Static Model

1. **Data Analysis:**

   Most distributions of features are mild and moderate left and right skewed. The 'lower' and 'entropy' features have very low skewness 0.3 and -0.1 respectively which are near zero-skew (normal distribution). One very strong right-skewed feature (sld) with value = 180.4. The distribution of the target variable and the skewness is Left-skewed = -0.197, almost zero skews (normal distribution).

   When to say the data is imbalanced? There are 3 degrees of imbalance based on the percentage of the minority class:

   - Mild (20-40% of the data set)
   - Moderate (1-20% of the data set)
   - Extreme (<1% of the data set)

   The percentage of our minority calss0 = 20895/ (147179+120895) = 49.09 %, which is more than 40% of the mild degree. So we can consider the data as balanced for now.

2. **Feature engineering and data cleaning:**

   The longest_word column has 8 Nans cells. Also, all these records are belonging the minority class 0. So, I have 2 options:

   - Remove these records as they are very few. 0.0066 % of the class 0.
   - Keep all the records of the minority class and just replace the 8 values with the mode (the most stable and frequent value) of the longest_word column for class 0.

   And I've applied the second option as it is the minority class and don't want to lose any of it. The data type of timestamp, longest_word, and sld is an object so we need to convert them into Integers type.

   We can ignore the timestamp column as it's just a record ID, moreover, we are dealing with stateless features. And for longest_word, and sld I converted their string values with the length, as those features were meant for.

3. **Split the data:**

   Split the data into Train: Test as 70:30, then divide the training data into Train: Validation as 70:30 respectively.

   The validation data is very important for Feature selection and hyperparameter tuning. Test data for champion model selection and final results evaluation.

   Note: I could have used K-fold cross-validation but it will take more time and I will have to implement it from scratch to apply feature selection methods on the training data only of each loop. (More complex).

4. **Baseline model:**

More intuitive to have a baseline performance. For the model selection:

- I need a fast training model. I don't care about accuracy at this point because I need this model to apply different feature selection methods with different K numbers of features.
- I am not interested in model explainability at this point.
- The training data is too large.

So, the answer is the Naive Bayes model. (figure 8,9 shows the baseline performance)

5. **Feature Filtering/Selection:**
   - **Anova**: for numerical features and categorical targets Anova is widely used as a feature selection algorithm. It separates the valuable features from the noise. Figure 10 shows the results of Anova compared with the baseline, and we can use only 3 features instead of the whole 14 features and get the same results as the baseline.
   - **MRMR (Maximum Relevance - Minimum Redundancy):** as the name states, it tries to select the K features that have the highest F-statistics with the target (Maximum Relevance) with the lowest Pearson correlation with each other (Minimum Redundancy). Produced by the Uber research team.

   From the results of MRMR (figure 11), with only 5 features we can get a higher f1-score than the baseline.

   MRMR feature selection has proven to be more effective than Anova as even with the same number of features 3 it was able to get the most important features and have a higher f1-score.

   MRMR best features = ['numeric', 'len', 'special', 'lower', 'subdomain_length'], those are the most important features that I will use in training the models. And the selected features make sense of removing the redundant features and using the most relevant to the target class like what I have observed in Figure 12.

   Note: The results are in ascending order of feature importance.

6. **Model Training:**
   - After splitting the data into training, validation, and testing, Some ML models assume input data with Gaussian distribution like Gaussian Naive Bayes (The baseline model). So, I applied Standard Scalar on the features to have mean =0 and standard deviation =1.
     Note: the Standard Scalar is fitted only on the training data then transforming the validation and test.
   - Another important use of the **validation** data is the model's hyperparameters tuning.

- Although the data is nearly balanced, I've used the F1-score metric in feature selection, hyperparameters tuning, and model selection. It's best to include both precision and recall because the costs of false positives and false negatives differ significantly and they have a different impact on the enterprise networks.
- **First model CatBoostClassifier:** CatBoost has a significant amount of performance potential and works remarkably well with the default settings, improving performance when tuned. It has many valuable features, but the one that's very powerful for this project, specifically after deployment is that it has the fastest prediction time when compared with XGBoost and LightBGM. Moreover, it's less prone to overfitting. The speed of processing and prediction time is as important as the prediction accuracy. The solutions in cybersecurity come in favor of one over the other, and to find one that combines both of them is a success.
- **Second model Multi-layer Perceptron**: When it comes to accuracy and performance, then we should try NN especially when a large dataset is available like that we have.
- For hyperparameters tuning I've used **BayesSearchCV** which is an informed search algorithm that learns from the previous iterations and makes use of it. It's much more efficient than the grid search as it doesn't try all the possibilities.

Figure 13, and 14 shows the results of before and after tuning the two models concerning the baseline.

7. Model evaluation:

Using the best features and the best hyperparameters I've trained both of the models on the training + validation, then compared the results using the test data. The evaluation metrics are f1-score, Recall, Precision, and the prediction time in ascending order of their importance. From the results, at figure 15 we can see that CatBoost has a higher F1 score and both of them are having identical classification reports (same Recall, and Precision). So we are left with F1-score and the prediction time. From Figures 16, and 17 CatBoost has the highest F1 score. Also, it has a lower prediction time than MLP and slightly higher than the baseline prediction time. F1-score is more important than the prediction time, and the difference between the baseline time and CatBoost is just 20 ms.

The champion model is **CatBoost**. Now, there's no need for splitting and wasting the data. So, I trained the selected model on the whole data and saved it.

Dynamic Model

After Deploying, the model will face some changes in the new data like new data distribution and concept drift which badly affects the performance. It's very important to set up the model so that it can adapt to the changes and perform better.

Analysis preparation: Simulating streaming data and using a window of 1000 records for each iteration. The evaluation metric is the F1-score for the same reasons as explained before, specifically, the mean of the F1-score over the iterations, plus the count of re-training and the sum of re-training times. Each applied method should have strengths and weaknesses related to accuracy and to the resources and time limitations. Both the dynamic and static models are initialized with the saved model and use the same features.

Model Re-training: I've created 2 functions for retraining.

- **Forgetting Learner:** We know that it's impractical to use all the historical and upcoming data to maintain and retrain the model as it requires infinite storage and runtime. I decided to use a fixed-length window to retrain the model. This method has drawbacks also and at some point, the model will forget about the old dataset and learn from the new one only.
- **Remember Learner**: To solve the drawback issue of the Forgetting Learner, I've split the training window into two halves; the first is randomly selected from the static data, and the second half is the most recent data. This method also has a weakness when the static data is no longer needed.

Briefly, Forgetting Learner is for the type of change that is always new, and Remember Learner is for the type of change that is reoccurring. (Figure 18 from lecture notes)

When to re-train?

1- **Performance Degradation – Forgetting learner :**
When the dynamic model F1-score decreases for 2 iterations respectively. It has drawbacks when the decrease is much larger than the increase, moreover, if the F1-score increases then decrease every consecutive iteration, the model will never retrain (worst-case scenario). The training window is equal to the length of the static data (268074) as the static model was trained. Figure 20, shows the Dynamic and Static model performance.

2- Performance Degradation – Forgetting learner: but with less training window = 50000. Figure 21, shows the Dynamic and Static model performance.

3- Performance Degradation – Remember learner: training window = 268074.
Figure 22, shows the Dynamic and Static model performance.

4- When the F1-score gets lower than a threshold – Remember learner: back to the F1-score of the static model it was 0.86 so it's not acceptable to have a very low value far

from 0.86. A threshold of 0.84 would be relatively acceptable. Training window = 268074.
Figure 23, shows the Dynamic and Static model performance.

Methods comparisons:

Methods	Static F1 mean	Dynamic F1 mean	Re-train time	Re-train Count
Number 1	0.8557170433903344	0.8558752739336878	1376.23	49
Number 2	0.8557170433903344	0.855781449676285	511.33	46
Number 3	0.8557170433903344	0.8558833611565444	1382.85	49
Number 4	0.8557170433903344	0.855874727319266	339.86	12

The first thing to notice is that all the Dynamic F1-score means are greater than the static model. However, it comes with the re-training overhead and re-training time with the resources. Method number 3 has the highest F1 mean and when comparing it with Method number 1 we find that Remember learner is slightly better than the Forgetting learner. Again the highest F1-mean has the highest re-training time.

Methods 1, and 3 are specifically for comparing the Forgetting learner and the Remember Learner (Remember Learner is better in our case meaning the type of change is reoccurring. figure 18).

Method 2 is to see the impact of using fewer training window sizes and compared them with 1. The F1-mean is slightly less but the re-training time difference is large.

