Introduction to Statistical Learning

Prototype Methods & kNN

Alexey Boldyrev, Maksim Karpov, Oleg Melnikov

28/11/2023

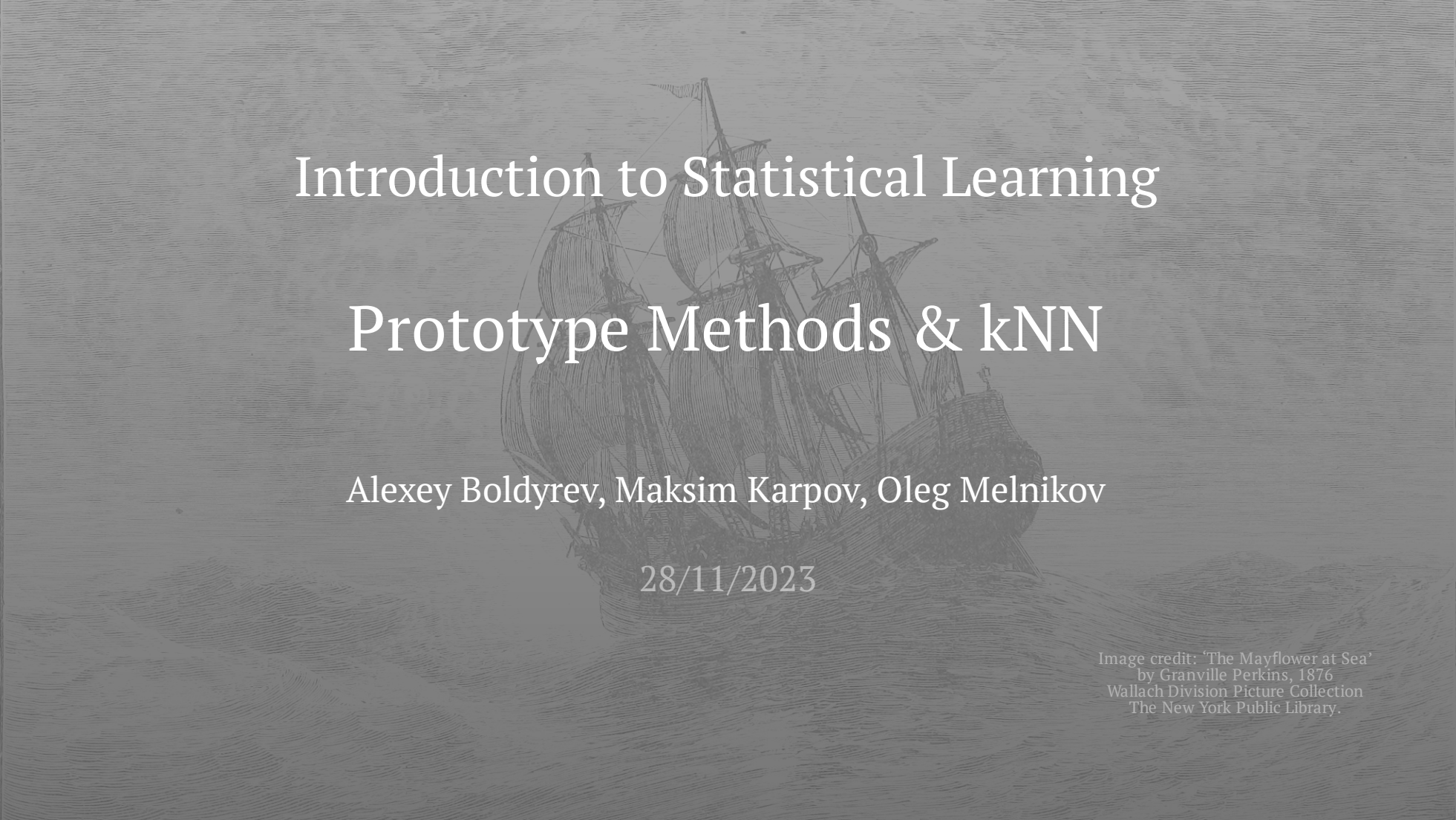


Image credit: 'The Mayflower at Sea'
by Granville Perkins, 1876
Wallach Division Picture Collection
The New York Public Library.

Lecture Attendance 28/11/2023

Log in with QR Code

Outline

- Prototype Methods
- Gaussian Mixtures
- k-Nearest Neighbors
- Invariant Transformations

Prototype Methods

Image credit: Midjourney

prompt: ‘prototype method in computer science’

Prototype Methods

Here we mix unsupervised and supervised methods

- Def.: $\mathcal{T} := \{(x_i, g_i\}_{1:N}$ taining observations with $g_i \in \{1, ..., K\}$ class labels, $x_i \in \mathbb{R}^p$
- Prototype models:
  - non-parametric, no fixed number of parameters
    - harder to interpret I/O relation (varies locally)
      - vs. logistic regression,

        where I/O relationship is global
  - effective black box prediction
  - represent $\mathcal{T}$ with **prototype** points in feature space $\mathbb{R}^p$, but (typically) **not in** $\mathcal{T}$
- Prototypes have class labels
- Test points are assigned to the "closest" prototype; Euclidean metric is "natural"

