Detection of Attacks on Power System Grids.

Stanislav Modrak[1]

1 University of Southampton

Introduction

In this coursework, we were asked to develop ML models for detecting attacks in power system grids based on analysis of system traces from various cyber-physical components of power system grids. A system trace contained features such as various phasor measurement unit readings, Snort intrusion detection system logs, logs of power generators, and logs of IEDs (Intelligent Electronic Devices) of relays and breakers.

Two models were to be developed, one for a binary classification task (part A) of detecting a particular trace to be a *normal event* or an *attack event* and another for a multi-class classification task (part B) of detecting a particular trace to be a *normal event* or an *attack event*, and if an attack event, whether it is *data injection attack* or *command injection attack*.

To facilitate this two labelled versions of the dataset were provided. `TrainingDataBinary.csv` (6000 samples) and `TrainingDataMulti.csv` (6000 samples) training sets, for binary and multi-class classification respectively. Unlabelled `TestingDataBinary.csv` and `TestingDataMulti.csv` test sets were provided to make predictions on using the final models and submit for later evaluation by a marker.

Method

Both *binary* (assignment part A) and *multi-class* (assignment part B) classification ML model development followed the same workflow. The development was done interactively using the Jupyter Lab environment hosted on the University of Southampton high-performance computing (HPC) IRIDIS 5 compute cluster. Jupyter Server instance was scheduled to run on the compute cluster each time work was undertaken. An `ssh` port forwarding connection over GlobalProtect VPN was made to enable local (from a personal workstation) GUI access.

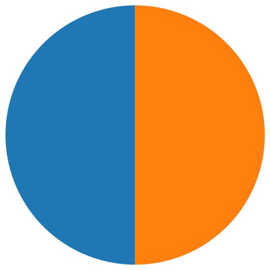
The technologies selected were Keras (TensorFlow) and scikit-learn for the ML development and Pandas for the data engineering. SciKeras wrapper for Keras was used to make Keras compatible with scikit-learn pipelines and interchangeable with its own estimators. These technologies were selected based on personal familiarity, the quality of documentation available and the technology's maturity.

EDA

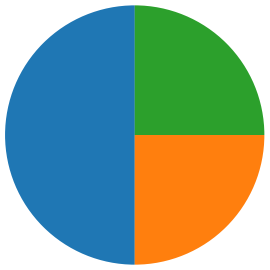
The development of the models was initiated by an EDA (exploratory data analysis) of the data available. The binary classification training set from `TrainingDataBinary.csv` was found to be perfectly balanced (50/50 distribution), and `TrainingDataMulti.csv` nearly balanced. The class distribution plots can be seen in figure 1. `TrainingDataMulti.csv` consisted of 50 % *normal event* and 25 % each *data injection attack* and *command injection attack* classes. The imbalanced-learn library for working with imbalanced data (applying random under-sampling on the majority class *normal event*) was used to resample the dataset to make the class proportions equal, 33.33 % each. However, this did not show to have a substantial effect on improving the classification performance metrics and was therefore abandoned, because throwing away any data is almost always to be avoided.

All features of the data were already numerical and thus require no additional encoding. There were no missing, `NaN` or other undesirable values identified in any of the features.

Figure 1. Class distribution of provided training datasets. On both sub-figures, the *blue* colour represents the *normal event* class. In sub-figure 1a, *orange* represents the *attack event*. In sub-figure 1b, minority classes *data injection attack* and *command injection attack* are represented by colours *orange* and *green*.



(a) Binary classification dataset.



(b) Multi-class classification dataset.

ML Engineering

The scikit-learn `sklearn.pipeline.Pipeline` composite estimator object was used as the high-level implementation wrapper for all the ML pipeline steps needed to produce a final ML model. The `Pipeline` can contain any pre-processing, feature engineering, feature scaling, encoding or other steps and the final ML algorithm estimator object. The advantage of using the scikit-learn pipeline API is the ability to perform an automated hyperparameter search (using `sklearn.model_selection.GridSearchCV` module) over the whole hyperparameter space of all the components of the ML pipeline at once, as well as an automated model selection of the best Ml algorithm amongst a selection of many. Using this method (`GridSearchCV`) enables a systematic approach to find the most performant and optimal model for the task.

