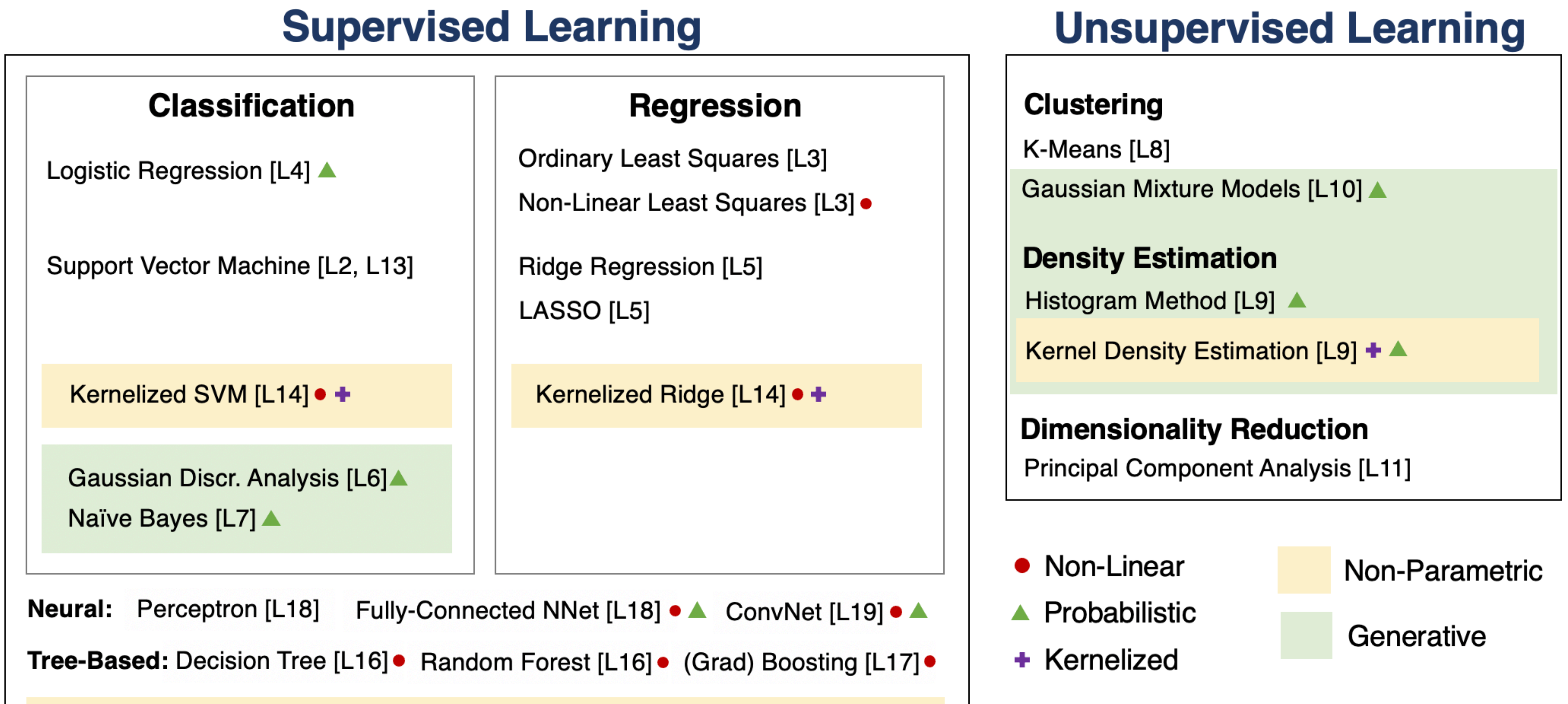
Lecture 24: Course Overview

A Map of Applied Machine Learning

We will go through the following map of algorithms from the course.



Supervised Learning

Classification

Logistic Regression [L4] ▲

Support Vector Machine [L2, L13]

Kernelized SVM [L14] ● ✚

Gaussian Discr. Analysis [L6] ▲

Naïve Bayes [L7] ▲

Regression

Ordinary Least Squares [L3]

Non-Linear Least Squares [L3] ●

Ridge Regression [L5]

LASSO [L5]

Kernelized Ridge [L14] ● ✚

Neural: Perceptron [L18] Fully-Connected NNet [L18] ● ▲ ConvNet [L19] ● ▲

Tree-Based: Decision Tree [L16] ● Random Forest [L16] ● (Grad) Boosting [L17] ●

Unsupervised Learning

Clustering

K-Means [L8]

Gaussian Mixture Models [L10] ▲

Density Estimation

Histogram Method [L9] ▲

Kernel Density Estimation [L9] ✚ ▲

Dimensionality Reduction

Principal Component Analysis [L11]

●	Non-Linear	🟨	Non-Parametric
▲	Probabilistic	🟩	Generative
✚	Kernelized		

Key Idea: Supervised Learning Recipe

To apply supervised learning, we define a dataset and a learning algorithm.

$$\underbrace{\text{Dataset}}_{\text{Features, Attributes, Targets}} + \underbrace{\text{Learning Algorithm}}_{\text{Model Class + Objective + Optimizer}} \to \text{Model}$$

Example: Linear Regression

In linear regression, we fit a model

$$f_\theta(x) := \theta^\top \phi(x)$$

that is linear in θ.

The features $\phi(x) : \mathbb{R} \to \mathbb{R}^p$ may be non-linear in x (e.g., polynomial).

We define the least squares objective for the model as:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \theta^\top x^{(i)})^2 = \frac{1}{2}(X\theta - y)^\top (X\theta - y)$$

We can set the gradient to zero to obtain the *normal equations*:

$$(X^\top X)\theta = X^\top y.$$

Hence, the value θ^* that minimizes this objective is given by:

$$\theta^* = (X^\top X)^{-1} X^\top y.$$

Supervised Learning

Regression

Ordinary Least Squares [L3]

Non-Linear Least Squares [L3] ●

● Non-Linear

Key Ideas: Overfitting & Underfitting

Overfitting is one of the most common failure modes of machine learning.

- An expressive model (a high degree polynomial) fits the training data perfectly.
- It makes incorrect predictions on held-out data (doesn't generalize).

How to avoid overfitting:

- Use a **simpler** model family (linear models vs. neural nets)
- Keep the same model, but **collect more training data**
- Use **regularization** to penalize overly complex models.

Key Idea: Regularization

Regularization penalizes complex models that may overfit the data.

Regularized least squares optimizes the following objective (**Ridge**).

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \left(y^{(i)} - \theta^\top x^{(i)} \right)^2 + \frac{\lambda}{2} \cdot ||\theta||_2^2.$$

If we use the L1 norm, we have the **LASSO**.

Supervised Learning

Regression

Ordinary Least Squares [L3]

Non-Linear Least Squares [L3] ●

Ridge Regression [L5]

LASSO [L5]

● Non-Linear

Regression vs. Classification

Consider a training dataset $\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})\}$.

