Machine Learning

Classification - Evaluation of a classifier

Claudio Sartori

DISI
Department of Computer Science and Engineering – University of Bologna, Italy
claudio.sartori@unibo.it

Model Selection

Evaluation and optimisation of a Classifier

Questions to be answered

- Which of the available models for classification is the best one?
- Which of the available algorithms is the best one?
- Which is the best parameters configuration?

⇒ Evaluation

The Oil Slick example I

- Detect oil slick (failures, illegal dumping) from satellite images, for early alarm
- Radar satellite images
    - Dark regions whose size and shape depend on weather and sea conditions
    - Look–alike dark regions can also be caused by local weather conditions, such as high winds
    - Manual detection by experts is definitely expensive and slow
- Scarcity of training data: oil spills are, fortunately, rare
- Unbalanced nature of data: the negative examples (non–spills) are predominant over the positive ones

The Oil Slick example II

- An automatic hazard detection system has been developed and marketed
  - Pre–selection of images for final manual processing
  - Necessary a tradeoff between undetected spills and false alarms
  - Evaluation of performance guides the tradeoff

The training set

- In supervised learning the training set performance is overoptimistic
- We need a lower bound for performance obtained by independent tests
- Supervised data are usually scarce, we need to balance the use of them between:
  - train
  - validation, to tune the parameter (sometimes it is omitted)
  - test
- Evaluate how much the theory fits the data
- Evaluate the cost generated by prediction errors

Learning and evaluation

- Empirically (and intuitively) the more training data we use the best performance we should expect
    - Statistically, we should expect a larger covering of the situation that can occur when classifying new data
- We must consider the effect of random changes
- The evaluation is independent from the algorithm used to generate the model

Controlling overfitting in a Decision Tree

- empirical observation: a simpler decision tree can reduce overfitting
- how can we simplify a decision tree?
- how can we decide the optimal level of simplification?

Decision tree pruning - the Scikit-Learn solution

Scikit-Learn allows to adjust pruning with one or more of the hyperparameters below

- `max_depth` - the maximum depth allowed for the tree; it is a horizontal cut, pruning all the branches below a given depth
  - this is the most important
- `min_samples_split` - either the minimum absolute number of samples or the minimum fraction of samples (with respect to the entire population in the dataset) in a node to make a split, if the threshold is not exceeded the node becomes a leaf
- `min_samples_leaf` - the minimum number of samples (or fraction, as above) required to be at a leaf node
- `min_impurity_decrease` - a node will be split if this split induces a decrease of the impurity greater than or equal to this value; if the weighted sum of the descendant leaves do not decrease from the node under consideration more than this threshold then the node becomes a leaf

Pruning and other

- pruning is a typical method to reduce overfitting and optimise a Decision tree
- we will see several other classification models
- each of them has several hyperparameters that can be tuned to reach the same objective

Accuracy of a classifier I

- The error frequency is the simplest indicator of the quality of a classifier
  - it is the sum of errors on any class divided by the number of tested records
- From now on, for simplicity, we will use the empirical error frequencies
  - remember that in real cases the maximum error frequencies should be used instead

Accuracy of a classifier II

- Accuracy and other more sophisticated indicators are used to:
    - compare different classifiers or hyperparameter settings
    - estimate the run–time performance we can expect, and therefore the cost of errors
        - with run–time performance we mean the performance observed when we will use the trained classification model with a classifier to produce value in some activity

The hyperparameters – a.k.a. model selection I

Optimising the model learned

- Every machine learning algorithm has one or more parameters that influence its behaviour
  - they are usually called hyperparameters
- It is crucial to obtain a highly reliable estimate of the run–time performance

The hyperparameters – a.k.a. model selection II

Optimising the model learned

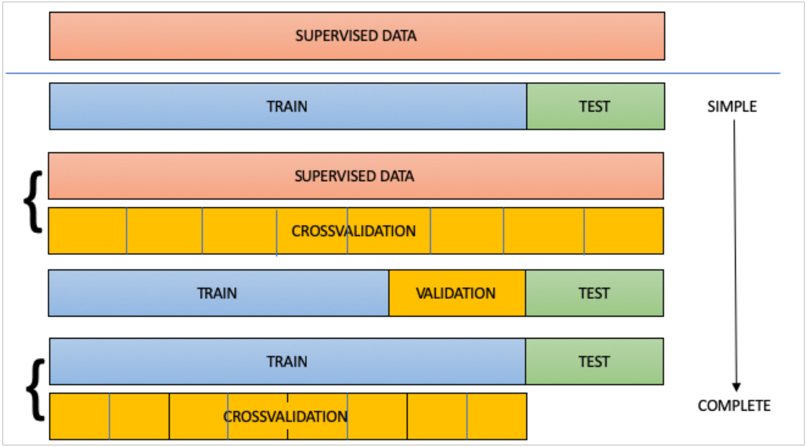
- Several train/test loops are in general necessary to find the best set of values for the hyperparameters
  - to be more general, the model selection includes the selection of the learning algorithm and its optimisation
- Sometimes it is necessary to find the best compromise between the optimisation step and the quality of the result
  - some learning algorithms require long computation times

Testing strategies

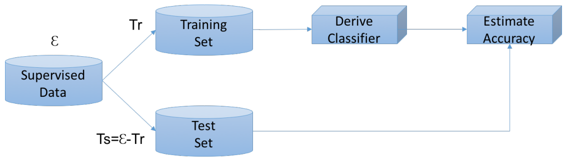
Getting the most out of the available supervised data

- as pointed out earlier, train and test should be done using different portions of the supervised data available
- in every step the data should be representative of the data that will be classified run–time
- a good strategy should
  - make the optimal usage of the supervised data
  - be compatible with the time constraints for the model selection

The train/test process: some alternatives



Holdout



- The split should be as random as possible
  - It may happen that the proportion of classes in the supervised dataset $\mathcal{X}$ is altered in the Training and Test sets, to prevent such cases the statistical sampling technique of stratification ensures the maintenance of the proportion of classes
- In this setting, the test set is used to obtain an estimation of the performance measures with new data

Dataset Size

- **Small Datasets**:
  - it's often beneficial to allocate more data for training to ensure the model has enough examples to learn from.
  - Typical splits might be 80/20 or even 90/10.
- **Large Datasets**:
  - you can afford to reserve more data for testing without compromising the training process.
  - Common splits are 70/30 or 80/20.

Model Complexity

- **Simple Models**:
  - require less data to train effectively, then you can afford a larger test set.
- **Complex Models**:
  - e.g. deep learning models often require more training data, so a split like 90/10 or 85/15 might be more appropriate.

Variance and Overfitting

- **High Variance Models**:
  - Models that are prone to overfitting may benefit from more training data to capture the underlying patterns better.
- **Cross-Validation**:
  - Using techniques like k-fold cross-validation can help mitigate overfitting and provide a more robust evaluation, reducing the dependence on a single train/test split.

Evaluation Stability

- **Stable Performance**:
  - A larger test set provides a more stable and reliable estimate of model performance, reducing variance in the evaluation metrics.
- **Rare Events**:
  - If the dataset contains rare events or classes, ensuring that these are adequately represented in both the training and test sets is crucial. Stratified sampling can help with this.

Computational Resources

- **Resource Constraints**:
  - Larger test sets mean more computational resources for evaluation. In resource-constrained environments, a smaller test set might be necessary.

Business and Domain Considerations

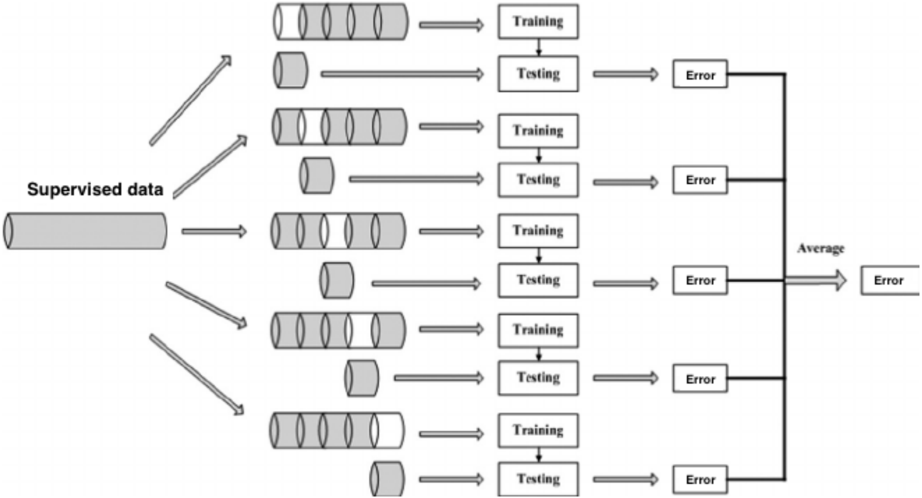
- **Critical Applications**:
  - In high-stakes applications (e.g., medical diagnosis, autonomous driving), ensuring a robust and reliable evaluation is crucial, often requiring a larger test set.
- **Regulatory Requirements**:
  - Some domains may have specific guidelines or standards that dictate the proportion of data to be used for testing and validation.

Cross Validation (k–fold)

A more complex strategy

- The training set is randomly partitioned into k subsets
  - If necessary, use stratified partitioning
- k iterations using one of the subsets for test and the others for training
- Combine the result of tests
- Generate the final model using the entire training set
- Optimal use of the supervised data
  - each record is used $k-1$ times for training and once for testing
- Typical value: $k = 10$

