13173510 MonzonD Word count: 700

5th Section - Evaluating the Model and Analysing Results (Delvin Monzon)

Evaluation Metrics - For all model training and testing we used a MacBook Pro with 2.3GHZ dual core i5 CPU, 8gb RAM running on macOS Catalina (10.15.2). With regards to the evaluation metrics, we opted to use Detection rate (DR), false alarm rate (FAR), detection accuracy (Acc), Matthews correlation coefficient (Mcc) to measure the quality of binary classification, time to build the model (TTB), time to test model (TTM). We also calculated the area under the ROC curve (AUC), to discriminate between positive and negative classes and the harmonic mean between precision and DR (F1). In turn we believed that these metrics would strengthen our classification and model evaluation. Ultimately, we were aiming for high Acc, DR, Mcc, AUC and F1 whilst maintaining a low FAR, FNR, TTB and TTM. The majority of these metrics were used in papers that used DFES methods [1][2] and/or other methods using a reduced [3][4] Aegean Wi-Fi Intrusion Dataset (AWID). As a result, a more credible comparison of our chosen model with benchmarks can be conducted.

After choosing our best 4 features via RFE, training and testing, it appeared that MLP was consistent with producing some of the best results in each evaluation metric. Namely, Acc (99.77%), MCC (99.54%), AUC (99.91%). We also believe that FNR is one of the most important metrics due to the possibility of an attack going undetected which could have detrimental effects, thus being a bigger factor to consider when choosing our algorithm; MLP produces the second lowest score of 0.06%. At this point, LDA was also a contender due to its even lower FNR and TTB. However, the deciding factor was the Acc between MLP and LDA, 99.77% and 95.76% respectively.

Evaluation Metrics for the models we selected

Classifier	*Acc*	*DR*	*TNR*	*FAR*	*FNR*	*F1*	*MCC*	*AUC*	*TTB(s)*	*TTM(s)*
MLP	0.9977	0.9977	0.996	0.0039	0.00064	0.9977	0.9954	0.9991	12.8229	0.0328
DT	0.9336	0.8508	0.9976	0.0024	0.1492	0.9238	0.8577	0.9242	0.2202	0.0018
RF	0.9051	0.7697	0.9981	0.0019	0.2302	0.8824	0.7887	0.9984	0.517	0.0207
ET	0.9139	0.8006	0.9958	0.0042	0.1994	0.8972	0.8119	0.9953	0.1706	0.0226
LR	0.9686	0.9945	0.9399	0.06	0.0054	0.9672	0.9358	0.9585	0.1778	0.0008
Gaussian	0.25	0	1	0	1	0.3333	0	0.4971	0.0311	0.0035
SVM	0.9006	0.8409	0.9507	0.0493	0.1591	0.8955	0.7964	0.9597	26.808	0.7834
LDA	0.9576	0.9999	0.9075	0.0925	4.98E-05	0.9536	0.9113	0.976	0.08815	0.0088
Red = ranking 1	**Green = ranking 2**									

A strength of MLP is that random training iterations may result in better classification, unfortunately this would increase the total training time which seems to be an already expensive computational model. Also, the user has to set the number of hidden neurons initially, the disadvantage is that a value too low can lead to underfitting and a value too high may result in overfitting.[5]

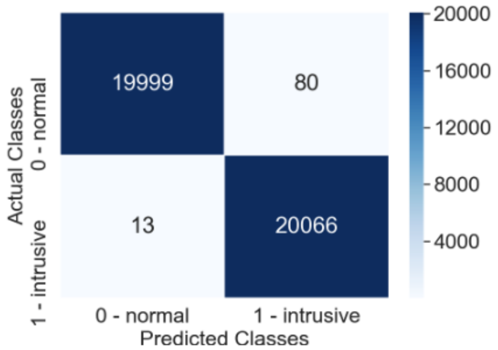
Additionally, the use of a sigmoid function is required for classification when using MLP. This could lead to long-term information being corrupted and causing vanishing gradients [6] – making the network harder to train. [7]

Despite MLP having one of the highest TTB (12.8229s), we decided that due to its excellent performance in several other metrics, as mentioned previously, we would select this to algorithm as our candidate. Once MLP has been tuned, the table below shows that the tuned version improves upon every evaluation metric we used. Most notable is the reduction of around 41% in TTB from 12.8229s to 7.6272s, which increases favourability to be used in real time IoT detection scenarios. Additionally, as seen in the confusion matrices, FNR has also decreased by 46%, illustrating less attacks going undetected thus less chance of intrusion.