The `GridSearchCV` automated hyper-parameter search and model selection tries all combinations of provided Ml algorithms and their hyper-parameters whilst trying to optimise an objective function of choice. The objective function can be any of the classification metrics such as accuracy, F1 score or a combination of them.

For both binary and multi-class classification tasks, I chose the *macro F1* score as the score to maximise, this is because it provides an excellent metric for multi-class classification (especially for unbalanced data) and works just as well as others for the binary classification.

The ML pipeline design (shared for binary and multi-class) was composed of components corresponding to the stages of *feature scaling*, *feature selection*, *dimensionality reduction*, and *classifier algorithm* (in this order). The `GridSearchCV` module performing the automated-hyper parameter search and model selection is provided with a parameter grid declaring which ML algorithms in combination with which hyper-parameters to test. The validation of the fitted candidate model is done using a selected validation method, in my case I chose *4-fold stratified cross-validation*. Before any training was undertaken all datasets were randomly shuffled to prevent any existing order of samples which might have been introduced during their production from affecting the training.

The parameter grid used in this assignment is presented below:

```
# model and corresponding hyper-parameter candidates
param_grid = [
    {  # Logistic Regression
        'feature_scaling': [StandardScaler()],
        'feature_selection': [VarianceThreshold(threshold=0)],
        'classifier': [LogisticRegression(max_iter=5000, n_jobs=CPUs, random_state=42)],
        'classifier__C': [1, 10, 100]
    },
    {  # Decision Tree
        'feature_scaling': [None],
        'feature_selection': [VarianceThreshold(threshold=0)],
        'dimensionality_reduction': [None],
        'classifier': [DecisionTreeClassifier(random_state=42)],
        'classifier__min_impurity_decrease': [0, 0.01],
        'classifier__criterion': ['gini', 'log_loss']
    },
    {  #  K-nearest neighbors
        'feature_scaling': [StandardScaler(), RobustScaler()],
        'feature_selection': [VarianceThreshold(threshold=0)],
        'dimensionality_reduction': [NeighborhoodComponentsAnalysis(random_state=42)],
        'classifier': [KNeighborsClassifier(n_jobs=CPUs)],
        'classifier__n_neighbors': [2, 3, 4, 5, 6, 10],
        'classifier__algorithm': ['auto']
    },
    {  # Support Vector Machine
        'feature_scaling': [RobustScaler()],
        'feature_selection': [VarianceThreshold(threshold=0)],
        'dimensionality_reduction': [None, PCA(n_components='mle', random_state=42)],
        'classifier': [SVC(random_state=42, cache_size=400)],
        'classifier__kernel': ['linear', 'rbf', 'poly','sigmoid'],
        'classifier__degree': [3, 4],
        'classifier__C': [1, 5, 8, 10, 100]
    },
    {  # Random Forest
        'feature_scaling': [None],
        'feature_selection': [VarianceThreshold(threshold=0)],
        'dimensionality_reduction': [None],
        'classifier': [RandomForestClassifier(random_state=42, n_jobs=CPUs)],
    },
    {  # AdaBoost
        'feature_scaling': [None],
        'feature_selection': [VarianceThreshold(threshold=0)],
        'dimensionality_reduction': [None],
        'classifier': [AdaBoostClassifier(random_state=42)],
        'classifier__estimator': [DecisionTreeClassifier(max_depth=1),
                                  DecisionTreeClassifier(max_depth=2)],
    },
    {  # Shallow/Deep Neural Network
        'feature_scaling': [RobustScaler(), StandardScaler(), None, Normalizer()],
            'feature_selection': [VarianceThreshold(threshold=0)],
            'dimensionality_reduction': [PCA(n_components='mle', random_state=42),
                                         None,
                                         LinearDiscriminantAnalysis(),
                                         PCA(random_state=42)],
            'classifier': [MLPClassifier(random_state=42, max_iter=4500)],
            'classifier__alpha': [0.0001, 0.001, 0.01],
            'classifier__hidden_layer_sizes': [(100,),
                                               (600,600,600),
                                               (300,),
                                               (400, 400),
                                               (400,),
                                               (500,),
                                               (300,300,300,300)],
            'classifier__batch_size': [512, 256, 128, 64],
            'classifier__activation': ['relu','tanh']
        },
        {  # Shallow/Deep Neural Network (TensorFlow)
            'feature_scaling': [StandardScaler()],
            'feature_selection': [VarianceThreshold(threshold=0)],
            'dimensionality_reduction': [None],
            'classifier': [KerasMLPClassifier(random_state=42)],
            'classifier__optimizer': ['adam'],
            'classifier__hidden_layer_sizes': [(100,),
                                               (100,100,100),
                                               (200,),
                                               (200,200),
                                               (100,100)],
            'classifier__epochs': [50, 100, 150],
            'classifier__batch_size': [64, 128, 256, 512, 1024]
        }
    ]
```

In total together for the binary and multi-class classification development, close to 5 600 candidate models were evaluated and due to the 4-fold cross-validation employed, close to 22 400 models were fitted. This was done using the parallel execution of various models simultaneously across 50 cores. The candidate hyper-parameters were chosen based on expert knowledge and from literature and scikit-learn recommendations/defaults.

Results

The classification performance metrics discussed in this section were obtained using *5-fold stratified cross-prediction* unless specified otherwise. This method allows obtaining accurate estimates of the performance expected of the ML models yet doesn't require putting aside a portion of the data as a test set before training and therefore can utilise the whole of the dataset and potentially capture more information. The source code for this project is available at `https://github.com/smith558/COMP3217-CW-2`, please follow `README.md`.

Definition: *macro F1 score.* The *macro F1 score* is an unweighted average of all per-class F1 scores. This variant is more sensitive to unbalanced data than a *weighted F1 score*.

Binary Classification

The most performant model (validated based on *macro F1 score*) for the binary classification (part A) was shown to be a *shallow feed-forward neural network* (also called *multi-layer perceptron)* with a *macro F1 score* of **0.98**, followed by *KNN* (*k-nearest neighbours*) with **0.96**. The comparison of other classification metrics of the best binary model is shown in table 1. The configuration of the neural network used is shown here:

```
{'classifier': MLPClassifier(batch_size=256, hidden_layer_sizes=(300,),
                             max_iter=4500, random_state=42,
                             alpha=0.001,),
 'classifier__activation': 'relu',
 'classifier__alpha': 0.001,
 'classifier__batch_size': 256,
 'classifier__hidden_layer_sizes': (300,),
 'dimensionality_reduction': PCA(n_components='mle', random_state=42),
 'feature_scaling': StandardScaler(),
 'feature_selection': VarianceThreshold(threshold=0)}
```

The neural network consisted of an input layer, output layer and 1 hidden layer of 300 units, the batch size was 256, regularization parameter $\alpha = 0.001$, activation function `ReLu`. The features of the data passed in were scaled using standard scaling (for each feature removing the mean and scaling to unit variance) and then stripped of features will a variance of 0 (single value features), finally, before training the neural network a PCA (principal component analysis) using Minka's MLE (dimensionality estimation method) was applied on the data to reduce the dimensionality of the data.

class	precision	recall	F1 score	accuracy
0	.97	.98	.98	
1	.98	.97	.98	
absolute				.98
macro average	.98	.98	.98	
weighted average	.98	.98	.98	

Table 1. Best model *binary classification* performance metrics comparison.

The confusion matrix can be seen in figure 2a. From this, we can see the classification performance is pretty much equal for both the positive class (*normal event*) and the negative class (*attack event*).

Multi-class Classification

The most performant model for the multi-class classification (part B) was shown to be a *deep feed-forward neural network* with a *macro F1 score* of **0.93** (obtained during automated model selection using 4-fold, the cross-validation value obtained using 5-fold was **0.92**), followed by *KNN* (*k-nearest neighbours*) with **0.92**. The comparison of other classification metrics of the best multi-class model is shown in table 2. The configuration of the neural network used is shown here:

```
{'classifier': MLPClassifier(activation='tanh', batch_size=512,
               hidden_layer_sizes=(600, 600, 600), max_iter=4500,
               random_state=42),
'classifier__activation': 'tanh',
'classifier__alpha': 0.0001,
'classifier__batch_size': 512,
'classifier__hidden_layer_sizes': (600, 600, 600),
'dimensionality_reduction': PCA(n_components='mle', random_state=42),
'feature_scaling': StandardScaler(),
'feature_selection': VarianceThreshold(threshold=0)}
```

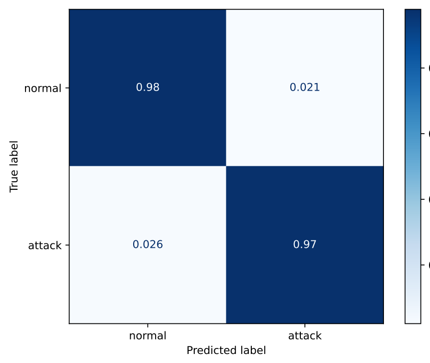
The neural network consisted of an input layer, output layer and 3 hidden layers of 600 units each, the batch size was 512, regularization parameter $\alpha = 0.0001$ (slightly smaller than for *binary classification*), activation function `Tanh`. The features of the data passed in were scaled using standard scaling (for each feature removing the mean and scaling to unit variance) and then stripped of features will a variance of 0 (single value features), finally, before training the neural network a PCA (principal component analysis) using Minka's MLE (dimensionality estimation method) was applied on the data to reduce the dimensionality of the data.

class	**precision**	**recall**	**F1 score**	**accuracy**
0	.96	.98	.97	
1	.9	.89	.9	
2	.89	.88	.89	
absolute				.93
macro average	.92	.92	.92	
weighted average	.93	.93	.93	

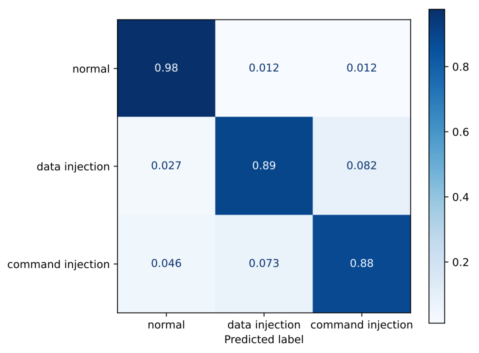
Table 2. Best model *multi-class classification* performance metrics comparison.

The confusion matrix can be seen in figure 2b. From this, we can see the classification performance is strongest for the *normal event* class for which 98 % of the sample are classified correctly, whilst for the *data injection attack* 89 % and *command injection attack* 88 %.

Figure 2. Classification confusion matrices. A *confusion matrix* is a table layout that allows comparison of classification performance on per-class granularity and spot problems. The grid matrix has a number of rows equal to the number of classes in the task, with the rows corresponding to true labels and columns to predicted labels. Therefore the intersection cell of each row/column identifies the number of samples of the dataset which were classified as either True Positive (TP), True Negative (TN), False Positive (FP) or False Negative (FN). The matrix presented here is row-normalised (cells in a row add up to 1).



(a) Binary classification.



(b) Multi-class classification.

Acknowledgements

The author acknowledges the use of the IRIDIS High-Performance Computing Facility, and associated support services at the UoS.