4 – Experimental Setup

Computational Intelligence for the Internet of Things (2019-20)

João Paulo Carvalho
joao.carvalho@inesc-id.pt

INESC-ID
Instituto Superior Técnico, Universidade de Lisboa

Experimental Setup

- Data Setup
  - Data Preprocessing and Feature Selection
  - Training, Validation and Test Sets
    - Cross Validation
- Evaluation and Validation
  - Error Measures
  - Confusion Matrix
  - Accuracy, Precision, Recall, F-Score, Specificity
  - AUC and ROC
  - Dataset balance
  - Baseline

Data Setup

Data Preprocessing

- Assumption – Data is Preprocessed:
  - Data cleaning
  - Data integration
  - Data reduction
  - Data transformation and data discretization
- Especially important:
  - Data Cleaning
    - Missing values; Noise removal; Outlier detection
  - Feature Selection
  - Data Normalization

Training, Validation and Test Sets

- Three data sets are commonly used in different stages of the creation of a model:
  - Training Dataset:
    - Used to fit the parameters of the model (e.g.: the weights in a NN)
  - Validation Dataset (or Development Dataset):
    - Used to test the model with the parameters selected during the training phase and to adjust hyper-parameters, i.e., the architecture, of the model (e.g.: how many hidden layers in a NN)
  - Test Dataset:
    - Used to provide an unbiased evaluation of a final model fit on the training dataset
    - If the data in the test dataset has never been used in training (for example in cross-validation), the test dataset is also called a holdout dataset

Some considerations

- The Test set should be independent of the Training set, but should follow the same probabilistic distribution
- Overfitting: most of the techniques tend to exploit relations in the training data that do not hold in general (i.e. they do not generalize to the rest of the data)
  - If a model performs (fits) better the training set than the test set, then overfitting has occurred
  - In order to avoid overfitting, when any classification parameter needs to be adjusted, it is necessary to have a validation dataset in addition to the training and test datasets.

Some considerations (II)

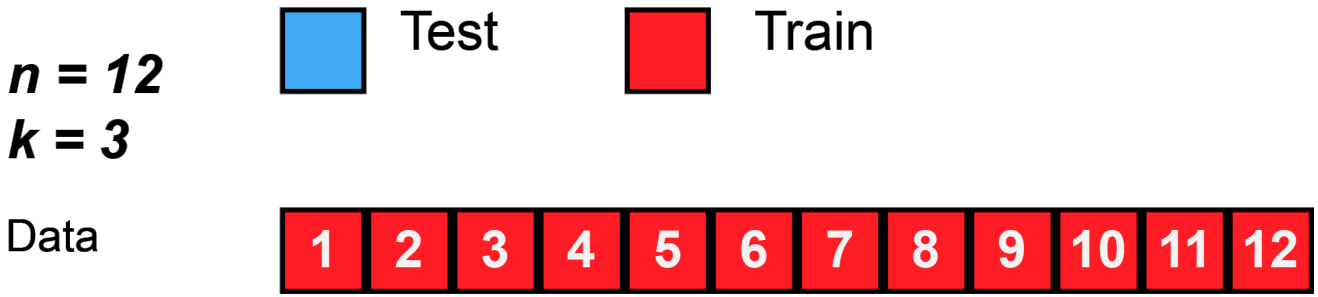
- When multiple algorithms are being compared, the training dataset is used to train candidate algorithms, the validation dataset is used to compare their performances and decide which one to take and, finally, the test dataset is used to calculate the performance of the selected model

Training, Validation and Test Set Ratios

- Most common ratios for splitting datasets:
  - 70% Train; 30% Test
  - 70% Train; 15% Validation; 15% Test
  - 80% Train; 10% Validation; 10% Test
  - 60% Train; 20% Validation; 20% Test

