In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Import Data, Select columns required in the model

files = "covid_hospitalization_sample.csv"
table = pd.read_csv(files, encoding = 'unicode_escape')
data = table.loc[:,['SARS-Cov-2 exam result','Patient age quantile','Proteina
            , 'Platelets', 'Eosinophils', 'Basophils', 'Leukocytes', 'Mean co
            'Mean corpuscular hemoglobin concentration', 'Lymphocytes', 'Hemo
             'Influenza A rapid test']]
# Have a look the data information
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1230 entries, 0 to 1229
Data columns (total 20 columns):
SARS-Cov-2 exam result                       1230 non-null object
Patient age quantile                         1230 non-null int64
Proteina C reativa                           175 non-null float64
Neutrophils                                  179 non-null float64
Mean platelet volume                         200 non-null float64
Monocytes                                    202 non-null float64
Red blood cell distribution width            202 non-null float64
Red blood Cells                              202 non-null float64
Platelets                                    202 non-null float64
Eosinophils                                  202 non-null float64
Basophils                                    202 non-null float64
Leukocytes                                   202 non-null float64
Mean corpuscular hemoglobin                  202 non-null float64
Mean corpuscular volume                      202 non-null float64
Mean corpuscular hemoglobin concentration    202 non-null float64
Lymphocytes                                  202 non-null float64
Hemoglobin                                   202 non-null float64
Hematocrit                                   202 non-null float64
Influenza B rapid test                       275 non-null object
Influenza A rapid test                       275 non-null object
dtypes: float64(16), int64(1), object(3)
memory usage: 192.3+ KB
None
```

Setting correct type to variables

As age column is an interval variable, we will invert it into a categorical variable.

In [3]:
```
data['Patient age quantile'] = data['Patient age quantile'].astype(str)

# Changing output variable(SARS-Cov-2 exam result) into binary 0/1
data['SARS-Cov-2 exam result'].value_counts ()
```

Out[3]:
```
negative    672
positive    558
Name: SARS-Cov-2 exam result, dtype: int64
```

In [4]:
```
SARS_Cov_2_result_map = {'positive':1,'negative':0}
data['SARS-Cov-2 exam result']=data['SARS-Cov-2 exam result'].map (SARS_Cov_2_
```

In [5]:
```
# Change output variable into non-object type
data['SARS-Cov-2 exam result'].astype(int)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1230 entries, 0 to 1229
Data columns (total 20 columns):
SARS-Cov-2 exam result                          1230 non-null int64
Patient age quantile                            1230 non-null object
Proteina C reativa                              175 non-null float64
Neutrophils                                     179 non-null float64
Mean platelet volume                            200 non-null float64
Monocytes                                       202 non-null float64
Red blood cell distribution width               202 non-null float64
Red blood Cells                                 202 non-null float64
Platelets                                       202 non-null float64
Eosinophils                                     202 non-null float64
Basophils                                       202 non-null float64
Leukocytes                                      202 non-null float64
Mean corpuscular hemoglobin                     202 non-null float64
Mean corpuscular volume                         202 non-null float64
Mean corpuscular hemoglobin concentration       202 non-null float64
Lymphocytes                                     202 non-null float64
Hemoglobin                                      202 non-null float64
Hematocrit                                      202 non-null float64
Influenza B rapid test                          275 non-null object
Influenza A rapid test                          275 non-null object
dtypes: float64(16), int64(1), object(3)
memory usage: 192.3+ KB
```

Imputation of missing values

We imputate the missing values with the columns except age quantile, influenza b rapid test and influenza a rapid test with its mean.

As influenza b rapid test and influenza a rapid test are categorical variables, we can imputate the missing values with its most frequent values.

In [6]:

```
# Imputate the missing values of the columns except age quantile, influenza b
data1 = data.copy()

data1['Proteina C reativa'].fillna(data1['Proteina C reativa'].mean(), inplac
data1['Neutrophils'].fillna(data1['Neutrophils'].mean(), inplace = True)
data1['Mean platelet volume'].fillna(data1['Mean platelet volume'].mean(), in
data1['Monocytes'].fillna(data1['Monocytes'].mean(), inplace = True)
data1['Red blood cell distribution width'].fillna(data1['Red blood cell distr
data1['Red blood Cells'].fillna(data1['Red blood Cells'].mean(), inplace = Tr
data1['Eosinophils'].fillna(data1['Eosinophils'].mean(), inplace = True)
data1['Basophils'].fillna(data1['Basophils'].mean(), inplace = True)
data1['Leukocytes'].fillna(data1['Leukocytes'].mean(), inplace = True)
data1['Mean corpuscular hemoglobin'].fillna(data1['Mean corpuscular hemoglobi
data1['Mean corpuscular volume'].fillna(data1['Mean corpuscular volume'].mean
data1['Mean corpuscular hemoglobin concentration'].fillna(data1['Mean corpusc
data1['Lymphocytes'].fillna(data1['Lymphocytes'].mean(), inplace = True)
data1['Hemoglobin'].fillna(data1['Hemoglobin'].mean(), inplace = True)
data1['Hematocrit'].fillna(data1['Hematocrit'].mean(), inplace = True)
data1['Platelets'].fillna(data1['Platelets'].mean(), inplace = True)

# Imputate influenza b rapid test and influenza a rapid test with its most fr

data1 = data1.fillna(data1['Influenza B rapid test'].value_counts().index[0])
data1 = data1.fillna(data1['Influenza A rapid test'].value_counts().index[0])
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1230 entries, 0 to 1229
```

```
Data columns (total 20 columns):
SARS-Cov-2 exam result                            1230 non-null int64
Patient age quantile                              1230 non-null object
Proteina C reativa                                1230 non-null float64
Neutrophils                                       1230 non-null float64
Mean platelet volume                              1230 non-null float64
Monocytes                                         1230 non-null float64
Red blood cell distribution width                 1230 non-null float64
Red blood Cells                                   1230 non-null float64
Platelets                                         1230 non-null float64
Eosinophils                                       1230 non-null float64
Basophils                                         1230 non-null float64
Leukocytes                                        1230 non-null float64
Mean corpuscular hemoglobin                       1230 non-null float64
Mean corpuscular volume                           1230 non-null float64
Mean corpuscular hemoglobin concentration         1230 non-null float64
Lymphocytes                                       1230 non-null float64
Hemoglobin                                        1230 non-null float64
Hematocrit                                        1230 non-null float64
Influenza B rapid test                            1230 non-null object
Influenza A rapid test                            1230 non-null object
dtypes: float64(16), int64(1), object(3)
memory usage: 192.3+ KB
```

Formatting categorical variable

In [7]:

```
data1 = pd.get_dummies(data1)
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1230 entries, 0 to 1229
Data columns (total 41 columns):
SARS-Cov-2 exam result                            1230 non-null int64
Proteina C reativa                                1230 non-null float64
Neutrophils                                       1230 non-null float64
Mean platelet volume                              1230 non-null float64
Monocytes                                         1230 non-null float64
Red blood cell distribution width                 1230 non-null float64
Red blood Cells                                   1230 non-null float64
Platelets                                         1230 non-null float64
Eosinophils                                       1230 non-null float64
Basophils                                         1230 non-null float64
Leukocytes                                        1230 non-null float64
Mean corpuscular hemoglobin                       1230 non-null float64
Mean corpuscular volume                           1230 non-null float64
Mean corpuscular hemoglobin concentration         1230 non-null float64
Lymphocytes                                       1230 non-null float64
Hemoglobin                                        1230 non-null float64
Hematocrit                                        1230 non-null float64
Patient age quantile_0                            1230 non-null uint8
Patient age quantile_1                            1230 non-null uint8
Patient age quantile_10                           1230 non-null uint8
Patient age quantile_11                           1230 non-null uint8
Patient age quantile_12                           1230 non-null uint8
Patient age quantile_13                           1230 non-null uint8
Patient age quantile_14                           1230 non-null uint8
Patient age quantile_15                           1230 non-null uint8
Patient age quantile_16                           1230 non-null uint8
Patient age quantile_17                           1230 non-null uint8
Patient age quantile_18                           1230 non-null uint8
Patient age quantile_19                           1230 non-null uint8
Patient age quantile_2                            1230 non-null uint8
Patient age quantile_3                            1230 non-null uint8
Patient age quantile_4                            1230 non-null uint8
Patient age quantile_5                            1230 non-null uint8
Patient age quantile_6                            1230 non-null uint8
Patient age quantile_7                            1230 non-null uint8
Patient age quantile_8                            1230 non-null uint8
```

```
Patient age quantile_9                          1230 non-null uint8
Influenza B rapid test_negative                 1230 non-null uint8
Influenza B rapid test_positive                 1230 non-null uint8
Influenza A rapid test_negative                 1230 non-null uint8
Influenza A rapid test_positive                 1230 non-null uint8
dtypes: float64(16), int64(1), uint8(24)
memory usage: 192.3 KB
```

Data Split & Partitioning

In [8]:

```
# target/input split
from sklearn.model_selection import train_test_split

y = data1['SARS-Cov-2 exam result']
x = data1.drop(['SARS-Cov-2 exam result'], axis = 1)
x_mat = x.to_numpy()

rs = 10

x_train, x_test, y_train, y_test = train_test_split(x_mat, y, test_size = 0.3
```

TASK 1

In [9]:

```
# Find proportion of covid positive cases
data1['SARS-Cov-2 exam result'].value_counts (1)
```

Out[9]:

```
0    0.546341
1    0.453659
Name: SARS-Cov-2 exam result, dtype: float64
```

> Hence, from above we get that there were 45.36% of COVID-positive cases.

TASK 2

Building a decision tree using the default setting

In [10]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score

rs = 10
# simple decision tree training
model = DecisionTreeClassifier(random_state=rs)
model.fit(x_train, y_train)
```

Out[10]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=10, splitter='best')
```

In [11]:

```
#Check the accuracy of the training set
print("Train accuracy:", model.score(x_train, y_train))

#Check the accuracy of the test set
print("Test accuracy:", model.score(x_test, y_test))
```

```
Train accuracy: 0.7502903600464577
```

```
Test accuracy: 0.6720867208672087
```

In [12]:

```
y_pred = model.predict(x_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.71      0.69      0.70       202
           1       0.63      0.65      0.64       167

    accuracy                           0.67       369
   macro avg       0.67      0.67      0.67       369
weighted avg       0.67      0.67      0.67       369
```

Feature importance

In [13]:

```
# grab feature importances from the model and feature name from the original 
importances = model.feature_importances_
feature_names = x.columns
# sort them out in descending order
indices = np.argsort(importances)
indices = np.flip(indices, axis=0)
# limit to 20 features, you can leave this out to print out everything
indices = indices[:20]
for i in indices:
   print(feature_names[i], ':', importances[i])
```

```
Patient age quantile_1 : 0.11933305088194286
Eosinophils : 0.10927416937595254
Influenza B rapid test_negative : 0.09305424090249394
Influenza A rapid test_negative : 0.09301560010020286
Leukocytes : 0.09159425714502227
Patient age quantile_2 : 0.06584563986622755
Patient age quantile_0 : 0.06295887839143101
Patient age quantile_17 : 0.039941548691240954
Hematocrit : 0.034458993919989
Patient age quantile_19 : 0.033307934536824534
Red blood Cells : 0.033043466348675964
Monocytes : 0.030452064082854287
Proteina C reativa : 0.023285426984821136
Patient age quantile_5 : 0.022178608480763626
Patient age quantile_4 : 0.020608885577744452
Lymphocytes : 0.018720363333874494
Mean platelet volume : 0.0158783292398496
Patient age quantile_3 : 0.012600018810485251
Patient age quantile_11 : 0.012411817573612406
Mean corpuscular hemoglobin concentration : 0.0121020530643229
```

Visualising decision tree

In [17]:

```
import pydot
from io import StringIO
from sklearn.tree import export_graphviz

# visualize
dotfile = StringIO()
export_graphviz(model, out_file=dotfile, feature_names=x.columns)
graph = pydot.graph_from_dot_data(dotfile.getvalue())
graph[0].write_png("dt_viz.png") # saved in the following file - will return
```

In [15]:

```
# Calculating the number of nodes
```

```
n_nodes = model.tree_.node_count
n_nodes
```

Out[15]: 107

In [16]:
```
print(model.tree_.max_depth)
```

30

Build a decision tree tuned with GridSearchCV

To perform GridSearchCV here, we will focus on three hyperparameters here :

1. Criterion
2. Max depth
3. Min samples leaf

In [18]:
```
from sklearn.model_selection import GridSearchCV
```

In [19]:
```
# grid search CV
params = {'criterion': ['gini', 'entropy'],
          'max_depth': range(1, 20),
          'min_samples_leaf': range(0, 25, 5)[1:]}

cv_1 = GridSearchCV(param_grid=params, estimator=DecisionTreeClassifier(randon
cv_1.fit(x_train, y_train)
```

Out[19]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight=None,
                                              criterion='gini', max_depth=Non
e,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort=False, random_state=10,
                                              splitter='best'),
             iid='warn', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(1, 20),
                         'min_samples_leaf': range(5, 25, 5)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [20]:
```
print(cv_1.best_params_)    #best hyperparameters
```

{'criterion': 'gini', 'max_depth': 18, 'min_samples_leaf': 15}

In [21]:
```
result_set = cv_1.cv_results_
# returns all the combinations of hyperparameters and their respective score
print(result_set)
```

```
{'mean_fit_time': array([0.00255549, 0.00181017, 0.00188367, 0.0049011 , 0.001
86734,
       0.00171857, 0.00141401, 0.00138674, 0.00178881, 0.00226407,
       0.00170779, 0.00150638, 0.00178082, 0.00178614, 0.00168676,
       0.00173192, 0.00192823, 0.00200882, 0.00192091, 0.00191073,
```

```
        0.00220048, 0.00218959, 0.00204303, 0.00205574, 0.0022542 ,
        0.0022635 , 0.0022155 , 0.00218568, 0.00243125, 0.00244498,
        0.0024467 , 0.00222256, 0.00376537, 0.00264637, 0.00240288,
        0.00235164, 0.00260804, 0.00341153, 0.0027056 , 0.00244331,
        0.0026989 , 0.00263069, 0.00262399, 0.00238895, 0.00282829,
        0.00274174, 0.0026475 , 0.00254784, 0.00294843, 0.00276306,
        0.00267541, 0.00255251, 0.00298042, 0.00281858, 0.00273862,
        0.00260727, 0.00311732, 0.00289376, 0.00302045, 0.00269272,
        0.00344141, 0.00296564, 0.00285954, 0.00267272, 0.00311038,
        0.00296268, 0.00295842, 0.00278082, 0.0030869 , 0.00307772,
        0.0029731 , 0.00273652, 0.00321214, 0.00299706, 0.00302687,
        0.00283353, 0.00116613, 0.00118008, 0.00115533, 0.00113485,
        0.00135779, 0.00137293, 0.00142379, 0.00161228, 0.00171072,
        0.00174851, 0.00167754, 0.00153811, 0.00188158, 0.00185661,
        0.00184412, 0.00175149, 0.00217104, 0.00201409, 0.00202255,
        0.00212862, 0.00235126, 0.00228701, 0.00220685, 0.00207992,
        0.00240967, 0.00251994, 0.00235057, 0.00239253, 0.0027333 ,
        0.00256763, 0.00250771, 0.00227938, 0.0027077 , 0.00263813,
        0.00252206, 0.00238366, 0.00296283, 0.00276697, 0.00286012,
        0.00265768, 0.00302989, 0.00280881, 0.00285618, 0.00249209,
        0.00304148, 0.00283601, 0.00278785, 0.00262671, 0.00312324,
        0.00294116, 0.00279677, 0.00256081, 0.00317428, 0.00297582,
        0.00282619, 0.00272779, 0.00323343, 0.00294144, 0.00293851,
        0.00274508, 0.0032753 , 0.0031673 , 0.00297761, 0.00280936,
        0.00372951, 0.00325551, 0.00301547, 0.00286744, 0.00339007,
        0.00317829, 0.00306537, 0.00284429, 0.00341117, 0.00315504,
        0.0030097 , 0.00275929]), 'std_fit_time': array([7.61295942e-04, 3.8309
7769e-04, 4.27380290e-04, 4.46961166e-03,
        2.54209595e-04, 1.87404216e-04, 1.41960629e-04, 1.68511992e-04,
        4.07919875e-04, 3.25387573e-04, 1.27046771e-04, 7.83571139e-05,
        1.59113710e-04, 1.56829820e-04, 7.95824503e-05, 1.57089614e-04,
        1.43396208e-04, 2.13906359e-04, 9.17990452e-05, 2.53612418e-04,
        1.69967408e-04, 2.59394611e-04, 6.85261160e-05, 1.83010269e-04,
        1.17265290e-04, 1.47136093e-04, 7.99044671e-05, 1.13723123e-04,
        6.36442041e-05, 1.75578895e-04, 2.27121693e-04, 1.64474693e-04,
        1.01321651e-03, 2.23892998e-04, 1.42296318e-04, 1.18299698e-04,
        7.71846375e-05, 7.72107597e-04, 2.37880157e-04, 1.32189676e-04,
        1.26535908e-04, 1.62326058e-04, 2.20241137e-04, 3.67863720e-05,
        1.57851504e-04, 1.31522548e-04, 1.36290379e-04, 1.64884098e-04,
        3.18693218e-04, 1.25136755e-04, 1.10142378e-04, 9.08979705e-05,
        1.45840853e-04, 1.04644791e-04, 1.27116201e-04, 1.02077686e-04,
        2.64999929e-04, 1.80099982e-04, 3.45027685e-04, 1.26687092e-04,
        3.99387231e-04, 1.94652930e-04, 1.35365108e-04, 1.72795279e-04,
        1.05162943e-04, 1.58111904e-04, 4.34718478e-04, 1.86722734e-04,
        8.52889741e-05, 3.03802089e-04, 1.85685540e-04, 1.27662710e-04,
        1.82285904e-04, 1.59200289e-04, 1.89596540e-04, 1.02755976e-04,
        1.83495013e-04, 8.55486395e-05, 1.77191681e-04, 1.32902799e-04,
        6.58450652e-05, 9.52227956e-05, 1.70892504e-04, 6.05670763e-04,
        1.66291927e-04, 3.55821239e-04, 1.62262418e-04, 5.32341279e-05,
        1.09645978e-04, 6.73640124e-05, 1.32874454e-04, 5.62316221e-05,
        2.85448161e-04, 4.79287824e-05, 8.72132295e-05, 2.35192417e-04,
        1.47151300e-04, 2.33904890e-04, 1.46000296e-04, 6.18783076e-05,
        4.12033178e-05, 4.55686309e-04, 8.78774142e-05, 3.20754927e-04,
        2.11568946e-04, 8.61862685e-05, 1.89200802e-04, 6.34607560e-05,
        9.29152075e-05, 8.22067434e-05, 8.01270221e-05, 8.41453309e-05,
        2.30749995e-04, 1.31401659e-04, 4.87486159e-04, 1.62794135e-04,
        2.14009278e-04, 9.68311751e-05, 3.18094670e-04, 3.65448633e-05,
        7.76707904e-05, 8.05436716e-05, 1.14239448e-04, 1.24906071e-04,
        1.17316516e-04, 1.17276761e-04, 7.41141516e-05, 1.15502904e-04,
        1.84209343e-04, 9.39324573e-05, 9.74379981e-05, 1.50476005e-04,
        1.47973795e-04, 8.69753493e-05, 1.34529360e-04, 8.57007215e-05,
        9.25752172e-05, 1.56792447e-04, 1.19008276e-04, 1.41160674e-04,
        3.93786911e-04, 1.91404009e-04, 1.18413357e-04, 2.23661953e-04,
        1.67793738e-04, 1.90195724e-04, 1.24305212e-04, 1.30735053e-04,
        1.10273902e-04, 9.54222402e-05, 1.32507513e-04, 1.46743296e-04]), 'mean
_score_time': array([0.00078909, 0.00068657, 0.00215411, 0.00113328, 0.0005480
1,
        0.00050669, 0.00038762, 0.00036762, 0.00047202, 0.00051827,
        0.00043194, 0.00039098, 0.00036862, 0.00037251, 0.00035706,
```

```
        0.00037701, 0.00038283, 0.00037935, 0.00038564, 0.00037045,
        0.00037768, 0.0003917 , 0.00037224, 0.00036528, 0.00037196,
        0.00039482, 0.00040176, 0.00037346, 0.00038323, 0.00041289,
        0.00040107, 0.00038106, 0.00055907, 0.00039554, 0.00037937,
        0.00038519, 0.00037234, 0.00047245, 0.00040379, 0.00040479,
        0.00036345, 0.00040612, 0.00038428, 0.00039885, 0.00039468,
        0.00038817, 0.00038078, 0.00039465, 0.00038552, 0.00038261,
        0.00036473, 0.00039372, 0.00038059, 0.00039964, 0.00038309,
        0.00040023, 0.00050151, 0.0003834 , 0.00042102, 0.00038548,
        0.00042005, 0.000386  , 0.00037796, 0.00038767, 0.00040891,
        0.00043473, 0.00040314, 0.0004118 , 0.00038757, 0.00037577,
        0.00038817, 0.00039749, 0.00038879, 0.00037355, 0.0003861 ,
        0.00042255, 0.00036721, 0.00036378, 0.00036342, 0.00035534,
        0.00037591, 0.00037658, 0.00037334, 0.00039489, 0.00038755,
        0.00038033, 0.00043116, 0.00035398, 0.00039043, 0.00036671,
        0.00037453, 0.00035813, 0.00036464, 0.00036032, 0.00035563,
        0.0004283 , 0.00036552, 0.00037103, 0.00037813, 0.00035782,
        0.00034704, 0.00044441, 0.00039003, 0.00041702, 0.00039821,
        0.00037887, 0.00037074, 0.00036025, 0.0003572 , 0.00036442,
        0.00036147, 0.00036652, 0.0003902 , 0.00038099, 0.00041647,
        0.00048976, 0.00037961, 0.00037599, 0.00040321, 0.00038731,
        0.00039496, 0.000367  , 0.00039294, 0.00036371, 0.00037546,
        0.00037029, 0.00035408, 0.00038664, 0.00038967, 0.00038924,
        0.00035279, 0.00037806, 0.00037763, 0.00036097, 0.00037208,
        0.00036469, 0.00036416, 0.000402  , 0.00036862, 0.00037882,
        0.00047302, 0.00039356, 0.00036867, 0.00040126, 0.00037689,
        0.00036736, 0.00037496, 0.00037427, 0.00039654, 0.00036409,
        0.00035009, 0.00036066]), 'std_score_time': array([2.10171435e-04, 2.94
313401e-04, 4.12199516e-03, 4.14849041e-04,
        1.48359445e-04, 1.38921938e-04, 2.49381275e-05, 2.24489672e-05,
        1.67439038e-04, 8.63940769e-05, 6.28959825e-05, 5.58759208e-05,
        3.73410156e-05, 3.69096602e-05, 9.64414204e-06, 5.16563495e-05,
        4.79606751e-05, 4.61533602e-05, 5.71200613e-05, 5.26090787e-05,
        4.05721527e-05, 6.15749342e-05, 4.48374077e-05, 1.98276121e-05,
        4.36249258e-05, 7.39754108e-05, 1.06410646e-04, 2.24544739e-05,
        4.69529906e-05, 7.99888328e-05, 6.05760383e-05, 6.44897401e-05,
        1.34610904e-04, 3.22271020e-05, 3.66055684e-05, 3.58620220e-05,
        3.05996198e-05, 8.84078595e-05, 4.35812207e-05, 7.09745160e-05,
        1.58189245e-05, 8.94219195e-05, 4.42435510e-05, 7.31602046e-05,
        4.35459901e-05, 2.92576513e-05, 2.75514203e-05, 4.33246708e-05,
        6.02685197e-05, 4.53613261e-05, 2.09034229e-05, 5.81071812e-05,
        2.56465814e-05, 9.10022698e-05, 3.01958425e-05, 7.52312476e-05,
        2.80831204e-04, 4.91478317e-05, 8.60428473e-05, 3.88715592e-05,
        6.01597496e-05, 4.25376431e-05, 3.26650943e-05, 5.87828247e-05,
        6.64803563e-05, 1.91761419e-04, 9.04949399e-05, 6.49606236e-05,
        5.10396435e-05, 2.81897986e-05, 5.14425807e-05, 7.95008388e-05,
        4.23603530e-05, 3.25965895e-05, 4.80518673e-05, 7.15506073e-05,
        4.52494093e-05, 2.95042004e-05, 3.52608182e-05, 1.66886874e-05,
        7.21153777e-05, 7.48682362e-05, 3.51973094e-05, 6.44545613e-05,
        5.16835446e-05, 6.07702534e-05, 2.02107451e-04, 1.11619879e-05,
        8.58048692e-05, 2.90674100e-05, 5.39976186e-05, 1.45033963e-05,
        2.96857844e-05, 2.34240764e-05, 1.05822403e-05, 1.06400856e-04,
        2.42152656e-05, 4.79102476e-05, 6.39245518e-05, 5.77917980e-06,
        5.53336599e-06, 1.78731009e-04, 5.00268370e-05, 1.07959579e-04,
        6.41279323e-05, 3.38481254e-05, 2.89493568e-05, 1.63594066e-05,
        1.00954433e-05, 1.92769199e-05, 1.49446615e-05, 1.89850815e-05,
        5.01779301e-05, 6.10598313e-05, 1.04428034e-04, 3.31405544e-04,
        5.40250187e-05, 4.12933821e-05, 7.60687395e-05, 5.63977876e-05,
        1.05660981e-04, 3.20349853e-05, 9.23529346e-05, 1.73496825e-05,
        4.32320406e-05, 4.69255312e-05, 6.21303447e-06, 1.20317791e-04,
        6.42030025e-05, 7.24204195e-05, 1.08442719e-05, 2.85871830e-05,
        3.97686835e-05, 3.94818575e-05, 3.00369942e-05, 1.01976284e-05,
        1.50049221e-05, 8.37254935e-05, 2.97797022e-05, 4.28985788e-05,
        9.44326603e-05, 5.30474695e-05, 2.27476939e-05, 8.80146331e-05,
        3.66532103e-05, 1.91192674e-05, 2.73629442e-05, 2.81305231e-05,
        8.00883318e-05, 2.17257941e-05, 5.83439054e-06, 2.11699186e-05]), 'para
m_criterion': masked_array(data=['gini', 'gini', 'gini', 'gini', 'gini', 'gin
i', 'gini',
                   'gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
```

```
                    'gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                    'gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                    'gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                    'gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                    'gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                    'gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                    'gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                    'gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                    'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy'],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object), 'param_max_depth': masked_array(data=[1, 1, 1, 1,
2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5,
                    5, 5, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 8, 9, 9, 9, 9,
                    10, 10, 10, 10, 11, 11, 11, 11, 12, 12, 12, 12, 13, 13,
                    13, 13, 14, 14, 14, 14, 15, 15, 15, 15, 16, 16, 16, 16,
                    17, 17, 17, 17, 18, 18, 18, 18, 19, 19, 19, 19, 1, 1,
                    1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5,
                    6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 8, 9, 9, 9, 9, 10, 10,
                    10, 10, 11, 11, 11, 11, 12, 12, 12, 12, 13, 13, 13, 13,
                    14, 14, 14, 14, 15, 15, 15, 15, 16, 16, 16, 16, 17, 17,
                    17, 17, 18, 18, 18, 18, 19, 19, 19, 19],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
```

```
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object), 'param_min_samples_leaf': masked_array(data=[5, 10,
15, 20, 5, 10, 15, 20, 5, 10, 15, 20, 5, 10, 15,
                     20, 5, 10, 15, 20, 5, 10, 15, 20, 5, 10, 15, 20, 5, 10,
                     15, 20, 5, 10, 15, 20, 5, 10, 15, 20, 5, 10, 15, 20, 5,
                     10, 15, 20, 5, 10, 15, 20, 5, 10, 15, 20, 5, 10, 15,
                     20, 5, 10, 15, 20, 5, 10, 15, 20, 5, 10, 15, 20, 5, 10,
                     15, 20, 5, 10, 15, 20, 5, 10, 15, 20, 5, 10, 15, 20, 5,
                     10, 15, 20, 5, 10, 15, 20, 5, 10, 15, 20, 5, 10, 15,
                     20, 5, 10, 15, 20, 5, 10, 15, 20, 5, 10, 15, 20, 5, 10,
                     15, 20, 5, 10, 15, 20, 5, 10, 15, 20, 5, 10, 15, 20, 5,
                     10, 15, 20, 5, 10, 15, 20, 5, 10, 15, 20, 5, 10, 15,
                     20, 5, 10, 15, 20],
               mask=[False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object), 'params': [{'criterion': 'gini', 'max_depth': 1, 'm
in_samples_leaf': 5}, {'criterion': 'gini', 'max_depth': 1, 'min_samples_lea
f': 10}, {'criterion': 'gini', 'max_depth': 1, 'min_samples_leaf': 15}, {'crit
erion': 'gini', 'max_depth': 1, 'min_samples_leaf': 20}, {'criterion': 'gini',
'max_depth': 2, 'min_samples_leaf': 5}, {'criterion': 'gini', 'max_depth': 2,
'min_samples_leaf': 10}, {'criterion': 'gini', 'max_depth': 2, 'min_samples_le
af': 15}, {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 20}, {'cri
terion': 'gini', 'max_depth': 3, 'min_samples_leaf': 5}, {'criterion': 'gini',
'max_depth': 3, 'min_samples_leaf': 10}, {'criterion': 'gini', 'max_depth': 3,
'min_samples_leaf': 15}, {'criterion': 'gini', 'max_depth': 3, 'min_samples_le
af': 20}, {'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 5}, {'crit
erion': 'gini', 'max_depth': 4, 'min_samples_leaf': 10}, {'criterion': 'gini',
'max_depth': 4, 'min_samples_leaf': 15}, {'criterion': 'gini', 'max_depth': 4,
'min_samples_leaf': 20}, {'criterion': 'gini', 'max_depth': 5, 'min_samples_le
af': 5}, {'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 10}, {'crit
erion': 'gini', 'max_depth': 5, 'min_samples_leaf': 15}, {'criterion': 'gini',
'max_depth': 5, 'min_samples_leaf': 20}, {'criterion': 'gini', 'max_depth': 6,
'min_samples_leaf': 5}, {'criterion': 'gini', 'max_depth': 6, 'min_samples_lea
f': 10}, {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 15}, {'crit
erion': 'gini', 'max_depth': 6, 'min_samples_leaf': 20}, {'criterion': 'gini',
'max_depth': 7, 'min_samples_leaf': 5}, {'criterion': 'gini', 'max_depth': 7,
'min_samples_leaf': 10}, {'criterion': 'gini', 'max_depth': 7, 'min_samples_le
af': 15}, {'criterion': 'gini', 'max_depth': 7, 'min_samples_leaf': 20}, {'cri
terion': 'gini', 'max_depth': 8, 'min_samples_leaf': 5}, {'criterion': 'gini',
'max_depth': 8, 'min_samples_leaf': 10}, {'criterion': 'gini', 'max_depth': 8,
'min_samples_leaf': 15}, {'criterion': 'gini', 'max_depth': 8, 'min_samples_le
af': 20}, {'criterion': 'gini', 'max_depth': 9, 'min_samples_leaf': 5}, {'crit
erion': 'gini', 'max_depth': 9, 'min_samples_leaf': 10}, {'criterion': 'gini',
'max_depth': 9, 'min_samples_leaf': 15}, {'criterion': 'gini', 'max_depth': 9,
'min_samples_leaf': 20}, {'criterion': 'gini', 'max_depth': 10, 'min_samples_l
```

```
eaf': 5}, {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 10}, {'cr
iterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 15}, {'criterion': 'gin
i', 'max_depth': 10, 'min_samples_leaf': 20}, {'criterion': 'gini', 'max_dept
h': 11, 'min_samples_leaf': 5}, {'criterion': 'gini', 'max_depth': 11, 'min_sa
mples_leaf': 10}, {'criterion': 'gini', 'max_depth': 11, 'min_samples_leaf': 1
5}, {'criterion': 'gini', 'max_depth': 11, 'min_samples_leaf': 20}, {'criterio
n': 'gini', 'max_depth': 12, 'min_samples_leaf': 5}, {'criterion': 'gini', 'ma
x_depth': 12, 'min_samples_leaf': 10}, {'criterion': 'gini', 'max_depth': 12,
'min_samples_leaf': 15}, {'criterion': 'gini', 'max_depth': 12, 'min_samples_l
eaf': 20}, {'criterion': 'gini', 'max_depth': 13, 'min_samples_leaf': 5}, {'cr
iterion': 'gini', 'max_depth': 13, 'min_samples_leaf': 10}, {'criterion': 'gin
i', 'max_depth': 13, 'min_samples_leaf': 15}, {'criterion': 'gini', 'max_dept
h': 13, 'min_samples_leaf': 20}, {'criterion': 'gini', 'max_depth': 14, 'min_s
amples_leaf': 5}, {'criterion': 'gini', 'max_depth': 14, 'min_samples_leaf': 1
0}, {'criterion': 'gini', 'max_depth': 14, 'min_samples_leaf': 15}, {'criterio
n': 'gini', 'max_depth': 14, 'min_samples_leaf': 20}, {'criterion': 'gini', 'm
ax_depth': 15, 'min_samples_leaf': 5}, {'criterion': 'gini', 'max_depth': 15,
'min_samples_leaf': 10}, {'criterion': 'gini', 'max_depth': 15, 'min_samples_l
eaf': 15}, {'criterion': 'gini', 'max_depth': 15, 'min_samples_leaf': 20}, {'c
riterion': 'gini', 'max_depth': 16, 'min_samples_leaf': 5}, {'criterion': 'gin
i', 'max_depth': 16, 'min_samples_leaf': 10}, {'criterion': 'gini', 'max_dept
h': 16, 'min_samples_leaf': 15}, {'criterion': 'gini', 'max_depth': 16, 'min_s
amples_leaf': 20}, {'criterion': 'gini', 'max_depth': 17, 'min_samples_leaf':
5}, {'criterion': 'gini', 'max_depth': 17, 'min_samples_leaf': 10}, {'criterio
n': 'gini', 'max_depth': 17, 'min_samples_leaf': 15}, {'criterion': 'gini', 'm
ax_depth': 17, 'min_samples_leaf': 20}, {'criterion': 'gini', 'max_depth': 18,
'min_samples_leaf': 5}, {'criterion': 'gini', 'max_depth': 18, 'min_samples_le
af': 10}, {'criterion': 'gini', 'max_depth': 18, 'min_samples_leaf': 15}, {'cr
iterion': 'gini', 'max_depth': 18, 'min_samples_leaf': 20}, {'criterion': 'gin
i', 'max_depth': 19, 'min_samples_leaf': 5}, {'criterion': 'gini', 'max_dept
h': 19, 'min_samples_leaf': 10}, {'criterion': 'gini', 'max_depth': 19, 'min_s
amples_leaf': 15}, {'criterion': 'gini', 'max_depth': 19, 'min_samples_leaf':
20}, {'criterion': 'entropy', 'max_depth': 1, 'min_samples_leaf': 5}, {'criter
ion': 'entropy', 'max_depth': 1, 'min_samples_leaf': 10}, {'criterion': 'entro
py', 'max_depth': 1, 'min_samples_leaf': 15}, {'criterion': 'entropy', 'max_de
pth': 1, 'min_samples_leaf': 20}, {'criterion': 'entropy', 'max_depth': 2, 'mi
n_samples_leaf': 5}, {'criterion': 'entropy', 'max_depth': 2, 'min_samples_lea
f': 10}, {'criterion': 'entropy', 'max_depth': 2, 'min_samples_leaf': 15}, {'c
riterion': 'entropy', 'max_depth': 2, 'min_samples_leaf': 20}, {'criterion':
'entropy', 'max_depth': 3, 'min_samples_leaf': 5}, {'criterion': 'entropy', 'm
ax_depth': 3, 'min_samples_leaf': 10}, {'criterion': 'entropy', 'max_depth':
3, 'min_samples_leaf': 15}, {'criterion': 'entropy', 'max_depth': 3, 'min_samp
les_leaf': 20}, {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf':
5}, {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 10}, {'criter
ion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 15}, {'criterion': 'entro
py', 'max_depth': 4, 'min_samples_leaf': 20}, {'criterion': 'entropy', 'max_de
pth': 5, 'min_samples_leaf': 5}, {'criterion': 'entropy', 'max_depth': 5, 'min
_samples_leaf': 10}, {'criterion': 'entropy', 'max_depth': 5, 'min_samples_lea
f': 15}, {'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 20}, {'c
riterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 5}, {'criterion': 'e
ntropy', 'max_depth': 6, 'min_samples_leaf': 10}, {'criterion': 'entropy', 'ma
x_depth': 6, 'min_samples_leaf': 15}, {'criterion': 'entropy', 'max_depth': 6,
'min_samples_leaf': 20}, {'criterion': 'entropy', 'max_depth': 7, 'min_samples
_leaf': 5}, {'criterion': 'entropy', 'max_depth': 7, 'min_samples_leaf': 10},
{'criterion': 'entropy', 'max_depth': 7, 'min_samples_leaf': 15}, {'criterio
n': 'entropy', 'max_depth': 7, 'min_samples_leaf': 20}, {'criterion': 'entrop
y', 'max_depth': 8, 'min_samples_leaf': 5}, {'criterion': 'entropy', 'max_dept
h': 8, 'min_samples_leaf': 10}, {'criterion': 'entropy', 'max_depth': 8, 'min_
samples_leaf': 15}, {'criterion': 'entropy', 'max_depth': 8, 'min_samples_lea
f': 20}, {'criterion': 'entropy', 'max_depth': 9, 'min_samples_leaf': 5}, {'cr
iterion': 'entropy', 'max_depth': 9, 'min_samples_leaf': 10}, {'criterion': 'e
ntropy', 'max_depth': 9, 'min_samples_leaf': 15}, {'criterion': 'entropy', 'ma
x_depth': 9, 'min_samples_leaf': 20}, {'criterion': 'entropy', 'max_depth': 1
0, 'min_samples_leaf': 5}, {'criterion': 'entropy', 'max_depth': 10, 'min_samp
les_leaf': 10}, {'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf':
15}, {'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 20}, {'crit
erion': 'entropy', 'max_depth': 11, 'min_samples_leaf': 5}, {'criterion': 'ent
ropy', 'max_depth': 11, 'min_samples_leaf': 10}, {'criterion': 'entropy', 'max
_depth': 11, 'min_samples_leaf': 15}, {'criterion': 'entropy', 'max_depth': 1
```

```
1, 'min_samples_leaf': 20}, {'criterion': 'entropy', 'max_depth': 12, 'min_sam
ples_leaf': 5}, {'criterion': 'entropy', 'max_depth': 12, 'min_samples_leaf':
10}, {'criterion': 'entropy', 'max_depth': 12, 'min_samples_leaf': 15}, {'crit
erion': 'entropy', 'max_depth': 12, 'min_samples_leaf': 20}, {'criterion': 'en
tropy', 'max_depth': 13, 'min_samples_leaf': 5}, {'criterion': 'entropy', 'max
_depth': 13, 'min_samples_leaf': 10}, {'criterion': 'entropy', 'max_depth': 1
3, 'min_samples_leaf': 15}, {'criterion': 'entropy', 'max_depth': 13, 'min_sam
ples_leaf': 20}, {'criterion': 'entropy', 'max_depth': 14, 'min_samples_leaf':
5}, {'criterion': 'entropy', 'max_depth': 14, 'min_samples_leaf': 10}, {'crite
rion': 'entropy', 'max_depth': 14, 'min_samples_leaf': 15}, {'criterion': 'ent
ropy', 'max_depth': 14, 'min_samples_leaf': 20}, {'criterion': 'entropy', 'max
_depth': 15, 'min_samples_leaf': 5}, {'criterion': 'entropy', 'max_depth': 15,
'min_samples_leaf': 10}, {'criterion': 'entropy', 'max_depth': 15, 'min_sample
s_leaf': 15}, {'criterion': 'entropy', 'max_depth': 15, 'min_samples_leaf': 2
0}, {'criterion': 'entropy', 'max_depth': 16, 'min_samples_leaf': 5}, {'criter
ion': 'entropy', 'max_depth': 16, 'min_samples_leaf': 10}, {'criterion': 'entr
opy', 'max_depth': 16, 'min_samples_leaf': 15}, {'criterion': 'entropy', 'max_
depth': 16, 'min_samples_leaf': 20}, {'criterion': 'entropy', 'max_depth': 17,
'min_samples_leaf': 5}, {'criterion': 'entropy', 'max_depth': 17, 'min_samples
_leaf': 10}, {'criterion': 'entropy', 'max_depth': 17, 'min_samples_leaf': 1
5}, {'criterion': 'entropy', 'max_depth': 17, 'min_samples_leaf': 20}, {'crite
rion': 'entropy', 'max_depth': 18, 'min_samples_leaf': 5}, {'criterion': 'entr
opy', 'max_depth': 18, 'min_samples_leaf': 10}, {'criterion': 'entropy', 'max_
depth': 18, 'min_samples_leaf': 15}, {'criterion': 'entropy', 'max_depth': 18,
'min_samples_leaf': 20}, {'criterion': 'entropy', 'max_depth': 19, 'min_sample
s_leaf': 5}, {'criterion': 'entropy', 'max_depth': 19, 'min_samples_leaf': 1
0}, {'criterion': 'entropy', 'max_depth': 19, 'min_samples_leaf': 15}, {'crite
rion': 'entropy', 'max_depth': 19, 'min_samples_leaf': 20}], 'split0_test_scor
e': array([0.54022989, 0.54022989, 0.54022989, 0.54022989, 0.54022989,
       0.54022989, 0.54022989, 0.54022989, 0.57471264, 0.57471264,
       0.57471264, 0.57471264, 0.57471264, 0.57471264, 0.57471264,
       0.57471264, 0.5862069 , 0.5862069 , 0.5862069 , 0.5862069 ,
       0.62068966, 0.62068966, 0.62068966, 0.5862069 , 0.62068966,
       0.62068966, 0.62068966, 0.5862069 , 0.62068966, 0.62068966,
       0.62068966, 0.5862069 , 0.62068966, 0.62068966, 0.62068966,
       0.5862069 , 0.62068966, 0.62068966, 0.62068966, 0.5862069 ,
       0.64367816, 0.62068966, 0.62068966, 0.59770115, 0.62068966,
       0.62068966, 0.62068966, 0.5862069 , 0.62068966, 0.62068966,
       0.62068966, 0.57471264, 0.64367816, 0.63218391, 0.6091954 ,
       0.59770115, 0.62068966, 0.6091954 , 0.5862069 , 0.6091954 ,
       0.63218391, 0.5862069 , 0.59770115, 0.62068966, 0.5862069 ,
       0.59770115, 0.59770115, 0.62068966, 0.62068966, 0.6091954 ,
       0.62068966, 0.62068966, 0.6091954 , 0.6091954 , 0.62068966,
       0.62068966, 0.54022989, 0.54022989, 0.54022989, 0.54022989,
       0.54022989, 0.54022989, 0.54022989, 0.54022989, 0.6091954 ,
       0.6091954 , 0.6091954 , 0.6091954 , 0.6091954 , 0.6091954 ,
       0.6091954 , 0.6091954 , 0.6091954 , 0.6091954 , 0.6091954 ,
       0.6091954 , 0.6091954 , 0.6091954 , 0.6091954 , 0.6091954 ,
       0.63218391, 0.62068966, 0.62068966, 0.6091954 , 0.63218391,
       0.62068966, 0.62068966, 0.6091954 , 0.63218391, 0.62068966,
       0.62068966, 0.6091954 , 0.64367816, 0.62068966, 0.62068966,
       0.6091954 , 0.64367816, 0.62068966, 0.62068966, 0.6091954 ,
       0.64367816, 0.62068966, 0.62068966, 0.59770115, 0.63218391,
       0.62068966, 0.62068966, 0.57471264, 0.64367816, 0.63218391,
       0.6091954 , 0.5862069 , 0.63218391, 0.6091954 , 0.5862069 ,
       0.6091954 , 0.63218391, 0.5862069 , 0.59770115, 0.6091954 ,
       0.6091954 , 0.59770115, 0.59770115, 0.62068966, 0.62068966,
       0.6091954 , 0.62068966, 0.62068966, 0.63218391, 0.6091954 ,
       0.62068966, 0.62068966]), 'split1_test_score': array([0.54651163, 0.546
51163, 0.54651163, 0.54651163, 0.48837209,
       0.48837209, 0.48837209, 0.48837209, 0.52325581, 0.52325581,
       0.52325581, 0.52325581, 0.54651163, 0.54651163, 0.54651163,
       0.54651163, 0.56976744, 0.56976744, 0.56976744, 0.60465116,
       0.62790698, 0.62790698, 0.62790698, 0.60465116, 0.62790698,
       0.62790698, 0.61627907, 0.54651163, 0.56976744, 0.56976744,
       0.55813953, 0.56976744, 0.59302326, 0.59302326, 0.58139535,
       0.56976744, 0.59302326, 0.59302326, 0.58139535, 0.56976744,
       0.59302326, 0.59302326, 0.58139535, 0.56976744, 0.59302326,
       0.59302326, 0.58139535, 0.58139535, 0.59302326, 0.59302326,
```

```
        0.58139535, 0.60465116, 0.59302326, 0.60465116, 0.60465116,
        0.60465116, 0.59302326, 0.62790698, 0.61627907, 0.60465116,
        0.59302326, 0.62790698, 0.60465116, 0.59302326, 0.61627907,
        0.61627907, 0.60465116, 0.59302326, 0.60465116, 0.61627907,
        0.63953488, 0.59302326, 0.60465116, 0.61627907, 0.63953488,
        0.59302326, 0.54651163, 0.54651163, 0.54651163, 0.54651163,
        0.48837209, 0.48837209, 0.48837209, 0.48837209, 0.52325581,
        0.52325581, 0.52325581, 0.52325581, 0.54651163, 0.54651163,
        0.54651163, 0.54651163, 0.56976744, 0.56976744, 0.56976744,
        0.60465116, 0.62790698, 0.62790698, 0.62790698, 0.60465116,
        0.62790698, 0.62790698, 0.61627907, 0.54651163, 0.56976744,
        0.56976744, 0.55813953, 0.56976744, 0.59302326, 0.59302326,
        0.58139535, 0.56976744, 0.59302326, 0.59302326, 0.58139535,
        0.56976744, 0.59302326, 0.59302326, 0.58139535, 0.56976744,
        0.59302326, 0.59302326, 0.58139535, 0.58139535, 0.59302326,
        0.59302326, 0.58139535, 0.60465116, 0.59302326, 0.60465116,
        0.60465116, 0.60465116, 0.59302326, 0.62790698, 0.61627907,
        0.60465116, 0.59302326, 0.62790698, 0.60465116, 0.59302326,
        0.61627907, 0.61627907, 0.60465116, 0.59302326, 0.60465116,
        0.61627907, 0.63953488, 0.59302326, 0.60465116, 0.61627907,
        0.63953488, 0.59302326]), 'split2_test_score': array([0.54651163, 0.546
51163, 0.54651163, 0.54651163, 0.54651163,
        0.54651163, 0.54651163, 0.54651163, 0.60465116, 0.60465116,
        0.60465116, 0.60465116, 0.60465116, 0.60465116, 0.60465116,
        0.60465116, 0.60465116, 0.60465116, 0.60465116, 0.60465116,
        0.62790698, 0.62790698, 0.62790698, 0.62790698, 0.65116279,
        0.65116279, 0.65116279, 0.61627907, 0.63953488, 0.63953488,
        0.63953488, 0.61627907, 0.63953488, 0.63953488, 0.63953488,
        0.61627907, 0.63953488, 0.63953488, 0.63953488, 0.61627907,
        0.63953488, 0.63953488, 0.63953488, 0.63953488, 0.63953488,
        0.63953488, 0.63953488, 0.6627907 , 0.65116279, 0.63953488,
        0.65116279, 0.6627907 , 0.65116279, 0.65116279, 0.6744186 ,
        0.6627907 , 0.65116279, 0.6744186 , 0.6744186 , 0.68604651,
        0.65116279, 0.6744186 , 0.6744186 , 0.68604651, 0.6744186 ,
        0.6627907 , 0.69767442, 0.68604651, 0.6744186 , 0.68604651,
        0.69767442, 0.68604651, 0.6627907 , 0.70930233, 0.69767442,
        0.68604651, 0.54651163, 0.54651163, 0.54651163, 0.54651163,
        0.54651163, 0.54651163, 0.54651163, 0.54651163, 0.60465116,
        0.60465116, 0.60465116, 0.60465116, 0.52325581, 0.52325581,
        0.52325581, 0.52325581, 0.62790698, 0.62790698, 0.62790698,
        0.63953488, 0.65116279, 0.65116279, 0.65116279, 0.6627907 ,
        0.6744186 , 0.6744186 , 0.6744186 , 0.65116279, 0.68604651,
        0.68604651, 0.68604651, 0.65116279, 0.6744186 , 0.6744186 ,
        0.6744186 , 0.65116279, 0.6744186 , 0.6744186 , 0.6744186 ,
        0.65116279, 0.6744186 , 0.6744186 , 0.6744186 , 0.65116279,
        0.6744186 , 0.6744186 , 0.6744186 , 0.65116279, 0.65116279,
        0.68604651, 0.68604651, 0.59302326, 0.65116279, 0.68604651,
        0.68604651, 0.59302326, 0.65116279, 0.68604651, 0.6744186 ,
        0.59302326, 0.65116279, 0.6744186 , 0.68604651, 0.61627907,
        0.6744186 , 0.69767442, 0.68604651, 0.6627907 , 0.6744186 ,
        0.72093023, 0.68604651, 0.68604651, 0.6744186 , 0.72093023,
        0.68604651, 0.69767442]), 'split3_test_score': array([0.54651163, 0.546
51163, 0.54651163, 0.54651163, 0.52325581,
        0.52325581, 0.52325581, 0.52325581, 0.59302326, 0.59302326,
        0.59302326, 0.59302326, 0.61627907, 0.61627907, 0.61627907,
        0.61627907, 0.61627907, 0.61627907, 0.61627907, 0.61627907,
        0.63953488, 0.63953488, 0.63953488, 0.6744186 , 0.70930233,
        0.69767442, 0.69767442, 0.68604651, 0.72093023, 0.70930233,
        0.70930233, 0.6627907 , 0.69767442, 0.68604651, 0.68604651,
        0.62790698, 0.69767442, 0.65116279, 0.65116279, 0.62790698,
        0.68604651, 0.68604651, 0.65116279, 0.62790698, 0.6627907 ,
        0.68604651, 0.65116279, 0.62790698, 0.63953488, 0.68604651,
        0.65116279, 0.61627907, 0.65116279, 0.6744186 , 0.63953488,
        0.61627907, 0.63953488, 0.6744186 , 0.63953488, 0.61627907,
        0.65116279, 0.69767442, 0.6627907 , 0.61627907, 0.65116279,
        0.70930233, 0.6744186 , 0.61627907, 0.65116279, 0.74418605,
        0.70930233, 0.61627907, 0.68604651, 0.74418605, 0.70930233,
        0.61627907, 0.54651163, 0.54651163, 0.54651163, 0.54651163,
        0.54651163, 0.54651163, 0.54651163, 0.54651163, 0.59302326,
```

```
0.59302326, 0.59302326, 0.59302326, 0.61627907, 0.61627907,
0.61627907, 0.61627907, 0.61627907, 0.61627907, 0.61627907,
0.61627907, 0.63953488, 0.63953488, 0.63953488, 0.6744186 ,
0.70930233, 0.69767442, 0.69767442, 0.68604651, 0.72093023,
0.70930233, 0.70930233, 0.6627907 , 0.69767442, 0.68604651,
0.68604651, 0.62790698, 0.69767442, 0.65116279, 0.65116279,
0.62790698, 0.68604651, 0.68604651, 0.65116279, 0.62790698,
0.68604651, 0.68604651, 0.65116279, 0.62790698, 0.68604651,
0.68604651, 0.65116279, 0.61627907, 0.6744186 , 0.6744186 ,
0.63953488, 0.61627907, 0.68604651, 0.6744186 , 0.63953488,
0.61627907, 0.68604651, 0.69767442, 0.6627907 , 0.61627907,
0.6744186 , 0.70930233, 0.6744186 , 0.61627907, 0.6744186 ,
0.74418605, 0.70930233, 0.61627907, 0.6744186 , 0.74418605,
0.70930233, 0.61627907]), 'split4_test_score': array([0.54651163, 0.546
51163, 0.54651163, 0.54651163, 0.54651163,
0.54651163, 0.54651163, 0.54651163, 0.62790698, 0.62790698,
0.62790698, 0.62790698, 0.61627907, 0.61627907, 0.61627907,
0.61627907, 0.62790698, 0.62790698, 0.62790698, 0.62790698,
0.63953488, 0.62790698, 0.62790698, 0.62790698, 0.6744186 ,
0.6627907 , 0.6627907 , 0.6744186 , 0.72093023, 0.70930233,
0.70930233, 0.6744186 , 0.72093023, 0.70930233, 0.70930233,
0.6744186 , 0.72093023, 0.70930233, 0.70930233, 0.69767442,
0.72093023, 0.70930233, 0.70930233, 0.72093023, 0.72093023,
0.73255814, 0.73255814, 0.74418605, 0.72093023, 0.69767442,
0.69767442, 0.70930233, 0.70930233, 0.72093023, 0.72093023,
0.70930233, 0.70930233, 0.74418605, 0.74418605, 0.72093023,
0.73255814, 0.74418605, 0.74418605, 0.73255814, 0.76744186,
0.75581395, 0.75581395, 0.73255814, 0.73255814, 0.76744186,
0.76744186, 0.73255814, 0.73255814, 0.76744186, 0.76744186,
0.73255814, 0.54651163, 0.54651163, 0.54651163, 0.54651163,
0.54651163, 0.54651163, 0.54651163, 0.54651163, 0.62790698,
0.62790698, 0.62790698, 0.62790698, 0.61627907, 0.61627907,
0.61627907, 0.61627907, 0.61627907, 0.61627907, 0.61627907,
0.61627907, 0.63953488, 0.62790698, 0.62790698, 0.62790698,
0.6744186 , 0.6627907 , 0.6627907 , 0.62790698, 0.6744186 ,
0.70930233, 0.70930233, 0.62790698, 0.72093023, 0.70930233,
0.70930233, 0.6744186 , 0.72093023, 0.70930233, 0.70930233,
0.69767442, 0.72093023, 0.70930233, 0.70930233, 0.72093023,
0.72093023, 0.73255814, 0.73255814, 0.74418605, 0.72093023,
0.73255814, 0.69767442, 0.70930233, 0.72093023, 0.74418605,
0.72093023, 0.70930233, 0.72093023, 0.77906977, 0.74418605,
0.72093023, 0.74418605, 0.74418605, 0.74418605, 0.73255814,
0.77906977, 0.74418605, 0.75581395, 0.73255814, 0.74418605,
0.75581395, 0.76744186, 0.73255814, 0.74418605, 0.76744186,
0.76744186, 0.73255814]), 'split5_test_score': array([0.54651163, 0.546
51163, 0.54651163, 0.54651163, 0.53488372,
0.53488372, 0.53488372, 0.53488372, 0.56976744, 0.56976744,
0.56976744, 0.56976744, 0.58139535, 0.58139535, 0.58139535,
0.58139535, 0.60465116, 0.60465116, 0.61627907, 0.61627907,
0.63953488, 0.62790698, 0.63953488, 0.61627907, 0.63953488,
0.62790698, 0.63953488, 0.6627907 , 0.68604651, 0.6744186 ,
0.68604651, 0.6627907 , 0.6744186 , 0.6744186 , 0.68604651,
0.6627907 , 0.6744186 , 0.6744186 , 0.68604651, 0.6627907 ,
0.6744186 , 0.6744186 , 0.68604651, 0.6627907 , 0.6627907 ,
0.65116279, 0.6627907 , 0.63953488, 0.6627907 , 0.6627907 ,
0.6744186 , 0.65116279, 0.6627907 , 0.6627907 , 0.6744186 ,
0.65116279, 0.6744186 , 0.65116279, 0.6627907 , 0.73255814,
0.6744186 , 0.63953488, 0.65116279, 0.74418605, 0.6627907 ,
0.70930233, 0.72093023, 0.77906977, 0.65116279, 0.72093023,
0.73255814, 0.76744186, 0.72093023, 0.72093023, 0.73255814,
0.73255814, 0.54651163, 0.54651163, 0.54651163, 0.54651163,
0.53488372, 0.53488372, 0.53488372, 0.53488372, 0.56976744,
0.56976744, 0.56976744, 0.56976744, 0.58139535, 0.58139535,
0.58139535, 0.58139535, 0.60465116, 0.60465116, 0.61627907,
0.61627907, 0.63953488, 0.62790698, 0.63953488, 0.61627907,
0.63953488, 0.62790698, 0.63953488, 0.61627907, 0.63953488,
0.6744186 , 0.68604651, 0.6627907 , 0.6744186 , 0.6744186 ,
0.68604651, 0.6627907 , 0.6744186 , 0.6744186 , 0.68604651,
0.6627907 , 0.6744186 , 0.6744186 , 0.68604651, 0.6627907 ,
```

```
        0.6627907 , 0.65116279, 0.6627907 , 0.63953488, 0.6627907 ,
        0.6627907 , 0.6744186 , 0.65116279, 0.6627907 , 0.6627907 ,
        0.6744186 , 0.65116279, 0.6744186 , 0.65116279, 0.6627907 ,
        0.73255814, 0.6744186 , 0.63953488, 0.65116279, 0.74418605,
        0.6627907 , 0.70930233, 0.72093023, 0.77906977, 0.65116279,
        0.72093023, 0.73255814, 0.76744186, 0.72093023, 0.72093023,
        0.73255814, 0.73255814]), 'split6_test_score': array([0.54651163, 0.546
51163, 0.54651163, 0.54651163, 0.51162791,
        0.51162791, 0.51162791, 0.51162791, 0.59302326, 0.59302326,
        0.59302326, 0.59302326, 0.62790698, 0.62790698, 0.62790698,
        0.62790698, 0.6744186 , 0.6744186 , 0.6744186 , 0.6744186 ,
        0.6744186 , 0.6744186 , 0.6744186 , 0.6744186 , 0.69767442,
        0.69767442, 0.69767442, 0.6627907 , 0.6627907 , 0.68604651,
        0.68604651, 0.6627907 , 0.6627907 , 0.68604651, 0.68604651,
        0.6627907 , 0.6627907 , 0.68604651, 0.68604651, 0.6627907 ,
        0.63953488, 0.68604651, 0.68604651, 0.63953488, 0.63953488,
        0.6627907 , 0.65116279, 0.62790698, 0.63953488, 0.6627907 ,
        0.65116279, 0.65116279, 0.6627907 , 0.68604651, 0.6744186 ,
        0.65116279, 0.6627907 , 0.69767442, 0.65116279, 0.65116279,
        0.63953488, 0.70930233, 0.6627907 , 0.60465116, 0.65116279,
        0.68604651, 0.6744186 , 0.60465116, 0.6627907 , 0.68604651,
        0.69767442, 0.63953488, 0.6627907 , 0.68604651, 0.69767442,
        0.6627907 , 0.54651163, 0.54651163, 0.54651163, 0.54651163,
        0.51162791, 0.51162791, 0.51162791, 0.51162791, 0.59302326,
        0.59302326, 0.59302326, 0.59302326, 0.62790698, 0.62790698,
        0.62790698, 0.62790698, 0.65116279, 0.65116279, 0.65116279,
        0.6744186 , 0.69767442, 0.69767442, 0.69767442, 0.6744186 ,
        0.69767442, 0.69767442, 0.69767442, 0.6627907 , 0.68604651,
        0.68604651, 0.68604651, 0.6627907 , 0.6627907 , 0.68604651,
        0.68604651, 0.6627907 , 0.6627907 , 0.68604651, 0.68604651,
        0.6627907 , 0.6627907 , 0.68604651, 0.68604651, 0.63953488,
        0.6627907 , 0.6627907 , 0.65116279, 0.62790698, 0.63953488,
        0.6627907 , 0.65116279, 0.65116279, 0.63953488, 0.68604651,
        0.6744186 , 0.65116279, 0.6627907 , 0.69767442, 0.65116279,
        0.65116279, 0.6627907 , 0.70930233, 0.6627907 , 0.60465116,
        0.63953488, 0.68604651, 0.6744186 , 0.60465116, 0.65116279,
        0.68604651, 0.69767442, 0.63953488, 0.6627907 , 0.68604651,
        0.69767442, 0.6627907 ]), 'split7_test_score': array([0.54651163, 0.546
51163, 0.54651163, 0.54651163, 0.54651163,
        0.54651163, 0.54651163, 0.54651163, 0.60465116, 0.60465116,
        0.60465116, 0.60465116, 0.60465116, 0.60465116, 0.60465116,
        0.63953488, 0.63953488, 0.63953488, 0.63953488, 0.68604651,
        0.68604651, 0.68604651, 0.68604651, 0.68604651, 0.6627907 ,
        0.6627907 , 0.65116279, 0.6627907 , 0.6627907 , 0.6627907 ,
        0.65116279, 0.6627907 , 0.6627907 , 0.6627907 , 0.65116279,
        0.6627907 , 0.6627907 , 0.6627907 , 0.65116279, 0.6627907 ,
        0.6627907 , 0.6627907 , 0.65116279, 0.6627907 , 0.6627907 ,
        0.6627907 , 0.65116279, 0.59302326, 0.6627907 , 0.65116279,
        0.63953488, 0.56976744, 0.6627907 , 0.65116279, 0.63953488,
        0.56976744, 0.6627907 , 0.65116279, 0.63953488, 0.61627907,
        0.6627907 , 0.59302326, 0.59302326, 0.65116279, 0.6744186 ,
        0.60465116, 0.63953488, 0.65116279, 0.6744186 , 0.60465116,
        0.6744186 , 0.65116279, 0.6744186 , 0.63953488, 0.6744186 ,
        0.65116279, 0.54651163, 0.54651163, 0.54651163, 0.54651163,
        0.54651163, 0.54651163, 0.54651163, 0.54651163, 0.60465116,
        0.60465116, 0.60465116, 0.60465116, 0.60465116, 0.60465116,
        0.60465116, 0.63953488, 0.63953488, 0.63953488, 0.63953488,
        0.68604651, 0.68604651, 0.68604651, 0.68604651, 0.68604651,
        0.68604651, 0.68604651, 0.6744186 , 0.6627907 , 0.6627907 ,
        0.6627907 , 0.65116279, 0.6627907 , 0.6627907 , 0.6627907 ,
        0.65116279, 0.6627907 , 0.6627907 , 0.6627907 , 0.65116279,
        0.6627907 , 0.6627907 , 0.6627907 , 0.65116279, 0.6627907 ,
        0.6627907 , 0.6627907 , 0.65116279, 0.59302326, 0.6627907 ,
        0.65116279, 0.63953488, 0.56976744, 0.6627907 , 0.65116279,
        0.63953488, 0.56976744, 0.6627907 , 0.65116279, 0.63953488,
        0.61627907, 0.6627907 , 0.59302326, 0.59302326, 0.65116279,
        0.6744186 , 0.60465116, 0.63953488, 0.65116279, 0.6744186 ,
        0.60465116, 0.6744186 , 0.65116279, 0.6744186 , 0.63953488,
        0.6744186 , 0.65116279]), 'split8_test_score': array([0.54651163, 0.546
```

```
51163, 0.54651163, 0.54651163, 0.51162791,
       0.51162791, 0.51162791, 0.51162791, 0.54651163, 0.54651163,
       0.54651163, 0.54651163, 0.54651163, 0.54651163, 0.54651163,
       0.54651163, 0.56976744, 0.56976744, 0.56976744, 0.56976744,
       0.55813953, 0.55813953, 0.56976744, 0.56976744, 0.60465116,
       0.59302326, 0.60465116, 0.61627907, 0.65116279, 0.63953488,
       0.65116279, 0.61627907, 0.65116279, 0.63953488, 0.65116279,
       0.61627907, 0.65116279, 0.63953488, 0.65116279, 0.61627907,
       0.65116279, 0.63953488, 0.65116279, 0.65116279, 0.6627907 ,
       0.65116279, 0.65116279, 0.6627907 , 0.6627907 , 0.65116279,
       0.6744186 , 0.6627907 , 0.6627907 , 0.62790698, 0.68604651,
       0.6627907 , 0.6627907 , 0.61627907, 0.65116279, 0.6627907 ,
       0.6627907 , 0.62790698, 0.63953488, 0.6627907 , 0.6627907 ,
       0.62790698, 0.65116279, 0.6627907 , 0.63953488, 0.62790698,
       0.65116279, 0.65116279, 0.63953488, 0.62790698, 0.65116279,
       0.6627907 , 0.54651163, 0.54651163, 0.54651163, 0.54651163,
       0.5       , 0.5       , 0.5       , 0.5       , 0.53488372,
       0.53488372, 0.53488372, 0.53488372, 0.54651163, 0.54651163,
       0.54651163, 0.54651163, 0.56976744, 0.56976744, 0.56976744,
       0.56976744, 0.55813953, 0.55813953, 0.56976744, 0.56976744,
       0.60465116, 0.59302326, 0.60465116, 0.61627907, 0.65116279,
       0.63953488, 0.65116279, 0.61627907, 0.65116279, 0.63953488,
       0.65116279, 0.61627907, 0.65116279, 0.63953488, 0.65116279,
       0.61627907, 0.65116279, 0.63953488, 0.65116279, 0.65116279,
       0.65116279, 0.63953488, 0.65116279, 0.6627907 , 0.65116279,
       0.63953488, 0.6744186 , 0.6627907 , 0.6627907 , 0.65116279,
       0.68604651, 0.6627907 , 0.6627907 , 0.6627907 , 0.65116279,
       0.6627907 , 0.6627907 , 0.6627907 , 0.63953488, 0.6627907 ,
       0.6744186 , 0.6627907 , 0.65116279, 0.6627907 , 0.6744186 ,
       0.6627907 , 0.65116279, 0.65116279, 0.6744186 , 0.6627907 ,
       0.65116279, 0.6627907 ]), 'split9_test_score': array([0.54651163, 0.546
51163, 0.54651163, 0.54651163, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.56976744, 0.56976744,
       0.56976744, 0.56976744, 0.60465116, 0.60465116, 0.60465116,
       0.60465116, 0.60465116, 0.60465116, 0.60465116, 0.60465116,
       0.60465116, 0.60465116, 0.60465116, 0.65116279, 0.6744186 ,
       0.65116279, 0.65116279, 0.65116279, 0.6744186 , 0.65116279,
       0.65116279, 0.65116279, 0.6744186 , 0.65116279, 0.65116279,
       0.55813953, 0.6744186 , 0.65116279, 0.65116279, 0.55813953,
       0.68604651, 0.55813953, 0.72093023, 0.55813953, 0.68604651,
       0.6627907 , 0.72093023, 0.55813953, 0.74418605, 0.55813953,
       0.55813953, 0.54651163, 0.74418605, 0.73255814, 0.55813953,
       0.55813953, 0.58139535, 0.73255814, 0.55813953, 0.59302326,
       0.58139535, 0.73255814, 0.56976744, 0.59302326, 0.58139535,
       0.55813953, 0.60465116, 0.59302326, 0.58139535, 0.59302326,
       0.60465116, 0.59302326, 0.59302326, 0.61627907, 0.60465116,
       0.59302326, 0.54651163, 0.54651163, 0.54651163, 0.54651163,
       0.5       , 0.5       , 0.5       , 0.5       , 0.56976744,
       0.56976744, 0.56976744, 0.56976744, 0.60465116, 0.60465116,
       0.60465116, 0.60465116, 0.60465116, 0.60465116, 0.60465116,
       0.60465116, 0.60465116, 0.60465116, 0.60465116, 0.65116279,
       0.65116279, 0.65116279, 0.65116279, 0.65116279, 0.65116279,
       0.65116279, 0.65116279, 0.65116279, 0.65116279, 0.65116279,
       0.65116279, 0.55813953, 0.65116279, 0.65116279, 0.65116279,
       0.55813953, 0.6627907 , 0.55813953, 0.72093023, 0.55813953,
       0.6627907 , 0.6627907 , 0.72093023, 0.55813953, 0.72093023,
       0.55813953, 0.55813953, 0.54651163, 0.72093023, 0.73255814,
       0.55813953, 0.55813953, 0.55813953, 0.73255814, 0.55813953,
       0.59302326, 0.55813953, 0.73255814, 0.56976744, 0.59302326,
       0.55813953, 0.55813953, 0.60465116, 0.59302326, 0.55813953,
       0.59302326, 0.60465116, 0.59302326, 0.58139535, 0.61627907,
       0.60465116, 0.59302326]), 'mean_test_score': array([0.54587689, 0.54587
689, 0.54587689, 0.54587689, 0.52497096,
       0.52497096, 0.52497096, 0.52497096, 0.58072009, 0.58072009,
       0.58072009, 0.58072009, 0.59233449, 0.59233449, 0.59233449,
       0.59581882, 0.6097561 , 0.6097561 , 0.61091754, 0.61904762,
       0.63182346, 0.62950058, 0.63182346, 0.63182346, 0.6562137 ,
       0.64924506, 0.64924506, 0.63646922, 0.66085947, 0.6562137 ,
       0.6562137 , 0.63646922, 0.65969803, 0.6562137 , 0.6562137 ,
```

```
        0.62369338, 0.65969803, 0.65272938, 0.65272938, 0.62601626,
        0.65969803, 0.64692218, 0.65969803, 0.6329849 , 0.65505226,
        0.6562137 , 0.6562137 , 0.62833914, 0.65969803, 0.64227642,
        0.63995354, 0.62485482, 0.66434379, 0.66434379, 0.64808362,
        0.62833914, 0.64576074, 0.66782811, 0.64227642, 0.64924506,
        0.64808362, 0.66318235, 0.63995354, 0.6504065 , 0.65272938,
        0.65272938, 0.66202091, 0.65389082, 0.64924506, 0.66550523,
        0.67944251, 0.65505226, 0.65853659, 0.67363531, 0.67944251,
        0.65505226, 0.54587689, 0.54587689, 0.54587689, 0.54587689,
        0.5261324 , 0.5261324 , 0.5261324 , 0.5261324 , 0.58304297,
        0.58304297, 0.58304297, 0.58304297, 0.58768873, 0.58768873,
        0.58768873, 0.59117305, 0.61091754, 0.61091754, 0.61207898,
        0.62369338, 0.63530778, 0.6329849 , 0.63530778, 0.63763066,
        0.65969803, 0.65389082, 0.65389082, 0.6329849 , 0.65737515,
        0.66085947, 0.66085947, 0.63763066, 0.66202091, 0.65969803,
        0.65969803, 0.62950058, 0.66318235, 0.6562137 , 0.6562137 ,
        0.63182346, 0.66318235, 0.6504065 , 0.66318235, 0.63530778,
        0.66202091, 0.65853659, 0.65969803, 0.62833914, 0.66202091,
        0.64924506, 0.64343786, 0.61788618, 0.66318235, 0.67247387,
        0.64924506, 0.62020906, 0.6504065 , 0.67711963, 0.64227642,
        0.63995354, 0.65272938, 0.66666667, 0.64111498, 0.64227642,
        0.6562137 , 0.65853659, 0.66085947, 0.65156794, 0.65272938,
        0.67131243, 0.67828107, 0.65505226, 0.66434379, 0.67828107,
        0.67828107, 0.6562137 ]), 'std_test_score': array([0.00189325, 0.001893
25, 0.00189325, 0.00189325, 0.02010758,
        0.02010758, 0.02010758, 0.02010758, 0.02900645, 0.02900645,
        0.02900645, 0.02900645, 0.02737107, 0.02737107, 0.02737107,
        0.03073127, 0.03044113, 0.03044113, 0.03044602, 0.03431659,
        0.0335314 , 0.03331882, 0.03102085, 0.03727377, 0.0320497 ,
        0.03154358, 0.02978181, 0.04199434, 0.04301171, 0.04026672,
        0.04317987, 0.03497599, 0.03477689, 0.03288167, 0.03564093,
        0.03986848, 0.03477689, 0.03134829, 0.03423138, 0.04328715,
        0.03307946, 0.04419345, 0.03984439, 0.04559244, 0.03324294,
        0.03525997, 0.04158665, 0.05105332, 0.04215029, 0.03993699,
        0.04069797, 0.0485427 , 0.03775614, 0.03833754, 0.04522939,
        0.04467917, 0.03652576, 0.04388624, 0.04787595, 0.04747713,
        0.04008758, 0.05361763, 0.04834748, 0.05260488, 0.05017147,
        0.05870729, 0.05020751, 0.05941544, 0.03986505, 0.06037925,
        0.04857727, 0.05498018, 0.04518098, 0.05606932, 0.04857727,
        0.04845166, 0.00189325, 0.00189325, 0.00189325, 0.00189325,
        0.02221525, 0.02221525, 0.02221525, 0.02221525, 0.0317369 ,
        0.0317369 , 0.0317369 , 0.0317369 , 0.03444446, 0.03444446,
        0.03444446, 0.03760361, 0.0249914 , 0.0249914 , 0.02494334,
        0.03272649, 0.03793625, 0.03796229, 0.0357365 , 0.03593522,
        0.03210063, 0.03396018, 0.03148385, 0.03724399, 0.03833876,
        0.04075458, 0.04363517, 0.02976889, 0.03306725, 0.03277775,
        0.03554508, 0.038793  , 0.03218907, 0.03162552, 0.03448543,
        0.04197856, 0.03090478, 0.04484403, 0.03945214, 0.04509306,
        0.03067835, 0.03561495, 0.04150454, 0.04955201, 0.03729015,
        0.04730188, 0.04294   , 0.0475951 , 0.0357663 , 0.04043529,
        0.04603289, 0.04495188, 0.04404595, 0.04753161, 0.04787595,
        0.04845813, 0.04812803, 0.05233005, 0.04929192, 0.05265941,
        0.05491394, 0.05768103, 0.0494988 , 0.05856135, 0.04745731,
        0.05926953, 0.04826548, 0.05498018, 0.04631439, 0.05496747,
        0.04826548, 0.04931243]), 'rank_test_score': array([137, 137, 137, 137,
149, 149, 149, 149, 133, 133, 133, 133, 122,
        122, 122, 121, 119, 119, 116, 113,  99, 103,  99,  99,  42,  70,
         70,  91,  25,  42,  42,  91,  29,  42,  42, 110,  29,  60,  60,
        108,  29,  78,  29,  96,  53,  42,  42, 105,  29,  81,  86, 109,
         13,  13,  76, 105,  79,  10,  81,  70,  76,  16,  86,  67,  60,
         60,  21,  57,  70,  12,   1,  53,  38,   7,   1,  53, 137, 137,
        137, 137, 145, 145, 145, 145, 129, 129, 129, 129, 126, 126, 126,
        125, 116, 116, 115, 110,  93,  96,  93,  89,  29,  57,  57,  96,
         41,  25,  25,  89,  21,  29,  29, 103,  16,  42,  42,  99,  16,
         67,  16,  93,  21,  38,  29, 105,  21,  70,  80, 114,  16,   8,
         70, 112,  67,   6,  81,  86,  60,  11,  85,  81,  42,  38,  25,
         66,  60,   9,   3,  53,  13,   3,   3,  42], dtype=int32), 'split0_tra
in_score': array([0.54651163, 0.54651163, 0.54651163, 0.54651163, 0.55297158,
        0.55297158, 0.55297158, 0.55297158, 0.58397933, 0.58397933,
```

```
       0.58397933, 0.58397933, 0.62144703, 0.62144703, 0.62144703,
       0.62144703, 0.64599483, 0.64599483, 0.64599483, 0.64599483,
       0.66795866, 0.66408269, 0.66408269, 0.64599483, 0.66795866,
       0.66408269, 0.66408269, 0.66795866, 0.68992248, 0.68604651,
       0.68604651, 0.66795866, 0.68992248, 0.68604651, 0.68604651,
       0.66795866, 0.68992248, 0.68604651, 0.68604651, 0.67958656,
       0.68992248, 0.68604651, 0.68604651, 0.68992248, 0.70155039,
       0.69767442, 0.69767442, 0.69767442, 0.70155039, 0.70671835,
       0.70671835, 0.70284238, 0.70930233, 0.71317829, 0.71447028,
       0.70542636, 0.71963824, 0.71963824, 0.72093023, 0.70801034,
       0.72739018, 0.7248062 , 0.72222222, 0.70930233, 0.73385013,
       0.7248062 , 0.72222222, 0.70930233, 0.7377261 , 0.7248062 ,
       0.7248062 , 0.70930233, 0.73901809, 0.7248062 , 0.7248062 ,
       0.70930233, 0.54651163, 0.54651163, 0.54651163, 0.54651163,
       0.54651163, 0.54651163, 0.54651163, 0.54651163, 0.58010336,
       0.58010336, 0.58010336, 0.58010336, 0.60981912, 0.60981912,
       0.60981912, 0.60981912, 0.64341085, 0.64341085, 0.64341085,
       0.64341085, 0.64341085, 0.64341085, 0.64341085, 0.64341085,
       0.66795866, 0.66408269, 0.66408269, 0.66537468, 0.69121447,
       0.68604651, 0.68604651, 0.66537468, 0.69121447, 0.68604651,
       0.68604651, 0.66537468, 0.69121447, 0.68604651, 0.68604651,
       0.67700258, 0.69121447, 0.68604651, 0.68604651, 0.68604651,
       0.69121447, 0.69767442, 0.69767442, 0.69379845, 0.69896641,
       0.70671835, 0.70671835, 0.7002584 , 0.71059432, 0.71317829,
       0.71447028, 0.70155039, 0.72093023, 0.71963824, 0.72093023,
       0.70284238, 0.72868217, 0.7248062 , 0.72222222, 0.70284238,
       0.73514212, 0.7248062 , 0.72222222, 0.70413437, 0.73901809,
       0.7248062 , 0.7248062 , 0.70413437, 0.74031008, 0.7248062 ,
       0.7248062 , 0.70413437]), 'split1_train_score': array([0.54580645, 0.54
580645, 0.54580645, 0.54580645, 0.55870968,
       0.55870968, 0.55870968, 0.55870968, 0.58967742, 0.58967742,
       0.58967742, 0.58967742, 0.61677419, 0.61677419, 0.61677419,
       0.61677419, 0.63612903, 0.63612903, 0.63612903, 0.64387097,
       0.66322581, 0.66322581, 0.66322581, 0.64387097, 0.66580645,
       0.66580645, 0.66451613, 0.66193548, 0.68903226, 0.68387097,
       0.68258065, 0.68129032, 0.7083871 , 0.70322581, 0.70193548,
       0.68129032, 0.7083871 , 0.70322581, 0.70193548, 0.68129032,
       0.7083871 , 0.70322581, 0.70193548, 0.68129032, 0.7083871 ,
       0.70322581, 0.70193548, 0.68903226, 0.71483871, 0.70322581,
       0.70193548, 0.69806452, 0.71483871, 0.70967742, 0.7083871 ,
       0.69806452, 0.72      , 0.71741935, 0.71612903, 0.69806452,
       0.72      , 0.71741935, 0.72      , 0.70193548, 0.73032258,
       0.72129032, 0.72129032, 0.70193548, 0.73548387, 0.72129032,
       0.72129032, 0.70193548, 0.73806452, 0.72258065, 0.72129032,
       0.70193548, 0.54580645, 0.54580645, 0.54580645, 0.54580645,
       0.55225806, 0.55225806, 0.55225806, 0.55225806, 0.58967742,
       0.58967742, 0.58967742, 0.58967742, 0.61677419, 0.61677419,
       0.61677419, 0.61677419, 0.63612903, 0.63612903, 0.63612903,
       0.64387097, 0.66322581, 0.66322581, 0.66322581, 0.64387097,
       0.66580645, 0.66580645, 0.66451613, 0.66193548, 0.68903226,
       0.68387097, 0.68258065, 0.68129032, 0.70967742, 0.70322581,
       0.70193548, 0.68129032, 0.70967742, 0.70322581, 0.70193548,
       0.68129032, 0.70967742, 0.70322581, 0.70193548, 0.68129032,
       0.70967742, 0.70322581, 0.70193548, 0.68903226, 0.71612903,
       0.70322581, 0.70193548, 0.69806452, 0.71612903, 0.70967742,
       0.7083871 , 0.69806452, 0.72129032, 0.71741935, 0.71612903,
       0.69806452, 0.72129032, 0.71741935, 0.72      , 0.70193548,
       0.7316129 , 0.72129032, 0.72129032, 0.70193548, 0.73677419,
       0.72129032, 0.72129032, 0.70193548, 0.73935484, 0.72258065,
       0.72129032, 0.70193548]), 'split2_train_score': array([0.54580645, 0.54
580645, 0.54580645, 0.54580645, 0.54580645,
       0.54580645, 0.54580645, 0.54580645, 0.58064516, 0.58064516,
       0.58064516, 0.58064516, 0.61548387, 0.61548387, 0.61548387,
       0.61548387, 0.61935484, 0.61548387, 0.61548387, 0.61548387,
       0.64516129, 0.64129032, 0.64129032, 0.64129032, 0.66451613,
       0.66064516, 0.66064516, 0.66451613, 0.68774194, 0.68387097,
       0.68387097, 0.66451613, 0.68774194, 0.68387097, 0.68387097,
       0.66451613, 0.68774194, 0.68387097, 0.68387097, 0.67612903,
       0.68774194, 0.68387097, 0.68387097, 0.68387097, 0.68774194,
```

```
       0.69548387, 0.69548387, 0.69290323, 0.69419355, 0.69548387,
       0.70193548, 0.69677419, 0.70580645, 0.70322581, 0.71096774,
       0.70064516, 0.70580645, 0.71354839, 0.71483871, 0.70064516,
       0.71354839, 0.71741935, 0.71870968, 0.70064516, 0.72387097,
       0.72129032, 0.71870968, 0.70064516, 0.72774194, 0.72258065,
       0.71870968, 0.70064516, 0.7316129 , 0.72258065, 0.71870968,
       0.70064516, 0.54580645, 0.54580645, 0.54580645, 0.54580645,
       0.54580645, 0.54580645, 0.54580645, 0.54580645, 0.58064516,
       0.58064516, 0.58064516, 0.58064516, 0.58193548, 0.58193548,
       0.58193548, 0.58193548, 0.60258065, 0.60129032, 0.60129032,
       0.61032258, 0.62967742, 0.6283871 , 0.6283871 , 0.63741935,
       0.64774194, 0.64645161, 0.64645161, 0.65935484, 0.66580645,
       0.66451613, 0.66451613, 0.65935484, 0.69032258, 0.68774194,
       0.68774194, 0.65935484, 0.69032258, 0.68774194, 0.68774194,
       0.67354839, 0.69032258, 0.68774194, 0.68774194, 0.67354839,
       0.69032258, 0.69935484, 0.69935484, 0.67354839, 0.69677419,
       0.70967742, 0.70967742, 0.67354839, 0.69677419, 0.70967742,
       0.71612903, 0.68387097, 0.70322581, 0.71612903, 0.72      ,
       0.68387097, 0.71483871, 0.72      , 0.72      , 0.69032258,
       0.72516129, 0.72      , 0.72      , 0.69806452, 0.72516129,
       0.72      , 0.72      , 0.70451613, 0.72774194, 0.72      ,
       0.72      , 0.70967742]), 'split3_train_score': array([0.54580645, 0.54
580645, 0.54580645, 0.54580645, 0.54967742,
       0.54967742, 0.54967742, 0.54967742, 0.58193548, 0.58193548,
       0.58193548, 0.58193548, 0.61419355, 0.61419355, 0.61419355,
       0.61419355, 0.61419355, 0.61419355, 0.61419355, 0.61419355,
       0.63870968, 0.63354839, 0.63354839, 0.63612903, 0.66064516,
       0.65548387, 0.65548387, 0.65677419, 0.68129032, 0.67612903,
       0.67612903, 0.67096774, 0.69548387, 0.69032258, 0.69032258,
       0.68258065, 0.70193548, 0.70064516, 0.70064516, 0.69419355,
       0.71096774, 0.71741935, 0.71225806, 0.69419355, 0.71741935,
       0.71741935, 0.71225806, 0.69419355, 0.72129032, 0.71741935,
       0.71225806, 0.69806452, 0.72516129, 0.72129032, 0.71612903,
       0.69806452, 0.72516129, 0.72129032, 0.71612903, 0.69806452,
       0.72516129, 0.72258065, 0.71741935, 0.69806452, 0.72516129,
       0.72258065, 0.71741935, 0.69806452, 0.72516129, 0.72258065,
       0.71741935, 0.69806452, 0.72645161, 0.72258065, 0.71741935,
       0.69806452, 0.54580645, 0.54580645, 0.54580645, 0.54580645,
       0.54580645, 0.54580645, 0.54580645, 0.54580645, 0.58193548,
       0.58193548, 0.58193548, 0.58193548, 0.61419355, 0.61419355,
       0.61419355, 0.61419355, 0.61419355, 0.61419355, 0.61419355,
       0.61419355, 0.63870968, 0.63354839, 0.63354839, 0.63612903,
       0.66193548, 0.65548387, 0.65548387, 0.65677419, 0.68258065,
       0.67612903, 0.67612903, 0.67096774, 0.69677419, 0.69032258,
       0.69032258, 0.68258065, 0.70322581, 0.70064516, 0.70064516,
       0.68258065, 0.70322581, 0.70580645, 0.70064516, 0.69419355,
       0.70967742, 0.71741935, 0.71225806, 0.69419355, 0.70967742,
       0.71741935, 0.71225806, 0.69806452, 0.71870968, 0.72129032,
       0.71612903, 0.69806452, 0.7316129 , 0.72129032, 0.71612903,
       0.69806452, 0.7316129 , 0.72258065, 0.71741935, 0.69806452,
       0.7316129 , 0.72258065, 0.71741935, 0.69806452, 0.73548387,
       0.72258065, 0.71741935, 0.69806452, 0.73548387, 0.72258065,
       0.71741935, 0.69806452]), 'split4_train_score': array([0.54580645, 0.54
580645, 0.54580645, 0.54580645, 0.54580645,
       0.54580645, 0.54580645, 0.54580645, 0.57806452, 0.57806452,
       0.57806452, 0.57806452, 0.61419355, 0.61419355, 0.61419355,
       0.61419355, 0.64258065, 0.64258065, 0.64129032, 0.64129032,
       0.64645161, 0.64258065, 0.64129032, 0.64129032, 0.66451613,
       0.66064516, 0.65935484, 0.65806452, 0.68129032, 0.67741935,
       0.67612903, 0.65806452, 0.68774194, 0.67741935, 0.67612903,
       0.65806452, 0.68774194, 0.67741935, 0.67612903, 0.66709677,
       0.68774194, 0.67741935, 0.67612903, 0.67483871, 0.68774194,
       0.68645161, 0.68516129, 0.68387097, 0.68774194, 0.69290323,
       0.6916129 , 0.6916129 , 0.69290323, 0.69935484, 0.69806452,
       0.69548387, 0.69290323, 0.7083871 , 0.70709677, 0.69677419,
       0.70193548, 0.71225806, 0.71096774, 0.69677419, 0.71096774,
       0.71354839, 0.71225806, 0.69677419, 0.71870968, 0.71354839,
       0.71225806, 0.69677419, 0.72258065, 0.71354839, 0.71225806,
       0.69677419, 0.54580645, 0.54580645, 0.54580645, 0.54580645,
```

```
        0.54580645, 0.54580645, 0.54580645, 0.54580645, 0.57806452,
        0.57806452, 0.57806452, 0.57806452, 0.61419355, 0.61419355,
        0.61419355, 0.61419355, 0.61548387, 0.61548387, 0.61419355,
        0.61419355, 0.64645161, 0.64258065, 0.64129032, 0.64129032,
        0.66451613, 0.66064516, 0.65935484, 0.64129032, 0.67225806,
        0.67741935, 0.67612903, 0.64129032, 0.68774194, 0.67741935,
        0.67612903, 0.65806452, 0.68774194, 0.67741935, 0.67612903,
        0.66709677, 0.68774194, 0.67741935, 0.67612903, 0.67483871,
        0.68774194, 0.68645161, 0.68516129, 0.68387097, 0.69419355,
        0.68645161, 0.6916129 , 0.6916129 , 0.69419355, 0.69419355,
        0.69806452, 0.69548387, 0.69419355, 0.70322581, 0.70709677,
        0.69677419, 0.70322581, 0.70967742, 0.71096774, 0.69677419,
        0.71225806, 0.71354839, 0.71225806, 0.69677419, 0.71870968,
        0.71612903, 0.71225806, 0.69677419, 0.72258065, 0.71741935,
        0.71225806, 0.69677419]), 'split5_train_score': array([0.54580645, 0.54
580645, 0.54580645, 0.54580645, 0.54709677,
        0.54709677, 0.54709677, 0.54709677, 0.58451613, 0.58451613,
        0.58451613, 0.58451613, 0.61806452, 0.61806452, 0.61806452,
        0.61806452, 0.64516129, 0.64516129, 0.64258065, 0.64258065,
        0.6683871 , 0.66451613, 0.66193548, 0.64258065, 0.6683871 ,
        0.66451613, 0.66193548, 0.65935484, 0.68516129, 0.68129032,
        0.67870968, 0.65935484, 0.6916129 , 0.68129032, 0.67870968,
        0.65935484, 0.6916129 , 0.68129032, 0.67870968, 0.65935484,
        0.6916129 , 0.68129032, 0.67870968, 0.65935484, 0.69806452,
        0.68903226, 0.68645161, 0.66709677, 0.69806452, 0.70064516,
        0.69806452, 0.67870968, 0.69806452, 0.70064516, 0.69806452,
        0.67870968, 0.7083871 , 0.70580645, 0.70322581, 0.67870968,
        0.71225806, 0.71096774, 0.7083871 , 0.68774194, 0.71741935,
        0.71483871, 0.71225806, 0.69677419, 0.72258065, 0.71741935,
        0.71483871, 0.70451613, 0.72645161, 0.71741935, 0.71483871,
        0.70580645, 0.54580645, 0.54580645, 0.54580645, 0.54580645,
        0.5483871 , 0.5483871 , 0.5483871 , 0.5483871 , 0.58451613,
        0.58451613, 0.58451613, 0.58451613, 0.61806452, 0.61806452,
        0.61806452, 0.61806452, 0.64516129, 0.64516129, 0.64258065,
        0.64258065, 0.6683871 , 0.66451613, 0.66193548, 0.64258065,
        0.6683871 , 0.66451613, 0.66193548, 0.64258065, 0.67612903,
        0.68129032, 0.67870968, 0.65935484, 0.6916129 , 0.68129032,
        0.67870968, 0.65935484, 0.6916129 , 0.68129032, 0.67870968,
        0.65935484, 0.6916129 , 0.68129032, 0.67870968, 0.65935484,
        0.69806452, 0.68903226, 0.68645161, 0.66709677, 0.69806452,
        0.70064516, 0.69806452, 0.67870968, 0.69806452, 0.70064516,
        0.69806452, 0.67870968, 0.7083871 , 0.70580645, 0.70322581,
        0.67870968, 0.71225806, 0.71096774, 0.7083871 , 0.68774194,
        0.71741935, 0.71483871, 0.71225806, 0.69677419, 0.72258065,
        0.71741935, 0.71483871, 0.70451613, 0.72645161, 0.71741935,
        0.71483871, 0.70580645]), 'split6_train_score': array([0.54580645, 0.54
580645, 0.54580645, 0.54580645, 0.54967742,
        0.54967742, 0.54967742, 0.54967742, 0.58193548, 0.58193548,
        0.58193548, 0.58193548, 0.60774194, 0.60774194, 0.60774194,
        0.60774194, 0.63612903, 0.63612903, 0.63612903, 0.63612903,
        0.63870968, 0.63870968, 0.63612903, 0.63612903, 0.66322581,
        0.65806452, 0.65548387, 0.65935484, 0.68774194, 0.68129032,
        0.67870968, 0.65935484, 0.68774194, 0.68129032, 0.67870968,
        0.65935484, 0.68774194, 0.68129032, 0.67870968, 0.65935484,
        0.69548387, 0.68129032, 0.67870968, 0.67225806, 0.69548387,
        0.68903226, 0.68774194, 0.68129032, 0.7083871 , 0.70193548,
        0.70064516, 0.69032258, 0.71741935, 0.71096774, 0.70967742,
        0.69032258, 0.72258065, 0.71483871, 0.71354839, 0.69032258,
        0.72645161, 0.71741935, 0.71612903, 0.69032258, 0.72903226,
        0.72      , 0.71870968, 0.70064516, 0.7316129 , 0.72      ,
        0.72      , 0.71225806, 0.7316129 , 0.72      , 0.72      ,
        0.71612903, 0.54580645, 0.54580645, 0.54580645, 0.54580645,
        0.54967742, 0.54967742, 0.54967742, 0.54967742, 0.58193548,
        0.58193548, 0.58193548, 0.58193548, 0.60774194, 0.60774194,
        0.60774194, 0.60774194, 0.62709677, 0.62709677, 0.62709677,
        0.63612903, 0.65548387, 0.65548387, 0.65548387, 0.63612903,
        0.65806452, 0.65806452, 0.65548387, 0.65935484, 0.68645161,
        0.68129032, 0.67870968, 0.65935484, 0.69419355, 0.68129032,
        0.67870968, 0.65935484, 0.69419355, 0.68129032, 0.67870968,
```

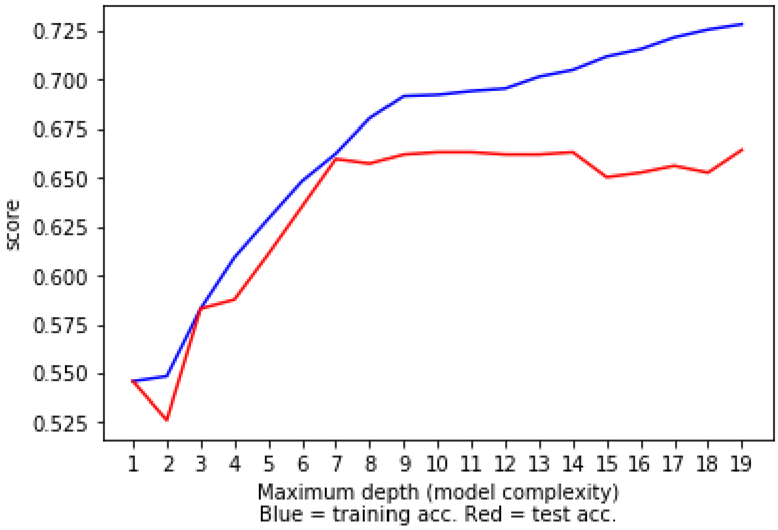
```
       0.65935484, 0.69419355, 0.68129032, 0.67870968, 0.67225806,
       0.69419355, 0.68903226, 0.68774194, 0.68129032, 0.70709677,
       0.70193548, 0.70064516, 0.69032258, 0.71225806, 0.71096774,
       0.70967742, 0.69032258, 0.72129032, 0.71483871, 0.71354839,
       0.69032258, 0.72129032, 0.71741935, 0.71612903, 0.69032258,
       0.72645161, 0.72      , 0.71870968, 0.70064516, 0.73032258,
       0.72      , 0.72      , 0.71225806, 0.73290323, 0.72      ,
       0.72      , 0.71612903]), 'split7_train_score': array([0.54580645, 0.54
580645, 0.54580645, 0.54580645, 0.54580645,
       0.54580645, 0.54580645, 0.54580645, 0.58064516, 0.58064516,
       0.58064516, 0.58064516, 0.60258065, 0.60258065, 0.60258065,
       0.60645161, 0.6283871 , 0.6283871 , 0.6283871 , 0.63483871,
       0.65677419, 0.65677419, 0.65677419, 0.63483871, 0.68129032,
       0.68129032, 0.68      , 0.65935484, 0.68645161, 0.68129032,
       0.68      , 0.65935484, 0.69290323, 0.68129032, 0.68      ,
       0.65935484, 0.69290323, 0.68129032, 0.68      , 0.65935484,
       0.69290323, 0.68129032, 0.68      , 0.65935484, 0.69290323,
       0.68129032, 0.68      , 0.66580645, 0.69290323, 0.6916129 ,
       0.68903226, 0.67870968, 0.69290323, 0.6916129 , 0.68903226,
       0.68516129, 0.69935484, 0.6916129 , 0.68903226, 0.6916129 ,
       0.69935484, 0.6916129 , 0.69548387, 0.69677419, 0.70967742,
       0.70451613, 0.70451613, 0.69677419, 0.70967742, 0.70451613,
       0.71096774, 0.69677419, 0.70967742, 0.71096774, 0.71741935,
       0.69677419, 0.54580645, 0.54580645, 0.54580645, 0.54580645,
       0.54580645, 0.54580645, 0.54580645, 0.54580645, 0.58064516,
       0.58064516, 0.58064516, 0.58064516, 0.60258065, 0.60258065,
       0.60258065, 0.60645161, 0.6283871 , 0.6283871 , 0.6283871 ,
       0.63483871, 0.65677419, 0.65677419, 0.65677419, 0.63483871,
       0.65935484, 0.65935484, 0.65806452, 0.65935484, 0.68903226,
       0.68387097, 0.68258065, 0.65935484, 0.69548387, 0.68387097,
       0.68258065, 0.65935484, 0.69548387, 0.68387097, 0.68258065,
       0.65935484, 0.69548387, 0.68387097, 0.68258065, 0.65935484,
       0.69548387, 0.68387097, 0.68258065, 0.66580645, 0.69548387,
       0.69419355, 0.6916129 , 0.67870968, 0.69548387, 0.69419355,
       0.6916129 , 0.68516129, 0.70193548, 0.69419355, 0.6916129 ,
       0.6916129 , 0.70193548, 0.69419355, 0.69806452, 0.69677419,
       0.71225806, 0.70709677, 0.70709677, 0.69677419, 0.71225806,
       0.70709677, 0.71354839, 0.69677419, 0.71225806, 0.71354839,
       0.72      , 0.69677419]), 'split8_train_score': array([0.54580645, 0.54
580645, 0.54580645, 0.54580645, 0.55612903,
       0.55612903, 0.55612903, 0.55612903, 0.58709677, 0.58709677,
       0.58709677, 0.58709677, 0.61677419, 0.61677419, 0.61677419,
       0.61677419, 0.64516129, 0.64516129, 0.64516129, 0.64516129,
       0.64903226, 0.64903226, 0.64516129, 0.64516129, 0.66967742,
       0.66709677, 0.66322581, 0.66193548, 0.68645161, 0.68387097,
       0.68      , 0.66193548, 0.69419355, 0.69548387, 0.6916129 ,
       0.66193548, 0.69419355, 0.70322581, 0.6916129 , 0.67354839,
       0.70580645, 0.70322581, 0.6916129 , 0.68258065, 0.71354839,
       0.71096774, 0.6916129 , 0.69290323, 0.72      , 0.71612903,
       0.69935484, 0.69290323, 0.72      , 0.72129032, 0.70967742,
       0.69290323, 0.72      , 0.72516129, 0.71483871, 0.69290323,
       0.72      , 0.72774194, 0.71870968, 0.69290323, 0.72387097,
       0.72774194, 0.72129032, 0.69290323, 0.72903226, 0.72774194,
       0.72129032, 0.69806452, 0.72903226, 0.72774194, 0.72129032,
       0.70322581, 0.54580645, 0.54580645, 0.54580645, 0.54580645,
       0.55225806, 0.55225806, 0.55225806, 0.55225806, 0.5883871 ,
       0.5883871 , 0.5883871 , 0.5883871 , 0.61677419, 0.61677419,
       0.61677419, 0.61677419, 0.64516129, 0.64516129, 0.64516129,
       0.64516129, 0.64903226, 0.64903226, 0.64516129, 0.64516129,
       0.66967742, 0.66709677, 0.66322581, 0.66193548, 0.68774194,
       0.68387097, 0.68      , 0.66193548, 0.69548387, 0.68387097,
       0.68      , 0.66193548, 0.69548387, 0.69548387, 0.6916129 ,
       0.67354839, 0.70709677, 0.69548387, 0.6916129 , 0.68258065,
       0.70709677, 0.69548387, 0.6916129 , 0.69290323, 0.71354839,
       0.70193548, 0.69935484, 0.69290323, 0.72258065, 0.70967742,
       0.70967742, 0.69290323, 0.72258065, 0.71483871, 0.71483871,
       0.69290323, 0.72258065, 0.71483871, 0.71870968, 0.69290323,
       0.72645161, 0.71483871, 0.72129032, 0.69290323, 0.72645161,
       0.71483871, 0.72129032, 0.69806452, 0.72645161, 0.71483871,
```

```
       0.72129032, 0.70322581]), 'split9_train_score': array([0.54580645, 0.54
580645, 0.54580645, 0.54580645, 0.55225806,
       0.55225806, 0.55225806, 0.55225806, 0.58451613, 0.58451613,
       0.58451613, 0.58451613, 0.61032258, 0.61032258, 0.61032258,
       0.61032258, 0.63225806, 0.63225806, 0.63225806, 0.61032258,
       0.63225806, 0.63225806, 0.63225806, 0.63870968, 0.66193548,
       0.66064516, 0.66064516, 0.63870968, 0.66193548, 0.66064516,
       0.66064516, 0.63870968, 0.66193548, 0.66064516, 0.66064516,
       0.65419355, 0.66193548, 0.66064516, 0.66064516, 0.66967742,
       0.66967742, 0.66451613, 0.67483871, 0.68258065, 0.66967742,
       0.67483871, 0.67483871, 0.6916129 , 0.68387097, 0.68129032,
       0.68      , 0.69935484, 0.68387097, 0.68903226, 0.69419355,
       0.7083871 , 0.6916129 , 0.70193548, 0.70322581, 0.71612903,
       0.69677419, 0.70193548, 0.71096774, 0.71612903, 0.69677419,
       0.70580645, 0.71741935, 0.71612903, 0.70967742, 0.71354839,
       0.71741935, 0.71612903, 0.71870968, 0.72258065, 0.71741935,
       0.71612903, 0.54580645, 0.54580645, 0.54580645, 0.54580645,
       0.55225806, 0.55225806, 0.55225806, 0.55225806, 0.58451613,
       0.58451613, 0.58451613, 0.58451613, 0.61032258, 0.61032258,
       0.61032258, 0.61032258, 0.63225806, 0.63225806, 0.63225806,
       0.61032258, 0.63225806, 0.63225806, 0.63225806, 0.63870968,
       0.66064516, 0.66064516, 0.66064516, 0.63870968, 0.66451613,
       0.66064516, 0.66064516, 0.63870968, 0.66451613, 0.66064516,
       0.66064516, 0.65419355, 0.66451613, 0.66064516, 0.66064516,
       0.66967742, 0.67225806, 0.66451613, 0.67483871, 0.68258065,
       0.67225806, 0.67483871, 0.67483871, 0.6916129 , 0.68645161,
       0.68129032, 0.68      , 0.69935484, 0.68645161, 0.68903226,
       0.69419355, 0.7083871 , 0.69419355, 0.70193548, 0.70322581,
       0.71612903, 0.69935484, 0.70193548, 0.71096774, 0.71612903,
       0.69935484, 0.70580645, 0.71741935, 0.71612903, 0.71096774,
       0.71354839, 0.71741935, 0.71612903, 0.72129032, 0.72258065,
       0.71741935, 0.71612903]), 'mean_train_score': array([0.54587697, 0.5458
7697, 0.54587697, 0.54587697, 0.55039393,
       0.55039393, 0.55039393, 0.55039393, 0.58330116, 0.58330116,
       0.58330116, 0.58330116, 0.61375761, 0.61375761, 0.61375761,
       0.6141447 , 0.63453497, 0.63414787, 0.63376077, 0.63298658,
       0.65066683, 0.64860182, 0.64756956, 0.64059948, 0.66679587,
       0.66382762, 0.6625373 , 0.65879587, 0.68370193, 0.67957239,
       0.67828207, 0.6621507 , 0.68976644, 0.68408852, 0.6827982 ,
       0.66486038, 0.6904116 , 0.68589497, 0.68383046, 0.67195866,
       0.69402451, 0.68795949, 0.6864111 , 0.67802451, 0.69725181,
       0.69454164, 0.69131583, 0.68563841, 0.70228407, 0.70073635,
       0.69815571, 0.69273585, 0.70602701, 0.70602751, 0.70486638,
       0.69531683, 0.71054447, 0.71196382, 0.70989947, 0.69712361,
       0.71428741, 0.7144161 , 0.71389964, 0.69905926, 0.72009469,
       0.71764191, 0.71660932, 0.70099475, 0.72474035, 0.7188032 ,
       0.71789997, 0.70344636, 0.72732116, 0.72048062, 0.71854514,
       0.70447862, 0.54587697, 0.54587697, 0.54587697, 0.54587697,
       0.54845761, 0.54845761, 0.54845761, 0.54845761, 0.58304259,
       0.58304259, 0.58304259, 0.58304259, 0.60923998, 0.60923998,
       0.60923998, 0.60962707, 0.62898625, 0.62885721, 0.62847012,
       0.62950238, 0.64834109, 0.64692173, 0.64614754, 0.63995399,
       0.66240877, 0.66021472, 0.6589244 , 0.6546665 , 0.68047629,
       0.67789497, 0.67660465, 0.65969876, 0.69170209, 0.68357239,
       0.68228207, 0.66408585, 0.69234725, 0.68576594, 0.68447562,
       0.6702809 , 0.69428274, 0.68666917, 0.68589497, 0.67660465,
       0.69557306, 0.69363841, 0.69196099, 0.68331533, 0.70163858,
       0.70034925, 0.69918796, 0.69015487, 0.70512395, 0.70525331,
       0.70564058, 0.69325181, 0.71196399, 0.71093157, 0.71067367,
       0.6949294 , 0.71570693, 0.71338385, 0.71428674, 0.69738101,
       0.72177228, 0.71648062, 0.71699642, 0.70021989, 0.72577278,
       0.71777094, 0.71828707, 0.70331666, 0.72848262, 0.71957739,
       0.71893223, 0.70486505]), 'std_train_score': array([0.00021155, 0.00021
155, 0.00021155, 0.00021155, 0.00432661,
       0.00432661, 0.00432661, 0.00432661, 0.00322082, 0.00322082,
       0.00322082, 0.00322082, 0.00521679, 0.00521679, 0.00521679,
       0.0044621 , 0.01055543, 0.01115877, 0.01083237, 0.01335437,
       0.01218041, 0.01206254, 0.01206435, 0.00377134, 0.00554842,
       0.00673278, 0.00654886, 0.00738517, 0.00776152, 0.00693494,
```

```
       0.00660757, 0.0103589 , 0.01095056, 0.01075357, 0.01043432,
       0.00923646, 0.01144719, 0.01263092, 0.0115758 , 0.01079345,
       0.01153223, 0.01465776, 0.01158483, 0.01108998, 0.01333679,
       0.0125271 , 0.01040722, 0.01062992, 0.01261853, 0.01053099,
       0.00874121, 0.00787166, 0.01295505, 0.01064299, 0.00878724,
       0.00852205, 0.01200817, 0.00962302, 0.00890938, 0.00963082,
       0.01090912, 0.01035157, 0.00746212, 0.00812912, 0.01081997,
       0.0073952 , 0.00519207, 0.00653215, 0.00923964, 0.00642927,
       0.00406781, 0.00656526, 0.00837525, 0.00487063, 0.00336874,
       0.0069216 , 0.00021155, 0.00021155, 0.00021155, 0.00021155,
       0.00276992, 0.00276992, 0.00276992, 0.00276992, 0.00352554,
       0.00352554, 0.00352554, 0.00352554, 0.01016912, 0.01016912,
       0.01016912, 0.00998018, 0.01376728, 0.01401793, 0.01386438,
       0.01446715, 0.0121394 , 0.01232394, 0.01200068, 0.00355659,
       0.00618069, 0.00574135, 0.00518242, 0.00932918, 0.00954468,
       0.00822518, 0.00763056, 0.01187556, 0.01067969, 0.01014613,
       0.01015738, 0.00931655, 0.01115108, 0.01170056, 0.01156727,
       0.00841228, 0.01021136, 0.01166879, 0.00918069, 0.01057527,
       0.01090991, 0.01119258, 0.01036556, 0.01044556, 0.00902087,
       0.01013234, 0.00911848, 0.00926992, 0.01171262, 0.00969721,
       0.00883583, 0.00850225, 0.01254769, 0.00854098, 0.00874276,
       0.0097816 , 0.01075372, 0.00901343, 0.00693558, 0.00784772,
       0.01063126, 0.00608617, 0.00468408, 0.00608269, 0.00934999,
       0.00487993, 0.003723  , 0.0062477 , 0.00827799, 0.00350535,
       0.00339005, 0.00683209])}
```

In [22]:

```python
dd = pd.DataFrame(result_set['params'])

index_ = list(dd.index[(dd['criterion']=='entropy') & (dd['min_samples_leaf']=

train_result = result_set['mean_train_score']
test_result = result_set['mean_test_score']

max_depth_train = []
max_depth_test = []
index_
for i in range(len(index_)):
    max_depth_train.append(train_result[index_[i]])
    max_depth_test.append(test_result[index_[i]])
plt.plot(range(1, len(max_depth_train)+1), max_depth_train, 'b', range(1,len(m
plt.xlabel('Maximum depth (model complexity)\nBlue = training acc. Red = test
plt.xticks(np.arange(1, len(max_depth_train)+1, 1))
plt.ylabel('score')
plt.show()
```



In [23]:

```python
cv_1.fit(x_train, y_train)
print("Train accuracy:", cv_1.score(x_train, y_train))
```

```
print("Test accuracy:", cv_1.score(x_test, y_test))
```

```
Train accuracy: 0.7177700348432056
Test accuracy: 0.6747967479674797
```

In [24]:

```
print(cv_1.best_params_)   #best hyperparameters
```

```
{'criterion': 'gini', 'max_depth': 18, 'min_samples_leaf': 15}
```

In [25]:

```
print(cv_1.best_estimator_)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=18,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=15, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=10, splitter='best')
```

In [26]:

```
print(cv_1.best_score_)
```

```
0.6794425087108014
```

In [27]:

```
# grab feature importances from the model and feature name from the original .
importances = cv_1.best_estimator_.feature_importances_
feature_names = x.columns
# sort them out in descending order
indices = np.argsort(importances)
indices = np.flip(indices, axis=0)
# limit to 20 features, you can leave this out to print out everything
indices = indices[:20]
for i in indices:
   print(feature_names[i], ':', importances[i])
```

```
Patient age quantile_1 : 0.1570030356594264
Influenza B rapid test_negative : 0.12242876717472596
Influenza A rapid test_negative : 0.1223779285913228
Eosinophils : 0.10892456465989904
Patient age quantile_2 : 0.08663119955059644
Leukocytes : 0.08590959598391089
Patient age quantile_0 : 0.0828331711635059
Patient age quantile_17 : 0.05254993773407766
Patient age quantile_5 : 0.029179752235000394
Patient age quantile_19 : 0.028757683744032734
Patient age quantile_4 : 0.027114513316328616
Patient age quantile_13 : 0.0192539114324268
Patient age quantile_3 : 0.01680834307569263
Red blood Cells : 0.012782006037245618
Patient age quantile_18 : 0.01171680381973257
Patient age quantile_14 : 0.011679333188114996
Patient age quantile_11 : 0.009979014634518463
Patient age quantile_16 : 0.008323475485032551
Hemoglobin : 0.005746962514409674
Platelets : 0.0
```

In [28]:

```
# Size of gridsearchCV tuned decision tree
a = cv_1.best_estimator_.tree_.node_count
a
```

Out[28]: 41

In [1]:

```
# inside `dm_tools.py' together with data_prep()
import numpy as np
```

```
import pydot
from io import StringIO
from sklearn.tree import export_graphviz

def analyse_feature_importance(dm_model, feature_names, n_to_display=20):
    # grab feature importances from the model
    importances = dm_model.feature_importances_

    # sort them out in descending order
    indices = np.argsort(importances)
    indices = np.flip(indices, axis=0)

    # limit to 20 features, you can leave this out to print out everything
    indices = indices[:n_to_display]

    for i in indices:
        print(feature_names[i], ':', importances[i])

def visualize_decision_tree(dm_model, feature_names, save_name):
    dotfile = StringIO()
    export_graphviz(dm_model, out_file=dotfile, feature_names=feature_names)
    graph = pydot.graph_from_dot_data(dotfile.getvalue())
    graph[0].write_png(save_name) # saved in the following file
```

In [30]:

```
# do the feature importance and visualization analysis on GridSearchCV
from dm_tools import analyse_feature_importance, visualize_decision_tree

analyse_feature_importance(cv_1.best_estimator_, x.columns, 20)
visualize_decision_tree(cv_1.best_estimator_, x.columns, "dt_gridsearchcv.png
```

```
Patient age quantile_1 : 0.1570030356594264
Influenza B rapid test_negative : 0.12242876717472596
Influenza A rapid test_negative : 0.1223779285913228
Eosinophils : 0.10892456465989904
Patient age quantile_2 : 0.08663119955059644
Leukocytes : 0.08590959598391089
Patient age quantile_0 : 0.0828331711635059
Patient age quantile_17 : 0.05254993773407766
Patient age quantile_5 : 0.029179752235000394
Patient age quantile_19 : 0.028757683744032734
Patient age quantile_4 : 0.027114513316328616
Patient age quantile_13 : 0.019253911432426 8
Patient age quantile_3 : 0.01680834307569263
Red blood Cells : 0.012782006037245618
Patient age quantile_18 : 0.01171680381973257
Patient age quantile_14 : 0.011679333188114996
Patient age quantile_11 : 0.009979014634518463
Patient age quantile_16 : 0.008323475485032551
Hemoglobin : 0.005746962514409674
Platelets : 0.0
```

Comparison

In [31]:

```
y_pred_dt = model.predict(x_test)
y_pred_dt_cv = cv_1.predict(x_test)

print(classification_report(y_test, y_pred_dt))
print(classification_report(y_test, y_pred_dt_cv))
```

```
              precision    recall  f1-score   support

           0       0.71      0.69      0.70       202
           1       0.63      0.65      0.64       167
```

```
    accuracy                           0.67       369
   macro avg       0.67      0.67      0.67       369
weighted avg       0.67      0.67      0.67       369

              precision    recall  f1-score   support

           0       0.70      0.71      0.70       202
           1       0.64      0.63      0.64       167

    accuracy                           0.67       369
   macro avg       0.67      0.67      0.67       369
weighted avg       0.67      0.67      0.67       369
```

In [32]:

```
from sklearn.metrics import roc_auc_score

dt_cv_best = cv_1.best_estimator_

y_pred_proba_dt = model.predict_proba(x_test)
y_pred_proba_dt_cv = dt_cv_best.predict_proba(x_test)

roc_index_dt = roc_auc_score(y_test, y_pred_proba_dt[:, 1])
roc_index_dt_cv = roc_auc_score(y_test, y_pred_proba_dt_cv[:, 1])

print("ROC index on test for DT_default:", roc_index_dt)
print("ROC index on test for DT_optimal:", roc_index_dt_cv)
```

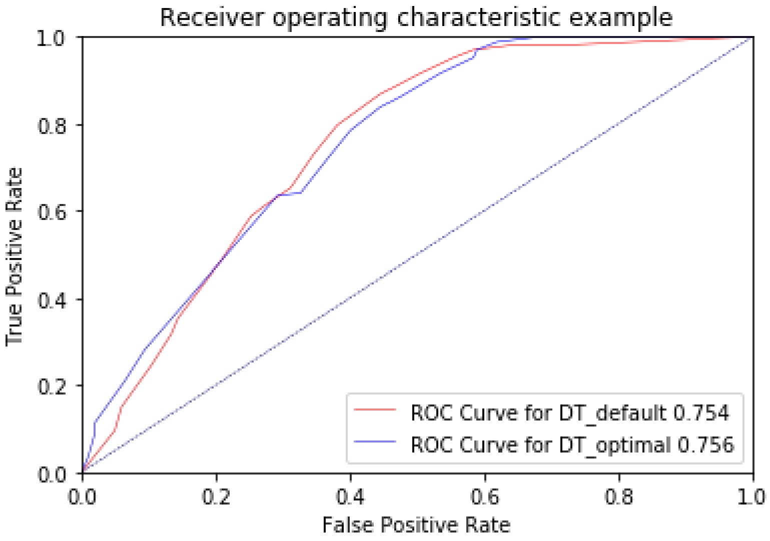
```
ROC index on test for DT_default: 0.7539426098298452
ROC index on test for DT_optimal: 0.7561807078911484
```

In [33]:

```
from sklearn.metrics import roc_curve

fpr_dt, tpr_dt, thresholds_dt = roc_curve(y_test, y_pred_proba_dt[:,1])
fpr_dt_cv, tpr_dt_cv, thresholds_dt_cv = roc_curve(y_test, y_pred_proba_dt_cv
```

In [34]:

```
# plotting on matplot

import matplotlib.pyplot as plt

plt.plot(fpr_dt, tpr_dt, label='ROC Curve for DT_default {:.3f}'.format(roc_i
plt.plot(fpr_dt_cv, tpr_dt_cv, label='ROC Curve for DT_optimal {:.3f}'.format

# plt.plot(fpr[2], tpr[2], color='darkorange',
#          lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[2])
plt.plot([0, 1], [0, 1], color='navy', lw=0.5, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```



In [35]:

```
# Save best DT model
import pickle
dt_best = cv_1
with open('DT.pickle', 'wb') as f:
    pickle.dump([dt_best, roc_index_dt_cv, fpr_dt_cv, tpr_dt_cv], f)
```

In [36]:

```
dt_cv_best = cv_1.best_estimator_
# probability prediction from decision tree
y_pred_proba_dt = dt_cv_best.predict_proba(x_test)

print("Probability produced by decision tree for each class vs actual predict
print("(Probs on zero)\t(probs on one)\t(prediction made)")
# print top 10
for i in range(20):
    print(y_pred_proba_dt[i][0], '\t', y_pred_proba_dt[i][1], '\t', y_pred[i]
```

```
Probability produced by decision tree for each class vs actual prediction on C
OVID test (0 = negative, 1 = positive). You should be able to see the default
threshold of 0.5.
(Probs on zero) (probs on one)  (prediction made)
0.9444444444444444       0.05555555555555555     0
0.5135135135135135       0.4864864864864865      0
0.25      0.75     1
0.36403508771929827      0.6359649122807017      1
0.9444444444444444       0.05555555555555555     0
0.36403508771929827      0.6359649122807017      1
0.6046511627906976       0.3953488372093023      0
0.5135135135135135       0.4864864864864865      0
0.2       0.8      1
0.36403508771929827      0.6359649122807017      1
0.36403508771929827      0.6359649122807017      1
0.7297297297297297       0.2702702702702703      0
0.9642857142857143       0.03571428571428571     0
0.36403508771929827      0.6359649122807017      1
1.0       0.0      0
0.5135135135135135       0.4864864864864865      0
0.24390243902439024      0.7560975609756098      1
0.24390243902439024      0.7560975609756098      1
0.24390243902439024      0.7560975609756098      1
0.6046511627906976       0.3953488372093023      0
```

Task 3 Predictive modeling using Regression

In [37]:

```
from sklearn.preprocessing import StandardScaler
```

```
# initialise a standard scaler object
scaler = StandardScaler()

# visualise min, max, mean and standard dev of data before scaling
print("Before scaling\n-------------")
for i in range(5):
    col = x_train[:,i]
    print("Variable #{}: min {}, max {}, mean {:.2f} and std dev {:.2f}".
          format(i, min(col), max(col), np.mean(col), np.std(col)))

# learn the mean and std.dev of variables from training data
# then use the learned values to transform training data
x_train_r = scaler.fit_transform(x_train, y_train)

print("After scaling\n-------------")
for i in range(5):
    col = x_train_r[:,i]
    print("Variable #{}: min {}, max {}, mean {:.2f} and std dev {:.2f}".
          format(i, min(col), max(col), np.mean(col), np.std(col)))

# use the statistic that you learned from training to transform test data
# NEVER learn from test data, this is supposed to be a set of dataset
# that the model has never seen before
x_test_r = scaler.transform(x_test)
```

```
Before scaling
-------------
Variable #0: min -0.5333752039999999, max 5.9462704660000005, mean 0.13 and st
d dev 0.40
Variable #1: min -3.3397746089999996, max 2.0524332519999997, mean 0.10 and st
d dev 0.38
Variable #2: min -1.896609068, max 2.7033133510000003, mean 0.10 and std dev
0.40
Variable #3: min -2.058668613, max 3.6404480930000003, mean 0.35 and std dev
0.45
Variable #4: min -1.33272469, max 4.947685719, mean -0.13 and std dev 0.33
After scaling
-------------
Variable #0: min -1.6558874437366855, max 14.640584773884214, mean -0.00 and s
td dev 1.00
Variable #1: min -9.172007001005262, max 5.189984710396853, mean -0.00 and std
dev 1.00
Variable #2: min -5.029834523257916, max 6.566421406080185, mean -0.00 and std
dev 1.00
Variable #3: min -5.367527157787365, max 7.339173450938573, mean -0.00 and std
dev 1.00
Variable #4: min -3.606588682308649, max 15.272491322537062, mean 0.00 and std
dev 1.00
```

Training logistic regression

In [38]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.tree import export_graphviz
```

In [39]:

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(random_state=rs)
```

```
# fit it to training data
model.fit(x_train_r, y_train)
```

```
/Users/juhijoshi/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_mode
l/logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in
0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[39]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=10, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [40]:
```
# training and test accuracy
print("Train accuracy:", model.score(x_train_r, y_train))
print("Test accuracy:", model.score(x_test_r, y_test))

# classification report on test data
y_pred = model.predict(x_test_r)
print(classification_report(y_test, y_pred))
```

```
Train accuracy: 0.7200929152148664
Test accuracy: 0.6883468834688347
              precision    recall  f1-score   support

           0       0.75      0.64      0.69       202
           1       0.63      0.75      0.68       167

    accuracy                           0.69       369
   macro avg       0.69      0.69      0.69       369
weighted avg       0.70      0.69      0.69       369
```

In [41]:
```
print (model.coef_)
```

```
[[ 0.03196252 -0.49981776 -0.20393726 -0.22500394 -0.0151027   0.35570774
  -0.37275488 -0.66118368  0.00440295 -0.34071599 -0.05984338 -0.09537778
   0.14658864 -0.3149829  -0.24470554 -0.208515   -0.45992371 -0.60627937
   0.15387141  0.01912862  0.13833858 -0.05402433  0.07016482  0.10046937
  -0.03124737 -0.14376656  0.04726437  0.42175694 -0.46685439 -0.11837888
   0.27856696  0.27834921  0.11363259  0.07433206  0.11441356  0.11085083
   0.36656191 -0.36656191  0.62555904 -0.62555904]]
```

In [42]:
```
feature_names = x.columns
coef = model.coef_[0]

for i in range(len(coef)):
    print(feature_names[i], ':', coef[i])
```

```
Proteina C reativa : 0.03196252211311077
Neutrophils : -0.4998177580343138
Mean platelet volume : -0.20393726062460812
Monocytes : -0.22500393977365635
Red blood cell distribution width : -0.015102703690835953
Red blood Cells : 0.3557077353372009
Platelets : -0.3727548835145546
Eosinophils : -0.6611836772382872
Basophils : 0.0044029504140295625
Leukocytes : -0.34071599042125006
Mean corpuscular hemoglobin : -0.0598433826801642
Mean corpuscular volume : -0.0953777825738466
Mean corpuscular hemoglobin concentration : 0.14658864271542002
Lymphocytes : -0.3149829041057197
Hemoglobin : -0.24470554322078925
```

```
Hematocrit : -0.20851500003561768
Patient age quantile_0 : -0.45992370956498985
Patient age quantile_1 : -0.606279365222407
Patient age quantile_10 : 0.15387141042882602
Patient age quantile_11 : 0.019128617561474439
Patient age quantile_12 : 0.138338578695753
Patient age quantile_13 : -0.05402433240901891
Patient age quantile_14 : 0.07016481783779789
Patient age quantile_15 : 0.10046936643045229
Patient age quantile_16 : -0.031247370132925476
Patient age quantile_17 : -0.14376655996890242
Patient age quantile_18 : 0.047264366756236346
Patient age quantile_19 : 0.4217569383776665
Patient age quantile_2 : -0.4668543926530027
Patient age quantile_3 : -0.11837887612976279
Patient age quantile_4 : 0.27856695809711607
Patient age quantile_5 : 0.2783492133091274
Patient age quantile_6 : 0.11363258875859865
Patient age quantile_7 : 0.07433205948462235
Patient age quantile_8 : 0.11441355749785796
Patient age quantile_9 : 0.11085082808128419
Influenza B rapid test_negative : 0.3665619145823581
Influenza B rapid test_positive : -0.36656191458234183
Influenza A rapid test_negative : 0.6255590360339798
Influenza A rapid test_positive : -0.6255590360339819
```

In [43]:

```
# grab feature importances from the model and feature name from the original
coef = model.coef_[0]
feature_names = x.columns

# sort them out in descending order
indices = np.argsort(np.absolute(coef))
indices = np.flip(indices, axis=0)

for i in indices:
    print(feature_names[i], ':', coef[i])
```

```
Eosinophils : -0.6611836772382872
Influenza A rapid test_positive : -0.6255590360339819
Influenza A rapid test_negative : 0.6255590360339798
Patient age quantile_1 : -0.606279365222407
Neutrophils : -0.4998177580343138
Patient age quantile_2 : -0.4668543926530027
Patient age quantile_0 : -0.45992370956498985
Patient age quantile_19 : 0.4217569383776665
Platelets : -0.3727548835145546
Influenza B rapid test_negative : 0.3665619145823581
Influenza B rapid test_positive : -0.36656191458234183
Red blood Cells : 0.3557077353372009
Leukocytes : -0.34071599042125006
Lymphocytes : -0.3149829041057197
Patient age quantile_4 : 0.27856695809711607
Patient age quantile_5 : 0.2783492133091274
Hemoglobin : -0.24470554322078925
Monocytes : -0.22500393977365635
Hematocrit : -0.20851500003561768
Mean platelet volume : -0.20393726062460812
Patient age quantile_10 : 0.15387141042882602
Mean corpuscular hemoglobin concentration : 0.14658864271542002
Patient age quantile_17 : -0.14376655996890242
Patient age quantile_12 : 0.138338578695753
Patient age quantile_3 : -0.11837887612976279
Patient age quantile_8 : 0.11441355749785796
Patient age quantile_6 : 0.11363258875859865
Patient age quantile_9 : 0.11085082808128419
Patient age quantile_15 : 0.10046936643045229
Mean corpuscular volume : -0.0953777825738466
Patient age quantile_7 : 0.07433205948462235
```

```
Patient age quantile_14 : 0.07016481783779789
Mean corpuscular hemoglobin : -0.0598433826801642
Patient age quantile_13 : -0.05402433240901891
Patient age quantile_18 : 0.047264366756236346
Proteina C reativa : 0.0319625221311077
Patient age quantile_16 : -0.031247370132925476
Patient age quantile_11 : 0.01912861756147439
Red blood cell distribution width : -0.015102703690835953
Basophils : 0.0044029504140295625
```

Finding optimal hyperparameters with GridSearchCV

In [44]:

```
# grid search CV
params = {'C': [pow(10, x) for x in range(-6, 4)]}

# use all cores to tune logistic regression with C parameter
cv = GridSearchCV(param_grid=params, estimator=LogisticRegression(random_stat
cv.fit(x_train_r, y_train)
```

```
/Users/juhijoshi/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_mode
l/logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in
0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[44]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fals
e,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='warn',
                                          n_jobs=None, penalty='l2',
                                          random_state=10, solver='warn',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=-1,
             param_grid={'C': [1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10,
                               100, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [43]:

```
result_set = cv.cv_results_
print(result_set)
```

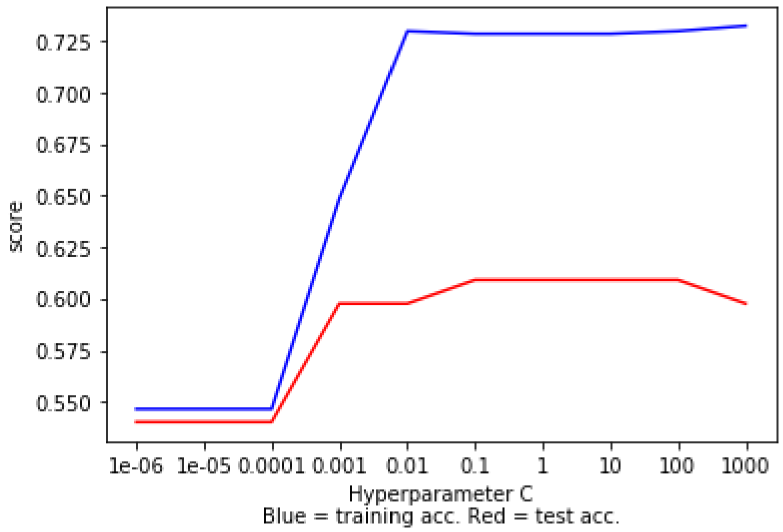
```
{'mean_fit_time': array([0.00717261, 0.00565956, 0.0043345 , 0.0081146 , 0.004
00882,
       0.00853028, 0.01240828, 0.02408087, 0.02760525, 0.02946064]), 'std_fit_
time': array([0.00557404, 0.0012688 , 0.00081027, 0.0074386 , 0.00014489,
       0.00426969, 0.00584624, 0.00576897, 0.00381496, 0.00473896]), 'mean_sco
re_time': array([0.00054216, 0.00052555, 0.00046353, 0.00046618, 0.00046294,
       0.00048997, 0.00052686, 0.00054221, 0.00050697, 0.00051818]), 'std_scor
e_time': array([1.03877199e-04, 1.15732909e-04, 1.59369222e-05, 3.00275020e-0
5,
       2.01930966e-05, 3.87634370e-05, 9.00260346e-05, 8.84296920e-05,
       6.97714934e-05, 7.60972348e-05]), 'param_C': masked_array(data=[1e-06,
1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100,
                   1000],
             mask=[False, False, False, False, False, False, False, False,
                   False, False],
       fill_value='?',
            dtype=object), 'params': [{'C': 1e-06}, {'C': 1e-05}, {'C': 0.000
1}, {'C': 0.001}, {'C': 0.01}, {'C': 0.1}, {'C': 1}, {'C': 10}, {'C': 100},
{'C': 1000}], 'split0_test_score': array([0.54022989, 0.54022989, 0.54022989,
0.59770115, 0.59770115,
       0.6091954 , 0.6091954 , 0.6091954 , 0.6091954 , 0.59770115]), 'split1_t
```

```
est_score': array([0.54651163, 0.54651163, 0.54651163, 0.58139535, 0.58139535,
       0.60465116, 0.62790698, 0.63953488, 0.63953488, 0.63953488]), 'split2_t
est_score': array([0.54651163, 0.54651163, 0.54651163, 0.62790698, 0.69767442,
       0.72093023, 0.72093023, 0.72093023, 0.72093023, 0.70930233]), 'split3_t
est_score': array([0.54651163, 0.54651163, 0.54651163, 0.63953488, 0.72093023,
       0.72093023, 0.72093023, 0.70930233, 0.70930233, 0.70930233]), 'split4_t
est_score': array([0.54651163, 0.54651163, 0.54651163, 0.6744186 , 0.74418605,
       0.75581395, 0.75581395, 0.74418605, 0.74418605, 0.74418605]), 'split5_t
est_score': array([0.54651163, 0.54651163, 0.54651163, 0.70930233, 0.73255814,
       0.72093023, 0.70930233, 0.72093023, 0.72093023, 0.72093023]), 'split6_t
est_score': array([0.54651163, 0.54651163, 0.54651163, 0.62790698, 0.69767442,
       0.70930233, 0.72093023, 0.72093023, 0.73255814, 0.72093023]), 'split7_t
est_score': array([0.54651163, 0.54651163, 0.54651163, 0.6744186 , 0.69767442,
       0.69767442, 0.69767442, 0.69767442, 0.69767442, 0.69767442]), 'split8_t
est_score': array([0.54651163, 0.54651163, 0.54651163, 0.65116279, 0.6744186 ,
       0.63953488, 0.65116279, 0.65116279, 0.65116279, 0.65116279]), 'split9_t
est_score': array([0.54651163, 0.54651163, 0.54651163, 0.56976744, 0.70930233,
       0.70930233, 0.70930233, 0.70930233, 0.70930233, 0.69767442]), 'mean_tes
t_score': array([0.54588345, 0.54588345, 0.54588345, 0.63535151, 0.68535151,
       0.68882652, 0.69231489, 0.69231489, 0.69347768, 0.68883988]), 'std_test
_score': array([0.00188452, 0.00188452, 0.00188452, 0.0418684 , 0.05156939,
       0.04934809, 0.044548  , 0.04140178, 0.04234175, 0.04274706]), 'rank_tes
t_score': array([8, 8, 8, 7, 6, 5, 2, 2, 1, 4], dtype=int32), 'split0_train_sc
ore': array([0.54651163, 0.54651163, 0.54651163, 0.64857881, 0.72997416,
       0.72868217, 0.72868217, 0.72868217, 0.72997416, 0.73255814]), 'split1_t
rain_score': array([0.54580645, 0.54580645, 0.54580645, 0.64645161, 0.7264516
1,
       0.72645161, 0.72645161, 0.72774194, 0.72516129, 0.72516129]), 'split2_t
rain_score': array([0.54580645, 0.54580645, 0.54580645, 0.64645161, 0.7122580
6,
       0.71225806, 0.71354839, 0.71354839, 0.71354839, 0.71354839]), 'split3_t
rain_score': array([0.54580645, 0.54580645, 0.54580645, 0.64387097, 0.7212903
2,
       0.71741935, 0.72      , 0.72129032, 0.72129032, 0.71870968]), 'split4_t
rain_score': array([0.54580645, 0.54580645, 0.54580645, 0.63870968, 0.7109677
4,
       0.71483871, 0.71096774, 0.71354839, 0.71612903, 0.71612903]), 'split5_t
rain_score': array([0.54580645, 0.54580645, 0.54580645, 0.63870968, 0.72
,
       0.72      , 0.71741935, 0.72      , 0.71612903, 0.71870968]), 'split6_t
rain_score': array([0.54580645, 0.54580645, 0.54580645, 0.64387097, 0.72
,
       0.71741935, 0.71483871, 0.71483871, 0.71612903, 0.71483871]), 'split7_t
rain_score': array([0.54580645, 0.54580645, 0.54580645, 0.64258065, 0.7161290
3,
       0.71741935, 0.71870968, 0.71870968, 0.71483871, 0.71483871]), 'split8_t
rain_score': array([0.54580645, 0.54580645, 0.54580645, 0.65419355, 0.7238709
7,
       0.72645161, 0.72516129, 0.72516129, 0.72258065, 0.72387097]), 'split9_t
rain_score': array([0.54580645, 0.54580645, 0.54580645, 0.65032258, 0.7187096
8,
       0.71741935, 0.71612903, 0.71483871, 0.71741935, 0.71741935]), 'mean_tra
in_score': array([0.54587697, 0.54587697, 0.54587697, 0.64537401, 0.71996516,
       0.71983596, 0.7191908 , 0.71983596, 0.71932   , 0.71957839]), 'std_trai
n_score': array([0.00021155, 0.00021155, 0.00021155, 0.00464773, 0.00562061,
       0.00521127, 0.00556998, 0.00549128, 0.00499698, 0.00563407])}
```

In [44]:

```
import matplotlib.pyplot as plt
train_result = result_set['split0_train_score']
test_result = result_set['split0_test_score']
print("Total number of models: ", len(test_result))
# plot Hyperparameter C values vs training and test accuracy score
plt.plot(range(0, len(train_result)), train_result, 'b', range(0,len(test_res
plt.xlabel('Hyperparameter C\nBlue = training acc. Red = test acc.')
plt.xticks(range(0, len(train_result)),[pow(10, x) for x in range(-6, 4)])
plt.ylabel('score')
plt.show()
```

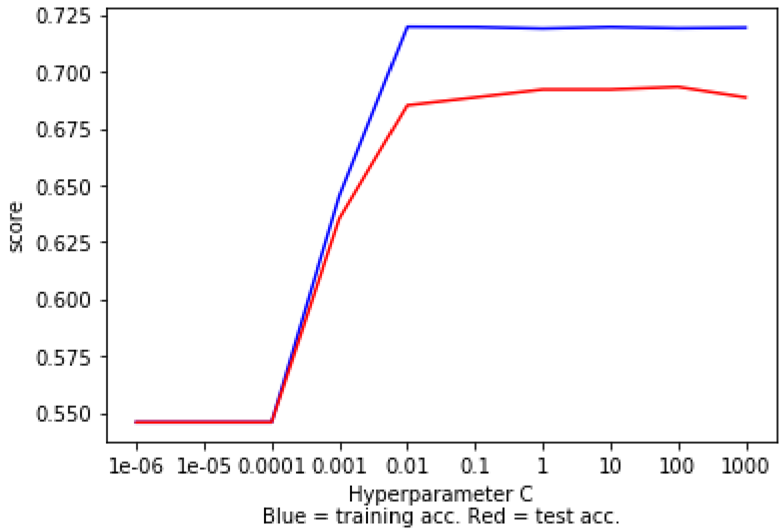
```
Total number of models:  10
```



In [45]:

```
import matplotlib.pyplot as plt

train_result = result_set['mean_train_score']
test_result = result_set['mean_test_score']
print("Total number of models: ", len(test_result))
# plot Hyperparameter C values vs training and test accuracy score
plt.plot(range(0, len(train_result)), train_result, 'b', range(0,len(test_res
plt.xlabel('Hyperparameter C\nBlue = training acc. Red = test acc.')
plt.xticks(range(0, len(train_result)),[pow(10, x) for x in range(-6, 4)])
plt.ylabel('score')
plt.show()
```

```
Total number of models:  10
```



In [46]:

```
print(cv.best_params_)
```

```
{'C': 100}
```

In [47]:

```
cv.fit(x_train_r, y_train)

print("Train accuracy:", cv.score(x_train_r, y_train))
print("Test accuracy:", cv.score(x_test_r, y_test))
```

```
Train accuracy: 0.7177700348432056
Test accuracy: 0.6883468834688347
```

Feature selection

Feature selection using Recursive Feature Elimination

In [48]:

```
from sklearn.feature_selection import RFECV

rfe = RFECV(estimator = LogisticRegression(random_state=rs), cv=10)
rfe.fit(x_train_r, y_train) # run the RFECV

# comparing how many variables before and after
print("Original feature set", x_train_r.shape[1])
print("Number of features after elimination", rfe.n_features_)
```

```
Original feature set 40
Number of features after elimination 38
```

In [53]:

```
X_train_sel = rfe.transform(x_train_r)
X_test_sel = rfe.transform(x_test_r)
```

In [54]:

```
dset = pd.DataFrame()
dset["fn"] = x.columns
dset["choose"] = rfe.support_

dset.sort_values(by=['choose'])
```

Out[54]:

	fn	choose
4	Red blood cell distribution width	False
8	Basophils	False
0	Proteina C reativa	True
23	Patient age quantile_15	True
24	Patient age quantile_16	True
25	Patient age quantile_17	True
26	Patient age quantile_18	True
27	Patient age quantile_19	True
28	Patient age quantile_2	True
29	Patient age quantile_3	True
31	Patient age quantile_5	True
22	Patient age quantile_14	True
32	Patient age quantile_6	True
33	Patient age quantile_7	True
34	Patient age quantile_8	True
35	Patient age quantile_9	True
36	Influenza B rapid test_negative	True
37	Influenza B rapid test_positive	True
30	Patient age quantile_4	True
21	Patient age quantile_13	True

	fn	choose
19	Patient age quantile_11	True
38	Influenza A rapid test_negative	True
1	Neutrophils	True
2	Mean platelet volume	True
3	Monocytes	True
5	Red blood Cells	True
6	Platelets	True
7	Eosinophils	True
9	Leukocytes	True
20	Patient age quantile_12	True
10	Mean corpuscular hemoglobin	True
12	Mean corpuscular hemoglobin concentration	True
13	Lymphocytes	True
14	Hemoglobin	True
15	Hematocrit	True
16	Patient age quantile_0	True
17	Patient age quantile_1	True
18	Patient age quantile_10	True
11	Mean corpuscular volume	True
39	Influenza A rapid test_positive	True

In [55]:

```
# Build logistic regression using selected feature using RFE

model_rfe = LogisticRegression(random_state=rs)

# fit it to training data
model_rfe.fit(X_train_sel, y_train)
```

Out[55]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=10, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [58]:

```
# training and test accuracy
print("Train accuracy:", model_rfe.score(X_train_sel, y_train))
print("Test accuracy:", model_rfe.score(X_test_sel, y_test))
```

```
Train accuracy: 0.7200929152148664
Test accuracy: 0.6883468834688347
```

In [59]:

```
# grab feature importances from the model and feature name from the original 
coef = model_rfe.coef_[0]
feature_names = x.columns

# sort them out in descending order
indices = np.argsort(np.absolute(coef))
```

```
indices = np.flip(indices, axis=0)

for i in indices:
    print(feature_names[i], ':', coef[i])
```

```
Platelets : -0.6614702338717633
Influenza B rapid test_positive : -0.644035850738945
Influenza B rapid test_negative : 0.6440358507389434
Hematocrit : -0.6090688306500966
Neutrophils : -0.5001950228143949
Patient age quantile_18 : -0.4688040590280442
Hemoglobin : -0.46228049553138795
Patient age quantile_17 : 0.4219129384737464
Red blood Cells : -0.3705070771971286
Patient age quantile_9 : -0.3672099182426332
Patient age quantile_8 : 0.3672099182426322
Red blood cell distribution width : 0.3535467983205428
Eosinophils : -0.34426974087380124
Mean corpuscular volume : -0.31741014916015176
Patient age quantile_2 : 0.2786543104919637
Patient age quantile_3 : 0.27834748362199696
Mean corpuscular hemoglobin concentration : -0.24477460563957157
Monocytes : -0.22546698829219655
Lymphocytes : -0.20625287747588755
Mean platelet volume : -0.2035696548186417
Patient age quantile_0 : 0.15431463252570007
Mean corpuscular hemoglobin : 0.15263119219044702
Patient age quantile_15 : -0.1436030716624471
Patient age quantile_10 : 0.13926294553586036
Patient age quantile_19 : -0.1180718773699202
Patient age quantile_6 : 0.11492473549697999
Patient age quantile_4 : 0.11445140250435595
Patient age quantile_7 : 0.11144629699926657
Patient age quantile_13 : 0.10078793919580194
Leukocytes : -0.09388028476765857
Patient age quantile_5 : 0.07543899986319438
Patient age quantile_12 : 0.07040080342469252
Basophils : -0.06214791587933037
Patient age quantile_11 : -0.05367116509682197
Patient age quantile_16 : 0.04733878307045071
Patient age quantile_14 : -0.030983253924591714
Proteina C reativa : 0.028823539745784457
Patient age quantile_1 : 0.01966768631650416
```

In [60]:

```
# grid search CV
params = {'C': [pow(10, x) for x in range(-6, 4)]}

rfe_cv = GridSearchCV(param_grid=params, estimator=LogisticRegression(max_ite
rfe_cv.fit(X_train_sel, y_train)

# test the best model
print("Train accuracy:", rfe_cv.score(X_train_sel, y_train))
print("Test accuracy:", rfe_cv.score(X_test_sel, y_test))

y_pred = rfe_cv.predict(X_test_sel)
print(classification_report(y_test, y_pred))

# print parameters of the best model
print(rfe_cv.best_params_)
```

```
Train accuracy: 0.7177700348432056
Test accuracy: 0.6883468834688347
              precision    recall  f1-score   support

           0       0.75      0.64      0.69       202
           1       0.63      0.75      0.68       167
```

```
    accuracy                           0.69       369
   macro avg       0.69      0.69      0.69       369
weighted avg       0.70      0.69      0.69       369

{'C': 1000}
```

In [61]:

```
# grab feature importances from the model and feature name from the original 
coef = rfe_cv.best_estimator_.coef_[0]
feature_names = x.columns

# sort them out in descending order
indices = np.argsort(np.absolute(coef))
indices = np.flip(indices, axis=0)

for i in indices:
    print(feature_names[i], ':', coef[i])
```

```
Red blood cell distribution width : 2.959107730141415
Mean corpuscular hemoglobin concentration : -2.648819619997219
Influenza B rapid test_negative : 2.5654734431194064
Influenza B rapid test_positive : -2.5654734431185537
Leukocytes : 0.7350668899000727
Platelets : -0.7085419981424834
Hematocrit : -0.625348307528325
Mean corpuscular hemoglobin : 0.5708695857872941
Neutrophils : -0.5370894216879268
Basophils : 0.5042288597461991
Hemoglobin : -0.47802689590079916
Patient age quantile_18 : -0.47401403143737353
Patient age quantile_17 : 0.4360407779641427
Patient age quantile_9 : -0.40987522571513574
Patient age quantile_8 : 0.409875225715131
Red blood Cells : -0.3969861882821646
Eosinophils : -0.3709499993339381
Mean corpuscular volume : -0.36046574597349607
Patient age quantile_2 : 0.2821082705134913
Patient age quantile_3 : 0.2819090146828276
Monocytes : -0.25748269803172025
Mean platelet volume : -0.21838387459953523
Patient age quantile_0 : 0.15876080816671487
Patient age quantile_15 : -0.14560666823857332
Patient age quantile_10 : 0.13804184086103913
Patient age quantile_19 : -0.11953321661137735
Patient age quantile_4 : 0.11631534894161387
Patient age quantile_7 : 0.11545284047724332
Patient age quantile_6 : 0.11542767023586811
Patient age quantile_13 : 0.09976882350608603
Patient age quantile_5 : 0.08161384356334257
Lymphocytes : -0.0758697465743787
Patient age quantile_12 : 0.07019274523361912
Patient age quantile_11 : -0.05458619370262486
Patient age quantile_16 : 0.05303375937670795
Patient age quantile_14 : -0.03094689372642316
Proteina C reativa : 0.022292140352526513
Patient age quantile_1 : 0.020013647421699737
```

Comparison and finding the best performing model

In [62]:

```
from sklearn.metrics import roc_auc_score

y_pred_proba_lr = model.predict_proba(x_test_r)
y_pred_proba_lr_cv = cv.predict_proba(x_test_r)
y_pred_proba_lr_rfe = model_rfe.predict_proba(X_test_sel)
```

```
y_pred_proba_rfe_cv = rfe_cv.predict_proba(X_test_sel)


roc_index_lr = roc_auc_score(y_test, y_pred_proba_lr[:, 1])
roc_index_lr_cv = roc_auc_score(y_test, y_pred_proba_lr_cv[:, 1])
roc_index_lr_rfe = roc_auc_score(y_test, y_pred_proba_lr_rfe[:, 1])
roc_index_rfe_cv = roc_auc_score(y_test, y_pred_proba_rfe_cv[:, 1])


print("ROC index on test for `model`:", roc_index_lr)
print("ROC index on test for `cv`:", roc_index_lr_cv)
print("ROC index on test for `model_rfe`:", roc_index_lr_rfe)
print("ROC index on test for `rfe_cv`:", roc_index_rfe_cv)
```

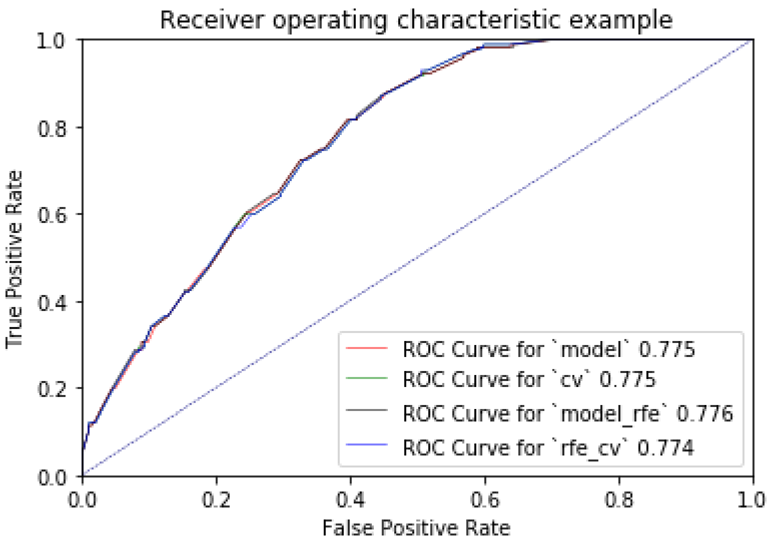
```
ROC index on test for `model`: 0.7747672970889903
ROC index on test for `cv`: 0.7746783660401969
ROC index on test for `model_rfe`: 0.7755380328452007
ROC index on test for `rfe_cv`: 0.7744115728938163
```

In [63]:

```
from sklearn.metrics import roc_curve

fpr_lr, tpr_lr, thresholds_lr = roc_curve(y_test, y_pred_proba_lr[:,1])
fpr_lr_cv, tpr_lr_cv, thresholds_lr_cv = roc_curve(y_test, y_pred_proba_lr_cv
fpr_lr_rfe, tpr_lr_rfe, thresholds_lr_rfe = roc_curve(y_test, y_pred_proba_lr_
fpr_rfe_cv, tpr_rfe_cv, thresholds_rfe_cv = roc_curve(y_test, y_pred_proba_rf
```

In [64]:

```
import matplotlib.pyplot as plt

plt.plot(fpr_lr, tpr_lr, label='ROC Curve for `model` {:.3f}'.format(roc_inde
plt.plot(fpr_lr_cv, tpr_lr_cv, label='ROC Curve for `cv` {:.3f}'.format(roc_i
plt.plot(fpr_lr_rfe, tpr_lr_rfe, label='ROC Curve for `model_rfe` {:.3f}'.for
plt.plot(fpr_rfe_cv, tpr_rfe_cv, label='ROC Curve for `rfe_cv` {:.3f}'.format


plt.plot([0, 1], [0, 1], color='navy', lw=0.5, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```



In [65]:

```
import pickle
```

```
lr_best = model_rfe
roc_index_lr_best = roc_index_lr_rfe
tpr_lr_best = tpr_lr_rfe
fpr_lr_best = fpr_lr_rfe
with open('LR.pickle', 'wb') as f:
    pickle.dump([lr_best,roc_index_lr_best, fpr_lr_best, tpr_lr_best], f)
```

Task 4 Predictive modeling using Neural Networks

In [66]:

```
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
```

1. Build Neural Networks with default setting

In [67]:

```
# Create neural networks model by fitting in the dataset

model_NN_1 = MLPClassifier(random_state = rs)
model_NN_1.fit(x_train, y_train)

print("Train accuracy:", model_NN_1.score(x_train,y_train))
print("Test accuracy:", model_NN_1.score(x_test, y_test))

y_pred = model_NN_1.predict(x_test)
print(classification_report(y_test, y_pred))

print(model_NN_1)
```

```
Train accuracy: 0.7502903600464577
Test accuracy: 0.6856368563685636
              precision    recall  f1-score   support

           0       0.71      0.71      0.71       202
           1       0.65      0.65      0.65       167

    accuracy                           0.69       369
   macro avg       0.68      0.68      0.68       369
weighted avg       0.69      0.69      0.69       369

MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=200,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=10, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
/opt/conda/lib/python3.7/site-packages/sklearn/neural_network/_multilayer_perc
eptron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (2
00) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

In [68]:

```
# As the maximum iterations (200) fails to reach the convergency, we will try

model_NN_2 = MLPClassifier(max_iter = 300, random_state =rs)
model_NN_2.fit(x_train, y_train)
```

```
print("Train accuracy:", model_NN_2.score(x_train,y_train))
print("Test accuracy:", model_NN_2.score(x_test, y_test))

y_pred = model_NN_2.predict(x_test)
print(classification_report(y_test,y_pred))

print(model_NN_2)
```

```
Train accuracy: 0.7502903600464577
Test accuracy: 0.6829268292682927
              precision    recall  f1-score   support

           0       0.71      0.71      0.71       202
           1       0.65      0.65      0.65       167

    accuracy                           0.68       369
   macro avg       0.68      0.68      0.68       369
weighted avg       0.68      0.68      0.68       369

MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=300,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=10, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

Discussion

The neural network after setting the iteration of 700 doesn't show train accuracy increase, and the test accuracy is slightly decreased. This indicates that the model has been fitted well when max_iter = 300.

2. Refine the network by tuning it with GridSearchCV

We will find the two optimal hyperparameters listed below using GridSearchCV:

1. hidden_layer_sizes
2. alpha

In [69]:

```
print("Total features included in the model now:", x_train.shape[1])
```

```
Total features included in the model now: 40
```

- 1.Set range of hidden layer

In [70]:

```
# Set range of hidden layer

params = {'hidden_layer_sizes': [(x,) for x in range(5,41,10)]}

cv_NN_1 = GridSearchCV(param_grid=params, estimator = MLPClassifier(random_st
cv_NN_1.fit(x_train,y_train)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/neural_network/_multilayer_perc
eptron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (2
00) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

Out[70]:

```
GridSearchCV(cv=10, error_score=nan,
             estimator=MLPClassifier(activation='relu', alpha=0.0001,
```

```
                                     batch_size='auto', beta_1=0.9,
                                     beta_2=0.999, early_stopping=False,
                                     epsilon=1e-08, hidden_layer_sizes=(100,),
                                     learning_rate='constant',
                                     learning_rate_init=0.001, max_fun=15000,
                                     max_iter=200, momentum=0.9,
                                     n_iter_no_change=10,
                                     nesterovs_momentum=True, power_t=0.5,
                                     random_state=10, shuffle=True,
                                     solver='adam', tol=0.0001,
                                     validation_fraction=0.1, verbose=False,
                                     warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'hidden_layer_sizes': [(5,), (15,), (25,), (35,)]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [71]:

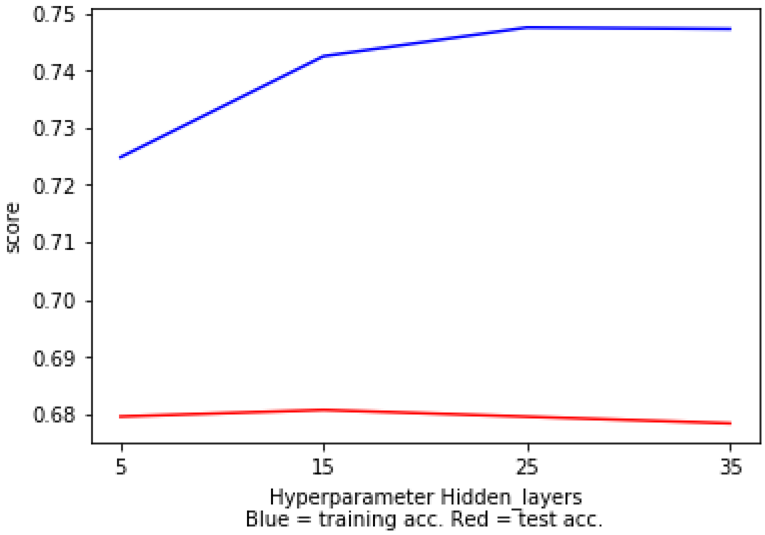
```
# Have a look at model accuracy and the best hyperparamiters

print("Train accuracy:", cv_NN_1.score(x_train,y_train))
print("Test accuracy:", cv_NN_1.score(x_test, y_test))

y_pred = cv_NN_1.predict(x_test)
print(classification_report(y_test, y_pred))

print(cv_NN_1.best_params_)
```

```
Train accuracy: 0.7409988385598142
Test accuracy: 0.6937669376693767
              precision    recall  f1-score   support

           0       0.75      0.67      0.70       202
           1       0.64      0.72      0.68       167

    accuracy                           0.69       369
   macro avg       0.69      0.70      0.69       369
weighted avg       0.70      0.69      0.69       369

{'hidden_layer_sizes': (15,)}
```

In [72]:

```
result_1 = cv_NN_1.cv_results_

train_result = result_1['mean_train_score']
test_result = result_1['mean_test_score']
print("Total number of models: ", len(test_result))
# plot hidden layers hyperparameter values vs training and test accuracy scor
plt.plot(range(0, len(train_result)), train_result, 'b', range(0,len(test_res
plt.xlabel('Hyperparameter Hidden_layers\nBlue = training acc. Red = test acc
plt.xticks(range(0, len(train_result)),range(5,41,10))
plt.ylabel('score')
plt.show()
```

```
Total number of models:  4
```



In [73]:

```
# We try the sizes around 15 for hidden_layer_sizes to

params = {'hidden_layer_sizes': [(x,) for x in range(13,20)]}

cv_NN_2 = GridSearchCV(param_grid=params, estimator = MLPClassifier(random_st
cv_NN_2.fit(x_train,y_train)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/neural_network/_multilayer_perc
eptron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (2
00) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

Out[73]:

```
GridSearchCV(cv=10, error_score=nan,
             estimator=MLPClassifier(activation='relu', alpha=0.0001,
                                     batch_size='auto', beta_1=0.9,
                                     beta_2=0.999, early_stopping=False,
                                     epsilon=1e-08, hidden_layer_sizes=(100,),
                                     learning_rate='constant',
                                     learning_rate_init=0.001, max_fun=15000,
                                     max_iter=200, momentum=0.9,
                                     n_iter_no_change=10,
                                     nesterovs_momentum=True, power_t=0.5,
                                     random_state=10, shuffle=True,
                                     solver='adam', tol=0.0001,
                                     validation_fraction=0.1, verbose=False,
                                     warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'hidden_layer_sizes': [(13,), (14,), (15,), (16,),
                                                (17,), (18,), (19,)]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [74]:

```
# Have a look at model accuracy and the best hyperparamiters

print("Train accuracy:", cv_NN_2.score(x_train,y_train))
print("Test accuracy:", cv_NN_2.score(x_test, y_test))

y_pred = cv_NN_2.predict(x_test)
print(classification_report(y_test, y_pred))

print(cv_NN_2.best_params_)
```

```
Train accuracy: 0.7444831591173054
Test accuracy: 0.6937669376693767
              precision    recall  f1-score   support

           0       0.75      0.67      0.70       202
```

```
           1       0.64      0.72      0.68       167

    accuracy                           0.69       369
   macro avg       0.69      0.70      0.69       369
weighted avg       0.70      0.69      0.69       369

{'hidden_layer_sizes': (16,)}
```

- 1. Set hidden_layer_sizes and alpha together

In [75]:
```
params = {'hidden_layer_sizes': [(x,) for x in range(13,20)], 'alpha': [0.01,

cv_NN_3 = GridSearchCV(param_grid=params, estimator = MLPClassifier(random_st
cv_NN_3.fit(x_train,y_train)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/neural_network/_multilayer_perc
eptron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (2
00) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

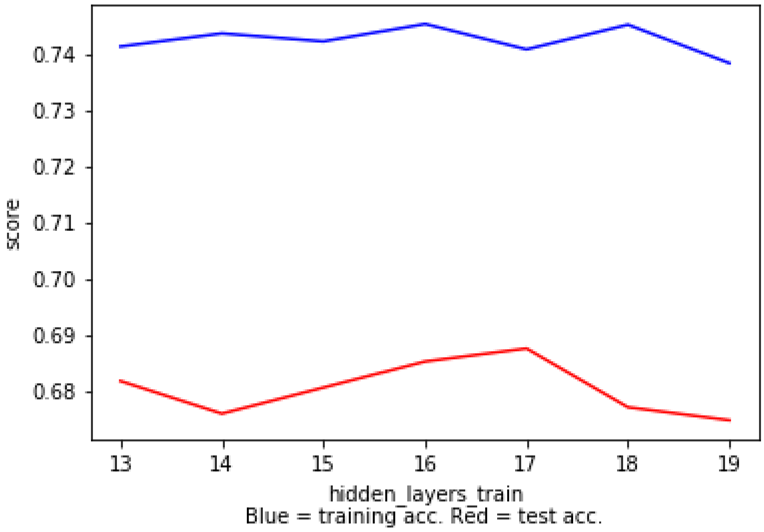
Out[75]:
```
GridSearchCV(cv=10, error_score=nan,
             estimator=MLPClassifier(activation='relu', alpha=0.0001,
                                     batch_size='auto', beta_1=0.9,
                                     beta_2=0.999, early_stopping=False,
                                     epsilon=1e-08, hidden_layer_sizes=(100,),
                                     learning_rate='constant',
                                     learning_rate_init=0.001, max_fun=15000,
                                     max_iter=200, momentum=0.9,
                                     n_iter_no_change=10,
                                     nesterovs_momentum=True, power_t=0.5,
                                     random_state=10, shuffle=True,
                                     solver='adam', tol=0.0001,
                                     validation_fraction=0.1, verbose=False,
                                     warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'alpha': [0.01, 0.001, 0.0001, 1e-05],
                         'hidden_layer_sizes': [(13,), (14,), (15,), (16,),
                                                (17,), (18,), (19,)]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [76]:
```
# Have a look at model accuracy and the best hyperparamiters

print("Train accuracy:", cv_NN_3.score(x_train,y_train))
print("Test accuracy:", cv_NN_3.score(x_test, y_test))

y_pred = cv_NN_3.predict(x_test)
print(classification_report(y_test, y_pred))

print(cv_NN_3.best_params_)
```

```
Train accuracy: 0.7363530778164924
Test accuracy: 0.6991869918699187
              precision    recall  f1-score   support

           0       0.75      0.68      0.71       202
           1       0.65      0.72      0.68       167

    accuracy                           0.70       369
   macro avg       0.70      0.70      0.70       369
weighted avg       0.70      0.70      0.70       369

{'alpha': 0.01, 'hidden_layer_sizes': (17,)}
```

In [77]:
```
# Have a look mean_train_score and mean_test_score
result_2 = cv_NN_3.cv_results_
```

```
ff = pd.DataFrame(result_2['params'])

train_result = result_2['mean_train_score']
test_result = result_2['mean_test_score']

index_ = list(ff.index[(ff['alpha']== 0.01)])
hidden_layers_train = []
hidden_layers_test = []
for i in range(0,len(index_)):
    hidden_layers_train.append(train_result[index_[i]])
    hidden_layers_test.append(test_result[index_[i]])
plt.plot(range(1, len(hidden_layers_train)+1), hidden_layers_train, 'b', rang
plt.xlabel('hidden_layers_train\nBlue = training acc. Red = test acc.')
plt.ylabel('score')
plt.xticks(np.arange(1, len(hidden_layers_train)+1, 1), range(13,20))
plt.show()
```



Conclusion

The GridSearch returned a hidden layer of 17 neurons and alpha value of 0.01, which shows no better test accuray of cv_NN_3 model compared to cv_NN_2 model.

3. Build model based on the best model of DT and tune with GridSearchCV

In [78]:

```
with open('DT.pickle', 'rb') as f:
    dt_best, roc_index_dt_cv, fpr_dt_cv, tpr_dt_cv = pickle.load(f)

print(dt_best.best_params_)
```

```
{'criterion': 'gini', 'max_depth': 18, 'min_samples_leaf': 15}
```

In [79]:

```
importances = dt_best.best_estimator_.feature_importances_
feature_names = x.columns
# sort them out in descending order
indices = np.argsort(importances)
indices = np.flip(indices, axis=0)

for i in indices:
   print(feature_names[i], ':', importances[i])
```

```
Patient age quantile_1 : 0.1570030356594264
Influenza B rapid test_negative : 0.12242876717472596
Influenza A rapid test_negative : 0.1223779285913228
Eosinophils : 0.10892456465989904
Patient age quantile_2 : 0.08663119955059644
Leukocytes : 0.08590959598391089
Patient age quantile_0 : 0.0828331711635059
Patient age quantile_17 : 0.05254993773407766
Patient age quantile_5 : 0.029179752235000394
Patient age quantile_19 : 0.028757683744032734
Patient age quantile_4 : 0.027114513316328616
Patient age quantile_13 : 0.0192539114324268
Patient age quantile_3 : 0.01680834307569263
Red blood Cells : 0.012782006037245618
Patient age quantile_18 : 0.01171680381973257
Patient age quantile_14 : 0.011679333188114996
Patient age quantile_11 : 0.009979014634518463
Patient age quantile_16 : 0.008323475485032551
Hemoglobin : 0.005746962514409674
Platelets : 0.0
Mean corpuscular volume : 0.0
Red blood cell distribution width : 0.0
Basophils : 0.0
Monocytes : 0.0
Mean platelet volume : 0.0
Neutrophils : 0.0
Mean corpuscular hemoglobin : 0.0
Influenza A rapid test_positive : 0.0
Mean corpuscular hemoglobin concentration : 0.0
Lymphocytes : 0.0
Hematocrit : 0.0
Patient age quantile_10 : 0.0
Patient age quantile_12 : 0.0
Patient age quantile_15 : 0.0
Patient age quantile_6 : 0.0
Patient age quantile_7 : 0.0
Patient age quantile_8 : 0.0
Patient age quantile_9 : 0.0
Influenza B rapid test_positive : 0.0
Proteina C reativa : 0.0
```

In [81]:

```
from sklearn.feature_selection import SelectFromModel

dt = SelectFromModel(dt_best.best_estimator_, prefit=True)
x_train_sel_NN = dt.transform(x_train)
x_test_sel_NN = dt.transform(x_test)

print(x_train_sel_NN.shape)
```

```
(861, 11)
```

In [82]:

```
# Build a Neural Network model with selected feature in DT best model.

model_NN_dt = MLPClassifier(random_state=rs)
model_NN_dt.fit(x_train_sel_NN, y_train)

print("Train accuracy:", model_NN_dt.score(x_train_sel_NN, y_train))
print("Test accuracy:", model_NN_dt.score(x_test_sel_NN, y_test))

y_pred = model_NN_dt.predict(x_test_sel_NN)
print(classification_report(y_test, y_pred))

print(model_NN_dt)
```

```
Train accuracy: 0.686411149825784
Test accuracy: 0.6802168021680217
```

```
              precision    recall  f1-score   support

           0       0.92      0.46      0.61       202
           1       0.59      0.95      0.73       167

    accuracy                           0.68       369
   macro avg       0.76      0.70      0.67       369
weighted avg       0.77      0.68      0.66       369

MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=200,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=10, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
/opt/conda/lib/python3.7/site-packages/sklearn/neural_network/_multilayer_perc
eptron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (2
00) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

In [83]:

```python
# Tune the model within gridSearchCV

params = {'hidden_layer_sizes': [(x,) for x in range(13,20)], 'alpha': [0.01,

model_NN_dt_cv = GridSearchCV(param_grid=params, estimator=MLPClassifier(max_
model_NN_dt_cv.fit(x_train_sel_NN, y_train)

print("Train accuracy:", model_NN_dt_cv.score(x_train_sel_NN, y_train))
print("Test accuracy:", model_NN_dt_cv.score(x_test_sel_NN, y_test))

y_pred = model_NN_dt_cv.predict(x_test_sel_NN)
print(classification_report(y_test, y_pred))

print(model_NN_dt_cv.best_params_)

print(model_NN_dt_cv)
```

```
Train accuracy: 0.6829268292682927
Test accuracy: 0.6802168021680217
              precision    recall  f1-score   support

           0       0.91      0.46      0.61       202
           1       0.59      0.95      0.73       167

    accuracy                           0.68       369
   macro avg       0.75      0.70      0.67       369
weighted avg       0.77      0.68      0.66       369

{'alpha': 0.01, 'hidden_layer_sizes': (17,)}
GridSearchCV(cv=10, error_score=nan,
             estimator=MLPClassifier(activation='relu', alpha=0.0001,
                                     batch_size='auto', beta_1=0.9,
                                     beta_2=0.999, early_stopping=False,
                                     epsilon=1e-08, hidden_layer_sizes=(100,),
                                     learning_rate='constant',
                                     learning_rate_init=0.001, max_fun=15000,
                                     max_iter=300, momentum=0.9,
                                     n_iter_no_change=10,
                                     nesterovs_momentum=True, power_t=0.5,
                                     random_state=10, shuffle=True,
                                     solver='adam', tol=0.0001,
                                     validation_fraction=0.1, verbose=False,
                                     warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'alpha': [0.01, 0.001, 0.0001, 1e-05],
```

```
                          'hidden_layer_sizes': [(13,), (14,), (15,), (16,),
                                                 (17,), (18,), (19,)]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/neural_network/_multilayer_perc
eptron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (3
00) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

In [84]:

```
result = model_NN_dt_cv.cv_results_
print(result)
```

```
{'mean_fit_time': array([0.61324914, 0.61918793, 0.65913343, 0.61996474, 0.636
19802,
       0.65792718, 0.65719886, 0.64656019, 0.64657795, 0.6361186 ,
       0.66964009, 0.66750636, 0.66328545, 0.67971849, 0.62723029,
       0.63210382, 0.72103479, 0.55256078, 0.61133943, 0.53081644,
       0.57010484, 0.62347307, 0.62977822, 0.62116704, 0.62736874,
       0.63831129, 0.64963059, 0.69190731]), 'std_fit_time': array([0.0395081
2, 0.10621519, 0.07038347, 0.02895592, 0.04879356,
       0.04200627, 0.02847453, 0.05380979, 0.04623752, 0.08875862,
       0.05448734, 0.0400822 , 0.06006486, 0.04264818, 0.01347596,
       0.02554043, 0.07216795, 0.06966528, 0.08290567, 0.07357171,
       0.09019988, 0.04367252, 0.04055431, 0.04433149, 0.03230526,
       0.02982427, 0.03246825, 0.08903765]), 'mean_score_time': array([0.00084
851, 0.00084734, 0.00084131, 0.00082324, 0.00095375,
       0.00093162, 0.0008008 , 0.00087121, 0.00080235, 0.00079904,
       0.00086627, 0.00092616, 0.00083783, 0.00092168, 0.00090683,
       0.0008651 , 0.00094292, 0.00075524, 0.00082719, 0.00070994,
       0.00084651, 0.00082438, 0.0008004 , 0.00085065, 0.00087233,
       0.00080578, 0.00088012, 0.00086715]), 'std_score_time': array([1.315921
73e-04, 1.46680644e-04, 1.61846991e-04, 1.35475741e-04,
       1.08503457e-04, 1.13816058e-04, 1.27999863e-04, 1.42794874e-04,
       1.10932353e-04, 1.74829507e-04, 1.22956518e-04, 1.55984039e-04,
       1.21912465e-04, 1.29512422e-04, 9.93266365e-05, 1.24013980e-04,
       8.25133911e-05, 1.35227397e-04, 1.25484989e-04, 1.22352320e-04,
       1.27163863e-04, 1.42215955e-04, 1.08400276e-04, 1.26940081e-04,
       8.24835805e-05, 8.06509865e-05, 1.32919128e-04, 1.08271358e-04]), 'para
m_alpha': masked_array(data=[0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.001,
0.001,
                   0.001, 0.001, 0.001, 0.001, 0.001, 0.0001, 0.0001,
                   0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 1e-05, 1e-05,
                   1e-05, 1e-05, 1e-05, 1e-05, 1e-05],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False],
       fill_value='?',
            dtype=object), 'param_hidden_layer_sizes': masked_array(data=[(1
3,), (14,), (15,), (16,), (17,), (18,), (19,), (13,),
                   (14,), (15,), (16,), (17,), (18,), (19,), (13,), (14,),
                   (15,), (16,), (17,), (18,), (19,), (13,), (14,), (15,),
                   (16,), (17,), (18,), (19,)],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'alpha': 0.01, 'hidden_layer_sizes': (1
3,)}, {'alpha': 0.01, 'hidden_layer_sizes': (14,)}, {'alpha': 0.01, 'hidden_la
yer_sizes': (15,)}, {'alpha': 0.01, 'hidden_layer_sizes': (16,)}, {'alpha': 0.
01, 'hidden_layer_sizes': (17,)}, {'alpha': 0.01, 'hidden_layer_sizes': (1
8,)}, {'alpha': 0.01, 'hidden_layer_sizes': (19,)}, {'alpha': 0.001, 'hidden_l
ayer_sizes': (13,)}, {'alpha': 0.001, 'hidden_layer_sizes': (14,)}, {'alpha':
0.001, 'hidden_layer_sizes': (15,)}, {'alpha': 0.001, 'hidden_layer_sizes': (1
6,)}, {'alpha': 0.001, 'hidden_layer_sizes': (17,)}, {'alpha': 0.001, 'hidden_
layer_sizes': (18,)}, {'alpha': 0.001, 'hidden_layer_sizes': (19,)}, {'alpha':
0.0001, 'hidden_layer_sizes': (13,)}, {'alpha': 0.0001, 'hidden_layer_sizes':
```

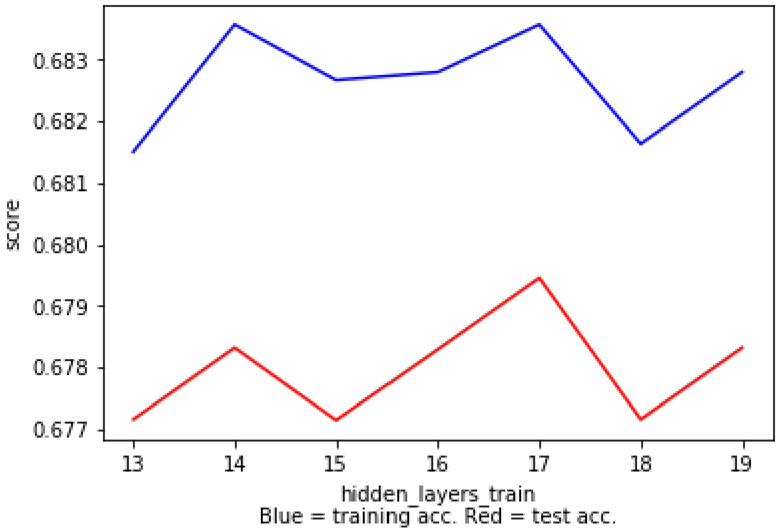
```
(14,)}, {'alpha': 0.0001, 'hidden_layer_sizes': (15,)}, {'alpha': 0.0001, 'hid
den_layer_sizes': (16,)}, {'alpha': 0.0001, 'hidden_layer_sizes': (17,)}, {'al
pha': 0.0001, 'hidden_layer_sizes': (18,)}, {'alpha': 0.0001, 'hidden_layer_si
zes': (19,)}, {'alpha': 1e-05, 'hidden_layer_sizes': (13,)}, {'alpha': 1e-05,
'hidden_layer_sizes': (14,)}, {'alpha': 1e-05, 'hidden_layer_sizes': (15,)},
{'alpha': 1e-05, 'hidden_layer_sizes': (16,)}, {'alpha': 1e-05, 'hidden_layer_
sizes': (17,)}, {'alpha': 1e-05, 'hidden_layer_sizes': (18,)}, {'alpha': 1e-0
5, 'hidden_layer_sizes': (19,)}], 'split0_test_score': array([0.64367816, 0.64
367816, 0.65517241, 0.66666667, 0.66666667,
       0.64367816, 0.64367816, 0.64367816, 0.64367816, 0.65517241,
       0.65517241, 0.66666667, 0.64367816, 0.64367816, 0.64367816,
       0.64367816, 0.65517241, 0.65517241, 0.66666667, 0.64367816,
       0.64367816, 0.64367816, 0.64367816, 0.65517241, 0.65517241,
       0.66666667, 0.64367816, 0.64367816]), 'split1_test_score': array([0.639
53488, 0.63953488, 0.63953488, 0.62790698, 0.63953488,
       0.63953488, 0.63953488, 0.63953488, 0.63953488, 0.63953488,
       0.62790698, 0.63953488, 0.63953488, 0.63953488, 0.63953488,
       0.63953488, 0.63953488, 0.62790698, 0.63953488, 0.63953488,
       0.63953488, 0.63953488, 0.63953488, 0.63953488, 0.62790698,
       0.63953488, 0.63953488, 0.63953488]), 'split2_test_score': array([0.697
67442, 0.6744186 , 0.68604651, 0.68604651, 0.68604651,
       0.68604651, 0.6744186 , 0.69767442, 0.6744186 , 0.68604651,
       0.68604651, 0.68604651, 0.68604651, 0.6744186 , 0.69767442,
       0.6744186 , 0.68604651, 0.68604651, 0.68604651, 0.68604651,
       0.6744186 , 0.69767442, 0.6744186 , 0.68604651, 0.68604651,
       0.68604651, 0.68604651, 0.6744186 ]), 'split3_test_score': array([0.709
30233, 0.72093023, 0.70930233, 0.70930233, 0.70930233,
       0.70930233, 0.72093023, 0.70930233, 0.72093023, 0.70930233,
       0.70930233, 0.70930233, 0.70930233, 0.70930233, 0.70930233,
       0.72093023, 0.70930233, 0.70930233, 0.70930233, 0.70930233,
       0.70930233, 0.70930233, 0.72093023, 0.70930233, 0.70930233,
       0.70930233, 0.70930233, 0.70930233]), 'split4_test_score': array([0.732
55814, 0.73255814, 0.73255814, 0.73255814, 0.73255814,
       0.73255814, 0.73255814, 0.73255814, 0.73255814, 0.73255814,
       0.73255814, 0.73255814, 0.72093023, 0.73255814, 0.73255814,
       0.73255814, 0.73255814, 0.73255814, 0.73255814, 0.73255814,
       0.73255814, 0.73255814, 0.73255814, 0.73255814, 0.73255814,
       0.73255814, 0.73255814, 0.73255814]), 'split5_test_score': array([0.639
53488, 0.6627907 , 0.65116279, 0.65116279, 0.65116279,
       0.6627907 , 0.65116279, 0.63953488, 0.6627907 , 0.65116279,
       0.6627907 , 0.65116279, 0.6627907 , 0.65116279, 0.63953488,
       0.6627907 , 0.65116279, 0.6627907 , 0.65116279, 0.6627907 ,
       0.65116279, 0.63953488, 0.6627907 , 0.65116279, 0.6627907 ,
       0.65116279, 0.6627907 , 0.65116279]), 'split6_test_score': array([0.697
67442, 0.68604651, 0.69767442, 0.68604651, 0.69767442,
       0.68604651, 0.68604651, 0.69767442, 0.68604651, 0.69767442,
       0.68604651, 0.69767442, 0.68604651, 0.68604651, 0.69767442,
       0.68604651, 0.69767442, 0.68604651, 0.69767442, 0.6744186 ,
       0.68604651, 0.69767442, 0.68604651, 0.69767442, 0.6744186 ,
       0.69767442, 0.68604651, 0.68604651]), 'split7_test_score': array([0.639
53488, 0.63953488, 0.63953488, 0.63953488, 0.63953488,
       0.63953488, 0.63953488, 0.63953488, 0.63953488, 0.63953488,
       0.63953488, 0.63953488, 0.63953488, 0.63953488, 0.63953488,
       0.63953488, 0.63953488, 0.63953488, 0.63953488, 0.63953488,
       0.63953488, 0.63953488, 0.63953488, 0.63953488, 0.63953488,
       0.63953488, 0.63953488, 0.63953488]), 'split8_test_score': array([0.651
16279, 0.6627907 , 0.65116279, 0.6627907 , 0.65116279,
       0.6627907 , 0.6627907 , 0.65116279, 0.6627907 , 0.65116279,
       0.6627907 , 0.65116279, 0.6627907 , 0.6627907 , 0.65116279,
       0.6627907 , 0.65116279, 0.6627907 , 0.65116279, 0.6627907 ,
       0.6627907 , 0.65116279, 0.6627907 , 0.65116279, 0.6627907 ,
       0.65116279, 0.6627907 , 0.6627907 ]), 'split9_test_score': array([0.720
93023, 0.72093023, 0.70930233, 0.72093023, 0.72093023,
       0.70930233, 0.73255814, 0.72093023, 0.72093023, 0.70930233,
       0.72093023, 0.72093023, 0.70930233, 0.73255814, 0.72093023,
       0.72093023, 0.70930233, 0.72093023, 0.72093023, 0.70930233,
       0.73255814, 0.72093023, 0.72093023, 0.70930233, 0.72093023,
       0.72093023, 0.70930233, 0.73255814]), 'mean_test_score': array([0.67715
851, 0.6783213 , 0.67714515, 0.67829457, 0.67945736,
```

```
       0.67715851, 0.6783213 , 0.67715851, 0.6783213 , 0.67714515,
       0.67830794, 0.67945736, 0.67599572, 0.67715851, 0.67715851,
       0.6783213 , 0.67714515, 0.67830794, 0.67945736, 0.67599572,
       0.67715851, 0.67715851, 0.6783213 , 0.67714515, 0.67714515,
       0.67945736, 0.67715851, 0.67715851]), 'std_test_score': array([0.035921
35, 0.03367329, 0.03209592, 0.03316163, 0.03279949,
       0.03107803, 0.03600195, 0.03592135, 0.03367329, 0.03209592,
       0.03301605, 0.03279949, 0.02914063, 0.03477382, 0.03592135,
       0.03367329, 0.03209592, 0.03301605, 0.03279949, 0.03094096,
       0.03477382, 0.03592135, 0.03367329, 0.03209592, 0.03292767,
       0.03279949, 0.03107803, 0.03477382]), 'rank_test_score': array([13,  5,
22, 12,  1, 13,  5, 13,  5, 22, 10,  1, 27, 13, 13,  5, 22,
       10,  1, 27, 13, 13,  5, 22, 26,  1, 13, 13], dtype=int32), 'split0_trai
n_score': array([0.68475452, 0.68992248, 0.68217054, 0.68604651, 0.68604651,
       0.68604651, 0.6873385 , 0.68475452, 0.68992248, 0.68217054,
       0.68346253, 0.68604651, 0.68604651, 0.6873385 , 0.68475452,
       0.68992248, 0.68346253, 0.68475452, 0.68604651, 0.68604651,
       0.6873385 , 0.68475452, 0.68992248, 0.68217054, 0.68475452,
       0.68604651, 0.68604651, 0.6873385 ]), 'split1_train_score': array([0.69
032258, 0.6916129 , 0.69032258, 0.68774194, 0.68774194,
       0.68645161, 0.68903226, 0.69032258, 0.6916129 , 0.68903226,
       0.68903226, 0.68774194, 0.68645161, 0.68774194, 0.69032258,
       0.6916129 , 0.68903226, 0.69032258, 0.68774194, 0.68645161,
       0.69032258, 0.69032258, 0.6916129 , 0.68774194, 0.68903226,
       0.68645161, 0.68645161, 0.68774194]), 'split2_train_score': array([0.67
870968, 0.68645161, 0.68387097, 0.68258065, 0.68387097,
       0.68258065, 0.68516129, 0.67741935, 0.68645161, 0.68387097,
       0.68258065, 0.68258065, 0.68258065, 0.68516129, 0.67741935,
       0.68645161, 0.68516129, 0.68129032, 0.68258065, 0.68258065,
       0.68645161, 0.68      , 0.68645161, 0.68387097, 0.68129032,
       0.68258065, 0.68258065, 0.68774194]), 'split3_train_score': array([0.67
870968, 0.67741935, 0.67741935, 0.68      , 0.68      ,
       0.67612903, 0.67741935, 0.67741935, 0.67741935, 0.67741935,
       0.68129032, 0.68      , 0.67483871, 0.67741935, 0.67612903,
       0.67870968, 0.67741935, 0.68      , 0.68      , 0.67741935,
       0.67741935, 0.67741935, 0.67870968, 0.67741935, 0.68      ,
       0.68      , 0.67741935, 0.67741935]), 'split4_train_score': array([0.67
354839, 0.68      , 0.67870968, 0.68      , 0.67870968,
       0.67741935, 0.67870968, 0.67354839, 0.68      , 0.67741935,
       0.68      , 0.67870968, 0.67741935, 0.67741935, 0.67483871,
       0.68      , 0.67741935, 0.68      , 0.67741935, 0.67741935,
       0.68      , 0.67483871, 0.68      , 0.67741935, 0.67870968,
       0.67741935, 0.67612903, 0.67870968]), 'split5_train_score': array([0.68
387097, 0.68129032, 0.68387097, 0.68645161, 0.68516129,
       0.68129032, 0.68129032, 0.68387097, 0.68129032, 0.68516129,
       0.68645161, 0.68516129, 0.68129032, 0.68129032, 0.68387097,
       0.68129032, 0.68387097, 0.68645161, 0.68516129, 0.68129032,
       0.68129032, 0.68387097, 0.68129032, 0.68387097, 0.68645161,
       0.68516129, 0.68      , 0.68258065]), 'split6_train_score': array([0.68
, 0.68      , 0.67870968, 0.67870968, 0.68129032,
       0.68      , 0.68258065, 0.67870968, 0.68129032, 0.67870968,
       0.67741935, 0.68129032, 0.68      , 0.68258065, 0.67870968,
       0.68      , 0.68      , 0.67741935, 0.68129032, 0.67741935,
       0.68258065, 0.67870968, 0.68      , 0.67870968, 0.67483871,
       0.68129032, 0.67612903, 0.68258065]), 'split7_train_score': array([0.68
516129, 0.68516129, 0.68774194, 0.68516129, 0.68774194,
       0.68516129, 0.68645161, 0.68516129, 0.68516129, 0.68774194,
       0.68387097, 0.68774194, 0.68387097, 0.68645161, 0.68516129,
       0.68516129, 0.68774194, 0.68516129, 0.68774194, 0.68516129,
       0.68774194, 0.68387097, 0.68516129, 0.68774194, 0.68387097,
       0.68774194, 0.68645161, 0.68774194]), 'split8_train_score': array([0.68
387097, 0.68645161, 0.68516129, 0.68645161, 0.68645161,
       0.68387097, 0.68387097, 0.68387097, 0.68516129, 0.68516129,
       0.68645161, 0.68645161, 0.68387097, 0.68516129, 0.68387097,
       0.68516129, 0.68516129, 0.68645161, 0.68516129, 0.68387097,
       0.68387097, 0.68387097, 0.68645161, 0.68516129, 0.68387097,
       0.68645161, 0.68387097, 0.68387097]), 'split9_train_score': array([0.67
612903, 0.67741935, 0.67870968, 0.67483871, 0.67870968,
       0.67741935, 0.67612903, 0.67612903, 0.67870968, 0.67870968,
```

```
        0.67483871, 0.67870968, 0.67741935, 0.67741935, 0.67612903,
        0.67870968, 0.67870968, 0.67483871, 0.67870968, 0.67741935,
        0.67741935, 0.67612903, 0.67870968, 0.67870968, 0.67483871,
        0.67870968, 0.67741935, 0.67741935]), 'mean_train_score': array([0.6815
0771, 0.68357289, 0.68266867, 0.6827982 , 0.68357239,
        0.68163691, 0.68279837, 0.68112061, 0.68370193, 0.68253963,
        0.6825398 , 0.68344336, 0.68137884, 0.68279837, 0.68112061,
        0.68370193, 0.68279787, 0.682669  , 0.6831853 , 0.68150788,
        0.68344353, 0.68137868, 0.68383096, 0.68228157, 0.68176577,
        0.6831853 , 0.68124981, 0.6833145 ]), 'std_train_score': array([0.00472
202, 0.00480135, 0.00410182, 0.00403792, 0.00342136,
        0.00360057, 0.00414737, 0.00494776, 0.00450365, 0.00408221,
        0.00410998, 0.00343198, 0.00370706, 0.00398356, 0.00487999,
        0.00442909, 0.00398088, 0.00450305, 0.00353414, 0.00363364,
        0.00424287, 0.0045283 , 0.00448813, 0.00381822, 0.00448561,
        0.00348671, 0.00411241, 0.00408773])}
```

In [85]:

```python
result_3 = model_NN_dt_cv.cv_results_

ff = pd.DataFrame(result_3['params'])

train_result = result_3['mean_train_score']
test_result = result_3['mean_test_score']

index_ = list(ff.index[(ff['alpha']== 0.01)])
hidden_layers_train = []
hidden_layers_test = []
for i in range(0,len(index_)):
    hidden_layers_train.append(train_result[index_[i]])
    hidden_layers_test.append(test_result[index_[i]])
plt.plot(range(1, len(hidden_layers_train)+1), hidden_layers_train, 'b', rang
plt.xlabel('hidden_layers_train\nBlue = training acc. Red = test acc.')
plt.ylabel('score')
plt.xticks(np.arange(1, len(hidden_layers_train)+1, 1), range(13,20))
plt.show()
```



4. Comparison and finding the best performing model

A total of

- 1. Default neural network: model_NN_1
- 1. Neural network with max_inter = 300: model_NN_2
- 1. Neural network with grid search: cv_NN_1
- 1. Neural network with grid search: cv_NN_2

- 1. Neural network with grid search: cv_NN_3
- 1. Neural network with feature selection using DT: model_NN_dt
- 1. Neural network with feature selection using DT with grid search: model_NN_dt_cv

In [86]:

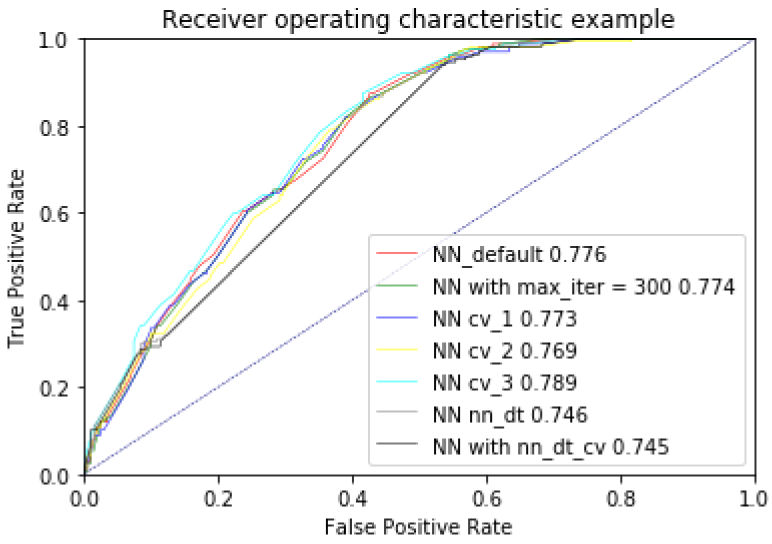
```
from sklearn.metrics import roc_auc_score

y_pred_proba_nn_1 = model_NN_1.predict_proba(x_test)
y_pred_proba_nn_2 = model_NN_2.predict_proba(x_test)
y_pred_proba_cv_1 = cv_NN_1.predict_proba(x_test)
y_pred_proba_cv_2 = cv_NN_2.predict_proba(x_test)
y_pred_proba_cv_3 = cv_NN_3.predict_proba(x_test)
y_pred_proba_nn_dt = model_NN_dt.predict_proba(x_test_sel_NN)
y_pred_proba_nn_dt_cv = model_NN_dt_cv.predict_proba(x_test_sel_NN)


roc_index_nn_1 = roc_auc_score(y_test, y_pred_proba_nn_1[:, 1])
roc_index_nn_2 = roc_auc_score(y_test, y_pred_proba_nn_2[:, 1])
roc_index_cv_1 = roc_auc_score(y_test, y_pred_proba_cv_1[:, 1])
roc_index_cv_2 = roc_auc_score(y_test, y_pred_proba_cv_2[:, 1])
roc_index_cv_3 = roc_auc_score(y_test, y_pred_proba_cv_3[:, 1])
roc_index_nn_dt = roc_auc_score(y_test, y_pred_proba_nn_dt[:, 1])
roc_index_nn_dt_cv = roc_auc_score(y_test, y_pred_proba_nn_dt_cv[:, 1])


print("ROC index on test for NN_default:", roc_index_nn_1)
print("ROC index on test for NN with with max_inter = 300:", roc_index_nn_2)
print("ROC index on test for NN with gridsearch 1:", roc_index_cv_1)
print("ROC index on test for NN with gridsearch 2:", roc_index_cv_2)
print("ROC index on test for NN with gridsearch 3:", roc_index_cv_3)
print("ROC index on test for NN with feature selection using DT:", roc_index_
print("ROC index on test for NN with feature selection using DT with grid sea

from sklearn.metrics import roc_curve

fpr_nn_1, tpr_nn_1, thresholds_nn_1 = roc_curve(y_test, y_pred_proba_nn_1[:,1
fpr_nn_2, tpr_nn_2, thresholds_nn_2 = roc_curve(y_test, y_pred_proba_nn_2[:,1
fpr_cv_1, tpr_cv_1, thresholds_cv_1 = roc_curve(y_test, y_pred_proba_cv_1[:,1
fpr_cv_2, tpr_cv_2, thresholds_cv_2 = roc_curve(y_test, y_pred_proba_cv_2[:,1
fpr_cv_3, tpr_cv_3, thresholds_cv_3 = roc_curve(y_test, y_pred_proba_cv_3[:,1
fpr_nn_dt, tpr_nn_dt, thresholds_nn_dt = roc_curve(y_test, y_pred_proba_nn_dt
fpr_nn_dt_cv, tpr_nn_dt_cv, thresholds_nn_dt_cv = roc_curve(y_test, y_pred_pr
```

```
ROC index on test for NN_default: 0.7763680559672734
ROC index on test for NN with with max_inter = 300: 0.7738186992351931
ROC index on test for NN with gridsearch 1: 0.7730183197960514
ROC index on test for NN with gridsearch 2: 0.7693425031125868
ROC index on test for NN with gridsearch 3: 0.7893519890911247
ROC index on test for NN with feature selection using DT: 0.7456423786091184
ROC index on test for NN with feature selection using DT with grid search: 0.7
452866544139444
```

In [87]:

```
import matplotlib.pyplot as plt

plt.plot(fpr_nn_1, tpr_nn_1, label='NN_default {:.3f}'.format(roc_index_nn_1)
plt.plot(fpr_nn_2, tpr_nn_2, label='NN with max_iter = 300 {:.3f}'.format(roc_
plt.plot(fpr_cv_1, tpr_cv_1, label='NN cv_1 {:.3f}'.format(roc_index_cv_1), c
plt.plot(fpr_cv_2, tpr_cv_2, label='NN cv_2 {:.3f}'.format(roc_index_cv_2), c
plt.plot(fpr_cv_3, tpr_cv_3, label='NN cv_3 {:.3f}'.format(roc_index_cv_3), c
plt.plot(fpr_nn_dt, tpr_nn_dt, label='NN nn_dt {:.3f}'.format(roc_index_nn_dt
plt.plot(fpr_nn_dt_cv, tpr_nn_dt_cv, label='NN with nn_dt_cv {:.3f}'.format(r


plt.plot([0, 1], [0, 1], color='navy', lw=0.5, linestyle='--')
```

```
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

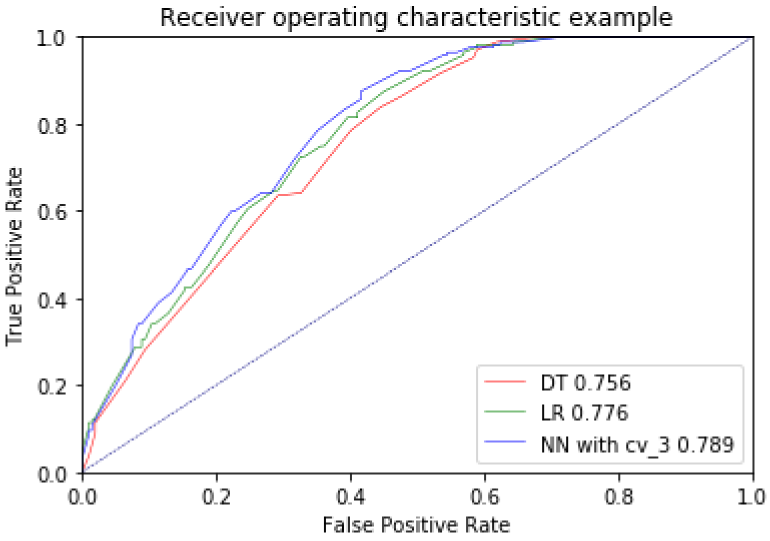


Based on the ROC curve as avove, the neural network model with grid search(hidden_layer_sizes = 17, alpha = 0.01, cv_NN_3) performs the best with ROC index of 0.789.

Task 5 Decision making

In [88]:

```
import pickle
with open('DT.pickle', 'rb') as f:
    dt_best,roc_index_dt_cv, fpr_dt_cv, tpr_dt_cv = pickle.load(f)
with open('LR.pickle', 'rb') as f:
    lr_best,roc_index_lr_cv, fpr_lr_cv, tpr_lr_cv = pickle.load(f)


plt.plot(fpr_dt_cv, tpr_dt_cv, label='DT {:.3f}'.format(roc_index_dt_cv), col
plt.plot(fpr_lr_cv, tpr_lr_cv, label='LR {:.3f}'.format(roc_index_lr_cv), col
plt.plot(fpr_cv_3, tpr_cv_3, label='NN with cv_3 {:.3f}'.format(roc_index_cv_

plt.plot([0, 1], [0, 1], color='navy', lw=0.5, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```



Overall, neural network with grid search (hidden layers = 17 and alpha = 0.01, cv_NN_3) are the best performing model of all the models.

In [ ]:

```

```