UNSW-NB15 Data Set Classification
Report for ITKST42
Tietoturvallisuustekniikka

SAMULI RAHKONEN
University of Jyvaskyla
May 5, 2017

Abstract

A model for classifying UNSW-NB15 data set samples was developed using a random forest and feed-forward neural network. The system uses the random forest that classifies data to normal or malicious data. This information is then used to train a neural network to further classify the attack data to different attack categories.

The results for attack detection were very good with approx. 0.88 precision for attacks and nearly 1.0 precision for normal data samples. Attack categorization had problems in differentiating between attack classes and could mostly classify the attacks to two different classes. However, it could accurately classify normal network data.

Development had many problems common in machine learning. Large data amount, lack of memory and unbalance between class sizes caused most trouble. Especially unbalanced data made it difficult to generalize data classes.

1 Introduction

This is the report for the large project work in ITKST42 Tietoturvallisuustekniikka course.

More and more data is produced by people, intelligent systems and sensors every year. To find anything relevant information from the vast amounts of data, some data mining methods has to take place. Machine learning techniques can automate the process of finding useful patterns in the data and make predictions on new data. They can be used in finding anomalies that differ from the usual data.

In case of internet network data, it could be a case of finding anomalies that could threaten privacy, data security and normal operation of vital systems, like banks, power plants and administrative systems. In network data these anomalies could be denial of service attacks, worms or exploits. Detecting and blocking them is a high priority especially in zero day vulnerabilities' case, when they are not publicly known before hand.

This report describes a method for detecting network attack types using UNSW-NB15 data set [6], random forests and a neural network.

2 Classified Network Data

The network data set included 2.5 million data points with 47 features and two target labels: Is the data point attack (referred as "attack or not") and the category. The category specifies what type the data point is. Table 2 lists the number of records in each category, which shows clear unbalance between categories.

The features are described in the file ″NUSW-NB15_features.csv″. The features were of different types: Integer, Float, Binary, Nominal and Timestamp. An example of an integer feature could be ′dloss′ which refers to ″destination packets retransmitted or dropped″. One float-type feature is ″Sload″ which means ″destination bits per seconds″. Binary type means that the feature can have either 1 or 0 as an value. Integers, floats and timestamps are non-limited continuous data values. Nominal type refers to categories. Nominal feature places the data point to some category. For example feature ″proto″ means the transaction protocol, like ″udp″ or ″tcp″. Depending on the used model, nominal types had to be converted to numeric form.

3 Techniques

This chapter introduces techniques used in the classification.

Category	No. of records
Normal	2 218 761
Fuzzers	24 246
Analysis	2 677
Backdoors	2 329
DoS	16 353
Exploits	44 525
Generic	215 481
Reconnaissance	13 987
Shellcode	1 511
Worms	174

3.1 Random Forests

Random Forest is an ensemble learning technique, which constructs many decision trees that work on different sets of features. The prediction is determined by the majority vote. A decision tree is a predictive model which is formed of decision nodes and leaf nodes which represent a prediction label. An input goes through a path of tests which lead to a prediction.

ExtraTreesClassifier (from Scikit-Learn package) was used.

3.2 Neural Network

Neural networks (NN) aim to mimic biological neural networks which approximate functions that depend on large number of inputs. In machine learning, neural networks are used to present a model that can make predictions and classify previously unseen data. NNs are used in many tasks, like computer vision and speech recognition. In this experiment, it is used to classify network data.

Neural networks are formed of a large number interconnected neurons. A typical neuron sums the input values with each having some learnt weight and passes the sum to an activation function that returns one value. The logistic funtion is a popular choice for the activation function. The layout of neurons depends on the architecture of the network.

Traditionally neural networks are divided into three different classes of network architectures: Single-Layer Feedforward Networks, Multilayer Feedforward Networks and Recurrent Networks. [1]

Feed-forward neural networks are built with one or more layers of neurons which connect to the following layers of neurons. There are no connections to previous layers. For a single-layer feed forward network, there is only the output layer of nodes which perform all the computations. In the case of multilayer feedforward networks, the layers that are in between the output layer and the source are said to be "hidden layers". [1]

Recurrent neural networks have at least one feedback loop where neuron's output value is fed back to one of neurons of its own or previous layers.

Training of multilayer feedforward networks is needed to make the model working. Training data with desired target responses are fed to the network to adjust the weights of the neurons. The weights of the network define the system model. A highly popular algorithm for this is "error back-propagation algorithm". The algorithm passes the different layers of the network twice: a forward pass and a backward pass. [1]

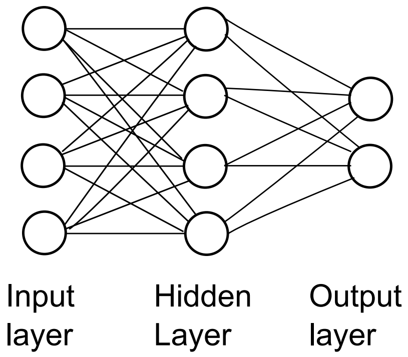


Figure 1: Feedforward network with

In the forward pass, an input vector (training data) is applied to the network's source and its effect propagates through the network until it reaches the output. The weights are fixed.

Backward pass then adjusts the weights according to the errors. The error signals are calculated by substracting the actual response from the desired response. The error signal is propagated backwards through the network. [1] The weights are adjusted according to the partial derivatives to the direction that reduces the errors.

3.3 Used Metrics

The confusion matrix is a way to visualize algorithm performance. Each column of the matrix represent instances in a predicted class and each row represent the instances of the true class. Confusion matrix makes it easy to see which classes the classification algorithm misclassifies to which classes.

Other used metrics are precision, recall and F1-score. Precision is the percentage of correct predictions over all predictions belonging to that class. Recall is the same as True-Positive Rate (probability of detection). It is the percentage of correct predictions over all true instances belonging to that class. F1-score is the harmonic mean of precision and recall, which provides one measurement for the class accuracy. Support is the number of data points used in calculating the scores.

4 Methods

The overall architecture is described in Figure 4. The used methods included data preprocessing, training two classifiers (a random forest and a neural network) and performance measurement.