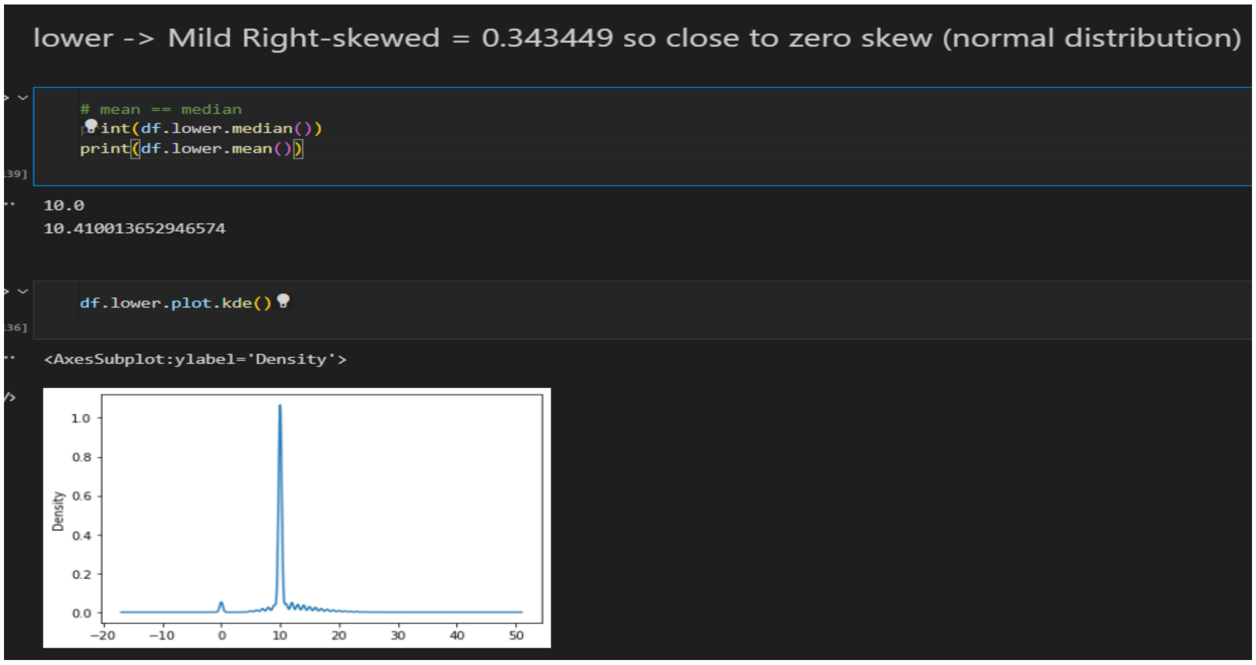
After measuring the trade-offs between The F1-mean and re-training time, I would choose method number 4 as it has the lowest re-training time and the difference in F1-mean between it and the highest F1-mean is very small (.00001).

However, if I am in a situation where I care the most about the F1-mean, I would use method number 4 by raising the threshold and maybe increasing the training window.

References

1- Check imbalance: https://developers.google.com/machine-learning/data-prep/construct/sampling-splitting/imbalanced-data#:~:text=A%20classification%20data%20set%20with,smaller%20proportion%20are%20minority%20classes
2- Feature understanding:http://ahlashkari.com/Datasets-DNS-Exfiltration-Traffic.asp
3- Model selection guide: https://blogs.sas.com/content/subconsciousmusings/2020/12/09/machine-learning-algorithm-use/
4- MRMR: https://lnkd.in/dHxhC3S
5- CatBoostClassifier: https://neptune.ai/blog/when-to-choose-catboost-over-xgboost-or-lightgbm
6- BayesSearchCV: https://towardsdatascience.com/grid-search-vs-random-search-vs-bayesian-optimization-2e68f57c3c46
7- Skewness: https://www.scribbr.com/statistics/skewness/

Appendix



lower -> Mild Right-skewed = 0.343449 so close to zero skew (normal distribution)

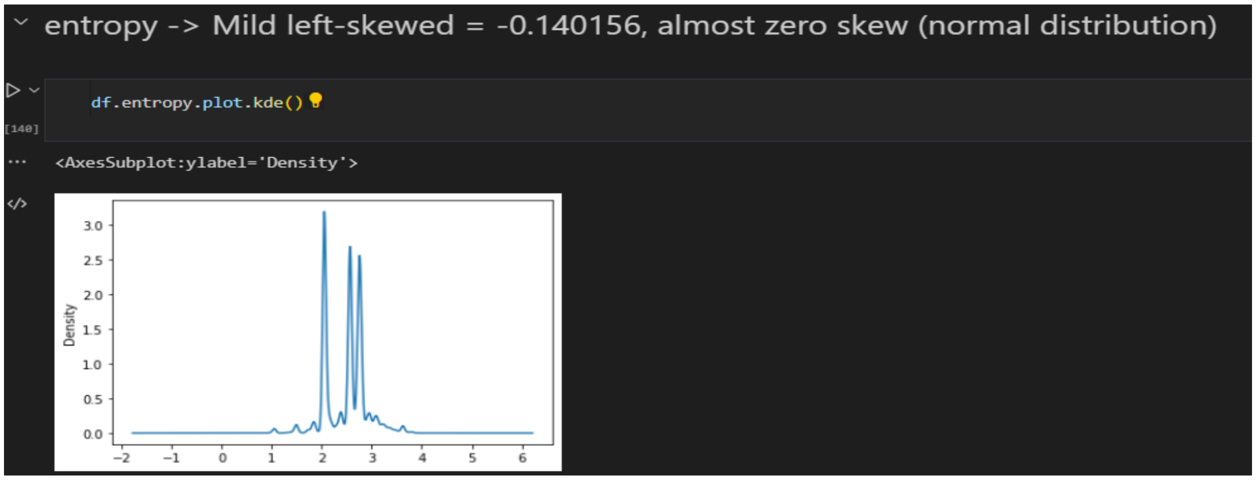
```
# mean == median
print(df.lower.median())
print(df.lower.mean())
```

```
10.0
10.410013652946574
```

```
df.lower.plot.kde()
```

```
<AxesSubplot:ylabel='Density'>
```

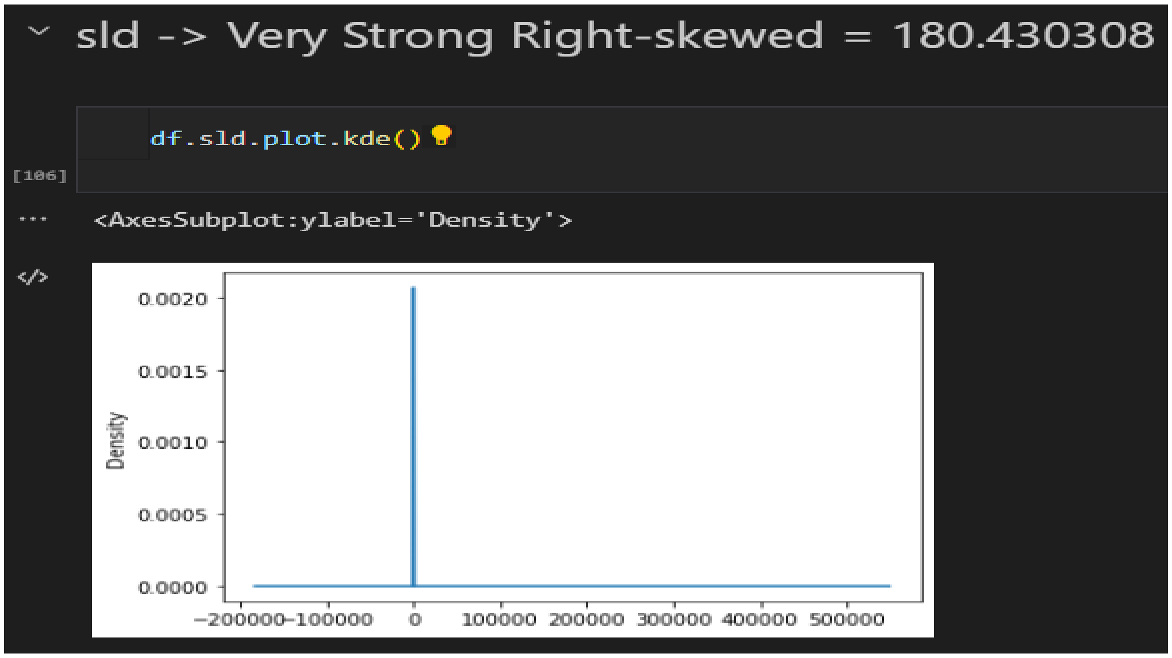
Figure 1. ‘lower’ feature distribution


entropy -> Mild left-skewed = -0.140156, almost zero skew (normal distribution)

```
[140]
df.entropy.plot.kde()
```

```
<AxesSubplot:ylabel='Density'>
```

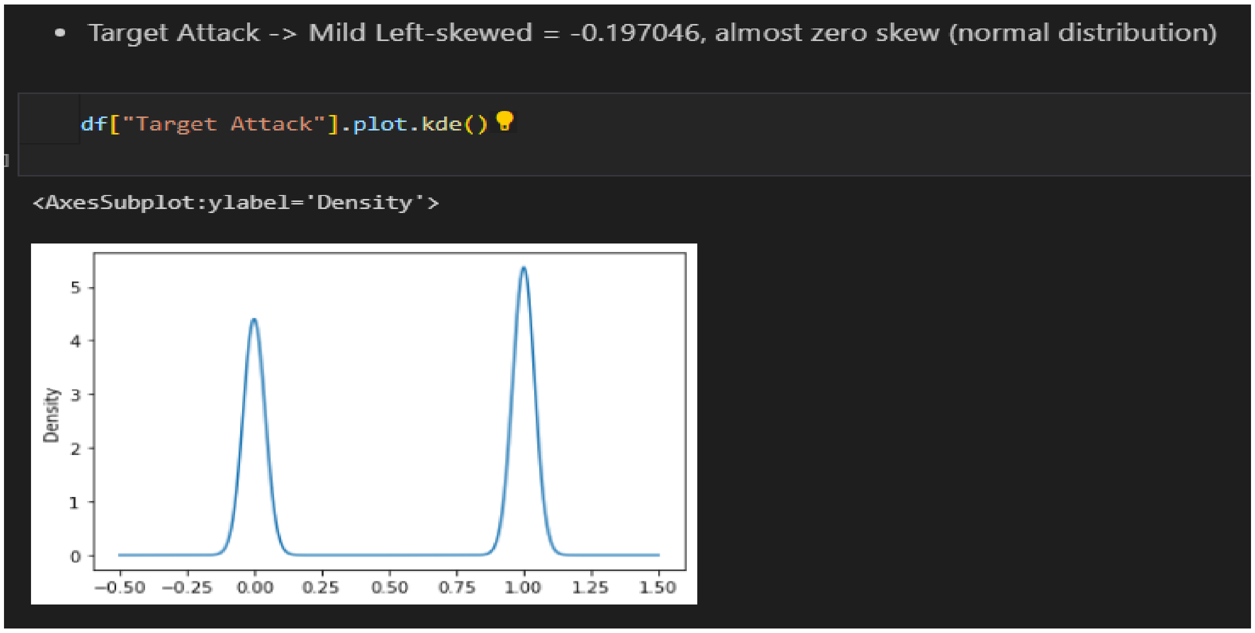
Figure 2. ‘entropy’ feature distribution


sld -> Very Strong Right-skewed = 180.430308

```
[106]
df.sld.plot.kde()
```

```
<AxesSubplot:ylabel='Density'>
```

Figure 3. ‘sld’ feature distribution



- Target Attack -> Mild Left-skewed = -0.197046, almost zero skew (normal distribution)

```
df["Target Attack"].plot.kde()
```

```
<AxesSubplot:ylabel='Density'>
```