K-fold Cross-Validation

- Partition the n data into K equal-sized subsets: $P_1, P_2, P_3, \ldots, P_K$
- For i=1 to K:
  - Train with $\sum_{j=1}^{K} P_j$, $j \neq i$, and test with P_i
  - Average testing results



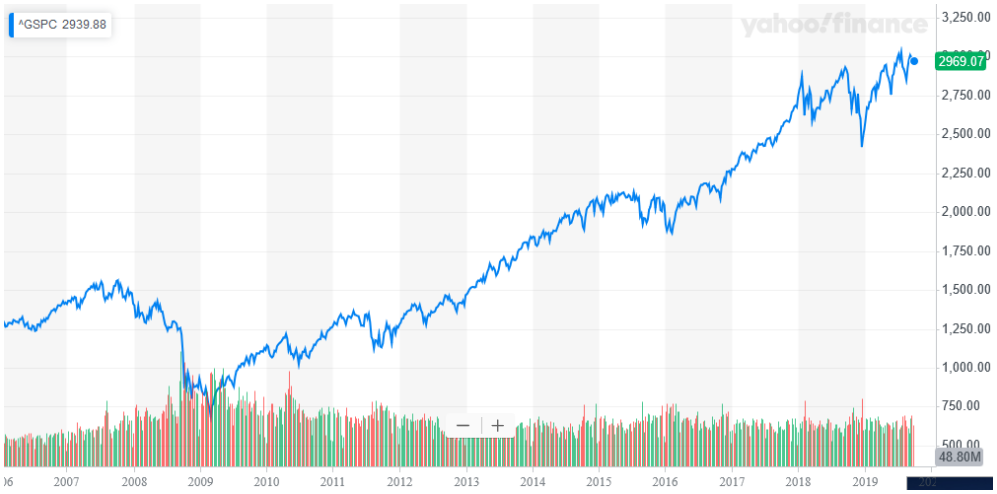
By MBanuelos22 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=87684542

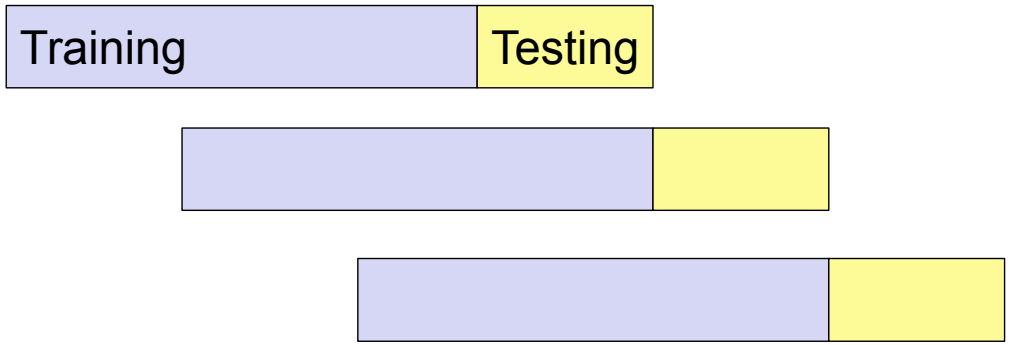
K-fold Cross-Validation with Validation and Test set

- The total data set is split in K sets
- One by one, a set is selected as test set
- Then, one by one, one of the remaining sets is used as a validation set and the other K-2 sets are used as training sets until all possible combinations have been evaluated
- Finally, for the selected parameter set, the test set is used to evaluate the model with the best parameter set
  - Two variants: (1) evaluate the model that was trained on the training set; (2) evaluate a new model that was fit on the combination of the train and the validation set.

Cross-Validation in Time Series

- Cross-Validation doesn't make sense in Time-Series data
- If data has a different behavior in different periods, use a sliding window:





Train, and Test Sets Made Easy...

