MACHINE LEARNING PROJECT

SAVE THE ATTACK !

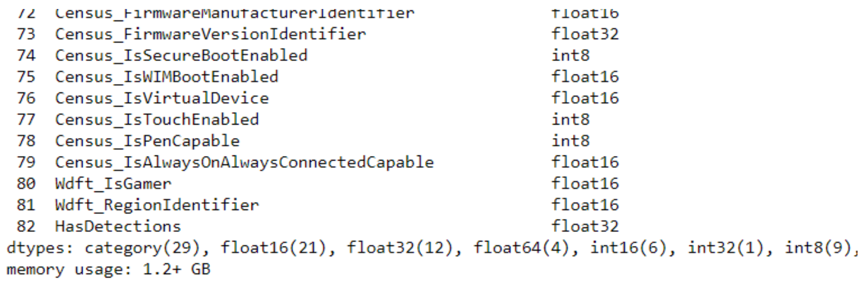
FINAL REPORT

Ayush Tiwari IMT2019015
Revanth Reddy IMT2019517

Dataset Details

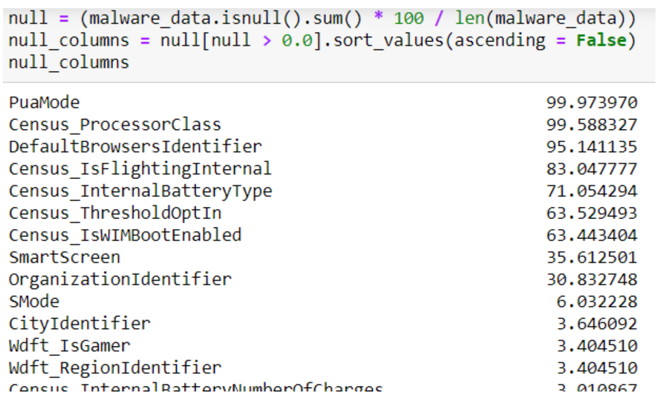
The training dataset namely 'train.csv' file provided has 7137943 rows.There are 83 columns where most of the columns are categorical.Each row corresponds to a machine uniquely identified by 'MachineIdentifier'. 'HasDetections' indicates whether malware was detected in the machine. Test data has 1783540 rows.

Preprocessing tasks



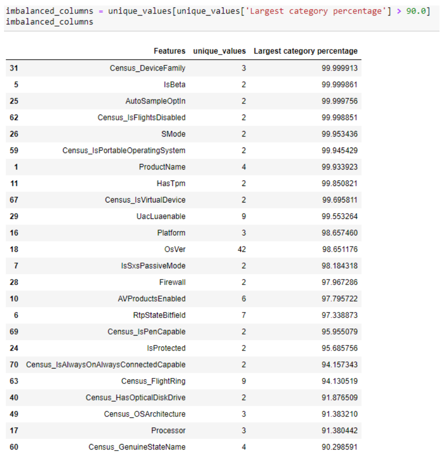
```
 72  Census_FirmwareManufacturerIdentifier              float16
 73  Census_FirmwareVersionIdentifier                   float32
 74  Census_IsSecureBootEnabled                         int8
 75  Census_IsWIMBootEnabled                            float16
 76  Census_IsVirtualDevice                             float16
 77  Census_IsTouchEnabled                              int8
 78  Census_IsPenCapable                                int8
 79  Census_IsAlwaysOnAlwaysConnectedCapable            float16
 80  Wdft_IsGamer                                       float16
 81  Wdft_RegionIdentifier                              float16
 82  HasDetections                                      float32
dtypes: category(29), float16(21), float32(12), float64(4), int16(6), int32(1), int8(9),
memory usage: 1.2+ GB
```

