Aprendizagem Aplicada à Segurança
(Mestrado em Cibersegurança-DETI-UA)

LECTURE 3
MODEL SELECTION AND VALIDATION – BIAS VS. VARIANCE

Petia Georgieva
(petia@ua.pt)

DETI/IEETA – UA

Outline

Model performance evaluation: perf. metrics

- **Model selection: Bias vs. variance**
- **Learning curves**
- **K –fold Cross Validation**

Performance Evaluation – Confusion Matrix

	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

a: TP (true positive)

b: FN (false negative)

c: FP (false positive)

d: TN (true negative)

Python: from sklearn.metrics import confusion_matrix

Performance metric - Accuracy

	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	(TP)	(FN)
	Class=No	(FP)	(TN)

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy - fraction of examples correctly classified.

1-Accuracy: Error rate (misclassification rate)

Limitation of Accuracy

- Consider binary classification (**Unbalanced data set**)
  - Class 0 has 9990 examples
  - Class 1 has 10 examples

- If model classify all examples as class 0, accuracy is 9990/10000 = 99.9 %

- Accuracy is misleading metrics because model does not classify correctly any example of class 1

  =>Use other performance metrics.

  => Find a way to balance the data set

(re-sampling methods: oversampling, under-sampling)

Other Performance Metrics

True Positive Rate (TPR), Sensitivity, Recall
of all positive examples the fraction of correctly classified

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

True Negative Rate (TNR), Specificity
of all negative examples the fraction of correctly classified

$$TNR = \frac{TN}{TN + FP}$$

False Positive Rate (FPR) - how often an actual negative instance will be classified as positive, i.e. "false alarm"

$$FPR = 1 - TNR = \frac{FP}{FP + TN}$$

Precision - the fraction of correctly classified positive samples from all classified as positive

$$\text{Precision} = \frac{TP}{TP + FP}$$

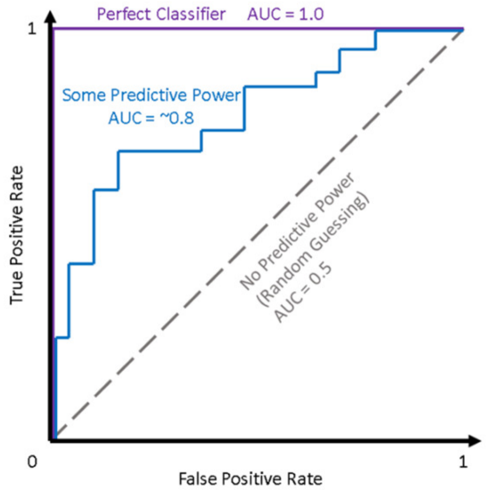
Combined performance metrics

F1 Score - weighted average of Precision and Recall

$$F1=2*(Recall * Precision) / (Recall + Precision)$$

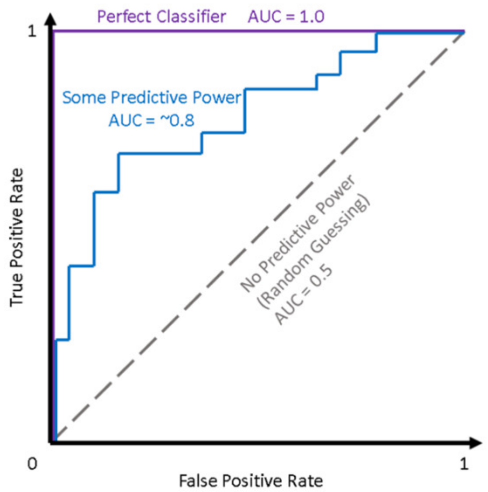
Balanced Accuracy= (Recall+Specificity)/2

Receiver Operating Characteristic (ROC) curve



ROC curve: **True Positive Rate (TPR)** against **False Positive Rate (FPR)** for a binary classifier changing the **thresholds** between positive and negative.
For example, in logistic regression, if an observation is predicted to be > 0.5, it is labelled as positive.
But, we can choose any threshold between 0 and 1 (0.1, 0.3, 0.6, 0.99, etc.).
ROC curves visualize how these choices affect classifier performance.
For multi K-class problem, draw K ROC curves.
Python: from sklearn.metrics import roc_curve

Area Under the (ROC) Curve - AUC

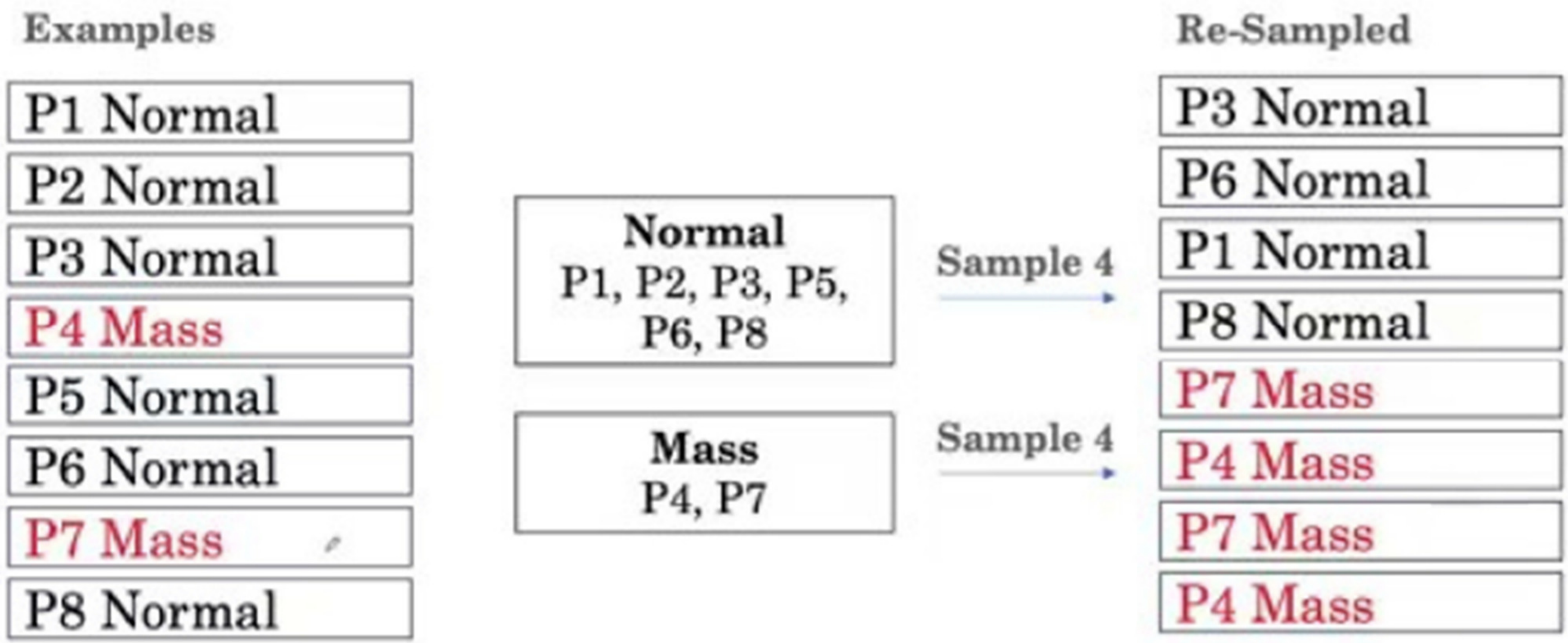


ROC curve is useful for visualization, but it's good to have also a single metric => AUC.
The higher the AUC score, the better a classifier performs for the given task.
For a classifier with no predictive power (i.e., random guessing) => AUC = 0.5.
For a perfect classifier => AUC = 1.0.
Most classifiers fall between 0.5 and 1.0.

Python: from sklearn.metrics import auc

Class Imbalance problem

Solution: Re-sampling methods (under-sampling, oversampling)



universidade de aveiro

Definitions for Epoch /Batch Size / Iterations / Train step

One Epoch is when an ENTIRE dataset is passed through the model (e.g. forward and backward in a neural network) only ONCE.
If data is too big to feed to the computer at once one epoch is divided in several smaller batches.