- Scikit learn Library (https://scikit-learn.org/):
  - sklearn.model_selection.train_test_split
- Automatically splits a dataset into train and test
  - Split is given as an argument
  - As it is the option of shuffling data before splitting it
- Use it twice to do a Train/Validation/Test split

Train, and Test Sets Made Easy... (II)

- sklearn.model_selection.train_test_split

```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> X, y = np.arange(10).reshape((5, 2)), range(5)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
>>> list(y)
[0, 1, 2, 3, 4]
```

Train, and Test Sets Made Easy... (III)

```
>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.33, random_state=42)
...
>>> X_train
array([[4, 5],
       [0, 1],
       [6, 7]])
>>> y_train
[2, 0, 3]
>>> X_test
array([[2, 3],
       [8, 9]])
>>> y_test
[1, 4]
```

```
>>> train_test_split(y, shuffle=False)
[[0, 1, 2], [3, 4]]
```

Cross-Validation Made Easy...

- sklearn.model_selection.cross_val_score
- Performs cross-validation on a given dataset and a given algorithm

```
>>> from sklearn import datasets, linear_model
>>> from sklearn.model_selection import cross_val_score
>>> diabetes = datasets.load_diabetes()
>>> X = diabetes.data[:150]
>>> y = diabetes.target[:150]
>>> lasso = linear_model.Lasso()
>>> print(cross_val_score(lasso, X, y, cv=3))
[0.33150734 0.08022311 0.03531764]
```

- Note: This function already includes the evaluation measures explained in the following slides

Evaluation and Validation

Error Measurement

- Applies to algorithms where the result is a numerical value
- Compute an error measurement for the training period and for the test period
  - The error in the both periods should be similar
- Error measures:
  - RMSE or RMSD (Root Mean Square Error/Deviation)
  - MAE (Mean Average Error)

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (e_i)^2}{n}} \qquad MAE = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

Logarithmic Loss

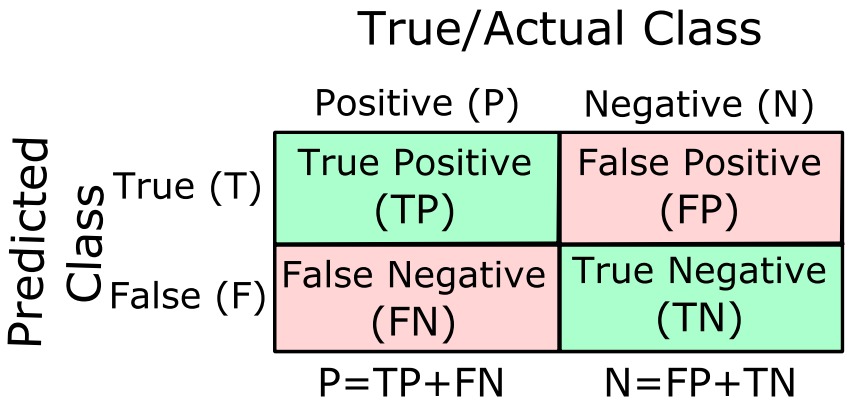
- Log loss: quantifies the accuracy of a classifier by penalising false classifications
  - measures the performance of a classification model where the prediction input is a probability value between 0 and 1

```
sklearn.metrics.log_loss(y_true, y_pred, eps=1e-15, normalize=True, sample_weight=None, labels=None)
```

Confusion Matrix

- Confusion Matrix: allows visualization of the performance of a supervised algorithm
- True Positives (TP), False Positives (FP), False Negatives (FN) and True Negatives (TN), are used to compute several metrics

Prediction \ Reference	Yes	No
Yes	True positive	False positive
No	False negative	True Negative



Classification Metrics

- Accuracy
- Precision / Positive Predictive Value (PPV)
- Recall / Sensitivity / True Positive Rate (TPR)
- F1 Score – Harmonic mean of PPV and TPR
- Specificity / True Negative Rate (TNR)

