Hands-on Lab Description

CS-ML-00201 – Feed-Forward Neural Network (FNN)

Category:

```
CS-ML: Machine Learning
```

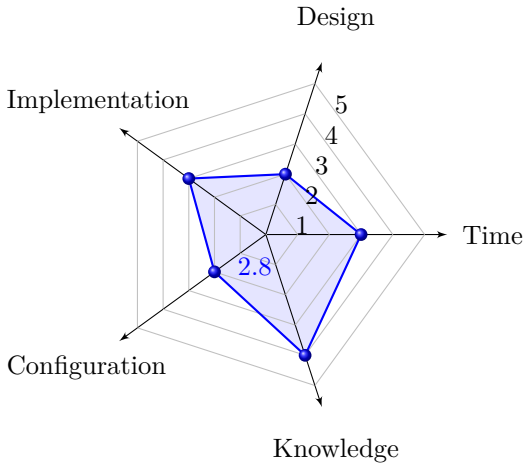
Objectives:

```
Learn Feed-Forward Neural Networks (FNN)
Learn how to setup FNN training model for network anomaly detection (using NSL-KDD
    dataset)
```

Estimated Lab Duration:

```
Expert: 180 minutes
Novice: 540 minutes
```

Difficulty Diagram:



Difficulty Table.

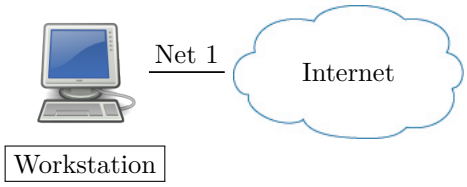
Measurements	Values (0-5)
Time	3
Design	2
Implementation	3
Configuration	2
Knowledge	4
Score (Average)	2.8

Required OS:

```
Linux: Ubuntu 18.04 LTS
```

Lab Running Environment:

```
VirtualBox https://www.virtualbox.org/ (Reference Labs: CS-SYS-00101)
```




```
1 Server: Linux (Ubuntu 18.04 LTS)
2 Network Setup: connected to the Internet
```

Lab Preparations:

```
Python and Anaconda software packages installed on the Linux VM. Reference Lab:
    CS-ML-00001
NSL-KDD dataset. Reference Lab: CS-ML-00101
Python Machine Learning basic concepts: CS-ML-00199
Python-based data pre-processing: CS-ML-00200
```

In this lab, we will practice supervised Machine Learning (ML) approaches. Particularly, we will focus on Feed-forward Neural Network (FNN) solutions. The majority of practical machine learning uses supervised learning. Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

$$Y = f(X).$$

The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data. It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers, the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

In this lab, will use NSL-KDD dataset as input (x), which provide labeled normal network traffic and labeled attack network traffic. FNN will use the well-labeled data to build FNN-based data pattern to differentiate normal and abnormal network traffic. Then, we use similar data set with labeled data to validate the accuracy of the generate anomaly detection pattern.

Task 1 Data Preprocessing

Data preprocessing is an important step to prepare the data for any Machine Learning (ML) models. There are many important steps in data preprocessing, such as data cleaning, data transformation, and feature selection.

Data cleaning and transformation are methods used to remove outliers and standardize the data so that they take a form that can be easily used to create a model. It is important to select only variables that contain unique and important information. Data mining procedures can be used to remove variables that do not contribute to the model. Another step in feature selection is feature fusion, the process by which multiple variables are combined using mathematical operations. This process is performed to reduce the number of variables in the model while still maintaining the level of information.

In this task, you will use one of four NSL-KDD datasets (refer to CS-ML-00101) as examples for training and testing. Here, we use *KDDTrain+.txt* data set as the illustrative example. First you need to import *pandas* and *numpy* for data preprocessing:

```
# To load a dataset file in Python, you can use Pandas. Import pandas
    using the line below
import pandas as pd
# Import numpy to perform operations on the dataset
import numpy as np
```

Next, you need to import dataset. Usually the dataset is given in *.csv* or *.txt* format. The following code is to load the dataset from the file *KDDTrain+.txt* and it is located in a sub-folder *NSL-KDD*:

```
dataset = pd.read_csv(./NSL-KDD/KDDTrain+.txt, header=None)
X = dataset.iloc[:, 0:-2].values
label_column = dataset.iloc[:, -2].values
y = []
for i in range(len(label_column)):
        if label_column[i] == 'normal':
                y.append(0)
        else:
                y.append(1)

# Convert i-st to array
y = np.array(y)
```

This code will ready data in X and y. Next, You need to encode categorical data, i.e., convert letters/words to numbers that FNN model can recognize. This can be done by using *ColumnTransformer* and *OneHotEncoder*. Note that python version earlier than 3.6 can use a bit more complicated approach such as *LabelEncoder* and *OneHotEncoder* for this purpose.

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
ct = ColumnTransformer(
        # The column numbers to be transformed ([1, 2, 3] represents
            three columns to be transferred)
        [('one_hot_encoder', OneHotEncoder(), [1,2,3])],
        # Leave the rest of the columns untouched
        remainder='passthrough'
)
X = np.array(ct.fit_transform(X), dtype=np.float)
```

Then, you can split the dataset into the training set and test set. For example, you can use 75% of data for training, and 25% of the data for testing. This can be done by creating several training and testing datasets as follows:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
    0.25, random_state = 0)
```

Finally, to improve the training accuracy, you need to perform feature scaling, such as *StandardScaler* method:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

# Scaling to the range [0,1]
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

Task 2 Building FNN Training Module

In this task, you will need to use *Keras* libaries to build up an FNN training module. First you need to importing the Keras libraries and packages and initialize the ANN module.

```
from keras.models import Sequential
from keras.layers import Dense

# Initialising the ANN
# Reference: https://machinelearningmastery.com/tutorial-first-neural-
    network-python-keras/
classifier = Sequential()
```

Now, you can build the neural network. For example, you can add 6 nodes in the input layer, i.e., the first

hidden layer; and you can use parameters such as uniform distribution for initializer, use Rectified Linear Unit (ReLU) function for activity function, and the total number of variables are specified in *input_dim*.

```
classifier.add(Dense(units = 6, kernel_initializer = 'uniform',
    activation = 'relu', input_dim = len(X_train[0])))

# Adding the second hidden layer, 6 nodes
classifier.add(Dense(units = 6, kernel_initializer = 'uniform',
    activation = 'relu'))

# Adding the output layer, 1 node,
# sigmoid on the output layer is to ensure the network output is
    between 0 and 1
classifier.add(Dense(units = 1, kernel_initializer = 'uniform',
    activation = 'sigmoid'))
```

Now, your FNN is constructed. The next step is to compile the neural network. You can use gradient decent algorithm '"adam", and the loss function can be formed by a binary classification problem:

```
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy',
    metrics = ['accuracy'])
# run training with batch size equals to 10 and 10 epochs will be
    performed
classifierHistory = classifier.fit(X_train, y_train, batch_size = 10,
    epochs = 10)
```

Finally, you need to evaluate the train results, such as showing the loss and accuracy of the training on the given dataset:

```
loss, accuracy = classifier.evaluate(X_train, y_train)
print('Print the loss and the accuracy of the model on the dataset')
print('Loss [0,1]: %.4f' % (loss), 'Accuracy [0,1]: %.4f' % (accuracy))
```

Task 3 Evaluate FNN Training Results

After finishing the training FNN, next, you need to use the test data set to evaluate the training model;

```
# using the trained model to predict the Test dataset results
y_pred = classifier.predict(X_test)
# y_pred equals to 0 if the prediction is less than 0.9 or equal to
    0.9,
# y_pred equals to 1 if the predication is greater than 0.9
y_pred = (y_pred > 0.9)
```

After finishing the predication based on the testing data set, you need to generate the confusion matrix in the format of

$$\begin{bmatrix} TN & FP \\ FN & TP \end{bmatrix}$$

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

```
print('Print the Confusion Matrix:')
print(cm)
```

Task 4 Visualize FNN Training and Testing Results

You can run the following code to visualize the training and testing results:

```
# Import matplot lib libraries for plotting the figures.
import matplotlib.pyplot as plt
```

Now, plot the accuracy:

```
# Nothe that Keras 2.2.4 recognizes 'acc' and 2.3.1 recognizes '
    accuracy'
# You can use the command python -c 'import keras; print(keras.
    __version__)' on MAC or Linux to check Keras' version
plt.plot(classifierHistory.history['accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.savefig('accuracy_sample.png')
plt.show()
```

Finally, you can plot the history of your loss function results:

```
print('Plot the loss')
plt.plot(classifierHistory.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.savefig('loss_sample.png')
plt.show()
```

Deliverable

Students need to submit a report to explain what they can accomplish based on the description presented in the *Lab Assessment* section. On how to submit screenshots, please refer to the *Guidance for Students* Section B.4 (Submit Lab Reports).

Related Information and Resource

```
Data preprocessing:
https://www.shanelynn.ie/select-pandas-dataframe-rows-and-columns-using-iloc-loc
-and-ix/
https://medium.com/@contactsunny/label-encoder-vs-one-hot-encoder-in-machine
-learning-3fc273365621
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train
_test_split.html
https://scikit-learn.org/stable/modules/preprocessing.html

Build ANN:
https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/
https://machinelearningmastery.com/rectified-linear-activation-function-for-deep
-learning-neural-networks/
https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep
-learning-neural-networks/
```

"nobreak

Lab Assessment (100 points)

In this lab, the students is required to use basic FNN model to create an anomaly detection model for network intrusion detection.

1. (50 points) Run the provided *fnn_sample.py* python program with different NSL-KDD datasets as the input, and compare their detection accuracy with the following neural network setup:

NN parameters	Test setup
Input layer	6
hidden layer	6
Output	1
Bachsize	10
Number of Epoch	10

   The compared study include the following datasets:

```
1  KDDTrain+.txt vs. KDDTrain+_20Percent.txt
2  KDDTest+.txt vs. KDDTest+_20Percent.txt
3  KDDTrain+.txt vs. KDDTest+.txt
4  KDDTrain+_20Percent.txt vs. KDDTest+_20Percent.txt
```

   Explain your observation based on the comparative results.

2. (50 points) Run the provided *fnn_sample.py* python program with different one NSL-KDD dataset: *KDDTrain+.txt* as the input, and compare their detection accuracy with the following different neural network setup:

NN parameters	Test 1 setup	Test 2 setup
Input layer	6	12
hidden layer	6	12
Output	1	1
Bachsize	10	20
Number of Epoch	10	20

Explain your observation based on the comparative results between two test setup.