FastHEAL- Malware Detection
Milestone-3

Team : Code Diva

Prarthana U Shanbhag – MT2022077

Karishma Chauhan – MT2022056

Contents

1. Recap
2. Changes in EDA
3. Models Implemented
4. Summary

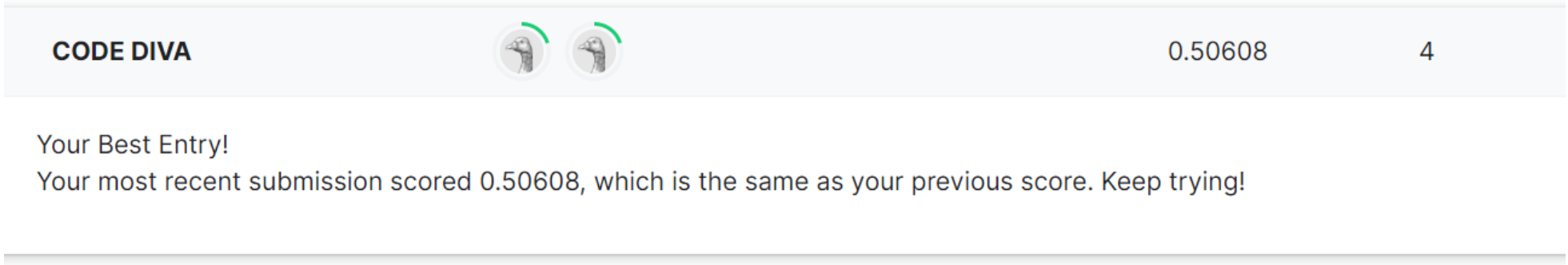
Where we were?

We have used Inbuilt Logistic Regression model in the implementation

- The score that we obtained is 0.50608 which is not the best possible solution model.

We realized that one of the reasons for such low accuracy could be highly unnormalized data

And Since logistic regression works by putting weights on the numeric values of data it does not perform well with categorical data and there are better alternative if data set is highly categorical



Revisiting EDA Process

1. Standard scaler: Since data was not normalized hence Standard scaler gave good improved accuracy.

2. Added Features: Few features we have eliminated on basis of the missing and imbalanced factor but did feature engineering and found few relevant features which have impact on malware detections need to be added.

3. Features added were :

   AVInstalled, Firewall,IsProtected, SMode and PuaMode,

4. Features added then removed were :

   Census_HasOpticalDiskDrive,Census_IsPenCapable, Processor,Census_OSArchitecture, and Platform

Models Implemented

1. Logistic Regression
2. Gaussian
3. SVM
4. Decision trees
5. Random forest
6. Adaboost
7. Multilayer Perceptron
8. Catboost
9. LGBM
10. XGBoost

Logistic Regression

So, we applied sklearn.preprocessing.StandardScaler on the data since data was not normalized

- We saw significant improvement in score for train(0.613) as well as on test submission on Kaggle (0.607)

Logistic Regression with default hyperparameters :

Train accuracy :

```
[125] # #finding accuracy
      print('Accuracy Score: ', (accuracy_score(Y_TEST, Y_pred)))

      Accuracy Score:  0.6127437930841226
```

Test accuracy :

M1_logistic.csv

Complete · Prarthana U Shanbhag · 22d ago

0.6079

Logistic Regression with hyperparameters tuning :

Logistic Regression Hyperparameters.

- The main hyperparameters we may tune in logistic regression are:
    1. solver,
    2. penalty, (l1 or l2)
    3. regularization strength(C)

Solver

- {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}, default='lbfgs'.
- **lbfgs** relatively performs well compared to other methods and it saves a lot of memory, however, sometimes it may have issues with convergence.
- **sag** is faster than other solvers for large datasets, when both the number of samples and the number of features are large.
- **saga** the solver of choice for sparse multinomial logistic regression and it's also suitable for very large datasets.
- **newton-cg** computationally expensive because of the Hessian Matrix.
- **liblinear** recommended when you have a high dimension dataset - solving large-scale classification problems.

