In [1]:
```
#from numpy.core.numeric import NaN
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter('ignore')
```

In [3]:
```
initial_Features=pd.read_csv('tox21_global_cdf_rdkit.csv')
initial_dataset=pd.read_csv('tox21.csv')
initial_Features=initial_Features.loc[:,initial_Features.apply(pd.Series.nunique)
initial_dataset=initial_dataset.iloc[initial_Features.dropna().index]
initial_dataset=initial_dataset.reset_index()
initial_Features=initial_Features.dropna()
initial_Features=initial_Features.reset_index()
index_array=[]
for i in np.arange (1,13):
   index_array.append(initial_dataset.iloc[:,i+1].dropna().index)
```

In [4]:
```
from pandas.core.frame import DataFrame
def label_ith(i):
    return pd.DataFrame(data=initial_dataset.iloc[index_array[i]].iloc[:,i+2])
def Feature_ith(i):
    return initial_Features.iloc[index_array[i]].drop('index',axis=1)
```

In [5]:
```
Feature_ith(3).head()
```

Out[5]:

	('BalabanJ', <class 'numpy.float64'>)	('BertzCT', <class 'numpy.float64'>)	('Chi0', <class 'numpy.float64'>)	('Chi0n', <class 'numpy.float64'>)	('Chi0v', <class 'numpy.float64'>)	('Chi 'numpy.
1	0.875932	0.047173	0.029397	0.031876	0.021488	
3	0.967576	0.059713	0.178132	0.308372	0.257215	
4	0.998591	0.009412	0.013743	0.002718	0.008883	
5	0.996279	0.014875	0.248635	0.451559	0.397276	
6	0.983223	0.023970	0.003767	0.001617	0.005040	

5 rows × 190 columns

In [7]:
```
# shape of dataset
for i in np.arange(0,12):
 print('Dimension of',i,'th labels: ',label_ith(i).shape,'    Dimension of feature
```

```
Dimension of 0 th labels:  (7166, 1)     Dimension of features:  (7166, 190)
Dimension of 1 th labels:  (6681, 1)     Dimension of features:  (6681, 190)
Dimension of 2 th labels:  (6475, 1)     Dimension of features:  (6475, 190)
Dimension of 3 th labels:  (5761, 1)     Dimension of features:  (5761, 190)
Dimension of 4 th labels:  (6128, 1)     Dimension of features:  (6128, 190)
Dimension of 5 th labels:  (6865, 1)     Dimension of features:  (6865, 190)
Dimension of 6 th labels:  (6373, 1)     Dimension of features:  (6373, 190)
Dimension of 7 th labels:  (5765, 1)     Dimension of features:  (5765, 190)
Dimension of 8 th labels:  (6991, 1)     Dimension of features:  (6991, 190)
Dimension of 9 th labels:  (6388, 1)     Dimension of features:  (6388, 190)
Dimension of 10 th labels:  (5735, 1)     Dimension of features:  (5735, 190)
Dimension of 11 th labels:  (6697, 1)     Dimension of features:  (6697, 190)
```

In [8]:
```
from sklearn.model_selection import train_test_split
X_training_data=[]
X_test=[]
y_training_data=[]
y_test=[]
for i in np.arange(0,12):
    X_training_data_tmp, X_test_tmp, y_training_data_tmp, y_test_tmp =train_test_s
    X_training_data.append(X_training_data_tmp)
    X_test.append(X_test_tmp)
    y_training_data.append(y_training_data_tmp)
    y_test.append(y_test_tmp)

from sklearn.decomposition import PCA
X_training_data_pca=[]
X_test_pca=[]
for i in np.arange(0,12):
  pca = PCA(n_components=71)
  principalComponents = pca.fit_transform(X_training_data[i])
  X_training_data_PCA_tmp = pd.DataFrame(data = principalComponents)
  X_training_data_pca.append(X_training_data_PCA_tmp)
  X_test_pca_tmp=pd.DataFrame(data = pca.transform(X_test[i]))
  X_test_pca.append(X_test_pca_tmp)
  print('PCA with 71 principal components retains',np.sum(pca.explained_variance_r
```

```
PCA with 71 principal components retains 95.15561720235434 % of data VAR. (It
is related for 0 th label)
PCA with 71 principal components retains 95.1845298900529 % of data VAR. (It i
s related for 1 th label)
PCA with 71 principal components retains 95.24634751131815 % of data VAR. (It
is related for 2 th label)
PCA with 71 principal components retains 95.25075875465564 % of data VAR. (It
is related for 3 th label)
PCA with 71 principal components retains 95.18393338742995 % of data VAR. (It
is related for 4 th label)
PCA with 71 principal components retains 95.18988397624564 % of data VAR. (It
is related for 5 th label)
PCA with 71 principal components retains 95.13612337438376 % of data VAR. (It
is related for 6 th label)
PCA with 71 principal components retains 95.21598716435155 % of data VAR. (It
is related for 7 th label)
PCA with 71 principal components retains 95.17490774284988 % of data VAR. (It
is related for 8 th label)
PCA with 71 principal components retains 95.16828621983608 % of data VAR. (It
is related for 9 th label)
PCA with 71 principal components retains 95.2322535963718 % of data VAR. (It i
s related for 10 th label)
PCA with 71 principal components retains 95.18388359013518 % of data VAR. (It
is related for 11 th label)
```

In [ ]:
```
#for i in (1,12):
#display(X_training_data[i].describe(),X_test[i].describe())

i=11
display(X_training_data[i].describe(),X_test[i].describe())
```

	('BalabanJ', <class 'numpy.float64'>)	('BertzCT', <class 'numpy.float64'>)	('Chi0', <class 'numpy.float64'>)	('Chi0n', <class 'numpy.float64'>)	('Chi0v', <class 'numpy.float64'>)	'n
count	6012.000000	6012.000000	6.012000e+03	6.012000e+03	6.012000e+03	
mean	0.797579	0.151504	1.830427e-01	1.888615e-01	1.916155e-01	
std	0.275032	0.228817	2.676787e-01	2.683001e-01	2.716296e-01	
min	0.000020	0.000577	3.875255e-15	2.722966e-11	1.025154e-07	
25%	0.743469	0.006519	5.928961e-03	6.301343e-03	5.810576e-03	
50%	0.929677	0.038143	4.001435e-02	4.750028e-02	4.479505e-02	
75%	0.975940	0.199407	2.553577e-01	2.785415e-01	2.895832e-01	
max	0.999740	1.000000	9.999127e-01	9.998083e-01	9.998569e-01	

8 rows × 190 columns

	('BalabanJ', <class 'numpy.float64'>)	('BertzCT', <class 'numpy.float64'>)	('Chi0', <class 'numpy.float64'>)	('Chi0n', <class 'numpy.float64'>)	('Chi0v', <class 'numpy.float64'>)	'n
count	669.000000	669.000000	669.000000	6.690000e+02	6.690000e+02	
mean	0.793070	0.162717	0.197346	2.035200e-01	2.028011e-01	
std	0.272745	0.242582	0.282832	2.823542e-01	2.830960e-01	
min	0.000020	0.000604	0.000009	1.211415e-08	3.818877e-07	
25%	0.728509	0.007945	0.006526	6.928929e-03	6.804298e-03	
50%	0.919290	0.042336	0.043001	5.065882e-02	5.005023e-02	
75%	0.977186	0.203237	0.264198	3.005023e-01	3.116237e-01	
max	0.999668	1.000000	0.999792	9.995754e-01	9.997444e-01	

8 rows × 190 columns

In [10]:
```
# Function to calculate 0 and 1 labels in test and train
def labels_counter(df1):
# count of 1 & 0 labels
  ones1=df1.sum()
  zeros1=df1.shape[0]-ones1
# Make a table with the results
  label_val_table1 = pd.concat([ones1,zeros1], axis=1)
# Rename the columns
  label_val_table_ren_columns1= label_val_table1.rename(
  columns = {0:'Label:1',1:'Label:0'} )
# Return the dataframe with missing information
  return label_val_table_ren_columns1
```

In [19]:
```
for i in np.arange(0,12):
  label_train=labels_counter(pd.DataFrame(y_training_data[i]))
  label_test=labels_counter(pd.DataFrame(y_test[i]))
  display("* trainingset labels *",label_train,' ** testset labels **' ,label_test
#  display(label_train,label_test)
```

'* trainingset labels *'

	Label:1	Label:0
NR-AR	276.0	6173.0

' ** testset labels **'

	Label:1	Label:0
NR-AR	31.0	686.0

'* trainingset labels *'

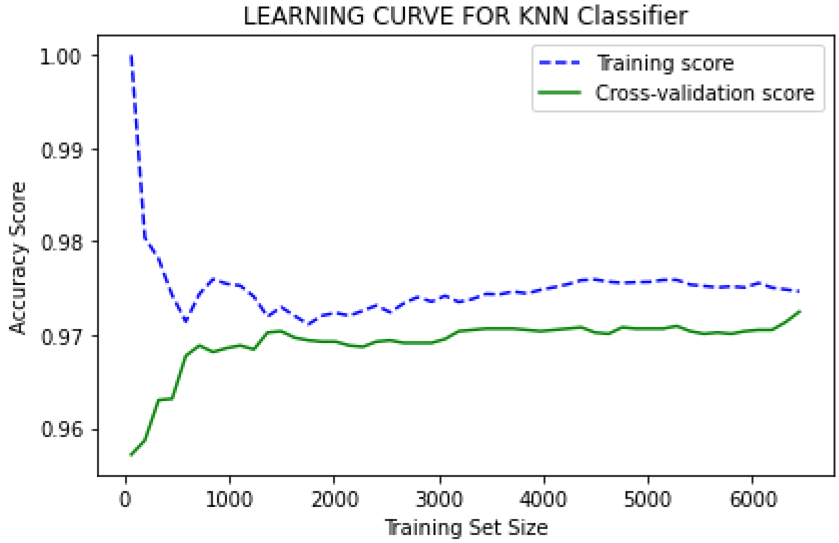
	Label:1	Label:0
NR-AR-LBD	205.0	5807.0

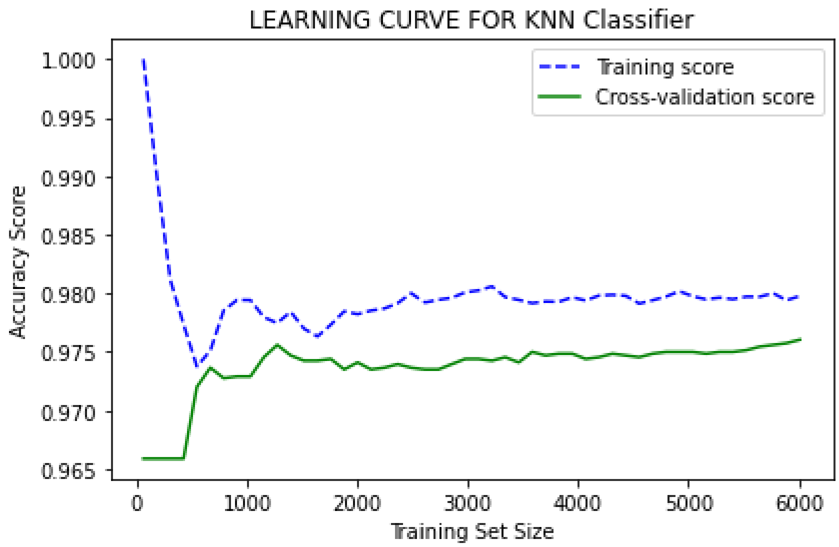
' ** testset labels **'

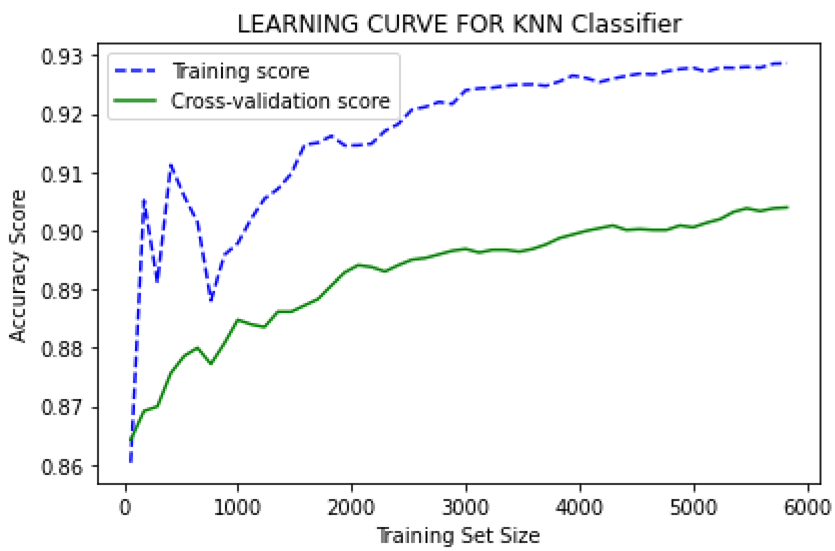
	Label:1	Label:0

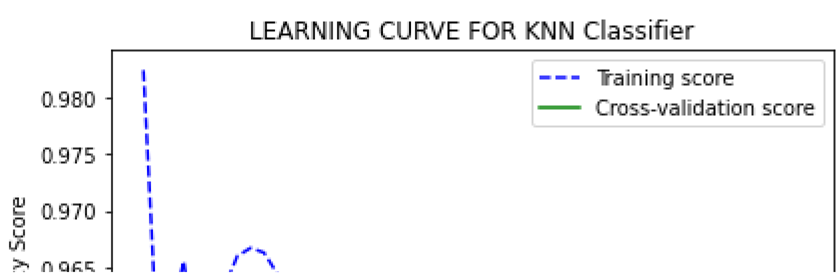
In [*]:

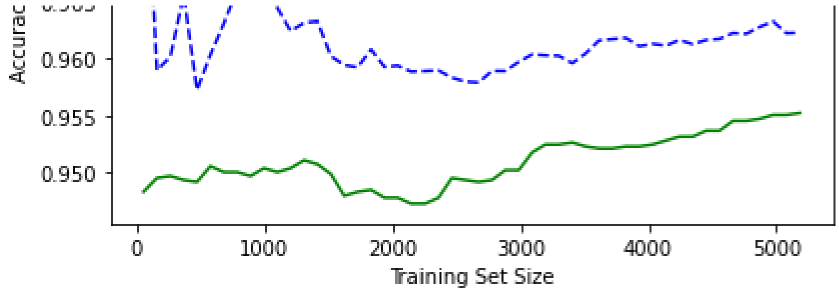
```
#Importing Required Libraries and Modules
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_digits
from sklearn.model_selection import learning_curve

for i in np.arange(0,12):
# X contains data and y contains labels
  X, y = Feature_ith(i),label_ith(i)

# Obtain scores from learning curve function
# cv is the number of folds while performing Cross Validation
  sizes, training_scores, testing_scores = learning_curve(KNeighborsClassifier(),

# Mean and Standard Deviation of training scores
  mean_training = np.mean(training_scores, axis=1)
  Standard_Deviation_training = np.std(training_scores, axis=1)

# Mean and Standard Deviation of testing scores
  mean_testing = np.mean(testing_scores, axis=1)
  Standard_Deviation_testing = np.std(testing_scores, axis=1)

# dotted blue line is for training scores and green line is for cross-validation s
  plt.plot(sizes, mean_training, '--', color="b", label="Training score")
  plt.plot(sizes, mean_testing, color="g", label="Cross-validation score")

# Drawing plot
  plt.title("LEARNING CURVE FOR KNN Classifier")
  plt.xlabel("Training Set Size"), plt.ylabel("Accuracy Score"), plt.legend(loc="b
  plt.tight_layout()
  plt.show()
```


LEARNING CURVE FOR KNN Classifier
Training score
Cross-validation score
Accuracy Score
1.00
0.99
0.98
0.97
0.96
0 1000 2000 3000 4000 5000 6000
Training Set Size

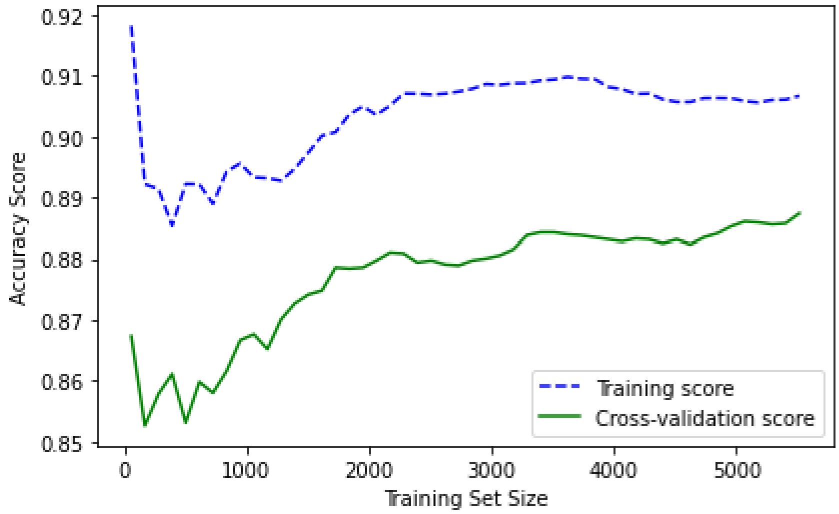

LEARNING CURVE FOR KNN Classifier
Training score
Cross-validation score
Accuracy Score
1.000
0.995
0.990
0.985
0.980
0.975
0.970
0.965
0 1000 2000 3000 4000 5000 6000
Training Set Size


LEARNING CURVE FOR KNN Classifier
Training score
Cross-validation score
Accuracy Score
0.93
0.92
0.91
0.90
0.89
0.88
0.87
0.86
0 1000 2000 3000 4000 5000 6000
Training Set Size

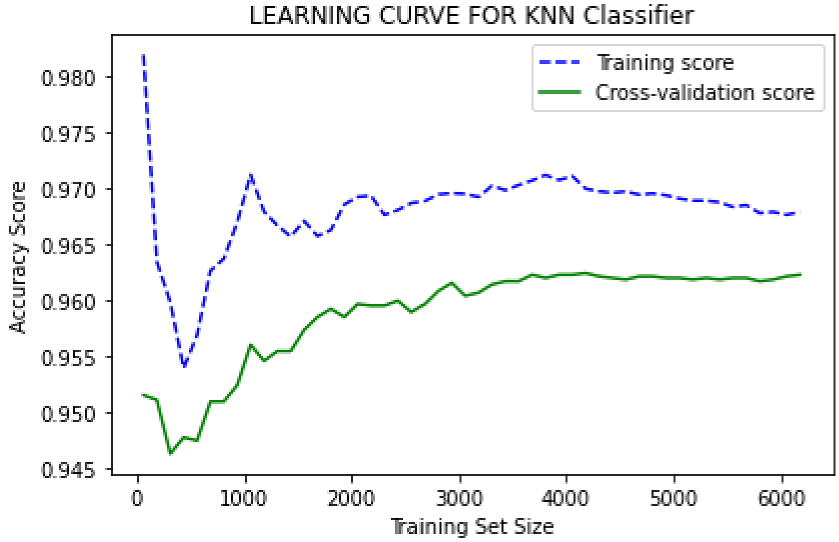

LEARNING CURVE FOR KNN Classifier
Training score
Cross-validation score
0.980
0.975
0.970



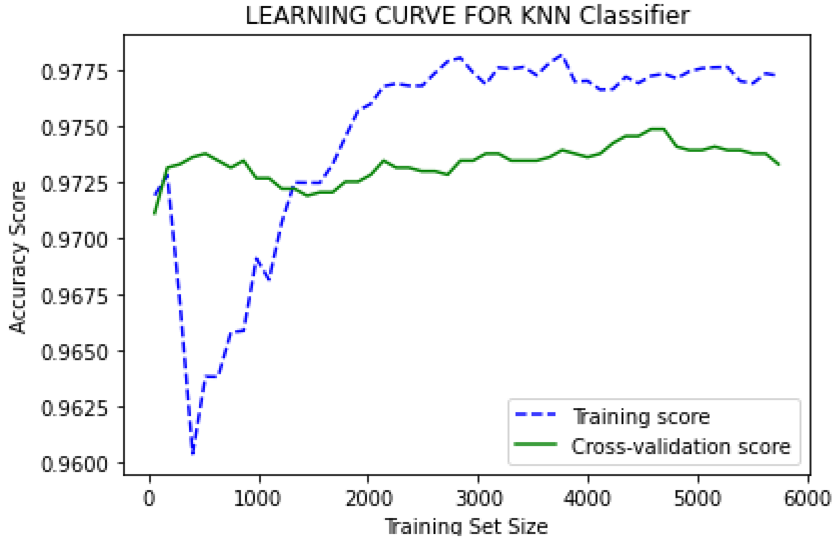
LEARNING CURVE FOR KNN Classifier

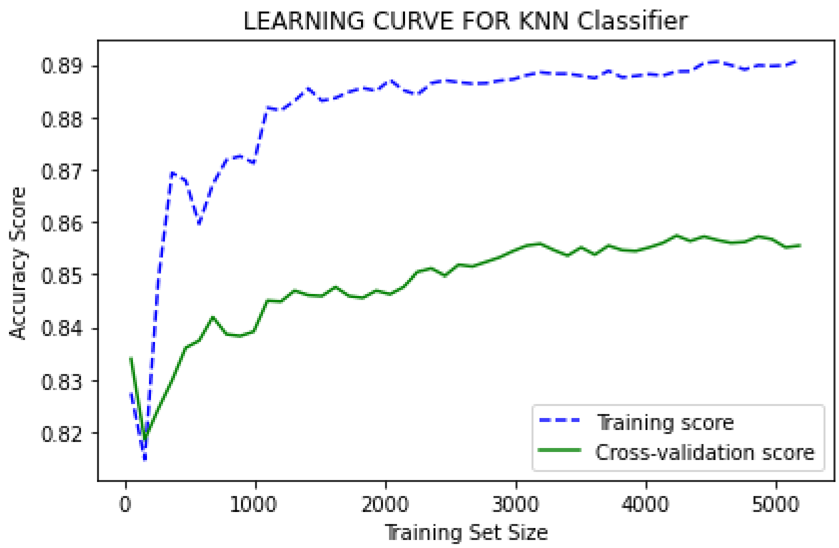


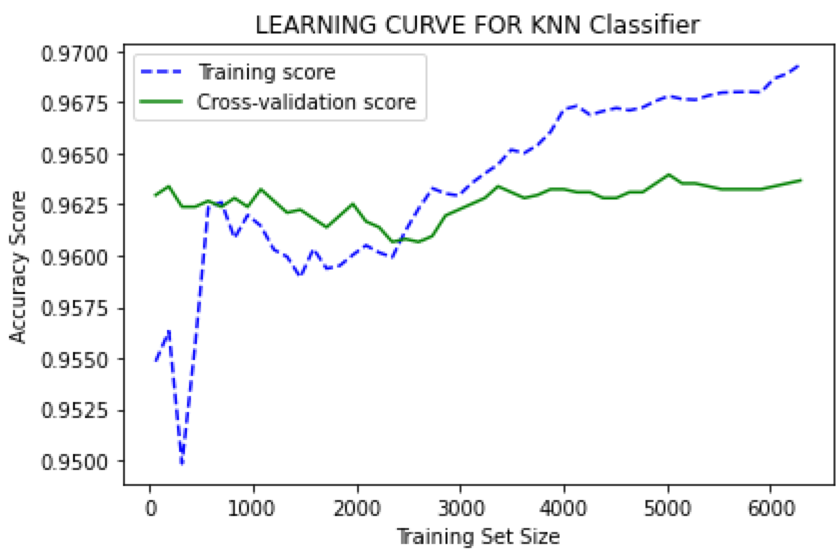
LEARNING CURVE FOR KNN Classifier

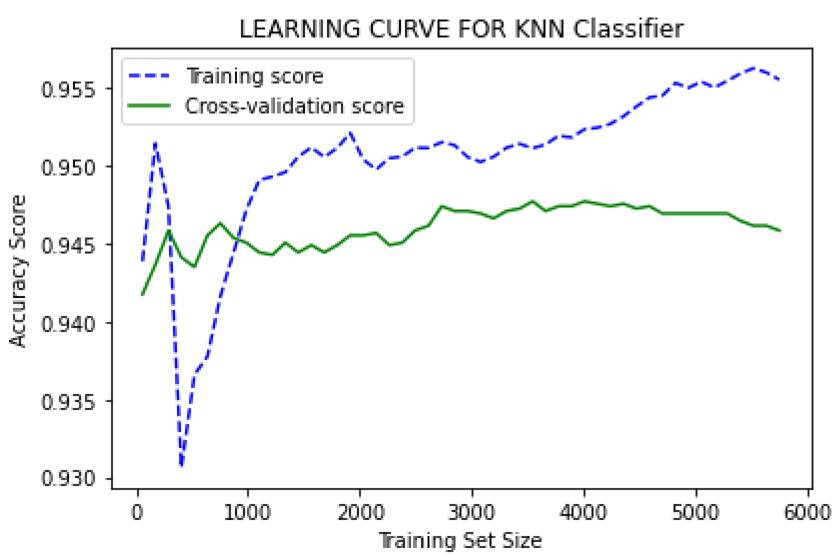


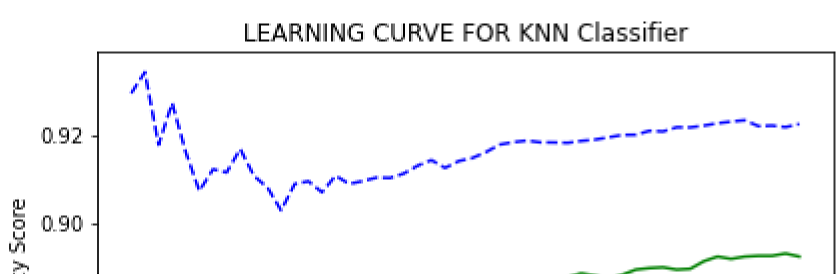
LEARNING CURVE FOR KNN Classifier



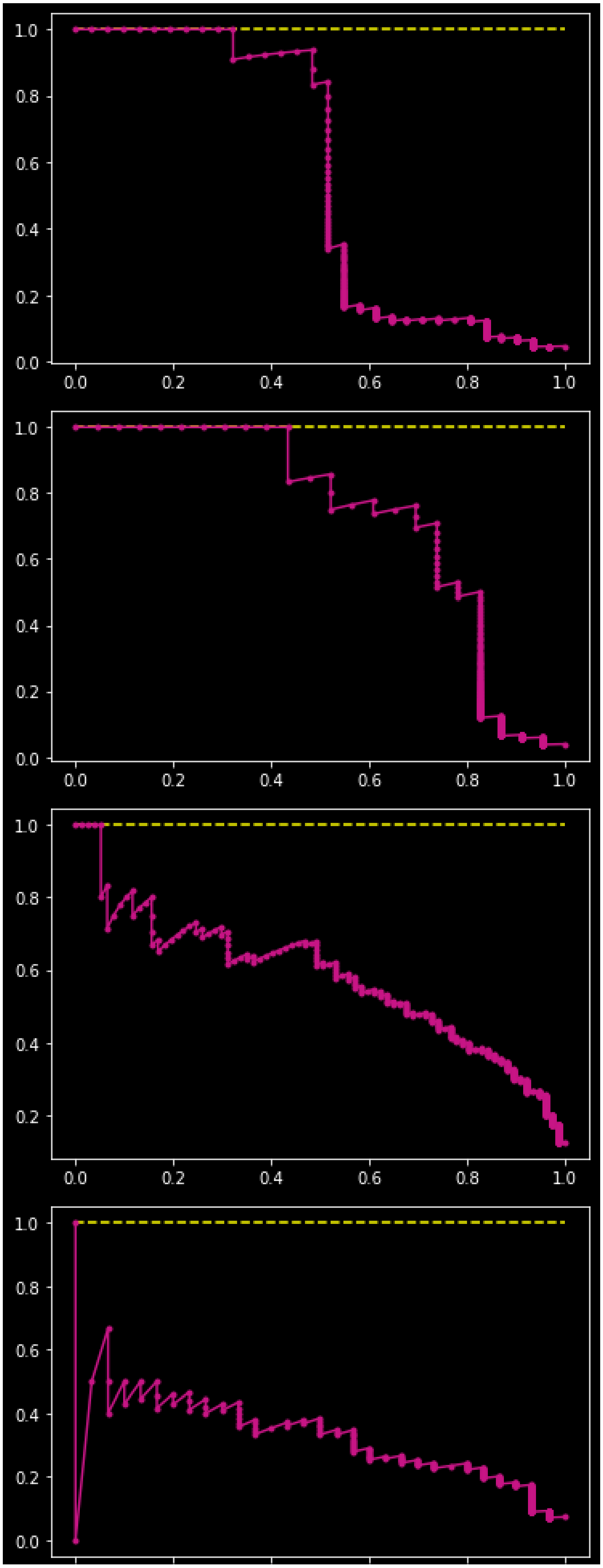

LEARNING CURVE FOR KNN Classifier
Accuracy Score
Training Set Size
Training score
Cross-validation score
0.89
0.88
0.87
0.86
0.85
0.84
0.83
0.82
0
1000
2000
3000
4000
5000

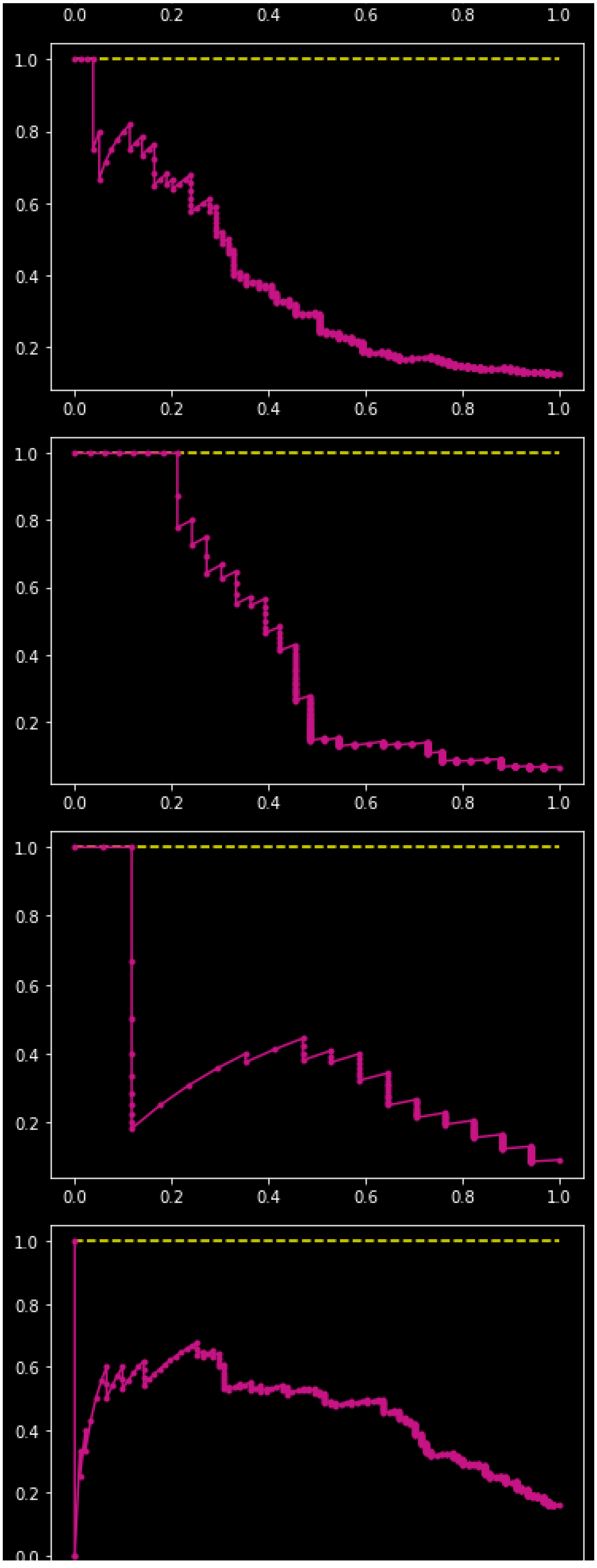

LEARNING CURVE FOR KNN Classifier
Accuracy Score
Training Set Size
Training score
Cross-validation score
0.9700
0.9675
0.9650
0.9625
0.9600
0.9575
0.9550
0.9525
0.9500
0
1000
2000
3000
4000
5000
6000

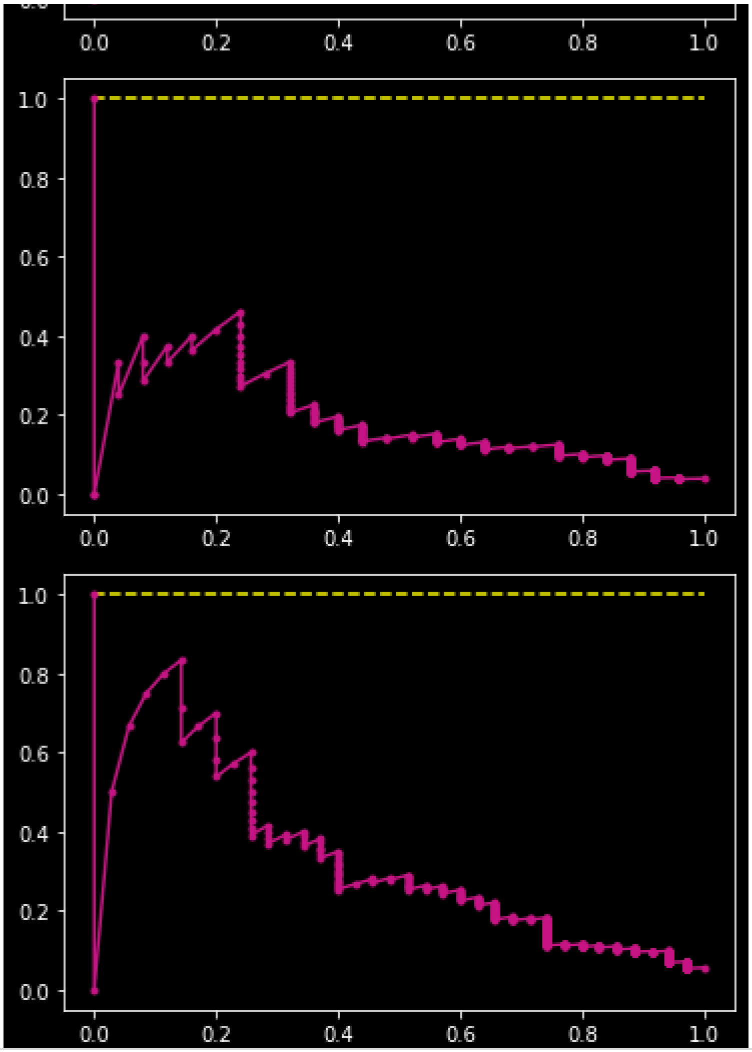

LEARNING CURVE FOR KNN Classifier
Accuracy Score
Training Set Size
Training score
Cross-validation score
0.955
0.950
0.945
0.940
0.935
0.930
0
1000
2000
3000
4000
5000
6000


LEARNING CURVE FOR KNN Classifier
0.92
0.90

In [ ]:
```
# precision-recall curve and f1 for an imbalanced dataset
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from matplotlib import pyplot
from matplotlib.pyplot import cm
for i in np.arange(0,12):
  model = LogisticRegression(solver='lbfgs')
  model.fit(X_training_data[i], y_training_data[i])
  dataset_probs = model.predict_proba(X_test[i])
  # keep probabilities for the positive outcome only
  dataset_probs = dataset_probs[:, 1]
  # predict class values
  yhat = model.predict(X_test[i])
  # calculate precision and recall for each threshold
  dataset_precision, dataset_recall, _ = precision_recall_curve(y_test[i], dataset
  # calculate scores
  dataset_f1, dataset_auc = f1_score(y_test[i], yhat), auc(dataset_recall, dataset
  # summarize scores
  print('Logistic: f1=%.3f auc=%.3f' % (dataset_f1, dataset_auc))

  # plot the precision-recall curves

  no_skill = len(y_test[i][y_test[i]==1]) / len(y_test[i])
  plt.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill',c='mediu
  plt.plot(dataset_recall, dataset_precision, marker='.', label='Logistic',c='yell
  plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0 0.2 0.4 0.6 0.8 1.0


0.0
0.2
0.4
0.6
0.8
1.0



✓ 5s completed at 8:51 PM