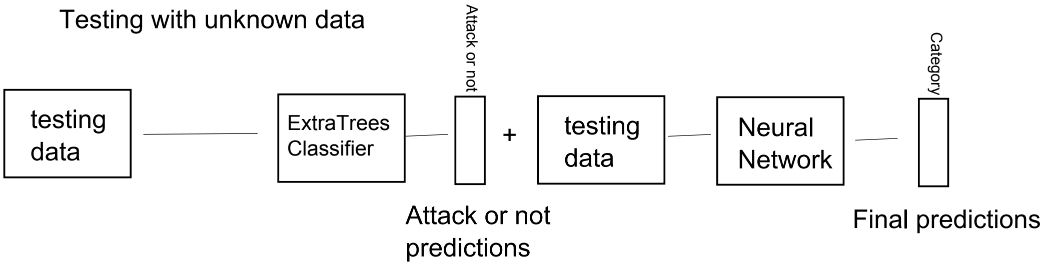


Figure 2: Overall architecture.

Python 3.5 with Pandas, Numpy, h5py, Scikit-Learn and Keras neural networking library with Theano backend were chosen for this task. Numpy and Pandas are used for accessing and rearranging the data. h5py is used for storing the data to HDF5 format in the hard drive between different steps. Scikit-Learn includes many ready-made machine learning algorithms and makes experimenting different techniques fast and easy. Keras is used for creating neural networks. Keras includes many different neural layer types and supports GPU. Nvidia Geforce GTX 770 was used for training the neural network.

Next subsections describe used methods in detail.

4.1 Idea

The network data classification was carried out by first classifying the data points as "attack or not" with a random forest technique. The classification class was then used as an additional data feature when classifying data points to attack categories.

The high-level process is described below:

1. Data preprocessing step.

2. Split data to training and testing data sets.
3. Train the random forest to predict ″attack or not″ label with training data without ″attack or not″ feature.
4. Use the random forest to generate additional ″attack or not″ feature to the testing data.
5. Train the neural network to predict attack categories with training data including ″attack or not″ feature.
6. Predict attack category with the testing data (with generated ″attack or not″ feature).
7. Apply performance metrics to measure how well the system generalizes the data.

4.2 Preprocessing

The data was read from the CSV files with Pandas. The large amount of networking data was a challenge in every step of the processing. The used computer had 8 GB of RAM. That is why HDF5 format was used between preprocessing steps.

Preprocessing was split to two phases. The first phase is described below (the file *h2_preprocess.py*):

1. Read the data
2. Split data by feature types
3. Convert each type accordingly
    (a) Replace NaN's with i.e. zero if possible, otherwise remove data point
    (b) Then trim nominal data from extra white spaces, set lower case, vectorize
4. Normalize data between [0, 1]
5. Save to HDF5

Nominal data was vectorized by one-hot encoding. For example, a feature with three possible nominal values was converted to three features, where value 1 means that the data point belongs to that category. Timestamp features were removed, because they can't be used in feed forward neural network. Designing a recurrent neural network is out of scope for this course. However, timestamps could have been used for generating new features.

The second phase (the file *h2_create_data_sets.py*) was splitting the data to training and testing parts. The category data is very unbalanced with normal network data having the major portion (about 90 %). Some attack categories have very little data points. Data can't be split to have for example 10 % for testing and 90 % per for training per category, because the categories would still be very unbalanced.

1. Read preprocessed data from HDF5
2. Split data to non-overlapping training and testing data sets for NN and RF models
3. Find feature importances for both models with random forest
4. Select most important features
5. Save data sets to HDF5

The solution is to undersample the data by selecting much smaller sets of data per category. Because we have two models (random forest and neural network) in the system, the data has to also be split for each one separately. Either of the models must not be trained with each other's testing data, because that way the results wouldn't proof anything about the system's ability to classify unseen data and generalize.

The random forest has only two target classes, but ideally we should have data points from each attack category so that we can test the "attack or not" categorization with all kinds of data. Neural network has much smaller data sets per category, because the category unbalance has very drastic effects on classification results. Figure 3 illustrates how data is split between random forest and neural network.

Neural network training data has at maximum 5000 data points or 90 % of the category size. For the random forest the maximum is 31000. These limits were determined by hand just looking at the category sizes. Oversampling by copying the data was observed to lead worse results.

The next step is the feature reduction. The data has this point 294 features after vectorization. ExtraTreesClassifier was used to select 10 most important features for the "attack or not" (random forest) classifier and 25 most important features for the neural network. Figures 4 and 5 illustrate which features have greater importance in both classification tasks.

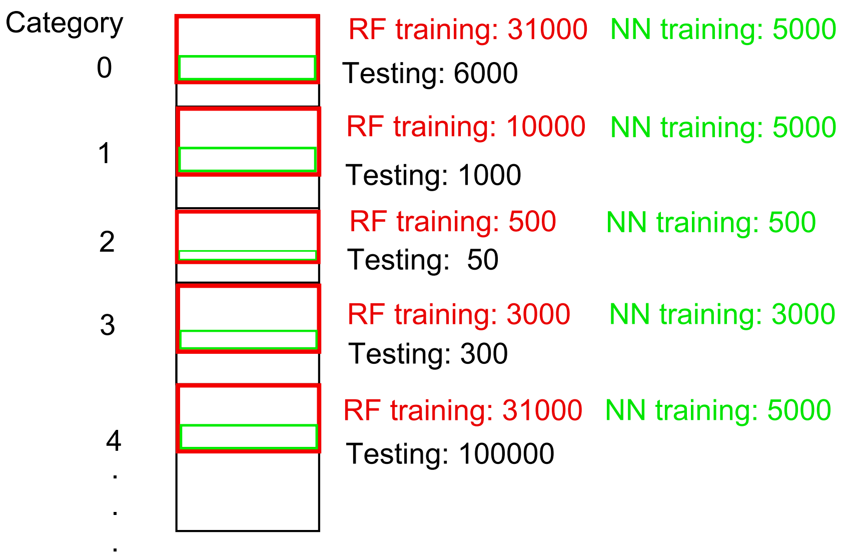


Figure 3: Splitting of each category of data to training and testing data for both random forest (RF) and the neural network (NN). Data point values are for illustration purposes only.

