OCCUPANCY DETECTION

Using Supervised Classification Algorithms

Naimish Sharma

18151

Department of Electrical Engineering and Computer Science

Project Guide – Dr Kundan Kandhway

ECS 330 LAB PROJECT

2020/2021 SEMESTER II

Understanding the problem

The aim of this project is to detect human presence in the room(i.e. Occupancy of the room) using Occupancy Detection datasets which is generated using IoT devices installed in the room .We are asked to classify occupancy status in an room i.e. 0 or 1 , 0 for not occupied and 1 for occupied status . There are three datasets . One for training and two for testing . This method of Occupancy detection can be of great use where because of privacy reasons we can't use cameras to check if there is somebody in the room or not . It is also useful in source control like lights and fans are open if room is occupied etc .

Data Understanding

Dataset Information –

Three datasets are available –

- Training Dataset – "datatraining.txt" containing 8143 samples
- Testing Dataset 1 – "datatest.txt" containing 2665 samples
- Testing Dataset 2 – "datatest2.txt" containing 9752 samples

Data set	Number of observations	Data Class Distribution (%)		Comment
		0 (non-occupied)	1 (occupied)	
Training	8143 of 7 variables	0.79	0.21	Measurements taken mostly with the door closed during occupied status
Testing 1	2665 of 7 variables	0.64	0.36	Measurements taken mostly with the door closed during occupied status
Testing 2	9752 of 7 variables	0.79	0.21	Measurements taken mostly with the door open during occupied status

Dataset is available on -

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

Attributes Information –

- Date time year-month-day hour:minute:second
- Light in Lux
- CO2 in ppm
- Temperature in Celsius
- Relative Humidity in %
- Humidity Ratio – No Unit . Derived quantity from temperature and relative humidity
- Occupancy , 0 or 1 , 0 for not occupied and 1 for occupied status .

Data Preparation

This project is implemented with Python Programming Language using various scikit learn machine learning libraries .

Importing Libraries –

```
[83] import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.preprocessing import StandardScaler
```

Importing dataset –

Importing dataset from local machine and reading dataset as Pandas dataframe df1 for "datatraining.txt" , df2 for "datatest.txt" and df3 for "datatest2.txt" and saving dataframes as .csv files .

```
[6]  from google.colab import files
     uploaded = files.upload()
```

Removed 'date' column from all 3 datasets

```
[7]  df1 = pd.read_csv("datatraining.txt")
     df1 = df1.drop(['date'], axis = 1)                            # df1 is for training
     df1.to_csv('datatraining.csv', index = None )                 # storing this dataframe in a csv file

     df2 = pd.read_csv("datatest.txt")
     df2 = df2.drop(['date'], axis = 1)                            # df2 is for testing
     df2.to_csv('datatest.csv',  index = None)                     # storing this dataframe in a csv file

     df3 = pd.read_csv("datatest2.txt")
     df3 = df3.drop(['date'], axis = 1)                            # df3 is for testing
     df3.to_csv('datatest2.csv',  index = None)                   # storing this dataframe in a csv file
```

Note – Column "date" is removed from all three datasets because I want to make model in which Occupancy is predicted independent of date and time .

```
print(df1.shape)
print(df2.shape)
print(df3.shape)
```

```
(8143, 6)
(2665, 6)
(9752, 6)
```

```python
df2.head()
```

	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
140	23.7000	26.272	585.200000	749.200000	0.004764	1
141	23.7180	26.290	578.400000	760.400000	0.004773	1
142	23.7300	26.230	572.666667	769.666667	0.004765	1
143	23.7225	26.125	493.750000	774.750000	0.004744	1
144	23.7540	26.200	488.600000	779.000000	0.004767	1

Now as index of dataframe df2 is starting from 140 so starting the dataframe index from 1

```python
#Changing index of df2
df2.index = np.arange(1,len(df2)+1)
df2.head()
```

	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
1	23.7000	26.272	585.200000	749.200000	0.004764	1
2	23.7180	26.290	578.400000	760.400000	0.004773	1
3	23.7300	26.230	572.666667	769.666667	0.004765	1
4	23.7225	26.125	493.750000	774.750000	0.004744	1
5	23.7540	26.200	488.600000	779.000000	0.004767	1

Checking for missing data –

```python
[43] df1.isna().any().sum()
```

```
0
```

```python
[44] df2.isna().any().sum()
```

```
0
```

```python
[45] df3.isna().any().sum()
```

```
0
```

There are no missing values in all three dataframes(i.e. in all 3 datasets)

Now using “.describe()” with all three dataframes to calculate statistical data about the given datasets .

```
df1.describe()
```

	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
count	8143.000000	8143.000000	8143.000000	8143.000000	8143.000000	8143.000000
mean	20.619084	25.731507	119.519375	606.546243	0.003863	0.212330
std	1.016916	5.531211	194.755805	314.320877	0.000852	0.408982
min	19.000000	16.745000	0.000000	412.750000	0.002674	0.000000
25%	19.700000	20.200000	0.000000	439.000000	0.003078	0.000000
50%	20.390000	26.222500	0.000000	453.500000	0.003801	0.000000
75%	21.390000	30.533333	256.375000	638.833333	0.004352	0.000000
max	23.180000	39.117500	1546.333333	2028.500000	0.006476	1.000000

```
[22] df2.describe()
```

