17
NOV
2020

ASSIGNMENT 2

SUBJECT : DEEP LEARNING [IT702]

SHANKARANARAYAN N

SUBJECT : DEEP LEARNING [IT702]

SHANKARANARAYAN N | M.TECH(RESEARCH) IT

ASSIGNMENT 2
SUBJECT : IT702 - DEEP LEARNING

AIM :

- **TASK1 :TO SPLIT THE DATA IN 8:2 RATION AND TABULATE THE RESULT**
- **TASK2 :TO USE 10-FOLD CROSS VERIFICATION AND TABULATE THE RESULT**
- **TASK3 :TO CHANGE THE HYPER AND TABULATE THE OBSERVATION**

-ALL THE ABOVE TASKS HAVE TO BE DONE WITH SVM AS WELL AS RANDOM FOREST-

DATASET :

COLUMNS :

- The dataset contains a total of 35 columns and 351 records.
- X -> column_a - column ah
- Y -> column_ai

-THE TRAINING DATA CONTAINS A TOTAL OF 351 DATASETS-

DATA SET INFORMATION:

This radar data was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. The targets data is the free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere.

Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the Goose Bay system. Instances in this database are described by 2 attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal.

ATTRIBUTE INFORMATION:

- All 34 are continuous
- The 35th attribute is either "good" or "bad" according to the definition summarized above. This is a binary classification task.

RELEVANT PAPERS:

Sigillito, V. G., Wing, S. P., Hutton, L. V., & Baker, K. B. (1989). Classification of radar returns from the ionosphere using neural networks. Johns Hopkins APL Technical Digest, 10, 262-266.

DATASET DESCRIPTION:

	column_a	column_b	column_c	column_d	column_e	column_f	column_g	column_h	column_i	column_j	column_k	colu
0	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.00000	0.03760	0.85243	-0.1
1	1	0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	0.50874	-0.6
2	1	0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	0.73082	0.0
3	1	0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	0.00000	0.0
4	1	0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	0.52798	-0.2
...	...	...	...	...	...	...	...	...	...	...	...	
346	1	0	0.83508	0.08298	0.73739	-0.14706	0.84349	-0.05567	0.90441	-0.04622	0.89391	0.1
347	1	0	0.95113	0.00419	0.95183	-0.02723	0.93438	-0.01920	0.94590	0.01606	0.96510	0.0
348	1	0	0.94701	-0.00034	0.93207	-0.03227	0.95177	-0.03431	0.95584	0.02446	0.94124	0.0
349	1	0	0.90608	-0.01657	0.98122	-0.01989	0.95691	-0.03646	0.85746	0.00110	0.89724	-0.0
350	1	0	0.84710	0.13533	0.73638	-0.06151	0.87873	0.08260	0.88928	-0.09139	0.78735	0.0

351 rows × 35 columns

ENVIRONMENT :

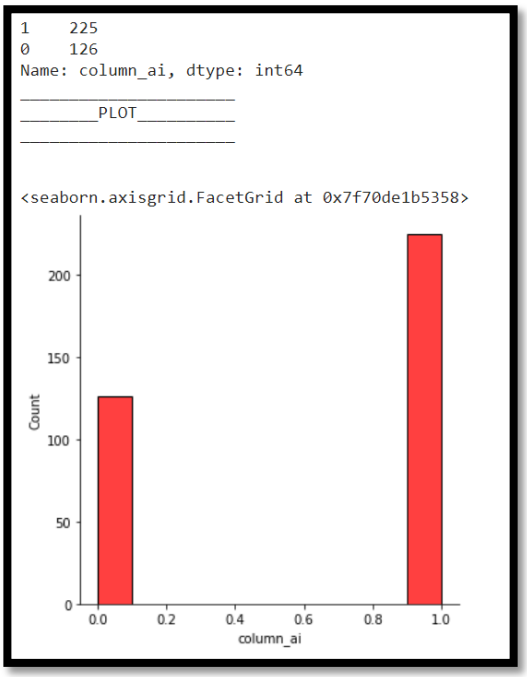
- The entire assignment is executed in Google colab.
- The data set is stored and retrieved from Google Drive.

DISTRIBUTION OF THE OUTCOME (BINARY)

1->Good

0->Bad

```
1    225
0    126
Name: column_ai, dtype: int64
_____________________
________PLOT__________
_____________________

<seaborn.axisgrid.FacetGrid at 0x7f70de1b5358>
```



FROM THE GRAPH PLOT WE CAN OBSERVE THAT,
WE HAVE EXACTLY 225-GOOD AND 126-BAD RECORDS TO TRAIN OUR DATA SET

OBSERVATIONS : TASK 1-TO SPLIT THE DATA IN 8:2 RATION AND TABULATE THE RESULT (SVM)

```
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=0.2)

from sklearn.svm import SVC
svc=SVC(kernel='linear')
svc.fit(X_train,Y_train)
ypredict=svc.predict(X_test)
print(classification_report(Y_test,ypredict))
print("_____________")
```

```
              precision    recall  f1-score   support

           0       0.92      0.71      0.80        34
           1       0.78      0.95      0.85        37

    accuracy                           0.83        71
   macro avg       0.85      0.83      0.83        71
weighted avg       0.85      0.83      0.83        71


_____________
```

- train_test_split() – splits the dataset into testing and training set.
- We are using linear kernel
- Once the svc.predict() is done we are calculating and displaying the accuracy.

OBSERVATIONS : TASK 1-TO SPLIT THE DATA IN 8:2 RATION AND TABULATE THE RESULT (Random forest)

```
rfc = RandomForestClassifier()
rfc.fit(X_train,Y_train)

rfc_predict = rfc.predict(X_test)
print(classification_report(Y_test,rfc_predict))
print("_____________")
```

```
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        34
           1       0.92      0.97      0.95        37

    accuracy                           0.94        71
   macro avg       0.95      0.94      0.94        71
weighted avg       0.94      0.94      0.94        71


_____________
```

- We use a similar procedure to calculate the accuracy here

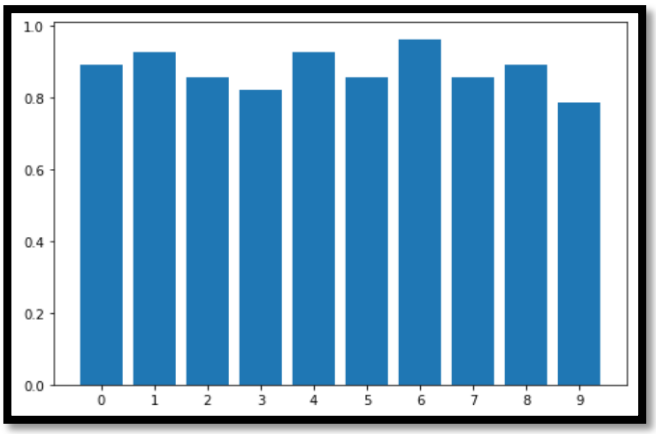
Here, along with accuracy/precision few other details are also displayed such as recall, f1-score and support.

- **Precision** : the number of positive class predictions that actually belong to the positive class.
- **Recall** : the number of positive class predictions made out of all positive examples in the dataset.
- **F-Measure** : a single score that balances both the concerns of precision and recall in one number.

OBSERVATIONS : TASK 2 - TO USE 10-FOLD CROSS VERIFICATION AND TABULATE THE RESULT (SVM)

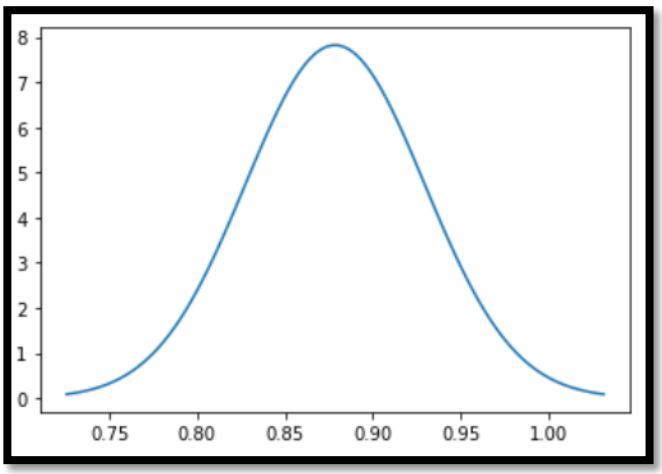
```
from sklearn.model_selection import cross_val_score
clf = svm.SVC(kernel='linear')
scores = cross_val_score(clf, X_train, Y_train, cv=10)
for i in range(10):
  print("ACCURACY OF FOLD",i+1,"is\t",scores[i])
print("\n\n Mean Accuracy: %0.2f (WITH STANDARD DEVIATION+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
ACCURACY OF FOLD 1 is    0.8928571428571429
ACCURACY OF FOLD 2 is    0.9285714285714286
ACCURACY OF FOLD 3 is    0.8571428571428571
ACCURACY OF FOLD 4 is    0.8214285714285714
ACCURACY OF FOLD 5 is    0.9285714285714286
ACCURACY OF FOLD 6 is    0.8571428571428571
ACCURACY OF FOLD 7 is    0.9642857142857143
ACCURACY OF FOLD 8 is    0.8571428571428571
ACCURACY OF FOLD 9 is    0.8928571428571429
ACCURACY OF FOLD 10 is   0.7857142857142857


 Mean Accuracy: 0.88 (WITH STANDARD DEVIATION+/- 0.10)
```



THIS GRAPH DISPLAYS THE ACCURACY VALUES OF EVERY FOLD(SVM).

THE ACCURACY VALUES ARE STORED IN AN ARRAY AND DISPLAYED VIA A BAR GRAPH.