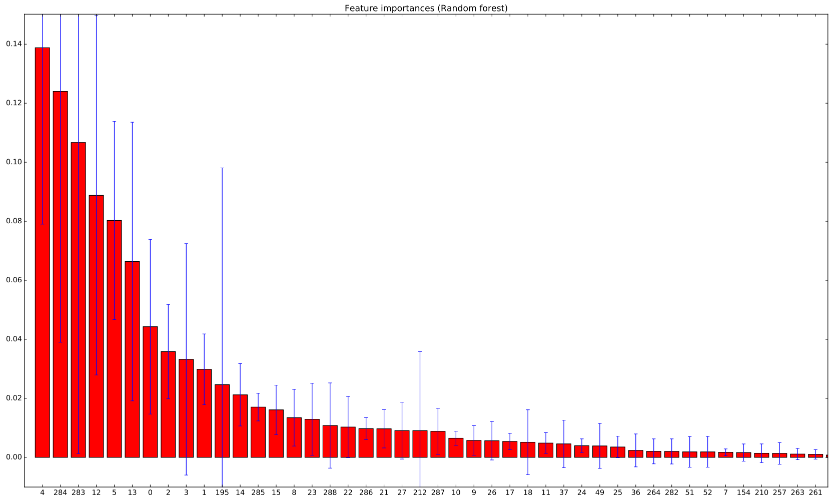


Figure 4: Feature importances for ”attack or not” (random forest) classifier. X axis values are feature indices.

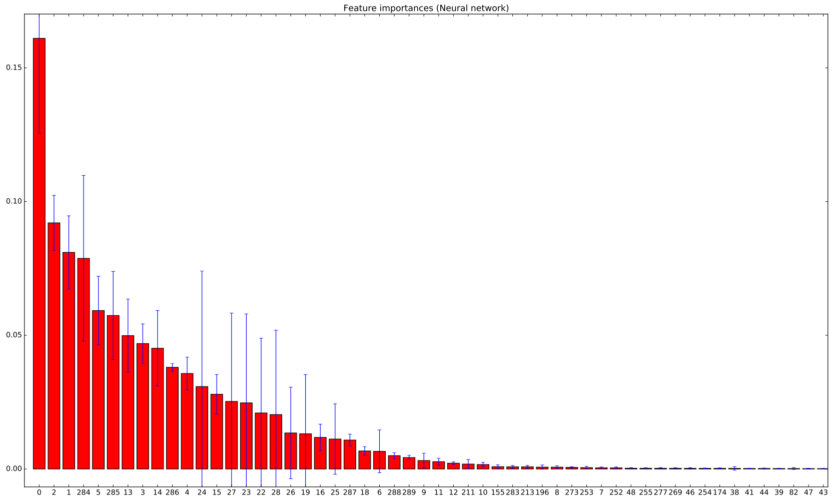


Figure 5: Feature importances for category (neural network) classifier. X axis values are feature indices.

Figure 5 shows how feature number 0 has much greater importance in attack classification compared to the others. The feature is the ”attack or not” label, so it makes sense that it can separate normal network data from other attack categories.

The resulting data sets were then saved to a HDF5 file.

4.3 Random Forest Classifier

For the attack detection case Scikit-Learn’s ExtraTreesClassifier was used with 31 estimators (decision trees) and the quality of a decision tree’s split was measured with the information gain (”entropy”). Other settings were left as default. Number of estimators and the criterion were selected by hand.

4.4 Neural Network Classifier

The neural network architecture is described in Table 1. This architecture was refined by trial and error. The input layer has the same dimensions (number of neurons) as the number of features in the input vector. Different activation functions, like tanh and ReLU were experimented, but sigmoid had always the best results.

Dense layers are fully connected layers. Fairly large number of neurons resulted to better classification results. Different deep residual layer techniques, bypasses between layers and bottlenecking for better generalization were experimented, but none of them resulted to noticably better classifications.

Table 1: Neural network architecture. Dimensions is the number of neurons

Layer	Dimensions	Activation	Dropout
Input	25	-	-
Dense	1250 (50*25)	Sigmoid	0.75
Dense	500 (20*25)	Sigmoid	0.5
Dense	25	Sigmoid	0.1
Output	10	Softmax	-

The output layer has softmax as the activation function, because the sample labels are categorical. The dimensions are the same as the number of different classes. The dropout is a regularization technique, which randomly shuts down a percentage of neurons (sets them to zero) [5]. It is used to prevent overfitting.

Because the classified data was categorial, the selected loss function was ″categorical cross-entropy″. Interestingly the usual ″go to″ optimizer, stochastic gradient descend optimization algorithm (SGD), performed very badly. Adaptive Moment Estimation (Adam) was the chosen method. The network was trained in 128 samples′ batches.

The neural network was run for 100 epochs, during which the learning converged fairly quickly after a few epochs as seen in Figure 6. This could be due to the optimizer getting stuck to a local minimum.

5 Results

This chapter describes the results for both the attack detection and attack categorization cases.

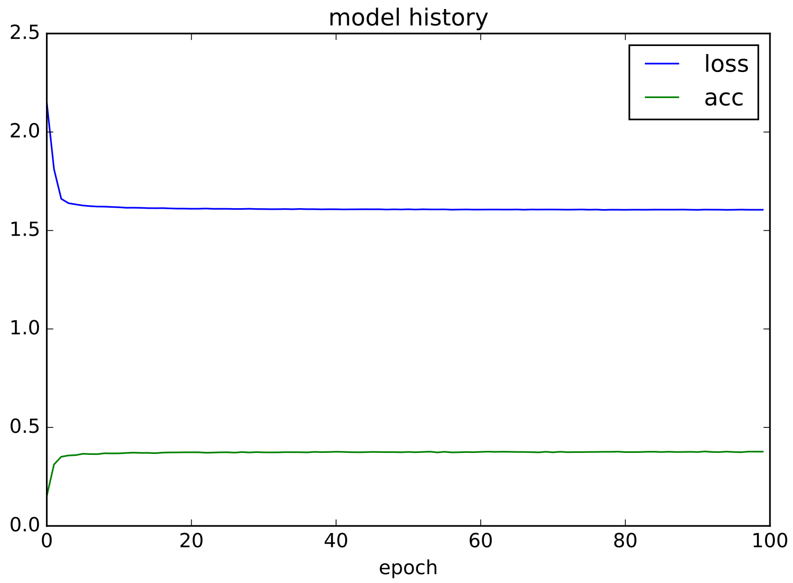


Figure 6: Neural networks training epochs. Loss is categorical cross-entropy, accuracy is overall accuracy for the training samples.

5.1 ”Attack or not” Classification Case

The classification results are in Table 2 and Figure 7. The class 0 means that the sample is not an attack. The class 1 is an attack.

Table 2: Class specific results for attack categorization.

