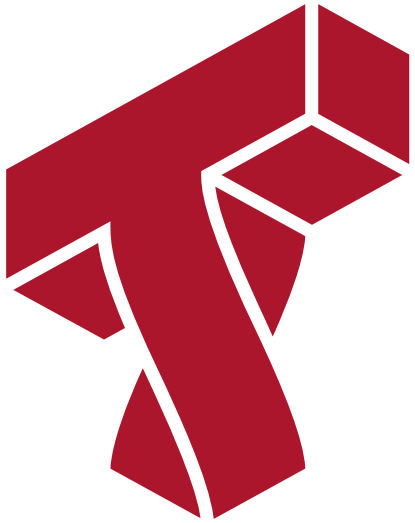


Mid-Semester Review

Part 1: Review of Supervised Learning

We start with an overview of the supervised learning algorithms seen in class.

Supervised Machine Learning

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

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

The output is a predictive model that maps inputs to targets. For instance, it can predict targets on new inputs.

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 features), allowing us to fit complex functions.

We define the least squares objective for the model as:

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

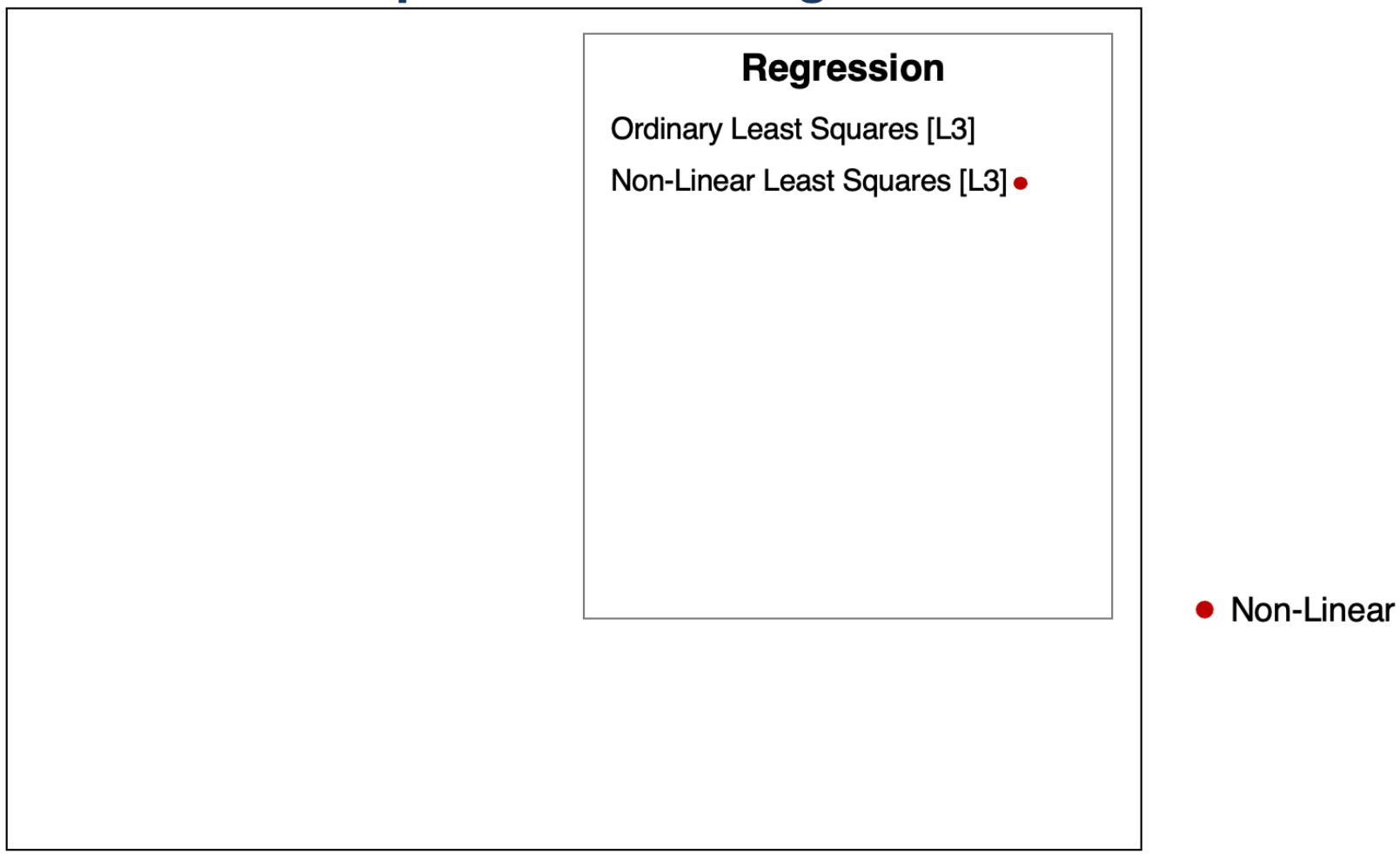
We can set the gradient to zero to obtain the *normal equations*:

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

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

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

Supervised Learning



Overfitting

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

- A very expressive model (a high degree polynomial) fits the training dataset perfectly.
- The model also makes highly incorrect predictions outside the training set, and doesn't generalize.

We can measure overfitting and underfitting by estimating accuracy on held out data and comparing it to the training data.

- If training performance is high but holdout performance is low, we are overfitting.
- If training performance is low and holdout performance is low, we are underfitting.