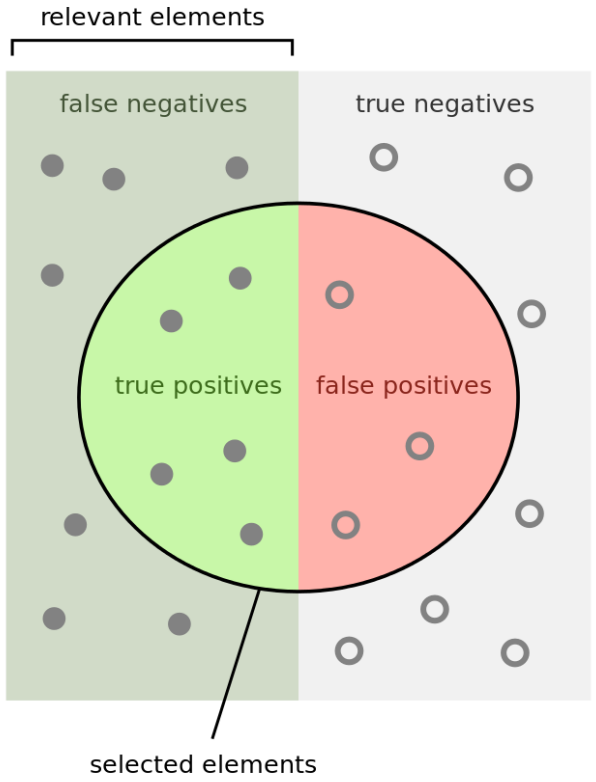
$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

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

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

$$\mathrm{F}_1 = 2 \cdot \frac{\mathrm{PPV} \cdot \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$



How many selected items are relevant?

Precision =

How many relevant items are selected?

Recall =

Confusion Matrix Example (I)

- Sick

		Real Class	
		Sick	Not Sick
Predicted Class	Predicted Sick	100 Sick (True Positive)	25 Not Sick (False Positive)
	Predicted Not Sick	20 Sick (False Negative)	1000 Not Sick (True Negative)

$$Precision = \frac{TP}{TP+FP} = \frac{1000}{100+25} = 0.8$$

$$Recall = \frac{TP}{TP+FN} = \frac{100}{100+20} = 0.83$$

$$Specificity = \frac{TN}{TN+FP} = \frac{1000}{1000+25} = 0.975$$

Note the imbalance of the dataset: many more “not sick” than “sick”. This leads to very high Specificity and Acc even if I say no one is sick

Confusion Matrix Example (II)

$$Precision = \frac{TP}{TP + FP} \quad Recall = \frac{TP}{TP + FN}$$

a) Precision = 1 Recall = 0.66

		Real Class: Sick	Real Class: Not Sick
Predicted Class	Predicted Sick	100 Sick (True Positive)	0 Not Sick (False Positive)
	Predicted Not Sick	50 Sick (False Negative)	10000 Not Sick (True Negative)

b) Precision = 0.66 Recall = 1

		Real Class: Sick	Real Class: Not Sick
Predicted Class	Predicted Sick	100 Sick (True Positive)	50 Not Sick (False Positive)
	Predicted Not Sick	0 Sick (False Negative)	10000 Not Sick (True Negative)

- a) I might be missing critical cases, but the probability of being sick is very high
- b) I am giving bad news to some, but I'm sure that I'm not missing anyone sick

Confusion Matrix Example (III)

- Stock Market: Which matrix do you prefer?

If I predict Up and it goes Down, I lose money

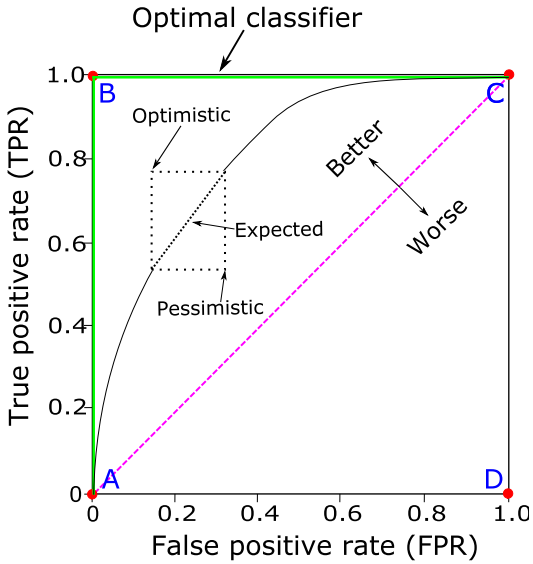
If I predict Down and it goes Up I lost the chance of making money

		Real Class	
		Up	Down
Predicted Class	Predicted Up	533 Up (True Positive)	450 Down (False Positive)
	Predicted Down	467 Up (False Negative)	480 Down (True Negative)

		Real Class	
		Up	Down
Predicted Class	Predicted Up	180 Up (True Positive)	160 Down (False Positive)
	Predicted Down	820 Up (False Negative)	770 Down (True Negative)

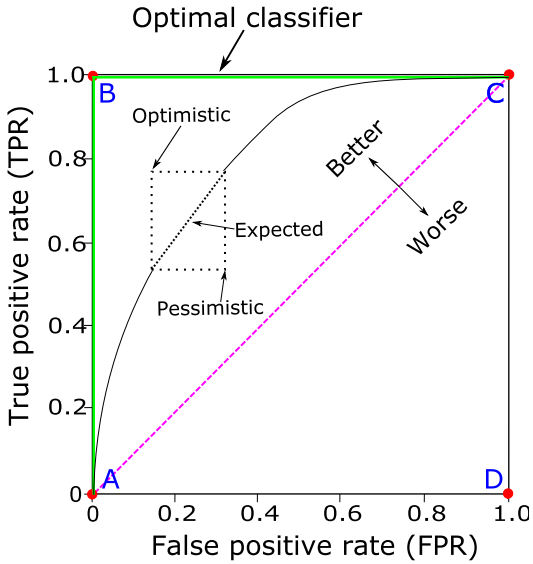
ROC and AUC

- The receiver operating characteristics (ROC) curve is a 2D graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The TPR represents the y-axis and FPR is the x-axis
  - TPR=Sensitivity=Recall
  - FPR=1-Specificity
- ROC is used to make a balance between the benefits, i.e., true positives, and costs, i.e., false positives
- AUC: Area Under the ROC (the bigger, the better)



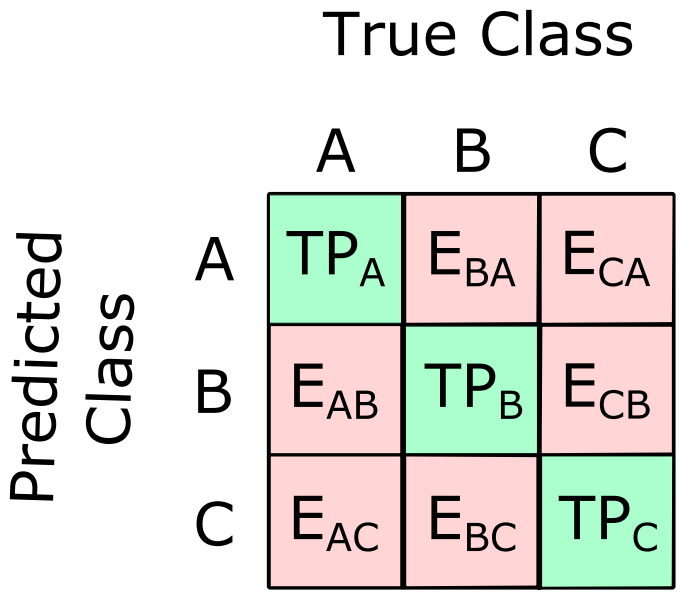
ROC and AUC (II)

