Lab 3

K-NN and Parzen windows

Introduction

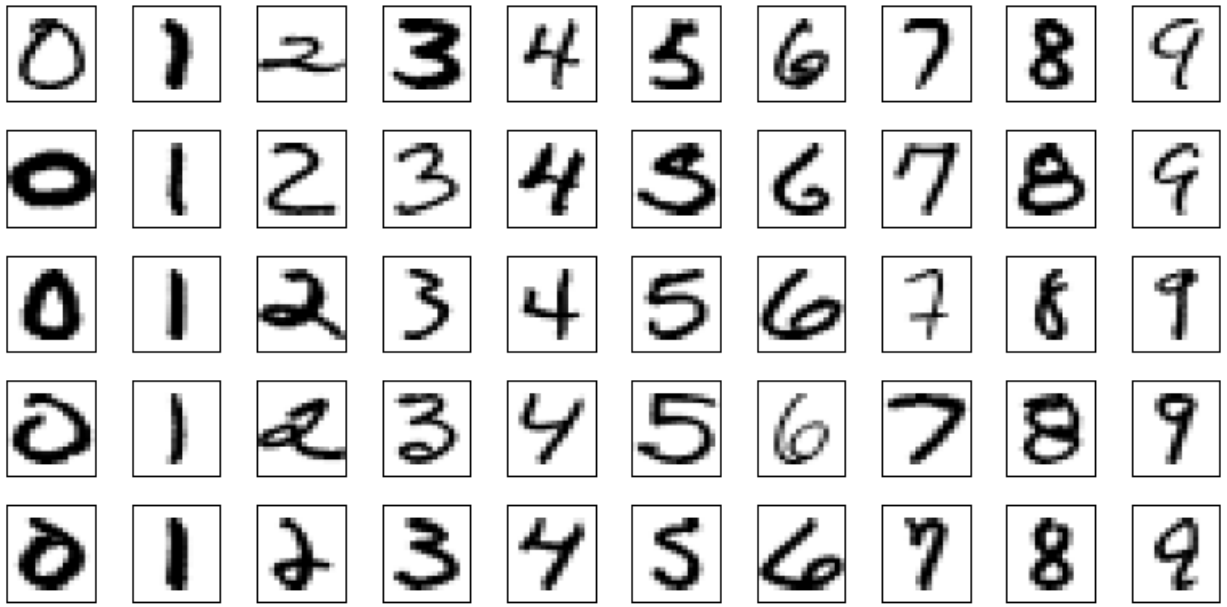
- Objectives

  – Use and compare two non-parametric classifiers, k-nearest neighbors and Parzen windows.
  – Test these methods on two real datasets, an image dataset of handwritten digits and a small high dimensional microarray dataset.
  – Implement a cross-validation strategy for hyperparameter selection.

ZIP dataset

- Each vector of d=256 features represents an image of size 16x16 (read by raws) from one of ten classes corresponding to handwritten digits 0 to 9.

- **Parameters**
  - total number of vectors in the Training dataset: 7291
  - total number of vectors in the Test dataset: 2007
  - initial dimension of the vectors d=256
  - number of classes c=10



Microarray datset

- Each vector of d=6830 features represents genetic information from different organs affected by cancer
- N= 64 samples
- c=14 cancer types