We distinguish between two types of supervised learning:

1. **Regression**: The target variable $y \in \mathcal{Y}$ is continuous.
2. **Classification**: The target variable y is discrete and takes on one of K possible values (classes): $\mathcal{Y} = \{y_1, y_2, \ldots y_K\}$.

Example: Logistic Regression

Logistic regression fits models of the form:

$$f(x) = \sigma(\theta^\top x) = \frac{1}{1 + \exp(-\theta^\top x)},$$

where

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

is known as the *sigmoid* or *logistic* function.

Example: K-Nearest Neighbors

Suppose we receive a query point x' and we want to predict its label y'.

1. Given a query x', find the K training examples $\mathcal{N} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(K)}, y^{(K)})\}$ closest to x'.
2. Return $y_{\mathcal{N}}$, the consensus label of the neighborhood $\mathcal{N}$.

Key Idea: Parametric vs. Non-Parametric Models

Nearest neighbors is an example of a *non-parametric* model.

- A parametric model has a finite set of parameters $\theta \in \Theta$ whose dimensionality is constant with data

- In a non-parametric model, the function f uses the entire training dataset to make predictions, and the complexity of the model increases with dataset size.

Supervised Learning

Classification

Logistic Regression [L4] ▲

Regression

Ordinary Least Squares [L3]

Non-Linear Least Squares [L3] ●

Ridge Regression [L5]

LASSO [L5]

Nearest Neighbors [L9] ●

● Non-Linear

▲ Probabilistic

🟨 Non-Parametric

Key Idea: Probabilistic Models

A probabilistic discriminative model outputs a vector of class probabilities

$$\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\text{input}} \rightarrow \text{model } P_\theta(y|x) \rightarrow \underbrace{\begin{bmatrix} 0.1 \\ 0.7 \\ 0.2 \end{bmatrix}}_{\text{output}}$$

The model takes x to be a fixed input.

For example, a logistic (softmax) model outputs a probability distribution

$$P_\theta(y|x) = \begin{bmatrix} P_\theta(y=0|x) \\ P_\theta(y=1|x) \end{bmatrix} = \begin{bmatrix} 1-\sigma(\theta^\top x) \\ \sigma(\theta^\top x) \end{bmatrix}$$

where $\theta^\top x$ is a linear model and

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

is the *sigmoid* or *logistic* function.

Key Idea: Discriminative vs. Generative Models

Another approach to classification is to use *generative* models.

- A generative approach first builds a model of x for each class:

$$P_\theta(x|y=k) \text{ for each class } k.$$

  $P_\theta(x|y=k)$ *scores* x according to how well it matches class k.

- A prior class probability $P_\theta(y=k)$.

In spam classification, we fit two models on spam/non-spam emails x:

$$P_\theta(x|y=0) \qquad \text{and} \qquad P_\theta(x|y=1)$$

- $P_\theta(x|y=1)$ *scores* x based on how much it looks like spam.
- $P_\theta(x|y=0)$ *scores* x based on whether it looks like non-spam.

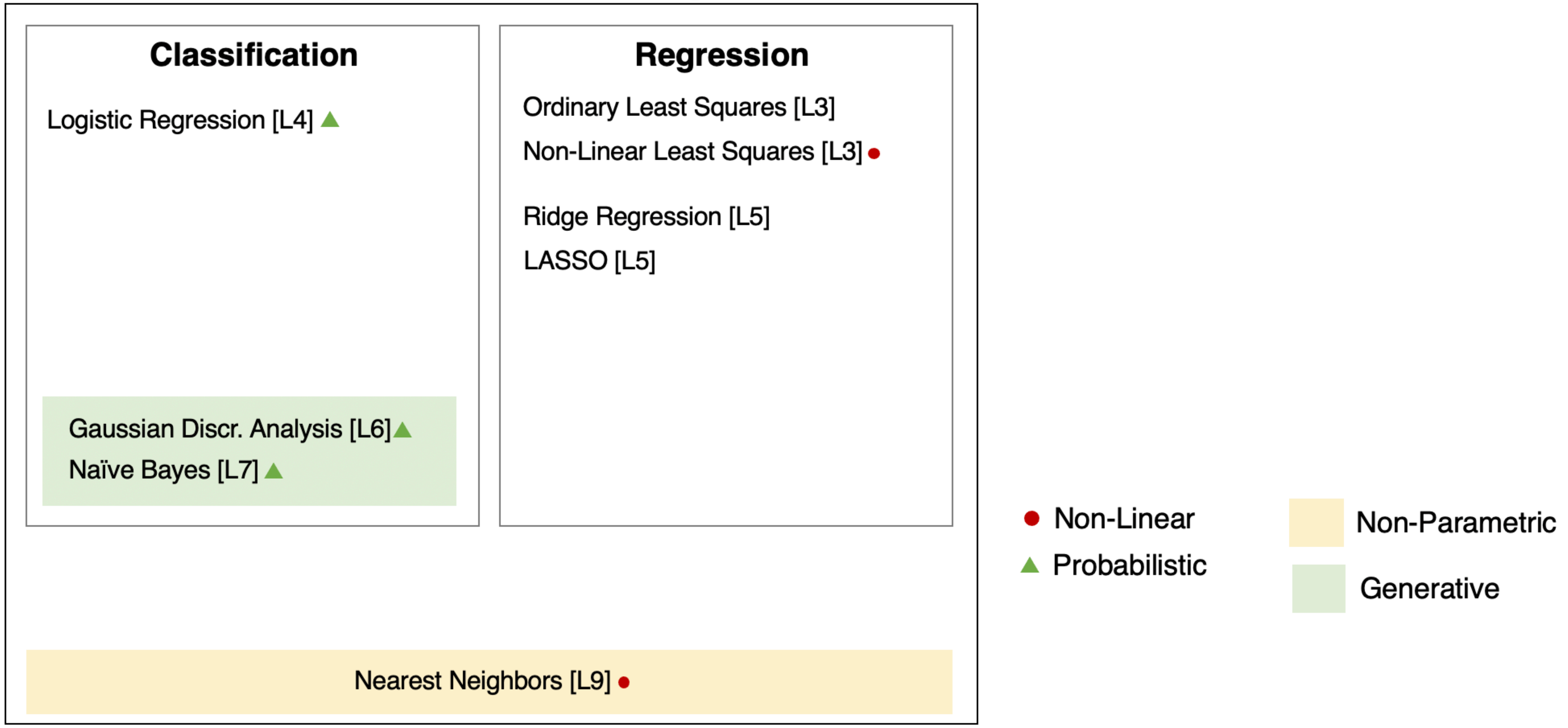
We also choose a prior $P(y)$.