- B represents the perfect classifier
- A random classifier would yield a result in the A-C line.
- The ROC curve has been used to evaluate many systems such as diagnostic systems, medical decision making systems, and other supervised learning systems
- For classifiers that produce a continuous output, the boundary between classes must be determined by a threshold value
  - Varying the threshold determines the curve
- The ROC is insensitive to the dataset balance



Confusion Matrix – Multi-class

- TP: Matrix Diagonal
- TP_i: True Positives of class i
- FP_i: The sum of the errors in the line of class i
  - $FP_A=E_{BA}+E_{CA}$
- FN_i: The sum of the errors in the column of class i
- TN_i: Add all values except those in line i and column i
  - $TN_A=TP_B+TP_C+E_{BC}+E_{CB}$



Confusion Matrix – Multi-class classification metrics

- All classification metrics can be calculated for each class i using TP_i, TN_i, FP_i and Fn_i
- The overall performance can be computed two ways:
  1. Micro-Average: Compute metrics based on the overall numbers (sum all TP, all TN, etc.). E.g.:

$$Pr_{micro} = \frac{TP_A + TP_B + TP_C}{TP_A + TP_B + TP_C + FP_A + FP_B + FP_C}$$

     - For single label problems Acc=Precision=Recall=F1
     - Note that Acc_{micro} = TP/(A+B+C)
  2. Macro-Average: Compute all metrics for each class and then average the results. E.g.:

$$Pr_{macro} = \frac{Pr_A + Pr_B + Pr_C}{3}$$

     - Note that F1 might produce a score that is not between Pr and Re

Predicted Class \ True Class	A	B	C
A	TP_A	E_{BA}	E_{CA}
B	E_{AB}	TP_B	E_{CB}
C	E_{AC}	E_{BC}	TP_C

Confusion Matrix – Multi-class classification metrics (II)

- Exercise – Calculate micro and macro Precision, Recall, F-Measure and Accuracy for this matrix:

Predicted \ True	A	B	C
A	12	22	21
B	25	998	15
C	5	15	32

- TP: Matrix Diagonal
- TP_i: True Positives of class i
- FP_i: The sum of the errors in the line of class i
  - $FP_A=E_{BA}+E_{CA}$
- FN_i: The sum of the errors in the column of class i
- TN_i: Add all values except those in line i and column i
  - $TN_A=TP_B+TP_C+E_{BC}+E_{CB}$

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

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

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

Confusion Matrix – Multi-class classification metrics (II)

- Exercise – Calculate micro and macro Precision, Recall, F-Measure and Accuracy for this matrix:

	TP	TN	FP	FN
A	12	1060	43	30
B	998	70	40	37
C	32	1057	20	36

$$PrA = \frac{TP}{TP+FP} = \frac{12}{12+43} = 0.22$$

$$ReA = \frac{TP}{TP+FN} = \frac{12}{12+30} = 0.29$$

$$AccA = \frac{TP+TN}{TP+TN+FP+FN} = \frac{12+1060}{12+1060+43+30} = 0.94$$

True

Predicted	A	B	C
A	12	22	21
B	25	998	15
C	5	15	32

	Pr	Re	F1	Acc
A	0.22	0.29	0.25	0.94
B	0.96	0.96	0.96	0.93
C	0.62	0.47	0.53	0.95
Micro	0.91	0.91	0.91	0.91
Macro	0.60	0.57	0.58	0.94

Dataset Balance

- Balanced datasets: all classes are equally represented
- Imbalanced datasets: some classes are more prevalent than others
- Some real-world data is heavily imbalanced. E.g.:
  - Number of cyberbullying tweets compared to non-cyberbullying (binary)
  - Number of positive cases for some diseases (binary);
  - News categorization (multi-class);
  - Etc.

Dataset Balance and Performance Metrics

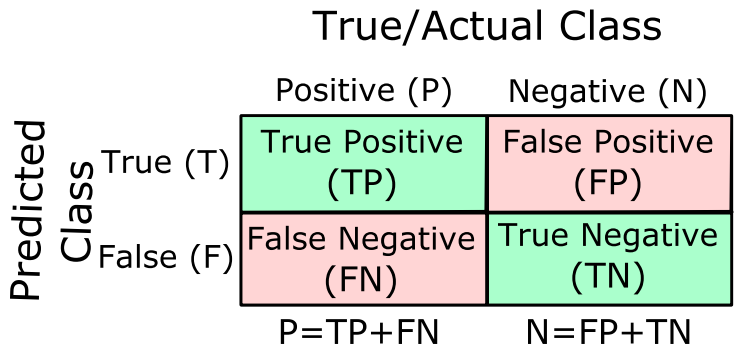
- Learning from imbalanced data is tricky!
  - Balancing the Training set is an option, **BUT** the Validation and Test sets should **ALWAYS** keep the imbalancing of the original dataset, otherwise the results are not reliable
  - Balancing techniques:
    - Duplicate minority class examples
    - SMOTE (Synthetic Minority Over-sampling Technique)
- Some metrics are insensitive to the data balance
  - Hence the results will be similar when testing on balanced or imbalanced datasets. However, using only such metrics hides the performance of the algorithm

Dataset Balance and Performance Metrics (II)

- Which metrics are insensitive to the data balance?
  - All that are calculated based on a single column of the confusion matrix
  - E.g.: Sensitivity (TPR, Recall), Specificity (TNR), ROC

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

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



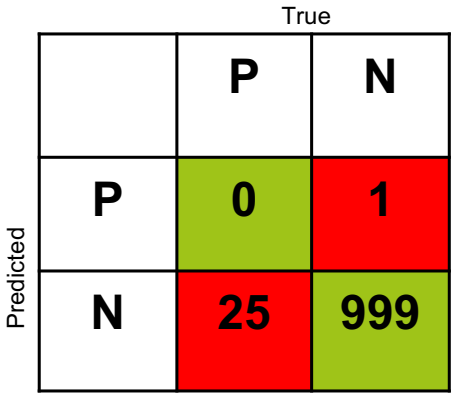
Dataset Balance and Performance Metrics (III)

- Precision is sensitive to data balance. Hence it is possible to have classifiers with excellent Pr in a balanced test set, and lousy Pr if the test set is imbalanced (while TPR and TNR don't change much)
- Q: What about Accuracy?
  - R: Is highly dependent on the majority class.
  - E.g. If the dataset is highly imbalanced towards the negative class, it is possible to have a very high Accuracy even if we don't get any positive correctly (25 positives, 1000 negatives)

