Faculty of Engineering

School of Electrical Engineering and Computer Science

uOttawa
L'Université canadienne
Canada's university

Assignment 3 Description Continuous Evaluation with Kafka

Part I (Static Model):

1. **Data Analysis**:

- First five rows of the static dataset. The dataset has 268074 rows and 16 columns.

```
df = pd.read_csv("/content/drive/MyDrive/AI_CS/Static_dataset.csv")
```

```
df.head()
```

	timestamp	FQDN_count	subdomain_length	upper	lower	numeric	entropy	special	labels	labels_max	labels_average	longest_word	sld	len	subdomain	Target Attack
0	56:19.8	27	10	0	10	11	2.570417	6	6	7	3.666667	2	192	14	1	1
1	07:23.9	27	10	0	10	11	2.767195	6	6	7	3.666667	2	192	14	1	1
2	23:15.1	26	9	0	10	10	2.742338	6	6	7	3.500000	2	192	13	1	0
3	04:51.9	27	10	0	10	11	2.570417	6	6	7	3.666667	2	192	14	1	1
4	12:44.0	15	9	0	11	0	2.929439	4	3	5	4.333333	local	local	15	1	1

- **The Distribution of the dataset**

➔ The image shows the Count, mean, standard deviation, min value, 25%, 50%(median), 70% quartiles, and max value of the dataset.

```
df.describe()
```

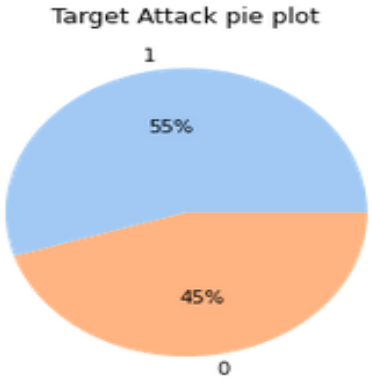
	FQDN_count	subdomain_length	upper	lower	numeric	entropy	special	labels	labels_max	labels_average	len	subdomain	Target Attack
count	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000
mean	22.286596	6.059021	0.845420	10.410014	6.497586	2.485735	4.533577	4.788823	8.252233	4.802239	12.576714	0.753497	0.549024
std	6.001205	3.899505	4.941929	3.207725	4.499866	0.407709	2.187683	1.803256	4.415355	4.573066	4.177828	0.430975	0.497592
min	2.000000	0.000000	0.000000	0.000000	0.000000	0.219195	0.000000	1.000000	2.000000	2.000000	2.000000	0.000000	0.000000
25%	18.000000	3.000000	0.000000	10.000000	0.000000	2.054029	2.000000	3.000000	7.000000	3.166667	11.000000	1.000000	0.000000
50%	24.000000	7.000000	0.000000	10.000000	8.000000	2.570417	6.000000	6.000000	7.000000	3.666667	12.000000	1.000000	1.000000
75%	27.000000	10.000000	0.000000	10.000000	10.000000	2.767195	6.000000	6.000000	7.000000	4.000000	14.000000	1.000000	1.000000
max	36.000000	23.000000	32.000000	34.000000	12.000000	4.216847	7.000000	7.000000	32.000000	32.000000	33.000000	1.000000	1.000000

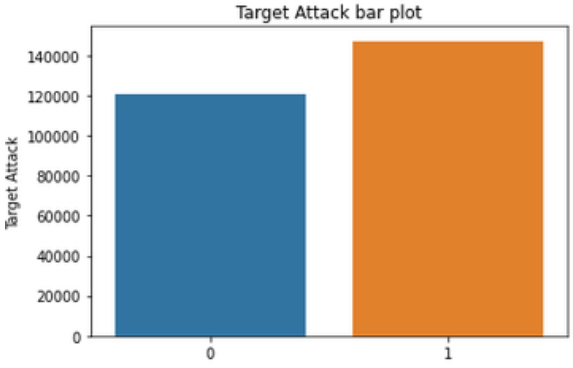
- **The number of values in each class to see if the data is balanced or imbalanced:**

We can see that the dataset is balanced as the number of values in each class is not too different than the other, 55% of the data is class 1 and 45% of the data is class 0, which is 10% difference only.

```
df['Target Attack'].value_counts()
```

```
1    147179
0    120895
Name: Target Attack, dtype: int64
```





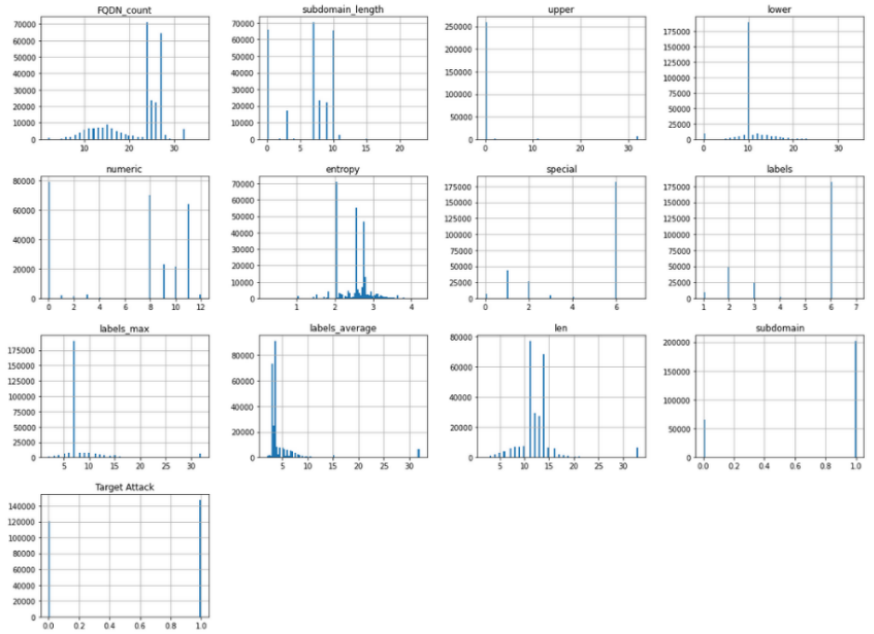
Each feature distribution, and data skewed pattern:

We can see that data features have different skewed patterns.

For example:

labels_average column is right skewed.

```
df.hist(bins=100, figsize=(20,15))
plt.show()
```



The skew of every feature:

Some of features are right skewed and others are left skewed.

```
FQDN_count          -1.101731
subdomain_length    -0.590480
upper                5.988737
lower                0.343449
numeric             -0.594384
entropy             -0.140156
special             -0.902972
labels              -0.903680
labels_max           3.979910
labels_average       5.087081
len                  2.634801
subdomain           -1.176397
Target Attack       -0.197046
dtype: float64
```

2. Feature engineering and data cleaning:

- **Check for Null values:**
  There are 8 missing values in **longest_word,**
  this is a very small number so I will remove them.

```
df.isna().sum()
```

```
timestamp          0
FQDN_count         0
subdomain_length   0
upper              0
lower              0
numeric            0
entropy            0
special            0
labels             0
labels_max         0
labels_average     0
longest_word       8
sld                0
len                0
subdomain          0
Target Attack      0
dtype: int64
```

```
df.dtypes
```

```
timestamp           object
FQDN_count           int64
subdomain_length     int64
upper                int64
lower                int64
numeric              int64
entropy            float64
special              int64
labels               int64
labels_max           int64
labels_average     float64
longest_word        object
sld                 object
len                  int64
subdomain            int64
Target Attack        int64
dtype: object
```

- **Check for missing values and categorical:**

  Timestamp, longest_word, sld columns is categorical values and we will need to encode it to feed it to the model.

  Encoding categorical Features:

  I used label encoding to encode the categorical values to numerical values.

```
from sklearn.preprocessing import LabelEncoder
df['sld'] = df[['sld']].apply(LabelEncoder().fit_transform)
df['longest_word'] = df[['longest_word']].apply(LabelEncoder().fit_transform)
df['timestamp'] = df[['timestamp']].apply(LabelEncoder().fit_transform)
```

```
df.dtypes
```

```
timestamp           int64
FQDN_count          int64
subdomain_length    int64
upper               int64
lower               int64
numeric             int64
entropy           float64
special             int64
labels              int64
labels_max          int64
labels_average    float64
longest_word        int64
sld                 int64
len                 int64
subdomain           int64
Target Attack       int64
dtype: object
```

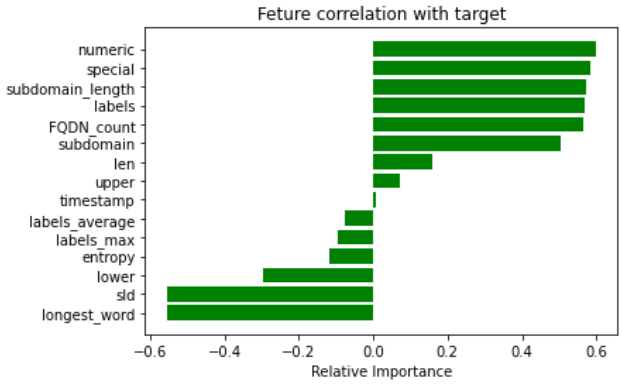
3. **Feature Filtering/Selection:** The dataset has 16 features, we want to reduce these features to make the model more general, faster, and eliminate the redundant values.