Given a new x', we would compare the probabilities of both models:

$$P_\theta(x'|y=0)P_\theta(y=0) \qquad \text{vs.} \qquad P_\theta(x'|y=1)P_\theta(y=1)$$

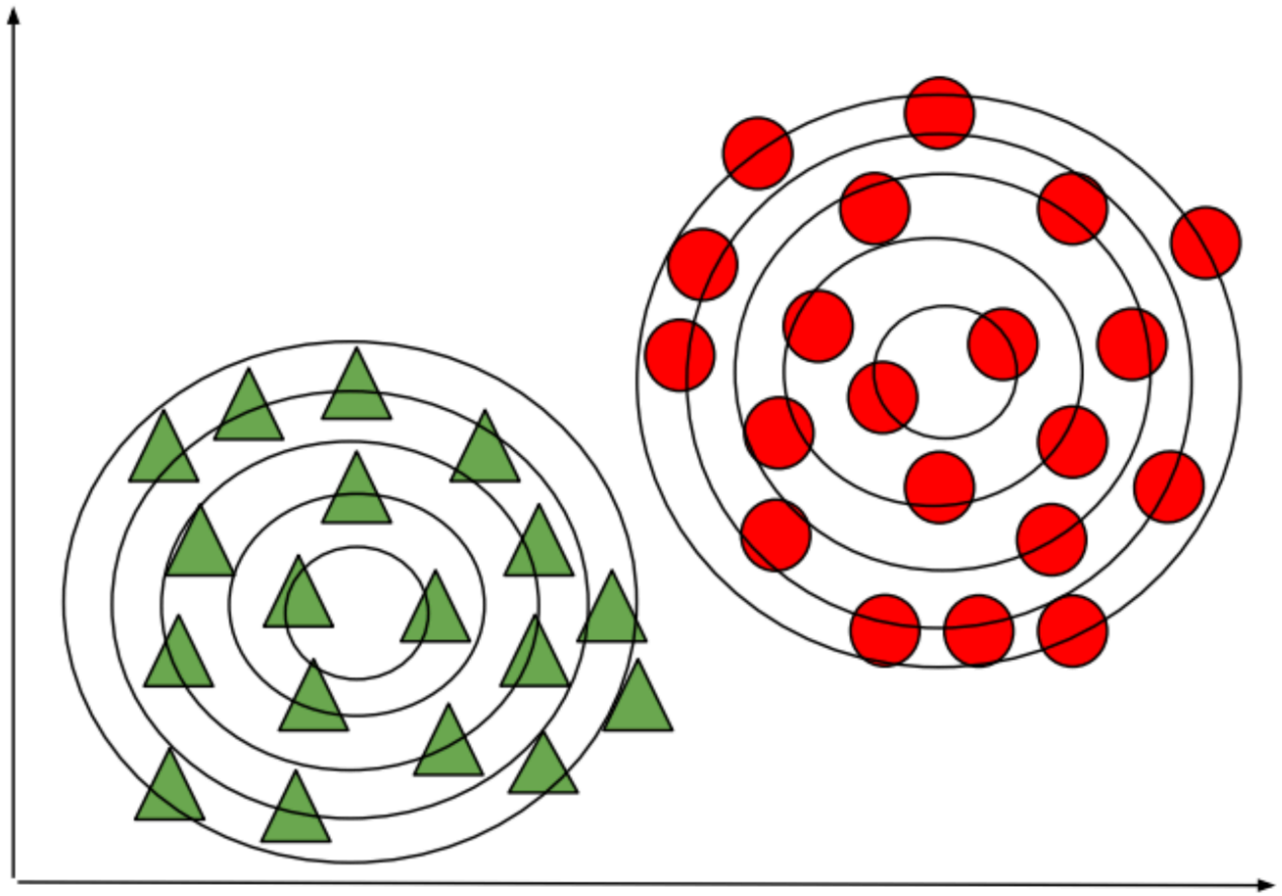
We output the class that's more likely to have generated x'.

Supervised Learning



Example: Gaussian Discriminant Analysis

The conditional probability of each class $P(x|y)$ is a Gaussian.



Formally, Gaussian Discriminant Analysis defines a model P_θ as follows:

- The conditional probability $P_\theta(x \mid y = k)$ of the data under class k is a multivariate Gaussian with mean and covariance μ_k, Σ_k.
- The distribution over classes is Categorical, $P_\theta(y = k) = \phi_k$.

How do we fit a GDA model?

1. The mean and covariance μ_k, Σ_k are the observed mean and variance of class k.

1. The probability ϕ_k of class k is the % of the data that has class k.

Key Idea: Maximum Likelihood Learning

We can learn a generative model $P_\theta(x, y)$ by maximizing the *maximum likelihood*:

$$\max_\theta \frac{1}{n} \sum_{i=1}^{n} \log P_\theta(x^{(i)}, y^{(i)}).$$

This seeks to find parameters θ such that the model assigns high probability to the training data.

Key Idea: The Max-Margin Principle

Linear classifiers can be generative or discriminative. They yield different decision boundaries.

Intuitively, we want to select linear decision boundaries with high *margin*: every point is as far as possible from the decision boundary.

Example: Support Vector Machines

Support Vector Machines fit linear models of the form

$$f_\theta(x) = \theta^\top x + \theta_0$$

such as to find the maximum margin hyperplane.

SVM: Constrained Primal

Recall that *primal* SVMs solve the following optimization problem.

$$\begin{aligned} \min_{\theta,\theta_0,\xi} \ & \frac{1}{2}||\theta||^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to } \ & y^{(i)}((x^{(i)})^\top\theta + \theta_0) \geq 1 - \xi_i \text{ for all } i \\ & \xi_i \geq 0 \end{aligned}$$