$$Pr = \frac{0}{0+1} = 0$$

$$Re = \frac{0}{0+25} = 0$$

$$Acc = \frac{TP+TN}{TP+TN+FP+FN} = \frac{0+999}{0+999+1+25} = 0.97$$

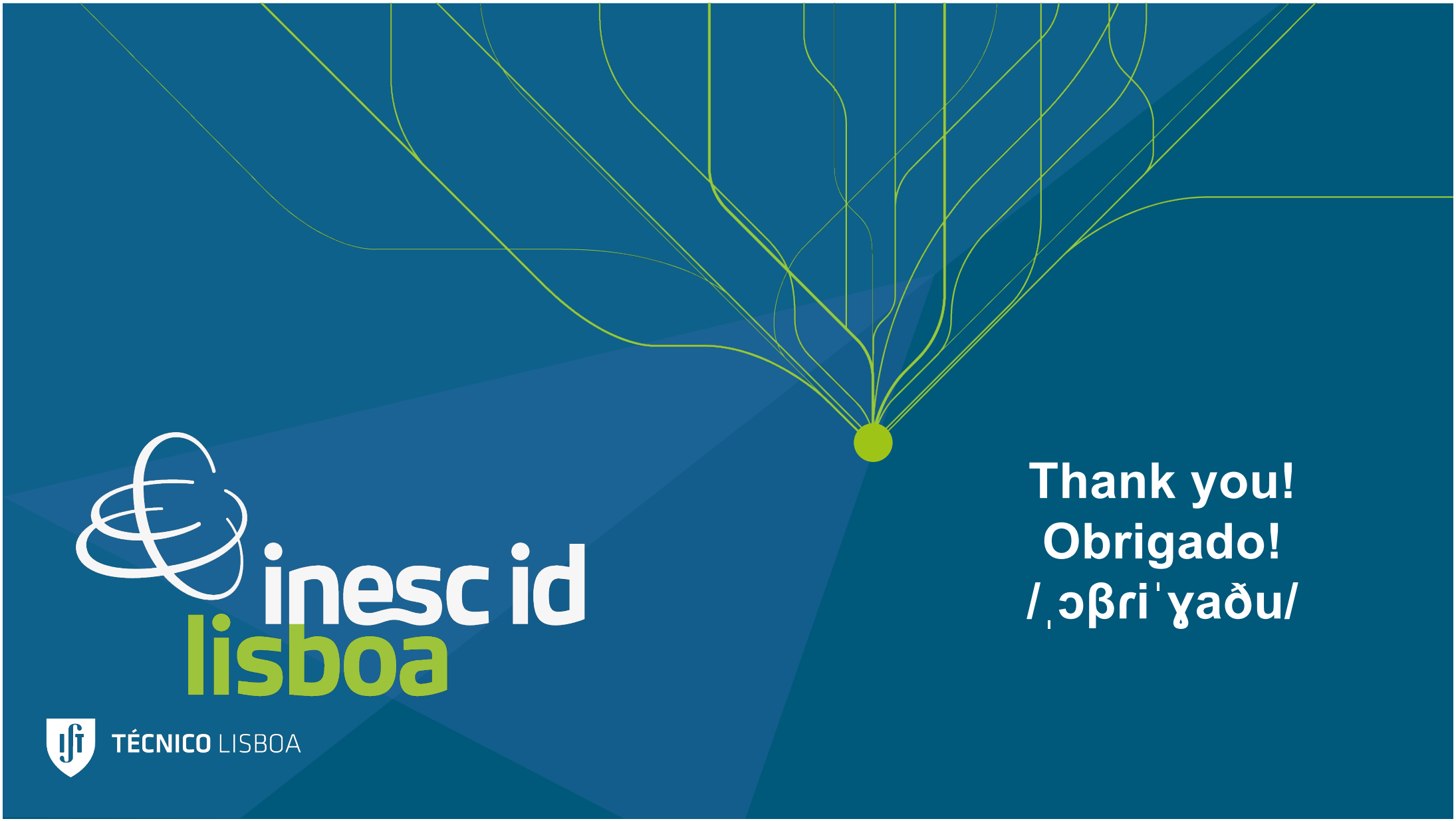


Baseline

- A baseline is the result of a very basic model/solution
- Your models should always get a better result than the baseline
- Simple baseline examples:
  - The most prevalent class in an imbalanced dataset
  - The Naïve forecast model
  - Naïve Bayes classifier (in text classification)
  - Etc.
- In forecasting problems, having a baseline is essential, since the RMSE by itself, doesn't say much about the performance of the model

Evaluation and Validation Made Easy...

- Scikit-learn.org offers and extensive and complete set of evaluation and evaluation metrics and tools
  - https://scikit-learn.org/stable/modules/model_evaluation.html#
- Using the available tools blindly should be avoided...
- ...but as long as you understand the basics underlying the evaluation and validation metrics, and know how to interpret them, feel free to use all the tools at your disposal!


inesc id
lisboa
TÉCNICO LISBOA
Thank you!
Obrigado!
/ˌɔβriˈɣaðu/