   Select independent features with:

   - High correlation with the target variable,
   - Higher information gain or mutual information of the independent variable,
   - Anova f-test.

- **Finding correlation with target variable of independent predictors:**

➔ I used this approach to filter the feature that has small relation with the target column and put a threshold for the correlation as I only keep the feature if its correlation with the target is greater than 0.2

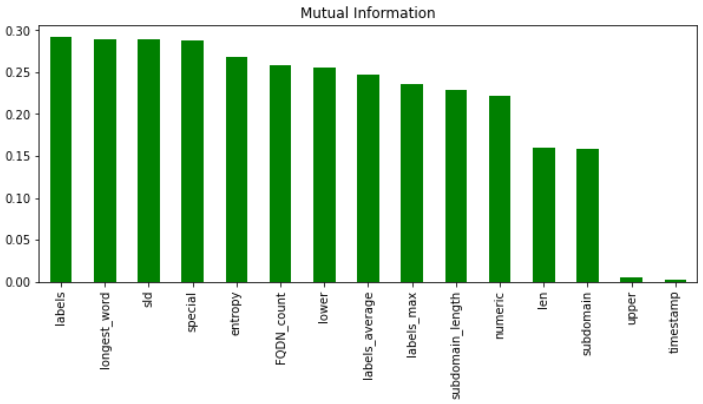


➔ The features I decided to keep from this approach are:

FQDN_count, subdomain_length, lower, numeric, special, labels, longest_word, sld, subdomain.

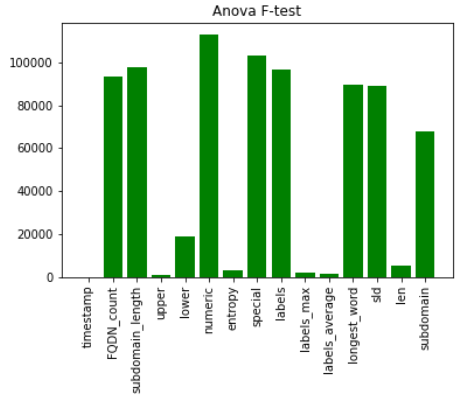
➔ **Mutual Information or Information Gain:**

This approach agrees to remove **upper, timestamp**



➔ **ANOVA f-test Feature Selection**

By plotting the score of each feature, I am confident to remove the **"entropy", "labels_max", "labels_average", "upper", "labels_average", 'len',"timestamp" columns.**



The selected features

```
X_selection.columns
```

```
Index(['FQDN_count', 'subdomain_length', 'lower', 'numeric', 'special',
       'labels', 'longest_word', 'sld', 'subdomain'],
      dtype='object')
```

4. **Model Training:**

I have already split the data before feature selection, but I need to split it again after it.

I split the dataset into the target and the features, then I used train_test_split from sklearn to split data to train and test with 25% of the data for testing and the rest for training and satisfied it.

I Normalize the training then the testing dataset using **Normalizer** from **sklearn**.

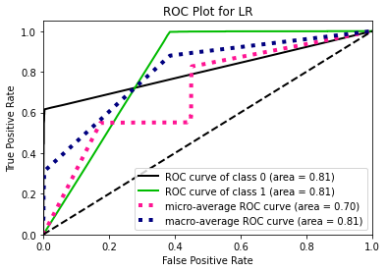
I used two learning algorithms and examen their performance before and after data normalization.

