ExitFactors

January 6, 2020

1 Introduction

The purpose of this analysis is to:

1) Examine which features are the most important in predicting when an employee will indicate on their Exit Interview survey that they *would* or *would not* recommend the compnay as a place of employment to a friend.

2) Sketch out code for modeling future data sets.

The predictor variables are the ratings, generally with values ranging from 1-5, which employees give for various facets of job satisfaction. The response variable is a binary global assessment of employee attitudes towards the organization, captured through the survey question, “Would you recommend working for this company to a friend?”

[1]:
```
# import basic analysis packages
import numpy as np
import pandas as pd
# RAND_STATE = 101
RAND_STATE = np.random.randint(1, 100000)
```

[2]:
```
pd.show_versions(as_json=False)
```

```
INSTALLED VERSIONS
------------------
commit           : None
python           : 3.7.4.final.0
python-bits      : 64
OS               : Windows
OS-release       : 10
machine          : AMD64
processor        : Intel64 Family 6 Model 85 Stepping 4, GenuineIntel
byteorder        : little
LC_ALL           : None
LANG             : None
LOCALE           : None.None

pandas           : 0.25.1
numpy            : 1.16.5
```

```
pytz             : 2019.3
dateutil         : 2.8.0
pip              : 19.2.3
setuptools       : 41.4.0
Cython           : 0.29.13
pytest           : 5.2.1
hypothesis       : None
sphinx           : 2.2.0
blosc            : None
feather          : None
xlsxwriter       : 1.2.1
lxml.etree       : 4.4.1
html5lib         : 1.0.1
pymysql          : 0.9.3
psycopg2         : None
jinja2           : 2.10.3
IPython          : 7.8.0
pandas_datareader: None
bs4              : 4.8.0
bottleneck       : 1.2.1
fastparquet      : None
gcsfs            : None
lxml.etree       : 4.4.1
matplotlib       : 3.1.1
numexpr          : 2.7.0
odfpy            : None
openpyxl         : 3.0.0
pandas_gbq       : None
pyarrow          : None
pytables         : None
s3fs             : None
scipy            : 1.3.1
sqlalchemy       : 1.3.9
tables           : 3.5.2
xarray           : None
xlrd             : 1.2.0
xlwt             : 1.3.0
xlsxwriter       : 1.2.1
```

[3]:
```
# import data set
from sqlalchemy import create_engine
eng = 'mysql+pymysql://[REDACTED]'

db = create_engine(eng)
emp_exit_raw = pd.read_sql_query("""SELECT
                                 SupPol, SupInf, SupFair, SupRec, SupCoop,␣
 ↪SupResolve, SupTrn,
```

```
                                  DeptCom, DeptCond, DeptCoop, DeptAdv,
                                  RatePay, AnnLeave, PdHoliday, DevEdu, MDVIns,
                                  Recommend
                         FROM empexit ORDER BY RespID DESC""", db)
```

2 Exploratory Data Analysis/Data Preparation

[4]:
```
emp_exit_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120 entries, 0 to 119
Data columns (total 17 columns):
SupPol        119 non-null float64
SupInf        120 non-null int64
SupFair       120 non-null int64
SupRec        120 non-null int64
SupCoop       119 non-null float64
SupResolve    120 non-null int64
SupTrn        116 non-null float64
DeptCom       119 non-null float64
DeptCond      120 non-null int64
DeptCoop      119 non-null float64
DeptAdv       119 non-null float64
RatePay       120 non-null int64
AnnLeave      117 non-null float64
PdHoliday     118 non-null float64
DevEdu        117 non-null float64
MDVIns        119 non-null float64
Recommend     119 non-null object
dtypes: float64(10), int64(6), object(1)
memory usage: 16.1+ KB
```

[5]:
```
emp_exit_raw.describe()
```

[5]:
```
           SupPol      SupInf     SupFair      SupRec     SupCoop  SupResolve  \
count  119.000000  120.000000  120.000000  120.000000  119.000000  120.000000
mean     4.319328    4.158333    3.950000    3.941667    3.991597    3.741667
std      0.956094    1.076844    1.377185    1.336594    1.189615    1.381120
min      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000
25%      4.000000    4.000000    3.000000    3.000000    3.000000    3.000000
50%      5.000000    5.000000    5.000000    5.000000    4.000000    4.000000
75%      5.000000    5.000000    5.000000    5.000000    5.000000    5.000000
max      5.000000    5.000000    5.000000    5.000000    5.000000    5.000000

           SupTrn     DeptCom    DeptCond    DeptCoop     DeptAdv     RatePay  \
count  116.000000  119.000000  120.000000  119.000000  119.000000  120.000000
```

```
mean     3.784483    3.571429    3.941667    3.932773    2.605042    2.591667
std      1.330616    1.266059    0.989829    1.169754    1.290240    1.260224
min      1.000000    1.000000    1.000000    1.000000    1.000000    0.000000
25%      3.000000    3.000000    3.000000    3.000000    1.000000    1.000000
50%      4.000000    4.000000    4.000000    4.000000    3.000000    3.000000
75%      5.000000    5.000000    5.000000    5.000000    4.000000    3.000000
max      5.000000    5.000000    5.000000    5.000000    5.000000    5.000000

         AnnLeave   PdHoliday      DevEdu      MDVIns
count  117.000000  118.000000  117.000000  119.000000
mean     3.641026    3.711864    3.017094    2.638655
std      1.392477    1.288355    1.828806    1.839911
min      0.000000    0.000000    0.000000    0.000000
25%      3.000000    3.000000    2.000000    1.000000
50%      4.000000    4.000000    3.000000    3.000000
75%      5.000000    5.000000    5.000000    4.000000
max      5.000000    5.000000    5.000000    5.000000
```

[6]:
```python
emp_exit_raw.head(10)
```

[6]:
```
   SupPol  SupInf  SupFair  SupRec  SupCoop  SupResolve  SupTrn  DeptCom  \
0     1.0       1        1       1      1.0           1     1.0      1.0
1     5.0       5        4       5      5.0           5     5.0      5.0
2     5.0       5        5       5      5.0           5     5.0      5.0
3     4.0       1        4       3      4.0           2     2.0      4.0
4     5.0       5        5       5      5.0           5     5.0      5.0
5     5.0       4        4       3      4.0           4     5.0      1.0
6     3.0       3        3       3      3.0           3     3.0      3.0
7     1.0       2        1       1      1.0           1     1.0      1.0
8     5.0       5        5       5      4.0           5     4.0      5.0
9     5.0       5        5       5      5.0           5     5.0      4.0

   DeptCond  DeptCoop  DeptAdv  RatePay  AnnLeave  PdHoliday  DevEdu  MDVIns  \
0         3       3.0      1.0        3       3.0        4.0     5.0     1.0
1         5       4.0      4.0        1       5.0        3.0     5.0     0.0
2         5       5.0      2.0        1       5.0        4.0     0.0     3.0
3         4       3.0      2.0        4       5.0        4.0     4.0     4.0
4         5       5.0      2.0        1       3.0        5.0     2.0     5.0
5         3       2.0      2.0        1       4.0        5.0     0.0     4.0
6         3       3.0      3.0        3       3.0        3.0     3.0     3.0
7         3       4.0      3.0        3       4.0        4.0     0.0     4.0
8         5       5.0      5.0        5       5.0        5.0     5.0     5.0
9         5       5.0      4.0        3       5.0        5.0     5.0     0.0

  Recommend
0         1
1         1
```

```
2          2
3          2
4          1
5          2
6          1
7          1
8          1
9          1
```

2.1 Indicator Variables, Missingness

[7]:
```
# function rec_word - takes a data frame row and converts 1s and 2s to words
# this is just in case we want to display the words.  Also, it will fix it so␣
↪that
# when we use get_dummies, responses of 'yes' will be encoded as 1s and 'no'␣
↪will be encoded as 0s
# rather than (the confusing) 1s and 2s, respectively.
def rec_word (row):
    if row['Recommend'] == '1':
        return 'yes'
    if row['Recommend'] == '2':
        return 'no'
    return 'Other'
```

[8]:
```
emp_exit_raw.isnull().sum()
```

[8]:
```
SupPol        1
SupInf        0
SupFair       0
SupRec        0
SupCoop       1
SupResolve    0
SupTrn        4
DeptCom       1
DeptCond      0
DeptCoop      1
DeptAdv       1
RatePay       0
AnnLeave      3
PdHoliday     2
DevEdu        3
MDVIns        1
Recommend     1
dtype: int64
```

[9]:
```python
# convert types, transform target variable
emp_exit = emp_exit_raw.dropna(axis=0)
emp_exit.info()
emp_exit = emp_exit.assign(SupPol = emp_exit['SupPol'].astype(int))
emp_exit = emp_exit.assign(SupCoop = emp_exit['SupCoop'].astype(int))
emp_exit = emp_exit.assign(SupTrn = emp_exit['SupTrn'].astype(int))
emp_exit = emp_exit.assign(DeptCom = emp_exit['DeptCom'].astype(int))
emp_exit = emp_exit.assign(DeptCoop = emp_exit['DeptCoop'].astype(int))
emp_exit = emp_exit.assign(DeptAdv = emp_exit['DeptAdv'].astype(int))

emp_exit = emp_exit.assign(AnnLeave = emp_exit['AnnLeave'].astype(int))
emp_exit = emp_exit.assign(PdHoliday = emp_exit['PdHoliday'].astype(int))
emp_exit = emp_exit.assign(DevEdu = emp_exit['DevEdu'].astype(int))
emp_exit = emp_exit.assign(MDVIns = emp_exit['MDVIns'].astype(int))

emp_exit.reset_index()
emp_exit = emp_exit.assign(rec_word = emp_exit.apply (lambda row:␣
↪rec_word(row), axis=1))
emp_exit.loc[:,'rec'] = pd.get_dummies(emp_exit['rec_word'],␣
↪drop_first=True)['yes']
del emp_exit['Recommend']
del emp_exit['rec_word']

emp_exit.head(10)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 110 entries, 0 to 119
Data columns (total 17 columns):
SupPol        110 non-null float64
SupInf        110 non-null int64
SupFair       110 non-null int64
SupRec        110 non-null int64
SupCoop       110 non-null float64
SupResolve    110 non-null int64
SupTrn        110 non-null float64
DeptCom       110 non-null float64
DeptCond      110 non-null int64
DeptCoop      110 non-null float64
DeptAdv       110 non-null float64
RatePay       110 non-null int64
AnnLeave      110 non-null float64
PdHoliday     110 non-null float64
DevEdu        110 non-null float64
MDVIns        110 non-null float64
Recommend     110 non-null object
dtypes: float64(10), int64(6), object(1)
memory usage: 15.5+ KB
```

[9]:
```
   SupPol  SupInf  SupFair  SupRec  SupCoop  SupResolve  SupTrn  DeptCom  \
0       1       1        1       1        1           1       1        1
1       5       5        4       5        5           5       5        5
2       5       5        5       5        5           5       5        5
3       4       1        4       3        4           2       2        4
4       5       5        5       5        5           5       5        5
5       5       4        4       3        4           4       5        1
6       3       3        3       3        3           3       3        3
7       1       2        1       1        1           1       1        1
8       5       5        5       5        4           5       4        5
9       5       5        5       5        5           5       5        4

   DeptCond  DeptCoop  DeptAdv  RatePay  AnnLeave  PdHoliday  DevEdu  MDVIns  \
0         3         3        1        3         3          4       5       1
1         5         4        4        1         5          3       5       0
2         5         5        2        1         5          4       0       3
3         4         3        2        4         5          4       4       4
4         5         5        2        1         3          5       2       5
5         3         2        2        1         4          5       0       4
6         3         3        3        3         3          3       3       3
7         3         4        3        3         4          4       0       4
8         5         5        5        5         5          5       5       5
9         5         5        4        3         5          5       5       0

   rec
0    1
1    1
2    0
3    0
4    1
5    0
6    1
7    1
8    1
9    1
```

[10]:
```
emp_exit.apply(pd.Series.value_counts)
```

[10]:
```
   SupPol  SupInf  SupFair  SupRec  SupCoop  SupResolve  SupTrn  DeptCom  \
0     NaN     NaN      NaN     NaN      NaN         NaN     NaN      NaN
1     2.0     4.0     12.0     8.0      5.0        11.0    10.0     11.0
2     3.0     6.0      7.0    12.0     11.0        13.0    12.0     12.0
3    17.0    16.0     14.0    17.0     18.0        20.0    20.0     24.0
4    24.0    27.0     20.0    16.0     22.0        16.0    19.0     33.0
5    64.0    57.0     57.0    57.0     54.0        50.0    49.0     30.0

   DeptCond  DeptCoop  DeptAdv  RatePay  AnnLeave  PdHoliday  DevEdu  MDVIns  \
```

```
0        NaN        NaN       NaN       NaN          6           3        22        23
1        2.0        7.0      30.0      28.0          2           6         4        13
2        5.0        8.0      24.0      20.0          9           6         5         6
3       27.0       19.0      27.0      38.0         29          28        28        22
4       39.0       33.0      20.0      15.0         25          29        21        22
5       37.0       43.0       9.0       9.0         39          38        30        24

    rec
0  37.0
1  73.0
2   NaN
3   NaN
4   NaN
5   NaN
```

Nearly a quarter of respondents did not have staff development funds or medical insurance. Remove those fields. Annual Leave did not apply to 6 employees and Paid Holidays did not apply to 3 employees (where score of 0 was entered). Response variable is fairly unbalanced with a 37/73 split between employees not recommending the company and recommending the company.

[11]:
```
emp_exit = emp_exit.drop(columns=['DevEdu', 'MDVIns'])
emp_exit.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 110 entries, 0 to 119
Data columns (total 15 columns):
SupPol        110 non-null int32
SupInf        110 non-null int64
SupFair       110 non-null int64
SupRec        110 non-null int64
SupCoop       110 non-null int32
SupResolve    110 non-null int64
SupTrn        110 non-null int32
DeptCom       110 non-null int32
DeptCond      110 non-null int64
DeptCoop      110 non-null int32
DeptAdv       110 non-null int32
RatePay       110 non-null int64
AnnLeave      110 non-null int32
PdHoliday     110 non-null int32
rec           110 non-null uint8
dtypes: int32(8), int64(6), uint8(1)
memory usage: 9.6 KB
```

[12]:
```
emp_exit.apply(pd.Series.value_counts)
```

[12]:
```
   SupPol  SupInf  SupFair  SupRec  SupCoop  SupResolve  SupTrn  DeptCom  \
0     NaN     NaN      NaN     NaN      NaN         NaN     NaN      NaN
1     2.0     4.0     12.0     8.0      5.0        11.0    10.0     11.0
2     3.0     6.0      7.0    12.0     11.0        13.0    12.0     12.0
3    17.0    16.0     14.0    17.0     18.0        20.0    20.0     24.0
4    24.0    27.0     20.0    16.0     22.0        16.0    19.0     33.0
5    64.0    57.0     57.0    57.0     54.0        50.0    49.0     30.0

   DeptCond  DeptCoop  DeptAdv  RatePay  AnnLeave  PdHoliday   rec
0       NaN       NaN      NaN      NaN         6          3  37.0
1       2.0       7.0     30.0     28.0         2          6  73.0
2       5.0       8.0     24.0     20.0         9          6   NaN
3      27.0      19.0     27.0     38.0        29         28   NaN
4      39.0      33.0     20.0     15.0        25         29   NaN
5      37.0      43.0      9.0      9.0        39         38   NaN
```

2.2 Anomalous Values

A score of 0 means that the element did not apply to the employee, e.g. they may not have had paid vacation time if they were a part-time employee. This affected columns Annual Leave and Paid Holiday. Because raw data is better visualized as categorical values, these 0s will be replaced by column modes for those two columns.

[13]:
```python
# replace zeros with modes for AnnLeave and PdHoliday columns
values = emp_exit['AnnLeave']
md_AL = values[values != 0].mode()
emp_exit.loc[emp_exit.AnnLeave < 1, 'AnnLeave'] = md_AL[0]

values = emp_exit['PdHoliday']
md_PH = values[values != 0].mode()
emp_exit.loc[emp_exit.PdHoliday < 1, 'PdHoliday'] = md_PH[0]

emp_exit.apply(pd.Series.value_counts)
```

[13]:
```
   SupPol  SupInf  SupFair  SupRec  SupCoop  SupResolve  SupTrn  DeptCom  \
0     NaN     NaN      NaN     NaN      NaN         NaN     NaN      NaN
1     2.0     4.0     12.0     8.0      5.0        11.0    10.0     11.0
2     3.0     6.0      7.0    12.0     11.0        13.0    12.0     12.0
3    17.0    16.0     14.0    17.0     18.0        20.0    20.0     24.0
4    24.0    27.0     20.0    16.0     22.0        16.0    19.0     33.0
5    64.0    57.0     57.0    57.0     54.0        50.0    49.0     30.0

   DeptCond  DeptCoop  DeptAdv  RatePay  AnnLeave  PdHoliday   rec
0       NaN       NaN      NaN      NaN       NaN        NaN  37.0
1       2.0       7.0     30.0     28.0       2.0        6.0  73.0
2       5.0       8.0     24.0     20.0       9.0        6.0   NaN
3      27.0      19.0     27.0     38.0      29.0       28.0   NaN
```

```
4      39.0      33.0     20.0     15.0      25.0       29.0   NaN
5      37.0      43.0      9.0      9.0      45.0       41.0   NaN
```

[14]:
```
emp_exit.isnull().sum()
```

[14]:
```
SupPol        0
SupInf        0
SupFair       0
SupRec        0
SupCoop       0
SupResolve    0
SupTrn        0
DeptCom       0
DeptCond      0
DeptCoop      0
DeptAdv       0
RatePay       0
AnnLeave      0
PdHoliday     0
rec           0
dtype: int64
```

2.3 Duplicates

[15]:
```
# are there duplicate values?
emp_exit.duplicated().value_counts()
```

[15]:
```
False    108
True       2
dtype: int64
```

[16]:
```
emp_exit.loc[emp_exit.duplicated() == True, :]
```

[16]:
```
     SupPol  SupInf  SupFair  SupRec  SupCoop  SupResolve  SupTrn  DeptCom  \
92        5       5        5       5        5           5       5        5
108       5       5        5       5        5           5       5        5

     DeptCond  DeptCoop  DeptAdv  RatePay  AnnLeave  PdHoliday  rec
92          5         5        4        4         5          5    1
108         5         5        5        5         5          5    1
```

It turns out that these were simply instances of matching values and represent valid observations. Will leave in the data set.

2.4 Data Partitioning/Balancing

This is a very small data set. Trial-and-error showed significant improvement when allowing a slightly larger training set. Will use 80/20 train/test split.

[17]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(emp_exit.drop('rec',␣
↪axis=1),
                                                    emp_exit['rec'],␣
↪test_size=0.20,
                                                    random_state=RAND_STATE)

X = emp_exit.drop('rec', axis=1)
y = emp_exit['rec']

print(X_train.shape, y_train.shape)
print(X.shape, y.shape)
features = list(X_train.columns.values) # for boruta
emp_exit['rec'].sum()
```

```
(88, 14) (88,)
(110, 14) (110,)
```

[17]: 73

[18]:
```
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

oversampling = False

# Note: Use of SMOTE changes data set from a data frame to a numpy array.
#   This interferes with code below that produces feature importances.
#   When using SMOTE, oversampling will be set to True

# oversampling = True
# sm = SMOTE(random_state = RAND_STATE)

# X_train_bal, y_train_bal = sm.fit_sample(X_train, y_train)
# print(X_train.shape, y_train.shape)
# print(type(X_train))

# X_train = X_train_bal  # comment to remove oversampling
# y_train = y_train_bal  # comment to remove oversampling
# print(X_train.shape, y_train.shape)
# print(type(X_train))
# print(X_train)
```

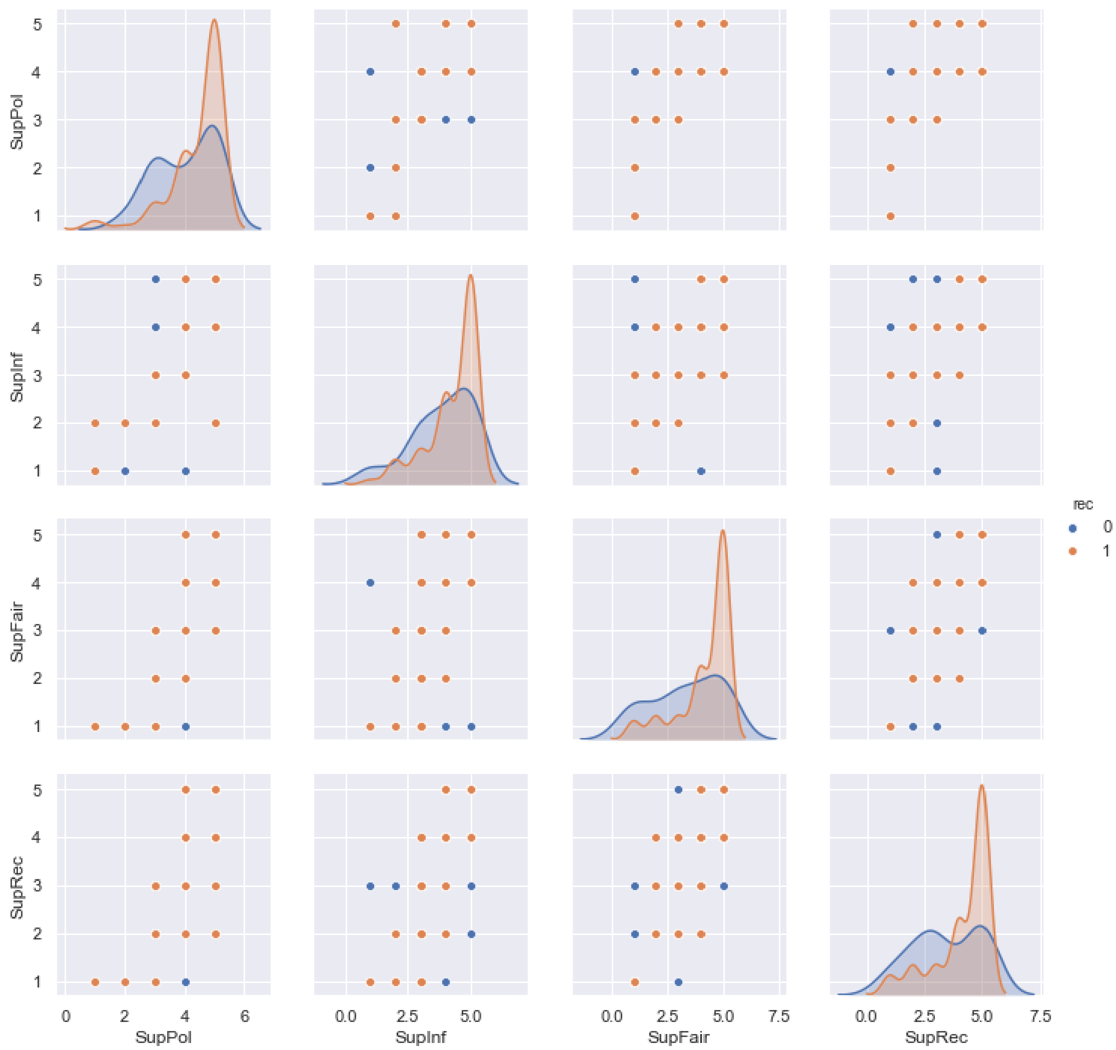
3 Feature Examination

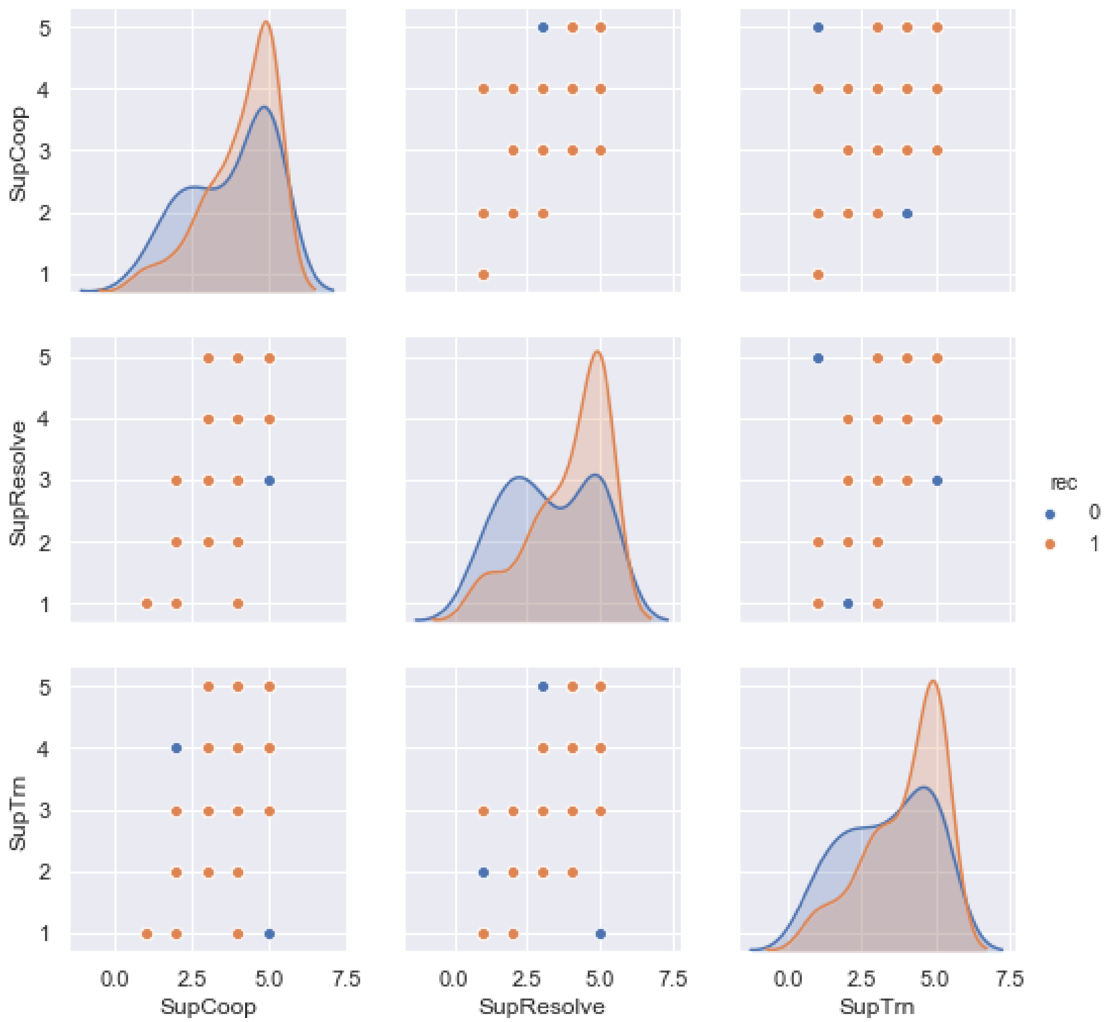
3.1 Comparison: those who recommend the organization vs. those who do not

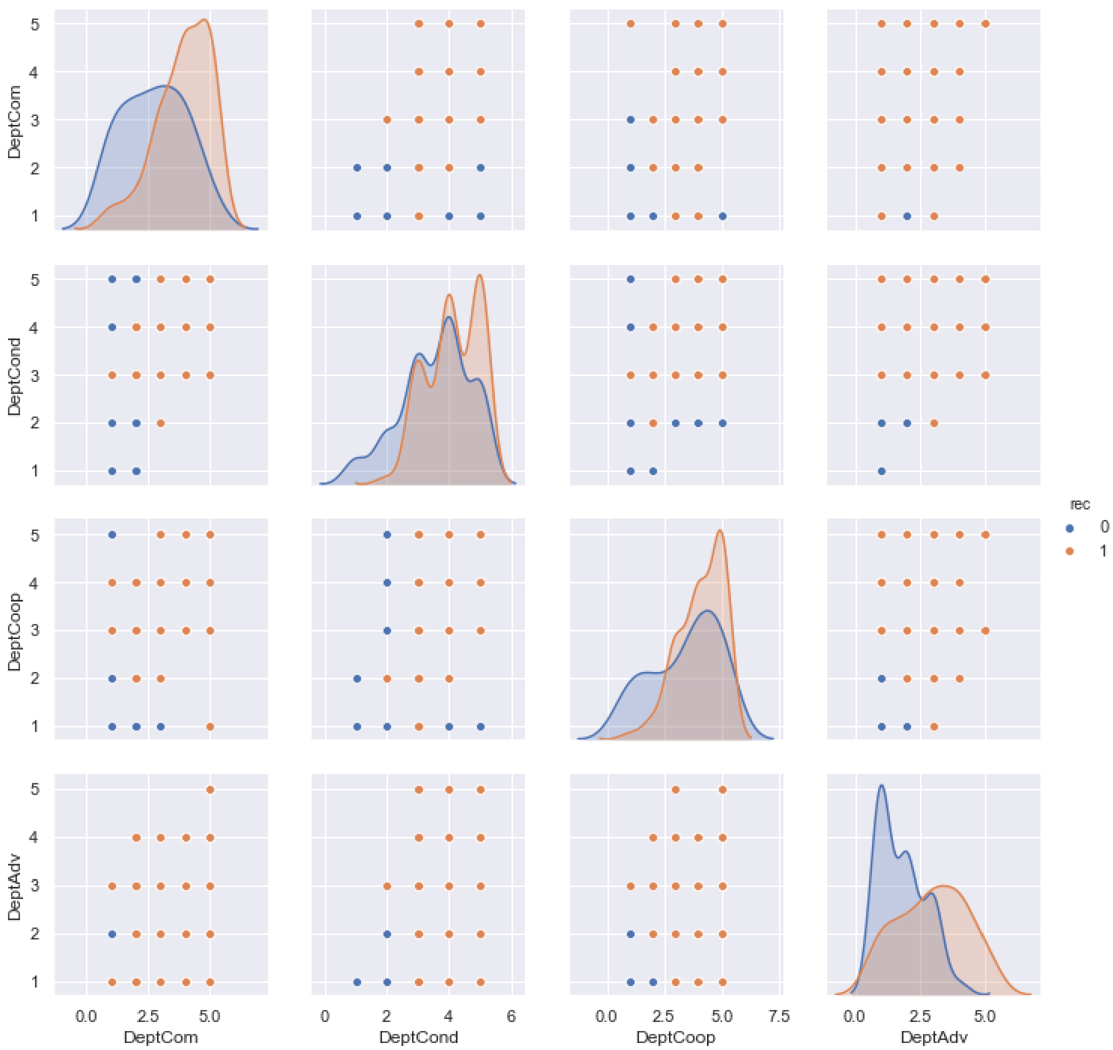
Let's examine the differences in the score distributions between those who indicate they would recommend the organization to a friend ('yay'), vs those who indicate they would not ('nay').

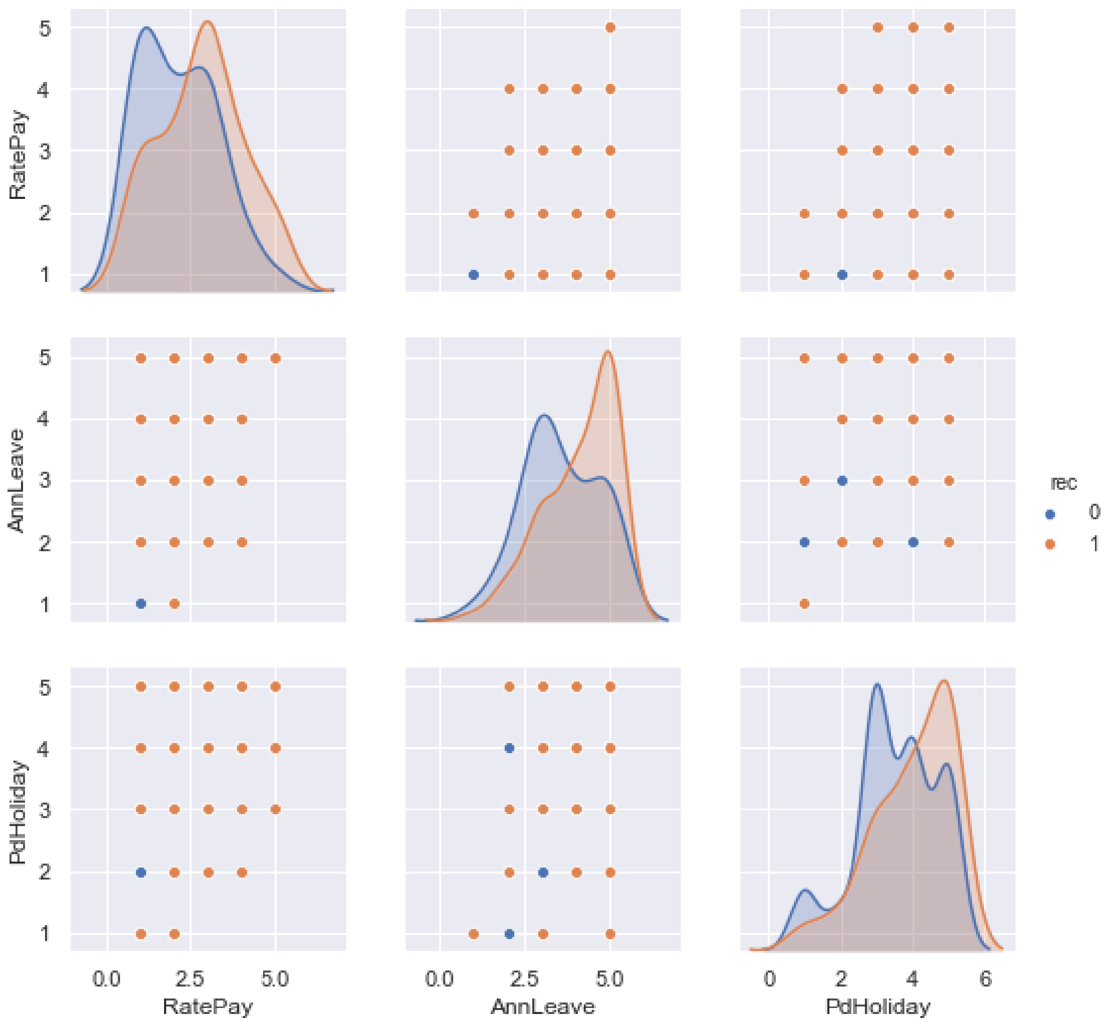
[65]:
```
# compare distributions of answers
import seaborn as sns