Comparison of evaluation metric between tuned and un-tuned MLP

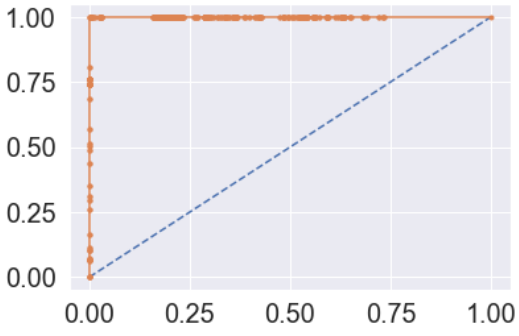
Classifier	***Acc***	***DR***	***TNR***	***FAR***	***FNR***	***F1***	***MCC***	***AUC***	***TTB(s)***	***TTM(s)***
MLP (tuned)	0.9984	0.9997	0.9972	0.0028	0.0003	0.9984	0.9968	0.9994	7.6272	0.0286
MLP	0.9977	0.9977	0.996	0.0039	0.0006	0.9977	0.9954	0.9991	12.8229	0.0328

Confusion Matrix of the un-tuned MLP

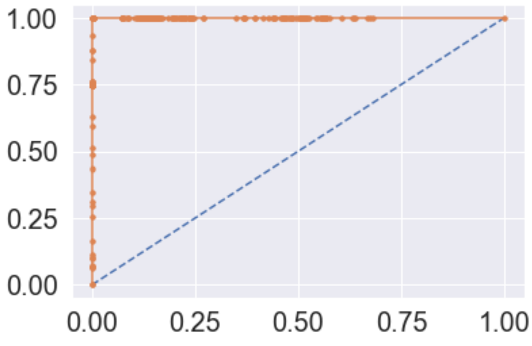


Confusion Matrix of the tuned MLP

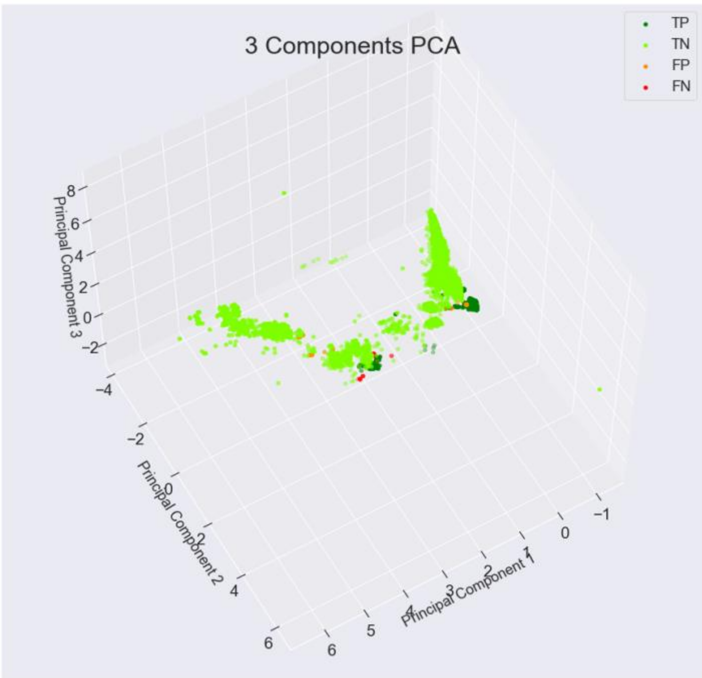
ROC Curve for of the un-tuned MLP



ROC Curve of the tuned MLP



3 Components PCA Plot showing the result of the Classification on the test set, for the tuned MLP (the different colours are used to highlight which points are TP, TN, FP and FN)



Evaluation Metrics for the tuned and untuned MLP

Benchmark Comparisons - Comparison of our tuned and untuned MLP algorithm with benchmark depicts promising progress: the Acc of our MLP is only beaten by SVM variants from Aminanto et al 2018 [1] and Parker et al [2]. The same models also produce a lower FAR of 0.01% and 0.012% respectively and also higher F1 and Mcc values. However, the use of feature extraction with RFE has allowed us to utilise only 4 features, substantially less than the DFES method used by Parker et al [2].

