

RandomForest

June 4, 2020

Note: This program trains Random Forest classifier on Microsoft malware dataset.

@author Saruul Khasar

```
[1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, roc_curve, confusion_matrix
from datetime import datetime

pd.options.display.float_format = '{:.2f}'.format

import warnings; warnings.simplefilter('ignore')
```

```
[2]: import nltk
import sklearn

print('The nltk version is {}'.format(nltk.__version__))
print('The scikit-learn version is {}'.format(sklearn.__version__))
```

The nltk version is 3.4.5.

The scikit-learn version is 0.20.4.

1 Importing dataset

1.1 Determining data type

As we can see above, most data types are objects. Let's specify the data types for these objects, and it will reduce the memory footprint from 5.2+ GB to 1.4 GB.

```
[3]: dtypes = {
    'MachineIdentifier': 'category',
    'ProductName': 'category',
    'EngineVersion': 'category',
    'AppVersion': 'category',
    'AvSigVersion': 'category',
    'IsBeta': 'category',
    'RtpStateBitfield': 'float16',
    'IsSxsPassiveMode': 'category',
    'DefaultBrowsersIdentifier': 'category',
    'AVProductStatesIdentifier': 'category',
    'AVProductsInstalled': 'float16',
    'AVProductsInstalled': 'float16',
    'HasTpm': 'category',
    'CountryIdentifier': 'category',
    'CityIdentifier': 'category',
    'OrganizationIdentifier': 'category',
    'GeoNameIdentifier': 'category',
    'LocaleEnglishNameIdentifier': 'category',
    'Platform': 'category',
    'Processor': 'category',
    'OsVer': 'category',
    'OsBuild': 'category',
    'OsSuite': 'category',
    'OsPlatformSubRelease': 'category',
    'OsBuildLab': 'category',
    'SkuEdition': 'category',
    'IsProtected': 'category',
    'AutoSampleOptIn': 'category',
    'PuaMode': 'category',
    'SMode': 'category',
    'IeVerIdentifier': 'category',
    'SmartScreen': 'category',
    'Firewall': 'category',
    'UacLuaenable': 'category',
    'Census_MDC2FormFactor': 'category',
    'Census_DeviceFamily': 'category',
    'Census_OEMNameIdentifier': 'category',
    'Census_OEMModelIdentifier': 'category',
    'Census_ProcessorCoreCount': 'float16',
    'Census_ProcessorManufacturerIdentifier': 'category',
```

```

'Census_ProcessorModelIdentifier': 'category',
'Census_ProcessorClass': 'category',
'Census_PrimaryDiskTotalCapacity': 'float32',
'Census_PrimaryDiskTypeName': 'category',
'Census_SystemVolumeTotalCapacity': 'float32',
'Census_HasOpticalDiskDrive': 'category',
'Census_TotalPhysicalRAM': 'float32',
'Census_ChassisTypeName': 'category',
'Census_InternalPrimaryDiagonalDisplaySizeInInches': 'float16',
'Census_InternalPrimaryDisplayResolutionHorizontal': 'float16',
'Census_InternalPrimaryDisplayResolutionVertical': 'float16',
'Census_PowerPlatformRoleName': 'category',
'Census_InternalBatteryType': 'category',
'Census_InternalBatteryNumberOfCharges': 'float32',
'Census_OSVersion': 'category',
'Census_OSSoftwareArchitecture': 'category',
'Census_OSBranch': 'category',
'Census_OSBUILDNumber': 'category',
'Census_OSBUILDRevision': 'category',
'Census_OSEdition': 'category',
'Census_OSSkuName': 'category',
'Census_OSSInstallType': 'category',
'Census_OSSInstallLanguageIdentifier': 'category',
'Census_OSUILocaleIdentifier': 'category',
'Census_OSWUAutoUpdateOptionsName': 'category',
'Census_IsPortableOperatingSystem': 'category',
'Census_GenuineStateName': 'category',
'Census_ActivationChannel': 'category',
'Census_IsFlightingInternal': 'category',
'Census_IsFlightsDisabled': 'category',
'Census_FlightRing': 'category',
'Census_ThresholdOptIn': 'category',
'Census_FirmwareManufacturerIdentifier': 'category',
'Census_FirmwareVersionIdentifier': 'category',
'Census_IsSecureBootEnabled': 'category',
'Census_IsWIMBootEnabled': 'category',
'Census_IsVirtualDevice': 'category',
'Census_IsTouchEnabled': 'category',
'Census_IsPenCapable': 'category',
'Census_IsAlwaysOnAlwaysConnectedCapable': 'category',
'Wdft_IsGamer': 'category',
'Wdft_RegionIdentifier': 'category',
'HasDetections': 'category'
}

```

```
[4]: %%time
df = pd.read_csv('train.csv', dtype=dtypes)
```

```
#df.info()
```

```
CPU times: user 2min 51s, sys: 4.19 s, total: 2min 55s
Wall time: 2min 53s
```

2 Inspect features

2.1 Relative frequency

Let's inspect relative frequency of each feature along with the infection rate

```
[5]: %%script false --no-raise-error

def rel_freq_infect(col):
    """
    input: feature/column name string
    output: cross tabulation of frequency & infection percentage
    """
    try:
        rel_freq_infect = pd.crosstab(df[col].cat.add_categories('NaN').
        ↪fillna('NaN'), df['HasDetections'], dropna=False, normalize=True, ↪
        ↪margins=True).sort_values('All', ascending=False) * 100
    except:
        rel_freq_infect = pd.crosstab(df[col], df['HasDetections'], ↪
        ↪dropna=False, normalize=True, margins=True).sort_values('All', ↪
        ↪ascending=False) * 100
    return rel_freq_infect
```

```
[6]: %%script false --no-raise-error
```

```
%%time
# Let's skip MachineIdentifier feature because it's all unique values (takes
# too time)
col_indices = df.columns
for idx, col in enumerate(col_indices[1:]):
    dat_type = df[col].dtypes
    print('{}: Relative frequency percentage: {} ({})'.format(idx+1, col, ↪
    ↪dat_type))
    print(rel_freq_infect(col))
```