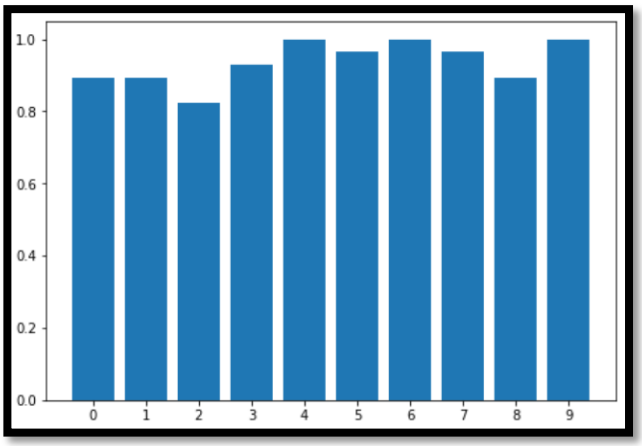


NORMAL DISTRIBUTION OF THE ACCURACY VALUES.

OBSERVATIONS : TASK 2 - TO USE 10-FOLD CROSS VERIFICATION AND TABULATE THE RESULT (Random forest)

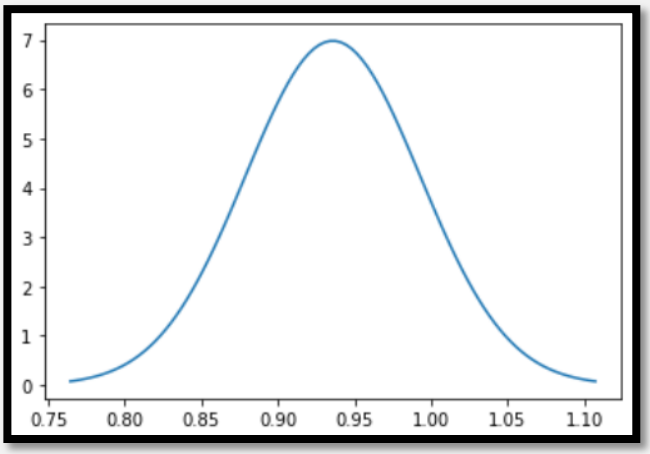
```
from sklearn.model_selection import cross_val_score
rfc = RandomForestClassifier()
scores = cross_val_score(rfc, X_train, Y_train, cv=10)
for i in range(10):
  print("ACCURACY OF FOLD",i+1,"is\t",scores[i])
print("\n\n Mean Accuracy: %0.2f (WITH STANDARD DEVIATION+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
ACCURACY OF FOLD 1 is    0.8928571428571429
ACCURACY OF FOLD 2 is    0.9285714285714286
ACCURACY OF FOLD 3 is    0.8571428571428571
ACCURACY OF FOLD 4 is    0.9285714285714286
ACCURACY OF FOLD 5 is    0.9642857142857143
ACCURACY OF FOLD 6 is    0.9642857142857143
ACCURACY OF FOLD 7 is    0.9642857142857143
ACCURACY OF FOLD 8 is    0.9642857142857143
ACCURACY OF FOLD 9 is    0.8571428571428571
ACCURACY OF FOLD 10 is   1.0


 Mean Accuracy: 0.93 (WITH STANDARD DEVIATION+/- 0.09)
```



THIS GRAPH DISPLAYS THE ACCURACY VALUES OF EVERY FOLD (RANDOM FOREST).

THE ACCURACY VALUES ARE STORED IN AN ARRAY AND DISPLAYED VIA A BAR GRAPH



NORMAL DISTRIBUTION OF THE ACCURACY VALUES.

IN TASK 2, SVM GIVES BETTER RESULT IN COMPARISON WITH RANDOM FOREST

TASK 2 : Procedure and Conclusion

- We are using the cross_val_score() method to perform the 10-fold cross verification.
- All the accuracy values of every respective fold is also printed.
- Applying 10 fold cross verification in SVM(LINEAR KERNAL) yields better result than Random forest

OBSERVATION :