We will see many ways of dealing with overfitting, but here are some ideas:

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

Regularization

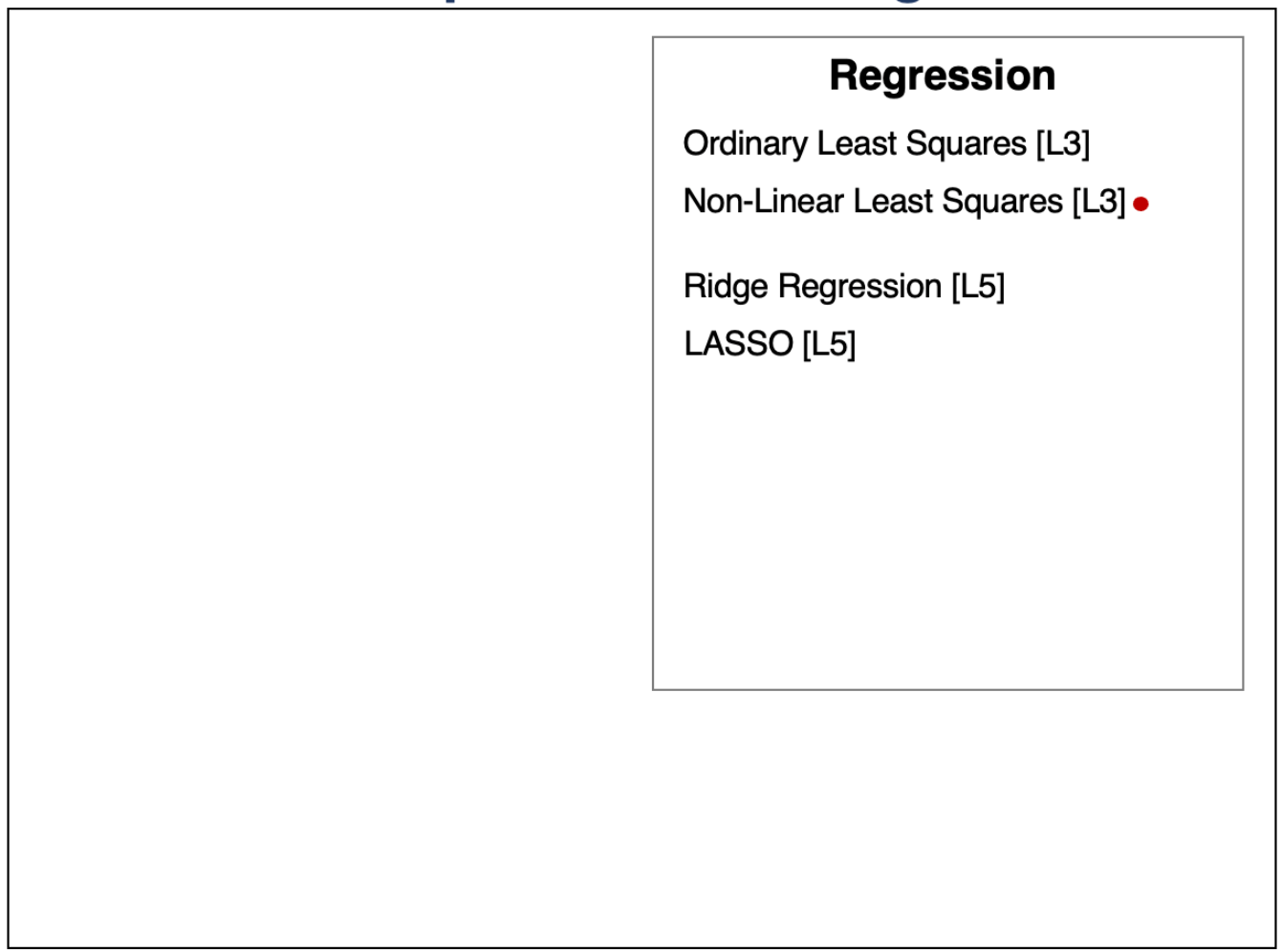
The idea of regularization is to penalize complex models that may overfit the data.

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

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

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

Supervised Learning



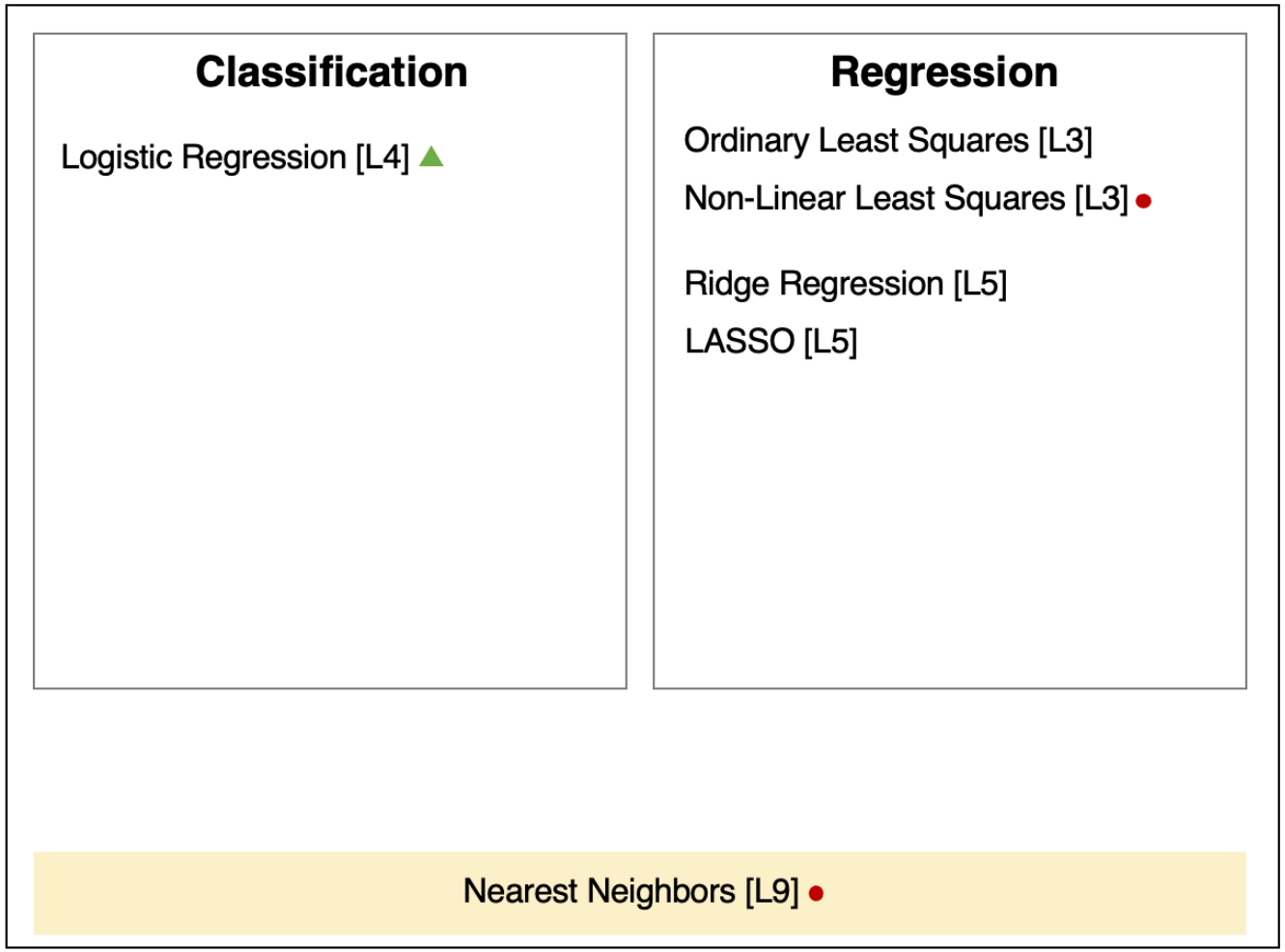
Regression vs. Classification

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 problems depending on the targets $y^{(i)}$.

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

Supervised Learning



● Non-Linear

▲ Probabilistic

Non-Parametric

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.

- Non-parametric models have the advantage of not losing any information at training time.
- However, they are also computationally less tractable and may easily overfit the training set.

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.

Generative Models: Intuition

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* each x according to how well it matches class k.

- A class probability $P_\theta(y=k)$ encoding our prior beliefs. These are often just the % of each class in the data.

In the context of spam classification, we would fit two models on a corpus of emails x with spam/non-spam labels y:

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

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

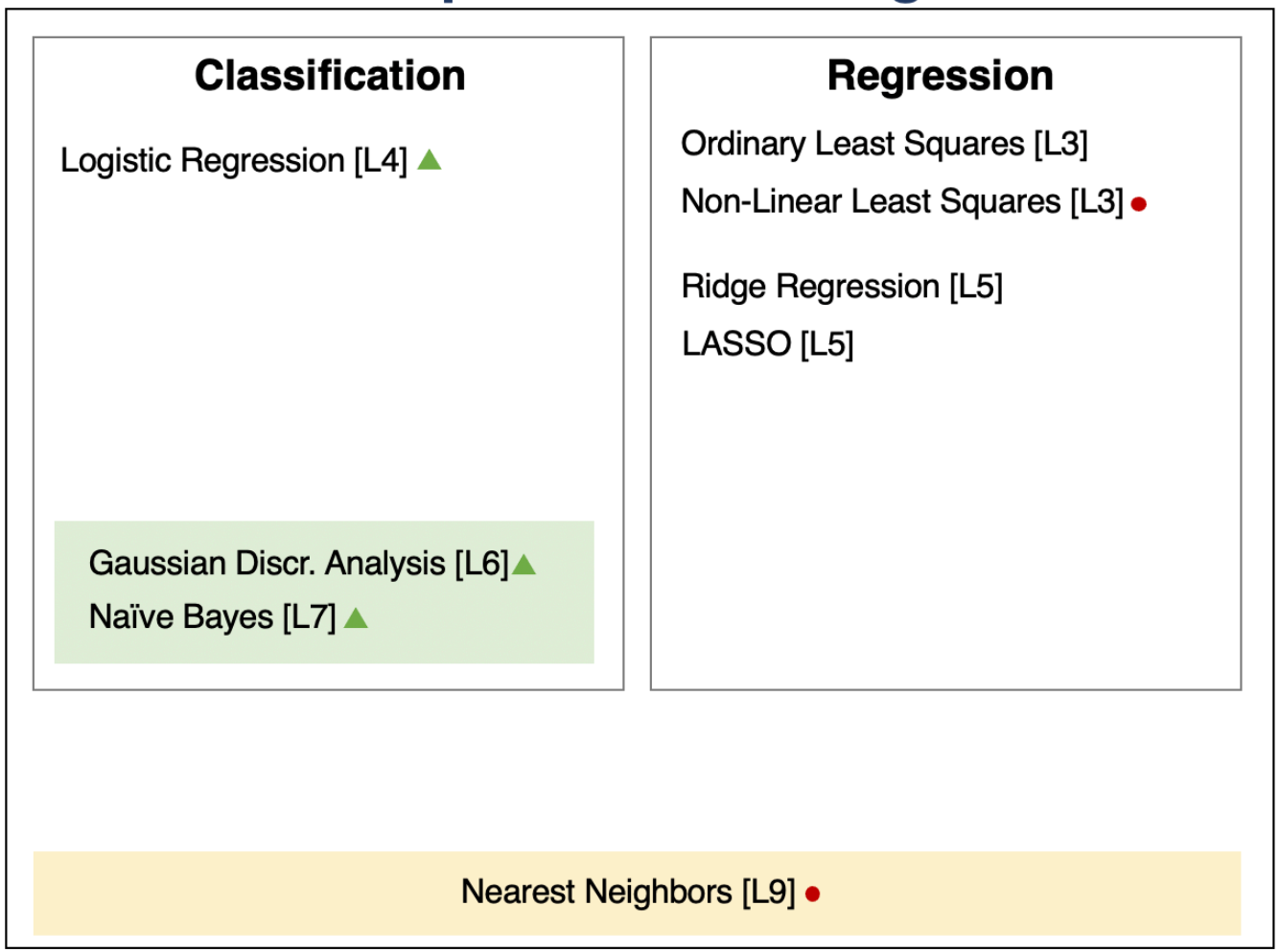
We also choose a prior $P(y)$.

