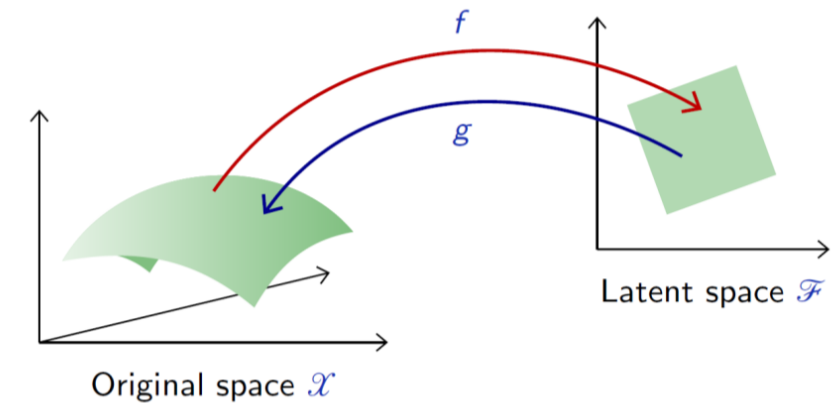
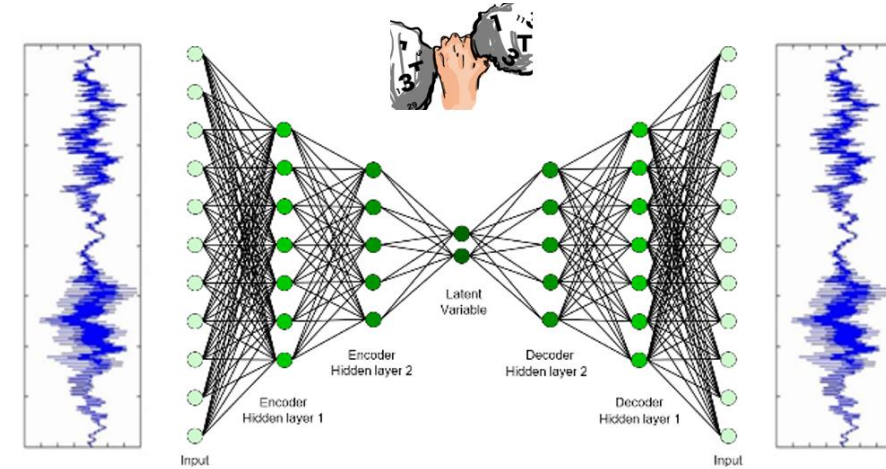
Autoencoder

- Nonlinear dimension reduction with NN
- Key idea:
 - An encoder function $z = f(x)$
 - A Decoder function $x = g(z)$
 - Learn to set $g(f(x)) \cong x$



Autoencoder

- Nonlinear dimension reduction with NN
- Key idea:
 - An encoder function $z = f(x)$
 - A Decoder function $x = g(z)$
 - Learn to set $g(f(x)) \cong x$
- Loss function: reconstruction error
 - Minimize $\|X - g \circ f(X)\|^2$



Clustering

The vast majority of data is unlabeled. The clustering algorithm tries to identify similar instances and assigning them to *clusters*