TASK 3- TO CHANGE THE HYPER PARAMETERS AND TABULATE THE OBSERVATION (SVM)

```
params={'C':[0.001,0.01],
        'gamma':[0.001,0.01],
        'kernel':['linear','poly','rbf','sigmoid']}

grid_search=GridSearchCV(SVC(random_state=0),params,n_jobs=-1)
grid_search.fit(X_train,Y_train)

print("Train Score"+str(grid_search.score(X_train,Y_train)))

print("TEST Score"+str(grid_search.score(X_test,Y_test)))

print(grid_search.best_params_)
```

```
Train Score0.8571428571428571
TEST Score0.7323943661971831
{'C': 0.01, 'gamma': 0.001, 'kernel': 'linear'}
```

```
means = grid_search.cv_results_['mean_test_score']
stds = grid_search.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, grid_search.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r"% (mean, std * 2, params))
```

```
0.671 (+/-0.017) for {'C': 0.001, 'gamma': 0.001, 'kernel': 'linear'}
0.671 (+/-0.017) for {'C': 0.001, 'gamma': 0.001, 'kernel': 'poly'}
0.671 (+/-0.017) for {'C': 0.001, 'gamma': 0.001, 'kernel': 'rbf'}
0.671 (+/-0.017) for {'C': 0.001, 'gamma': 0.001, 'kernel': 'sigmoid'}
0.671 (+/-0.017) for {'C': 0.001, 'gamma': 0.01, 'kernel': 'linear'}
0.671 (+/-0.017) for {'C': 0.001, 'gamma': 0.01, 'kernel': 'poly'}
0.671 (+/-0.017) for {'C': 0.001, 'gamma': 0.01, 'kernel': 'rbf'}
0.671 (+/-0.017) for {'C': 0.001, 'gamma': 0.01, 'kernel': 'sigmoid'}
0.836 (+/-0.052) for {'C': 0.01, 'gamma': 0.001, 'kernel': 'linear'}
0.671 (+/-0.017) for {'C': 0.01, 'gamma': 0.001, 'kernel': 'poly'}
0.671 (+/-0.017) for {'C': 0.01, 'gamma': 0.001, 'kernel': 'rbf'}
0.671 (+/-0.017) for {'C': 0.01, 'gamma': 0.001, 'kernel': 'sigmoid'}
0.836 (+/-0.052) for {'C': 0.01, 'gamma': 0.01, 'kernel': 'linear'}
0.671 (+/-0.017) for {'C': 0.01, 'gamma': 0.01, 'kernel': 'poly'}
0.671 (+/-0.017) for {'C': 0.01, 'gamma': 0.01, 'kernel': 'rbf'}
0.671 (+/-0.017) for {'C': 0.01, 'gamma': 0.01, 'kernel': 'sigmoid'}
```

- Here, We are using four different kernals with 2 different Gamma and C values,
    - Kernal : Linear,Poly,RBF,Sigmoid
    - Gamma : 0.001, 0.01
    - C : 0.001, 0.01

  These different parameters produce 12 different SVM classification function.
- C : Cost of miscalculation. Large C gives you low bias and high variance. Small C gives you higher bias and lower variance.
- Gamma : Parameter of Gaussian Kernel (to handle non-linear classification). Gamma controls the shape of the "peaks" where we raise the points for non linear classification. A small gamma gives you a pointed bump in the higher dimensions, a large gamma gives you a broader bump.

OBSERVATION :

TASK 3- TO CHANGE THE HYPER PARAMETERS AND TABULATE THE OBSERVATION (Random Forest)

```
# Number of trees in random forest
#n_estimators = [int(x) for x in np.linspace(start = 10, stop = 80, num = 10)]
n_estimators =[10,30]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [2,4]
param_grid = {'n_estimators': n_estimators,'max_features': max_features,'max_depth': max_depth}
rf_Model = RandomForestClassifier()
rf_Grid = GridSearchCV(estimator = rf_Model, param_grid = param_grid, cv = 3, verbose=2, n_jobs = 4)
resulthere=rf_Grid.fit(X_train, Y_train)
rf_Grid.best_params_
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  24 out of  24 | elapsed:    0.7s finished
{'max_depth': 4, 'max_features': 'auto', 'n_estimators': 10}
```

```
means = resulthere.cv_results_['mean_test_score']
stds = resulthere.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, resulthere.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r"% (mean, std * 2, params))
```

```
0.900 (+/-0.043) for {'max_depth': 2, 'max_features': 'auto', 'n_estimators': 10}
0.893 (+/-0.018) for {'max_depth': 2, 'max_features': 'auto', 'n_estimators': 30}
0.911 (+/-0.036) for {'max_depth': 2, 'max_features': 'sqrt', 'n_estimators': 10}
0.914 (+/-0.052) for {'max_depth': 2, 'max_features': 'sqrt', 'n_estimators': 30}
0.932 (+/-0.055) for {'max_depth': 4, 'max_features': 'auto', 'n_estimators': 10}
0.922 (+/-0.078) for {'max_depth': 4, 'max_features': 'auto', 'n_estimators': 30}
0.918 (+/-0.078) for {'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 10}
0.922 (+/-0.061) for {'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 30}
```

Similar to what we did in SCM… Here,

- We are using the GridSearchSV method to modify the hyper parameters and to find the parameter which gives the best accuracy.
- All the accuracy values of each parameter combination is also printed for better understanding.
- It is observed that (Max depth:4 | Max features:auto | estimator:10) gives the best accuracy.

ASSIGNMENT 2

SUBJECT : IT702 - DEEP LEARNING

> **SYBMITTED BY : SHANKARANARAYAN N, M.TECH(RESEARCH), DEPARTMENT OF INFORMATION TECHNOLOGY**

NECESSARY IMPORTS

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

x=np.array([0.001])
y=np.array([0.001])
z=np.array([0.001])
```

IMPORTING THE DATA AND MODIFYING IT TO OUR NEED

```
# Import PyDrive and associated libraries.
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)



downloaded = drive.CreateFile({'id':'1_mhzaq-G2Ounk6qxJq4XKsr092QcM4N7'})
downloaded.GetContentFile('assignment2data.csv')


df = pd.read_csv('assignment2data.csv')
df["column_a"].replace({True:1 }, inplace=True)
df["column_a"].replace({False:0 }, inplace=True)
df["column_b"].replace({True:1 }, inplace=True)
```

```
df["column_b"].replace({False:0 }, inplace=True)
df["column_ai"].replace({"g":1 }, inplace=True)
df["column_ai"].replace({"b":0 }, inplace=True)
df
```

	column_a	column_b	column_c	column_d	column_e	column_f	column_g	column_h	column_i
0	1	0	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.00000
1	1	0	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000
2	1	0	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965
3	1	0	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000
4	1	0	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152
...	...	...	...	...	...	...	...	...	...
346	1	0	0.83508	0.08298	0.73739	-0.14706	0.84349	-0.05567	0.90441
347	1	0	0.95113	0.00419	0.95183	-0.02723	0.93438	-0.01920	0.94590
348	1	0	0.94701	-0.00034	0.93207	-0.03227	0.95177	-0.03431	0.95584
349	1	0	0.90608	-0.01657	0.98122	-0.01989	0.95691	-0.03646	0.85746
350	1	0	0.84710	0.13533	0.73638	-0.06151	0.87873	0.08260	0.88928

351 rows × 35 columns

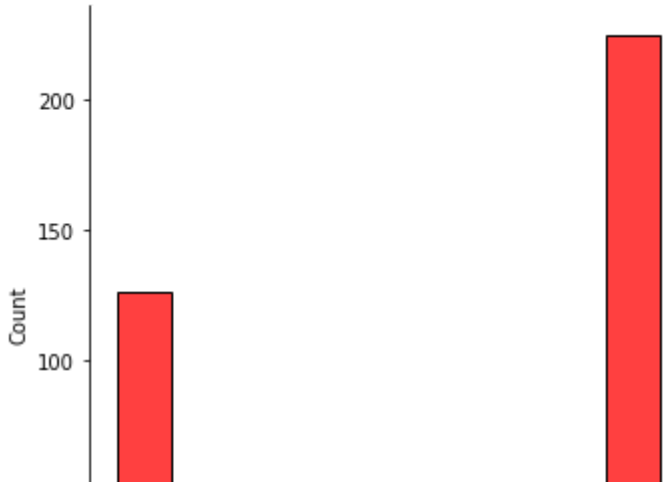
INFORMATION REGARDING THE DATASET AND THE DISTRIBUTION OF PREDICTION DATA

```
(df.groupby('column_ai')).count()
```

column_ai	column_a	column_b	column_c	column_d	column_e	column_f	column_g	column_h	colu
0	126	126	126	126	126	126	126	126	
1	225	225	225	225	225	225	225	225	

```
print(df['column_ai'].value_counts())
print("_____________________")
print("________PLOT__________")
print("_____________________")
print("\n")
sns.displot(df, x="column_ai",color='red')
```

```
1    225
0    126
Name: column_ai, dtype: int64
_________________________
_________PLOT____________
_________________________


<seaborn.axisgrid.FacetGrid at 0x7f70de1b5358>
```



SPLITTING THE DATA

```
df1=df.values
m2=len(df['column_ai'])
X=np.asarray(df1[:,0:34].reshape(m2,34))
y=np.asarray(df['column_ai'])
```

> Indented block

TASK 1 : TO SPLIT THE DATA IN 8:2 RATION AND TABULATE THE RESULT

TASK1:PART 1[SVM]

```
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=0.2)


from sklearn.svm import SVC
svc=SVC(kernel='linear')
svc.fit(X_train,Y_train)
ypredict=svc.predict(X_test)
print(classification_report(Y_test,ypredict))
print("_____________")
```

```
                precision    recall  f1-score   support

             0       0.92      0.71      0.80        34
             1       0.78      0.95      0.85        37
```

```
        accuracy                           0.83        71
       macro avg       0.85      0.83      0.83        71
    weighted avg       0.85      0.83      0.83        71

    _______________
```

TASK1:PART 2[RANDOM FOREST]

```
rfc = RandomForestClassifier()
rfc.fit(X_train,Y_train)

rfc_predict = rfc.predict(X_test)
print(classification_report(Y_test,rfc_predict))
print("_______________")
```

```
                  precision    recall  f1-score   support

               0       0.97      0.91      0.94        34
               1       0.92      0.97      0.95        37

        accuracy                           0.94        71
       macro avg       0.95      0.94      0.94        71
    weighted avg       0.94      0.94      0.94        71

    _______________
```

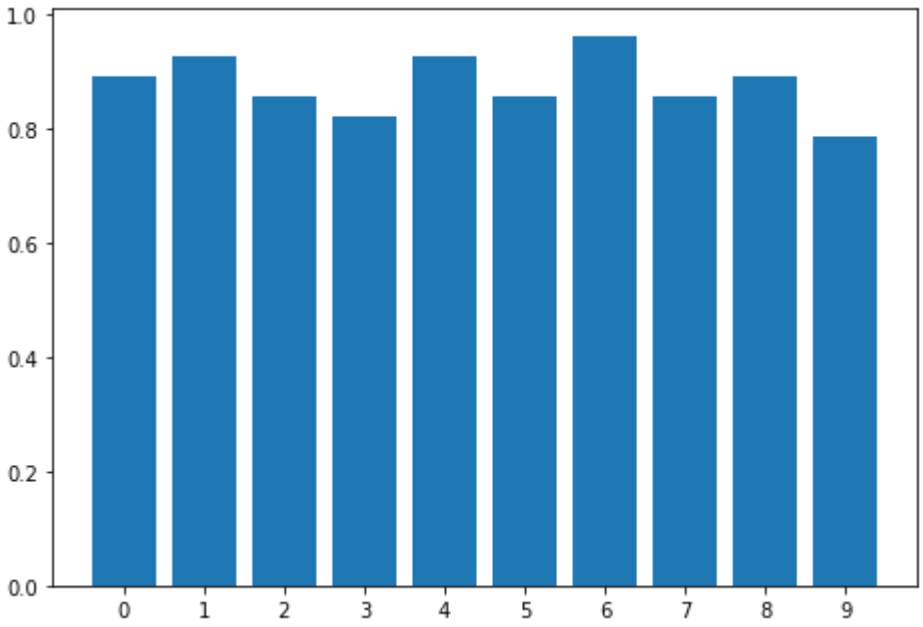
TASK 2:TO USE 10-FOLD CROSS VERIFICATION AND TABULATE THE RESULT

TASK2:PART 1[SVM-10FOLD]

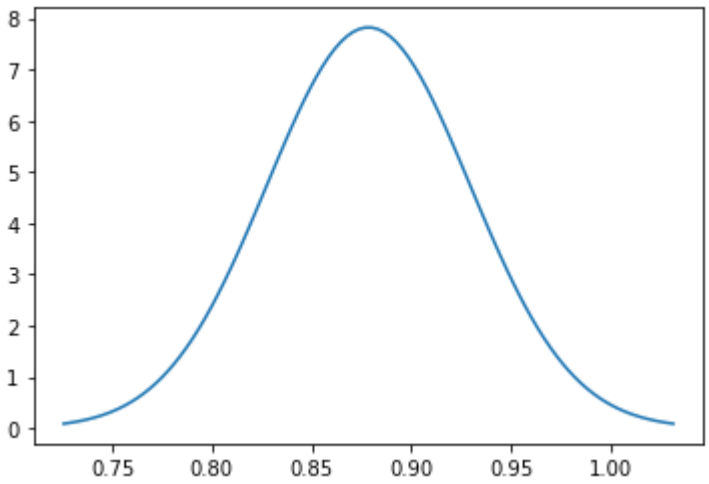
```
from sklearn.model_selection import cross_val_score
clf = svm.SVC(kernel='linear')
scores = cross_val_score(clf, X_train, Y_train, cv=10)
for i in range(10):
  print("ACCURACY OF FOLD",i+1,"is\t",scores[i])
print("\n\n Mean Accuracy: %0.2f (WITH STANDARD DEVIATION+/- %0.2f)" % (scores.mean(), scores.std()
```

```
    ACCURACY OF FOLD 1 is    0.8928571428571429
    ACCURACY OF FOLD 2 is    0.9285714285714286
    ACCURACY OF FOLD 3 is    0.8571428571428571
    ACCURACY OF FOLD 4 is    0.8214285714285714
    ACCURACY OF FOLD 5 is    0.9285714285714286
    ACCURACY OF FOLD 6 is    0.8571428571428571
    ACCURACY OF FOLD 7 is    0.9642857142857143
    ACCURACY OF FOLD 8 is    0.8571428571428571
    ACCURACY OF FOLD 9 is    0.8928571428571429
    ACCURACY OF FOLD 10 is   0.7857142857142857


     Mean Accuracy: 0.88 (WITH STANDARD DEVIATION+/- 0.10)
```

```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
langs = ['0', '1', '2', '3', '4','5','6','7','8','9']
ax.bar(langs,scores)
plt.show()
```




```
import scipy.stats as stats
import math

mu = scores.mean()
variance = scores.std()**2
sigma = scores.std()
x = np.linspace(mu - 3*sigma, mu + 3*sigma, 100)
plt.plot(x, stats.norm.pdf(x, mu, sigma))
plt.show()
```



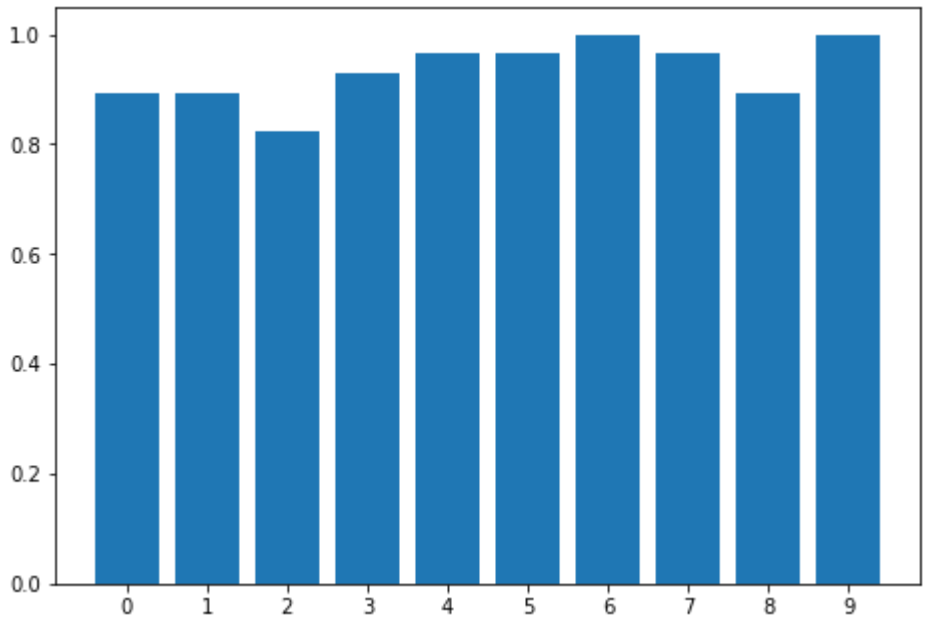
TASK2:PART 2[RANDOM FOREST-10FOLD]

```
from sklearn.model_selection import cross_val_score
rfc = RandomForestClassifier()
```

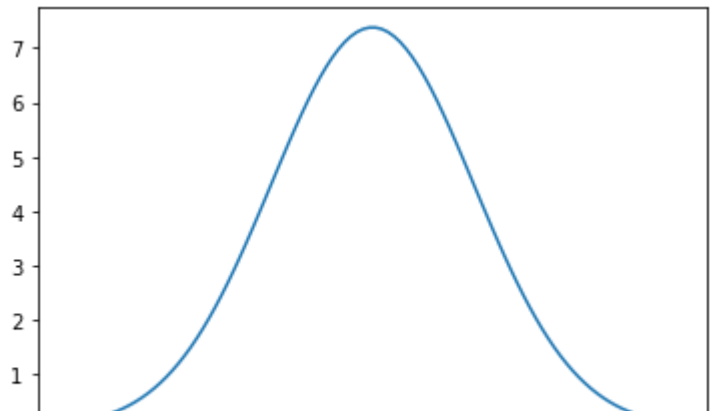
```
rfc = RandomForestClassifier()
scores = cross_val_score(rfc, X_train, Y_train, cv=10)
for i in range(10):
  print("ACCURACY OF FOLD",i+1,"is\t",scores[i])
print("\n\n Mean Accuracy: %0.2f (WITH STANDARD DEVIATION+/- %0.2f)" % (scores.mean(), scores.std()
```

```
ACCURACY OF FOLD 1 is    0.8928571428571429
ACCURACY OF FOLD 2 is    0.8928571428571429
ACCURACY OF FOLD 3 is    0.8214285714285714
ACCURACY OF FOLD 4 is    0.9285714285714286
ACCURACY OF FOLD 5 is    0.9642857142857143
ACCURACY OF FOLD 6 is    0.9642857142857143
ACCURACY OF FOLD 7 is    1.0
ACCURACY OF FOLD 8 is    0.9642857142857143
ACCURACY OF FOLD 9 is    0.8928571428571429
ACCURACY OF FOLD 10 is   1.0


 Mean Accuracy: 0.93 (WITH STANDARD DEVIATION+/- 0.11)
```

```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
langs = ['0', '1', '2', '3', '4','5','6','7','8','9']
ax.bar(langs,scores)
plt.show()
```




```
import scipy.stats as stats
import math

mu = scores.mean()
variance = scores.std()**2
sigma = scores.std()
x = np.linspace(mu - 3*sigma, mu + 3*sigma, 100)
plt.plot(x, stats.norm.pdf(x, mu, sigma))
plt.show()
```



TASK 3 : TO CHANGE THE HYPER PARAMETERS OF SVM AND RANDOM FOREST AND TABULATE THE OBSERVATION *italicized text*

TASK3:PART 1[SVM]

