Assignment – 1, Report

COL-865

-by Harsh Pandey

Under the supervision of

Prof. Srikanta Bedathur

**Dataset1**: BitcoinHeist Ransomware Dataset

We are using this dataset for classification task. This has details of fraudulent transactions that occur in transferring of crypto currencies.

This prediction label has 29 classes, categorical + numerical features and no missing values.

Note that this is a highly skewed dataset with class distribution as-

**Few Initial details**

* The categorical features are encoded into one-hot encoding and the numerical values are normalized before giving as input to the model.
* For target labels they are also converted to one-hot encoding and then pass the training dataset to model for train.

I am always data from training using stratified sampling. (i.e take same percentage from each class so that no class is missed )

1. **Random Forest** –

I am using Sklearn random forest implementation for this.

**Training and Testing**

* I used **cross validation** to get best hyperparameters for-
  + n\_estimators: number of trees in Random Forest
  + max\_depth: max depth of each tree.
* I got very high accuracies of 98.79403 for

Number of estimator: 20

Max depth: 20

Train accuracy: 99.04613

Test accuracy: 98.79403

But this results although looks very promising is actually not very good. Because of highly skewed data the model learns very few important and usable details. For this let’s plot the confusion matrix and check what is happening.

**Visualization**

This is the confusing matrix on train dataset.

A screenshot of a computer

Description automatically generated with low confidence

This is the confusion matrix on the test dataset.

A picture containing text, light, electronics, dark

Description automatically generated

As we can see we mainly get ‘white’ i.e the most dominant class as the prediction. But still Random Forest can at least some values when compare to the later models.

1. **Oblique Trees-**

I am using **scikit-obliquetree 0.1.4** implementation for this.

I have also used ensembled oblique trees in expectation to get a better result.

**Training and Testing**

* For Hyperparameters I selected-
  + n\_estimators: 10, number of trees in Random Forest
  + max\_depth: 5, max depth of each tree.
* This took Wall time: 5min 19s, to run.
* Here as well we see high accuracies of *97.360* for

Number of estimator: 10

Max depth: 5

Train accuracy: 97.360

Test accuracy: 95.138261

Here max\_depth is chosen smaller than random forest because oblique tree uses a combination of attributes, so might overfit faster.

But the accuracies show it can’t properly even fit the dataset.

Note the accuracies are a bit less compared to random forest. Plotting the confusing matrix clearly show what is happening.

**Visualization**

This is for the testing dataset

Graphical user interface, text

Description automatically generated

This is interesting as from the confusion matrix it looks like this can actually learn some significant information.

1. **TabNet -**

I am using [**TabNet’s pytorch**](https://github.com/dreamquark-ai/tabnet)implementation for this.

**Training and Testing**

* Here I used Adam and trained TabNet until there is no significant decrease in loss values.
* This was the accuracies train set: 98.5804
* And for Test we got an accuracy: 98.5782

**Epoch wise Result**

Device used : cuda

epoch 0 | loss: 0.00797 | val\_0\_logloss: 0.00475 | 0:02:53s

epoch 1 | loss: 0.00474 | val\_0\_logloss: 0.00469 | 0:05:46s

epoch 2 | loss: 0.00465 | val\_0\_logloss: 0.00472 | 0:08:39s

epoch 3 | loss: 0.00465 | val\_0\_logloss: 0.00457 | 0:11:31s

epoch 4 | loss: 0.00453 | val\_0\_logloss: 0.00457 | 0:14:22s

epoch 5 | loss: 0.00452 | val\_0\_logloss: 0.00456 | 0:17:13s

epoch 6 | loss: 0.0045 | val\_0\_logloss: 0.00466 | 0:20:05s

epoch 7 | loss: 0.00459 | val\_0\_logloss: 0.0046 | 0:22:56s

epoch 8 | loss: 0.00458 | val\_0\_logloss: 0.00469 | 0:25:45s

epoch 9 | loss: 0.00456 | val\_0\_logloss: 0.00461 | 0:28:38s

best\_epoch = 5 and best\_val\_0\_logloss = 0.00456

Wall time: 31min 39s

**Visualization**

This is for the training dataset, as visible this is clearly not learning anything because of the highly scaled dataset.