Train data accuracy :

```
Accuracy Score:  0.6127437930841226
```

	Train Accuracy	Test Accuracy	Precission	Recall	AUC
0	0.61381	0.61273	0.61143	0.61722	0.61273
1	0.61380	0.61273	0.61143	0.61722	0.61273
2	0.61381	0.61273	0.61143	0.61723	0.61273
3	0.61381	0.61273	0.61143	0.61723	0.61273
4	0.61380	0.61273	0.61143	0.61722	0.61273

Test data accuracy :

M1_logistic_hyper.csv

Complete · Prarthana U Shanbhag · 21d ago · Logistic regression with hyperparameters

0.61151

Moving on from Logistic Regression….

We further explored below models :

1. Gaussian Model:

   The data set is not normally distributes and does follow the gaussian curve and hence as a result we results

2. SVM :

   SVM model did not converge : It is not suitable here because SVMs perform poorly in imbalanced datasets and data set is so high that SVM didn’t converge even after 10 hours of model training since

   SVMs are based around a kernel function. Most implementations explicitly store this as an NxN matrix of distances between the training points to avoid computing entries over and over again

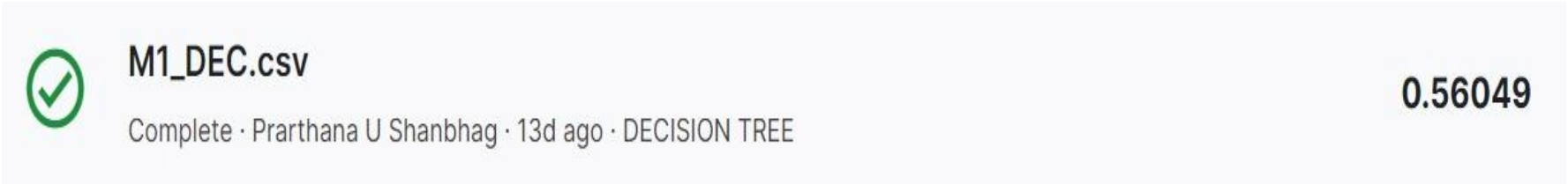
Decision trees

- Decision trees supports non linearity, where LR supports only linear solutions.
- When there are large number of features with less data-sets(with low noise), LR may outperform

Decision trees with default hyperparameters :

Train data : `accuracy_score 0.5716807711707672`

Test data :



Decision trees with hyperparameters tuning :

The few other hyperparameters that would restrict the structure of the decision tree are:

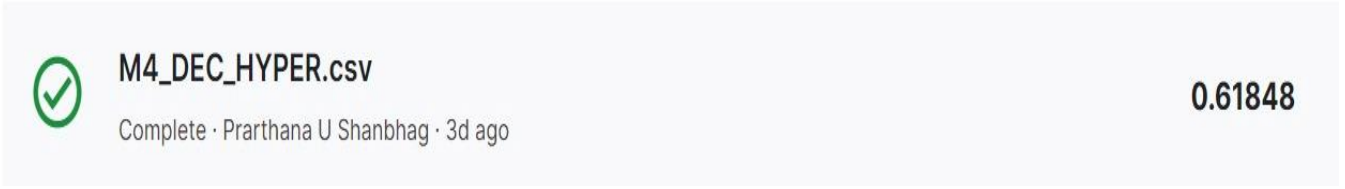
- min_samples_split – Minimum number of samples a node must possess before splitting.
- min_samples_leaf – Minimum number of samples a leaf node must possess.
- min_weight_fraction_leaf – Minimum fraction of the sum total of weights required to be at a leaf node.
- max_leaf_nodes – Maximum number of leaf nodes a decision tree can have.
- max_features – Maximum number of features that are taken into the account for splitting each node.
- max_depth = max number of levels in decision tree

With manual tuning :

