CG3002

Multi-Class Classification Tutorial with the Scikit-learn Library

Zhao Na

Overview

- **Scikit-learn** is a Python library for machine learning that features various classification, regression and clustering algorithms including support vector machines, random forests, *k*-means, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

- In this tutorial, you will discover how you can use **Scikit-learn** to develop and evaluate **SVM** model for multi-class classification problem.

Problem Description

In this tutorial, we will use the standard machine learning dataset called the iris flowers dataset.

- This dataset is well studied and is a good problem for practicing on machine learning because all of the 4 input variables are numeric and have the same scale in centimeters. Each instance describes the properties of an observed flower measurements and the output variable is specific iris species.
- This is a multi-class classification problem, with three classes (flower species) to be predicted.
- The iris flower dataset is a well-studied problem and a such we can expect to achieve a model accuracy in the range of 95% to 97%. This provides a good target to aim for when developing our models.

Step-by-step tutorial

- ❑ Step 0: Data segmentation
- ❑ Step 1: Data pre-processing
- ❑ Step 2: Model selection: choosing estimators and parameters
- ❑ Step 3: Evaluate the model with K-fold cross validation

Step 0: Data segmentation

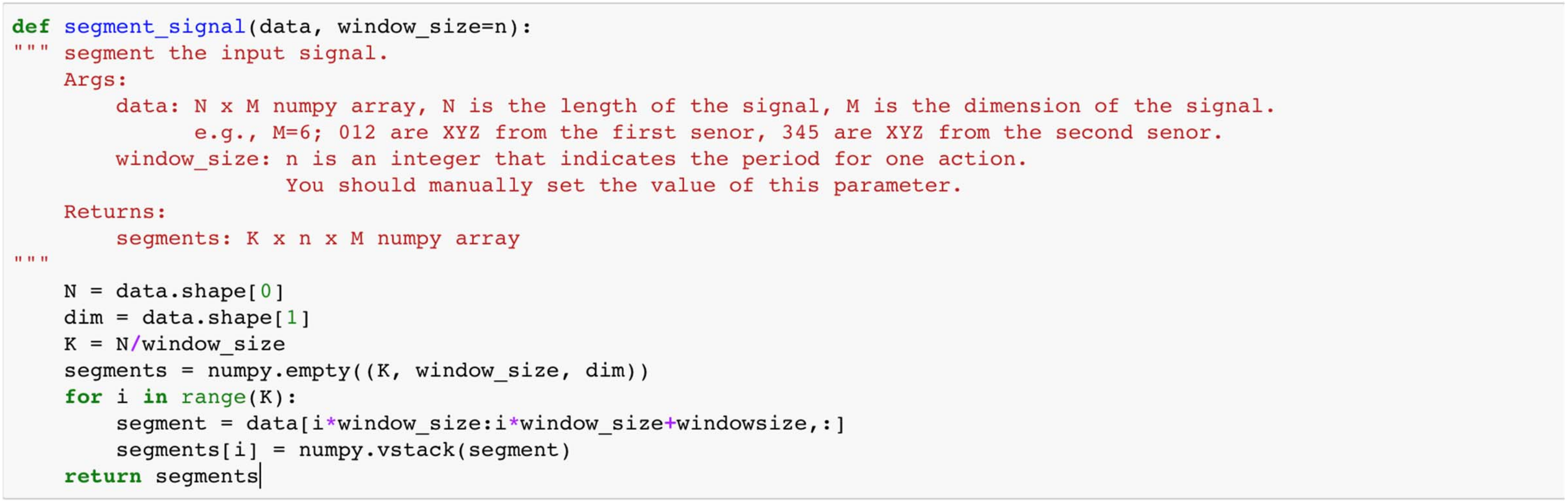
In the iris flower dataset, there is no need to segment data. However, in your actual project data segmentation is necessary, because identifying the start and end of the actual activity from signals is important for recognizing the activity accurately.