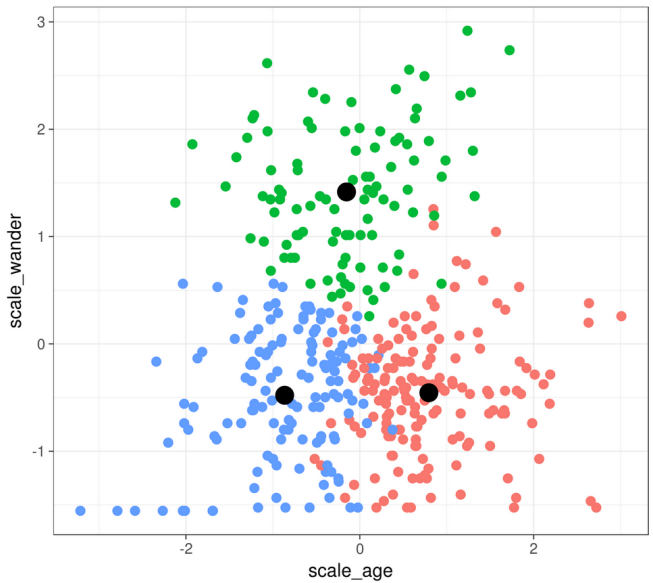


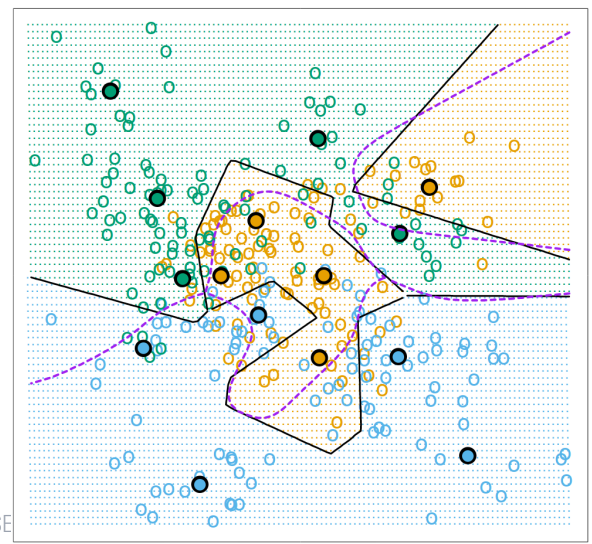
Image source: https://rpubs.com/cyobero/k-means

k-Means, k-Medians, k-Medoids

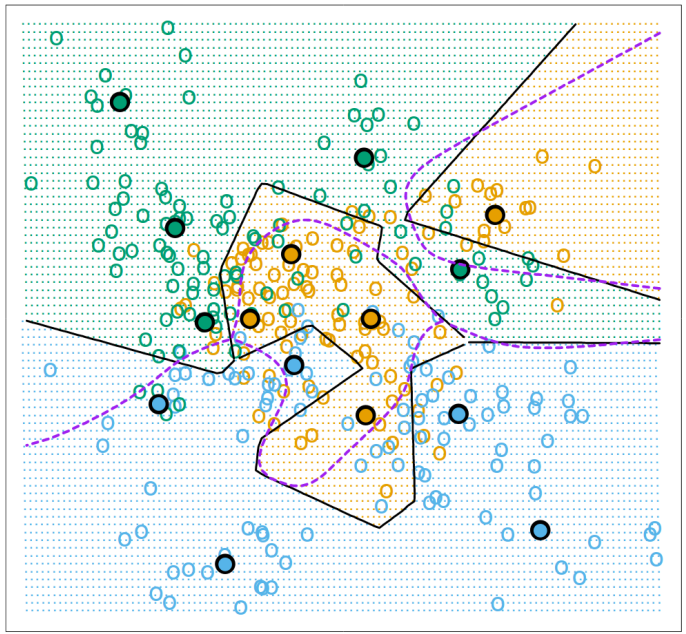
- **k-Means Algorithm**:
  - Initialize with R random cluster centers (each centroid is a **mean** of points in a cluster)
  - Assign each point to the L_2-closest centroid
    - This yields the smallest **intra-cluster variance** (sum of squared errors)
  - Recompute centroids and repeat till convergence
- **k-Medoids**:
  - Same as k-means, but allows **any** distance metric and cluster centers **must be** medoids
    - Def: **medoid** is a representative point of the sample and from the sample
    - Easier to interpret
- **k-Medians** (often confused with k-medoids!):
  - Same as k-means, but assigns each point to the L_1-closest centroid (minimizes L_1 within-cluster error)
    - Uses median of each attribute of points in a cluster.
      - I.e. $\text{med}\left(\begin{bmatrix} 0 \\ 3 \end{bmatrix}, \begin{bmatrix} 2 \\ 3 \end{bmatrix} \right) = \begin{bmatrix} 0 \\ 3 \end{bmatrix}$

