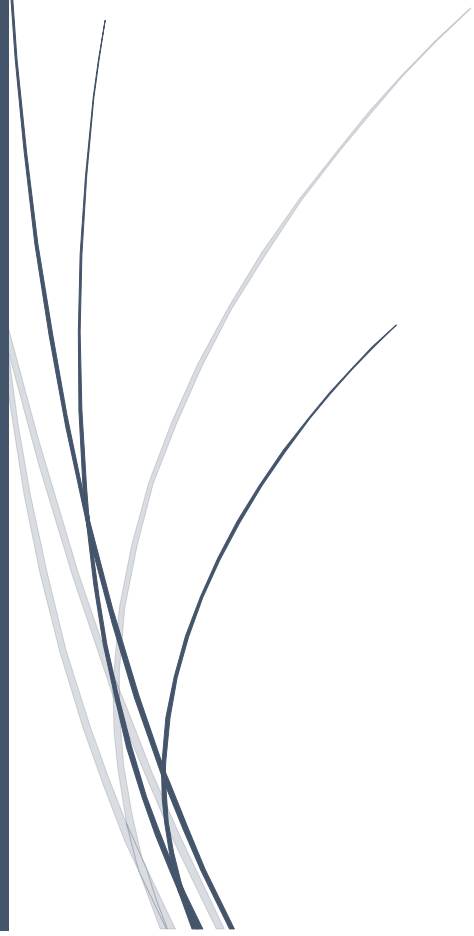
ELG7186[EG] Ai For Cyber Security 20219

Assignment 2 - Kafka Server

Hosna Eltarras

STUDENT ID: 300267032



Problem 1: Network Intrusion

I. Algorithm Description

- Static:

Firstly, we started by exploring the data where some null values and infinities were found. After investigation, they appeared in the "Flow Bytes" and "Flow Packets" features. These were cleaned by dropping these null-containing-rows as they were about only "6" rows out of over 25k.

Secondly, the data was split using Hold-Out Split into train and test sets. The train set was used in K-Fold Cross Validation (10-folds) to evaluate the training performance. For the classifiers, two Algorithms were experimented from sklearn → (1) Gradient Boost Classifier. (2) Decision Tree (DT). Both with parameters discussed below.

Thirdly, After training and validation by 10-Fold CV, prediction was made on the test set for each classifier and evaluated by below metrics.

- Dynamic:

The cleaned data was loaded with 2 model copies for the best performing one from the static case; which was "Decision Tree".
One of the 2 model copies is updated and the other remains as is to compare both models' performance on the new stream (original static and updated one; dynamic). The columns' names were modified by removing spaces to match the new stream with the original data.

The Adaptation algorithm was performed manually by the Single Classifier Evolving mechanism for adaptive learning (i.e. Forgetting) where we have a sliding window of 1000 packets. Each coming packet *-from kafka-* is stored in a dataframe until we reach 1000 packets, then prediction is done on the 1000 packets via the static and dynamic models*(same as the static in the 1st iteration)*. The dynamic model is retrained by the sliding window mechanism; where we drop the 1st 1000 rows from the original data and concatenate to them *(from the end)* the new 1000 packets. Thus, we maintain the same size of data at each iteration. This is repeated for 100 times to consume a total of 100k packets.

II. Algorithm Evaluation

Hyperparameters: Both algorithms were trained using the hyperparameters proposed in sklearn examples as a 1st trial, and as they provided good results there was no need to change or modify any.

(1) Gradient boost: [Number of estimators: 100, max depth: 1, learning rate: 1]
(2) Decision Tree: [all as default: criterion: "gini", max_depth: None, min_samples_split: 2, max_features=n_features (78), min_samples_leaf: 1, class weight: 'balanced']
The class_weight parameter is set to balanced to account for the dataset imbalance and return unbiased results. Random state of 0 for reproducibility.

Metrics and Performance Estimation: Evaluation was performed on 2 steps in case of static model; (1) Using 3 scores: precision/recall/macro-f1 in 10-Fold cross validation during training *(CV required encoding of the labels; here it was performed manually).* (2) Using the classification report to provide the same scores on the test data. Macro-F1 score was used over the weighted as it seemed to give more fair results consistent with the imbalance.

Then, during streaming the same 3 metrics were calculated for the static and dynamic model's predictions, in addition to the Attack f1-score (positive class) as it's the most significant class. Also, Average precision-recall was used as it's recommended in case of imbalanced dataset. Accuracy wasn't used due to its inadequacy in the context of imbalance as it blindly account for correct predictions regardless the class scarcity or significance.