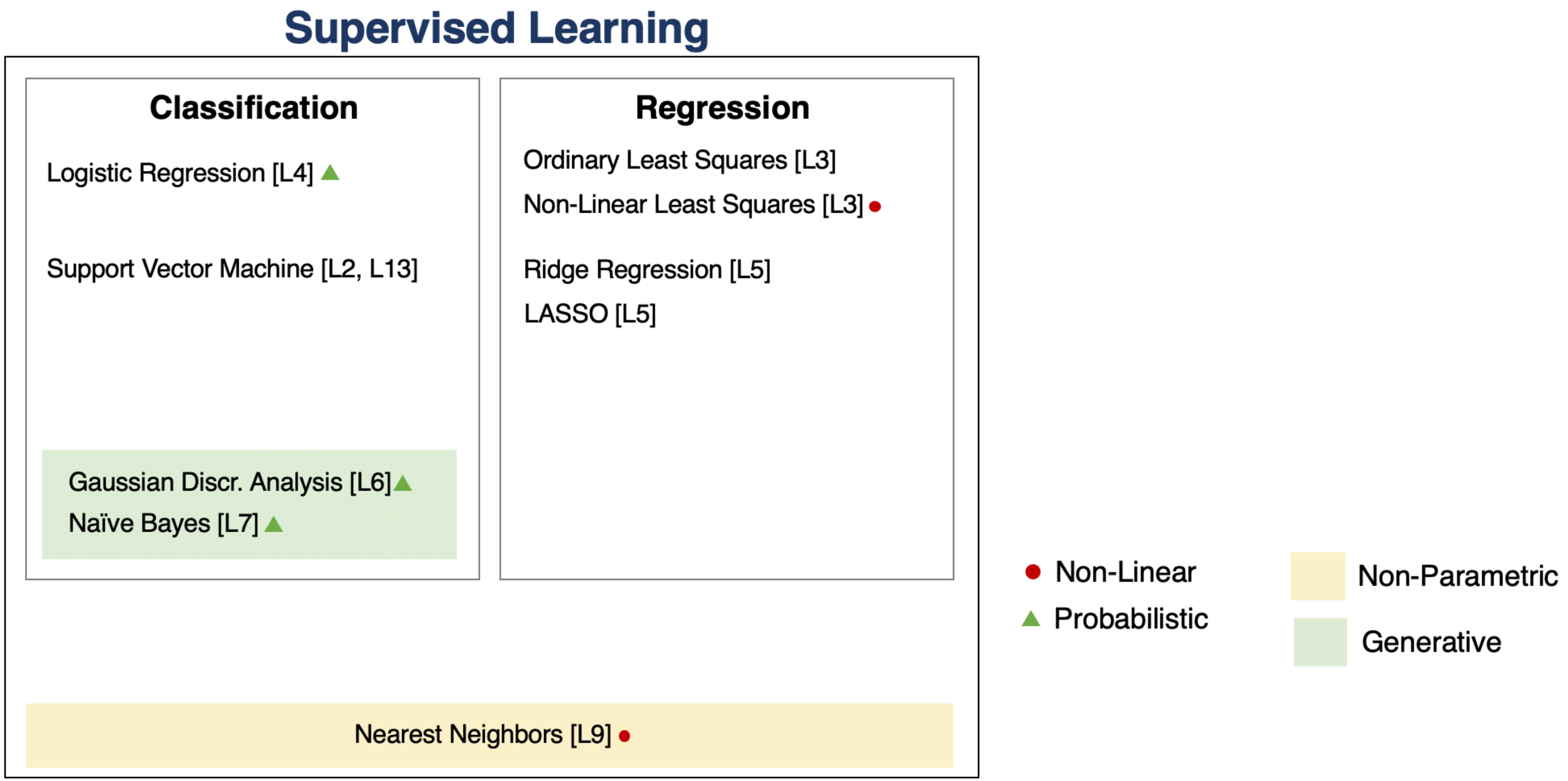
SVM: Unconstrained Primal

We can turn our optimization problem into an unconstrained form:

$$\min_{\theta,\theta_0} \sum_{i=1}^{n} \underbrace{\left(1 - y^{(i)}\left((x^{(i)})^\top \theta + \theta_0\right)\right)^+}_{\text{hinge loss}} + \underbrace{\frac{\lambda}{2}||\theta||^2}_{\text{regularizer}}$$

- The hinge loss penalizes incorrect predictions.
- The L2 regularizer ensures the weights are small and well-behaved.

Supervised Learning



Classification

Logistic Regression [L4] ▲

Support Vector Machine [L2, L13]

Gaussian Discr. Analysis [L6] ▲

Naïve Bayes [L7] ▲

Regression

Ordinary Least Squares [L3]

Non-Linear Least Squares [L3] ●

Ridge Regression [L5]

LASSO [L5]

Nearest Neighbors [L9] ●

● Non-Linear

▲ Probabilistic

🟨 Non-Parametric

🟩 Generative

Key Idea: The Kernel Trick

Many algorithms in machine learning only involve dot products $\phi(x)^\top \phi(z)$ but not the features ϕ themselves.

We can often compute $\phi(x)^\top \phi(z)$ very efficiently for complex ϕ using a kernel function $K(x, z) = \phi(x)^\top \phi(z)$. This is the **kernel trick**.

Example: The Kernel Trick in SVMs

Notice that in the SVM dual, the features $\phi(x)$ are never used directly. Only their *dot product* is used.

$$J(\lambda) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{n} \lambda_i \lambda_k y^{(i)} y^{(k)} \phi(x^{(i)})^\top \phi(x^{(k)})$$

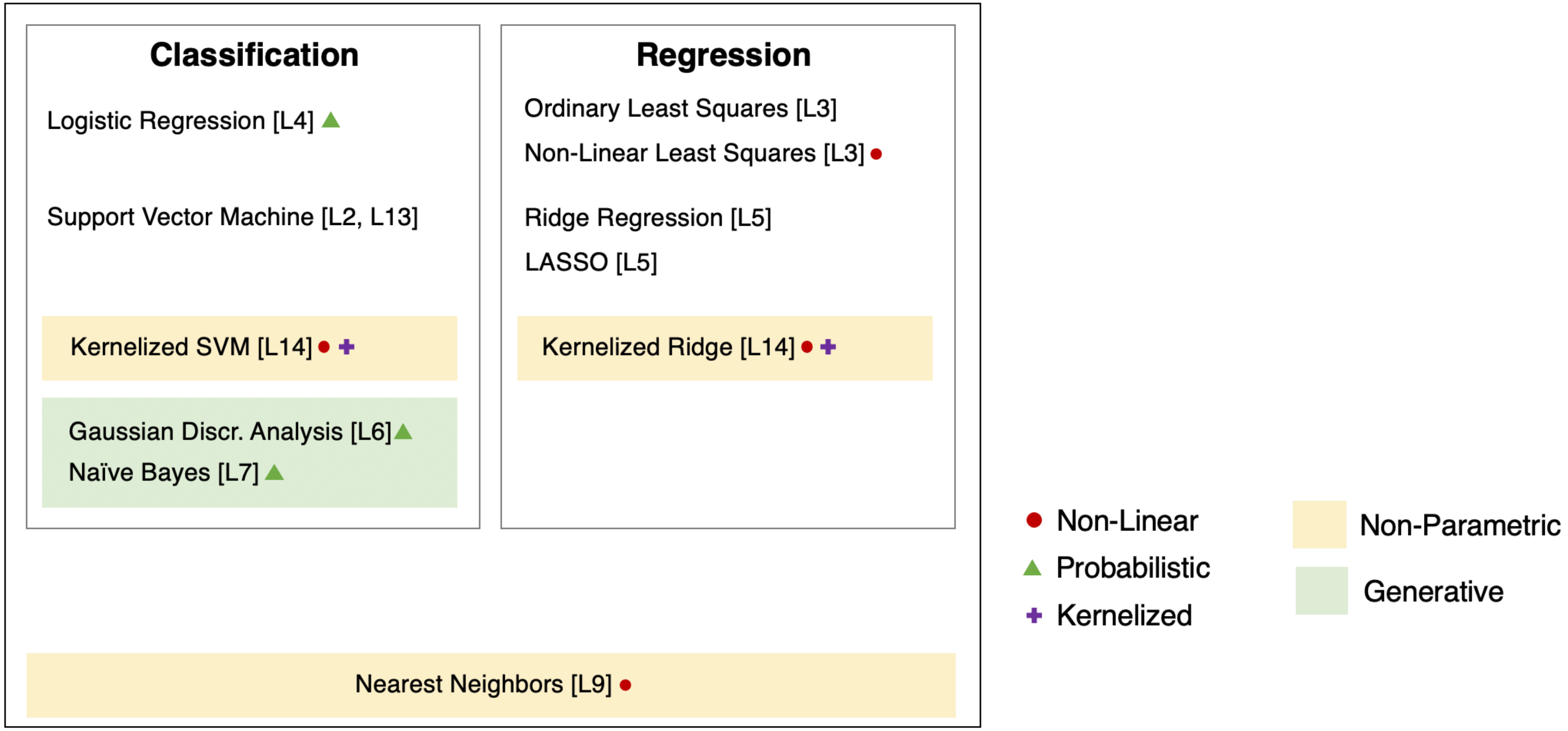
If we can compute the dot product efficiently, we can potentially use very complex features.

