```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
print("Success")
```

```
Success
```

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473

Enter your authorization code:
··········
Mounted at /content/drive
```

```
"------------------reading the data set and setting the headers------------------"

data_bcw = "/content/drive/My Drive/BCW.csv"
data = pd.read_csv(data_bcw, header=None)
data.columns = ["ID","A1","A2","A3","A4","A5","A6","A7","A8","A9","C"]
#data.head(24)
```

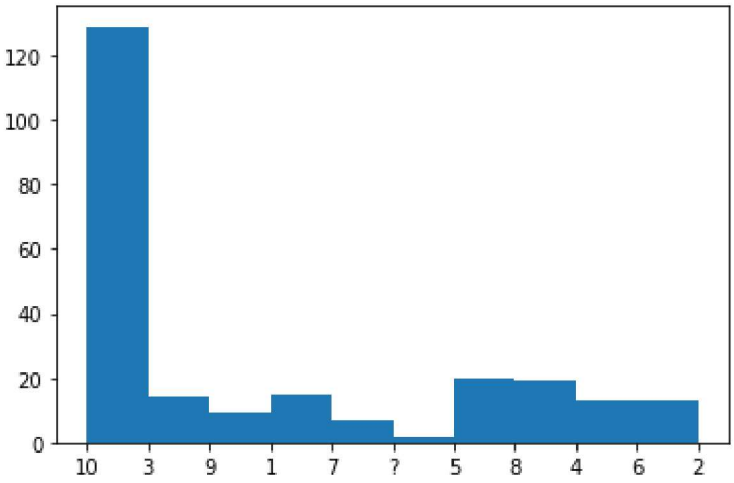
```
data.head(24)
```

	ID	A1	A2	A3	A4	A5	A6	A7	A8	A9	C
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2
5	1017122	8	10	10	8	7	10	9	7	1	4
6	1018099	1	1	1	1	2	10	3	1	1	2
7	1018561	2	1	2	1	2	1	3	1	1	2
8	1033078	2	1	1	1	2	1	1	1	5	2
9	1033078	4	2	1	1	2	1	2	1	1	2
10	1035283	1	1	1	1	1	1	3	1	1	2
11	1036172	2	1	1	1	2	1	2	1	1	2
12	1041801	5	3	3	3	2	3	4	4	1	4
13	1043999	1	1	1	1	2	3	3	1	1	2
14	1044572	8	7	5	10	7	9	5	5	4	4
15	1047630	7	4	6	4	6	1	4	3	1	4
16	1048672	4	1	1	1	2	1	2	1	1	2
17	1049815	4	1	1	1	2	1	3	1	1	2
18	1050670	10	7	7	6	4	10	4	1	2	4
19	1050718	6	1	1	1	2	1	3	1	1	2
20	1054590	7	3	2	10	5	10	5	4	4	4
21	1054593	10	5	5	3	6	7	7	10	1	4
22	1056784	3	1	1	1	2	1	2	1	1	2
23	1057013	8	4	5	1	2	?	7	3	1	4

```
"----------------Class value = 2 implies Beingn-----------------"
"--------------Class value = 4 implies Malignant-------------"

B=data[data["C"]==2]
M=data[data["C"]==4]
```

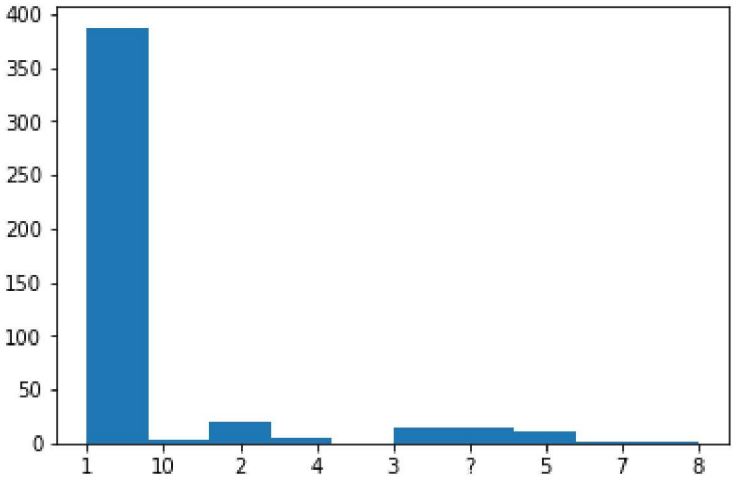
```
"------------------There are missing values in the attribute A6----------------"
"--------------------Plotting Histograms for both the classes------------------"
"-----------------Replace missing values with mode of the class-------------------"

plt.hist(M["A6"],bins=10)
```

```
(array([129.,  14.,   9.,  15.,   7.,   2.,  20.,  19.,  13.,  13.]),
 array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.]),
 <a list of 10 Patch objects>)
```




```
plt.hist(B["A6"],bins=10)
```

```
(array([387.,   3.,  21.,   6.,   0.,  14.,  14.,  10.,   1.,   2.]),
 array([0. , 0.8, 1.6, 2.4, 3.2, 4. , 4.8, 5.6, 6.4, 7.2, 8. ]),
 <a list of 10 Patch objects>)
```




```
"-------------------------Marking the class---------------------"
"---------------------------2 (Benign) as 1 --------------------------"
"------------------------ 4 (Malignant) as -1-----------------------"

Y=data["C"]
print(Y)
for i in range(len(Y)):
  if(Y.iloc[i]==2):
    Y.iloc[i]=1
  else:
    Y.iloc[i]=-1
print(Y)
```