k-Means for Classification of Labeled Observations

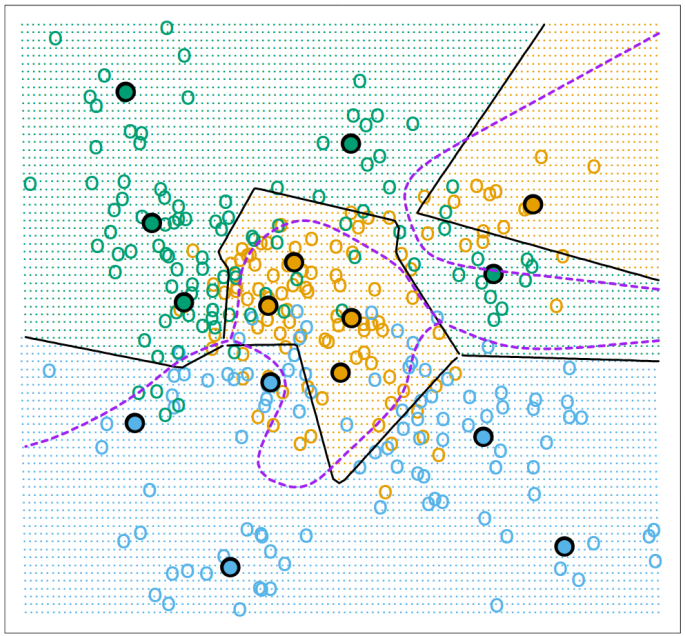
- For each class k:
  - Use k-Means to find R centroids (i.e. prototypes) and label them with class k
- Classify new test point to the class of the closest prototype
- Decision boundary is equidistant between two neighboring prototypes (of differing classes)
- Inter-class prototypes
  - near boundaries yield more misclassifications
  - are *constructed* independently
- Ex. ESL Fig 13.1
  - x_1 is among greens, but is classified to blue
    - b/c nearest prototype is blue
  - x_2 is among greens, but is classified to orange



Learning Vector Quantization (LVQ)



k-Means 5 prototypes & 3 classes



LVQ 5 prototypes & 3 classes

- Prototypes are away from decision boundary: this improves classification near boundary
- Train points attract intra-class prototypes and repel inter-class prototypes
- LVQ is an **online** algorithm, i.e. observations are processed one at a time

LVQ1 Algorithm

1. Initialize:
   1. R prototypes for each class, $m_1(k), ..., m_R(k)$ for $k = 1 : K$
   2. Learning rate $\epsilon > 0$
2. Bootstrap (x_i, g_i) and assign x_i it to the **closest prototype** $m_j(k)$
   - Update prototype:

$$m_j(k) \leftarrow m_j(k) + \epsilon\big(x_i - m_j(k)\big) \cdot \begin{cases} 1\,, & \text{if } g_i = k \\ -1, & \text{if } g_i \neq k \end{cases}$$

     - If true and predicted labels match, move the prototype closer to x_i, otherwise, move it away from x_i
   - Reduce ϵ
3. Repeat step 2

Gaussian Mixtures

Image credit: Midjourney

prompt: ‘gaussian mixtures’

Gaussian Distribution

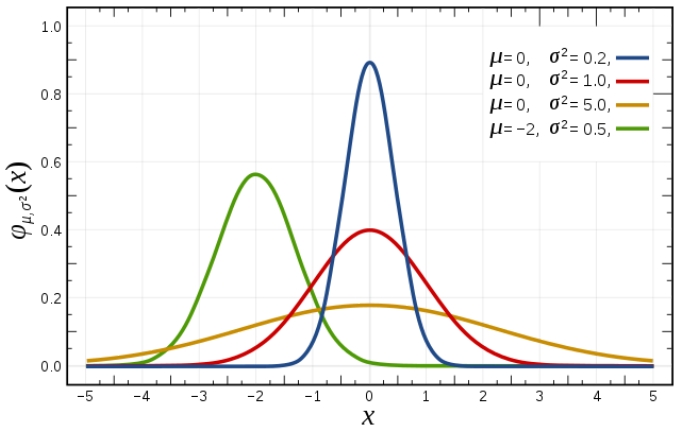


Image source: https://en.wikipedia.org/wiki/Normal_distribution

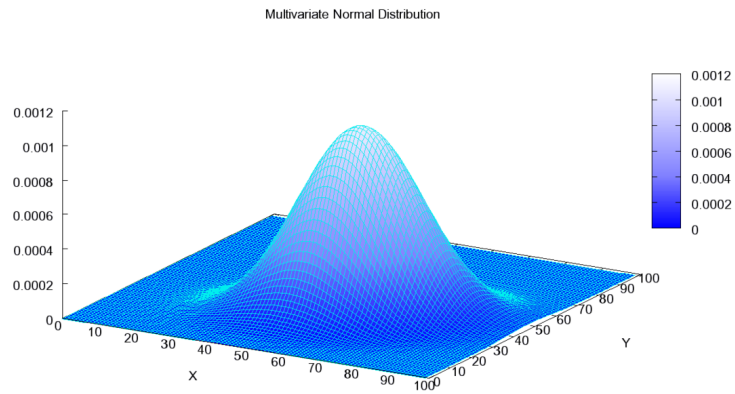


Image source: https://en.wikipedia.org/wiki/Multivariate_normal_distribution

- For d dimensions, the Gaussian distribution of a vector $x = (x_1, x_2, ..., x_d)^T$ is:

$$N(x|\mu, \Sigma) := \frac{1}{(2\pi)^{d/2}\sqrt{|\Sigma|}} \exp(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu))$$