	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
count	2665.000000	2665.000000	2665.000000	2665.000000	2665.000000	2665.000000
mean	21.433876	25.353937	193.227556	717.906470	0.004027	0.364728
std	1.028024	2.436842	250.210906	292.681718	0.000611	0.481444
min	20.200000	22.100000	0.000000	427.500000	0.003303	0.000000
25%	20.650000	23.260000	0.000000	466.000000	0.003529	0.000000
50%	20.890000	25.000000	0.000000	580.500000	0.003815	0.000000
75%	22.356667	26.856667	442.500000	956.333333	0.004532	1.000000
max	24.408333	31.472500	1697.250000	1402.250000	0.005378	1.000000

```
[23] df3.describe()
```

	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
count	9752.000000	9752.000000	9752.000000	9752.000000	9752.000000	9752.000000
mean	21.001768	29.891910	123.067930	753.224832	0.004589	0.210111
std	1.020693	3.952844	208.221275	297.096114	0.000531	0.407408
min	19.500000	21.865000	0.000000	484.666667	0.003275	0.000000
25%	20.290000	26.642083	0.000000	542.312500	0.004196	0.000000
50%	20.790000	30.200000	0.000000	639.000000	0.004593	0.000000
75%	21.533333	32.700000	208.250000	831.125000	0.004998	0.000000
max	24.390000	39.500000	1581.000000	2076.500000	0.005769	1.000000

So , it is clearly visible that is all 3 datasets more than 50% of values under column "Light" are zero . We have to see why these values are zero otherwise these values can effect our model predictions .

```
# There are lots of zero values in column "Light" in all 3 datasets
print(sum(df1.Light  ==  0))
print(sum(df2.Light  ==  0))
print(sum(df3.Light  ==  0))
```

```
5160
1615
5997
```

There are 5160 datapoints in df1 , 1615 datapoints in df2 and 5997 datapoints in df3 in which Light value is zero .

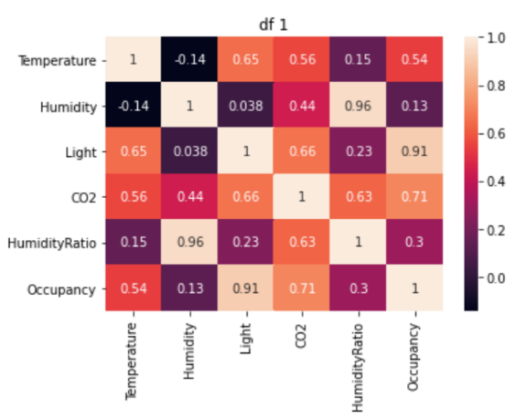
As Occupancy is either 0 or 1 so I added value of Occupancy for datapoints in each of dataframe for which Light value is zero .

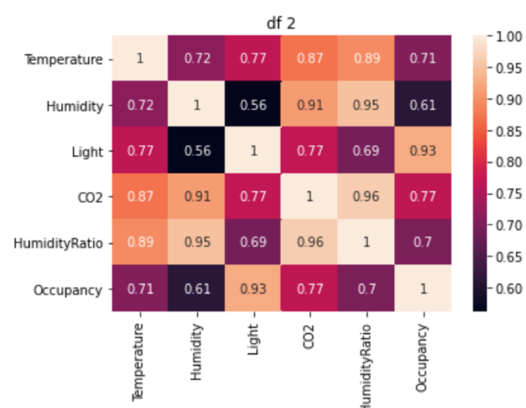
```
[27] print(sum(df1[df1.Light  ==  0].Occupancy))
     print(sum(df2[df2.Light  ==  0].Occupancy))
     print(sum(df3[df3.Light  ==  0].Occupancy))

     0
     0
     1
```

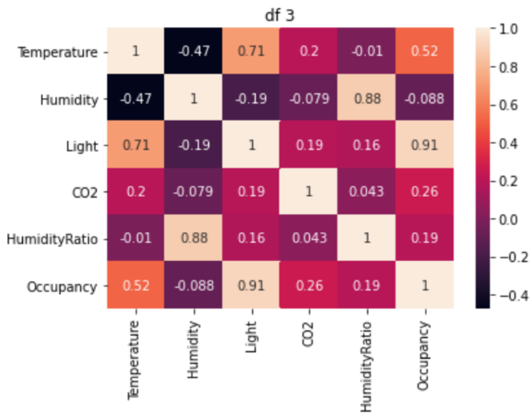
It comes out that for almost all the datapoints for which Light value is zero in each of the dataframe Occupancy is also zero . So , **It means that whenever Light is Off there is no one in room i.e. Occupancy is zero .**

It is also justified by following heatmap that Light and Occupancy are highly correlated .

```
corrMatrix1 = df1.corr()
corrMatrix2 = df2.corr()
corrMatrix3 = df3.corr()
sns.heatmap(corrMatrix1, annot=True)
plt.title("df 1")
plt.show()
sns.heatmap(corrMatrix2, annot=True)
plt.title("df 2")
plt.show()
sns.heatmap(corrMatrix3, annot=True)
plt.title("df 3")
plt.show()
```







Now extracting source and target domains for modelling .

X_train includes all columns except Occupancy of train dataframe df1 . X_test1 includes all columns except Occupancy of test dataframe df2 .