```
0      2
1      2
2      2
3      2
4      2
      ..
694    2
695    2
696    4
697    4
698    4
Name: C, Length: 699, dtype: int64
0      1
1      1
2      1
3      1
4      1
      ..
694    1
695    1
696   -1
697   -1
698   -1
Name: C, Length: 699, dtype: int64
/usr/local/lib/python3.6/dist-packages/pandas/core/indexing.py:205: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/use
  self._setitem_with_indexer(indexer, value)
```

```
"------------replacing the missing values in A6 with respective mode of the corresponding

for i in range (0,len(data)):
  if(data.iloc[i,6]=='?'):
    if(data.iloc[i,9]==2):
      data.iloc[i,6]='1'
    else:
      data.iloc[i,6]='10'
data.dtypes
```

```
ID     int64
A1     int64
A2     int64
A3     int64
A4     int64
A5     int64
A6    object
A7     int64
A8     int64
A9     int64
C      int64
dtype: object
```

```
"----------------------data type conversion of A6 values from object to int64-------------

data['A6']=data["A6"].astype(str).astype(int)
data.dtypes
```

```
ID    int64
A1    int64
A2    int64
A3    int64
A4    int64
A5    int64
A6    int64
A7    int64
A8    int64
A9    int64
C     int64
dtype: object
```

```
"-------------------------dropping the last column from the set to build X---------------
"-----------------------------dropping the feature of ID from the data-------------------


X=data.drop(["ID"],axis=1)
X=X.iloc[:,:-1].values
print(X)
```

```
[[ 5  1  1 ...  3  1  1]
 [ 5  4  4 ...  3  2  1]
 [ 3  1  1 ...  3  1  1]
 ...
 [ 5 10 10 ...  8 10  2]
 [ 4  8  6 ... 10  6  1]
 [ 4  8  8 ... 10  4  1]]
```

```
"------------------test-train split function-------------------------"
"------------top fraction as train and lower as test----------"

def test_train_part(X,Y,frac):

  train_size=int(len(X)*frac)

  X_train=np.zeros((train_size,9))
  X_test=np.zeros((len(X)-train_size,9))
  Y_train=np.zeros((train_size,1))
  Y_test=np.zeros((len(X)-train_size,1))

  j=0
  while j < int(train_size):
    X_train[j]=X[j]
    Y_train[j]=Y[j]
    if(j< int(len(X)-train_size)):
      X_test[j]=X[len(X)-1-j]
      Y_test[j]=Y[len(X)-1-j]
    j=j+1

  return X_train,X_test,Y_train,Y_test
```

```
"-----------------------------Linear SVM Training----------------------------"

def svm_train(X,Y,epochs):
```

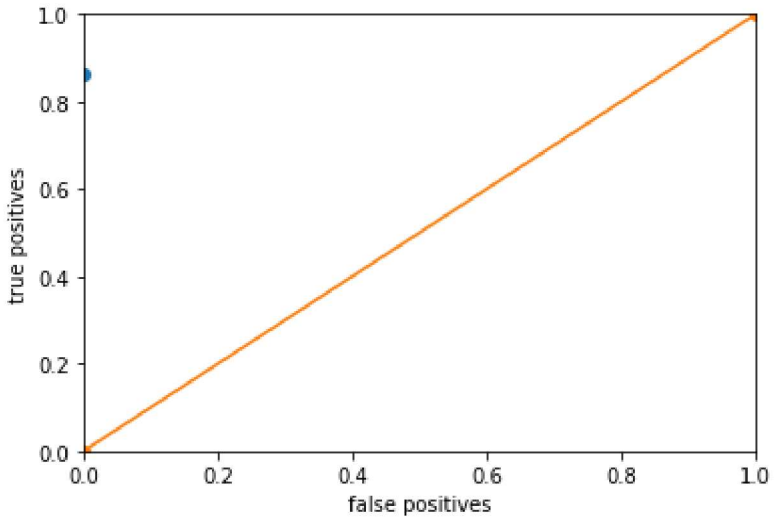
```
def svm_train(X,Y,epochs):

  w=np.zeros(len(X[0]),dtype=int)
  alpha = 0.08

  for e in range(0,epochs):
    error = 0
    for i, x in enumerate(X):
      if(Y[i]*np.dot(X[i],w) < 1):
        w = w + alpha * ((X[i]*Y[i])+(-2*(1/epochs)*w))
      else:
        w = w + alpha * (-2 * (1/epochs) * w)
  return w
```

```
"------------------------------Test Run-------------------------"

def svm_test(X,w1,Y):
  i=0
  Yp=np.zeros((len(X),1))
  error=0
  tn=0
  tp=0
  fp=0
  fn=0

  while i < len(X):
    Yp[i]=np.dot(X[i],w1)
    if Yp[i]<-1:
      Yp[i]=-1
      if(Y[i]==-1):
        tn=tn+1
      elif(Y[i]==1):
        fn=fn+1

    else:
      Yp[i]=1
      if(Y[i]==1):
        tp=tp+1
      elif(Y[i]==-1):
        fp=fp+1
    if(Y[i]!=Yp[i]):
      error+=1
    i=i+1
  predict=(error/len(Y))*100
  accuracy=100-predict
  return accuracy,tn/len(X),tp/len(X),fp/len(X),fn/len(X)
```

```
"------------------------------Tuned parameter W1-------------------------------"
"-----------------------------------Accuracy-------------------------------------"


X_train,X_test,Y_train,Y_test=test_train_part(X,Y,0.7)

w1 = svm_train(X_train,Y_train,epochs=1000)
print(w1)
```

```
print(w1)
ac,tn,tp,fp,fn = svm_test(X_test,w1,Y_test)
print('accuracy =',ac)
print('true negative=',tn)
print('true positive=',tp)
print('false negative=',fn)
print('false positive=',fp)

plt.xlabel('false positives')
plt.ylabel('true positives')
plt.xlim(0,1)
plt.ylim(0,1)
plt.plot(fp/(tn+fp),tp/(tp+fn),'-o')
plt.plot([0,1],[0,1],'-o')
plt.show()
```

```
[ 1.03522802 -3.51768025 -0.87700068 -1.21000387  3.06409758 -1.24100392
  0.11266515 -0.2984383   1.30905927]
accuracy = 89.52380952380952
true negative= 0.22380952380952382
true positive= 0.6714285714285714
false negative= 0.10476190476190476
false positive= 0.0
```




```
def conf_mat(tneg,tpos,fneg,fpos,k):
  CF=np.zeros((2,2))
  CF[0][0]=tpos/k
  CF[0][1]=fneg/k
  CF[1][0]=fpos/k
  CF[1][1]=tneg/k

  return CF
```

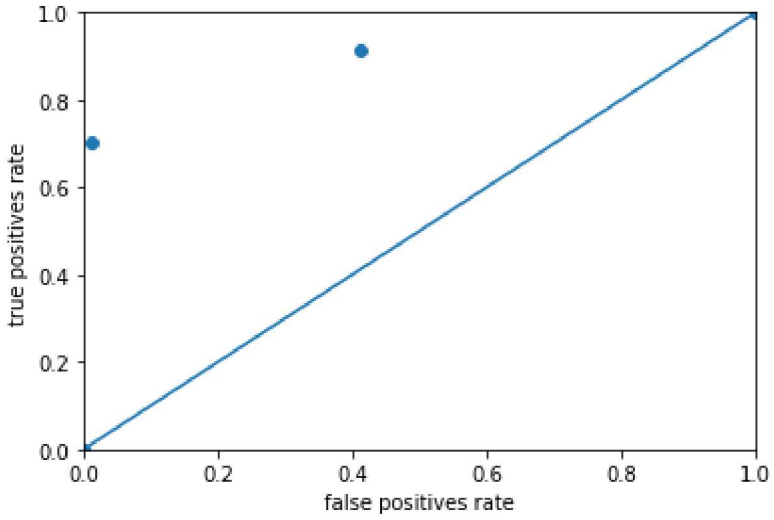
```
"----------------------------K-cross fold validation--------------------------"
"------ A function is defined with the Feature matrix and Classes as parameters-----------
"----- 'k' can be passed to the function or can be taken as input ------------------"
"----------------- The functions returns the final accuracy -----------------"

def k_fold_val(X,Y,k):

  #k=int(input('Enter vakue of k:'))
```

```
subsize=int(len(X)/k)

ac_tot=0
tpos=np.zeros((k,1))
fpos=np.zeros((k,1))
tneg=np.zeros((k,1))
fneg=np.zeros((k,1))
fpr=np.zeros((k,1))
tpr=np.zeros((k,1))

X_train=np.zeros((len(X)-subsize,9))
X_test=np.zeros((subsize,9))
Y_train=np.zeros((len(X)-subsize,1))
Y_test=np.zeros((subsize,1))

tne=0
fne=0
fpo=0
tpo=0

for i in range (0,k):
  j=0
  s=0

  for j in range (0,subsize):
    X_test[0+j]=X[i*subsize+j]
    Y_test[0+j]=Y[i*subsize+j]

  m=subsize*(i+1)

  for j in range (0,len(X)-subsize):
    if(m+j < len(X)):
      X_train[j]=X[m+j]
      Y_train[j]=Y[m+j]
      continue
    else:
      X_train[j]=X[s]
      Y_train[j]=Y[s]
      s=s+1
      continue
  w1=svm_train(X_train,Y_train,epochs=1000)
  accuracy=0
  accuracy,tn,tp,fp,fn=svm_test(X_test,w1,Y_test)
  print('accuracy ',i+1,'=',accuracy)

  tneg[i]=tn
  tpos[i]=tp
  fneg[i]=fn
  fpos[i]=fp

  fpr[i]=fp/(tn+fp)
  tpr[i]=tp/(tp+fn)

  tne=tne+tn
  tpo=tpo+tp
```

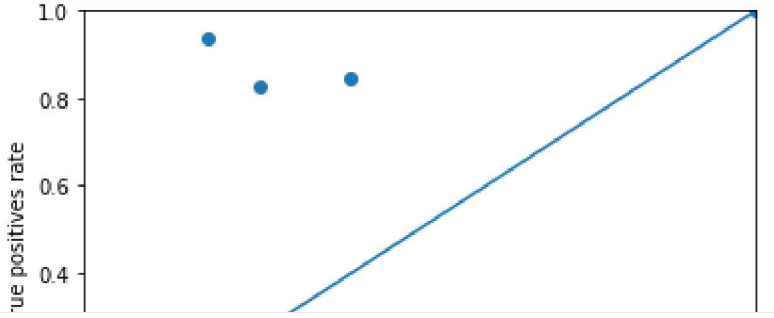
```
    tpo=tpo+tp
    fne=fne+fn
    fpo=fpo+fp

    CF=conf_mat(tn,tp,fn,fp,1)
    print('Confusion Matrix :\n',CF)

    ac_tot=ac_tot+accuracy

  CF=conf_mat(tne,tpo,fne,fpo,k)
  print('Confusion Matrix :\n',CF)

  recall=CF[0][0]/(CF[0][0]+CF[0][1])
  precision=CF[0][0]/(CF[0][0]+CF[1][0])
  f1=(2*CF[0][0])/((2*CF[0][0])+CF[0][1]+CF[1][0])

  print('Recall: ',recall,'\nPrecision: ',precision,'\nF1 Score: ',f1,'\n')

  plt.xlabel('false positives rate')
  plt.ylabel('true positives rate')
  plt.xlim(0,1)
  plt.ylim(0,1)
  plt.scatter(fpr,tpr,marker='o')
  plt.plot([0,1],[0,1],'-o')
  plt.show()

  return ac_tot/k
```

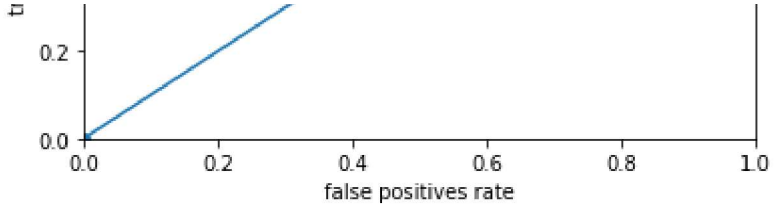
```
"--------------------------------Avereage accuracy for different k values-----------------

ac_avg=0
av=np.zeros((9,1))
kol=np.zeros((9,1))

for i in range (0,9):
  ac_avg=k_fold_val(X,Y,i+2)
  av[i]=ac_avg
  kol[i]=i+2
  print('final accuracy with k =',i+2,':',ac_avg,'\n')
```

```
accuracy  1 = 76.79083094555874
Confusion Matrix :
 [[0.50143266 0.04584527]
 [0.18624642 0.26647564]]
accuracy  2 = 77.07736389684814
Confusion Matrix :
 [[0.53868195 0.22636103]
 [0.00286533 0.23209169]]
Confusion Matrix :
 [[0.52005731 0.13610315]
 [0.09455587 0.24928367]]
Recall:  0.7925764192139737
Precision:  0.8461538461538461
F1 Score:  0.8184892897406989
```

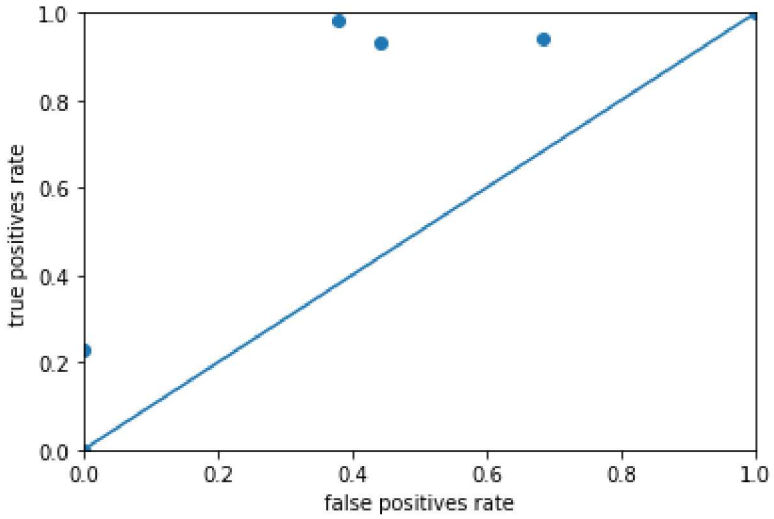



```
final accuracy with k = 2 : 76.93409742120343

accuracy  1 = 73.81974248927038
Confusion Matrix :
 [[0.472103   0.08583691]
 [0.17596567 0.26609442]]
accuracy  2 = 79.39914163090128
Confusion Matrix :
 [[0.527897   0.11158798]
 [0.0944206  0.26609442]]
accuracy  3 = 90.98712446351931
Confusion Matrix :
 [[0.72103004 0.0472103 ]
 [0.04291845 0.1888412 ]]
Confusion Matrix :
 [[0.57367668 0.08154506]
 [0.10443491 0.24034335]]
Recall:  0.8755458515283843
Precision:  0.8459915611814346
F1 Score:  0.8605150214592274
```



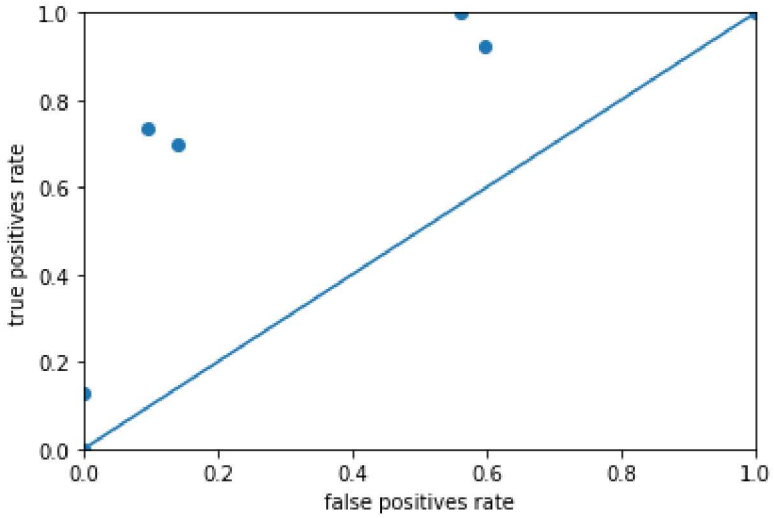



```
final accuracy with k = 3 : 81.40200286123032

accuracy  1 = 67.816091954023
Confusion Matrix :
 [[0.54597701 0.03448276]
 [0.28735632 0.13218391]]
accuracy  2 = 75.28735632183908
Confusion Matrix :
 [[0.48275862 0.03448276]
 [0.21264368 0.27011494]]
accuracy  3 = 42.52873563218391
Confusion Matrix :
 [[0.17241379 0.57471264]
 [0.         0.25287356]]
accuracy  4 = 90.80459770114942
Confusion Matrix :
 [[0.77586207 0.01149425]
 [0.08045977 0.13218391]]
Confusion Matrix :
 [[0.49425287 0.1637931 ]
 [0.14511494 0.19683908]]
Recall:  0.7510917030567685
Precision:  0.7730337078651687
F1 Score:  0.761904761904762
```

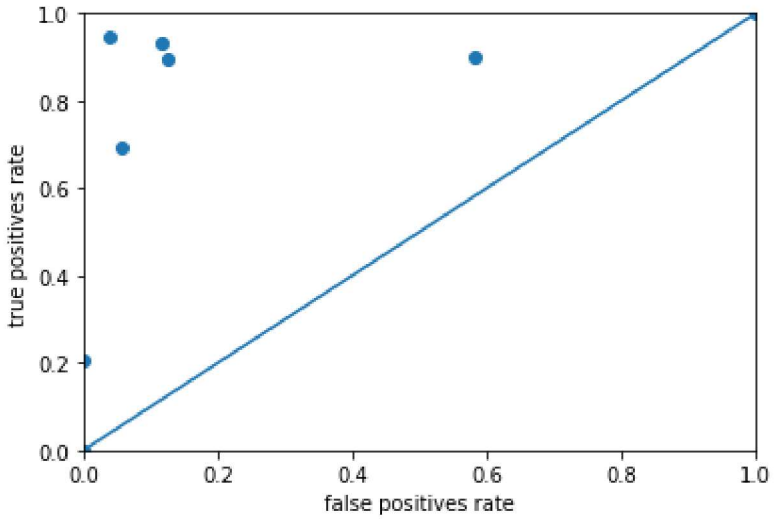



```
final accuracy with k = 4 : 69.10919540229885

accuracy  1 = 69.06474820143885
Confusion Matrix :
 [[0.51079137 0.04316547]
 [0.26618705 0.17985612]]
accuracy  2 = 81.29496402877697
Confusion Matrix :
 [[0.4028777  0.14388489]
 [0.04316547 0.41007194]]
accuracy  3 = 75.53956834532374
Confusion Matrix :
 [[0.44604317 0.1942446 ]
 [0.05035971 0.30935252]]
accuracy  4 = 32.37410071942446
```

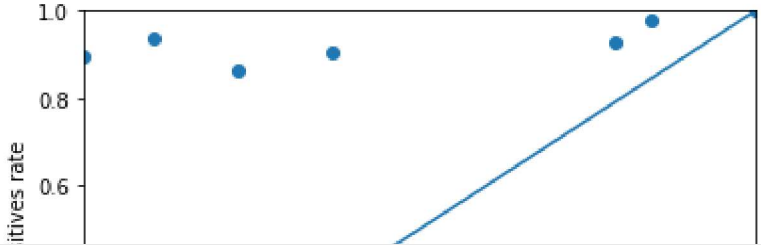
```
Confusion Matrix :
 [[0.10071942 0.67625899]
 [0.         0.22302158]]
accuracy  5 = 87.05035971223022
Confusion Matrix :
 [[0.76978417 0.        ]
 [0.1294964  0.10071942]]
Confusion Matrix :
 [[0.44604317 0.21151079]
 [0.09784173 0.24460432]]
Recall:  0.6783369803063458
Precision:  0.8201058201058201
F1 Score:  0.7425149700598802
```

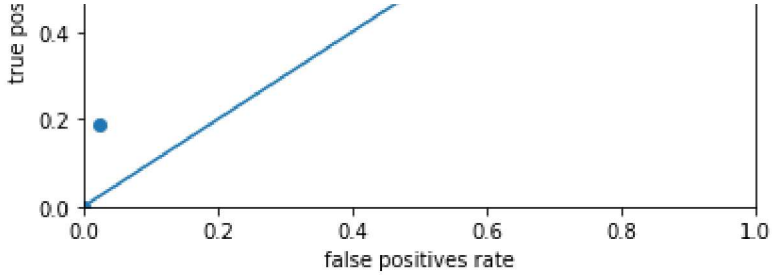



