E-Surveillance Alert Classification

Prepared by,

Sridhar Ponnusamy

Kamakshi Mansukhani

Sarah Nowlan

Param Jashwal

Table of Contents

1. Business Problem

Description:

Prevent break-ins before they occur using IoT security cameras with built-in computer vision capabilities, reducing the need for human intervention. Automated security to safeguard and alert against threats from intrusion or fire using multi-capability sensors such as vibration, motion, smoke, fire, etc. Ensure the safety of both monetary and intellectual assets with round-the-clock surveillance and controlled access management.

2. Problem Statement

We are tasked with classifying the alert whether it is Critical, Normal, or Testing which is received from the various sensors. Such as vibration, motion, smoke, fire, Panic, shutter(Door sensor).

True_Normal

Testing the sensors(Smoke, Panic, fire), Smoke Alert due to AC maintenance / UPS maintenance, Cash loading, Pressing the panic switches unknowingly by bank staffs…

True_Critical

Smoke, Fire, Network Connection error, PIR, Panic(Fire, Theft Attempt, in ATM and Bank)

Breaking the ATM Machine(Vibration sensor will be activated)

Thieves are showing the Weapons to Bank Staff (Panic switch must be pressed by Bank Staff to get the alert)

Breaking the Doors(In this case, the Shutter sensor will send the alert)

False_Normal

While opening the bank, make sure to enter the password to change the mode. Otherwise, PIR and Alarm will be generated.

Some interesting reasons:

*During renovation work at the bank (smoke and fire sensor will be activated due to dust)

*During sanitization of the branch.

* Cleaning or sweeping the branch.

* Auditing(Auditor will check by burning the papers inside the branch)

* Birthday celebration(candle smoke will generate the alert)

False_Critcal

False_Critical = No activity, Sensor Malfunctioning (keep on getting the alert). Alert is received from the critical sensor even though there is no activity. Such as Smoke, fire, PIR(Motion detection sensor)

3.Real-world/Business Objectives and Constraints

1. The cost of misclassification can be very high.

2. No strict latency concerns.

4. Mapping the real-world problem to an ML problem

Type of Machine Learning Problem

Supervised Learning:

It is a Multi classification problem, for a given sensor data we need to classify if it is critical, Normal, or Testing.

Train and Test Construction

We build train and test by randomly splitting in the ratio of 70:30 or 80:20 whatever we choose as we have sufficient points to work with.

Importing Necessary Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import rcParams
from matplotlib.cm import rainbow
%matplotlib inline
import warnings

warnings.filterwarnings('ignore')
```

5. Reading the Dataset

```
data1= pd.read_csv('Master - Ack Time Analysis From April-20 to August-20 - Details.csv')
data=data1.drop(['Branch'],axis=1)
data
```

Showing first 5 rows of the dataset

```
In [ ]: data.head(5)

In [ ]: #Diiferent types of sensor are used in the bank

In [133]: sensors_type=data['SENSOR NAME (Standard)'].drop_duplicates()
          sensors_type

Out[133]: 0               Network
          1                   PIR
          5             Vibration
          9                Hooter
          10         Front Shutter
          42           Stop Panic
          43           Arm Dis-Arm
          59                Panic
          92                Smoke
          119         Energy Vault
          197           Chest Door
          7985       UPS Room Door
          11871      ATM Back Door
          82659         Chest Door
          82666           Network
          82667         Vibration
          82713              Smoke
          82719      Front Shutter
          82737              Panic
          113385               NaN
          Name: SENSOR NAME (Standard), dtype: object
```

Shape

Data. shape

(185687(Rows), 16(Columns))

Counting the output values for each category

data.Status.value_counts().count

<bound method Series.count of
True_Normal 110395
Testing 36442
False_False 23755
True_Critical 10505
FALSE 4590
Name: Status, dtype: int64>

6. Data Preprocessing

Features of the data set

In [6]:

```
data.columns
```

Out[6]:

```
Index(['DATE', 'SOL ID', 'Branch', 'SENSOR (As of Portal)', 'Region', 'STATE',
       'SENSOR NAME (Standard)', ' EVENT', 'EVENT DATE AND TIME',
       'AKNOWLEDGE STATUS', 'Confirmation', 'LOG ID', '2nd AKNOWLEDGE STATUS',
       'Status', 'Month', 'Reason'],
      dtype='object')
```

Number of distinct observations

In [13]:

```
data.nunique()
```

Out[13]:

```
DATE                        177
SOL ID                     1721
Branch                     1793
SENSOR (As of Portal)      1327
Region                        4
STATE                        22
SENSOR NAME (Standard)       19
 EVENT                       14
EVENT DATE AND TIME       73423
AKNOWLEDGE STATUS          9864
Confirmation                 36
LOG ID                   185475
2nd AKNOWLEDGE STATUS     10078
Status                        5
Month                         6
Reason                       18
dtype: int64
```

Checking for NaN/null values

```
p.isna().sum()
```

```
SOL ID                    0
Region                    0
STATE                     0
 EVENT                    0
AKNOWLEDGE STATUS         0
Confirmation              0
2nd AKNOWLEDGE STATUS     0
Status                    0
dtype: int64
```

Feature Importance

Feature Importance

```
]: #importing the ExtraTreesClassifier
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt
```

```
']: model = ExtraTreesClassifier()
```

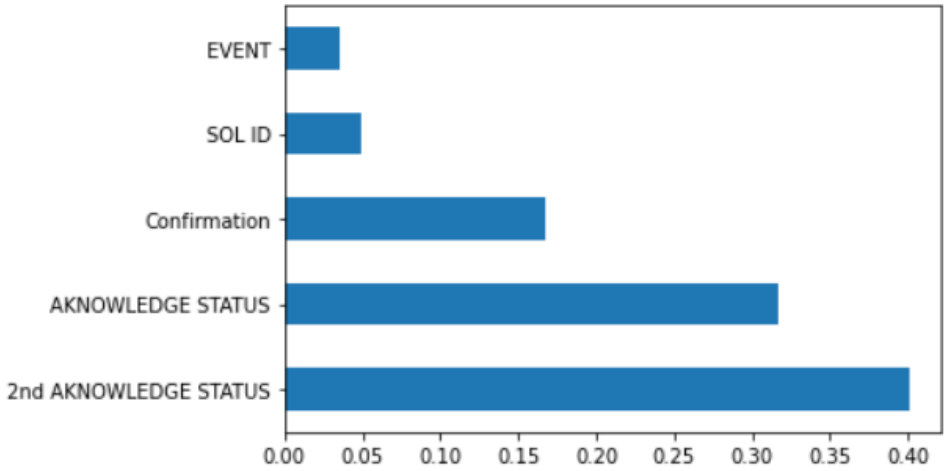
```
]: model.fit(X,y)

print(model.feature_importances_)
```

```
[0.04941096 0.01162244 0.01898071 0.03501246 0.31675196 0.16743766
 0.40078382]
```

Which are the best features?

```
: #Top 5 important features
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(5).plot(kind='barh')
plt.show()
```




```
#columns to list
feature_names = X.columns.tolist()
feature_names
```

```
['SOL ID',
 'Region',
 'STATE',
 ' EVENT',
 'AKNOWLEDGE STATUS',
 'Confirmation',
 '2nd AKNOWLEDGE STATUS']
```

Removing the Unnecessary Columns

```
p=data.drop(['LOG ID','SENSOR (As of Portal)','DATE','SENSOR NAME (Standard)', 'Month','EVENT DATE AND TIME','Reason'], axis=1)
```

Handling Categorical Data

Here we are converting categorical data to numerical data using a label encoder

```
from sklearn.preprocessing import LabelEncoder
labelencoder_X=LabelEncoder()
xm=p.apply(LabelEncoder().fit_transform)
xm
```

	SOL ID	Region	STATE	EVENT	AKNOWLEDGE STATUS	Confirmation	2nd AKNOWLEDGE STATUS	Status
0	1255	3	11	6	7872	14	8079	3
1	1166	2	17	8	434	15	442	0
2	1092	2	0	6	7872	14	8079	3
3	1525	3	11	6	7872	14	8079	3
4	11	1	15	6	7872	14	8079	3
...	...	...	...	...	...	...	...	...
185682	1369	3	16	5	8337	19	8587	4
185683	1369	3	16	12	8337	19	8587	4
185684	807	0	13	10	2968	19	3045	1
185685	807	0	13	5	2968	19	3045	1
185686	807	0	13	12	2968	19	3045	1

185687 rows × 8 columns

Splitting input features

```
X=xm.iloc[:,:-1]
X
```

	SOL ID	Region	STATE	EVENT	AKNOWLEDGE STATUS	Confirmation	2nd AKNOWLEDGE STATUS
0	1255	3	11	6	7872	14	8079
1	1166	2	17	8	434	15	442
2	1092	2	0	6	7872	14	8079
3	1525	3	11	6	7872	14	8079
4	11	1	15	6	7872	14	8079
...	...	...	...	...	...	...	...
185682	1369	3	16	5	8337	19	8587
185683	1369	3	16	12	8337	19	8587
185684	807	0	13	10	2968	19	3045
185685	807	0	13	5	2968	19	3045
185686	807	0	13	12	2968	19	3045

185687 rows × 7 columns

Output feature

```
y=xm.iloc[:,7]
y
```

```
0          3
1          0
2          3
3          3
4          3
          ..
185682     4
185683     4
185684     1
185685     1
185686     1
Name: Status, Length: 185687, dtype: int32
```

Calculate the correlation with y for each feature

```
xm.corr()
```

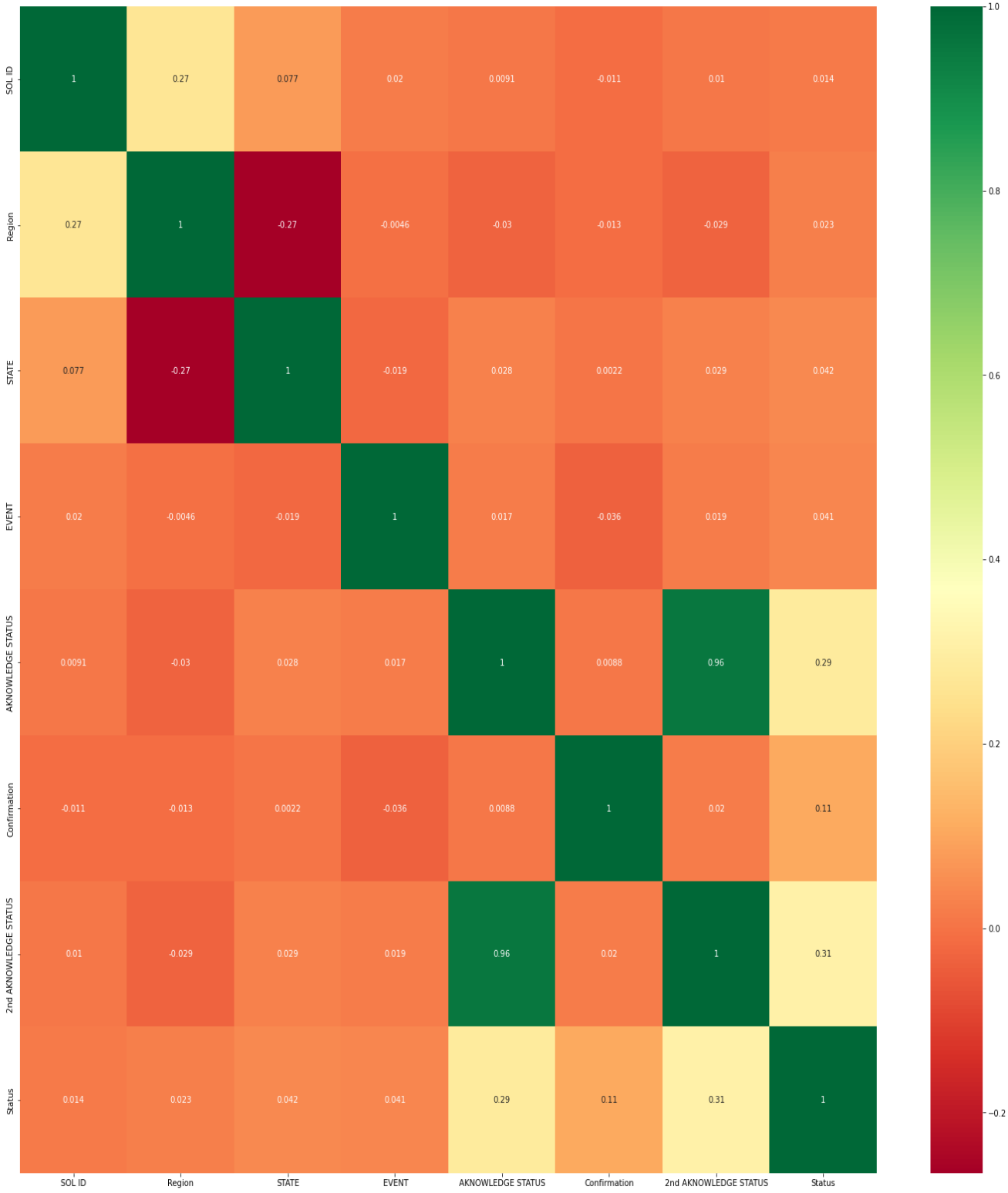
	SOL ID	Region	STATE	EVENT	AKNOWLEDGE STATUS	Confirmation	2nd AKNOWLEDGE STATUS	Status
SOL ID	1.000000	0.265381	0.076566	0.019894	0.009091	-0.011106	0.010405	0.013946
Region	0.265381	1.000000	-0.265645	-0.004633	-0.030057	-0.013371	-0.028609	0.023269
STATE	0.076566	-0.265645	1.000000	-0.018737	0.028354	0.002211	0.028944	0.042123
EVENT	0.019894	-0.004633	-0.018737	1.000000	0.017059	-0.036335	0.019393	0.040683
AKNOWLEDGE STATUS	0.009091	-0.030057	0.028354	0.017059	1.000000	0.008753	0.959627	0.287432
Confirmation	-0.011106	-0.013371	0.002211	-0.036335	0.008753	1.000000	0.019503	0.109857
2nd AKNOWLEDGE STATUS	0.010405	-0.028609	0.028944	0.019393	0.959627	0.019503	1.000000	0.308584
Status	0.013946	0.023269	0.042123	0.040683	0.287432	0.109857	0.308584	1.000000

Heat Map

In [151]:

```
#get correlations of each features in dataset
corrmat = xm.corr()
corr_features = corrmat.index
plt.figure(figsize=(25,25))
#plot heat map

c=sns.heatmap(xm[corr_features].corr(),annot=True,cmap="RdYlGn")
```


SOL ID
Region
STATE
EVENT
AKNOWLEDGE STATUS
Confirmation
2nd AKNOWLEDGE STATUS
Status
1 0.27 0.077 0.02 0.0091 -0.011 0.01 0.014
0.27 1 -0.27 -0.0046 -0.03 -0.013 -0.029 0.023
0.077 -0.27 1 -0.019 0.028 0.0022 0.029 0.042
0.02 -0.0046 -0.019 1 0.017 -0.036 0.019 0.041
0.0091 -0.03 0.028 0.017 1 0.0088 0.96 0.29
-0.011 -0.013 0.0022 -0.036 0.0088 1 0.02 0.11
0.01 -0.029 0.029 0.019 0.96 0.02 1 0.31
0.014 0.023 0.042 0.041 0.29 0.11 0.31 1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

Split Train and Test data

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state=5, stratify=y)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(129980, 7)
(55707, 7)
(129980,)
(55707,)
```

7. Building machine learning model

1. KNearest Neighbour

```
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier()
knn_classifier.fit(X_train, y_train)
knn_predictions = knn_classifier.predict(X_test)
print(knn_classifier.score(X_test, y_test))
```

```
Accuracy: 0.9670777460642289
```

```
#Checking model prediction
X_= np.array([1255,3,11,6,7872,14,8079])
y_pred =knn_classifier.predict([X_])


X1 = np.array([807,0,13,10,2968,19,3045])
y_pred1 =knn_classifier.predict([X1])

print(y_pred)
print(y_pred1)
```

```
[3]
```

[1]

2. Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_train, y_train)
print(rf_classifier.score(X_test, y_test))
```

Accuracy: 0.9806128493726103

```
X_= np.array([807,0,13,10,2968,19,3045])
y_pred =rf_classifier.predict([X_])
y_pred
```

array([1])

Evaluate the Model

Classification Report

In [109]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred, target_names =data_features ))
```

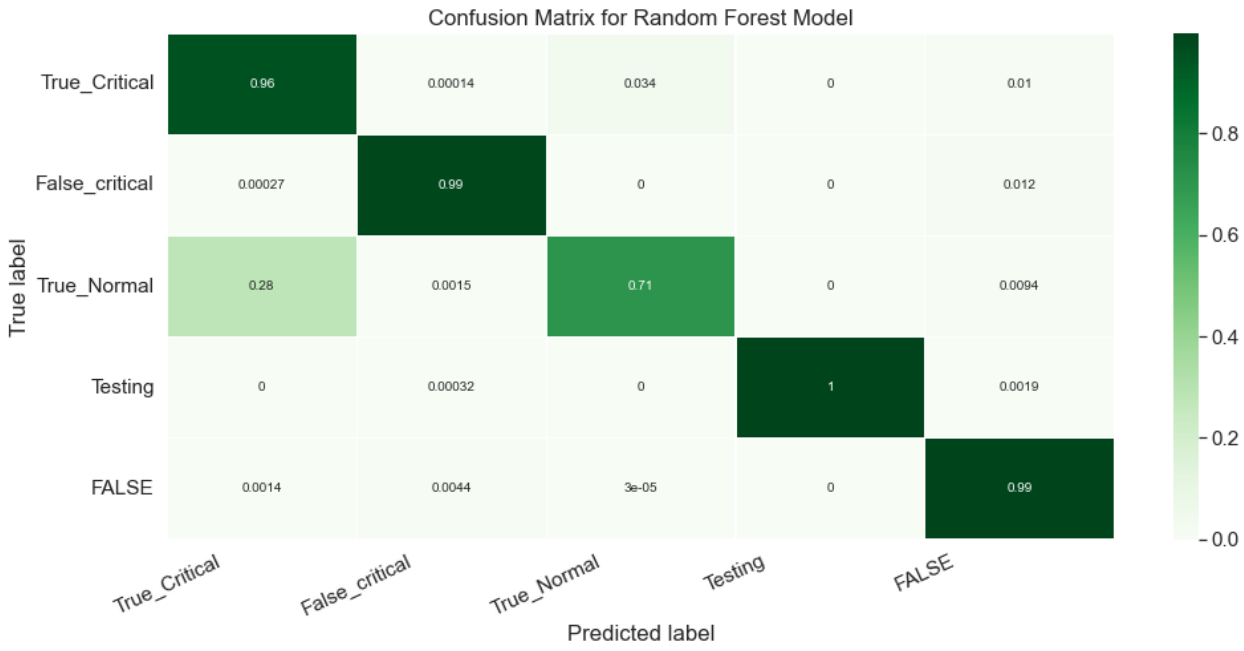
	precision	recall	f1-score	support
True_Critical	0.94	0.96	0.95	7127
False_False	0.99	0.99	0.99	10933
True_Normal	0.80	0.71	0.75	1377
Testing	1.00	1.00	1.00	3151
FALSE	0.99	0.99	0.99	33119
accuracy			0.98	55707
macro avg	0.94	0.93	0.94	55707
weighted avg	0.98	0.98	0.98	55707

Confusion Matrix

In [110]:

```
confusion_matrix(y_test, y_pred)
```

```
array([[ 6810,     1,   242,     0,    74],
       [    3, 10796,     0,     0,   134],
       [  385,     2,   977,     0,    13],
       [    0,     1,     0,  3144,     6],
       [   46,   146,     1,     0, 32926]], dtype=int64)
```



Random Forest with RandomizedSearchCV

In [112]:

```
rf = RandomForestClassifier(random_state= RSEED)
from pprint import pprint
# Look at parameters used by our current forest
#print('Parameters currently in use:\n')
pprint(rf.get_params())
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 50,
 'verbose': 0,
 'warm_start': False}
```

Set the parameter grid

In [113]:

```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 200, num =
10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']+list(np.arange(0.5, 1, 0.1))
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(3, 20, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
```

```
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [3, 4, 6, 8, 9, 11, 13, 14, 16, 18, 20, None],
 'max_features': ['auto',
                  'sqrt',
                  0.5,
                  0.6,
                  0.7,
                  0.7999999999999999,
                  0.8999999999999999],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [10, 31, 52, 73, 94, 115, 136, 157, 178, 200]}
```

Create the Random Search model

In [116]:

```
rf_random = RandomizedSearchCV(estimator = rf, param_distributions =
random_grid, n_iter = 10, cv = 3, verbose=2, random_state=RSEED,n_jobs = -1)
```

Fit the model

In [121]:

```
rf_random.fit(X_train, y_train)
```

```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(random_state=50),
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [3, 4, 6, 8, 9, 11, 13, 1
4,
                                                      16, 18, 20, None],
                                        'max_features': ['auto', 'sqrt', 0.5,
                                                         0.6, 0.7,
                                                         0.7999999999999999,
                                                         0.8999999999999999],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [10, 31, 52, 73, 94,
                                                         115, 136, 157, 178,
                                                         200]},
                   random_state=50, verbose=2)
```

Explore the best parameters

```
print(rf_random.best_params_)
```

{'n_estimators': 94, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 0.8999999999999999, 'max_depth': 18, 'bootstrap': True}

Use the Best Model

```
random_cv=rf_random.best_estimator_
random_cv
```

Out[123]:

**RandomForestClassifier(max_depth=18, max_features=0.8999999999999999,
n_estimators=94, random_state=50)**

In [124]:

```
y_pred1 = random_cv.predict(X_test)
y_pred1
```

Out[124]:

array([4, 4, 4, ..., 4, 4, 1])

In [125]:

```
print(random_cv.score(X_test,y_test))
```

Accuracy: 0.9828387814816809

Evaluate the Model

Classfication Reports

In [126]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred1, target_names =data_features ))
```

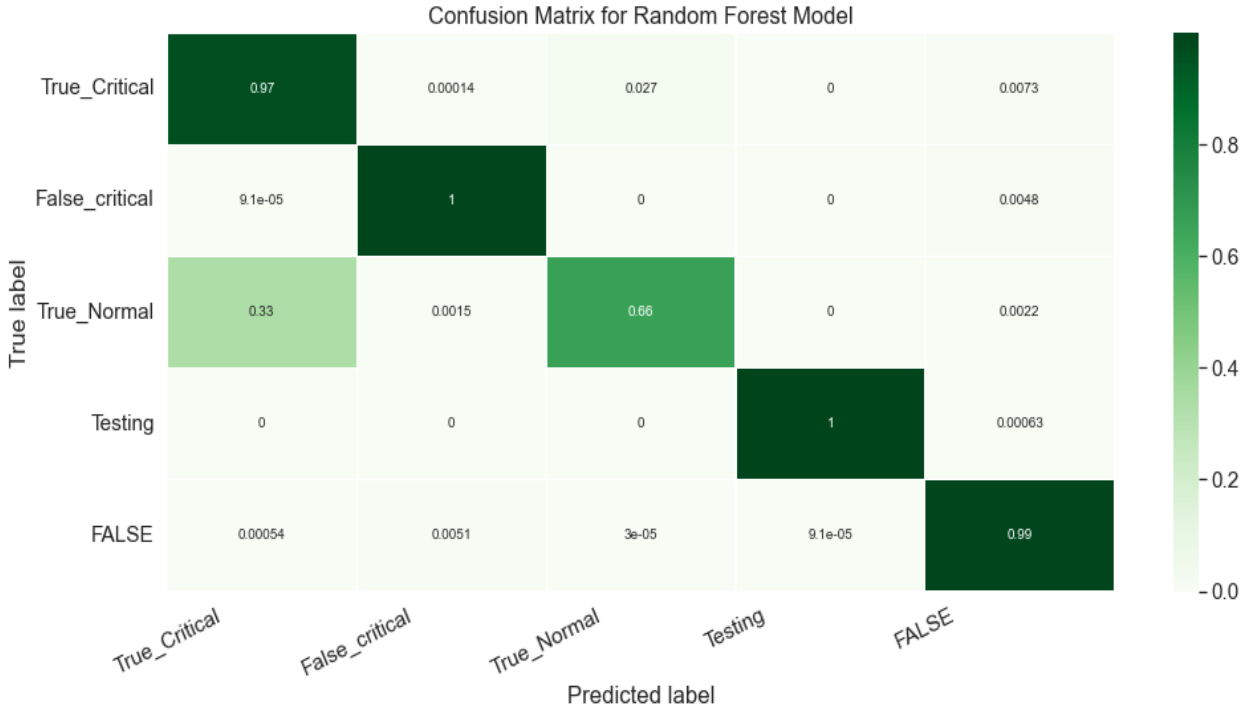
```
                precision    recall  f1-score   support

 True_Critical       0.93      0.97      0.95      7127
   False_False       0.98      1.00      0.99     10933
   True_Normal       0.82      0.66      0.73      1377
       Testing       1.00      1.00      1.00      3151
         FALSE       1.00      0.99      1.00     33119

      accuracy                           0.98     55707
     macro avg       0.95      0.92      0.93     55707
  weighted avg       0.98      0.98      0.98     55707
```

```
confusion_matrix(y_test, y_pred1)
```

```
array([[ 6881,     1,   193,     0,    52],
       [    1, 10880,     0,     0,    52],
       [  460,     2,   912,     0,     3],
       [    0,     0,     0,  3149,     2],
       [   18,   168,     1,     3, 32929]], dtype=int64)
```


Confusion Matrix for Random Forest Model
True_Critical
False_critical
True_Normal
Testing
FALSE
True label
0.97
0.00014
0.027
0
0.0073
9.1e-05
1
0
0
0.0048
0.33
0.0015
0.66
0
0.0022
0
0
0
1
0.00063
0.00054
0.0051
3e-05
9.1e-05
0.99
0.8
0.6
0.4
0.2
0.0
True_Critical
False_critical
True_Normal
Testing
FALSE
Predicted label

What is the issue with imbalanced dataset?

Most models trained on imbalanced data will have a bias towards predicting the larger class(es) and, in many cases, may ignore the smaller class(es) altogether.

As a result, the instances belonging to the smaller class(es) are typically misclassified more often than those belonging to the larger class(es)

How to deal with imbalance dataset?

Resample the training set

Oversampling — Duplicating samples from the minority class

Undersampling — Deleting samples from the majority class

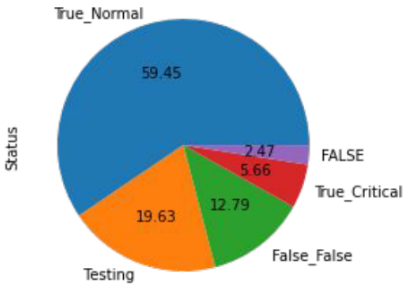
Combining Both Random Sampling Techniques

Combining both random sampling methods can occasionally result in overall improved performance in comparison to the methods being performed in isolation.

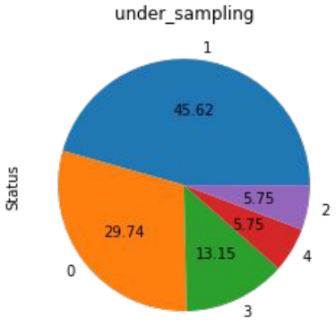
Under Sampling

Deleting samples from the majority class

Before Under Sampling



After under sampling



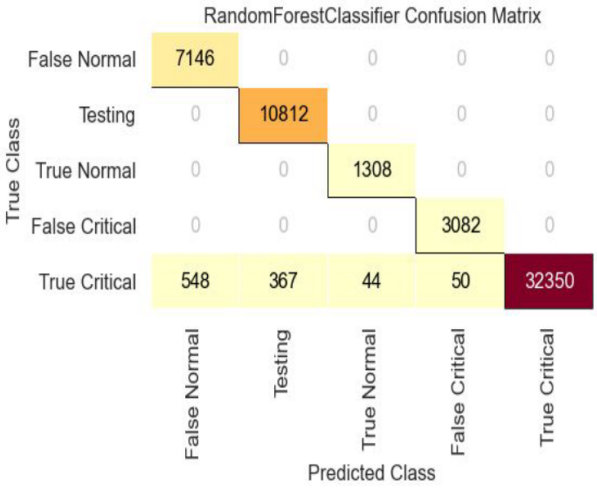
Before undersampling: Counter({4: 77036, 1: 25630, 0: 16609, 3: 7423, 2: 3282})

After undersampling: Counter({1: 36442, 0: 23755, 3: 10505, 2: 4590, 4: 4590})

RandomForestClassifier with under sampling

Test Accuracy-----0.9802538280646956

Train Accuracy ----1.0



It is overfitting

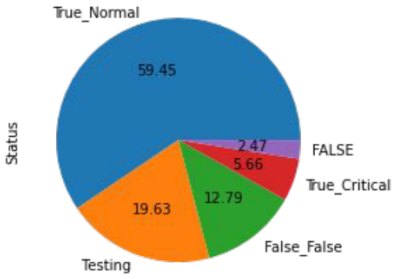
it can discard potentially useful information which could be important for building rule classifiers.

The sample chosen by random under sampling may be a **biased sample**. And it will not be an **accurate representative of the population**. Thereby, resulting in inaccurate results with the actual test data set.

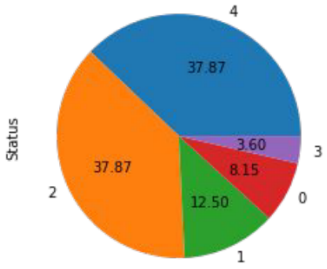
Oversampling

Duplicating samples from the minority class

Before Oversampling



After Oversampling

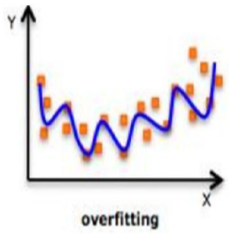


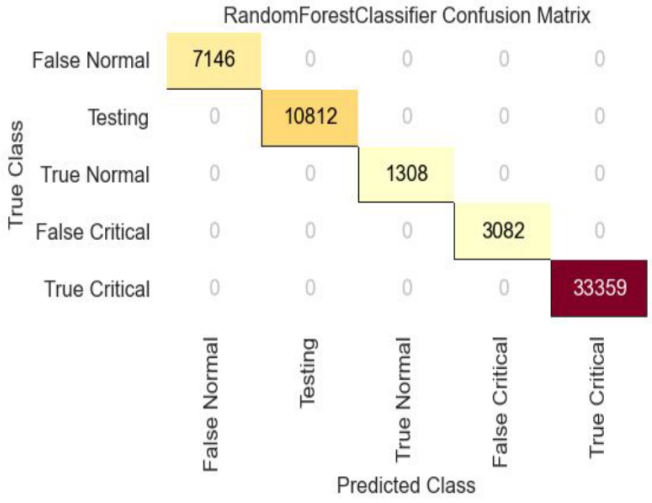
Random Forest Classifier with Oversampling

It increases the likelihood of overfitting since it replicates the minority class events.

Test Accuracy =1.0

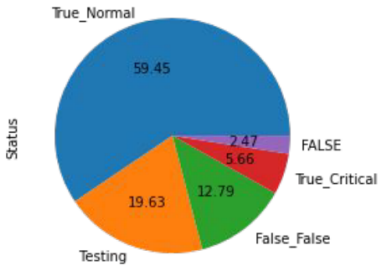
Train Accuracy=1.0



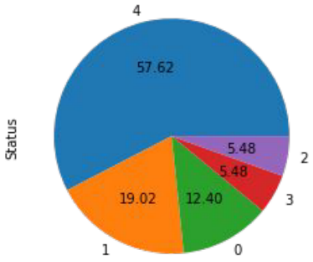


Combined Sampling

Before



After Combined Sampling



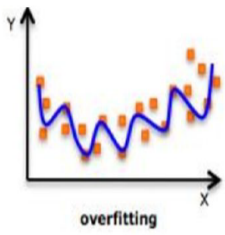
The concept is that we can apply a modest amount of oversampling to the minority class, which improves the bias to the minority class examples, whilst we also perform a modest amount of undersampling on the majority class to reduce the bias on the majority class examples.

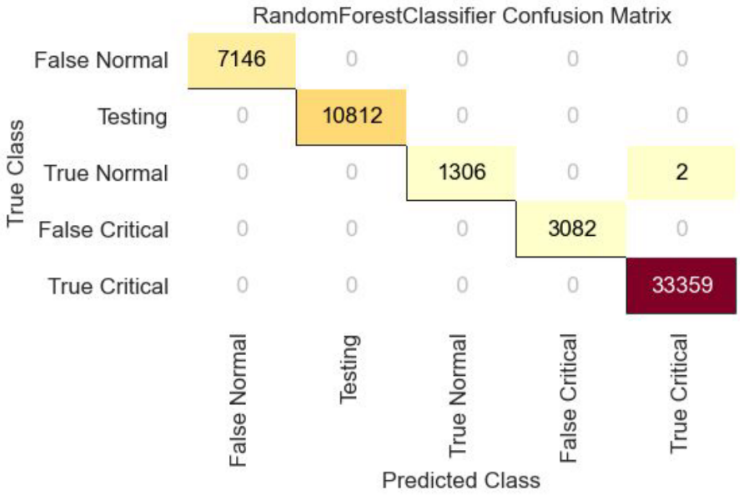
RandomForestClassifier with combined sampling

Test Accuracy=0.9999640978692086

Train Accuracy= 1.0

It is totally overfitting



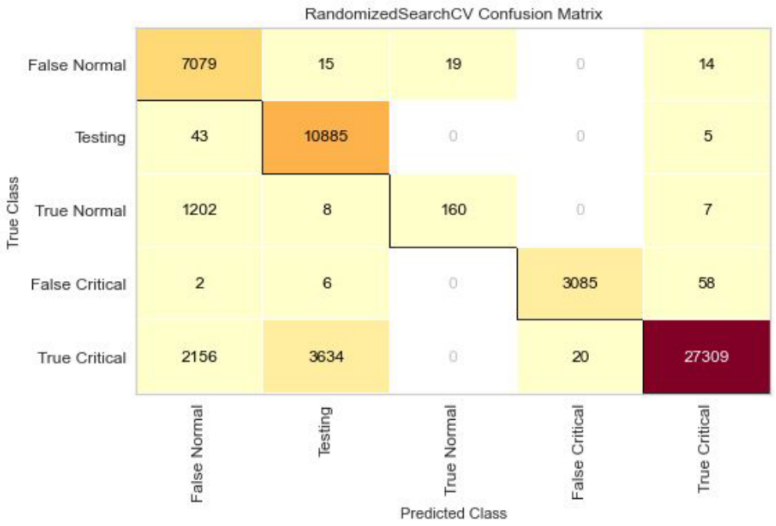


RandomForest Classifier with Undersampling

RandomizedSearchCV

Test Accuracy=0.8709497908700882
Train Accuracy=0.9321123657394658

Misclassification is high for
True Normal and True Critical



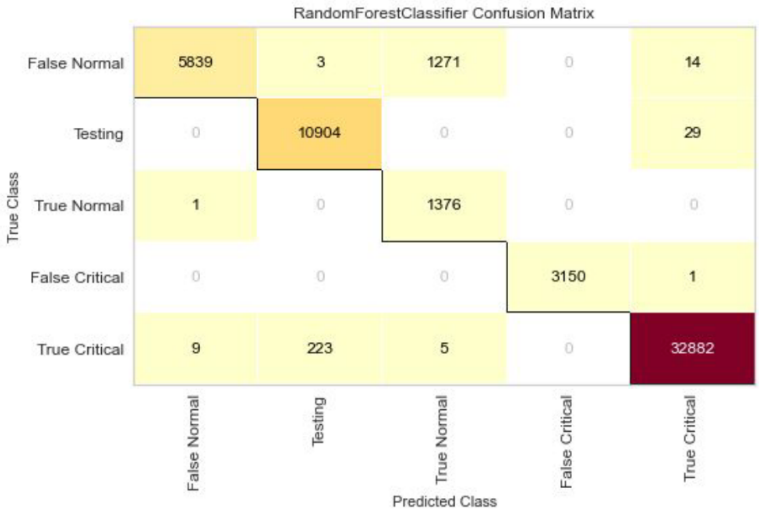
RandomForestClassifier with oversampling

RandomizedSearchCV

Test=0.9720681422442422
Train=0.9823082623193775

All classes classified perfectly except False Normal

Optimal



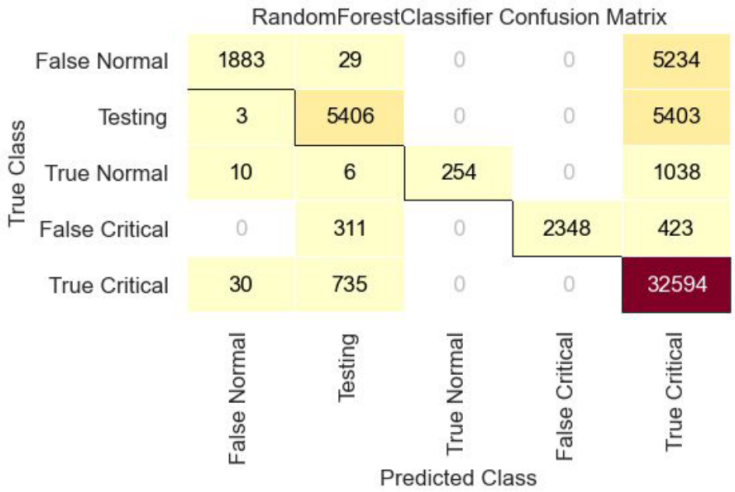
RandomForestClassifier with combined sampling

RandomizedSearchCV

Test-0.7626510133376416

Train-0.7823095792319495

Here, False Normal, True Normal and Testing are heavily misclassified



Summary

Balanced/Imbalanced data	Model	Sampling	Accuracy	Accuracy	Fitting
Imbalanced data	KNN	None	97.8	96.74	balanced
Imbalanced data	RandomForest	None	98.25	98.09	balanced
Imbalanced data	RandomForest with RadomizedSearchCV	None	98.34	98.28	balanced
Balanced data	RandomForest	Under Sampling	100	98	overfitting
Balanced data	RandomForest	OverSampling	100	100	overfitting
Balanced data	RandomForest	Combined Sampling	100	99.9	overfitting
Balanced data	RandomForest with RadomizedSearchCV	Under Sampling	93	87	balanced
Balanced data	RandomForest with RadomizedSearchCV	OverSampling	98	97	balanced
Balanced data	RandomForest with RadomizedSearchCV	Combined Sampling	78	76	balanced

8. Conclusion

In [155]:

```
print('\n                                    Accuracy     Error')
print('                                  ----------   --------')

print('KNN                                : {:.04}%       {:.04}%
'.format(knn_classifier.score(X_test, y_test) * 100,\
```

```
100-(knn_classifier.score(X_test, y_test) * 100)))

print('Random Forest                          : {:.04}%        {:.04}%
'.format(rf_classifier.score(X_test, y_test)* 100,\

100-(rf_classifier.score(X_test, y_test)* 100)))

print('Random Forest with RandomizedSearchCV : {:.04}%        {:.04}%
'.format(random_cv.score(X_test, y_test)* 100,\

100-(random_cv.score(X_test, y_test)* 100)))
```

Imbalanced data

	Accuracy	Error
KNN	: 96.74%	3.264%
Random Forest	: 98.06%	1.939%
Random Forest with RandomizedSearchCV	: 98.28%	1.716%

Balanced Data

Results

- RandomForest (Oversampling) with RandomizedSearchCV is the optimal model for this dataset.
- Most of the models are overfitting due to resampling the training data.
- Most importantly, Random Forest Classifier worked well even with imbalanced dataset.

9. References

1. https://medium.com/analytics-vidhya/evaluating-a-random-forest-model-9d165595ad56
2. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
3. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
4. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

5. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

Advanced Level Project:

Bank Theft attempt detection using image processing

Challenges in this project:

1. The main task is to differentiate between the customers and thieves.
2. We need to train thousands of images which has many objects like weapons, knives, hockey sticks, bombs,
3. Labelling the images.

This project can be deployed into ATMs and banks.

If anyone tries to break the doors, it will generate the alert and notified to the nearest police station or control rooms of respective banks.

Theft attempt in ATM

Most of the thieves are covering their faces using mask

Thief threatening the bank staff

Objects used in the crime