6-MicrosoftMalwarePrediction-GradientBoosting

May 22, 2020

1 Microsoft Malware Prediction

1.1 Gradient Boosting y sus variantes

1.1.1 Importamos las librerías

[13]:

```
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
import json
import plotly.express as px
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import metrics
from time import time
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import metrics
from sklearn.metrics import f1_score, precision_score, recall_score
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
```

Lectura de los datos

[ ]:

```
# Leemos los datos originales (para el submission necesitamos la variable␣
 ↪identificadora de test)

# Leemos el fichero json
import json

with open('datos/datatype.json', 'r') as myfile:
    data = myfile.read()
```

```
# Obtenemos los tipos de datos para el train
dtypes_train = json.loads(data) # Parse file

# Hacemos una copia de los tipos de datos a modificar para test
dtypes_test = dtypes_train.copy()

# Eliminamos la variable 'target'
del dtypes_test['HasDetections']

# Lectura de nuevo del conjunto de train y test, con los tipos de datos que hemos definido
train = pd.read_csv("./datos/train_malware.csv", dtype = dtypes_train)
test = pd.read_csv("./datos/test_malware.csv", dtype = dtypes_test)
```

[ ]:

```
# Leemos los datos con label encoding
train_label_encoding = pd.read_csv("./datos/train_filtrado_encoding.csv")
test_label_encoding = pd.read_csv("./datos/test_filtrado_encoding.csv")
```

Partición

[3]:

```
# Dividimos la variable target de
x = train_label_encoding.drop('HasDetections', axis=1)
y = train_label_encoding['HasDetections']
```

[4]:

```
# Creamos el conjunto de validación
X_train, X_val, y_train, y_val = train_test_split(x, y, test_size=0.25, random_state = 3)
print(X_train.shape, y_train.shape, X_val.shape, y_val.shape)
```

```
(6580545, 58) (6580545,) (2193515, 58) (2193515,)
```

1.1.2 Lectura del conjunto de datos particionados

[3]:

```
# Lectura del conjunto de datos particionado
X_train = pd.read_csv("./datos/X_train.csv")
X_val = pd.read_csv("./datos/X_val.csv")
y_train = pd.read_csv("./datos/y_train.csv")
y_val = pd.read_csv("./datos/y_val.csv")
```

1.1.3 Algoritmo de `GradientBoostingClassifier`

Partición 75-25

```
gb_model = GradientBoostingClassifier(n_estimators=100, learning_rate=1,
                                      max_depth=3, validation_fraction=0.15, random_state=9)
```

	learning_rate	n_estimators	tiempo (seg.)	tiempo	accuracy (training)	accuracy (validation)
1	0.05	100	6248.7365028858	1.73 horas	0.628	0.628
2	0.1	100	5736.4154019356	1.60 horas	0.634	0.634
3	0.5	100	5800.8757534027	1.61 horas	0.645	0.645

```
gb_model = GradientBoostingClassifier(n_estimators=50, learning_rate=1,
                                      max_depth=4, validation_fraction=0.2, random_state=9)
```

	learning_rate	n_estimators	tiempo (seg.)	tiempo	accuracy (training)	accuracy (validation)
1	0.05	50	3653.2919328213	1 hora	0.626	0.627
2	0.1	50	2985.6868040562	49.76 minutos	0.634	0.634
3	0.5	50	3337.0875909328	55.61 minutos	0.646	0.646
4	0.75	50	3804.4561002265	64 minutos	0.648	0.648
5	1	50	3502.4089672565	59 minutos	0.647	0.647

[6]:
```
# Configuración del modelo de Gradient Boosting
gb_model = GradientBoostingClassifier(n_estimators=50, learning_rate=0.75,
                                      max_depth=4, validation_fraction=0.2,␣
 ↪random_state=9)
```

[7]:
```
# Entrenamiento del modelo
start_time = time()
gb_model.fit(X_train, y_train)
time_gb_model = time() - start_time

y_pred = gb_model.predict(X_val)

print("Tiempo de entrenamiento: %.10f segundos" % time_gb_model)
print("Accuracy: ", metrics.accuracy_score(y_val, y_pred))
```

```
/Users/gema/anaconda3/lib/python3.7/site-
packages/sklearn/utils/validation.py:761: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the
shape of y to (n_samples, ), for example using ravel().


Tiempo de entrenamiento: 3280.8205292225 segundos
Accuracy:  0.6475674887110414
```

[11]:

```
# Imprimimos algunas métricas
logloss = metrics.log_loss(y_val, y_pred)
accuracy = metrics.accuracy_score(y_val, y_pred)
F1 = metrics.f1_score(y_val, y_pred)
precision = precision_score(y_val, y_pred, average='binary')
recall = recall_score(y_val, y_pred, average='binary')
auc = metrics.roc_auc_score(y_val, y_pred)

metricas = [logloss, accuracy, F1, precision, recall, auc, time_gb_model]
nombre_metricas = ['Log loss', 'Accuracy', 'F1 Score', 'Precision', 'Recall', 'AUC', 'Tiempo de entrenamiento']

pd.DataFrame(metricas, nombre_metricas, columns = ['Gradient Boosting']).T
```

[11]:

```
                    Log loss  Accuracy  F1 Score  Precision    Recall  \
Gradient Boosting  12.172721  0.647567  0.640917   0.653346  0.628952

                       AUC  Tiempo de entrenamiento
Gradient Boosting  0.64757              3280.820529
```

[14]:

```
# Guardar el modelo
pkl_filename = "modelos/gradient_boosting.pkl"
with open(pkl_filename, 'wb') as file:
    pickle.dump(gb_model, file)
```

1.1.4 Sacamos las variables más importantes del mejor modelo

[15]:

```
feature_importance = pd.DataFrame(sorted(zip(gb_model.feature_importances_,X_train.columns)),
                                  columns=['Valor','Variable'])
```

[16]:

```
feature_importance = feature_importance.sort_values('Valor', ascending=False)
feature_importance.head(10)
```

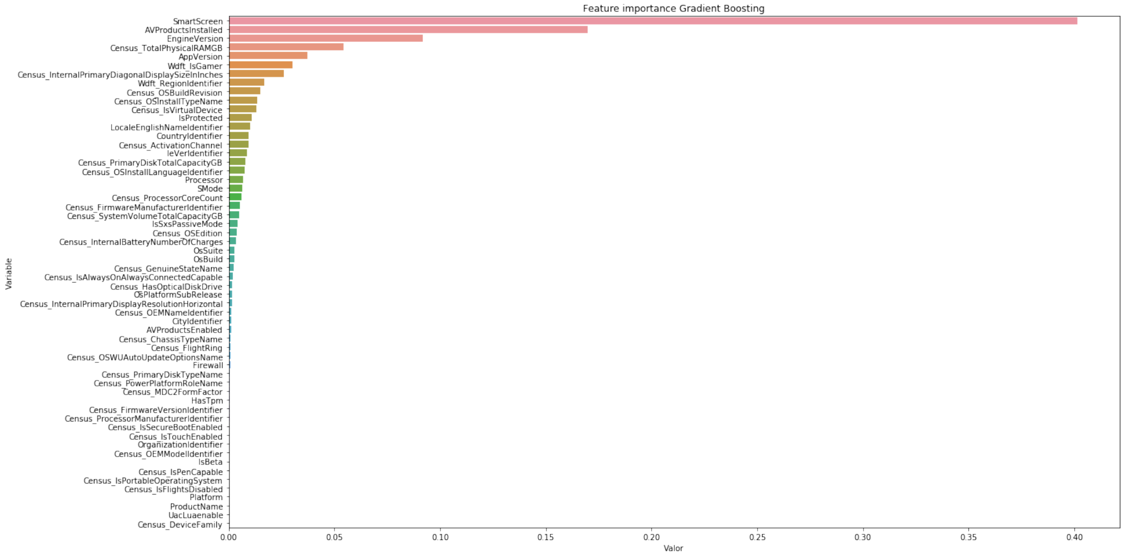
[16]:

```
       Valor                                           Variable
57  0.401244                                        SmartScreen
56  0.169904                                 AVProductsInstalled
55  0.091885                                       EngineVersion
54  0.054166                             Census_TotalPhysicalRAMGB
53  0.037232                                          AppVersion
52  0.030377                                        Wdft_IsGamer
51  0.026119  Census_InternalPrimaryDiagonalDisplaySizeInInches
50  0.016714                                 Wdft_RegionIdentifier
49  0.014922                               Census_OSBuildRevision
48  0.013348                              Census_OSInstallTypeName
```

[17]:

```
fig = px.bar(feature_importance, x='Valor', y='Variable', orientation='h')
fig.update_layout(title_text='Feature importance Gradient Boosting', title_x=0, xaxis=dict(title='Valor'),
                 margin=dict(l=10, r=10, t=100, b=0), template='seaborn',
                  uniformtext_minsize=6,)
fig.show()
```

[18]:
```
plt.figure(figsize=(20, 10))
sns.barplot(x="Valor", y="Variable",
            data=feature_importance.sort_values(by="Valor", ascending=False))
plt.title('Feature importance Gradient Boosting')
plt.tight_layout()
plt.show()
```



1.1.5 Algoritmo de XGBoost

https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/

[13]:
```
# Configuración del algoritmo XGBoost
xgb_model = XGBClassifier()
```

[14]:
```
# Entrenamiento del modelo
start_time = time()
xgb_model.fit(X_train, y_train)
elapsed_time = time() - start_time

y_pred = xgb_model.predict(X_val)

print("Tiempo de entrenamiento: %.10f segundos" % elapsed_time)
print("Accuracy score (training): {0:.3f}".format(xgb_model.score(X_train,␣
 ↪y_train)))
```

```
print("Accuracy score (validation): {0:.3f}".format(xgb_model.score(X_val,␣
 ↪y_val)))
print("Accuracy: ", metrics.accuracy_score(y_val, y_pred))
```

```
Tiempo de entrenamiento: 1679.1951990128 segundos
Accuracy score (training): 0.634
Accuracy score (validation): 0.634
Accuracy:  0.6337941614258393
```

