# Model_Tirocinio

July 19, 2021

[1]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn import svm
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
%matplotlib inline
import pickle
from sklearn import tree
import numpy as np
from sklearn.model_selection import GridSearchCV
from matplotlib import pyplot
from sklearn.inspection import permutation_importance
import seaborn as sns
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings("ignore")

def plot_cv_results(cv_results, param_x, param_z, metric='mean_test_score'):
    """
    cv_results - cv_results_ attribute of a GridSearchCV instance (or similar)
    param_x - name of grid search parameter to plot on x axis
    param_z - name of grid search parameter to plot by line color
    """
    cv_results = pd.DataFrame(cv_results)
    col_x = 'param_' + param_x
    col_z = 'param_' + param_z
    fig, ax = plt.subplots(1, 1, figsize=(11, 8))
```

```
    sns.pointplot(x=col_x, y=metric, hue=col_z, data=cv_results, ci=99,␣
 ↪n_boot=64, ax=ax)
    ax.set_title("CV Grid Search Results")
    ax.set_xlabel(param_x)
    ax.set_ylabel(metric)
    ax.legend(title=param_z)
    return fig
def plotonlyC_cv_results(cv_results, param_x, metric='mean_test_score'):
    """
    cv_results - cv_results_ attribute of a GridSearchCV instance (or similar)
    param_x - name of grid search parameter to plot on x axis
    param_z - name of grid search parameter to plot by line color
    """
    cv_results = pd.DataFrame(cv_results)
    col_x = 'param_' + param_x
    fig, ax = plt.subplots(1, 1, figsize=(11, 8))
    sns.pointplot(x=col_x, y=metric, data=cv_results, ci=99, n_boot=64, ax=ax)
    ax.set_title("CV Grid Search Results")
    ax.set_xlabel(param_x)

    return fig
```

[2]:
```
from sklearn.linear_model import LogisticRegression
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', None)
```

[3]:
```
data=pd.read_csv(r'C:\Users\Alessandro\Desktop\provaARFF\fileAARF.csv', sep=',')
```

[4]:
```
data.head()
```

[4]:
```
         meanX        meanY        meanZ      meanHB     meanBR    MeanPos  \
0  2008.427891  2058.039396  2103.906217   65.244186  14.838372 -54.848837
1  2013.012667  2053.901000  2105.963000   55.966667   8.103333 -59.466667
2  2014.027647  2053.500000  2104.404118   61.529412  22.567647 -58.941176
3  2014.306471  2051.794706  2104.563824   61.823529  20.038235 -61.323529
4  2024.815610  2044.073415  2115.720000  104.219512  10.492683 -71.000000

   Xzero  Yzero  Zzero         VarX        VarY        VarZ   Time  maxPos  \
0     59     62     54  1193.142185  123.193218  706.889636  86.05   -31.0
1     20     19     20  1255.106506   93.559199  794.514964  30.00   -35.0
2     18     22     20  1392.578647   95.840588  936.901395  34.00   -32.0
3     20     23     20  1371.921370   74.270207  949.737691  34.00   -34.0
4     40     19     40   946.144049   47.727049  382.041600  41.00   -55.0

   minPos    maxX    minX    maxY    minY    maxZ    minZ       Class
```

```
0   -71.0  2073.0  1960.0  2082.0  2037.0  2178.0  2045.0  Addominali
1   -96.0  2071.0  1964.0  2082.0  2038.0  2190.0  2048.0  Addominali
2   -94.0  2077.0  1963.0  2082.0  2023.0  2224.0  2031.0  Addominali
3   -93.0  2082.0  1963.0  2087.0  1991.0  2213.0  2032.0  Addominali
4   -88.0  2077.0  1968.0  2070.0  2024.0  2178.0  2069.0  Addominali
```

[5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 516 entries, 0 to 515
Data columns (total 22 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   meanX    516 non-null    float64
 1   meanY    516 non-null    float64
 2   meanZ    516 non-null    float64
 3   meanHB   516 non-null    float64
 4   meanBR   516 non-null    float64
 5   MeanPos  516 non-null    float64
 6   Xzero    516 non-null    int64
 7   Yzero    516 non-null    int64
 8   Zzero    516 non-null    int64
 9   VarX     516 non-null    float64
 10  VarY     516 non-null    float64
 11  VarZ     516 non-null    float64
 12  Time     516 non-null    float64
 13  maxPos   516 non-null    float64
 14  minPos   516 non-null    float64
 15  maxX     516 non-null    float64
 16  minX     516 non-null    float64
 17  maxY     516 non-null    float64
 18  minY     516 non-null    float64
 19  maxZ     516 non-null    float64
 20  minZ     516 non-null    float64
 21  Class    516 non-null    object
dtypes: float64(18), int64(3), object(1)
memory usage: 88.8+ KB
```

[6]:
```
data.isnull().sum()
```

[6]:
```
meanX      0
meanY      0
meanZ      0
meanHB     0
meanBR     0
MeanPos    0
Xzero      0
```

```
Yzero      0
Zzero      0
VarX       0
VarY       0
VarZ       0
Time       0
maxPos     0
minPos     0
maxX       0
minX       0
maxY       0
minY       0
maxZ       0
minZ       0
Class      0
dtype: int64
```

[7]:
```
Label=data['Class'].unique()
data['Class'].unique()
```

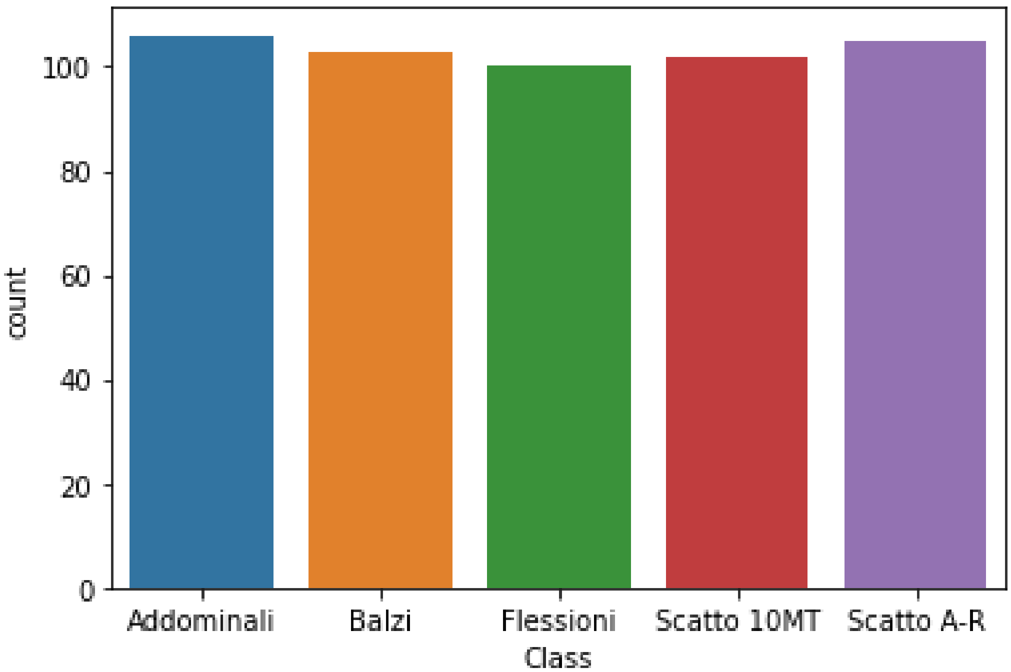
[7]:
```
array(['Addominali', 'Balzi', 'Flessioni', 'Scatto 10MT', 'Scatto A-R'],
      dtype=object)
```

[8]:
```
data['Class'].value_counts()
```

[8]:
```
Addominali     106
Scatto A-R     105
Balzi          103
Scatto 10MT    102
Flessioni      100
Name: Class, dtype: int64
```

