ML-prac-1

September 10, 2023

[2]:
```python
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/breast-cancer-wisconsin-data/data.csv
/kaggle/input/pngpng/breast-cancer-awareness-campaign-4375584-3649338.png
```

1 ML Feature Selection Techniques | Breast Cancer Diagnosis

1.1 In this Project we will cover following sections

- **Import Dataset using pandas.read_csv**
- **Visualization of dataset in different ways and check for null values, undesired columns and remove them**
- **Declare the dependent (y) and independent (X) parameters from the dataset**
- **Convert the dependent (y) parameter into 0,1 using a label encoder to classify it not two classes**

- **Use a Standard scaler to normally distribute the data**
- **Train the model and check the accuracy without reducing the learning parameters**
- **Apply Different correlation Techniques to reduce the learning parameter**

- **Filter Method:**
  i) Pearson Correlation Coefficient
  ii) Spearman's Rank Correlation Coefficient
  iii) Kendall's Rank Correlation Coefficient
- **Statistical and Ranking Filter Method:**
  i) Mutual Information or Information Gain
  ii) ANNOVA Univariate Test
- **Wrapper Method:**
  i) Step Forward Feature Selection
  ii) Step Backward Feature Selection
  iii) Step Floating Forward Feature Selection
  iv) Step Floating Backward Feature Selection
- **Train the model and check the accuracy after reducing the learning parameters**
- **Do the above step for all the feature selection technique**
- **Compare the correlation technique using a Bar graph and ROC curve**

1.2 Importing Libraries

[3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split as tts
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪mutual_info_score, classification_report, ConfusionMatrixDisplay
from sklearn.feature_selection import mutual_info_classif as MIC
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
```

[4]:
```python
#importing the dataset and displaying above 5 rows
df = pd.read_csv('/kaggle/input/breast-cancer-wisconsin-data/data.csv')
df.head()
```

[4]:
```
         id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302         M        17.99         10.38          122.80     1001.0
1    842517         M        20.57         17.77          132.90     1326.0
2  84300903         M        19.69         21.25          130.00     1203.0
3  84348301         M        11.42         20.38           77.58      386.1
4  84358402         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   …  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0  …          17.33           184.60      2019.0            0.1622
1  …          23.41           158.80      1956.0            0.1238
2  …          25.53           152.50      1709.0            0.1444
3  …          26.50            98.87       567.7            0.2098
4  …          16.67           152.20      1575.0            0.1374

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0             0.6656           0.7119                0.2654          0.4601
1             0.1866           0.2416                0.1860          0.2750
2             0.4245           0.4504                0.2430          0.3613
3             0.8663           0.6869                0.2575          0.6638
4             0.2050           0.4000                0.1625          0.2364

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
1                  0.08902          NaN
2                  0.08758          NaN
3                  0.17300          NaN
4                  0.07678          NaN

[5 rows x 33 columns]
```

1.3 Visualization in different ways

[5]:
```python
df.shape #shape of out dataset
```

[5]:
```
(569, 33)
```

[6]:
```
df.isnull().sum()#checking for nulls
```

[6]:
```
id                          0
diagnosis                   0
radius_mean                 0
texture_mean                0
perimeter_mean              0
area_mean                   0
smoothness_mean             0
compactness_mean            0
concavity_mean              0
concave points_mean         0
symmetry_mean               0
fractal_dimension_mean      0
radius_se                   0
texture_se                  0
perimeter_se                0
area_se                     0
smoothness_se               0
compactness_se              0
concavity_se                0
concave points_se           0
symmetry_se                 0
fractal_dimension_se        0
radius_worst                0
texture_worst               0
perimeter_worst             0
area_worst                  0
smoothness_worst            0
compactness_worst           0
concavity_worst             0
concave points_worst        0
symmetry_worst              0
fractal_dimension_worst     0
Unnamed: 32               569
dtype: int64
```

[7]:
```
df.describe() #displaying statistical data of our dataset
```

[7]:
```
                 id  radius_mean  texture_mean  perimeter_mean   area_mean  \
count  5.690000e+02   569.000000    569.000000      569.000000  569.000000
mean   3.037183e+07    14.127292     19.289649       91.969033  654.889104
std    1.250206e+08     3.524049      4.301036       24.298981  351.914129
min    8.670000e+03     6.981000      9.710000       43.790000  143.500000
25%    8.692180e+05    11.700000     16.170000       75.170000  420.300000
50%    9.060240e+05    13.370000     18.840000       86.240000  551.100000
75%    8.813129e+06    15.780000     21.800000      104.100000  782.700000
```

```
max    9.113205e+08    28.110000     39.280000      188.500000  2501.000000

       smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
count       569.000000        569.000000      569.000000           569.000000
mean          0.096360          0.104341        0.088799             0.048919
std           0.014064          0.052813        0.079720             0.038803
min           0.052630          0.019380        0.000000             0.000000
25%           0.086370          0.064920        0.029560             0.020310
50%           0.095870          0.092630        0.061540             0.033500
75%           0.105300          0.130400        0.130700             0.074000
max           0.163400          0.345400        0.426800             0.201200

       symmetry_mean  …  texture_worst  perimeter_worst   area_worst  \
count     569.000000  …     569.000000       569.000000   569.000000
mean        0.181162  …      25.677223       107.261213   880.583128
std         0.027414  …       6.146258        33.602542   569.356993
min         0.106000  …      12.020000        50.410000   185.200000
25%         0.161900  …      21.080000        84.110000   515.300000
50%         0.179200  …      25.410000        97.660000   686.500000
75%         0.195700  …      29.720000       125.400000  1084.000000
max         0.304000  …      49.540000       251.200000  4254.000000

       smoothness_worst  compactness_worst  concavity_worst  \
count        569.000000         569.000000       569.000000
mean           0.132369           0.254265         0.272188
std            0.022832           0.157336         0.208624
min            0.071170           0.027290         0.000000
25%            0.116600           0.147200         0.114500
50%            0.131300           0.211900         0.226700
75%            0.146000           0.339100         0.382900
max            0.222600           1.058000         1.252000

       concave points_worst  symmetry_worst  fractal_dimension_worst  \
count            569.000000      569.000000               569.000000
mean               0.114606        0.290076                 0.083946
std                0.065732        0.061867                 0.018061
min                0.000000        0.156500                 0.055040
25%                0.064930        0.250400                 0.071460
50%                0.099930        0.282200                 0.080040
75%                0.161400        0.317900                 0.092080
max                0.291000        0.663800                 0.207500

       Unnamed: 32
count          0.0
mean           NaN
std            NaN
min            NaN
```

```
25%             NaN
50%             NaN
75%             NaN
max             NaN