sup_cols1 = ['SupPol', 'SupInf', 'SupFair', 'SupRec']
sup_cols2 = ['SupCoop', 'SupResolve', 'SupTrn']
dept_cols = ['DeptCom', 'DeptCond', 'DeptCoop', 'DeptAdv']
ben_cols  = ['RatePay', 'AnnLeave', 'PdHoliday']

sns.pairplot(emp_exit, vars=sup_cols1, hue='rec')
sns.pairplot(emp_exit, vars=sup_cols2, hue='rec')
sns.pairplot(emp_exit, vars=dept_cols, hue='rec')
sns.pairplot(emp_exit, vars=ben_cols, hue='rec')
```

[65]:
```
<seaborn.axisgrid.PairGrid at 0x2306eaae9c8>
```


SupPol
SupInf
SupFair
SupRec
rec
0
1


SupCoop
SupResolve
SupTrn
rec
0
1
0.0
2.5
5.0
7.5
1
2
3
4
5


DeptCom
DeptCond
DeptCoop
DeptAdv
rec
0
1

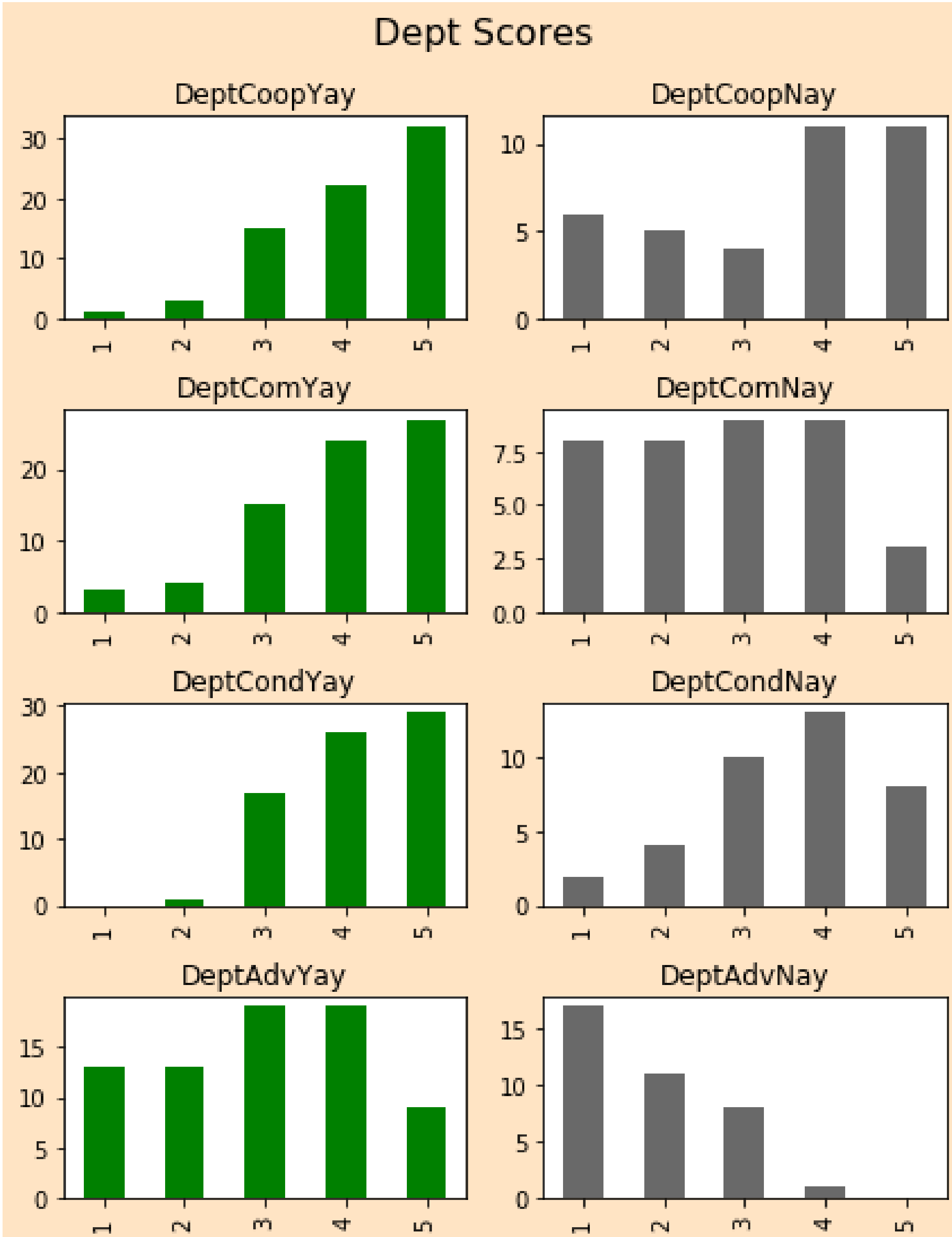


[20]:
```
# or...manually generate histograms for comparison

# separate into those who recommend VOH and those who do not
emp_ex_yay = emp_exit.loc[emp_exit['rec'] == 1, : ]
emp_ex_nay = emp_exit.loc[emp_exit['rec'] == 0, : ]
```

[21]:
```
# produce plottable series
def make_1to5_series(col):
    ones = len(col[col == 1])
    twos = len(col[col == 2])
    threes = len(col[col == 3])
    fours = len(col[col == 4])
    fives = len(col[col == 5])
    return pd.Series(data=(ones, twos, threes, fours, fives), index=(1, 2, 3,␣
↪4, 5))
```

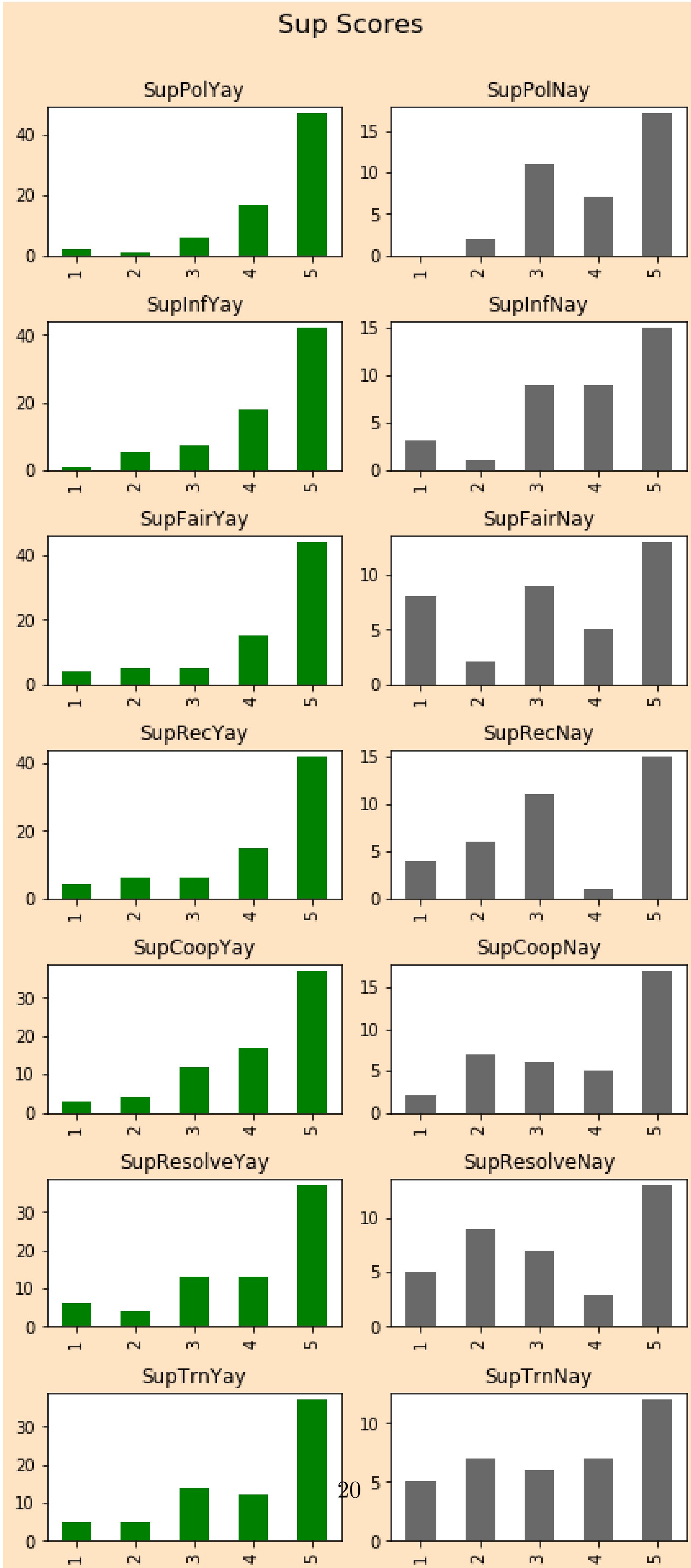
[22]:
```
# visualization prep
import matplotlib.pyplot as plt
%matplotlib inline
color_yay = 'green'
color_nay = 'dimgray'
color_face = 'bisque'
```

[23]:
```
DeptCoopYay_totals = make_1to5_series(emp_ex_yay['DeptCoop'])
DeptCoopNay_totals = make_1to5_series(emp_ex_nay['DeptCoop'])
DeptComYay_totals = make_1to5_series(emp_ex_yay['DeptCom'])
DeptComNay_totals = make_1to5_series(emp_ex_nay['DeptCom'])
DeptCondYay_totals = make_1to5_series(emp_ex_yay['DeptCond'])
DeptCondNay_totals = make_1to5_series(emp_ex_nay['DeptCond'])
DeptAdvYay_totals = make_1to5_series(emp_ex_yay['DeptAdv'])
DeptAdvNay_totals = make_1to5_series(emp_ex_nay['DeptAdv'])

fig, axes = plt.subplots(4, 2, figsize=(6, 8), facecolor=color_face)
fig.suptitle('Dept Scores', size=16)
DeptCoopYay_totals.plot.bar(ax=axes[0,0], color=color_yay, alpha=1, title =␣
 ↪'DeptCoopYay')
DeptCoopNay_totals.plot.bar(ax=axes[0,1], color=color_nay, alpha=1, title =␣
 ↪'DeptCoopNay')
DeptComYay_totals.plot.bar(ax=axes[1,0], color=color_yay, alpha=1, title =␣
 ↪'DeptComYay')
DeptComNay_totals.plot.bar(ax=axes[1,1], color=color_nay, alpha=1, title =␣
 ↪'DeptComNay')
DeptCondYay_totals.plot.bar(ax=axes[2,0], color=color_yay, alpha=1, title =␣
 ↪'DeptCondYay')
DeptCondNay_totals.plot.bar(ax=axes[2,1], color=color_nay, alpha=1, title =␣
 ↪'DeptCondNay')
DeptAdvYay_totals.plot.bar(ax=axes[3,0], color=color_yay, alpha=1, title =␣
 ↪'DeptAdvYay')
DeptAdvNay_totals.plot.bar(ax=axes[3,1], color=color_nay, alpha=1, title =␣
 ↪'DeptAdvNay')
fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```



[24]:
```
SupPolYay_totals = make_1to5_series(emp_ex_yay['SupPol'])
SupPolNay_totals = make_1to5_series(emp_ex_nay['SupPol'])
SupInfYay_totals = make_1to5_series(emp_ex_yay['SupInf'])
SupInfNay_totals = make_1to5_series(emp_ex_nay['SupInf'])
```

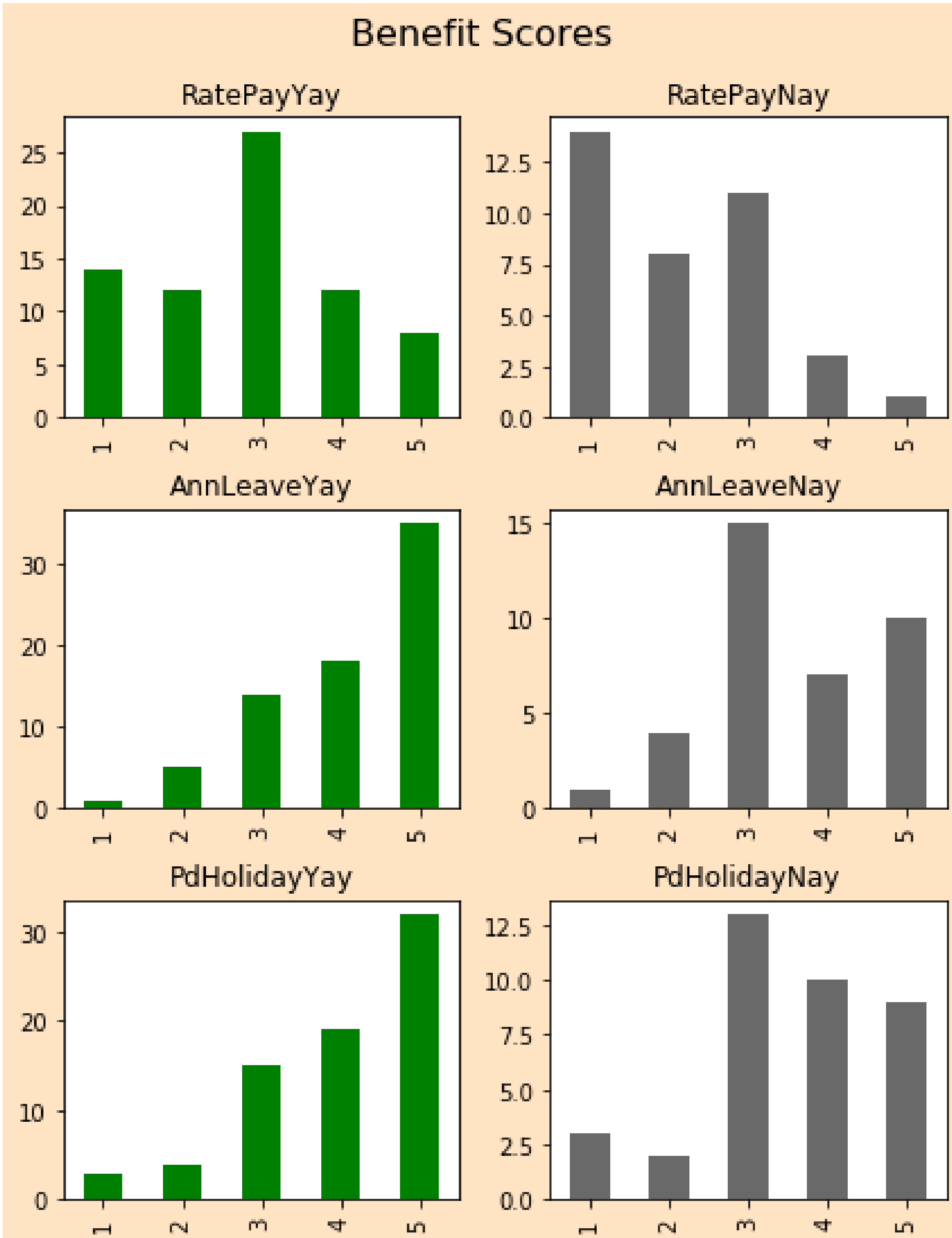
```
SupFairYay_totals = make_1to5_series(emp_ex_yay['SupFair'])
SupFairNay_totals = make_1to5_series(emp_ex_nay['SupFair'])
SupRecYay_totals = make_1to5_series(emp_ex_yay['SupRec'])
SupRecNay_totals = make_1to5_series(emp_ex_nay['SupRec'])
SupCoopYay_totals = make_1to5_series(emp_ex_yay['SupCoop'])
SupCoopNay_totals = make_1to5_series(emp_ex_nay['SupCoop'])
SupResolveYay_totals = make_1to5_series(emp_ex_yay['SupResolve'])
SupResolveNay_totals = make_1to5_series(emp_ex_nay['SupResolve'])
SupTrnYay_totals = make_1to5_series(emp_ex_yay['SupTrn'])
SupTrnNay_totals = make_1to5_series(emp_ex_nay['SupTrn'])

fig, axes = plt.subplots(7, 2, figsize=(6,14), facecolor=color_face)
fig.suptitle('Sup Scores', size=16)
SupPolYay_totals.plot.bar(ax=axes[0,0], color=color_yay, alpha=1,␣
↪title='SupPolYay')
SupPolNay_totals.plot.bar(ax=axes[0,1], color=color_nay, alpha=1,␣
↪title='SupPolNay')
SupInfYay_totals.plot.bar(ax=axes[1,0], color=color_yay, alpha=1,␣
↪title='SupInfYay')
SupInfNay_totals.plot.bar(ax=axes[1,1], color=color_nay, alpha=1,␣
↪title='SupInfNay')
SupFairYay_totals.plot.bar(ax=axes[2,0], color=color_yay, alpha=1,␣
↪title='SupFairYay')
SupFairNay_totals.plot.bar(ax=axes[2,1], color=color_nay, alpha=1,␣
↪title='SupFairNay')
SupRecYay_totals.plot.bar(ax=axes[3,0], color=color_yay, alpha=1,␣
↪title='SupRecYay')
SupRecNay_totals.plot.bar(ax=axes[3,1], color=color_nay, alpha=1,␣
↪title='SupRecNay')
SupCoopYay_totals.plot.bar(ax=axes[4,0], color=color_yay, alpha=1,␣
↪title='SupCoopYay')
SupCoopNay_totals.plot.bar(ax=axes[4,1], color=color_nay, alpha=1,␣
↪title='SupCoopNay')
SupResolveYay_totals.plot.bar(ax=axes[5,0], color=color_yay, alpha=1,␣
↪title='SupResolveYay')
SupResolveNay_totals.plot.bar(ax=axes[5,1], color=color_nay, alpha=1,␣
↪title='SupResolveNay')
SupTrnYay_totals.plot.bar(ax=axes[6,0], color=color_yay, alpha=1,␣
↪title='SupTrnYay')
SupTrnNay_totals.plot.bar(ax=axes[6,1], color=color_nay, alpha=1,␣
↪title='SupTrnNay')
fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```



Sup Scores
SupPolYay
SupPolNay
SupInfYay
SupInfNay
SupFairYay
SupFairNay
SupRecYay
SupRecNay
SupCoopYay
SupCoopNay
SupResolveYay
SupResolveNay
SupTrnYay
SupTrnNay

[25]:

```
RatePayYay_totals = make_1to5_series(emp_ex_yay['RatePay'])
RatePayNay_totals = make_1to5_series(emp_ex_nay['RatePay'])
AnnLeaveYay_totals = make_1to5_series(emp_ex_yay['AnnLeave'])
AnnLeaveNay_totals = make_1to5_series(emp_ex_nay['AnnLeave'])
PdHolidayYay_totals = make_1to5_series(emp_ex_yay['PdHoliday'])
PdHolidayNay_totals = make_1to5_series(emp_ex_nay['PdHoliday'])

fig, axes = plt.subplots(3, 2, figsize=(6,8), facecolor=color_face)
fig.suptitle('Benefit Scores', size=16)
RatePayYay_totals.plot.bar(ax=axes[0,0], color=color_yay, alpha=1,␣
↪title='RatePayYay')
RatePayNay_totals.plot.bar(ax=axes[0,1], color=color_nay, alpha=1,␣
↪title='RatePayNay')
AnnLeaveYay_totals.plot.bar(ax=axes[1,0], color=color_yay, alpha=1,␣
↪title='AnnLeaveYay')
AnnLeaveNay_totals.plot.bar(ax=axes[1,1], color=color_nay, alpha=1,␣
↪title='AnnLeaveNay')
PdHolidayYay_totals.plot.bar(ax=axes[2,0], color=color_yay, alpha=1,␣
↪title='PdHolidayYay')
PdHolidayNay_totals.plot.bar(ax=axes[2,1], color=color_nay, alpha=1,␣
↪title='PdHolidayNay')
fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```


Benefit Scores
RatePayYay
RatePayNay
AnnLeaveYay
AnnLeaveNay
PdHolidayYay
PdHolidayNay

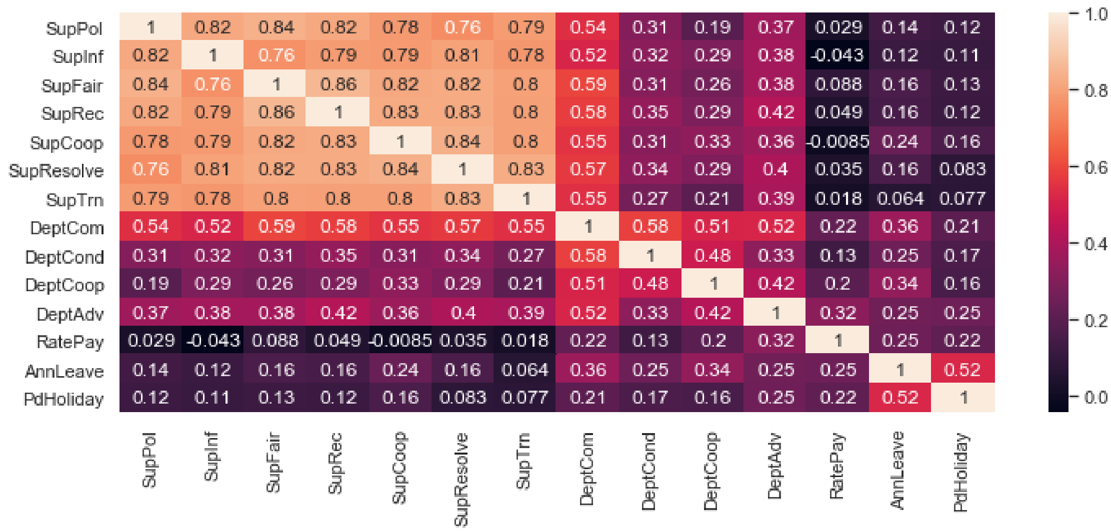
3.2 Feature Independence

[26]:
```python
# does the data set meet the assumptions of logistic regression?
# pd.plotting.scatter_matrix(X, alpha=0.5, figsize=(14, 8), diagonal='kde')
corr = X.corr()
print(corr)
```

```
              SupPol    SupInf   SupFair    SupRec   SupCoop  SupResolve  \
SupPol      1.000000  0.820232  0.844072  0.816991  0.783293    0.755152
SupInf      0.820232  1.000000  0.762225  0.788657  0.787951    0.806342
SupFair     0.844072  0.762225  1.000000  0.857976  0.816738    0.824832
SupRec      0.816991  0.788657  0.857976  1.000000  0.826965    0.826471
SupCoop     0.783293  0.787951  0.816738  0.826965  1.000000    0.840308
SupResolve  0.755152  0.806342  0.824832  0.826471  0.840308    1.000000
SupTrn      0.789581  0.776900  0.801341  0.796548  0.798609    0.831893
DeptCom     0.542721  0.518755  0.594334  0.579286  0.553552    0.568335
DeptCond    0.306747  0.320739  0.307888  0.346306  0.312223    0.335532
DeptCoop    0.193711  0.288053  0.263244  0.288656  0.334265    0.294156
DeptAdv     0.369437  0.378526  0.383341  0.420649  0.360991    0.401625
RatePay     0.028580 -0.043171  0.087780  0.049489 -0.008503    0.035388
AnnLeave    0.137811  0.118215  0.163292  0.155542  0.236882    0.163094
PdHoliday   0.120003  0.113906  0.126924  0.118270  0.162969    0.082722

              SupTrn   DeptCom  DeptCond  DeptCoop   DeptAdv   RatePay  \
SupPol      0.789581  0.542721  0.306747  0.193711  0.369437  0.028580
SupInf      0.776900  0.518755  0.320739  0.288053  0.378526 -0.043171
SupFair     0.801341  0.594334  0.307888  0.263244  0.383341  0.087780
SupRec      0.796548  0.579286  0.346306  0.288656  0.420649  0.049489
SupCoop     0.798609  0.553552  0.312223  0.334265  0.360991 -0.008503
SupResolve  0.831893  0.568335  0.335532  0.294156  0.401625  0.035388
SupTrn      1.000000  0.547263  0.270055  0.209401  0.390773  0.017644
DeptCom     0.547263  1.000000  0.582484  0.511734  0.517577  0.221710
DeptCond    0.270055  0.582484  1.000000  0.479551  0.328181  0.128123
DeptCoop    0.209401  0.511734  0.479551  1.000000  0.420931  0.204794
DeptAdv     0.390773  0.517577  0.328181  0.420931  1.000000  0.323199
RatePay     0.017644  0.221710  0.128123  0.204794  0.323199  1.000000
AnnLeave    0.063572  0.361152  0.250955  0.341432  0.254700  0.253388
PdHoliday   0.077053  0.206968  0.165677  0.160136  0.253121  0.215332

            AnnLeave  PdHoliday
SupPol      0.137811   0.120003
SupInf      0.118215   0.113906
SupFair     0.163292   0.126924
SupRec      0.155542   0.118270
SupCoop     0.236882   0.162969
SupResolve  0.163094   0.082722
SupTrn      0.063572   0.077053
DeptCom     0.361152   0.206968
```

```
DeptCond     0.250955    0.165677
DeptCoop     0.341432    0.160136
DeptAdv      0.254700    0.253121
RatePay      0.253388    0.215332
AnnLeave     1.000000    0.521737
PdHoliday    0.521737    1.000000
```

[27]:
```python
# import seaborn as sns
sns.set(rc={'figure.figsize':(12,4.5)})
ax = sns.heatmap(corr, xticklabels=corr.columns.values, annot=True,
                 yticklabels=corr.columns.values)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.show()
```



We see signicant correlation among the supervisor rating categories. The features of this data set fail the independence test. Also, the sample size is small given the number of predictors. Logistic regression is probably not appropriate here. Regardless, code will be sketched out for this for future analyses.

3.3 Feature Selection

[28]:
```python
print(oversampling)
```

```
False
```

[29]:
```python
# create frame to which to add feature importance results for various methods
importance_compare = pd.DataFrame(index = X_train.columns.values)
print(importance_compare)
```

```
Empty DataFrame
Columns: []
Index: [SupPol, SupInf, SupFair, SupRec, SupCoop, SupResolve, SupTrn, DeptCom,
DeptCond, DeptCoop, DeptAdv, RatePay, AnnLeave, PdHoliday]
```

[30]:
```
from boruta import BorutaPy
from sklearn.ensemble import RandomForestClassifier




max_it    = 100
perc_args = [50, 60, 70, 80, 90]
# perc_args = [90]

if (oversampling == False):
    for perc_arg in perc_args:
        rf = RandomForestClassifier(n_jobs=-1, max_depth=6)
        boruta_feature_selector = BorutaPy(rf, n_estimators='auto', verbose=0,␣
↪random_state=RAND_STATE,
                                           max_iter=max_it, perc=perc_arg)

        boruta_feature_selector.fit(X_train.values, y_train.values)

        selected = list()
        indexes = np.where(boruta_feature_selector.support_ == True)

        #print(f'selected size: {np.sum(boruta_feature_selector.support_)}')
        if (np.sum(boruta_feature_selector.support_ > 0)):
            for x in np.nditer(indexes):
                selected.append(features[x])

        tentative = list()
        indexes = np.where(boruta_feature_selector.support_weak_ == True)
        #print(f'tentatives size: {np.sum(boruta_feature_selector.
↪support_weak_)}')
        if (np.sum(boruta_feature_selector.support_weak_ > 0)):
            for x in np.nditer(indexes):
                tentative.append(features[x])

        print(f'\n#### Results for perc = {perc_arg}, max_iter = {max_it} ####')
        feat_ranks = pd.DataFrame(index = features)
        feat_ranks = feat_ranks.assign(Ranking = boruta_feature_selector.
↪ranking_)

        #feat_ranks = pd.DataFrame(index = features, {'Features': features,␣
↪'Ranking': boruta_feature_selector.ranking_})
        feat_ranks = feat_ranks.sort_values(by='Ranking')
```

```
        #feat_ranks = feat_ranks.sort_values(by='Ranking').
↪reset_index(drop=True)
        print(feat_ranks)

        print(f'\nSelected Features:')
        print(selected)
        print(f'\nTentative Features:')
        print(tentative)


        if (perc_arg == 90):
            importance_compare = importance_compare.assign(boruta_rank =␣
↪feat_ranks['Ranking'])
            print(importance_compare)
```

```
#### Results for perc = 50, max_iter = 100 ####
            Ranking
SupFair           1
SupRec            1
SupCoop           1
SupResolve        1
DeptCom           1
DeptCond          1
DeptCoop          1
DeptAdv           1
RatePay           1
SupTrn            2
PdHoliday         3
AnnLeave          4
SupPol            5
SupInf            5

Selected Features:
['SupFair', 'SupRec', 'SupCoop', 'SupResolve', 'DeptCom', 'DeptCond',
'DeptCoop', 'DeptAdv', 'RatePay']

Tentative Features:
['SupTrn']

#### Results for perc = 60, max_iter = 100 ####
            Ranking
SupFair           1
SupRec            1
```

```
SupResolve        1
DeptCom           1
DeptCond          1
DeptCoop          1
DeptAdv           1
RatePay           1
SupCoop           2
SupTrn            3
PdHoliday         4
SupPol            5
AnnLeave          5
SupInf            7

Selected Features:
['SupFair', 'SupRec', 'SupResolve', 'DeptCom', 'DeptCond', 'DeptCoop',
'DeptAdv', 'RatePay']

Tentative Features:
['SupCoop']

#### Results for perc = 70, max_iter = 100 ####
            Ranking
SupFair           1
SupRec            1
SupResolve        1
DeptCom           1
DeptCoop          1
DeptAdv           1
RatePay           2
DeptCond          3
SupCoop           4
PdHoliday         5
SupTrn            6
AnnLeave          6
SupPol            8
SupInf            9

Selected Features:
['SupFair', 'SupRec', 'SupResolve', 'DeptCom', 'DeptCoop', 'DeptAdv']

Tentative Features:
['RatePay']

#### Results for perc = 80, max_iter = 100 ####
            Ranking
SupFair           1
SupRec            1
DeptCom           1
```

```
DeptCoop          1
DeptAdv           1
SupResolve        2
RatePay           3
DeptCond          4
SupCoop           5
SupTrn            6
PdHoliday         7
SupPol            8
AnnLeave          8
SupInf           10

Selected Features:
['SupFair', 'SupRec', 'DeptCom', 'DeptCoop', 'DeptAdv']

Tentative Features:
['SupResolve']

#### Results for perc = 90, max_iter = 100 ####
            Ranking
SupRec            1
DeptCom           1
DeptCoop          1
DeptAdv           1
SupFair           2
SupResolve        3
RatePay           4
SupCoop           5
DeptCond          5
SupTrn            7
PdHoliday         8
AnnLeave          9
SupPol           10
SupInf           11

Selected Features:
['SupRec', 'DeptCom', 'DeptCoop', 'DeptAdv']

Tentative Features:
[]
            boruta_rank
SupPol               10
SupInf               11
SupFair               2
SupRec                1
SupCoop               5
SupResolve            3
SupTrn                7
```

```
DeptCom              1
DeptCond             5
DeptCoop             1
DeptAdv              1
RatePay              4
AnnLeave             9
PdHoliday            8
```

It may be of value to also examine whether or not the most important features for the older data are the same as the important features for the later data.

[31]:
```
# old vs new - the data is ordered, most recent observations first
emp_old = emp_exit.iloc[56:, :]
emp_new = emp_exit.iloc[:56, :]
X_old = emp_old.drop('rec', axis=1)
y_old = emp_old['rec']
X_new = emp_new.drop('rec', axis=1)
y_new = emp_new['rec']

max_it   = 80
perc_args = [60, 70, 80, 90]
#perc_args = [90]

print(emp_old['rec'].sum())
print(emp_new['rec'].sum())

old_yes_percent = emp_old['rec'].sum() / len(emp_old['rec'])
new_yes_percent = emp_new['rec'].sum() / len(emp_new['rec'])

print(f'Recommendation rate, prior to January 2019: {old_yes_percent}')
print(f'Recommendation rate, after January 2019: {new_yes_percent}')

if (oversampling == False):
    for perc_arg in perc_args:
        rf = RandomForestClassifier(n_jobs=-1, max_depth=6)
        boruta_feature_selector = BorutaPy(rf, n_estimators='auto', verbose=0,␣
 ↪random_state=RAND_STATE,
                                      max_iter=max_it, perc=perc_arg)

        boruta_feature_selector.fit(X_old.values, y_old.values)

        selected = list()
        indexes = np.where(boruta_feature_selector.support_ == True)

        #print(f'selected size: {np.sum(boruta_feature_selector.support_)}')
        if (np.sum(boruta_feature_selector.support_ > 0)):
            for x in np.nditer(indexes):
                selected.append(features[x])
```

```
        tentative = list()
        indexes = np.where(boruta_feature_selector.support_weak_ == True)
        #print(f'tentatives size: {np.sum(boruta_feature_selector.support_weak_)}')
        if (np.sum(boruta_feature_selector.support_weak_ > 0)):
            for x in np.nditer(indexes):
                tentative.append(features[x])

        print(f'\n#### Results for old records, perc = {perc_arg}, max_iter = {max_it} ####')
        feat_ranks = pd.DataFrame({'Features': features, 'Ranking': boruta_feature_selector.ranking_})
        feat_ranks = feat_ranks.sort_values(by='Ranking').reset_index(drop=True)
        print(feat_ranks)

        print(f'\nSelected Features:')
        print(selected)
        print(f'\nTentative Features:')
        print(tentative)


    for perc_arg in perc_args:
        rf = RandomForestClassifier(n_jobs=-1, max_depth=6)
        boruta_feature_selector = BorutaPy(rf, n_estimators='auto', verbose=0, random_state=RAND_STATE,
                                      max_iter=max_it, perc=perc_arg)

        boruta_feature_selector.fit(X_new.values, y_new.values)

        selected = list()
        indexes = np.where(boruta_feature_selector.support_ == True)

        #print(f'selected size: {np.sum(boruta_feature_selector.support_)}')
        if (np.sum(boruta_feature_selector.support_ > 0)):
            for x in np.nditer(indexes):
                selected.append(features[x])

        tentative = list()
        indexes = np.where(boruta_feature_selector.support_weak_ == True)
        #print(f'tentatives size: {np.sum(boruta_feature_selector.support_weak_)}')
        if (np.sum(boruta_feature_selector.support_weak_ > 0)):
            for x in np.nditer(indexes):
                tentative.append(features[x])
```

```
        print(f'\n#### Results for new records, perc = {perc_arg}, max_iter =␣
↪{max_it} ####')
        feat_ranks = pd.DataFrame({'Features': features, 'Ranking':␣
↪boruta_feature_selector.ranking_})
        feat_ranks = feat_ranks.sort_values(by='Ranking').reset_index(drop=True)
        print(feat_ranks)

        print(f'\nSelected Features:')
        print(selected)
        print(f'\nTentative Features:')
        print(tentative)
```

```
34
39
Recommendation rate, prior to January 2019: 0.6296296296296297
Recommendation rate, after January 2019: 0.6964285714285714

#### Results for old records, perc = 60, max_iter = 80 ####
      Features  Ranking
0       SupPol        1
1      SupFair        1
2       SupRec        1
3   SupResolve        1
4       SupTrn        1
5      DeptCom        1
6     DeptCond        1
7     DeptCoop        1
8      DeptAdv        1
9     AnnLeave        1
10   PdHoliday        1
11     RatePay        2
12     SupCoop        3
13      SupInf        4

Selected Features:
['SupPol', 'SupFair', 'SupRec', 'SupResolve', 'SupTrn', 'DeptCom', 'DeptCond',
'DeptCoop', 'DeptAdv', 'AnnLeave', 'PdHoliday']

Tentative Features:
[]

#### Results for old records, perc = 70, max_iter = 80 ####
      Features  Ranking
0       SupPol        1
```

```
1      SupFair        1
2       SupRec        1
3   SupResolve        1
4      DeptCom        1
5     DeptCond        1
6     DeptCoop        1
7      DeptAdv        1
8     AnnLeave        1
9    PdHoliday        1
10     RatePay        2
11     SupCoop        3
12      SupTrn        3
13      SupInf        5

Selected Features:
['SupPol', 'SupFair', 'SupRec', 'SupResolve', 'DeptCom', 'DeptCond', 'DeptCoop',
'DeptAdv', 'AnnLeave', 'PdHoliday']

Tentative Features:
[]

#### Results for old records, perc = 80, max_iter = 80 ####
      Features  Ranking
0      SupFair        1
1       SupRec        1
2   SupResolve        1
3      DeptCom        1
4     DeptCoop        1
5      DeptAdv        1
6     AnnLeave        1
7       SupPol        2
8     DeptCond        2
9    PdHoliday        2
10     RatePay        3
11     SupCoop        4
12      SupTrn        4
13      SupInf        6

Selected Features:
['SupFair', 'SupRec', 'SupResolve', 'DeptCom', 'DeptCoop', 'DeptAdv',
'AnnLeave']

Tentative Features:
['SupPol', 'DeptCond', 'PdHoliday']

#### Results for old records, perc = 90, max_iter = 80 ####
      Features  Ranking
0      SupFair        1
```

```
1       SupRec        1
2   SupResolve        1
3      DeptCom        1
4     DeptCoop        1
5      DeptAdv        1
6     AnnLeave        1
7       SupPol        2
8     DeptCond        2
9    PdHoliday        2
10      SupTrn        4
11     RatePay        4
12     SupCoop        6
13      SupInf        7

Selected Features:
['SupFair', 'SupRec', 'SupResolve', 'DeptCom', 'DeptCoop', 'DeptAdv',
'AnnLeave']

Tentative Features:
[]

#### Results for new records, perc = 60, max_iter = 80 ####
      Features  Ranking
0      DeptAdv        1
1      RatePay        1
2       SupRec        2
3      DeptCom        3
4     DeptCond        3
5     DeptCoop        4
6      SupFair        5
7       SupTrn        6
8     AnnLeave        7
9      SupCoop        9
10   PdHoliday        9
11      SupInf       11
12  SupResolve       11
13      SupPol       12

Selected Features:
['DeptAdv', 'RatePay']

Tentative Features:
['SupRec']

#### Results for new records, perc = 70, max_iter = 80 ####
      Features  Ranking
0      DeptAdv        1
1      RatePay        1
```

```
2       SupRec        2
3     DeptCond        2
4      DeptCom        3
5     DeptCoop        4
6       SupTrn        5
7     AnnLeave        6
8      SupCoop        7
9    PdHoliday        7
10     SupFair        9
11      SupInf       10
12  SupResolve       10
13      SupPol       12

Selected Features:
['DeptAdv', 'RatePay']

Tentative Features:
['SupRec', 'DeptCond']

#### Results for new records, perc = 80, max_iter = 80 ####
      Features  Ranking
0      DeptAdv        1
1      RatePay        2
2      DeptCom        3
3     DeptCond        4
4     DeptCoop        5
5       SupRec        6
6       SupTrn        7
7     AnnLeave        8
8      SupCoop        9
9    PdHoliday        9
10     SupFair       11
11      SupInf       12
12  SupResolve       12
13      SupPol       14

Selected Features:
['DeptAdv']

Tentative Features:
['RatePay']

#### Results for new records, perc = 90, max_iter = 80 ####
      Features  Ranking
0      DeptAdv        1
1      RatePay        2
2      DeptCom        3
3     DeptCond        3
```

```
4      DeptCoop        5
5        SupRec        6
6        SupTrn        7
7      AnnLeave        8
8       SupCoop        9
9     PdHoliday        9
10      SupFair       11
11       SupInf       12
12   SupResolve       12
13       SupPol       14

Selected Features:
['DeptAdv']

Tentative Features:
[]
```

[ ]:
```
# Hypothesis Test: has there been a change in the proportion of positive␣
↪recommendation ratings for the organization?

# $\pi_1 = $ proportion of people who would recommend the organization before␣
↪mid-January 2019

# $\pi_2 = $ proportion of people who would recommend the organization after␣
↪mid-January 2019

# $H_0 : \pi_1 - \pi_2 = 0$ No change in recommendation proportion

# $H_a : \pi_1 - \pi_2 \neq 0$ Recommendation proportion has changed
```

[32]:
```
# # z-test for difference in proportions is not appropriate here, because␣
↪sample size is
# # large in relation to population size, so before and after %s will be␣
↪compared directly.
# # However, possible code for situations like this might look something like:
# import math
# import scipy
# def z_prop(p1, n1, p2, n2):
#     Z = (p1 - p2) / (math.sqrt( (p1*(1-p1)/n1) + (p2*(1-p2)/n2) ))
#     # or use pooled p
#     p_value = 1 - ndtr(abs(Z))
#     p_scipy = scipy.stats.norm.sf(abs(Z))
#     print(f'n1(old n): {l_old}')
#     print(f'n2(new n): {l_new}')
#     print(f'p1(old p): {old_yes_percent}')
#     print(f'p2(new p): {new_yes_percent}')
#     print(f'Z-score  : {Z}')
```

```
#     print(f'p-value  : {p_value}')
#     print(f'p-scipy  : {p_scipy}')
#     return 1

# n1 = len(emp_old['rec'])
# n2 = len(emp_new['rec'])

# z_prop(old_yes_percent, n1, new_yes_percent, n2)
```

[33]:
```
# boruta feature selection
if (oversampling == False):
    # subset training and test sets to Boruta-selected, perc = 60
    X_train60 = X_train.reindex(columns=['SupFair', 'SupRec', 'SupResolve',␣
↪'DeptCom',
                              'DeptCond', 'DeptCoop', 'DeptAdv', 'RatePay'])
    X_test60  = X_test.reindex(columns=['SupFair', 'SupRec', 'SupResolve',␣
↪'DeptCom',
                             'DeptCond', 'DeptCoop', 'DeptAdv', 'RatePay'])


    # subset training and test sets to Boruta-selected, perc = 70
    X_train70 = X_train.reindex(columns=['SupFair', 'SupRec', 'SupResolve',␣
↪'DeptCom',
                              'DeptCoop', 'DeptAdv'])
    X_test70  = X_test.reindex(columns=['SupFair', 'SupRec', 'SupResolve',␣
↪'DeptCom',
                             'DeptCoop', 'DeptAdv'])


    # subset training and test sets to Boruta-selected, perc = 80
    X_train80 = X_train.reindex(columns=['SupFair', 'SupRec', 'DeptCom',␣
↪'DeptCoop',
                              'DeptAdv'])
    X_test80 = X_test.reindex(columns=['SupFair', 'SupRec', 'DeptCom',␣
↪'DeptCoop',
                            'DeptAdv'])

    #print(X_train60)
    #print(X_train70)
    #print(X_train80)

    # choose one; if none are selected, all features will be used to build␣
↪models
    #X_train, X_test = X_train60, X_test60
    #X_train, X_test = X_train70, X_test70
    #X_train, X_test = X_train80, X_test80
```

```
X_train.info()
X_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 88 entries, 98 to 101
Data columns (total 14 columns):
SupPol        88 non-null int32
SupInf        88 non-null int64
SupFair       88 non-null int64
SupRec        88 non-null int64
SupCoop       88 non-null int32
SupResolve    88 non-null int64
SupTrn        88 non-null int32
DeptCom       88 non-null int32
DeptCond      88 non-null int64
DeptCoop      88 non-null int32
DeptAdv       88 non-null int32
RatePay       88 non-null int64
AnnLeave      88 non-null int32
PdHoliday     88 non-null int32
dtypes: int32(8), int64(6)
memory usage: 7.6 KB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 22 entries, 102 to 70
Data columns (total 14 columns):
SupPol        22 non-null int32
SupInf        22 non-null int64
SupFair       22 non-null int64
SupRec        22 non-null int64
SupCoop       22 non-null int32
SupResolve    22 non-null int64
SupTrn        22 non-null int32
DeptCom       22 non-null int32
DeptCond      22 non-null int64
DeptCoop      22 non-null int32
DeptAdv       22 non-null int32
RatePay       22 non-null int64
AnnLeave      22 non-null int32
PdHoliday     22 non-null int32
dtypes: int32(8), int64(6)
memory usage: 1.9 KB
```

4 Model Creation and Evaluation

4.1 Model Evaluation

[34]:
```python
# obtain accuracy and classification report
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
# import matplotlib.pyplot as plt

model_results = pd.DataFrame(columns=['Model', 'Tuning', 'Scoring', 'TrainAcc',␣
↪'TestAcc', 'AUC'])

def evaluate_model(model, tuning, scoring, X_train, y_train, X_test, y_test,␣
↪model_type):
    global model_results
    test_predict = model.predict(X_test)
    train_predict = model.predict(X_train)
    acc_train = format(accuracy_score(y_train, train_predict), '.3f')
    acc_test  = format(accuracy_score(y_test, test_predict), '.3f')
    FP_rate, recall, thresholds = roc_curve(y_test, test_predict)
    roc_auc = round(auc(FP_rate, recall), 3)
    model_results = model_results.append({'Model' : model_type, 'Tuning' :␣
↪tuning, 'Scoring' : scoring,
                                    'TrainAcc' : acc_train, 'TestAcc' :␣
↪acc_test,
                                    'AUC' : roc_auc}, ignore_index=True)

    print(f'\n{model_type} Model Results:\nTraining Accuracy: {acc_train}    ␣
↪Test Accuracy: {acc_test}    AUC: {roc_auc}\n')

    # confusion matrix
    cf_mx = confusion_matrix(y_test, test_predict)
    print(cf_mx)
    plt.matshow(cf_mx)
    main_title = 'Confusion Matrix - ' + model_type + ' - ' + tuning + ' - ' +␣
↪scoring
    plt.title(main_title)
    plt.colorbar()
    plt.ylabel('True Value')
    plt.xlabel('Predicted Value')
    plt.show()


    # classification report
```

```
    main_title = '\nClassification Report - ' + model_type + ' - ' + tuning + '␣
↪- ' + scoring
    print(main_title)
    print('Train:')
    train_pred = model.predict(X_train)
    print(classification_report(y_train, train_predict))
    print('Test:')
    test_pred = model.predict(X_test)
    print(classification_report(y_test, test_predict))

    # roc
    sns.set(rc={'figure.figsize':(12,8)})
    main_title = 'ROC - ' + model_type + ' - ' + tuning + ' - ' + scoring
    plt.title(main_title)
    plt.plot(FP_rate, recall, 'b', label='AUC = %0.2f' % roc_auc)
    plt.legend(loc='lower right')
    plt.plot([0,1], [0,1], 'r--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.0])
    plt.ylabel('True Positive Rate (Recall)')
    plt.xlabel('False Positive Rate (Fall-Out)')
    plt.show()

    model_results = model_results.sort_values(by='AUC', ascending=False,␣
↪kind='mergesort').reset_index(drop=True).head(20)


def show_best_params(params, best_est):
    best_params = best_est.get_params()
    print(f'Best parameters: ')
    for param_name in sorted(params.keys()):
        print('%s: %r' % (param_name, best_params[param_name]))
```

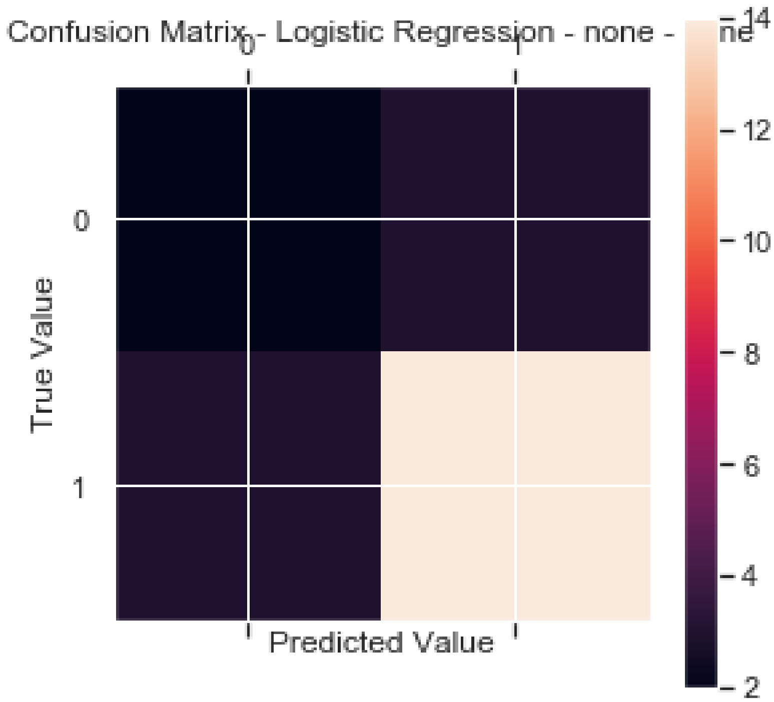
4.2 Logistic Regression

Several features are highly correlated, and the data set is small, so Logistic Regression may not be appropriate here. However, code for LR model will be sketched out for future use with other data sets and to establish a baseline accuracy to which other models can be compared.

[35]:
```
from sklearn.linear_model import LogisticRegression

model_type = 'Logistic Regression'
tuning = 'none'
scoring = 'none'
```

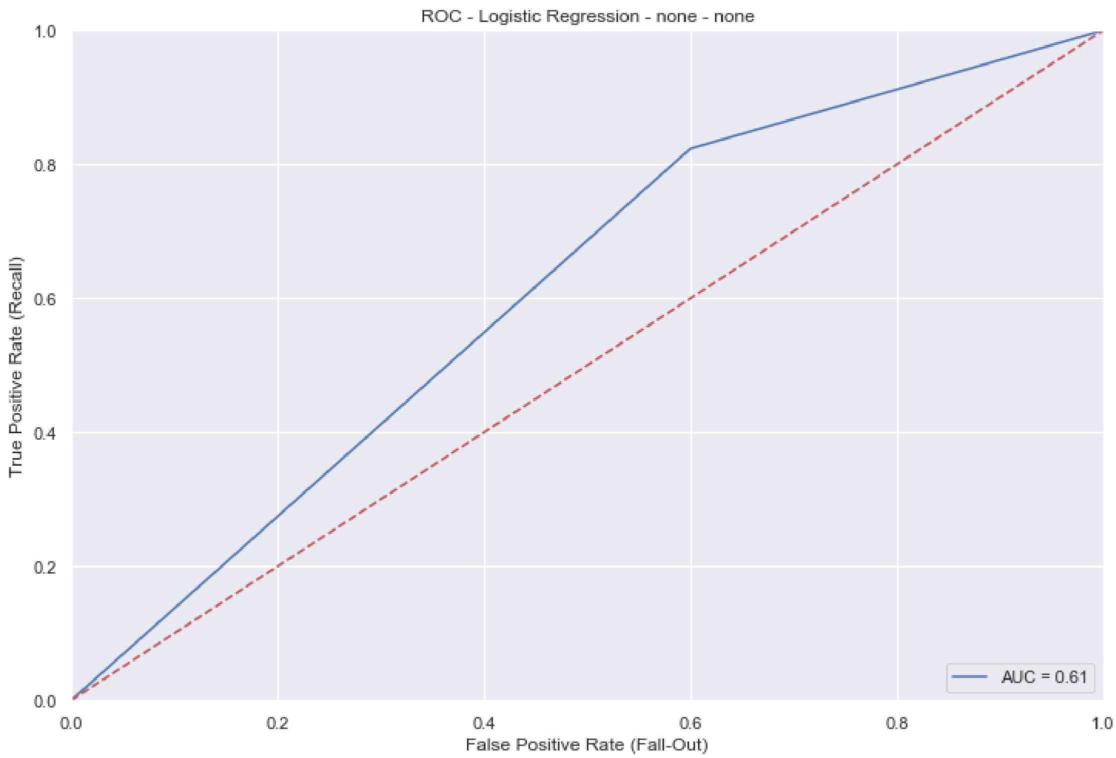
```
logmodel = LogisticRegression(solver='liblinear') # will try liblinear for␣
↪small data sets
logmodel.fit(X_train, y_train)
evaluate_model(logmodel, tuning, scoring, X_train, y_train, X_test, y_test,␣
↪model_type)
```

```
Logistic Regression Model Results:
Training Accuracy: 0.750    Test Accuracy: 0.727    AUC: 0.612

[[ 2  3]
 [ 3 14]]
```




```
Classification Report - Logistic Regression - none - none
Train:
              precision    recall  f1-score   support

           0       0.71      0.53      0.61        32
           1       0.77      0.88      0.82        56

    accuracy                           0.75        88
   macro avg       0.74      0.70      0.71        88
weighted avg       0.74      0.75      0.74        88
```

```
Test:
              precision    recall  f1-score   support

           0       0.40      0.40      0.40         5
           1       0.82      0.82      0.82        17

    accuracy                           0.73        22
   macro avg       0.61      0.61      0.61        22
weighted avg       0.73      0.73      0.73        22
```



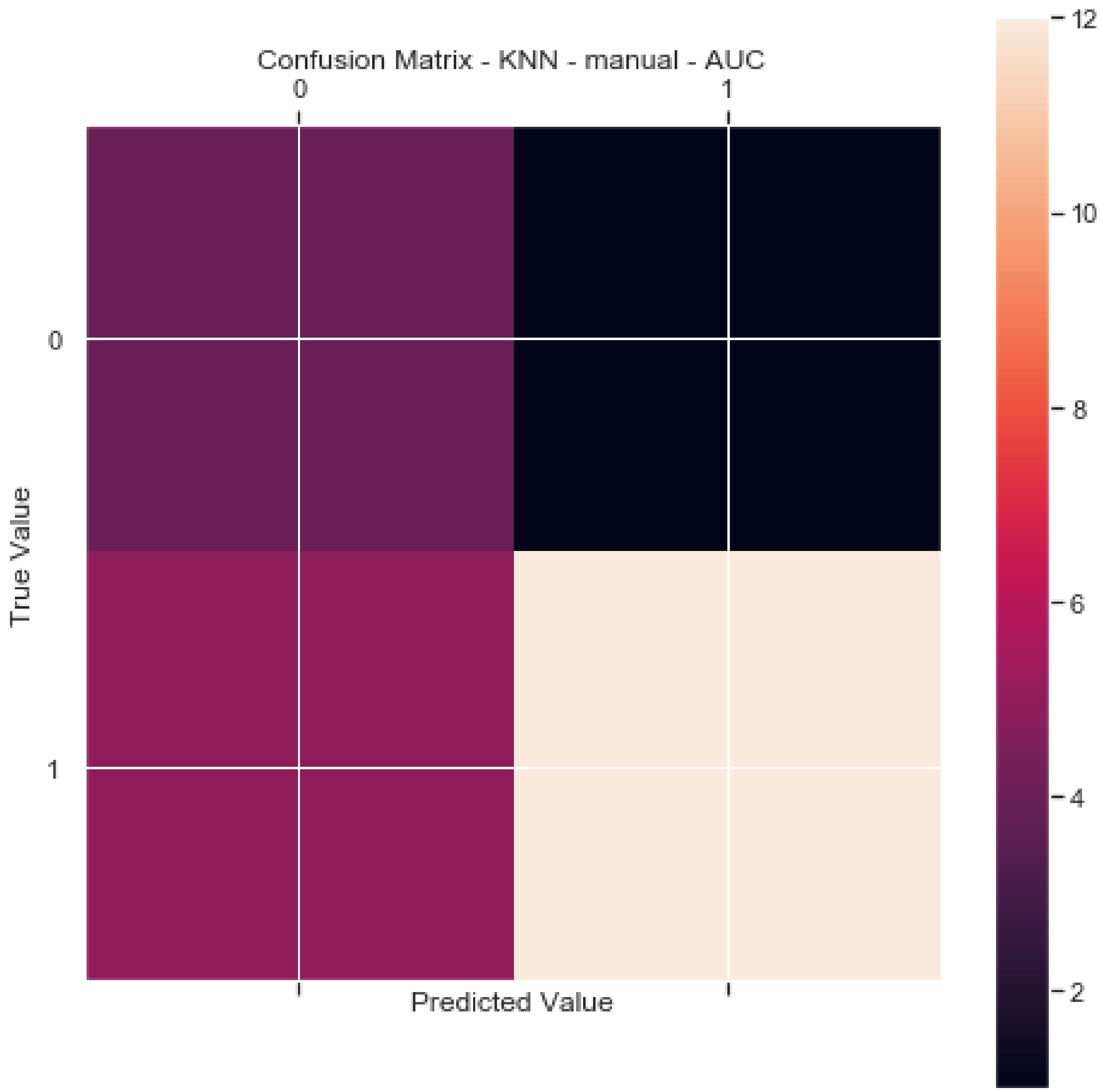
4.3 K Nearest Neighbors

[36]:
```
# create and evaluate KNN model; manual tuning

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score

model_type = "KNN"
tuning = 'manual'
```

```python
scoring = 'AUC'
kmin = 3
kmax = 7

knn_results = pd.DataFrame(columns=['K', 'TrainAcc', 'TestAcc', 'AUC'])

for K in range(kmin, kmax + 1):
    knn_model = KNeighborsClassifier(n_neighbors=K)
    knn_model.fit(X_train, y_train)
    knn_train_pred = knn_model.predict(X_train)
    knn_test_pred = knn_model.predict(X_test)
    acc_train = format(accuracy_score(y_train, knn_train_pred), '.3f')
    acc_test  = format(accuracy_score(y_test, knn_test_pred), '.3f')
    FP_rate, recall, thresholds = roc_curve(y_test, knn_test_pred)
    knn_roc_auc = auc(FP_rate, recall)
    knn_results = knn_results.append({'K' : K, 'TrainAcc' : acc_train,
                               'TestAcc' : acc_test, 'AUC' : knn_roc_auc},␣
 ↪ignore_index=True)

print('Model Evaluation ' + model_type + ' - ' + tuning + ' - ' + scoring)
print(knn_results)

# best KNN model
auc_max_id = knn_results['AUC'].idxmax()
best_K   = knn_results.iloc[auc_max_id, 0]
print(f'\nBest KNN model parameters:  K = {best_K}')
knn_best_model_acc  = format(accuracy_score(y_test, knn_test_pred), '.3f')

knn_model = KNeighborsClassifier(n_neighbors=best_K)
knn_model.fit(X_train, y_train)
knn_test_pred = knn_model.predict(X_test)

evaluate_model(knn_model, tuning, scoring, X_train, y_train, X_test, y_test,␣
 ↪model_type)
```

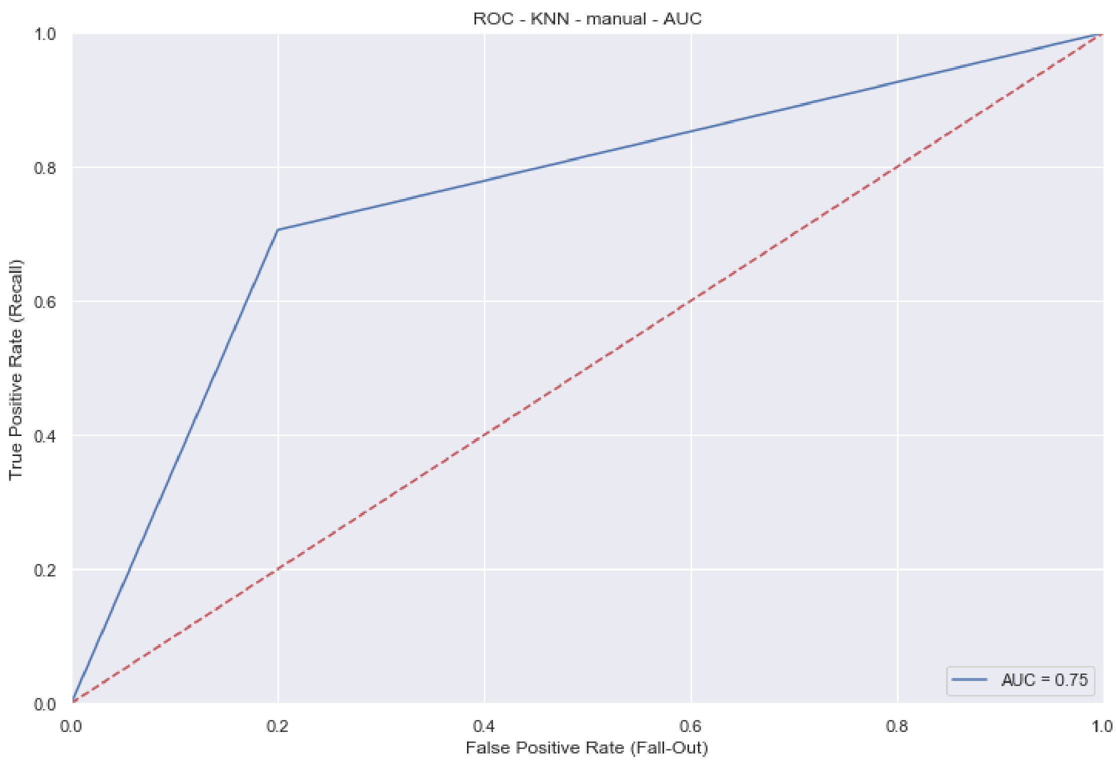
```
Model Evaluation KNN - manual - AUC
   K TrainAcc TestAcc       AUC
0  3    0.898   0.727  0.611765
1  4    0.830   0.727  0.752941
2  5    0.818   0.682  0.511765
3  6    0.795   0.727  0.682353
4  7    0.761   0.682  0.511765

Best KNN model parameters:  K = 4

KNN Model Results:
Training Accuracy: 0.830    Test Accuracy: 0.727    AUC: 0.753
```

```
[[ 4  1]
 [ 5 12]]
```




```
Classification Report - KNN - manual - AUC
Train:
              precision    recall  f1-score   support

           0       0.76      0.78      0.77        32
           1       0.87      0.86      0.86        56

    accuracy                           0.83        88
   macro avg       0.82      0.82      0.82        88
```

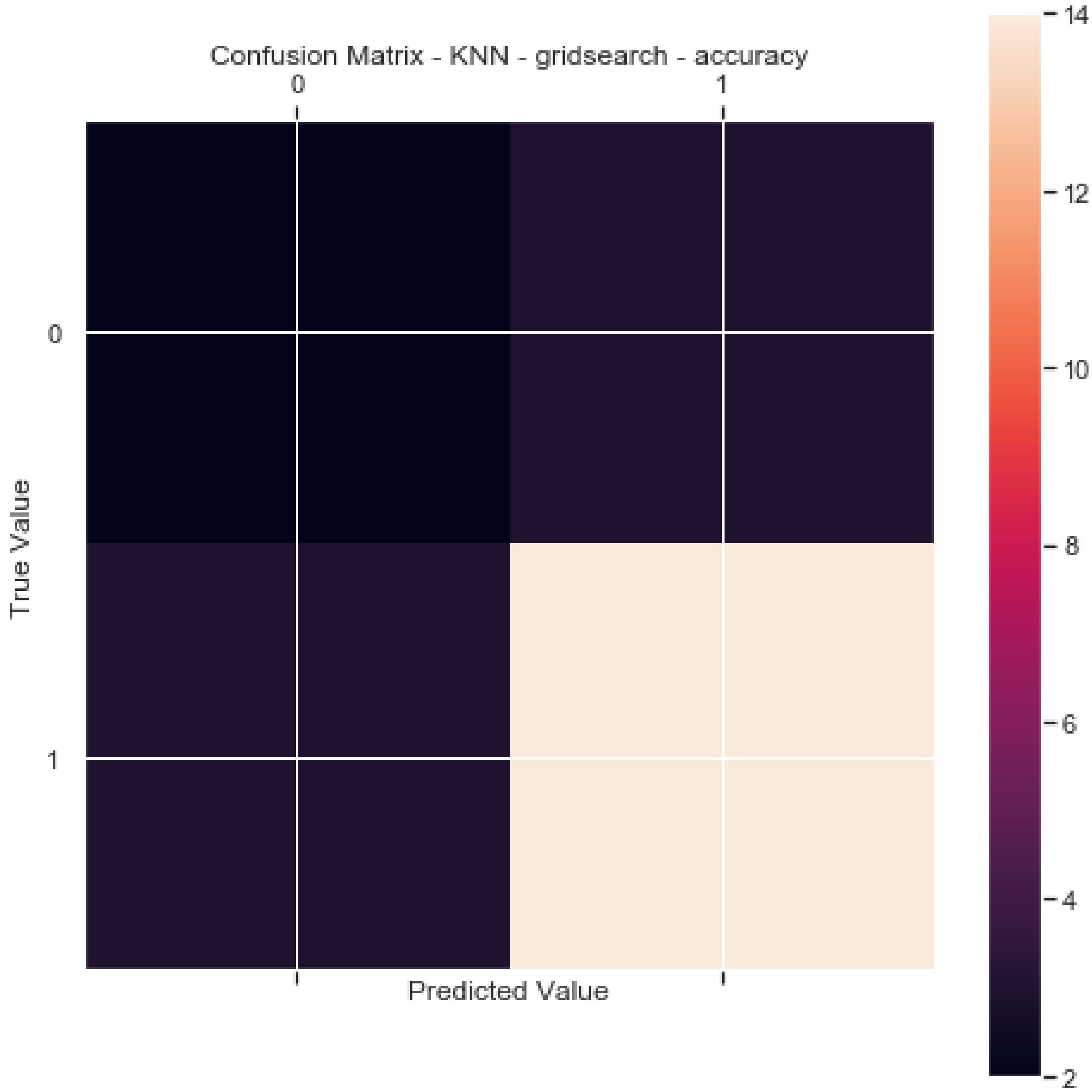
```
weighted avg       0.83      0.83      0.83        88

Test:
              precision    recall  f1-score   support

           0       0.44      0.80      0.57         5
           1       0.92      0.71      0.80        17

    accuracy                           0.73        22
   macro avg       0.68      0.75      0.69        22
weighted avg       0.81      0.73      0.75        22
```



[37]:
```
# KNN with GridSearch hyperparametric tuning
from sklearn.model_selection import GridSearchCV
grid_param = {'n_neighbors': range(3,8)}
tuning = 'gridsearch'
scoring = 'accuracy'
if __name__ == '__main__':
    model_type = 'KNN'
    knn_model_grid = KNeighborsClassifier()
    grid_search = GridSearchCV(estimator=knn_model_grid, param_grid=grid_param,
                               scoring = 'accuracy', cv=5, n_jobs=-1, iid=False)
```

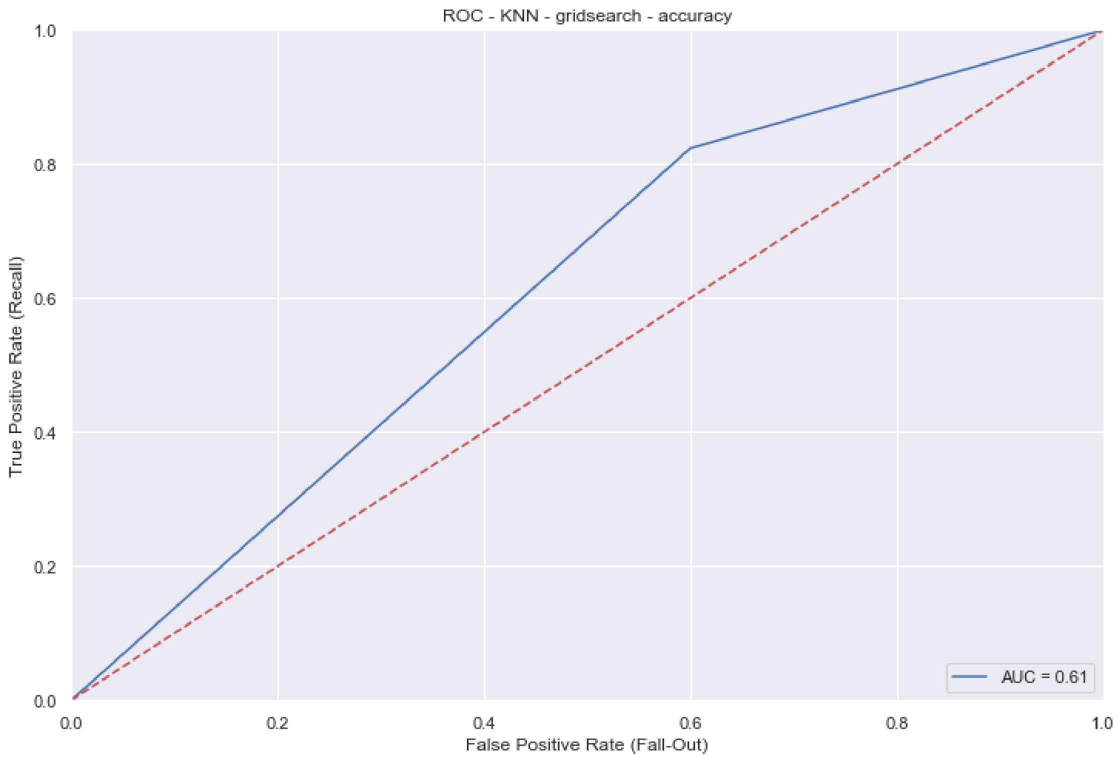
```
    grid_search.fit(X_train, y_train)
    knn_model_gs = grid_search.best_estimator_

    show_best_params(grid_param, grid_search.best_estimator_)
    evaluate_model(knn_model_gs, tuning, scoring, X_train, y_train, X_test,␣
↪y_test, model_type)
```

```
Best parameters:
n_neighbors: 3

KNN Model Results:
Training Accuracy: 0.898    Test Accuracy: 0.727    AUC: 0.612

[[ 2  3]
 [ 3 14]]
```




```
Classification Report - KNN - gridsearch - accuracy
Train:
              precision    recall  f1-score   support

           0       0.93      0.78      0.85        32
           1       0.89      0.96      0.92        56

    accuracy                           0.90        88
   macro avg       0.91      0.87      0.89        88
weighted avg       0.90      0.90      0.90        88

Test:
```

```
              precision    recall  f1-score   support

           0       0.40      0.40      0.40         5
           1       0.82      0.82      0.82        17

    accuracy                           0.73        22
   macro avg       0.61      0.61      0.61        22
weighted avg       0.73      0.73      0.73        22
```

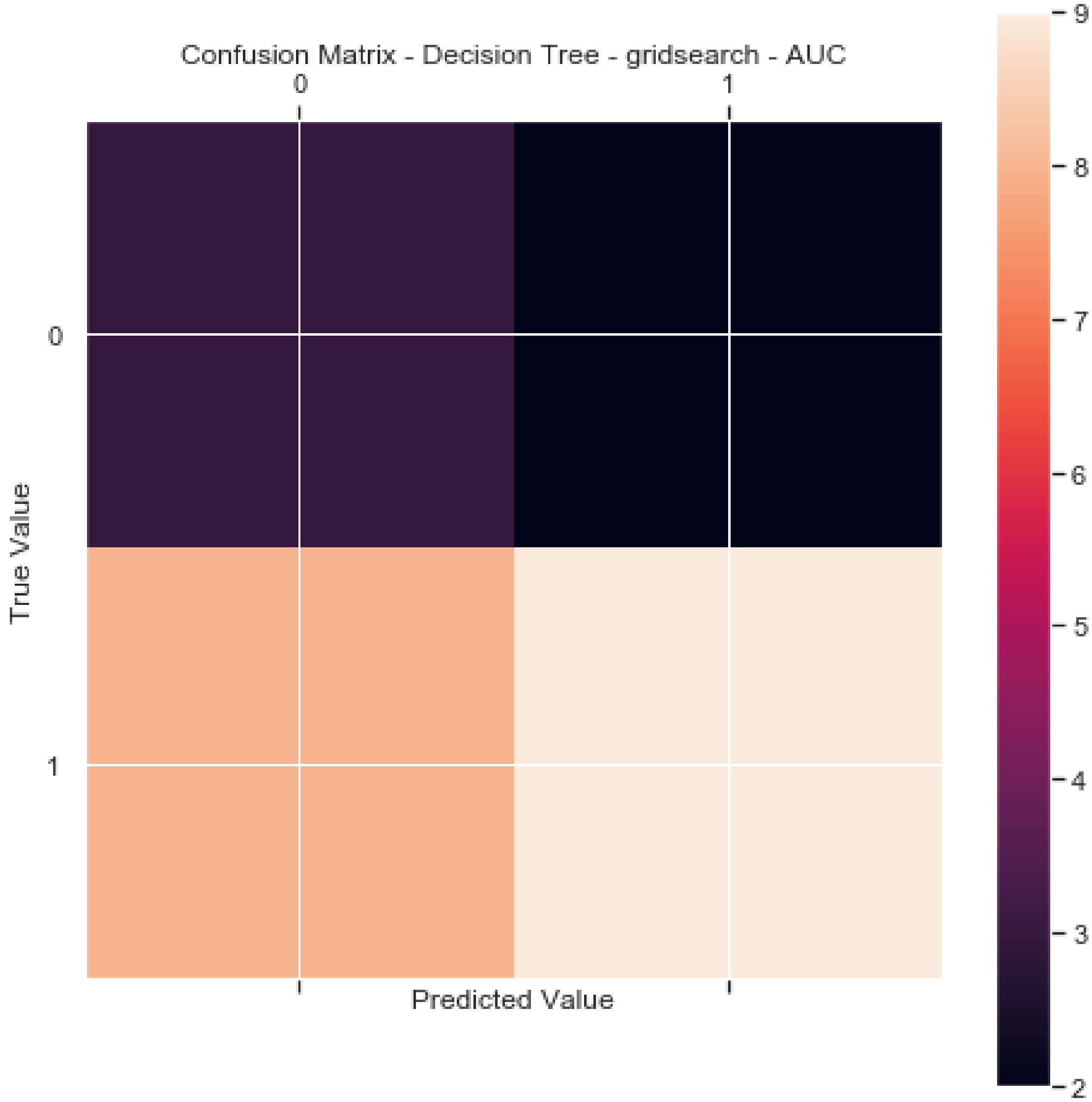


4.4 Decision Tree

[38]:
```python
# Decision Tree - gridsearch - best AUC
from sklearn.tree import DecisionTreeClassifier
# from sklearn.model_selection import train_test_split
# from sklearn.metrics import classification_report
from sklearn.pipeline import Pipeline
# from sklearn.model_selection import GridSearchCV

if __name__ == '__main__':
    model_type = "Decision Tree"
    tuning = 'gridsearch'
    scoring = 'AUC'
```

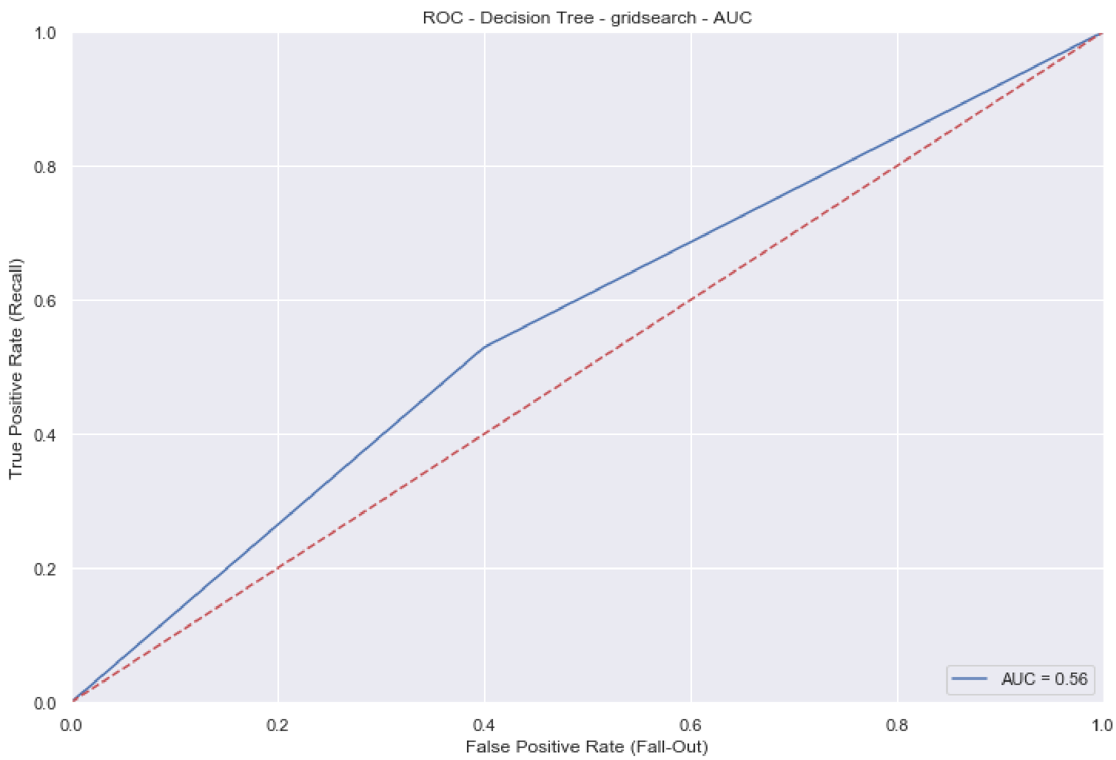
```
    max_depths = range(2, 15)
    min_samp_spl = range(2, 4)
    min_samp_leaf = range(1, 4)

    pipeline = Pipeline([('dt_model',␣
↪DecisionTreeClassifier(criterion='entropy'))])
    parameters = { 'dt_model__max_depth': max_depths,
                  'dt_model__min_samples_split': min_samp_spl,
                  'dt_model__min_samples_leaf': min_samp_leaf
                  }

    grid_search = GridSearchCV(pipeline, parameters, n_jobs=-1, verbose=1, cv=5,
                               scoring='roc_auc', refit=True, iid=False)
    grid_search.fit(X_train, y_train)

    dt_best_auc = grid_search.best_estimator_

    show_best_params(parameters, grid_search.best_estimator_)
    evaluate_model(dt_best_auc, tuning, scoring, X_train, y_train, X_test,␣
↪y_test, model_type)
```

```
Fitting 5 folds for each of 78 candidates, totalling 390 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 20 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:    0.1s

Best parameters:
dt_model__max_depth: 3
dt_model__min_samples_leaf: 1
dt_model__min_samples_split: 2

Decision Tree Model Results:
Training Accuracy: 0.773    Test Accuracy: 0.545    AUC: 0.565

[[3 2]
 [8 9]]

[Parallel(n_jobs=-1)]: Done 390 out of 390 | elapsed:    0.4s finished
```




```
Classification Report - Decision Tree - gridsearch - AUC
Train:
              precision    recall  f1-score   support

           0       0.68      0.72      0.70        32
           1       0.83      0.80      0.82        56

    accuracy                           0.77        88
   macro avg       0.75      0.76      0.76        88
weighted avg       0.78      0.77      0.77        88

Test:
```

```
              precision    recall  f1-score   support

           0       0.27      0.60      0.37         5
           1       0.82      0.53      0.64        17

    accuracy                           0.55        22
   macro avg       0.55      0.56      0.51        22
weighted avg       0.69      0.55      0.58        22
```

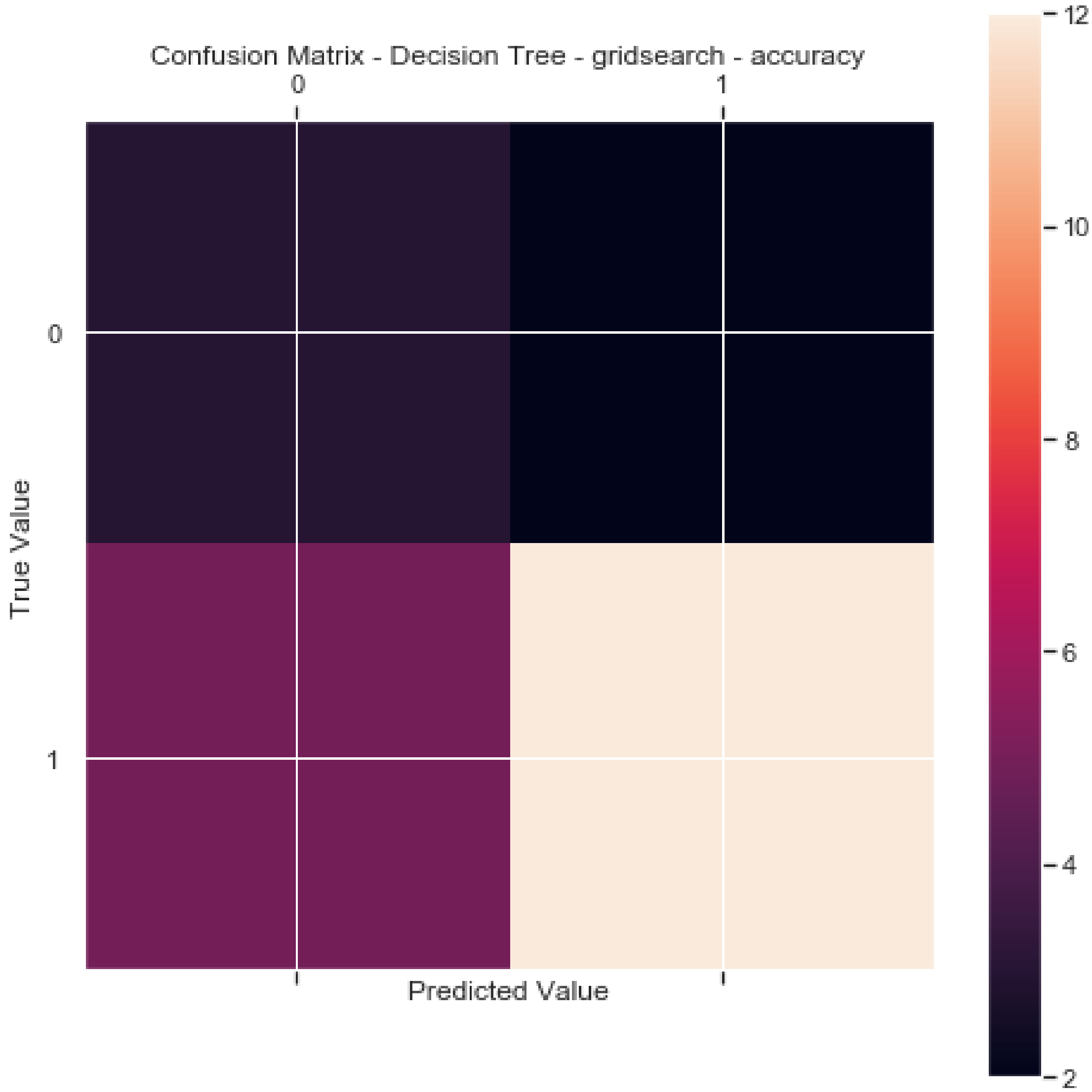


[39]:

```
# Decision Tree - gridsearch - most accurate
from sklearn.tree import DecisionTreeClassifier
# from sklearn.model_selection import train_test_split
# from sklearn.metrics import classification_report
from sklearn.pipeline import Pipeline
# from sklearn.model_selection import GridSearchCV

if __name__ == '__main__':
    model_type = "Decision Tree"
    tuning = 'gridsearch'
    scoring = 'accuracy'
    max_depths = range(2, 15)
    min_samp_spl = range(2, 4)
```

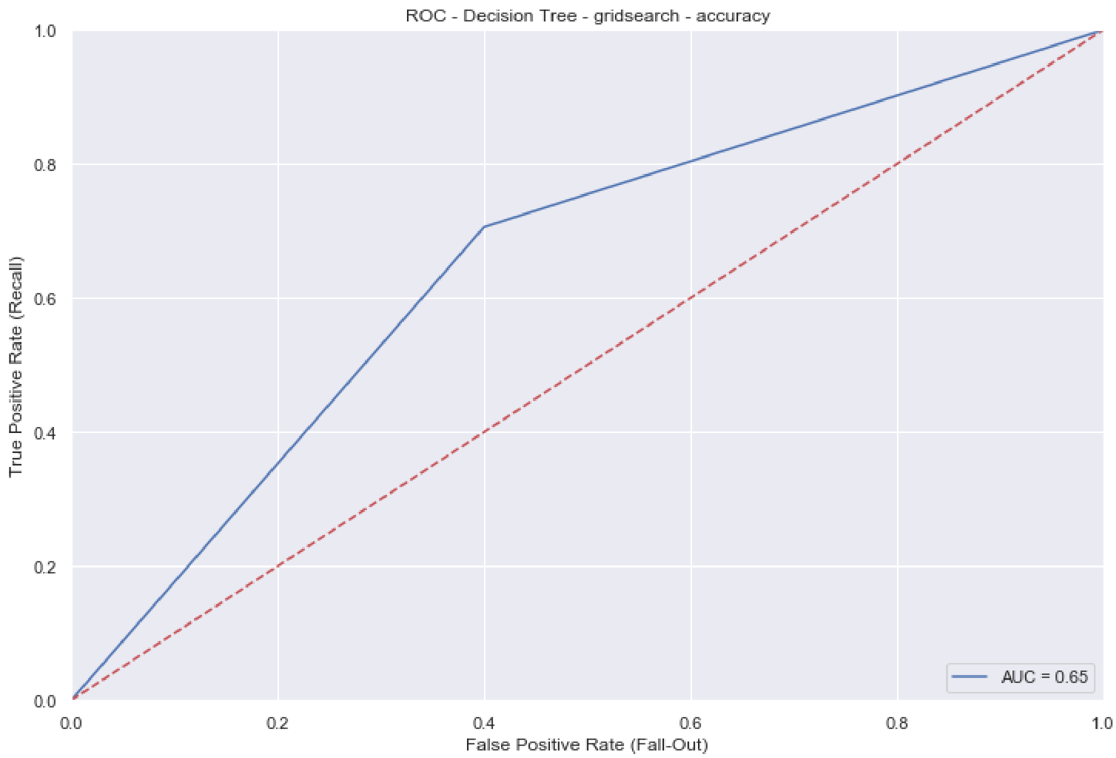
```
    min_samp_leaf = range(1, 4)

    pipeline = Pipeline([('dt_model',␣
↪DecisionTreeClassifier(criterion='entropy'))])
    parameters = { 'dt_model__max_depth': max_depths,
                   'dt_model__min_samples_split': min_samp_spl,
                   'dt_model__min_samples_leaf': min_samp_leaf
                   }

    grid_search = GridSearchCV(pipeline, parameters, n_jobs=-1, verbose=1, cv=5,
                               scoring='accuracy', iid=False)
    grid_search.fit(X_train, y_train)

    dt_most_acc = grid_search.best_estimator_

    show_best_params(parameters, grid_search.best_estimator_)
    evaluate_model(dt_most_acc, tuning, scoring, X_train, y_train, X_test,␣
↪y_test, model_type)
```

```
Fitting 5 folds for each of 78 candidates, totalling 390 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 20 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:    0.1s

Best parameters:
dt_model__max_depth: 3
dt_model__min_samples_leaf: 3
dt_model__min_samples_split: 3

Decision Tree Model Results:
Training Accuracy: 0.761    Test Accuracy: 0.682    AUC: 0.653

[[ 3  2]
 [ 5 12]]

[Parallel(n_jobs=-1)]: Done 390 out of 390 | elapsed:    0.4s finished
```




```
Classification Report - Decision Tree - gridsearch - accuracy
Train:
              precision    recall  f1-score   support

           0       0.67      0.69      0.68        32
           1       0.82      0.80      0.81        56

    accuracy                           0.76        88
   macro avg       0.74      0.75      0.74        88
weighted avg       0.76      0.76      0.76        88

Test:
```

```
              precision    recall  f1-score   support

           0       0.38      0.60      0.46         5
           1       0.86      0.71      0.77        17

    accuracy                           0.68        22
   macro avg       0.62      0.65      0.62        22
weighted avg       0.75      0.68      0.70        22
```



Results for Decision Tree are wildly volatile.

[40]:
```
#  best decision tree
# from sklearn.tree import export_graphviz
# import graphviz
# import pydot

# dt_most_acc = DecisionTreeClassifier(criterion='entropy', max_depth=8,␣
↪min_samples_leaf=3, min_samples_split=2)

# dt_most_acc.fit(X_train, y_train)

# export_graphviz(dt_most_acc, out_file="file.dot")
```

```
# with open("file.dot") as f:
#     dot_graph = f.read()
# graphviz.Source(dot_graph)

# (graph,) = pydot.graph_from_dot_file('file.dot')
# graph.write_png('tree.png')
```

[41]:
```
# feature importances (Decision Tree class) for best model

if (oversampling == False):

    best_params = grid_search.best_estimator_.get_params()
    for param_name in sorted(parameters.keys()):
        print('%s: %r' % (param_name, best_params[param_name]))

    print(type(X_train))
    best_max_depth = best_params['dt_model__max_depth']
    best_min_samp_spl = best_params['dt_model__min_samples_split']
    best_min_samp_lf = best_params['dt_model__min_samples_leaf']

    dt_most_acc = DecisionTreeClassifier(max_depth = best_max_depth,
                                         min_samples_split = best_min_samp_spl,
                                         min_samples_leaf = best_min_samp_lf)

    dt_most_acc.fit(X_train, y_train)

    dt_importances = pd.DataFrame(data=dt_most_acc.feature_importances_, index␣
↪= X_train.columns,
                                  columns=['importance'])
    dt_importances = dt_importances.sort_values(by='importance',␣
↪ascending=False, kind='mergesort')
    print(dt_importances)
```

```
dt_model__max_depth: 3
dt_model__min_samples_leaf: 3
dt_model__min_samples_split: 3
<class 'pandas.core.frame.DataFrame'>
            importance
DeptCom       0.411322
SupRec        0.230962
RatePay       0.142935
SupCoop       0.128651
DeptCoop      0.086129
SupPol        0.000000
SupInf        0.000000
SupFair       0.000000
SupResolve    0.000000
```

```
SupTrn        0.000000
DeptCond      0.000000
DeptAdv       0.000000
AnnLeave      0.000000
PdHoliday     0.000000
```

[42]:
```
importance_compare = importance_compare.assign(dt_importance =␣
↪dt_importances['importance'])
# print(importance_compare)
```

4.5 Random Forest

[43]:
```
# random forest - manual loop

# from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
# from sklearn.metrics import classification_report
# from sklearn.metrics import roc_curve, auc
from sklearn.metrics import accuracy_score

# parameters:  number of estimators (trees), maximum depth
estims = [10, 20, 40, 80, 150, 200, 250, 300, 400]
max_depths = range(8, 14)


rf_results = pd.DataFrame(columns=['Trees', 'MaxDepth', 'TrainAcc', 'TestAcc',␣
↪'AUC'])

for est in estims:
    for max_d in max_depths:
        rf_model = RandomForestClassifier(n_estimators=est, max_depth = max_d,
                                          random_state=RAND_STATE)
        rf_model.fit(X_train, y_train)
        rf_train_pred = rf_model.predict(X_train)
        rf_test_pred = rf_model.predict(X_test)
        acc_train = format(accuracy_score(y_train, rf_train_pred), '.3f')
        acc_test  = format(accuracy_score(y_test, rf_test_pred), '.3f')
        FP_rate, recall, thresholds = roc_curve(y_test, rf_test_pred)
        rf_roc_auc = auc(FP_rate, recall)
        rf_results = rf_results.append({'Trees' : int(est), 'MaxDepth' : max_d,
                                   'TrainAcc' : acc_train, 'TestAcc' :␣
↪acc_test,
                                   'AUC' : rf_roc_auc}, ignore_index=True)
        print(f'Forest with {est} trees, max_depth = {max_d} completed.')

rf_results = rf_results.assign(Estimators = rf_results.Trees.astype(int))
rf_results = rf_results.assign(MaxDepth = rf_results.MaxDepth.astype(int))
```

```python
rf_results = rf_results.drop('Trees', axis=1)
rf_results.columns.tolist()
```

```
Forest with 10 trees, max_depth = 8 completed.
Forest with 10 trees, max_depth = 9 completed.
Forest with 10 trees, max_depth = 10 completed.
Forest with 10 trees, max_depth = 11 completed.
Forest with 10 trees, max_depth = 12 completed.
Forest with 10 trees, max_depth = 13 completed.
Forest with 20 trees, max_depth = 8 completed.
Forest with 20 trees, max_depth = 9 completed.
Forest with 20 trees, max_depth = 10 completed.
Forest with 20 trees, max_depth = 11 completed.
Forest with 20 trees, max_depth = 12 completed.
Forest with 20 trees, max_depth = 13 completed.
Forest with 40 trees, max_depth = 8 completed.
Forest with 40 trees, max_depth = 9 completed.
Forest with 40 trees, max_depth = 10 completed.
Forest with 40 trees, max_depth = 11 completed.
Forest with 40 trees, max_depth = 12 completed.
Forest with 40 trees, max_depth = 13 completed.
Forest with 80 trees, max_depth = 8 completed.
Forest with 80 trees, max_depth = 9 completed.
Forest with 80 trees, max_depth = 10 completed.
Forest with 80 trees, max_depth = 11 completed.
Forest with 80 trees, max_depth = 12 completed.
Forest with 80 trees, max_depth = 13 completed.
Forest with 150 trees, max_depth = 8 completed.
Forest with 150 trees, max_depth = 9 completed.
Forest with 150 trees, max_depth = 10 completed.
Forest with 150 trees, max_depth = 11 completed.
Forest with 150 trees, max_depth = 12 completed.
Forest with 150 trees, max_depth = 13 completed.
Forest with 200 trees, max_depth = 8 completed.
Forest with 200 trees, max_depth = 9 completed.
Forest with 200 trees, max_depth = 10 completed.
Forest with 200 trees, max_depth = 11 completed.
Forest with 200 trees, max_depth = 12 completed.
Forest with 200 trees, max_depth = 13 completed.
Forest with 250 trees, max_depth = 8 completed.
Forest with 250 trees, max_depth = 9 completed.
Forest with 250 trees, max_depth = 10 completed.
Forest with 250 trees, max_depth = 11 completed.
Forest with 250 trees, max_depth = 12 completed.
Forest with 250 trees, max_depth = 13 completed.
Forest with 300 trees, max_depth = 8 completed.
Forest with 300 trees, max_depth = 9 completed.
```

```
Forest with 300 trees, max_depth = 10 completed.
Forest with 300 trees, max_depth = 11 completed.
Forest with 300 trees, max_depth = 12 completed.
Forest with 300 trees, max_depth = 13 completed.
Forest with 400 trees, max_depth = 8 completed.
Forest with 400 trees, max_depth = 9 completed.
Forest with 400 trees, max_depth = 10 completed.
Forest with 400 trees, max_depth = 11 completed.
Forest with 400 trees, max_depth = 12 completed.
Forest with 400 trees, max_depth = 13 completed.
```

[43]:
```
['MaxDepth', 'TrainAcc', 'TestAcc', 'AUC', 'Estimators']
```

[44]:
```
rf_results = rf_results[['MaxDepth', 'Estimators', 'TrainAcc', 'TestAcc',␣
 ↪'AUC']]
rf_results.sort_values(by='TestAcc', ascending=False, kind='mergesort').head(20)
```

[44]:

	MaxDepth	Estimators	TrainAcc	TestAcc	AUC
42	8	300	1.000	0.864	0.841176
48	8	400	1.000	0.864	0.841176
49	9	400	1.000	0.864	0.841176
12	8	40	1.000	0.818	0.741176
13	9	40	1.000	0.818	0.741176
14	10	40	1.000	0.818	0.741176
15	11	40	1.000	0.818	0.741176
18	8	80	1.000	0.818	0.811765
19	9	80	1.000	0.818	0.811765
20	10	80	1.000	0.818	0.811765
21	11	80	1.000	0.818	0.811765
22	12	80	1.000	0.818	0.811765
23	13	80	1.000	0.818	0.811765
24	8	150	1.000	0.818	0.811765
25	9	150	1.000	0.818	0.811765
26	10	150	1.000	0.818	0.811765
27	11	150	1.000	0.818	0.811765
28	12	150	1.000	0.818	0.811765
29	13	150	1.000	0.818	0.811765
30	8	200	1.000	0.818	0.811765

[45]:
```
rf_results.sort_values(by='AUC', ascending=False, kind='mergesort').head(20)
```

[45]:

	MaxDepth	Estimators	TrainAcc	TestAcc	AUC
42	8	300	1.000	0.864	0.841176
48	8	400	1.000	0.864	0.841176
49	9	400	1.000	0.864	0.841176
18	8	80	1.000	0.818	0.811765
19	9	80	1.000	0.818	0.811765

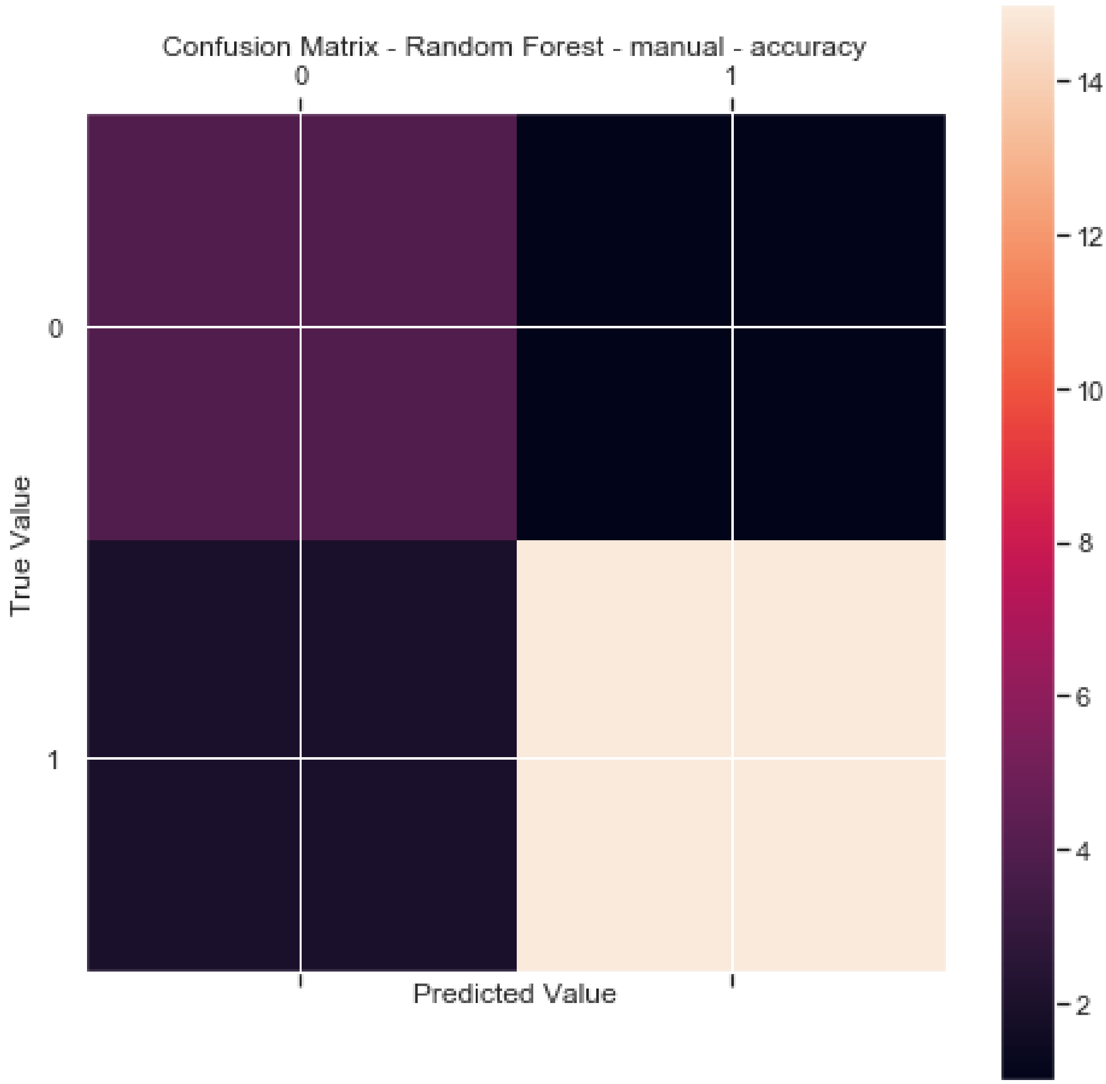
```
20        10        80    1.000    0.818  0.811765
21        11        80    1.000    0.818  0.811765
22        12        80    1.000    0.818  0.811765
23        13        80    1.000    0.818  0.811765
24         8       150    1.000    0.818  0.811765
25         9       150    1.000    0.818  0.811765
26        10       150    1.000    0.818  0.811765
27        11       150    1.000    0.818  0.811765
28        12       150    1.000    0.818  0.811765
29        13       150    1.000    0.818  0.811765
30         8       200    1.000    0.818  0.811765
31         9       200    1.000    0.818  0.811765
32        10       200    1.000    0.818  0.811765
33        11       200    1.000    0.818  0.811765
34        12       200    1.000    0.818  0.811765
```

[46]:

```
from sklearn.model_selection import cross_val_score
rf_results = rf_results.assign(TestAcc = rf_results.TestAcc.astype(float))
acc_max_id = rf_results['TestAcc'].idxmax()

best_MaxDepth   = rf_results.iloc[acc_max_id, 0]
best_Estimators = rf_results.iloc[acc_max_id, 1]
print(f'Best Parameters:  \nMaxDepth = {best_MaxDepth}            Estimators =␣
↪{best_Estimators}')

model_type = 'Random Forest'
tuning = 'manual'
scoring = 'accuracy'
rf_most_acc = RandomForestClassifier(n_estimators=best_Estimators,␣
↪max_depth=best_MaxDepth,
                                       random_state=RAND_STATE)

rf_most_acc.fit(X_train,y_train)
predictions = rf_most_acc.predict(X_test)
FP_rate, recall, thresholds = roc_curve(y_test, predictions)

#train_scores = format(cross_val_score(rf_most_acc, X_train, y_train, cv=5).
↪mean(), '.3f')
#test_scores = format(cross_val_score(rf_most_acc, X_test, y_test, cv=5).
↪mean(), '.3f')
#print(f'Cross Validation:       Train Accuracy = {train_scores}  Test Accuracy␣
↪= {test_scores}')

evaluate_model(rf_most_acc, tuning, scoring, X_train, y_train, X_test, y_test,␣
↪model_type)
```

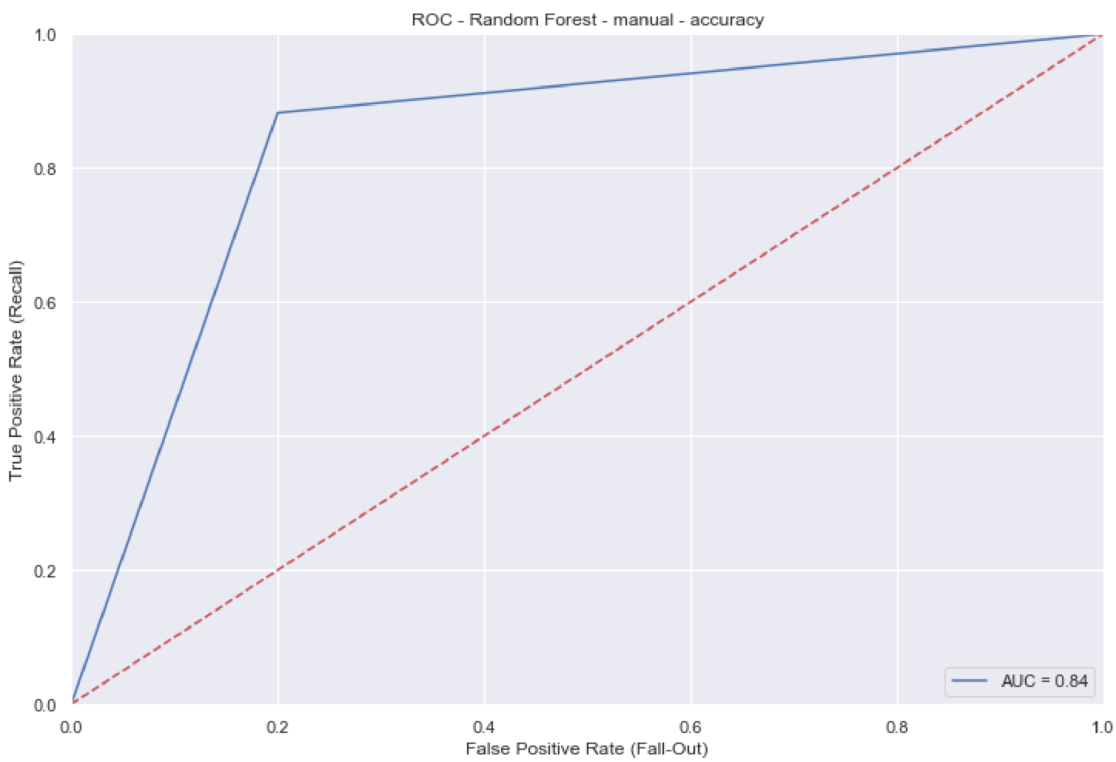
```
Best Parameters:
```

```
MaxDepth = 8            Estimators = 300

Random Forest Model Results:
Training Accuracy: 1.000    Test Accuracy: 0.864    AUC: 0.841

[[ 4  1]
 [ 2 15]]
```




```
Classification Report - Random Forest - manual - accuracy
Train:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        32
```

```
           1       1.00      1.00      1.00        56

    accuracy                           1.00        88
   macro avg       1.00      1.00      1.00        88
weighted avg       1.00      1.00      1.00        88

Test:
              precision    recall  f1-score   support

           0       0.67      0.80      0.73         5
           1       0.94      0.88      0.91        17

    accuracy                           0.86        22
   macro avg       0.80      0.84      0.82        22
weighted avg       0.88      0.86      0.87        22
```

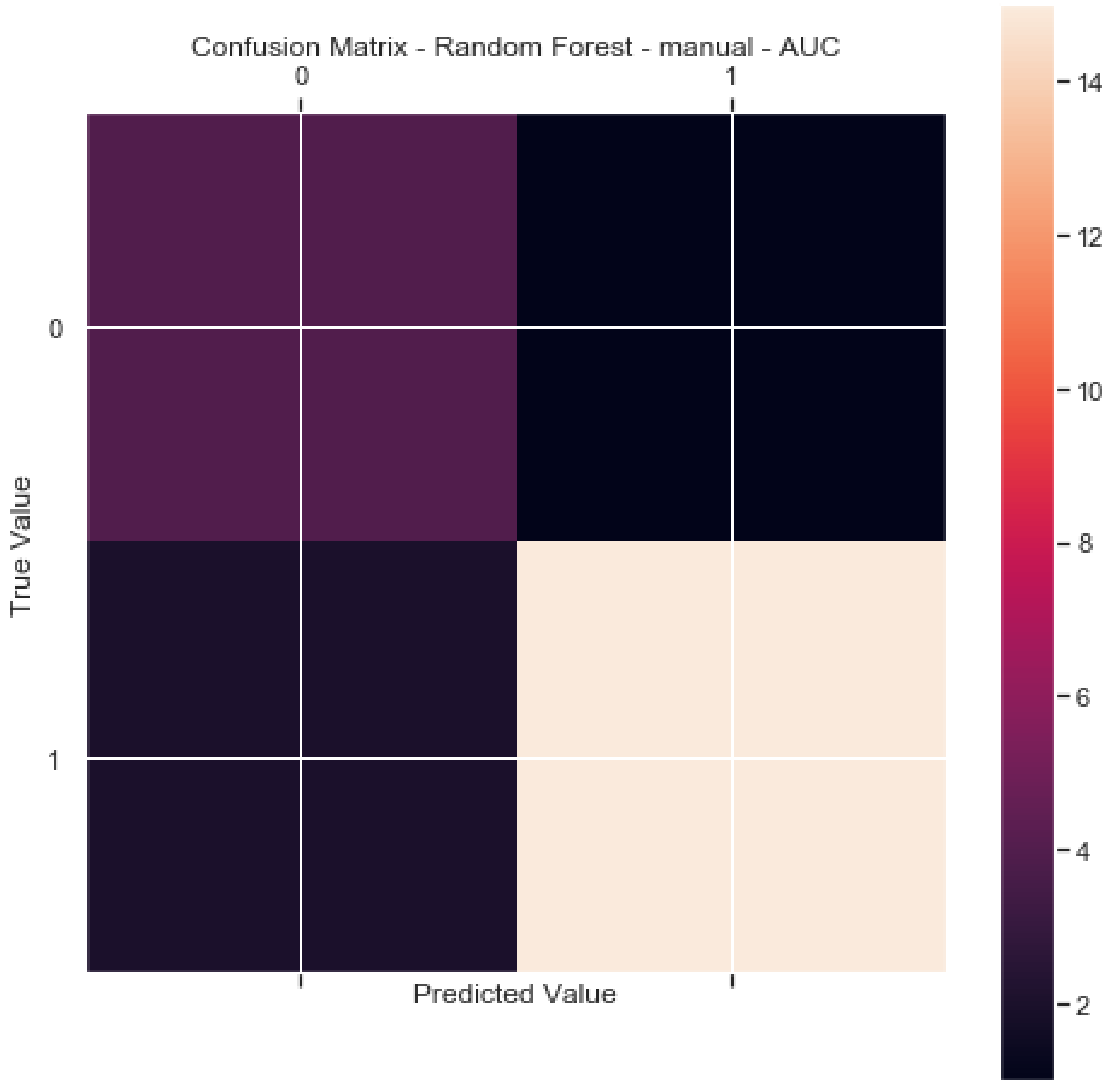


[47]:

```
rf_results = rf_results.assign(TestAcc = rf_results.TestAcc.astype(float))
auc_max_id = rf_results['AUC'].idxmax()

best_MaxDepth    = rf_results.iloc[auc_max_id, 0]
best_Estimators = rf_results.iloc[auc_max_id, 1]
```

```
print(f'Best Parameters:  \nMaxDepth = {best_MaxDepth}            Estimators =␣
↪{best_Estimators}')

model_type = 'Random Forest'
tuning = 'manual'
scoring = 'AUC'
rf_best_auc = RandomForestClassifier(n_estimators=best_Estimators,␣
↪max_depth=best_MaxDepth,
                                       random_state=RAND_STATE)

rf_best_auc.fit(X_train,y_train)
predictions = rf_best_auc.predict(X_test)
FP_rate, recall, thresholds = roc_curve(y_test, predictions)

evaluate_model(rf_best_auc, tuning, scoring, X_train, y_train, X_test, y_test,␣
↪model_type)
```

```
Best Parameters:
MaxDepth = 8            Estimators = 300

Random Forest Model Results:
Training Accuracy: 1.000    Test Accuracy: 0.864    AUC: 0.841

[[ 4  1]
 [ 2 15]]
```



Classification Report - Random Forest - manual - AUC

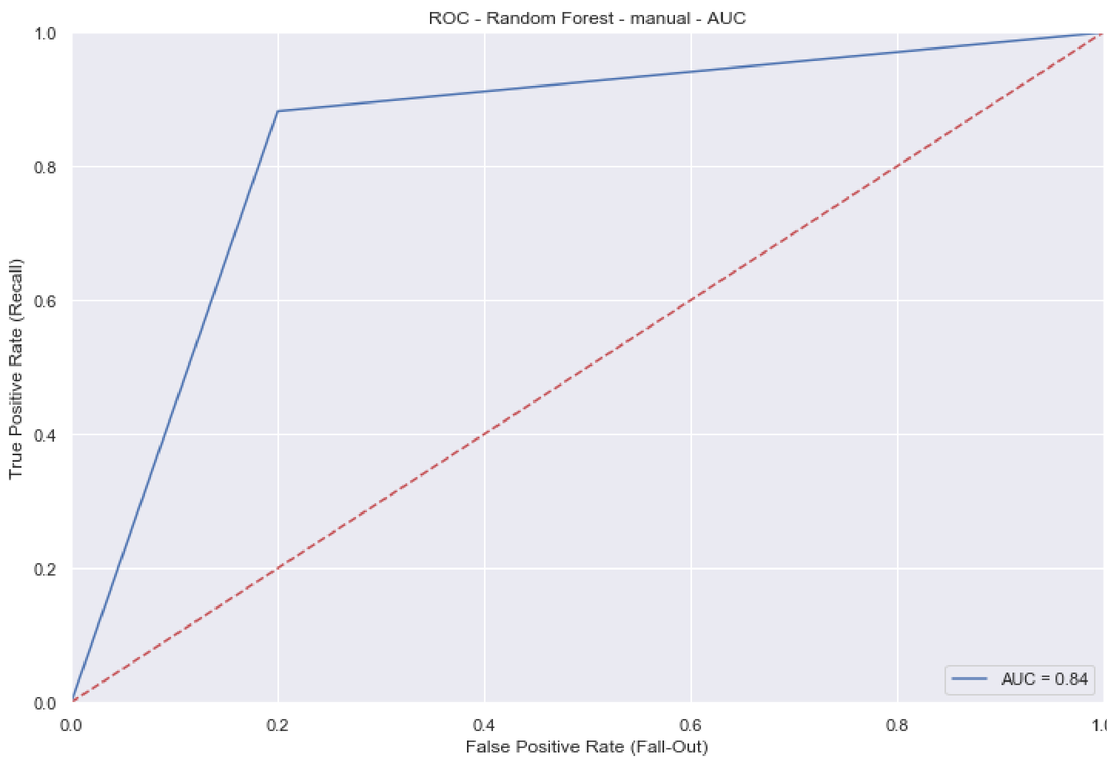
Train:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	32
1	1.00	1.00	1.00	56
accuracy			1.00	88
macro avg	1.00	1.00	1.00	88
weighted avg	1.00	1.00	1.00	88

Test:

	precision	recall	f1-score	support

```
           0       0.67      0.80      0.73         5
           1       0.94      0.88      0.91        17

    accuracy                           0.86        22
   macro avg       0.80      0.84      0.82        22
weighted avg       0.88      0.86      0.87        22
```



[48]:
```
# feature importance - Random Forest - mean decrease in impurity
rf_best_auc.feature_importances_

print(f'Feature Importance - mean descrease in impurity algorithm')
rf_importances = pd.DataFrame(data=rf_best_auc.feature_importances_, index =␣
 ↪X_train.columns,
                             columns=['importance'])
rf_importances = rf_importances.sort_values(by='importance', ascending=False,␣
 ↪kind='mergesort')
print(rf_importances)

importance_compare = importance_compare.assign(rf_importance =␣
 ↪rf_importances['importance'])
```

```
Feature Importance - mean descrease in impurity algorithm
              importance
DeptAdv         0.116103
DeptCom         0.111713
DeptCoop        0.109654
SupRec          0.079908
SupFair         0.073546
RatePay         0.069208
SupResolve      0.066679
PdHoliday       0.065288
AnnLeave        0.062718
DeptCond        0.061862
SupCoop         0.056299
SupInf          0.046542
SupTrn          0.043195
SupPol          0.037285
```

[49]:

```python
# feature importance - Random Forest - permutation importance
def permutation_importances(rf, X_test, y_test): #, metric):
    # baseline = metric(rf, X_train, y_train)
    rf_test_pred = rf.predict(X_test)
    baseline = accuracy_score(y_test, rf_test_pred)
    imp = []
    for col in X_test.columns:
        save = X_test[col].copy()
        X_test[col] = np.random.permutation(X_test[col])
        rf_test_pred = rf.predict(X_test)
        m = accuracy_score(y_test, rf_test_pred)
        #m = metric(rf, X_train, y_train)
        X_test[col] = save
#          print(baseline)
#          print(m)
#          print(type(baseline))
#          print(type(m))
        imp.append(baseline - m)
        #imp = [3,3,3]
    return np.array(imp)

imp = permutation_importances(rf_best_auc, X_test, y_test)

print(imp)

print(f'Feature Importance - permutation importance')
perm_importances = pd.DataFrame(imp, index = X_test.columns,
                             columns=['importance'])
#perm_importances = perm_importances.sort_values(by='importance',␣
 ↪ascending=False, kind='mergesort')
```

```
print(perm_importances)
importance_compare = importance_compare.assign(permutation =␣
↪perm_importances['importance'])
# print(importance_compare)
```

```
[0.04545455 0.         0.04545455 0.         0.04545455 0.
 0.         0.13636364 0.09090909 0.04545455 0.22727273 0.04545455
 0.09090909 0.        ]
Feature Importance - permutation importance
            importance
SupPol        0.045455
SupInf        0.000000
SupFair       0.045455
SupRec        0.000000
SupCoop       0.045455
SupResolve    0.000000
SupTrn        0.000000
DeptCom       0.136364
DeptCond      0.090909
DeptCoop      0.045455
DeptAdv       0.227273
RatePay       0.045455
AnnLeave      0.090909
PdHoliday     0.000000
```

[50]:
```
# feature importance - Random Forest - drop-column
def dropcol_importances(rf, X_train, y_train, X_test, y_test):
    #rf_ = clone(rf)
    rf_ = RandomForestClassifier(n_estimators=best_Estimators,␣
↪max_depth=best_MaxDepth,
                                  random_state=RAND_STATE)
    #rf_.random_state = 999
    rf_.fit(X_train, y_train)
    # baseline = rf_.oob_score_
    rf_test_pred = rf_.predict(X_test)
    baseline = accuracy_score(y_test, rf_test_pred)
    imp = []
    for col in X_train.columns:
        X_train_drop = X_train.drop(col, axis=1)
        X_test_drop = X_test.drop(col, axis=1)

        rf_ = RandomForestClassifier(n_estimators=best_Estimators,␣
↪max_depth=best_MaxDepth,
                                      random_state=RAND_STATE)
        #rf_.random_state = 999
        rf_.fit(X_train_drop, y_train)
```

```
        rf_test_pred = rf_.predict(X_test_drop)
        o = accuracy_score(y_test, rf_test_pred)
        #o = rf_.oob_score_
        imp.append(baseline - o)
    imp = np.array(imp)
    I = pd.DataFrame(
            data={'Feature':X_train.columns,
                  'Importance':imp})
    I = I.set_index('Feature')
    I = I.sort_values('Importance', ascending=False)
    return I

drop_col_imp = dropcol_importances(rf_best_auc, X_train, y_train, X_test,␣
 ↪y_test)
importance_compare = importance_compare.assign(drop_col =␣
 ↪drop_col_imp['Importance'])
print(importance_compare)
```

```
             boruta_rank  dt_importance  rf_importance  permutation  drop_col
SupPol                10       0.000000       0.037285     0.045455  0.090909
SupInf                11       0.000000       0.046542     0.000000  0.090909
SupFair                2       0.000000       0.073546     0.045455  0.045455
SupRec                 1       0.230962       0.079908     0.000000  0.045455
SupCoop                5       0.128651       0.056299     0.045455  0.045455
SupResolve             3       0.000000       0.066679     0.000000  0.000000
SupTrn                 7       0.000000       0.043195     0.000000  0.045455
DeptCom                1       0.411322       0.111713     0.136364  0.181818
DeptCond               5       0.000000       0.061862     0.090909  0.090909
DeptCoop               1       0.086129       0.109654     0.045455  0.045455
DeptAdv                1       0.000000       0.116103     0.227273  0.090909
RatePay                4       0.142935       0.069208     0.045455  0.045455
AnnLeave               9       0.000000       0.062718     0.090909  0.090909
PdHoliday              8       0.000000       0.065288     0.000000  0.090909
```

4.5.1 Random Forest using GridSearch

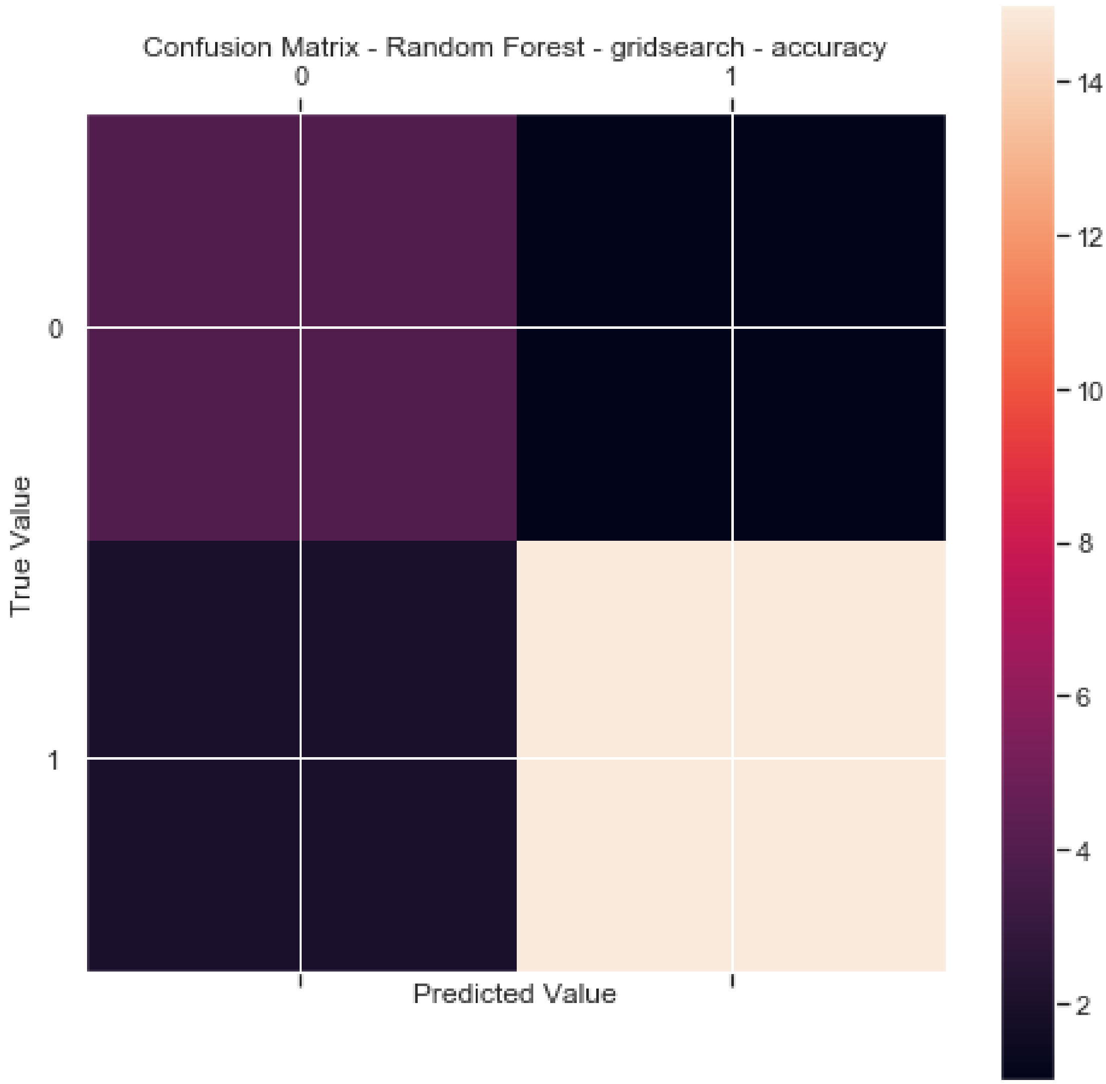
[51]:
```
# random forest with grid search, best accuracy
if __name__ == '__main__':
    model_type = 'Random Forest'
    tuning = 'gridsearch'
    scoring = 'accuracy'
    max_d = range(2, 15)
    rf_gs_model = RandomForestClassifier(random_state=RAND_STATE)
    grid = {'n_estimators':[30, 50, 80, 100, 150, 200, 300, 400, 500, 600, 700,␣
 ↪800],
            'max_depth': max_d  #, 'max_features' : max_feat
                  }
```

```
    grid_search = GridSearchCV(estimator=rf_gs_model, param_grid=grid,␣
↪n_jobs=-1,
                               verbose=2, scoring='accuracy', cv=5, iid=False)

    grid_search.fit(X_train, y_train)

    rf_most_acc = grid_search.best_estimator_

    rf_test_pred = grid_search.best_estimator_.predict(X_test)

    show_best_params(grid, grid_search.best_estimator_)
    evaluate_model(rf_most_acc, tuning, scoring, X_train, y_train, X_test,␣
↪y_test, model_type)
```

```
Fitting 5 folds for each of 156 candidates, totalling 780 fits
```

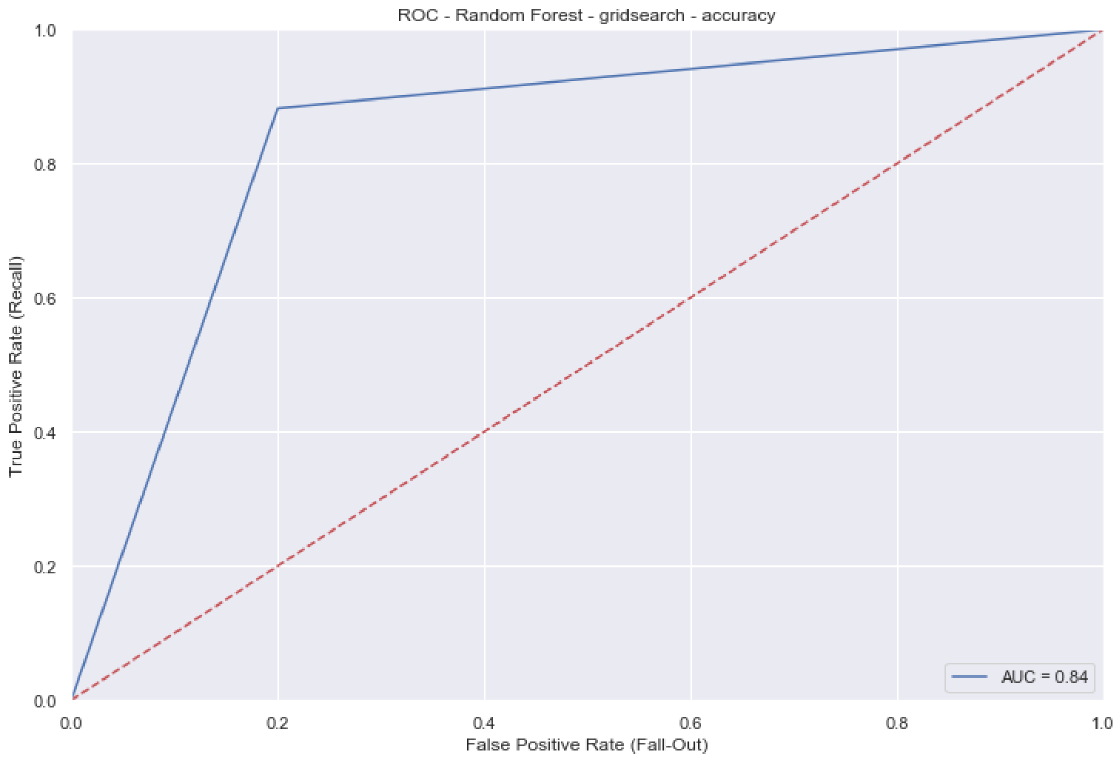
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 20 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 457 tasks      | elapsed:    6.4s
[Parallel(n_jobs=-1)]: Done 714 tasks      | elapsed:    9.9s
[Parallel(n_jobs=-1)]: Done 780 out of 780 | elapsed:   11.0s finished
```

```
Best parameters:
max_depth: 5
n_estimators: 150

Random Forest Model Results:
Training Accuracy: 0.920    Test Accuracy: 0.864    AUC: 0.841

[[ 4  1]
 [ 2 15]]
```




```
Classification Report - Random Forest - gridsearch - accuracy
Train:
              precision    recall  f1-score   support

           0       1.00      0.78      0.88        32
           1       0.89      1.00      0.94        56

    accuracy                           0.92        88
   macro avg       0.94      0.89      0.91        88
weighted avg       0.93      0.92      0.92        88

Test:
              precision    recall  f1-score   support
```

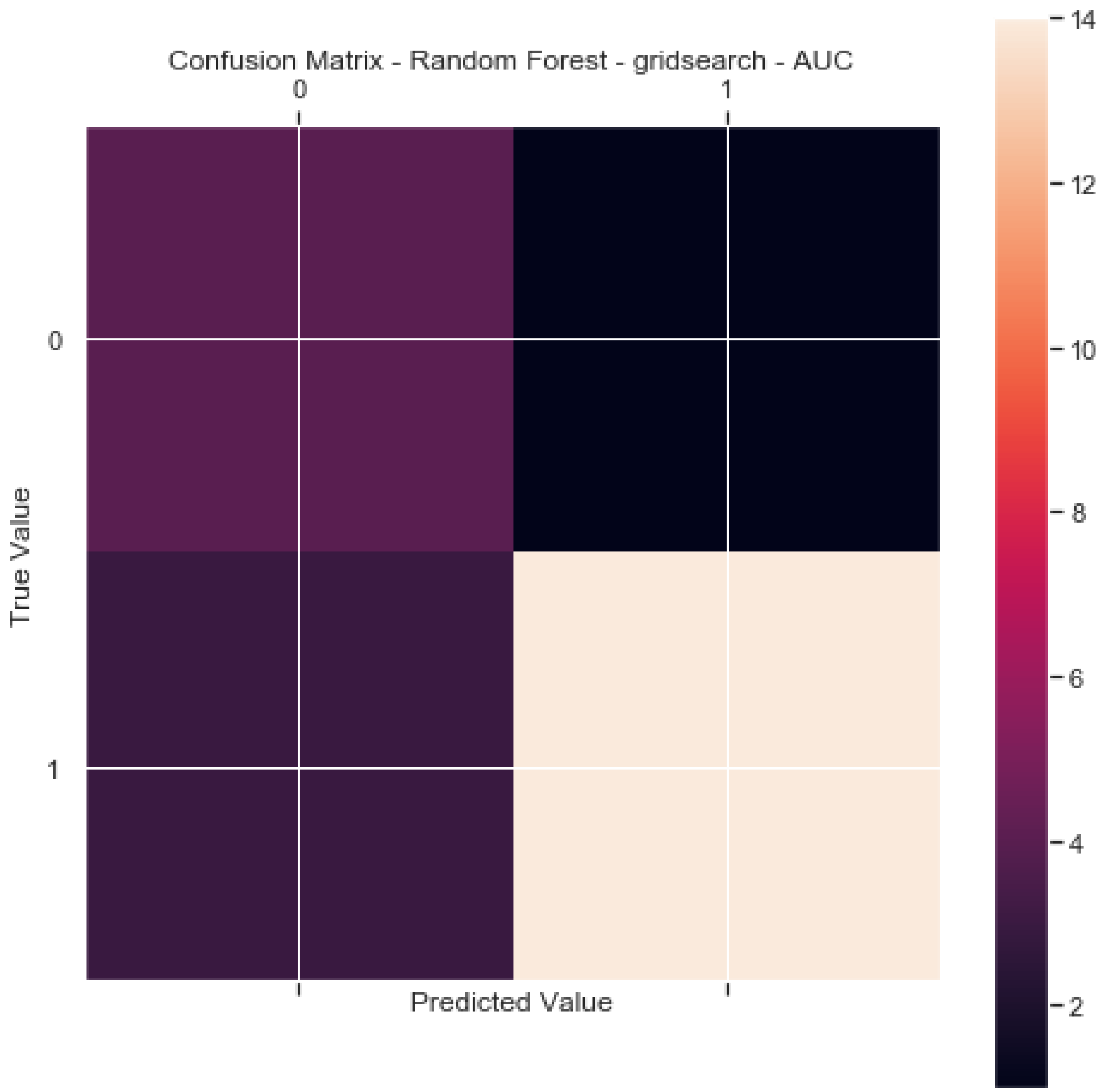
```
           0       0.67      0.80      0.73         5
           1       0.94      0.88      0.91        17

    accuracy                           0.86        22
   macro avg       0.80      0.84      0.82        22
weighted avg       0.88      0.86      0.87        22
```



[52]:
```
# random forest with grid search, best auc
if __name__ == '__main__':
    model_type = 'Random Forest'
    tuning = 'gridsearch'
    scoring = 'AUC'
    max_d = range(2, 15)
    rf_gs_model = RandomForestClassifier(random_state=RAND_STATE)

    grid = {'n_estimators':[30, 50, 80, 100, 150, 200, 300, 400, 500, 600, 700,␣
↪800],
            'max_depth': max_d  #, 'max_features' : max_feat
                  }
    grid_search = GridSearchCV(estimator=rf_gs_model, param_grid=grid,␣
↪n_jobs=-1,
```

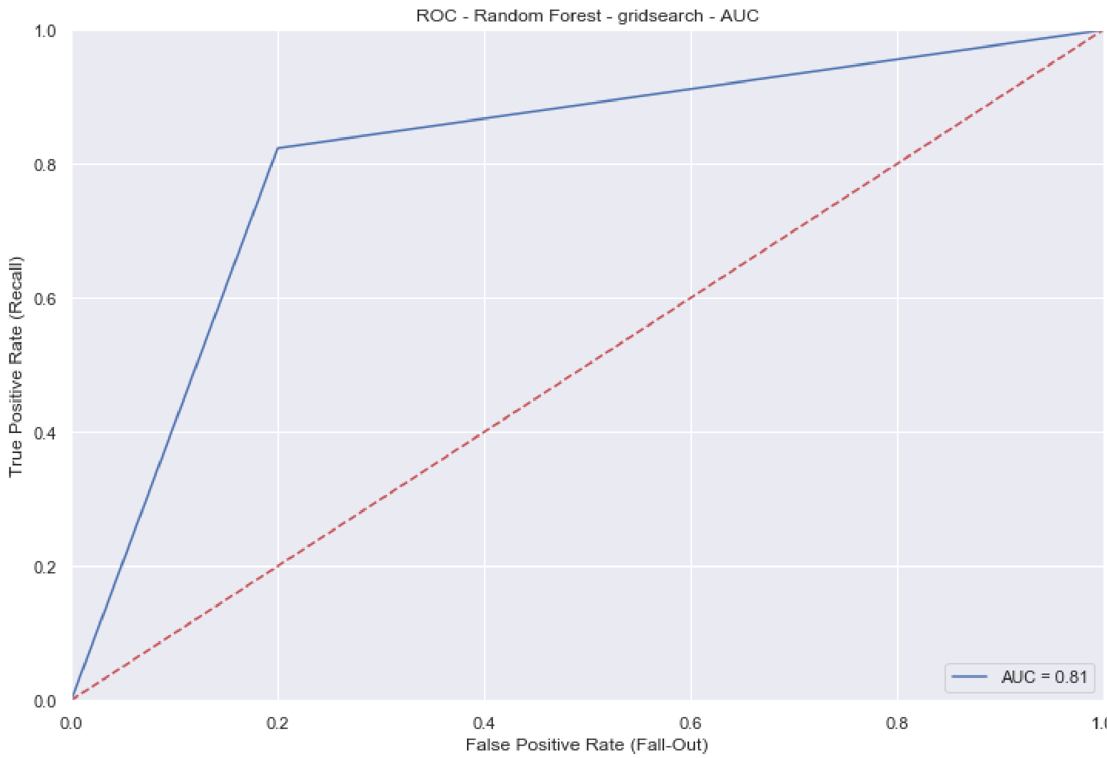
```
                              verbose=2, scoring='roc_auc', cv=5, iid=False)
    grid_search.fit(X_train, y_train)

    rf_best_auc = grid_search.best_estimator_
    rf_test_pred = grid_search.best_estimator_.predict(X_test)
    show_best_params(grid, grid_search.best_estimator_)
    evaluate_model(rf_best_auc, tuning, scoring, X_train, y_train, X_test,␣
↪y_test, model_type)
```

```
Fitting 5 folds for each of 156 candidates, totalling 780 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 20 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 457 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done 712 tasks      | elapsed:    9.9s

Best parameters:
max_depth: 4
n_estimators: 100

Random Forest Model Results:
Training Accuracy: 0.909    Test Accuracy: 0.818    AUC: 0.812

[[ 4  1]
 [ 3 14]]

[Parallel(n_jobs=-1)]: Done 780 out of 780 | elapsed:   11.0s finished
```




```
Classification Report - Random Forest - gridsearch - AUC
Train:
              precision    recall  f1-score   support

           0       0.96      0.78      0.86        32
           1       0.89      0.98      0.93        56

    accuracy                           0.91        88
   macro avg       0.92      0.88      0.90        88
weighted avg       0.91      0.91      0.91        88

Test:
```

```
              precision    recall  f1-score   support

           0       0.57      0.80      0.67         5
           1       0.93      0.82      0.87        17

    accuracy                           0.82        22
   macro avg       0.75      0.81      0.77        22
weighted avg       0.85      0.82      0.83        22
```



4.6 Adaptive Boost

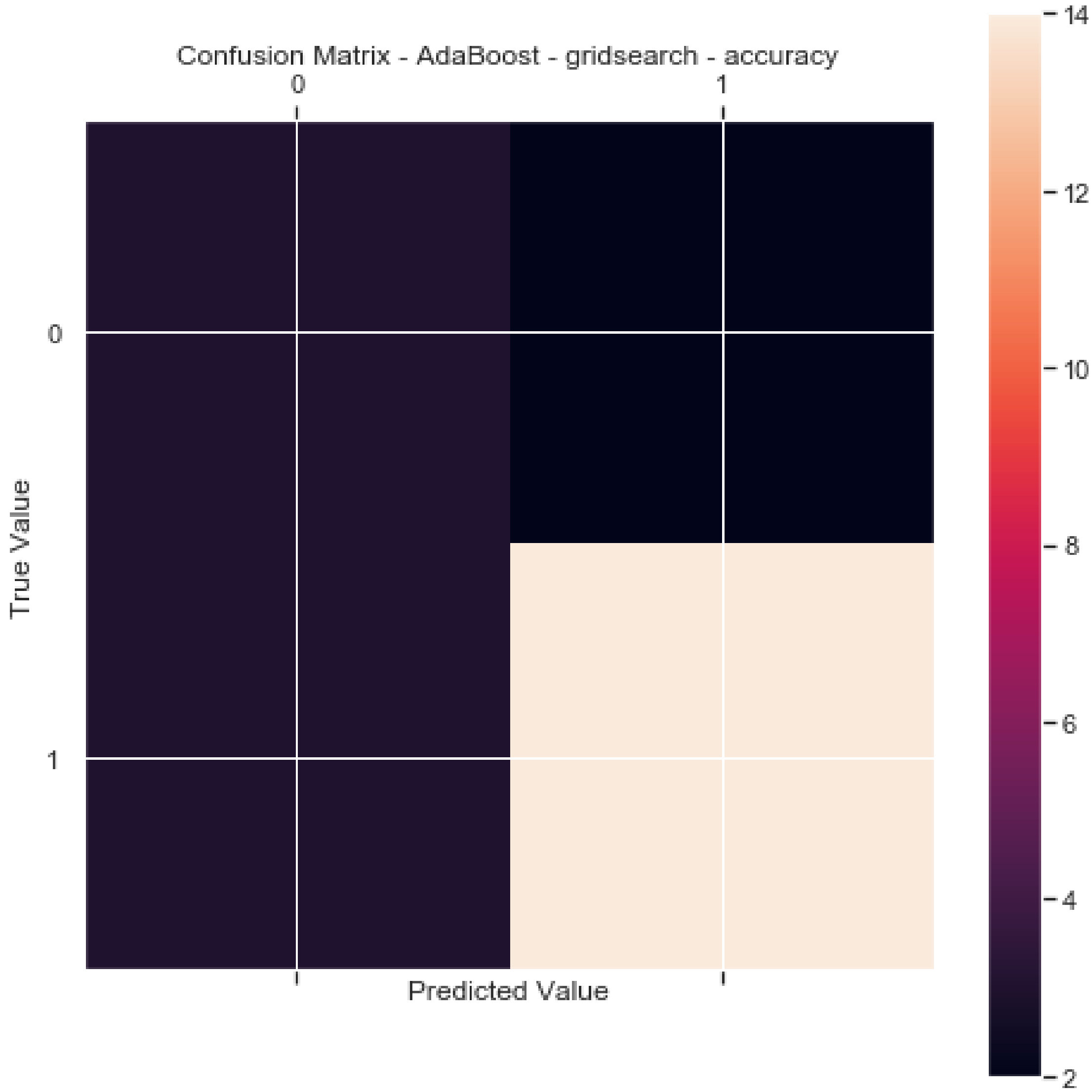
[53]:
```python
from sklearn.ensemble import AdaBoostClassifier

if __name__ == '__main__':
    model_type = 'AdaBoost'
    tuning = 'gridsearch'
    scoring = 'accuracy'
    grid = {'n_estimators':[100, 200, 300, 400, 500, 600, 700, 800, 1000],
            'learning_rate':[0.001, 0.01, 0.1, 1] }
    ada_model = AdaBoostClassifier(random_state=RAND_STATE)
    grid_search = GridSearchCV(estimator=ada_model, param_grid=grid, n_jobs=-1,
                               verbose=1, scoring='accuracy', cv=5, iid=False)
```

```
    grid_search.fit(X_train, y_train)
    ada_most_acc = grid_search.best_estimator_

    show_best_params(grid, grid_search.best_estimator_)
    evaluate_model(ada_most_acc, tuning, scoring, X_train, y_train, X_test,␣
↪y_test, model_type)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
```

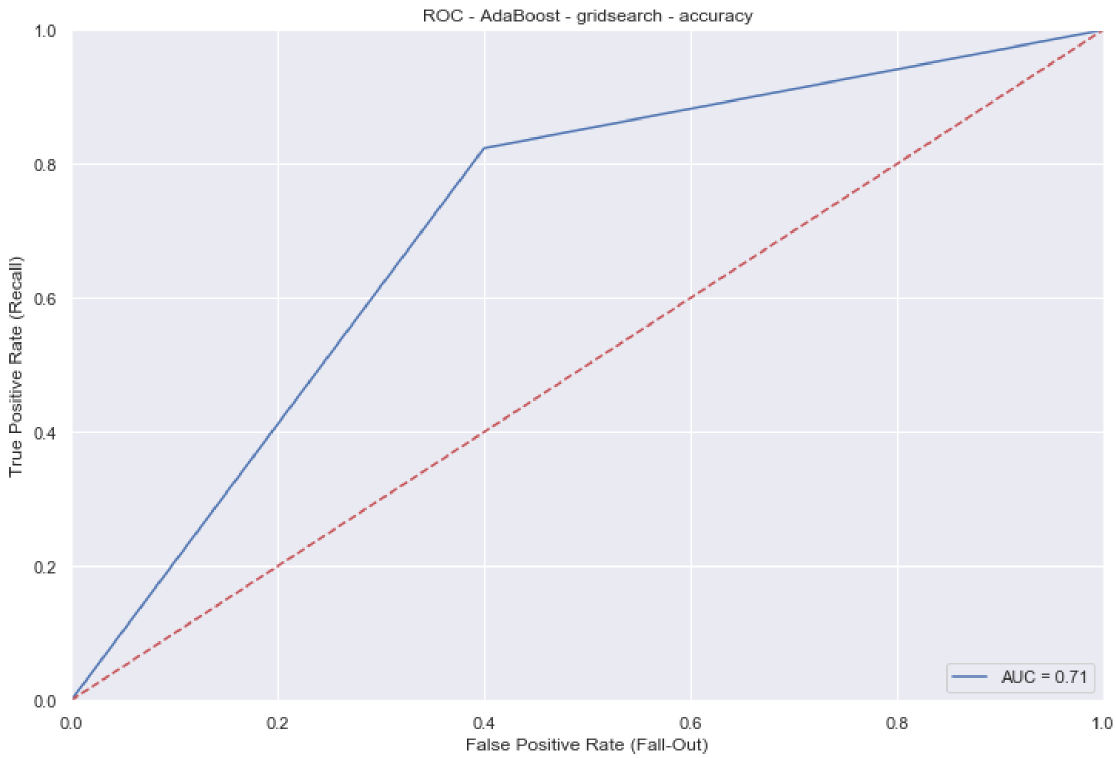
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 20 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:    0.3s
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed:    4.9s finished
```

```
Best parameters:
learning_rate: 0.1
n_estimators: 200

AdaBoost Model Results:
Training Accuracy: 0.898    Test Accuracy: 0.773    AUC: 0.712

[[ 3  2]
 [ 3 14]]
```




```
Classification Report - AdaBoost - gridsearch - accuracy
Train:
              precision    recall  f1-score   support

           0       0.93      0.78      0.85        32
           1       0.89      0.96      0.92        56

    accuracy                           0.90        88
   macro avg       0.91      0.87      0.89        88
weighted avg       0.90      0.90      0.90        88

Test:
```

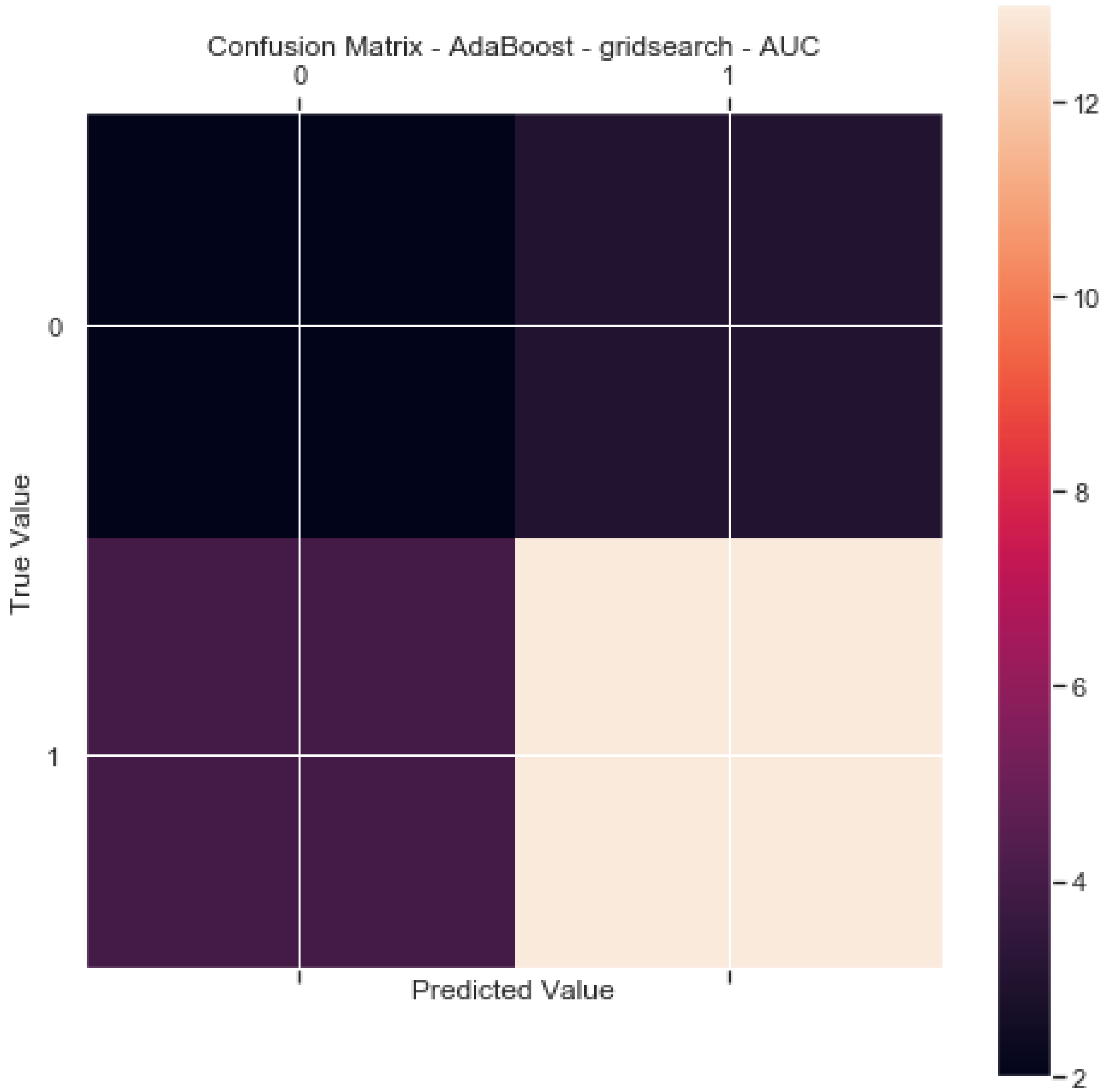
```
              precision    recall  f1-score   support

           0       0.50      0.60      0.55         5
           1       0.88      0.82      0.85        17

    accuracy                           0.77        22
   macro avg       0.69      0.71      0.70        22
weighted avg       0.79      0.77      0.78        22
```



[54]:
```
if __name__ == '__main__':
    model_type = 'AdaBoost'
    tuning = 'gridsearch'
    scoring = 'AUC'
    grid = {'n_estimators':[100, 200, 300, 400, 500, 600, 700, 800, 1000],
            'learning_rate':[0.001, 0.01, 0.1, 1] }
    ada_model = AdaBoostClassifier(random_state=RAND_STATE)
    grid_search = GridSearchCV(estimator=ada_model, param_grid=grid, n_jobs=-1,
                               verbose=1, scoring='roc_auc', cv=5, iid=False)
    grid_search.fit(X_train, y_train)

    ada_best_auc = grid_search.best_estimator_
    show_best_params(grid, grid_search.best_estimator_)
```

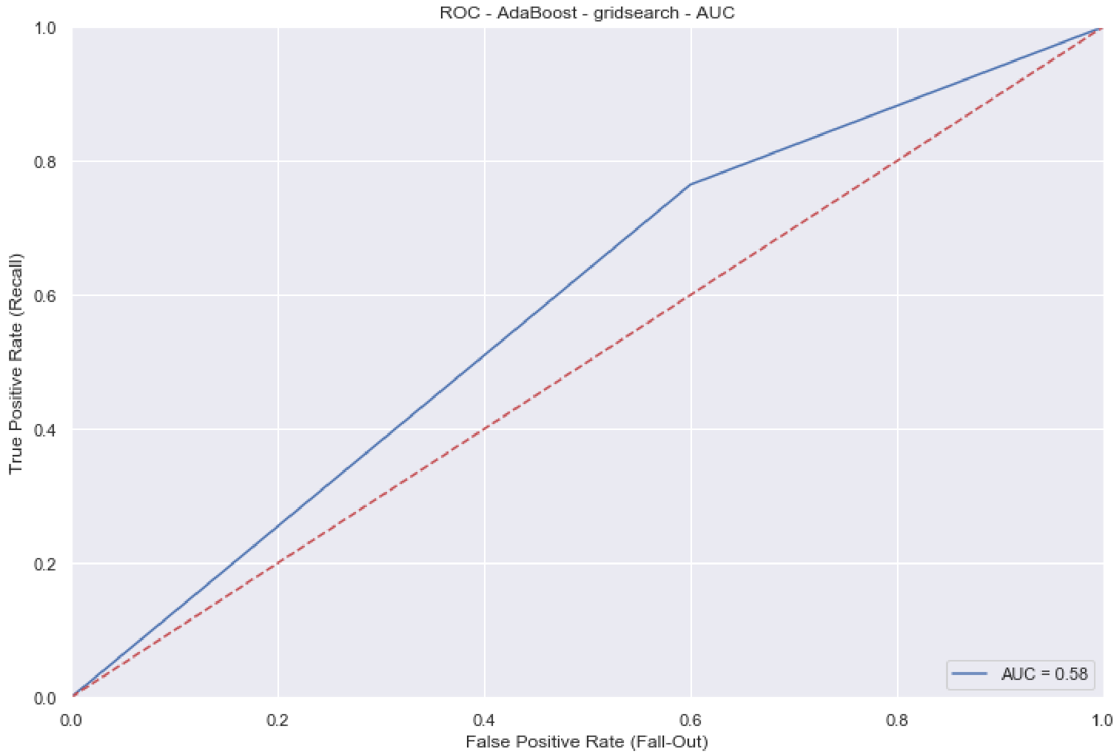
```
   evaluate_model(ada_best_auc,  tuning, scoring, X_train, y_train, X_test,␣
↪y_test, model_type)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 20 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:    0.2s

Best parameters:
learning_rate: 0.1
n_estimators: 100

AdaBoost Model Results:
Training Accuracy: 0.875    Test Accuracy: 0.682    AUC: 0.582

[[ 2  3]
 [ 4 13]]

[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed:    5.1s finished
```




```
Classification Report - AdaBoost - gridsearch - AUC
Train:
              precision    recall  f1-score   support

           0       0.92      0.72      0.81        32
           1       0.86      0.96      0.91        56

    accuracy                           0.88        88
   macro avg       0.89      0.84      0.86        88
weighted avg       0.88      0.88      0.87        88

Test:
```

```
              precision    recall  f1-score   support

           0       0.33      0.40      0.36         5
           1       0.81      0.76      0.79        17

    accuracy                           0.68        22
   macro avg       0.57      0.58      0.58        22
weighted avg       0.70      0.68      0.69        22
```

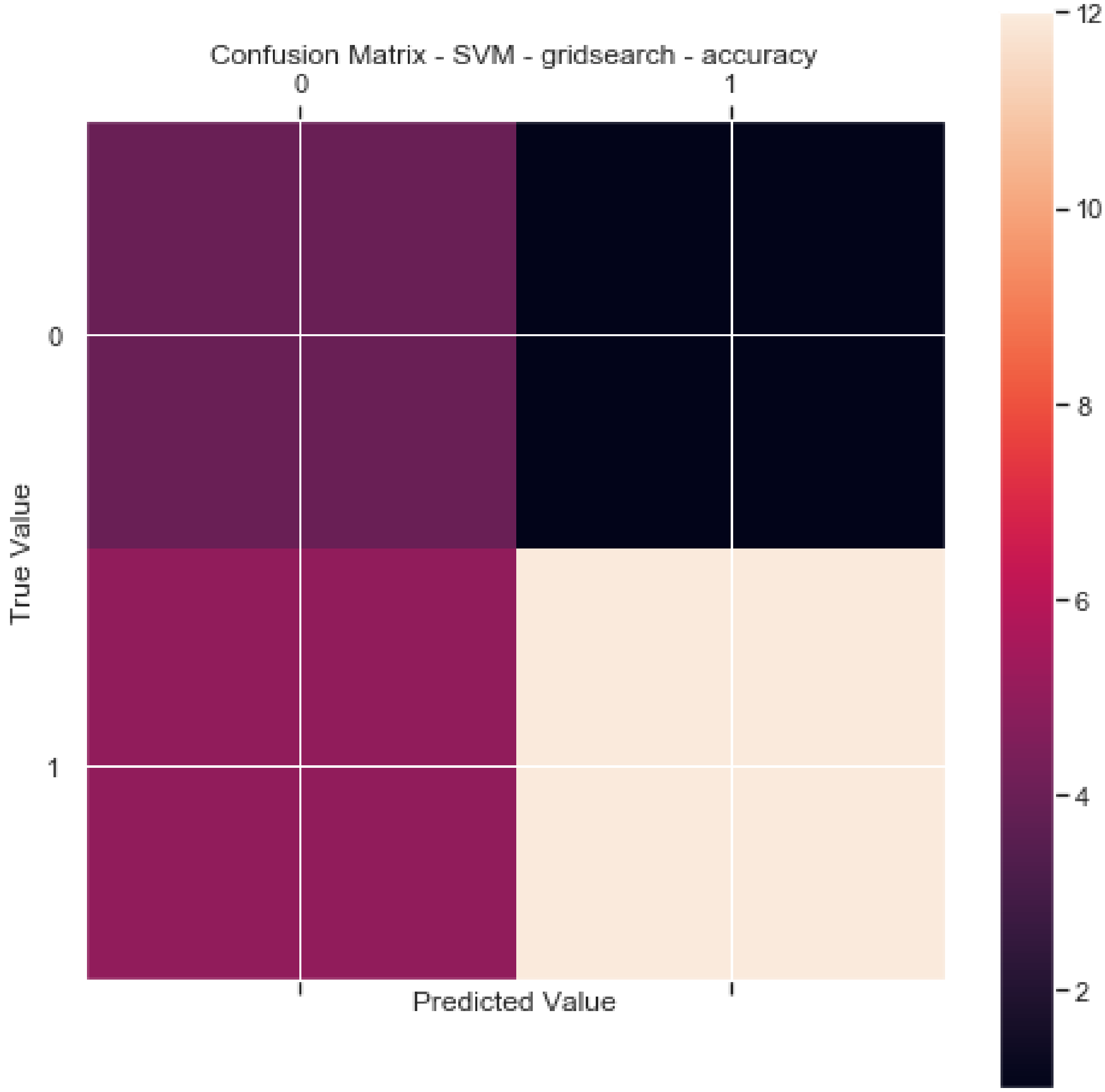


4.7 Support Vector Machine

[55]:
```
# svm - most accurate

#from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
#from sklearn.grid_search import GridSearchCV
#from sklearn.metrics import classification_report

if __name__ == '__main__':
    model_type = 'SVM'
    tuning = 'gridsearch'
    scoring = 'accuracy'
```

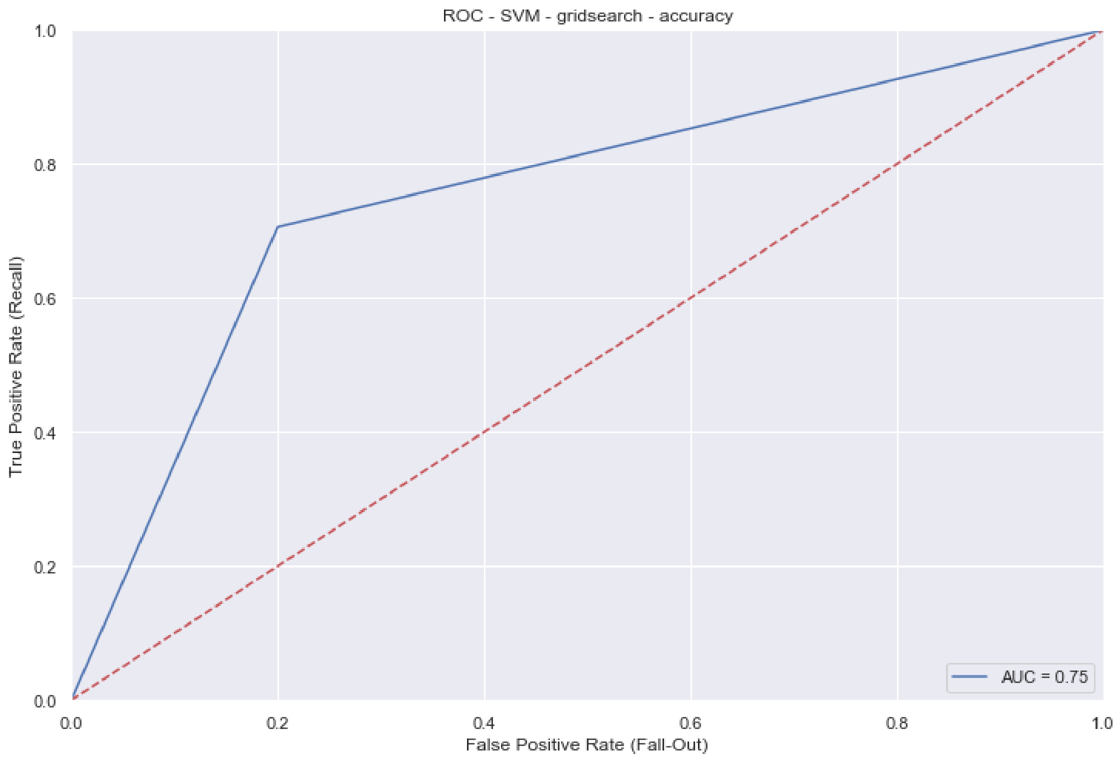
```python
    pipeline = Pipeline([
        ('svm', SVC(kernel='rbf', gamma=0.01, C=100))
    ])
    parameters = {
        'svm__gamma': (0.01, 0.03, 0.1, 0.3, 1),
        'svm__C': (0.1, 0.3, 1, 3, 10, 20, 30)
    }

    grid_search = GridSearchCV(pipeline, parameters, n_jobs=-1,
                               verbose=2, scoring='accuracy', cv=5, iid=False)
    grid_search.fit(X_train, y_train)

    svm_most_acc = grid_search.best_estimator_
    show_best_params(parameters, grid_search.best_estimator_)
    evaluate_model(svm_most_acc, tuning, scoring, X_train, y_train, X_test,␣
↪y_test, model_type)
```

```
Fitting 5 folds for each of 35 candidates, totalling 175 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 20 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    0.0s

Best parameters:
svm__C: 20
svm__gamma: 0.01

SVM Model Results:
Training Accuracy: 0.886    Test Accuracy: 0.727    AUC: 0.753

[[ 4  1]
 [ 5 12]]

[Parallel(n_jobs=-1)]: Done 175 out of 175 | elapsed:    0.6s finished
```




```
Classification Report - SVM - gridsearch - accuracy
Train:
              precision    recall  f1-score   support

           0       0.96      0.72      0.82        32
           1       0.86      0.98      0.92        56

    accuracy                           0.89        88
   macro avg       0.91      0.85      0.87        88
weighted avg       0.90      0.89      0.88        88

Test:
```

```
              precision    recall  f1-score   support

           0       0.44      0.80      0.57         5
           1       0.92      0.71      0.80        17

    accuracy                           0.73        22
   macro avg       0.68      0.75      0.69        22
weighted avg       0.81      0.73      0.75        22
```



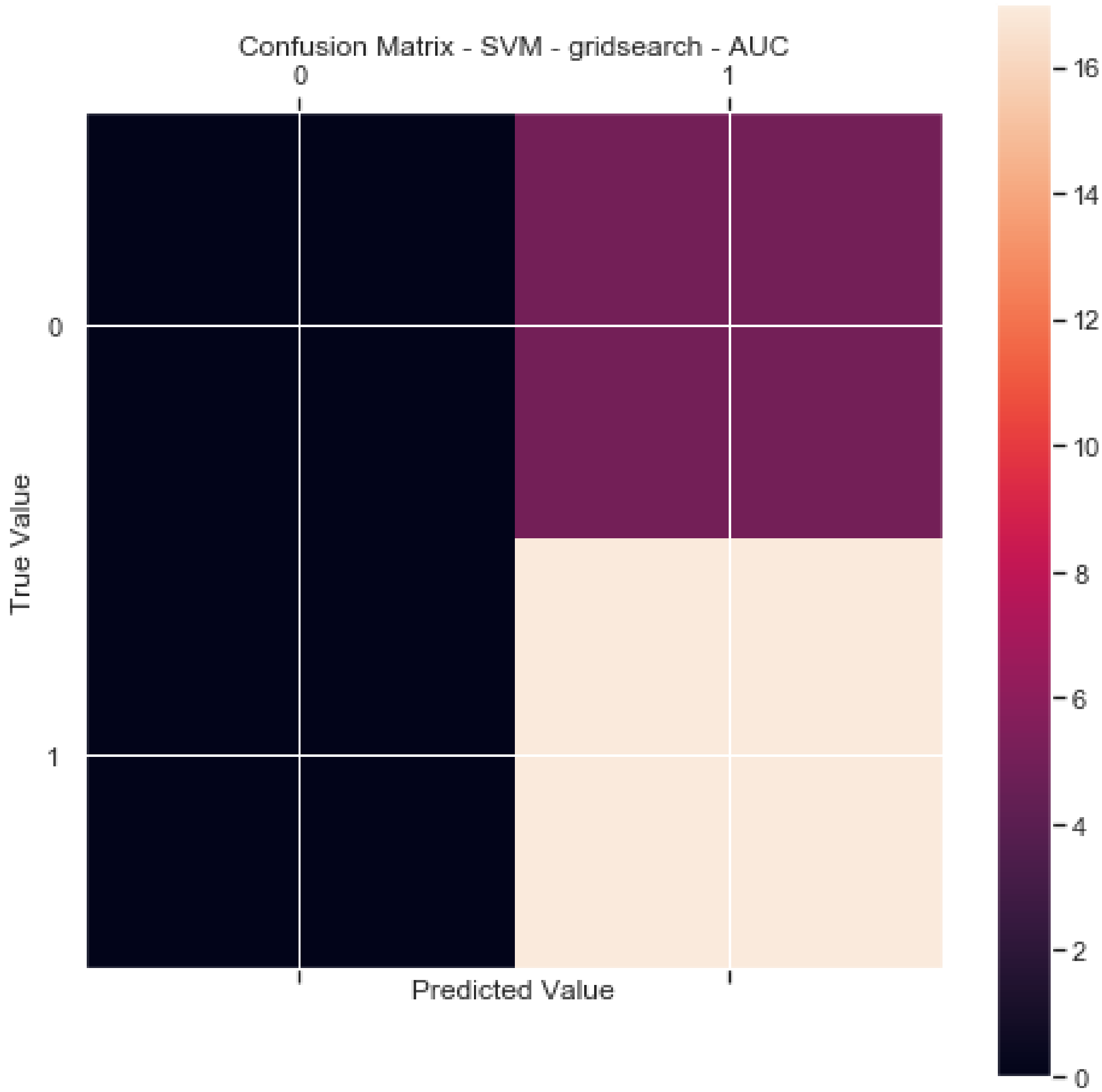
[56]:
```
# svm - best auc

#from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
#from sklearn.grid_search import GridSearchCV
#from sklearn.metrics import classification_report

if __name__ == '__main__':
    model_type = 'SVM'
    tuning = 'gridsearch'
    scoring = 'AUC'
    pipeline = Pipeline([
        ('clf', SVC(kernel='rbf', gamma=0.01, C=100))
```

```
    ])
    parameters = {
        'clf__gamma': (0.01, 0.03, 0.1, 0.3, 1),
        'clf__C': (0.1, 0.3, 1, 3, 10, 20, 30)
    }

    grid_search = GridSearchCV(pipeline, parameters, n_jobs=-1,
                               verbose=2, scoring='roc_auc', cv=5, iid=False)
    grid_search.fit(X_train, y_train)

    svm_best_auc = grid_search.best_estimator_
    show_best_params(parameters, grid_search.best_estimator_)
    evaluate_model(svm_best_auc, tuning, scoring, X_train, y_train, X_test,␣
↪y_test, model_type)
```

```
Fitting 5 folds for each of 35 candidates, totalling 175 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 20 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done  48 out of 175 | elapsed:    0.2s remaining:    0.8s
[Parallel(n_jobs=-1)]: Done 136 out of 175 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done 175 out of 175 | elapsed:    0.4s finished
```

```
Best parameters:
clf__C: 0.3
clf__gamma: 1

SVM Model Results:
Training Accuracy: 0.636    Test Accuracy: 0.773    AUC: 0.5

[[ 0  5]
 [ 0 17]]
```

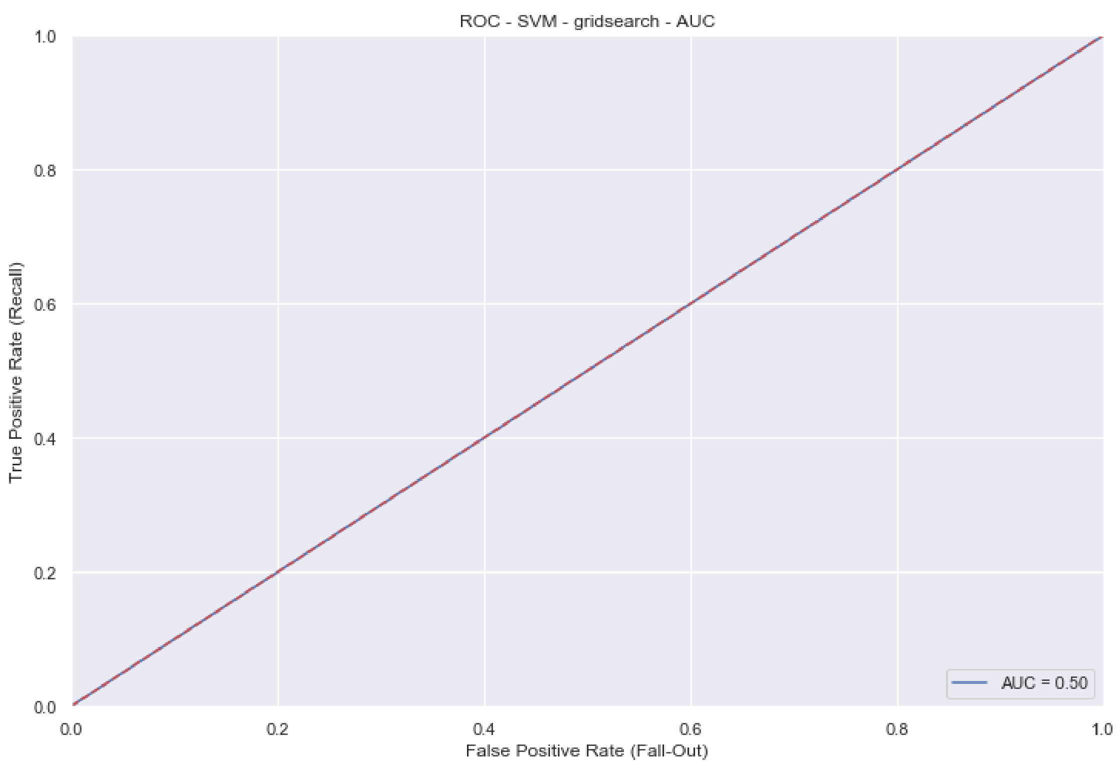


Classification Report - SVM - gridsearch - AUC
Train:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	32
1	0.64	1.00	0.78	56
accuracy			0.64	88
macro avg	0.32	0.50	0.39	88
weighted avg	0.40	0.64	0.49	88

Test:

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         5
           1       0.77      1.00      0.87        17

    accuracy                           0.77        22
   macro avg       0.39      0.50      0.44        22
weighted avg       0.60      0.77      0.67        22

C:\Users\op97dan\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\metrics\classification.py:1437: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\op97dan\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\metrics\classification.py:1437: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
```



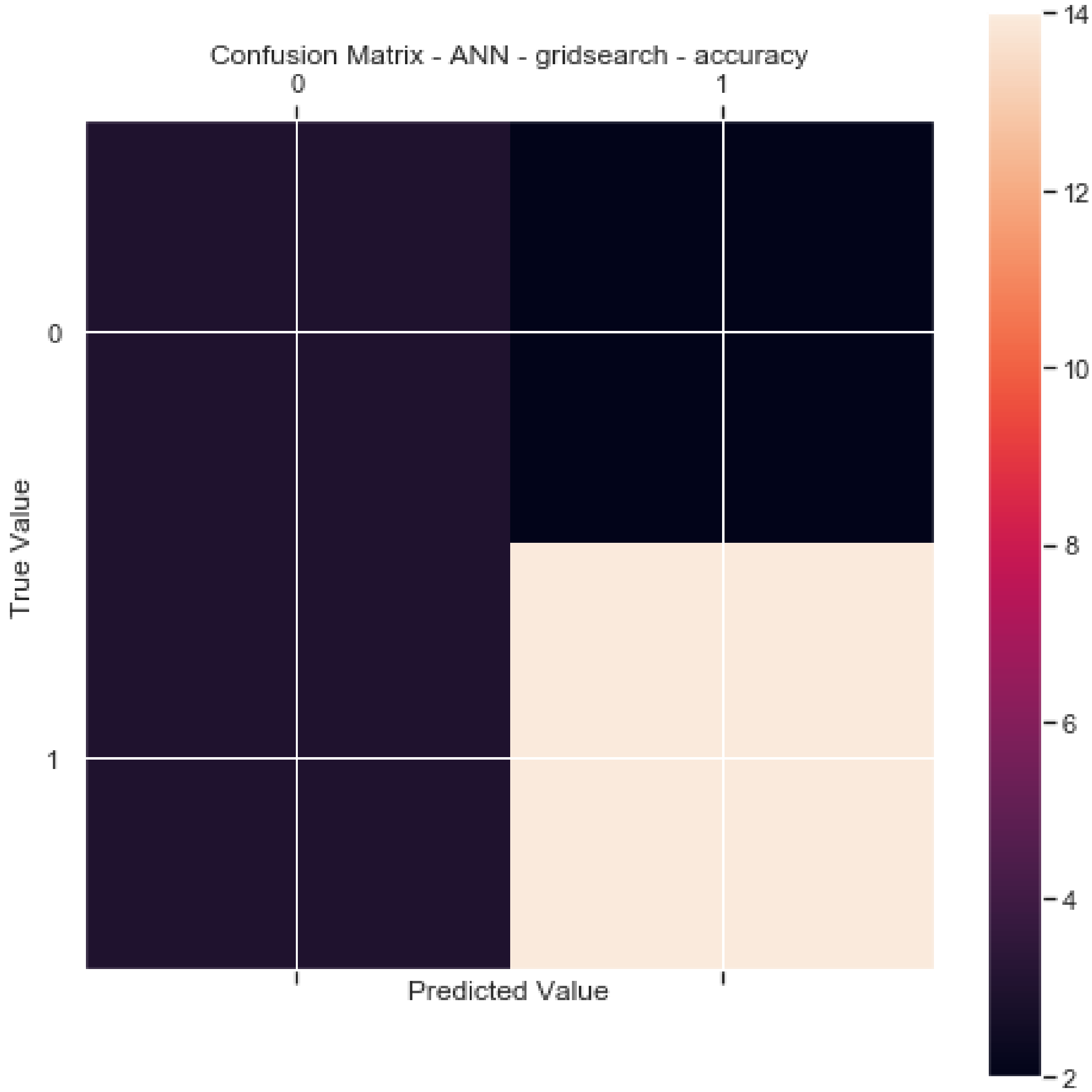
4.8 Artificial Neural Network

[57]:
```
# most accurate ANN, gridsearch
from sklearn.neural_network import MLPClassifier

if __name__ == '__main__':
    model_type = 'ANN'
    tuning = 'gridsearch'
    scoring = 'accuracy'
    pipeline = Pipeline([
        ('mlp', MLPClassifier(hidden_layer_sizes=(5, 5), alpha=0.1,␣
 ↪max_iter=300, random_state=RAND_STATE))
    ])
    neur_levels = []
    for x in range(2,21,2):
        for y in range(2,21,2):
            pair = (x, y)
            neur_levels.append(pair)
    alph_levels = [0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1]

    parameters = {
        'mlp__hidden_layer_sizes': neur_levels,
        'mlp__alpha': alph_levels
    }

    grid_search = GridSearchCV(pipeline, parameters, n_jobs=-1,
                               verbose=2, scoring='accuracy', cv=5, iid=False)
    grid_search.fit(X_train, y_train)
    ann_most_acc = grid_search.best_estimator_
    show_best_params(parameters, grid_search.best_estimator_)
    evaluate_model(ann_most_acc, tuning, scoring, X_train, y_train, X_test,␣
 ↪y_test, model_type)
```

```
Fitting 5 folds for each of 600 candidates, totalling 3000 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 20 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 204 tasks      | elapsed:    1.7s
[Parallel(n_jobs=-1)]: Done 610 tasks      | elapsed:    4.8s
[Parallel(n_jobs=-1)]: Done 1176 tasks      | elapsed:    9.2s
[Parallel(n_jobs=-1)]: Done 1906 tasks      | elapsed:   14.6s
[Parallel(n_jobs=-1)]: Done 2796 tasks      | elapsed:   21.4s
[Parallel(n_jobs=-1)]: Done 3000 out of 3000 | elapsed:   22.8s finished
C:\Users\op97dan\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\neural_network\multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
```

```
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Best parameters:
mlp__alpha: 0.001
mlp__hidden_layer_sizes: (12, 8)

ANN Model Results:
Training Accuracy: 0.795    Test Accuracy: 0.773    AUC: 0.712

[[ 3  2]
 [ 3 14]]
```



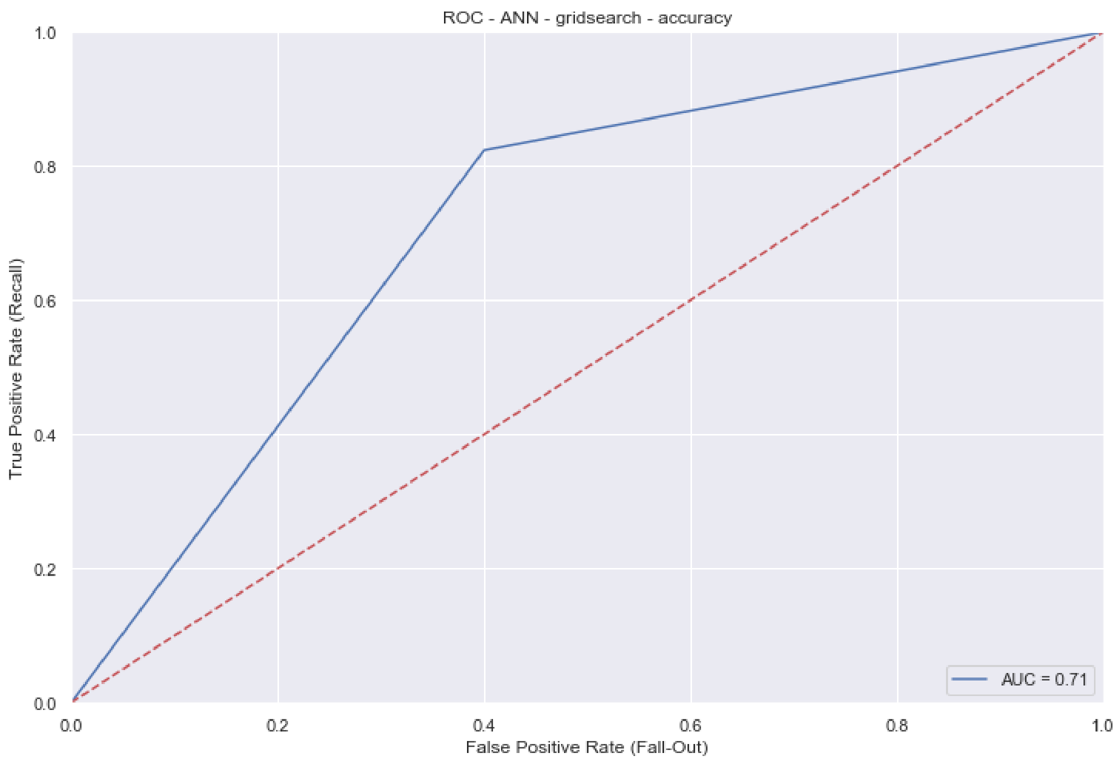
Classification Report - ANN - gridsearch - accuracy

Train:

	precision	recall	f1-score	support
0	0.85	0.53	0.65	32
1	0.78	0.95	0.85	56
accuracy			0.80	88
macro avg	0.81	0.74	0.75	88
weighted avg	0.81	0.80	0.78	88

Test:

	precision	recall	f1-score	support
0	0.50	0.60	0.55	5
1	0.88	0.82	0.85	17
accuracy			0.77	22
macro avg	0.69	0.71	0.70	22
weighted avg	0.79	0.77	0.78	22

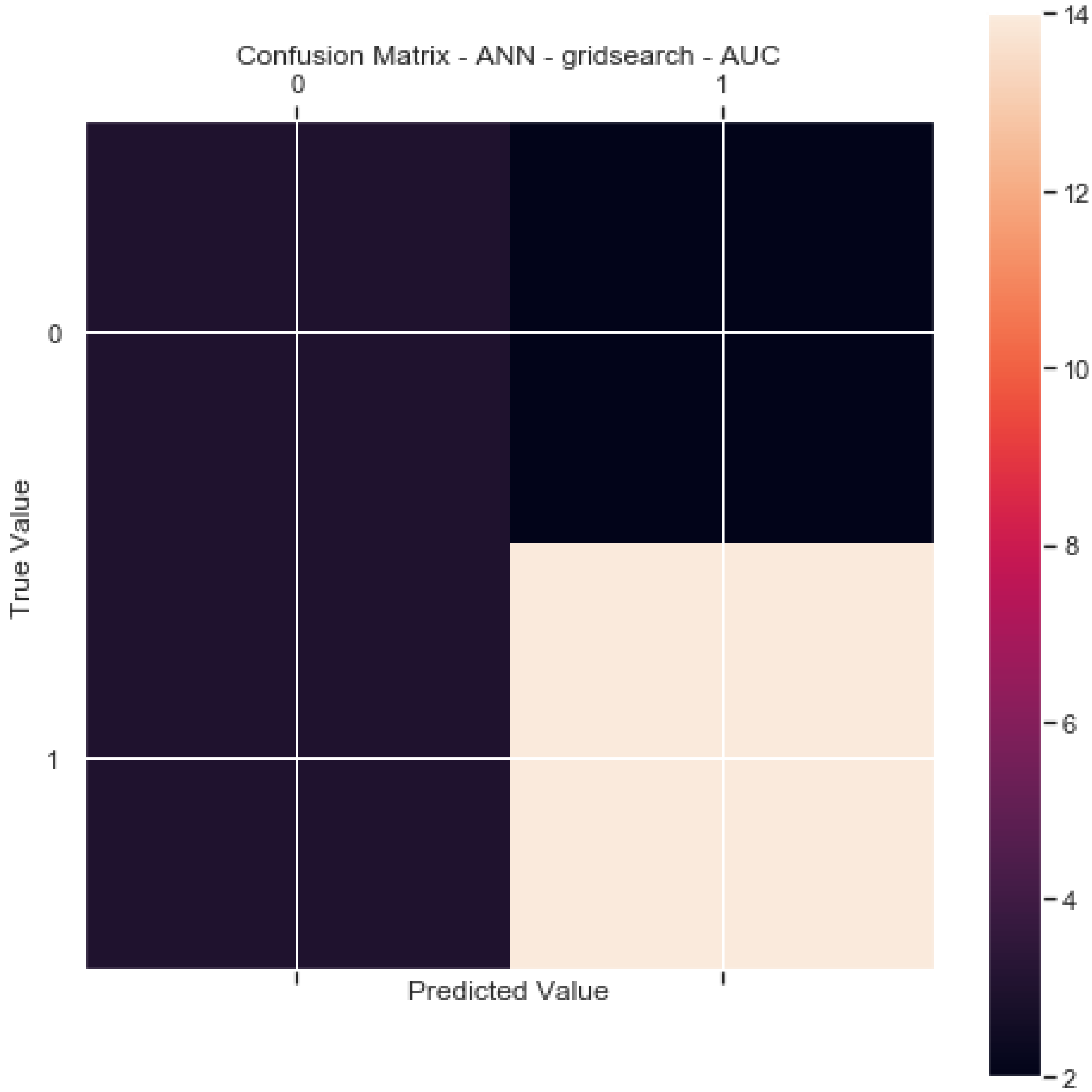


[58]:
```python
# highest AUC ANN, gridsearch
#from sklearn.neural_network import MLPClassifier
ann_model = MLPClassifier(solver='lbfgs', activation='logistic',␣
↪hidden_layer_sizes=(2, 3), random_state=RAND_STATE)

if __name__ == '__main__':
    model_type = 'ANN'
    tuning = 'gridsearch'
    scoring = 'AUC'
    pipeline = Pipeline([
        ('mlp', MLPClassifier(hidden_layer_sizes=(5, 5), alpha=0.1,␣
↪max_iter=300, random_state=RAND_STATE))
    ])
    neur_levels = []
    for x in range(2,21,2):
        for y in range(2,21,2):
            pair = (x, y)
            neur_levels.append(pair)
    alph_levels = [0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1]

    parameters = {
        'mlp__hidden_layer_sizes': neur_levels,
        'mlp__alpha': alph_levels
    }

    grid_search = GridSearchCV(pipeline, parameters, n_jobs=-1,
                               verbose=2, scoring='roc_auc', cv=5, iid=False)
    grid_search.fit(X_train, y_train)
    ann_best_auc = grid_search.best_estimator_

    show_best_params(parameters, grid_search.best_estimator_)
    evaluate_model(ann_most_acc, tuning, scoring, X_train, y_train, X_test,␣
↪y_test, model_type)
```

```
Fitting 5 folds for each of 600 candidates, totalling 3000 fits
```

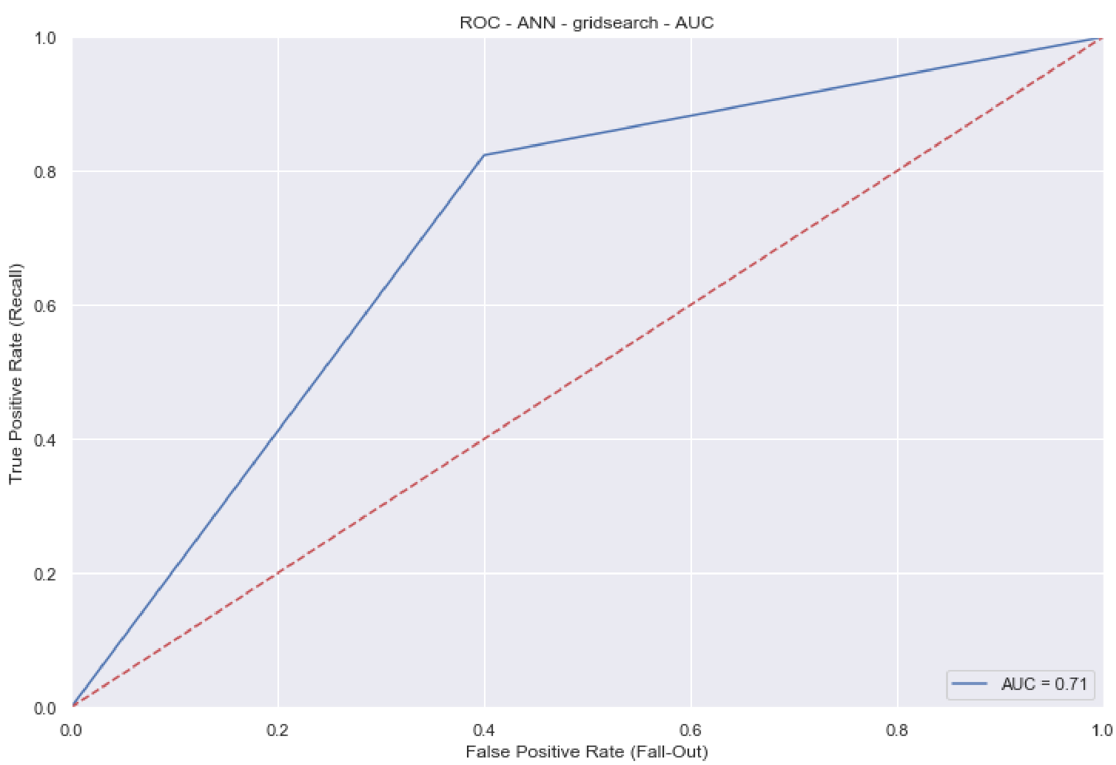
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 20 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 368 tasks      | elapsed:    2.9s
[Parallel(n_jobs=-1)]: Done 1180 tasks      | elapsed:    9.0s
[Parallel(n_jobs=-1)]: Done 2312 tasks      | elapsed:   17.6s
[Parallel(n_jobs=-1)]: Done 3000 out of 3000 | elapsed:   22.5s finished
C:\Users\op97dan\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\neural_network\multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Best parameters:
mlp__alpha: 0.001
mlp__hidden_layer_sizes: (16, 18)

ANN Model Results:
Training Accuracy: 0.795    Test Accuracy: 0.773    AUC: 0.712

[[ 3  2]
 [ 3 14]]
```




```
Classification Report - ANN - gridsearch - AUC
Train:
```

```
              precision    recall  f1-score   support

           0       0.85      0.53      0.65        32
           1       0.78      0.95      0.85        56

    accuracy                           0.80        88
   macro avg       0.81      0.74      0.75        88
weighted avg       0.81      0.80      0.78        88

Test:
              precision    recall  f1-score   support

           0       0.50      0.60      0.55         5
           1       0.88      0.82      0.85        17

    accuracy                           0.77        22
   macro avg       0.69      0.71      0.70        22
weighted avg       0.79      0.77      0.78        22
```



[59]:
```
# kmeans for future analysis
#from sklearn.cluster import KMeans
#from sklearn import metrics
```

```
#import matplotlib.pyplot as plt
#import seaborn as sns

#kmeans_model = KMeans(n_clusters=2).fit(X)
#centroids = kmeans_model.cluster_centers_
#print(centroids)

#plt.scatter(X['SupPol'], X['SupFair'], c=kmeans_model.labels_.astype(float),␣
↪s=50, alpha=0.8)
#plt.scatter(centroids[:, 0], centroids[:, 2], c='red', s=50)
```

5 Model Results

[60]:
```
model_results.head(20)
```

[60]:

	Model	Tuning	Scoring	TrainAcc	TestAcc	AUC
0	Random Forest	manual	accuracy	1.000	0.864	0.841
1	Random Forest	manual	AUC	1.000	0.864	0.841
2	Random Forest	gridsearch	accuracy	0.920	0.864	0.841
3	Random Forest	gridsearch	AUC	0.909	0.818	0.812
4	KNN	manual	AUC	0.830	0.727	0.753
5	SVM	gridsearch	accuracy	0.886	0.727	0.753
6	AdaBoost	gridsearch	accuracy	0.898	0.773	0.712
7	ANN	gridsearch	accuracy	0.795	0.773	0.712
8	ANN	gridsearch	AUC	0.795	0.773	0.712
9	Decision Tree	gridsearch	accuracy	0.761	0.682	0.653
10	Logistic Regression	none	none	0.750	0.727	0.612
11	KNN	gridsearch	accuracy	0.898	0.727	0.612
12	AdaBoost	gridsearch	AUC	0.875	0.682	0.582
13	Decision Tree	gridsearch	AUC	0.773	0.545	0.565
14	SVM	gridsearch	AUC	0.636	0.773	0.500

6 Feature Importance Summary

[61]:
```
importance_compare
```

[61]:

	boruta_rank	dt_importance	rf_importance	permutation	drop_col
SupPol	10	0.000000	0.037285	0.045455	0.090909
SupInf	11	0.000000	0.046542	0.000000	0.090909
SupFair	2	0.000000	0.073546	0.045455	0.045455
SupRec	1	0.230962	0.079908	0.000000	0.045455
SupCoop	5	0.128651	0.056299	0.045455	0.045455
SupResolve	3	0.000000	0.066679	0.000000	0.000000
SupTrn	7	0.000000	0.043195	0.000000	0.045455
DeptCom	1	0.411322	0.111713	0.136364	0.181818

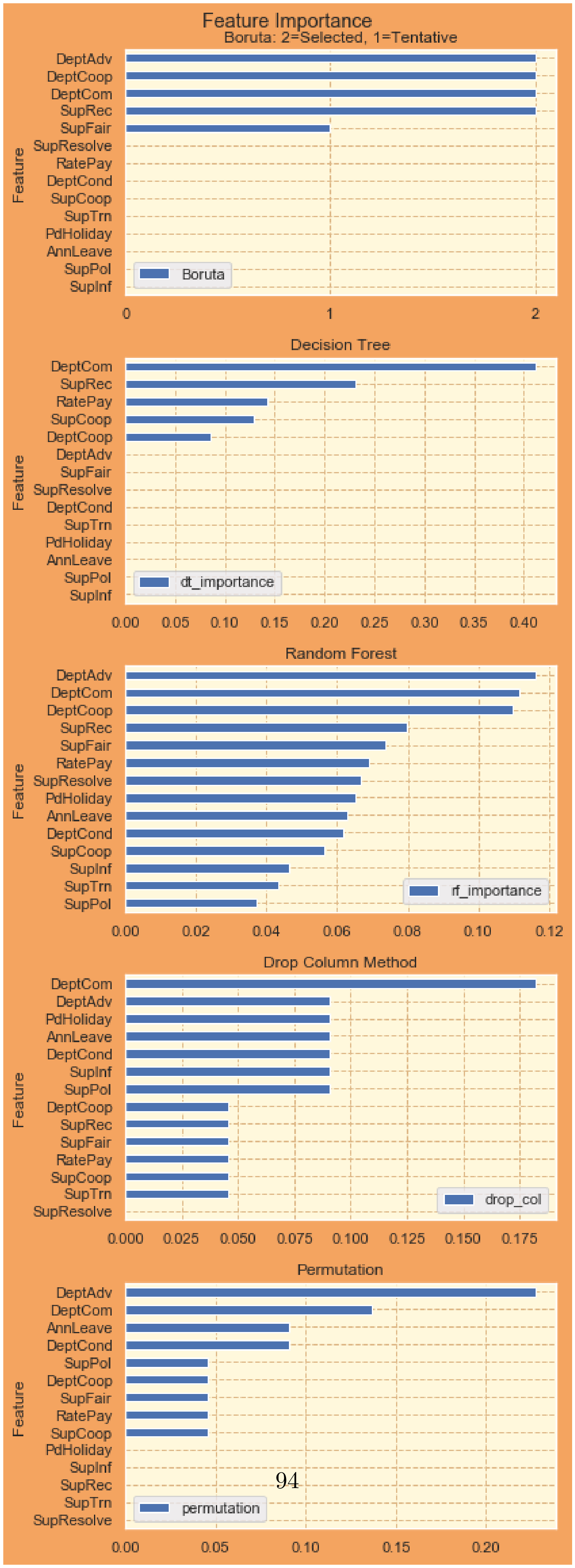
DeptCond	5	0.000000	0.061862	0.090909	0.090909
DeptCoop	1	0.086129	0.109654	0.045455	0.045455
DeptAdv	1	0.000000	0.116103	0.227273	0.090909
RatePay	4	0.142935	0.069208	0.045455	0.045455
AnnLeave	9	0.000000	0.062718	0.090909	0.090909
PdHoliday	8	0.000000	0.065288	0.000000	0.090909

[62]:
```
def boruta_plot_val(row):
    if (row['boruta_rank'] == 1):
        plot_value = 2
    elif (row['boruta_rank'] == 2):
        plot_value = 1
    else:
        plot_value = 0
    return plot_value;
```

[63]:
```
importance_compare = importance_compare.reset_index()
importance_compare.rename(columns={'index':'Feature'}, inplace=True)
importance_compare = importance_compare.sort_values(by='boruta_rank',␣
 ↪ascending=False)
importance_compare['Boruta'] = importance_compare.apply(lambda row:␣
 ↪boruta_plot_val(row), axis=1)
```

[64]:
```
subplot_facecolor = 'cornsilk'
grid_color = 'burlywood'
fig, axes = plt.subplots(5, 1, subplot_kw=dict(polar=False), sharex=False,␣
 ↪figsize=(6, 16))
fig.suptitle('Feature Importance', y=1)
fig.set_facecolor('sandybrown')
#axes[0].xaxis.set_visible(False)

importance_compare.plot.barh(ax=axes[0], x='Feature', y='Boruta')
axes[0].xaxis.set_major_locator(plt.MultipleLocator(1))
axes[0].set_title('Boruta: 2=Selected, 1=Tentative')

importance_compare = importance_compare.sort_values(by='dt_importance',␣
 ↪ascending=True)
importance_compare.plot.barh(ax=axes[1], x='Feature', y='dt_importance')
axes[1].set_title('Decision Tree')

importance_compare = importance_compare.sort_values(by='rf_importance',␣
 ↪ascending=True)
importance_compare.plot.barh(ax=axes[2], x='Feature', y='rf_importance')
axes[2].set_title('Random Forest')

importance_compare = importance_compare.sort_values(by='drop_col',␣
 ↪ascending=True)
```

```
importance_compare.plot.barh(ax=axes[3], x='Feature', y='drop_col')
axes[3].set_title('Drop Column Method')

importance_compare = importance_compare.sort_values(by='permutation',␣
↪ascending=True)
importance_compare.plot.barh(ax=axes[4], x='Feature', y='permutation')
axes[4].set_title('Permutation')

for i in [0,1,2,3,4]:
    axes[i].grid(b=True, which='major', color=grid_color, linestyle='--')
    axes[i].set_facecolor(subplot_facecolor)
plt.tight_layout()
```


Feature Importance
Boruta: 2=Selected, 1=Tentative
Boruta
Decision Tree
dt_importance
Random Forest
rf_importance
Drop Column Method
drop_col
Permutation
permutation
Feature
DeptAdv
DeptCoop
DeptCom
SupRec
SupFair
SupResolve
RatePay
DeptCond
SupCoop
SupTrn
PdHoliday
AnnLeave
SupPol
SupInf