2.2 Unique values

Let's find number of unique values in each feature/column.

```
[7]: %%script false --no-raise-error
```

```

%%time
col_indices = df.columns
print("NUMBER OF UNIQUE VALUES")
for idx, col in enumerate(col_indices):
    dat_type = df[col].dtypes
    num_unique = df[col].nunique()
    print('{} . {} ({}): {}'.format(idx, col, dat_type, num_unique))

```

2.3 Relative frequency ratio

Let's find the ratio between percentage of infection over the percentage of frequency for each unique value and rank the features by highest ratio for any certain unique value. If this ratio is high, this would tell us the indicator has high chance of correlation with the infection.

```

[8]: %%script false --no-raise-error

def rel_freq_ratio(col):
    """
    input: feature/column name string
    output: cross tabulation of frequency & infection percentage
    """
    try:
        rel_freq = pd.crosstab(df[col].cat.add_categories('NaN').fillna('NaN'), df['HasDetections'], dropna=False, normalize=True, margins=True).
        sort_values('All', ascending=False) * 100
    except:
        rel_freq = pd.crosstab(df[col], df['HasDetections'], dropna=False, normalize=True, margins=True).sort_values('All', ascending=False) * 100

    rel_freq = rel_freq.loc[rel_freq['All'] > 0.02, :]
    rel_freq['detected/all'] = rel_freq.iloc[:, 1] / rel_freq.iloc[:, 2]
    rel_freq = rel_freq.sort_values('detected/all', ascending=False)
    return rel_freq

```

```

[9]: %%script false --no-raise-error

# Let's skip MachineIdentifier feature because it's all unique values (takes too time)
col_indices = df.columns
for idx, col in enumerate(col_indices[1:3]):
    dat_type = df[col].dtypes
    print('{} . Relative frequency ratio: {}'.format(idx+1, col, dat_type))
    print(rel_freq_ratio(col))

```

3 Drop/transform features

3.1 Drop features by percentage of null values

Let's drop **features/columns** with more than 20 percent of the values are Null.

```
[10]: null_check = df.isnull().sum() / len(df) * 100
print('Percentage of rows has null value %')
print(null_check.loc[null_check != 0].sort_values(ascending=False))
```

	Percentage of rows has null value %
PuaMode	99.97
Census_ProcessorClass	99.59
DefaultBrowsersIdentifier	95.14
Census_IsFlightingInternal	83.04
Census_InternalBatteryType	71.05
Census_ThresholdOptIn	63.53
Census_IsWIMBootEnabled	63.44
SmartScreen	35.61
OrganizationIdentifier	30.84
SMode	6.03
CityIdentifier	3.65
Wdft_RegionIdentifier	3.40
Wdft_IsGamer	3.40
Census_InternalBatteryNumberOfCharges	3.01
Census_FirmwareManufacturerIdentifier	2.05
Census_IsFlightsDisabled	1.80
Census_FirmwareVersionIdentifier	1.79
Census_OEMModelIdentifier	1.14
Census_OEMNameIdentifier	1.07
Firewall	1.02
Census_TotalPhysicalRAM	0.90
Census_IsAlwaysOnAlwaysConnectedCapable	0.80
Census_OSInstallLanguageIdentifier	0.67
IeVerIdentifier	0.66
Census_PrimaryDiskTotalCapacity	0.59
Census_SystemVolumeTotalCapacity	0.59
Census_InternalPrimaryDiagonalDisplaySizeInInches	0.53
Census_InternalPrimaryDisplayResolutionVertical	0.53
Census_InternalPrimaryDisplayResolutionHorizontal	0.53
Census_ProcessorModelIdentifier	0.46
	...
OsBuild	0.00
CountryIdentifier	0.00
LocaleEnglishNameIdentifier	0.00
Platform	0.00
Processor	0.00
SkuEdition	0.00

```

OsVer                         0.00
HasDetections                  0.00
AutoSampleOptIn                 0.00
Census_MDC2FormFactor          0.00
Census_IsPenCapable            0.00
Census_IsTouchEnabled           0.00
Census_IsSecureBootEnabled      0.00
Census_FlightRing                0.00
Census_ActivationChannel        0.00
Census_GenuineStateName         0.00
Census_IsPortableOperatingSystem 0.00
Census_OSWUAutoUpdateOptionsName 0.00
Census_OSUILocaleIdentifier     0.00
Census_OSIInstallTypeName       0.00
Census_OSSkuName                 0.00
Census_OSEdition                  0.00
Census_OSBuildRevision           0.00
Census_OSBuildNumber              0.00
Census_OSBranch                   0.00
Census_OSSArchitecture            0.00
Census_OSVersioin                  0.00
Census_HasOpticalDiskDrive       0.00
Census_DeviceFamily                0.00
AppVersion                        0.00
Length: 80, dtype: float64

```

```

[11]: col_drop = null_check.loc[null_check > 20].index
df.drop(col_drop, axis=1, inplace=True)

print("Initial number of features:  {} \
      \nRemaining number of features:  {} \
      \nNumber of columns dropped:    {}".format(83, len(df.keys()), 83-len(df.
      ↪keys())))
initial_features = len(df.keys())
remaining_features = len(df.keys())

```

```

Initial number of features:  83
Remaining number of features:  74
Number of columns dropped:    9

```

3.2 Drop duplicated features

- Census_OSVersioin: Census_OSVersioin = OsBuild + Census_OSBuildRevision
($10.0.17134.165 = 10.0 + 17134 + 165$)
- OsBuildLab: OsBuildLab = OsBuild + Census_OSSArchitecture + Census_OSBranch
($17134.1.amd64fre.rs4_release.180410-1804 = OsBuild + 1 + Census_OSSArchitecture + Census_OSBranch + 180410-1804$)
- Census_OSSkuName: almost the same as Census_OSEdition

- Processor: almost same as Census_OSSearchArchitecture
- Platform: OsVer contains Platform information
- Census_ChassisTypeName: almost the same as Census_MDC2FormFactor
- OsBuild: Census_OSBuildNumber contains OsBuild information
- OsPlatformSubRelease: Census_OSBranch contains OsPlatformSubRelease information

```
[12]: duplicated_features = ['Census_OSVersion', 'Census_OSSkuName', 'Processor', ↴
    ↴'Platform',
    'Census_ChassisTypeName', 'OsBuildLab', 'OsBuild', ↴
    ↴'OsPlatformSubRelease']
for i in duplicated_features:
    if i in df.keys():
        df.drop(i, axis=1, inplace=True)

print("Initial number of features:  {} \
    \nRemaining number of features:  {} \
    \nNumber of columns dropped:  {}".format(initial_features, len(df.
    ↴keys()), initial_features-len(df.keys())))
initial_features = len(df.keys())
remaining_features = len(df.keys())
```

Initial number of features: 74
 Remaining number of features: 66
 Number of columns dropped: 8

3.3 Drop meaningless features

Drop features that have useless information or have potential errors (3 features)

- MachineIdentifier is just a machine ID
- UacLuaenable has strange values rather than 0, 1 values
- Census_InternalBatteryNumberOfCharges has 4294967296 value in 26% of total rows

```
[13]: meaningless_features = ['MachineIdentifier', 'UacLuaenable', ↴
    ↴'Census_InternalBatteryNumberOfCharges']
df.drop(meaningless_features, axis=1, inplace=True)

print("Initial number of features:  {} \
    \nRemaining number of features:  {} \
    \nNumber of columns dropped:  {}".format(initial_features, len(df.
    ↴keys()), initial_features-len(df.keys())))
initial_features = len(df.keys())
remaining_features = len(df.keys())
```

Initial number of features: 66
 Remaining number of features: 63
 Number of columns dropped: 3

3.4 Fix some category values

- Census_PrimaryDiskTypeName: merged ‘UNKNOWN’ category to ‘Unspecified’
- Census_PowerPlatformRoleName: merged ‘UNKNOWN’ category to ‘Unspecified’

```
[14]: df['Census_PrimaryDiskTypeName'].replace('UNKNOWN', 'Unspecified', inplace=True)
df['Census_PrimaryDiskTypeName'].cat.remove_unused_categories(inplace=True)

df['Census_PowerPlatformRoleName'].replace('UNKNOWN', 'Unspecified', ▾
    ↳inplace=True)
df['Census_PowerPlatformRoleName'].cat.remove_unused_categories(inplace=True)
```

3.5 Split version some features into numeric features

Following features are combination of numbers that each number represent certain version of a machine parts; therefore, each number has meaning and we'll split these numbers to get broader information - EngineVersion - AppVersion - AvSigVersion - OsVer

```
[15]: df['EngineVersion_2'] = df['EngineVersion'].str.split('.', expand=True)[2].
    ↳astype('float16')
df['EngineVersion_3'] = df['EngineVersion'].str.split('.', expand=True)[3].
    ↳astype('float16')
df.drop('EngineVersion', axis=1, inplace=True)

df['AppVersion_1'] = df['AppVersion'].str.split('.', expand=True)[1].
    ↳astype('float16')
df['AppVersion_2'] = df['AppVersion'].str.split('.', expand=True)[2].
    ↳astype('float16')
df['AppVersion_3'] = df['AppVersion'].str.split('.', expand=True)[3].
    ↳astype('float16')
df.drop('AppVersion', axis=1, inplace=True)

df = df.loc[df['AvSigVersion'] != '1.2\x17;3.1144.0']
df['AvSigVersion_0'] = df['AvSigVersion'].str.split('.', expand=True)[0].
    ↳astype('float16')
df['AvSigVersion_1'] = df['AvSigVersion'].str.split('.', expand=True)[1].
    ↳astype('float16')
df['AvSigVersion_2'] = df['AvSigVersion'].str.split('.', expand=True)[2].
    ↳astype('float16')
df.drop('AvSigVersion', axis=1, inplace=True)

df['OsVer_0'] = df['OsVer'].str.split('.', expand=True)[0].astype('float16')
df['OsVer_1'] = df['OsVer'].str.split('.', expand=True)[1].astype('float16')
df['OsVer_2'] = df['OsVer'].str.split('.', expand=True)[2].astype('float16')
df['OsVer_3'] = df['OsVer'].str.split('.', expand=True)[3].astype('float16')
df.drop('OsVer', axis=1, inplace=True)
```

```
[16]: print("Initial number of features:  {} \
    \nRemaining number of features:  {} \
    \nNumber of columns added:  {}".format(initial_features, len(df.
    ↪keys()), len(df.keys())-initial_features))

initial_features = len(df.keys())
remaining_features = len(df.keys())
```

Initial number of features: 63
 Remaining number of features: 71
 Number of columns added: 8

3.6 Drop features with not enough variation

Drop categorical features if a single category takes more than 95% of rows: Not enough variation

```
[17]: columns = df.columns
columns_not_var = []

for col in columns:
    if df[col].dtypes.name == 'category' and df[col].
    ↪value_counts(normalize=True).sort_values(ascending=False).
    ↪reset_index(drop=True)[0] > 0.95:
        columns_not_var.append(col)

df.drop(columns_not_var, axis=1, inplace=True)

print("Initial number of features:  {} \
    \nRemaining number of features:  {} \
    \nNumber of columns dropped:  {}".format(initial_features, len(df.
    ↪keys()), initial_features-len(df.keys())))
initial_features = len(df.keys())
remaining_features = len(df.keys())
```

Initial number of features: 71
 Remaining number of features: 59
 Number of columns dropped: 12

3.7 Drop rows if categorical features have missing values

```
[18]: types = df.dtypes
columns_cat = list(types[types == 'category'].index)
df.dropna(subset=columns_cat, inplace=True)
```