Mixture Models

- Formally a Mixture Model is the weighted sum of a number of pdfs where the weights are determined by a distribution, π

$$p(x) = \pi_0 f_0(x) + \pi_1 f_1(x) + \ldots + \pi_k f_k(x) = \sum_{i=0}^{k} \pi_i f_i,$$

where $\sum_{i=0}^{k} \pi_i = 1$

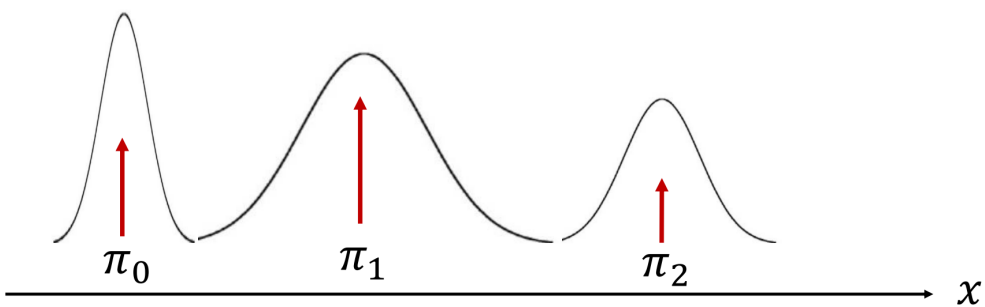


Image source: https://nakulgopalan.github.io/cs4641/course/20-gaussian-mixture-model.pdf

Mixture Models

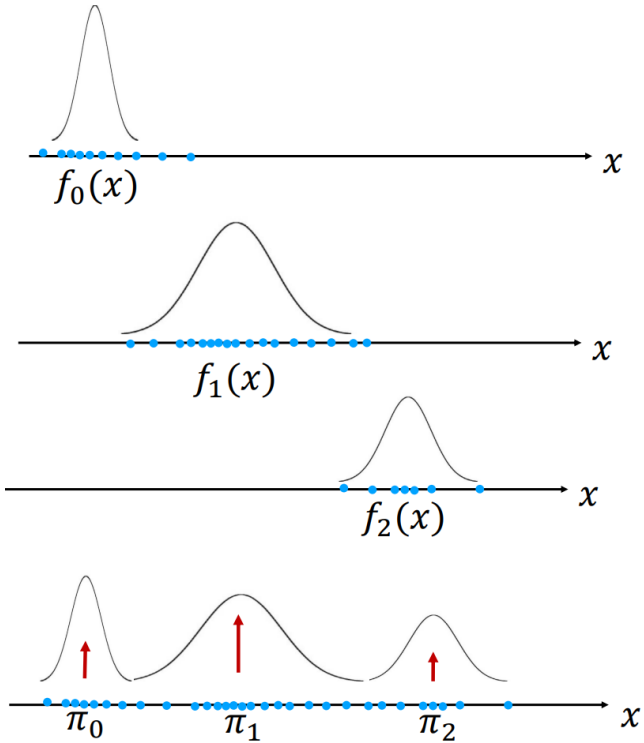


Image source: https://nakulgopalan.github.io/cs4641/course/20-gaussian-mixture-model.pdf

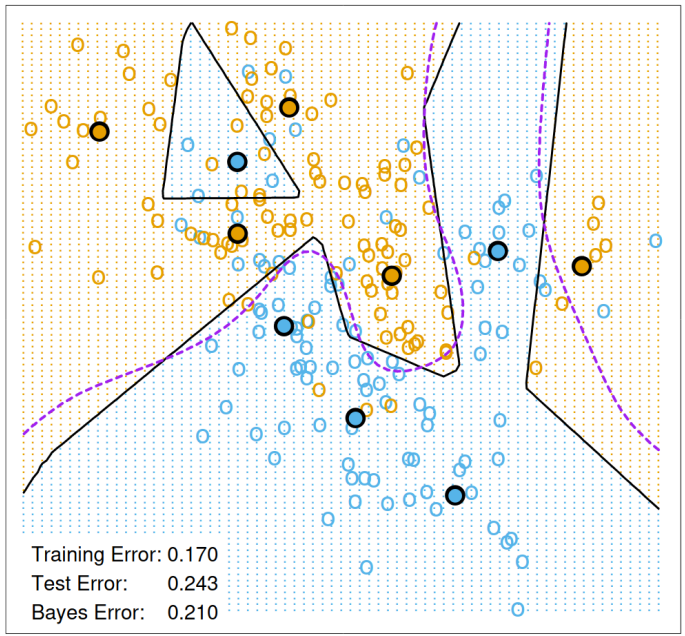
$$p(x) = \pi_0 f_0(x) + \pi_1 f_1(x) + \pi_2 f_2(x)$$

- Mixture model creates a new pdf for us to generate random variables. It is a generative model.
- It clusters different components using a Gaussian distribution.
  - So it provides us the inferring opportunity. Soft assignment!

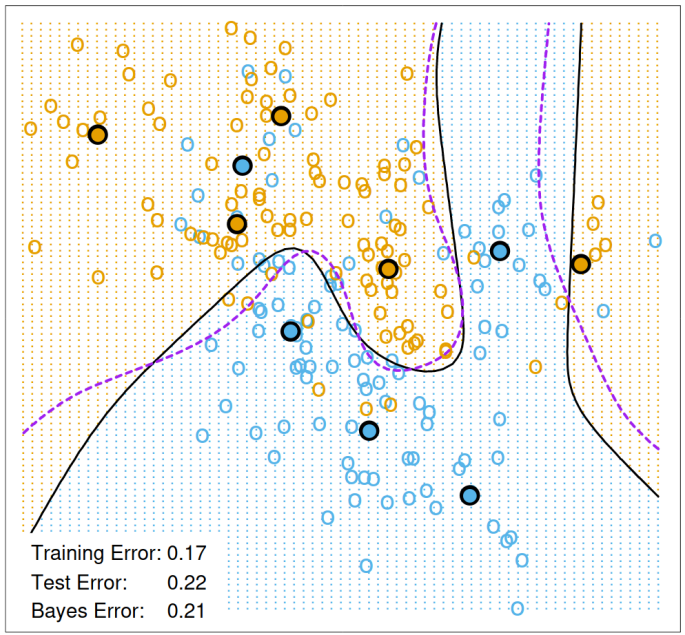
Gaussian Mixtures Model (GMM)

- Each cluster r is represented as $\mathcal{N}\left(\mu_{r,p\times 1}, \Sigma_{r,p\times p}\right)$
- We use **expectation maximization (EM) algorithm** to estimate R Gaussian PDFs:
  - **E-step**: assign **responsibility** p to each x_i
    - $p' = [p_1, ..., p_R]$ are probability weights with $p_r := \mathbb{P}[g_i = r | \text{all params}]$ that x_i is in each of R clusters
  - **M-step**: update R Gaussian parameters $\left(\mu_r, \Sigma_r\right)$ for the current responsibilities
    - each observation contributes to weighted means and covariances for each cluster
- GMM is **a soft clustering** prototype model
  - Recall: K-Means is **hard clustering**

Example: GMM vs K-Means (ESL Fig. 13.2)



K-Means 5 prototypes & 2 classes



GMM 5 prototypes & 2 classes

- GMM has smoother boundaries vs K-Means has piecewise linear decision boundary
- GMM derives a better decision boundary. Both models estimate one wrong prototype

k-Nearest Neighbors

Image credit: Midjourney

prompt: ‘k-Nearest Neighbors’

k-Nearest Neighbor (kNN) Classifiers

- All train points are used in testing
  - There is no reduction in train set
- In 1-NN, every point is a prototype
- kNN algorithm
  1. Standardize features as $x_i \leftarrow \frac{x_i - \bar{x}}{\sigma_{x_i}}$ (to have 0 mean, unit variance)
     - Otherwise, **smaller scale** features are **dominate** the neighborhoods
     - Typically, any L_2 distance-based algorithm can benefit from such feature normalization
  2. Classify the query point x_0, to the **mode (majority)** of labels of k closest training points
- kNN performs well with
  - "irregular" (complex) decision boundaries
  - many prototypes (i.e. many clusters)
- We use Cross Validation (CV) to choose k

Comparative Study

- K-Means vs LVQ vs kNN on 2 simulated problems:
  - All have $X_1, \ldots, X_{10} \overset{\text{iid}}{\sim} \mathcal{U}(0, 1)$
- **Easy**: $Y := I(X_1 > 0.5)$
  - 9 features are ignored (just noise in 9 dimensions)
- **Hard**: $Y := I\big(\text{sgn}\big[\prod_{j=1}^{3}(X_j - 0.5)\big] > 0\big)$
  - 7 features are ignored

Comparative Study: Misclassification Error

- kNN:
  - outperforms on difficult problem
  - underperforms on easy problem
- K-Means and LVQ perform similarly

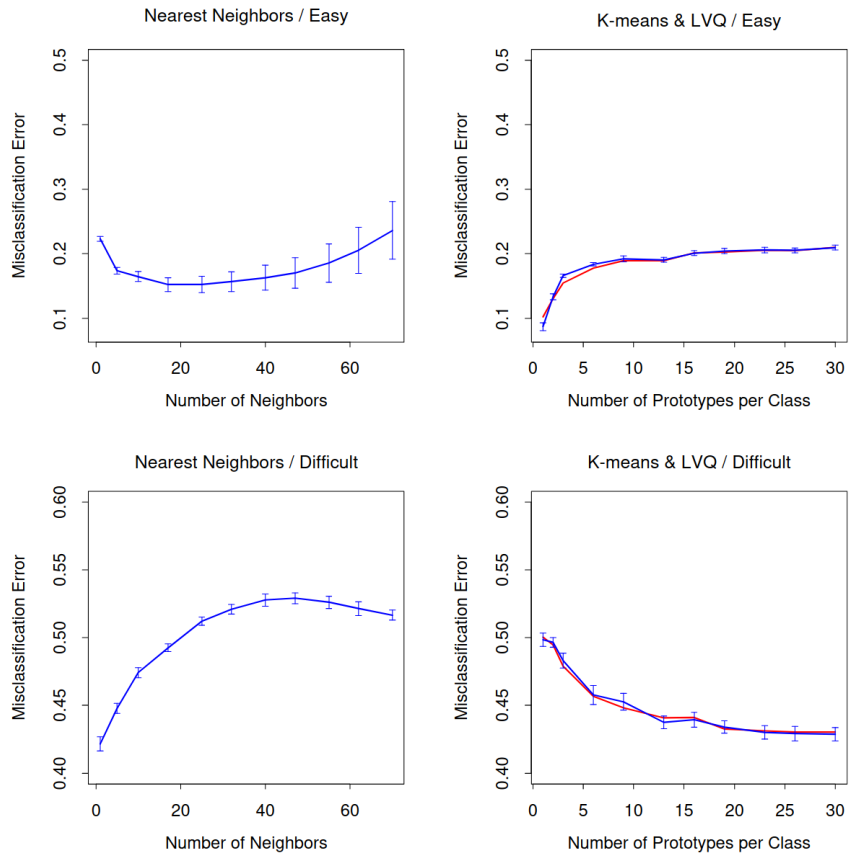
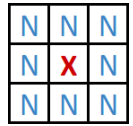


Image source: ESL Fig. 13.5

Ex: kNN for STATLOG Project



- **Goal**: classify pixels of aerial images into 7 agricultural land use classes: cotton, vegetation stubble, mix, red soil, gray soil, damp gray soil

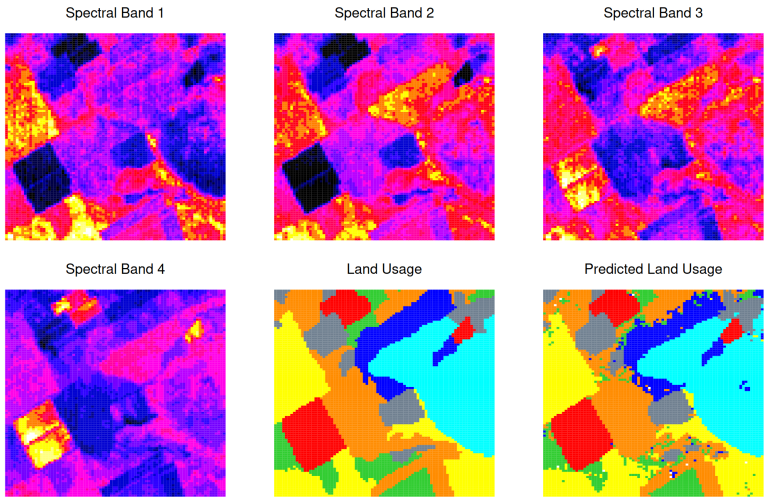


Image source: ESL Fig. 13.6

- Each $x_i \in \mathbb{R}^{36}$ is built with 9 adjacent pixels from 4 spectral images
- 5-NN resulted in 9.5% error
  - Better than many classifiers:

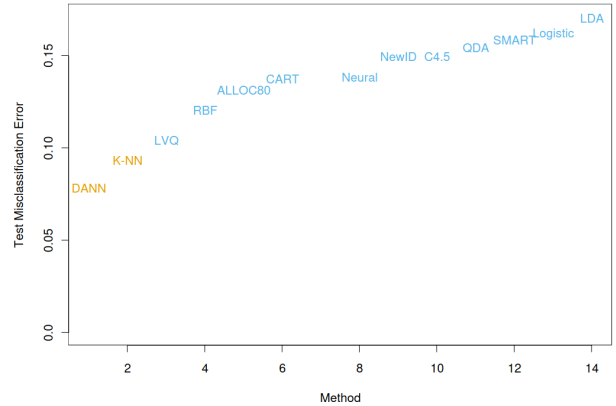


Image source: ESL Fig. 13.8

Invariant Transformations

Image credit: Midjourney

prompt: ‘invariant transformations’

Invariant Transformations

- Digit classifier should ignore these transforms:
  1. Small clockwise/anticlockwise rotations
     - Large rotations work for digits other than 6-9
  2. Horizontal/vertical translation (i.e. shifts)
  3. Horizontal/vertical scaling
  4. Shear
  5. Character thickness
- However, these transforms produce digit vectors, which are far apart in $\mathbb{R}^{256}$

16 x 16 = 256 pixels, gray scale

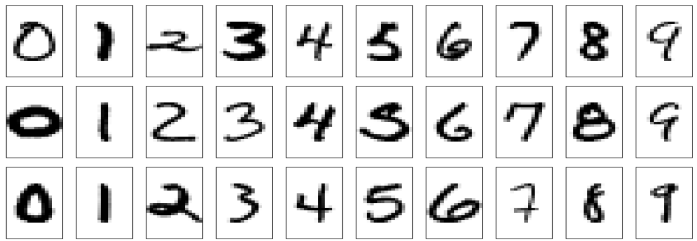


Image source: ESL Fig. 13.9

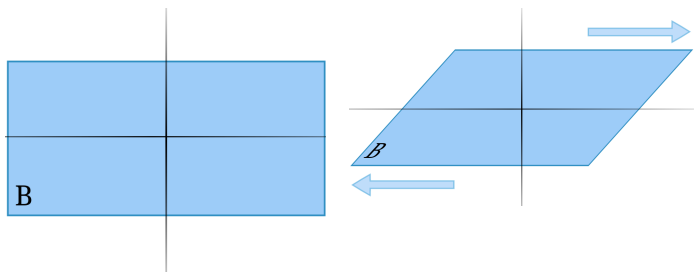


Image source: https://en.wikipedia.org/wiki/Shear_force

Image source: ESL Fig. 13.10 (top)

Invariance Manifolds

- As we rotate the image by angle α, we generate a 1D trajectory curve (**invariance manifold**) in $\mathbb{R}^{256}$ pixel space
- Let $x_1, x_2 \in \mathbb{R}^{256}$ be digit pixel vectors with invariance manifolds m_1, m_2
- **Invariant metric** is the shortest Euclidean distance b/w curves m_1, m_2
- Problems:
  - It's very computation intensive
  - It works better with longer curves, i.e. large transforms, but large transforms cause problems (e.g. 6 vs 9)

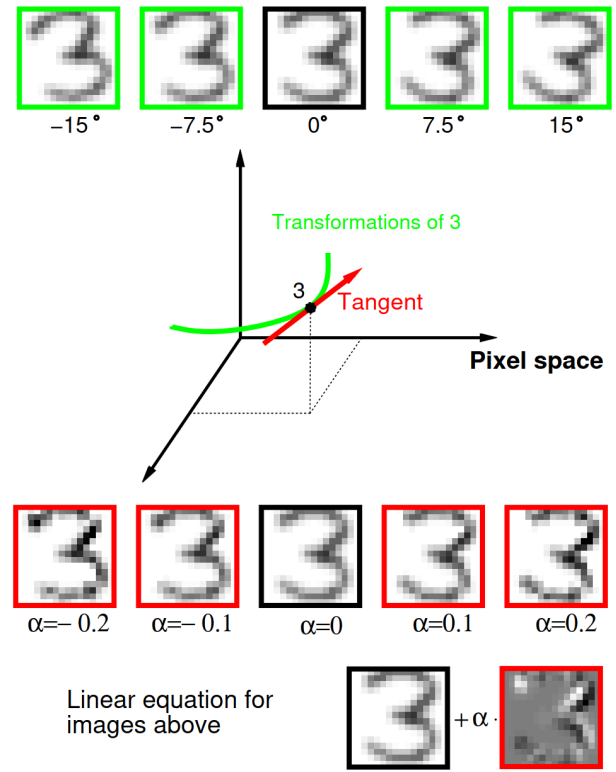


Image source: ESL Fig. 13.10

Tangent Distance

- **Invariant tangent line** t_i estimates the direction vector from **small** rotations of the image x_i
  - Solves both problems of invariant metric
  - This is a tangent line at the point of original image
- **Tangent distance** t_{12} is the shortest distance between tangent lines t_1, t_2
- To apply 1-NN:
  - Given a query image x_0 compute its t_0
  - Find training image x_i with smallest t_{i0}
  - $\hat{g}_0 := g_{\hat{\imath}}$, where $\iota := \underset{i}{\operatorname{argmin}} t_{i0}$ where $\hat{g}_0$ is the predicted class of x_0

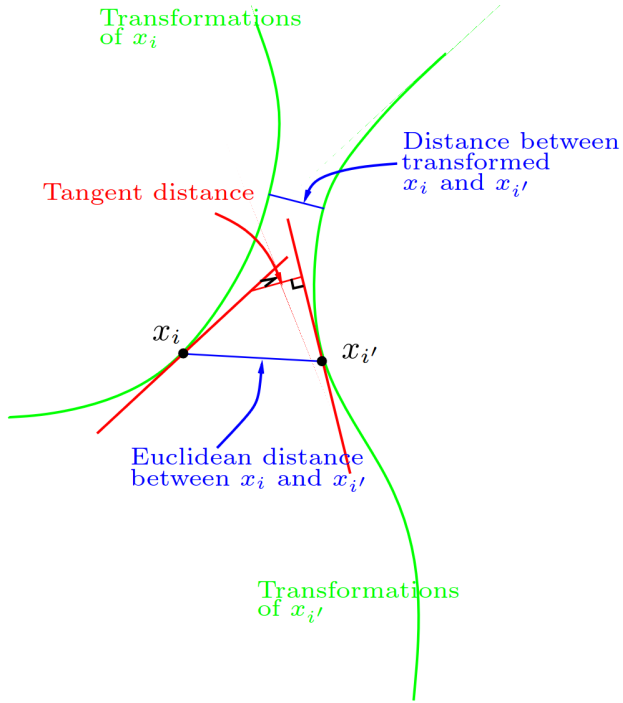


Image source: ESL Fig. 13.11

Back to 7 Transformations

- Transforms: rotations, 2x translations, 2x scaling, sheer, character thickness
- Each adds one more dimension to the curve in
- So, we compute **7D tangent hyperplanes** and distances among their tangent hyperplanes
- Takeaway:
  - Never underestimate **feature engineering**
  - Always investigate:
    - **Why** the observations are misclassified and
    - **How** you can express the distinctive features of misclassified observations

Method	Error rate
Neural-Net	0.049
1-NN / Euclidean distance	0.055
1-NN / tangent distance	0.026

kNN in High Dimensions (HD)

- Curse of dimensionality: in HD, points are spread out and are "near" the boundaries
  - kNN performance degrades
- Consider unit cube in $\mathbb{R}^p$, $\left[-\frac{1}{2}, \frac{1}{2}\right]^p$
- R = radius of 1-NN at $(0, 0)$, which approaches 0.5 as p grows
  - Median of R is computed over different training sample sizes

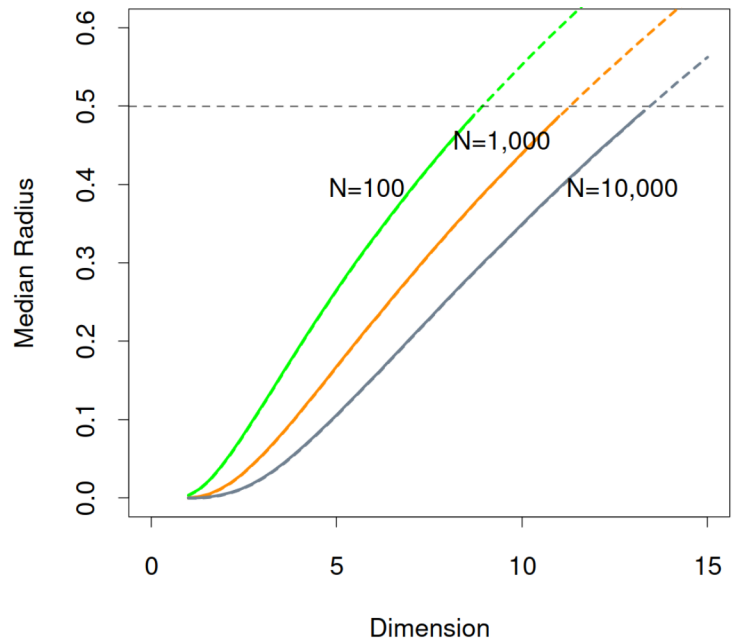


Image source: ESL Fig. 13.12

A Case for Adaptive Metric (Ex. ESL Fig. 13.13)

- A spherical neighborhood misclassifies x_0 as red, but it's in green region
- b/c Euclidean metric assumes

  uniform distribution of points in the disk
- but green points are clustered at the top
- A rectangular neighborhood classifies x_0 correctly as green
- Goal: adapt a metric to stretch the neighborhood in directions where class probabilities are stable
- How:
  - For a query point x_0 find N_0 (say 50) neighbors
  - Distribution of neighbors determines the metric
- Stretch the neighborhood in the direction **orthogonal to the line** ℓ joining the class centroids

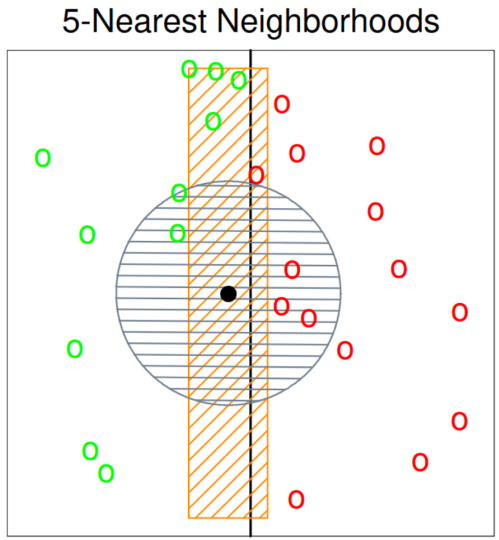


Image source: ESL Fig. 13.13

Discriminant Adaptive Nearest-Neighbor (DANN)

- DANN metric is $D(x, \nu) = (x - \nu)'\Sigma(x - \nu)$,

  where:

  - $\Sigma := W^{-\frac{1}{2}}\left[W^{-\frac{1}{2}}BW^{-\frac{1}{2}} + \epsilon I\right]W^{-\frac{1}{2}} = W^{-\frac{1}{2}}\left[B^* + \epsilon I\right]W^{-\frac{1}{2}}$
  - $W := \sum_{k=1}^{K} \pi_k W_k$ is the pooled **intra-class covariance**
  - $B = \sum_k \pi_k(\bar{x}_k - \bar{x})(\bar{x}_k - \bar{x})'$ is **inter-class covariance**
  - ϵ rounds the rectangular neighborhood to an ellipsoid to avoid use of points in corners of a rectangle
    - $\epsilon = 1$ works well

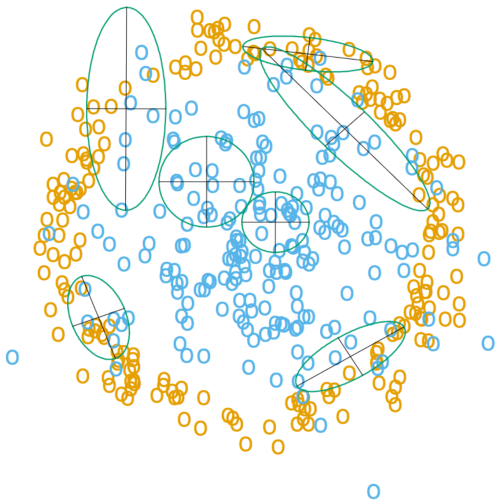


Image source: ESL Fig. 13.14

- The formula:
  - Spheres the data (pulls distant points closer to x_0)
  - Then stretches the neighborhood in the zero-eigenvalue direction of B^*

END