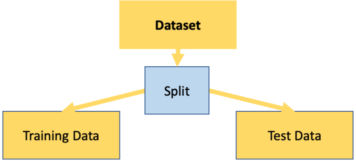
Cross Validation



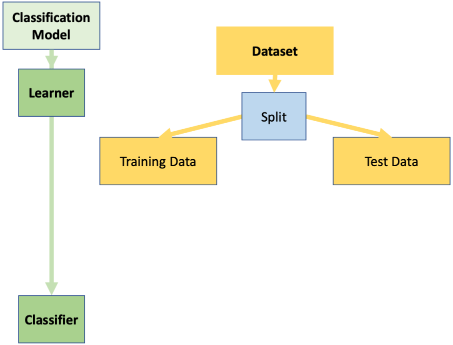
Cross–validation workflow

Dataset

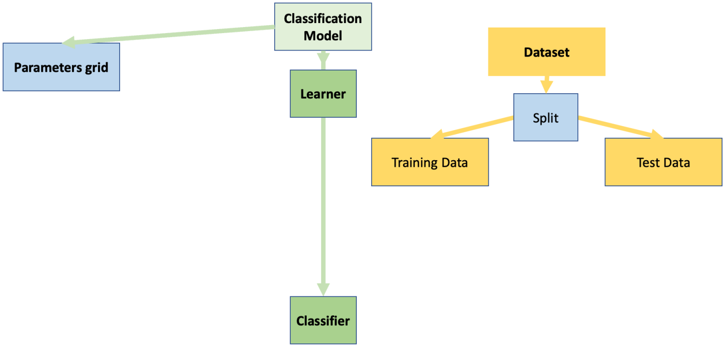
Cross–validation workflow



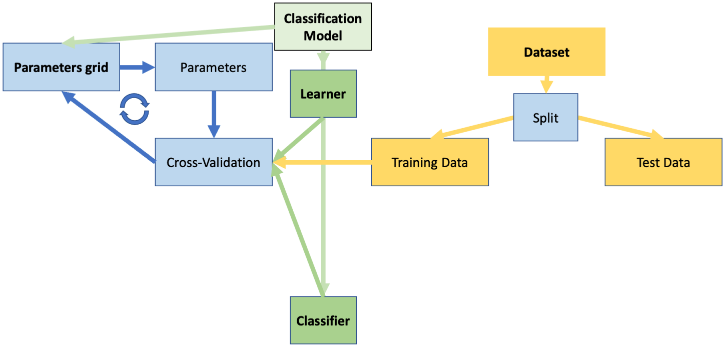
Cross–validation workflow



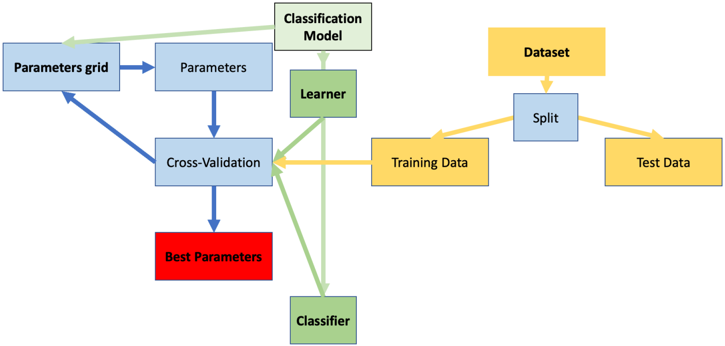
Cross–validation workflow



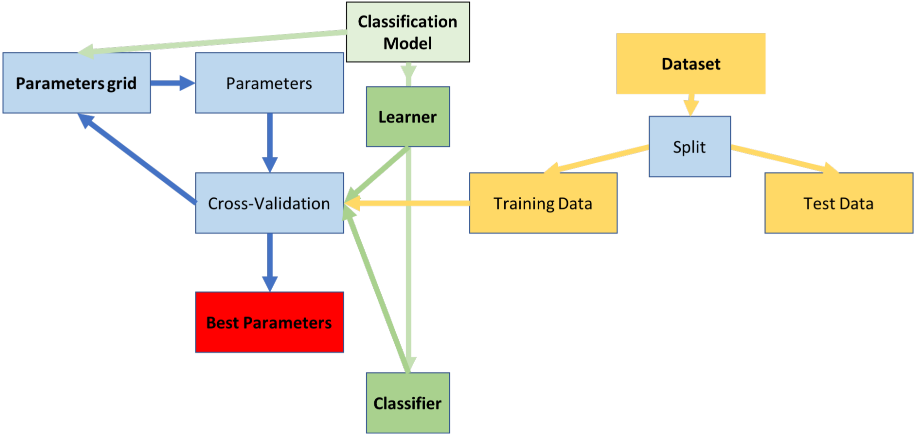
Cross–validation workflow



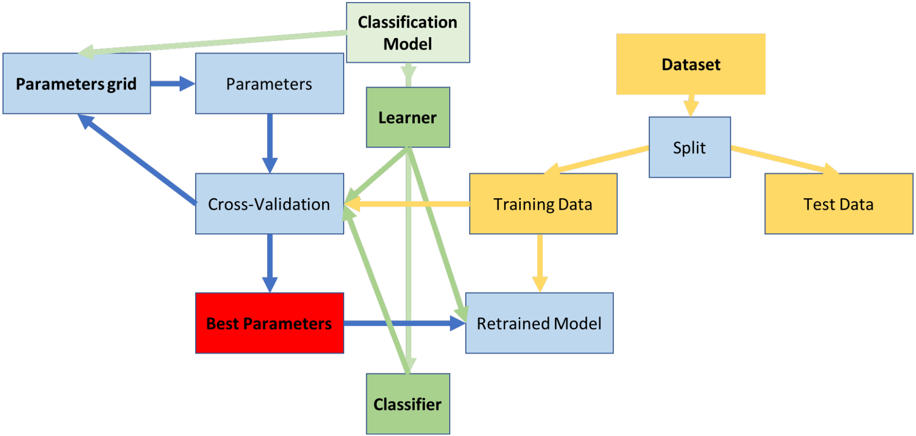
Cross–validation workflow



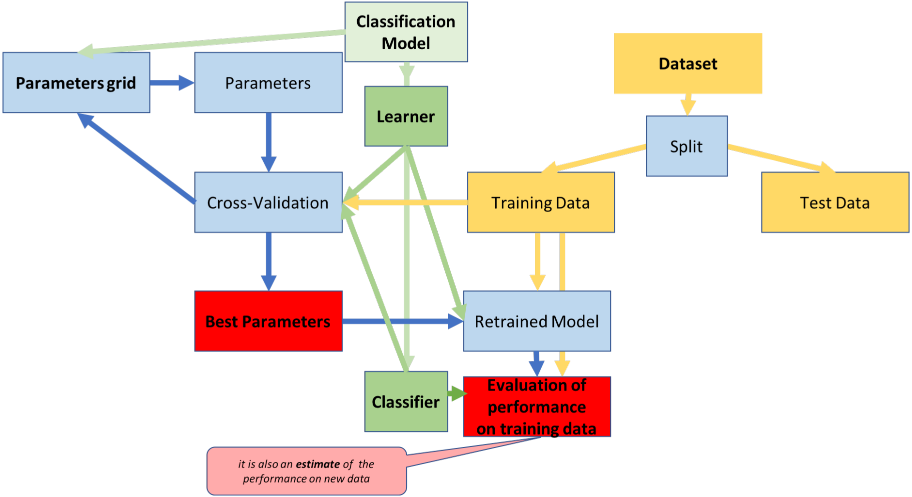
Cross–validation workflow



Cross–validation workflow

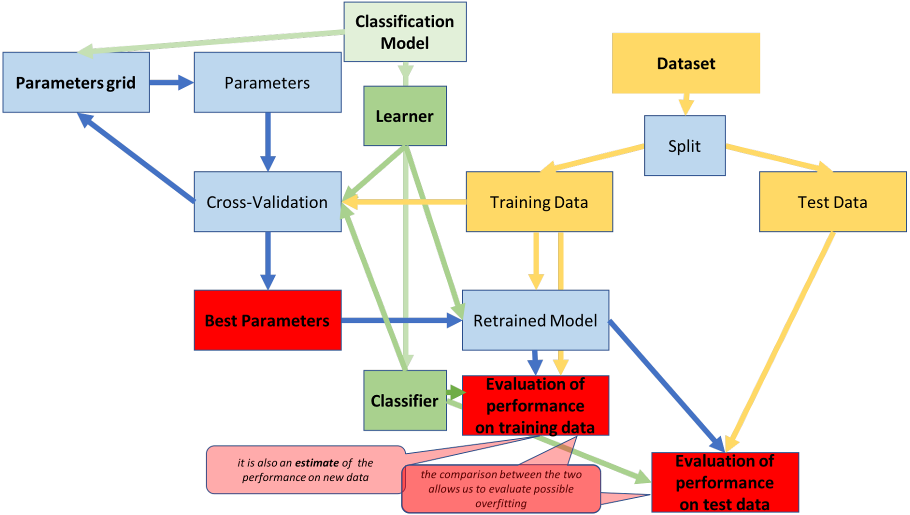


Cross–validation workflow



*it is also an **estimate** of the performance on new data*

Cross–validation workflow


Classification Model
Parameters grid
Parameters
Learner
Dataset
Split
Cross-Validation
Training Data
Test Data
Best Parameters
Retrained Model
Classifier
Evaluation of performance on training data
Evaluation of performance on test data
it is also an ***estimate*** of the performance on new data
the comparison between the two allows us to evaluate possible overfitting