4 Feature selection

4.1 Drop HasDetections feature

'HasDetection' is our label feature (dependent variable).

```
[19]: label = df['HasDetections'].astype('int8')
df.drop(['HasDetections'], axis=1, inplace=True)
```

4.2 List of remaining features

```
[20]: for idx, col in enumerate(df.keys()):
    print('{}. {}'.format(idx+1, col))
```

1. RtpStateBitfield
2. AVProductStatesIdentifier
3. AVProductsInstalled
4. AVProductsEnabled
5. CountryIdentifier
6. CityIdentifier
7. GeoNameIdentifier
8. LocaleEnglishNameIdentifier
9. OsSuite
10. SkuEdition
11. IsProtected
12. IeVerIdentifier
13. Census_MDC2FormFactor
14. Census_OEMNameIdentifier
15. Census_OEMModelIdentifier
16. Census_ProcessorCoreCount
17. Census_ProcessorManufacturerIdentifier
18. Census_ProcessorModelIdentifier
19. Census_PrimaryDiskTotalCapacity
20. Census_PrimaryDiskTypeName
21. Census_SystemVolumeTotalCapacity
22. Census_HasOpticalDiskDrive
23. Census_TotalPhysicalRAM
24. Census_InternalPrimaryDiagonalDisplaySizeInInches
25. Census_InternalPrimaryDisplayResolutionHorizontal
26. Census_InternalPrimaryDisplayResolutionVertical
27. Census_PowerPlatformRoleName
28. Census_OSSoftwareArchitecture
29. Census_OSBranch
30. Census_OSBUILDNumber
31. Census_OSBUILDRevision
32. Census_OSEdition
33. Census_OSIInstallTypeName
34. Census_OSIInstallLanguageIdentifier

35. Census_OSUILocaleIdentifier
 36. Census_OSWUAutoUpdateOptionsName
 37. Census_GenuineStateName
 38. Census_ActivationChannel
 39. Census_FlightRing
 40. Census_FirmwareManufacturerIdentifier
 41. Census_FirmwareVersionIdentifier
 42. Census_IsSecureBootEnabled
 43. Census_IsTouchEnabled
 44. Census_IsAlwaysOnAlwaysConnectedCapable
 45. Wdft_IsGamer
 46. Wdft_RegionIdentifier
 47. EngineVersion_2
 48. EngineVersion_3
 49. AppVersion_1
 50. AppVersion_2
 51. AppVersion_3
 52. AvSigVersion_0
 53. AvSigVersion_1
 54. AvSigVersion_2
 55. OsVer_0
 56. OsVer_1
 57. OsVer_2
 58. OsVer_3

```
[21]: settings_list = [
    'IsBeta', 'RtpStateBitfield', 'IsSxsPassiveMode', 'AVProductsInstalled',
    'AVProductsEnabled', 'HasTpm', 'LocaleEnglishNameIdentifier', 'Platform',
    'OsVer', 'OsBuild', 'OsSuite', 'OsPlatformSubRelease',
    'OsBuildLab', 'SkuEdition', 'IsProtected', 'AutoSampleOptIn',
    'PuaMode', 'SMode', 'IeVerIdentifier', 'SmartScreen',
    □
    ↳ 'Firewall', 'UacLuaenable', 'Census_MDC2FormFactor', 'Census_OEMNameIdentifier',
    □
    ↳ 'Census_OEMModelIdentifier', 'Census_PrimaryDiskTotalCapacity', 'Census_PrimaryDiskTypeName',
    □
    ↳ 'Census_HasOpticalDiskDrive', 'Census_PowerPlatformRoleName', 'Census_OSVersions', 'Census_OSSAri',
    □
    ↳ 'Census_OSSBranch', 'Census_OSBUILDNumber', 'Census_OSBUILDRevision', 'Census_OSEdition',
    □
    ↳ 'Census_OSSkuName', 'Census_OSSInstallType', 'Census_OSSInstallLanguageIdentifier', 'Census_OSS',
    □
    ↳ 'Census_OSWUAutoUpdateOptionsName', 'Census_IsPortableOperatingSystem', 'Census_GenuineStateN,
    □
    ↳ 'Census_IsFlightingInternal', 'Census_IsFlightsDisabled', 'Census_FlightRing', 'Census_Threshold
    □
    ↳ 'Census_IsSecureBootEnabled', 'Census_IsWIMBootEnabled', 'Census_IsVirtualDevice', 'Census_IsT
```

```
    ↵'Census_IsPenCapable', 'Census_IsAlwaysOnAlwaysConnectedCapable', 'Wdft_IsGamer'  
]
```

```
[22]: # We will train the model on three sets of features  
"""  
1. All features  
2. All settings  
3. All features except settings  
"""  
  
df_all = df  
  
settings = []  
for i in df.keys():  
    if i in settings_list:  
        settings.append(i)  
df_settings = df.loc[:,(settings)]  
  
notsettings = []  
for i in df.keys():  
    if i not in settings_list:  
        notsettings.append(i)  
df_notsettings = df.loc[:,(notsettings)]
```

```
[23]: %%time  
  
# Transform numeric and categorical features through different pipelines  
"""  
1. Numeric features are imputed with median value then scaled to standard  
↳normal distribution.  
2. Categorical features are encoded by one hot encoder  
  
ColumnTransformer allows to implement pipelines together.  
"""  
  
num_pipeline = Pipeline([  
    ('imputer', SimpleImputer(strategy='median')),  
    ('std_scaler', StandardScaler())  
])  
  
cat_pipeline = Pipeline([  
    ('oh_encoder', OneHotEncoder())  
])  
  
types_all = df_all.dtypes
```

```

types_settings = df_settings.dtypes
types_notsettings = df_notsettings.dtypes

col_num_all = types_all[types_all != 'category'].index
col_cat_all = types_all[types_all == 'category'].index
col_num_settings = types_settings[types_settings != 'category'].index
col_cat_settings = types_settings[types_settings == 'category'].index
col_num_notsettings = types_notsettings[types_notsettings != 'category'].index
col_cat_notsettings = types_notsettings[types_notsettings == 'category'].index

pipeline_all = ColumnTransformer([
    ('num', num_pipeline, col_num_all),
    ('cat', cat_pipeline, col_cat_all)
])

pipeline_settings = ColumnTransformer([
    ('num', num_pipeline, col_num_settings),
    ('cat', cat_pipeline, col_cat_settings)
])

pipeline_notsettings = ColumnTransformer([
    ('num', num_pipeline, col_num_notsettings),
    ('cat', cat_pipeline, col_cat_notsettings)
])

# Use the prepared pipeline on our 3 sets of features
data_all = pipeline_all.fit_transform(df_all)
data_settings = pipeline_settings.fit_transform(df_settings)
data_notsettings = pipeline_notsettings.fit_transform(df_notsettings)

