University of Stavanger

Stavanger, November 19, 2021

Laboratory exercise 5

ELE520 Machine learning

A PDF version of the report of the student solution to the exercise including figures and answers to questions shall be submitted on CANVAS.[1]

In this laboratory project the objective is to get familiar with some libraries for machine learning. We will start out solving the problem 1 from laboratory exercise 4 with library functions similar to the ones we have developed ourselves. We will put focus on *performance evaluation*. In problem 2 we will continue with new approaches to solving the same problem but now we will apprach the problem with some basic experiments with *neural networks*.

When we design a classifier, it is very important to know how its expected performance will be when it will classify new unlabeled data in the future. One such performance measure is the error rate.

To get an idea of how the classifier will perform on unlabelled data, we will classify data not included in training the classifier. This is the role of the test data set. We will let the classifier label all the test data, and we will compare these proposed labels with the true ones.

If we want to compare several classifier designs, we can evaluate the performance of each and then select the one with best performance.

If we want to compare several classifier designs, we can evaluate the performance of each and then select the one with best performance.

Problem 1

In this problem, you will be working with the same data as in problem 1, laboratory exercise 4. You will use scikit-learn[2], the Scientific Python Machine learning library. The focus will be on solving some of the problems from exercise 4 at varying levels: pdf estimation, classifier design, cross validation and grid search. The solution for this problem can possibly be made by modifying the solution to laboratory exercise 4.

[1]If it is not possible to export the Jupyter notebook to PDF, the ipynb and py files can be submitted.
[2]This is a live link

Trygve Eftestøl, Professor
Faculty of Science and Technology
Department of Electrical and Computer Engineering
Kjølv Egelands hus

University of Stavanger
N-4036 Stavanger.

Telephone: +47 51 83 10 00.
Telefax: +47 51 83 17 50.
E-mail: trygve.eftestol@uis.no
www.ux.his.no/~trygve-e

a) In the instructor's solution, the pdf functions are estimated in the `classify` function. Add a scikit-learn variant of the Parzen window classifier, KernelDensity, which can be imported as follows[3]:

```
# from sklearn.neighbors.kde import KernelDensity
from sklearn.neighbors import KernelDensity
```

You need to check the description of the function at the reference page which you will find by following the live link above. You will need to provide the proper *parameters* to choose a window function and window width. You must also provide the appropriate *methods* to fit the model from the training data and then to evaluate the model on the data you want to classify. Note theat the model will generate log-transformed density values, so it is advisable to applyt the Numpy `exp()` function.

Experiment with the window widt to see if you can get approximetely the same results as in problem 1, lab4. We used the following window widths ($h_1 = 0.1$ and $h_1 = 5$). These parameters can not be transferred directly to the library function. You will need to identify the corresponding parameter and try different values. An approach might be to reduce the window width until the reclassifcation error is 0 and then start increasing until you get a similar coorespondence between the reclassification and test results.

You can use the data structures in the file *lab4.p* with simulated data for a two class problem. In the cell structures `X` and `Y` you will find data that can be used for training and testing of the classifier as used in laboratory exercise 4.

```
        pfile = 'lab4.p'
        with open(pfile, "rb") as fp:
            X, Y   = pickle.load(fp)
```

b) For the following problems you will use higher level library functions that requires the data to be formatted differently from what we have used so far.

You need to make a function (call it `list2mat`) to restructure the data so that all feature vectors are placed in a common matrix, one feature vector per row. In addition, a separate vector should contain the corresponding class labels.

The data can be loaded as shown in the following lines of code:

```
if prb == '1':
    pfile = 'lab4.p'
    with open(pfile, "rb") as fp:
        Xo, Yo = pickle.load(fp)
```

The following code show how such a function can be called when the original data was loaded as shown in the previous listing and the function `list2mat` added to *labsol5.py*.

```
        prm = [[]]
        X, yx = labsol5.list2mat(Xo)
```

[3]The commented line is for an older version of scikit-learn.

c) The following listings shows a suggested code for the reclassification and the testing.

```
    for k in range(0, M):
        gx[k], Cx[k], pxwx[k], Pwx = classify(Y, X[k], met,
    discr, prm)
        gy[k], Cy[k], pxwy[k], Pwy = classify(Y, Y[k], met,
    discr, prm)
        CNx[k, :] = np.bincount(Cx[k], minlength=M)
        CNy[k, :] = np.bincount(Cy[k], minlength=M)
```

and following computation of the confusion matrices.

From now on you will use the `list2mat` function to convert the lists to data matrices with label vectors. You will not need to use the for loop anymore as the scikit-learn based classifiers will handle data with multiple classes.

Now, you can replace this with a scikit-learn based version of the ML classifier, QuadraticDiscriminantAnalysis, which can be imported as follows:

```
from sklearn.discriminant_analysis import
    QuadraticDiscriminantAnalysis
```

You do not need any parameters. You need to fit the model to the training data and classify both the data sets as before.

The next step will be to determine the confusion matrices from the classification results. For this purpose, use the scikit-learn based method for this, confusion_matrix, which can be imported as follows:

```
from sklearn.metrics import confusion_matrix
```

This function is applied with the classification results as input. You should get exactly the same results as with the ML classifier.

d) Repeat the previous subtask, but now with a scikit-learn based nearest neighbour classifier KNeighborsClassifier, which can be imported as follows:

```
from sklearn.neighbors import KNeighborsClassifier
```

Note that this is a method that uses a majority vote, in contrast to the method implemented in lab4, using pdf-estimates. Experiment with different choices of k_N to see how close you can get to the corresponding results of lab4.

e) Repeat the two previous subtasks, now using 5-fold crossvalidation. Report the mean and 2 times the standard deviation of the performance metrics. Comment on the standard deviation. You can use cross_validate, which can be imported as follows:

```
from sklearn.model_selection import cross_validate
```

Experiment with different choices of $k_n$for the nearest neighbor classifier. How is the standard deviation affected?

f) You might have noticed that experimenting with parameter settings is a bit like tweaking the instruments. It is a challenge to keep track of the results of the different settings. And itr gets more challenging as the number of parameters to tweak increases. Scikit-learn offers GridSearchCV to handle exhaustive searches over several parameters. The function can be imported as follows:

```
import labsol4
```

. Do a grid search with the nearest neighbor classifier where you vary k_n from 1 to 10 and also vary the algorithm used between *ball_tree* and *kd_tree*. Apply the best model to the test date and explore the type of results that are provided.

Problem 2

We will still try to solve the same problem, but this time we will use code from a working example and adapt it to our purpose. You will use tensorflow, Google's library for deep learning. The github companion site to the *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by Aurélion Geron contains jupyter notebooks with code for all examples presented in the book.

a) We start with tensorflow's web site. After looking a bit around, we find a tutorial giving an example of basic classification. This shows a straightforward example with a popular dataset for demonstrating neural networks on the fashion MNIST database, using the *keras* API. A modified version is shown in the following listing:

```
# Adapted from https://www.tensorflow.org/tutorials/keras/
    basic_classification
import tensorflow as tf

# Load data set
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Building the model

# Setup the layers - one hidden layer and one output layer
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])

# Compile the model - choosing optimizer, loss function and
# evaluation metric
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
# Train the model
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)

# Evaluate the accuracy
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

Adapt the shown code to handle your data set from problem 1. You will need to adapt your data set so it has the same structure as the data used in the example. You can omit the Flatten command in line 12 in the listing. Run the example code and study the data structure used there first.

You will also need to change the number of output nodes and you should also change the number of hidden nodes to a reasonable amount.

b) Experiment with the number of hidden nodes and hidden layers. See how it affects the results.

c) To get a better idea on how the neural networks actually are implemented, you can study the part, *Using plain TensorFlow*, of the notebook Chapter 10 - Introduction to artificial neural networks where a network is trained to handle the MNIST data set of handwritten digits. Looking at the following part of the code, you will actually see the weight matrix and bias vector and output from the layer.

```
def neuron_layer(X, n_neurons, name, activation=None):
    with tf.name_scope(name):
        n_inputs = int(X.get_shape()[1])
        stddev = 2 / np.sqrt(n_inputs)
        init = tf.truncated_normal((n_inputs, n_neurons),
   stddev=stddev)
        W = tf.Variable(init, name="kernel")
        b = tf.Variable(tf.zeros([n_neurons]), name="bias")
        Z = tf.matmul(X, W) + b
        if activation is not None:
            return activation(Z)
        else:
            return Z
```

In the following listing, you can see how this function is used to create the actual three layer network.

```
with tf.name_scope("dnn"):
    hidden1 = neuron_layer(X, n_hidden1, name="hidden1",
                           activation=tf.nn.relu)
    hidden2 = neuron_layer(hidden1, n_hidden2, name="hidden2"
   ,
                           activation=tf.nn.relu)
    logits = neuron_layer(hidden2, n_outputs, name="outputs")
```

The learning hyperparameters and other setup can be found in *mnist_demo2.py*.

Repeat (b), but now adapting this code.

d) For those interested, check out convolutional neural networks, recurrent neural networks, and autoencoders.