1.1.6 Vamos a sacar las variables más importantes

[16]:

```
feature_importance = pd.DataFrame(sorted(zip(xgb_model.
 ↪feature_importances_,X_train.columns)),
                                  columns=['Valor','Variable'])
```

[17]:

```
feature_importance = feature_importance.sort_values('Valor', ascending=False)
feature_importance.head()
```

[17]:

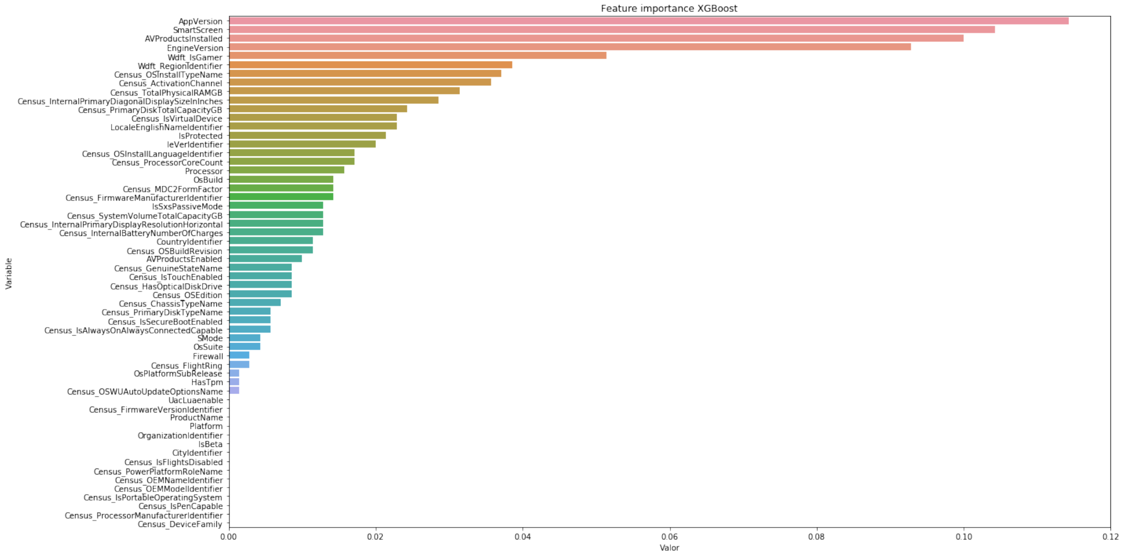
```
       Valor             Variable
57  0.114286           AppVersion
56  0.104286          SmartScreen
55  0.100000  AVProductsInstalled
54  0.092857        EngineVersion
53  0.051429          Wdft_IsGamer
```

[20]:

```
fig = px.bar(feature_importance, x='Valor', y='Variable', orientation='h')
fig.update_layout(title_text='Feature importance XGBoost', title_x=0,␣
 ↪xaxis=dict(title='Valor'),
                 margin=dict(l=10, r=10, t=100, b=0), template='seaborn',
                  uniformtext_minsize=6,)
fig.show()
```

[21]:

```
plt.figure(figsize=(20, 10))
sns.barplot(x="Valor", y="Variable",
            data=feature_importance.sort_values(by="Valor", ascending=False))
plt.title('Feature importance XGBoost')
plt.tight_layout()
plt.show()
```



Submission en Kaggle

```
[22]: pred_xgb_model = xgb_model.predict(test_label_encoding)
      (pred_xgb_model, len(y_pred))
```

```
[22]: (array([1, 1, 1, ..., 1, 0, 0]), 2193515)
```

```
[23]: # Cogemos los identificadores del conjunto test
      id_test = test['MachineIdentifier']

      # Leemos el CSV para realizar el submission
      submission = pd.read_csv("./datos/Submissions/GradientBoosting/XGBoost/
       ↪sample_submission.csv")
      # Vemos que 'submission.head()' coincide con 'id_test' de manera ordenada

      # Pegamos la lista de los identificadores a la columna␣
       ↪submission['HasDetections']
      submission['HasDetections'] = pred_xgb_model
      submission.head()
```

```
[23]:                 MachineIdentifier  HasDetections
      0  0000010489e3af074adeac69c53e555e              1
      1  00000176ac758d54827acd545b6315a5              1
      2  0000019dcefc128c2d4387c1273dae1d              1
      3  0000055553dc51b1295785415f1a224d              1
      4  00000574cefffeca83ec8adf9285b2bf              1
```

```
[24]: # Guardamos el fichero CSV
      submission.to_csv('./datos/Submissions/GradientBoosting/XGBoost/
       ↪sample_submission.csv',
                        index = False, header = True)
```