A picture containing text

Description automatically generated

This is for the testing dataset, here as well we only get prediction as ‘white’ (the majority class)

A picture containing text

Description automatically generated

This is the worst of all, as TabNet is specifically predicting just 1 class.

**Dataset2**: Traffic Violations Dataset

Again we will be using this dataset for classification task. This has details traffic violations that occurred from all electronic traffic violations issued. (of some country)

**Few Initial details**

This prediction label has 4 classes,42 categorical + numerical features and lots of missing values.

Especially for this dataset, each column is checked whether it can be kept as categorical or should be changed into numerical.

* I found 2 columns-
  + date\_of\_stop: date when the vehicle was stopped for voilation
  + time\_of\_stop: time when the vehicle was stopped for violation
* These columns were converted to date, time and from there to integer values in seconds.

There is also some heavy filtering done like

* Categorical columns with unique values more than 100 are dropped.
* Categorical columns with more than 1000 None values are dropped.
* For reset of Categorical columns with None values they are assigned a new class ***None\_class,*** .
  + Eg- columns with values [‘yes’, ‘no’, None] (2 unique values and some None values) now have 3 unique values [‘yes’, ‘no’, ‘None\_class’]
* Now I encode all categorical features in one-hot encoding.

List of removed columns-

['agency', 'description', 'model', 'geolocation', 'location', 'driver\_city', 'search\_conducted', 'search\_disposition', 'search\_outcome', 'search\_reason', 'search\_reason\_for\_stop', 'search\_type', 'search\_arrest\_reason', 'make', 'model', 'color', 'article']

1. **Random Forest** –

I am using Sklearn random forest implementation for this.

**Training and Testing**

1. I used **cross validation** to get best hyperparameters for-
   * n\_estimators: number of trees in Random Forest
   * max\_depth: max depth of each tree.
2. I got very high accuracies of 69.3868175 for

Number of estimator: 60

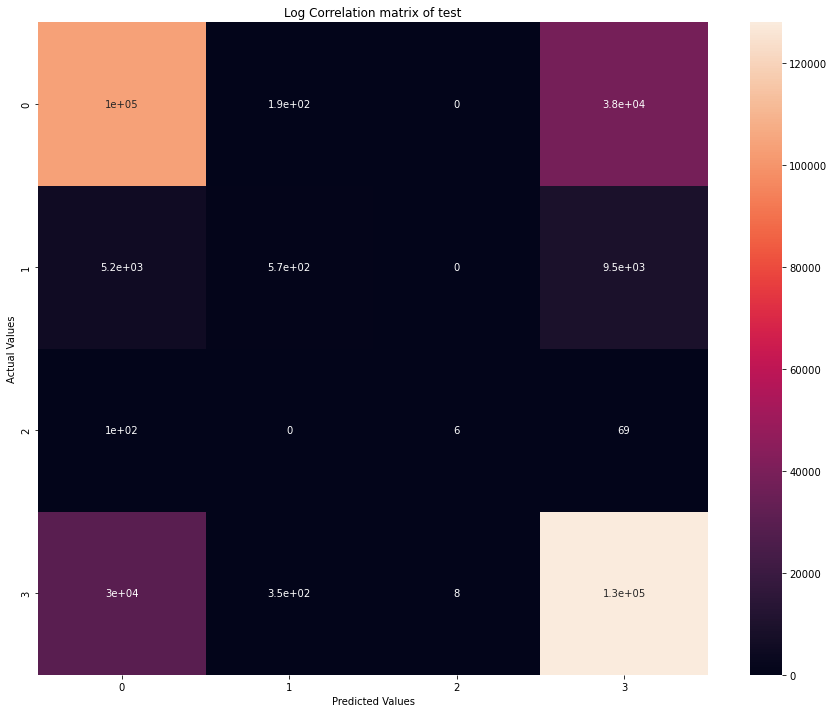
Max depth: 40

Test accuracy: **69.3868175**

This is a fine accuracy as compared to the later models.

**Visualization**

This is the confusion matrix on the test dataset.



As visible the model does learns to distinguish 2 classes

1. **Oblique Trees-**

I am using **scikit-obliquetree 0.1.4** implementation for this.

**This section’s code takes very long time because of the high number of feature/ columns.**

**Training and Testing**

1. I used Hyper parameters as

Number of estimator: 50

Max depth: 20

1. It took Wall time: **2h 18min 4s.** On Asterix when running at almost all cores.
2. The test accuracy: 46.35366502762711
3. And just 1 forward pass over test data took Wall time: 20min 48s

This shows that oblique tree is not running well

**Visualization**

This is for the testing dataset

Background pattern

Description automatically generated with low confidence

This does not look good as well. Maybe the trees are facing difficulty in finding best attribute across all the features.

**Dataset3**: German Credit Card Dataset

I used **the numerical version of the dataset** where they already quoting “This file has been edited and several indicator variables added to make it suitable for algorithms which cannot cope with categorical variables. Several attributes that are ordered categorical (such as attribute 17) have been coded as integer”.

This is a binary classification task. And uses a custom cost matrix

1. **Random Forest** –

I am using Sklearn random forest implementation for this.

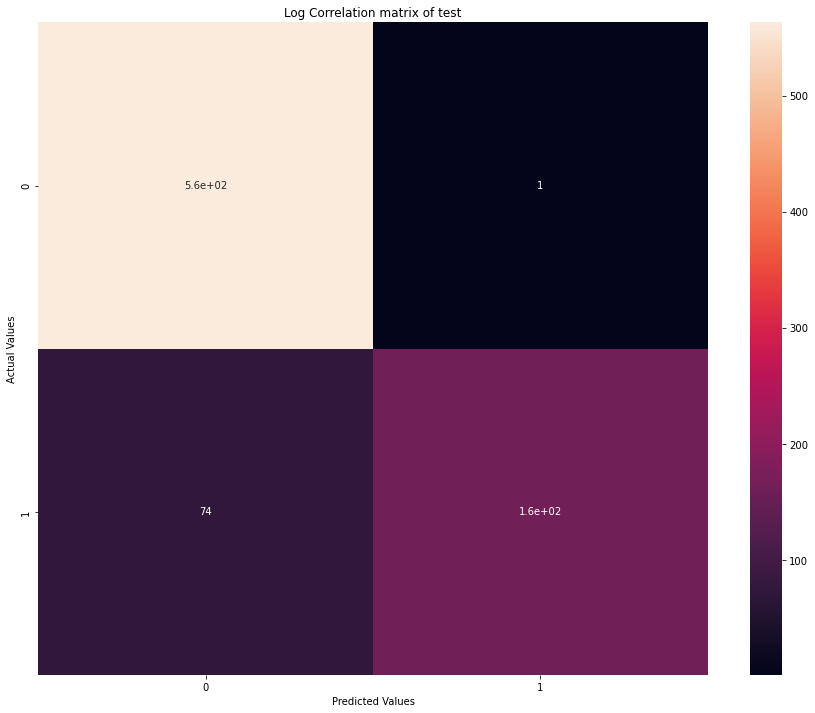
**Training and Testing**

I got very high accuracies of 90.625 for

Number of estimator: 10

Max depth: 10

Test accuracy: **90.625**



1. TabNet

First the data is fetched and normalized.

Then I directly use the TabNet model for this task.

For cost function I used the class weight parameter to assign different class weights when one class is predicted.

Accuracy : 0.425