Class	Precision	Recall	F1-score	Support
0	1.00	0.98	0.99	2187456
1	0.88	1.00	0.93	290263
avg / total	0.99	0.98	0.98	2477719

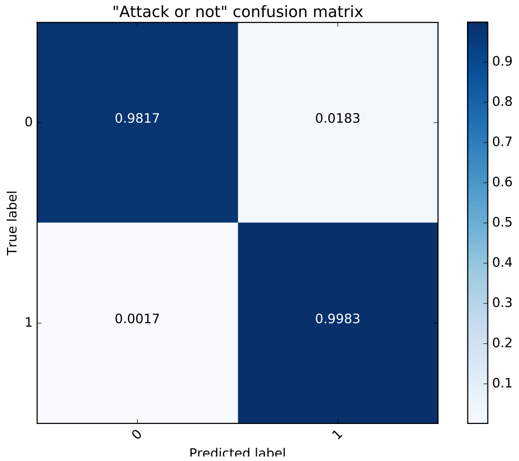


Figure 7: Confusion matrix for attack detection.

The results suggest that the ExtraTreesClassifiers works very well with this data. The scores improved even more after feature reduction with the same classifier. Recall scores are very good (0.98 for class 0 and 1.00 for class 1). Precision for class 1 is lower (0.88).

5.2 Attack Categorization Case

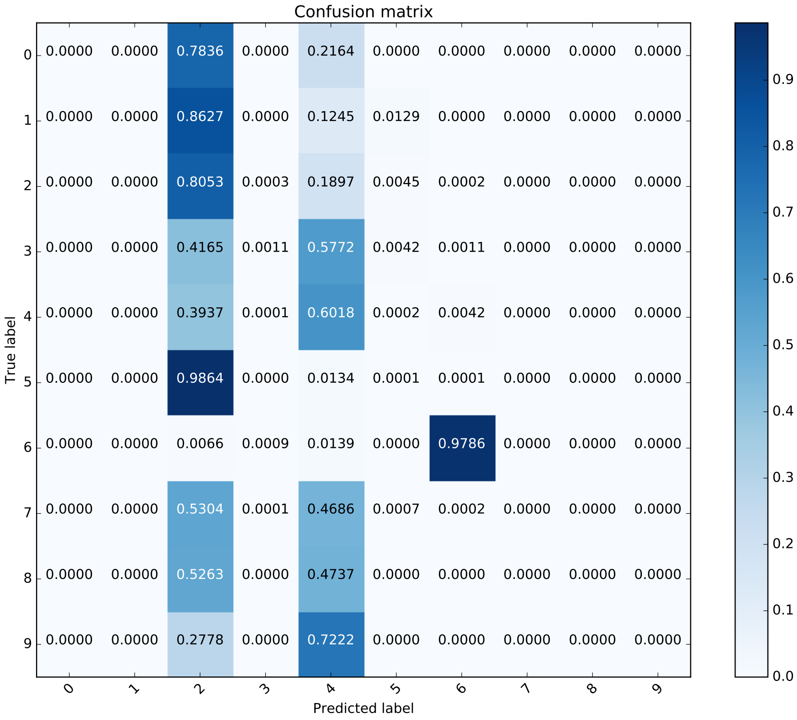
The classification results are in Table 3 and Figure 8. The attack classes are represented with numbers 0–9. Number 6 is a ″normal″ data point and rest of the numbers are different attack categories.

The results suggest that the neural network has learnt to classify samples mostly to the classes 2, 4 and 6. The class 6 is almost always predicted correctly. The precision is approx. 1.0 and recall is approx. 0.98. The confusion matrix shows only little overlapping. This suggests that there are strongly differentiating features present for the class 6 (normal data). The generated ″attack or not″ feature is the cause, as can be seen in Figure 5.

Differentiation between different attack categories is much worse. The confusion matrix shows that predictions fall mostly under classes 2 and 4. Therefore instances of the classes 2 and 4 have quite high recall score (0.81 for class 2 and 0.60 for class 4). Most of the data points are predicted to be instances of class 2, so the precision is low. Especially class 5 is very similar to class 2.

Table 3: Class specific results for attack categorization.

Class	Precision	Recall	F1-score	Support
0	0.00	0.00	0.00	268
1	0.00	0.00	0.00	233
2	**0.04**	**0.81**	**0.07**	11353
3	**0.02**	0.00	0.00	39525
4	**0.16**	**0.60**	**0.25**	19246
5	**0.06**	0.00	0.00	210481
6	**1.00**	**0.98**	**0.99**	2187456
7	0.00	0.00	0.00	8987
8	0.00	0.00	0.00	152
9	0.00	0.00	0.00	18
avg / total	0.89	0.87	0.88	2477719



	0	1	2	3	4	5	6	7	8	9
0	0.0000	0.0000	0.7836	0.0000	0.2164	0.0000	0.0000	0.0000	0.0000	0.0000
1	0.0000	0.0000	0.8627	0.0000	0.1245	0.0129	0.0000	0.0000	0.0000	0.0000
2	0.0000	0.0000	0.8053	0.0003	0.1897	0.0045	0.0002	0.0000	0.0000	0.0000
3	0.0000	0.0000	0.4165	0.0011	0.5772	0.0042	0.0011	0.0000	0.0000	0.0000
4	0.0000	0.0000	0.3937	0.0001	0.6018	0.0002	0.0042	0.0000	0.0000	0.0000
5	0.0000	0.0000	0.9864	0.0000	0.0134	0.0001	0.0001	0.0000	0.0000	0.0000
6	0.0000	0.0000	0.0066	0.0009	0.0139	0.0000	0.9786	0.0000	0.0000	0.0000
7	0.0000	0.0000	0.5304	0.0001	0.4686	0.0007	0.0002	0.0000	0.0000	0.0000
8	0.0000	0.0000	0.5263	0.0000	0.4737	0.0000	0.0000	0.0000	0.0000	0.0000
9	0.0000	0.0000	0.2778	0.0000	0.7222	0.0000	0.0000	0.0000	0.0000	0.0000

Figure 8: Confusion matrix for attack categorization.

Changing the neural network architecture had an impact on which classes the data points were classified to. However, the overall classification accuracy stayed mostly the same. The neural network seemed to learn to classify attack instances always to two classes, with some variation on which two classes. Different deep residual layer techniques, bypasses between layers and bottlenecking for better generalization were experimented but they didn't have any meaningful positive impact to the overall accuracy.