Cross Validation – pros and cons

- ☹ The train/test loop is repeated k times
- ☺ The estimate of the performance is averaged on k runs

  ⇒ more reliablity
- ☺ All the examples are used once for testing
- ☺ The final model is obtained using all the examples

  ⇒ best use of the examples

Train, Validation, Test – pros and cons

- ☺ The train/validation loop is faster than the Cross Validation
- ☺ The optimisation of the hyperparameters is done with the validation set, independent from the final evaluation

  ⇒ more reliable than the simple holdout
- ☹ The test during the hyperparameters optimisation is done on a portion of the examples (the validation set)

  ⇒ less reliable with respect to Cross Validation

Model selection – Zero level

- *Use of supervised data*
  - Entire dataset
- *Hyperparameters*
  - defaults
- *Action*
  - Train the model with the entire dataset, make predictions on the entire dataset and compare the predictions with the ground truth
- *Quality of results*
  - probably not the best possible with that model
- *Reliability of evaluation for generalisation*
  - Not reliable
- *Observations*
  - Just a first trial
- *`sklearn` model selection procedures*
  - none

Model selection – Quick attempt

- *Use of supervised data*
  - split dataset into Train and Test
- *Hyperparameters*
  - defaults
- *Action*
  - train the model with the train set, make predictions on the test set and compare the predictions with the ground truth
- *Quality of results*
  - probably not the best possible with that model
- *Reliability of evaluation for generalisation*
  - better estimation of the expected performance
- *Observations*
  - If you are in a hurry and/or the supervised dataset is small
- *`sklearn` model selection procedures*
  - `train_test_split`

Model selection – Standard

- *Use of supervised data*
  - Split dataset into Train and Test
- *Hyperparameters*
  - Find appropriate ranges of hyperparameters
- *Action*
  - For a number of combinations of hyperparameter values train the model with the Train set, predict with the Test set and evaluate the result. Choose the combination of hyperparameters that gives the best result, train the model with the Train set, predict and evaluate with the test set
- *Quality of results*
  - improved with respect to the defaults
- *Reliability of evaluation for generalisation*
  - possible overestimation, because the test data are involved in the optimisation
- *Observations*
  - Not too time consuming: the number of train/validate opeations is the number of combinations of hyperparameter values.

    Not good for very small datasets.
- *`sklearn` model selection procedures*
  - `train_test_split`, `ParameterGrid`

Model selection – Train/Validation/Test

- *Use of supervised data*
  - Split into Train-Validate and Test, then split Train-Validate into Train and Validate
- *Hyperparameters*
  - Find appropriate ranges of hyperparameters
- *Action*
  - For a number of combinations of hyperparameter values train with the Train set, predict with the Validate set and evaluate the result. Choose the combination of hyperparameters that gives the best result, train with the Train-Validate set and evaluate with the Test set
- *Quality of results*
  - Optimised for the model
- *Reliability of evaluation for generalisation*
  - Good reliability of the estimated performance
- *Observations*
  - Not too time consuming: the number of train/validate operations is the number of combinations of hyperparameter values.

    Not good for very small datasets
- *`sklearn` model selection procedures*
  - `train_test_split` two times, `ParameterGrid`

Model selection – Cross Validation

- *Use of supervised data*
  - Split dataset into Train and Test
- *Hyperparameters*
  - Find appropriate ranges of hyperparameters
- *Action*
  - For a number of combinations of hyperparameter values and do Cross Validation. Choose the combination of hyperparameters that gives the best result, train the model with the Train set and evaluate with the Test set
- *Quality of results*
  - Optimised for the model
- *Reliability of evaluation for generalisation*
  - Best reliability of the estimated performance
- *Observations*
  - Time consuming: the number of train/validate operations is the number of combinations of hyperparameter values times the number of cross validation folds.

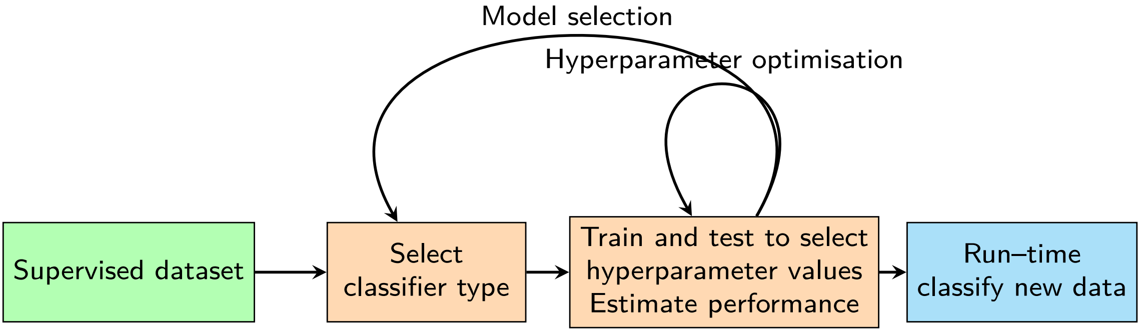
    Not good for very small datasets
- *`sklearn` model selection procedures*
  - `train_test_split`, `GridSearchCV` or `RandomizedSearchCV`

The model–selection process for the development of a classifier – I

A wider selection process

- We can use different models for doing classification
  - decision tree is one of them
- The general process implies to try several models, for each of them find the best hyperparameters and, at the end, use the model giving the best results with the best hyperparameters to classify new (unsupervised) data
  - later we will refer to this "productive" phase as run–time

The model–selection process for the development of a classifier – II


Model selection
Hyperparameter optimisation
Supervised dataset
Select classifier type
Train and test to select hyperparameter values Estimate performance
Run–time classify new data

Performance measures of a classifier

What does best in page 44 mean?

