### **FLOOD MONITORING SYSTEM**

###### 622621121034: NAVEENKUMAR.R

**INTRODUCTION :**

##### A flood monitoring system is used to monitor a rise in water levels. The system comprises sensors that are deployed in cities or any area of interest. The sensors can be connected to either the main electricity or can be solar-powered. These sensors are deployed on bridges, wells, lakes, or beaches to measure water levels in real-time and continuously send data remotely to the centralized data system management via different networks such as GSM, mobile cell networks, or Wi-Fi.roduction:

##### 

**DATA SET:**

In [39]:

*#Import some basic libraries*

import numpy as np

import pandas as pd

# Data Insight

In [40]:

*#Read the data present in dataset*

data = pd.read\_csv('../input/kerela-flood/kerala.csv')

*#Using data.head() we can see the top 5 rows of the dataset*

data.head()

Out[40]:

|  | SUBDIVISION | YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | NOV | DEC | ANNUAL RAINFALL | FLOODS |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | KERALA | 1901 | 28.7 | 44.7 | 51.6 | 160.0 | 174.7 | 824.6 | 743.0 | 357.5 | 197.7 | 266.9 | 350.8 | 48.4 | 3248.6 | YES |
| 1 | KERALA | 1902 | 6.7 | 2.6 | 57.3 | 83.9 | 134.5 | 390.9 | 1205.0 | 315.8 | 491.6 | 358.4 | 158.3 | 121.5 | 3326.6 | YES |
| 2 | KERALA | 1903 | 3.2 | 18.6 | 3.1 | 83.6 | 249.7 | 558.6 | 1022.5 | 420.2 | 341.8 | 354.1 | 157.0 | 59.0 | 3271.2 | YES |
| 3 | KERALA | 1904 | 23.7 | 3.0 | 32.2 | 71.5 | 235.7 | 1098.2 | 725.5 | 351.8 | 222.7 | 328.1 | 33.9 | 3.3 | 3129.7 | YES |
| 4 | KERALA | 1905 | 1.2 | 22.3 | 9.4 | 105.9 | 263.3 | 850.2 | 520.5 | 293.6 | 217.2 | 383.5 | 74.4 | 0.2 | 2741.6 | NO |

In [41]:

*#Now we will cheak if any colomns is left empty*

data.apply(lambda x:sum(x.isnull()), axis=0)

Out[41]:

SUBDIVISION 0

YEAR 0

JAN 0

FEB 0

MAR 0

APR 0

MAY 0

JUN 0

JUL 0

AUG 0

SEP 0

OCT 0

NOV 0

DEC 0

ANNUAL RAINFALL 0

FLOODS 0

dtype: int64

In [42]:

*#We want the data in numbers, therefore we will replace the yes/no in floods coloumn by 1/0*

data['FLOODS'].replace(['YES','NO'],[1,0],inplace=True)

In [43]:

*#Let's see how are data looks like now*

data.head()

Out[43]:

|  | SUBDIVISION | YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | NOV | DEC | ANNUAL RAINFALL | FLOODS |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | KERALA | 1901 | 28.7 | 44.7 | 51.6 | 160.0 | 174.7 | 824.6 | 743.0 | 357.5 | 197.7 | 266.9 | 350.8 | 48.4 | 3248.6 | 1 |
| 1 | KERALA | 1902 | 6.7 | 2.6 | 57.3 | 83.9 | 134.5 | 390.9 | 1205.0 | 315.8 | 491.6 | 358.4 | 158.3 | 121.5 | 3326.6 | 1 |
| 2 | KERALA | 1903 | 3.2 | 18.6 | 3.1 | 83.6 | 249.7 | 558.6 | 1022.5 | 420.2 | 341.8 | 354.1 | 157.0 | 59.0 | 3271.2 | 1 |
| 3 | KERALA | 1904 | 23.7 | 3.0 | 32.2 | 71.5 | 235.7 | 1098.2 | 725.5 | 351.8 | 222.7 | 328.1 | 33.9 | 3.3 | 3129.7 | 1 |
| 4 | KERALA | 1905 | 1.2 | 22.3 | 9.4 | 105.9 | 263.3 | 850.2 | 520.5 | 293.6 | 217.2 | 383.5 | 74.4 | 0.2 | 2741.6 | 0 |

In [44]:

*#Now let's seperate the data which we are gonna use for prediction*

x = data.iloc[:,1:14]

x.head()

Out[44]:

|  | YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | NOV | DEC |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1901 | 28.7 | 44.7 | 51.6 | 160.0 | 174.7 | 824.6 | 743.0 | 357.5 | 197.7 | 266.9 | 350.8 | 48.4 |
| 1 | 1902 | 6.7 | 2.6 | 57.3 | 83.9 | 134.5 | 390.9 | 1205.0 | 315.8 | 491.6 | 358.4 | 158.3 | 121.5 |
| 2 | 1903 | 3.2 | 18.6 | 3.1 | 83.6 | 249.7 | 558.6 | 1022.5 | 420.2 | 341.8 | 354.1 | 157.0 | 59.0 |
| 3 | 1904 | 23.7 | 3.0 | 32.2 | 71.5 | 235.7 | 1098.2 | 725.5 | 351.8 | 222.7 | 328.1 | 33.9 | 3.3 |
| 4 | 1905 | 1.2 | 22.3 | 9.4 | 105.9 | 263.3 | 850.2 | 520.5 | 293.6 | 217.2 | 383.5 | 74.4 | 0.2 |

In [45]:

*#Now seperate the flood label from the dataset*

y = data.iloc[:, -1]

y.head()

Out[45]:

0 1

1 1

2 1

3 1

4 0

Name: FLOODS, dtype: int64

In [46]:

*#Let's see hoe the rainfall index vary during rainy season*

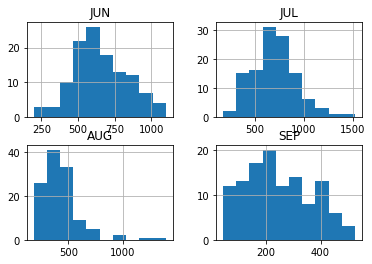
import matplotlib.pyplot as plt

%matplotlib inline

c = data[['JUN','JUL','AUG','SEP']]

c.hist()

plt.show()



In [47]:

*#Data might be widely distributed so let's scale it between 0 and 1*

from sklearn import preprocessing

minmax = preprocessing.MinMaxScaler(feature\_range=(0,1))

minmax.fit(x).transform(x)

Out[47]:

array([[0. , 0.34371257, 0.56582278, ..., 0.39727673, 0.95570189,

0.2388724 ],

[0.00854701, 0.08023952, 0.03291139, ..., 0.5804966 , 0.37952709,

0.60039565],

[0.01709402, 0.03832335, 0.23544304, ..., 0.57188626, 0.37563604,

0.29129575],

...,

[0.98290598, 0.02874251, 0.04810127, ..., 0.31517821, 0.28105358,

0.11622156],

[0.99145299, 0.02275449, 0.08607595, ..., 0.24809772, 0.18258007,

0.18793274],

[1. , 0.34850299, 0.65949367, ..., 0.57589107, 0.28105358,

0.3214639 ]])

In [48]:

*#Let's divide the dataset into 2 sets:train and test in ratio (4:1)*

from sklearn import model\_selection,neighbors

from sklearn.model\_selection import train\_test\_split

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2)

In [49]:

*#Let's see how our train set looks like*

x\_train.head()

Out[49]:

|  | YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | NOV | DEC |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 37 | 1938 | 0.3 | 79.0 | 53.3 | 164.5 | 179.6 | 681.6 | 648.6 | 287.9 | 223.2 | 223.7 | 69.5 | 22.9 |
| 94 | 1995 | 10.3 | 6.5 | 37.3 | 134.9 | 355.6 | 493.4 | 702.5 | 457.3 | 280.0 | 198.3 | 182.6 | 0.1 |
| 58 | 1959 | 3.0 | 21.4 | 6.3 | 150.7 | 347.2 | 872.8 | 1155.7 | 397.3 | 405.5 | 200.4 | 151.9 | 34.0 |
| 91 | 1992 | 2.4 | 0.9 | 0.1 | 43.0 | 218.4 | 819.3 | 767.8 | 508.0 | 297.5 | 290.7 | 287.6 | 3.7 |
| 55 | 1956 | 7.9 | 11.7 | 15.1 | 151.6 | 351.3 | 755.4 | 466.8 | 319.5 | 178.4 | 353.3 | 178.2 | 9.1 |

In [50]:

y\_train.head()

Out[50]:

37 0

94 0

58 1

91 1

55 0

Name: FLOODS, dtype: int64

# Prediction Algorithms:

# 1. KNN Classifier

In [51]:

clf = neighbors.KNeighborsClassifier()

knn\_clf = clf.fit(x\_train,y\_train)

In [52]:

*#Let's predict chances of flood*

y\_predict = knn\_clf.predict(x\_test)

print('predicted chances of flood')

print(y\_predict)

predicted chances of flood

[1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 1 1 1 0 0 0 1 0]

In [53]:

*#Actual chances of flood*

print("actual values of floods:")

print(y\_test)

actual values of floods:

70 1

46 1

102 0

9 0

116 0

92 0

42 1

12 0

14 1

72 0

34 0

17 0

78 0

86 0

109 1

15 1

1 1

108 0

22 1

89 0

65 0

5 0

24 1

85 0

Name: FLOODS, dtype: int64

In [54]:

from sklearn.model\_selection import cross\_val\_score

In [55]:

knn\_accuracy = cross\_val\_score(knn\_clf,x\_test,y\_test,cv=3,scoring='accuracy',n\_jobs=-1)

In [56]:

knn\_accuracy.mean()

Out[56]:

0.7083333333333334

# 2. Logistic Regression

In [57]:

x\_train\_std = minmax.fit\_transform(x\_train)

x\_test\_std = minmax.transform(x\_test)

In [58]:

from sklearn.model\_selection import cross\_val\_score

from sklearn.linear\_model import LogisticRegression

lr = LogisticRegression()

lr\_clf = lr.fit(x\_train\_std,y\_train)

lr\_accuracy = cross\_val\_score(lr\_clf,x\_test\_std,y\_test,cv=3,scoring='accuracy',n\_jobs=-1)

In [59]:

lr\_accuracy.mean()

Out[59]:

0.625

In [60]:

y\_predict = lr\_clf.predict(x\_test\_std)

print('Predicted chances of flood')

print(y\_predict)

Predicted chances of flood

[1 0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 1 0 0 0 1 0]

In [61]:

print('Actual chances of flood')

print(y\_test.values)

Actual chances of flood

[1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 1 1 0 1 0 0 0 1 0]

In [62]:

from sklearn.metrics import accuracy\_score,recall\_score,roc\_auc\_score,confusion\_matrix

print("**\n**accuracy score: **%f**"%(accuracy\_score(y\_test,y\_predict)\*100))

print("recall score: **%f**"%(recall\_score(y\_test,y\_predict)\*100))

print("roc score: **%f**"%(roc\_auc\_score(y\_test,y\_predict)\*100))

accuracy score: 83.333333

recall score: 88.888889

roc score: 84.444444

# 3. Decision tree classification

In [63]:

from sklearn.tree import DecisionTreeClassifier

dtc\_clf = DecisionTreeClassifier()

dtc\_clf.fit(x\_train,y\_train)

dtc\_clf\_acc = cross\_val\_score(dtc\_clf,x\_train\_std,y\_train,cv=3,scoring="accuracy",n\_jobs=-1)

dtc\_clf\_acc

Out[63]:

array([0.71875 , 0.64516129, 0.61290323])

In [64]:

*#Predicted flood chances*

y\_pred = dtc\_clf.predict(x\_test)

print(y\_pred)

[1 1 0 0 0 0 1 0 1 0 0 1 1 1 1 0 0 1 1 0 1 0 1 0]

In [65]:

*#Actual flood chances*

print("actual values:")

print(y\_test.values)

actual values:

[1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 1 1 0 1 0 0 0 1 0]

In [66]:

from sklearn.metrics import accuracy\_score,recall\_score,roc\_auc\_score,confusion\_matrix

print("**\n**accuracy score:**%f**"%(accuracy\_score(y\_test,y\_pred)\*100))

print("recall score:**%f**"%(recall\_score(y\_test,y\_pred)\*100))

print("roc score:**%f**"%(roc\_auc\_score(y\_test,y\_pred)\*100))

accuracy score:70.833333

recall score:77.777778

roc score:72.222222

# 4. Random Forest Classification

In [67]:

from sklearn.ensemble import RandomForestClassifier

rmf = RandomForestClassifier(max\_depth=3,random\_state=0)

rmf\_clf = rmf.fit(x\_train,y\_train)

rmf\_clf

Out[67]:

RandomForestClassifier(max\_depth=3, random\_state=0)

In [68]:

rmf\_clf\_acc = cross\_val\_score(rmf\_clf,x\_train\_std,y\_train,cv=3,scoring="accuracy",n\_jobs=-1)

*#rmf\_proba = cross\_val\_predict(rmf\_clf,x\_train\_std,y\_train,cv=3,method='predict\_proba')*

In [69]:

rmf\_clf\_acc

Out[69]:

array([0.8125 , 0.67741935, 0.87096774])

In [70]:

y\_pred = rmf\_clf.predict(x\_test)

In [71]:

from sklearn.metrics import accuracy\_score,recall\_score,roc\_auc\_score,confusion\_matrix

print("**\n**accuracy score:**%f**"%(accuracy\_score(y\_test,y\_pred)\*100))

print("recall score:**%f**"%(recall\_score(y\_test,y\_pred)\*100))

print("roc score:**%f**"%(roc\_auc\_score(y\_test,y\_pred)\*100))

accuracy score:79.166667

recall score:100.000000

roc score:83.333333

# 5. Enseble Learning

In [72]:

from sklearn.ensemble import VotingClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

log\_clf = LogisticRegression(solver="liblinear", random\_state=42)

rnd\_clf = RandomForestClassifier(n\_estimators=10, random\_state=42)

knn\_clf = KNeighborsClassifier()

voting = VotingClassifier(

estimators=[('lr', log\_clf), ('rf', rnd\_clf), ('knn', knn\_clf)],

voting='hard')

In [73]:

voting\_clf = voting.fit(x\_train, y\_train)

In [74]:

from sklearn.metrics import accuracy\_score

for clf **in** (log\_clf, rnd\_clf, knn\_clf, voting\_clf):

clf.fit(x\_train, y\_train)

y\_pred = clf.predict(x\_test)

print(clf.\_\_class\_\_.\_\_name\_\_, accuracy\_score(y\_test, y\_pred))

LogisticRegression 0.9583333333333334

RandomForestClassifier 0.7083333333333334

KNeighborsClassifier 0.9166666666666666

VotingClassifier 0.9166666666666666

# Comparing all the prediction models

In [75]:

models = []

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import VotingClassifier

models.append(('KNN', KNeighborsClassifier()))

models.append(('LR', LogisticRegression()))

models.append(('DT', DecisionTreeClassifier()))

models.append(('RF', RandomForestClassifier()))

models.append(('EL', VotingClassifier(

estimators=[('lr', log\_clf), ('rf', rnd\_clf), ('knn', knn\_clf)],

voting='hard')))

names = []

scores = []

for name, model **in** models:

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

scores.append(accuracy\_score(y\_test, y\_pred))

names.append(name)

tr\_split = pd.DataFrame({'Name': names, 'Score': scores})

print(tr\_split)

Name Score

0 KNN 0.916667

1 LR 0.958333

2 DT 0.750000

3 RF 0.916667

4 EL 0.916667

/opt/conda/lib/python3.7/site-packages/sklearn/linear\_model/\_logistic.py:764: ConvergenceWarning: lbfgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

https://scikit-learn.org/stable/modules/preprocessing.html

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression

extra\_warning\_msg=\_LOGISTIC\_SOLVER\_CONVERGENCE\_MSG)

In [76]:

import seaborn as sns

axis = sns.barplot(x = 'Name', y = 'Score', data =tr\_split )

axis.set(xlabel='Classifier', ylabel='Accuracy')

for p **in** axis.patches:

height = p.get\_height()

axis.text(p.get\_x() + p.get\_width()/2, height + 0.005, '**{:1.4f}**'.format(height), ha="center")

plt.show()

