Assignment 2 - Scikit-Learn

Davide Pedranz (189295)
davide.pedranz@studenti.unitn.it

***Abstract*—Scikit-Learn is a simple and powerful library to do Machine Learning in Python. In this assignment, we will use Scikit-Learn to compare the performances of the Naive Bayes, Support Vector Machine and Random Forest learning algorithms on a binary classification problem.**

I. Introduction

Machine Learning techniques allow to extract knowledge from training data and use it to classify or predict new examples. There are different learning algorithms for different problems. In this assignment, we will focus on the classification problem. In particular, we will run the following algorithms and compare their performances:

- Naive Bayes,
- Support Vector Machine,
- Random Forest.

Most algorithm require to set some parameters in advance, before the learning phase. These parameters are called hyper-parameters. It is usually tricky to choose this parameters in advance, so a common way to overcome this problem is to use a part of the training data to tune their values. We used the process called k-fold cross validation, which is described in Section III.

We use the Python library `Scikit-Learn`[1]. Scikit-Learn implement most of the commonly used Machine Learning algorithms and provides utilities to generate learning problems, split the data and measure the performances.

II. Learning problem

We focus on a binary classification problem. The sample data are generated using the `make_classification` function in the `scikit.datasets` package. We generated a problem with 5000 examples, each of them with 15 features. Only 10 of them actually influence the class, the others are either a linear combination of the other, repeated or just noise.

III. k-fold cross validation

Two common problems of the learning algorithms are underfitting and overfitting. The former appears when the model is too generic to accurately fit the data, the latter means that the model is too complex that perfectly fit training data but fail to generalize on new examples. Most algorithm has some hyperparameter that allow to specify the complexity of the model to use or the regularization term, tuning the trade-off between underfitting and overfitting.

Hyperparameters make the learning algorithms very flexible but also difficult to use, since their choice is not trivial. An obvious solution is to use a part of the available training examples to tune those parameters. The training set is split in two parts, the first one used for the training, while the second one for the validation. In other works, a different classifier is trained for each combination of hyperparameters, then the one performing better on the validation set is selected. This approach has the downside of "wasting" a significant part of the available examples.

The k-fold cross validation technique allow to obtain a similar result without wast of training examples. The training set is randomly partitioned in k equal sized sets. For each combination of hyperparameters, k different learners are trained, each time taking as training set the union of all partitions except the k^{th} one. The k^{th} partition is used to test the performances of the k^{th} learner. The performances of the k learners are then averaged and the best one is selected. Finally, the best learner is trained on the entire training set using its hyperparameters. The real performances are measured on the test set, which was never used before, neither for the learning nor for the hyperparameters choice. This guarantees that the choice of the hyperparameters is not baised.

The same technique can be used in a similar way to get a higher confidence when comparing different algorithms. This time, the entire dataset is partitioned in k subset. For each k, a learner for each algorithm is trained on $k-1$ folds and tested on the last one. The performances are collected for each k and then averaged for each algorithm. The means are used to compare the learning algorithms, so that the comparison does not dependent on a single random split of the data. In fact, an algorithm could perform well on a certain split, whereas a second one could perform much better on a different one.

The described tecniques is commonly used in many Machine Learning problems where the available examples are limited.

IV. Evaluation Scores

An important phase of the learning process is the evaluation. After training an algorithm, we want to measure how good it is performing. These measures are used for both the selection of the best learning algorithm on the given problem and the choice of its hyperparameters. Commonly used ones are:

- Accuracy,
- F1-score,
- AUC ROC.

The accuracy score focuses on the correct predicted samples, whereas F1 and AUC ROC scores take into account both the correct and wrong predicted ones. They can be combined to obtain more robust measures.

[1] http://scikit-learn.org/

Table I
PERFORMANCES OF NAIVE BAYES

k-fold	Accuracy	F1	AUC ROC
1	0.786	0.802	0.787
2	0.744	0.758	0.744
3	0.764	0.773	0.765
4	0.764	0.785	0.761
5	0.764	0.785	0.763
6	0.778	0.791	0.780
7	0.748	0.771	0.745
8	0.738	0.747	0.746
9	0.742	0.774	0.736
10	0.734	0.750	0.738
avg	0.756	0.774	0.756

Table II
PERFORMANCES OF SVM

k-fold	Accuracy	F1	AUC ROC
1	0.970	0.969	0.970
2	0.972	0.972	0.972
3	0.984	0.984	0.984
4	0.966	0.967	0.966
5	0.968	0.969	0.968
6	0.980	0.979	0.980
7	0.970	0.971	0.970
8	0.972	0.970	0.972
9	0.972	0.974	0.972
10	0.966	0.965	0.966
avg	0.972	0.972	0.972

All these metrics are provided by `Scikit-Learn` in the `sklearn.metrics` package.

V. NAIVE BAYES

The Naive Bayes classification algorithm tries to model the probabilistic structure of the dataset under the assumption that every feature is indipendent on each other given the class. The predicted class is the one that maximizes the probability of having the given configuration.

For this specific classification problem quite bad. In fact, all scores are around 0.75. The details can be found in Table I. The bad performances are probably due to the presence of redundant and repeated features, which break the conditional indipendence assumption of the algorithm.

VI. SUPPORT VECTOR MACHINE

The Support Vector Machine (SVM) algorithm tries to find the hyperplane which linearly separates the two classes with the highest margin. In the simpler implementation, it uses the standard dot product to measure the similarity between two points. It is also possibile to use a different function, denominated kernel, as similarity function. This allows to tranform the features space and perform a linear separation in the new space. The result is a non linear separation in the original space. with a shape that depends on the used kernel. In this case, we used the popular Gaussian radial basis function kernel.

Table III
PERFORMANCES OF RANDOM FOREST

k-fold	Accuracy	F1	AUC ROC
1	0.936	0.936	0.936
2	0.940	0.941	0.940
3	0.958	0.958	0.959
4	0.944	0.947	0.944
5	0.948	0.950	0.948
6	0.968	0.967	0.968
7	0.938	0.942	0.937
8	0.950	0.947	0.952
9	0.950	0.953	0.949
10	0.946	0.944	0.946
avg	0.948	0.948	0.948

Table IV
PERFORMANCES' SUMMARY

Algorithm	Accuracy	F1	AUC ROC
Naive Bayes	0.756	0.774	0.756
Support Vector Machine	0.972	0.972	0.972
Random Forest	0.948	0.948	0.948

The performances of this algorithm are reported in Table II. Both the accuracy, F1 and AUC ROC scores are very high: 0.972. SVM was able to discriminate very well the samples and performed much better than Naive Bayes.

VII. RANDOM FOREST

Random Forest is an enseble learning method that combines multiple Decision Trees. The aim is to reduce the variance of each tree and thus better generealize on new examples. A specified number of Decision Trees are learned in a sequence and combined togheter. The resulting prediction of the Random Forest is simply the average of the prediction of all trees.

The performances of Random Forest are reported in table Table III. It scored on average 0.948 on each indicator, perfoming slightly worse then SVM but much better than Naive Bayes.

VIII. CONCLUSION

For this classification problem, SVM was the best choice, performing better then Naive Bayes and Random Forest. Random Forest scored second, performing only slighly worse than SVM. Naive Bayes performed worst, with a significant difference from both the other two algorithms. The performances of the three learning algorithms are summariezed in Table IV.