There are several criteria to segment signal, such as:

- ❖ Time
- ❖ Abrupt change

Step 0: Data segmentation

- In this tutorial, we give an example that how to segment signal with non-overlap time window.



```
def segment_signal(data, window_size=n):
"""  segment the input signal.
    Args:
        data: N x M numpy array, N is the length of the signal, M is the dimension of the signal.
              e.g., M=6; 012 are XYZ from the first senor, 345 are XYZ from the second senor.
        window_size: n is an integer that indicates the period for one action.
                     You should manually set the value of this parameter.
    Returns:
        segments: K x n x M numpy array
"""
    N = data.shape[0]
    dim = data.shape[1]
    K = N/window_size
    segments = numpy.empty((K, window_size, dim))
    for i in range(K):
        segment = data[i*window_size:i*window_size+windowsize,:]
        segments[i] = numpy.vstack(segment)
    return segments
```

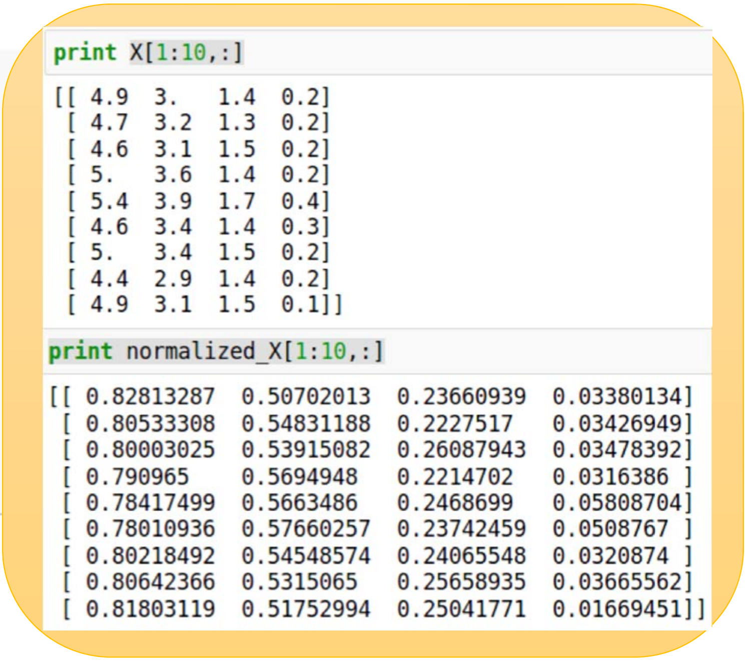
Step 1: Data pre-processing

- In terms of modality of your own input data, different preprocessing techniques can be applied to the raw data to smooth the signal or remove the noise.

❑ Normalization

Here, we will introduce how to use to scale input data into a certain range.

```python
# Normalize the data attributes for the Iris dataset.
from sklearn.datasets import load_iris
from sklearn import preprocessing
# load the iris dataset
iris = load_iris()
print(iris.data.shape)
# separate the data from the target attributes
X = iris.data
y = iris.target
# normalize the data attributes
normalized_X = preprocessing.normalize(X)
```



```
print X[1:10,:]

[[ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]
 [ 5.4  3.9  1.7  0.4]
 [ 4.6  3.4  1.4  0.3]
 [ 5.   3.4  1.5  0.2]
 [ 4.4  2.9  1.4  0.2]
 [ 4.9  3.1  1.5  0.1]]
```

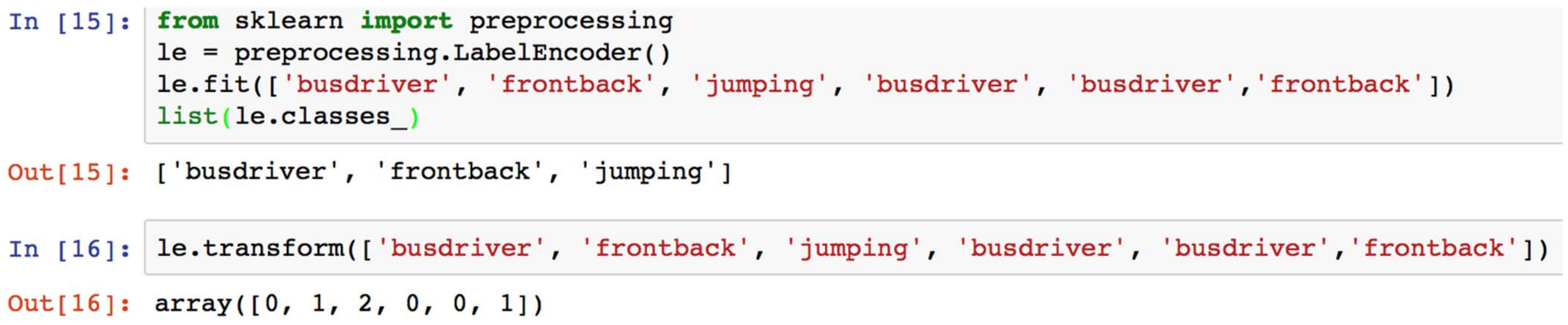
```
print normalized_X[1:10,:]

[[ 0.82813287  0.50702013  0.23660939  0.03380134]
 [ 0.80533308  0.54831188  0.2227517   0.03426949]
 [ 0.80003025  0.53915082  0.26087943  0.03478392]
 [ 0.790965    0.5694948   0.2214702   0.0316386 ]
 [ 0.78417499  0.5663486   0.2468699   0.05808704]
 [ 0.78010936  0.57660257  0.23742459  0.0508767 ]
 [ 0.80218492  0.54548574  0.24065548  0.0320874 ]
 [ 0.80642366  0.5315065   0.25658935  0.03665562]
 [ 0.81803119  0.51752994  0.25041771  0.01669451]]
```