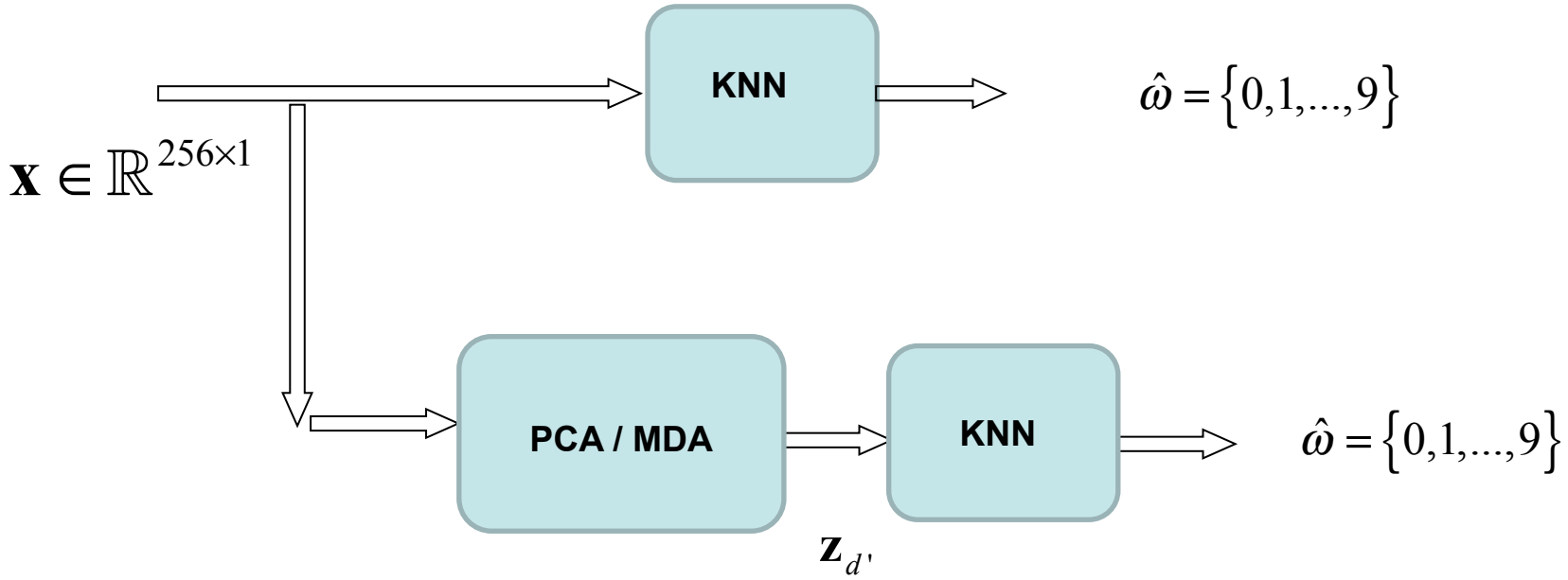
1: CNS 5 samples
2: RENAL 7 samples
3: BREAST 9 samples
4: NSCLC 9 samples
5: UNKNOWN 1 samples
6: OVARIAN 6 samples
7: MELANOMA 8 samples
8: PROSTATE 2 samples
9: LEUKEMIA 6 samples
10:K562B-repro 1 samples
11:K562A-repro 1 samples
12:COLON 7 samples
13:MCF7A-repro 1 samples
14:MCF7D-repro 1 samples

k-nearest neighbors

For each test vector find the ”k” closest training vectors and predict the most popular class among the K training vectors (majority voting)

Example: ZIP dataset. There are 10 classes (digits 0 to 9). Vectors to classify are images of handwritten digits

lab3_zip.m



Feature selection

- PCA

$$\mathbf{z}_n = \underbrace{\left[\mathbf{w}_1,..,\mathbf{w}_{d'}\right]}_{d' \text{ direcciones principales}}^T \left(\mathbf{x}_n - \mathbf{m}\right) = \mathbf{W}_{PCA}^T \left(\mathbf{x}_n - \mathbf{m}\right) \qquad \hat{\mathbf{x}}_n = \mathbf{W}_{PCA}\mathbf{z}_n + \mathbf{m}$$