Since the size of traindata.csv was 4.4 gb , we made a few simple changes to the datatypes of the columns that reduced the total memory occupied by the dataset .
Modifying the datatype from int64 to int32 or lesser wherever possible , similar for float values and converting object dtype to categorical dtype which eventually helps us save a lot of memory. This reduces the memory occupied to nearly 1.2gb



```
null = (malware_data.isnull().sum() * 100 / len(malware_data))
null_columns = null[null > 0.0].sort_values(ascending = False)
null_columns
```

```
PuaMode                                  99.973970
Census_ProcessorClass                    99.588327
DefaultBrowsersIdentifier                95.141135
Census_IsFlightingInternal               83.047777
Census_InternalBatteryType               71.054294
Census_ThresholdOptIn                    63.529493
Census_IsWIMBootEnabled                  63.443404
SmartScreen                              35.612501
OrganizationIdentifier                   30.832748
SMode                                     6.032228
CityIdentifier                            3.646092
Wdft_IsGamer                              3.404510
Wdft_RegionIdentifier                     3.404510
Census_InternalBatteryNumberOfCharges     3.010867
```

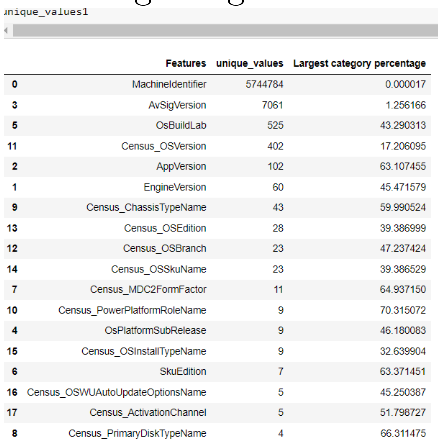


```
imbalanced_columns = unique_values[unique_values['Largest category percentage'] > 90.0]
imbalanced_columns
```

	Features	unique_values	Largest category percentage
31	Census_DeviceFamily	3	99.999913
5	IsBeta	2	99.999861
25	AutoSampleOptIn	2	99.999756
62	Census_IsFlightsDisabled	2	99.998851
26	SMode	2	99.953436
59	Census_IsPortableOperatingSystem	2	99.945429
1	ProductName	4	99.933923
11	HasTpm	2	99.850821
67	Census_IsVirtualDevice	2	99.695811
29	UacLuaenable	9	99.553264
16	Platform	3	98.657460
18	OsVer	42	98.651176
7	IsSxsPassiveMode	2	98.184318
28	Firewall	2	97.967286
10	AVProductsEnabled	6	97.795722
6	RtpStateBitfield	7	97.338873
69	Census_IsPenCapable	2	95.955079
24	IsProtected	2	95.685756
70	Census_IsAlwaysOnAlwaysConnectedCapable	2	94.157343
63	Census_FlightRing	9	94.130519
40	Census_HasOpticalDiskDrive	2	91.876509
49	Census_OSArchitecture	3	91.383210
17	Processor	3	91.380442
60	Census_GenuineStateName	4	90.298591

Removed columns more with more than 40% missing data Removed columns having the same category for most (90% or more) of the data. Now in total we have 52 columns.

Reducing categorical dimensions-



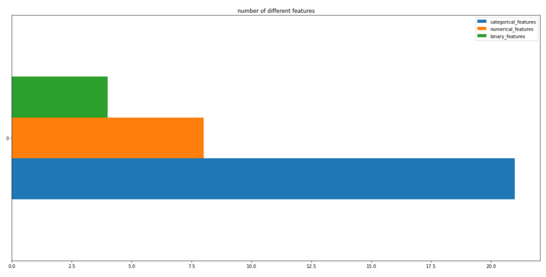
```
unique_values1
```

	Features	unique_values	Largest category percentage
0	MachineIdentifier	5744784	0.000017
3	AvSigVersion	7061	1.256166
5	OsBuildLab	525	43.290313
11	Census_OSVersion	402	17.206095
2	AppVersion	102	63.107455
1	EngineVersion	60	45.471579
9	Census_ChassisTypeName	43	59.990524
13	Census_OSEdition	28	39.386999
12	Census_OSBranch	23	47.237424
14	Census_OSSkuName	23	39.386529
7	Census_MDC2FormFactor	11	64.937150
10	Census_PowerPlatformRoleName	9	70.315072
4	OsPlatformSubRelease	9	46.180083
15	Census_OSInstallTypeName	9	32.639904
6	SkuEdition	7	63.371451
16	Census_OSWUAutoUpdateOptionsName	5	45.250387
17	Census_ActivationChannel	5	51.798727
8	Census_PrimaryDiskTypeName	4	66.311475

