Handwritten symbol recognition using Scikit-learn

Peter-Tibor Zavaczki

march 7, 2018

Chapter 1

About Scikit-learn

1.1 Tool Purpose

Scikit-learn is a machine learning library for Python, which features various classification, regression and clustering algorithms, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

1.2 Installing scikit-learn

Prior to using scikit-learn, Python (>= 2.7 or >= 3.3) has to be installed along with the NumPy (>= 1.8.2) and SciPy (>= 0.13.3) libraries.

1.2.1 Installing steps

Some versions of Ubuntu come installed with Python 2.7.12 and Python 3.5.2, so for this installation we will consider that and install scikit-learn for Python 3.5.2. To ease installing packages for Python we will use pip. To install pip, we need the command *sudo apt install python3-pip*. Please note that we have a 3 after python to signal that we will install pip for Python 3.x. After the previous step we install NumPy and SciPy by using the command *sudo pip3 install numpy scipy*. This will download the libraries' latest version and automatically install them. As a final step, we use the command *sudo pip3 install scikit-learn* to install scikit-learn.

1.3 Studied example

The studied example is **Recognizing hand-written digits** by **Gael Varoquaux**, a handwritten digit classificator by machine learning. It can recognize the 0-9 handwritten digits and convert them to digital characters.

1.3.1 How to run the example(s)

To run the given example, you need to have matplotlib, installed with *sudo pip3 install matplotlib* and python3-tk, installed with *sudo apt-get install python3-tk*.

Then just use the command *python3 ./plot_digits_classification.py* from the folder of origin to run the example.

1.3.2 Algorithm

The given example relies on a few libraries which it imports and works with. These are matplotlib.pyplot, and from sklearn, the datasets, svm, metrics libraries. After the libraries have been loaded, the application loads the processed dataset using the datasets.load_digits() command.

The images and the targets of the digits dataset is zipped into touples and added to a lists, so that it can be worked with in the following nested for-each loop (first level iterating by index, second level iterating by (image, prediction) touple). Please note that the loop only takes the first 4 (image, target) touples! In the mentioned for loop, at each iteration a new subplot is activated, the axis' are turned off (we are displaying an image and labelling it to see what it is, we are not actually displaying an actual plot), an image is shown on the axis with the gray_r colormap set and the interpolation set to nearest (this is the best choice when a small image is enlarged), then a title is set for each sepparate subplot, which signals the character trained using that data sample.

In the following step the number of image samples is saved in the n_samples variable. In the data variable a reshaped version of the digits' images' array is stored in the (samples, feature) matrix format. A Support Vector Classifier is instantiated with a gamma of value 0.001 and then the first half of the dataset is used for training.

After training with the first half of the dataset, the second half's targets are stored in the 'expected' variable and the predictions from the second half of the dataset are stored in the 'predicted' variable.

After predicting the second half of the dataset, we print the SVC's prameters, which in this case is all default, except for gamma and the classification report, which consists of listing the possible cases and the precision, recall and f1-scores calculated for them, along the number of samples of that case in the predicted dataset. The second part of printing the prediction data is the confusion matrix, which represents the expected values on the rows and the predicted values on the columns.

In the next step, the second half of the dataset and the predictions are zipped into a list of touples, so that the first four predicted images can be displayed similarly to the first four training images before.

As a last step, the plot is shown so that we can see the results.

Chapter 2

Proposed project

The proposed project is a handwritten letter recognizer, working on the "EMINST handwritten letters dataset, from kaggle".

2.1 The dataset's format

The above mentioned dataset is split into two csv files, representing a training set of 171.610.185 bytes of data and a testing set of 28.625.756 bytes. Each row of the csv has 785 columns, which represents an instance of data. On the first column, the number represents the position of the letter in the english alphabet (1 for a, 26 for z). The rest of the row's columns represent the values of a flattened 28x28 pixel grayscale image (784 values on a row, each representing a pixel), with values ranging from 0 to 255.

2.2 The code

The code is written in Python, as such the lines starting with a hash symbol (#) are considered comments.
The implementation boils down to a few little snippets regarding the manipulation of the dataset, creating the classifier, fitting the training data, then predicting the testing data, then outputting the classification report and confusion matrix generated by the classifier.

2.2.1 Importing the necessary libraries

In this section we import the libraries, with which we will work later on. Namely: numpy, svm and metrics from sklearn and joblib from sklearn.externals.

```
# Import numpy / pandas for csv loading
import numpy as np

# Import classifiers and performance metrics
from sklearn import svm, metrics

# Import pandas or joblib for storing data
```

```
# import pandas as pd
from sklearn.externals import joblib
```

2.2.2 Manipulating the training dataset

In this section we load the raw training dataset using numpy, then we split it by the first column and the rest (the letter and its representation). After we have split the data into the specific parts, we flatten the letter representations into a single line, so that, in case it is represented as an nxn matrix, it will be a vector of length nxn.

```
# The letters dataset
lettersTrainRaw = np.loadtxt(fname = "./emnist-letters-train.csv", delimiter = ',')
lettersTrainTarget = lettersTrainRaw[:, 0]
lettersTrainData = lettersTrainRaw[:, 1:]

# To apply a classifier on this data, we need to flatten the image,
# to turn the data in a (samples, feature) matrix:
n_samplesTrain = len(lettersTrainData)
dataTrain = lettersTrainData.reshape((n_samplesTrain, -1))
```

2.2.3 The classifier

In this step we create a SVC (Support Vector Classifier) to which we will fit our training data and then predict the testing data.
In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to a category of a given set, an SVM training algorithm builds a model that assigns new examples to a category of the learned sets.
The support vector clustering algorithm created by Hava Siegelmann and Vladimir Vapnik, applies the statistics of support vectors, developed in the support vector machines algorithm, to categorize unlabeled data, and is one of the most widely used clustering algorithms in industrial applications.
In SciKit we have SVC as a subclass of svm. Svm represents support vector machine, and SVC is a Support Vector Classifier. SVC can be of 4 types based on its kernel function: linear, polynomial, rbf (radial basis function) or sigmoid. In our case a rbf based SVC is used, which means the equation it is based on is $(-\gamma \left\|x - x'\right\|^2)$ with $\gamma = 0.001$.

```
# Create a classifier:  a support vector classifier
classifier = svm.SVC(gamma=0.001)
# We learn the letters in the training dataset
classifier.fit(dataTrain[:n_samplesTrain // 1], lettersTrainTarget[:n_samplesTrain // 1])
```

2.2.4 Manipulating the testing dataset

In this secton we modify the loaded testing dataset similarly to how we did with the training dataset.

```
# Now predict the value of the letters in the tresting set
lettersTestRaw = np.loadtxt(fname = "./emnist-letters-train.csv", delimiter = ',')
lettersTestTarget = lettersTestRaw[:, 0]
lettersTestData = lettersTestRaw[:, 1:]

n_samplesTest = len(lettersTestData)
dataTest = lettersTestData.reshape((n_samplesTest, -1))
```

2.2.5 Predicting the testing dataset and showing the results

In this section we take the already trained classifier and use it to predict a different dataset, called the testing dataset. After the prediction we get some results, such as a report, containing the precision, recall, f1-score and the support of each predicted value. After the report has been printed, we print a confusion matrix which shows each value and what has been predicted.

```
predicted = classifier.predict(dataTest)
print("Classification report for classifier %s:\n%s \n"
% (classifier, metrics.classification_report(lettersTestTarget, predicted)))
print("Confusion matrix:\n%s" % metrics.confusion_matrix(lettersTestTarget, predicted))
```

2.3 Results

The results are composed of a report, containing the precision, recall, f1-score and the support of each predicted value and a confusion matrix. For the given dataset, the result looks as can be seen in the result.txt file.