Binary prediction

For simplicity: Positive/Negative

- success rate = accuracy

$$\frac{TP + TN}{N_{test}}$$

- error rate

$$1 - \text{success rate}$$

		Predicted class		
		pos	neg	Total
True class	pos	TP	FN	T_{pos}
	neg	FP	TN	T_{neg}
	Total	P_{pos}	P_{neg}	N_{test}

Confusion matrix

Is accuracy enough? I

- Is the accuracy the only performance indicator for a classifier? Other possible indicators:
  - Velocity
  - Robustness w.r.t. noise
    - i.e. training data with bad class label
  - Scalability
  - Interpretability

Is accuracy enough? II

- A classification error can have different consequences, depending on the class of the individual
    - when forecasting an illness a false positive can be less dangerous than a false negative
        - unless the cares or the additional examinations are dangerous or invasive
    - consider the cost of retiring a machinery as damaged, while it is ok (false positive) and the cost of an unpredicted failure (false negative)

A summary of measures I

Precision – $TP/(TP + FP)$

the rate of true positives among the positive classifications

Recall – $TP/(TP + FN)$

the rate of the positives that I can catch (a.k.a. Sensitivity)

Specificity – $TN/(TN + FP)$

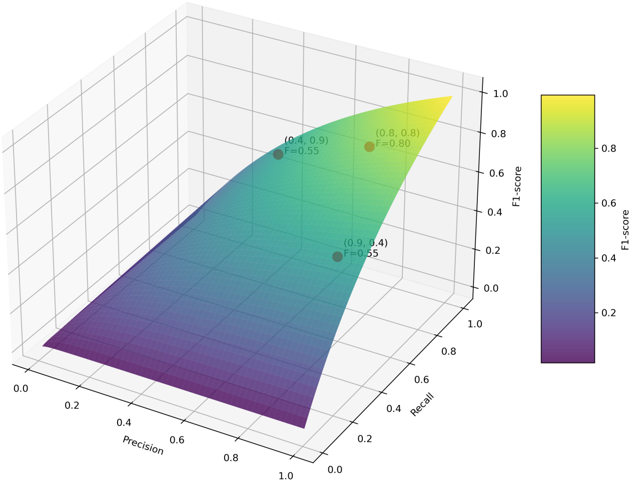
the rate of the negatives that I can catch

A summary of measures II

F1–score – the armonic mean of precision and recall, a.k.a. balanced F-score

$$F = 2\frac{\text{prec} \cdot \text{rec}}{\text{prec} + \text{rec}}$$

F1–score as a function of Precision and Recall



Which measure should we use?

A rule of thumb

- Accuracy gives an initial feeling of the effectiveness of the classifier, but can be heavily misleading when classes are highly imbalanced
    - it considers globally all the classes, also when $C > 2$ (non-binary case, also called multi–class)
- F1–score is always interesting, because has higher values when precision and recall are reasonably balanced
- if the the costs of errors on positives and negatives are significantly different, then it is necessary to evaluate precision and recall.
- for multi–class problems see page 57.

Multi–class case

- The confusion matrix of page 48 is easily extended when there are more than two classes
- Each cell contains the number of test records of class i and predicted as class j
- The numbers in the main diagonal are the true predictions

Confusion matrix with three classes, say a, b, c

$T_i =$ true number of i labels in the dataset

$P_i =$ total number of i predictions by a given classifier, say $\overline{C}$

$TP_i =$ number of true predictions for class i given by classifier $\overline{C}$

$FP_{i-j} =$ number of false prediction for class i predicted as j

$$\text{accuracy} = \frac{\sum_i TP_i}{N}$$

$$\text{precision}_i = \frac{TP_i}{P_i}$$

$$\text{recall}_i = \frac{TP_i}{T_i}$$

		Predicted class			
		a	b	c	Total
True class	a	TP_a	FP_{a-b}	FP_{a-c}	T_a
	b	FP_{b-a}	TP_b	FP_{b-c}	T_b
	c	FP_{c-a}	FP_{c-b}	TP_c	T_c
	Total	P_a	P_b	P_c	N

Multi-class evaluation I

The number of classes is more than two

- precision, recall and f1–score are intrinsically defined for a single class;
- in binary classification they are commonly referred to the so–called positive class;
- in multi-class cases, the scorers of `scikit-learn` produce an array of values, one for each class
- when a single value is necessary to optimise the hyper parameters, as in `GridSearchCV`, if we need to maximise one of precision, recall, f1–score an average value is required

Multi-class evaluation II

The number of classes is more than two

Averaging of measure f for classes $c_i \in \mathcal{C}$, each class with frequency C_i

- macro average: $f(\mathcal{C}) = \frac{\sum f(c_i)}{C}$
- weighted average: $f(\mathcal{C}) = \frac{\sum f(c_i) * C_i}{C}$
- micro average: the definition differs for the different measures:

$$\text{precision\_micro} = \frac{\sum TP_i}{\sum(TP_i + FP_i)} \qquad \text{recall\_micro} = \frac{\sum TP_i}{\sum(TP_i + FN_i)}$$

$$\text{f1\_micro} = \frac{2 * \text{precision\_micro} * \text{recall\_micro}}{\text{precision\_micro} + \text{recall\_micro}}$$

Multi-class evaluation III

The number of classes is more than two

- with macro average the measure of each class has the same impact on the average value, therefore in case of imbalancing the minority classes have influence bigger than their size
- with wighted average the measure of each class influences the result in proportion with its size, therefore in case of imbalancing the minority classes have smaller influence
- with micro average precision and recall are computed globally across all samples, f1_micro is defined on the basis of precision_micro and recall_micro, in analogy with the binary case

Use cases for micro averaging

- When class imbalance is not a concern
- When evaluating overall system performance
- Search engines, multi-class classification, medical diagnosis (in case of discrimination among many diseases)

Use cases for macro averaging

- When all classes should be treated equally
- When class imbalance exists but all classes must contribute equally
- Sentiment analysis, rare disease detection, multi-class classification

Use cases for weighted averaging

- When class imbalance exists, and larger classes should influence the result more
- Spam filtering, product recommendation, customer churn prediction

Comparison Table

Metric	Handles Imbalance?	Emphasizes Large Classes?	Treats All Classes Equally?
*measure*_micro	No	Yes	No
*measure*_macro	Yes	No	Yes
*measure*_weighted	Yes	Yes	No