[8 rows x 32 columns]
```

[8]:
```
#dropping the id and Unnamed: 32 columns as it will not contirbute to the␣
↪training of model
df.drop('id',axis=1,inplace=True)
df.drop('Unnamed: 32',axis=1,inplace=True)
```

[9]:
```
#Value counts of the target column diagnosis
df['diagnosis'].value_counts()
```

[9]:
```
diagnosis
B    357
M    212
Name: count, dtype: int64
```

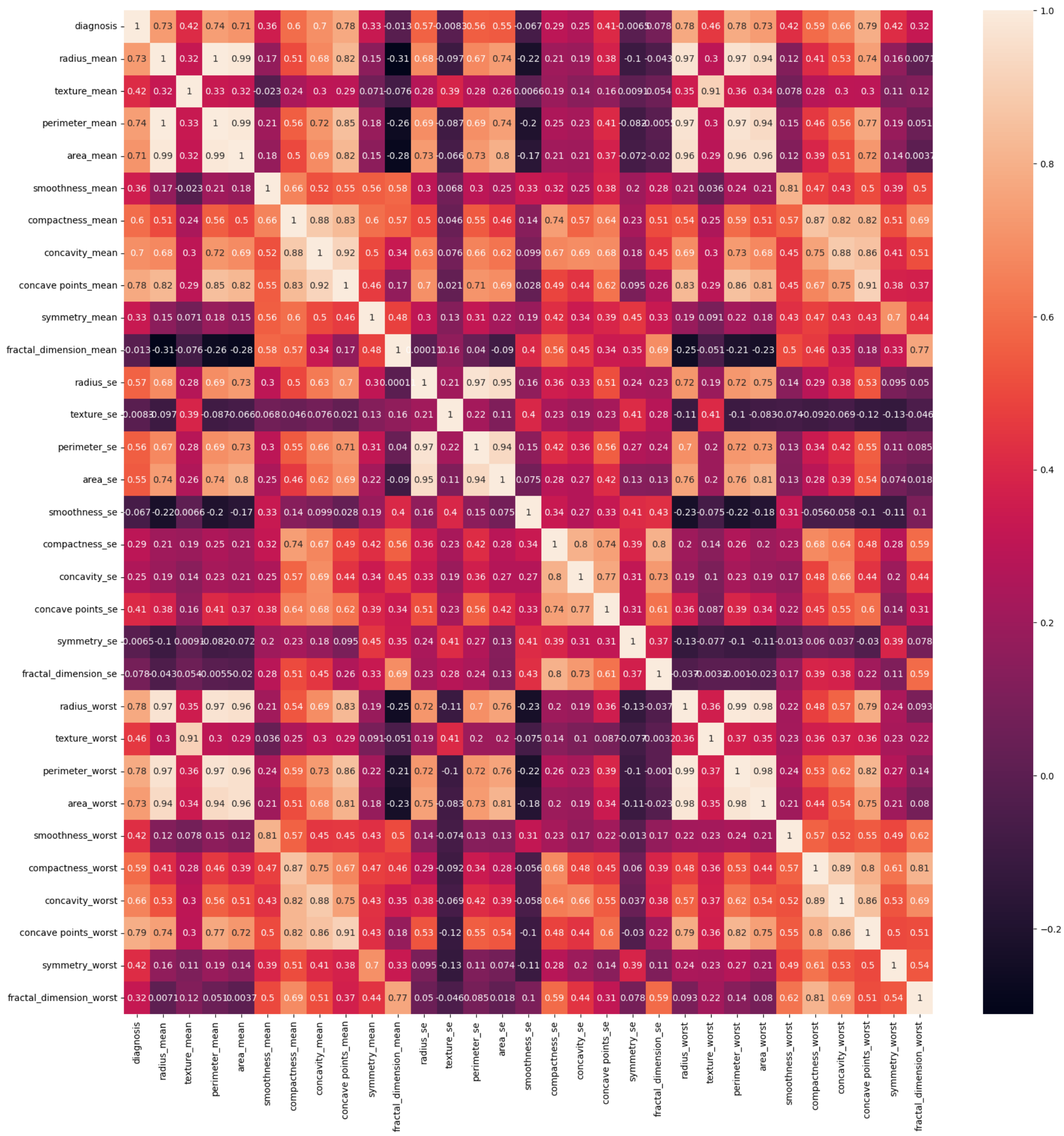
1.4 Convert the dependent (y) parameter into 0,1

[10]:
```
label_encoder = preprocessing.LabelEncoder()
df['diagnosis'] = label_encoder.fit_transform(df['diagnosis'])
df['diagnosis'].unique()
```

[10]:
```
array([1, 0])
```

[11]:
```
#plotting heatmap to see the correlation of the features of the dataset
plt.figure(figsize=(20,20))
sns.heatmap(df.corr(),annot=True)
```

[11]:
```
<Axes: >
```



1.5 Declare the dependent (y) and independent (X) parameters from the dataset

[12]:
```
#splitting the dataset into X (independent) and y (dependent)
data = np.array(df)
Y, X = np.split(data,[1],axis=1)
X_train,X_test,y_train,y_test = tts(X,Y,test_size=0.3,random_state=40)
```

1.6 Using Standard scaler to normally distribute the data

[13]:
```
ss=StandardScaler()
X_train=ss.fit_transform(X_train)
X_test=ss.transform(X_test)
```

1.7 Train and check accuracy

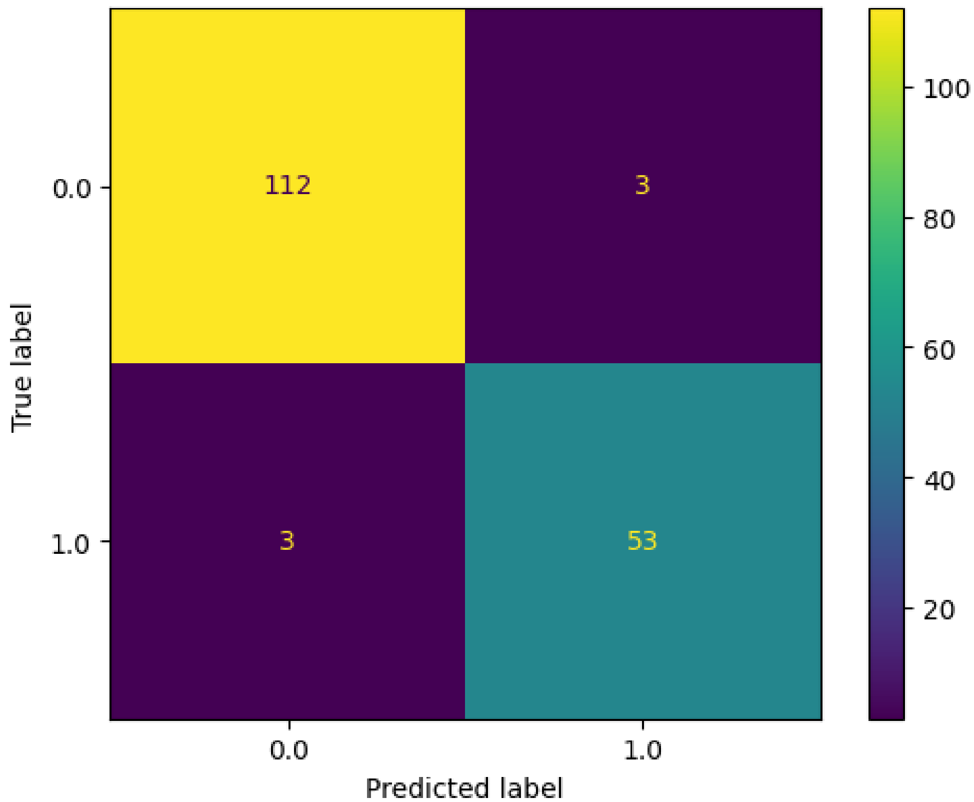
[14]:
```
#running our GaussianNB algorithm to check the accuracy (without feature␣
↪selection)
model = GaussianNB()
model.fit(X_train,np.ravel(y_train))
predict = model.predict(X_test)
accuracy = accuracy_score(y_test,predict)
print('Accuracy without feature selection:',accuracy)
classif_report = pd.DataFrame(classification_report(y_test, predict,␣
↪output_dict=True))
classif_report
```

```
Accuracy without feature selection: 0.9649122807017544
```

[14]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.973913	0.946429	0.964912	0.960171	0.964912
recall	0.973913	0.946429	0.964912	0.960171	0.964912
f1-score	0.973913	0.946429	0.964912	0.960171	0.964912
support	115.000000	56.000000	0.964912	171.000000	171.000000

[15]:
```
cm = confusion_matrix(y_test, predict, labels=model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.
↪classes_)
disp.plot()
plt.grid(False)
plt.show()
```



1.8 Applying Feature Selection Techniques and Training

FEATURE SELECTION TECHNIQUES USED: 1. Filter Method: a) Pearson Correlation Coefficient b) Spearman's Rank Correlation Coefficient c) Kendall's Rank Correlation Cofficient 2. Statistical and Ranking Filter Methods: a) Mutual Information or Information Gain b) ANNOVA Univariate Test 3. Wrapper Method: a) Step Forward Feature Selection
b) Step Backward Feature Selection c) Step Floating Forward Feature Selection d) Step Floating Backward Feature Selection

1.8.1 Filter Method

`a) Pearson Correlation Coefficient`

[16]:
```
pcc_X_train = pd.DataFrame(X_train)
pcc_X_test = pd.DataFrame(X_test)
pcc_matrix = pcc_X_train.corr(method = "pearson")
pcc_features = set()
for i in range(len(pcc_matrix)):
    for j in range(i):
        if(abs(pcc_matrix.iloc[i,j]>0.9)):
            column = pcc_matrix.columns[i]
            pcc_features.add(column)
```

```python
print('Features selected for Pearson Correlation Coefficient method:
 ↪',pcc_features)
pcc_X_train.drop(labels = pcc_features, axis = 1, inplace=True)
pcc_X_test.drop(labels = pcc_features, axis = 1, inplace=True)
```

```
Features selected for Pearson Correlation Coefficient method: {2, 3, 7, 12, 13,
20, 21, 22, 23, 27}
```

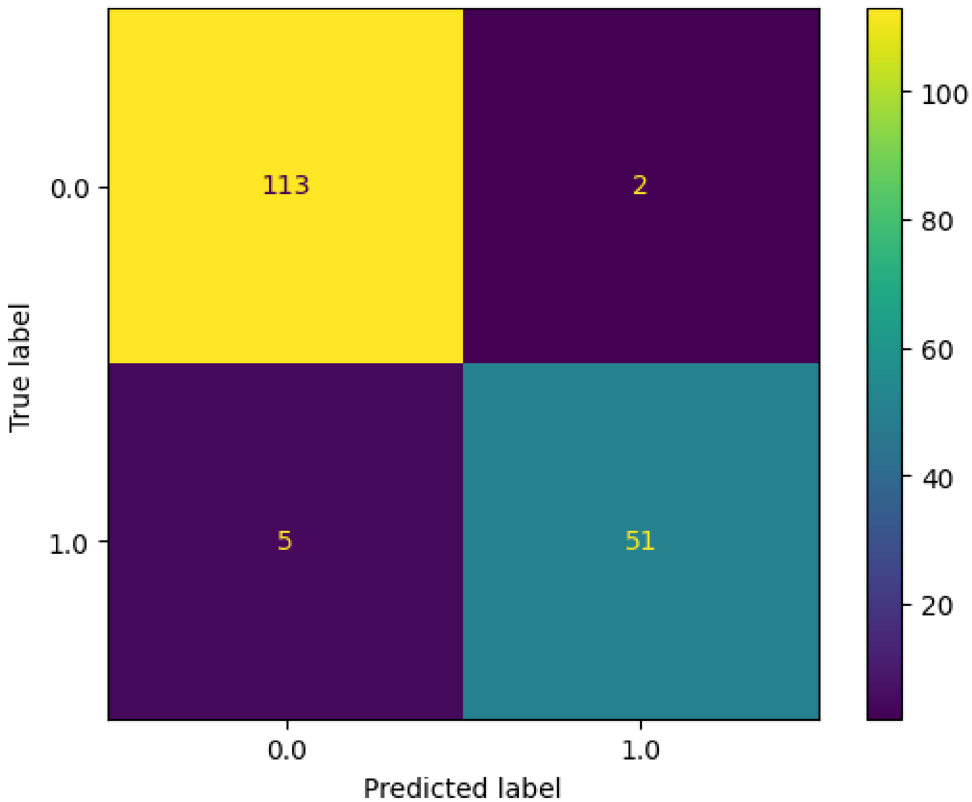
[17]:
```python
pcc_model = GaussianNB()
pcc_model.fit(pcc_X_train,np.ravel(y_train))
pcc_predict = pcc_model.predict(pcc_X_test)
pcc_accuracy = accuracy_score(y_test,pcc_predict)
print('Accuracy with Pearson Correlation Coefficient:',pcc_accuracy)
pcc_report = pd.DataFrame(classification_report(y_test, pcc_predict,␣
 ↪output_dict=True))
pcc_report
```