As a result, despite the slight loss in the previously mentioned values, the TTB for our MLP model is significantly quicker at 7.6272s compared to their 12073s and would not fare as well in an environment where speed may be paramount. Lastly, our tuned MLP model performed the best DR of 99.97%.

Performance comparison between our MLP model and algorithms from other papers

Classifier	*Acc (%)*	*DR (%)*	*FAR (%)*	*F1(%)*	*MCC (%)*	*TTB(s)*
MLP (tuned)	99.84	99.97	0.28	99.84	99.68	7.6272
MLP	99.77	99.77	0.39	99.77	99.54	12.8229
Parker et al 2019 (DETEReD RBFC) [2]	98.04	99.07	2.96	98.01	96.09	603.33
Parker et al 2019 (DFES SVM) [2]	99.97	99.92	0.01	99.94	99.92	12073
Aminanto et al 2018 [1]	99.97	99.92	0.01	99.94	99.92	12073
Aminanto and Kim 2017 [4]	97.6	85	2.36	NF	NF	350
Kolias et al 2015 [3]	96.26	96.3	43.68	94.8	NF	568.92

Red = ranking 1 **Green = ranking 2** ***NF = Not Found***

Resources

[1] ME Aminanto, R Choi, HC Tanuwidjaja, PD Yoo and K Kim (2018) Deep abstraction and weighted feature selection for Wi-Fi impersonation detection, IEEE Transactions on Information Forensics and Security, 13(3), 621–636.

[2] LR Parker, PD Yoo, TA Asyhari, L Chermak, Y Jhi and K Taha, DEMISe: Interpretable Deep Extraction and Mutual Information Selection Techniques for IoT Intrusion Detection, The 14th ACM International Conference on Availability, Reliability and Security (ARES), 26-29 Aug. 2019, U.K.

[3] C Kolias, G Kambourakis, A Stavrou and S Gritzalis (2016a) Intrusion detection in 802.11 networks: empirical evaluation of threats and a public dataset, IEEE Communication Surveys and Tutorials, 18(1), 184–208.

[4] ME Aminanto and K Kim (2017) Detecting impersonation attack in WiFi networks using deep learning approach, Information Security Applications 17th International Workshop. Jeju Island, South Korea, 25-27 August 2016, 136–147.

[5] N.Gillian, "MLP", nickgillian.com.
https://www.nickgillian.com/wiki/pmwiki.php/GRT/MLP (date accessed Jan 17th, 2020).

[6] P. Yoo. (2019). Regression-based ML Algorithms (LR and NN) [Pdf]. Available: https://moodle.bbk.ac.uk/mod/folder/view.php?id=637295.

[7] C-F Wang, "The Vanishing gradient Problem", Towardsdatascience.com.
https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484
(accessed Jan. 16th, 2020).

Appendix

---

1) **Criteria checklist used (for Evaluation section and Future works) to decide a success criterion and do progressive checks throughout the project lifetime.**

Planning (Group 30%):

Presenting a coherent plan that focuses on the practical side of the project. E.g. how we work together, our decisions, justifications and contingencies put in place before we move onto the next step. Creativity!