  - each column w_i represents an eigen-image
  - function pca.m

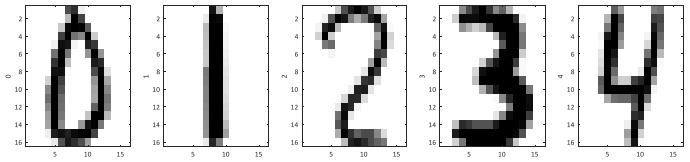
- MDA

$$\mathbf{z}_n = \mathbf{W}_{MDA}^T \mathbf{x}_n \qquad \hat{\mathbf{x}}_n = \mathbf{W}_{MDA}\left(\mathbf{W}_{MDA}^T \mathbf{W}_{MDA}\right)^{-1}\mathbf{z}_n$$

$$d' \leq \min(N_{clases} - 1, d)$$

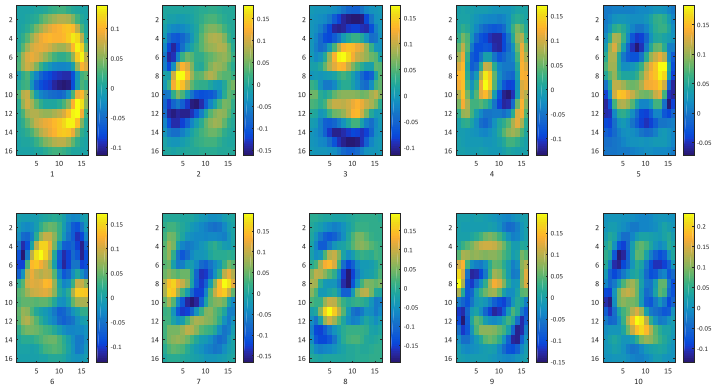
  - function mda_ml.m
  - note: labels are numbered 1... $N_{classes}$

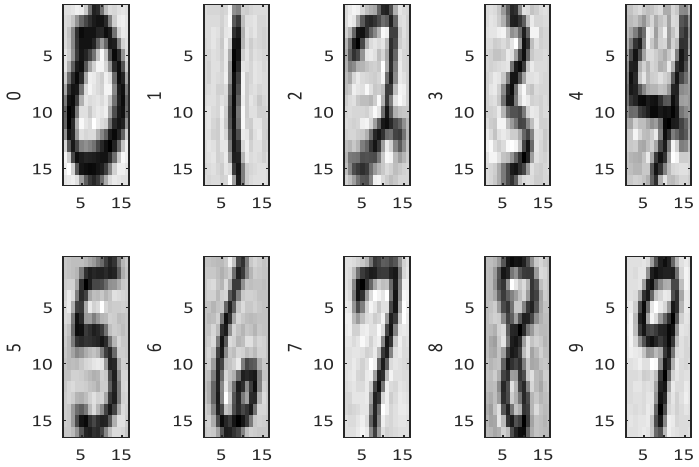
Example: dimensionality reduction from 256 to 64 features using PCA



Sample original images

Eigenimages

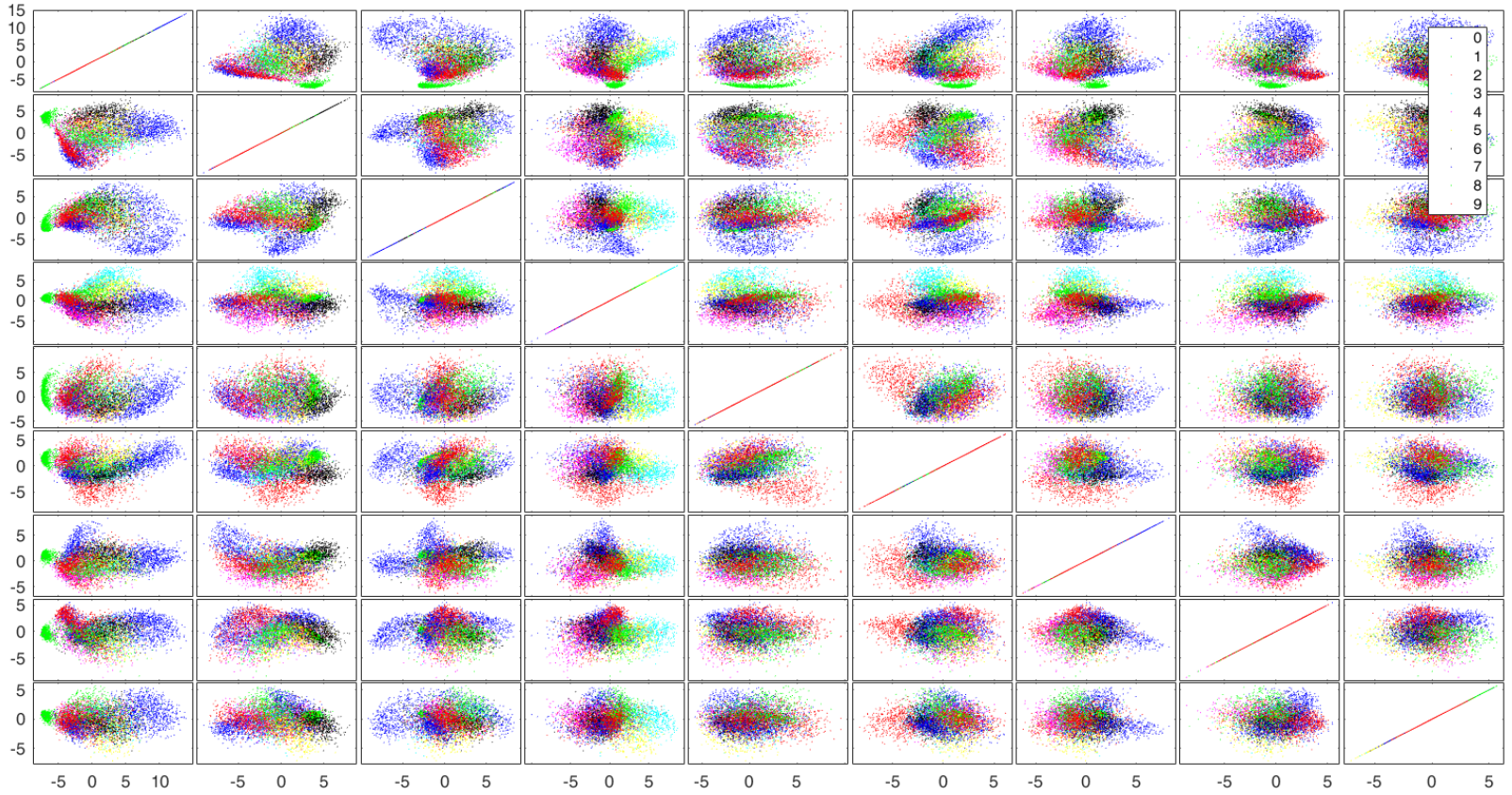




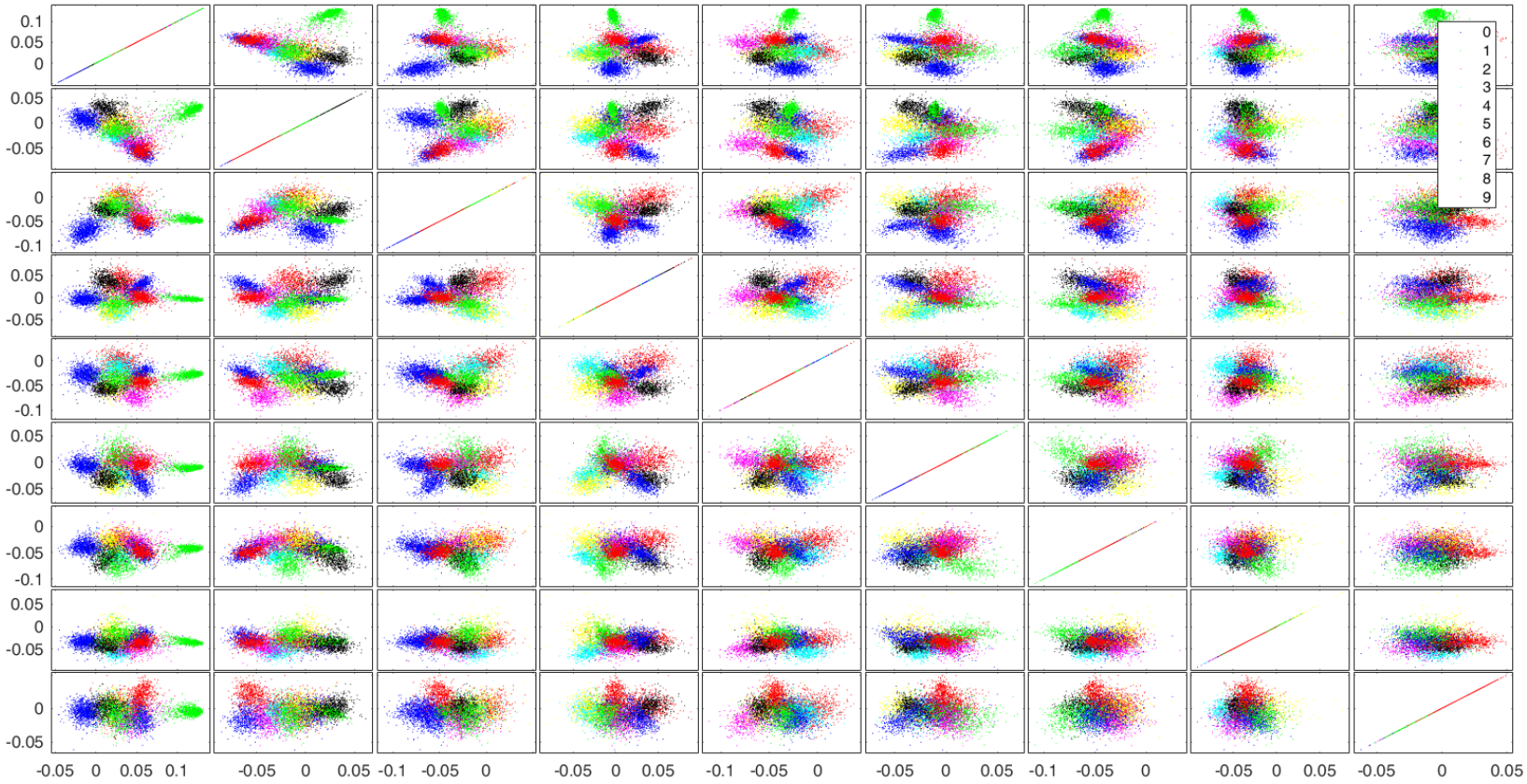
Sample reconstructed images

Can we still recognize digits using the reconstructed images?

Example: scatter plot using PCA (9 features)



Example: scatter plot using MDA (9 features)



Parzen windows

Estimation of pdf conditioned to class c using n training vectors:

$$\hat{f}(\mathbf{x}|c) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{V_{R,n}}\varphi\left(\frac{\mathbf{x}-\mathbf{x}_i}{h_n}\right) = \frac{1}{n}\sum_{i=1}^{n}\gamma_c\left(\mathbf{x}-\mathbf{x}_i\right)$$

test vector to classify

training vectors

The function `predict_parzen.m`, uses a Gaussian window:

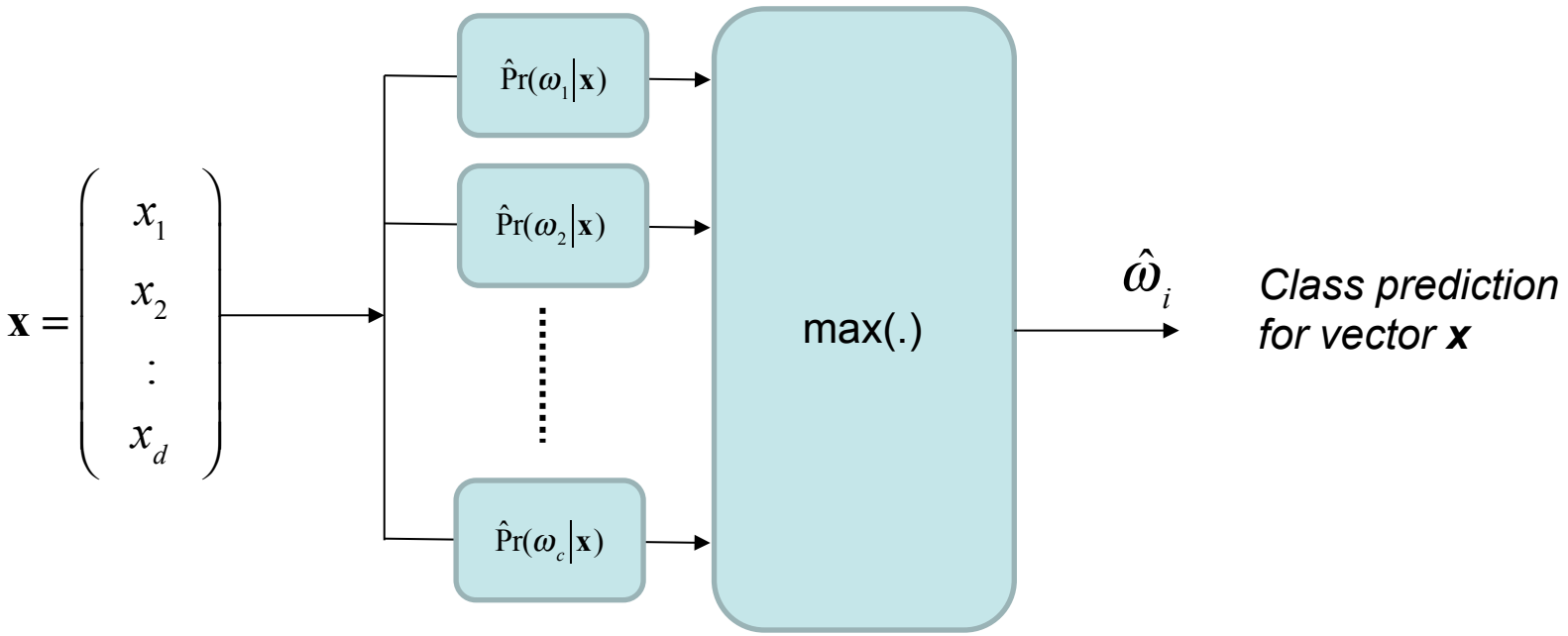
$$\gamma_c\left(\mathbf{x}-\mathbf{x}_i\right) = \frac{1}{h_n^{\,d}}\frac{1}{\sqrt{(2\pi)^d|\mathbf{C}_c|}}\exp\left(-\frac{1}{2}\left(\frac{\mathbf{x}-\mathbf{x}_i}{h_n}\right)^T\mathbf{C}_c^{-1}\left(\frac{\mathbf{x}-\mathbf{x}_i}{h_n}\right)\right) \qquad \int\gamma_c(\mathbf{x})\,d\mathbf{x} = 1$$

where *d* is the number of features.

The parameter that determines the width of the window is $h_n = \frac{h}{\sqrt{n}}$