	LR without Normalization	LR with Norm	Naïve Bayes without Norm	Naïve Bayes with Norm
Accuracy	82.44	81.56	80.30	80.30
F1-score	1: 86 0: 76	1: 85 0: 75	1: 84 0: 74	1: 84 0: 74
Precision	1: 76 0: 99	1: 76 0: 95	1: 76 0: 90	1: 76 0: 90
Recall	1: 100 0: 62 Micro avg: 81	1: 97 0: 63 Micro avg: 80	1: 94 0: 64 Micro avg: 79	1: 94 0: 64 Micro avg: 79

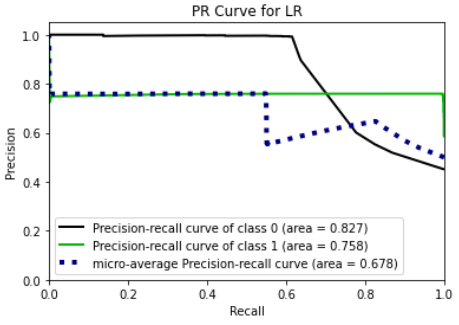
The model with best accuracy is Logistic regression before normalization, but the accuracy is not the best metric to use to evaluate our models, even if our data is balanced, it just gives as an intuition about how both positive and negative were correctly classified, but here in this problem I want to make sure that there is less attacks were classified as normal action PS: recall, but I don't want all the action be classified as attacks, so I also care about precision, F1 score is a good metric to balance precision and recall, and the Logistic regression model without normalized data is the winner here.

5. **Model evaluation:**

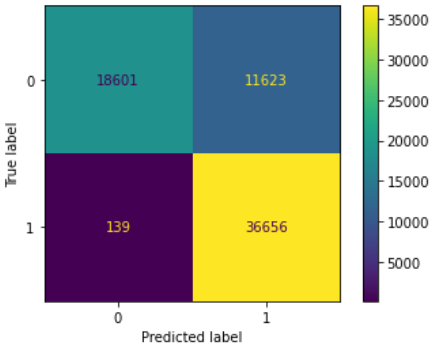
- **AUC-ROC:**
  - Is the measure of the ability of a classifier to distinguish between classes.
  - The ROC is 0.81 which is a good score for our problem.
  - The higher value of y-axis indicates that the TP is higher than FN.



- **Precision Recall Curve:**
- A good classifier will maintain both a high precision and high recall, which we can see in the PR graph.



- **Confusion matrix:** It shows combinations of predicted and actual values.

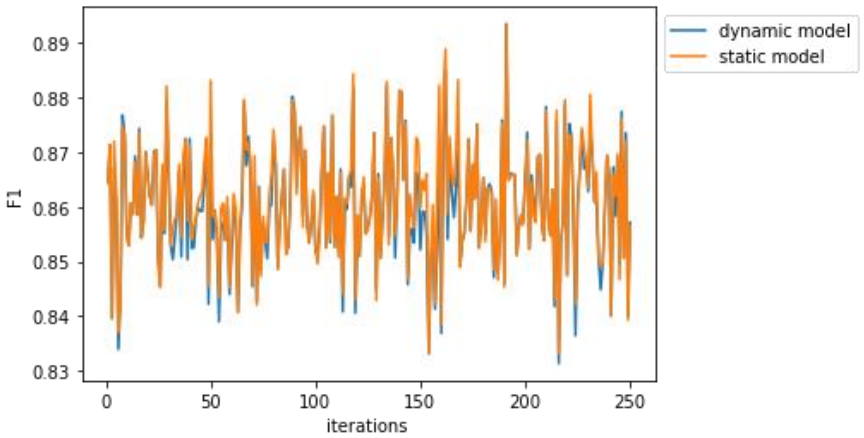


We can see that our model didn't misclassify the class as class 0, it only misclassified 139 points, and 11623 classified as 1 and it was 0.

➔ **At the end, I saved the LR model to use it in dynamic part.**

Part II (Dynamic Model):

1. I loaded the model from the static part and then preprocessed the "Kafka_dataset.csv" as previous.
2. I Run the consumer's code and validate you are receiving the data stream.
3. I appended 1,000 observations of data streaming as a window.
4. Create a function to preprocess the streaming data like the static data before.
5. Create a pipeline with the same architecture model as static part.
6. The dynamic part will have 250 windows, as the data streamed, it will calculate its Accuracy, F1 score, compare it with the threshold (F1=85%) and retrain the model if it is lower than it.
7. I evaluate the F1 score as a chosen metric from the last part for each model on each window and draw the F1 vs iteration plot.



This is the result of the last window generated, as we care about the F1 score, we can see that it is not changing much, but these small changes can be important depending on the case and the user requirements, so saying dynamic implementation is better or not depends on the situation.

```
Window 250
ACC of Dynamic Model without retrain = 81.8%
F1 score of Dynamic Model without retrain = 85.71428571428571%
**********
ACC of Static Model = 81.69999999999999%
F1 score of Static Model = 85.6470588235294%
**********
```