- K-means
- Hierarchical clustering
- DBSCAN



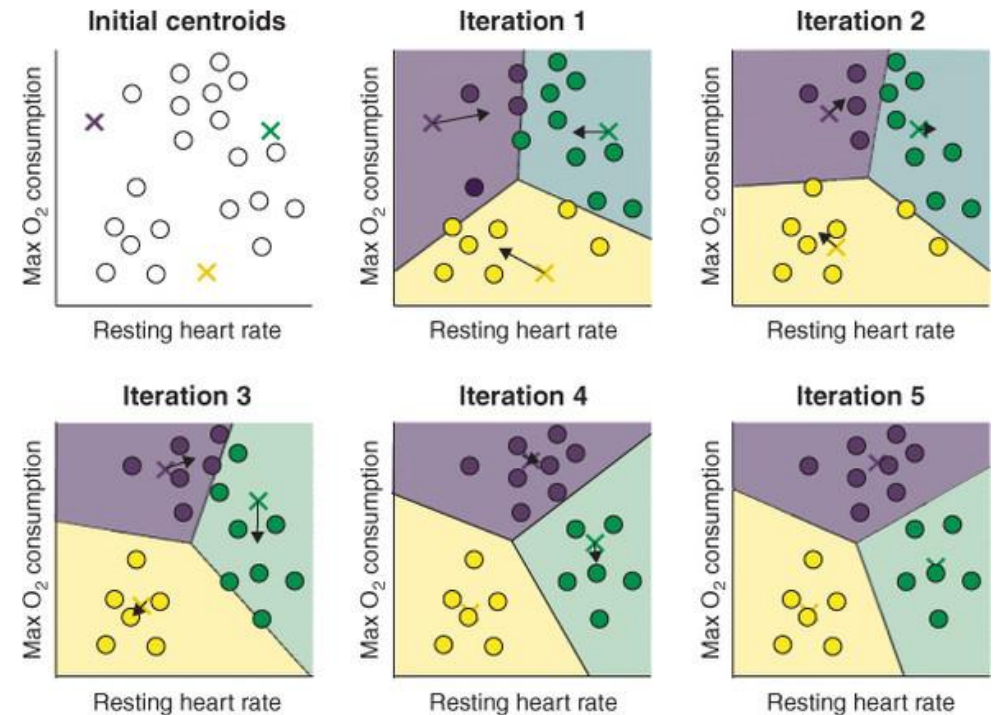
K-means

- look for instances centered around a particular point (centroid)

- Core algorithm

Algorithm 1 k -means algorithm

- 1: Specify the number k of clusters to assign.
 - 2: Randomly initialize k centroids.
 - 3: **repeat**
 - 4: **expectation:** Assign each point to its closest centroid.
 - 5: **maximization:** Compute the new centroid (mean) of each cluster.
 - 6: **until** The centroid positions do not change.
-



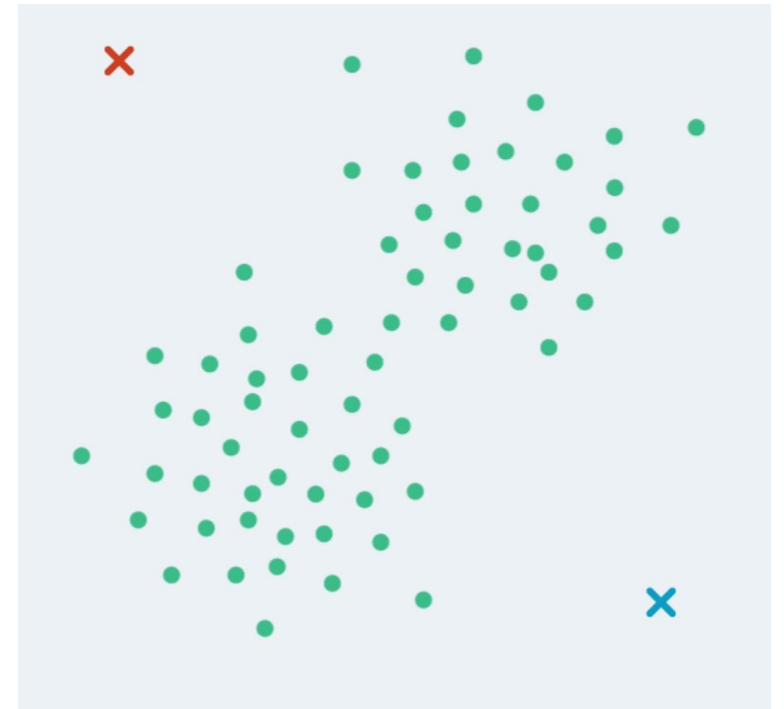
K-means

- look for instances centered around a particular point (centroid)
- Core algorithm

Algorithm 1 k -means algorithm

- 1: Specify the number k of clusters to assign.
 - 2: Randomly initialize k centroids.
 - 3: **repeat**
 - 4: **expectation:** Assign each point to its closest centroid.
 - 5: **maximization:** Compute the new centroid (mean) of each cluster.
 - 6: **until** The centroid positions do not change.
-

The algorithm is guaranteed to converge in a finite number of steps (usually quite small).



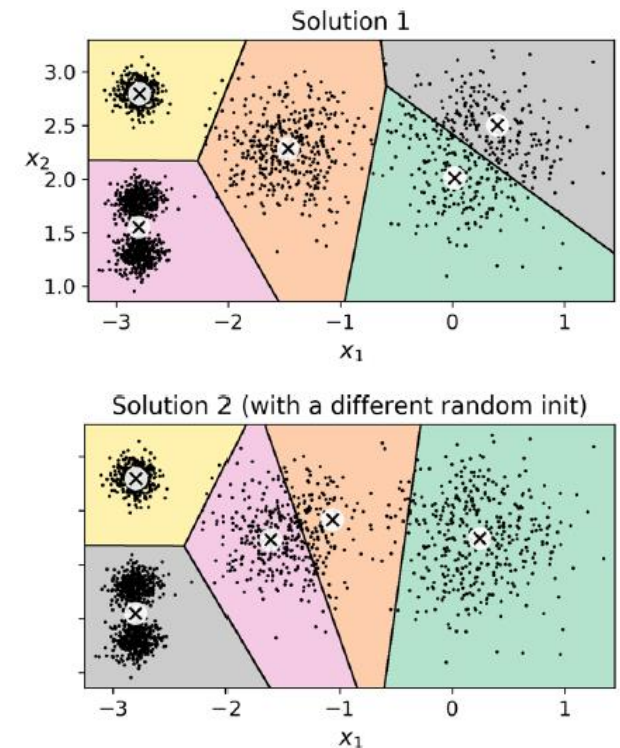
K-means

- look for instances centered around a particular point (centroid)
- Core algorithm

Algorithm 1 k -means algorithm

- 1: Specify the number k of clusters to assign.
 - 2: Randomly initialize k centroids.
 - 3: **repeat**
 - 4: **expectation:** Assign each point to its closest centroid.
 - 5: **maximization:** Compute the new centroid (mean) of each cluster.
 - 6: **until** The centroid positions do not change.
-

The algorithm is guaranteed to converge in a finite number of steps (usually quite small). But it might not converge to a right solution (local optimum)



K-means

- look for instances centered around a particular point (centroid)
- Core algorithm

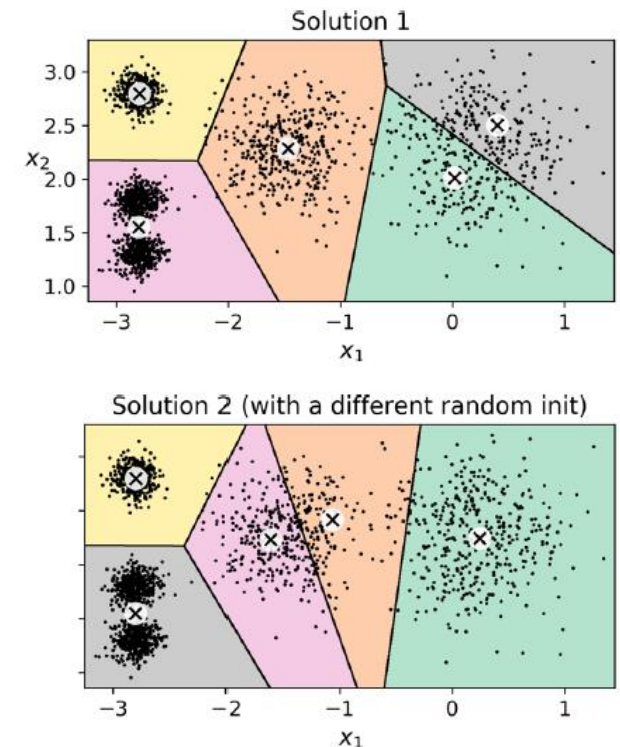
Algorithm 1 k -means algorithm

- 1: Specify the number k of clusters to assign.
 - 2: Randomly initialize k centroids.
 - 3: **repeat**
 - 4: **expectation:** Assign each point to its closest centroid.
 - 5: **maximization:** Compute the new centroid (mean) of each cluster.
 - 6: **until** The centroid positions do not change.
-

The algorithm is guaranteed to converge in a finite number of steps (usually quite small). But it might not converge to a right solution (local optimum)

Solution:

- If you know the approximate position of centroids, manually set them during initialization
- Run the algorithm multiple times with different random initialization, keep the best solution



K-means

- look for instances centered around a particular point (centroid)
- Core algorithm

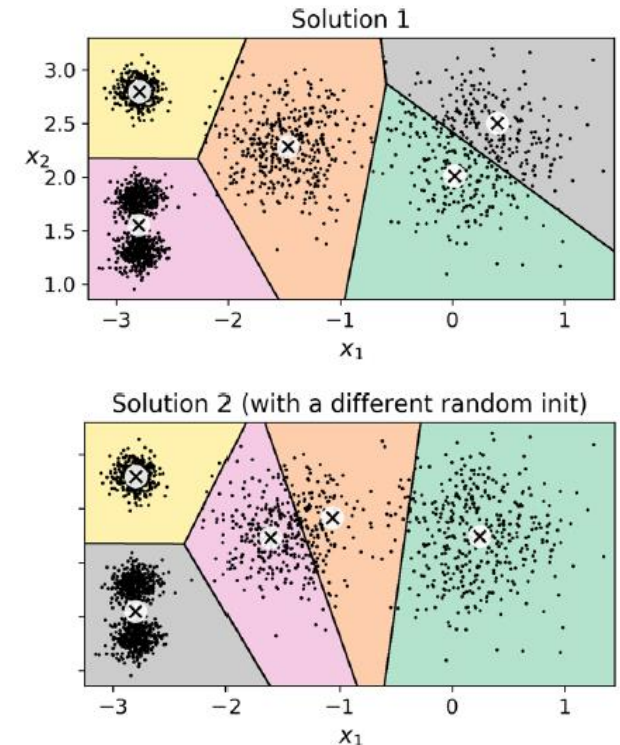
Algorithm 1 k -means algorithm

- 1: Specify the number k of clusters to assign.
 - 2: Randomly initialize k centroids.
 - 3: **repeat**
 - 4: **expectation:** Assign each point to its closest centroid.
 - 5: **maximization:** Compute the new centroid (mean) of each cluster.
 - 6: **until** The centroid positions do not change.
-

The algorithm is guaranteed to converge in a finite number of steps (usually quite small). But it might not converge to a right solution (local optimum)

Note:

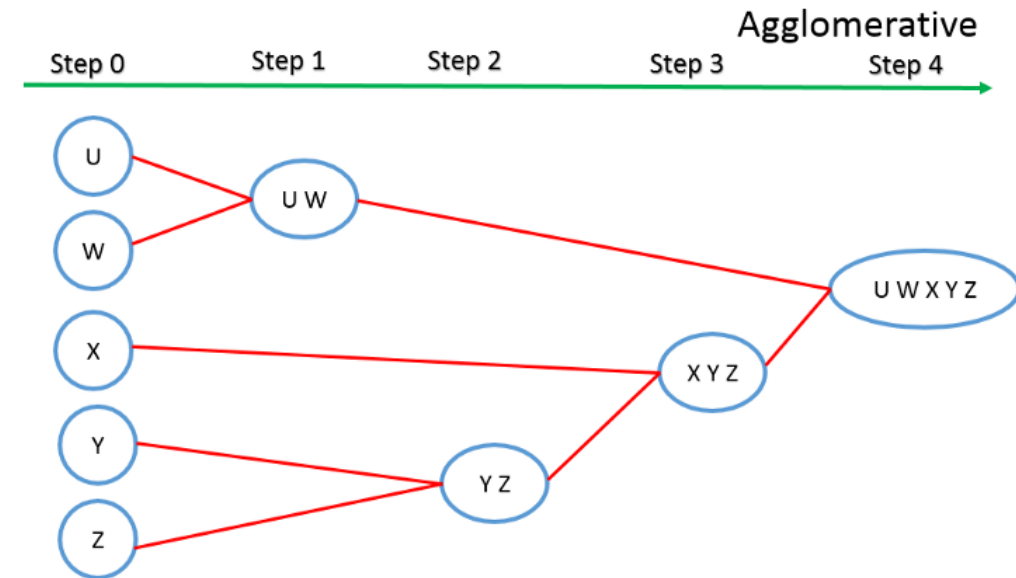
- K-means favor cluster with similar size, it doesn't perform well with varying sizes
- Need to specify the number of clusters



Hierarchical (Agglomerative) clustering

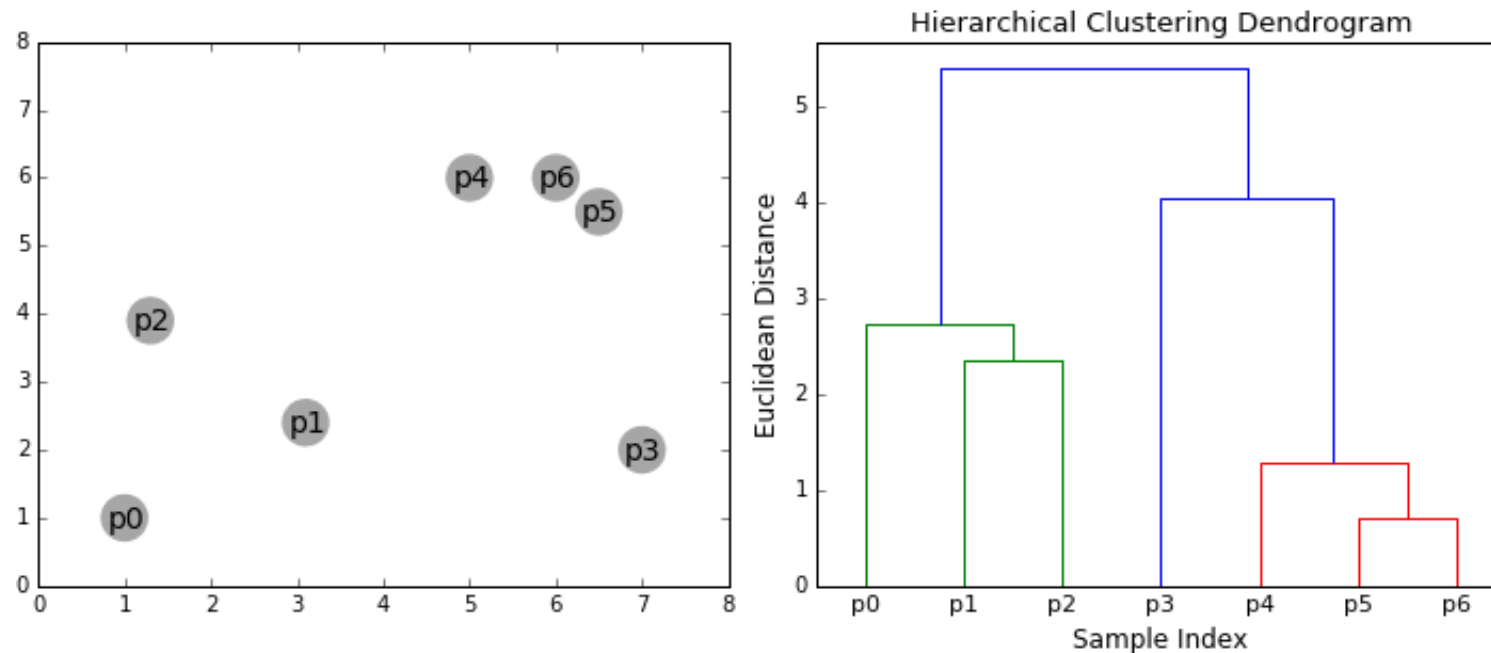
- A bottom-Up approach to connect the nearest pair of clusters

$t = 0$
Choose $R_0 = [C_i = x_i, i = 1, \dots, N]$ as initial clustering
Repeat
 $t = t + 1$
 Find the closest clusters C_i, C_j in the existing clustering R_{t-1} such that
 $g(C_i, C_j) = \max_{r,s}(C_r, C_s)$ if g is similarity function
 $g(C_i, C_j) = \min_{r,s}(C_r, C_s)$ if g is dissimilarity function
 Define $C_q = C_i \cup C_j$ and produce the new clustering $R_t = [R_{t-1} - C_i - C_j] \cup C_q$
Until only one cluster is left



Hierarchical (Agglomerative) clustering

- A bottom-Up approach to connect the nearest pair of clusters



Scale up well to large number of instance or clusters

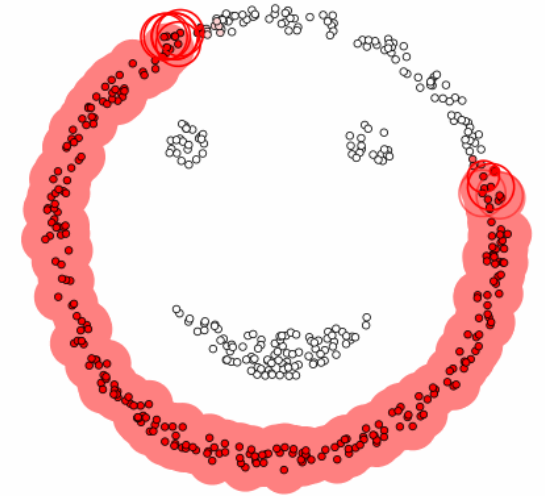
DBSCAN

- defines clusters as continuous regions of high density
- Core algorithm

Algorithm 3 : DBSCAN Clustering

Input: *2D_Data* obtained by *Algorithm1* as the input data, $|Data|$ objects to be clustered, the neighborhood radius (ϵ) and minimum points (μ)

- 1: Randomly select a point P
 - 2: Retrieve all points density-reachable from P based on ϵ and μ and Similarity Metric (*Algorithm2*)
 - 3: If P is a core point, a cluster is formed.
 - 4: If P is a border point, no points are density-reachable from P and DBSCAN selects the next no-visited point randomly.
 - 5: Continue the procedure until all points have been processed.
-



- Works pretty well if clusters are dense enough and separated well by low-density regions
- Robust to outliers
- Computation complexity is $O(m \log m)$

Summary

We have knowledge about

- ❑ Machine learning's definition and categories
- ❑ The workflow for train a machine learning model
- ❑ Rationale of several major machine learning algorithms!
- ❑ Performance measure for evaluating models
- ❑ Challenges in machine learning and potential solutions

And we have experience in

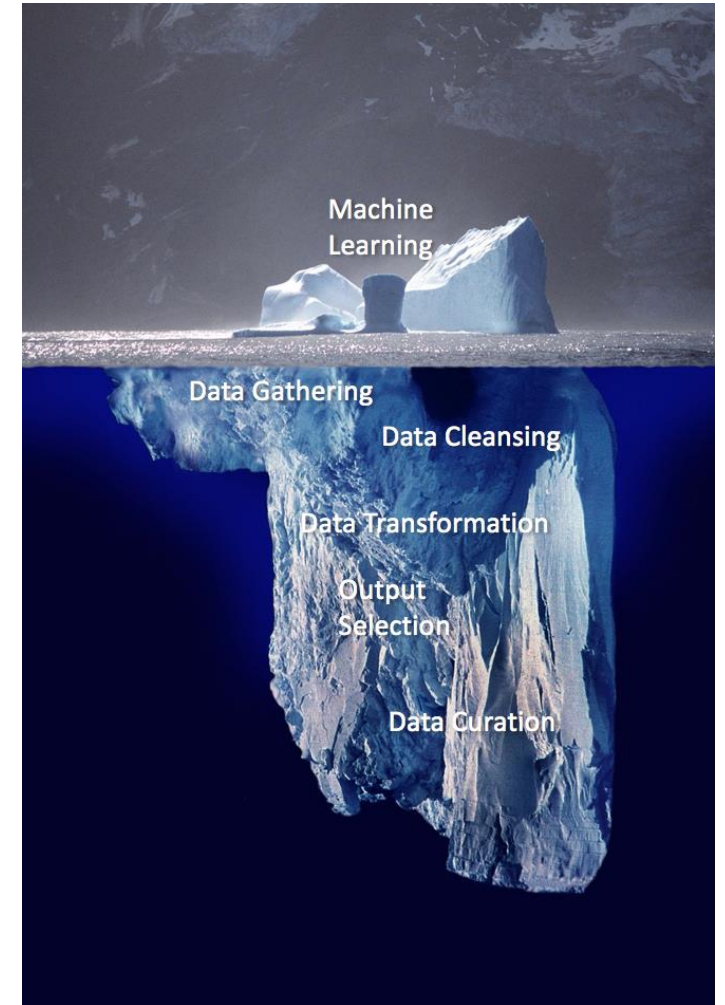
- ❑ Jupyter notebook
- ❑ NumPy, matplotlib, scikit-learn, keras
- ❑ Build, train, evaluate a classifier or regressor
- ❑ Tune hyperparameters
- ❑ Unsupervised learning (PCA, K-Means)

Summary of algorithms

Supervised learning		Unsupervised learning	
Classification	Regression	Dimension reduction	Clustering
Logistic regression	Linear regression	PCA	K-means
KNN	Polynomial regression	t-SNE	Hierarchical
Naïve bayes	SVR	Autoencoder	DBSCAN
SVM	Tree-based		
Decision tree	GBM		
Random forest	Neural Network		
Adaboost			
Gradient boosting			
Neural network			

Beyond machine learning algorithms

- Problem formulation
- Data cleansing
- Feature engineering
 - Feature encoding
 - Imputation: Missing data handling
 - Transformation/Normalization/Standardization
 - ...

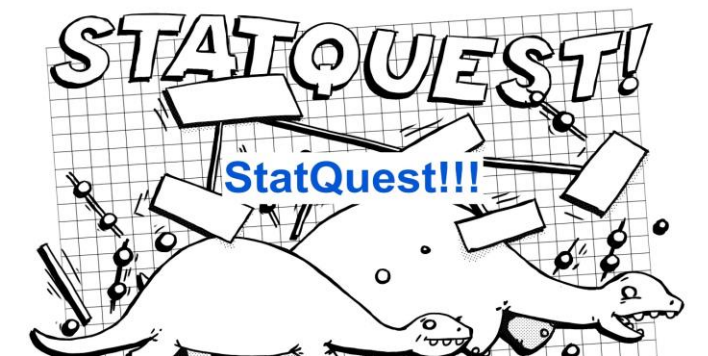


Where to get help?

- <https://www.google.com>
- <https://stackoverflow.com>
- <https://stats.stackexchange.com/>
- <https://towardsdatascience.com/>
- <https://www.3blue1brown.com>
- <https://statquest.org/>



towards
data science



Lastly, GLHF!



“good luck, have fun with
machine learning”



Wenbin × DALL-E
Human & AI

Q&A