[9]:
```
sns.countplot(data['Class'])
```

[9]:
```
<AxesSubplot:xlabel='Class', ylabel='count'>
```




[10]:
```
label_class= LabelEncoder()
```

[11]:
```
label_genere= LabelEncoder()
```

[12]:
```
data['Class']=label_class.fit_transform(data['Class'])
```

[13]:
```
data.head(20)
```

[13]:
```
          meanX        meanY        meanZ      meanHB     meanBR    MeanPos  \
0   2008.427891  2058.039396  2103.906217   65.244186  14.838372 -54.848837
1   2013.012667  2053.901000  2105.963000   55.966667   8.103333 -59.466667
2   2014.027647  2053.500000  2104.404118   61.529412  22.567647 -58.941176
3   2014.306471  2051.794706  2104.563824   61.823529  20.038235 -61.323529
4   2024.815610  2044.073415  2115.720000  104.219512  10.492683 -71.000000
5   2016.042376  2044.220198  2108.527525  104.705882  20.649020 -62.352941
6   2016.139029  2043.190291  2109.035728  109.627451  16.888235 -62.098039
7   2005.954141  2055.136364  2077.586061   93.448980  14.397959 -33.428571
8   2006.933333  2053.238180  2079.213650   94.020000  22.272000 -35.300000
9   2007.954949  2052.552323  2083.253535   94.680000  20.388000 -40.640000
10  2008.424800  2050.085867  2102.408533  104.157895  17.757895 -54.526316
11  2009.172184  2050.571954  2103.185057  101.186047  17.153488 -54.395349
12  2009.457179  2051.845641  2102.937179  100.769231  16.771795 -55.333333
13  2010.194030  2048.829851  2092.597313  117.970588   9.285294 -49.529412
14  2008.010475  2046.316687  2091.416565  115.585366  13.578049 -46.731707
15  2006.413456  2046.980836  2090.078695  114.081633  18.875510 -45.061224
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 16 | 2001.802853 | 2044.423865 | 2080.749935 | 72.512821 | 17.951282 | -36.435897 |
| 17 | 2003.046702 | 2046.548663 | 2083.756863 | 92.607143 | 22.771429 | -38.000000 |
| 18 | 2000.948019 | 2045.009192 | 2082.543582 | 103.516129 | 20.825806 | -35.161290 |
| 19 | 2012.439259 | 2050.293086 | 2104.257284 | 96.975610 | 10.014634 | -58.219512 |

| | Xzero | Yzero | Zzero | VarX | VarY | VarZ | Time | maxPos \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 59 | 62 | 54 | 1193.142185 | 123.193218 | 706.889636 | 86.05 | -31.0 |
| 1 | 20 | 19 | 20 | 1255.106506 | 93.559199 | 794.514964 | 30.00 | -35.0 |
| 2 | 18 | 22 | 20 | 1392.578647 | 95.840588 | 936.901395 | 34.00 | -32.0 |
| 3 | 20 | 23 | 20 | 1371.921370 | 74.270207 | 949.737691 | 34.00 | -34.0 |
| 4 | 40 | 19 | 40 | 946.144049 | 47.727049 | 382.041600 | 41.00 | -55.0 |
| 5 | 40 | 48 | 40 | 1222.090481 | 53.973295 | 668.888054 | 50.50 | -46.0 |
| 6 | 46 | 57 | 40 | 1189.293681 | 50.352139 | 688.491927 | 51.50 | -31.0 |
| 7 | 39 | 47 | 39 | 1063.335473 | 139.845849 | 3262.370270 | 49.50 | -9.0 |
| 8 | 40 | 62 | 40 | 1075.078642 | 118.554546 | 3245.838033 | 50.55 | -7.0 |
| 9 | 40 | 77 | 39 | 1225.390496 | 147.402818 | 2836.488245 | 49.50 | -14.0 |
| 10 | 30 | 32 | 30 | 1124.346745 | 79.368094 | 1000.739234 | 37.50 | -37.0 |
| 11 | 34 | 34 | 34 | 1151.977939 | 122.736777 | 928.217018 | 43.50 | -37.0 |
| 12 | 32 | 32 | 31 | 1164.881500 | 124.835148 | 963.326566 | 39.00 | -36.0 |
| 13 | 26 | 32 | 26 | 1522.979069 | 105.736422 | 1874.502023 | 33.50 | -29.0 |
| 14 | 30 | 36 | 30 | 1474.906833 | 91.204460 | 1808.040359 | 41.05 | -22.0 |
| 15 | 32 | 36 | 34 | 1375.276149 | 101.161305 | 1879.880046 | 49.05 | -18.0 |
| 16 | 42 | 51 | 38 | 815.230653 | 94.260286 | 3394.947584 | 38.55 | -7.0 |
| 17 | 28 | 39 | 28 | 815.034896 | 113.513229 | 3132.859957 | 28.05 | -13.0 |
| 18 | 30 | 40 | 30 | 811.506648 | 92.686762 | 2899.529242 | 31.55 | -12.0 |
| 19 | 29 | 28 | 28 | 1291.502607 | 110.244224 | 817.300719 | 40.50 | -39.0 |

| | minPos | maxX | minX | maxY | minY | maxZ | minZ | Class |
|---|---|---|---|---|---|---|---|---|
| 0 | -71.0 | 2073.0 | 1960.0 | 2082.0 | 2037.0 | 2178.0 | 2045.0 | 0 |
| 1 | -96.0 | 2071.0 | 1964.0 | 2082.0 | 2038.0 | 2190.0 | 2048.0 | 0 |
| 2 | -94.0 | 2077.0 | 1963.0 | 2082.0 | 2023.0 | 2224.0 | 2031.0 | 0 |
| 3 | -93.0 | 2082.0 | 1963.0 | 2087.0 | 1991.0 | 2213.0 | 2032.0 | 0 |
| 4 | -88.0 | 2077.0 | 1968.0 | 2070.0 | 2024.0 | 2178.0 | 2069.0 | 0 |
| 5 | -102.0 | 2087.0 | 1969.0 | 2073.0 | 2022.0 | 2262.0 | 2060.0 | 0 |
| 6 | -80.0 | 2099.0 | 1962.0 | 2081.0 | 1997.0 | 2267.0 | 2058.0 | 0 |
| 7 | -58.0 | 2104.0 | 1925.0 | 2101.0 | 2030.0 | 2176.0 | 1938.0 | 0 |
| 8 | -57.0 | 2104.0 | 1888.0 | 2105.0 | 2016.0 | 2194.0 | 1952.0 | 0 |
| 9 | -66.0 | 2110.0 | 1943.0 | 2100.0 | 2024.0 | 2187.0 | 1968.0 | 0 |
| 10 | -76.0 | 2070.0 | 1961.0 | 2075.0 | 2037.0 | 2179.0 | 2034.0 | 0 |
| 11 | -67.0 | 2070.0 | 1962.0 | 2084.0 | 2034.0 | 2184.0 | 2041.0 | 0 |
| 12 | -78.0 | 2085.0 | 1965.0 | 2079.0 | 2037.0 | 2183.0 | 2025.0 | 0 |
| 13 | -77.0 | 2093.0 | 1956.0 | 2082.0 | 2030.0 | 2200.0 | 2004.0 | 0 |
| 14 | -72.0 | 2099.0 | 1956.0 | 2077.0 | 2030.0 | 2188.0 | 2001.0 | 0 |
| 15 | -99.0 | 2096.0 | 1956.0 | 2076.0 | 2026.0 | 2207.0 | 1998.0 | 0 |
| 16 | -101.0 | 2078.0 | 1948.0 | 2111.0 | 1950.0 | 2306.0 | 1939.0 | 0 |
| 17 | -69.0 | 2078.0 | 1956.0 | 2098.0 | 2004.0 | 2243.0 | 1958.0 | 0 |
| 18 | -56.0 | 2100.0 | 1954.0 | 2088.0 | 1982.0 | 2230.0 | 1963.0 | 0 |

```
19  -104.0  2081.0  1965.0  2072.0  2035.0  2172.0  2043.0      0
```

[14]:
```
X= data.drop('Class',axis=1)
y=data['Class']
```

[15]:
```
data['Class'].value_counts()
```

[15]:
```
0    106
4    105
1    103
3    102
2    100
Name: Class, dtype: int64
```

[16]:
```
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.20,␣
 ↪random_state=42)
```

[17]:
```
sc= StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

# 1 RANDOM FOREST CLASSIFIER

[18]:
```
#Con grid Search
rfc=RandomForestClassifier(random_state=42)
param_grid = {
    'n_estimators': [20, 50, 100, 200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid,refit=True, cv= 5,␣
 ↪return_train_score=True)
#fitting
CV_rfc.fit(X_train, y_train)
#Parametri Migliori
CV_rfc.best_params_
```

[18]:
```
{'criterion': 'gini',
 'max_depth': 7,
 'max_features': 'auto',
 'n_estimators': 500}
```

[46]:
```
#Prestazioni di tutte le combinazioni
```

```python
x=pd.concat([pd.DataFrame(CV_rfc.cv_results_["params"]),pd.DataFrame(CV_rfc.
 ↪cv_results_["mean_train_score"], columns=["Training"]),pd.DataFrame(CV_rfc.
 ↪cv_results_["mean_test_score"], columns=["Validation"])],axis=1)
print(x)
```

```
   criterion  max_depth max_features  n_estimators  Training  Validation
0       gini          4         auto            20  0.983019    0.982986
1       gini          4         auto            50  0.992116    0.982956
2       gini          4         auto           100  0.992718    0.985395
3       gini          4         auto           200  0.993932    0.985395
4       gini          4         auto           500  0.993326    0.987834
5       gini          4         sqrt            20  0.983019    0.982986
6       gini          4         sqrt            50  0.992116    0.982956
7       gini          4         sqrt           100  0.992718    0.985395
8       gini          4         sqrt           200  0.993932    0.985395
9       gini          4         sqrt           500  0.993326    0.987834
10      gini          4         log2            20  0.983019    0.982986
11      gini          4         log2            50  0.992116    0.982956
12      gini          4         log2           100  0.992718    0.985395
13      gini          4         log2           200  0.993932    0.985395
14      gini          4         log2           500  0.993326    0.987834
15      gini          5         auto            20  0.993328    0.978108
16      gini          5         auto            50  0.994540    0.982986
17      gini          5         auto           100  0.994538    0.987834
18      gini          5         auto           200  0.995148    0.985395
19      gini          5         auto           500  0.995754    0.987834
20      gini          5         sqrt            20  0.993328    0.978108
21      gini          5         sqrt            50  0.994540    0.982986
22      gini          5         sqrt           100  0.994538    0.987834
23      gini          5         sqrt           200  0.995148    0.985395
24      gini          5         sqrt           500  0.995754    0.987834
25      gini          5         log2            20  0.993328    0.978108
26      gini          5         log2            50  0.994540    0.982986
27      gini          5         log2           100  0.994538    0.987834
28      gini          5         log2           200  0.995148    0.985395
29      gini          5         log2           500  0.995754    0.987834
30      gini          6         auto            20  0.996966    0.982956
31      gini          6         auto            50  0.998180    0.987834
32      gini          6         auto           100  0.999392    0.985395
33      gini          6         auto           200  1.000000    0.987834
34      gini          6         auto           500  1.000000    0.987834
35      gini          6         sqrt            20  0.996966    0.982956
36      gini          6         sqrt            50  0.998180    0.987834
37      gini          6         sqrt           100  0.999392    0.985395
38      gini          6         sqrt           200  1.000000    0.987834
39      gini          6         sqrt           500  1.000000    0.987834
40      gini          6         log2            20  0.996966    0.982956
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 41 | gini | 6 | log2 | 50 | 0.998180 | 0.987834 |
| 42 | gini | 6 | log2 | 100 | 0.999392 | 0.985395 |
| 43 | gini | 6 | log2 | 200 | 1.000000 | 0.987834 |
| 44 | gini | 6 | log2 | 500 | 1.000000 | 0.987834 |
| 45 | gini | 7 | auto | 20 | 0.999394 | 0.982956 |
| 46 | gini | 7 | auto | 50 | 1.000000 | 0.987834 |
| 47 | gini | 7 | auto | 100 | 1.000000 | 0.985395 |
| 48 | gini | 7 | auto | 200 | 1.000000 | 0.987834 |
| 49 | gini | 7 | auto | 500 | 1.000000 | 0.990273 |
| 50 | gini | 7 | sqrt | 20 | 0.999394 | 0.982956 |
| 51 | gini | 7 | sqrt | 50 | 1.000000 | 0.987834 |
| 52 | gini | 7 | sqrt | 100 | 1.000000 | 0.985395 |
| 53 | gini | 7 | sqrt | 200 | 1.000000 | 0.987834 |
| 54 | gini | 7 | sqrt | 500 | 1.000000 | 0.990273 |
| 55 | gini | 7 | log2 | 20 | 0.999394 | 0.982956 |
| 56 | gini | 7 | log2 | 50 | 1.000000 | 0.987834 |
| 57 | gini | 7 | log2 | 100 | 1.000000 | 0.985395 |
| 58 | gini | 7 | log2 | 200 | 1.000000 | 0.987834 |
| 59 | gini | 7 | log2 | 500 | 1.000000 | 0.990273 |
| 60 | gini | 8 | auto | 20 | 1.000000 | 0.982956 |
| 61 | gini | 8 | auto | 50 | 1.000000 | 0.985395 |
| 62 | gini | 8 | auto | 100 | 1.000000 | 0.985395 |
| 63 | gini | 8 | auto | 200 | 1.000000 | 0.987834 |
| 64 | gini | 8 | auto | 500 | 1.000000 | 0.990273 |
| 65 | gini | 8 | sqrt | 20 | 1.000000 | 0.982956 |
| 66 | gini | 8 | sqrt | 50 | 1.000000 | 0.985395 |
| 67 | gini | 8 | sqrt | 100 | 1.000000 | 0.985395 |
| 68 | gini | 8 | sqrt | 200 | 1.000000 | 0.987834 |
| 69 | gini | 8 | sqrt | 500 | 1.000000 | 0.990273 |
| 70 | gini | 8 | log2 | 20 | 1.000000 | 0.982956 |
| 71 | gini | 8 | log2 | 50 | 1.000000 | 0.985395 |
| 72 | gini | 8 | log2 | 100 | 1.000000 | 0.985395 |
| 73 | gini | 8 | log2 | 200 | 1.000000 | 0.987834 |
| 74 | gini | 8 | log2 | 500 | 1.000000 | 0.990273 |
| 75 | entropy | 4 | auto | 20 | 0.992718 | 0.970820 |
| 76 | entropy | 4 | auto | 50 | 0.993324 | 0.985395 |
| 77 | entropy | 4 | auto | 100 | 0.993324 | 0.987834 |
| 78 | entropy | 4 | auto | 200 | 0.992718 | 0.985395 |
| 79 | entropy | 4 | auto | 500 | 0.993326 | 0.985395 |
| 80 | entropy | 4 | sqrt | 20 | 0.992718 | 0.970820 |
| 81 | entropy | 4 | sqrt | 50 | 0.993324 | 0.985395 |
| 82 | entropy | 4 | sqrt | 100 | 0.993324 | 0.987834 |
| 83 | entropy | 4 | sqrt | 200 | 0.992718 | 0.985395 |
| 84 | entropy | 4 | sqrt | 500 | 0.993326 | 0.985395 |
| 85 | entropy | 4 | log2 | 20 | 0.992718 | 0.970820 |
| 86 | entropy | 4 | log2 | 50 | 0.993324 | 0.985395 |
| 87 | entropy | 4 | log2 | 100 | 0.993324 | 0.987834 |
| 88 | entropy | 4 | log2 | 200 | 0.992718 | 0.985395 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 89 | entropy | 4 | log2 | 500 | 0.993326 | 0.985395 |
| 90 | entropy | 5 | auto | 20 | 0.994540 | 0.980547 |
| 91 | entropy | 5 | auto | 50 | 0.995146 | 0.982956 |
| 92 | entropy | 5 | auto | 100 | 0.996362 | 0.987834 |
| 93 | entropy | 5 | auto | 200 | 0.996968 | 0.985395 |
| 94 | entropy | 5 | auto | 500 | 0.996968 | 0.985395 |
| 95 | entropy | 5 | sqrt | 20 | 0.994540 | 0.980547 |
| 96 | entropy | 5 | sqrt | 50 | 0.995146 | 0.982956 |
| 97 | entropy | 5 | sqrt | 100 | 0.996362 | 0.987834 |
| 98 | entropy | 5 | sqrt | 200 | 0.996968 | 0.985395 |
| 99 | entropy | 5 | sqrt | 500 | 0.996968 | 0.985395 |
| 100 | entropy | 5 | log2 | 20 | 0.994540 | 0.980547 |
| 101 | entropy | 5 | log2 | 50 | 0.995146 | 0.982956 |
| 102 | entropy | 5 | log2 | 100 | 0.996362 | 0.987834 |
| 103 | entropy | 5 | log2 | 200 | 0.996968 | 0.985395 |
| 104 | entropy | 5 | log2 | 500 | 0.996968 | 0.985395 |
| 105 | entropy | 6 | auto | 20 | 0.999394 | 0.978108 |
| 106 | entropy | 6 | auto | 50 | 0.998786 | 0.987834 |
| 107 | entropy | 6 | auto | 100 | 1.000000 | 0.987834 |
| 108 | entropy | 6 | auto | 200 | 1.000000 | 0.982956 |
| 109 | entropy | 6 | auto | 500 | 1.000000 | 0.985395 |
| 110 | entropy | 6 | sqrt | 20 | 0.999394 | 0.978108 |
| 111 | entropy | 6 | sqrt | 50 | 0.998786 | 0.987834 |
| 112 | entropy | 6 | sqrt | 100 | 1.000000 | 0.987834 |
| 113 | entropy | 6 | sqrt | 200 | 1.000000 | 0.982956 |
| 114 | entropy | 6 | sqrt | 500 | 1.000000 | 0.985395 |
| 115 | entropy | 6 | log2 | 20 | 0.999394 | 0.978108 |
| 116 | entropy | 6 | log2 | 50 | 0.998786 | 0.987834 |
| 117 | entropy | 6 | log2 | 100 | 1.000000 | 0.987834 |
| 118 | entropy | 6 | log2 | 200 | 1.000000 | 0.982956 |
| 119 | entropy | 6 | log2 | 500 | 1.000000 | 0.985395 |
| 120 | entropy | 7 | auto | 20 | 1.000000 | 0.980517 |
| 121 | entropy | 7 | auto | 50 | 1.000000 | 0.987834 |
| 122 | entropy | 7 | auto | 100 | 1.000000 | 0.987834 |
| 123 | entropy | 7 | auto | 200 | 1.000000 | 0.985395 |
| 124 | entropy | 7 | auto | 500 | 1.000000 | 0.987834 |
| 125 | entropy | 7 | sqrt | 20 | 1.000000 | 0.980517 |
| 126 | entropy | 7 | sqrt | 50 | 1.000000 | 0.987834 |
| 127 | entropy | 7 | sqrt | 100 | 1.000000 | 0.987834 |
| 128 | entropy | 7 | sqrt | 200 | 1.000000 | 0.985395 |
| 129 | entropy | 7 | sqrt | 500 | 1.000000 | 0.987834 |
| 130 | entropy | 7 | log2 | 20 | 1.000000 | 0.980517 |
| 131 | entropy | 7 | log2 | 50 | 1.000000 | 0.987834 |
| 132 | entropy | 7 | log2 | 100 | 1.000000 | 0.987834 |
| 133 | entropy | 7 | log2 | 200 | 1.000000 | 0.985395 |
| 134 | entropy | 7 | log2 | 500 | 1.000000 | 0.987834 |
| 135 | entropy | 8 | auto | 20 | 1.000000 | 0.982956 |
| 136 | entropy | 8 | auto | 50 | 1.000000 | 0.987834 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 137 | entropy | 8 | auto | 100 | 1.000000 | 0.987834 |
| 138 | entropy | 8 | auto | 200 | 1.000000 | 0.982956 |
| 139 | entropy | 8 | auto | 500 | 1.000000 | 0.990273 |
| 140 | entropy | 8 | sqrt | 20 | 1.000000 | 0.982956 |
| 141 | entropy | 8 | sqrt | 50 | 1.000000 | 0.987834 |
| 142 | entropy | 8 | sqrt | 100 | 1.000000 | 0.987834 |
| 143 | entropy | 8 | sqrt | 200 | 1.000000 | 0.982956 |
| 144 | entropy | 8 | sqrt | 500 | 1.000000 | 0.990273 |
| 145 | entropy | 8 | log2 | 20 | 1.000000 | 0.982956 |
| 146 | entropy | 8 | log2 | 50 | 1.000000 | 0.987834 |
| 147 | entropy | 8 | log2 | 100 | 1.000000 | 0.987834 |
| 148 | entropy | 8 | log2 | 200 | 1.000000 | 0.982956 |
| 149 | entropy | 8 | log2 | 500 | 1.000000 | 0.990273 |

[43]:
```
CV_rfc.cv_results_["mean_train_score"]
CV_rfc.best_score_
```

[43]:
```
array([0.98301925, 0.99211569, 0.99271806, 0.99393203, 0.99332596,
       0.98301925, 0.99211569, 0.99271806, 0.99393203, 0.99332596,
       0.98301925, 0.99211569, 0.99271806, 0.99393203, 0.99332596,
       0.99332781, 0.99453993, 0.99453809, 0.99514783, 0.99575389,
       0.99332781, 0.99453993, 0.99453809, 0.99514783, 0.99575389,
       0.99332781, 0.99453993, 0.99453809, 0.99514783, 0.99575389,
       0.99696601, 0.99817998, 0.9993921 , 1.        , 1.        ,
       0.99696601, 0.99817998, 0.9993921 , 1.        , 1.        ,
       0.99696601, 0.99817998, 0.9993921 , 1.        , 1.        ,
       0.99939394, 1.        , 1.        , 1.        , 1.        ,
       0.99939394, 1.        , 1.        , 1.        , 1.        ,
       0.99939394, 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       0.99271806, 0.99332412, 0.99332412, 0.99271806, 0.99332596,
       0.99271806, 0.99332412, 0.99332412, 0.99271806, 0.99332596,
       0.99271806, 0.99332412, 0.99332412, 0.99271806, 0.99332596,
       0.99453993, 0.99514599, 0.99636179, 0.99696785, 0.99696785,
       0.99453993, 0.99514599, 0.99636179, 0.99696785, 0.99696785,
       0.99453993, 0.99514599, 0.99636179, 0.99696785, 0.99696785,
       0.99939394, 0.99878604, 1.        , 1.        , 1.        ,
       0.99939394, 0.99878604, 1.        , 1.        , 1.        ,
       0.99939394, 0.99878604, 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ])
```

[20]:
```
#Feature Importance
CV_rfc.best_estimator_.feature_importances_
```

[20]:
```
array([0.09089185, 0.01158459, 0.07365019, 0.00514455, 0.00367066,
       0.09889455, 0.05894107, 0.00881113, 0.05930028, 0.07668151,
       0.02464817, 0.02434091, 0.05318446, 0.08593782, 0.05843887,
       0.02148258, 0.06980539, 0.0233242 , 0.01614501, 0.10819311,
       0.0269291 ])
```

[21]:
```
#TEST Miglior Risultato
print(CV_rfc.best_score_)
BestRF=CV_rfc.best_estimator_
grid_predictions = BestRF.predict(X_test)
print(confusion_matrix(y_test,grid_predictions))
print(classification_report(y_test,grid_predictions))
accuracy_score(y_test,grid_predictions)
```

```
0.9902732882750513
[[28  0  0  0  0]
 [ 0 18  0  0  0]
 [ 0  0 21  0  0]
 [ 0  0  0 17  0]
 [ 0  0  0  0 20]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        28
           1       1.00      1.00      1.00        18
           2       1.00      1.00      1.00        21
           3       1.00      1.00      1.00        17
           4       1.00      1.00      1.00        20

    accuracy                           1.00       104
   macro avg       1.00      1.00      1.00       104
weighted avg       1.00      1.00      1.00       104
```

[21]:
```
1.0
```

## 2 SVM

[51]:
```
#Kernel lineare Con Grid Search
param_grid = {'C': [0.1,1, 10, 100]}
grid = GridSearchCV(SVC(kernel='linear'),param_grid,refit=True,verbose=0,␣
 ↪return_train_score=True)
grid.fit(X_train,y_train)
#Parametri Migliori
```

```
grid.best_params_
print(grid.best_score_)
```

```
0.9829856009403468
```

[52]:
```
x=pd.concat([pd.DataFrame(grid.cv_results_["params"]),pd.DataFrame(grid.
 ↪cv_results_["mean_train_score"], columns=["Training"]),pd.DataFrame(grid.
 ↪cv_results_["mean_test_score"], columns=["Validation"])],axis=1)
print(x)
```

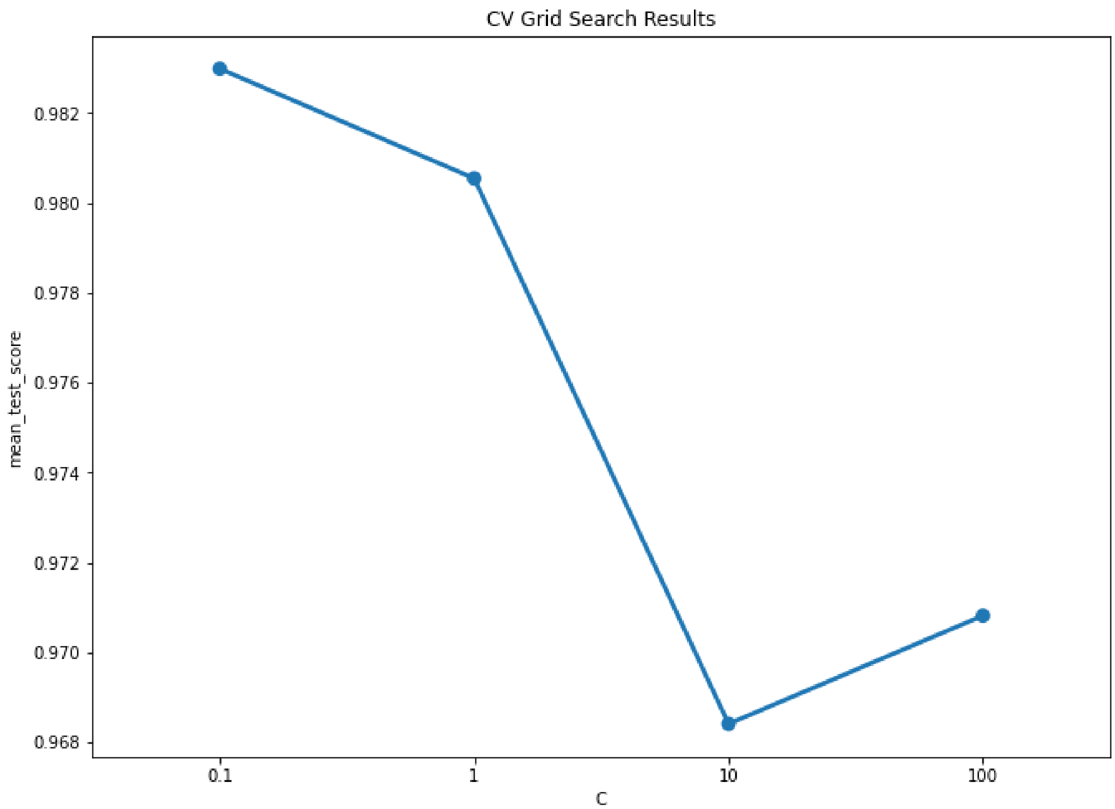
```
       C  Training  Validation
0    0.1  0.988472    0.982986
1    1.0  0.994540    0.980547
2   10.0  0.998180    0.968410
3  100.0  1.000000    0.970820
```

[53]:
```
#Prestazioni di tutte le combinazioni
x=pd.concat([pd.DataFrame(grid.cv_results_["params"]),pd.DataFrame(grid.
 ↪cv_results_["mean_test_score"], columns=["Accuracy"])],axis=1)
print(x)
#TEST Miglior Risultato
print(grid.best_estimator_)
grid_predictions = grid.predict(X_test)
print(confusion_matrix(y_test,grid_predictions))
print(classification_report(y_test,grid_predictions))
fig = plotonlyC_cv_results(grid.cv_results_,'C')
print("Accuracy")
accuracy_score(y_test,grid_predictions)
```

```
       C  Accuracy
0    0.1  0.982986
1    1.0  0.980547
2   10.0  0.968410
3  100.0  0.970820
SVC(C=0.1, kernel='linear')
[[28  0  0  0  0]
 [ 0 18  0  0  0]
 [ 0  0 21  0  0]
 [ 0  0  0 17  0]
 [ 0  0  0  0 20]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        28
           1       1.00      1.00      1.00        18
           2       1.00      1.00      1.00        21
           3       1.00      1.00      1.00        17
           4       1.00      1.00      1.00        20
```

```
    accuracy                           1.00       104
   macro avg       1.00      1.00      1.00       104
weighted avg       1.00      1.00      1.00       104

Accuracy
```

[53]: 1.0




[54]:
```
"""
def f_importances(coef, names):
    imp = coef
    imp,names = zip(*sorted(zip(imp,names)))
    plt.barh(range(len(names)), imp, align='center')
    plt.yticks(range(len(names)), names)
    plt.show()

features_names = ['0', '1','2','3','4','5','6','7',␣
↪'8','9','10','11','12','13','14', '15','16','17','18','19','20']
svm = grid.best_estimator_
f_importances(svm.coef_, features_names)
"""
```

[54]: "\ndef f_importances(coef, names):\n    imp = coef\n    imp,names = zip(*sorted(zip(imp,names)))\n    plt.barh(range(len(names)), imp, align='center')\n    plt.yticks(range(len(names)), names)\n    plt.show()\n\nfeatures_names = ['0', '1','2','3','4','5','6','7', '8','9','10','11','12','13','14', '15','16','17','18','19','20']\nsvm = grid.best_estimator_\nf_importances(svm.coef_, features_names)\n"

[55]:
```
#Polynomial kernel Con Grid Search
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],}
grid =␣
 ↪GridSearchCV(SVC(kernel='poly'),param_grid,refit=True,verbose=0,return_train_score=True)
grid.fit(X_train,y_train)
#Parametri Migliori
print(grid.best_params_)
print(grid.best_score_)
```

```
{'C': 10, 'gamma': 0.1}
0.9854246253305906
```

[56]:
```
x=pd.concat([pd.DataFrame(grid.cv_results_["params"]),pd.DataFrame(grid.
 ↪cv_results_["mean_train_score"], columns=["Training"]),pd.DataFrame(grid.
 ↪cv_results_["mean_test_score"], columns=["Validation"])],axis=1)
print(x)
```

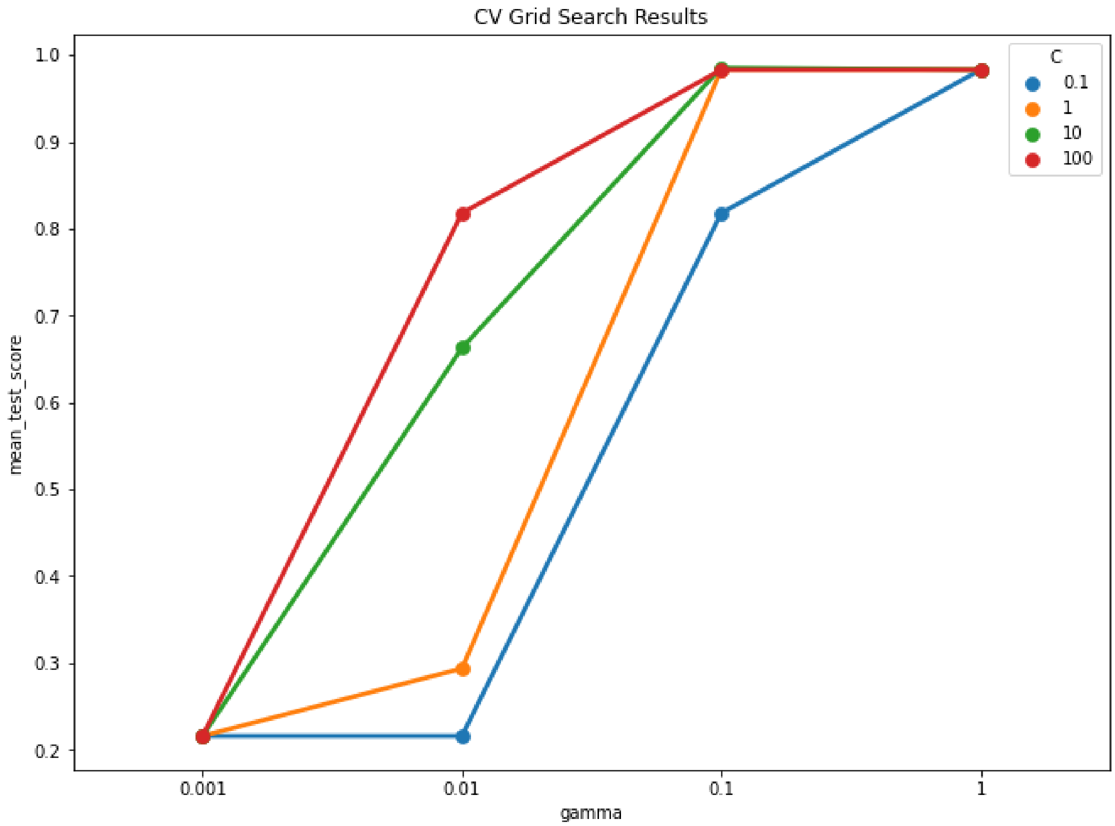
```
        C  gamma  Training  Validation
0     0.1  1.000  1.000000    0.982986
1     0.1  0.100  0.840402    0.818043
2     0.1  0.010  0.216625    0.216015
3     0.1  0.001  0.216625    0.216015
4     1.0  1.000  1.000000    0.982986
5     1.0  0.100  0.996360    0.982986
6     1.0  0.010  0.312521    0.293770
7     1.0  0.001  0.216625    0.216015
8    10.0  1.000  1.000000    0.982986
9    10.0  0.100  1.000000    0.985425
10   10.0  0.010  0.667483    0.662739
11   10.0  0.001  0.216625    0.216015
12  100.0  1.000  1.000000    0.982986
13  100.0  0.100  1.000000    0.982986
14  100.0  0.010  0.839794    0.818043
15  100.0  0.001  0.216625    0.216015
```

[57]:
```
#Prestazioni di tutte le combinazioni
x=pd.concat([pd.DataFrame(grid.cv_results_["params"]),pd.DataFrame(grid.
 ↪cv_results_["mean_test_score"], columns=["Accuracy"])],axis=1)
print(x)
#TEST Miglior Risultato
```

```python
print(grid.best_estimator_)
grid_predictions = grid.predict(X_test)
print(confusion_matrix(y_test,grid_predictions))
print(classification_report(y_test,grid_predictions))
fig = plot_cv_results(grid.cv_results_, 'gamma','C')
print("Accuracy")
accuracy_score(y_test,grid_predictions)
```

```
        C  gamma  Accuracy
0     0.1  1.000  0.982986
1     0.1  0.100  0.818043
2     0.1  0.010  0.216015
3     0.1  0.001  0.216015
4     1.0  1.000  0.982986
5     1.0  0.100  0.982986
6     1.0  0.010  0.293770
7     1.0  0.001  0.216015
8    10.0  1.000  0.982986
9    10.0  0.100  0.985425
10   10.0  0.010  0.662739
11   10.0  0.001  0.216015
12  100.0  1.000  0.982986
13  100.0  0.100  0.982986
14  100.0  0.010  0.818043
15  100.0  0.001  0.216015
SVC(C=10, gamma=0.1, kernel='poly')
[[28  0  0  0  0]
 [ 0 18  0  0  0]
 [ 0  0 21  0  0]
 [ 0  0  0 16  1]
 [ 0  0  0  0 20]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        28
           1       1.00      1.00      1.00        18
           2       1.00      1.00      1.00        21
           3       1.00      0.94      0.97        17
           4       0.95      1.00      0.98        20

    accuracy                           0.99       104
   macro avg       0.99      0.99      0.99       104
weighted avg       0.99      0.99      0.99       104

Accuracy
```

[57]: 0.9903846153846154




[58]:
```
#RBF Kernel Con Grid Search
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001]}
grid =␣
↪GridSearchCV(SVC(kernel='rbf'),param_grid,refit=True,verbose=0,return_train_score=True)
grid.fit(X_train,y_train)
#Parametri Migliori
print(grid.best_params_)
print(grid.best_score_)
```

```
{'C': 100, 'gamma': 0.01}
0.9878930355568617
```

[59]:
```
x=pd.concat([pd.DataFrame(grid.cv_results_["params"]),pd.DataFrame(grid.
↪cv_results_["mean_train_score"], columns=["Training"]),pd.DataFrame(grid.
↪cv_results_["mean_test_score"], columns=["Validation"])],axis=1)
print(x)
```

```
       C  gamma  Training  Validation
0    0.1  1.000  0.604977    0.429621
1    0.1  0.100  0.979372    0.970849
2    0.1  0.010  0.918069    0.915016
```

```
3      0.1  0.001  0.397443    0.395680
4      1.0  1.000  1.000000    0.864061
5      1.0  0.100  0.998178    0.975669
6      1.0  0.010  0.989688    0.985454
7      1.0  0.001  0.823408    0.822892
8     10.0  1.000  1.000000    0.871290
9     10.0  0.100  1.000000    0.978108
10    10.0  0.010  0.992114    0.975698
11    10.0  0.001  0.989688    0.983015
12   100.0  1.000  1.000000    0.871290
13   100.0  0.100  1.000000    0.978108
14   100.0  0.010  1.000000    0.987893
15   100.0  0.001  0.990294    0.978108
```

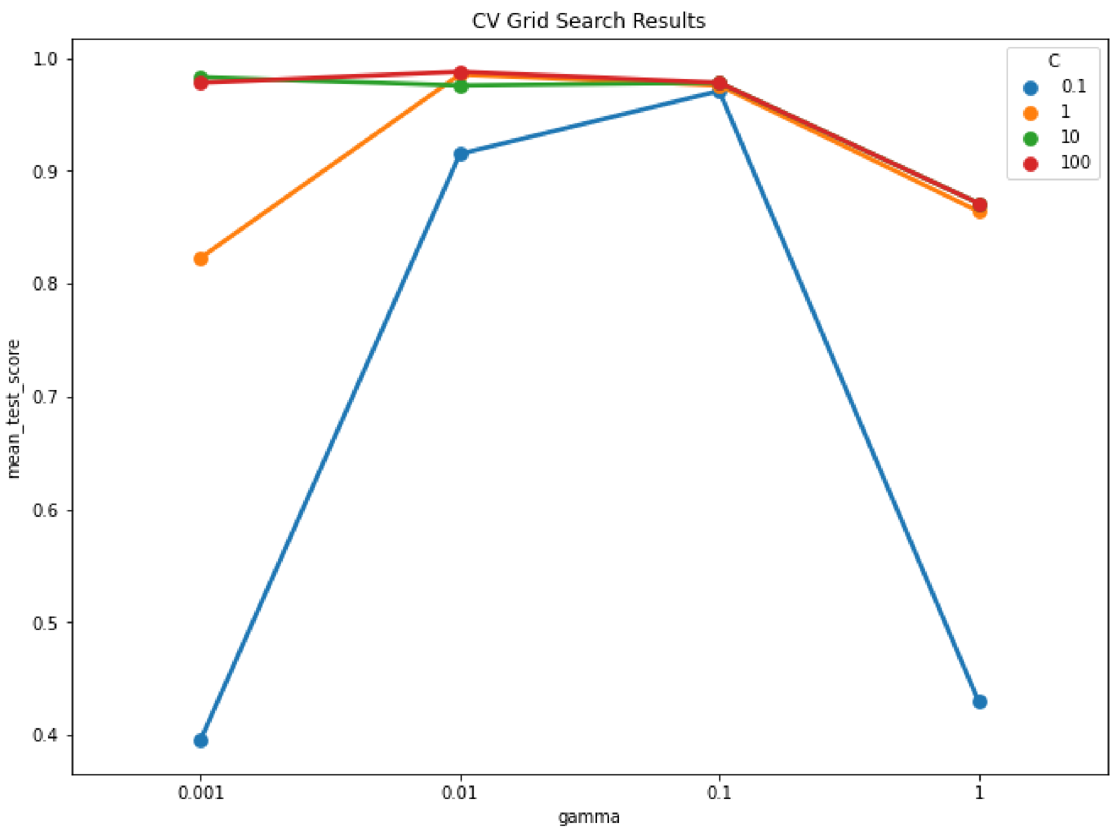
[60]:

```
#Prestazioni di tutte le combinazioni
x=pd.concat([pd.DataFrame(grid.cv_results_["params"]),pd.DataFrame(grid.
 ↪cv_results_["mean_test_score"], columns=["Accuracy"])],axis=1)
print(x)
#TEST Miglior Risultato
print(grid.best_estimator_)
grid_predictions = grid.predict(X_test)
print(confusion_matrix(y_test,grid_predictions))
print(classification_report(y_test,grid_predictions))
fig = plot_cv_results(grid.cv_results_, 'gamma','C')
print("Accuracy")
accuracy_score(y_test,grid_predictions)
```

```
         C  gamma  Accuracy
0      0.1  1.000  0.429621
1      0.1  0.100  0.970849
2      0.1  0.010  0.915016
3      0.1  0.001  0.395680
4      1.0  1.000  0.864061
5      1.0  0.100  0.975669
6      1.0  0.010  0.985454
7      1.0  0.001  0.822892
8     10.0  1.000  0.871290
9     10.0  0.100  0.978108
10    10.0  0.010  0.975698
11    10.0  0.001  0.983015
12   100.0  1.000  0.871290
13   100.0  0.100  0.978108
14   100.0  0.010  0.987893
15   100.0  0.001  0.978108
SVC(C=100, gamma=0.01)
[[28  0  0  0  0]
 [ 0 18  0  0  0]
 [ 0  0 21  0  0]
```

```
 [ 0  0  0 17  0]
 [ 0  0  0  0 20]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        28
           1       1.00      1.00      1.00        18
           2       1.00      1.00      1.00        21
           3       1.00      1.00      1.00        17
           4       1.00      1.00      1.00        20

    accuracy                           1.00       104
   macro avg       1.00      1.00      1.00       104
weighted avg       1.00      1.00      1.00       104

Accuracy
```

[60]: 1.0




[61]:
```
#Sigmoid Kernel Con Grid Search
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001]}
```

```
grid =␣
↪GridSearchCV(SVC(kernel='sigmoid'),param_grid,refit=True,verbose=0,return_train_score=True)
grid.fit(X_train,y_train)
#Parametri Migliori
print(grid.best_params_)
print(grid.best_score_)
```

```
{'C': 10, 'gamma': 0.01}
0.9878636497208346
```

[62]:
```
x=pd.concat([pd.DataFrame(grid.cv_results_["params"]),pd.DataFrame(grid.
↪cv_results_["mean_train_score"], columns=["Training"]),pd.DataFrame(grid.
↪cv_results_["mean_test_score"], columns=["Validation"])],axis=1)
print(x)
```

```
        C  gamma  Training  Validation
0     0.1  1.000  0.632243    0.645636
1     0.1  0.100  0.970270    0.968440
2     0.1  0.010  0.812503    0.800911
3     0.1  0.001  0.379849    0.373905
4     1.0  1.000  0.635319    0.628504
5     1.0  0.100  0.922935    0.922363
6     1.0  0.010  0.987864    0.983015
7     1.0  0.001  0.771841    0.762210
8    10.0  1.000  0.660182    0.645754
9    10.0  0.100  0.897443    0.881164
10   10.0  0.010  0.986652    0.987864
11   10.0  0.001  0.990292    0.985425
12  100.0  1.000  0.648043    0.645813
13  100.0  0.100  0.891377    0.876315
14  100.0  0.010  0.981803    0.970820
15  100.0  0.001  0.988472    0.982986
```

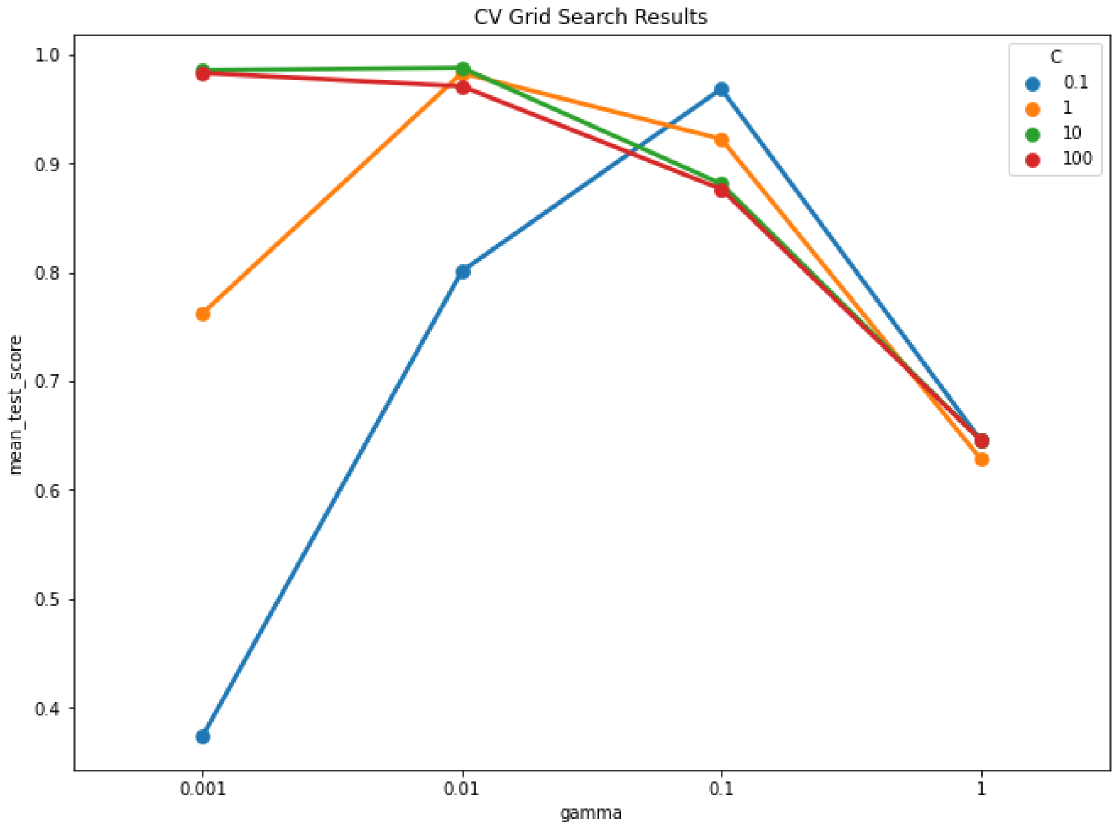
[63]:
```
#Prestazioni di tutte le combinazioni
x=pd.concat([pd.DataFrame(grid.cv_results_["params"]),pd.DataFrame(grid.
↪cv_results_["mean_test_score"], columns=["Accuracy"])],axis=1)
print(x)
#TEST Miglior Risultato
print(grid.best_estimator_)
grid_predictions = grid.predict(X_test)
print(confusion_matrix(y_test,grid_predictions))
print(classification_report(y_test,grid_predictions))
print("Accuracy")
accuracy_score(y_test,grid_predictions)
```

```
        C  gamma  Accuracy
0     0.1  1.000  0.645636
```

```
1      0.1  0.100  0.968440
2      0.1  0.010  0.800911
3      0.1  0.001  0.373905
4      1.0  1.000  0.628504
5      1.0  0.100  0.922363
6      1.0  0.010  0.983015
7      1.0  0.001  0.762210
8     10.0  1.000  0.645754
9     10.0  0.100  0.881164
10    10.0  0.010  0.987864
11    10.0  0.001  0.985425
12   100.0  1.000  0.645813
13   100.0  0.100  0.876315
14   100.0  0.010  0.970820
15   100.0  0.001  0.982986
SVC(C=10, gamma=0.01, kernel='sigmoid')
[[28  0  0  0  0]
 [ 0 18  0  0  0]
 [ 0  0 21  0  0]
 [ 0  0  0 17  0]
 [ 0  0  0  0 20]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        28
           1       1.00      1.00      1.00        18
           2       1.00      1.00      1.00        21
           3       1.00      1.00      1.00        17
           4       1.00      1.00      1.00        20

    accuracy                           1.00       104
   macro avg       1.00      1.00      1.00       104
weighted avg       1.00      1.00      1.00       104

Accuracy
```

[63]:
```
1.0
```

[64]:
```
fig = plot_cv_results(grid.cv_results_, 'gamma','C')
```




## 3 K-Nearest Neighbour

[65]:
```
#Con Grid Search
k_range = list(range(1, 31))
param_grid = dict(n_neighbors=k_range)
grid = GridSearchCV(KNeighborsClassifier(), param_grid,refit=True, cv=5,␣
 ↪scoring='accuracy',return_train_score=True)
grid.fit(X_train, y_train)
# Miglior Modello
print(grid.best_params_)
print(grid.best_score_)
```

```
{'n_neighbors': 3}
0.9830149867763737
```

[66]:
```
x=pd.concat([pd.DataFrame(grid.cv_results_["params"]),pd.DataFrame(grid.
 ↪cv_results_["mean_train_score"], columns=["Training"]),pd.DataFrame(grid.
 ↪cv_results_["mean_test_score"], columns=["Validation"])],axis=1)
print(x)
```

|  | n_neighbors | Training | Validation |
|---|---|---|---|
| 0 | 1 | 1.000000 | 0.980547 |
| 1 | 2 | 0.993326 | 0.975727 |
| 2 | 3 | 0.989686 | 0.983015 |
| 3 | 4 | 0.989074 | 0.978196 |
| 4 | 5 | 0.983010 | 0.973347 |
| 5 | 6 | 0.984832 | 0.975757 |
| 6 | 7 | 0.980582 | 0.973347 |
| 7 | 8 | 0.983616 | 0.980635 |
| 8 | 9 | 0.983012 | 0.975757 |
| 9 | 10 | 0.984830 | 0.980635 |
| 10 | 11 | 0.984224 | 0.978196 |
| 11 | 12 | 0.984830 | 0.978196 |
| 12 | 13 | 0.983008 | 0.975727 |
| 13 | 14 | 0.979974 | 0.980605 |
| 14 | 15 | 0.979370 | 0.973288 |
| 15 | 16 | 0.979978 | 0.973288 |
| 16 | 17 | 0.977548 | 0.973288 |
| 17 | 18 | 0.978154 | 0.970879 |
| 18 | 19 | 0.973906 | 0.968440 |
| 19 | 20 | 0.974510 | 0.968440 |
| 20 | 21 | 0.973300 | 0.970849 |
| 21 | 22 | 0.973296 | 0.968440 |
| 22 | 23 | 0.973296 | 0.970849 |
| 23 | 24 | 0.974510 | 0.970849 |
| 24 | 25 | 0.973901 | 0.970849 |
| 25 | 26 | 0.973298 | 0.965971 |
| 26 | 27 | 0.973298 | 0.968440 |
| 27 | 28 | 0.972690 | 0.970849 |
| 28 | 29 | 0.972689 | 0.970849 |
| 29 | 30 | 0.973903 | 0.970849 |

[67]:
```python
#Prestazione di ogni Combinazione
pd.concat([pd.DataFrame(grid.cv_results_["params"]),pd.DataFrame(grid.
 ↪cv_results_["mean_test_score"], columns=["Accuracy"])],axis=1)
```

[67]:

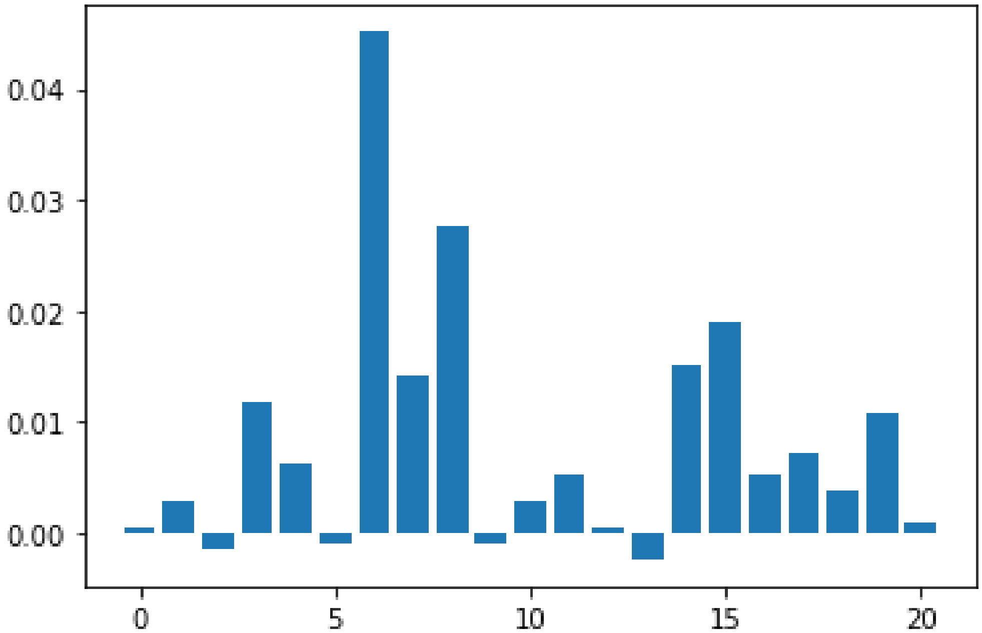
|  | n_neighbors | Accuracy |
|---|---|---|
| 0 | 1 | 0.980547 |
| 1 | 2 | 0.975727 |
| 2 | 3 | 0.983015 |
| 3 | 4 | 0.978196 |
| 4 | 5 | 0.973347 |
| 5 | 6 | 0.975757 |
| 6 | 7 | 0.973347 |
| 7 | 8 | 0.980635 |
| 8 | 9 | 0.975757 |
| 9 | 10 | 0.980635 |

```
10          11   0.978196
11          12   0.978196
12          13   0.975727
13          14   0.980605
14          15   0.973288
15          16   0.973288
16          17   0.973288
17          18   0.970879
18          19   0.968440
19          20   0.968440
20          21   0.970849
21          22   0.968440
22          23   0.970849
23          24   0.970849
24          25   0.970849
25          26   0.965971
26          27   0.968440
27          28   0.970849
28          29   0.970849
29          30   0.970849
```

[68]:
```python
#TEST Miglior Risultato

Bestknn=grid.best_estimator_
grid_predictions = Bestknn.predict(X_test)
print(confusion_matrix(y_test,grid_predictions))
print(classification_report(y_test,grid_predictions))
 # perform permutation importance
results = permutation_importance(Bestknn, X_train, y_train, scoring='accuracy')


# get importance
importance = results.importances_mean
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
print("Accuracy")
print(accuracy_score(y_test,grid_predictions))
```

```
[[28  0  0  0  0]
 [ 0 18  0  0  0]
 [ 0  0 21  0  0]
 [ 0  0  0 16  1]
 [ 0  0  0  1 19]]
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        28
           1       1.00      1.00      1.00        18
           2       1.00      1.00      1.00        21
           3       0.94      0.94      0.94        17
           4       0.95      0.95      0.95        20

    accuracy                           0.98       104
   macro avg       0.98      0.98      0.98       104
weighted avg       0.98      0.98      0.98       104

Feature: 0, Score: 0.00049
Feature: 1, Score: 0.00291
Feature: 2, Score: -0.00146
Feature: 3, Score: 0.01165
Feature: 4, Score: 0.00631
Feature: 5, Score: -0.00097
Feature: 6, Score: 0.04515
Feature: 7, Score: 0.01408
Feature: 8, Score: 0.02767
Feature: 9, Score: -0.00097
Feature: 10, Score: 0.00291
Feature: 11, Score: 0.00534
Feature: 12, Score: 0.00049
Feature: 13, Score: -0.00243
Feature: 14, Score: 0.01505
Feature: 15, Score: 0.01893
Feature: 16, Score: 0.00534
Feature: 17, Score: 0.00728
Feature: 18, Score: 0.00388
Feature: 19, Score: 0.01068
Feature: 20, Score: 0.00097
```

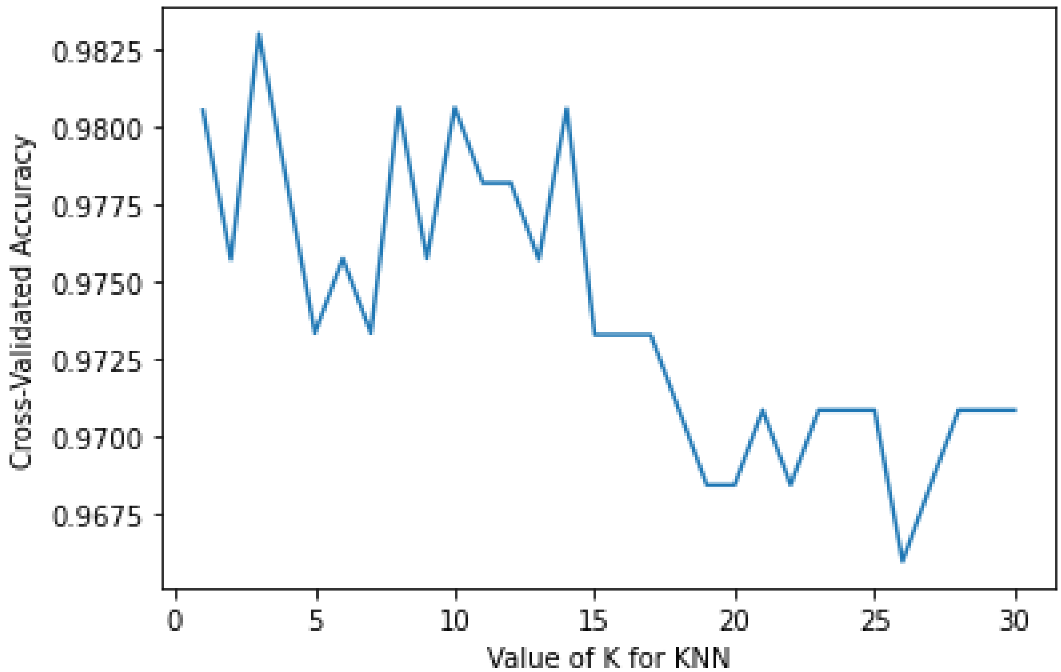



```
Accuracy
0.9807692307692307
```

[69]:
```
# plot the results
# this is identical to the one we generated above
grid_mean_scores = grid.cv_results_['mean_test_score']
print(grid_mean_scores)
plt.plot(k_range, grid_mean_scores)

plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
```

```
[0.98054658 0.9757273  0.98301499 0.97819571 0.97334705 0.97575669
 0.97334705 0.98063473 0.97575669 0.98063473 0.97819571 0.97819571
 0.9757273  0.98060535 0.97328828 0.97328828 0.97328828 0.97087864
 0.96843961 0.96843961 0.97084925 0.96843961 0.97084925 0.97084925
 0.97084925 0.9659712  0.96843961 0.97084925 0.97084925 0.97084925]
```

[69]:
```
Text(0, 0.5, 'Cross-Validated Accuracy')
```




# 4 Logistic Regression

[74]:
```
#Grid Search
C=np.logspace(-3,3,7)
penalty=["none","l2"]
grid={"C":np.logspace(-3,3,7), "penalty":["none","l2"]}
logreg=LogisticRegression()
logreg_cv=GridSearchCV(logreg,grid,refit=True,cv=5,return_train_score=True)
logreg_cv.fit(X_train,y_train)
print(logreg_cv.best_params_)
print(logreg_cv.best_score_)
```

```
{'C': 0.1, 'penalty': 'l2'}
0.9829856009403468
```

[77]:
```
x=pd.concat([pd.DataFrame(logreg_cv.cv_results_["params"]),pd.
 ↪DataFrame(logreg_cv.cv_results_["mean_train_score"], columns=["Training"]),pd.
 ↪DataFrame(logreg_cv.cv_results_["mean_test_score"],␣
 ↪columns=["Validation"])],axis=1)
print(x)
```

```
          C penalty  Training  Validation
0     0.001    none  1.000000    0.970820
1     0.001      l2  0.936288    0.932001
```

```
2       0.010    none  1.000000    0.970820
3       0.010      l2  0.983618    0.980605
4       0.100    none  1.000000    0.970820
5       0.100      l2  0.988470    0.982986
6       1.000    none  1.000000    0.970820
7       1.000      l2  0.993326    0.975698
8      10.000    none  1.000000    0.970820
9      10.000      l2  0.996360    0.975698
10    100.000    none  1.000000    0.970820
11    100.000      l2  1.000000    0.970849
12   1000.000    none  1.000000    0.970820
13   1000.000      l2  1.000000    0.965971
```

[78]:

```
#Prestazione di ogni Combinazione
pd.concat([pd.DataFrame(logreg_cv.cv_results_["params"]),pd.DataFrame(logreg_cv.
 ↪cv_results_["mean_test_score"], columns=["Accuracy"])],axis=1)
```

[78]:

```
            C penalty  Accuracy
0       0.001    none  0.970820
1       0.001      l2  0.932001
2       0.010    none  0.970820
3       0.010      l2  0.980605
4       0.100    none  0.970820
5       0.100      l2  0.982986
6       1.000    none  0.970820
7       1.000      l2  0.975698
8      10.000    none  0.970820
9      10.000      l2  0.975698
10    100.000    none  0.970820
11    100.000      l2  0.970849
12   1000.000    none  0.970820
13   1000.000      l2  0.965971
```

[79]:

```
#TEST Miglior Risultato
BestLog=logreg_cv.best_estimator_
grid_predictions = BestLog.predict(X_test)
print(confusion_matrix(y_test,grid_predictions))
print(classification_report(y_test,grid_predictions))
print("Accuracy")
accuracy_score(y_test,grid_predictions)
```

```
[[28  0  0  0  0]
 [ 0 18  0  0  0]
 [ 0  0 21  0  0]
 [ 0  0  0 17  0]
 [ 0  0  0  1 19]]
              precision    recall  f1-score   support
```

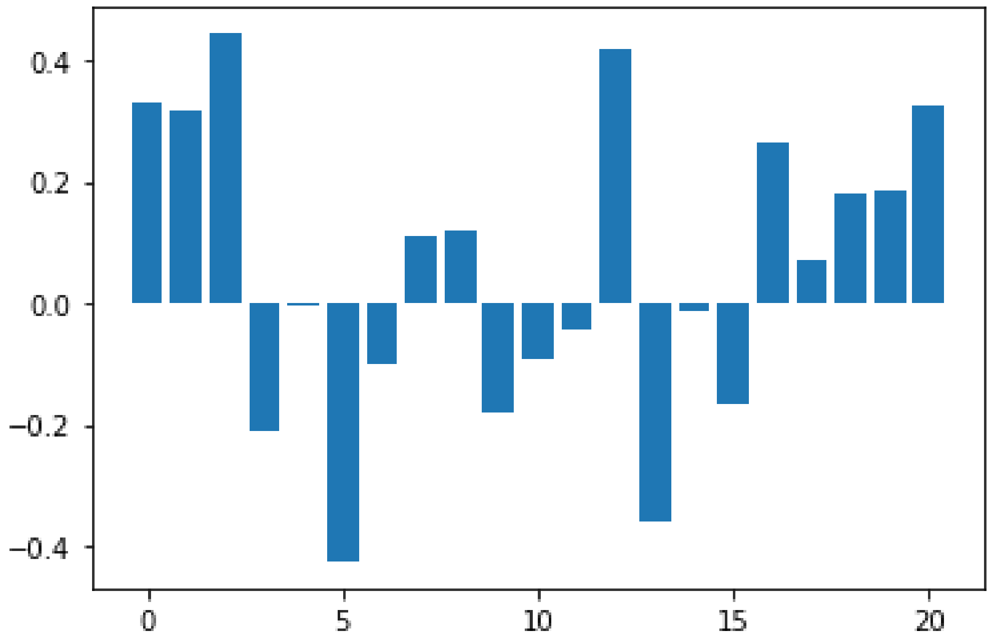
```
           0       1.00      1.00      1.00        28
           1       1.00      1.00      1.00        18
           2       1.00      1.00      1.00        21
           3       0.94      1.00      0.97        17
           4       1.00      0.95      0.97        20

    accuracy                           0.99       104
   macro avg       0.99      0.99      0.99       104
weighted avg       0.99      0.99      0.99       104
```

```
Accuracy
```

[79]: 0.9903846153846154

[80]:
```
#Feature Importance
importance = BestLog.coef_[0]
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

```
Feature: 0, Score: 0.32786
Feature: 1, Score: 0.31720
Feature: 2, Score: 0.44280
Feature: 3, Score: -0.21083
Feature: 4, Score: -0.00398
Feature: 5, Score: -0.42530
Feature: 6, Score: -0.09909
Feature: 7, Score: 0.10973
Feature: 8, Score: 0.11909
Feature: 9, Score: -0.17826
Feature: 10, Score: -0.09077
Feature: 11, Score: -0.04398
Feature: 12, Score: 0.41809
Feature: 13, Score: -0.35945
Feature: 14, Score: -0.01445
Feature: 15, Score: -0.16436
Feature: 16, Score: 0.26267
Feature: 17, Score: 0.06973
Feature: 18, Score: 0.18252
Feature: 19, Score: 0.18680
Feature: 20, Score: 0.32559
```




[ ]:

```

```

## 5 Decision tree

[ ]:

```
dtree = tree.DecisionTreeClassifier()
```

[ ]:

```
dtree.fit(X_train,y_train)
plt.figure(figsize=(15,15))
tree.plot_tree(dtree,filled=True,fontsize=10)
plt.savefig('tree.jpg',format='jpg',bbox_inches = "tight")
pred_clf=dtree.predict(X_test)
print(classification_report(y_test, pred_clf))
print(confusion_matrix(y_test, pred_clf))
Label=data['Class'].unique()
cmtx = pd.DataFrame(
    confusion_matrix(y_test, pred_clf, labels=Label),
    index=['true:{:}'.format(x) for x in Label],
    columns=['pred:{:}'.format(x) for x in Label]
)
print(cmtx)
```

[ ]:

```
'''
# save the model to disk
filename = 'finalized_model.sav'
pickle.dump(dtree, open(filename, 'wb'))
```

```
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(X_test, y_test)
print(result)
pred_clf=loaded_model.predict(X_test)
print(classification_report(y_test, pred_clf))
print(confusion_matrix(y_test, pred_clf))
'''
```