Batch Size: Total number of training examples present in a single batch.

Iterations is the number of batches needed to complete one epoch.

Example: Let's say we have 2000 training examples.
We can divide the dataset of 2000 examples into batches of 500 then it will take 4 iterations to complete 1 epoch.

Training run/step - is one update of the model parameters.
We update the parameters after one batch or after one epoch.

Deciding what to do next ?

Suppose you have trained a ML model on some data. When you test the trained model on a new set of data, it makes unacceptably large errors. What should you do ?

-- Get more training examples ?
-- Try smaller sets of features (feature selection) ?
-- Try getting additional features (feature engineering) ?
-- Try using different/nonlinear kernels ?
-- Try other values of the hyper parameters (e.g. regul. parameter) ?

Run tests to gain insight what isn't working with the learning algorithm and how to improve its performance.
Diagnostics is time consuming , but can be a very good use of your time.

Simplest division: Train & Test subsets

- Training set (70%-80 %) : used to train the model
- Test set (30%-20%) : used to test the trained model

- **Optimize the model parameters with training data**
  (minimize some cost/loss function J)

<u>After the training stage is over (i.e. the cost function J converged)</u>

- Compute the MSE on test data (for regression problems)

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

or

- **Compute the model accuracy or some other metric from the confusion matrix, on test data (for classification problems)**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

3 way split: Train/Dev/Test Sets

Choose ML model: Logit, SVM, K-NN, etc. ?
Choose model hyper-parameters:
- What is the best learning rate ?
- What is the best regularization parameter (*λ)* ?
- What is the best polinomial degree ?
- ……

Devide dataset in 3 sub-sets:
- Training set
- Cross Validation (CV) set = Development set = 'dev' set
- Test set

Traditional division for Small data set (up to 10000 examples) :
60% - 20% - 20%

Big data (1 million. examples): 98% - 1% - 1%

Model /hyper parameter selection

Step 1: Optimize parameters θ (to minimize some cost function J) using the same training set for all models. Compute some perf. metrics with the training data (i.e. error, accuracy) :

Training error => $$E_{train}(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2\right]$$

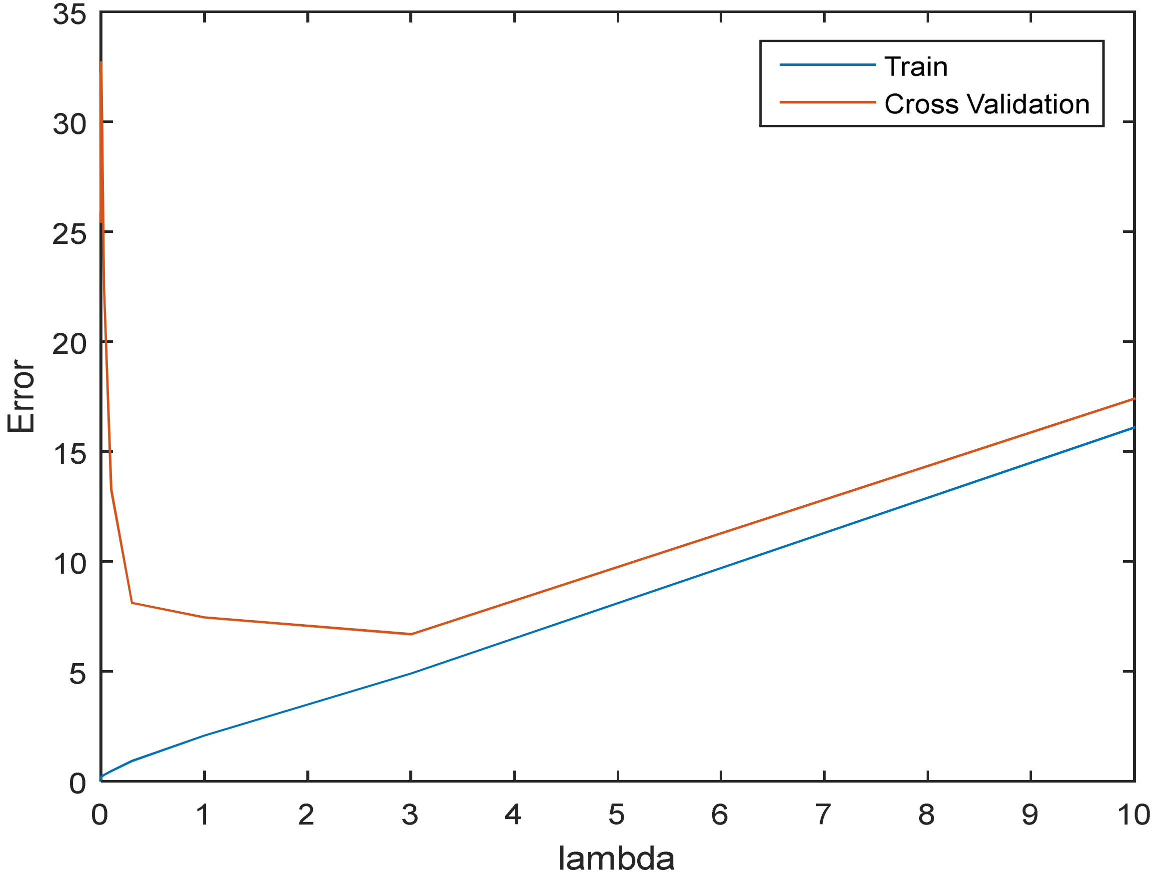
Step 2: Test the optimized models from step 1 with the CV set and choose the model with the min CV error (or other performance metric with dev data):

Cross validation (CV)/dev error => $$E_{cv}(\theta) = \frac{1}{2m_{cv}}\left[\sum_{i=1}^{m_{cv}}\left(h_\theta\left(x_{cv}^{(i)}\right) - y_{cv}^{(i)}\right)^2\right]$$

Step 3: Retrain the best model from step 2 with both train and CV sets starting from the parameters got at step 2. Test the retrained model with test set and compute test data perf. metric (***the real model performance !!!***):

Test error => $$E_{test}(\theta) = \frac{1}{2m_{test}}\left[\sum_{i=1}^{m_{test}}\left(h_\theta\left(x_{test}^{(i)}\right) - y_{test}^{(i)}\right)^2\right]$$

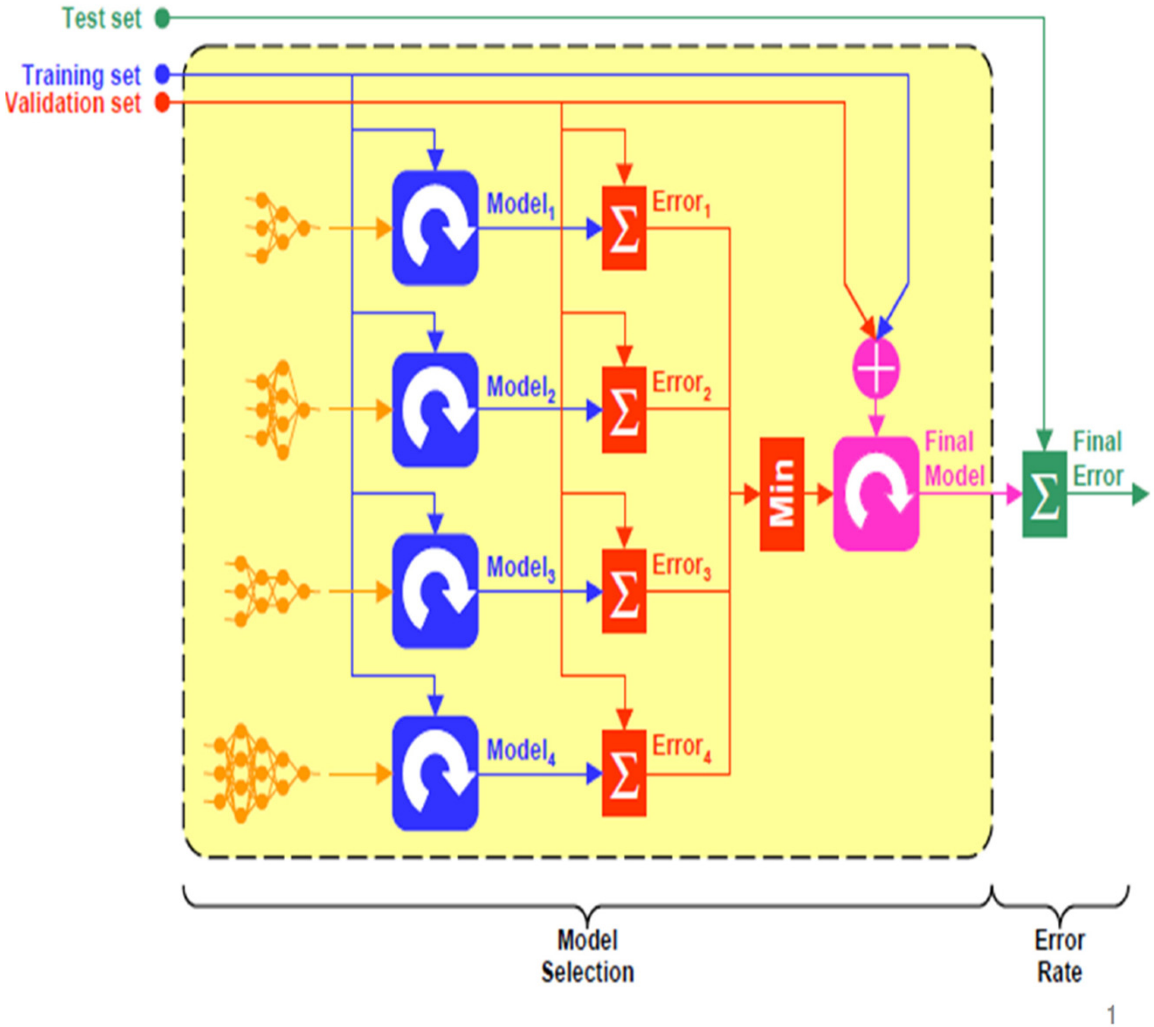
Example: Select best hyper-param. λ



Best λ=3

universidade de aveiro

Training/Valid (Dev)/Test subsets



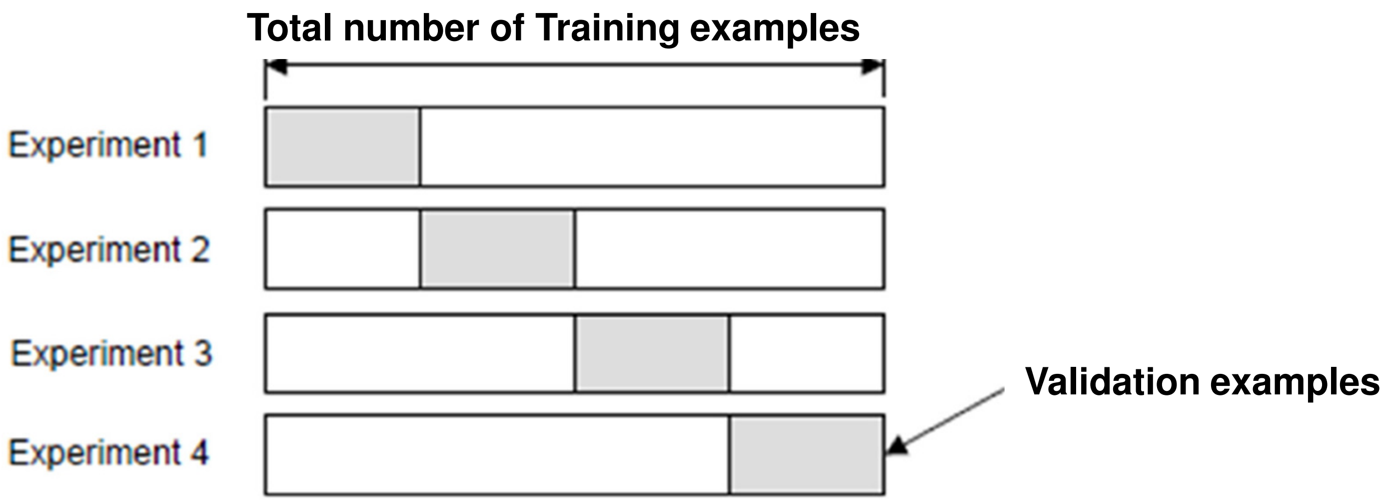
The most credible is the performance metric with test data, not used for training or validation of the model.

universidade de aveiro

K –fold Cross Validation

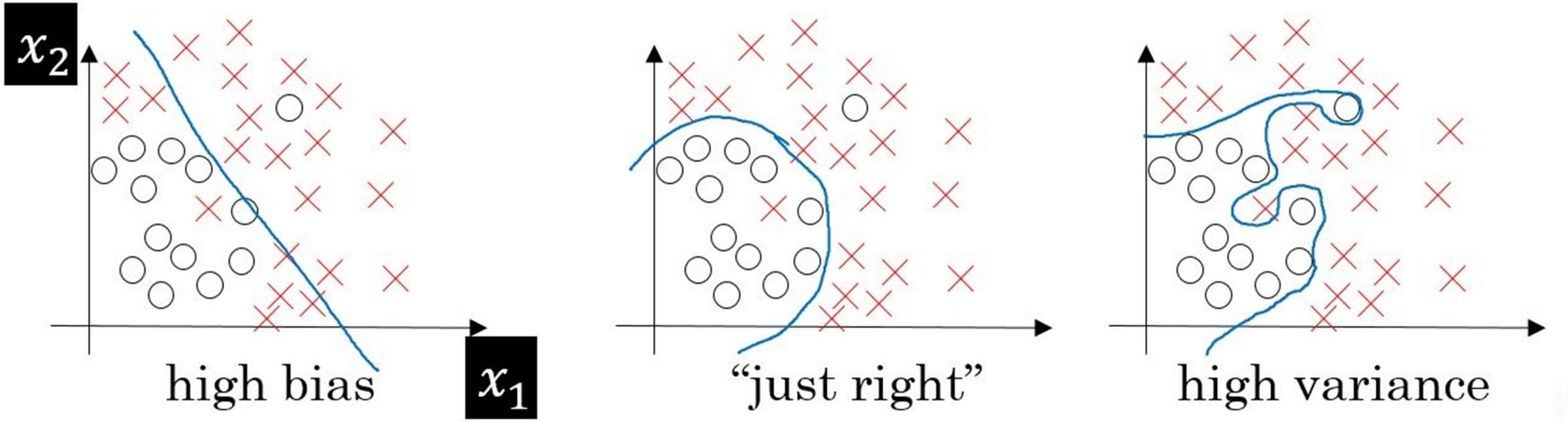
- Divide data into Training and Test subsets.
- Divide Training data into K subsets (K-fold).
- Use K-1 subsets for training and the remaining subset for CV.
- The final validation error is the average CV error of K experiments.
- Choose the best model /hyper-parameter the one that minimise the average CV error.

$$E_{cv} = \frac{1}{K}\sum_{i=1}^{K} E_{validation\_i}$$



Bias vs. Variance

An important concept in ML is the bias-variance tradeoff.
Models with **high bias** are not complex enough and **underfit** the training data.
Models with **high variance** are too complex and **overfit** the training data**.**



underfiting data		**overfiting data**
(very simple model)	(good model)	(very complex model)

Diagnosing Bias vs. Variance

How to diagnose if we have a high bias problem or high variance problem ?

High Bias (underfiting) problem:

Training error (*Etrain)* and Validation/dev error (*Ecv)* are both high

High Variance (overfiting) problem:

Training error (*Etrain)* is low
and Validation/dev error (*Ecv)* is much higher than *Etrain*

Hints to improve ML model

Suppose you have learned a data model (hypothesis). However, when you test your hypothesis on a new set of data, you find that it makes unacceptably large errors in its prediction (regression or classification). What should you try next?

-- Get more training examples – fixes high variance

-- Try smaller sets of features – fixes high variance

-- Try getting additional features – fixes high bias

-- Try adding polynomial features - fixes high bias

-- Try decreasing λ – fixes high bias

- Try increasing λ – fixes high variance