6 Conclusions

A system for classifying UNSW-NB15 data set samples was developed using a random forest and feed-forward neural network. The data set was split to training and testing sets. The created system relied on the random forest to classify data as normal network data or an attack. This information was then used to train a neural network to further classify the attack data to different attack categories.

The results for attack detection were very good with approx. 0.88 precision for attacks and nearly 1.0 precision for normal data samples.

Attack categorization had problems in differentiating between attack classes and could mostly classify the attacks to two different classes. However, it could accurately classify normal network data.

Development had many common problems in machine learning. Large data amount, lack of memory and unbalance between class sizes were most common problems. Especially unbalanced data made it difficult to generalize data classes. Most development time was spent on fixing these problems.

Further experimenting could have been done on tuning the hyper parameters automatically. Some timestamp features were discarded in the beginning. They could have been used to create new features that could have changed the outcome. Another feature reduction schemes could have been tried, like principal component analysis. Using an autoencoder neural network to produce more data was considered, but it was decided to be out of scope for this work. Final experimentation showed that the results of developed method were equally as good or slightly better than the ones that could be achieved with just a random forest (Scikit-Learn's ExtraTreesClassifier) using same data sets and features.

References

1. Simon Haykin, Neural Networks - A Comprehensive Foundation, Second Edition, Pearson Education Inc, 1999.
2. Heikki Huttunen, ”SGN-41006 Signal Interpretation Methods lecture slides”, Tampere University of Technology, Spring 2016.
3. Dumitru Erhan et al., ”Why Does Unsupervised Pre-training Help Deep Learning?”, Journal of Machine Learning Research 11, 2010, Available at: http://www.jmlr.org/papers/volume11/erhan10a/erhan10a.pdf
4. Glorot, Bordes and Bengio, ”Deep sparse rectifier neural networks”, Available at: http://www.jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf
5. Srivastava et al., ”Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, Available at: https://www.cs.toronto.edu/ hinton/absps/JMLRdropout.pdf
6. Moustafa, Nour, and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)."Military Communications and Information Systems Conference (MilCIS), 2015. IEEE, 2015.

APPENDIX

Listing 1: h2_preprocess.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jan 29 16:52:20 2017

@author: Samuli
"""


import numpy as np
import pandas

from sklearn.preprocessing import normalize # ,
    LabelEncoder

# Read the info about features
print('Reading feature info...')
data_info = pandas.read_csv("NUSW-NB15_features.csv",
    encoding = "ISO-8859-1", header=None).values
features = data_info[1:-2,:]
feature_names = features[:, 1]  # Names of the features in
    a list
feature_types = np.array([item.lower() for item in features
    [:, 2]])  # The types of the corresponding features in
    'features_names'

# index arrays for different types of features
print('Finding column indices for feature types...')
nominal_cols = np.where(feature_types == "nominal")[0]
integer_cols = np.where(feature_types == "integer")[0]
binary_cols = np.where(feature_types == "binary")[0]
float_cols = np.where(feature_types == "float")[0]

# arrays for names of the different types of features
nominal_names = feature_names[nominal_cols]
integer_names = feature_names[integer_cols]
binary_names = feature_names[binary_cols]
float_names = feature_names[float_cols]

print('Reading csv files...')
dataframe1 = pandas.read_csv("UNSW-NB15_1.csv", header=None
    )
dataframe2 = pandas.read_csv("UNSW-NB15_2.csv", header=None
   )
dataframe3 = pandas.read_csv("UNSW-NB15_3.csv", header=None
   )
dataframe4 = pandas.read_csv("UNSW-NB15_4.csv", header=None
   )

print(’Concatenating...’)
dataframe = pandas.concat([dataframe1, dataframe2,
   dataframe3, dataframe4])

del dataframe1
del dataframe2
del dataframe3
del dataframe4

print(’Preprocessing...’)
print(’Converting data...’)
dataframe[integer_cols] = dataframe[integer_cols].
   convert_objects(convert_numeric=True)
dataframe[binary_cols] = dataframe[binary_cols].
   convert_objects(convert_numeric=True)
dataframe[float_cols] = dataframe[float_cols].
   convert_objects(convert_numeric=True)
dataframe[48] = dataframe[48].convert_objects(
   convert_numeric=True)
#dataframe[nominal_cols] = dataframe[nominal_cols].astype(
   str)

print(’Replacing NaNs...’)
dataframe.loc[:,47] = dataframe.loc[:,47].replace(np.nan,’
   normal’, regex=True).apply(lambda x: x.strip().lower())
dataframe.loc[:,binary_cols] = dataframe.loc[:,binary_cols
   ].replace(np.nan, 0, regex=True)
dataframe.loc[:,37:39] = dataframe.loc[:,37:39].replace(np.
   nan, 0, regex=True)
# dataframe.loc[:,float_cols] = dataframe.loc[:,float_cols
   ].replace(np.nan, 0, regex=True)

print(’Stripping nominal columns and setting them lower
   case...’)
dataframe.loc[:,nominal_cols] = dataframe.loc[:,
   nominal_cols].applymap(lambda x: x.strip().lower())

print(’Changing targets \’backdoors\’ to \’backdoor\’...’)
dataframe.loc[:,47] = dataframe.loc[:,47].replace(’
   backdoors’,’backdoor’, regex=True).apply(lambda x: x.
   strip().lower())
```

```
dataset = dataframe.values

del dataframe

# Subsets of the dataset which have data that is only of
    the corresponding data type (nominal, integer etc)
# Columns don't include the target classes (the two last
    columns of the dataset)
print('Slicing dataset...')
nominal_x = dataset[:, nominal_cols][:,:]
integer_x = dataset[:, integer_cols][:,:].astype(np.float32
    )
binary_x = dataset[:, binary_cols][:,:].astype(np.float32)
float_x = dataset[:, float_cols][:,:].astype(np.float32)
# aon_x = dataset[:, 48][np.newaxis,:].astype(np.float32).
    transpose()  # Attack or not (binary)