```
params={'C':[0.001,0.01],
        'gamma':[0.001,0.01],
        'kernel':['linear','poly','rbf','sigmoid']}

grid_search=GridSearchCV(SVC(random_state=0),params,n_jobs=-1)
grid_search.fit(X_train,Y_train)

print("Train Score"+str(grid_search.score(X_train,Y_train)))

print("TEST Score"+str(grid_search.score(X_test,Y_test)))
print(grid_search.best_params_)
```

```
Train Score0.8571428571428571
TEST Score0.7323943661971831
{'C': 0.01, 'gamma': 0.001, 'kernel': 'linear'}
```

```
means = grid_search.cv_results_['mean_test_score']
stds = grid_search.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, grid_search.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r"% (mean, std * 2, params))
```

```
0.671 (+/-0.017) for {'C': 0.001, 'gamma': 0.001, 'kernel': 'linear'}
0.671 (+/-0.017) for {'C': 0.001, 'gamma': 0.001, 'kernel': 'poly'}
0.671 (+/-0.017) for {'C': 0.001, 'gamma': 0.001, 'kernel': 'rbf'}
0.671 (+/-0.017) for {'C': 0.001, 'gamma': 0.001, 'kernel': 'sigmoid'}
0.671 (+/-0.017) for {'C': 0.001, 'gamma': 0.01, 'kernel': 'linear'}
0.671 (+/-0.017) for {'C': 0.001, 'gamma': 0.01, 'kernel': 'poly'}
0.671 (+/-0.017) for {'C': 0.001, 'gamma': 0.01, 'kernel': 'rbf'}
0.671 (+/-0.017) for {'C': 0.001, 'gamma': 0.01, 'kernel': 'sigmoid'}
0.836 (+/-0.052) for {'C': 0.01, 'gamma': 0.001, 'kernel': 'linear'}
0.671 (+/-0.017) for {'C': 0.01, 'gamma': 0.001, 'kernel': 'poly'}
0.671 (+/-0.017) for {'C': 0.01, 'gamma': 0.001, 'kernel': 'rbf'}
0.671 (+/-0.017) for {'C': 0.01, 'gamma': 0.001, 'kernel': 'sigmoid'}
0.836 (+/-0.052) for {'C': 0.01, 'gamma': 0.01, 'kernel': 'linear'}
0.671 (+/-0.017) for {'C': 0.01, 'gamma': 0.01, 'kernel': 'poly'}
```

```
0.671 (+/-0.017) for {'C': 0.01, 'gamma': 0.01, 'kernel': 'rbf'}
0.671 (+/-0.017) for {'C': 0.01, 'gamma': 0.01, 'kernel': 'sigmoid'}
```

TASK3:PART 2[RANDOM FOREST]

```
# Number of trees in random forest
#n_estimators = [int(x) for x in np.linspace(start = 10, stop = 80, num = 10)]
n_estimators =[10,30]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [2,4]
param_grid = {'n_estimators': n_estimators,'max_features': max_features,'max_depth': max_depth}
rf_Model = RandomForestClassifier()
rf_Grid = GridSearchCV(estimator = rf_Model, param_grid = param_grid, cv = 3, verbose=2, n_jobs = 4)
resulthere=rf_Grid.fit(X_train, Y_train)
rf_Grid.best_params_
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  24 out of  24 | elapsed:    0.7s finished
{'max_depth': 4, 'max_features': 'auto', 'n_estimators': 10}
```

```
print (f'Train Accuracy - : {rf_Grid.score(X_train,Y_train):.3f}')
print (f'Test Accuracy - : {rf_Grid.score(X_test,Y_test):.3f}')
```

```
Train Accuracy - : 0.964
Test Accuracy - : 0.887
```

```
means = resulthere.cv_results_['mean_test_score']
stds = resulthere.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, resulthere.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r"% (mean, std * 2, params))
```

```
0.900 (+/-0.043) for {'max_depth': 2, 'max_features': 'auto', 'n_estimators': 10}
0.893 (+/-0.018) for {'max_depth': 2, 'max_features': 'auto', 'n_estimators': 30}
0.911 (+/-0.036) for {'max_depth': 2, 'max_features': 'sqrt', 'n_estimators': 10}
0.914 (+/-0.052) for {'max_depth': 2, 'max_features': 'sqrt', 'n_estimators': 30}
0.932 (+/-0.055) for {'max_depth': 4, 'max_features': 'auto', 'n_estimators': 10}
0.922 (+/-0.078) for {'max_depth': 4, 'max_features': 'auto', 'n_estimators': 30}
0.918 (+/-0.078) for {'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 10}
0.922 (+/-0.061) for {'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 30}
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, plot_confusion_
target_names = ["Kama(1)", "Rosa(2)", "Canadian(3)"]
disp = plot_confusion_matrix(resulthere, X_test, Y_test, cmap=plt.cm.Blues, )
disp.ax_.set_title("Confusion matrix, without normalization")
plt.show()
```