24 November 2020

TD – From data preparation to model selection and evaluation of the classification performance

Introduction

This directed work is a tutorial that will lead you though the process of managing a machine learning problem from the data preparation to the evaluation of the chosen classifier.
We will go through the following steps:

1. Load the dataset
2. Explore and visualize the dataset
3. Balance an unbalanced dataset (data augmentation)
4. Data preparation:
    a. Split the data in X (the features matrix) and Y (the label matrix)
    b. Split the dataset in Training and Test set (we will create and use the Validation set directly in step 6)
5. Rescaling/standardization (and how to deal with outliers!)
6. k-fold cross-validation to search the "best" hyperparameters (using grid search)
7. Detect under/overfitting using scikit-learn function "learning_curve"
8. Compute the confusion matrix on the test set
9. Compute accuracy, precision, recall, f1_score

For each of the previous steps, we will provide some links and tips to go deeper in your understanding.

NOTA BENE: this tutorial presents a problem of multiclass classification using Random Forest but it could be adapted to any classifier (even deep learning) or a regression problem with only few changes.

Table of Contents

1 Loading the dataset

In this practical work, you will use the **Iris dataset** (for more info, check the related Wikipedia page: https://en.wikipedia.org/wiki/Iris_flower_data_set)
This dataset is considered the "hello world" dataset in machine learning and statistics. The dataset contains 150 observations of iris flowers. There are four columns of measurements of the flowers in centimeters. The fifth column is the species of the flower observed (the classes). All observed flowers belong to one of three species: *Iris-setosa*, *Iris-versicolor*, and *Iris-virginica*.

In this step, we are going to: first, import the needed modules; second, we load the Iris dataset from scikit learn. Optionally you can load it from a URL (you need an internet connection) or from a CSV file.

```
# 0. import libraries
import pandas as pd
from pandas.plotting import scatter_matrix

import numpy as np

import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import GridSearchCV

from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import learning_curve

import itertools
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, confusion_matrix
```

```
# 1. Load dataset
# option 1 - load directily from scikit-learn and copy it to a pandas dataframe
# (of course, this is a viable option only if the dataset is available in this package)
from sklearn.datasets import load_iris
iris = load_iris(as_frame=True)
dataset = iris['frame']

# option 2  -  load the dataset from a file
# - the internet connection is no required
# - you will probably need to rename de columns

# dataset = pd.read_csv('path_to_the_dataset_folder/example_dataset.csv')

# option 3  - load via url
# url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
# names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
# dataset = pd.read_csv(url, names=names)
```

2 Explore and visualize the dataset

(*The code presented in this section is inspired by: https://machinelearningmastery.com/machine-learning-in-python-step-by-step/. Have a look to the previous link for an interesting step by step machine learning tutorial*)

Visualizing the dataset or a summary of it should be the first step before even starting to think about machine learning algorithms. This step allows understanding basics information such as the number of samples, the number of classes, the number of features, how the samples are distributed among the classes, etc. From this basic information you can start thinking ahead: is the dataset balanced? Are there probable outliers? Are there missing values?

```
# 2. Explore dataset (source check https://machinelearningmastery.com/machine-learning-in-python-step-by-step/)

# 2.1 visualize shape
print(dataset.shape)

# 2.2 Get a peek of the data
print(dataset.head(10))

# 2.3 Get Statistical Summary
print(dataset.describe())

# 2.4 Visualize Class Distribution (here called "target")
print(dataset.groupby('target').size())

# 2.5 plot data and relationships among variables
# 2.5.1 box and whisker plots
dataset.plot(kind='box', subplots=True, layout=(3, 2), sharex=False, sharey=False)
plt.show()

# 2.5.2  histograms
dataset.hist()
plt.show()

# 2.5.3  scatter plot matrix
scatter_matrix(dataset)
plt.show()
```

Question 1: now that you had a look to the data, you can fill the following information:

- Number of samples:
- Number of classes:
- Number of features:
- Are there probable outliers? (yes/no & why)

3 Unbalanced dataset

In point 2.4, we visualized how the data are distributed in the different classes. In this way, we can detect if a dataset is unbalanced.
Question 2: is the Iris dataset balanced? (yes/no & why)

[Optional reading]
The possibility to balance an unbalanced dataset depends on the data that we are using. For instance:

- If the dataset is large, one possibility is to simply exclude some samples
- If the dataset is small, it is possible to change the loss function (i.e., the function that we have to minimize during the training) in order to penalize more errors on the less represented classes
- Avoid using "accuracy" as the unique metric for the algorithm performance
- You can use data augmentation on the missing data (typical if the dataset is composed by pictures)

The following code provide a simple example of data augmentation using Keras (a library typically used for deep learning). Even if **it is not relevant for the Iris dataset**, it could be interesting for many other problems and it is provided as a reference (https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html).

```
## 3. Data augmentation (in order to use the code below, you need Keras installed in your venv, (https://keras.io/#installation))

# from keras.preprocessing.image import ImageDataGenerator
# datagen = ImageDataGenerator(
#         rotation_range=40,
#         width_shift_range=0.2,
#         height_shift_range=0.2,
#         rescale=1./255,
#         shear_range=0.2,
#         zoom_range=0.2,
#         horizontal_flip=True,
#         fill_mode='nearest')
```

4 Data preparation

Before moving forward, it is necessary to manipulate the dataset in order to prepare the data for the *cross-validation* and the *classification* using a supervised approach. This is done in two steps: we split the data in X (the features matrix) and Y (the label matrix); then, we split the dataset in Training and Test set. The Test set will be used again only at the end of the workflow to get an unbiased evaluation of the chosen classifier.
NOTA BENE: we do not talk yet of the "Validation set". The Validation set (used to tune the hyper-parameters) will be managed directly in later steps.

```
# 4. Data preparation
# 4.1 Split the dataset in features and labels
# X = dataset.loc[:, 'sepal-length (cm)':'petal-width (cm)']
X = dataset.iloc[:, 0:3]
Y = dataset.loc[:, 'target']

# 4.2 Split into train and test sets
test_size = 0.20
seed = 46
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
```

Question 3: In your opinion, why do we fix the seed?

5 Rescaling/standardization

During Step 2, while visualizing the data, we could notice that all the features have a quite similar range. This means that, for this dataset, a rescaling (or normalization) is not really necessary (you could test the rest of the code without rescaling the data to check the impact on performance).
On the other hand, rescaling rarely impact negatively the performance of an algorithm and, usually, it is a safe choice to do it anyway. However, this is true only if **there are not outliers** among our data (in Step 2.5.1 the box and whisker plots showed some potential outliers).
In the following example, we use a scaler called RobustScaler to remap all the data in our dataset in a small interval around the 0 value. This scaler is designed to be resilient to outliers.
For more information about the RobustScaler, check http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html#sklearn.preprocessing.RobustScaler

```
# 5. Rescaling/normalization
# we suspect the presence of outliers, so we use the RobustScaler

# 5.1 calculate the scaler using only the training set
scaler_X = RobustScaler()
# 5.2 apply the scaler on the training set
X_train_scaled = scaler_X.fit_transform(X_train)

# 5.3 apply the scaler (calculated on the training set) also to the test set
X_test_scaled = scaler_X.transform(X_test)

## 5.4 in case of a Regression problem for timeseries, you may need to rescale also the Y
## scaler_y = RobustScaler()
## y_train_scaled = scaler_y.fit_transform(Y_train)
## y_test_scaled = scaler_y.transform(Y_test)

# 5.5 [optional] check the new values after the rescaling
print(X_train_scaled.shape)
print(X_train_scaled[:10,:])
```

Question 4: Select one feature and plot the data before and after rescaling the dataset. Modify the previous code to use the "QuantileTransformer" and "MinMaxScaler" and plot again the data (hint: http://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#sphx-glr-auto-examples-preprocessing-plot-all-scaling-py). Are these methods (QuantileTransformer, MinMaxScaler) a good alternative? Why?
In your opinion, why in a regression problem should you rescale also the Y?

6 k-fold cross-validation to search the "best" hyperparameters (using grid search)

In this step, we combine k-fold cross-validation and grid search (http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) to look for the optimal hyper parameters of a Random Forest classifier.
With the k-fold cross-validation we split iteratively the training data set in training set and validation set and, at each iteration, we look for the best parameters. If not specified differently, the function `grid_search.fit()` automatically refit (train) the classifier using the best parameters it found.
Finally, we provide a simple script to visualize which parameter provided the best results.

```
# 6. k-fold cross-validation + grid search

# 6.1 choose your classifier (in this example, we use Random Forest)
classifier = RandomForestClassifier()

# 6.2 create the grid of parameters that you want to explore
# of course, the list of parameters depends on the chosen classifier
param_grid = {"max_depth": [2, None],
              "n_estimators": [5, 20, 50],
              "min_samples_split": [2, 3, 5],
              "min_samples_leaf": [1, 5],
              "bootstrap": [True, False]}

# 6.3 use k-fold cross-validation to select the best set of parameters
# in this example (k=5), at each iteration, 4/5 of data will be used for the training set
# and 1/5 of data for the VALIDATION set

grid_search = GridSearchCV(classifier, param_grid, cv=5)
grid_search.fit(X_train_scaled, Y_train)

results = grid_search.cv_results_


#--------------------#
# 6.4 check the best scores for the best parameters
n_top = 5
for i in range(1, n_top + 1):
    candidates = np.flatnonzero(results['rank_test_score'] == i)
    for candidate in candidates:
        print("Model with rank: {0}".format(i))
        print("Mean validation score: {0:.6f} (std: {1:.3f})".format(
              results['mean_test_score'][candidate],
              results['std_test_score'][candidate]))
        print("Parameters: {0}".format(results['params'][candidate]))
        print("")
```

Question 5: let's review together the Random Forest algorithm. Explain the role on the algorithm of the following parameters:

- max_dept:

- n_estimators:

- min_samples_split:

- min_samples_leaf:

7 Detect under/overfitting using scikit-learn function "learning_curve"

While the goal of the previous step was to find the best hyper parameters, in this step, we want to be sure that our model is not underfitting or overfitting the data. This could be done by drawing a learning curve of the model using the training and the validation set.

NOTA BENE: during the lecture, we compared the error of the model on the training and on the validation set (i.e., a higher value corresponds to worse performance). The function that we used in this section return the accuracy (i.e., a higher value corresponds to better performance). However, the analysis remains fairly similar.

```
# 7.0 detect under/overfitting using learning_curve:
# 7.1 create a classifier (for instance, with the best parameters found in step 6)

classifier = RandomForestClassifier(bootstrap=True, max_depth=1, min_samples_leaf=1,
min_samples_split=2, n_estimators=5)

# 7.2 define a k-fold classifier object
cv = StratifiedKFold(n_splits=10, random_state=None, shuffle=True)

# 7.3 calculate a learning curve
train_sizes, train_scores, test_scores = learning_curve(classifier, X_train_scaled, Y_train, cv=cv)



# 7.4 print the learning courve (source: http://scikit-
learn.org/stable/auto_examples/model_selection/plot_learning_curve.html )
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)
plt.grid()

plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.1,
                 color="r")
plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.1, color="g")
plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
         label="Training score")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
         label="Cross-validation score")

plt.legend(loc="best")
plt.show()
```

Question 6: In your opinion, is the system overfitting, underfitting or doing well? What do you think and why?

8 Compute the confusion matrix using the test set

It is time to exhume the Test set put aside in Step 4 and finally provide the final evaluation of the system using by computing the confusion matrix. The code below can be set to provide a normalized version of the confusion matrix. It is also completed by a script that draws a better-looking version of the matrix (source: http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py). Try it out!
NOTA BENE: We are evaluating the performance of the model with the parameters set in step 7.1 (i.e., not the best parameters that we found in Step 6).

```
# 8. compute the confusion matrix on the test set
# warning!! in case of regression you should invert the rescaling. e.g.:
# yhat = regressor.predict(X_test_scaled)
# y_pred = scaler_y.inverse_transform(yhat)

# 8.1 refit the classifier on the whole training set and compute the prediction on test set
classifier.fit(X_train_scaled, Y_train)
y_pred = classifier.predict(X_test_scaled)
# 8.2 compute and print the confusion matrix
cnf_matrix = confusion_matrix(Y_test, y_pred)

print("on the x (horizontal) axis: Predicted label")
print("on the y (vertical) axis: True label")

normalize = False
if normalize:
    cnf_matrix = cnf_matrix.astype('float') / cnf_matrix.sum(axis=1)[:, np.newaxis]
    print("Normalized confusion matrix")
else:
    print('Confusion matrix, without normalization')

print(cnf_matrix)

# 8.3 [optional] advanced print for the confusion matrix
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    #print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
```

```
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

plt.figure()
class_names = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=False,
                      title='Normalized confusion matrix')
plt.show()
```

Question 7: explain the line `y_pred = classifier.predict(X_test_scaled);` Why we did not use `y_pred = classifier.predict(X_train_scaled)`?

9 Compute accuracy, precision, recall, f1_score

The Confusion matrix is one of the best methods to obtain an overview of the performance of an algorithm. However, other metrics are often used because they facilitate the comparison of results. In the following few lines, we compute the *accuracy*, *precision*, *recall*, and *f1_score*. Since this is a multiclass problem these metrics are calculated for each class. In a second step, we use the attribute "average" to get global scores. In this case, we set average="macro" but many other options are available; check for instance: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html.

```
# 9. accuracy and beyond
# 9.1 compute accuracy, precision, recall, f1_score (per class)
print("Metrics per classes")
print("accuracy_score: " + str(accuracy_score(Y_test, y_pred)))
print("precision_score: " + str(precision_score(Y_test, y_pred, average=None)))
print("recall_score: " + str(recall_score(Y_test, y_pred, average=None)))
print("f1_score: " + str(f1_score(Y_test, y_pred, average=None)))

# 9.2 Compute accuracy, precision, recall, f1_score (in average - only Multiclass!)
print("Metrics (average)")
print("accuracy_score: " + str(accuracy_score(Y_test, y_pred)))
print("precision_score: " + str(precision_score(Y_test, y_pred, average="macro")))
print("recall_score: " + str(recall_score(Y_test, y_pred, average="macro")))
print("f1_score: " + str(f1_score(Y_test, y_pred, average="macro")))
```

Question 8: Go back to Step 7 and substitute the parameters used by the classifiers with the best parameters you found in Step 6. Do not forget to re-train the classifier (using `classifier.fit()`) on the training set. Is the performance of the Classifier improved?

10 Optional exercises – Going deeper

Now that you have all the complete code, hereafter you will find some additional exercises that you can try out to get a better understanding of the process:

- Modify the code to compare different classifiers (hint: *see points 5.3 and 5.4 in https://machinelearningmastery.com/machine-learning-in-python-step-by-step/*)
- Modify the code to make it work for a regression problem (for instance, you can use the bike prediction dataset used last week)