Criteria	Complete	Comments
Documenting progress: ○ GitHub (Commit, etc.) ○ Gant charts (efficiency, delegation, schedule) ○ Monitoring	☐	YOO is looking for "**creativity**" in how we document, keep a log of this project. Therefore, the criteria for 'Planning' is not restricted.
Contingencies (preventing risks): ○ Regular updates to GitHub/commits	☐	
Justifications: ○ Estimates ○ Resources	☐	
Delegation of tasks:	☐	
Discuss model trade-offs: accuracy vs speed	☐	

Pre-processing (Individual 40%):

Criteria	Complete	Comments
Data visualisation: ○ Descriptive statistics ○ Graphical visualisation	☐	
Very good **justification** of selection techniques: ○ Explain why we used it. ○ Effective techniques ○ If technique was relevant **but not used,** then explain why.	☐	
Technique (not limited to): ○ Transformation ○ Discretisation	☐	

○ Cleaning ○ Normalisation ○ Standardisation ○ Smoothing ○ Feature construction Creativity!		
Selection and application of technique is effective:	☐	

Selecting features (Individual 40%):

Criteria	Complete	Comments
Consider techniques from following methods: ○ Filter ○ Wrapper ○ Embedded	☐	
Dimensionality techniques (Linear and non-linear): ○ PCA ○ Factor analysis ○ LDA	☐	
Feature construction techniques: ○ Autoencoder ○ GANS ○ Literature research	☐	
Strong evidence of wider research and reading: ○ Novel feature selection techniques and their justifications.	☐	
Justification for feature selection techniques: ○ Application of FS ○ Effectiveness ○ Consideration of alternatives.	☐	

Exploring and selecting ML algorithms (Individual 40%):

Criteria	Complete	Comments
Select candidate algorithms: 5-10 + justification	☐	
Establish baselines for model performance: ○ Progress from simple model using initial data pipeline.	☐	

Justify and discuss selection strategies for algorithms: ○ Effectiveness ○ Selection strategies of algorithms	☐	
Evidence of wider reading of literature: ○ Novel algorithms outside the scope of lecture content.	☐	

Refining algorithms (Individual 40%):

Criteria	Complete	Comments
Justification of selection and best configuration for hyperparameters (effectivity, insight, creativity): ○ High dimensional space ○ Learning rate ○ Dropout rate ○ Batch size ○ Wider reading hyperparameters	☐	
Justification and consider model design components: ○ Number of layers ○ Number of unit per layers ○ Loss functions ○ Activations ○ Optimisers ○ Drop out layer ○ Any novel wider components from lit.	☐	
Perform model-specific optimisations: reproducibility	☐	
Debugging model as complexity is added.	☐	
Discuss selection strategies for searching for best configuration: ○ Trial and error ○ Grid search ○ Random search ○ Bayesian optimisation ○ Wider reading ideas.	☐	

Evaluating model and analysing the results (Individual 40%):

Criteria	Complete	Comments

Evaluate classification performance and explain why these are chosen: o Accuracy o Detection rate o False alarm o Type II error o MCC o TBM (Time has taken to build model) o TTM (Time taken to test model) o Wider resources to go beyond lectures.	☐	
Compare model's performance with benchmarks.	☐	
Strong justification on evaluation methods: o All models are fairly evaluated. o All evaluation measures are correctly calculated. o **<u>Interpretation of results</u>** is very good o Compared with some well-chosen sources.	☐	
Evidence of wider reading and wider choice of extra measures.	☐	Log loss?

Future group work (Group 15%)