III. Results And Discussion:

Average scores of 100 iterations are shown in table below for streaming case.

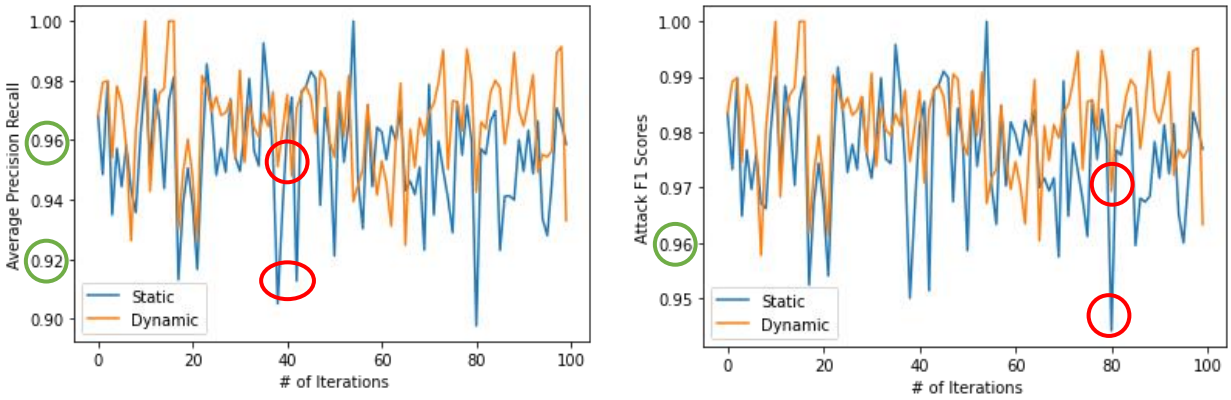
Average Scores	Evaluation Case	(Static) GradientBoost	(Static) Decision Tree	Dynamic Model
Precision	Training (10-Fold) Testing (test set) Streaming	0.971 0.98 -	0.986 0.99 **0.9876**	- - **0.9901**
Recall	Training (10-Fold) Testing (test set) Streaming	0.953 **0.96** -	**0.989** **0.99** 0.9856	- - 0.9896
Macro-F1	Training (10-Fold) Testing (test set) Streaming	0.968 0.97 -	0.987 0.99 0.9865	- - 0.9898

Insights: We can observe 3 main things from the table above:

1st: How indictive the cross validation is, where the average performance of 10 fold validations is similar to the performance in **test set**. (DT Recall: **0.989, 0.99**)

2nd: The outperformance of the Decision tree over the gradient boost in all metrics specifically the recall (which concerns with the positive class; attack)

3rd: The dynamic and static models on the streaming cases *(Highlighted in yellow)* We can see the dynamic model is performing **slightly better** than the static model. **Better**: for it's being retrained/updated on the stream data, thus learning the new data distribution. **Slightly**: as it seems the new stream had a very similar distribution to the original data, thus the static model maintained to capture the stream's pattern easily. This is depicted in the below graphs.



In general, we can see from both graphs the trend in the performance, at some iterations there were some hard packets on both models (as in iteration 40, 80) where the models find some difficulty identifying these packets. Also, it's obvious that the dynamic model is mostly performing better (see gap between the red circles). The Avg-precision-recall ***(Fig 1: left)*** seems to be a good metric were at iteration 40 or 80, it reaches to 92% in the static model with 95% dynamic.

Problem 2: IOT Botnet

I. Algorithm Description

- Static: Here, while exploring the data it seemed to be clean. However, there was a categorical feature ["Source"]. We can encode it, but as we searched for its importance it didn't seem to be very influential. So, with 115 other features, we chose to drop this column to facilitate the process. The same algorithms and classifiers were used as in the 1st problem.

- Dynamic: Same algorithm -as in 1st problem- was used (i.e. Forgetting). Data was loaded while removing "source" column and spaces in feature names.

II. Algorithm Evaluation

Hyperparameters: Same parameters were used as in the 1st problem.
Metrics and Performance Estimation: Same criteria was used with few changes. For static model in CV part; Macro-precision/Macro-Recall were used

instead of precision/recall as these aren't supported for multi-class in cv (Encoding for classes in CV was made via label encoder).
During Streaming, again same metrics were used. But as we've multiple attacks here we can't retrieve one (Attack f1-score), so we chose F1-score of the rarest attack classes (Mirai and Gafygt_combo attacks). However, since these were sometimes unavailable in any of the 1000 streamed packets, in such cases we'd manually add their F1-scores as 0 to denote their complete absence.