```
final accuracy with k = 5 : 69.06474820143885

accuracy  1 = 67.24137931034483
Confusion Matrix :
 [[0.47413793 0.05172414]
 [0.27586207 0.19827586]]
accuracy  2 = 88.79310344827586
Confusion Matrix :
 [[0.52586207 0.06034483]
 [0.05172414 0.36206897]]
accuracy  3 = 81.0344827586207
Confusion Matrix :
 [[0.37068966 0.1637931 ]
 [0.02586207 0.43965517]]
accuracy  4 = 40.51724137931034
Confusion Matrix :
 [[0.15517241 0.59482759]
 [0.         0.25      ]]
accuracy  5 = 92.24137931034483
Confusion Matrix :
 [[0.72413793 0.05172414]
 [0.02586207 0.19827586]]
accuracy  6 = 94.82758620689656
Confusion Matrix :
 [[0.73275862 0.04310345]
 [0.00862069 0.21551724]]
Confusion Matrix :
 [[0.49712644 0.16091954]
 [0.06465517 0.27729885]]
Recall:  0.7554585152838429
Precision:  0.8849104859335039
F1 Score:  0.8150765606595995
```

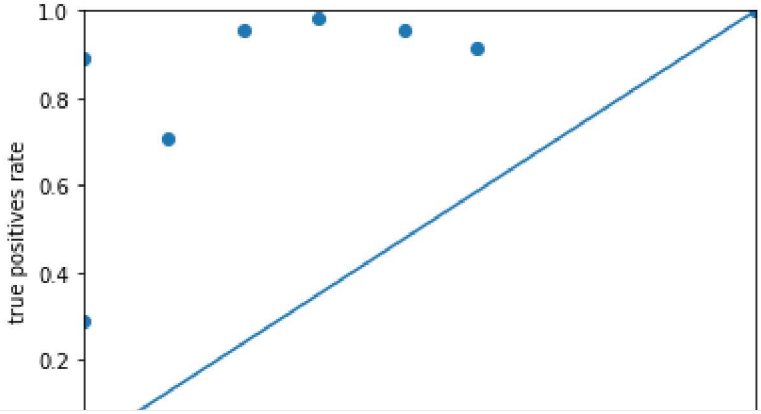


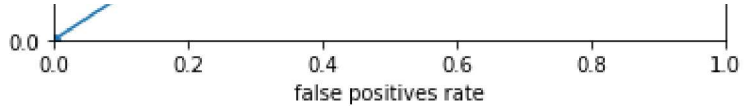

```
final accuracy with k = 6 : 77.4425287356322

accuracy  1 = 61.61616161616162
Confusion Matrix :
 [[0.52525253 0.04040404]
 [0.34343434 0.09090909]]
accuracy  2 = 51.515151515151516
Confusion Matrix :
 [[0.11111111 0.47474747]
 [0.01010101 0.4040404 ]]
accuracy  3 = 54.54545454545455
Confusion Matrix :
 [[0.46464646 0.01010101]
 [0.44444444 0.08080808]]
accuracy  4 = 80.8080808080808
Confusion Matrix :
 [[0.58585859 0.06060606]
 [0.13131313 0.22222222]]
accuracy  5 = 83.83838383838383
Confusion Matrix :
 [[0.63636364 0.1010101 ]
 [0.06060606 0.2020202 ]]
accuracy  6 = 91.91919191919192
Confusion Matrix :
 [[0.6969697  0.08080808]
 [0.         0.22222222]]
accuracy  7 = 92.92929292929293
Confusion Matrix :
 [[0.75757576 0.05050505]
 [0.02020202 0.17171717]]
Confusion Matrix :
 [[0.53968254 0.11688312]
 [0.14430014 0.1991342 ]]
Recall:  0.8219780219780219
Precision:  0.7890295358649789
F1 Score:  0.8051668460710442
```



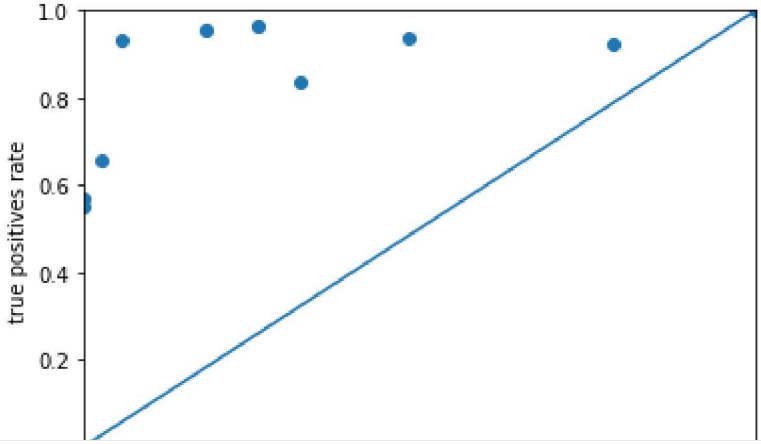


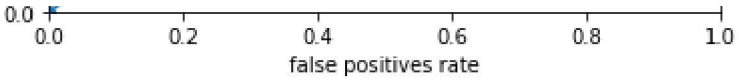

```
final accuracy with k = 7 : 73.88167388167389

accuracy  1 = 67.816091954023
Confusion Matrix :
 [[0.48275862 0.04597701]
 [0.27586207 0.1954023 ]]
accuracy  2 = 77.01149425287356
Confusion Matrix :
 [[0.44827586 0.18390805]
 [0.04597701 0.32183908]]
accuracy  3 = 74.71264367816092
Confusion Matrix :
 [[0.49425287 0.02298851]
 [0.22988506 0.25287356]]
accuracy  4 = 86.20689655172414
Confusion Matrix :
 [[0.49425287 0.02298851]
 [0.11494253 0.36781609]]
accuracy  5 = 45.97701149425287
Confusion Matrix :
 [[0.2183908  0.54022989]
 [0.         0.24137931]]
accuracy  6 = 73.5632183908046
Confusion Matrix :
 [[0.73563218 0.        ]
 [0.26436782 0.        ]]
accuracy  7 = 89.65517241379311
Confusion Matrix :
 [[0.72413793 0.01149425]
 [0.09195402 0.17241379]]
accuracy  8 = 90.80459770114942
Confusion Matrix :
 [[0.74712644 0.09195402]
 [0.         0.16091954]]
Confusion Matrix :
 [[0.54310345 0.11494253]
 [0.12787356 0.21408046]]
Recall:  0.8253275109170306
Precision:  0.8094218415417558
F1 Score:  0.8172972972972973
```



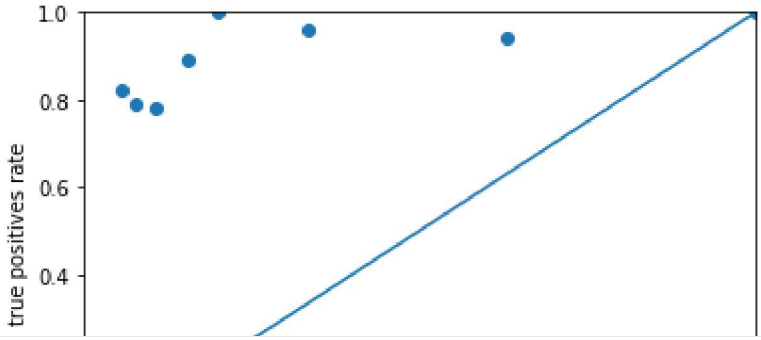


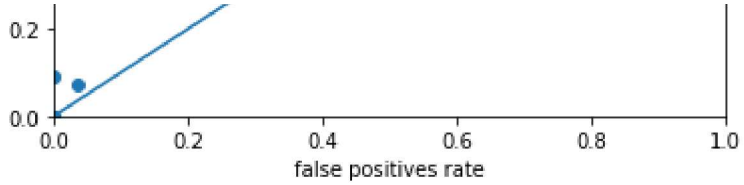

```
final accuracy with k = 8 : 75.7183908045977

accuracy  1 = 57.142857142857146
Confusion Matrix :
 [[0.46753247 0.03896104]
 [0.38961039 0.1038961 ]]
accuracy  2 = 77.92207792207792
Confusion Matrix :
 [[0.58441558 0.03896104]
 [0.18181818 0.19480519]]
accuracy  3 = 76.62337662337663
Confusion Matrix :
 [[0.31168831 0.23376623]
 [0.         0.45454545]]
accuracy  4 = 81.81818181818181
Confusion Matrix :
 [[0.32467532 0.16883117]
 [0.01298701 0.49350649]]
accuracy  5 = 77.92207792207792
Confusion Matrix :
 [[0.53246753 0.1038961 ]
 [0.11688312 0.24675325]]
accuracy  6 = 64.93506493506493
Confusion Matrix :
 [[0.42857143 0.35064935]
 [0.         0.22077922]]
accuracy  7 = 93.50649350649351
Confusion Matrix :
 [[0.71428571 0.05194805]
 [0.01298701 0.22077922]]
accuracy  8 = 89.6103896103896
Confusion Matrix :
 [[0.67532468 0.02597403]
 [0.07792208 0.22077922]]
accuracy  9 = 93.50649350649351
Confusion Matrix :
 [[0.81818182 0.03896104]
 [0.02597403 0.11688312]]
Confusion Matrix :
 [[0.53968254 0.11688312]
 [0.09090909 0.25252525]]
Recall:  0.821978021978022
Precision:  0.8558352402745996
F1 Score:  0.8385650224215248
```





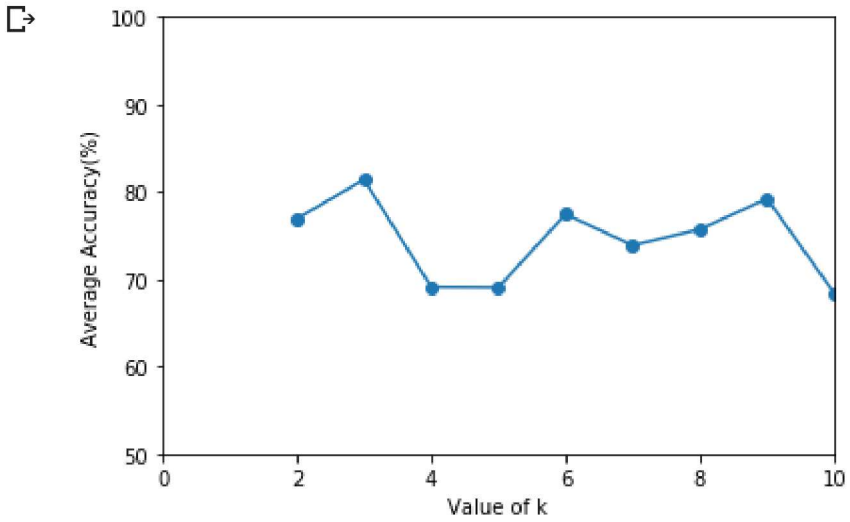

```
final accuracy with k = 9 : 79.2207792207792

accuracy  1 = 65.21739130434783
Confusion Matrix :
 [[0.46376812 0.02898551]
 [0.31884058 0.1884058 ]]
accuracy  2 = 42.028985507246375
Confusion Matrix :
 [[0.04347826 0.56521739]
 [0.01449275 0.37681159]]
accuracy  3 = 84.05797101449275
Confusion Matrix :
 [[0.49275362 0.13043478]
 [0.02898551 0.34782609]]
accuracy  4 = 84.05797101449275
Confusion Matrix :
 [[0.36231884 0.10144928]
 [0.05797101 0.47826087]]
accuracy  5 = 86.95652173913044
Confusion Matrix :
 [[0.47826087 0.05797101]
 [0.07246377 0.39130435]]
accuracy  6 = 23.188405797101453
Confusion Matrix :
 [[0.         0.76811594]
 [0.         0.23188406]]
accuracy  7 = 85.5072463768116
Confusion Matrix :
 [[0.60869565 0.13043478]
 [0.01449275 0.24637681]]
accuracy  8 = 28.985507246376812
Confusion Matrix :
 [[0.07246377 0.71014493]
 [0.         0.2173913 ]]
accuracy  9 = 86.95652173913044
Confusion Matrix :
 [[0.66666667 0.02898551]
 [0.10144928 0.20289855]]
accuracy  10 = 97.10144927536231
Confusion Matrix :
 [[0.85507246 0.        ]
 [0.02898551 0.11594203]]
Confusion Matrix :
 [[0.40434783 0.25217391]
 [0.06376812 0.27971014]]
Recall:  0.6158940397350994
Precision:  0.8637770897832817
F1 Score:  0.7190721649484536
```






```
final accuracy with k = 10 : 68.40579710144928
```

```
"--------plot of average accuracy for differnt values of k----------------------"
"---------------------k-cross validation method----------------------"
"-----------------k ranging in the interval of 2-10-------------------------"

plt.xlabel('Value of k')
plt.ylabel('Average Accuracy(%)')
plt.xlim(0,10)
plt.ylim(50,100)
plt.plot(kol,av,'-o')
plt.show()
```




```
"----------------------------Boot Strap Method--------------------------"
"------------Sampling randomly with replacment N times, N=len(data)-----------------------


def bootstrap(X):
  X_sam=np.zeros((len(data),9))
  Y_sam=np.zeros((len(data),1))

  for i in range (0,len(data)):
      X_rep=data.sample(n=1,replace=True)
      X_rep=X_rep.drop(["ID"],axis=1)
      X_sam[i]=X_rep.iloc[:,:-1].values
      Y_sam[i]=X_rep["C"].values

  X_sam=X_sam.astype(int)

  return X_sam,Y_sam
```

```
"--------------------------Accuracy in Bootstrap Sampling method-------------------------
```

```
iter=10

print('***************Boot strap Method**************')

tposr=np.zeros((iter,1))
fposr=np.zeros((iter,1))

for i in range(0,iter):
  X_resampled,Y_resampled=bootstrap(data)

  X_train,X_test,Y_train,Y_test = test_train_part(X_resampled,Y_resampled,0.632)

  w1=svm_train(X_train,Y_train,epochs=1000)
  ac=svm_test(X_test,w1,Y_test)

  CFb=np.zeros((2,2))
  CFb[0][0]=ac[2]
  CFb[0][1]=ac[4]
  CFb[1][0]=ac[3]
  CFb[1][1]=ac[1]

  tposr[i]=ac[2]/(ac[2]+ac[4])
  fposr[i]=ac[3]/(ac[1]+ac[3])
  print('Accuracy=',ac[0])
  recall=ac[2]/(ac[2]+ac[4])
  precision=ac[2]/(ac[2]+ac[3])
  f1=(2*ac[2])/((2*ac[2])+ac[3]+ac[4])
  print('Confusion Matrix :\n',CFb)
  print('Recall: ',recall,'\nPrecision: ',precision,'\nF1 Score: ',f1,'\n')

plt.xlabel('false positives rate')
plt.ylabel('true positives rate')
plt.xlim(0,1)
plt.ylim(0,1)
plt.scatter(fposr,tposr,marker='o')
plt.plot([0,1],[0,1],'-o')
plt.show()
```

```
**************Boot strap Method**************
Accuracy= 89.92248062015504
Confusion Matrix :
 [[0.62403101 0.03875969]
 [0.0620155  0.2751938 ]]
Recall:  0.9415204678362574
Precision:  0.9096045197740114
F1 Score:  0.925287356321839

Accuracy= 84.49612403100775
Confusion Matrix :
 [[0.49224806 0.13178295]
 [0.02325581 0.35271318]]
Recall:  0.7888198757763975
Precision:  0.9548872180451127
F1 Score:  0.8639455782312925

Accuracy= 83.33333333333334
Confusion Matrix :
 [[0.49612403 0.10465116]
 [0.0620155  0.3372093 ]]
Recall:  0.8258064516129032
Precision:  0.8888888888888888
F1 Score:  0.8561872909698997

Accuracy= 83.33333333333334
Confusion Matrix :
 [[0.47674419 0.15891473]
 [0.00775194 0.35658915]]
Recall:  0.75
Precision:  0.984
F1 Score:  0.8512110726643599

Accuracy= 86.82170542635659
Confusion Matrix :
 [[0.57364341 0.10465116]
 [0.02713178 0.29457364]]
Recall:  0.8457142857142858
Precision:  0.9548387096774192
F1 Score:  0.896969696969697

Accuracy= 84.49612403100775
Confusion Matrix :
 [[0.53100775 0.14341085]
 [0.01162791 0.31395349]]
Recall:  0.7873563218390804
Precision:  0.9785714285714285
F1 Score:  0.8726114649681529

Accuracy= 82.55813953488372
Confusion Matrix :
 [[0.58914729 0.07751938]
 [0.09689922 0.23643411]]
Recall:  0.8837209302325582
Precision:  0.8587570621468926
F1 Score:  0.8710601719197709

Accuracy= 58.13953488372093
Confusion Matrix :
 [[0.2751938  0.41860465]
 [0.         0.30620155]]
```

```
Recall:  0.3966480446927374
Precision:  1.0
F1 Score:  0.5680000000000001

Accuracy= 82.94573643410853
Confusion Matrix :
 [[0.63565891 0.08527132]
 [0.08527132 0.19379845]]
Recall:  0.8817204301075269
Precision:  0.8817204301075269
F1 Score:  0.881720430107527

Accuracy= 83.33333333333334
Confusion Matrix :
 [[0.5503876  0.10465116]
 [0.0620155  0.28294574]]
Recall:  0.8402366863905326
Precision:  0.8987341772151899
F1 Score:  0.8685015290519879
```