Demo

In our project, we used the dataset that contains 500,000 individual machines information.

However, in the demo, we created the dataset (in the text file "projectdemo1.txt")that consists of 49,999 individual machines, which is approximately 25 MB.

Libraries we use in the demo:

1. pandas
2. numpy
3. sklearn
4. matplotlib

1. Loading the dataset and Preprocessing

As we cited in the report, in preprocessing, we used an external code from “Load the Totality of the Data.” Kaggle, © 2019 Kaggle Inc., www.kaggle.com/theoviel/load-the-totality-of-the-data.

In [1]:

```
import pandas as pd
# Code to read csv file into Colaboratory:
# !pip install -U -q PyDrive
# from pydrive.auth import GoogleAuth
# from pydrive.drive import GoogleDrive
# from google.colab import auth
# from oauth2client.client import GoogleCredentials
# # Authenticate and create the PyDrive client.
# auth.authenticate_user()
# gauth = GoogleAuth()
# gauth.credentials = GoogleCredentials.get_application_default()
# drive = GoogleDrive(gauth)

# link = 'https://drive.google.com/open?id=1kR3TcMccX8m3aScfno4wjY15vkqH1s_a'
# fluff, id = link.split('=')
# print (id) # Verify that you have everything after '='
# downloaded = drive.CreateFile({'id':id})
# downloaded.GetContentFile('test_file1.txt')

#https://www.kaggle.com/theoviel/load-the-totality-of-the-data
dtypes = {
        'MachineIdentifier':                                    'category',
        'ProductName':                                          'category',
        'EngineVersion':                                        'category',
        'AppVersion':                                           'category',
        'AvSigVersion':                                         'category',
        'IsBeta':                                               'int8',
        'RtpStateBitfield':                                     'float16',
        'IsSxsPassiveMode':                                     'int8',
        'DefaultBrowsersIdentifier':                            'float32',
        'AVProductStatesIdentifier':                            'float32',
        'AVProductsInstalled':                                  'float16',
        'AVProductsEnabled':                                    'float16',
        'HasTpm':                                               'int8',
        'CountryIdentifier':                                    'int16',
        'CityIdentifier':                                       'float32',
        'OrganizationIdentifier':                               'float16',
        'GeoNameIdentifier':                                    'float16',
        'LocaleEnglishNameIdentifier':                          'int16',
        'Platform':                                             'category',
        'Processor':                                            'category',
        'OsVer':                                                'category',
        'OsBuild':                                              'int16',
        'OsSuite':                                              'int16',
        'OsPlatformSubRelease':                                 'category',
        'OsBuildLab':                                           'category',
        'SkuEdition':                                           'category',
        'IsProtected':                                          'float16',
```

```
        'AutoSampleOptIn':                                  'int8',
        'PuaMode':                                          'category',
        'SMode':                                            'float16',
        'IeVerIdentifier':                                  'float16',
        'SmartScreen':                                      'category',
        'Firewall':                                         'float16',
        'UacLuaenable':                                     'float64', # wa
s 'float32'
        'Census_MDC2FormFactor':                            'category',
        'Census_DeviceFamily':                              'category',
        'Census_OEMNameIdentifier':                         'float32', # wa
s 'float16'
        'Census_OEMModelIdentifier':                        'float32',
        'Census_ProcessorCoreCount':                        'float16',
        'Census_ProcessorManufacturerIdentifier':           'float16',
        'Census_ProcessorModelIdentifier':                  'float32', # wa
s 'float16'
        'Census_ProcessorClass':                            'category',
        'Census_PrimaryDiskTotalCapacity':                  'float64', # wa
s 'float32'
        'Census_PrimaryDiskTypeName':                       'category',
        'Census_SystemVolumeTotalCapacity':                 'float64', # wa
s 'float32'
        'Census_HasOpticalDiskDrive':                       'int8',
        'Census_TotalPhysicalRAM':                          'float32',
        'Census_ChassisTypeName':                           'category',
        'Census_InternalPrimaryDiagonalDisplaySizeInInches':    'float32', # wa
s 'float16'
        'Census_InternalPrimaryDisplayResolutionHorizontal':    'float32', # wa
s 'float16'
        'Census_InternalPrimaryDisplayResolutionVertical':      'float32', # wa
s 'float16'
        'Census_PowerPlatformRoleName':                     'category',
        'Census_InternalBatteryType':                       'category',
        'Census_InternalBatteryNumberOfCharges':            'float64', # wa
s 'float32'
        'Census_OSVersion':                                 'category',
        'Census_OSArchitecture':                            'category',
        'Census_OSBranch':                                  'category',
        'Census_OSBuildNumber':                             'int16',
        'Census_OSBuildRevision':                           'int32',
        'Census_OSEdition':                                 'category',
        'Census_OSSkuName':                                 'category',
        'Census_OSInstallTypeName':                         'category',
        'Census_OSInstallLanguageIdentifier':               'float16',
        'Census_OSUILocaleIdentifier':                      'int16',
        'Census_OSWUAutoUpdateOptionsName':                 'category',
        'Census_IsPortableOperatingSystem':                 'int8',
        'Census_GenuineStateName':                          'category',
        'Census_ActivationChannel':                         'category',
```

```
        'Census_IsFlightingInternal':                               'float16',
        'Census_IsFlightsDisabled':                                 'float16',
        'Census_FlightRing':                                        'category',
        'Census_ThresholdOptIn':                                    'float16',
         'Census_FirmwareManufacturerIdentifier':                    'float16',
        'Census_FirmwareVersionIdentifier':                         'float32',
        'Census_IsSecureBootEnabled':                               'int8',
        'Census_IsWIMBootEnabled':                                  'float16',
        'Census_IsVirtualDevice':                                   'float16',
        'Census_IsTouchEnabled':                                    'int8',
        'Census_IsPenCapable':                                      'int8',
        'Census_IsAlwaysOnAlwaysConnectedCapable':                  'float16',
        'Wdft_IsGamer':                                             'float16',
        'Wdft_RegionIdentifier':                                    'float16',
        'HasDetections':                                            'int8'
        }

def load(x):
    if x in ['MachineIdentifier','IsBeta','Census_IsPortableOperatingSystem','P
uaMode','Census_IsVirtualDevice',
 'Census_ProcessorClass','Census_IsFlightsDisabled','Census_IsWIMBootEnabled',
 'Census_DeviceFamily','Census_IsFlightingInternal','UacLuaenable',
 'AutoSampleOptIn','SMode','Census_ThresholdOptIn','Platform',
'Census_OSInstallLanguageIdentifier','Census_OSArchitecture',
'Census_OSSkuName']:
        return False
    else:
        return True
train = pd.read_csv('projectdemo1.txt', delimiter=',', dtype=dtypes, usecols=lo
ad)
```