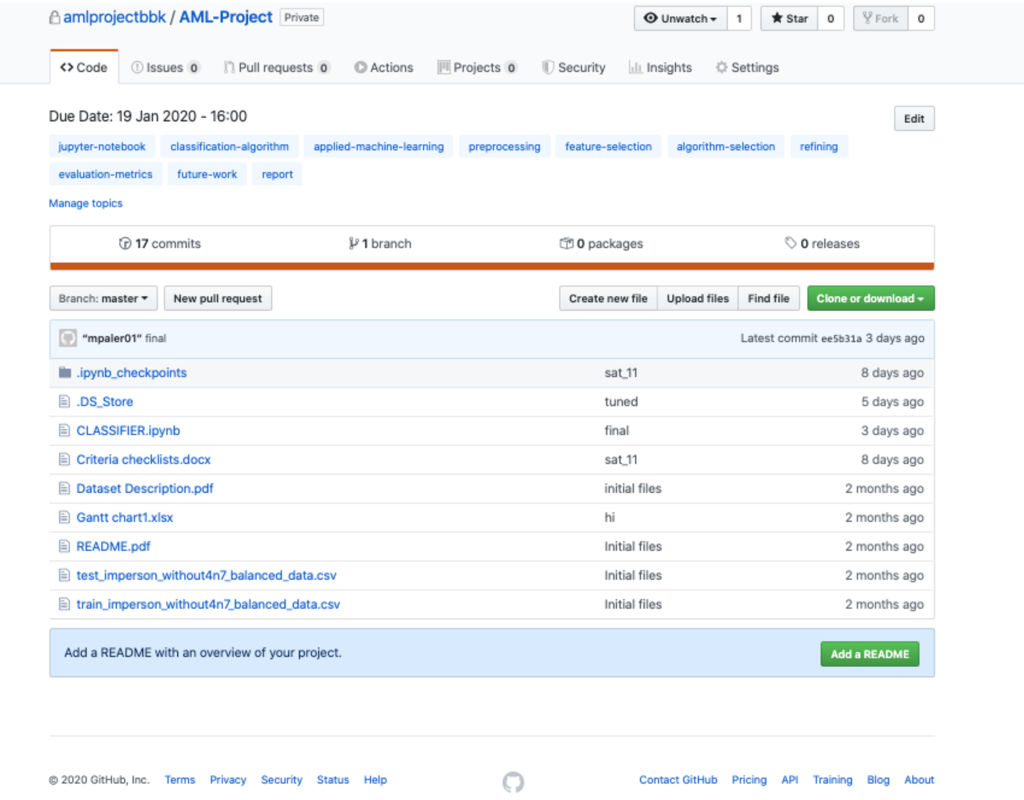
Criteria	Complete	Comments
Explain where results can lead to:	☐	
Strengths and weaknesses:	☐	
Next possible steps to take:	☐	
Possible questions raised by results:	☐	
What paths were more promising? Justify.	☐	
What are the future plans? Collaboration. o Unachievable in the time we spent on project o Optimistic	☐	
Strongly addresses question, strong evidence for future work.	☐	

Report documentation (Group 15%):

Criteria	Complete	Comments

Organisation of work is: o Well ordered o Concise o Coherent	☐	
Excellent use of appendices and illustrations	☐	
No mistakes (spelling, terminology)	☐	
Correct IEEE referencing	☐	
4000 words (+- 10%) = around 850 each	☐	
Arial 10 point or Times New Roman 11	☐	
1.5 line spacing, 1 inch margins	☐	
Submitted code in .ipynb or .py format	☐	

2) GitHub collaboration between our group. We used a basic account so maximum number of contributers were two. As a result we all used the same log in details to commit. **Currently set on private but if you wish to access it please get in touch.**



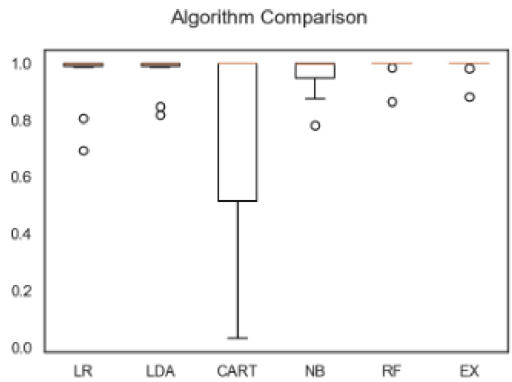
3) Baseline accuracy comparison of some candidate models.

Baseline Accuracy

```
#compare_models(df_backup)
```

```
LR: 0.947115 (0.101375)
LDA: 0.964304 (0.065782)
CART: 0.760872 (0.371827)
NB: 0.956431 (0.069650)
RF: 0.984831 (0.040355)
EX: 0.986243 (0.035194)

<IPython.core.display.Javascript object>
```



4) Report on test data for MLP

```
best_params_MLP={'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.01,
 'max_iter': 250,
 'momentum': 0.1,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': None,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.00000001,
 'warm_start': False}

to_keep = ['38','50','107','LDA1','LDA_tuned1','target']
df_best, df_test_best = df.loc[:,to_keep], df_test.loc[:,to_keep]
class_acc_report(MLPClassifier, df_best, df_test_best, best_params_MLP)
```

```
              precision    recall  f1-score   support

         0.0     0.9979    0.9979    0.9979     20079
         1.0     0.9979    0.9979    0.9979     20079

    accuracy                         0.9979     40158
   macro avg     0.9979    0.9979    0.9979     40158
weighted avg     0.9979    0.9979    0.9979     40158


(MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
               beta_2=0.999, early_stopping=False, epsilon=1e-08,
               hidden_layer_sizes=(100,), learning_rate='constant',
               learning_rate_init=0.01, max_iter=250, momentum=0.1,
               n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
               random_state=None, shuffle=True, solver='adam', tol=0.0001,
               validation_fraction=1e-08, verbose=False, warm_start=False),
 array([0., 0., 0., ..., 1., 1., 1.]))
```

5) Report on test data for KNN.

```
: 1 class_acc_report(KNeighborsClassifier, df_best, df_test_best)
```

```
              precision    recall  f1-score   support

         0.0     0.9993    0.9907    0.9950     20079
         1.0     0.9908    0.9994    0.9950     20079

    accuracy                         0.9950     40158
   macro avg     0.9951    0.9950    0.9950     40158
weighted avg     0.9951    0.9950    0.9950     40158
```

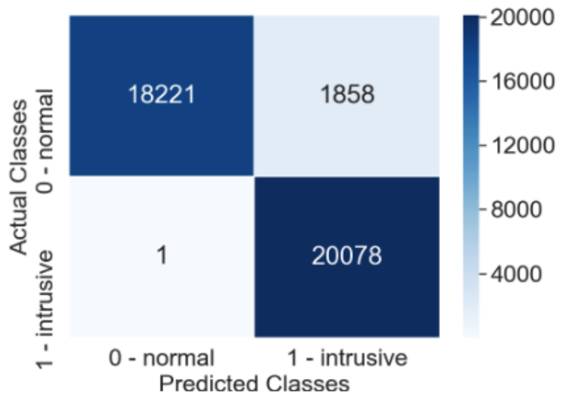
```
: (KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                       metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                       weights='uniform'), array([0., 0., 0., ..., 1., 1., 1.]))
```

6) LDA analysis and ROC curve.

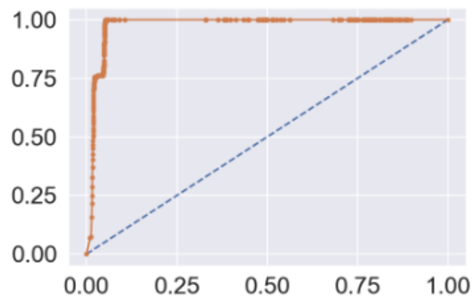
```
: 1 get_acc_measures(LinearDiscriminantAnalysis)
```

```
train Acc: 0.9841412142945467
test Acc: 0.9576220865344918
f1-score: 0.9536086528418459
TPR: 0.9999501967229444
TNR: 0.907465511230639
FPR: 0.09253448876936099
FNR: 4.98032770556156e-05
AUC: 0.976025667535988
MCC: 0.9113215345067753
TTB: 0.08815622329711914
TTM: 0.0008790493011474609

<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>
```

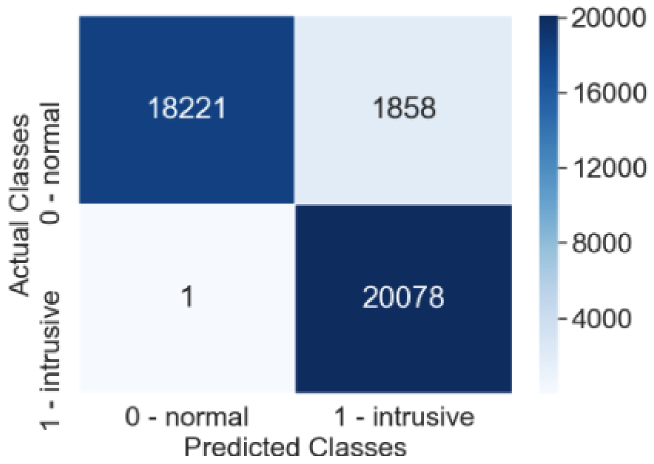


7) Gaussian NB analysis and ROC curve.