Summary on averaging multi–class performance for single–class scores

precision, recall, f1–score

- **Use score_micro** when overall performance matters
  - for recall the micro and weighted average are *mathematically equivalent*
- **Use score_macro** when class importance should be equal
  - therefore the influence of smaller classes is bigger than their actual presence
- **Use score_weighted** when larger classes should have more influence

Beyond pure counting

Taking into account the “a priori” information

- Is it likely to obtain a correct prediction by chance?
- Example: early diagnosis
  - let us consider a disease affecting 2% of patients
  - a prediction saying always “no disease” has 98% precision, which, in general would be a good result
  - the evaluation of a prediction should take this as a baseline, and, possibly, look for improvements

Example of confusion matrix

- Confusion matrix of classifier $\overline{C}$ on a given dataset
- 140 correct predictions
- The predicted proportion of classes is 100:60:40

		Predicted class			
		a	b	c	Total
True class	a	88	14	18	120
	b	10	40	10	60
	c	2	6	12	20
	Total	100	60	40	200

Confusion matrix of a random classifier $R_{\overline{C}}$

A virtual experiment

- A random classifier $R_{\overline{C}}$ producing the same proportion of classes as $\overline{C}$
  - the horizontal margin is the same as $\overline{C}$
- The rows have all the same proportion as the horizontal margin
- 82 predictions are exact by chance
  - the sum of the main diagonal of $R_{\overline{C}}$

		Predicted class			
		a	b	c	Total
True class	a	60	36	24	120
	b	30	18	12	60
	c	10	6	4	20
	Total	100	60	40	200

Taking into account the random component

- The improvement of $\overline{C}$ over $R_{\overline{C}}$ is $140 - 82 = 58$
- The improvement of the perfect classifier is $200 - 82 = 118$
- We define $\kappa(\overline{C})$ the improvement of the classifier at hand w.r.t. the improvement of the perfect classifier

$$\kappa(\overline{C}) = 58/118 = 0.492$$

κ statistic[Cohen(1960)]

- Evaluates the concordance between two classifications
  - in our case between the predicted and the true one
- Fraction of concordance observed $\mathbf{Pr}(o) = \frac{TP_a + TP_b + TP_c}{N}$
- Expected fraction of concordance for a random assignment $\mathbf{Pr}(e) = \frac{T_a * P_a + T_b * P_b + T_c * P_c}{N^2}$
- κ is the ratio between the concordance exceeding the random component and the maximum surplus possible

$$-1 \leqslant \kappa = \frac{\mathbf{Pr}(o) - \mathbf{Pr}(e)}{1 - \mathbf{Pr}(e)} \leqslant 1$$

Definition of Cohen's Kappa for Binary Classification

Cohen's Kappa Score can be expressed in terms of the confusion matrix components *TP*, *FP*, *TN*, and *FN* for binary classification.

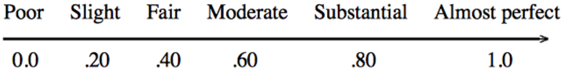
Formula

$$\kappa = \frac{2 \times (TP \times TN - FP \times FN)}{(TP + FP)(FP + TN) + (TP + FN)(FN + TN)}$$

Range of κ

- 1 for perfect agreement
  - $TP_a + TP_b + TP_c = N$
- -1 for total disagreement
  - $TP_a + TP_b + TP_c = 0$ and there is a perfect swap between predictions and true labels
    - if all classes have non–zero counts -1 is possible only if the number of labels is two
- 0 for random agreement

$\kappa =$



Definition of Matthews Correlation Coefficient (MCC)

It is a metric for evaluating binary classifications, especially effective when classes are imbalanced. It measures the correlation between observed and predicted classifications.

Formula

$$\mathrm{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Properties of MCC

- **Range:** MCC values range from -1 to 1.
  - 1: Perfect prediction
  - 0: Prediction no better than random
  - -1: Complete disagreement between predictions and actuals
- **Balanced Evaluation:** MCC is a balanced metric, particularly useful for imbalanced datasets since it accounts for all components of the confusion matrix.

The cost of errors

- Our decisions are driven by predictions
- Bad predictions imply a cost
- Examples
  - grant a loan to a person who turns out to be a bad payer costs more than denying a loan to a person that could be a good payer
  - a false "oil spill" alarm is less expensive than an undetected spill
  - a wrong "fault prediction" in an industrial plant is in general less expensive than an unexpected fault disabling the plant and creating damages
  - in direct marketing, sending advertisement material without redemption is less harmful than the loss of business if a promising customer is ignored

Cost sensitive learning I

Weight the errors

- Alternative 1: alterate the proportion of classes in the supervised data, duplicating the examples for which the classification error is higher
    - In this way, the classifier will became more able to classify the classes for which the classification error cost is higher
    - This solution is useful also when the classes are imbalanced, that is the frequencies of the class labels in $\mathcal{X}$ are not equal

Cost sensitive learning II

- Alternative 2: some learning schemes allow to add weights to the instances
  - e.g. the `DecisionTreeClassifier` of Scikit-Learn has the hyperparameter **class_weight**: it allows to define a dictionary, with one key per distinct class, specifying the relative weight to be assigned to each class, the optimisation of the fitting will be adjusted accordingly
    - the `balance` option balances the classes automatically

1 Model selection 2

2 Model selection for a classifier 11

3 Performance measures of a classifier 46

4 Evaluation of a probabilistic classifier 77

- Lift Chart 81
- ROC Curve 84

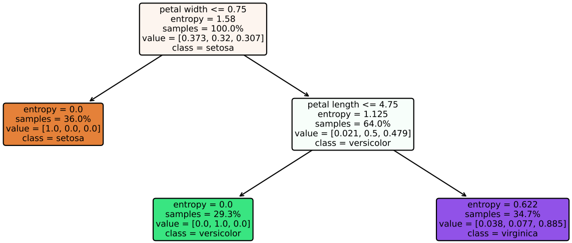
5 Statistical issues about the evaluation of a classifier OPTIONAL 88

Predicting probabilities of classes I

- Many classifiers produce, rather than a class label (crisp prediction), a tuple of probabilities, one for each possible class, (probabilistic, or soft prediction)
- The adequacy of one output or the other depends on the application domain
  - when an immediate decision is required the crisp output is necessary
  - when the classification is part of a process including several evaluation/action steps the probabilistic output can be more appropriated

Crisp values sometimes hide probabilities

- For example, when a leaf of a decision tree has non–zero counts for the minority classes a less–than–one probability, the probabilities of the examples falling in that leaf can be assigned on the basis of the fractions of the training data elements in that leaf belonging to each class
- Consider the rightmost leaf in the figure (it is the pruned tree of the first module on Classification) [1]



[1] Since it is quite common to have leaves with a small number of examples and/or minority classes with frequencies near to zero, smoothing techniques are used to adjust the probabilities

Probabilities to crisp

- Probabilities can be converted to a crisp value with different techniques, depending on the number of classes (binary or multiclass)
    - binary – set a threshold for the positive class
    - multiclass – output the class with the maximum probability

Binary – Lift Chart

- Used to evaluate various scenarios, depending on the application
- Consider a dataset with 1000 positives and apply a probabilistic classification scheme
- Sort all the classified elements for decreasing probability of positive class
- Make a bi-dimensional chart with axes
  x = sample size, y = number of positives in sample
- Only the rank is important, not the specific probability

Lift and Cumulative Gains Chart – I

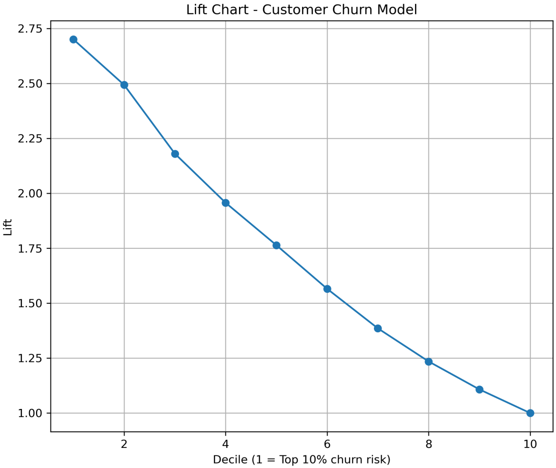
Lift Chart

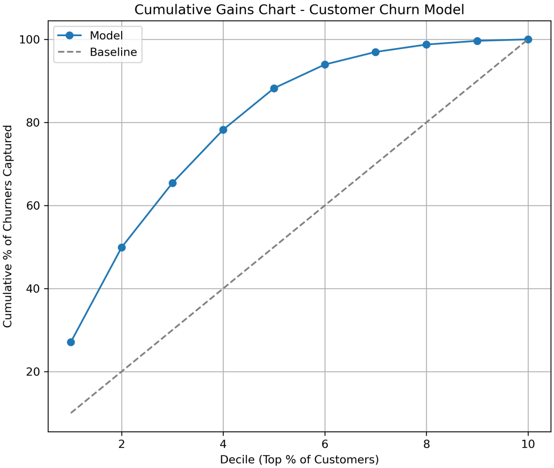
- for each fraction of data, with decreasing probability of positivity, shows a multiplication factor of the likelihood of positivity with respect to random choice

Cumulative Gains Chart

- The straight line plots the number of positives obtained with a random choice of a sample of test data
- The curve plots the number of positives obtained drawing a fraction of test data with decreasing probability
- The larger the area between the two curves, the best the classification model

Lift and Cumulative Gains Chart – II





ROC Curve

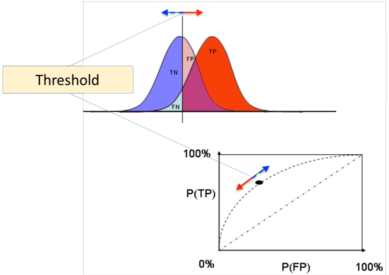
Receiver-Operator Characteristic

History: interpretation of radar signals during WWII

- Tradeoff between hit rate and false alarm in a noisy channel
- The noise can be such that the recognition of the transmission is altered
- The noise alters the two levels according to a gaussian distribution
- Problem: set the positive/negative threshold in order to maximize the tradeoff above, according to application–dependent requirements

ROC Curve

- With less noise the two gaussian curves are better separated
- Moving the threshold towards right increases both the rate of true positives and false positives caught
- The area between the non–discrimination line and the ROC curve is a quality index of the line
- The maximum area is the upper left triangle



[a]

[a] Image from Wikipedia
TN = blue + cyan = probability of a negative to be caught
FN = cyan = probability of a negative to be missed
FP = pink + purple = probability of a positive to be missed
TP = red + purple = probability of a positive to be caught

ROC for a soft classifier

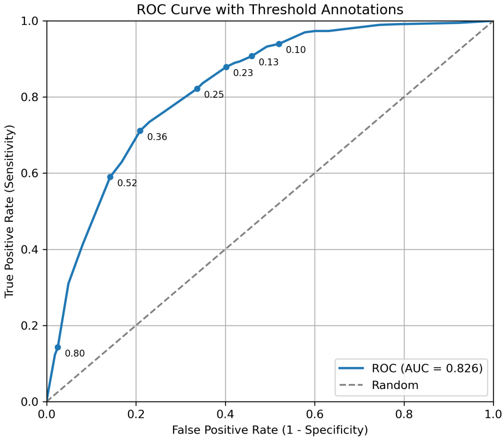
- The soft classifier can be converted into a crisp one by choosing a threshold ⇒ predict positive if the probability of the test record exceeds the threshold
- Varying the threshold the behavior of the classifier changes, by changing the ratio of TP and FP
- Threshold steps allow to track the ROC curve
    - sort the test elements by decreasing positive probability
    - set the threshold to the highest probability, set TP and FP to zero
    - repeat
        - update the number of TP and FP with probability from the threshold to 1
        - draw a point in the curve
        - move to next top probability of positive
    - end repeat

Drawing the ROC curve for a soft classifier

Optional

Sampled records, with label, predicted probability and positive rate

original_index	y	y_pred_prob	positive_rate
2797	1	0.840	1.000
6365	1	0.840	1.000
2708	1	0.795	1.000
5706	1	0.681	1.000
5716	0	0.562	0.800
1446	0	0.562	0.667
2699	0	0.521	0.571
4028	0	0.284	0.500
3762	0	0.227	0.444
1259	1	0.127	0.500
2810	0	0.104	0.455
4552	0	0.091	0.417
2525	0	0.039	0.385
4361	0	0.027	0.357
4886	0	0.027	0.333
1116	0	0.027	0.312
839	0	0.017	0.294
5756	0	0.017	0.278
405	0	0.000	0.263



The meaning of the test error

- let us suppose that the test set is a good representation, on the average, of the entire dataset $\mathcal{X}$ (i.e. run–time)
- the relationship between the training set and $\mathcal{X}$ will be subject to probabilistic variability
- the evaluation can be done either at different levels
  - general – the whole performance of the classifier
  - local – the local performance of a component of the model, i.e. a node of a decision tree
- if the test set error ratio is x, we should expect a run–time error $x \pm$???

$\Rightarrow$ $\boxed{\text{confidence interval}}$

Confidence interval in error estimation

Bernoulli process

- forecasting each element of the test set is like one experiment of a Bernoulli process
  - good prediction ⇒ success
  - bad prediction ⇒ error
- the same as N independent binary random events of the same type
- $f = S/N$ = empirical frequency of error
- which is the probability p of error?

Empirical frequency and true frequency

- Deviations of the empirical frequency from the true frequency are due to noise
- Usually, noise is assumed to have a normal distribution around the true probability (for $N \geqslant 30$)
- We need statistical reasoning to determine a pessimistic evaluation of p given f, N, and the amount of risk that we can tolerate
  - we can estimate the interval centred in f such that p will be in that interval with confidence $1 - \alpha$, where α is typically .05 or 0.01

$$P\left(z_{\alpha/2} \leqslant \frac{f - p}{\sqrt{p(1-p)/N}} \leqslant z_{1-\alpha/2}\right) = 1 - \alpha$$

Range error estimate

Wilson score interval

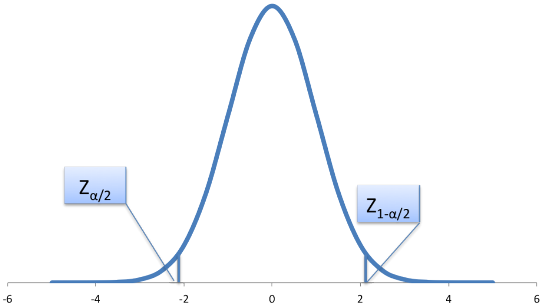
- z depends on the desired confidence level $1 - \alpha$
- it is the abscissa delimiting the area $1 - \alpha$ for a normal distribution
  - i.e. the inverse of the standard cumulative normal distribution
- with a little of algebra

$$\frac{1}{1 + \frac{1}{N}z^2}\left[f + \frac{1}{2N}z^2 \pm z\sqrt{\frac{1}{N}f(1-f) + \frac{1}{4N^2}z^2}\right]$$

The pessimistic error is obtained by substituting ± with +

Confidence level in error estimation

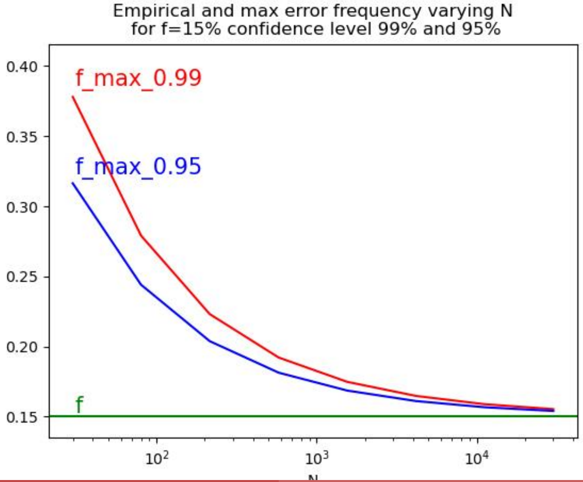
The confidence level $1 - \alpha$ is the probability that the estimation of the frequency of good predictions is good, i.e. the probability that the true frequency is inside the confidence interval around the empirical frequency



$1-\alpha$	Z
0.99	2.58
0.98	2.33
0.95	1.96
0.90	1.65
0.75	1.04
0.50	0.67

Confidence interval in error estimation

Increasing N, with constant empirical frequency, the uncertainty for p narrows.

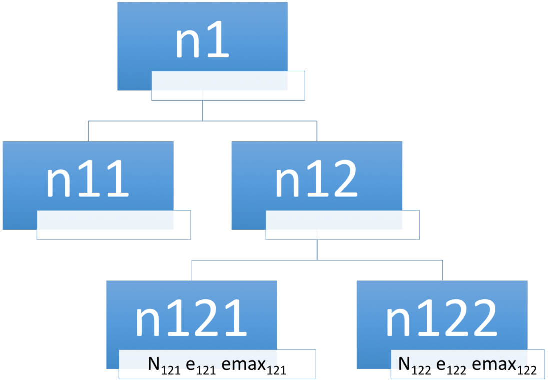


Statistical pruning of DT with error estimation

The C4.5 strategy

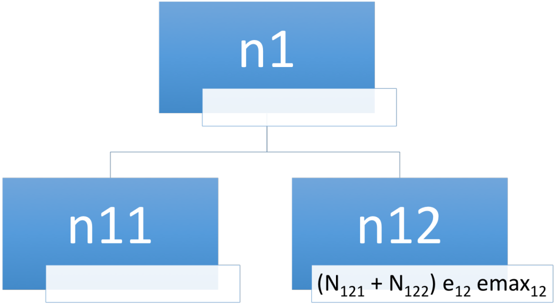
- Consider a subtree near the leaves
- Compute its maximum error e_l as a weighted sum of the maximum error of the leaves
- Compute the maximum error e_r of the root of the subtree transformed into a leaf
- Prune if $e_r \leqslant e_l$
- With pruning, the error frequency increases, but the number of records in node also increases, therefore the maximum error can decrease

DT before pruning


n1
n11
n12
n121
N_{121} e_{121} $emax_{121}$
n122
N_{122} e_{122} $emax_{122}$

N_x = Records in node f_x = Error frequency in node e_x = Maximum error frequency in node

DT after pruning



$e_x - f_x$ increases as N_x decreases

Pruning is done if $(N_{121} + N_{121}) * e_{12} < N_{121} * e_{121} + N_{122} * e_{122}$

Bibliography

- Jacob Cohen.
  A coefficient of agreement for nominal scales.
  Educational and Psychological Measurement, 20(1):37–46, 1960.
  doi: 10.1177/001316446002000104.
  URL http://dx.doi.org/10.1177/001316446002000104.