```
Accuracy with Pearson Correlation Coefficient: 0.9590643274853801
```

[17]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.957627	0.962264	0.959064	0.959946	0.959146
recall	0.982609	0.910714	0.959064	0.946661	0.959064
f1-score	0.969957	0.935780	0.959064	0.952868	0.958765
support	115.000000	56.000000	0.959064	171.000000	171.000000

[18]:
```python
cm = confusion_matrix(y_test, pcc_predict, labels=pcc_model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=pcc_model.
 ↪classes_)
disp.plot()
plt.grid(False)
plt.show()
```



1.8.2 Filter Method

b) Spearman's Rank Correlation Coefficient

[19]:
```
src_X_train = pd.DataFrame(X_train)
src_X_test = pd.DataFrame(X_test)
src_matrix = src_X_train.corr(method = "spearman")
src_features = set()
for i in range(len(src_matrix)):
    for j in range(i):
        if(abs(src_matrix.iloc[i,j]>0.8)):
            column = src_matrix.columns[i]
            src_features.add(column)
print('Features selected for Spearmans Rank Correlation method',src_features)
src_X_train.drop(labels = src_features, axis = 1, inplace=True)
src_X_test.drop(labels = src_features, axis = 1, inplace=True)
```

```
Features selected for Spearmans Rank Correlation method {2, 3, 6, 7, 12, 13, 15,
16, 17, 20, 21, 22, 23, 25, 26, 27}
```

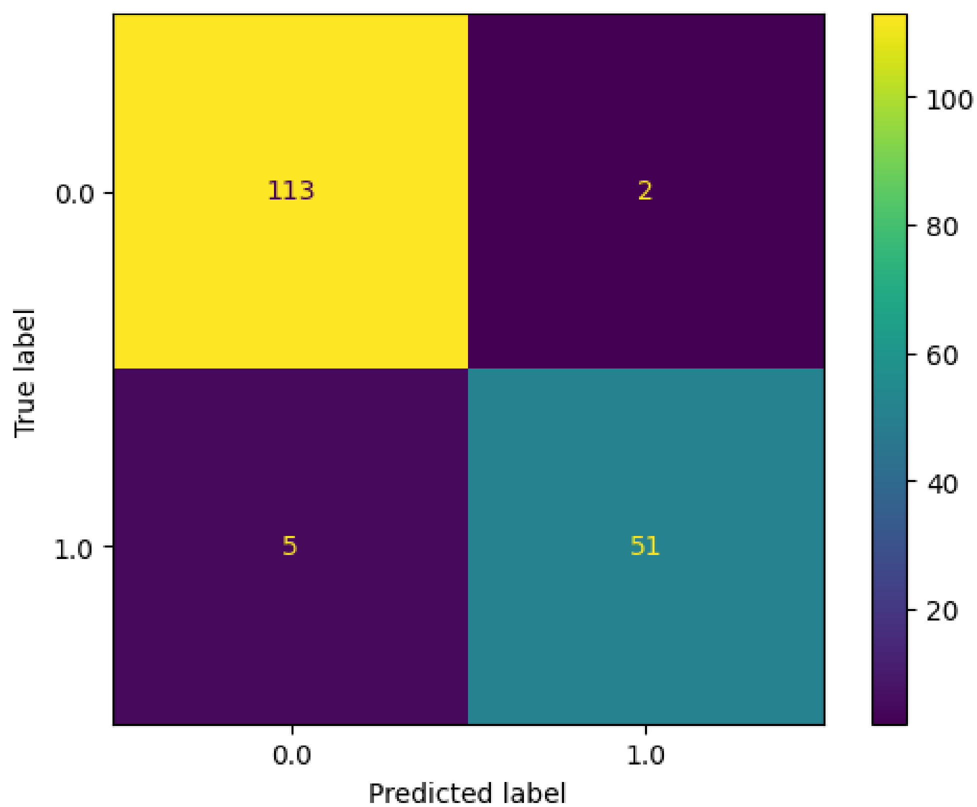
[20]:
```
src_model = GaussianNB()
src_model.fit(src_X_train,np.ravel(y_train))
src_predict = src_model.predict(src_X_test)
src_accuracy = accuracy_score(y_test,src_predict)
print('Accuracy with Spearmans Rank Correlation Coefficient:',src_accuracy)
src_report = pd.DataFrame(classification_report(y_test, src_predict,␣
↪output_dict=True))
src_report
```

```
Accuracy with Spearmans Rank Correlation Coefficient: 0.9590643274853801
```

[20]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.957627	0.962264	0.959064	0.959946	0.959146
recall	0.982609	0.910714	0.959064	0.946661	0.959064
f1-score	0.969957	0.935780	0.959064	0.952868	0.958765
support	115.000000	56.000000	0.959064	171.000000	171.000000

[21]:
```
cm = confusion_matrix(y_test, src_predict, labels=src_model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=src_model.
↪classes_)
disp.plot()
plt.grid(False)
plt.show()
```



1.8.3 Filter Method

c) Kendall's Rank Correlation Coefficient

[22]:
```
krc_X_train = pd.DataFrame(X_train)
krc_X_test = pd.DataFrame(X_test)
krc_matrix = krc_X_train.corr(method = "kendall")
krc_features = set()
for i in range(len(krc_matrix)):
    for j in range(i):
        if(abs(krc_matrix.iloc[i,j]>0.8)):
            column = krc_matrix.columns[i]
            krc_features.add(column)
print('Features selected for Kendalls Rank Correlation method',krc_features)
krc_X_train.drop(labels = krc_features, axis = 1, inplace=True)
krc_X_test.drop(labels = krc_features, axis = 1, inplace=True)
```

```
Features selected for Kendalls Rank Correlation method {2, 3, 12, 13, 20, 22,
23}
```

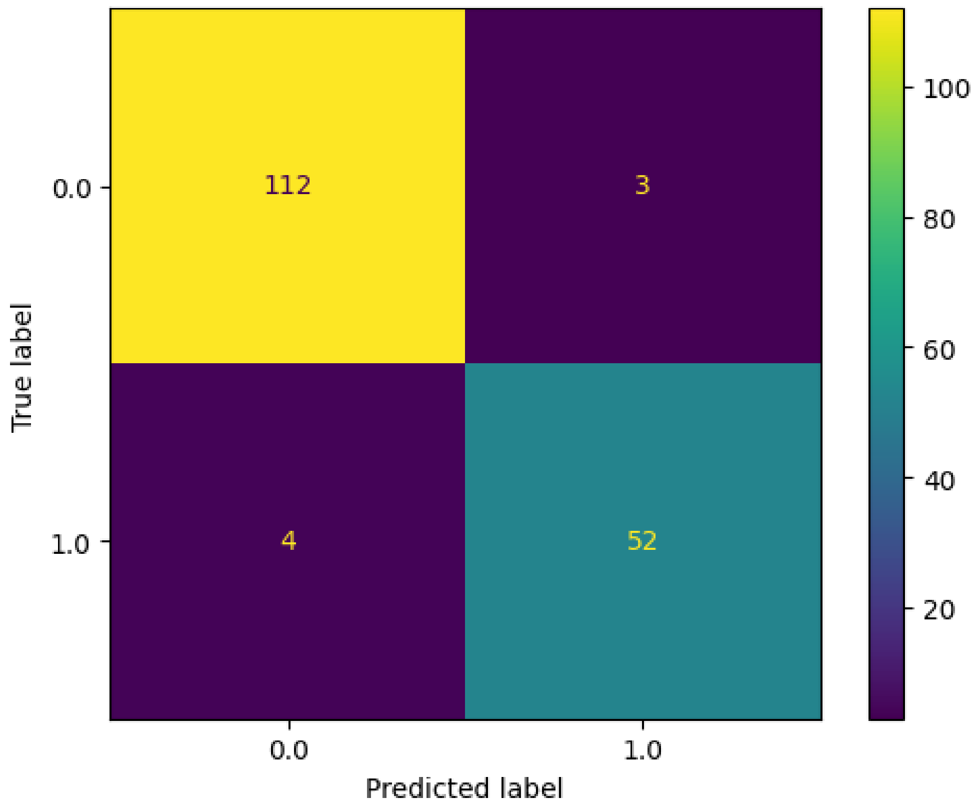
[23]:
```
krc_model = GaussianNB()
krc_model.fit(krc_X_train,np.ravel(y_train))
krc_predict = krc_model.predict(krc_X_test)
krc_accuracy = accuracy_score(y_test,krc_predict)
print('Accuracy with Kendalls Rank Correlation Coefficient:',krc_accuracy)
krc_report = pd.DataFrame(classification_report(y_test, krc_predict,␣
 ↪output_dict=True))
krc_report
```