# Make nominal (textual) data binary vectors
print('Vectorizing nominal data...')
from sklearn.feature_extraction import DictVectorizer
v = DictVectorizer(sparse=False)
# D = [{'foo': 1, 'bar': 2}, {'foo': 3, 'baz': 1}]
D = map(lambda dataline: dict(zip(nominal_names, dataline))
    , nominal_x)
labeled_nominal_x = v.fit_transform(D).astype(np.float32)
del nominal_x

print('Concatenating X...')
X = np.concatenate((integer_x, labeled_nominal_x, float_x,
    binary_x), axis=1)

del integer_x
del labeled_nominal_x
del float_x
del binary_x

# Find rows that have NaNs
print('Removing NaNs if any...')
nan_indices = []
for feature_i in range(X.shape[1]):
    nan_indices.extend(list(np.where(np.isnan(X[:,
        feature_i]))[0]))
nan_indices = np.unique(nan_indices)

# Remove rows that have NaNs
X_no_nans = np.delete(X, nan_indices, axis=0)
```

```
del X

print('Normalizing X...')
normalized_X = normalize(X_no_nans, copy=False)

del X_no_nans


data_dim = normalized_X.shape
print('Data dimensions are', data_dim)

print('Creating target Y matrix...')
Y = np.delete(dataset[:, -2], nan_indices)
Y_A = np.delete(dataset[:, -1], nan_indices).astype(np.
    int16) # Is attack or not

del dataset

'''
# Remove same rows as in X to have correct y's
Y_no_nans = np.delete(Y, nan_indices, axis=0)
'''
print('Vectorizing Y labels...')
D = [{'attack_cat': y} for y in Y]
labeled_Y = v.fit_transform(D)

del D

print('Saving normalized X and labeled Y to HDF5')
import h5py
h5f = h5py.File('data.h5', 'w')
h5f.create_dataset('normalized_X', data=normalized_X)
h5f.create_dataset('labeled_Y', data=labeled_Y)
dt = h5py.special_dtype(vlen=str)
h5f.create_dataset('Y', data=Y, dtype=dt)
h5f.create_dataset('Y_A', data=Y_A)
h5f.close()

del Y
del normalized_X
del labeled_Y
```

Listing 2: h2_create_data_sets.py

```
# -*- coding: utf-8 -*-
```

```
"""
Created on Sun Feb 12 14:05:11 2017

@author: Samuli
"""

import numpy as np

print('Loading normalized data from HDF5...')
import h5py
h5f = h5py.File('data.h5', 'r')
normalized_X = h5f['normalized_X'].value
labeled_Y = h5f['labeled_Y'].value
Y = h5f['Y'].value
Y_A = h5f['Y_A'].value
h5f.close()


print('Splitting X to test and train datasets...')
# X_test, X_train, Y_test, Y_train = []
rf_normal_inds = []
nn_normal_inds = []
inds = []

attack_cats = np.unique(Y)
cat_sizes = []
np.random.seed(1337)
max_training_samples = 5000
print('Select max', max_training_samples, 'samples for
    training...')
for cat in attack_cats:
    indices = np.ix_(Y == cat)[0]
    # total_num_of_samples = indices.shape[0]
    np.random.shuffle(indices)
    if cat == 'normal':
        len_of_subset = min(np.floor(len(indices)*0.9),
            31000)
        rf_normal_inds = indices[:len_of_subset]
        nn_normal_inds = indices[:len_of_subset][:
            max_training_samples]
        cat_size = len(indices)
        print(cat, ': training samples =', len_of_subset, '
            | total samples =', cat_size)
    else:
        len_of_subset = min(np.floor(len(indices)*0.9),
            max_training_samples)
        cat_size = len(indices)
        print(cat, ': training samples =', len_of_subset, '
            | total samples =', cat_size)
        cat_sizes.append(cat_size)
        #inds.extend(oversampled_indices)
        inds.extend(indices[:int(len_of_subset)])


print('Number of categories is', len(cat_sizes), '| Total
    samples in categories:\n|', '\n|'.join([str(i)+': '+str
    (c) for i, c in enumerate(cat_sizes)]))
print('normal samples for rf:', len(rf_normal_inds))
print('normal samples for nn:', len(nn_normal_inds))

# Attack or not learning data
rf_inds = []
rf_inds.extend(inds)
rf_inds.extend(rf_normal_inds)
X_rf_train = normalized_X[rf_inds, :]
X_rf_test = np.delete(normalized_X, rf_inds, axis=0)

Y_rf_train = Y_A[rf_inds]
Y_rf_test = np.delete(Y_A, rf_inds, axis=0)

# Category learning data
nn_inds = []
nn_inds.extend(inds)
nn_inds.extend(nn_normal_inds)
X_nn_train = normalized_X[nn_inds, :]
# Remove rf indices because nn indices is a subset and we
    dont want to test and train with same data
X_nn_test = np.delete(normalized_X, rf_inds, axis=0)

del normalized_X

Y_nn_train = labeled_Y[nn_inds]
Y_nn_train_string = Y[nn_inds]

Y_nn_test = np.delete(labeled_Y, rf_inds, axis=0)
Y_nn_test_string = np.delete(Y, rf_inds, axis=0)

del labeled_Y
del Y

Y_nn_A_train = Y_A[nn_inds]
Y_nn_A_test = np.delete(Y_A, rf_inds, axis=0)
```

```
del Y_A

print('Finding feature importances with
   ExtraTreesClassifier')
from sklearn.ensemble import ExtraTreesClassifier

def find_importances(X_train, Y_train):
    model = ExtraTreesClassifier()
    model = model.fit(X_train, Y_train)

    importances = model.feature_importances_
    std = np.std([tree.feature_importances_ for tree in
       model.estimators_],
                 axis=0)
    indices = np.argsort(importances)[::-1]  # Top ranking
       features' indices
    return importances, indices, std

import matplotlib.pyplot as plt
# Plot the feature importances of the forest
def plot_feature_importances(X_train, importances, indices,
    std, title):
    plt.figure()
    plt.title(title)
    plt.bar(range(X_train.shape[1]), importances[indices],
           color="r", yerr=std[indices], align="center")
    plt.xticks(range(X_train.shape[1]), indices)
    plt.xlim([-1, X_train.shape[1]])
    plt.show()



rf_importances, rf_indices, rf_std = find_importances(
   X_rf_train, Y_rf_train)

plot_feature_importances(X_rf_train, rf_importances,
   rf_indices, rf_std, title='Feature importances (Random
   forest)')

# Neural network is classified with correct 'attack or not'
    labels
X_nn_train = np.concatenate((Y_nn_A_train[:,np.newaxis],
   X_nn_train), axis=1)
nn_importances, nn_indices, nn_std = find_importances(
   X_nn_train,
                                                      Y_nn_train
                                                         )
```