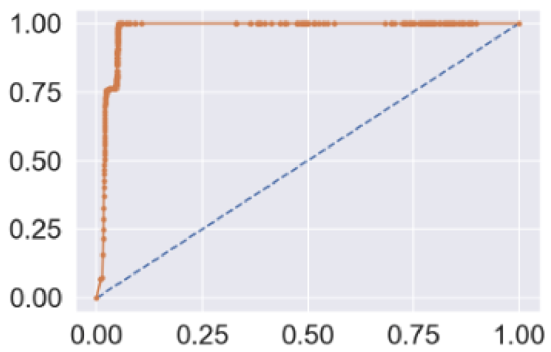
```
get_acc_measures(LinearDiscriminantAnalysis)
```

```
train Acc: 0.9841412142945467
test Acc: 0.9576220865344918
f1-score: 0.9536086528418459
TPR: 0.9999501967229444
TNR: 0.907465511230639
FPR: 0.09253448876936099
FNR: 4.98032770556156e-05
AUC: 0.976025667535988
MCC: 0.9113215345067753
TTB: 0.08815622329711914
TTM: 0.0008790493011474609

&lt;IPython.core.display.Javascript object&gt;
```




```
&lt;IPython.core.display.Javascript object&gt;
```

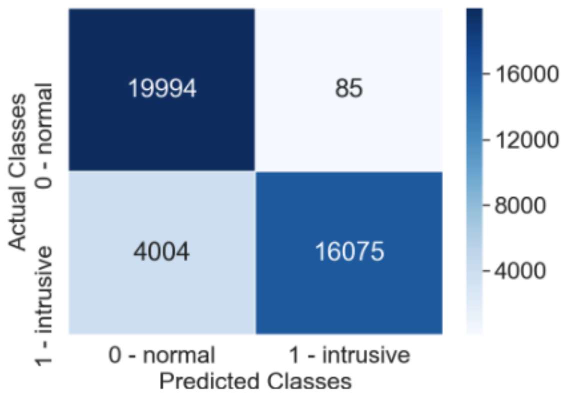


6) Extra Trees analysis and ROC curve.

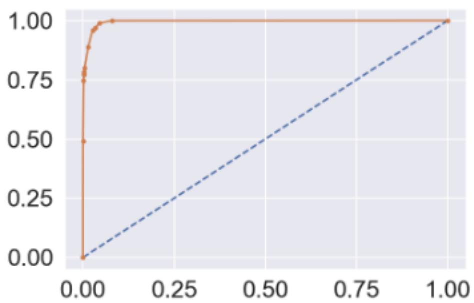
```
1  get_acc_measures(ExtraTreesClassifier)
```

```
train Acc: 0.9425209183463171
test Acc: 0.9139464308700643
f1-score: 0.8971981445190512
TPR: 0.8005876786692564
TNR: 0.9957667214502715
FPR: 0.0042332785497285474
FNR: 0.1994123213307436
AUC: 0.9953882373797265
MCC: 0.8119705187223852
TTB: 0.17059707641601562
TTM: 0.022600889205932617

<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>
```

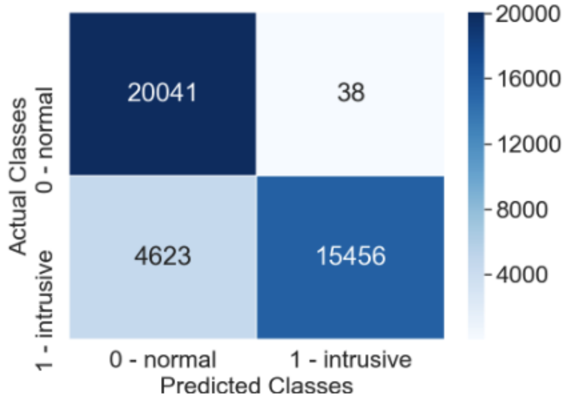


7) Random Forest analysis and ROC curve.