This function returns the predicted class for each input vector

Parzen classifier with c classes



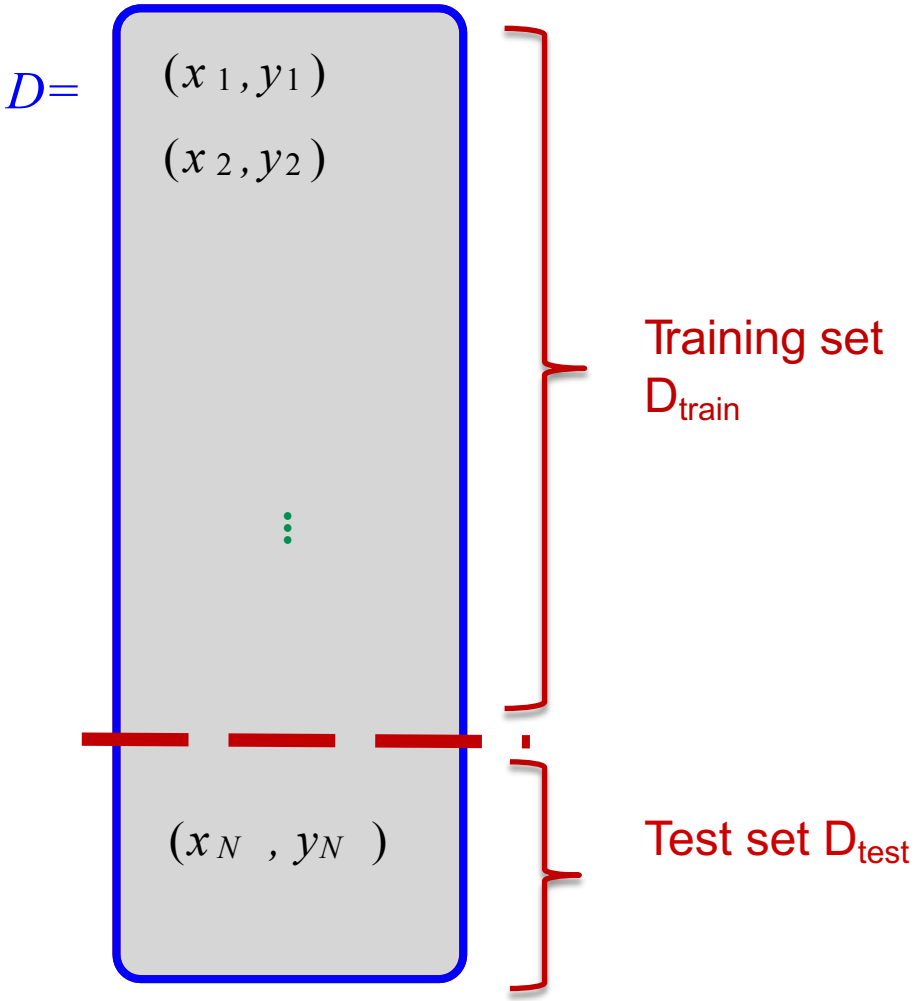
```
function Predict_test = predict_parzen
   (X_train,Labels_train,N_classes,h,X_test)
```

Parzen Windows are computed from the Train set

Introduce Train, Test or any set of vectors to be labeled

```
% Parzen classifier with gaussian window
% X_train: matrix, rows contain train vectors
% Labels_train: Assumed to be 0,1,2,....N_classes-1
% N_classes: Number of different classes
% h: Window width parameter
% X_test: matrix, rows contain vectors to be labeled
% Predict_test: Labels given to the test set
```