III. Results And Discussion:

Average scores of 100 iterations are shown in table below for streaming case.

Average Scores	Evaluation Case	(Static) GradientBoost	(Static) Decision Tree	Dynamic Model
Macro Precision	Training (10-Fold) Testing (test set) Streaming	0.647 0.85 -	0.916 0.96 0.9665	- - 0.9659
Macro Recall	Training (10-Fold) Testing (test set) Streaming	0.66 **0.89** -	0.9157 **0.97** 0.9674	- - 0.9645
Macro-F1	Training (10-Fold) Testing (test set) Streaming	0.641 0.87 -	0.914 0.96 0.966	- - 0.9638

Insights: From the table above we can see: (1) The power of DT in contrast to gradient boost. (2) Both static and dynamic models had very close results with the dynamic model a bit lower, this appears more clearly in the next graph *(Figure 2)* where at iteration 70 the recall for dynamic model was horribly lower than the static model.
However, the power of dynamic model is manifested more in the rare attack classes.

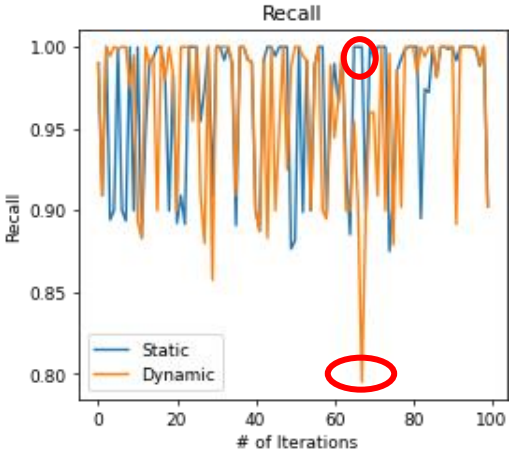
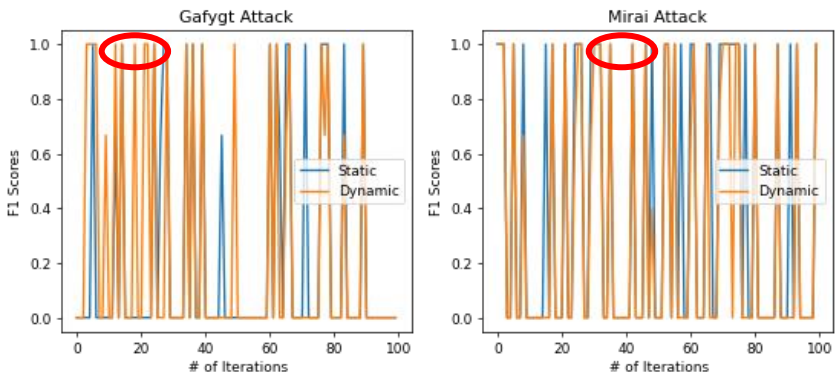


Figure 2: Recall on 100 iterations for stream Botnet



At iterations (10-30) Gafygt *(left)* attack, it seems the static model nearly misses most of the packets while the dynamic one detects them. Same for "Mirai"

Advantages and Limitations:

We can see the advantage of using simple models as Decision tree in some contexts, which implies that complex models (as XGBoost) are not always better. The pro in using cross validation for evaluation before testing that gives a prior understanding on what would happen in the test set.
Moreover, while the difference between the performance of the static and dynamic models is not that significant. However, we can see how the static model is more limited in terms of rare classes -especially in the multiclass problems- where the dynamic model is performing better even though they nearly reach the same values in the overall average metrics.
The advantage, also, of calculating the performance on specific classes rather than an overall average as this gives more insights on how the model is performing on the most crucial classes.
Finally, the fact that 100k packets were used has assisted in assessing the model performance well.

Knowledge Learned:
Adapting models on new streams might be beneficial at some points, specifically when you have different distribution or multiple classes that were imbalanced in the original data. Dynamic models can detect such classes better as it'll learn on them from the new stream.
Nevertheless, if the production/stream data distribution is similar to the in-hand data distribution, your model -without adaptation- can be sufficient to some point.
Not all the classes/features would be always found in the new data coming in. Thus, you have to account for these possibilities by observing the trend of the production/new streams and dealing with them.
Lastly, trying an adaptive model and comparing it with the static model was much informative as it's always thought that definitely the dynamic model will always perform better, but this is not always true. So don't take everything for granted until you try it yourself. At least in such areas.