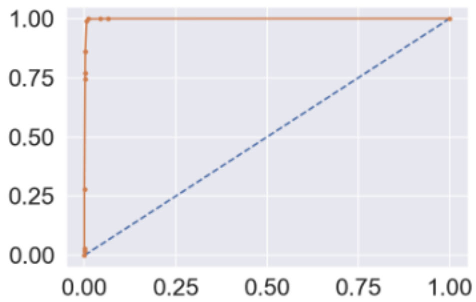
```
get_acc_measures(RandomForestClassifier)
```

```
train Acc: 0.7717736284571947
test Acc: 0.9050541275517449
f1-score: 0.882400470021558
TPR: 0.7697594501718213
TNR: 0.998107475471886
FPR: 0.001892524528113948
FNR: 0.23024054982817865
AUC: 0.998423980146691
MCC: 0.7887048467487091
TTB: 0.5170192718505859
TTM: 0.020718812942504883

<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>
```

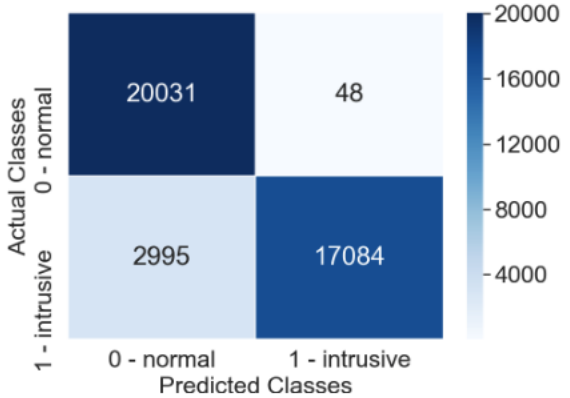


8) Decision Tree analysis and ROC curve

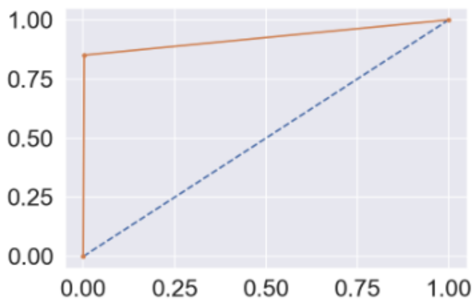
```
1  get_acc_measures(DecisionTreeClassifier)
```

```
train Acc: 0.8699043732739788
test Acc: 0.9335639351461288
f1-score: 0.923814023746447
TPR: 0.8508391852183874
TNR: 0.9976094427013298
FPR: 0.002390557298670215
FNR: 0.14916081478161258
AUC: 0.9242243139598586
MCC: 0.8577374025775093
TTB: 0.22018933296203613
TTM: 0.0018210411071777344

<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>
```

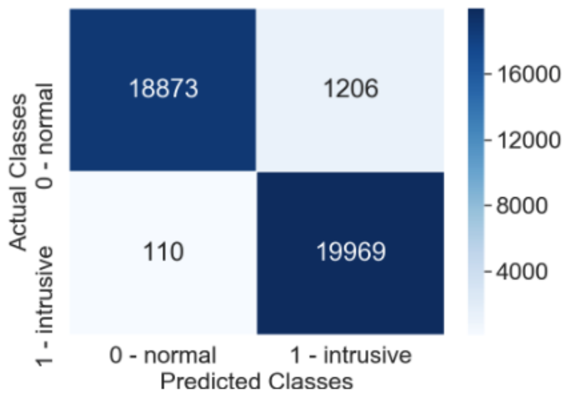


9) Logistic regression analysis and ROC curve.

```
1  get_acc_measures(LogisticRegression)
```

```
train Acc: 0.9306191006141544
test Acc: 0.9686256932428389
f1-score: 0.967205015895896
TPR: 0.9945216395238806
TNR: 0.9399372478709099
FPR: 0.06006275212909007
FNR: 0.005478360476119382
AUC: 0.9585312191354831
MCC: 0.9358540953720462
TTB: 0.17781400680541992
TTM: 0.0008089542388916016

<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>
```