X_test2 includes all columns except Occupancy of test2 dataFrame df2 . And Y_train , Y_test1 , Y_test2 contains target Series(Occupancy Column) of dataframe df1 , df2 , df3 respectively . And converted the X_train , X_test1 and X_test2 into StandardScalar Form as X_scaled_train , X_scaled_test1 and X_scaled_test2 respectively .

```
#For training dataset "training.txt"
X_train = df1.iloc[:,0:5]
Y_train = df1.iloc[:,5]
scalar = StandardScaler()
X_scaled_train=scalar.fit(X_train).transform(X_train)
```

```
[110] #For testing dataset 1 "datatest.txt"
X_test1 = df2.iloc[:,0:5]
Y_test1 = df2.iloc[:,5]
scalar = StandardScaler()
X_scaled_test1=scalar.fit(X_test1).transform(X_test1)

#For testing dataset 2 "datatest2.txt"
X_test2 = df3.iloc[:,0:5]
Y_test2 = df3.iloc[:,5]
scalar = StandardScaler()
X_scaled_test2=scalar.fit(X_test2).transform(X_test2)
```

Printing shape of X_scaled_train , X_scaled_test1 and X_scaled_test2 respectively .

```
[113] print(X_scaled_train.shape)
print(X_scaled_test1.shape)
print(X_scaled_test2.shape)

(8143, 5)
(2665, 5)
(9752, 5)
```

Data Modelling

1) Logistic Regression –

It measures the relationship between the categorical dependent variable and one or more independent variables by estimating the probability of occurrence of an event using its logistics function.

Used different solvers available for Logistic Regression to check which gives the best accuracy so that we can choose among them the best solver for optimization of model .

```
[135] # importing Logistic regression model
      from sklearn.linear_model import LogisticRegression
      from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
      # we can also add penalty
      solv = {"newton-cg", "lbfgs", "liblinear", "sag", "saga"} # It represents which algorithm to use in the optimization problem.
      for a in solv:
          LR = LogisticRegression(random_state=0, solver = a)
          print(LR)
          LR.fit(X_scaled_train, Y_train)
          Y_pred1 = LR.predict(X_scaled_test1)
          Y_pred2 = LR.predict(X_scaled_test2)
          print('Coefficients: \n', LR.coef_)

          # accuracy
          print('Accuracy on Train : ', round(LR.score(X_scaled_train, Y_train)*100, 2))
          print('Accuracy on Test1 : ', round(LR.score(X_scaled_test1, Y_test1)*100, 2))
          print('Accuracy on Test2 : ', round(LR.score(X_scaled_test2, Y_test2)*100, 2))
```

Also calculated the confusion matrix , classification report , accuracy score on each of the testing datasets .

```
# Confusion Matrix
result = confusion_matrix(Y_test1, Y_pred1)
print("Confusion Matrix:")
print(result)
# Classification report
result1 = classification_report(Y_test1, Y_pred1)
print("\nClassification Report:")
print(result1)
# Accuracy score
result2 = accuracy_score(Y_test1, Y_pred1)
print("\nAccuracy:",result2)
print("For Test2 dataset")
# Confusion Matrix
result = confusion_matrix(Y_test2, Y_pred2)
print("Confusion Matrix:")
print(result)
# Classification report
result1 = classification_report(Y_test2, Y_pred2)
print("\nClassification Report:")
print (result1)
# Accuracy score
result2 = accuracy_score(Y_test2, Y_pred2)
print("\nAccuracy:",result2)
print('')
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=0, solver='saga', tol=0.0001, verbose=0,
                   warm_start=False)
Coefficients:
 [[-1.25609781  0.18181663  3.91327515  1.89551697 -0.38408122]]
Accuracy on Train :  98.6
Accuracy on Test1 :  89.04
Accuracy on Test2 :  95.63
For Test1 dataset
Confusion Matrix:
[[1671   22]
 [ 270  702]]

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.99      0.92      1693
           1       0.97      0.72      0.83       972

    accuracy                           0.89      2665
   macro avg       0.92      0.85      0.87      2665
weighted avg       0.90      0.89      0.89      2665


Accuracy: 0.8904315196998124
For Test2 dataset
Confusion Matrix:
[[7651   52]
 [ 374 1675]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      7703
           1       0.97      0.82      0.89      2049

    accuracy                           0.96      9752
   macro avg       0.96      0.91      0.93      9752
weighted avg       0.96      0.96      0.95      9752


Accuracy: 0.9563166529942576
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=0, solver='sag', tol=0.0001, verbose=0,
                   warm_start=False)
Coefficients:
 [[-1.25448032  0.18764232  3.91326452  1.89570778 -0.389882  ]]
Accuracy on Train :  98.6
Accuracy on Test1 :  89.04
Accuracy on Test2 :  95.61
For Test1 dataset
Confusion Matrix:
[[1671   22]
 [ 270  702]]

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.99      0.92      1693
           1       0.97      0.72      0.83       972

    accuracy                           0.89      2665
   macro avg       0.92      0.85      0.87      2665
weighted avg       0.90      0.89      0.89      2665


Accuracy: 0.8904315196998124
For Test2 dataset
Confusion Matrix:
[[7651   52]
 [ 376 1673]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      7703
           1       0.97      0.82      0.89      2049

    accuracy                           0.96      9752
   macro avg       0.96      0.90      0.93      9752
weighted avg       0.96      0.96      0.95      9752


Accuracy: 0.9561115668580804
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
Coefficients:
 [[-1.25311124  0.19245655  3.91322812  1.89585145 -0.39468199]]
Accuracy on Train :  98.6
Accuracy on Test1 :  89.04
Accuracy on Test2 :  95.61
For Test1 dataset
Confusion Matrix:
[[1671   22]
 [ 270  702]]

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.99      0.92      1693
           1       0.97      0.72      0.83       972

    accuracy                           0.89      2665
   macro avg       0.92      0.85      0.87      2665
weighted avg       0.90      0.89      0.89      2665


Accuracy: 0.8904315196998124
For Test2 dataset
Confusion Matrix:
[[7651   52]
 [ 376 1673]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      7703
           1       0.97      0.82      0.89      2049

    accuracy                           0.96      9752
   macro avg       0.96      0.90      0.93      9752
weighted avg       0.96      0.96      0.95      9752


Accuracy: 0.9561115668580804
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=0, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
Coefficients:
 [[-1.24832926  0.17094084  3.83620881  1.88247633 -0.38236267]]
Accuracy on Train :  98.6
Accuracy on Test1 :  89.23
Accuracy on Test2 :  95.65
For Test1 dataset
Confusion Matrix:
[[1671   22]
 [ 265  707]]

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.99      0.92      1693
           1       0.97      0.73      0.83       972

    accuracy                           0.89      2665
   macro avg       0.92      0.86      0.88      2665
weighted avg       0.90      0.89      0.89      2665


Accuracy: 0.8923076923076924
For Test2 dataset
Confusion Matrix:
[[7649   54]
 [ 370 1679]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      7703
           1       0.97      0.82      0.89      2049

    accuracy                           0.96      9752
   macro avg       0.96      0.91      0.93      9752
weighted avg       0.96      0.96      0.96      9752


Accuracy: 0.9565217391304348
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=0, solver='newton-cg', tol=0.0001, verbose=0,
                   warm_start=False)
Coefficients:
 [[-1.25313994  0.19245224  3.91325952  1.8958613  -0.39467167]]
Accuracy on Train :  98.6
Accuracy on Test1 :  89.04
Accuracy on Test2 :  95.61
For Test1 dataset
Confusion Matrix:
[[1671   22]
 [ 270  702]]

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.99      0.92      1693
           1       0.97      0.72      0.83       972

    accuracy                           0.89      2665
   macro avg       0.92      0.85      0.87      2665
weighted avg       0.90      0.89      0.89      2665


Accuracy: 0.8904315196998124
For Test2 dataset
Confusion Matrix:
[[7651   52]
 [ 376 1673]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      7703
           1       0.97      0.82      0.89      2049

    accuracy                           0.96      9752
   macro avg       0.96      0.90      0.93      9752
weighted avg       0.96      0.96      0.95      9752


Accuracy: 0.9561115668580804
```

2) Naïve Bayes –

Naïve Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with a strong assumption that all the predictors are in dependent to each other i.e. the presence of a feature in a class is independent to the presence of any other feature in the same class.

```
[156] from sklearn.naive_bayes import GaussianNB
      # GNB Classifier
      GNB = GaussianNB()

      # training
      GNB.fit(X_scaled_train, Y_train)

      # predictions
      Y_pred1 = GNB.predict(X_scaled_test1)
      Y_pred2 = GNB.predict(X_scaled_test2)
      # accuracy
      acc1 = GNB.score(X_scaled_test1, Y_test1)
      acc2 = GNB.score(X_scaled_test2, Y_test2)
      print(acc1)
      print(acc2)
```

```
0.9039399624765478
0.9914889253486464
```

Different performance metrices used for evaluating machine learning model -

```
# Confusion Matrix
result = confusion_matrix(Y_test1, Y_pred1)
print("Confusion Matrix:")
print(result)
# Classification report
result1 = classification_report(Y_test1, Y_pred1)
print("\nClassification Report:")
print (result1)
# Accuracy score
result2 = accuracy_score(Y_test1, Y_pred1)
print("\nAccuracy:",result2)

# Confusion Matrix
result = confusion_matrix(Y_test2, Y_pred2)
print("Confusion Matrix:")
print(result)
# Classification report
result1 = classification_report(Y_test2, Y_pred2)
print("\nClassification Report:")
print (result1)
 # Accuracy score
result2 = accuracy_score(Y_test2, Y_pred2)
print("\nAccuracy:",result2)
```

```
Confusion Matrix:
[[1682   11]
 [ 245  727]]

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.99      0.93      1693
           1       0.99      0.75      0.85       972

    accuracy                           0.90      2665
   macro avg       0.93      0.87      0.89      2665
weighted avg       0.91      0.90      0.90      2665


Accuracy: 0.9039399624765478
Confusion Matrix:
[[7635   68]
 [  15 2034]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      7703
           1       0.97      0.99      0.98      2049

    accuracy                           0.99      9752
   macro avg       0.98      0.99      0.99      9752
weighted avg       0.99      0.99      0.99      9752


Accuracy: 0.9914889253486464
```

3) K–Nearest Neighbors

It's nonparametric and lazy in nature. Nonparametric means that there is no assumption for the underlying data distribution i.e. the model structure is determined from the dataset. Lazy or instance learning means that for the purpose of model generation, it does not require any training data points and whole training data is used in the testing phase.

```
from sklearn.neighbors import KNeighborsClassifier
knnr = KNeighborsClassifier(n_neighbors = 5, weights='uniform')
knnr.fit(X_scaled_train, Y_train)
# accuracy
print('Accuracy on Train : ', round(knnr.score(X_scaled_train, Y_train)*100, 2))
print('Accuracy on Test1 : ', round(knnr.score(X_scaled_test1, Y_test1)*100, 2))
print('Accuracy on Test2 : ', round(knnr.score(X_scaled_test2, Y_test2)*100, 2))
```

```
Accuracy on Train :  99.58
Accuracy on Test1 :  96.06
Accuracy on Test2 :  96.67
```

Different performance metrices are used for evaluating machine learning model -

```
# predictions
Y_pred1 = knnr.predict(X_scaled_test1)
Y_pred2 = knnr.predict(X_scaled_test2)
# Confusion Matrix
result = confusion_matrix(Y_test1, Y_pred1)
print("Confusion Matrix:")
print(result)
# Classification report
result1 = classification_report(Y_test1, Y_pred1)
print("\nClassification Report:")
print (result1)
# Accuracy score
result2 = accuracy_score(Y_test1, Y_pred1)
print("\nAccuracy:",result2)

# Confusion Matrix
result = confusion_matrix(Y_test2, Y_pred2)
print("Confusion Matrix:")
print(result)
# Classification report
result1 = classification_report(Y_test2, Y_pred2)
print("\nClassification Report:")
print (result1)
 # Accuracy score
result2 = accuracy_score(Y_test2, Y_pred2)
print("\nAccuracy:",result2)
```

```
Confusion Matrix:
[[1648   45]
 [  60  912]]

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.97      0.97      1693
           1       0.95      0.94      0.95       972

    accuracy                           0.96      2665
   macro avg       0.96      0.96      0.96      2665
weighted avg       0.96      0.96      0.96      2665


Accuracy: 0.9606003752345216
Confusion Matrix:
[[7568  135]
 [ 190 1859]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      7703
           1       0.93      0.91      0.92      2049

    accuracy                           0.97      9752
   macro avg       0.95      0.94      0.95      9752
weighted avg       0.97      0.97      0.97      9752


Accuracy: 0.9666735028712059
```

4) Random Forest –

It computes the locally optimal feature/split combination. In Random forest, each decision tree in the ensemble is built from a sample drawn with replacement from the training set and then gets the prediction from each of them and finally selects the best solution by means of voting. It can be used for both classification as well as regression tasks.
Random forest is used over decision tree to prevent overfitting of model on training dataset .

```
from sklearn.ensemble import RandomForestClassifier
# classifier
RF = RandomForestClassifier(n_estimators = 50)

# training
RF.fit(X_scaled_train, Y_train)

# predictions
Y_pred1 = RF.predict(X_scaled_test1)
Y_pred2 = RF.predict(X_scaled_test2)
```

```
# Confusion Matrix
result = confusion_matrix(Y_test1, Y_pred1)
print("Confusion Matrix:")
print(result)
# Classification report
result1 = classification_report(Y_test1, Y_pred1)
print("\nClassification Report:")
print (result1)
# Accuracy score
result2 = accuracy_score(Y_test1, Y_pred1)
print("\nAccuracy:",result2)

# Confusion Matrix
result = confusion_matrix(Y_test2, Y_pred2)
print("Confusion Matrix:")
print(result)
# Classification report
result1 = classification_report(Y_test2, Y_pred2)
print("\nClassification Report:")
print (result1)
 # Accuracy score
result2 = accuracy_score(Y_test2, Y_pred2)
print("\nAccuracy:",result2)
```

```
Accuracy on Train :  100.0
Accuracy on Test1 :  92.83
Accuracy on Test2 :  96.32
Confusion Matrix:
[[1657   36]
 [ 155  817]]

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.98      0.95      1693
           1       0.96      0.84      0.90       972

    accuracy                           0.93      2665
   macro avg       0.94      0.91      0.92      2665
weighted avg       0.93      0.93      0.93      2665


Accuracy: 0.9283302063789869
Confusion Matrix:
[[7663   40]
 [ 319 1730]]

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.99      0.98      7703
           1       0.98      0.84      0.91      2049

    accuracy                           0.96      9752
   macro avg       0.97      0.92      0.94      9752
weighted avg       0.96      0.96      0.96      9752


Accuracy: 0.9631870385561936
```

5) Support Vector Machine –

Support vector machines (SVMs) are powerful yet flexible supervised machine learning methods used for classification, regression, and, outliers' detection. SVMs are very efficient in high dimensional spaces and generally are used in classification problems.The main goal of SVMs is to divide the datasets into number of classes in order to find a maximum marginal hyperplane (MMH)

```python
from sklearn.svm import SVC
kernels ={ "linear", "poly", "rbf", "sigmoid"}
for k in kernels:
    svm = SVC(kernel = k,gamma = 'scale')
    svm.fit(X_scaled_train, Y_train)
    Y_pred1 = RF.predict(X_scaled_test1)
    Y_pred2 = RF.predict(X_scaled_test2)
    # accuracy
    print(k)
    print('Accuracy on Train : ', round(svm.score(X_scaled_train, Y_train)*100, 2))
    print('Accuracy on Test1 : ', round(svm.score(X_scaled_test1, Y_test1)*100, 2))
    print('Accuracy on Test2 : ', round(svm.score(X_scaled_test2, Y_test2)*100, 2))
```

```
sigmoid
Accuracy on Train :  91.53
Accuracy on Test1 :  81.43
Accuracy on Test2 :  84.84
poly
Accuracy on Train :  97.8
Accuracy on Test1 :  86.15
Accuracy on Test2 :  81.6
linear
Accuracy on Train :  98.62
Accuracy on Test1 :  91.74
Accuracy on Test2 :  99.24
rbf
Accuracy on Train :  98.88
Accuracy on Test1 :  97.79
Accuracy on Test2 :  97.87
```

It is visible that accuracy is higher when we use "linear" and "rbf" kernel
Now checking performance metrices for **"linear" and "rbf"** kernels.

```
linear

Accuracy on Train :  98.62
Accuracy on Test1 :  91.74
Accuracy on Test2 :  99.24
Confusion Matrix:
[[1667   26]
 [ 194  778]]

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      1693
           1       0.97      0.80      0.88       972

    accuracy                           0.92      2665
   macro avg       0.93      0.89      0.91      2665
weighted avg       0.92      0.92      0.92      2665


Accuracy: 0.9174484052532833
Confusion Matrix:
[[7645   58]
 [  16 2033]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      7703
           1       0.97      0.99      0.98      2049

    accuracy                           0.99      9752
   macro avg       0.99      0.99      0.99      9752
weighted avg       0.99      0.99      0.99      9752


Accuracy: 0.9924118129614438

rbf

Accuracy on Train :  98.88
Accuracy on Test1 :  97.79
Accuracy on Test2 :  97.87
Confusion Matrix:
[[1638   55]
 [   4  968]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.98      1693
           1       0.95      1.00      0.97       972

    accuracy                           0.98      2665
   macro avg       0.97      0.98      0.98      2665
weighted avg       0.98      0.98      0.98      2665


Accuracy: 0.9778611632270169
Confusion Matrix:
[[7640   63]
 [ 145 1904]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      7703
           1       0.97      0.93      0.95      2049

    accuracy                           0.98      9752
   macro avg       0.97      0.96      0.97      9752
weighted avg       0.98      0.98      0.98      9752


Accuracy: 0.9786710418375718
```

6) Artificial Neural Network(ANNs)-

Importing libraries and building model architecture -

```
[143] import keras
      import tensorflow as tf
      from keras.models import Sequential
      from keras.layers import Dense
      from keras.layers import Dropout
      from tensorflow.keras.optimizers import Adam
```

```
[144] model = Sequential()
      model.add(Dense(64, input_dim =5, activation='relu'))
      model.add(Dropout(rate=0.3))
      model.add(Dense(32, activation='relu'))
      model.add(Dense(8, activation='relu'))
      model.add(Dense(1, activation='sigmoid'))
```

Choosing optimizer , loss function -

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy']) #binary_crossentropy is used for binary classification
history = model.fit(X_scaled_train,Y_train,epochs=20,verbose=True)
model.summary()
```

```
Epoch 1/20
255/255 [==============================] - 1s 1ms/step - loss: 0.2719 - accuracy: 0.9352
Epoch 2/20
255/255 [==============================] - 0s 1ms/step - loss: 0.0520 - accuracy: 0.9870
Epoch 3/20
255/255 [==============================] - 0s 1ms/step - loss: 0.0440 - accuracy: 0.9881
Epoch 4/20
255/255 [==============================] - 0s 2ms/step - loss: 0.0422 - accuracy: 0.9886
Epoch 5/20
255/255 [==============================] - 0s 2ms/step - loss: 0.0371 - accuracy: 0.9889
Epoch 6/20
255/255 [==============================] - 0s 2ms/step - loss: 0.0425 - accuracy: 0.9872
Epoch 7/20
255/255 [==============================] - 0s 1ms/step - loss: 0.0417 - accuracy: 0.9874
Epoch 8/20
255/255 [==============================] - 0s 2ms/step - loss: 0.0324 - accuracy: 0.9892
Epoch 9/20
255/255 [==============================] - 0s 1ms/step - loss: 0.0320 - accuracy: 0.9888
Epoch 10/20
255/255 [==============================] - 0s 2ms/step - loss: 0.0358 - accuracy: 0.9871
Epoch 11/20
255/255 [==============================] - 0s 2ms/step - loss: 0.0350 - accuracy: 0.9876
Epoch 12/20
255/255 [==============================] - 0s 1ms/step - loss: 0.0320 - accuracy: 0.9871
Epoch 13/20
255/255 [==============================] - 0s 1ms/step - loss: 0.0299 - accuracy: 0.9874
Epoch 14/20
255/255 [==============================] - 0s 1ms/step - loss: 0.0337 - accuracy: 0.9882
Epoch 15/20
255/255 [==============================] - 0s 2ms/step - loss: 0.0236 - accuracy: 0.9901
Epoch 16/20
255/255 [==============================] - 0s 1ms/step - loss: 0.0316 - accuracy: 0.9880
Epoch 17/20
255/255 [==============================] - 0s 1ms/step - loss: 0.0294 - accuracy: 0.9903
Epoch 18/20
255/255 [==============================] - 0s 2ms/step - loss: 0.0290 - accuracy: 0.9877
Epoch 19/20
255/255 [==============================] - 0s 1ms/step - loss: 0.0286 - accuracy: 0.9876
```

```
Epoch 20/20
255/255 [==============================] - 0s 1ms/step - loss: 0.0245 - accuracy: 0.9881
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_62 (Dense)             (None, 64)                384
_________________________________________________________________
dropout_15 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_63 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_64 (Dense)             (None, 8)                 264
_________________________________________________________________
dense_65 (Dense)             (None, 1)                 9
=================================================================
Total params: 2,737
Trainable params: 2,737
Non-trainable params: 0
```

Checking accuracy of trained ANN on testing datasets .

```
[146] y_pred1 = model.predict(X_scaled_test1)
      y_pred2 = model.predict(X_scaled_test2)
```

```
[147] loss1, accuracy1  = model.evaluate(X_scaled_test1,
                                Y_test1, verbose=True)
      loss1  , accuracy1
```

```
84/84 [==============================] - 0s 1ms/step - loss: 0.1241 - accuracy: 0.9606
(0.12409702688455582, 0.9606003761291504)
```

```
[148] loss2, accuracy2  = model.evaluate(X_scaled_test2,
                                Y_test2, verbose=True)
      loss2  , accuracy2
```

```
305/305 [==============================] - 0s 958us/step - loss: 0.0732 - accuracy: 0.9746
(0.07323317229747772, 0.9745693206787109)
```

Model Analysis

In Logistic Regression "liblinear" can be used as solver(Optimization algorithm used) for best accuracy on testing datasets .

Solver	Train	Test1	Test2
liblinear	98.6	89.23	95.65
sag	98.6	89.04	95.61
newton-cg	98.6	89.04	95.61
saga	98.6	89.04	95.63
lbfgs	98.6	89.04	95.61

In SVM "rbf" kernel can be used as it gives best accuracy among all kernels for SVM .

Solver	Train	Test1	Test2
linear	98.62	91.74	99.24
rbf	98.88	97.79	97.87

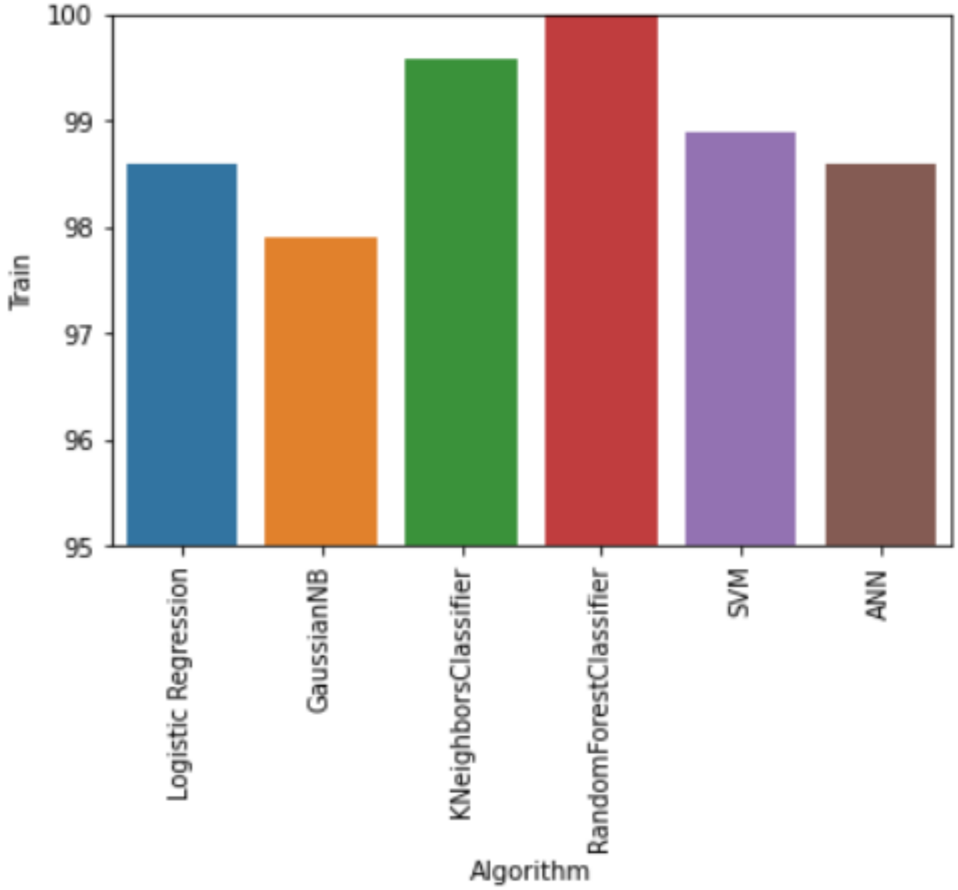
For ANN accuracy and loss on training and testing datasets are -

		Train	Test1	Test2
0	Accuracy	98.580	96.510	97.260
1	Loss	0.037	0.126	0.095

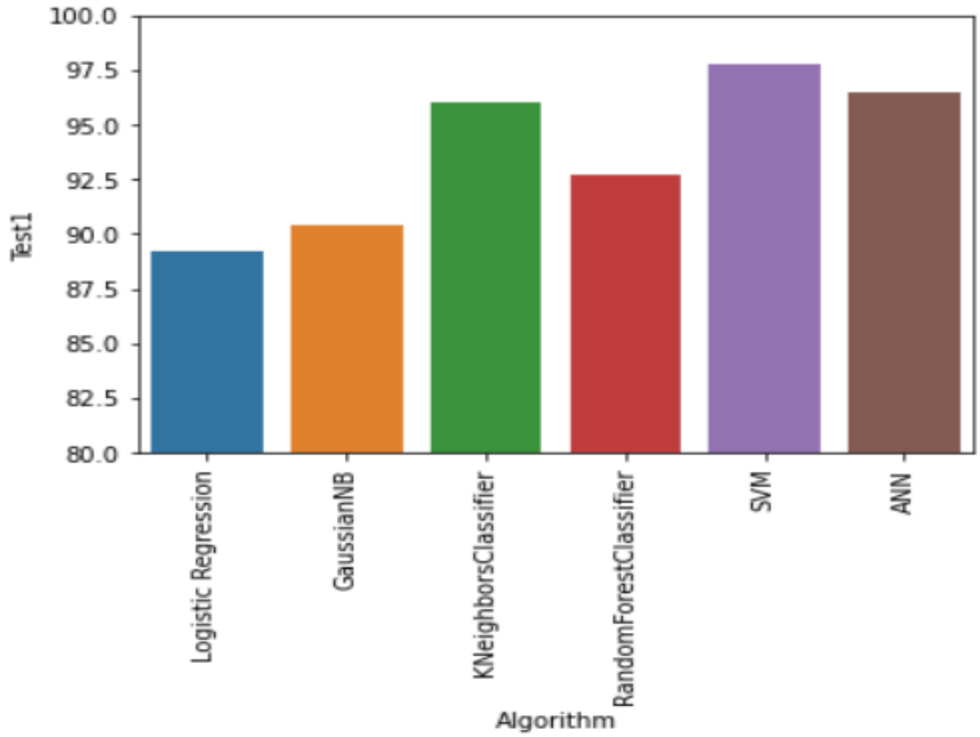
Overall accuracy across all the datasets –

Algorithm	Train	Test1	Test2
Logistic Regression	98.60	89.23	95.65
GaussianNB	97.89	90.39	99.15
KNeighborsClassifier	99.58	96.06	96.67
RandomForestClassifier	100.00	92.72	96.32
SVM	98.88	97.79	97.87
ANN	98.58	96.51	97.26

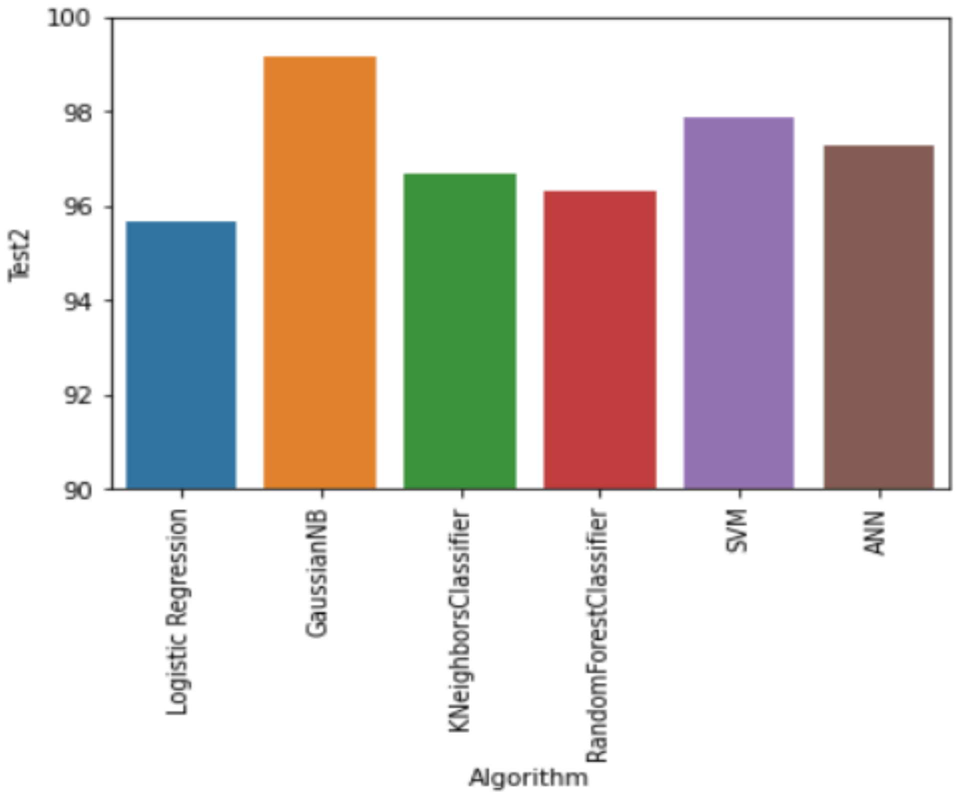
For Training dataset-



For Test1 dataset -



For Test2 dataset -



All in all , all models in general did a great job but we can say that SVM gives highest accuracy on Test1 dataset and GaussianNaive Bayes gives highest accuracy on Test 2 dataset .