In [1]:
```
#Installing the required Libraries
!pip install seaborn
!pip install sklearn
```

```
Collecting seaborn
  Downloading seaborn-0.11.1-py3-none-any.whl (285 kB)
     |████████████████████████████████| 285 kB 3.8 MB/s eta 0:00:01
Collecting matplotlib>=2.2
  Downloading matplotlib-3.4.1-cp38-cp38-manylinux1_x86_64.whl (10.3 MB)
     |████████████████████████████████| 10.3 MB 6.0 MB/s eta 0:00:01
Collecting pandas>=0.23
  Downloading pandas-1.2.4-cp38-cp38-manylinux1_x86_64.whl (9.7 MB)
     |████████████████████████████████| 9.7 MB 7.3 MB/s eta 0:00:01     |████
████                                | 2.2 MB 5.5 MB/s eta 0:00:02
Collecting scipy>=1.0
  Downloading scipy-1.6.3-cp38-cp38-manylinux1_x86_64.whl (27.2 MB)
     |████████████████████████████████| 27.2 MB 10.5 MB/s eta 0:00:01
Collecting numpy>=1.15
  Downloading numpy-1.20.2-cp38-cp38-manylinux2010_x86_64.whl (15.4 MB)
     |████████████████████████████████| 15.4 MB 7.1 MB/s eta 0:00:01
Collecting pillow>=6.2.0
  Downloading Pillow-8.2.0-cp38-cp38-manylinux1_x86_64.whl (3.0 MB)
     |████████████████████████████████| 3.0 MB 4.8 MB/s eta 0:00:01
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/lib/python
3.8/site-packages (from matplotlib>=2.2->seaborn) (2.8.1)
Collecting kiwisolver>=1.0.1
  Downloading kiwisolver-1.3.1-cp38-cp38-manylinux1_x86_64.whl (1.2 MB)
     |████████████████████████████████| 1.2 MB 11.1 MB/s eta 0:00:01
Requirement already satisfied: pyparsing>=2.2.1 in /opt/conda/lib/python3.8/s
ite-packages (from matplotlib>=2.2->seaborn) (2.4.7)
Collecting cycler>=0.10
  Downloading cycler-0.10.0-py2.py3-none-any.whl (6.5 kB)
Requirement already satisfied: six in /opt/conda/lib/python3.8/site-packages
(from cycler>=0.10->matplotlib>=2.2->seaborn) (1.15.0)
Requirement already satisfied: pytz>=2017.3 in /opt/conda/lib/python3.8/site-
packages (from pandas>=0.23->seaborn) (2021.1)
Installing collected packages: pillow, numpy, kiwisolver, cycler, scipy, pand
as, matplotlib, seaborn
Successfully installed cycler-0.10.0 kiwisolver-1.3.1 matplotlib-3.4.1 numpy-
1.20.2 pandas-1.2.4 pillow-8.2.0 scipy-1.6.3 seaborn-0.11.1
Collecting sklearn
  Downloading sklearn-0.0.tar.gz (1.1 kB)
Collecting scikit-learn
  Downloading scikit_learn-0.24.2-cp38-cp38-manylinux2010_x86_64.whl (24.9 M
B)
     |████████████████████████████████| 24.9 MB 9.6 MB/s eta 0:00:011
Requirement already satisfied: numpy>=1.13.3 in /opt/conda/lib/python3.8/site
-packages (from scikit-learn->sklearn) (1.20.2)
Requirement already satisfied: scipy>=0.19.1 in /opt/conda/lib/python3.8/site
-packages (from scikit-learn->sklearn) (1.6.3)
Collecting threadpoolctl>=2.0.0
  Downloading threadpoolctl-2.1.0-py3-none-any.whl (12 kB)
Collecting joblib>=0.11
  Downloading joblib-1.0.1-py3-none-any.whl (303 kB)
     |████████████████████████████████| 303 kB 8.4 MB/s eta 0:00:01
Building wheels for collected packages: sklearn
```

```
  Building wheel for sklearn (setup.py) ... done
  Created wheel for sklearn: filename=sklearn-0.0-py2.py3-none-any.whl size=1
316 sha256=d161716cd4bf6498f84b3b0718c5cae520786f78ff10bbcdbfa26d46accc5b7e
  Stored in directory: /home/jovyan/.cache/pip/wheels/22/0b/40/fd3f795caaa1fb
4c6cb738bc1f56100be1e57da95849bfc897
Successfully built sklearn
Installing collected packages: threadpoolctl, joblib, scikit-learn, sklearn
Successfully installed joblib-1.0.1 scikit-learn-0.24.2 sklearn-0.0 threadpoo
lctl-2.1.0
```

In [ ]:

```
import numpy as np
import pandas as pd

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import KFold, cross_validate, cross_val_predict, val
from sklearn.metrics import confusion_matrix, make_scorer
```

In [259]:

```
#Load and print data
data = pd.read_csv('./data/diabetes.csv')
data.head()
```

Out[259]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288

In [260]:

```
data.describe()
```

Out[260]:

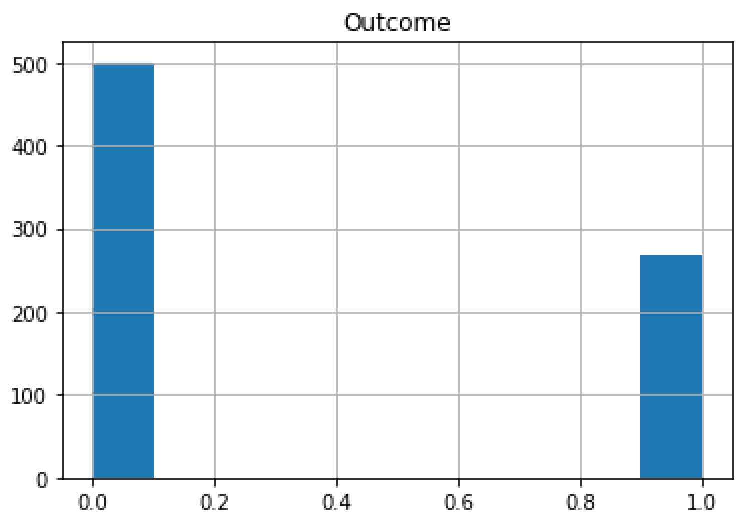
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [261]:
```
for column in list(data):
    print(f'Column name:  {column} , no of null : {data[column].size - data[colu
```

```
Column name:  Pregnancies , no of null : 0
Column name:  Glucose , no of null : 0
Column name:  BloodPressure , no of null : 0
Column name:  SkinThickness , no of null : 0
Column name:  Insulin , no of null : 0
Column name:  BMI , no of null : 0
Column name:  DiabetesPedigreeFunction , no of null : 0
Column name:  Age , no of null : 0
Column name:  Outcome , no of null : 0
```

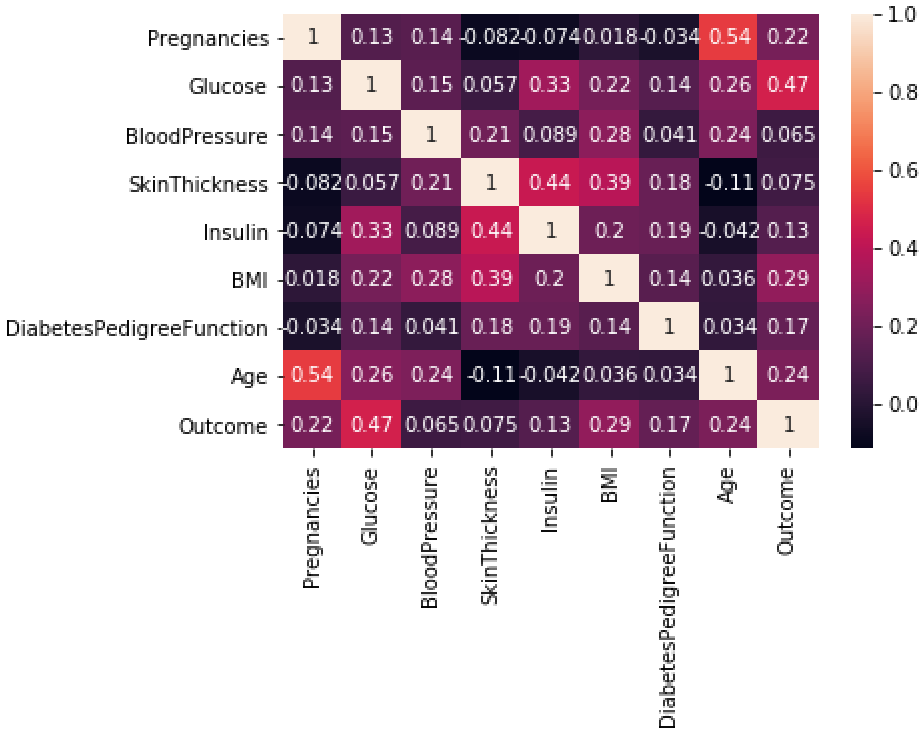
In [262]:
```
# Diabetics.csv has 2 times non diabetic to 1 time diabetic data
data.hist(column='Outcome'))
```

Out[262]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f306c3c6ba8>]],
      dtype=object)
```



In [263]:
```
sns.heatmap(data.corr(),annot=True)
# Corelation b/w fields
```

Out[263]: <matplotlib.axes._subplots.AxesSubplot at 0x7f306c513cc0>



In [264]:
```
# features on the 'x' axis
X = data.drop("Outcome",axis = 1)

# label in 'y' axis
y = data.Outcome
```

In [265]:
```
X.head()
```

Out[265]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288

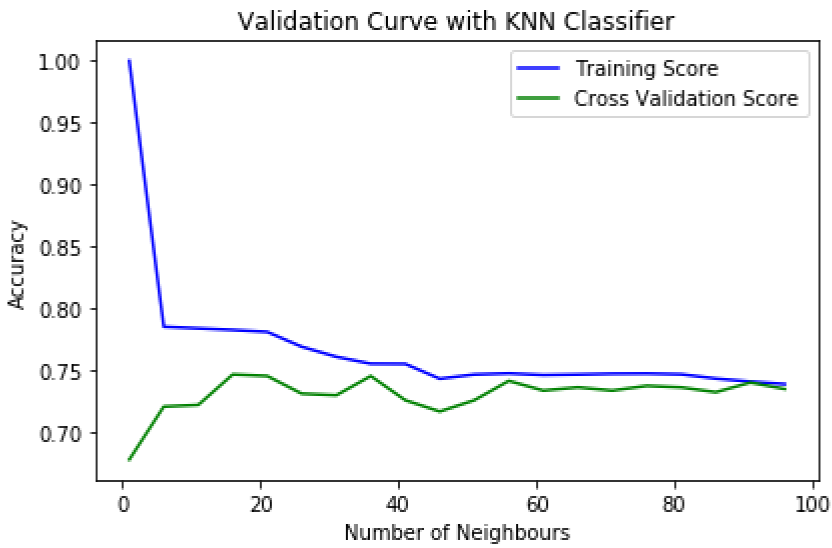
In [266]:
```
y.head()
```

Out[266]:
```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

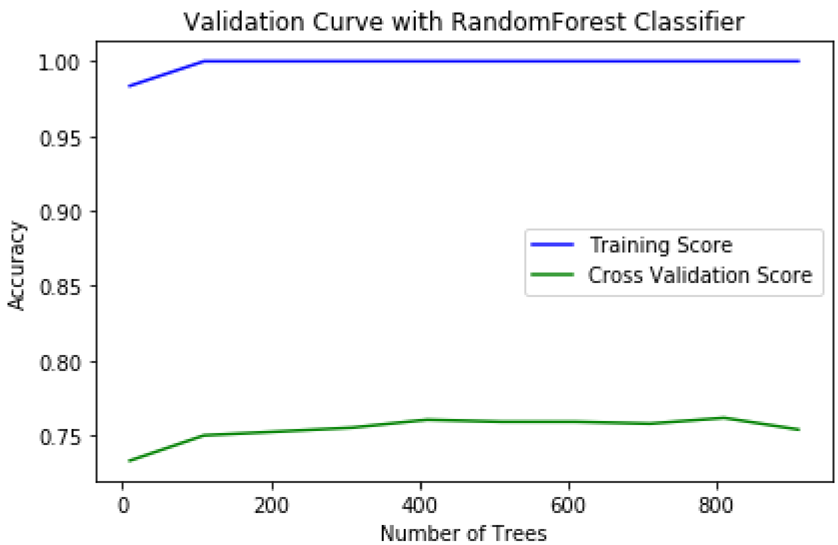
In [267]:
```
cv = KFold(n_splits=10, random_state=10, shuffle=True)
```

In [268]:
```
def plot_results(train_score, test_score, title, xlabel):
    #standard deviation and mean calculated for testing and trainig scores
    mean_train_score = np.mean(train_score, axis = 1)
    std_train_score = np.std(train_score, axis = 1)


    mean_test_score = np.mean(test_score, axis = 1)
    std_test_score = np.std(test_score, axis = 1)

    # Creating the Plot for above
    plt.plot(parameter_range, mean_train_score, label = "Training Score", color =
    plt.plot(parameter_range, mean_test_score, label = "Cross Validation Score",

    # values considered for algo comparison
    best_neighbor = parameter_range[np.argmax(mean_test_score)]

    # Creating the plot
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel("Accuracy")
    plt.tight_layout()
    plt.legend(loc = 'best')
    plt.show()
```

In [269]:
```
# best n_neighbors for knn algo
parameter_range = np.arange(1, 100, 5)

# 10-fold cross validation
train_score, test_score = validation_curve(KNeighborsClassifier(), X, y,
                                       param_name = "n_neighbors",
                                       param_range = parameter_range,
                                        cv = cv, scoring = "accuracy")

plot_results(train_score, test_score, "Validation Curve with KNN Classifier", "Nu
```

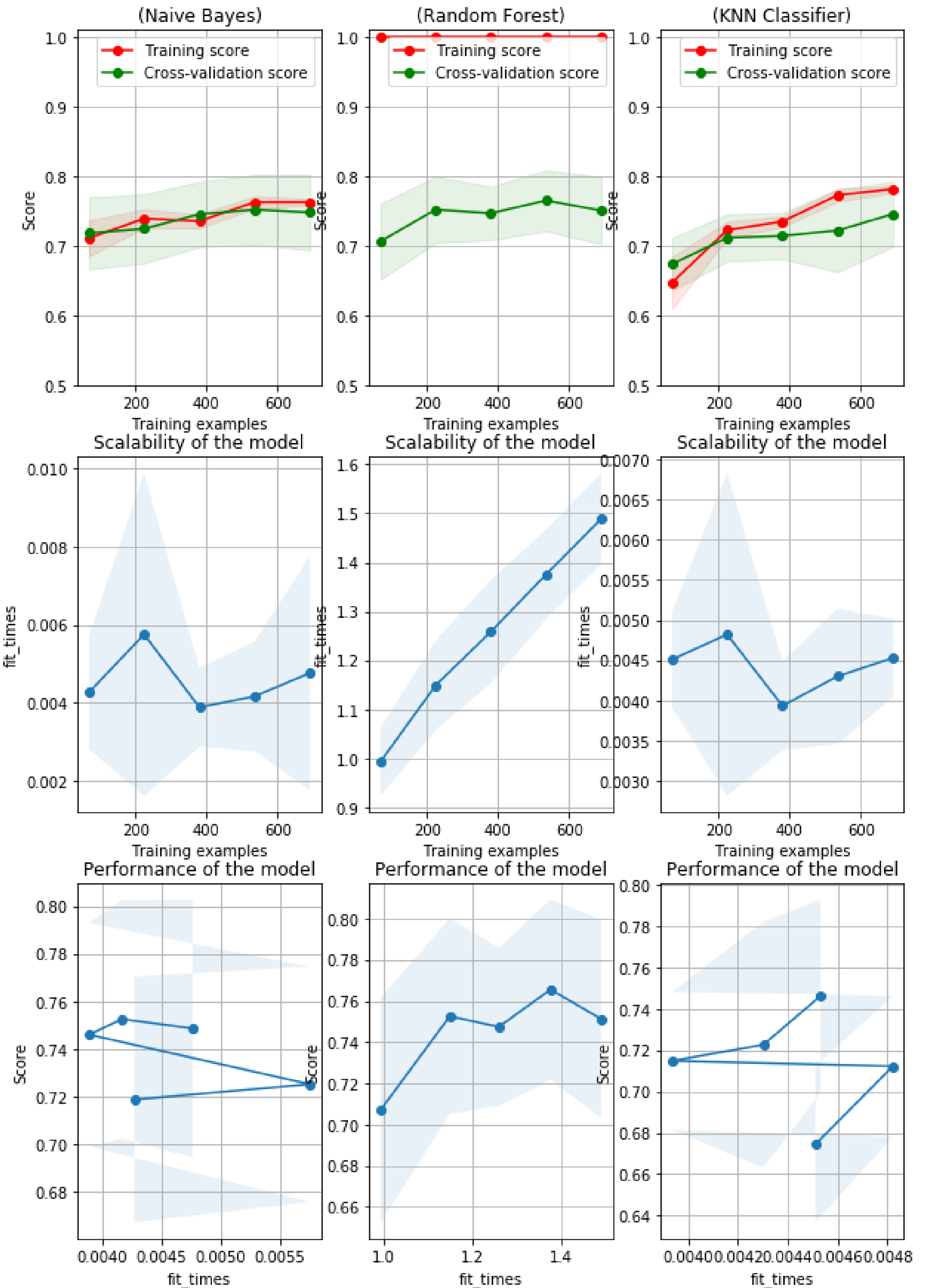


In [270]:
```
# best n_estimators in random forest
parameter_range = np.arange(10, 1000, 100)

# 10-fold cross validation
train_score, test_score = validation_curve(RandomForestClassifier(), X, y,
                                           param_name = "n_estimators",
                                           param_range = parameter_range,
                                            cv = cv, scoring = "accuracy")

plot_results(train_score, test_score, "Validation Curve with RandomForest Classi
```



In [271]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit


def plot_learning_curve(estimator, title, X, y, axes=None, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
    if axes is None:
        _, axes = plt.subplots(1, 3, figsize=(20, 5))

    axes[0].set_title(title)
    if ylim is not None:
        axes[0].set_ylim(*ylim)
    axes[0].set_xlabel("Training examples")
    axes[0].set_ylabel("Score")

    train_sizes, train_scores, test_scores, fit_times, _ = \
        learning_curve(estimator, X, y, cv=cv, n_jobs=n_jobs,
                       train_sizes=train_sizes,
                       return_times=True)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    # Learning curve plot
    axes[0].grid()
    axes[0].fill_between(train_sizes, train_scores_mean - train_scores_std,
                         train_scores_mean + train_scores_std, alpha=0.1,
                         color="r")
    axes[0].fill_between(train_sizes, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1,
                         color="g")
    axes[0].plot(train_sizes, train_scores_mean, 'o-', color="r",
                 label="Training score")
    axes[0].plot(train_sizes, test_scores_mean, 'o-', color="g",
                 label="Cross-validation score")
    axes[0].legend(loc="best")


    axes[1].grid()
    axes[1].plot(train_sizes, fit_times_mean, 'o-')
    axes[1].fill_between(train_sizes, fit_times_mean - fit_times_std,
                         fit_times_mean + fit_times_std, alpha=0.1)
    axes[1].set_xlabel("Training examples")
    axes[1].set_ylabel("fit_times")
    axes[1].set_title("Scalability of the model")

    # Plot fit_time vs score
    axes[2].grid()
    axes[2].plot(fit_times_mean, test_scores_mean, 'o-')
    axes[2].fill_between(fit_times_mean, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1)
```

```
    axes[2].set_xlabel("fit_times")
    axes[2].set_ylabel("Score")
    axes[2].set_title("Performance of the model")

    return plt


fig, axes = plt.subplots(3, 3, figsize=(10, 15))

# Comparison of Naive Bayes, RF and KNN algo
clf_gauss = GaussianNB()
plot_learning_curve(clf_gauss, "(Naive Bayes)", X, y, axes=axes[:, 0], ylim=(0.5,
                    cv=cv, n_jobs=4)

clf_rf = RandomForestClassifier(n_estimators = best_estimator)
plot_learning_curve(clf_rf, "(Random Forest)", X, y, axes=axes[:, 1], ylim=(0.5,
                    cv=cv, n_jobs=4)

clf_knn = KNeighborsClassifier(n_neighbors = best_neighbor)
plot_learning_curve(clf_knn, "(KNN Classifier)", X, y, axes=axes[:, 2], ylim=(0.5
                    cv=cv, n_jobs=4)

plt.show()

# Naive Bayes, RF and KNN have similar performance. Naives Bayes and KNN algorith
```


(Naive Bayes)
(Random Forest)
(KNN Classifier)
Training score
Cross-validation score
Score
Training examples
Scalability of the model
fit_times
Performance of the model

In [272]:
```
def cal_tn(Y_test, y_pred): return confusion_matrix(Y_test, y_pred)[0,0]
def cal_fp(Y_test, y_pred): return confusion_matrix(Y_test, y_pred)[0,1]
def cal_fn(Y_test, y_pred): return confusion_matrix(Y_test, y_pred)[1,0]
def cal_tp(Y_test, y_pred): return confusion_matrix(Y_test, y_pred)[1,1]

def tpr(Y_test,y_pred):
    tp = cal_tp(Y_test,y_pred)
    fn = cal_fn(Y_test,y_pred)
    return round((tp / (tp + fn)),2)

def tnr(Y_test,y_pred):
    tn = cal_tn(Y_test,y_pred)
    fp = cal_fp(Y_test,y_pred)
    return round((tn / (tn + fp)),2)

def fpr(Y_test,y_pred):
    tn = cal_tn(Y_test,y_pred)
    fp = cal_fp(Y_test,y_pred)

    return round((fp / (tn + fp)),2)

def fnr(Y_test,y_pred):
    tp = cal_tp(Y_test,y_pred)
    fn = cal_fn(Y_test,y_pred)
    return round((fn / (tp + fn)),2)

def Recall(Y_test,y_pred):
    tp = cal_tp(Y_test,y_pred)
    fn = cal_fn(Y_test,y_pred)
    return round((tp / (tp + fn)),2)

def Precision(Y_test,y_pred):
    tp = cal_tp(Y_test,y_pred)
    fp = cal_fp(Y_test,y_pred)
    return round((tp / (tp + fp)),2)

def F1Score(Y_test,y_pred):
    tp = cal_tp(Y_test,y_pred)
    fp = cal_fp(Y_test,y_pred)
    fn = cal_fn(Y_test,y_pred)
    return round(((2*tp) / ((2*tp) + fp+fn)),2)

def Accuracy(Y_test,y_pred):
    tn = cal_tn(Y_test,y_pred)
    tp = cal_tp(Y_test,y_pred)
    fp = cal_fp(Y_test,y_pred)
    fn = cal_fn(Y_test,y_pred)
    return round(((tp + tn) / (tp + fp + fn + tn)),2)

def Error(Y_test,y_pred):
    tn = cal_tn(Y_test,y_pred)
    tp = cal_tp(Y_test,y_pred)
    fp = cal_fp(Y_test,y_pred)
    fn = cal_fn(Y_test,y_pred)
    return round(((fp + fn) / (tp + fp + fn + tn)),2)
```

```
def BACC(Y_test,y_pred):
    tn = cal_tn(Y_test,y_pred)
    tp = cal_tp(Y_test,y_pred)
    fp = cal_fp(Y_test,y_pred)
    fn = cal_fn(Y_test,y_pred)
    return round(0.5*((tp / (tp + fn))+(tn / (fp + tn))),2)

def TSS(Y_test,y_pred):
    tn = cal_tn(Y_test,y_pred)
    tp = cal_tp(Y_test,y_pred)
    fp = cal_fp(Y_test,y_pred)
    fn = cal_fn(Y_test,y_pred)
    return round((tp / (tp + fn))-(fp / (fp + tn)),2)

def HSS(Y_test,y_pred):
    tn = cal_tn(Y_test,y_pred)
    tp = cal_tp(Y_test,y_pred)
    fp = cal_fp(Y_test,y_pred)
    fn = cal_fn(Y_test,y_pred)
    return round((2*((tp * tn)-(fp * fn)))/(((tp + fn)*(fn + tn))+((tp + fp)*(fp

def cal_mean(dict_score):
    df = pd.DataFrame.from_dict(dict_score, orient='index')
    df['mean'] = df.mean(axis=1)
    return df
```

In [273]:

```
from sklearn.metrics import confusion_matrix,make_scorer
scoring = {'tp': make_scorer(tp),'tn': make_scorer(tn),'fp': make_scorer(fp),'fn'
           'tnr':make_scorer(tnr),'fpr':make_scorer(fpr),'fnr':make_scorer(fnr),'
           'Accuracy':make_scorer(Accuracy),'Error':make_scorer(Error),'BACC':mak
           'HSS':make_scorer(HSS)}
```

In [274]:
```
clf_gauss_score = cross_validate(clf_gauss,X,y,scoring = scoring,cv=cv)
df_gauss = cal_mean(clf_gauss_score)
df_gauss.head(20)
```

Out[274]:

	0	1	2	3	4	5	6	
fit_time	0.011970	0.002966	0.002934	0.00289	0.004081	0.005298	0.003251	0.0028
score_time	0.029084	0.022756	0.023649	0.02103	0.024744	0.023822	0.023445	0.0204
test_tp	12.000000	20.000000	15.000000	14.00000	16.000000	14.000000	16.000000	20.0000
test_tn	43.000000	38.000000	40.000000	48.00000	45.000000	40.000000	38.000000	47.0000
test_fp	8.000000	6.000000	9.000000	9.00000	6.000000	10.000000	10.000000	4.0000
test_fn	14.000000	13.000000	13.000000	6.00000	10.000000	13.000000	13.000000	6.0000
test_tpr	0.460000	0.610000	0.540000	0.70000	0.620000	0.520000	0.550000	0.7700
test_tnr	0.840000	0.860000	0.820000	0.84000	0.880000	0.800000	0.790000	0.9200
test_fpr	0.160000	0.140000	0.180000	0.16000	0.120000	0.200000	0.210000	0.0800
test_fnr	0.540000	0.390000	0.460000	0.30000	0.380000	0.480000	0.450000	0.2300
test_recall	0.460000	0.610000	0.540000	0.70000	0.620000	0.520000	0.550000	0.7700
test_precision	0.600000	0.770000	0.620000	0.61000	0.730000	0.580000	0.620000	0.8300
test_F1Score	0.520000	0.680000	0.580000	0.65000	0.670000	0.550000	0.580000	0.8000
test_Accuracy	0.710000	0.750000	0.710000	0.81000	0.790000	0.700000	0.700000	0.8700
test_Error	0.290000	0.250000	0.290000	0.19000	0.210000	0.300000	0.300000	0.1300
test_BACC	0.650000	0.730000	0.680000	0.77000	0.750000	0.660000	0.670000	0.8500
test_TSS	0.300000	0.470000	0.350000	0.54000	0.500000	0.320000	0.340000	0.6900
test_HSS	0.320000	0.480000	0.360000	0.52000	0.520000	0.330000	0.350000	0.7000

In [275]:
```
clf_rf_score = cross_validate(clf_rf,X,y,scoring = scoring,cv=cv)
df_rf = cal_mean(clf_rf_score)
df_rf.head(20)
```

Out[275]:

	0	1	2	3	4	5	6	
fit_time	1.064124	1.038974	1.030510	1.019342	1.008097	0.996995	1.021667	1.009
score_time	0.106313	0.109850	0.104925	0.102989	0.109176	0.103296	0.101269	0.104
test_tp	13.000000	19.000000	15.000000	13.000000	14.000000	16.000000	14.000000	17.000
test_tn	44.000000	40.000000	44.000000	45.000000	45.000000	43.000000	37.000000	48.000
test_fp	7.000000	4.000000	5.000000	12.000000	6.000000	7.000000	11.000000	3.000
test_fn	13.000000	14.000000	13.000000	7.000000	12.000000	11.000000	15.000000	9.000
test_tpr	0.500000	0.580000	0.540000	0.650000	0.540000	0.590000	0.480000	0.650
test_tnr	0.860000	0.910000	0.900000	0.790000	0.880000	0.860000	0.770000	0.940
test_fpr	0.140000	0.090000	0.100000	0.210000	0.120000	0.140000	0.230000	0.060
test_fnr	0.500000	0.420000	0.460000	0.350000	0.460000	0.410000	0.520000	0.350
test_recall	0.500000	0.580000	0.540000	0.650000	0.540000	0.590000	0.480000	0.650
test_precision	0.650000	0.830000	0.750000	0.520000	0.700000	0.700000	0.560000	0.850
test_F1Score	0.570000	0.680000	0.620000	0.580000	0.610000	0.640000	0.520000	0.740
test_Accuracy	0.740000	0.770000	0.770000	0.750000	0.770000	0.770000	0.660000	0.840
test_Error	0.260000	0.230000	0.230000	0.250000	0.230000	0.230000	0.340000	0.160
test_BACC	0.680000	0.740000	0.720000	0.720000	0.710000	0.730000	0.630000	0.800
test_TSS	0.360000	0.480000	0.430000	0.440000	0.420000	0.450000	0.250000	0.600
test_HSS	0.380000	0.500000	0.460000	0.410000	0.450000	0.470000	0.260000	0.630

In [276]:
```
clf_knn_score = cross_validate(clf_knn,X,y,scoring = scoring,cv=cv)
df_knn = cal_mean(clf_knn_score)
df_knn.head(20)
```

Out[276]:

	0	1	2	3	4	5	6	
fit_time	0.006313	0.00418	0.002788	0.002858	0.003150	0.002965	0.002766	0.0030
score_time	0.033054	0.03295	0.025597	0.025110	0.023521	0.024060	0.023784	0.0255
test_tp	10.000000	13.00000	14.000000	11.000000	15.000000	11.000000	14.000000	13.0000
test_tn	45.000000	38.00000	42.000000	47.000000	47.000000	46.000000	41.000000	50.0000
test_fp	6.000000	6.00000	7.000000	10.000000	4.000000	4.000000	7.000000	1.0000
test_fn	16.000000	20.00000	14.000000	9.000000	11.000000	16.000000	15.000000	13.0000
test_tpr	0.380000	0.39000	0.500000	0.550000	0.580000	0.410000	0.480000	0.5000
test_tnr	0.880000	0.86000	0.860000	0.820000	0.920000	0.920000	0.850000	0.9800
test_fpr	0.120000	0.14000	0.140000	0.180000	0.080000	0.080000	0.150000	0.0200
test_fnr	0.620000	0.61000	0.500000	0.450000	0.420000	0.590000	0.520000	0.5000
test_recall	0.380000	0.39000	0.500000	0.550000	0.580000	0.410000	0.480000	0.5000
test_precision	0.620000	0.68000	0.670000	0.520000	0.790000	0.730000	0.670000	0.9300
test_F1Score	0.480000	0.50000	0.570000	0.540000	0.670000	0.520000	0.560000	0.6500
test_Accuracy	0.710000	0.66000	0.730000	0.750000	0.810000	0.740000	0.710000	0.8200
test_Error	0.290000	0.34000	0.270000	0.250000	0.190000	0.260000	0.290000	0.1800
test_BACC	0.630000	0.63000	0.680000	0.690000	0.750000	0.660000	0.670000	0.7400
test_TSS	0.270000	0.26000	0.360000	0.370000	0.500000	0.330000	0.340000	0.4800
test_HSS	0.290000	0.27000	0.380000	0.370000	0.530000	0.360000	0.360000	0.5400

In [277]:
```
final_result = {'Type of classifier':['KNN', 'RF', 'Gaussian NB'],
        'Test Accuracy Mean':[df_knn.loc["test_Accuracy"]["mean"],df_rf.loc["test

df_final_result = pd.DataFrame(final_result)

df_final_result.head()
```

Out[277]:

	Type of classifier	Test Accuracy Mean
0	KNN	0.745
1	RF	0.758
2	Gaussian NB	0.748

In [ ]: