Chapter 1

A Tour of Machine Learning Classifiers Using Scikit-Learn

1.1 Introduction

The goal of this chapter is to use Scikit-Learn to impelement some classifiers in order to learn how to use them, his advantages, disadvantages and different use scenarios.

Specifically, we will take a look of 4 popular machine learning models commonly used in academia and in industry. In addition, we will take a look at the Scikit-Learn library, which offers a user-friendly and consistent interface for using those algorithms efficiently and productively.

1.1.1 Choosing a Classification Algorithm

Each algorithm has its own quirks and relies on certain assumptions. no single classifier works best across all possible scenarios (The Lack of A Priori Distinctions Between Learning Algorithms, Wolpert, David H, Neural Computation 8.7 (1996): 1341-1390).

Observation 1.1.1 (Comparasion)
In practice, it is always recomended to compare the behaviour of different algorithms in order to find the best model suitable for a particular problem; these may differ in the number of features or examples, the amount of noise in a dataset, and whether the classes are linearly separable.

Idea 1.1.1
The performance of a classifier relies upon the data that is available for learning. The five main steps that are involved in training a supervised machine learning algorithm can be summarized as follows:

1. Selecting features and collecting labeled training examples
2. Choosing a performance metric
3. Choosing a learning algorithm and training a model
4. Evaluating the performance of the model
5. Changing the settings of the algorithm and tuning the model.

We will mainly focus on the main concepts of the different algorithms in this chapter and revisit topics such as feature selection and preprocessing, performance metrics, and hyperparameter tuning for more detailed discussions later in the book.

1.1.2 First Steps with scikit-learn Training a Perceptron

Before we learn about two related learning algorithms: the perceptron and adaline, both implemented in Python using NumPy and other libraries by ourselves.

Now we will take a look at the **scikit-learn API**.

Cristo Daniel Alvarado ESFM

Cristo Daniel Alvarado ESFM

Observation 1.1.2
One of the advantages of using scikit-learn is that combines a user-friendly and consistent interface with a highly optimized implementation of several classification algorithms. Also, this library offers a not only a large variety of learning algorithms, but also many convenient functions to preprocess data and to fine-tune and evaluate our models.

1.1.3 Training a Model Using Scikit-Learn

To get started with the scikit-learn library, we will train a perceptron model similar to the one that we implemented in Chapter 2. For simplicity, we will use the already familiar Iris dataset throughout the following sections.

Observation 1.1.3 (**Iris Dataset and Its Uses**)
Conveniently, the Iris dataset is already available via scikit-learn, since it is a simple yet popular dataset that is frequently used for testing and experimenting with algorithms. Similar to the previous chapter, we will only use two features from the Iris dataset for visualization purposes.

We will assign the petal length and petal width of the 150 flower examples to the feature matrix, `X`, and the corresponding class labels of the flower species to the vector array, `y`:

```
Class labels: [0 1 2]
Class labels: [0 1 2]
```

Code 1.1: Class Labels of Matrix `X`

The `np.unique(y)` function returned the three unique class labels stored in `iris.target`, and as we can see, the Iris flower class names, `Iris-setosa`, `Iris-versicolor`, and `Iris-virginica`, are already stored as integers (here: `0`, `1`, `2`).

Observation 1.1.4 (**Scikit-Learn with Class Labels and String Formats**)
Although many scikit-learn functions and class methods also work with *class labels in string format, using integer labels is a recommended approach to avoid technical glitches and improve computational performance due to a smaller memory footprint*; furthermore, *encoding class labels as integers is a common convention among most machine learning libraries.*

To evaluate how well a trained model performs on unseen data, we will *further split the dataset into separate training and test datasets.*

Idea 1.1.2 (**More Info About Best Practices around Model Evaluation**)
In Chapter 6, *Learning Best Practices for Model Evaluation and Hyperparameter Tuning*, we **will discuss the best practices around model evaluation in more detail**.

Using the `train_test_split` function from scikit-learn's `model_selection` module, we randomly split the `X` and `y` arrays into 30 percent test data (45 examples) and 70 percent training data (105 examples):

```
train_test_split(X, y, test_size=0.3, random_state=1, stratify=y
    )
```

Code 1.2: Train Test Split Function

Observation 1.1.5

Note that the `train_test_split` function already shuffles the training datasets internally before splitting; otherwise, all examples from class `0` and class `1` would have ended up in the training datasets, and the test dataset would consist of 45 examples from class `2`. Via the `random_state` parameter, we provided a fixed random seed (`random_state=1`) for the internal pseudo-random number generator that is used for shuffling the datasets prior to splitting. Using such a fixed `random_state` ensures that our results are reproducible.

Cristo Daniel Alvarado ESFM