```
plot_feature_importances(X_nn_train,
                        nn_importances, nn_indices, nn_std,
                            title='Feature importances (
                           Neural network)')

NB_RF_FEATURES = 10
NB_NN_FEATURES = 25

reduced_X_nn_train = X_nn_train[:, nn_indices[0:
   NB_NN_FEATURES]]
reduced_Y_nn_train = Y_nn_train
reduced_Y_nn_train_string = Y_nn_train_string
reduced_Y_nn_test_string = Y_nn_test_string
reduced_Y_nn_A_train = Y_nn_A_train

# Test set has 1 less because we get the one from RF

reduced_X_nn_test = X_nn_test[:, nn_indices[1:
   NB_NN_FEATURES]]
reduced_Y_nn_test = Y_nn_test


reduced_X_rf_train = X_rf_train[:, rf_indices[0:
   NB_RF_FEATURES]]
reduced_Y_rf_train = Y_rf_train

reduced_X_rf_test = X_rf_test[:, rf_indices[0:
   NB_RF_FEATURES]]
reduced_Y_rf_test = Y_rf_test



print('Saving X and Y to HDF5')
import h5py
h5f = h5py.File('datasets.h5', 'w')
h5f.create_dataset('X_rf_train', data=reduced_X_rf_train)
h5f.create_dataset('X_rf_test',  data=reduced_X_rf_test)
h5f.create_dataset('Y_rf_train', data=reduced_Y_rf_train)
h5f.create_dataset('Y_rf_test',  data=reduced_Y_rf_test)

h5f.create_dataset('X_nn_train', data=reduced_X_nn_train)
h5f.create_dataset('X_nn_test',  data=reduced_X_nn_test)
h5f.create_dataset('Y_nn_train', data=reduced_Y_nn_train)
h5f.create_dataset('Y_nn_test',  data=reduced_Y_nn_test)
dt = h5py.special_dtype(vlen=str)
h5f.create_dataset('Y_nn_train_string', data=
   reduced_Y_nn_train_string, dtype=dt)
```

```
h5f.create_dataset('Y_nn_test_string', data=
   reduced_Y_nn_test_string, dtype=dt)

h5f.create_dataset('Y_nn_A_train', data=
   reduced_Y_nn_A_train)
h5f.create_dataset('Y_nn_A_test', data=Y_nn_A_test)

h5f.close()
```

Listing 3: h2_fit_neural.py

```
# -*- coding: utf-8 -*-
"""
Created on Sat Feb 11 11:24:58 2017

@author: Samuli
"""
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Input,
    merge, Lambda
from keras.models import Model
from keras.optimizers import SGD
import numpy as np

from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from keras.regularizers import l2, activity_l2

from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

import confusion_matrix as cm
import roc

print('Loading normalized data from HDF5...')
import h5py
h5f = h5py.File('datasets.h5', 'r')
X_train = h5f['X_nn_train'].value
Y_train = h5f['Y_nn_train'].value.astype(np.float32)
X_test = h5f['X_nn_test'].value
Y_test = h5f['Y_nn_test'].value.astype(np.float32)

X_train2 = h5f['X_rf_train'].value
Y_train2 = h5f['Y_rf_train'].value.astype(np.float32)
```

```
X_test2 = h5f['X_rf_test'].value
Y_test2 = h5f['Y_rf_test'].value.astype(np.float32)
h5f.close()


from sklearn.ensemble import ExtraTreesClassifier

print('Training ExtraTreesClassifier for "attack or not" labels...')
model2 = ExtraTreesClassifier(n_estimators=31, criterion='entropy')
model2 = model2.fit(X_train2, Y_train2)

Y_pred2 = model2.predict_proba(X_test2)[:,1]

print('Testing accuracy...')
score2 = accuracy_score(Y_test2, np.around(Y_pred2))
print(score2)
print(classification_report(Y_test2, np.around(Y_pred2)))

def perf_measure(y_actual, y_hat):
    TP = 0
    FP = 0
    TN = 0
    FN = 0

    for i in range(len(y_hat)):
        if y_actual[i]==y_hat[i]==1:
           TP += 1
    for i in range(len(y_hat)):
        if y_hat[i]==1 and y_actual[i]!=y_hat[i]:
           FP += 1
    for i in range(len(y_hat)):
        if y_actual[i]==y_hat[i]==0:
           TN += 1
    for i in range(len(y_hat)):
        if y_hat[i]==0 and y_actual[i]!=y_hat[i]:
           FN += 1

    return(TP, FP, TN, FN)

TP, FP, TN, FN = perf_measure(np.around(Y_pred2), Y_test2)

fp_rate = FP/(TN+FP)
tn_rate = TN/(TN+FP)

accuracy = (TN+TP)/(TN+FP+TP+TN)
```

```
precision = TP/(TN+FP)
hitrate = TP/(TN+FN)

print('TP:', TP, 'FP:', FP, 'TN:', TN, 'FN:', FN)
print('Accuracy:', accuracy)
print('False Positive rate:', fp_rate, 'True Negative Rate'
    , tn_rate)

def to_cat(y):
    y_tmp = np.ndarray(shape=(y.shape[0], 2), dtype=np.
        float32)
    for i in range(y.shape[0]):
        y_tmp[i, :] = np.array([1-y[i], y[i]])    # np.array
            ([0,1]) if y[i] else np.array([1,0])
    return y_tmp

cm.plot_confusion_matrix(Y_test2, np.round(Y_pred2),
    classes=list(range(2)),
                            normalize=True,
                            title='"Attack or not" confusion
                                matrix')
roc.plot_roc_curve(to_cat(Y_test2), to_cat(Y_pred2), 2, 0,
    title='Receiver operating characteristic (attack_or_not
     = 0)')
roc.plot_roc_curve(to_cat(Y_test2), to_cat(Y_pred2), 2, 1,
    title='Receiver operating characteristic (attack_or_not
     = 1)')


print('Combining predicted "attack or not" labels to neural
    network testing data...')
X_test = np.concatenate((Y_pred2[:,np.newaxis], X_test),
    axis=1)

print('Creating neural network...')
num_of_features = X_train.shape[1]
nb_classes = Y_train.shape[1]


def residual_layer(size, x):

    y = Dense(size, activation='sigmoid', W_regularizer=l2
        (0.01), activity_regularizer=activity_l2(0.01))(x)
    # x = Dropout(0.5)(x)
    # print(x.get_shape().as_list()[1])
    y = Dense(x.get_shape().as_list()[1], activation='
        sigmoid',  W_regularizer=l2(0.01),
       activity_regularizer=activity_l2(0.01))(y)
    res = merge([y, x], mode='sum')
    return res


def baseline_model():
    def branch2(x):

        x = Dense(np.floor(num_of_features*50), activation=
           'sigmoid')(x)
        x = Dropout(0.75)(x)

        x = Dense(np.floor(num_of_features*20), activation=
           'sigmoid')(x)
        x = Dropout(0.5)(x)

        x = Dense(np.floor(num_of_features), activation='
           sigmoid')(x)
        x = Dropout(0.1)(x)
        return x

    main_input = Input(shape=(num_of_features,), name='
       main_input')

    x = main_input
    x = branch2(x)
    main_output = Dense(nb_classes, activation='softmax')(x
       )
    model = Model(input=main_input, output=main_output)
    model.compile(loss='categorical_crossentropy',
       optimizer='adam', metrics=['accuracy', '
       categorical_crossentropy'])
    return model

model = baseline_model()

print('Training neural network...')
history = model.fit(X_train, Y_train,
                    nb_epoch=100,
                    batch_size=128
                    )

print('Plotting training history data...')
print(history.history.keys())

from epoch_history_plot import plot_hist
```

```
plot_hist(history, [’loss’, ’acc’])

# summarize history for accuracy
import matplotlib.pyplot as plt
plt.figure()
plt.plot(history.history[’acc’])
# plt.plot(history.history[’val_acc’])
plt.title(’model accuracy’)
plt.ylabel(’accuracy’)
plt.xlabel(’epoch’)
#plt.legend([’train’, ’test’], loc=’upper left’)
plt.show()
plt.figure()
# summarize history for loss
plt.plot(history.history[’loss’])
# plt.plot(history.history[’val_loss’])
plt.title(’model loss’)
plt.ylabel(’loss’)
plt.xlabel(’epoch’)
#plt.legend([’train’, ’test’], loc=’upper left’)
plt.show()

print(’Testing neural network...’)
Y_predicted = model.predict(X_test)

max_probs = np.argmax(Y_predicted, axis=1)
Y_pred = np.zeros(Y_predicted.shape)
for row, col in enumerate(max_probs):
    Y_pred[row,col] = 1

score = accuracy_score(Y_test, Y_pred)
print(score)
print(classification_report(Y_test.argmax(axis=-1), Y_pred.
   argmax(axis=-1)))


cm.plot_confusion_matrix(Y_test.argmax(axis=-1), Y_pred.
   argmax(axis=-1), classes=list(range(10)),
                          normalize=True,
                          title=’Confusion matrix’)

print(’Saving neural network model...’)
json_string = model.to_json()
with open(’neural_model1.json’, ’w’) as f:
    f.write(json_string)
model.save_weights(’neural_model_weights1.h5’)
model.save('neural_model1.h5')

roc.plot_roc_curve(Y_test, Y_predicted, nb_classes, 6,
   title='Receiver operating characteristic (class 6)')
roc.plot_roc_curve(Y_test, Y_predicted, nb_classes, 4,
   title='Receiver operating characteristic (class 4)')
roc.plot_roc_curve(Y_test, Y_predicted, nb_classes, 2,
   title='Receiver operating characteristic (class 2)')
roc.plot_roc_curve(Y_test, Y_predicted, nb_classes, 0,
   title='Receiver operating characteristic (class 0)')


model3 = ExtraTreesClassifier(n_estimators=5, criterion='
   entropy')
print('Fitting...')
model3 = model2.fit(X_train, Y_train.argmax(axis=-1))
print('Predicting...')
Y_predicted3 = model3.predict(X_test)

print('Testing accuracy...')
score3 = accuracy_score(Y_test.argmax(axis=-1),
   Y_predicted3)
print(score3)
print(classification_report(Y_test.argmax(axis=-1),
   Y_predicted3))

cm.plot_confusion_matrix(Y_test.argmax(axis=-1),
   Y_predicted3, classes=list(range(10)),
                          normalize=True,
                          title='Extratrees Confusion
                             matrix')


'''
print('Saving X and Y to HDF5')

h5f = h5py.File('results.h5', 'w')
h5f.create_dataset('Y_predicted', data=Y_pred)
h5f.create_dataset('Y_expected', data=Y_test)
h5f.close()
'''
```

Listing 4: confusion_matrix.py

```
import itertools
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(y_test, y_pred, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True
       `.
    """
    plt.figure()
    cm = confusion_matrix(y_test, y_pred)
    # np.set_printoptions(precision=2)


    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.
           newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range
       (cm.shape[1])):
        plt.text(j, i, "{0:.4f}".format(cm[i, j]),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "
                    black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

Listing 5: epoch_history_plot.py

```
# -*- coding: utf-8 -*-

import matplotlib.pyplot as plt

def plot_hist(hist, metrics, num=None):
    if num is not None:
        fig = plt.figure(num)
    else:
        fig = plt.figure()
    ax1 = fig.add_subplot(111)
    for metric in metrics:
        ax1.plot(hist.history[metric], label=metric)
    # plt.plot(history.history['val_acc'])
    # ax.ylabel(metric)
    plt.title('model history')
    plt.xlabel('epoch')
    #plt.legend(['train', 'test'], loc='upper left')
    plt.legend(loc='upper right');
    plt.show()
```

Listing 6: roc.py

```

from sklearn import metrics
import matplotlib.pyplot as plt

def plot_roc_curve(Y_test, Y_pred, nb_classes, class_index,
    title='Receiver operating characteristic'):

    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(nb_classes):
        fpr[i], tpr[i], _ = metrics.roc_curve(Y_test[:, i],
            Y_pred[:, i])
        roc_auc[i] = metrics.auc(fpr[i], tpr[i])

    # Compute micro-average ROC curve and ROC area
    fpr["micro"], tpr["micro"], _ = metrics.roc_curve(
       Y_test.ravel(), Y_pred.ravel())
    roc_auc["micro"] = metrics.auc(fpr["micro"], tpr["micro
       "])

    plt.figure()
```

```
lw = 2
plt.plot(fpr[class_index], tpr[class_index], color='
   darkorange',
         lw=lw, label='ROC curve (area = %0.4f)' %
            roc_auc[class_index])
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle
   ='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(title)
plt.legend(loc="lower right")
plt.show()
```