Train data

```
accuracy_score 0.6203413103177716
```

Test data

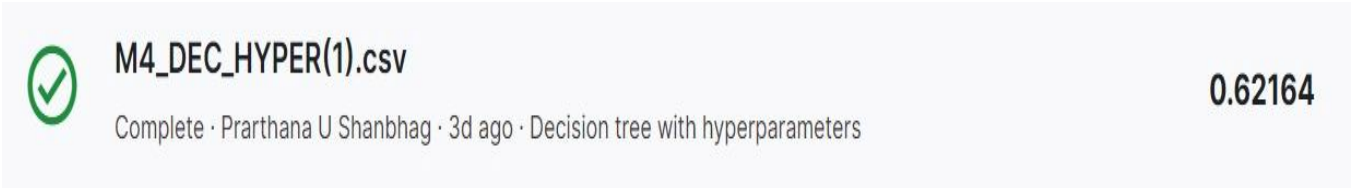


With Grid Search :

Train data

```
accuracy_score 0.6309407050383904
```

Test data



Random Forest

- A decision tree combines some decisions, whereas a random forest combines several decision trees and hence assembles randomized decisions based on many decisions and then creates a final decision depending on the majority.
- Random Forest uses the bagging technique. It is an ensemble method that consists in generating many little decision trees taking different random samples of the original dataset. Each decision tree makes its own prediction, which are combined to generate a much more accurate prediction.

Random Forest with default hyperparameters :

Train data : `accuracy_score 0.63094`

Test data :

Random Forest with hyperparameters tuning :

Hyper parameters :

- n_estimators = number of trees in the foreset
- max_features = max number of features considered for splitting a node
- max_depth = max number of levels in each decision tree
- min_samples_split = min number of data points placed in a node before the node is split
- min_samples_leaf = min number of data points allowed in a leaf node
- bootstrap = method for sampling data points (with or without replacement)

Train data

```
accuracy_score 0.6514984868015469
```

Test data

M5_Randomforest.csv

Complete · Prarthana U Shanbhag · 2d ago · Randomforest with manual hyperparameter tuning

0.64558

We successfully manages to achieve accuracy improvement from 0.629 to 0.645 with Manual tuning

ADABoost

- This is a type of ensemble technique, where a number of weak learners are combined together to form a strong learner. In contrary to the random forest, here each classifier has different weights assigned to it based on the classifier's performance.
- Adaboost increases the predictive accuracy by assigning weights to both observations at end of every tree and weights(scores) to every classifier. Hence, in Adaboost, every classifier has a different weightage on final prediction.

Train data : `Accuracy: 0.6299949560051561`

Test data :

M7_adaboost.csv

Complete · Prarthana U Shanbhag · 2d ago

0.62876

MultiLayered Perceptron

- Multilayer perceptron (MLP) is a technique of feed-forward artificial neural networks using a back propagation learning method to classify the target variable used for supervised learning.
- MLP's can be applied to complex non-linear problems, and it also works well with large input data with a relatively faster performance.

Train data :

```
clf = MLPClassifier(hidden_layer_sizes=(120),
                    random_state=1,
                    verbose=True,
                    activation='logistic',solver='sgd',
                    learning_rate_init=0.01)
```

```
Accuracy: 0.6659614414616376
```

Test data :

3.csv

Complete · Prarthana U Shanbhag · 11d ago · Multilayer Perceptron

0.65062

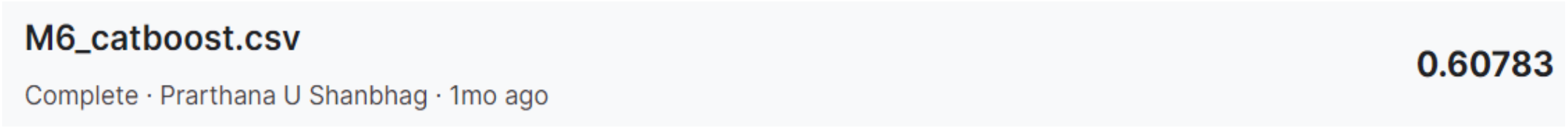
CatBoost

- CatBoost uses the boosting technique. It is also an ensemble method, but it consists in generating decision trees one after another, where the results of one tree are used to improve the next one, and so on.
- CatBoost is great for processing categorical features but tuning is more complex for boosting based techniques hence we are working on hyper tuning the boosting technique based models

CatBoostwith default hyperparameters :

Train data `accuracy_score 0.6342228044611332`

Test data



CatBoostwith hyperparameters tuning :

