Intelligent and Interactive Systems
Spring 2019

Lab 2: Software for Machine Learning

Scikit-learn is an open source package for Machine learning algorithms in Python. In this lab two supervised classifers, namely k-NN and Support Vector Machine(SVM) are 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.

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 Linux: Log on to one of the workstations on the ThinLinc cluster. You can use any editor, but I suggest you to use Spyder for Python 2.7.

Download the tutorial material from the Student portal, 'File areas' → 'Lab material' → Lab2

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 code in separate sections. A skeleton file named `Lab2_ToDo.py` is given in the Lab2 folder and you may use it as a starting point. In the end compare your files with `Lab2_Completed.py` available on Studentportalen. The codes presented in this document are not complete, you may add/modify/import things in order to make them work.

Supervised learning: classification of digits dataset

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 representation.

```
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 Mainfold 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 Classifer training and evaluation using k-NN

Task 1: We will maually 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 classifer 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 k-fold cross validation

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

0.5 Classifer training and evaluation using SVM

Task 1: We now use SVM to train the classifier and compare the result with using kNN.

```
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
clf_svm = LinearSVC()
clf_svm.fit(trainData, trainLabels)
y_pred_svm = clf_svm.predict(testData)
acc_svm = accuracy_score(expectedLabels, fnDigits.target)
print "Linear SVM accuracy:  ",acc_svm
```

Task 2: Use `cross_validation module` to do k-fold cross validation

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

Deep Learning: digits classification (Extra task)

Simply speaking, deep learning models are machine learning models that consist at least one hidden layer. These models have drawn people's attentions in many fields in recent years because of its superb performance.

You are required to explore the deep learning models and use them for this classification tasks. In the area of deep learning, a better tool called tensorflow[1] is used. A good starting point could be found at this tutorial[2]. By following the instructions on the webpage, you should be able to at least construct a neural network model with one hidden layer to complete this task.

[1]https://www.tensorflow.org/
[2]https://www.tensorflow.org/get_started/mnist/beginners