Figure 4. ‘Target Attack’ label class distribution

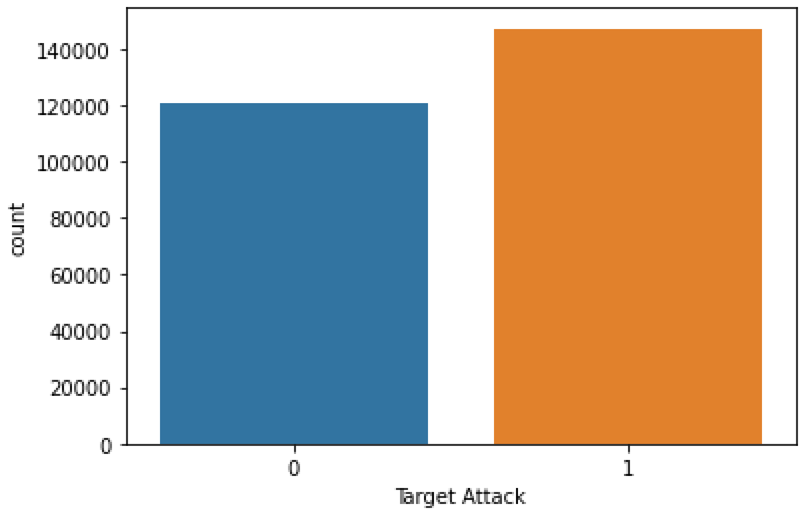


Figure 5. ‘Target Attack’ label class count distribution

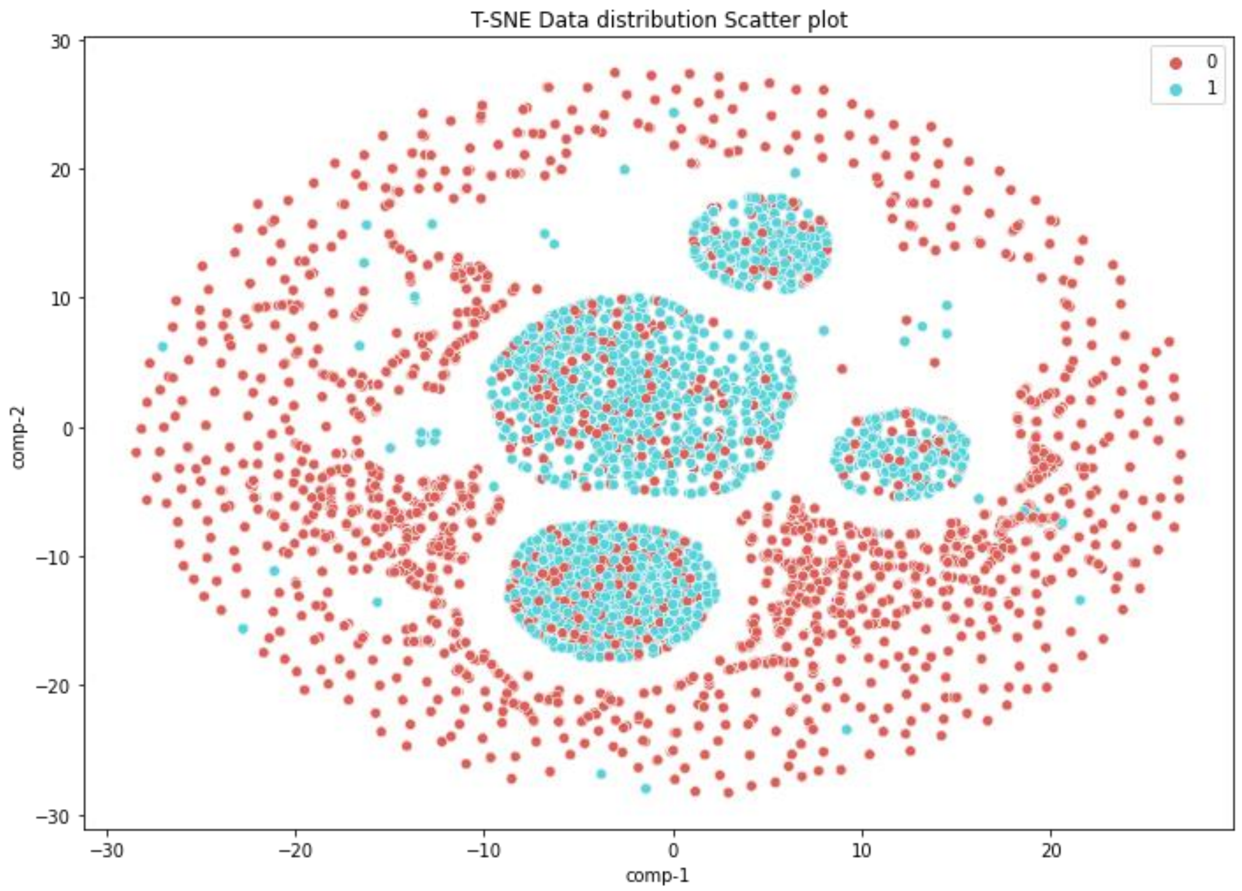


Figure 6. Whole data distribution using TSNE

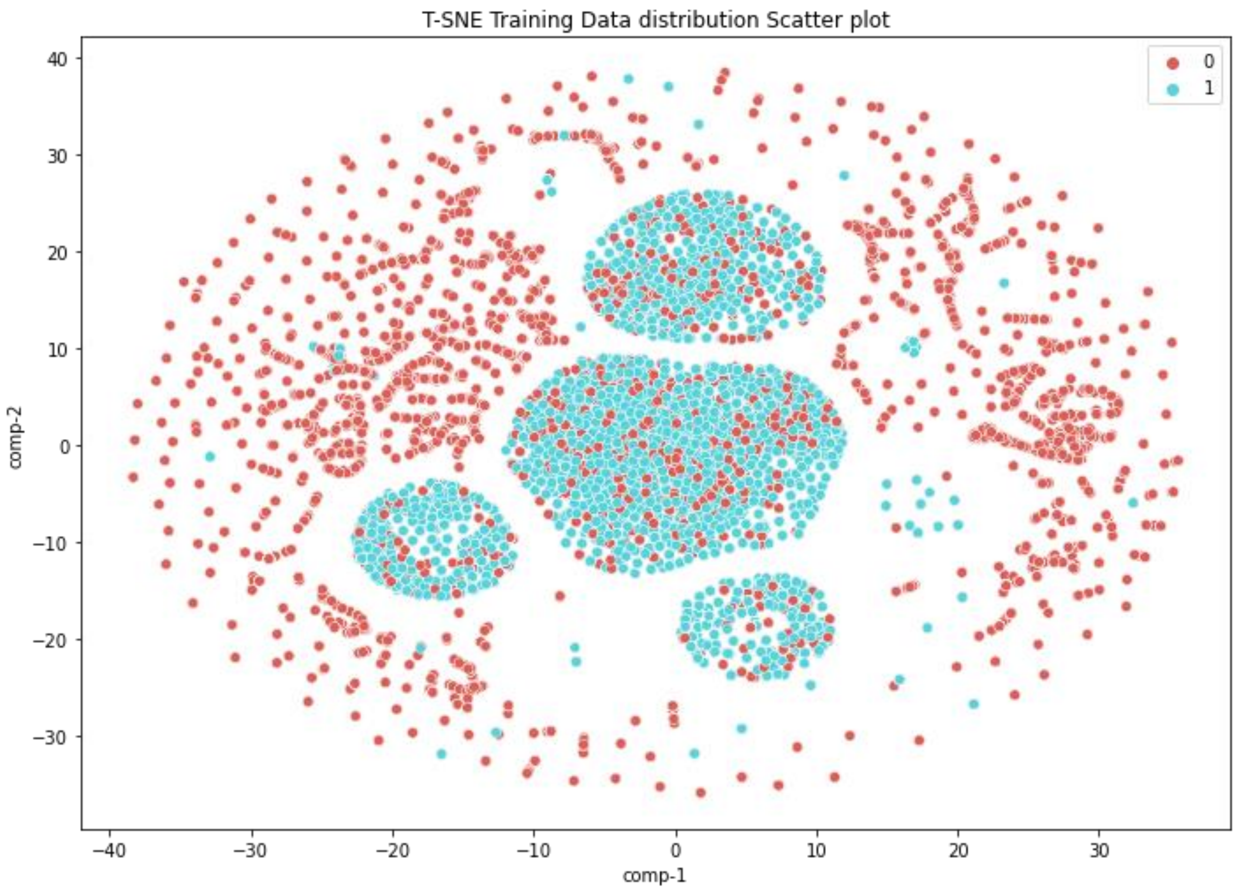
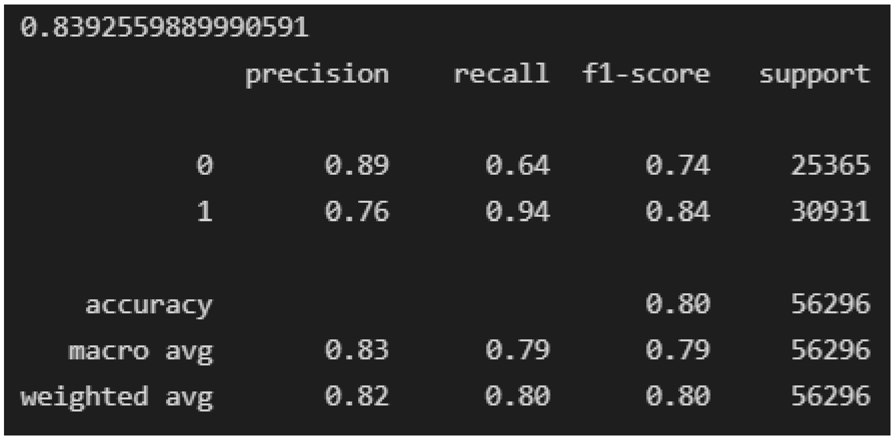
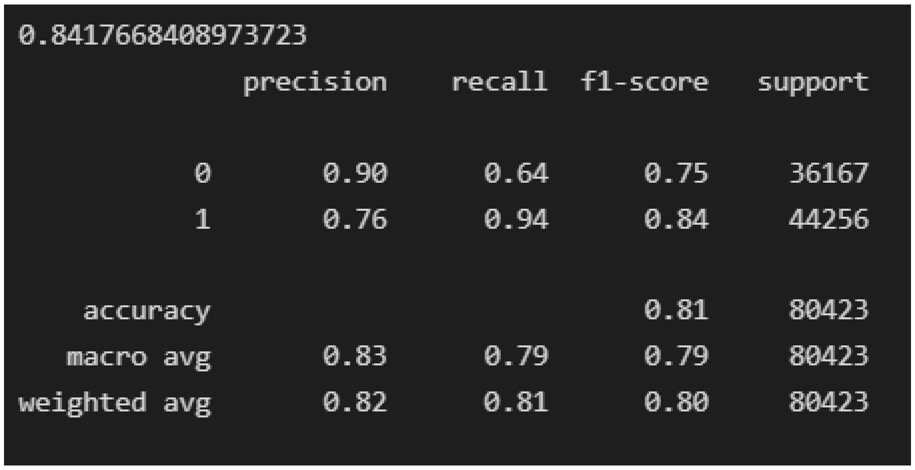


Figure 7. Training data distribution using TSNE



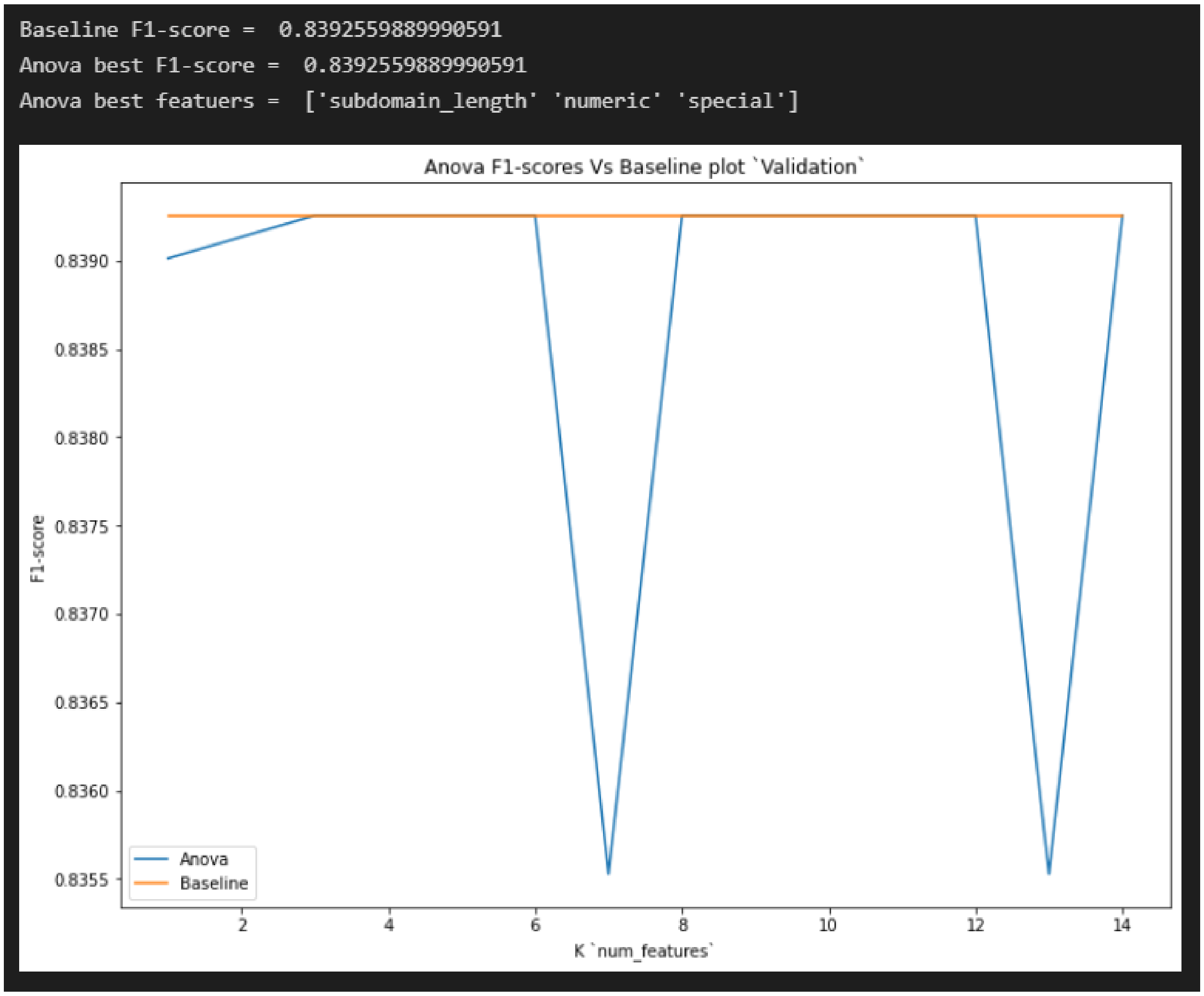
```
0.8392559889990591
              precision    recall  f1-score   support

           0       0.89      0.64      0.74     25365
           1       0.76      0.94      0.84     30931

    accuracy                           0.80     56296
   macro avg       0.83      0.79      0.79     56296
weighted avg       0.82      0.80      0.80     56296
```

Figure 8. Baseline Validation results f1-score = 0.839



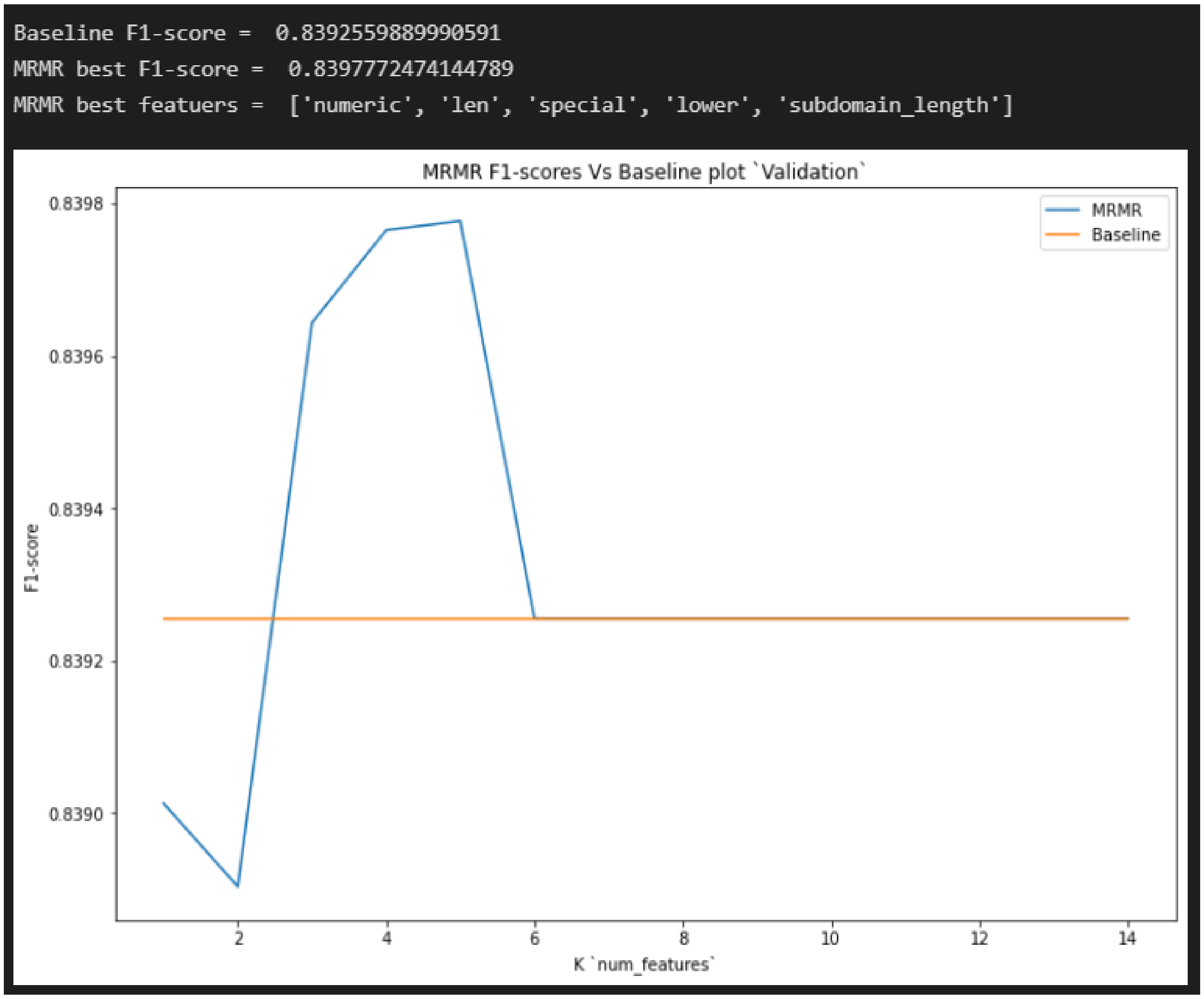
```
0.8417668408973723
              precision    recall  f1-score   support

           0       0.90      0.64      0.75     36167
           1       0.76      0.94      0.84     44256

    accuracy                           0.81     80423
   macro avg       0.83      0.79      0.79     80423
weighted avg       0.82      0.81      0.80     80423
```

Figure 9. Baseline Test results f1-score = 0.8417



```
Baseline F1-score =  0.8392559889990591
Anova best F1-score =  0.8392559889990591
Anova best featuers =  ['subdomain_length' 'numeric' 'special']
```

Figure 10. Anova Vs Baseline validation results



```
Baseline F1-score =  0.8392559889990591
MRMR best F1-score =  0.8397772474144789
MRMR best featuers =  ['numeric', 'len', 'special', 'lower', 'subdomain_length']
```

Figure 11. MRMR Vs Baseline validation results

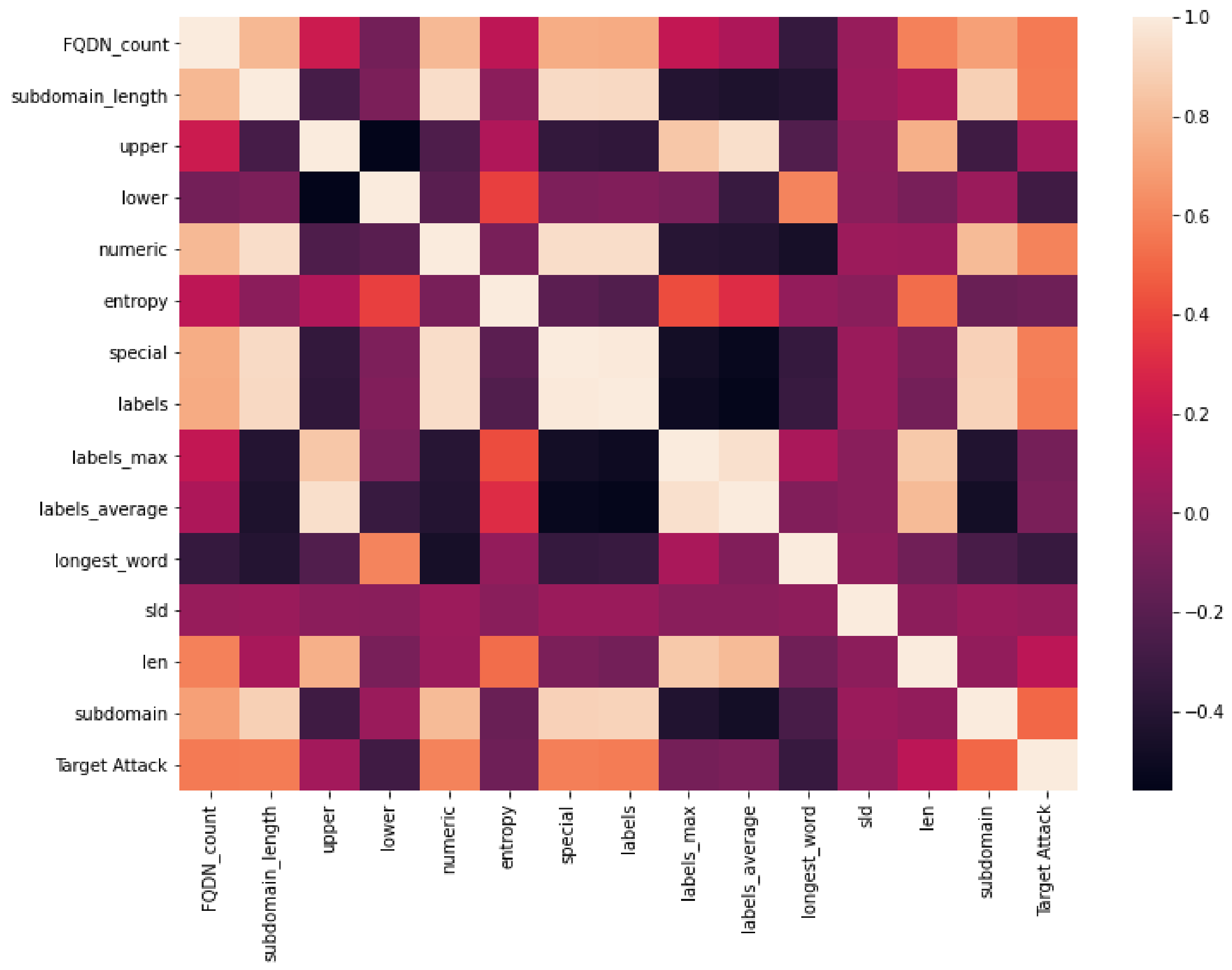


Figure 12. Heat map. Obviousely there are some redundant features which are highly correlated to each other like labels_max with labels_average, and special with labels. However, I leave them to see how the feature selection algorithms will act.

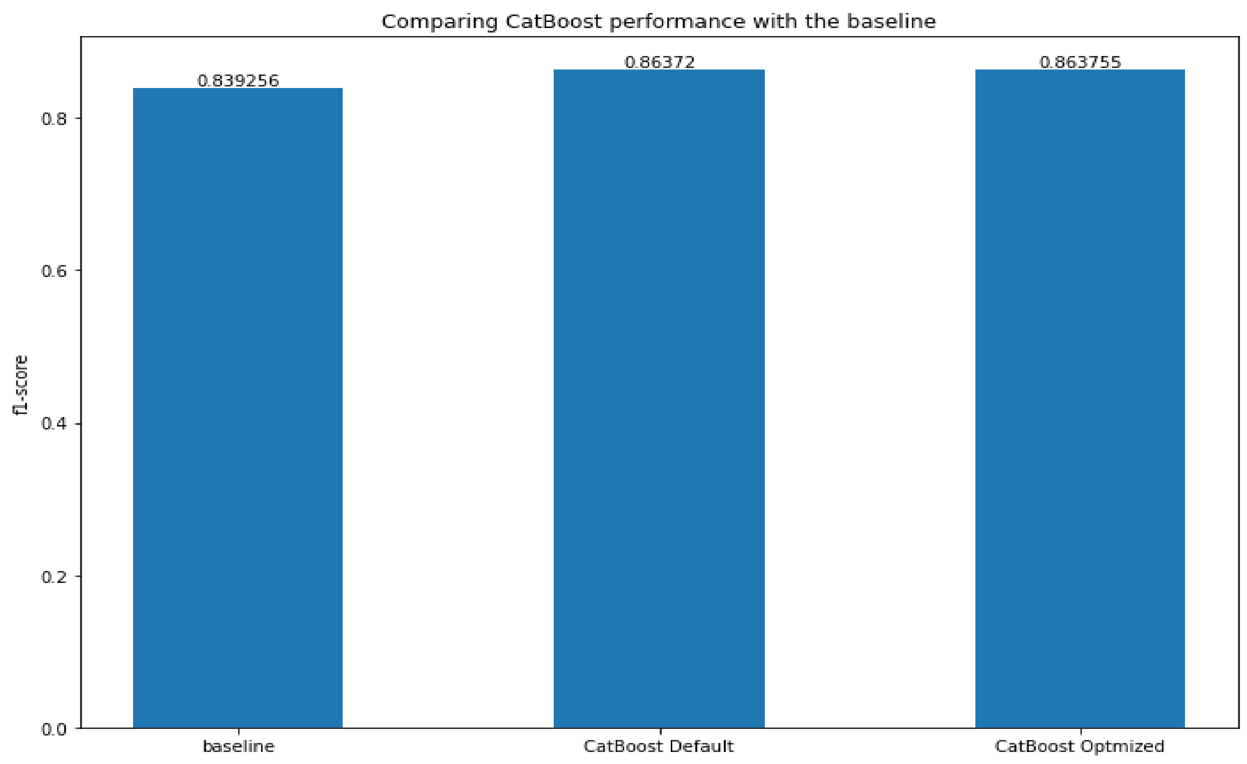


Figure 13. CatBoost has a higher f1-score than the baseline even with its default parameters. After tuning, the score didn't improve that much. The best hyperparameters [('depth', 7), ('iterations', 1000), ('l2_leaf_reg', 1), ('learning_rate', 0.15)]

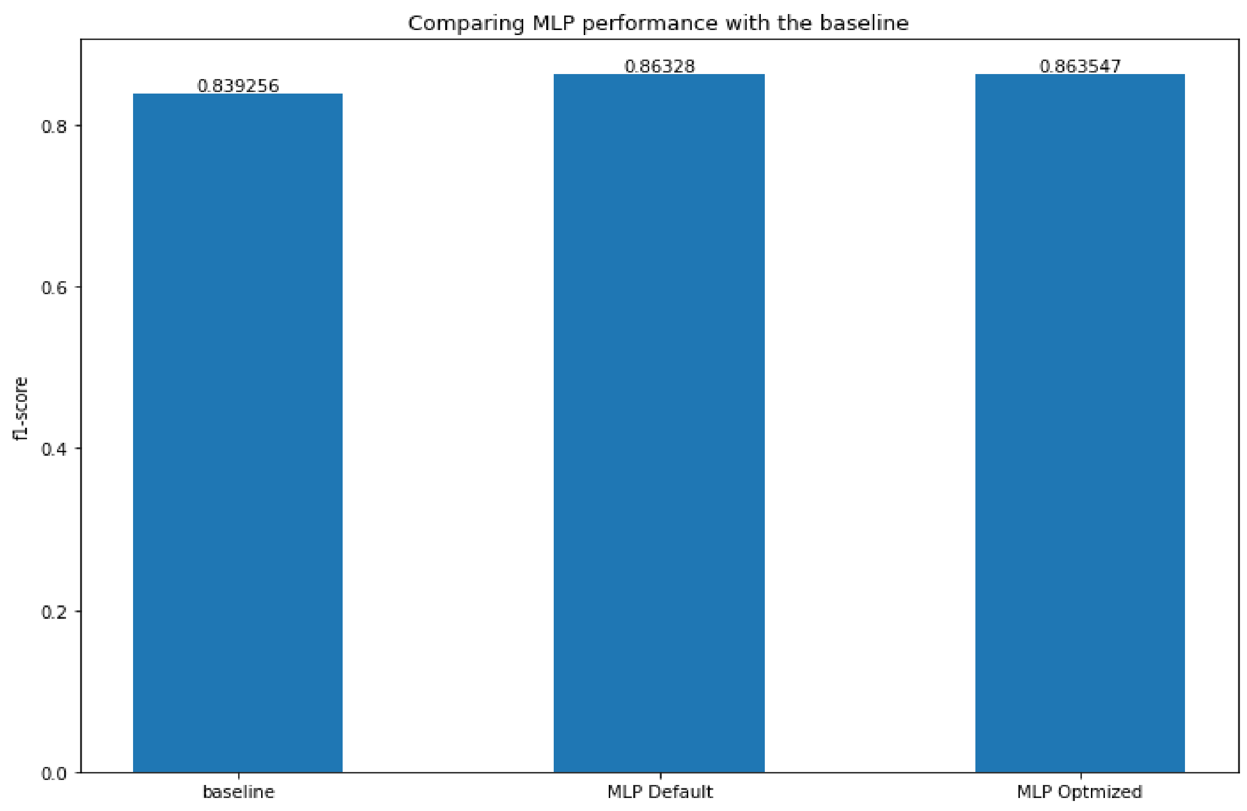
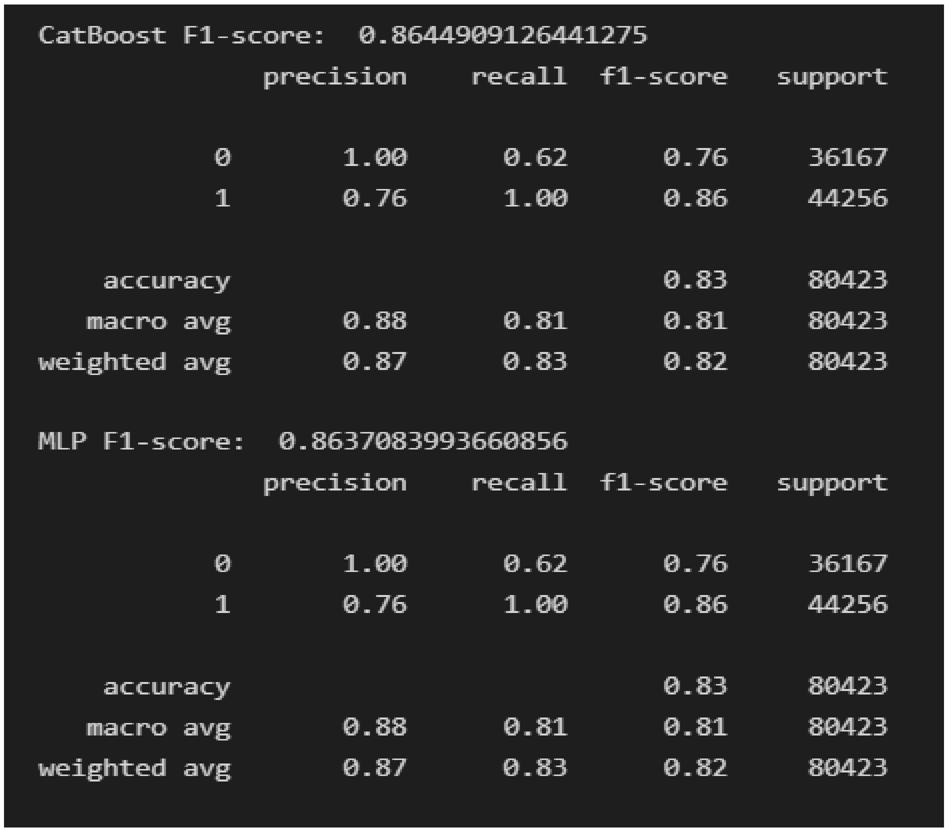


Figure 14. MLP has a higher f1-score than the baseline even with its default parameters. After tuning, the score didn't improve that much. The best hyperparameters: [('activation', 'logistic'), ('alpha', 0.0001), ('learning_rate', 'constant'), ('learning_rate_init', 0.01), ('max_iter', 100)]



```
CatBoost F1-score:  0.8644909126441275
              precision    recall  f1-score   support

           0       1.00      0.62      0.76     36167
           1       0.76      1.00      0.86     44256

    accuracy                           0.83     80423
   macro avg       0.88      0.81      0.81     80423
weighted avg       0.87      0.83      0.82     80423

MLP F1-score:  0.8637083993660856
              precision    recall  f1-score   support

           0       1.00      0.62      0.76     36167
           1       0.76      1.00      0.86     44256

    accuracy                           0.83     80423
   macro avg       0.88      0.81      0.81     80423
weighted avg       0.87      0.83      0.82     80423
```