```
Accuracy with Kendalls Rank Correlation Coefficient: 0.9590643274853801
```

[23]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.965517	0.945455	0.959064	0.955486	0.958947
recall	0.973913	0.928571	0.959064	0.951242	0.959064
f1-score	0.969697	0.936937	0.959064	0.953317	0.958969
support	115.000000	56.000000	0.959064	171.000000	171.000000

[24]:
```
cm = confusion_matrix(y_test, krc_predict, labels=krc_model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=krc_model.
 ↪classes_)
disp.plot()
plt.grid(False)
plt.show()
```



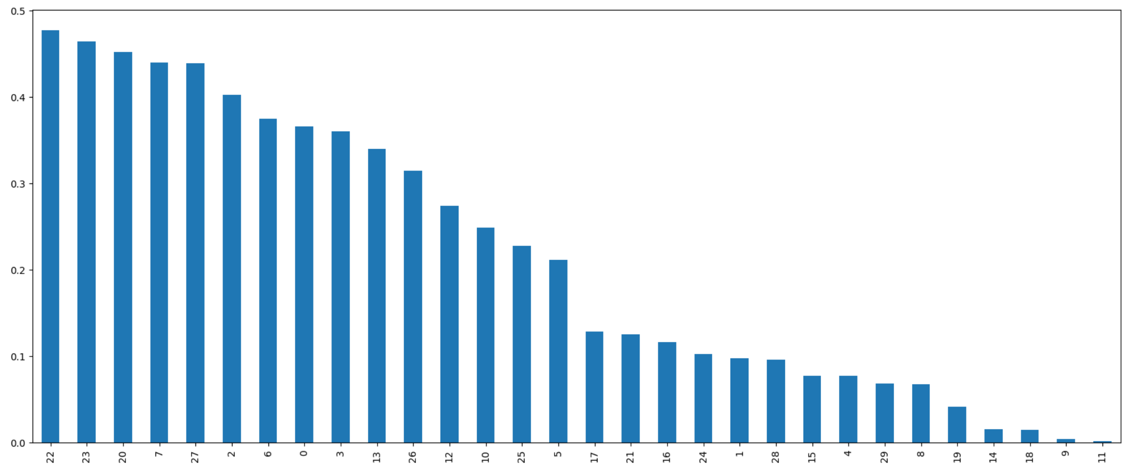
1.8.4 Statistical and Ranking Filter Methods

a) Mutual Information or Information Gain

[25]:
```
mic_scores = MIC(X,np.ravel(Y))
print('Estimated mutual information between each feature and the␣
 ↪target\n',mic_scores)
mutual_info = pd.Series(mic_scores)
mutual_info.sort_values(ascending=False)
mutual_info.sort_values(ascending=False).plot.bar(figsize=(20, 8)) #bar graph␣
 ↪to check Estimated mutual information between each feature and the target.
```

```
Estimated mutual information between each feature and the target
 [0.3654233  0.09727465 0.40212302 0.35941978 0.07658193 0.21109952
 0.37461926 0.43933328 0.06740813 0.00401972 0.2481417  0.00092304
 0.27361804 0.3393041  0.01530332 0.0766155  0.11560666 0.12769907
 0.01454752 0.04111067 0.45184226 0.12517228 0.47701961 0.46372746
 0.1019735  0.2268808  0.31467776 0.43852798 0.09537434 0.06827815]
```

[25]: `<Axes: >`



[26]:
```
mic_score_selected = np.where(mic_scores > 0.2)
print('Selected features where mutual information > 0.2',mic_score_selected)
mic_X = np.delete(X,[1,4,8,9,11,14,15,16,17,18,19,21,24,28,29],axis = 1)
mic_X_train, mic_X_test, mic_y_train, mic_y_test = tts(mic_X,Y, random_state =
  ↪0, stratify = Y)
```

```
Selected features where mutual information > 0.2 (array([ 0,  2,  3,  5,  6,  7,
10, 12, 13, 20, 22, 23, 25, 26, 27]),)
```

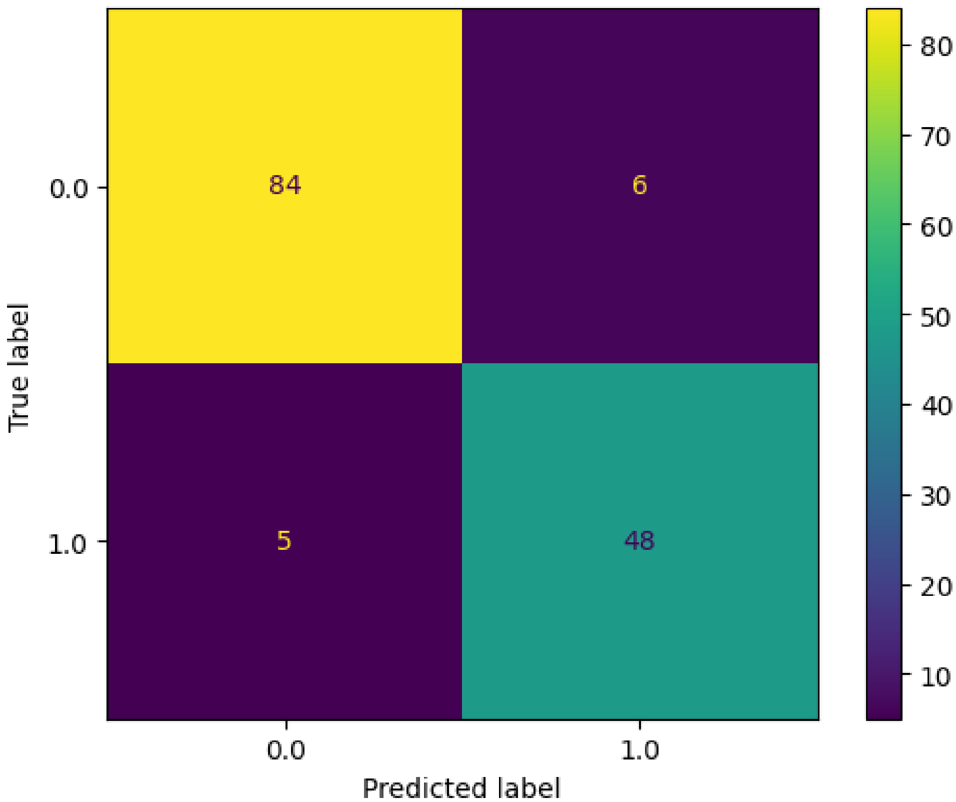
[27]:
```
mic_model = GaussianNB()
mic_model.fit(mic_X_train,np.ravel(mic_y_train))
mic_predict = mic_model.predict(mic_X_test)
mic_accuracy = accuracy_score(mic_y_test,mic_predict)
print('Accuracy with Mutual Information:',mic_accuracy)
mic_report = pd.DataFrame(classification_report(mic_y_test, mic_predict,
  ↪output_dict=True))
mic_report
```

```
Accuracy with Mutual Information: 0.9230769230769231
```

[27]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.943820	0.888889	0.923077	0.916355	0.923461
recall	0.933333	0.905660	0.923077	0.919497	0.923077
f1-score	0.938547	0.897196	0.923077	0.917872	0.923222
support	90.000000	53.000000	0.923077	143.000000	143.000000

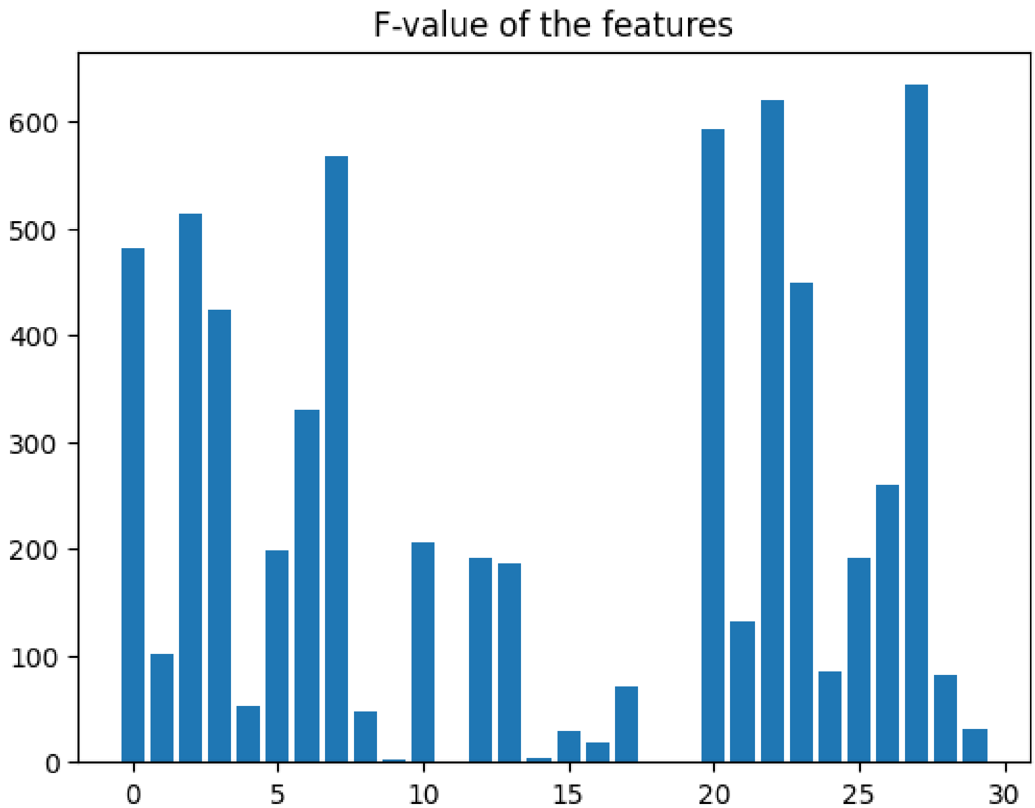
[28]:
```
cm = confusion_matrix(mic_y_test, mic_predict, labels=mic_model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=mic_model.
  ↪classes_)
disp.plot()
plt.grid(False)
plt.show()
```



1.8.5 Statistical and Ranking Filter Methods

b) ANNOVA Univariate Test

[29]:
```
fs = SelectKBest(score_func=f_classif, k='all')
fs.fit(X_train, np.ravel(y_train))
X_train_fs = fs.transform(X_train)
X_test_fs = fs.transform(X_test)
plt.bar([i for i in range(len(fs.scores_))], fs.scores_)
plt.title('F-value of the features')
plt.show() #bar graph to analyze f-statitic value for each feature
```



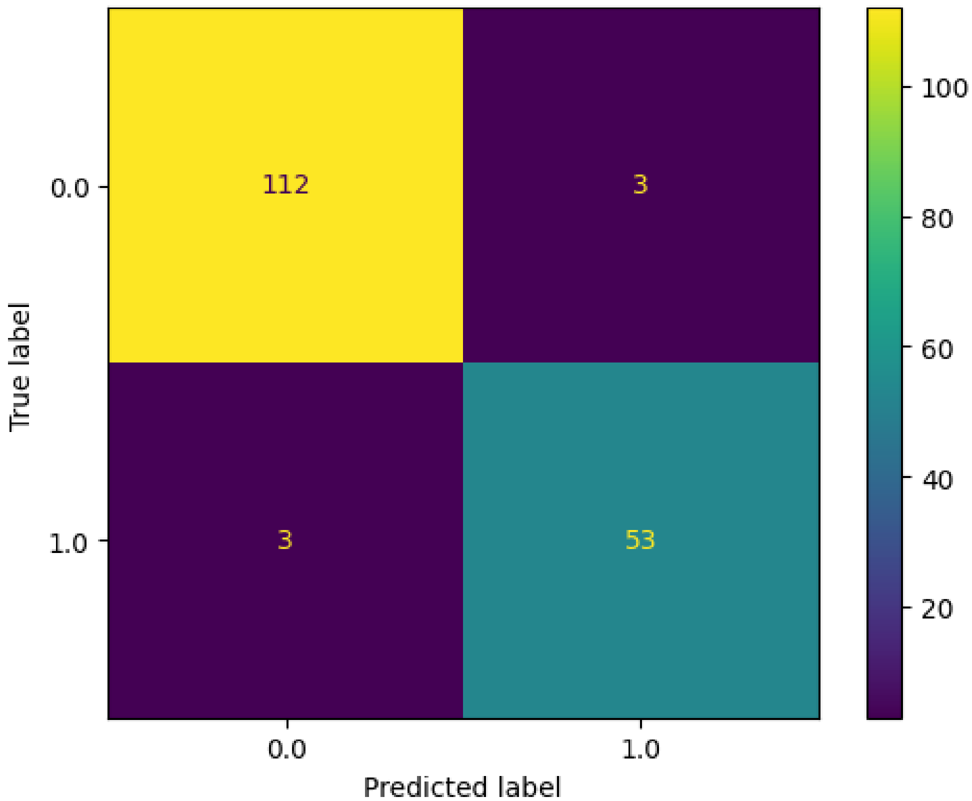
[30]:
```
fs = SelectKBest(score_func=f_classif, k=18) #selecting 18 best features
fs.fit(X_train, np.ravel(y_train))
X_train_fs = fs.transform(X_train)
X_test_fs = fs.transform(X_test)
annova_model = GaussianNB()
annova_model.fit(X_train_fs,np.ravel(y_train))
annova_predict = annova_model.predict(X_test_fs)
annova_accuracy = accuracy_score(y_test,annova_predict)
print('Accuracy with ANNOVA Univariate Test:',annova_accuracy)
annova_report = pd.DataFrame(classification_report(y_test, annova_predict,␣
↪output_dict=True))
annova_report
```

Accuracy with ANNOVA Univariate Test: 0.9649122807017544

[30]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.973913	0.946429	0.964912	0.960171	0.964912
recall	0.973913	0.946429	0.964912	0.960171	0.964912
f1-score	0.973913	0.946429	0.964912	0.960171	0.964912
support	115.000000	56.000000	0.964912	171.000000	171.000000

[31]:
```
cm = confusion_matrix(y_test, annova_predict, labels=annova_model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=annova_model.
 ↪classes_)
disp.plot()
plt.grid(False)
plt.show()
```



1.8.6 Wrapper Method

a) Step Forward Feature Selection

[32]:
```
X_train_wrap,X_test_wrap,y_train_wrap,y_test_wrap = tts(X,Y,test_size=0.2)␣
 ↪#splitting the dataset
```

[33]:
```
sfs = SFS(GaussianNB(), k_features = 15, forward = True, floating = False,␣
 ↪verbose = 2, scoring = 'accuracy',cv=0)
sfs.fit(X_train_wrap, np.ravel(y_train_wrap))
sffs = sfs.transform(X_train_wrap)
```

```
[2023-09-10 14:47:48] Features: 1/15 -- score: 0.9142857142857143
[2023-09-10 14:47:48] Features: 2/15 -- score: 0.9472527472527472
```

```
[2023-09-10 14:47:48] Features: 3/15 -- score: 0.9604395604395605
[2023-09-10 14:47:48] Features: 4/15 -- score: 0.9626373626373627
[2023-09-10 14:47:48] Features: 5/15 -- score: 0.9648351648351648
[2023-09-10 14:47:48] Features: 6/15 -- score: 0.9714285714285714
[2023-09-10 14:47:48] Features: 7/15 -- score: 0.967032967032967
[2023-09-10 14:47:48] Features: 8/15 -- score: 0.9648351648351648
[2023-09-10 14:47:48] Features: 9/15 -- score: 0.9692307692307692
[2023-09-10 14:47:48] Features: 10/15 -- score: 0.9648351648351648
[2023-09-10 14:47:48] Features: 11/15 -- score: 0.9648351648351648
[2023-09-10 14:47:48] Features: 12/15 -- score: 0.9692307692307692
[2023-09-10 14:47:48] Features: 13/15 -- score: 0.9736263736263736
[2023-09-10 14:47:48] Features: 14/15 -- score: 0.9692307692307692
[2023-09-10 14:47:48] Features: 15/15 -- score: 0.9692307692307692
```

[34]:

```python
sf_model = GaussianNB()
sf_model.fit(sffs,np.ravel(y_train_wrap))
sf_predict = sf_model.predict(sfs.transform(X_test_wrap))
sf_accuracy = accuracy_score(y_test_wrap,sf_predict)
print('Accuracy with Step Forward Feature Selection:',sf_accuracy)
sf_report = pd.DataFrame(classification_report(y_test_wrap, sf_predict,
 ↪output_dict=True))
sf_report
```

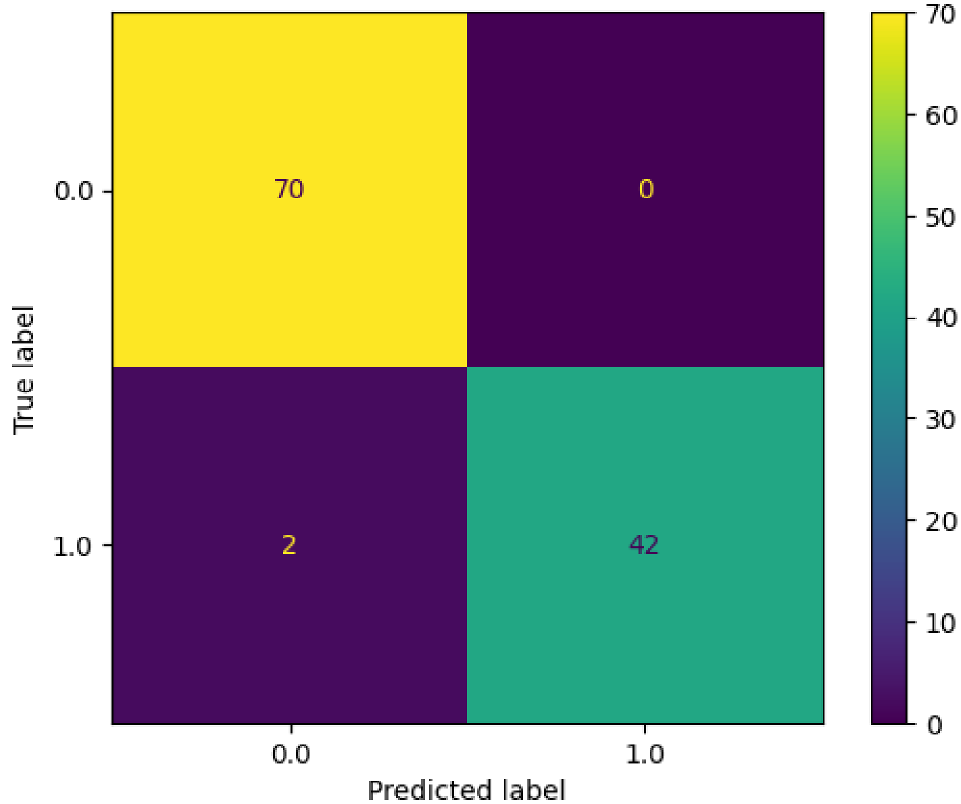
```
Accuracy with Step Forward Feature Selection: 0.9824561403508771
```

[34]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.972222	1.000000	0.982456	0.986111	0.982943
recall	1.000000	0.954545	0.982456	0.977273	0.982456
f1-score	0.985915	0.976744	0.982456	0.981330	0.982376
support	70.000000	44.000000	0.982456	114.000000	114.000000

[35]:

```python
cm = confusion_matrix(y_test_wrap, sf_predict, labels=sf_model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=sf_model.
 ↪classes_)
disp.plot()
plt.grid(False)
plt.show()
```



1.8.7 Wrapper Method

b) Step Backward Feature Selection

[36]:
```
sbs = SFS(GaussianNB(), k_features = 24, forward = False, floating = False,␣
↪verbose = 2, scoring = 'accuracy',cv=0)
sbs.fit(X_train_wrap, np.ravel(y_train_wrap))
sbfs = sbs.transform(X_train_wrap)
```

```
[2023-09-10 14:47:49] Features: 29/24 -- score: 0.9428571428571428
[2023-09-10 14:47:49] Features: 28/24 -- score: 0.9494505494505494
[2023-09-10 14:47:49] Features: 27/24 -- score: 0.9494505494505494
[2023-09-10 14:47:49] Features: 26/24 -- score: 0.9516483516483516
[2023-09-10 14:47:49] Features: 25/24 -- score: 0.9538461538461539
[2023-09-10 14:47:49] Features: 24/24 -- score: 0.9538461538461539
```

[37]:
```
sb_model = GaussianNB()
sb_model.fit(sbfs,np.ravel(y_train_wrap))
sb_predict = sb_model.predict(sbs.transform(X_test_wrap))
sb_accuracy = accuracy_score(y_test_wrap,sb_predict)
```

```python
print('Accuracy with Step Backward Feature Selection:',sb_accuracy)
sb_report = pd.DataFrame(classification_report(y_test_wrap, sb_predict,␣
    ↪output_dict=True))
sb_report
```

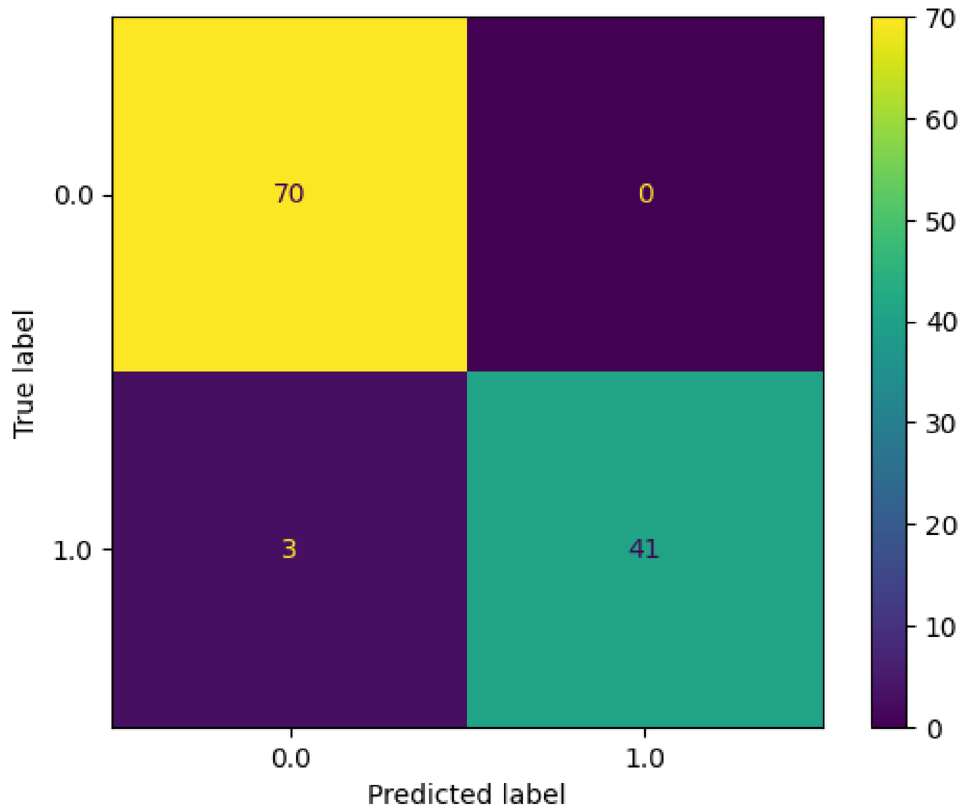
```
Accuracy with Step Backward Feature Selection: 0.9736842105263158
```

[37]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.958904	1.000000	0.973684	0.979452	0.974766
recall	1.000000	0.931818	0.973684	0.965909	0.973684
f1-score	0.979021	0.964706	0.973684	0.971863	0.973496
support	70.000000	44.000000	0.973684	114.000000	114.000000

[38]:

```python
cm = confusion_matrix(y_test_wrap, sb_predict, labels=sb_model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=sb_model.
    ↪classes_)
disp.plot()
plt.grid(False)
plt.show()
```



1.8.8 Wrapper Method

c) Step Floating Forward Feature Selection

[39]:
```
sfloat = SFS(GaussianNB(), k_features = 15, forward = True, floating = True,␣
↪verbose = 2, scoring = 'accuracy',cv=0)
sfloat.fit(X_train_wrap, np.ravel(y_train_wrap))
sfloat_fw = sfloat.transform(X_train_wrap)
```

```
[2023-09-10 14:47:49] Features: 1/15 -- score: 0.9142857142857143
[2023-09-10 14:47:49] Features: 2/15 -- score: 0.9472527472527472
[2023-09-10 14:47:49] Features: 3/15 -- score: 0.9604395604395605
[2023-09-10 14:47:49] Features: 4/15 -- score: 0.9626373626373627
[2023-09-10 14:47:50] Features: 5/15 -- score: 0.9648351648351648
[2023-09-10 14:47:50] Features: 6/15 -- score: 0.9714285714285714
[2023-09-10 14:47:50] Features: 7/15 -- score: 0.967032967032967
[2023-09-10 14:47:50] Features: 8/15 -- score: 0.9648351648351648
[2023-09-10 14:47:50] Features: 9/15 -- score: 0.9692307692307692
[2023-09-10 14:47:50] Features: 10/15 -- score: 0.9648351648351648
[2023-09-10 14:47:50] Features: 10/15 -- score: 0.967032967032967
[2023-09-10 14:47:50] Features: 10/15 -- score: 0.9736263736263736
[2023-09-10 14:47:50] Features: 11/15 -- score: 0.9758241758241758
[2023-09-10 14:47:50] Features: 12/15 -- score: 0.9736263736263736
[2023-09-10 14:47:50] Features: 11/15 -- score: 0.978021978021978
[2023-09-10 14:47:50] Features: 12/15 -- score: 0.9758241758241758
[2023-09-10 14:47:50] Features: 13/15 -- score: 0.9714285714285714
[2023-09-10 14:47:50] Features: 14/15 -- score: 0.9714285714285714
[2023-09-10 14:47:50] Features: 14/15 -- score: 0.9736263736263736
[2023-09-10 14:47:50] Features: 15/15 -- score: 0.9692307692307692
```

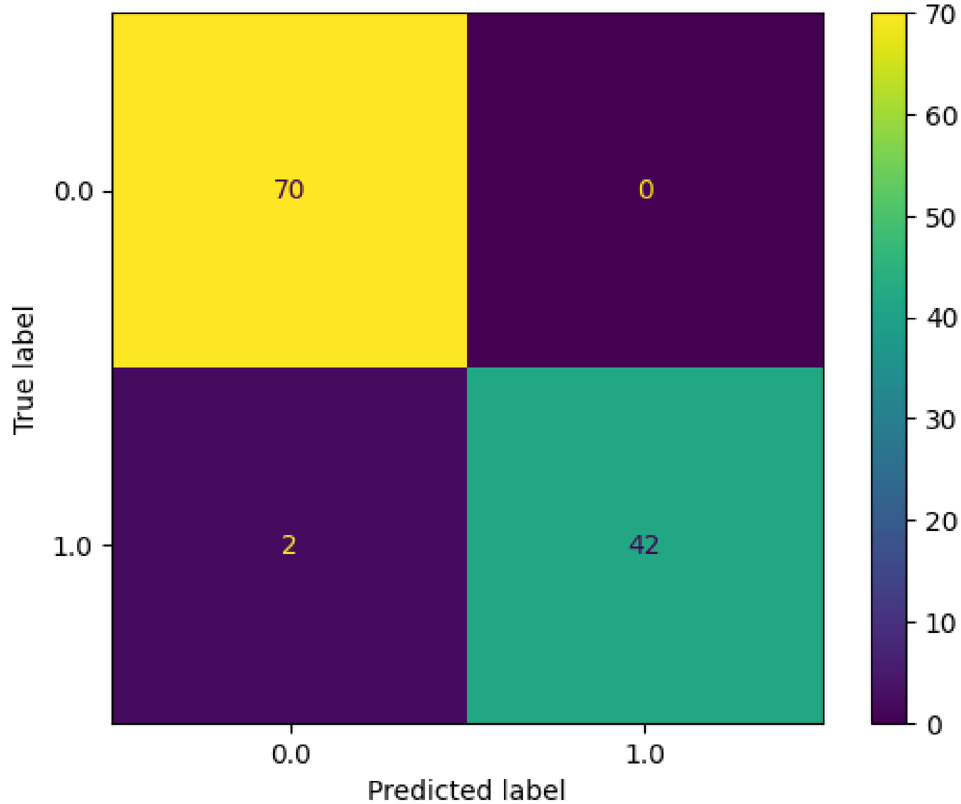
[40]:
```
sfloat_model = GaussianNB()
sfloat_model.fit(sfloat_fw,np.ravel(y_train_wrap))
sfloat_predict = sfloat_model.predict(sfloat.transform(X_test_wrap))
sfloat_accuracy = accuracy_score(y_test_wrap,sfloat_predict)
print('Accuracy with Step Floating Forward Feature Selection:',sfloat_accuracy)
sfloat_report = pd.DataFrame(classification_report(y_test_wrap, sfloat_predict,␣
↪output_dict=True))
sfloat_report
```

Accuracy with Step Floating Forward Feature Selection: 0.9824561403508771

[40]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.972222	1.000000	0.982456	0.986111	0.982943
recall	1.000000	0.954545	0.982456	0.977273	0.982456
f1-score	0.985915	0.976744	0.982456	0.981330	0.982376
support	70.000000	44.000000	0.982456	114.000000	114.000000

[41]:
```
cm = confusion_matrix(y_test_wrap, sfloat_predict, labels=sfloat_model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=sfloat_model.
 ↪classes_)
disp.plot()
plt.grid(False)
plt.show()
```



1.8.9 Wrapper Method

d) Step Floating Backward Feature Selection

[42]:
```
sfloatbw = SFS(GaussianNB(), k_features = 12, forward = False, floating = True,␣
 ↪verbose = 2, scoring = 'accuracy',cv=0)
sfloatbw.fit(X_train_wrap, np.ravel(y_train_wrap))
sfloat_bw = sfloatbw.transform(X_train_wrap)
```

```
[2023-09-10 14:47:51] Features: 29/12 -- score: 0.9428571428571428
[2023-09-10 14:47:51] Features: 28/12 -- score: 0.9494505494505494
[2023-09-10 14:47:51] Features: 27/12 -- score: 0.9494505494505494
[2023-09-10 14:47:51] Features: 26/12 -- score: 0.9516483516483516
```

```
[2023-09-10 14:47:51] Features: 25/12 -- score: 0.9538461538461539
[2023-09-10 14:47:51] Features: 24/12 -- score: 0.9538461538461539
[2023-09-10 14:47:51] Features: 23/12 -- score: 0.9538461538461539
[2023-09-10 14:47:51] Features: 22/12 -- score: 0.9538461538461539
[2023-09-10 14:47:51] Features: 21/12 -- score: 0.9538461538461539
[2023-09-10 14:47:51] Features: 21/12 -- score: 0.9604395604395605
[2023-09-10 14:47:51] Features: 20/12 -- score: 0.9626373626373627
[2023-09-10 14:47:51] Features: 19/12 -- score: 0.9648351648351648
[2023-09-10 14:47:51] Features: 18/12 -- score: 0.9648351648351648
[2023-09-10 14:47:52] Features: 17/12 -- score: 0.9648351648351648
[2023-09-10 14:47:52] Features: 16/12 -- score: 0.9648351648351648
[2023-09-10 14:47:52] Features: 15/12 -- score: 0.9648351648351648
[2023-09-10 14:47:52] Features: 15/12 -- score: 0.9692307692307692
[2023-09-10 14:47:52] Features: 14/12 -- score: 0.967032967032967
[2023-09-10 14:47:52] Features: 13/12 -- score: 0.9692307692307692
[2023-09-10 14:47:52] Features: 12/12 -- score: 0.9648351648351648
```

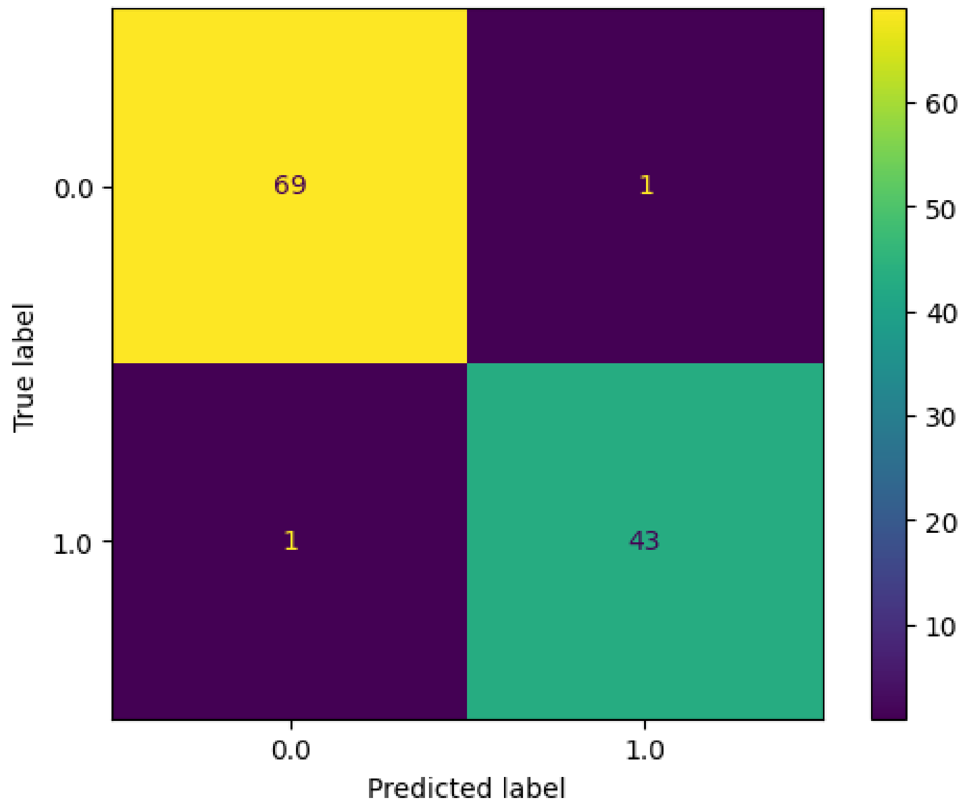
[43]:
```
sfloatbw_model = GaussianNB()
sfloatbw_model.fit(sfloat_bw,np.ravel(y_train_wrap))
sfloatbw_predict = sfloatbw_model.predict(sfloatbw.transform(X_test_wrap))
sfloatbw_accuracy = accuracy_score(y_test_wrap,sfloatbw_predict)
print('Accuracy with Step Floating Backward Feature Selection:
 ↪',sfloatbw_accuracy)
sfloatbw_report = pd.DataFrame(classification_report(y_test_wrap,␣
 ↪sfloatbw_predict, output_dict=True))
sfloatbw_report
```