```

CPU times: user 5min 56s, sys: 1min 2s, total: 6min 59s
Wall time: 4min 51s

5 Training the dataset

5.1 Random Forest classifier on all features

5.1.1 Model 1: All features

```
[24]: %%time
# %%script false --no-raise-error

# Let's split the data into training & testing set
X_train, X_test, y_train, y_test = train_test_split(data_all, np.array(label), test_size=0.3, random_state=42)
```

```

print('Train X: {}'.format(X_train.shape))
print('Test X:  {}'.format(X_test.shape))
print('Train y:  {}'.format(y_train.shape))
print('Test y:   {}'.format(y_test.shape))

# Let's train the model
clf = RandomForestClassifier(max_depth=50,
                             n_estimators=100,
                             min_samples_split=10,
                             n_jobs=-1)
clf.fit(X_train, y_train)

```

```

Train X: (5224247, 346233)
Test X: (2238963, 346233)
Train y: (5224247,)
Test y: (2238963,)
CPU times: user 4h 38min 29s, sys: 50.1 s, total: 4h 39min 19s
Wall time: 25min 32s

```

[25]:

```

%%time
# %%script false --no-raise-error

# AUC score
pred_train = clf.predict(X_train)
pred_test = clf.predict(X_test)

print('Accuracy in training set: {:.3f}'.format(accuracy_score(y_train, pred_train)))
print('Accuracy in testing set: {:.3f}'.format(accuracy_score(y_test, pred_test)))

print('\nAUC in training set: {:.3f}'.format(roc_auc_score(y_train, pred_train)))
print('AUC in testing set: {:.3f}'.format(roc_auc_score(y_test, pred_test)))

```

```

Accuracy in training set: 0.630
Accuracy in testing set: 0.613

AUC in training set: 0.629
AUC in testing set: 0.613
CPU times: user 6min 48s, sys: 7.77 s, total: 6min 56s
Wall time: 44.1 s

```

```
[26]: %%script false --no-raise-error

# Prediction score/probability
pred_score_test = clf.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, pred_score_test)
auc_score = roc_auc_score(y_test, pred_score_test)

def plot_roc_curve(fpr, tpr, auc_score, legend=None):
    plt.plot(fpr, tpr, linewidth=2, c='red')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.legend(['{}', 'AUC: {:.2f}'.format(legend, auc_score)], 'Random guess'],
               loc='lower right')
    plt.title('ROC Curve', weight='bold')

plot_roc_curve(fpr, tpr, auc_score, 'Random Forest')


# Confusion matrix
def plot_confusion_matrix(cm, classes,
                         normalize=False,
                         title='Confusion matrix',
                         cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        cm = np.round(cm, 2)
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

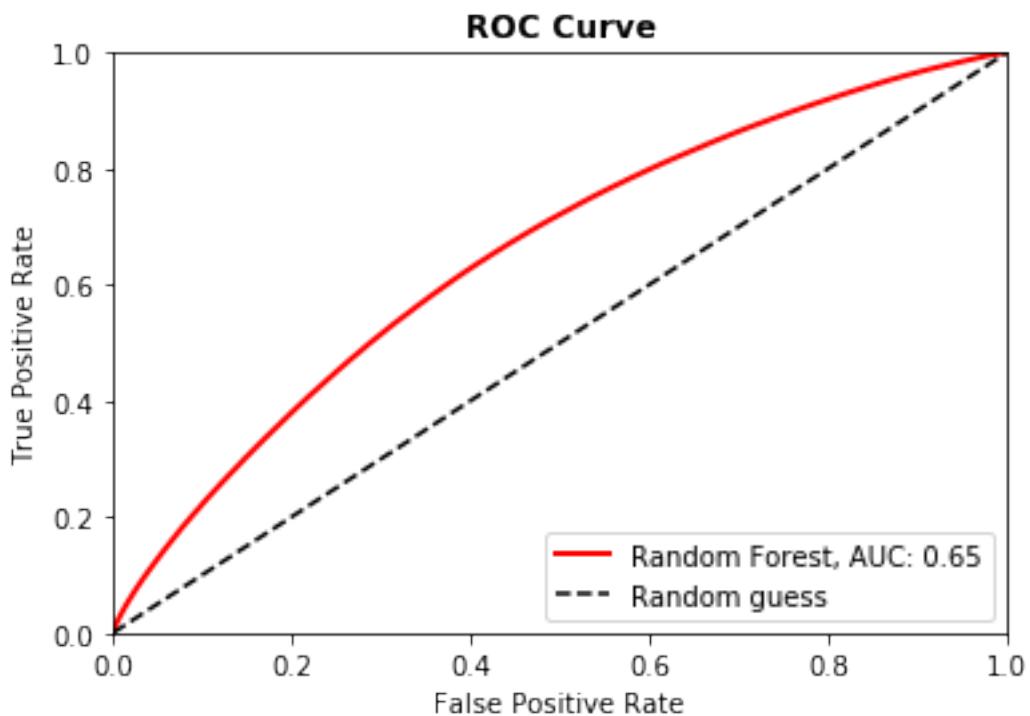
    print(cm)
```

```

thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')