Simple train-test procedure

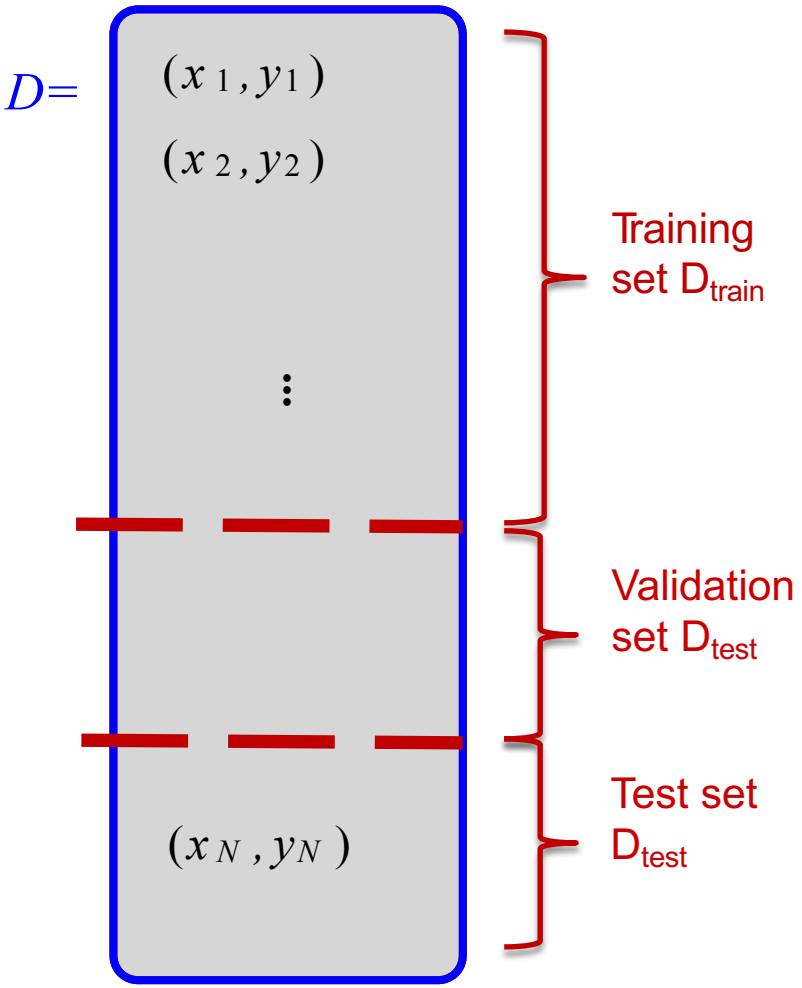


- Provided large enough dataset D drown from p_{data}
- Arrange samples in random order
- Split dataset in two: D_{train} and D_{test}
- Use D_{train} to find the best predictor f
- Use D_{test} to evaluate the generalization performance of f

Cross validation

Make sure examples are in random order

Split data D in 3: D_{train} D_{valid} D_{test}



- Provided large enough dataset D drown from p_{data}
- A general procedure for estimating the true error of a predictor
- Data is split into subsets
- Training and Validation set used only to find the right predictor (**optimize hyperparameters**)
- A Test set used to report the prediction error of the algorithm
- The sets must be disjoint
- The process is repeated several times, and the results are averaged to provide error estimates

k-fold cross validation

- For small datasets (but enough data to keep an independent test set)
- Divide training data into k equal sized folds: train on k-1 folds,and validate on the remainder fold
- Compute average performance across the k iterations

How many folds?

- too few, and effective training set much smaller
- too many, and test performance estimates noisy
- **Cost:** must run training algorithm once per fold
- **Practical rule of thumb:** 5-fold or 10-fold cross validation
- **LOOCV:** leave-one-out cross-validation, K=N (when the dataset is very small)

K-fold cv for parameter selection

- We want to find the optimum value of a parameter p (e.g. h for Parzen, k for k-NN) from a set of possible values $\{p_1,...,p_N\}$
- We want this value to be as independent as possible from the (random) train/test partition

- For K_0=1,...,K (folds)
  - for p=p1,...p_N (parameter values)
    - train the classifier using parameter p on the K-1 training folds
    - validate on the remaining fold
  - end

  end
- Compute average validation errors for each value of p. Select p_{opt} the parameter with min average error
- Test the classifier (using p_{opt}) on the test set (retrain on the whole training set)

K-fold cv for parameter selection

How to implement cv in Matlab?

- given traning dataset with N labeled samples

- **cp = cvpartition(N,'kfold',K_folds);**

  performs a random partition of dataset into *K_folds* independent folds. Returns results in cp struct.

- **logic_vector_valid = test(cp,K_0);**

  returns a vector of size N with values '0' and '1'. Positions with '1' correspond to elements in fold K_0 to be used for validation

- **logic_vector_train = training(cp,K_0);**

  returns a vector of size N with values '0' and '1'. Positions with '1' correspond to elements to be used for training (all folds except k0)