POLITECNICO
MILANO 1863

Machine Learning Methods for Communication Networks and Systems

Francesco Musumeci

Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB)

Politecnico di Milano, Milano, Italy

Part I – 6: General concepts

Outline

- How to address overfitting
- Evaluating a learning algorithm
- Model selection
- Error metrics for unbalanced classes
- Dimensionality reduction
- Building a ML project

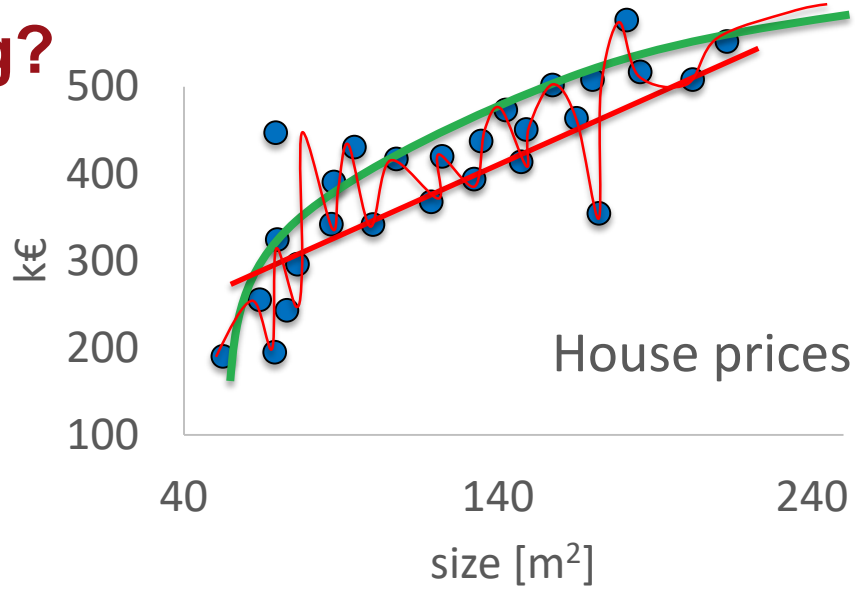
Outline

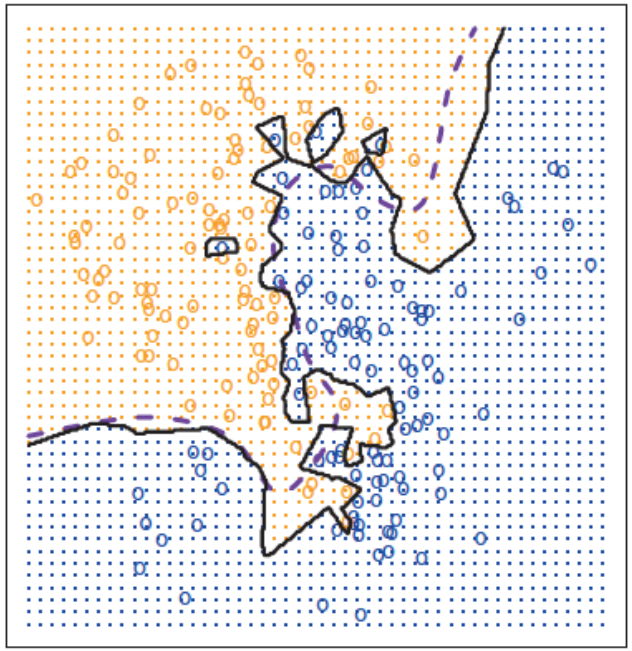
- How to address overfitting
- Evaluating a learning algorithm
- Model selection
- Error metrics for unbalanced classes
- Dimensionality reduction
- Building a ML project

How do we address overfitting?

The problem

- Very simple hypotheses can provide high MSE in the training set (underfit)
  - High bias
- Adding polynomial features or using “deep” NN improves training MSE but can fail in generalizing the response for future test examples (overfit)
  - High variance
- Trade-off between bias and variance is desirable
  - For both regression and classification problems



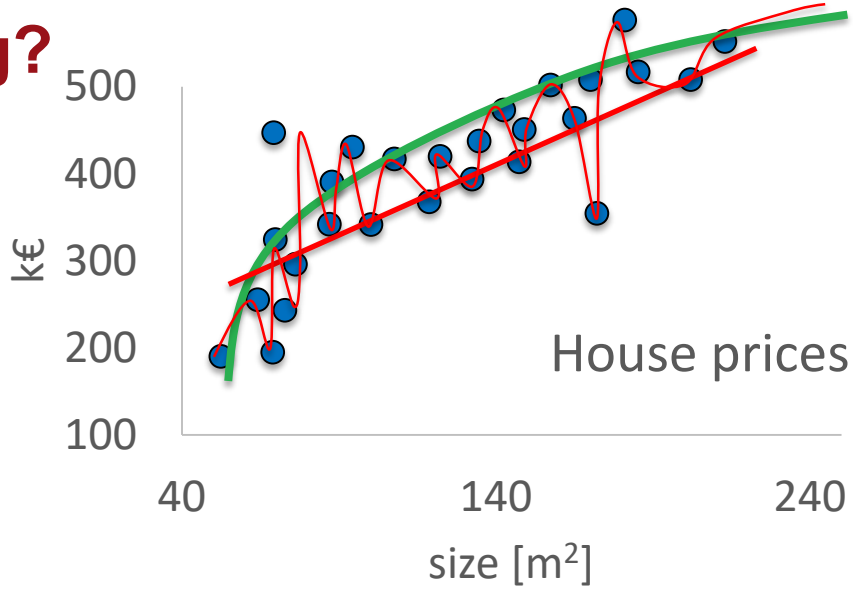


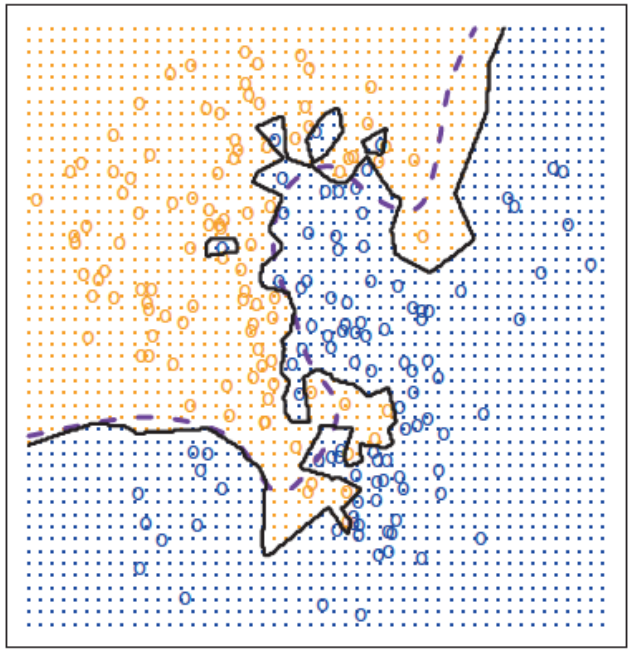
Source: ISLR

How do we address overfitting?

The solution(s)

- Use many features *while reducing the magnitude of parameters θ* (shrinkage methods)
  - Regularization (ridge regression)
  - Lasso
  - …
- Reduce dimensionality of the features space
  - "manually" select only a subset of features (can be very hard)
  - Get "principal" components (via PCA)
    - Transform original features in a new set of features which "sinthesize" the original information





Source: ISLR

How do we address overfitting?

Shrinkage methods: Regularization

- Aka "Ridge regression"
  - Change cost function minimizing *simultaneously* MSE and parameters θ

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h(\theta) - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

  - λ: regularization parameter
  - *λ is used to tune bias/variance*
    - High λ ➔ all θ's will tend to zero ➔ underfit (high bias – low variance)
      - N.B. a very large λ would give approx. $h(x)=\theta_0$
    - Low λ ➔ MSE will tend to zero ➔ overfit (high variance – low bias)

How do we address overfitting?

Shrinkage methods: Regularization

- Regularized **linear regression**
  - Cost function

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(\theta^T x^{(i)} - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

  - Gradient descent update rule (repeat until convergence):

$$\begin{cases} \theta_0 = \theta_0 - \alpha\dfrac{1}{m}\displaystyle\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)} \\ \theta_j = \theta_j - \alpha\left[\dfrac{1}{m}\displaystyle\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \dfrac{\lambda}{m}\theta_j\right] \end{cases}$$

$$h_\theta(x^{(i)}) = \theta^T x^{(i)}$$

simultaneous update

How do we address overfitting?

Shrinkage methods: Regularization

- Regularized **logistic regression**
  - Cost function

$$J(\theta) = -\left[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

  - Gradient descent update rule (repeat until convergence):

$$\begin{cases} \theta_0 = \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)} \\ \theta_j = \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j\right] \end{cases}$$

$$h_\theta(x^{(i)}) = \frac{1}{1+e^{-\theta^T x^{(i)}}}$$

simultaneous update

How do we address overfitting?

Shrinkage methods: Regularization

- Regularized cost function in **neural networks**
  - Cost function

$$h_\theta(x) \in R^K; \quad (h_\theta(x))_k : k^{th} \text{ element of vector } h_\theta(x)$$

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)(i)} \log(h_\theta(x^{(i)}))_k + (1 - y_k^{(i)(i)}) \log(1 - (h_\theta(x^{(i)}))_k)\right]\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\theta_{ij}^{(l)})^2$$

L : nr of layers in the NN

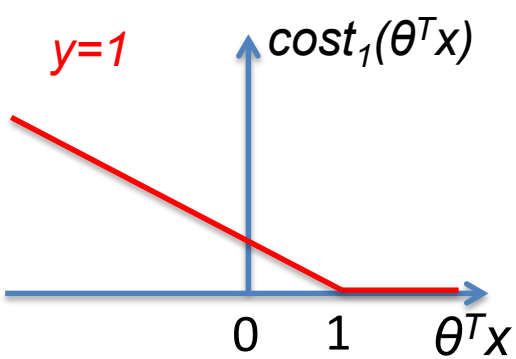
s_p : nr of neurons in p - th layer

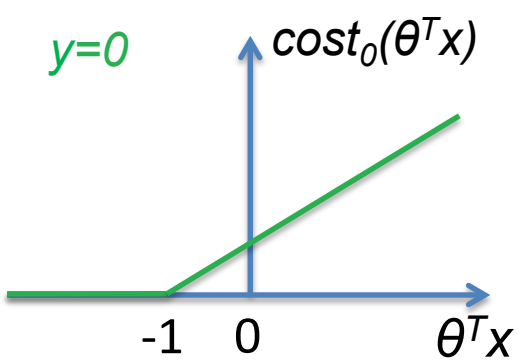
How do we address overfitting?
Shrinkage methods: Regularization

- Regularization in **SVM**
  - Cost function

$$J(\theta) = C\sum_{i=1}^{m}\left[y^{(i)}\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\text{cost}_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$$

Role similar to λ in regularized linear/logistic regression and Neural Nets





How do we address overfitting?

Shrinkage methods: Lasso

- Alternative to regularization
  - Change cost function minimizing *simultaneously* MSE and parameters θ

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{n}|\theta_j|\right]$$

  - Minimizing absolute values instead of squared values leads to have some of the parameters = 0 (not just near to 0)

How do we address overfitting?
Other techniques

- Dropout
  - Used in DNNs
  - At <u>training</u> time: for each layer of the DNN, define a "*keep_prob*" as the probability of keeping (=1-P{removing}) the nodes of the layer
    - Training at each epoch (or minibatch) is done by keeping nodes with such probabilities
  - At <u>test</u> time: no dropout (keep all the nodes), but reduce the activation values by a value *keep_prob* (*a=a/keep_prob*)
- Data augmentation
  - Create new examples as "distortions" of the training examples
  - Often used in image classification

Outline

- How to address overfitting
- Evaluating a learning algorithm
- Model selection
- Error metrics for unbalanced classes
- Dimensionality reduction
- Building a ML project

Evaluating a learning algorithm

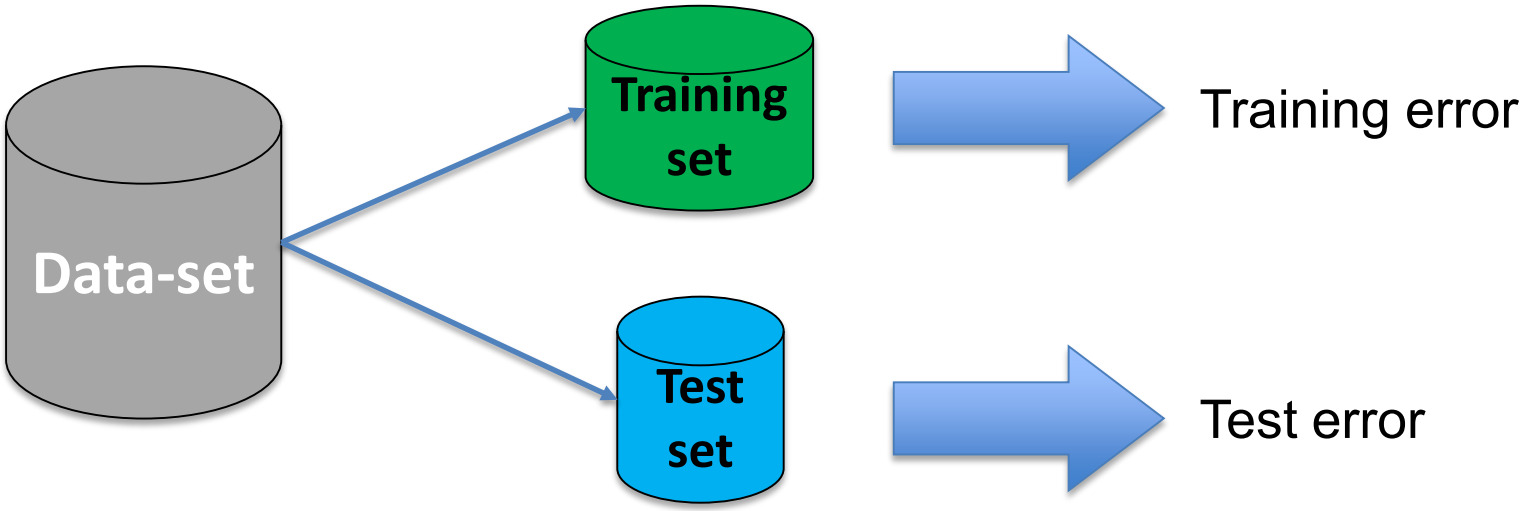
The problem

- Is it always good to get the minimum cost function $J(\theta)$?
  - How do we know we are not overfitting?
- Solution: use shrinkage methods!

- However…
  - How can we state that our final (regularized) hypothesis is good?
  - How do we measure the performance of our model?
  - How to choose the (hyper)parameters of each model?
    - Degree in a polynomial regression
    - Nr of hidden layers and neurons in neural networks
    - Type of Kernel in SVM
    - K in KNN
    - …

Evaluating a learning algorithm
The solution

- Available dataset is split into 2 separate sets:
  - Training set: used for model "creation", e.g., to choose parameters θ
  - Test set: used for model "selection", e.g., assess which model best fits to our data



Evaluating a learning algorithm

Test error

- Regression problems

$$J_{test}(\theta) = \frac{1}{2m_{test}}\left[\sum_{i=1}^{m_{test}}\left(h(x_{test}^{(i)}) - y_{test}^{(i)}\right)^2\right]$$

Test error: MSE

- Classification problems:

$$J_{test}(\theta) = \frac{1}{m_{test}}\left[\sum_{i=1}^{m_{test}} err\left(h(x_{test}^{(i)}), y_{test}^{(i)}\right)\right]$$

Test error: Misclassification proportion

where

$$err\left(h(x_{test}^{(i)}), y_{test}^{(i)}\right) = \begin{cases} 1 \text{ if } [h(x_{test}^{(i)}) = 1 \text{ and } y_{test}^{(i)} = 0] \text{ or } [h(x_{test}^{(i)}) = 0 \text{ and } y_{test}^{(i)} = 1] \\ 0 \text{ otherwise} \end{cases}$$

Outline

- How to address overfitting
- Evaluating a learning algorithm
- Model selection
- Error metrics for unbalanced classes
- Dimensionality reduction
- Building a ML project

Model selection
The problem

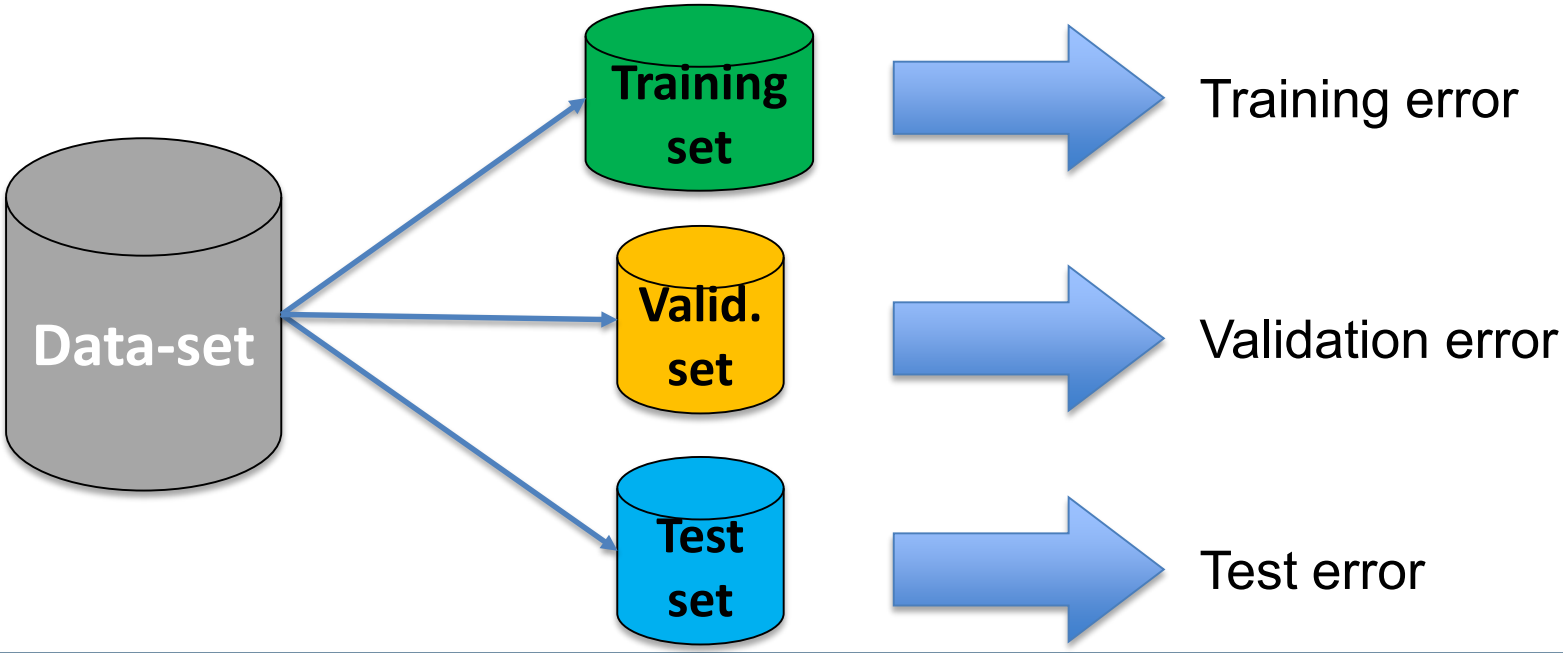
- Example: select the regularization (hyper)parameter λ
  - Try different values of λ and for each of them:
    - optimize parameters θ using examples in the training set (minimize cost function $J_{train}(\theta)$ for that specific value of λ)
    - compute corresponding test error $J_{test}(\theta)$ using examples in the test set
  - Select the value of λ which provides the lowest $J_{test}(\theta)$

- *Does this really generalize our problem?*
  - The value of λ we selected *depends* on the examples in the test set !!!

Model selection

The solution

- Split data into **3 separate sets**:
  - Training set: used for model ***fitting***, e.g., to choose parameters θ
  - Validation set: used for model ***selection***, e.g., assess which model best fits to our data (tune hyperparameters)
  - Test set: used to assess model ***performance***



Model selection

A better approach

GOOD

- Example: select the regularization (hyper)parameter λ
  - Try different values of λ and for each of them:
    - optimize parameters θ using examples in the training set (minimize cost function $J_{train}(\theta)$ for <u>that specific</u> value of λ)
    - compute corresponding validation error $J_{val}(\theta)$ using examples **<u>in the validation set</u>**
  - Select the value of λ which provides the **lowest $J_{val}(\theta)$**
  - Compute $J_{test}(\theta)$ using the test set
- The value of λ we selected *does NOT depend* on examples in the test set (i.e., those used for algorithm performace evaluation)
- $J_{test}(\theta)$ is a good estimation of the (generalized) performance of your algorithm
- N.B.: in principle, to select different hyperparameters (e.g., λ, polynomial degree, etc.) of the same model, the procedure should be repeated once for each hyperparameter separately

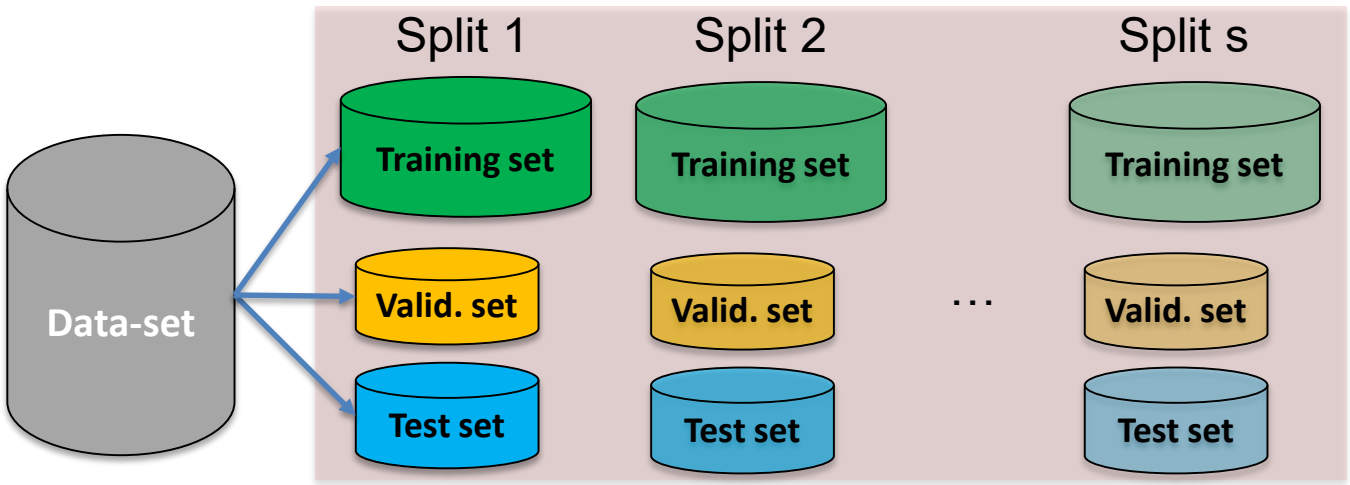
Outline

- How to address overfitting
- Evaluating a learning algorithm
- Model selection
  - Evaluating a learning algorithm, revisited
- Error metrics for unbalanced classes
- Dimensionality reduction
- Building a ML project

Evaluating a learning algorithm – revisited

Problem

- The model performance assessed through the validation-set approach still depends on the <u>particular split</u> we created between training/validation/test sets
  - A different split would give a different estimate for model performance (different J_{test})
- *Cross-Validation*: use several splits and then consider "average" values

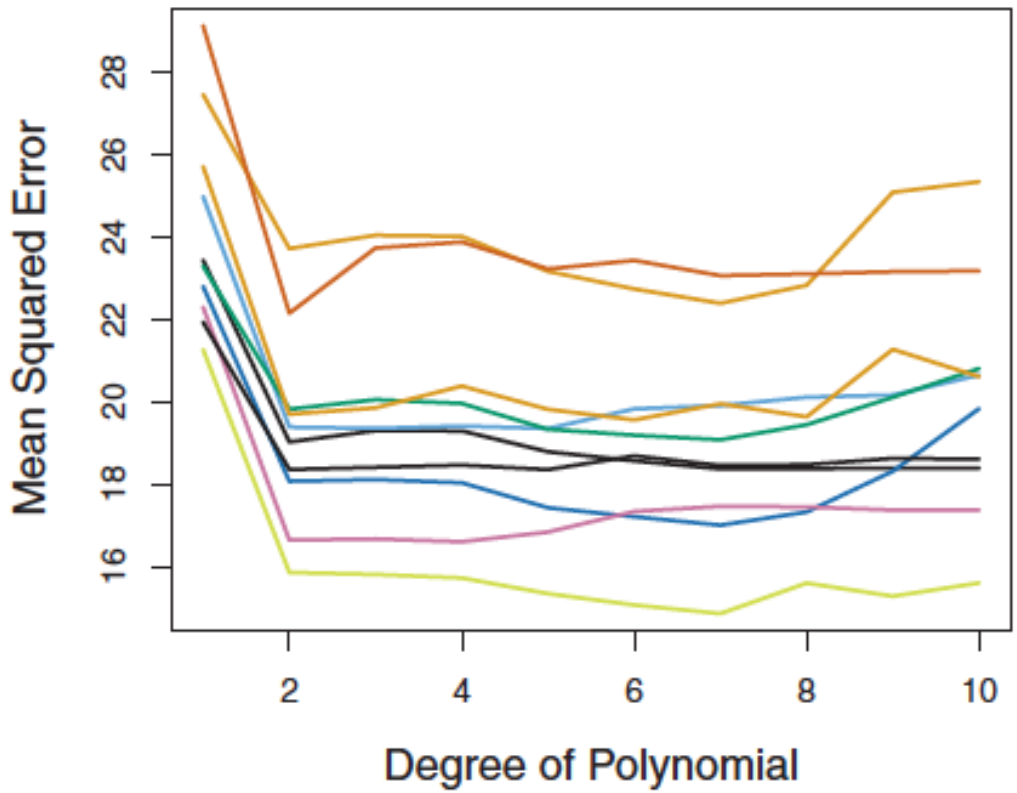


Model performance:
$J_{test} = 1/s \, \Sigma_i \, J_{test,i}$

Evaluating a learning algorithm - revisited

Example of using the "simple" validation-set approach

- Selecting the degree in polynomial regression
  - Model selection can be good with validation-set approach, but performance evaluation can fail