```



```

[27]: %%script false --no-raise-error

y_pred = pred_test
class_names = ['Not Detected', 'Detected']

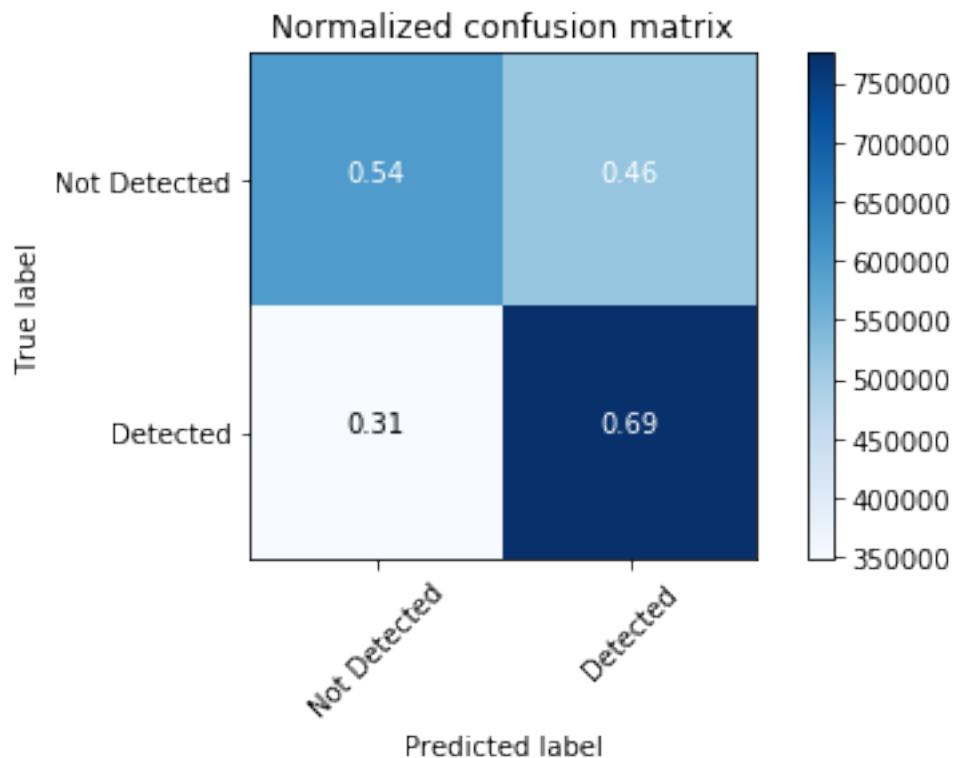
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure()
plot_confusion_matrix(cm, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

```

```
plt.show()
```

```
Normalized confusion matrix  
[[0.54 0.46]  
 [0.31 0.69]]
```



5.1.2 Model 2: All settings

```
[28]: %%time  
#%%script false --no-raise-error  
  
# Let's split the data into training & testing set  
X_train, X_test, y_train, y_test = train_test_split(data_settings, np.  
array(label), test_size=0.3, random_state=42)  
  
print('Train X: {}'.format(X_train.shape))  
print('Test X:  {}'.format(X_test.shape))  
print('Train y: {}'.format(y_train.shape))  
print('Test y:  {}'.format(y_test.shape))
```

```
# Let's train the model
clf = RandomForestClassifier(max_depth=50,
                            n_estimators=100,
                            min_samples_split=10,
                            n_jobs=-1)
clf.fit(X_train, y_train)
```

Train X: (5224247, 165189)
Test X: (2238963, 165189)
Train y: (5224247,)
Test y: (2238963,)
CPU times: user 4h 51min 48s, sys: 47.7 s, total: 4h 52min 36s
Wall time: 26min 38s

```
[29]: %%script false --no-raise-error

# AUC score
pred_train = clf.predict(X_train)
pred_test = clf.predict(X_test)

print('Accuracy in training set: {:.3f}'.format(accuracy_score(y_train, pred_train)))
print('Accuracy in testing set: {:.3f}'.format(accuracy_score(y_test, pred_test)))

print('\nAUC in training set: {:.3f}'.format(roc_auc_score(y_train, pred_train)))
print('AUC in testing set: {:.3f}'.format(roc_auc_score(y_test, pred_test)))
```