Step 1: Data pre-processing

❑ Encode The Output Variable

In your project, the output variable contains 10 different string values, such as “jumping” and “busdriver”.

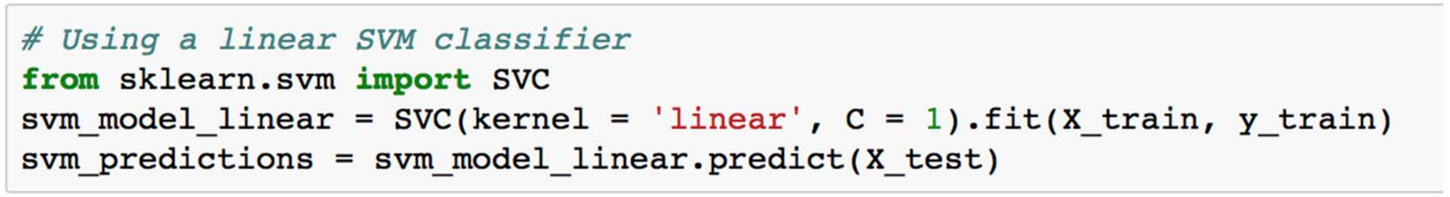
When modeling multi-class classification problems, it is good practice to transform non-numerical labels to numerical labels.



```
In [15]: from sklearn import preprocessing
         le = preprocessing.LabelEncoder()
         le.fit(['busdriver', 'frontback', 'jumping', 'busdriver', 'busdriver','frontback'])
         list(le.classes_)

Out[15]: ['busdriver', 'frontback', 'jumping']

In [16]: le.transform(['busdriver', 'frontback', 'jumping', 'busdriver', 'busdriver','frontback'])

Out[16]: array([0, 1, 2, 0, 0, 1])
```

Step 2: Model selection

- Here, we give an example of using SVM classifier. Also, you can select other models (such as Decision tree, KNN, and Neural network) to fit your data.

