Introduction to Data Science

Project Report

Type B. Applying ML Classification algorithms on the data set and getting inferences from the data. You may use the appropriate ML algorithm and know the concept behind it.

Group Members:

Dev Bansal 20UCS244

Sahil Gupta 20UCS166

Prateek Sharma 20UCS146

Shubham Singh 20UCS191

Contents

Introduction

This is our group project for the course Introduction to Data Science. We have chosen a dataset that ….

Dataset Link:

https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+

The code is implemented using Google Colab notebook in Python language and can be accessed through the google drive link:

https://drive.google.com/drive/u/1/folders/1fBAh8LQEsn8nknDfB7Mb0hAN1xIIBTeW

Objective

Our purpose in this project is to apply suitable ML Classification algorithms to the dataset and obtain inferences from the data using Python. After pre-processing, we will conduct a preliminary analysis of the dataset and classify our data into appropriate categories with proper validation.

Our aim is to obtain inference of accurate occupancy detection of an office room from light, temperature, humidity and CO2 measurements using statistical learning models

System Requirements

The code can be run through Google Colab without installing anything on the local system.

To run code on the local system:

1. Python3 needs to be installed on the system.

2. Important Statistical libraries such as NumPy, pandas, scikit-learn, etc. need to be installed on the system.

Dataset Description

Following are details of our dataset:-

Data Set Characteristics:	Multivariate, Time-Series	**Number of Instances:**	20560	**Area:**	Computer
Attribute Characteristics:	Real	**Number of Attributes:**	7	**Date Donated**	2016-02-29
Associated Tasks:	Classification	**Missing Values?**	N/A	**Number of Web Hits:**	193749

Data Set Information:

Three data sets are submitted, for training and testing. Ground-truth occupancy was obtained from time stamped pictures that were taken every minute. For the journal publication, the processing R scripts can be found in:

https://github.com/LuisM78/Occupancy-detection-data

Attribute Information:

date time year-month-day hour:minute:second
Temperature, in Celsius
Relative Humidity, %
Light, in Lux
CO_2, in ppm
Humidity Ratio, Derived quantity from temperature and relative humidity, in kgwater-vapor/kg-air
Occupancy, 0 or 1, 0 for not occupied, 1 for occupied status

There are total of 2 classes i.e., 0 and 1 with 20560 instances where: -

Class 0 have 15810

Class 1 have 4750

Libraries used in our project

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split as TTS
from matplotlib import pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
```

Drive mount and storing of dataset in variable data

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

[ ] data = pd.read_csv('/content/drive/MyDrive/IDS Project/datatesting.csv')
```

Overview of Dataset

```
[ ] data.shape

    (20560, 7)
```

```
[ ] data
```

	date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
0	04-02-2015 17:51	23.180	27.2720	426.00	721.25	0.004793	1
1	04-02-2015 17:51	23.150	27.2675	429.50	714.00	0.004783	1
2	04-02-2015 17:53	23.150	27.2450	426.00	713.50	0.004779	1
3	04-02-2015 17:54	23.150	27.2000	426.00	708.25	0.004772	1
4	04-02-2015 17:55	23.100	27.2000	426.00	704.50	0.004757	1
...	...	...	...	...	...	...	...
20555	18-02-2015 09:15	20.815	27.7175	429.75	1505.25	0.004213	1
20556	18-02-2015 09:16	20.865	27.7450	423.50	1514.50	0.004230	1
20557	18-02-2015 09:16	20.890	27.7450	423.50	1521.50	0.004237	1
20558	18-02-2015 09:17	20.890	28.0225	418.75	1632.00	0.004279	1
20559	18-02-2015 09:19	21.000	28.1000	409.00	1864.00	0.004321	1

20560 rows × 7 columns

Count of class wise instances

```
data['Occupancy'].value_counts()

0    15810
1     4750
Name: Occupancy, dtype: int64
```

Checking count of Missing/Null values of each attribute

```
[ ]  data.isnull().sum()

     Temperature      0
     Humidity         0
     Light            0
     CO2              0
     HumidityRatio    0
     Occupancy        0
     dtype: int64
```

As there are no missing values, we need not replace the missing values with their respective appropriate values.

Dataset Preprocessing

We have dropped the date attribute as date attribute doesn't contribute to the processing and signifies only the date when the data was recorded.

```
[4]  data = data.drop(columns=['date'])
```

We have computed several measures of central tendencies. The following contingency table includes the results of this calculation:

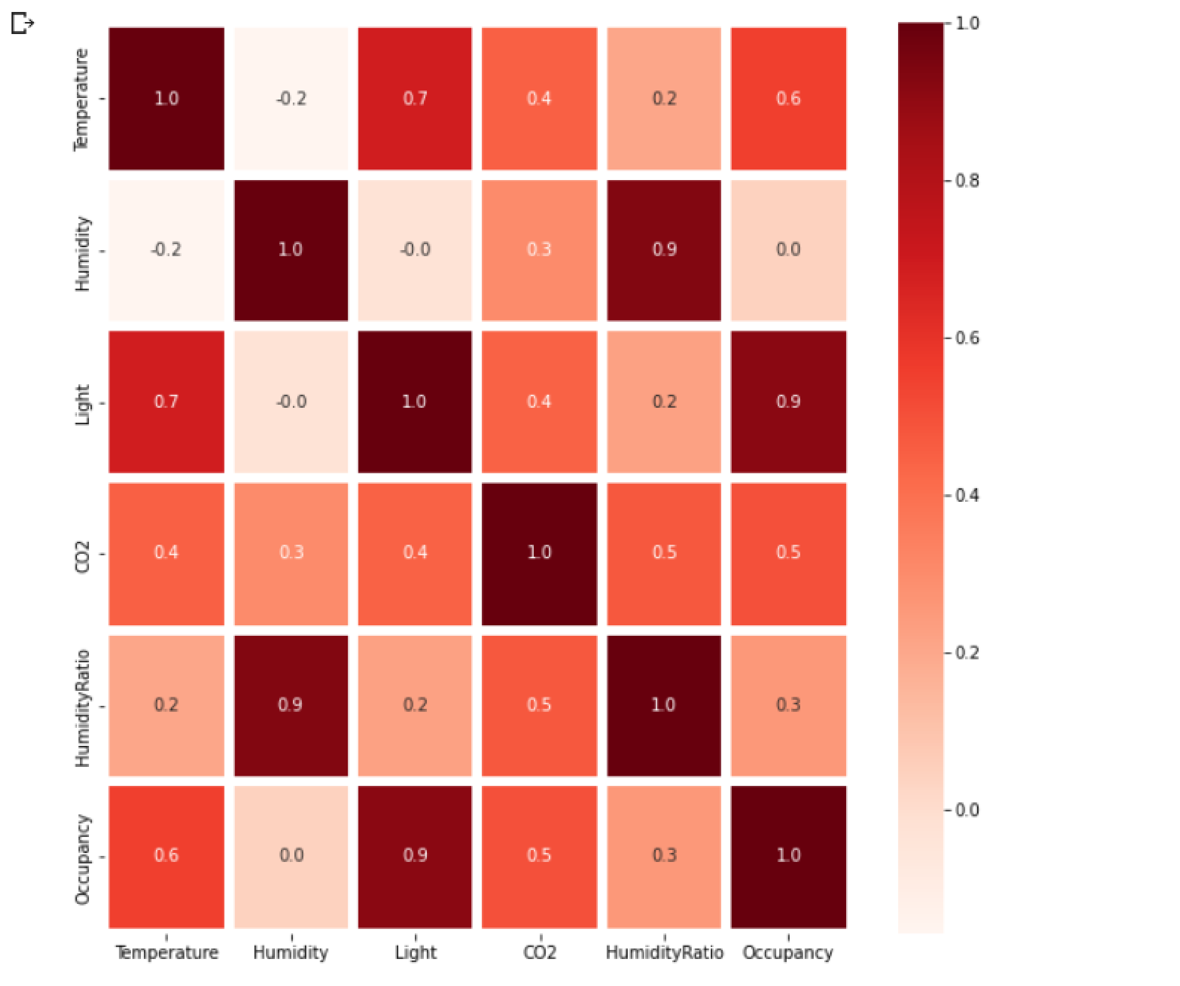
```
[ ]  data.describe()
```

	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
count	20560.000000	20560.000000	20560.000000	20560.000000	20560.000000	20560.000000
mean	20.906212	27.655925	130.756622	690.553276	0.004228	0.231031
std	1.055315	4.982154	210.430875	311.201281	0.000768	0.421503
min	19.000000	16.745000	0.000000	412.750000	0.002674	0.000000
25%	20.200000	24.500000	0.000000	460.000000	0.003719	0.000000
50%	20.700000	27.290000	0.000000	565.416667	0.004292	0.000000
75%	21.525000	31.290000	301.000000	804.666667	0.004832	0.000000
max	24.408333	39.500000	1697.250000	2076.500000	0.006476	1.000000

Preliminary & Analysis of Data

Data in the dataset is not sorted based on any attributes. But to carry out the best results, we must randomize it before splitting.

```
f,ax=plt.subplots(figsize=(10,10))
sns.heatmap(data.corr(),annot=True,linewidths=5,fmt='.1f',ax=ax, cmap='Reds')
plt.show()
```



By observing this Heatmap, we can infer that the features are not closely related to each other. For example, attribute **Light** dominates overall tendency measures over other attributes such as **Humidity** or **HumidityRatio**.

Hence, we can infer that there is a need for normalizing or standardizing the attribute values to be contained in a similar domain. Thus, we decided to normalize the data after splitting it into test and training sets.

Training Data vs Test Data

Here we used a simple holdout method. The data has been randomized before split as observed in the preliminary stage.

```
[ ] Train, Test = TTS(data, test_size = 0.2, random_state = 4)

[ ] Train.shape

    (16448, 6)

    Test.shape

    (4112, 6)
```

The data has been split into training and testing data as above. The training data now contains 16448 instances and testing data contains 4112 instances with all the features included.

Now, we would split both the datasets into two parts: features and outcome. For this we drop the attribute **Occupancy** from the datasets and name the new datasets as X_train and X_test and then use the dropped columns **Occupancy** to form another datasets Y_train and Y_test which contains the outcome feature.

```
X_train = Train.drop(columns=['Occupancy'])
```

```
[ ] Y_train = Train['Occupancy']
    X_test = Test.drop(columns=['Occupancy'])
    Y_test = Test['Occupancy']
```

The feature dataset is as below:

```
X_train
```

	Temperature	Humidity	Light	CO2	HumidityRatio
13318	21.200	25.200	19.0	519.00	0.003920
18734	19.890	30.500	0.0	722.00	0.004380
6215	19.500	27.290	0.0	465.00	0.003821
10046	20.945	25.890	0.0	596.50	0.003965
18596	20.200	30.390	0.0	711.00	0.004449
...	...	...	...	...	...
16840	20.390	32.790	0.0	657.00	0.004860
11863	20.575	21.995	24.0	836.25	0.003289
17093	20.200	29.890	0.0	727.50	0.004375
8366	22.390	24.912	418.6	782.80	0.004169
17530	20.390	24.890	0.0	804.50	0.003682

The Outcome dataset is as below:

```
Y_train
```

```
13318    0
18734    0
6215     0
10046    0
18596    0
        ..
16840    0
11863    0
17093    0
8366     1
17530    0
```

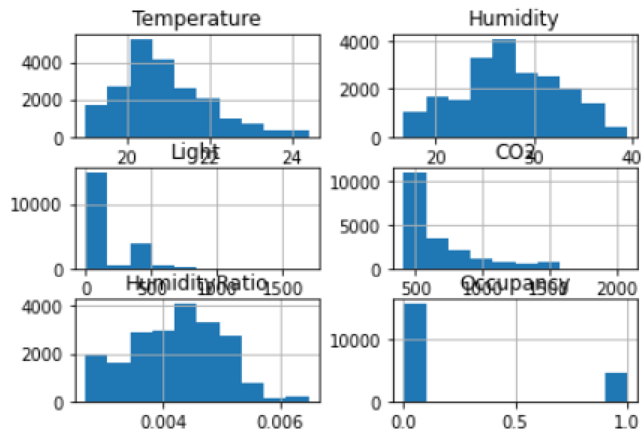
Normalization of Training & Test Data

There are two types of Scaling: **Standard Scaling** and **MinMax Scaling**.

Scaling is done as some models of machine learning are based on distance between the points(features). So, a feature having more distance than the other feature may be dominant in the model but practically it may not be so.

```
data.hist()
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad7d4ea00>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad7ca8e50>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad7c672b0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad7c926a0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad7c3ea90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f8ad7bf0dc0>]],
      dtype=object)
```

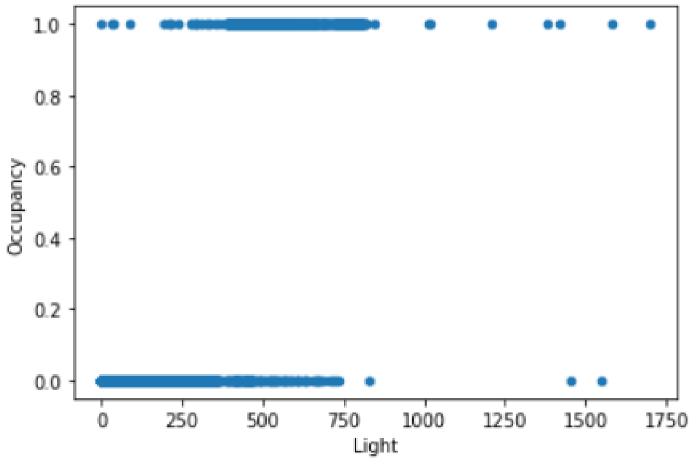


As we see that the features are not on same scale, we need scaling.

We preferred MinMax Scaling as the curves are not following Gaussian Distribution which is preferred for Standard Scaling. Also, it doesn't change the shape of the data. Also, our dominant feature is Light that has some outliers and Standard Scaler doesn't handle outliers well.

```python
data.plot.scatter(y='Occupancy',x='Light')
```

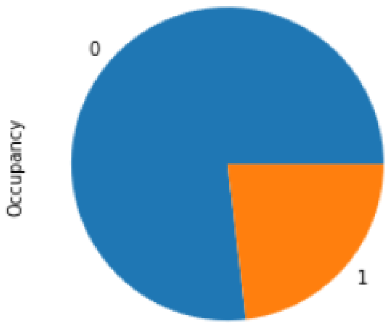
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6085a95b20>
```



<u>Re-Analysis of Data After Partitioning & Normalizing</u>

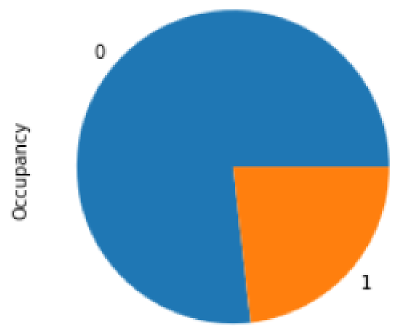
```python
data['Occupancy'].value_counts().plot(kind='pie')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ad73dd4f0>
```



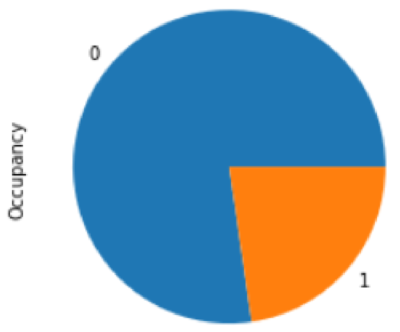
```
Y_train.value_counts().plot(kind='pie')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ad739d2e0>
```




```
Y_test.value_counts().plot(kind='pie')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8ad7ab9490>
```

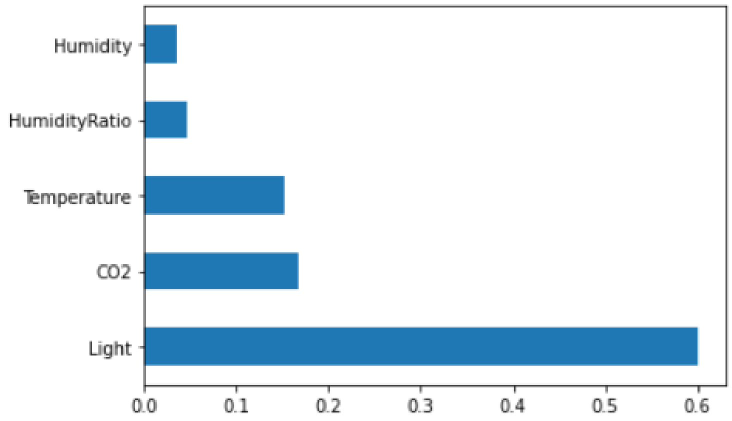


We can see that test-class distribution is roughly equivalent in all three datasets (Original, test, and training). This means that accuracy is a good way of measuring the classifiers (due to the absence of bias).

Next, we checked if some attributes have a low influence on classifications using ExtraTreesClassifier from the sklearn library in order to drop them which are least important.

```python
from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier()
model.fit(X_train,Y_train)
print(model.feature_importances_)
feat_importances = pd.Series(model.feature_importances_,index=X_train.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```

```
[0.15162621 0.03508592 0.60028798 0.16648977 0.04651013]
```



From this figure, we see that Humidity and HumidityRatio are not so significant features and can be dropped so that the computation is done faster and unnecessary computation is not done.

```python
X_train = X_train.drop(columns=['Humidity','HumidityRatio'])
X_test = X_test.drop(columns=['Humidity','HumidityRatio'])
```

Classification & Choosing Appropriate Classifier

We have used Logistic Regression, SVM, Naïve Bayes, Random Forest, KNN, Boosting, Bagging classifier for our dataset.

Following is the output table for Accuracy, Precision, and Recall of all the classifiers mentioned above.

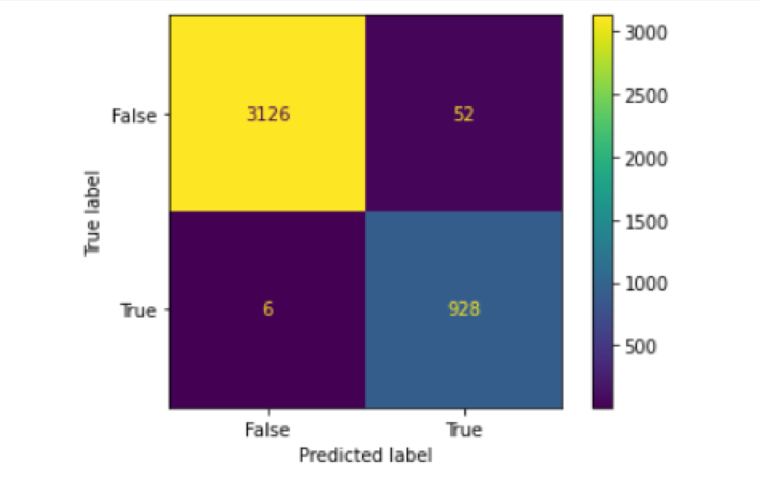
	Logistic Regression	SVM	Naïve Bayes	Random Forest	KNN	Boosting	Bagging
Accuracy	0.9858	0.9858	0.9627	0.9861	0.9868	0.9863	0.9854
Precision	0.9864	0.9862	0.9678	0.9866	0.9870	0.9868	0.9859
Recall	0.9858	0.9858	0.9627	0.9861	0.9868	0.9863	0.9854

Code of Logistic Regression Classifier

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
model = LogisticRegression()
model.fit(X_train,Y_train)
x_test_pred = model.predict(X_test)
test_acc = accuracy_score(Y_test,x_test_pred)
test_pre = precision_score(Y_test,x_test_pred,average='weighted')
test_rec = recall_score(Y_test,x_test_pred,average='weighted')
print("Accuracy: {:f}".format(test_acc))
print("Precision: {:f}".format(test_pre))
print("Recall: {:f}".format(test_rec))
```

```
Accuracy: 0.985895
Precision: 0.986467
Recall: 0.985895
```

Confusion Matrix:

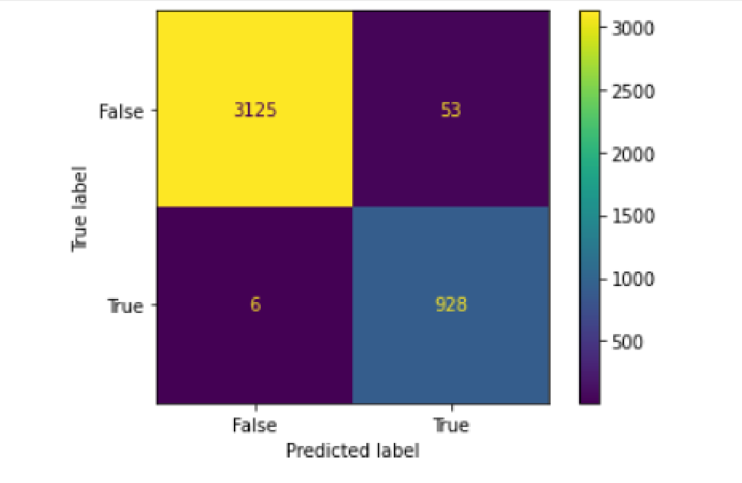


Code of SVM Classifier

```
[ ]  from sklearn import svm
     clf = svm.SVC(kernel='linear')
     clf.fit(X_train,Y_train)
     Y_pred = clf.predict(X_test)
     test_acc = accuracy_score(Y_test,Y_pred)
     test_pre = precision_score(Y_test,Y_pred,average='weighted')
     test_rec = recall_score(Y_test,Y_pred,average='weighted')
     print("Accuracy: {:f}".format(test_acc))
     print("Precision: {:f}".format(test_pre))
     print("Recall: {:f}".format(test_rec))
```

```
Accuracy: 0.985652
Precision: 0.986247
Recall: 0.985652
```

Confusion Matrix:

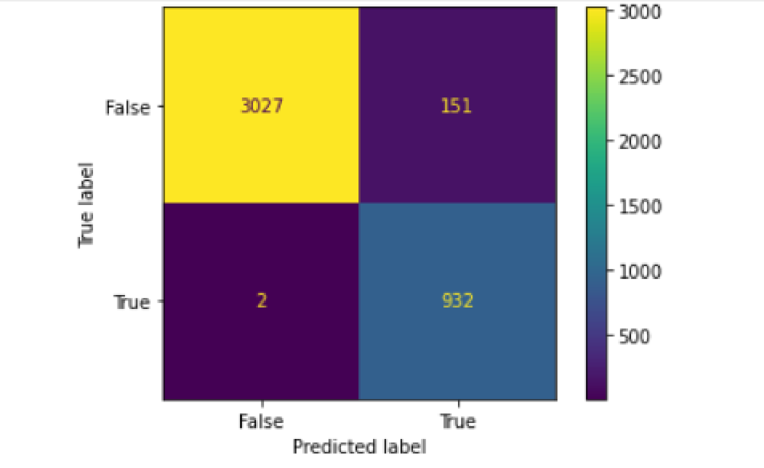


Code of Naïve Bayes Classifier

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, Y_train)
Y_pred_nb = gnb.predict(X_test)
test_acc = accuracy_score(Y_test,Y_pred_nb)
test_pre = precision_score(Y_test,Y_pred_nb,average='weighted')
test_rec = recall_score(Y_test,Y_pred_nb,average='weighted')
print("Accuracy: {:f}".format(test_acc))
print("Precision: {:f}".format(test_pre))
print("Recall: {:f}".format(test_rec))
```

```
Accuracy: 0.962792
Precision: 0.967820
Recall: 0.962792
```

Confusion Matrix:

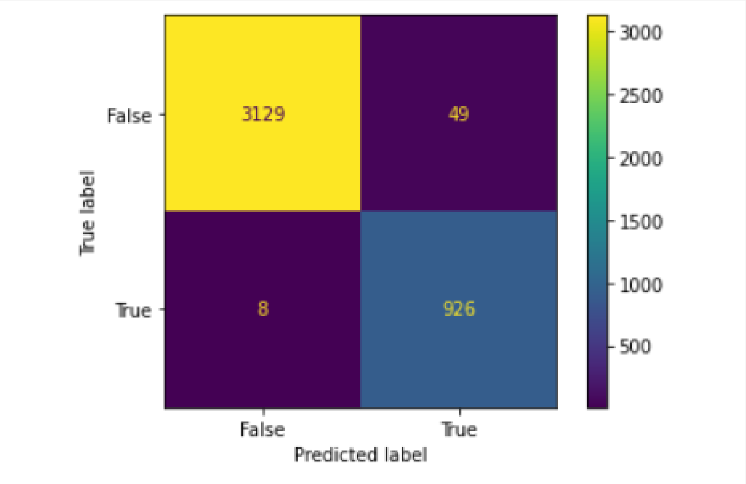


Code of Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
clf_r = RandomForestClassifier(max_depth=4, random_state=0)
clf_r.fit(X_train,Y_train)
Y_pred_rf = clf_r.predict(X_test)
test_acc = accuracy_score(Y_test,Y_pred_rf)
test_pre = precision_score(Y_test,Y_pred_rf,average='weighted')
test_rec = recall_score(Y_test,Y_pred_rf,average='weighted')
print("Accuracy: {:f}".format(test_acc))
print("Precision: {:f}".format(test_pre))
print("Recall: {:f}".format(test_rec))
```

```
Accuracy: 0.986138
Precision: 0.986614
Recall: 0.986138
```

Confusion Matrix:

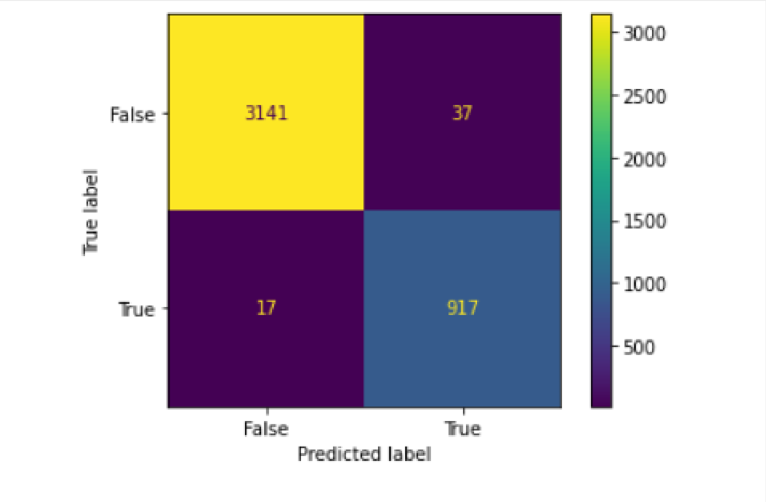


Code of KNN Classifier

```
[ ] from sklearn.neighbors import KNeighborsClassifier
    model = KNeighborsClassifier(n_neighbors=7)
    model.fit(X_train,Y_train)
    Y_pred_KNN = model.predict(X_test)
    test_acc = accuracy_score(Y_test,Y_pred_KNN)
    test_pre = precision_score(Y_test,Y_pred_KNN,average='weighted')
    test_rec = recall_score(Y_test,Y_pred_KNN,average='weighted')
    print("Accuracy: {:f}".format(test_acc))
    print("Precision: {:f}".format(test_pre))
    print("Recall: {:f}".format(test_rec))

    Accuracy: 0.986868
    Precision: 0.987030
    Recall: 0.986868
```

Confusion Matrix:

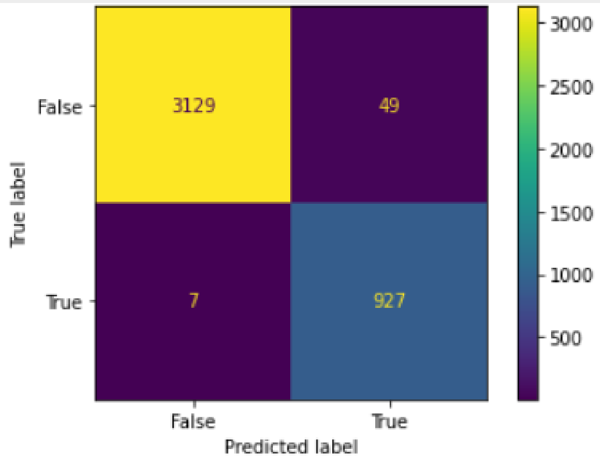


Code of Boosting Classifier

```
from xgboost.sklearn import XGBClassifier
model = XGBClassifier()
model.fit(X_train,Y_train)
Y_pred_boost = model.predict(X_test)
test_acc = accuracy_score(Y_test,Y_pred_boost)
test_pre = precision_score(Y_test,Y_pred_boost,average='weighted')
test_rec = recall_score(Y_test,Y_pred_boost,average='weighted')
print("Accuracy: {:f}".format(test_acc))
print("Precision: {:f}".format(test_pre))
print("Recall: {:f}".format(test_rec))
```

```
Accuracy: 0.986381
Precision: 0.986871
Recall: 0.986381
```

Confusion Matrix:

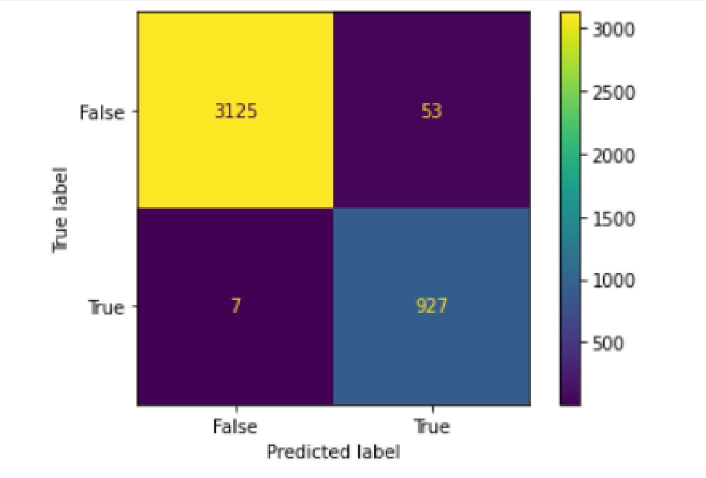


Code of Bagging Classifier

```
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier
model = BaggingClassifier(base_estimator=SVC(),n_estimators=10,random_state=0)
model.fit(X_train,Y_train)
Y_pred_bag = model.predict(X_test)
test_acc = accuracy_score(Y_test,Y_pred_bag)
test_pre = precision_score(Y_test,Y_pred_bag,average='weighted')
test_rec = recall_score(Y_test,Y_pred_bag,average='weighted')
print("Accuracy: {:f}".format(test_acc))
print("Precision: {:f}".format(test_pre))
print("Recall: {:f}".format(test_rec))
```

```
Accuracy: 0.985409
Precision: 0.985989
Recall: 0.985409
```

Confusion Matrix:



Conclusion

As we have seen the accuracies of the above classifiers, KNN classifier showed the highest accuracy (0.9868), highest precision (0.9870) & highest recall value (0.9868) it may be the best classifier in this case.

References

1. https://scikit-learn.org/stable/modules/svm.html
2. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
3. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
4. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
5. https://scikit-learn.org/stable/modules/naive_bayes.html
6. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
7. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html
8. https://matplotlib.org/
9. https://stackoverflow.com/questions/35277075/python-pandas-counting-the-occurrences-of-a-specific-value