In the context of spam classification, 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



● Non-Linear

▲ Probabilistic

Non-Parametric

Generative

Text Classification & Bag of Words

Perhaps the most widely used approach for representing text documents is called "bag of words".

We start by defining a vocabulary V containing all the possible words we are interested in, e.g.:

$$V = \{\text{church}, \text{doctor}, \text{fervently}, \text{purple}, \text{slow}, \dots\}$$

A bag of words representation of a document x is a function $\phi(x) \to \{0,1\}^{|V|}$ that outputs a feature vector

$$\phi(x) = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ \vdots \end{pmatrix} \begin{matrix} \text{church} \\ \text{doctor} \\ \text{fervently} \\ \\ \text{purple} \\ \\ \end{matrix}$$

of dimension V. The j-th component $\phi(x)_j$ equals 1 if x contains the j-th word in V and 0 otherwise.

Bernoulli Naive Bayes Model

The *Bernoulli Naive Bayes* model $P_\theta(x, y)$ is defined for *binary data* $x \in \{0, 1\}^d$ (e.g., bag-of-words documents).

The probability of the data $x' = [0, 1, 0, \dots]$ for each class equals

$$P_\theta(x = x' | y = k) = \prod_{j=1}^{d} P_\theta(x_j = x'_j \mid y = k)$$

where each $P_\theta(x_j \mid y = k)$ is Bernoulli:

$$\begin{aligned} P_\theta(x_j = 1 \mid y = k) &= \psi_{jk} \\ P_\theta(x_j = 0 \mid y = k) &= 1 - \psi_{jk} \end{aligned}$$

The probability over y is Categorical: $P_\theta(y = k) = \phi_k$.

For example, suppose that $x' = [0, 1, 0, 0, 1]$. Then we have:

$$\begin{aligned} P_\theta(x = x' | y = k) &= \prod_{j=1}^{d} P_\theta(x_j = x'_j \mid y = k) \\ &= (1 - \psi_{1k}) \cdot \psi_{2k} \cdot (1 - \psi_{3k}) \cdot (1 - \psi_{4k}) \cdot \psi_{5k} \end{aligned}$$

i.e., we multiply the probabilities of each word being present or absent.

Maximum Likelihood Learning

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

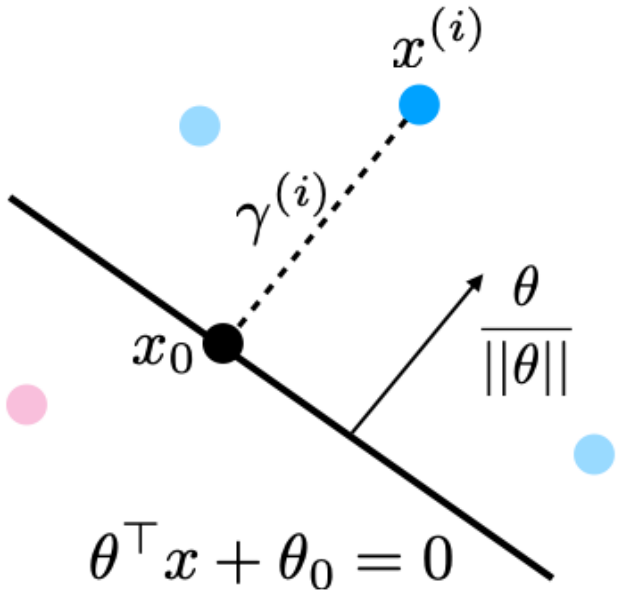
$$\max_\theta \frac{1}{n} \sum_{i=1}^{n} \log P_\theta(x^{(i)}, y^{(i)}).$$

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

Max-Margin Classification

Maximizing margins is a principle for choosing good decision boundaries.

- Margin: distance from the decision boundary to closest datapoint.



- We want the margin to be as large as possible.

Support Vector Machine: Primal Form

We saw that maximizing the margin of a linear model amounts to solving the following optimization problem.

$$\min_{\theta,\theta_0} \frac{1}{2}||\theta||^2$$
$$\text{subject to } \; y^{(i)}((x^{(i)})^\top \theta + \theta_0) \geq 1 \text{ for all } i$$

We minimize the L2 norm among all correct decision boundaries.