The prediction of an dual SVM is:

$$y_{\text{pred}} = \sum_{i=1}^{n} \lambda_i^* \cdot K(x_{\text{pred}}, x^{(i)}).$$

This is a weighted average of the training data. Thus, dual SVMs are **non-parametric**.

Supervised Learning

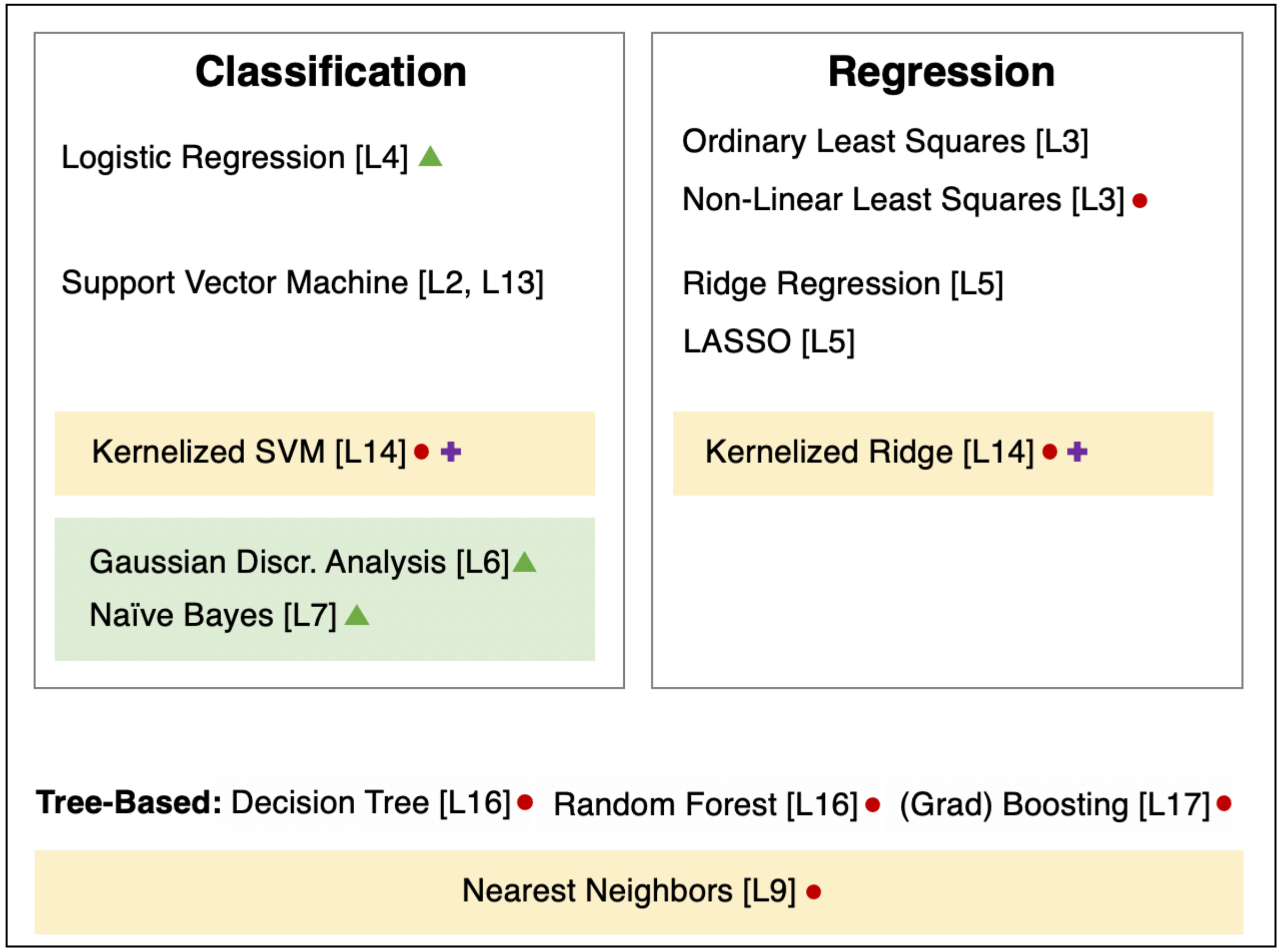


Tree-Based Models

Decision trees output target based on a tree of human-interpretable decision rules.

- **Random forests** combine large trees using *bagging* to reduce overfitting.
- **Boosted trees** combine small trees to reduce underfitting.

Supervised Learning



Classification

Logistic Regression [L4] ▲

Support Vector Machine [L2, L13]

Kernelized SVM [L14] ● ✚

Gaussian Discr. Analysis [L6] ▲

Naïve Bayes [L7] ▲

Regression

Ordinary Least Squares [L3]

Non-Linear Least Squares [L3] ●

Ridge Regression [L5]

LASSO [L5]

Kernelized Ridge [L14] ● ✚

Tree-Based: Decision Tree [L16] ● Random Forest [L16] ● (Grad) Boosting [L17] ●

Nearest Neighbors [L9] ●

● Non-Linear

▲ Probabilistic

✚ Kernelized

🟨 Non-Parametric

🟩 Generative

Neural Networks

Neural network models are inspired by the brain.

- A Perceptron is an artificial model of a neuron.
- MLPs stack multiple layers of artificial neurons.
- ConvNets tie the weights of neighboring neurons into receptive fields that implement the convolution operation.

Supervised Learning

Classification

Logistic Regression [L4] ▲

Support Vector Machine [L2, L13]

Kernelized SVM [L14] ● ✚

Gaussian Discr. Analysis [L6] ▲

Naïve Bayes [L7] ▲

Regression

Ordinary Least Squares [L3]

Non-Linear Least Squares [L3] ●

Ridge Regression [L5]

LASSO [L5]

Kernelized Ridge [L14] ● ✚

Neural: Perceptron [L18] Fully-Connected NNet [L18] ● ▲ ConvNet [L19] ● ▲

Tree-Based: Decision Tree [L16] ● Random Forest [L16] ● (Grad) Boosting [L17] ●

Nearest Neighbors [L9] ●

● Non-Linear

▲ Probabilistic

✚ Kernelized

🟨 Non-Parametric

🟩 Generative

Unsupervised Learning

We have a dataset *without* labels. We looked at three tasks:

- **Density estimation**: Recover the data distribution from data.
- **Clustering**: Identify clusters in the dataset.
- **Dimensionality reduction**: Low-dimensional data representations.