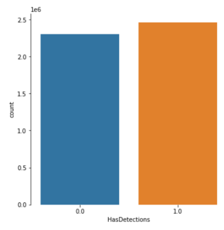
For each of the columns-AvSigVersion,ChassisTypeName,CensusOSEdition ,OsBuildLab,CensusSSkuName,CensusOSVersion,AppVersion, we made a new category and pushed all the categories occurring less than 0.1% out of the total length of records.
Encoded categorical data using the cat.codes method , this not only label encodes but also is a memory efficient technique to encode categorical columns as it uses hashing to encode.

When all rows were removed where a numerical column had an outlier(detected using z scores) ,the number of rows reduced significantly and any model did not perform better , hence we left this part as it could even lead to information loss.

EDA

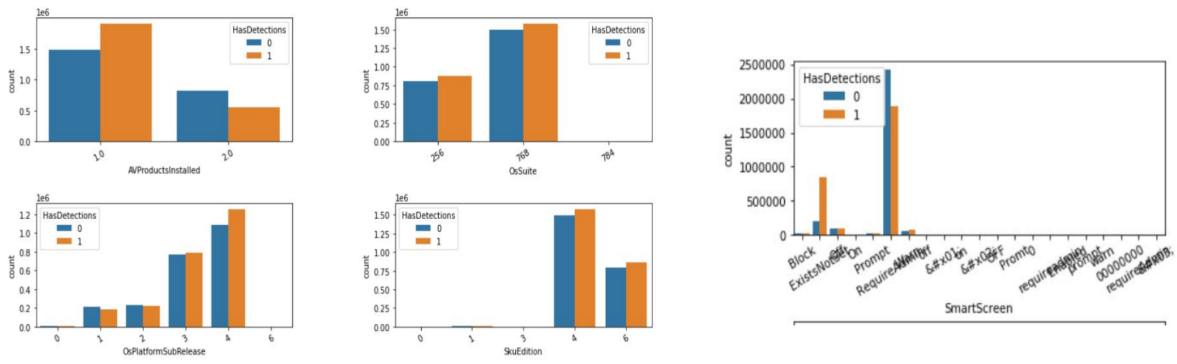


Most of the columns are categorical



Our target variable is mostly balanced

Countplots like below were made for every categorical column



As a conclusion about these categorical features, we may say the

target variable had similar distribution in most columns. But some columns like AVproductsInstalled , Smartscreen,censusOSVersion ,OS-buildlab,AvSigversion etc seemed relevant as their categories showed significant difference in the target variable distribution.Several inferences could be made just by looking at these plots as in which category influenced the target variable significantly.
For example in AvSigversion -Anti virus signature version of the windows defender , particular versions performed much better compared to others which performed much poorer , as in high number of machines which had a particular version had detected malware whereas for few other particular versions it was the opposite , the malware detected was very low . Therefore , these types of plots truly helped analyze each categorical column and its influence on the target variable . Wherever the distribution of 0s and 1s was equal on almost every category of the column it indicated that the feature did not have much influence on the target variable .