Figure 15. Catboost Vs MLP results

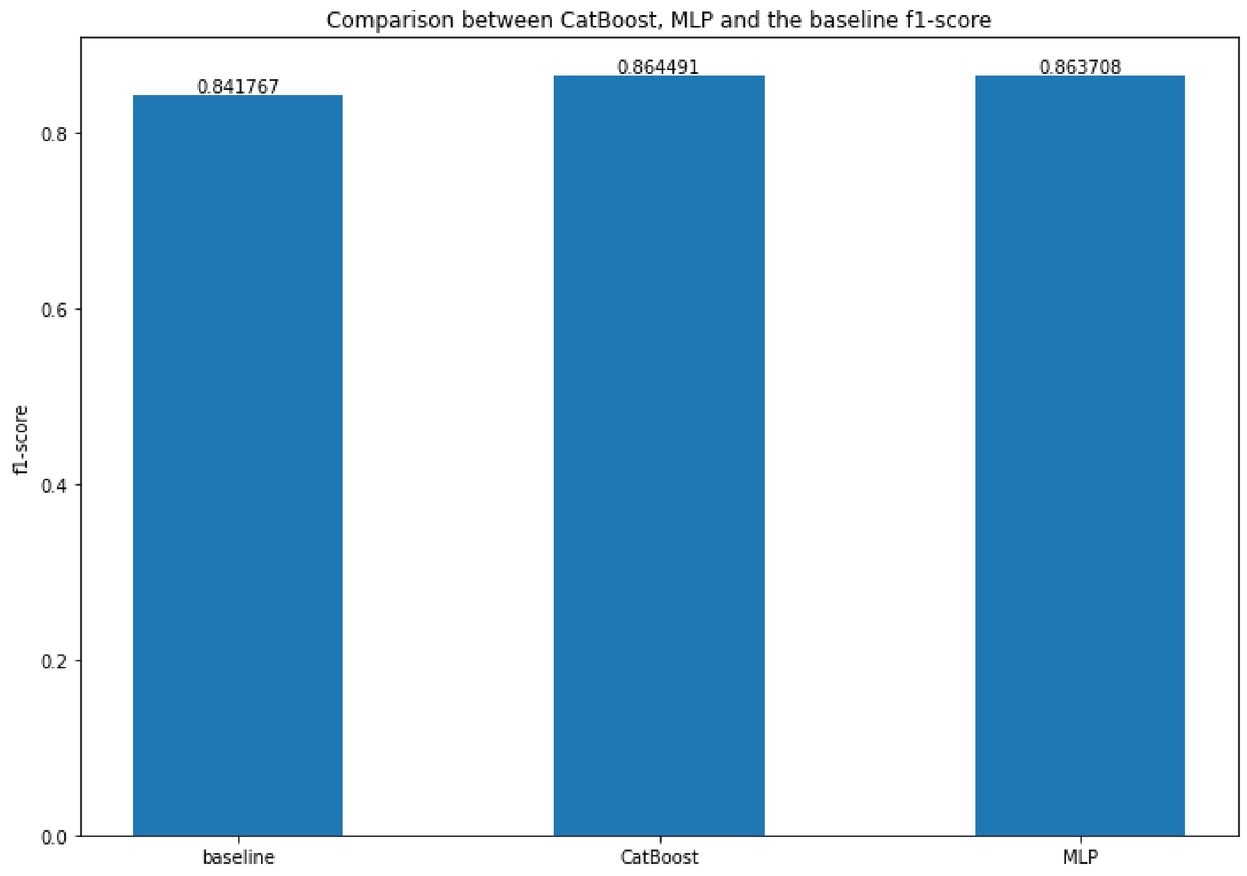


Figure 16. Baseline Vs CatBoost Vs MLP (F1-score metric)

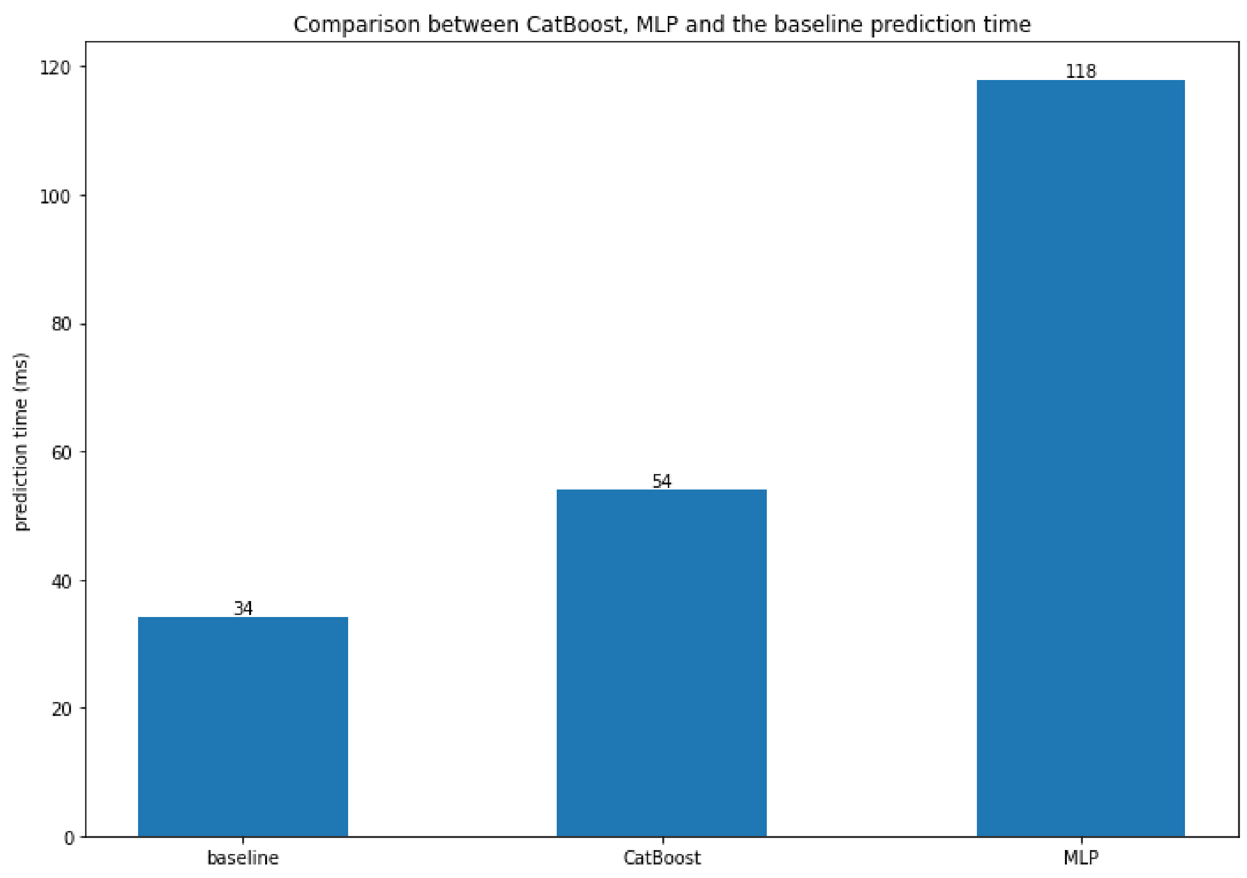


Figure 17. Baseline Vs CatBoost Vs MLP (prediction time)

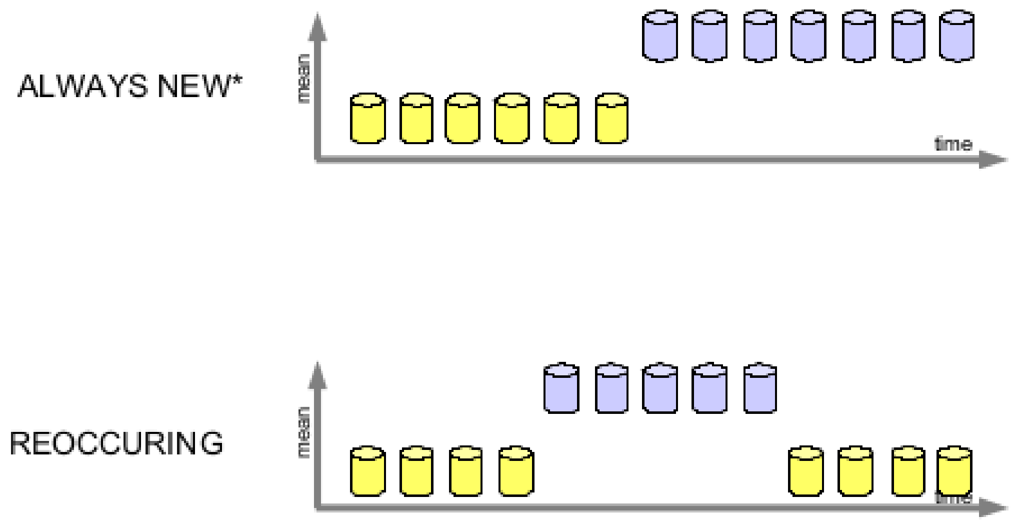


Figure 18. Types of Change. From lecture notes

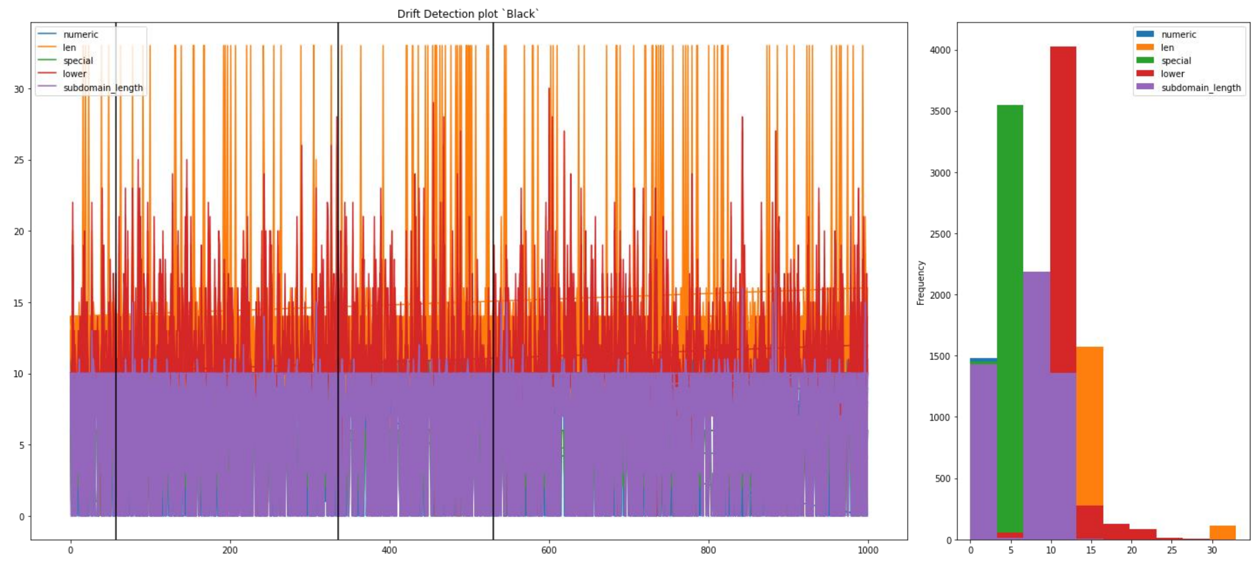


Figure 19. Concept drift detection.

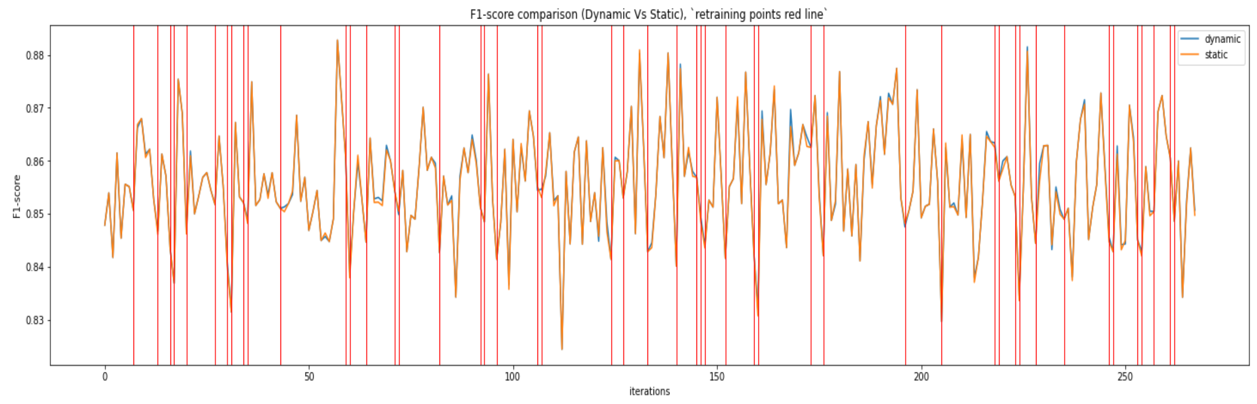


Figure 20. Dynamic Vs Static (F1-score). Vertical red lines for retraining points.

Performance Degradation – Forgetting learner (268074)

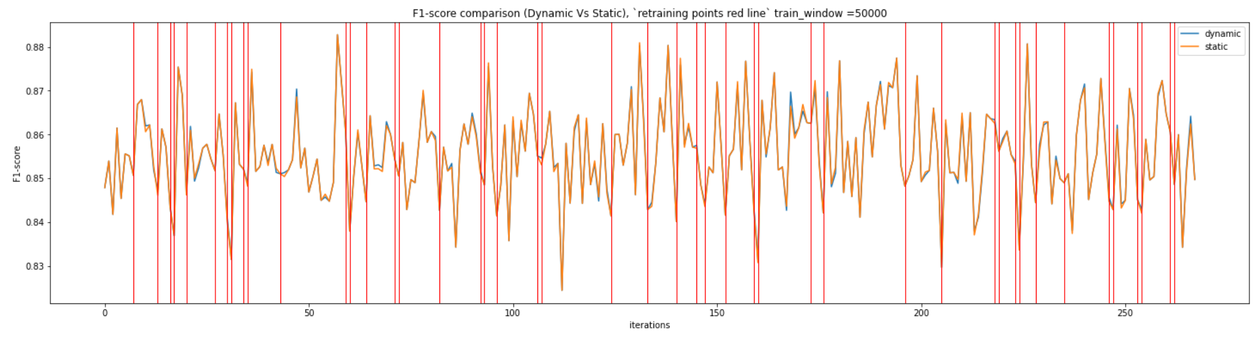


Figure 21. Dynamic Vs Static (F1-score). Vertical red lines for retraining points.

Performance Degradation – Forgetting learner (5000)

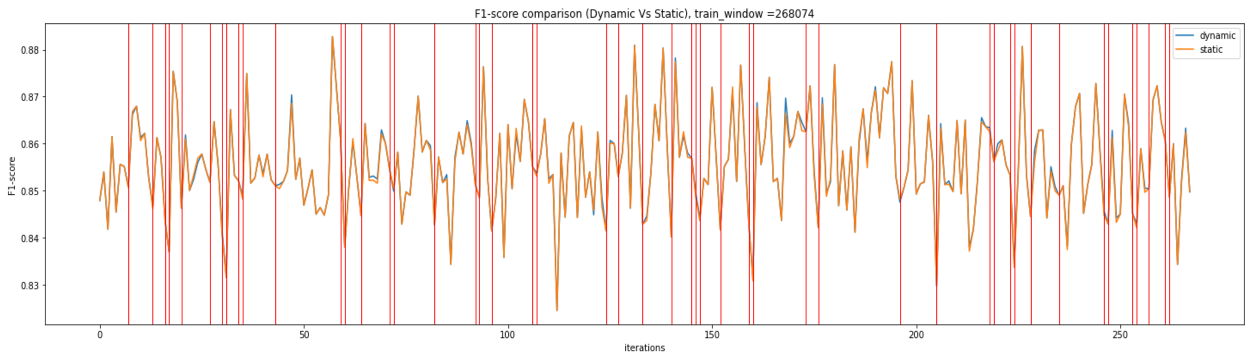


Figure 22. Dynamic Vs Static (F1-score). Vertical red lines for retraining points.

Performance Degradation – Remember learner (268074)

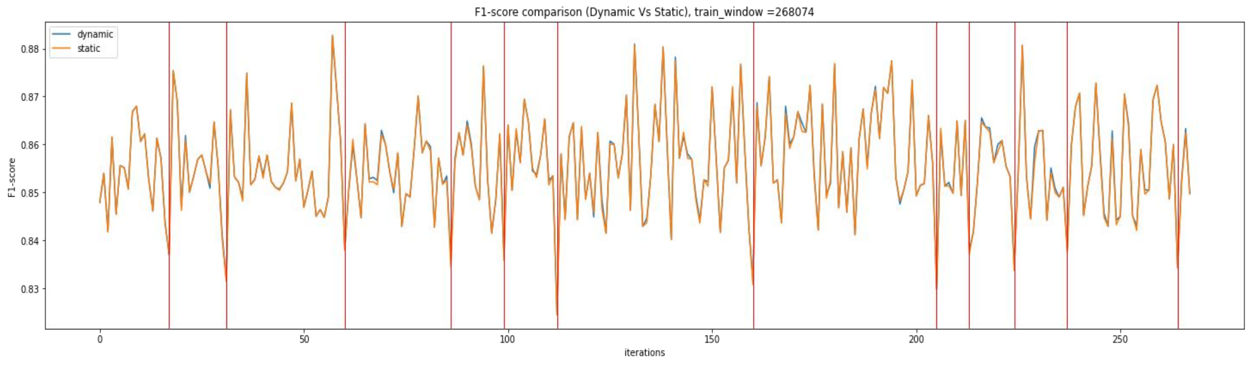


Figure 23. Dynamic Vs Static (F1-score). Vertical red lines for retraining points.

Threshold of 0.84 F1-score – training window = 268074

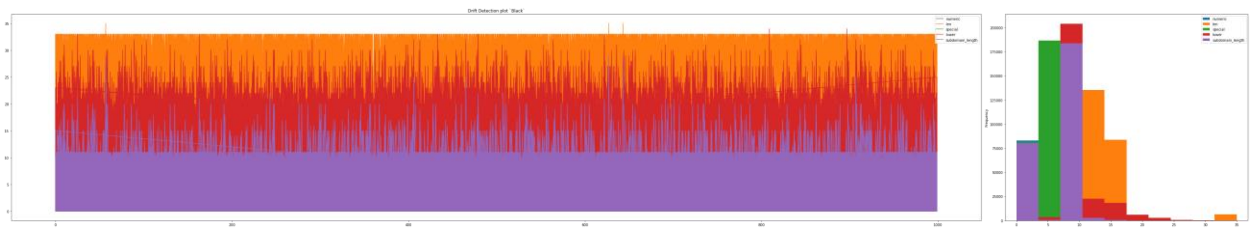


Figure 24. Streaming data distribution changing