Accuracy in training set: 0.610
Accuracy in testing set: 0.587

AUC in training set: 0.610
AUC in testing set: 0.587

```
[30]: %%script false --no-raise-error

# Prediction score/probability
pred_score_test = clf.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, pred_score_test)
auc_score = roc_auc_score(y_test, pred_score_test)

def plot_roc_curve(fpr, tpr, auc_score, legend=None):
    plt.plot(fpr, tpr, linewidth=2, c='red')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False Positive Rate')
```

```

plt.ylabel('True Positive Rate')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.legend(['{} , AUC: {:.2f}'.format(legend, auc_score), 'Random guess'], loc='lower right')
plt.title('ROC Curve', weight='bold')

plot_roc_curve(fpr, tpr, auc_score, 'Random Forest')


# Confusion matrix
def plot_confusion_matrix(cm, classes,
                           normalize=False,
                           title='Confusion matrix',
                           cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

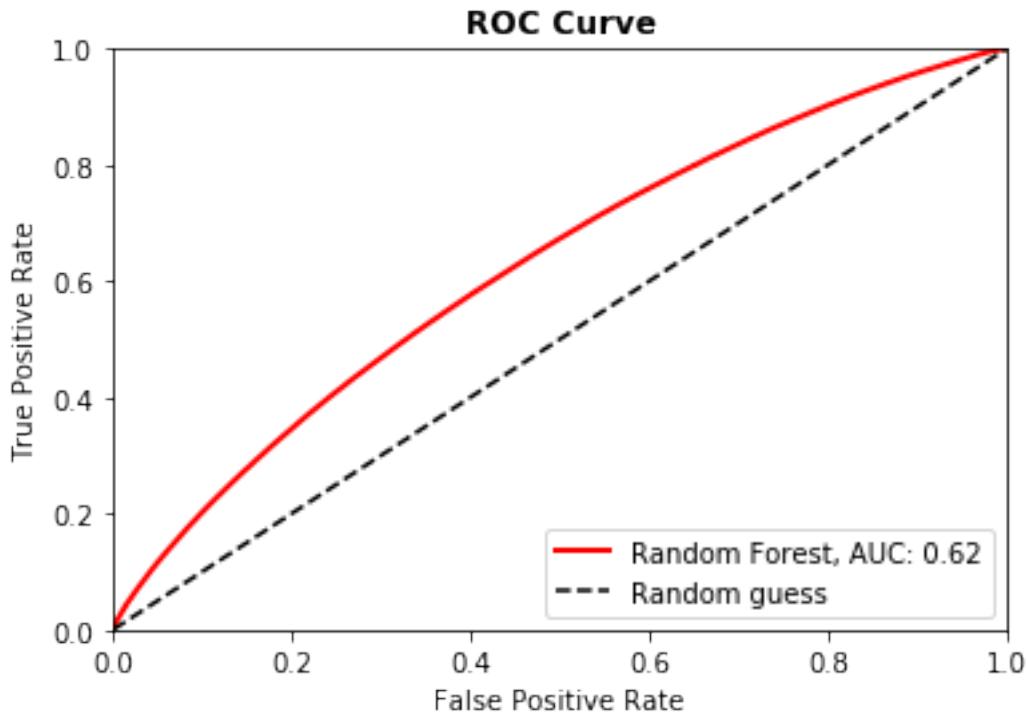
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        cm = np.round(cm, 2)
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, cm[i, j],
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

```



```
[31]: %%script false --no-raise-error

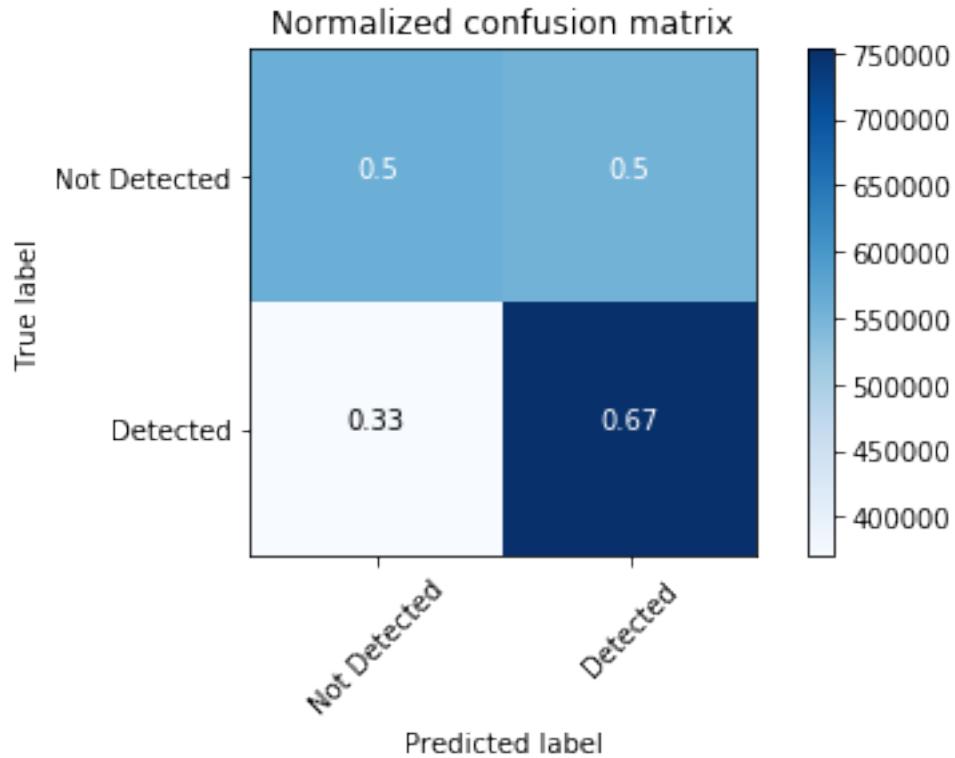
y_pred = pred_test
class_names = ['Not Detected', 'Detected']

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure()
plot_confusion_matrix(cm, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```

```
Normalized confusion matrix
[[0.5  0.5 ]
 [0.33 0.67]]
```



5.1.3 Model 3: All features except settings

```
[32]: %%time

# Let's split the data into training & testing set
X_train, X_test, y_train, y_test = train_test_split(data_notsettings, np.
    ↪array(label), test_size=0.3, random_state=42)

print('Train X: {}'.format(X_train.shape))
print('Test X:  {}'.format(X_test.shape))
print('Train y: {}'.format(y_train.shape))
print('Test y:  {}'.format(y_test.shape))

# Let's train the model
clf = RandomForestClassifier(max_depth=50,
                            n_estimators=100,
                            min_samples_split=10,
                            n_jobs=-1)
clf.fit(X_train, y_train)
```

Train X: (5224247, 181044)

```
Test X: (2238963, 181044)
Train y: (5224247,)
Test y: (2238963,)
CPU times: user 2h 44min 12s, sys: 33.3 s, total: 2h 44min 46s
Wall time: 14min 51s
```

```
[33]: %%script false --no-raise-error

# AUC score
pred_train = clf.predict(X_train)
pred_test = clf.predict(X_test)

print('Accuracy in training set: {:.3f}'.format(accuracy_score(y_train, pred_train)))
print('Accuracy in testing set: {:.3f}'.format(accuracy_score(y_test, pred_test)))

print('\nAUC in training set: {:.3f}'.format(roc_auc_score(y_train, pred_train)))
print('AUC in testing set: {:.3f}'.format(roc_auc_score(y_test, pred_test)))
```