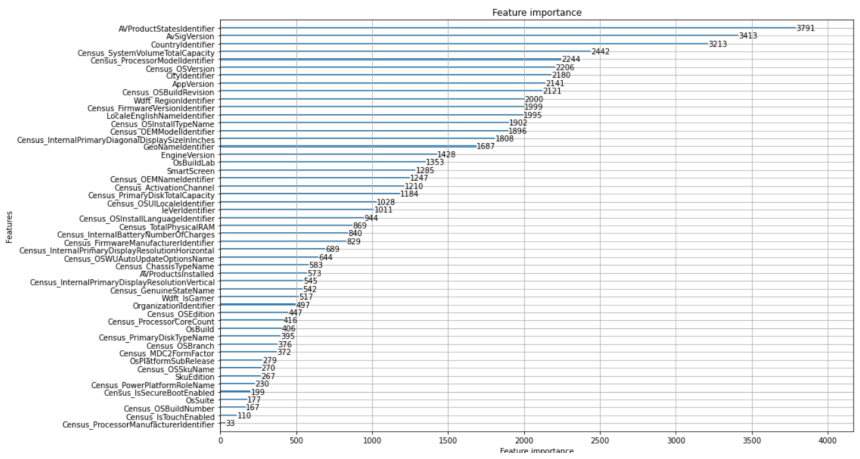
MODELS AND EXPERIMENTS

We had initially used dask to train our models but now due to memory optimizations our dataset occupied lesser space and hence were able to run models like logistic regression without any issues. Our models train within 10 mins , whereas dask took hours to run a logistic regression model as it divides the data and tasks amongst different clusters and passing data between the clusters for computations takes up lot of time.
When we used logistic regression on our training data, we faced memory issues hence we had sorted to dask , we were able to train the entire dataset after normalizing the data. Later after the memory optimizations we were able to train using sklearn itself , it relatively took much lesser time whereas dask took 3 hours to train .
We trained XGboost model on our train data with default parameters keeping n-estimators as 600 , we trained it by trying out both normally as well as with dask framework. Dask framework took more than an hour to train.Whereas it was much more convenient to train it using

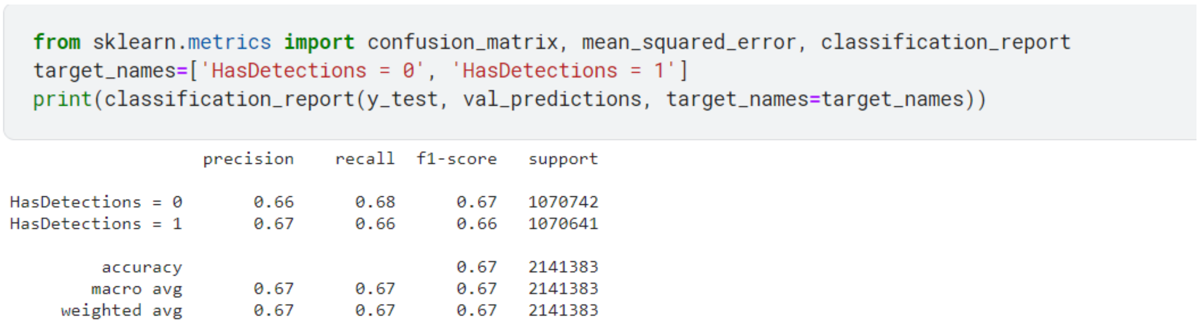
pandas and sklearn itself .
Lightgbm is a fast, distributed, high performance gradient boosting framework based on decision tree algorithm. It splits the tree leaf wise while others(like XGboost) split level wise or tree wise .

Since the tree grows leafwise , it can lead to overfitting and hence poorer score .We can control this by defining the depth of the splitting . We chose values of the parameters like num-leaves and n-estimators by hit and trial accordingly whichever values gave us the best score . We defined our learning rate to be 0.1 , num-leaves to be 60 , max-depth as -1 itself and estimators as 1000.

We plotted graph for feature importance for our best model -



RESULTS
The logistic regression model scored 0.57 on test data without the normalization while training the model on train-data , whereas it scored 0.61 after train-data was normalized .
XGboost model scored 0.69 on test data.
We mainly worked on Lightgbm model and tuned its parameters to get the best score of 0.71 on the train data.(train-test split)
Confusion matrix of the lgbm model tested on 0.15 % of the test data-



```python
from sklearn.metrics import confusion_matrix, mean_squared_error, classification_report
target_names=['HasDetections = 0', 'HasDetections = 1']
print(classification_report(y_test, val_predictions, target_names=target_names))
```

```
                   precision    recall  f1-score   support

HasDetections = 0       0.66      0.68      0.67   1070742
HasDetections = 1       0.67      0.66      0.66   1070641

         accuracy                           0.67   2141383
        macro avg       0.67      0.67      0.67   2141383
     weighted avg       0.67      0.67      0.67   2141383
```

THANK YOU

SUBMITTED BY -
AYUSH TIWARI (IMT2019015) AND REVANTH REDDY (IMT2019517)

-LONERBOIS