Support Vector Machine: Dual Form

The solution to this problem also happens to be given by the following *dual* problem:

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

$$\text{subject to } \sum_{i=1}^{n} \lambda_i y^{(i)} = 0$$

- It's better to use the primal when we have $n > d$
- Use the dual when we have $d > n$
- The dual allows us to use kernels

Kernelized SVM

We can replace each dot product via a kernel function

$$\max_{\lambda} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{n} \lambda_i \lambda_k y^{(i)} y^{(k)} K(x^{(i)}, x^{(k)})$$

$$\text{subject to } \sum_{i=1}^{n} \lambda_i y^{(i)} = 0$$

$$C \geq \lambda_i \geq 0 \text{ for all } i$$

Kernels can compute the dot product more efficiently.

How to choose features or kernels?

- Most often you want to start with a kernel (don't worry about the features)
- There are conditions that guarantee that a kernel corresponds to some dot product

Also, the predictions at a new point x' are given by

$$y' = \sum_{i=1}^{n} \lambda_i^* y^{(i)} K(x^{(i)}, x').$$

Are kernelized SVMs:

- For classification or regression?
- Discriminative or generative?
- Parametric or non-parametric?

Consider an update rule for kernelized k-NN:

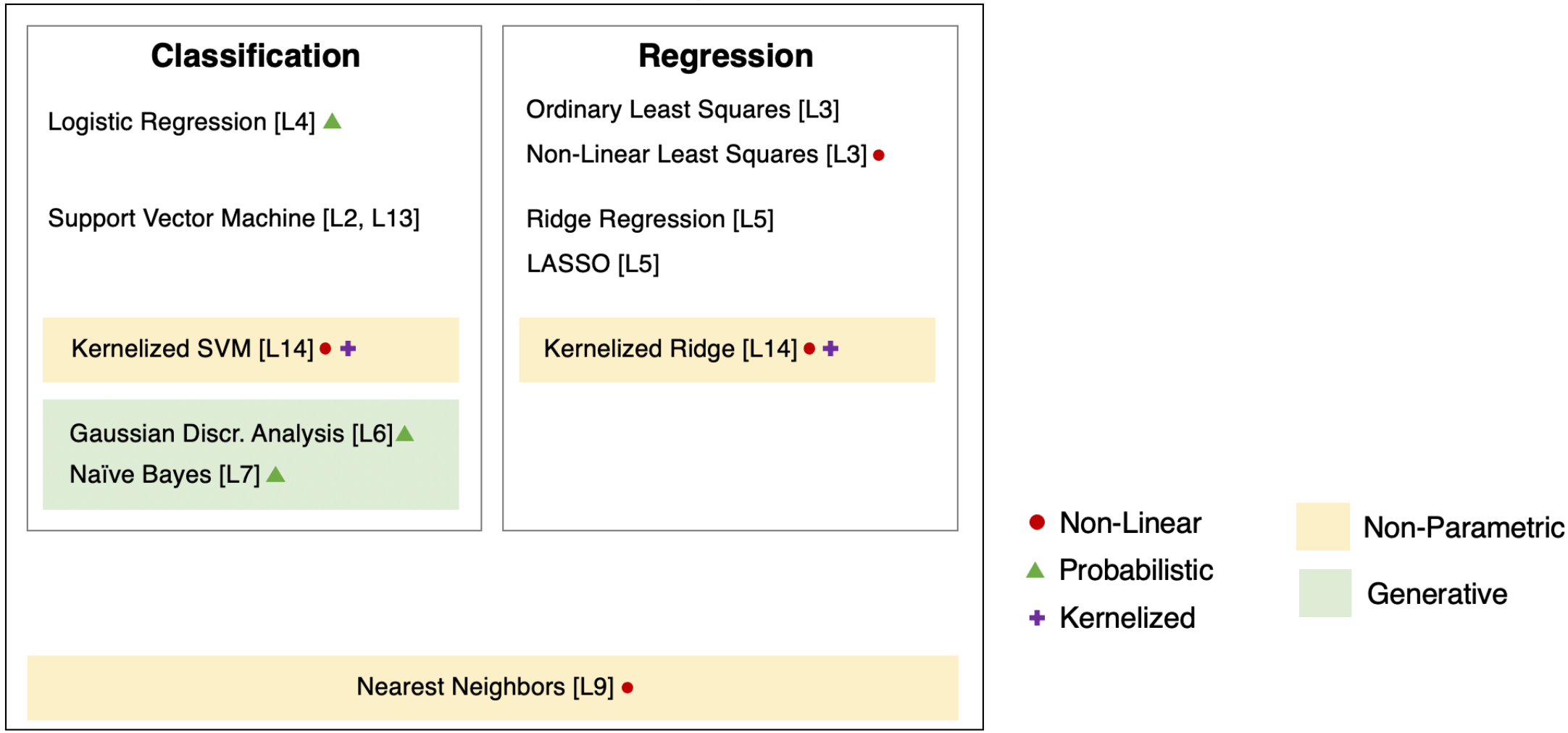
$$y' = \sum_{i=1}^{n} y^{(i)} K(x^{(i)}, x').$$

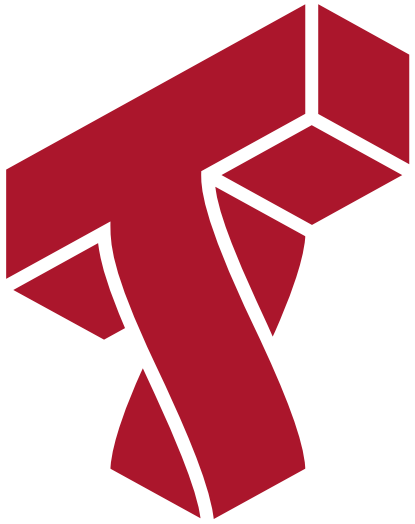
This is practically the same as kernelized SVM:

$$y' = \sum_{i=1}^{n} \lambda_i^* y^{(i)} K(x^{(i)}, x').$$

You can view the SVM as learning an optimal weighting of data in kernelized k-NN (hence, it's non-parametric).

Supervised Learning





Part 2: Unsupervised Learning

Next, we review algorithms for unsupervised learning.

Density Estimation

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

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

It's also a general learning task. We can solve many downstream tasks using a good model P:

- Outlier and novelty detection
- Generating new samples x
- Visualizing and understanding the structure of P_{data}

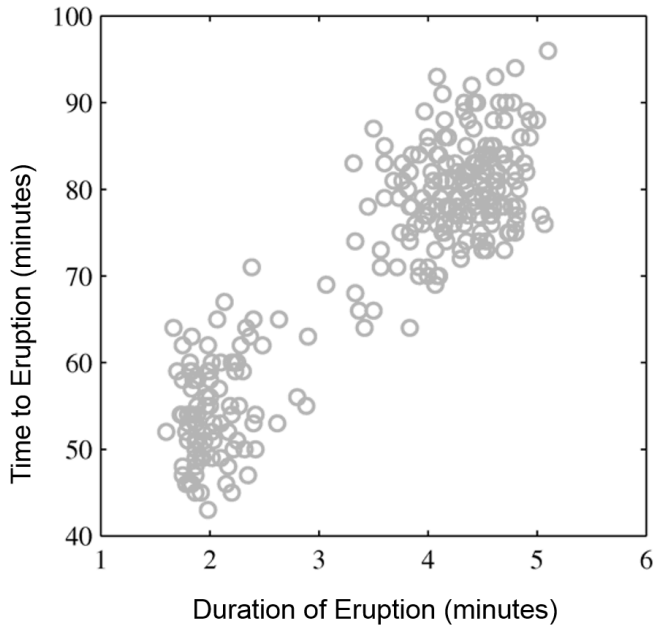
Clustering

Clustering is the problem of identifying distinct components in the data.

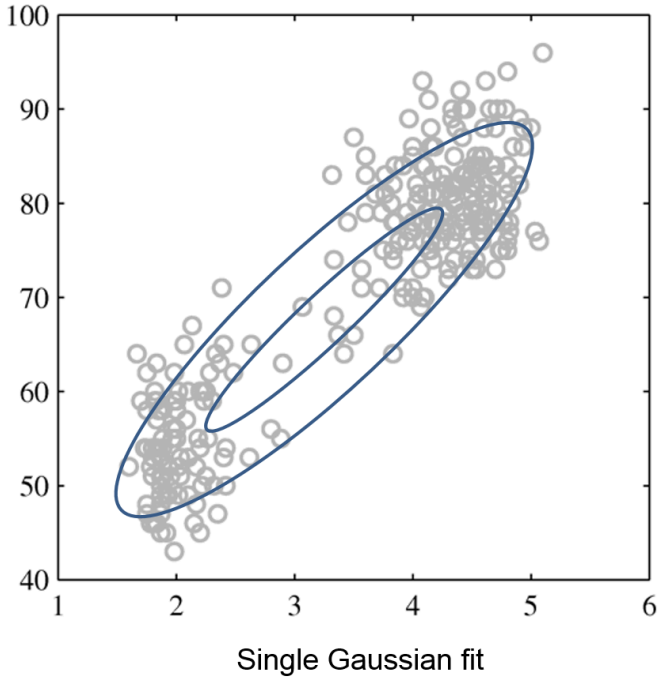
- A cluster $C_k \subseteq \mathcal{X}$ can be thought of as a subset of the space $\mathcal{X}$.
- Datapoints in a cluster are more similar to each other than to points in other clusters

Intuitively, a GMM represents well the two clusters in the geyser dataset:

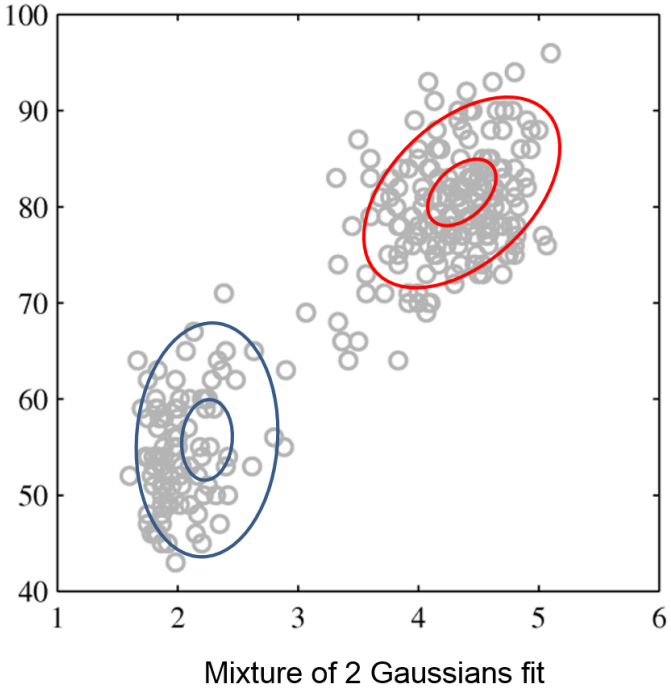
Raw data



Single Gaussian



Mixture of Gaussians



Linear Dimensionality Reduction

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

$$f_\theta : \mathcal{X} \to \mathcal{Z}$$

is a linear function with parameters $\theta = W \in \mathbb{R}^{d \times p}$:

$$z = f_\theta(x) = W^\top \cdot x.$$

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

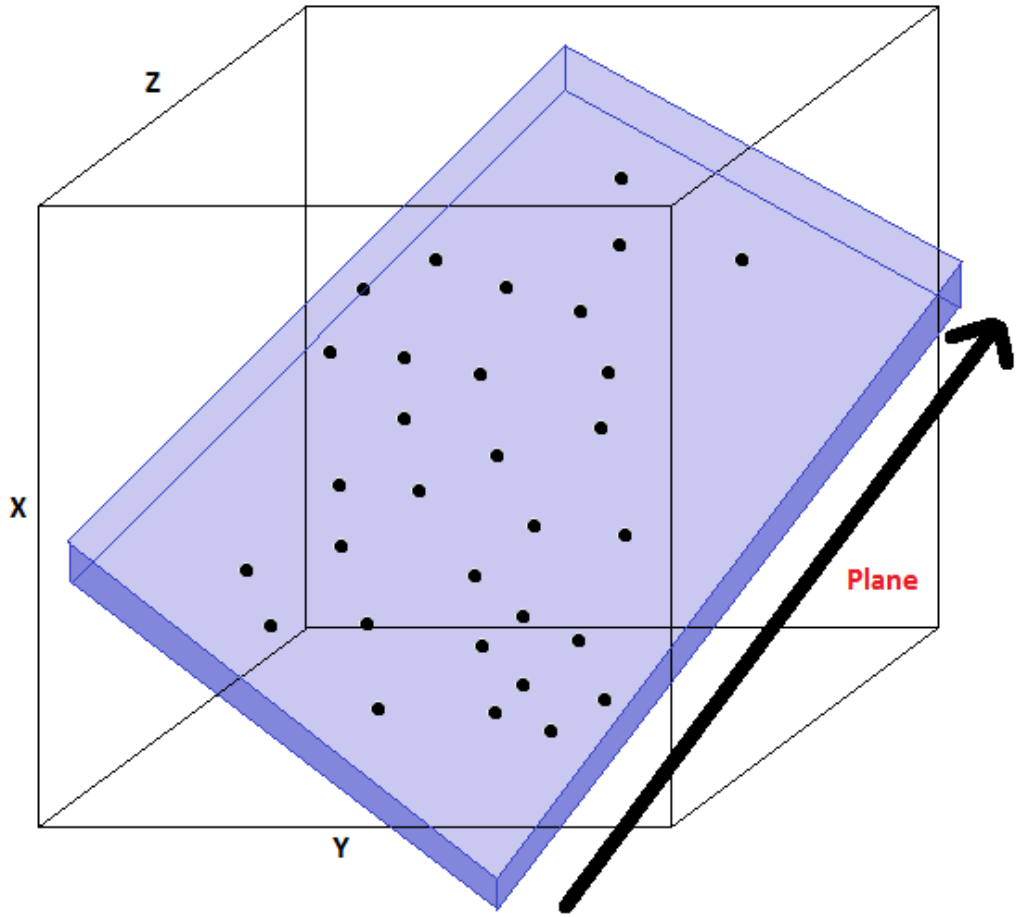
Principal component analysis (PCA) assumes that

- The projection subspace is spanned by a set of orthonormal vectors $w^{(1)}, w^{(2)}, \ldots, w^{(p)}$
- The data x are approximated by a linear combination $\tilde{x}$ of the $w^{(k)}$

$$x \approx \tilde{x} = \sum_{k=1}^{p} w^{(k)} z_k = Wz$$

for some $z \in \mathcal{X}$ that are the coordinates of $\tilde{x}$ in the basis W.

In this example, the data lives in a lower-dimensional 2D plane within a 3D space (image credit).



A natural objective is to minimize the reconstruction error

$$J_1(W) = \sum_{i=1}^{n} \|x^{(i)} - \tilde{x}^{(i)}\|_2^2 = \sum_{i=1}^{n} \|x^{(i)} - WW^\top x^{(i)}\|_2^2$$

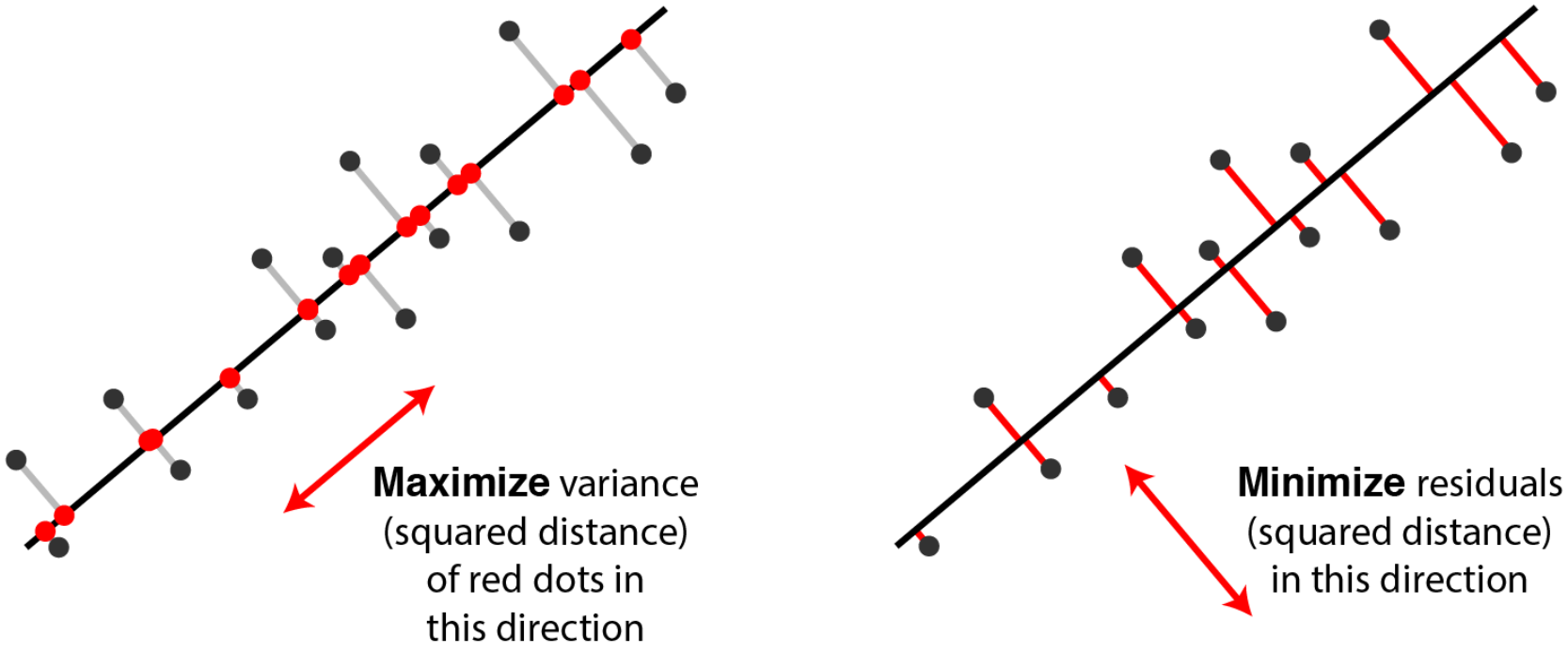
between each input $x^{(i)}$ and its approximate reconstruction

$$\tilde{x}^{(i)} = W \cdot z^{(i)} = W \cdot W^\top \cdot x^{(i)}.$$

The variance objective is simply

$$J_2(W) = \hat{\mathbb{E}}\left[\|W^\top x\|^2\right] = \frac{1}{n}\sum_{i=1}^{n} \|W^\top x^{(i)}\|_2^2.$$

The two are equivalent (figure credit: Alex Williams)



Recall that the positive semidefinite matrix $\hat{\Sigma} = \frac{1}{n} X^\top X$ has an *eigendecomposition*

$$\hat{\Sigma} = Q \Lambda Q^\top = \sum_{j=1}^{d} \lambda_j q^{(j)} (q^{(j)})^\top.$$

- Q is a matrix whose columns are orthonormal eigenvectors $q^{(j)}$ for $j = 1, 2, \ldots, d$.
- Λ is a diagonal matrix of positive eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d$.

The variance objective can be written as follows.

$$\max_w J(w) = \max_w \sum_{j=1}^{d} \lambda_j (w^\top q^{(j)})^2$$

Its optimum is attained by the top eigenvector $w = q^{(1)}$. The optimum is $J(q^{(1)}) = \lambda_1$.

More generally when $p > 1$, our objective is

$$J(W) = \sum_{k=1}^{p} \sum_{j=1}^{d} \lambda_j ((w^{(k)})^\top q^{(j)})^2$$

where W is a matrix of orthonormal columns $w^{(1)}, w^{(2)}, \ldots, w^{(p)}$.

Unsupervised Learning

Clustering

K-Means [L8]

Gaussian Mixture Models [L10] ▲

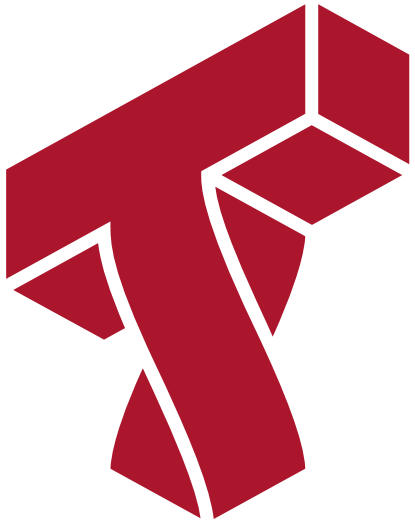
Density Estimation

Histogram Method [L9] ▲

Kernel Density Estimation [L9] ✚ ▲

Dimensionality Reduction

Principal Component Analysis [L11]



Part 3: Machine Learning in Practice

We conclude with high-level considerations about how to apply machine learning.

Machine learning is an interactive process. At each iteration, the machine learning engineer needs to make a number of decisions.

- Add more data?
- Train the algorithm for longer?
- Use a bigger model?
- Add regularization?
- Add new features?

We prioritize these using a principled process (more on this in a few weeks).

Datasets for Model Development

When developing machine learning models, it is customary to work with three datasets:

- **Training set**: Data on which we train our algorithms.
- **Development set** (validation or holdout set): Data used for tuning algorithms.
- **Test set**: Data used to evaluate the final performance of the model.

Model Development Workflow

The typical way in which these datasets are used is:

1. **Training:** Try a new model and fit it on the training set.

1. **Model Selection**: Estimate performance on the development set using metrics. Based on results, try a new model idea in step #1.

1. **Evaluation**: Finally, estimate real-world performance on test set.

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 following lectures, we will see algorithms for the **big data** regime.

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.