

Intelligent and Interactive Systems

Spring 2016

Lab 2: Software for Machine Learning

Scikit-learn is an open source package for Machine learning algorithms in Python. In this lab a supervised classifier (k-NN classifier) is introduced for digit classification. Working with high dimensional data and some visualization techniques are discussed. An overview of the classification accuracy and confusion matrix of overall classifier are presented. K-Means clustering is demonstrated to introduce the idea unsupervised learning.

This lab will be handled as a tutorial as the code is made available. The lab session is more of discussion into the implementation aspects complemented with some presentation whenever necessary.

Formality

- You need an account for the computer systems at the Dept. of Information Technology.
- You should work together in groups of two people. It may be advantageous to have someone to discuss issues with.
- The tutorials are not mandatory but you are highly recommended to take them. The topics in the tutorials will not only complement the learning during the lectures but will prove directly useful during the assignments and project.

Getting the material

In a lab at the Dept. of Information Technology running WINDOWS: Log on to one of the workstations. PYTHON is found under ITC-Application Explorer(Zenworks)

Download the tutorial material from the Student portal, 'File areas' → 'Lab material'

We will be working with the digits dataset that is available in SCIKIT-LEARN. We will develop the code as we go along in this tutorial, it is recommended to start coding in separate file for the supervised and unsupervised tasks and in the end compare your files with `supervisedClassify.py`, `unsupervisedClassify.py` respectively available on STUDENTPORTALEN.

Supervised learning: Nearest neighbour classification

0.1 Data visualization

```
from sklearn import datasets
digits = datasets.load_digits()
```

We load the digits into the digits data structure here. We will need `digits.images`, which is an NUMPY array of digit images and their corresponding labels in `digits.target`.

We will begin by displaying the data that we will be working with using the code snippet.

```
i = 0
for image in digits.images:
    if(i < 10):
        imMax = np.max(image)
        image = 255*(np.abs(imMax-image)/imMax)
        res = cv2.resize(image,(100, 100), interpolation = cv2.INTER_CUBIC)
        cv2.imwrite('digit_'+str(i)+'.png',res)
        i+=1
    else:
        break
```

0.2 Dimensionality reduction using PCA

Here each image is 8×8 pixels, which is considered as 64 dimensional feature. This is reduced to a 2D plot to find any clustering pattern in the resulting data representaion.

```
from sklearn.decomposition import PCA
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))
pca = PCA(n_components=2)
X_trans = pca.fit_transform(data)
plt.scatter(X_trans[:,0], X_trans[:,1])
plt.show()
```

0.3 Manifold embedding with tSNE

We can also visualize the data taking into account the underlying variance in the feature vectors using tSNE.

```
tsne = manifold.TSNE(n_components=2, init='pca', random_state=0)
X_trans = tsne.fit_transform(data)
```

You can plot this data and check if the visualization improves over the previous method.

0.4 Classifier training and evaluation

TASK 1: We will manually split our data into training and testing sets (say 70-30).

Create a function

```
def holdOut(fnDigits,fnData,nSamples,percentSplit=0.8):
for this task. Use the follow code snippet to get things running.
n_trainSamples = int(nSamples*percentSplit)
trainData = fnData[:n_trainSamples,:]
trainLabels = fnDigits.target[:n_trainSamples]
testData = fnData[n_trainSamples:,:]
expectedLabels = fnDigits.target[n_trainSamples:]
```

Now train a kNN classifier on the data

```
n_neighbors = 10
kNNClassifier = neighbors.KNeighborsClassifier(n_neighbors, weights='distance')
# trains the model
kNNClassifier.fit(trainData, trainLabels)
predictedLabels = kNNClassifier.predict(testData)
print("Classification report for classifier %s: \n %s \n"
      % ('k-NearestNeighbour', metrics.classification_report(expectedLabels, predictedLabels)))
print("Confusion matrix:\n %s" % metrics.confusion_matrix(expectedLabels, predictedLabels))
```

TASK 2: Use cross_validation module to do training-testing split hold out for us

```
trainData, testData, trainLabels, expectedLabels = cross_validation.train_test_split(fnData, fnDigits.target,
test_size=(1.0-percentSplit), random_state=0)
```

TASK 3: Use cross_validation module to do *k*-fold cross validation

```
scores = cross_validation.cross_val_score(kNNClassifier, fnData, fnDigits.target, cv=kFold)
print(scores)
```

Unsupervised learning: k-Means clustering (Extra task)

Unsupervised learning entails finding structure without having explicit mapping between input and output data. We then set up some assumption of structure and let the algorithm find something with those constraints.

In a feature space, points might be grouped. If we are lucky, the points in a classification problem will be grouped according to class (this is rare though). Load the data as in the previous exercise. But now we cluster the data.

```
kclass = KMeans(init='k-means++', n_clusters=10, n_init=10)
y_pred = kclass.fit(data)
viewDataImg=True
if(viewDataImg):
    for i in range(y_pred.cluster_centers_.shape[0]):
        centroid = y_pred.cluster_centers_[i,:]
        imMax = np.max(centroid)
        imMin = np.min(centroid)
        image = (255*(np.abs(imMax-centroid+imMin)/imMax)).astype(np.uint8)
        image = image.reshape(8,-1)
        res = cv2.resize(image,(100, 100), interpolation = cv2.INTER_CUBIC)
        cv2.imwrite('digit_'+str(i)+'.png',res)
```