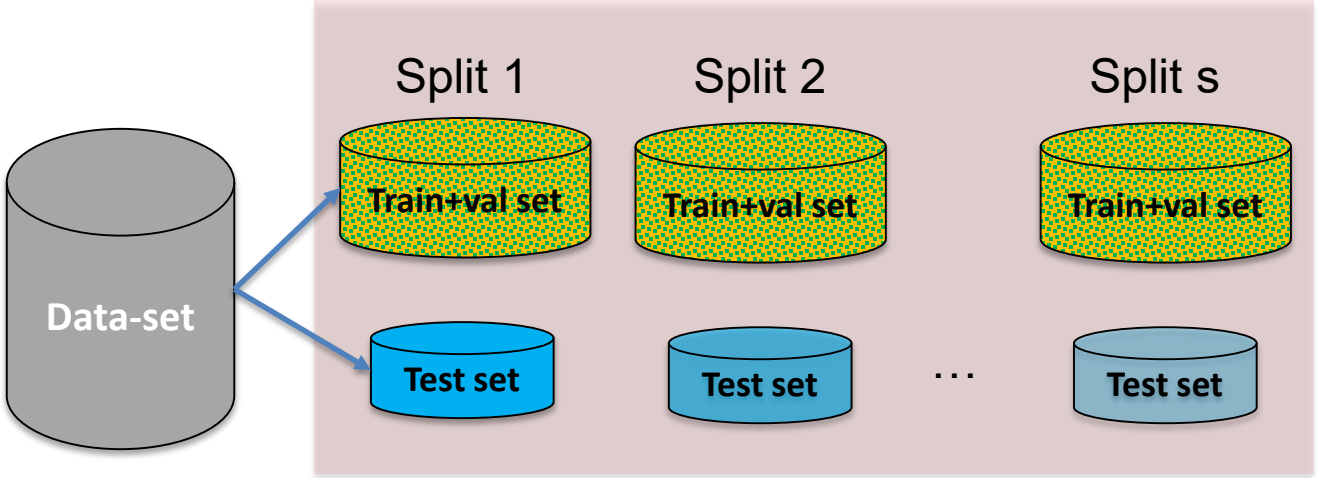


Source: ISLR

Evaluating a learning algorithm - revisited

Cross-validation

- *Cross-Validation*: use several splits and then consider "average" values



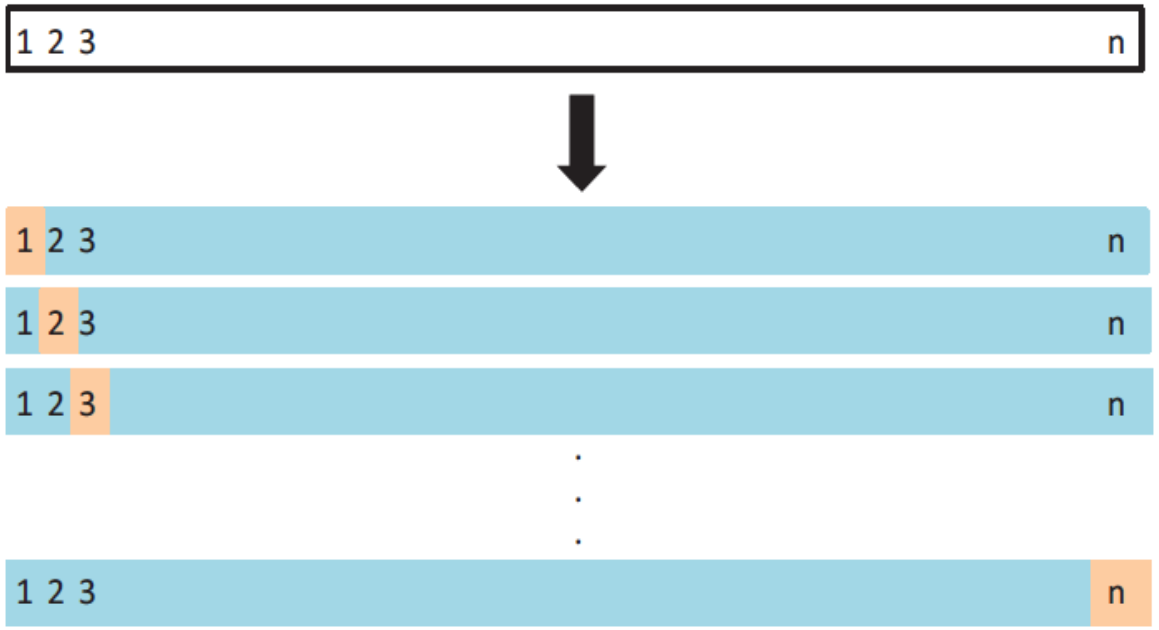
Model performance:
$J_{test} = 1/s \; \Sigma_i \; J_{test,i}$

- How do we split data into these 3 sets?
  - (easiest approach: perform "several" random independent splits and compute the average error)
  - More schematic approach: consider a <u>fixed (independent) test-set</u>, then split remaining set into train/valid by using either
    - Leave-one-out cross-validation (LOOCV), or
    - K-fold cross validation

Cross-validation

Leave-one-out cross-validation (LOOCV)

- Dataset with ***n* examples**: consider *n* different splits
  - At each split: leave one (example) out



For regression:

$$J_{test} = 1/n \, \Sigma_i \, J_{test,i}$$

For classification:

$$Err_{test} = 1/n \, \Sigma_i \, Err_{test,i}$$

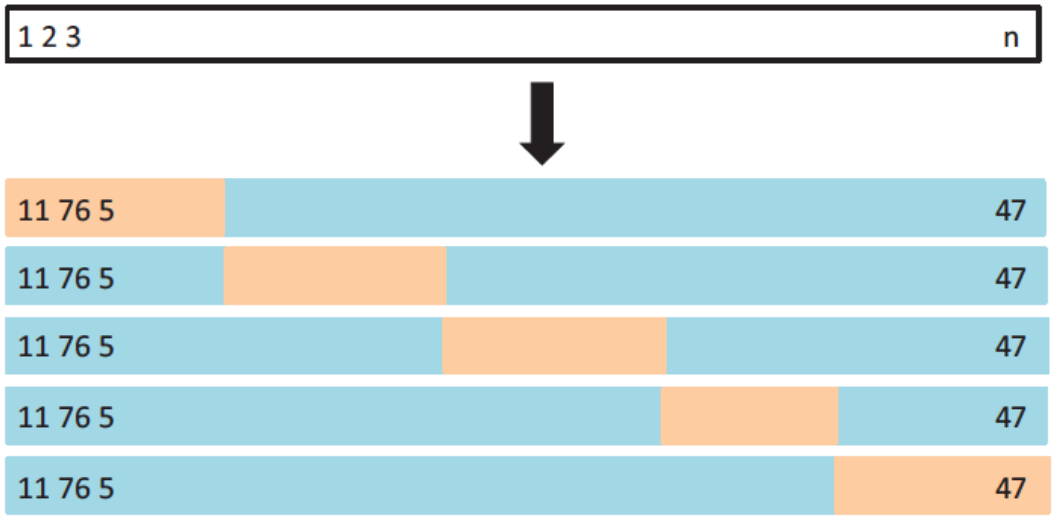
Source: ISLR

- Does not depend on the split (we consider *all* splits)
- Can be time consuming with large *n* (fit the model *n* times)

Cross-validation

k-fold cross-validation

- Dataset with ***n examples***: consider *k* different randomly defined groups (folds)
  - At each iteration: leave one fold out
    - we need to fit the model k times
    - LOOCV is a "special" k-fold cv with k=n



For regression:

$$J_{test} = 1/k \, \Sigma_i \, J_{test,i}$$

For classification

$$Err_{test} = 1/k \, \Sigma_i \, Err_{test,i}$$

Source: ISLR

- Lower variance wrt LOOCV
  - Fitted models are less correlated

Summarizing…
Cross-validation & performance evaluation

- In the same development, two cross-validations can be nested for
  - Performance assessment (outer cross-validation)
  - Model selection & hyperparameters tuning (inner cross-validation)

Outline

- How to address overfitting
- Evaluating a learning algorithm
- Model selection
- Error metrics for unbalanced classes
- Dimensionality reduction
- Building a ML project

Error metrics for unbalanced classes

- Accuracy: typical metric in classification problems

$$A = \frac{\text{tot. nr of correctly classified example}}{\text{tot. nr of examples}}$$

  – can be applied to binary or multi-class classifiers
- Problem: suppose we have a dataset with two skewed classes, e.g.:
  – 99% negative class (*y=0*)
  – 1% positive clss (*y=1*)
- A "trivial" classifier which *always* predicts *y=0* (i.e., negative examples) will have 99% accuracy!!!

Error metrics for unbalanced classes

Confusion Matrix

- Used to breakdown classification errors

		Predicted class	
		y=1	*y=0*
Actual class	*y=1*	True positive (TP)	False negative (FN)
	y=0	False positive (FP)	True negative (TN)

Accuracy

$$A = \frac{TP + TN}{TP + FP + TN + FN}$$

- Precision

$$P = \frac{\text{true positive}}{\text{nr of predicted positive}} = \frac{TP}{TP + FP}$$

- Recall (Sensitivity)

$$R = \frac{\text{true positive}}{\text{nr of actual positive}} = \frac{TP}{TP + FN}$$

The higher, the better!

Error metrics for unbalanced classes

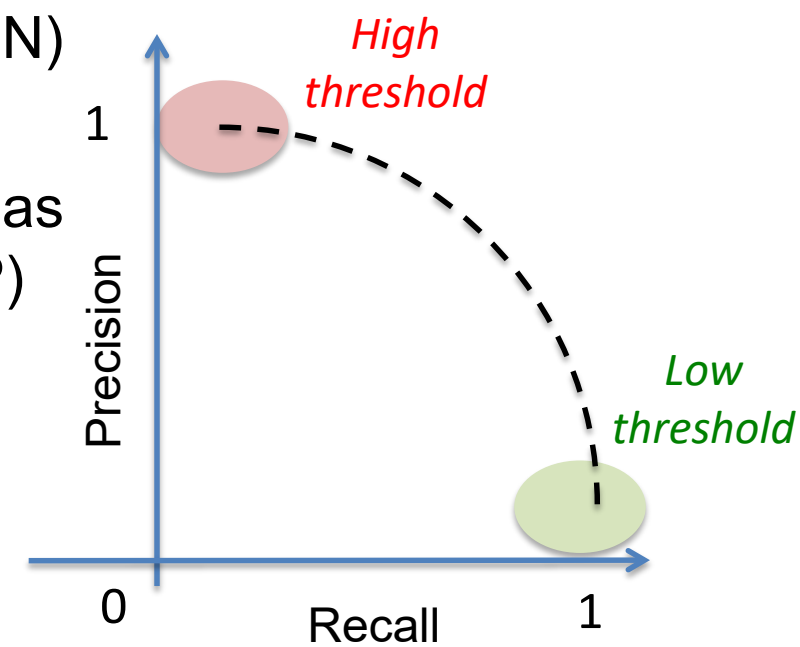
Precision or Recall?

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN}$$

- Assume we have trained a logistic regression classifier
  - *y=1 if $h_\theta(x) \geq 0.5$*
  - *y=0 if $h_\theta(x) < 0.5$*
- If we predict y=1 only if *$h_\theta(x) \geq 0.9$* (high confidence for positive class, more FN)
  - High precision, low recall
- If we predict y=1 if *$h_\theta(x) \geq 0.1$* (catch as many positives as possible, more FP)
  - High recall, low precision
- What to privilege?
- F-score

$$F = 2\frac{PR}{P + R}$$



Error metrics for unbalanced classes

Receiver Operating Characteristics (ROC) curves

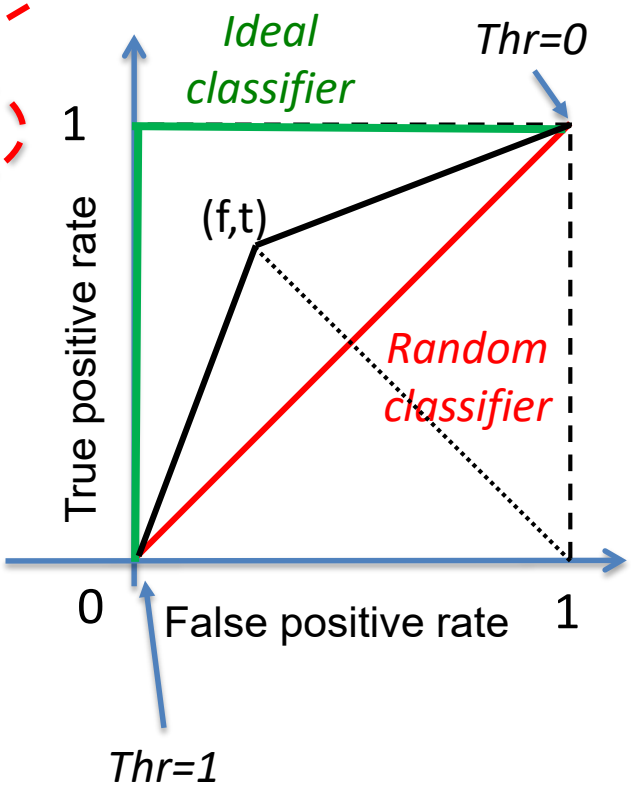
- Used to evaluate the performance of a classifier for various threshold values

Recall

$$specificity = \frac{TN}{TN + FP} \qquad sensitivity = \frac{TP}{TP + FN}$$

$$f = false\ positive\ rate = 1 - specificity = \frac{FP}{TN + FP}$$

$$t = true\ positive\ rate = sensitivity = \frac{TP}{TP + FN}$$



- Given a generic classifier we can obtain the corresponding (f,t)
  - (f,t) changes with the threshold
- Area under ROC curve (AUC)

for a given thr. (a given (f,t)) → $AUC = \frac{t}{2} + \frac{(1-f)}{2} = \frac{1}{2}\frac{TP}{TP+FN} + \frac{1}{2}\frac{TN}{TN+FP}$

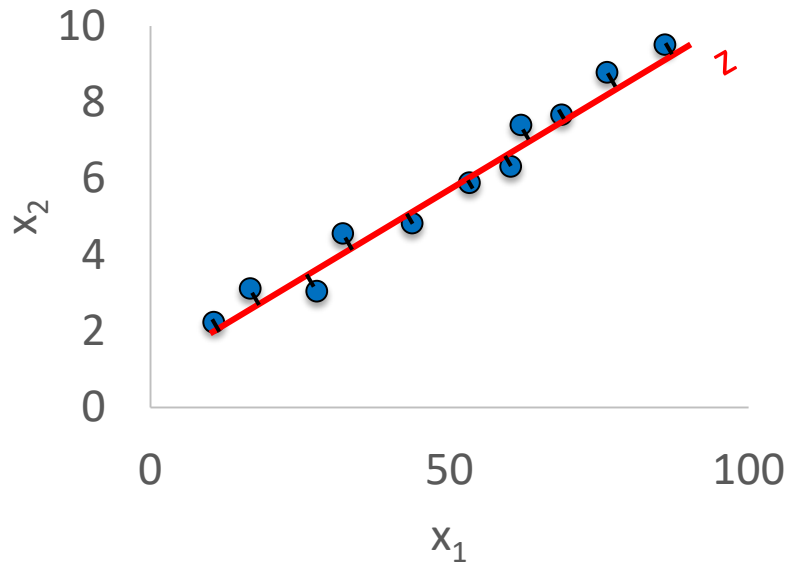
Outline

- How to address overfitting
- Evaluating a learning algorithm
- Model selection
- Error metrics for unbalanced classes
- Dimensionality reduction
- Building a ML project

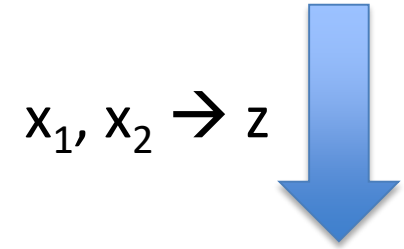
Dimensionality reduction

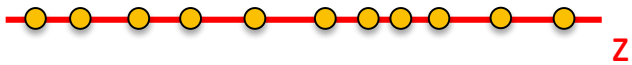
Motivation 1

- <u>Features compression</u>
  - Useful when we have redundant (correlated) features
    - cm vs inch
    - nr of users vs density of users
    - total traffic vs traffic per user
    - …
- Find a direction (vector or component) where to "project" the original features
- Reduces problem complexity
- Reduces storage requirements



$x_1, x_2 \rightarrow z$

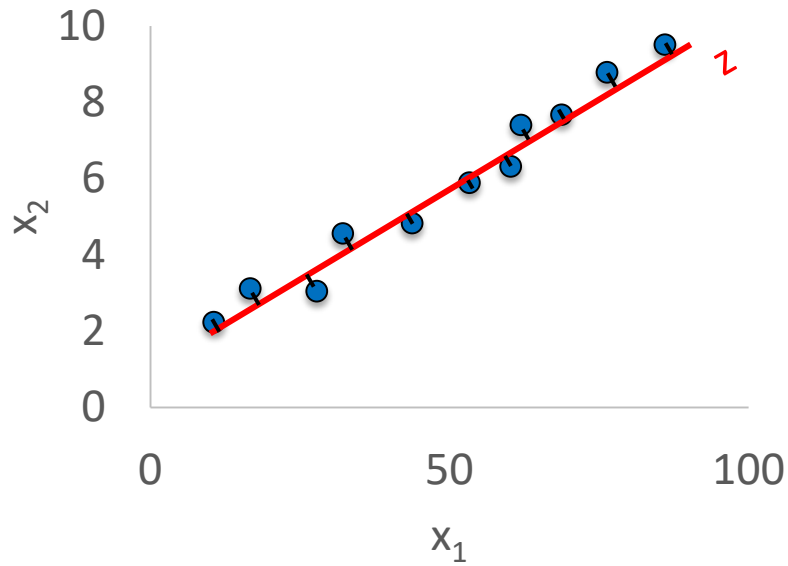




Dimensionality reduction

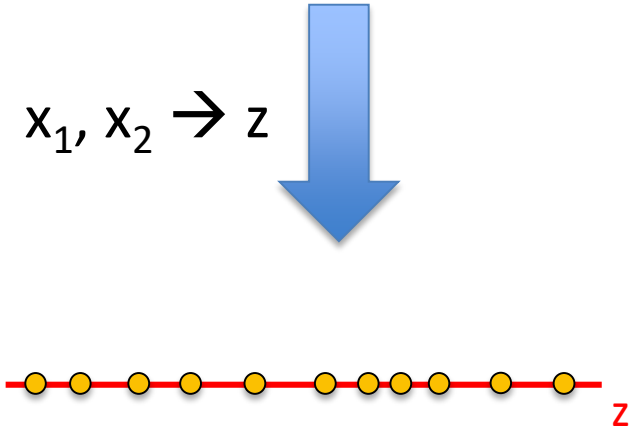
Motivation 2

- <u>Data visualization</u>
  - Useful to observe examples when we have many features (e.g., >3)



- We get a set of **<u>different</u>** features $z_1, \ldots, z_k$, (k<<n) which "synthesize" our dataset
- N.B. We are accepting to loose some information
- N.B.2 The transformation we do is NOT as in linear regression!

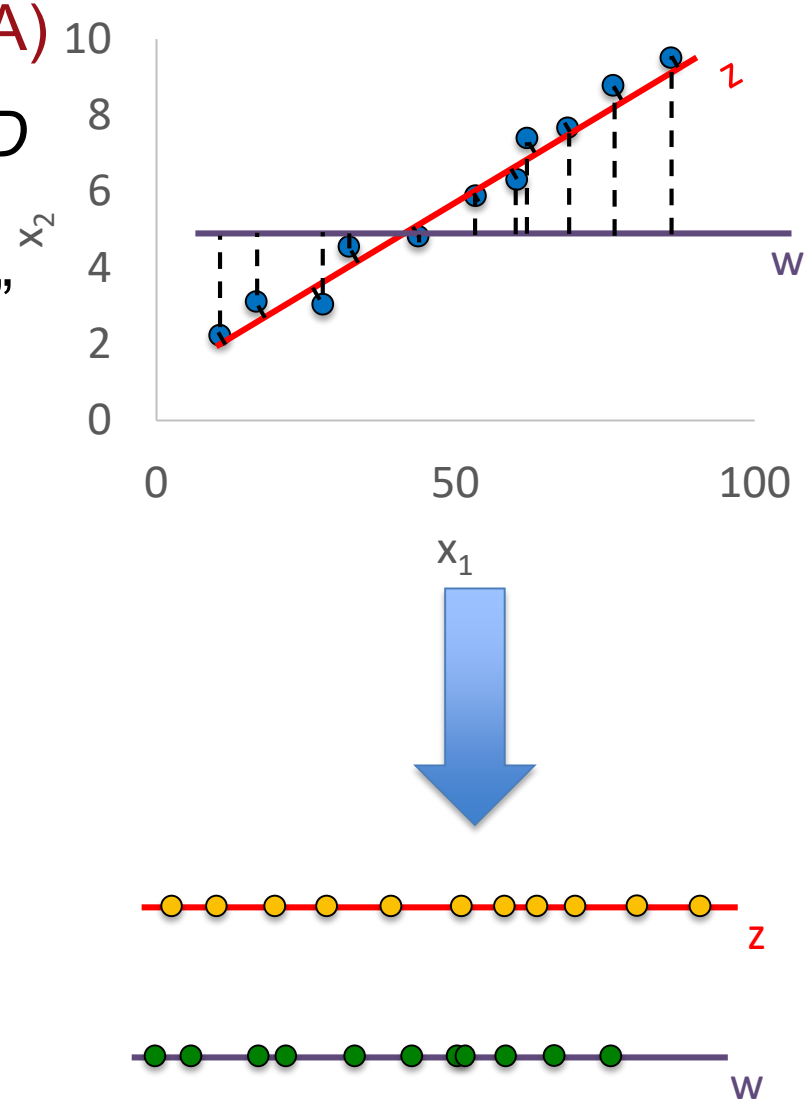
$x_1, x_2 \rightarrow z$



Dimensionality reduction

Principal Component Analysis (PCA)

- Reduce dimension from *2D* to *1D*
  - Find a line (vector or component) where to "project" the original features so as to *minimize the projection error*
    - Minimize the overall distance of every features to the obtained projection line
- Reduce dimension from *n* to *k*
  - Find *k* "directions" (vectors or components)
    - E.g.: k=2 → we project features onto a 2D plane



Dimensionality reduction

PCA algorithm

1. Data preprocessing (mean normalization + features scaling), i.e., perform replacement:

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j} \quad \mathrm{i} = 1,\ldots,\mathrm{m};\ \mathrm{j} = 1,\ldots,\mathrm{n}$$

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$$

$$s_j = \max_i x_j^{(i)} - \min_i x_j^{(i)}$$

2. Compute covariance matrix…

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T$$

nxn matrix

$$x^{(i)} = \begin{pmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \ldots \\ x_n^{(i)} \end{pmatrix}$$

3. …and the matrix *U* of its eigenvectors

$$U = \begin{pmatrix} u_1^{(1)} & u_1^{(2)} & \ldots & u_1^{(n)} \\ u_2^{(1)} & u_2^{(2)} & \ldots & u_2^{(n)} \\ \ldots & \ldots & \ldots & \ldots \\ u_n^{(1)} & u_n^{(2)} & \ldots & u_n^{(n)} \end{pmatrix} = \begin{pmatrix} u^{(1)} & u^{(2)} & \ldots & u^{(n)} \end{pmatrix}$$

nxn matrix

Dimensionality reduction

PCA algorithm (ctd.)

4. Take the first *k* eigenvectors of U (the "*principal components*")

$$U_{k-components} = \begin{pmatrix} u_1^{(1)} & u_1^{(2)} & \dots & u_1^{(k)} \\ u_2^{(1)} & u_2^{(2)} & \dots & u_2^{(k)} \\ \dots & \dots & \dots & \dots \\ u_n^{(1)} & u_n^{(2)} & \dots & u_n^{(k)} \end{pmatrix} = \begin{pmatrix} u^{(1)} & u^{(2)} & \dots & u^{(k)} \end{pmatrix}$$

nxk matrix

5. Use them to compute "new" features *z*

$$\underset{kx1}{z^{(i)}} = \underset{kxn}{\begin{pmatrix} u^{(1)} & u^{(2)} & \dots & u^{(k)} \end{pmatrix}^T} \underset{nx1}{x^{(i)}} \quad i = 1,2,\dots,m$$

6. To recompute back the original features:

$$x_{approx}^{(i)} = U_{k-components} z^{(i)} \quad i = 1,2,\dots,m$$

Dimensionality reduction

How to select the number of components *k*?

- Objective: reduce dimensions and retain as much information as possible
  - Average squared projection error $E = \frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)} - x_{approx}^{(i)}\right\|^2$
  - Variability in the data set $V = \frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}\right\|^2$
- Typical choice: select the *lowest k* s.t.

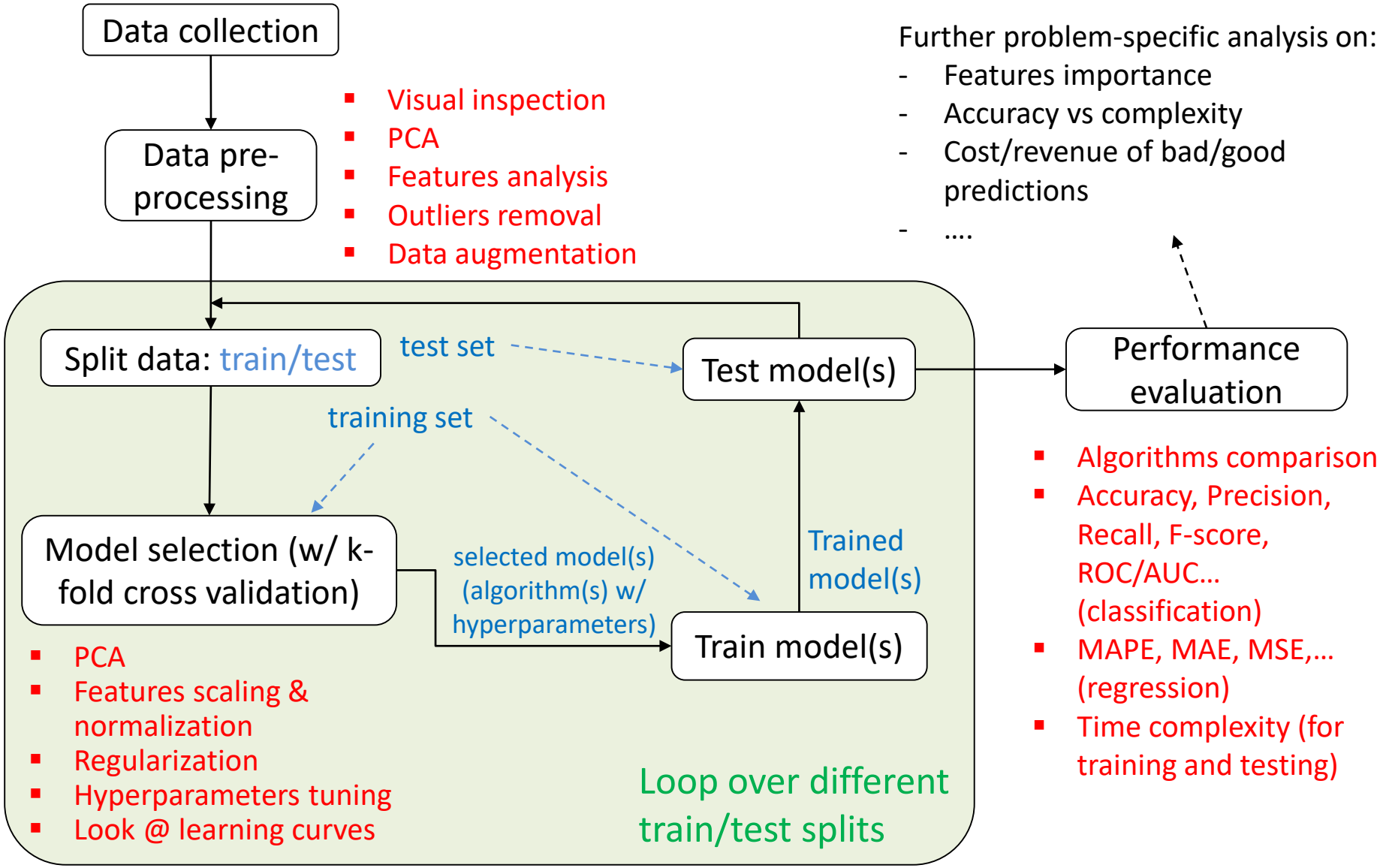
$$\frac{E}{V} \leq \varphi$$

  - E.g., 0.01; 0.05; 0.1 …
  - "retained variance" = (1-*E*/*V*); e.g., 99%, 95%, 90%...

Outline

- How to address overfitting
- Evaluating a learning algorithm
- Model selection
- Error metrics for unbalanced classes
- Dimensionality reduction
- Building a ML project

Building a ML project (supervised problems)



Building a ML project (unsupervised problems)

Data collection

Data pre-processing

- Visual inspection
- PCA
- Features analysis
- Outliers removal
- Data augmentation

Further problem-specific analysis:

- Impact of clustering on supervised problems
- Features importance
- Cost/revenue of bad/good clustering solutions
- ....

Features engineering

- PCA
- Features removal
- Autoencoders
- Features scaling & normalization

Inference

Clustering

- Nr of clusters
- Hyperparameters tuning
- Look @ clustering metrics
  - Inter/intra cluster distance
  - Silhouette
- Elbow method

- Observe clusters "manually"
- Compare different clustering
  - ARI, Rand index

Loop over different subsets of data and combinations of algorithms/parameters

Emerging concepts

- **Active learning**
  - Data for scarce portions of the features space can be actively collected
    - E.g., inject *probe* traffic to measure its QoT; purposedly alter equipment behaviour to understand its aging characteristics…
    - Can be very expensive → trade off knowledge vs cost
- **Concept drift (online learning)**
  - Training is continuously performed while retrieving new data from the field
  - The input/output relationship changes over time
    - Can happen w/ or w/o changes in the input distribution
    - E.g., traffic patterns on a given area can change over time; network devices performance may change due to aging
- **Transfer learning**
  - Perform training on a given source domain and apply the knowledge on a target domain
    - Partly retrain with few data on the target domain
    - E.g., is the training performed over a network/link/failure still valid on a different scenario?

Emerging concepts (2)

- **Collaborative/hierarchical learning**
  - Different domains use their own learning to take local decisions
  - Domain abstractions can be shared with a hierarchically-higher learning engine to take global decisions
    - E.g., modulation format of some lightpaths is adapted based on QoT measurement for that lightpath; change of modulation format can be shared with network controller to eventually re-route the lightpath and save spectrum
- **Federated learning**
  - Model development with huge datasets can be computationally intense and suffer from security issues
  - Alternative: develop different models using small datasets, then "merge" the knowledge models
    - E.g., different ANNs are trained in different devices to detect failures; knowledge is shared among the different parties (e.g., averaging ANN weights) to obtain a (hopefully) more general model
- **Explainable Artificial Intelligence (XAI)**
  - How to justify wrong decision? What features should we remove/keep? Can we improve models a-posteriori?