```
Accuracy in training set: 0.612
Accuracy in testing set: 0.604

AUC in training set: 0.612
AUC in testing set: 0.603
```

```
[34]: %%script false --no-raise-error

# Prediction score/probability
pred_score_test = clf.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, pred_score_test)
auc_score = roc_auc_score(y_test, pred_score_test)

def plot_roc_curve(fpr, tpr, auc_score, legend=None):
    plt.plot(fpr, tpr, linewidth=2, c='red')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.legend(['{} AUC: {:.2f}'.format(legend, auc_score), 'Random guess'],
              loc='lower right')
    plt.title('ROC Curve', weight='bold')

plot_roc_curve(fpr, tpr, auc_score, 'Random Forest')
```

```

# Confusion matrix
def plot_confusion_matrix(cm, classes,
                         normalize=False,
                         title='Confusion matrix',
                         cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

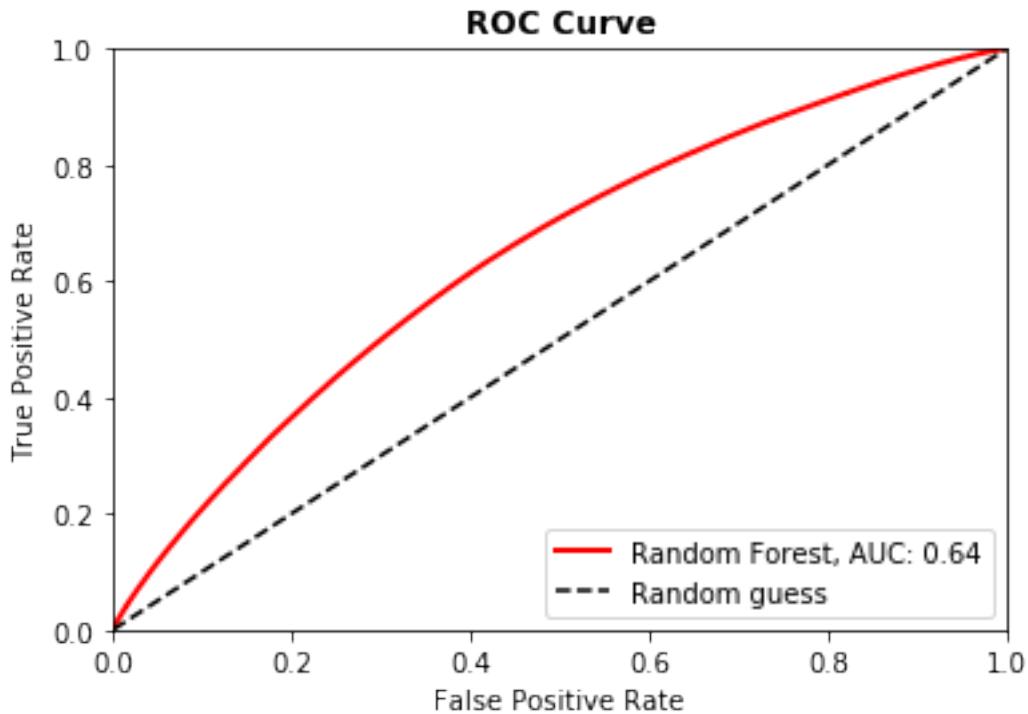
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        cm = np.round(cm, 2)
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, cm[i, j],
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

```



```
[35]: %%script false --no-raise-error

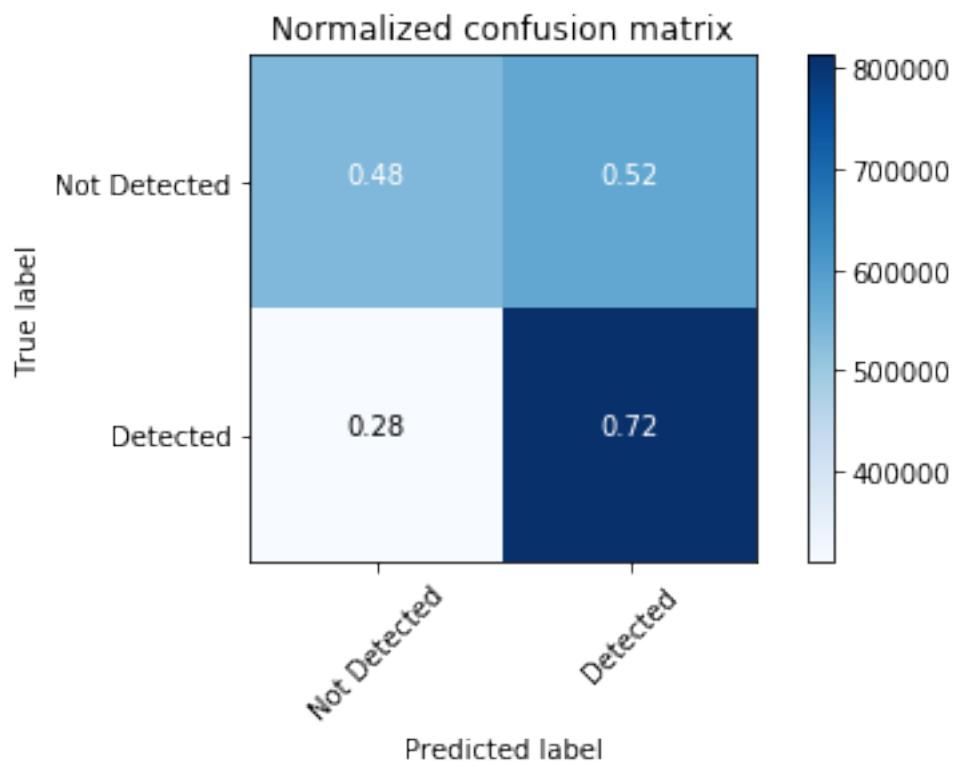
y_pred = pred_test
class_names = ['Not Detected', 'Detected']

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure()
plot_confusion_matrix(cm, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```

```
Normalized confusion matrix
[[0.48 0.52]
 [0.28 0.72]]
```



[]: