Natural Language and Information Retrieval
Degree in Data Science
ETSINF. Academic year 2022-2023

Practical Work 3

Session 3: Validation of the Results

Reynier Ortega Bueno
rortega@prhlt.upv.es

Goal

The main aim of this lab session is to equip students with a comprehensive understanding of the various quality measures and validation techniques used to evaluate the performance of classification or regression algorithms. The session covers the main quality measures and validation schemes currently in use. In addition, these criteria will be used to evaluate the models developed to participate in the HUHU shared task.
For that, the scikit-learn Python library should be used.

Bibliography

- https://scikit-learn.org/stable/modules/model_evaluation.html
- https://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection
- https://scikit-learn.org/stable/modules/cross_validation.html
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearcphCV.html#sklearn.model_selection.RandomizedSearchCV
- Distributions: https://docs.scipy.org/doc/scipy/reference/stats.html

EVALUATION MEASURES

Evaluation measures in machine learning are used to quantify the performance of a model in terms of its ability to make accurate predictions on new unseen data. The specific evaluation measures used depend on the type of problem being solved (e.g., classification, regression, clustering) and the desired predictions. The basic idea of evaluating a classifier is to measure the number of cases that are correctly classified and the number of misclassifications. In case of regression models the aim is to evaluate the divergence between the predicted and the real value of the dependent variable.

A binary classifier can obtain the following types of predictions

	Ground Truth (Test data)		
Model (Test data)	**Positive**	**Negative**	**Total**
Positive	TP	FP	TP+FP
Negative	FN	TN	FN+TN
Total	TP+FN	FP+TN	TOTAL

The following terms are used to describe the predictions of a classification algorithm:

- True positives (TP): Examples that belong to the class and are correctly classified by the algorithm.
- True negatives (TN): Examples that do not belong to the class and are correctly classified as not belonging to it.
- False positives (FP): Examples are incorrectly classified as belonging to the class when they do not belong.
- False negatives (FN): Examples incorrectly classified as not belonging to the class when they actually belong to it.

Based on these types of predictions, several evaluation measures can be defined, including **accuracy (Acc)**, **precision (Pr)**, **recall (Rc)**, and **F1-score**.

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

The *Acc* may be inappropriate when dealing with unbalanced data in binary classification. For instance, consider a scenario with 9990 training examples in the positive class and only 10 in the negative class. In such cases, relying solely on *Acc* as a performance metric could be misleading. For instance, if a classifier always predicts the positive class, it would achieve a high accuracy of 99.9%. However, this would not necessarily mean that the classifier effectively captures patterns in the training data or makes accurate predictions for the negative class. Therefore, alternative metrics such as precision, recall, or F1-score may be more informative in evaluating the performance of classifiers on unbalanced data.

$$Pr = \frac{TP}{TP+FP} \qquad Rc = \frac{TP}{TP+FN} \qquad F_1 = 2\frac{Pr*Rc}{Pr+Rc}$$

Conversely to the *Acc,* the *F1-score* can be calculated by class or overall for the test set. Two ways to compute the overall F1-score are **micro-averaged F1 (*F1-micro*)** or **macro-averaged F1 (*F1-macro*)**.

Micro-averaged F1 is calculated by first computing the overall true positives (TP), false positives (FP), and false negatives (FN) across all classes and then computing the F1 score using these overall values. This approach gives equal importance to each prediction regardless of the class it belongs to. Micro-averaged F1 is useful when you have balanced classes and care more about overall performance than individual classes' performance.

On the other hand, macro-averaged F1 is calculated by computing the F1 score for each class separately and then taking the average across all classes. This approach gives equal importance to each class, regardless of the number of data points in it. Macro-averaged F1 is useful when you want to independently evaluate the model's performance for each class.

The idea of regression metrics is to quantify how well a regression model fits the data. There are several commonly used evaluation metrics, including:

Mean Squared Error (*MSE*): This is the average of the squared differences between the predicted value $\hat{y}_i$ and actual values of the dependent variable y_i. It measures how far the predicted values are from the actual values, with higher values indicating poorer performance.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (*RMSE*): This is the square root of the MSE and is often used to make the error metric easier to interpret.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$$

Mean Absolute Error (*MAE*): This is the average of the absolute differences between the predicted and actual values of the dependent variable. It measures how far the predicted values are from the actual values, but unlike MSE, it doesn't penalize large errors.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

HUHU Evaluation Metrics

Subtask 1 (HUrtful HUmour Detection)
It consists in determining whether a prejudicial tweet is intended to cause humor. Systems will have to distinguish between tweets that using humor express prejudice and tweets that express prejudice without using humor. For that, systems will be evaluated employing the F1-score

achieved in the positive class. Below is an example of how this measure can be calculated using the Sklearn library. It is important to remember that this measure must be calculated for the positive class.

EXAMPLE 1

```
from sklearn.metrics import f1_score, accuracy_score
y_true = [0, 1, 1, 0, 1, 1, 0]
y_pred = [0, 0, 1, 0, 0, 0, 1]
prejudice=f1_score(y_true, y_pred, pos_label=0)
humorAndPrejudice=f1_score(y_true, y_pred, pos_label=1)
a=accuracy_score(y_true, y_pred)
print("F1-Score for the prejudice", prejudice)
print("F1-Score for the  by means of humor",
humorAndPrejudice)
print("Classifier's Accuracy: ", a)
```

Subtask 2a (Prejudice Target Detection)

In this subtask, systems are asked to identify the targeted groups on each tweet as a multilabel classification task. The metric employed will be F1-score macro-averaged.

EXAMPLE 2

```
from sklearn.metrics import f1_score, accuracy_score
y_true = [1, 2, 1, 3, 3, 1, 2, 2, 1]
y_pred = [2, 1, 1, 3, 2, 1, 2, 2, 3]
macro=f1_score(y_true, y_pred, average="macro")
micro=f1_score(y_true, y_pred, average="micro")
a=accuracy_score(y_true, y_pred)
print("F1-Macro", macro)
print("F1-Micro", micro)
print("Classifier's Accuracy: ", a)
```

Subtask 2b (Degree of Prejudice Prediction)
It consists of predicting on a continuous scale from 1 to 5 to evaluate how prejudicial the message is on average among minority groups. Systems will be evaluated employing the Root Mean Squared Error.

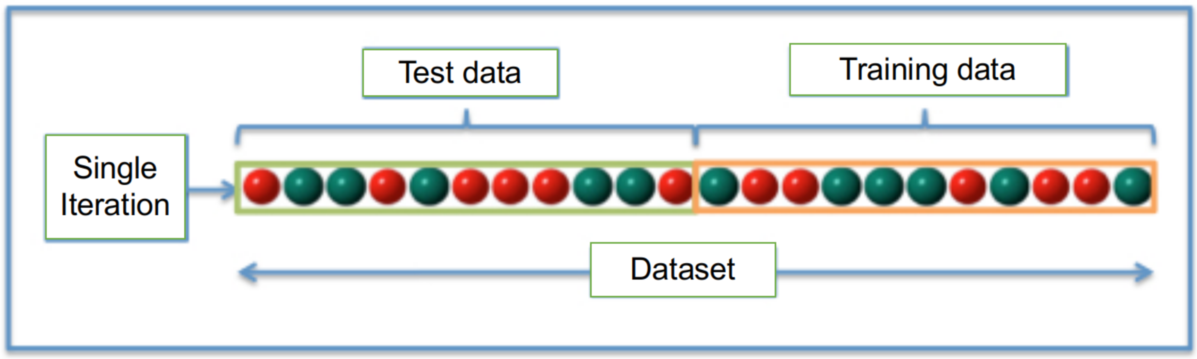
EXAMPLE 3

```
from sklearn.metrics import mean_absolute_error,
mean_squared_error
y_true = [1, 2, 1, 3, 4, 5, 2, 5, 1]
y_pred = [1, 1, 2, 2, 4, 4, 1, 1, 5]
MAE=f1_score(y_true, y_pred, average="macro")
MSE=mean_squared_error(y_true, y_pred, squared=True)
RMSE=mean_squared_error(y_true, y_pred, squared=False)
a=accuracy_score(y_true, y_pred)
print("MAE", MAE)
print("MSE", MSE)
print("RMSE", RMSE)
```

VALIDATION SCHEMAS

One commonly used approach to train and validate a classifier involves using two distinct subsets of data, one for training and one for testing. However, in some cases, the available data may not be separated into these two sets, making it necessary to use techniques for partitioning the data set. The literature shows four strategies commonly employed to accomplish this goal. They are presented below.

Holdout
It consists of randomly dividing the data set into two sets: a training set (⅔) and a test set (⅓), although (½) can be used for training and (½) for evaluation.



Advantages:

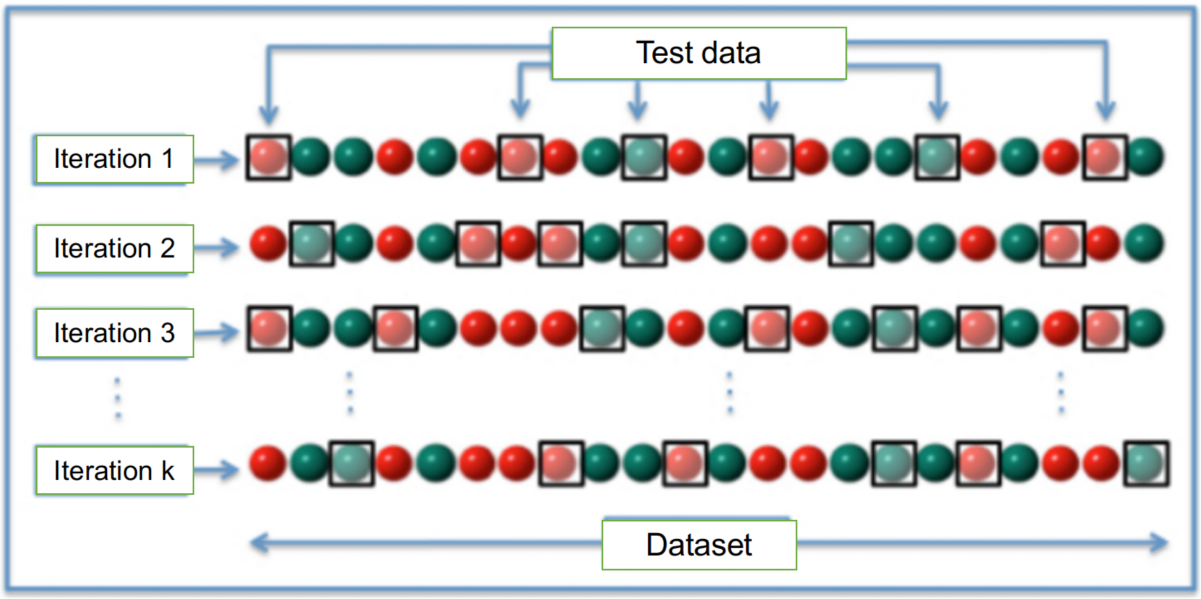
1. It is very fast and useful in the early stages of classifier development.
2. The error is validated with a statistically independent set to those used for parameter adjustment (statistical independence).

Disadvantages

1. Highly dependent on the division to be performed.
2. It obtains relatively pessimistic estimates.
3. A classifier trained with the full dataset will perform better.

Random Subsampling

This is a variation of the holdout method. Here the holdout method is repeated k times and the evaluation measure is averaged.



Advantage:

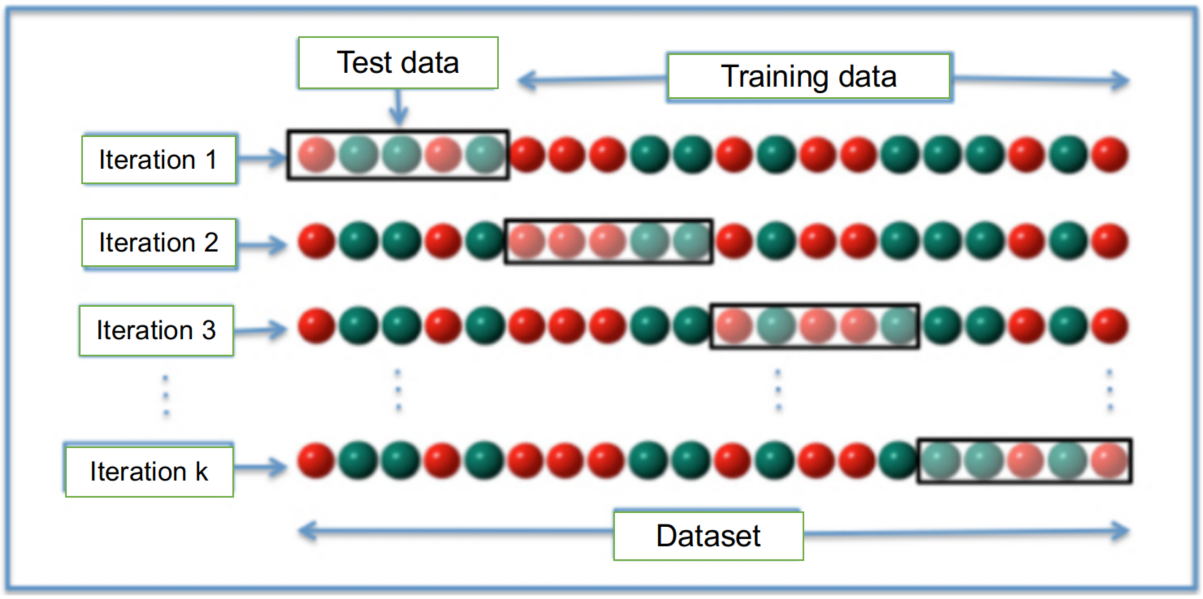
1. The results do not depend on a single division of the dataset.

Disadvantages:

1. High computational cost, due to the number of training sets generated.
2. The randomness in selection causes some data not to be evaluated and others to be evaluated more than once.
3. Overlapping training and test datasets are obtained.

K-fold Cross-Validation

The dataset is randomly divided into k disjoint subsets S1, S2, ..., Sk of approximately the same size. The training and testing are performed k times. At each iteration i, Si is taken as the test dataset and the union of the Sj, for all j distinct from i, as the training dataset. The evaluation metric for each iteration is averaged.



Advantages:

1. All data points are evaluated.
2. It is not dependent on a single division.

Disadvantages:

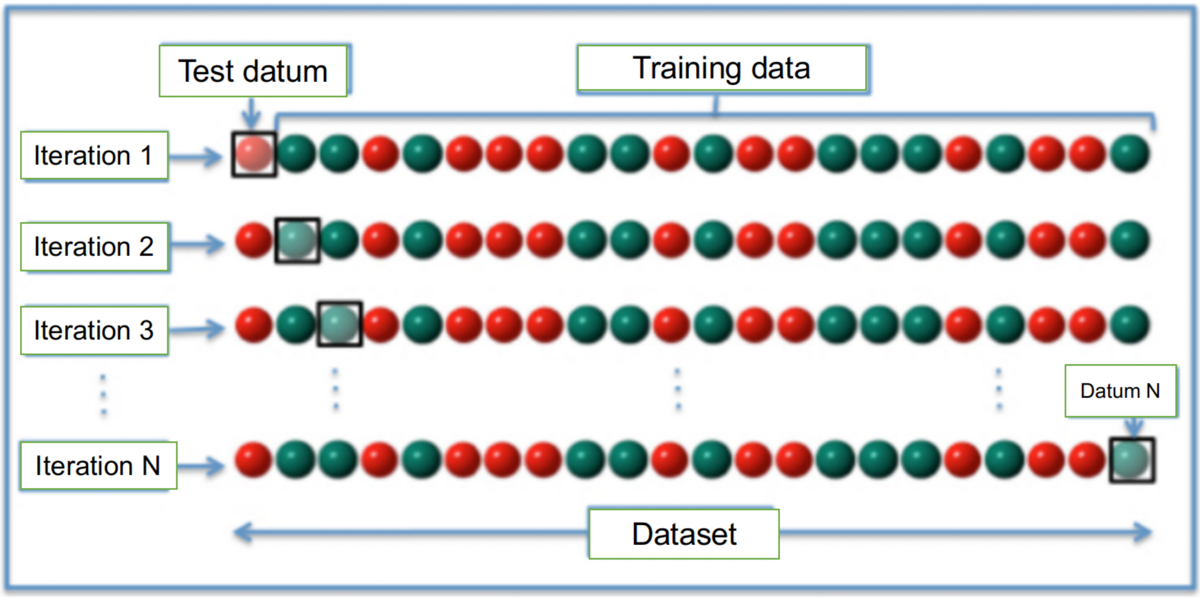
1. It is computationally expensive.
2. Depending on the value of k, the results may vary somewhat.
3. It does not take into consideration the distribution of classes in the generated partitions.

Stratified Sampling (Stratified K-fold cross-validation)

This is a variation of the k-fold cross-validation method. Here each fold is constructed in such a way that the initial class distribution is maintained in each partition.

Leave one-out

This is a variation of k-fold cross-validation. Here k=N, where N is the number of data points and a single element is always left as the test set.



Advantage:

1. More information is available for the construction of the classifier.
2. Applicable for small datasets.

Disadvantage:

1. It is very slow because a large number of iterations must be performed.

Calculate metrics with cross-validation

EXAMPLE 4

```
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from sklearn import svm
X_train = np.array([[0.1, 0.1], [0., 0.], [1., 1.], [-1.,
-1.], [2., 2.], [0.5, 0.5], [1.5, 1.5], [-1.5, -1.5], [2.5,
2.5], [2.56, 2.53]])
y_train = np.array([0, 1, 0, 0, 1, 1, 0, 1, 0, 0])
```

```
clf = SVC(kernel='linear', C=1, random_state=42)
scores = cross_val_score(clf, X_train, y_train, cv=5,
scoring='f1_macro')
scores
```

K-fold: Generate indexes that can be used to obtain splits of datasets

EXAMPLE 5

```
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import KFold
from sklearn import metrics
X_train = np.array([[0.1, 0.1], [0., 0.], [1., 1.], [-1.,
-1.], [2., 2.], [0.5, 0.5], [1.5, 1.5], [-1.5, -1.5], [2.5,
2.5], [2.56, 2.53]])
y_train = np.array([0, 1, 0, 0, 1, 1, 0, 1, 0, 0])
kf = KFold(n_splits=3)
for train, test in kf.split(X_train):
    X_train_i, X_test_i, y_train_i, y_test_i = X_train[train],
X_train[test], y_train[train], y_train[test]
```

Stratified K-fold: Requires the variable containing the manual annotations to be passed in order to distribute the data.

EXAMPLE 6

```
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from sklearn import metrics
X_train = np.array([[0.1, 0.1], [0., 0.], [1., 1.], [-1.,
-1.], [2., 2.], [0.5, 0.5], [1.5, 1.5], [-1.5, -1.5], [2.5,
2.5], [2.56, 2.53]])
y_train = np.array([0, 1, 0, 0, 1, 1, 0, 1, 0, 0])
kf = StratifiedKFold(n_splits=3)
```

```
for train, test in kf.split(X_train,y_train):
    X_train_i, X_test_i, y_train_i, y_test_i = X_train[train],
X_train[test], y_train[train], y_train[test]
```

HYPERPARAMETERS SETTING

Hyperparameters are parameters that are not learned directly within the classifier. It is possible and recommended to search the hyperparameter space for better results. Two generic approaches can be used for parameter search:

Grid Search: exhaustively considers all parameter combinations.

EXAMPLE 7

```
from sklearn import svm
from sklearn.model_selection import GridSearchCV
clf = GridSearchCV(estimator=svm.SVC(), param_grid={'C':
[2,3,4,5,10], 'kernel': ('linear', 'rbf')})
clf.fit(X_train, y_train)
#Obtaining the best hyper-parameters
print(clf.best_params_)
#Obtain the best classifier
print(clf.best_estimator_)
```

Randomized Search: samples a parameter space with a specific distribution.

EXAMPLE 8

```
#- uniform: uniform distribution in: [loc, loc + scale]
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform
logistic = LogisticRegression()
distributions = dict(C=uniform(loc=0, scale=4), penalty=['l2',
'l1'])
```

```
clf = RandomizedSearchCV(logistic, distributions)
clf.fit(X_train, y_train)
print(clf.best_params_)
print(clf.best_estimator_)
```

GridSearchCV and RandomSearchCV have successive equivalents for halving the search: HalvingGridSearchCV and HalvingRandomSearchCV.

Note: A small subset of these parameters may have a large impact on the classifier performance and others can be left with their default values.

ACTIVITY

Adjust the parameters of the proposed models for the HUHU subtasks using the F1 evaluation measure for classification tasks and RMSE for the regression task, and using a cross-validation scheme. Deliver code and report with explanation of the strategy used before the next session.

The Google collaborative environment (Google Colab) can be used to implement the machine learning models: https://colab.research.google.com/notebooks/welcome.ipynb#scrollTo=5fCEDCU_qrC0