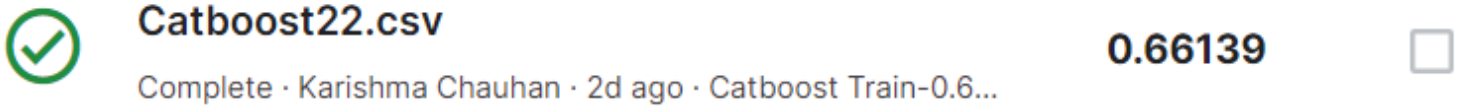
Train data

```
bestTest = 0.6048492057
bestIteration = 2499

CatBoost model is fitted: True
CatBoost model parameters:
{'iterations': 2500, 'learning_rate': 0.1}
```

```
accuracy_score 0.6623255618449813
```

Test data



GBM

- Gradient Boosting technique also combines a no. of weak learners to form a strong learner. The residuals are captured in a step-by-step manner by the classifiers, in order to capture the maximum variance within the data, this is done by introducing the learning rate to the classifiers.
- Hence in this method, we are slowly inching towards better prediction (This is done by identifying negative gradient and moving in the opposite direction to reduce the loss, hence it is called Gradient Boosting in line with Gradient Descent where similar logic is employed).
- Thus, by no. of classifiers, we arrive at a predictive value very close to the observed value.
- Two possible model using this technique :
  - Light GBM
  - XGBoost

LGBM

LGBM Manual hyperparameters tuning :

Train data

```
accuracy_score 0.6580031384856807
```

Test data

M1.csv

Complete · Prarthana U Shanbhag · 1d ago · LGBM test

0.65857

LGBM hyperparameters tuning using Grid Search :

Train data

```
Best score reached: 0.7224198690988102 with params: {'colsample_bytree': 0.9731668400523877, 'min_child_samples': 171, 'min_child_weight': 1e-05, 'num_leaves': 41, 'reg_alpha': 10, 'reg_lambda': 100, 'subsample': 0.5575732396028996}
accuracy_score 0.662875497393936
```

Test data

lgbm_hyper2.csv

Complete · Prarthana U Shanbhag · 2d ago · train-0.66387

0.66383

XGBoost

XGBoost default hyperparameters :

Train data Accuracy: 0.632853219750042

Test data

M5_test.csv

Complete · Prarthana U Shanbhag · 1mo ago · XGBoost

0.6291

XGBoost hyperparameters tuning using Grid Search :

Train data

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.4,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0.2, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.03, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=11, max_leaves=0, min_child_weight=7,
              missing=nan, monotone_constraints='()', n_estimators=4000,
              n_jobs=-1, nthread=-1, num_parallel_tree=1, predictor='auto',
              random_state=42, reg_alpha=0.6, ...)
```

Accuracy: 0.6659614414616376

Test data

xgboost_alltrain.csv

Complete · Karishma Chauhan · 2d ago · train-0.67514

0.66652

Summary of Models

Model Implement	Accuracy in training	Accuracy in testing
1.Logistic Regression		
a) Logistic with default hyperparameters	0.61274	0.6079
b) Logistic with hyperparameter tuning	0.61274	0.6115
2. Gaussian model	0.5747	0.59796
3. SVM	Did not converge	Did not converge
4. Decision trees		
a) With default hyperparameters	0.5716	0.5604
b) with manual hyperparameter tuning	0.62034	0.6184
c) with hyperparameter tuning using grid search	0.6309	0.6216
5. Random forest		
a) with default hyperparameter	0.63094	0.62917
b) with hyperparameter tuning	0.6514	0.64558

Model Implement	Accuracy in training	Accuracy in testing
6. Adaboost	0.6299	0.6287
7. Multilayer Perceptron	0.6659	0.65062
8. Catboost		
a) with default hyperparameter	0.6342	0.6078
b) with hyperparameter tuning	0.6623	0.66139
9. LGBM		
a) with manual hyperparameter tuning	0.6580	0.6585
b) with hyperparameter tuning	0.6628	0.6638
10. XGBoost		
a) with default hyperparameter	0.6328	0.6291
b) with hyperparameter tuning	0.6659	0.6665