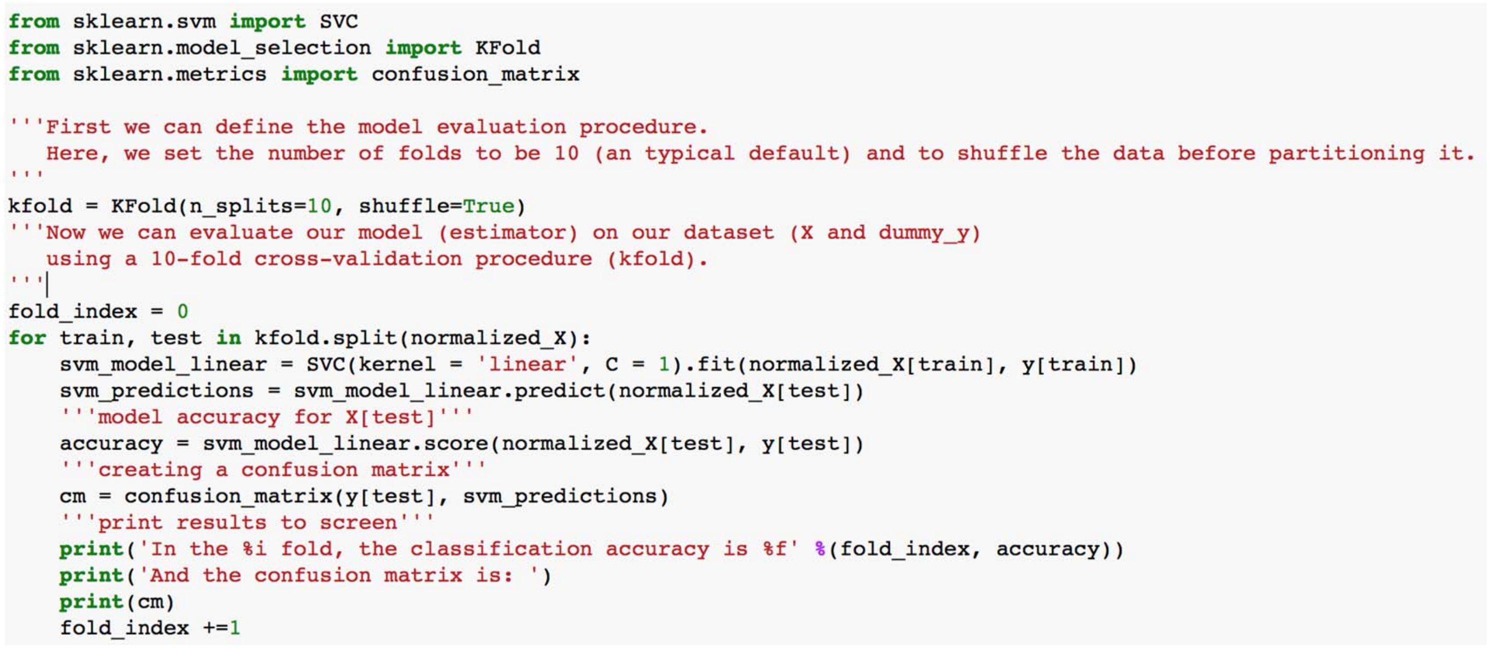


```
# Using a linear SVM classifier
from sklearn.svm import SVC
svm_model_linear = SVC(kernel = 'linear', C = 1).fit(X_train, y_train)
svm_predictions = svm_model_linear.predict(X_test)
```

Step 3: Evaluate the model

❖Evaluate the model with K-fold cross validation

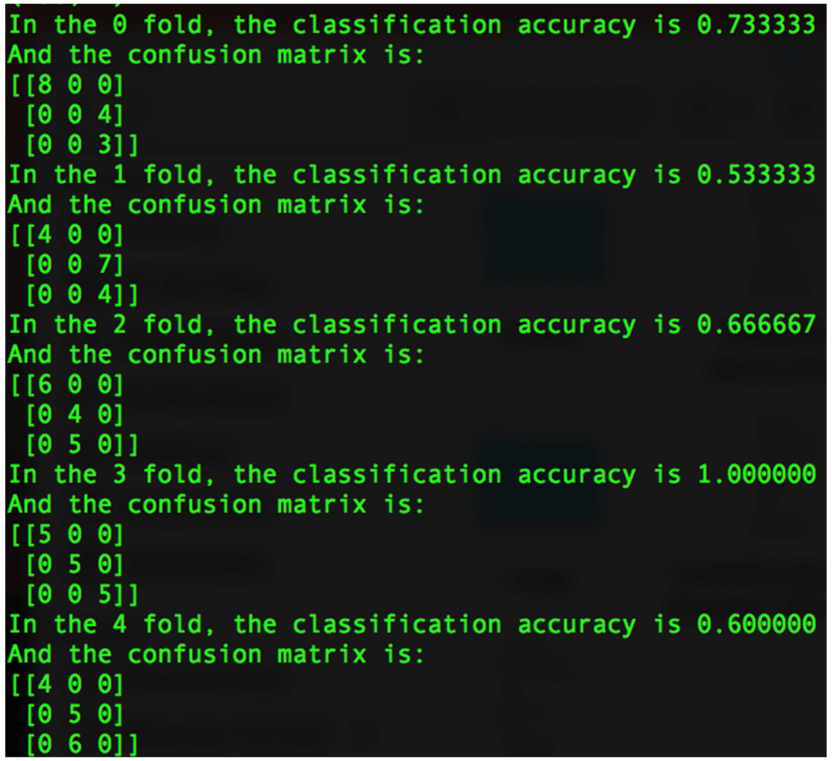
- We can now evaluate the SVM model on our training data.
- The scikit-learn has excellent capability to evaluate models using a suite of techniques. The gold standard for evaluating machine learning models is k-fold cross validation.



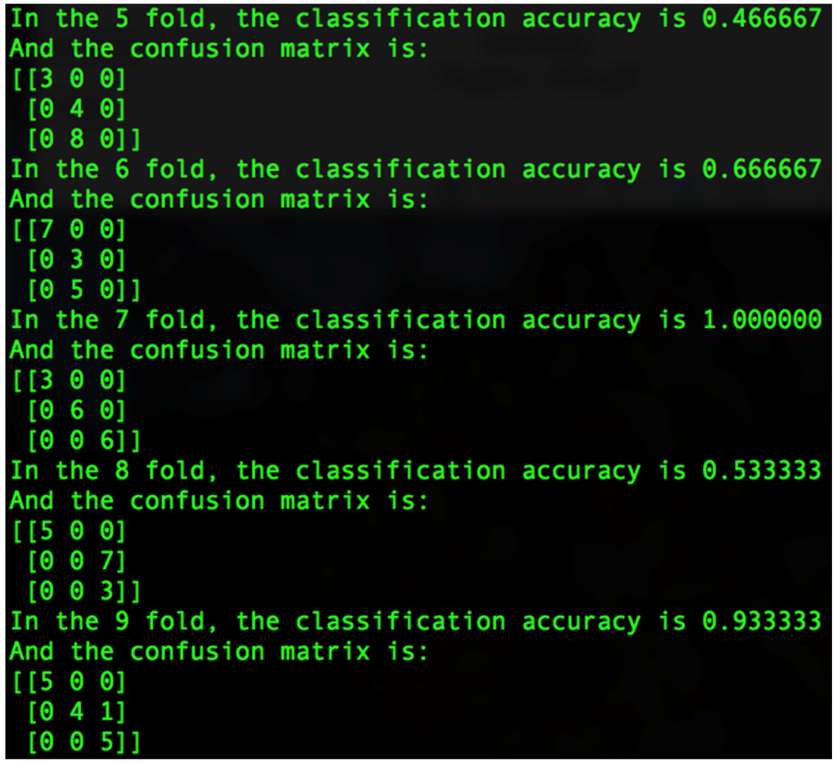
```python
from sklearn.svm import SVC
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix

'''First we can define the model evaluation procedure.
   Here, we set the number of folds to be 10 (an typical default) and to shuffle the data before partitioning it.
'''
kfold = KFold(n_splits=10, shuffle=True)
'''Now we can evaluate our model (estimator) on our dataset (X and dummy_y)
   using a 10-fold cross-validation procedure (kfold).
'''
fold_index = 0
for train, test in kfold.split(normalized_X):
    svm_model_linear = SVC(kernel = 'linear', C = 1).fit(normalized_X[train], y[train])
    svm_predictions = svm_model_linear.predict(normalized_X[test])
    '''model accuracy for X[test]'''
    accuracy = svm_model_linear.score(normalized_X[test], y[test])
    '''creating a confusion matrix'''
    cm = confusion_matrix(y[test], svm_predictions)
    '''print results to screen'''
    print('In the %i fold, the classification accuracy is %f' %(fold_index, accuracy))
    print('And the confusion matrix is: ')
    print(cm)
    fold_index +=1
```

Step 3: Evaluate the model

- The output:



```
In the 0 fold, the classification accuracy is 0.733333
And the confusion matrix is:
[[8 0 0]
 [0 0 4]
 [0 0 3]]
In the 1 fold, the classification accuracy is 0.533333
And the confusion matrix is:
[[4 0 0]
 [0 0 7]
 [0 0 4]]
In the 2 fold, the classification accuracy is 0.666667
And the confusion matrix is:
[[6 0 0]
 [0 4 0]
 [0 5 0]]
In the 3 fold, the classification accuracy is 1.000000
And the confusion matrix is:
[[5 0 0]
 [0 5 0]
 [0 0 5]]
In the 4 fold, the classification accuracy is 0.600000
And the confusion matrix is:
[[4 0 0]
 [0 5 0]
 [0 6 0]]
```



```
In the 5 fold, the classification accuracy is 0.466667
And the confusion matrix is:
[[3 0 0]
 [0 4 0]
 [0 8 0]]
In the 6 fold, the classification accuracy is 0.666667
And the confusion matrix is:
[[7 0 0]
 [0 3 0]
 [0 5 0]]
In the 7 fold, the classification accuracy is 1.000000
And the confusion matrix is:
[[3 0 0]
 [0 6 0]
 [0 0 6]]
In the 8 fold, the classification accuracy is 0.533333
And the confusion matrix is:
[[5 0 0]
 [0 0 7]
 [0 0 3]]
In the 9 fold, the classification accuracy is 0.933333
And the confusion matrix is:
[[5 0 0]
 [0 4 1]
 [0 0 5]]
```