Microsoft Malware Prediction with XGBoost Decision Tree Algorithm

April 3, 2020

[1]:
```
import numpy as np
import pandas as pd
import json, datetime as dt
import matplotlib.pyplot as plt
```

[2]:
```
from sklearn.model_selection import train_test_split as tts
from xgboost.sklearn import XGBClassifier
```

[3]:
```
d1 = dt.datetime.now()
print("Data processing started at", "%02d:%02d" % (d1.hour, d1.minute))
```

```
Data processing started at 11:43
```

1 Data loading & exploring

1.1 Loading

Opening dataset

[4]:
```
with open('./data/datatypes.json') as file:
    dtype = json.load(file)
```

[5]:
```
df = pd.read_csv('./data/microsoft-malware.csv', dtype=dtype)
```

[6]:
```
df.shape
```

[6]:
```
(999999, 83)
```

1.2 Cleaning

Dropping categorical data

[7]:
```
binary = []
categorical = []
numerical = []
```

```
for key, value in dtype.items():
    if value in ['int8']:
        binary.append(key)
    if value in ['int16','category']:
        categorical.append(key)
    else:
        numerical.append(key)

categorical.remove('MachineIdentifier') # Déjà enlevé par iloc
df = df.drop(columns=list(categorical))
```

Cleaning NaN

[8]:
```
for i in df.columns:
    s = df.loc[:, i]
    if i in numerical:  # set NaNs in numerical features to -1
        s.fillna(-1, inplace=True)
    elif i in binary:  # set NaNs in binary feature to the most frequent one
        s.fillna(s.mode().iloc[0], inplace=True)
    df[i] = s.values
    if df[i].dtype == "int64" or df[i].dtype == "float64":
        df.loc[df[i].value_counts(normalize=True)[df[i]].values < 0.05, i] = -1
```

2 Splitting dataset into train/test sets

Splitting into `data` (for training) and `target` (for testing), then splitting again into `x` and `y` values

[9]:
```
data = df.iloc[:,1:-1] # Dropping MachineIdentifier & HasDetections
target = df.iloc[:,-1] # Selecting HasDetections
```

[10]:
```
xtrain, xtest, ytrain, ytest = tts(data, target, train_size=0.8)
```

3 Training classifier

Fitting one million lines would take about 2 minutes on my work computer (Core i5, 16GB RAM)
Total number of lines (approx. 9 million) should take ~20min (hopefully).
Edit : Actually took 47min. Lol.

[11]:
```
boost = XGBClassifier(max_depth=4, n_estimators=500)
```

[12]:
```
boost.fit(xtrain, ytrain)
```

[12]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=4,
              min_child_weight=1, missing=None, n_estimators=500, n_jobs=1,
```

```
nthread=None, objective='binary:logistic', random_state=0,
reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
silent=None, subsample=1, verbosity=1)
```

[13]:
```
boost_prediction = boost.predict(xtest)
```

[14]:
```
print("Score Train :", round(boost.score(xtest, ytest)*100, 2), " %")
```

```
Score Train : 59.78  %
```

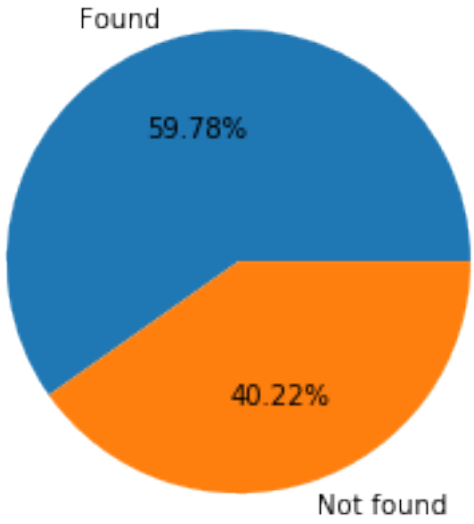
[15]:
```
d2 = dt.datetime.now()
print("Took ", d2-d1)
```

```
Took  0:00:29.139907
```

Plotting result

[16]:
```
from scipy.spatial import distance
```

[17]:
```
labels = 'Found', 'Not found'
hamming = distance.hamming(ytest, boost_prediction)
rates = [1-hamming, hamming]
fig1, ax1 = plt.subplots()
ax1.pie(rates, labels=labels, autopct='%0.2f%%')
plt.show()
```

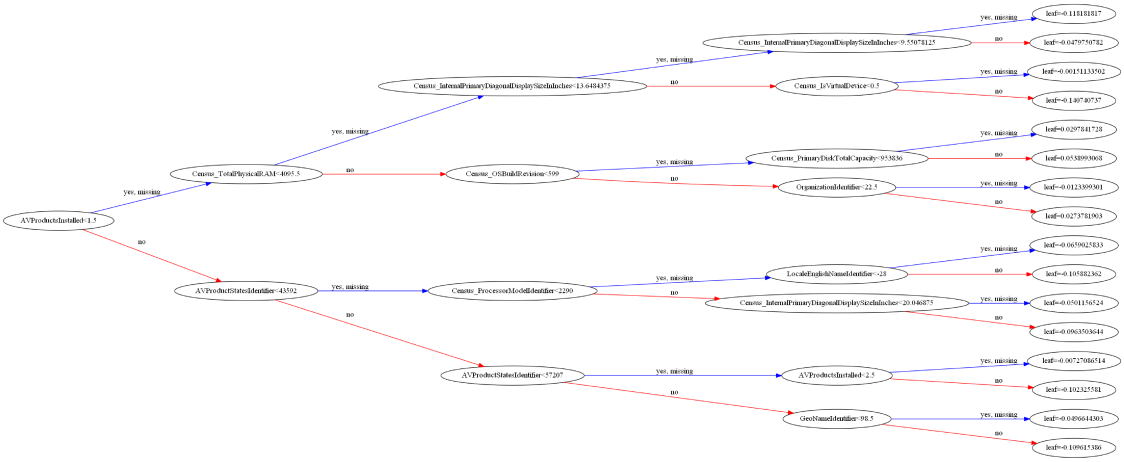


4 Decision tree

We can view the decision tree made by the classifier by using `plot_tree` method provided in `xgboost`

[18]:
```
from xgboost import plot_tree
```

[19]:
```
plot_tree(boost, rankdir='LR')
fig = plt.gcf()
fig.set_size_inches(150,50)
# fig.savefig("tree.png")
plt.show()
```



5 Performance study by varying parameters

5.1 Varying `n_estimators`

Testing 10 classifiers with estimator values from 50 to 500.
Memory error when tested on all dataset → must train models on a a subset.

[20]:
```
est_values = [50*i for i in range(1,11)]
xgb_estimators = [XGBClassifier(n_estimators=i) for i in est_values]
```

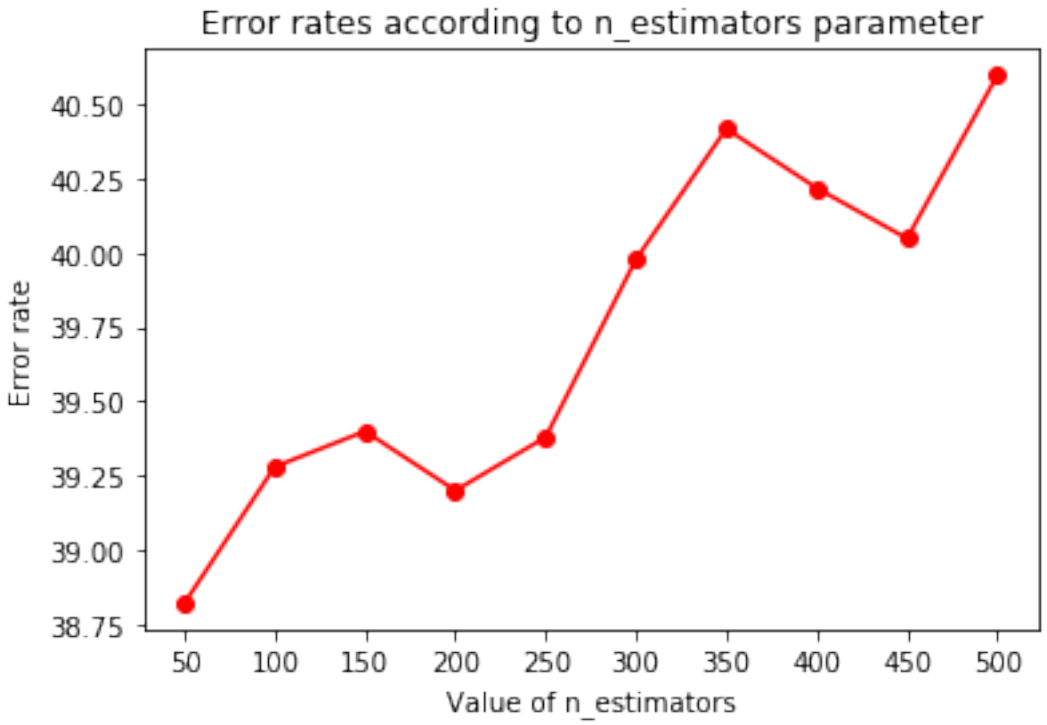
Calculate error rate of each classifier

[21]:
```
xgb_estimators_results = []
for i in range(0,10):
    xgb_estimators[i].fit(xtrain, ytrain)
    xgb_estimators[i].predict(xtest)
    xgb_estimators_results.append(
        round(100 - xgb_estimators[i].score(xtest, ytest)*100, 2)
    )
```

```
min_estimators = xgb_estimators_results.index(min(xgb_estimators_results))
```

Plotting results

[22]:
```
plt.plot(est_values, xgb_estimators_results, 'ro-')
plt.xlabel('Value of n_estimators')
plt.ylabel('Error rate')
plt.title('Error rates according to n_estimators parameter')
plt.xticks(est_values)
plt.show()
```



As we can see, we can get a minimum error rate by setting `n_estimators` to 50.
May change when shuffling dataset.

5.2 Varying `max_depth`

Testing 7 classifiers with `max_depth` values from 1 to 8

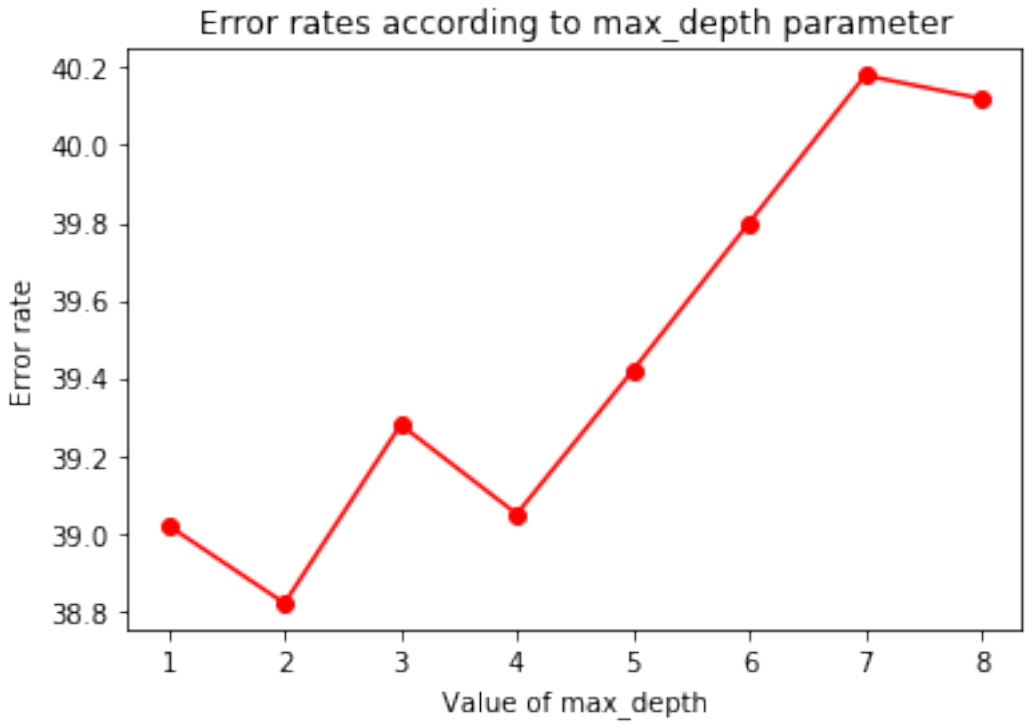
[23]:
```
depth_values = range(1,9)
xgb_depth = [XGBClassifier(max_depth=i) for i in depth_values]
```

Calculate error rate of each classifier

[24]:
```
xgb_depth_results = []
for i in depth_values:
    xgb_depth[i-1].fit(xtrain, ytrain)
    xgb_depth[i-1].predict(xtest)
    xgb_depth_results.append(
        round(100 - xgb_depth[i-1].score(xtest, ytest)*100, 2)
    )
min_depth = xgb_depth_results.index(min(xgb_depth_results))
```

Plotting results

[25]:
```
plt.plot(depth_values, xgb_depth_results, 'ro-')
plt.xlabel('Value of max_depth')
plt.ylabel('Error rate')
plt.title('Error rates according to max_depth parameter')
plt.xticks(depth_values)
plt.show()
```



A minimum error rate can be get by setting `max_depth` to 2.
May change when shuffling dataset.

5.3 Using the study results

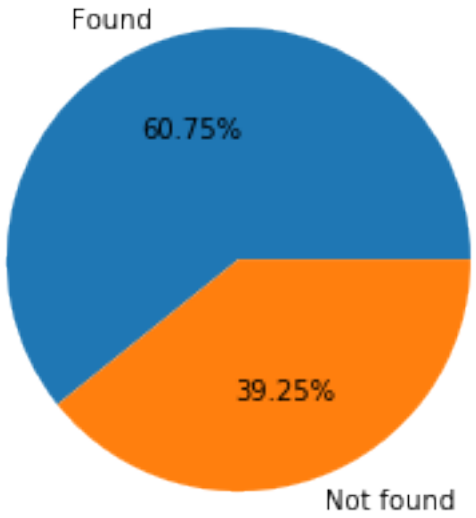
[26]:
```
min_err_est = est_values[min_estimators]
min_err_dep = depth_values[min_depth]
print('Training with parameters:\nn_estimators\t', min_err_est,␣
↪'\nmax_depth\t',min_err_dep)
```

```
Training with parameters:
n_estimators     50
max_depth        2
```

[27]:
```
boost = XGBClassifier(n_estimators=min_err_est, max_depth=min_err_dep,␣
↪tree_method='exact', predictor='cpu_predictor')
boost.fit(xtrain, ytrain)
boost_prediction = boost.predict(xtest)
```

Result

[28]:
```
labels = 'Found', 'Not found'
hamming = distance.hamming(ytest, boost_prediction)
rates = [1-hamming, hamming]
fig1, ax1 = plt.subplots()
ax1.pie(rates, labels=labels, autopct='%0.2f%%')
plt.show()
```



Decision tree

[29]:
```
plot_tree(boost, rankdir='LR')
fig = plt.gcf()
fig.set_size_inches(150,50)
# fig.savefig("tree.png")
plt.show()
```