In [2]:

```
train.shape
```

Out[2]:

```
(49999, 65)
```

Replacing Nan values

In [3]:

```
train.DefaultBrowsersIdentifier.fillna(0,inplace=True)

import numpy as np
SmartScreen_dict = {
    'off': 'Off', '&#x02;': '2', '&#x01;': '1', 'on': 'On', 'requireadmin': 'Re
quireAdmin', 'OFF': 'Off',
    'Promt': 'Prompt', 'requireAdmin': 'RequireAdmin', 'prompt': 'Prompt', 'war
n': 'Warn',
    '00000000': '0', '&#x03;': '3', np.nan: 'NoExist'
}
train.replace({'SmartScreen': SmartScreen_dict}, inplace=True)

train.replace({'OrganizationIdentifier': {np.nan: 0.0}}, inplace=True)
census_bt_dict = {
    '□`````': 'unknown', 'unkn': 'unknown', np.nan: 'unknown'
}

train.replace({'Census_InternalBatteryType': census_bt_dict}, inplace=True)

train['SmartScreen'] = train.SmartScreen.astype('category')
train['Census_InternalBatteryType'] = train.Census_InternalBatteryType.astype(
'category')

category_cols = train.select_dtypes(include='category').columns.tolist()
```

In [4]:

```
train.dropna(inplace=True)
train.shape
```

Out[4]:

```
(43164, 65)
```

Replacing categorical values to numerical values

In [5]:

```
#Encode labels with value between 0 and n_classes-1.
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

for col in category_cols:
    train[col] = le.fit_transform(train[col])
```

2. Generating train and test datasets

In [6]:

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.ensemble import AdaBoostClassifier
import matplotlib.pyplot as plt
from matplotlib.legend_handler import HandlerLine2D
from sklearn.metrics import roc_curve, auc
from sklearn.ensemble import BaggingClassifier
```

In [7]:

```
#This function generates train and test datsets.
#As a default, we used an 80/20 train/test split on the dataset
def train_test_generator(train, train_size=0.8, random_state=100):
    features = train.loc[:, ~train.columns.isin(['HasDetections']) ]
    target = target = train.loc[:, train.columns == 'HasDetections']
    train_x, test_x, train_y, test_y = train_test_split(features, target, train
_size=0.8,shuffle=True,random_state=100)
    return train_x, test_x, train_y, test_y
```

In [8]:

```
train_x, test_x, train_y, test_y = train_test_generator(train)
```

```
C:\Users\yamam\Anaconda3\lib\site-packages\sklearn\model_selection
\_split.py:2179: FutureWarning: From version 0.21, test_size will
always complement train_size unless both are specified.
  FutureWarning)
```

4. Building Decision Tree, Bagging, Random Forest, AdaBoost models

In our demo, we used 1/10 of datasets that we used in our project. Thus, the accuracy rate we got in the demo is worse than the one in the project. Also, our hyperparameters change as our dataset changes (in the demo, we commented out the code for tuning hyperparameters for the speed of runtime execution).

a) Decision Tree

In [9]:

```
# from sklearn.model_selection import GridSearchCV

# param = {
#     'max_depth': np.linspace(1, 20, 10, endpoint=False, dtype=int),
#     'min_samples_split': np.linspace(0.1, 1.0, 5, endpoint=True)}

# clf_tree = DecisionTreeClassifier(criterion='entropy')
# # instantiate the grid
# grid = GridSearchCV(clf_tree, param, cv=5, scoring='accuracy')

# # fit the grid with data
# grid.fit(train_x, train_y)
# # summarize results
# print("Best: %f using %s" % (grid.best_score_, grid.best_params_))
### Best: 0.619513 using {'max_depth': 12, 'min_samples_split': 0.1} ###

clf = DecisionTreeClassifier(criterion = "entropy", max_depth=12, min_samples_s
plit=0.1,random_state = 100)
clf.fit(train_x, train_y)
pred_y = clf.predict(test_x)
clf_acur = accuracy_score(test_y, pred_y)
print("Accuracy (Decision Tree):", clf_acur)
```

```
Accuracy (Decision Tree): 0.625159272558786
```

b) Bagging

In [10]:

```
cart = clf = DecisionTreeClassifier(criterion = "entropy", max_depth=12, min_sa
mples_split=0.1,random_state = 100)
# param = {
#     'n_estimators': [1, 10, 25, 50, 100]
# }
# bag = BaggingClassifier(base_estimator=cart, random_state=100)

# # instantiate the grid
# grid = GridSearchCV(bag, param, cv=5, scoring='accuracy')

# # fit the grid with data
# grid.fit(train_x, train_y)
# summarize results
# print("Best: %f using %s" % (grid.best_score_, grid.best_params_))
# Best: 0.624607 using {'n_estimators': 100}
bag = BaggingClassifier(base_estimator=cart, n_estimators=100, random_state=100
)
bag.fit(train_x, train_y)
pred_y = bag.predict(test_x)
bag_acur = accuracy_score(test_y, pred_y)
print("Accuracy (Bagging):", bag_acur)
```

```
C:\Users\yamam\Anaconda3\lib\site-packages\sklearn\ensemble\baggin
g.py:621: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_sample
s, ), for example using ravel().
  y = column_or_1d(y, warn=True)

Accuracy (Bagging): 0.6246959342059539
```

c) Random Forest

In [11]:

```
# param = {
#     'n_estimators': [1, 10, 25, 50, 75, 100],
#     'max_depth': [12],
#     }

# rf = RandomForestClassifier(random_state=100)
# # instantiate the grid
# grid = GridSearchCV(rf, param, cv=5, scoring='accuracy')

# # fit the grid with data
# grid.fit(train_x, train_y)
# summarize results
# print("Best: %f using %s" % (grid.best_score_, grid.best_params_))
# Best: 0.634445 using {'max_depth': 12, 'n_estimators': 75}
rf = RandomForestClassifier(n_estimators=75, max_depth=12, random_state=100)
rf.fit(train_x, train_y)
pred_y = rf.predict(test_x)
rf_acur = accuracy_score(test_y, pred_y)
print("Accuracy (Random Forest):", rf_acur)
```

```
C:\Users\yamam\Anaconda3\lib\site-packages\ipykernel_launcher.py:1
6: DataConversionWarning: A column-vector y was passed when a 1d a
rray was expected. Please change the shape of y to (n_samples,), f
or example using ravel().
  app.launch_new_instance()

Accuracy (Random Forest): 0.6322251824394765
```

d) AdaBoost

In [12]:

```
# param = {
#     'n_estimators': [1, 10, 25, 50, 75, 100],
#     'learning_rate': [0.05]
# }

# rf = AdaBoostClassifier(base_estimator=cart, random_state=100)
# # instantiate the grid
# grid = GridSearchCV(rf, param, cv=5, scoring='accuracy')

# # fit the grid with data
# grid.fit(train_x, train_y)
# summarize results
# print("Best: %f using %s" % (grid.best_score_, grid.best_params_))
# Best: 0.638346 using {'learning_rate': 0.05, 'n_estimators': 100}
Ada_clf = AdaBoostClassifier(base_estimator=cart,n_estimators=100, learning_rat
e=0.05, random_state=100)
Ada_clf.fit(train_x, train_y)
pred_y = Ada_clf.predict(test_x)
Ada_acur = accuracy_score(test_y, pred_y)
print("Accuracy (AdaBoost):", Ada_acur)
```

```
C:\Users\yamam\Anaconda3\lib\site-packages\sklearn\utils\validatio
n.py:761: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_sample
s, ), for example using ravel().
  y = column_or_1d(y, warn=True)

Accuracy (AdaBoost): 0.6367427313795899
```

In [13]:

```
print("Final Results")
d = {"accuracy" : pd.Series([clf_acur, bag_acur, rf_acur, Ada_acur],
                            index=["Decision Tree", "Bagging", "Random Forest",
"AdaBoost"])}
df = pd.DataFrame(d)
print(df)
```

```
Final Results
               accuracy
Decision Tree  0.625159
Bagging        0.624696
Random Forest  0.632225
AdaBoost       0.636743
```

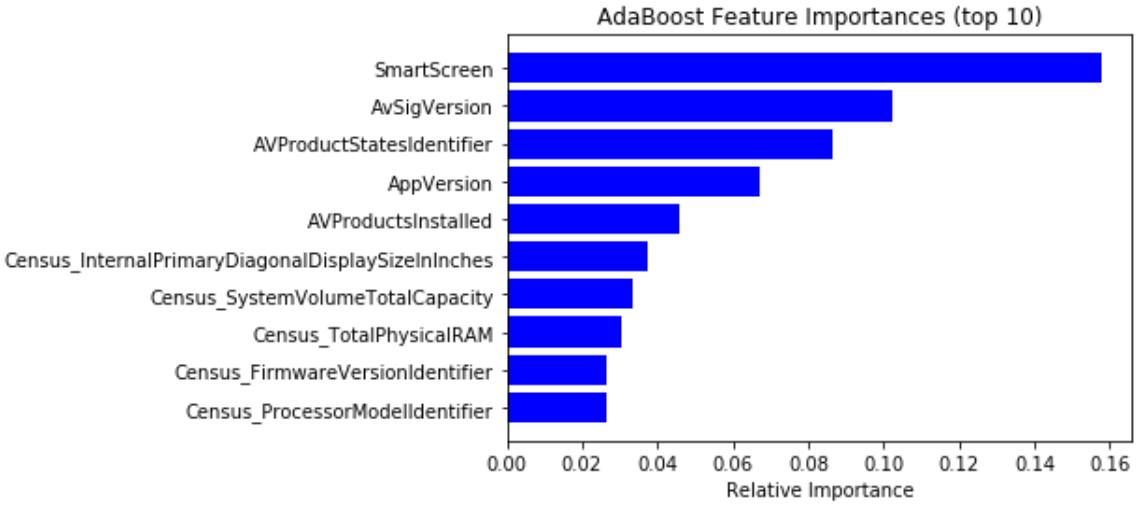
Overall, AdaBoost performed the best.

5. Feature Selection

1) Comparing feature importances in AdaBoost and Random Forest

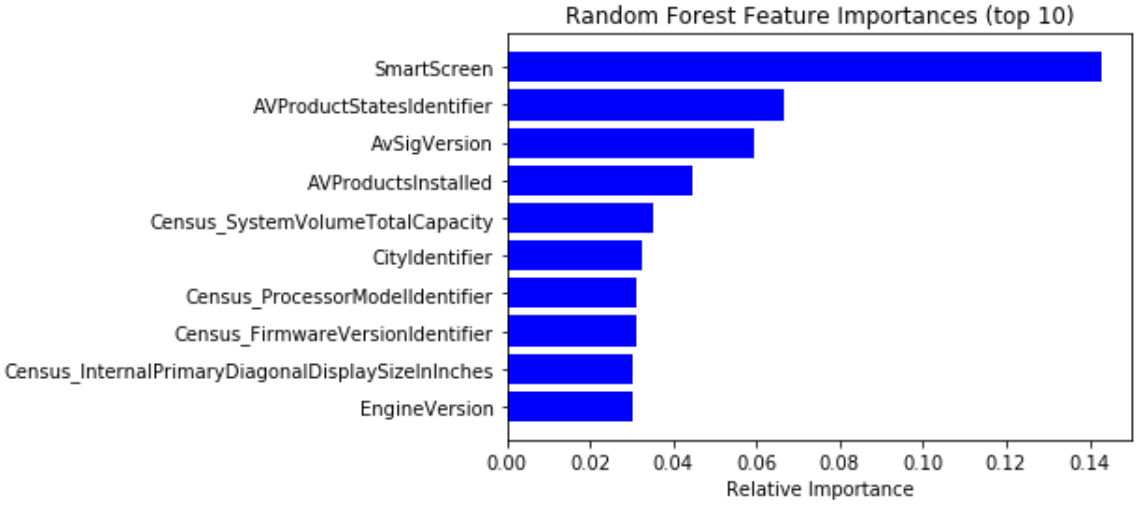
In [14]:

```
features = train.loc[:, train.columns != 'HasDetections'].columns

ada_importances = Ada_clf.feature_importances_
ada_indices = np.argsort(ada_importances)[-10:]  # top 10 features
plt.title('AdaBoost Feature Importances (top 10)')
plt.barh(range(len(ada_indices)), ada_importances[ada_indices], color='b', alig
n='center')
plt.yticks(range(len(ada_indices)), [features[i] for i in ada_indices])
plt.xlabel('Relative Importance')
plt.show()
```



In [15]:

```
features = train.loc[:, train.columns != 'HasDetections'].columns

rf_importances = rf.feature_importances_
rf_indices = np.argsort(rf_importances)[-10:]  # top 10 features
plt.title('Random Forest Feature Importances (top 10)')
plt.barh(range(len(rf_indices)), rf_importances[rf_indices], color='b', align=
'center')
plt.yticks(range(len(rf_indices)), [features[i] for i in rf_indices])
plt.xlabel('Relative Importance')
plt.show()
```



2) Comparing the Accuracy rate with the top 10 features using Logistic Regression

In [16]:

```
ad_important = [ada_importances[i] for i in ada_indices]
feature_top = [features[i] for i in ada_indices]
dropf = [i for i in features if i not in feature_top]

copytrain_x = train_x.copy()
copytest_x = test_x.copy()
copytrain_x.drop(dropf, axis=1, inplace=True)
copytest_x.drop(dropf, axis=1, inplace=True)

from sklearn.linear_model import LogisticRegression
lg = LogisticRegression(solver = 'lbfgs', ).fit(copytrain_x, train_y)
ada_log = lg.score(copytest_x, test_y)
print(ada_log)
```

```
0.5348082937565157


C:\Users\yamam\Anaconda3\lib\site-packages\sklearn\utils\validatio
n.py:761: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_sample
s, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [17]:

```
rf_important = [rf_importances[i] for i in rf_indices]
feature_top = [features[i] for i in rf_indices]
dropf = [i for i in features if i not in feature_top]

copytrain_x1 = train_x.copy()
copytest_x1 = test_x.copy()
copytrain_x1.drop(dropf, axis=1, inplace=True)
copytest_x1.drop(dropf, axis=1, inplace=True)

lg = LogisticRegression(solver = 'lbfgs', ).fit(copytrain_x1, train_y)
rf_log = lg.score(copytest_x1, test_y)
print(rf_log)
```

```
C:\Users\yamam\Anaconda3\lib\site-packages\sklearn\utils\validatio
n.py:761: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_sample
s, ), for example using ravel().
  y = column_or_1d(y, warn=True)

0.5224139928182555
```

In [18]:

```
print("Comparing Accuracy with the Top 10 Featuress")
d = {"accuracy" : pd.Series([ada_log, rf_log],
                            index=["AdaBoost", "Random Forest", ])}
df = pd.DataFrame(d)
print(df)
```

```
Comparing Accuracy with the Top 10 Featuress
               accuracy
AdaBoost       0.534808
Random Forest  0.522414
```