Key Idea: Density Estimation

The problem of density estimation is to approximate the data distribution P_{data} with the model P.

$$P \approx P_{\text{data}}.$$

In kernel density estimation we will fit a model of the form

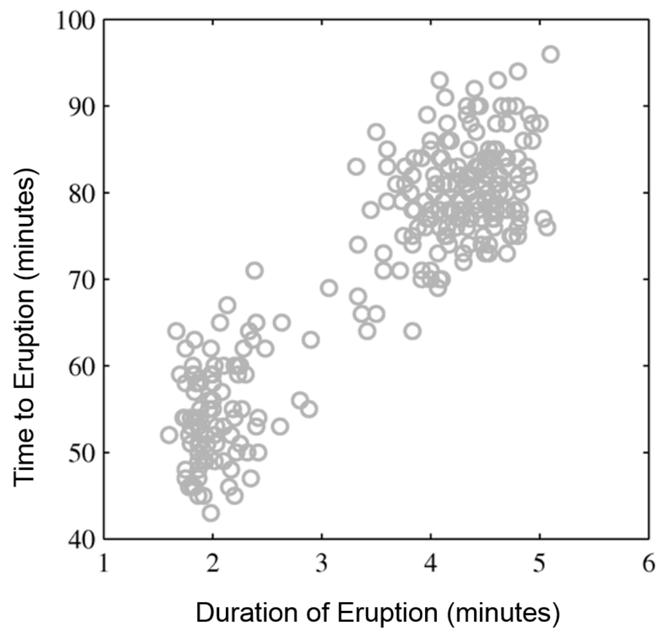
$$P(x) \propto \sum_{i=1}^{n} K(x, x^{(i)})$$

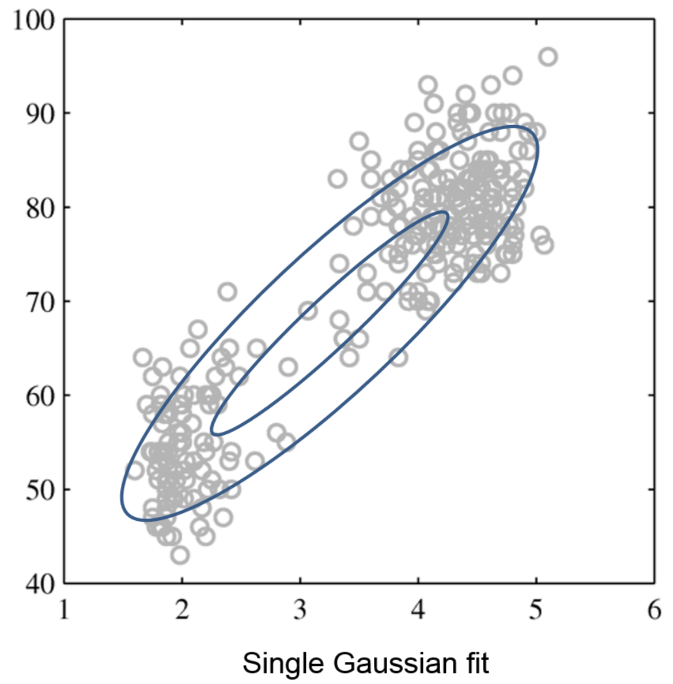
One example is the Gaussian kernel $K(x, z; \delta) \propto \exp(-||x - z||^2/2\delta^2)$.

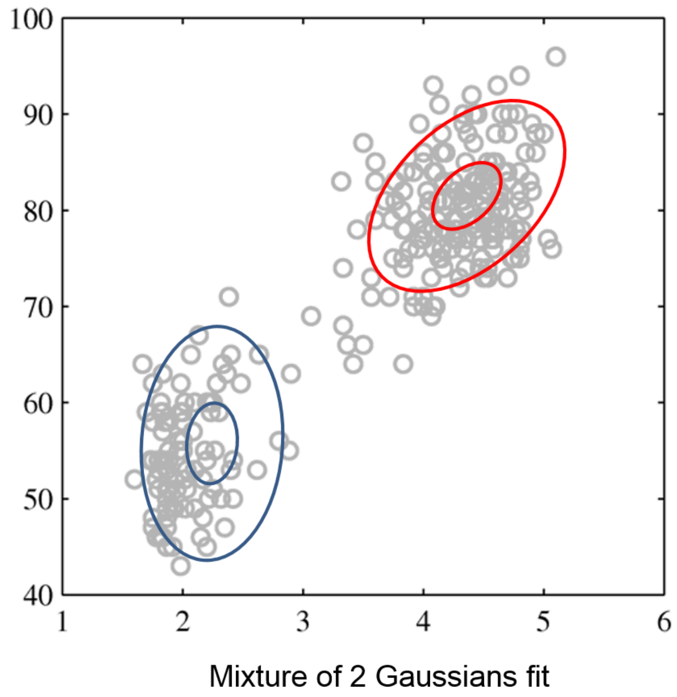
Key Idea: Clustering & Gaussian Mixtures

We can perform clustering by fitting a mixtures of Gaussians model (GDA model) without labels.

Raw data **Single Gaussian** **Mixture of Gaussians**







But how do we fit a GMM (GDA model) without labels?

We can use expectation maximization. We repeat until convergence:

1. Given parameters θ, "hallucinate" class labels z using $P_\theta(z|x)$.

1. Given labels z, fit parameters θ.

Supervised Learning

Classification

Logistic Regression [L4] ▲

Support Vector Machine [L2, L13]

Kernelized SVM [L14] ● ✚

Gaussian Discr. Analysis [L6] ▲

Naïve Bayes [L7] ▲

Regression

Ordinary Least Squares [L3]

Non-Linear Least Squares [L3] ●

Ridge Regression [L5]

LASSO [L5]

Kernelized Ridge [L14] ● ✚

Neural: Perceptron [L18] Fully-Connected NNet [L18] ● ▲ ConvNet [L19] ● ▲

Tree-Based: Decision Tree [L16] ● Random Forest [L16] ● (Grad) Boosting [L17] ●

Nearest Neighbors [L9] ●

Unsupervised Learning

Clustering

K-Means [L8]

Gaussian Mixture Models [L10] ▲

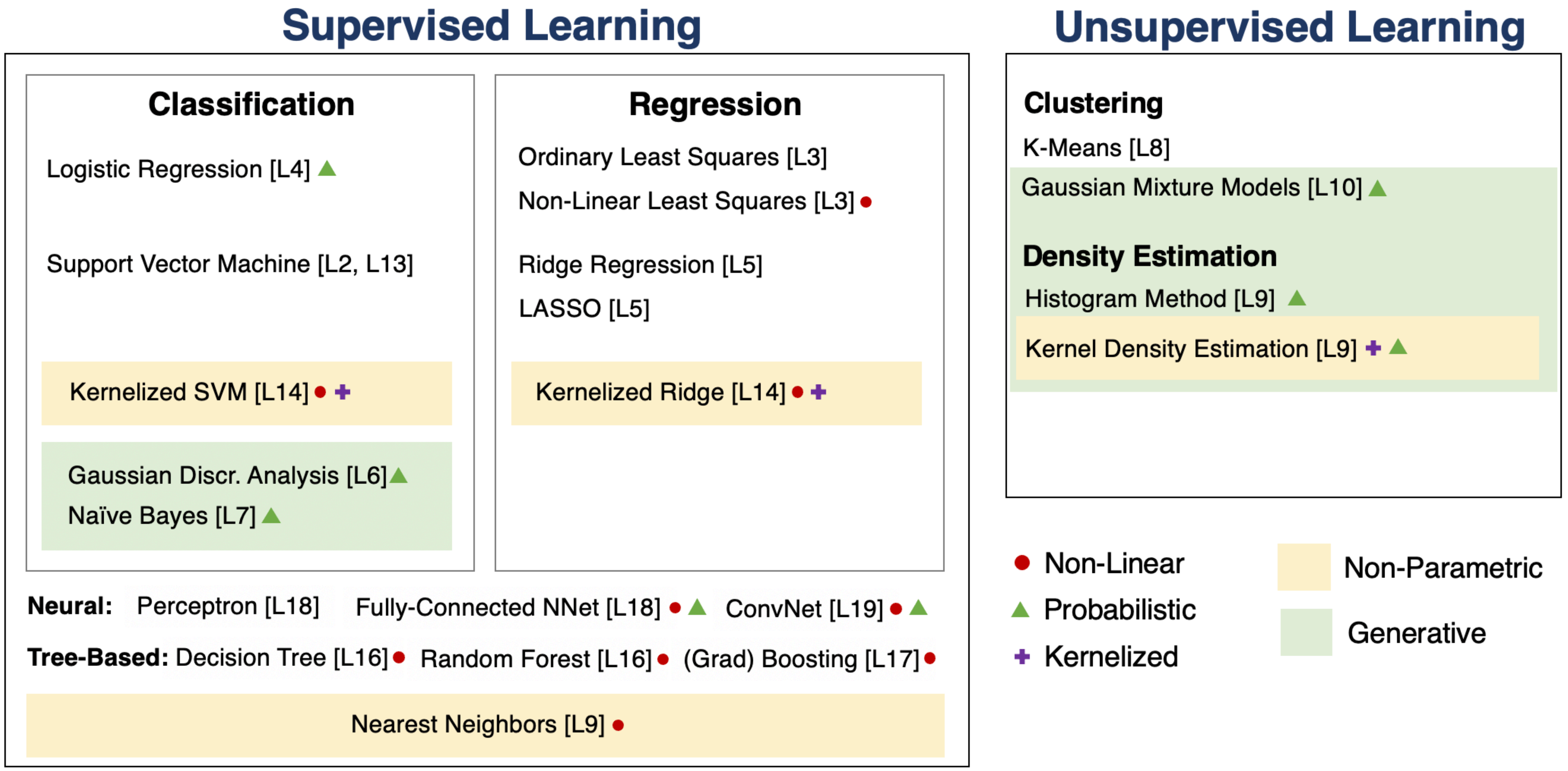
● Non-Linear

▲ Probabilistic

✚ Kernelized

🟨 Non-Parametric

🟩 Generative



Supervised Learning

Classification

Logistic Regression [L4] ▲

Support Vector Machine [L2, L13]

Kernelized SVM [L14] ● ✚

Gaussian Discr. Analysis [L6] ▲
Naïve Bayes [L7] ▲

Regression

Ordinary Least Squares [L3]
Non-Linear Least Squares [L3] ●

Ridge Regression [L5]
LASSO [L5]

Kernelized Ridge [L14] ● ✚

Neural: Perceptron [L18] Fully-Connected NNet [L18] ● ▲ ConvNet [L19] ● ▲

Tree-Based: Decision Tree [L16] ● Random Forest [L16] ● (Grad) Boosting [L17] ●

Nearest Neighbors [L9] ●

Unsupervised Learning

Clustering

K-Means [L8]
Gaussian Mixture Models [L10] ▲

Density Estimation

Histogram Method [L9] ▲
Kernel Density Estimation [L9] ✚ ▲

● Non-Linear
▲ Probabilistic
✚ Kernelized

🟨 Non-Parametric
🟩 Generative

Key Idea: Linear Dimensionality Reduction

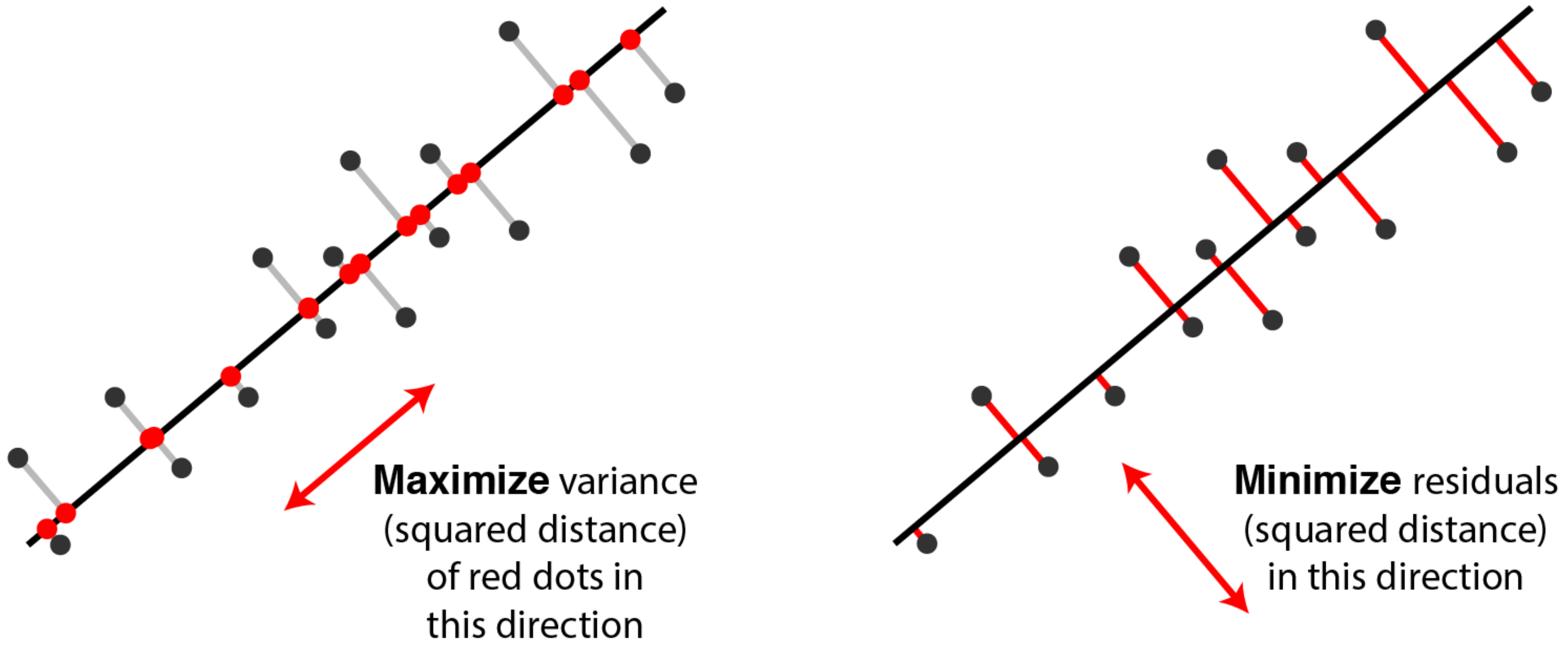
Suppose $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Z} = \mathbb{R}^p$ for some $p < d$.

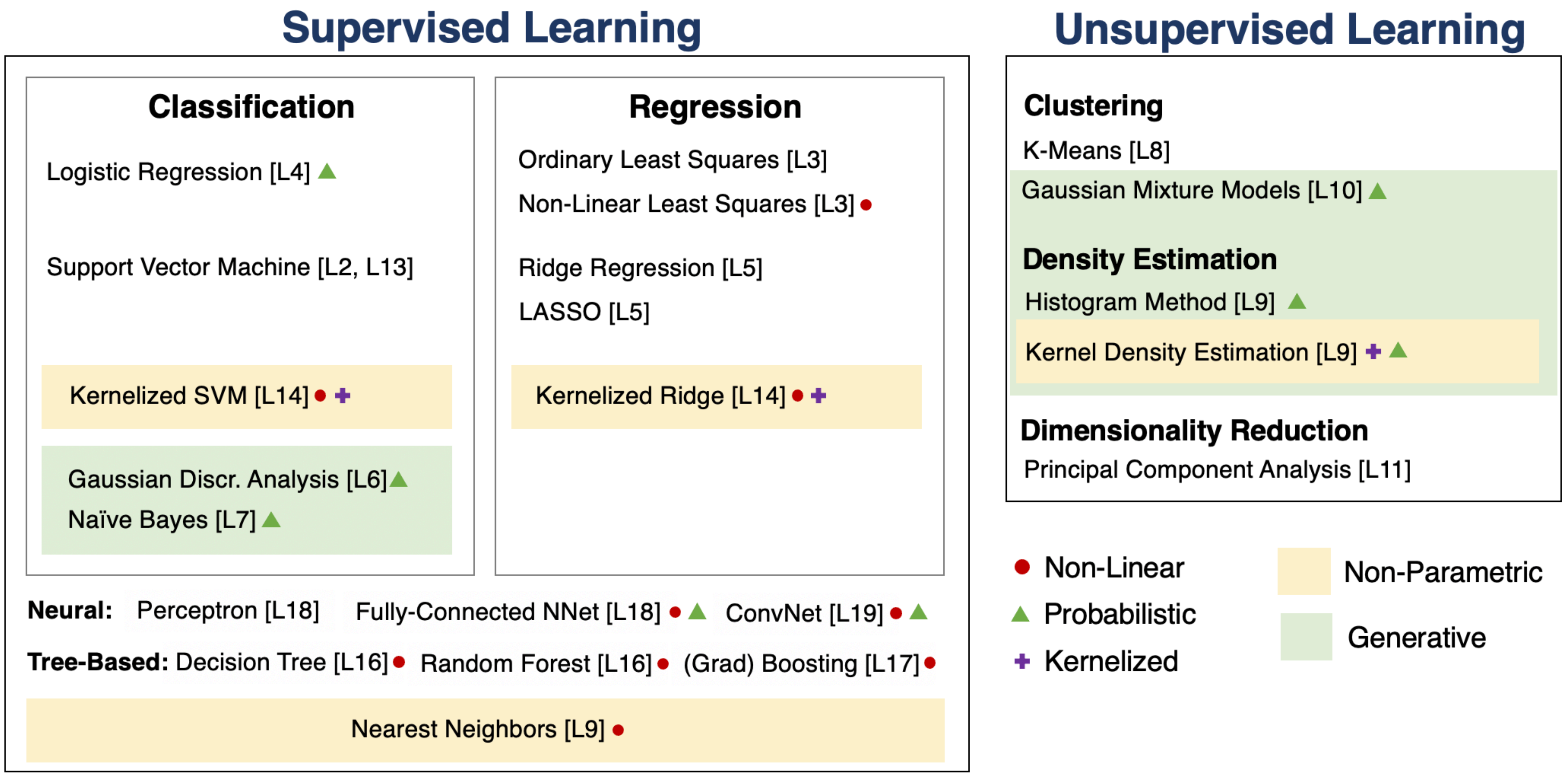
We want to learn a linear projection

$$z = W^\top \cdot x.$$

The latent dimension z is obtained from x via a matrix W.

We find W using one of the following equivalent objectives (figure credit: Alex Williams)





Supervised Learning

Classification
Logistic Regression [L4] ▲
Support Vector Machine [L2, L13]
Kernelized SVM [L14] ● ✚
Gaussian Discr. Analysis [L6] ▲
Naïve Bayes [L7] ▲

Regression
Ordinary Least Squares [L3]
Non-Linear Least Squares [L3] ●
Ridge Regression [L5]
LASSO [L5]
Kernelized Ridge [L14] ● ✚

Neural: Perceptron [L18] Fully-Connected NNet [L18] ● ▲ ConvNet [L19] ● ▲
Tree-Based: Decision Tree [L16] ● Random Forest [L16] ● (Grad) Boosting [L17] ●
Nearest Neighbors [L9] ●

Unsupervised Learning

Clustering
K-Means [L8]
Gaussian Mixture Models [L10] ▲

Density Estimation
Histogram Method [L9] ▲
Kernel Density Estimation [L9] ✚ ▲

Dimensionality Reduction
Principal Component Analysis [L11]

● Non-Linear
▲ Probabilistic
✚ Kernelized
🟨 Non-Parametric
🟩 Generative

How To Decide Which Algorithm to Use

One factor is how much data you have. In the **small data** (<10,000) regime, consider:

- Linear models with hand-crafted features (LASSO, LR, NB, SVMs)
- Kernel methods often work best (e.g., SVM + RBF kernel)
- Non-parametric methods (kernels, nearest neighbors) are also powerful

In the **big data** regime,

- If using "high-level" features, gradient boosted trees are state-of-the-art
- When using "low-level" representations (images, sound signals), neural networks work best
- Linear models with good features are also good and reliable

Some additional advice:

- If interpretability matters, use decision trees or LASSO.
- When uncertainty estimates are important use probabilistic methods.
- If you know the data generating process, use generative models.

What’s Next? Ideas for Courses

Consider the following courses to keep learning about ML:

- Graduate courses in the Spring semester at Cornell (generative models, NLP, etc.)
- Masters courses: Deep Learning, ML Engineering, Data Science, etc.
- Online courses, e.g. Full Stack Deep Learning

What’s Next? Ideas for Research

In order to get involved in research, I recommend:

- Contacting research groups at Cornell for openings
- Watching online ML tutorials, e.g. NeurIPS
- Reading and implementing ML papers on your own

What’s Next? Ideas for Industry Projects

Finally, a few ideas for how to get more practice applying ML in the real world:

- Participate in Kaggle competitions and review solutions
- Build an open-source project that you like and host it on Github

Thank You For Taking Applied ML!