Machine Learning in Python

A Jupyter Notebook Example

Introduction

This presentation serves as a walk-through and detailed explanation of the provided Example Jupyter Notebook covering machine learning in Python.
This Notebook loads pre-generated datasets provided by the libraries used to perform the machine learning tasks and performs those tasks.
Here, we will step through the example code and explain each block’s function.

Part 1 - K Means

```
import numpy
import scipy
import sklearn
from sklearn.datasets import
    load_wine
from scipy.cluster.vq import
    kmeans2, whiten
```

This sequence of statements import the necessary libraries for clustering using K Means. The *numpy* library is not used directly, but is used by the other libraries for better handling of matrices than base Python. The *load_wine* function contains a dataset of wine and its features that will be used to demonstrate machine learning techniques.

K Means Cont.

```
[data, target] = load_wine(
    return_X_y=True)
whitened = whiten(data)
```

These two lines load the wine dataset and perform whitening, or feature-wise normalization. The parameter passed to the load function causes it to return the feature matrix and class labels separately. Whitening the data is needed since kmeans considers all features numerically equal, so a feature with a range of 1000-9000 will be given greater weight than one with a 1-9 range without whitening.

Kmeans Cont.

```
start = [whitened[0],
    whitened[int(len(whitened)/2)],
    whitened[len(whitened)-1]]
[centroid, label] =
    kmeans2(whitened, start)
```

These three lines determine the starting centroids of the three clusters and perform the clustering. Without defining starting centroids, the kmeans algorithm will select random centroids, which will cause unpredictable behavior and inconsistent clustering.

K Means Cont.

```
errors = 0.0
for i in range(len(label)):
    if target[i] != label[i]:
        errors += 1
acc = (len(data) - errors)
      /len(data)
print (acc)
```

These lines evaluate and report the accuracy of the clustering. In this case, we simply count up the data instances that were assigned to a different cluster than their true label. This is a simple example, and more complex assessments of clustering reliability would be necessary for more complex datasets.

Part 2 - Decision Trees

```
import numpy
import scipy
from sklearn.datasets import
    load_wine
from sklearn import tree
```

These first lines serve the same function as in the kmeans example. Here, the Decision Tree function *tree* is part of the *sklearn* or Scikit Learn library as opposed to kmeans coming from *scipy*.

Decision Trees Cont.

```
[data, target] = load_wine(
      return_X_y=True)
```

Again, load the dataset as separate feature values and true class labels. Since Decision Trees consider each feature individually, it is not necessary to whiten the feature matrix in this case.

Decision Trees Cont.

```
data_train = data[0:len(data)-2]
target_train = target[0:len(data)-2]
```

To demonstrate the effectiveness of Decision Trees, we create a training set using leave-one-out cross validation. In this case, the the final data instance is left out of the data.

Decision Trees Cont.

```
dtc = tree.DecisionTreeClassifier()
dtc = dtc.fit(data_train,
      target_train)
print (dtc.predict(
      data[len(data)-1]
            .reshape(1,-1)))
```

The last three lines initialize, train, and test a Decision Tree using the training data set. The final line is the testing step and also reports the prediction of the Decision Tree. If we were performing leave-one-out cross validation for real, we would randomly select a large number of samples to leave out and, training a new tree for each sample left out, accumulate the results of the testing over all the samples.

Part 3 - Support Vector Machines (SVM)

```
import numpy
import scipy
from sklearn.datasets import
    load_wine
from sklearn import svm
```

These first lines serve the same function as in the kmeans example. Here, the Support Vector Machine function *svm* is part of the *sklearn* or Scikit Learn library as opposed to kmeans coming from *scipy*.

SVM Cont.

```
[data, target] = load_wine(
    return_X_y=True)
data_train =
    data[0:len(data)-2]
target_train =
    target[0:len(target)-2]
```

Again, the dataset is loaded as features and labels and divided into a training and test set. Like Decision Trees, SVMs do not generally require data whitening.

SVM Cont.

```
clf = svm.LinearSVC()
clf.fit(data_train,
    target_train)
print (clf.predict(
    data[len(data)-1]
        .reshape(1,-1)))
```

Again, we initialize, train, and test the SVM classifier. In these two examples, the Decision Tree and SVM classifier both correctly predict the class of the last sample. Can you find a sample that one or both classifiers does not correctly predict?