```
Accuracy with Step Floating Backward Feature Selection: 0.9824561403508771
```

[43]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.985714	0.977273	0.982456	0.981494	0.982456
recall	0.985714	0.977273	0.982456	0.981494	0.982456
f1-score	0.985714	0.977273	0.982456	0.981494	0.982456
support	70.000000	44.000000	0.982456	114.000000	114.000000

[44]:
```
cm = confusion_matrix(y_test_wrap, sfloatbw_predict, labels=sfloatbw_model.
 ↪classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,␣
 ↪display_labels=sfloatbw_model.classes_)
disp.plot()
plt.grid(False)
plt.show()
```



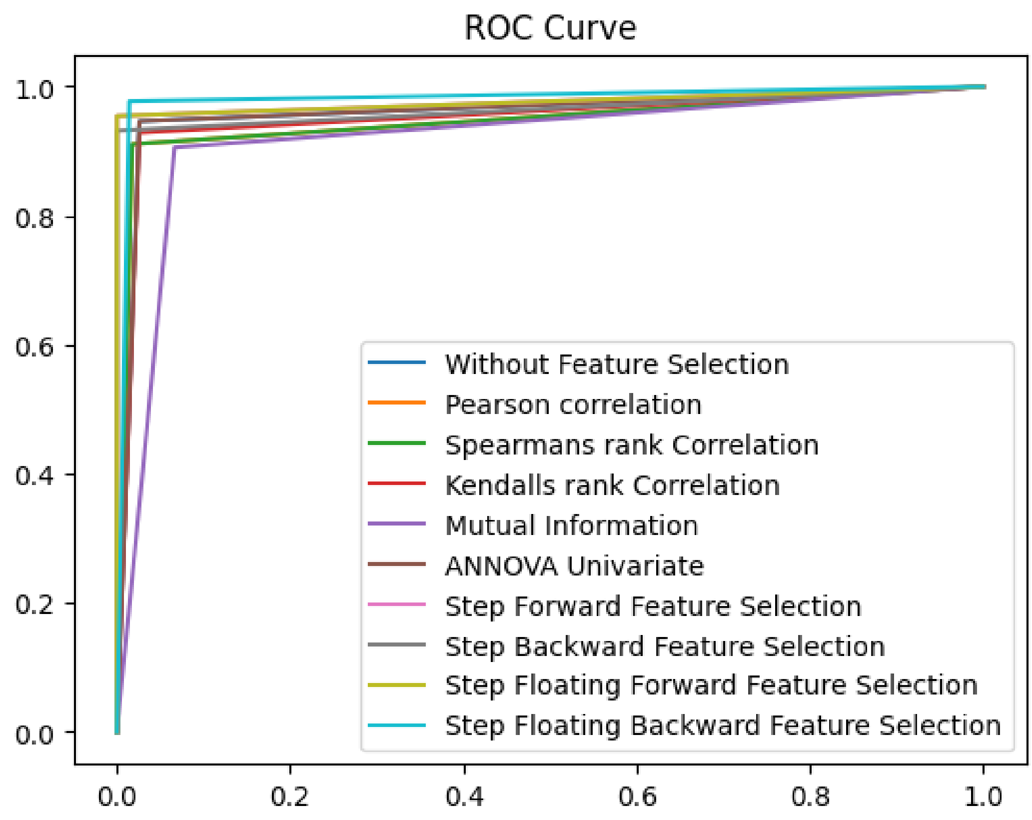
1.8.10 Comparisons

ROC Curve

[45]:
```
fpr, tpr, _ = metrics.roc_curve(y_test,predict)
plt.plot(fpr, tpr)
fpr_pcc, tpr_pcc, _ = metrics.roc_curve(y_test,pcc_predict)
plt.plot(fpr_pcc, tpr_pcc)
fpr_src, tpr_src, _ = metrics.roc_curve(y_test,src_predict)
plt.plot(fpr_src, tpr_src)
fpr_krc, tpr_krc, _ = metrics.roc_curve(y_test,krc_predict)
plt.plot(fpr_krc, tpr_krc)
fpr_mic, tpr_mic, _ = metrics.roc_curve(mic_y_test,mic_predict)
plt.plot(fpr_mic, tpr_mic)
fpr_annova, tpr_annova, _ = metrics.roc_curve(y_test,annova_predict)
plt.plot(fpr_annova, tpr_annova)
fpr_sf, tpr_sf, _ = metrics.roc_curve(y_test_wrap,sf_predict)
plt.plot(fpr_sf, tpr_sf)
fpr_sb, tpr_sb, _ = metrics.roc_curve(y_test_wrap,sb_predict)
plt.plot(fpr_sb, tpr_sb)
fpr_sfloar, tpr_sfloat, _ = metrics.roc_curve(y_test_wrap,sfloat_predict)
plt.plot(fpr_sfloar, tpr_sfloat)
```

```
fpr_sfloatbw, tpr_sfloatbw, _ = metrics.roc_curve(y_test_wrap,sfloatbw_predict)
plt.plot(fpr_sfloatbw, tpr_sfloatbw)
plt.title('ROC Curve')
plt.legend(['Without Feature Selection','Pearson correlation','Spearmans rank␣
↪Correlation','Kendalls rank Correlation', 'Mutual Information', 'ANNOVA␣
↪Univariate', 'Step Forward Feature Selection','Step Backward Feature␣
↪Selection','Step Floating Forward Feature Selection','Step Floating Backward␣
↪Feature Selection'])
```

[45]: <matplotlib.legend.Legend at 0x78a034cf3e80>

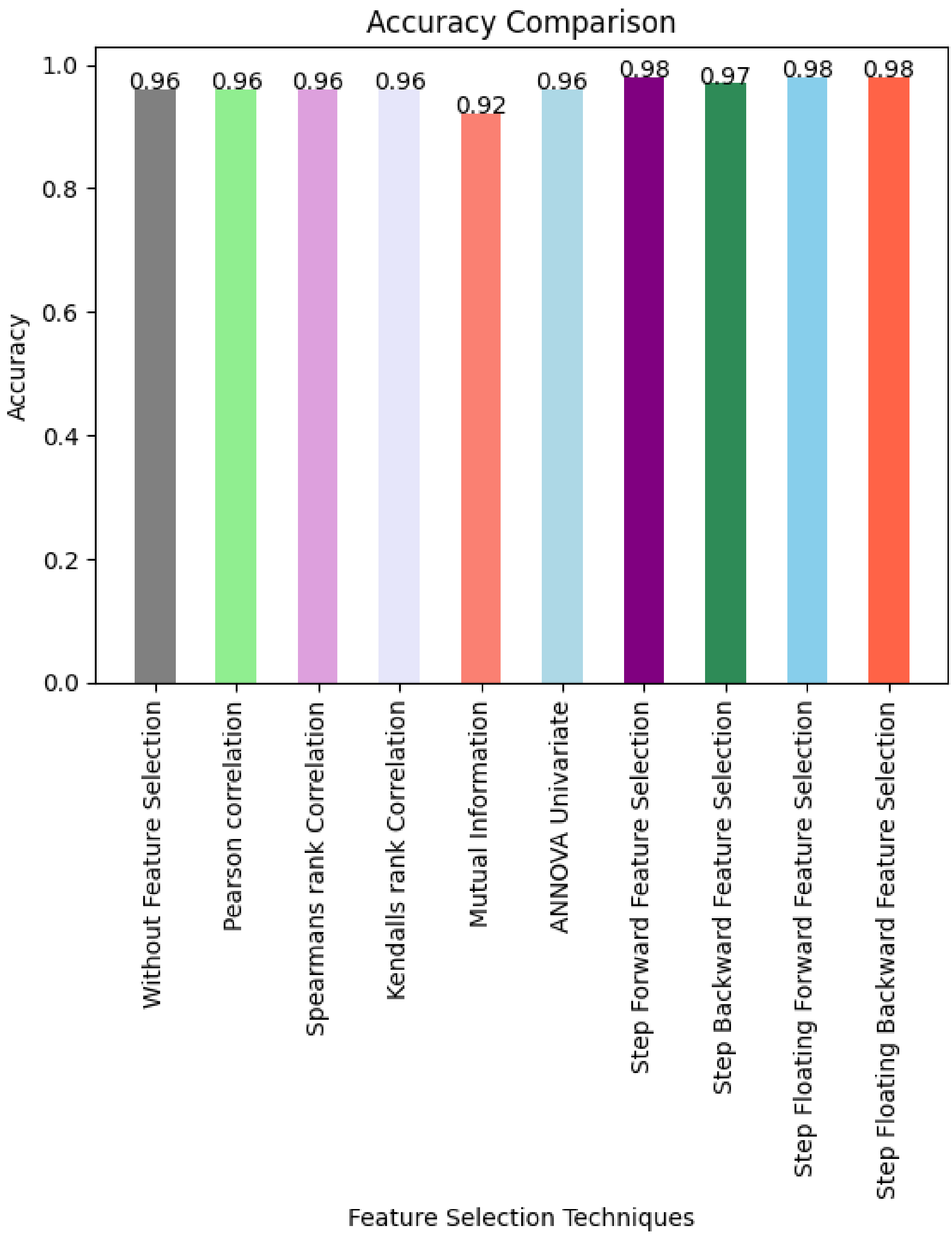


1.8.11 Comparisons

Accuracy Comparison

[46]:
```
x_labels = ['Without Feature Selection','Pearson correlation','Spearmans rank␣
↪Correlation','Kendalls rank Correlation', 'Mutual Information', 'ANNOVA␣
↪Univariate', 'Step Forward Feature Selection','Step Backward Feature␣
↪Selection','Step Floating Forward Feature Selection','Step Floating Backward␣
↪Feature Selection']
```

```python
y_labels = [round(accuracy,2),round(pcc_accuracy,2), round(src_accuracy,2),␣
↪round(krc_accuracy,2), round(mic_accuracy,2), round(annova_accuracy,2),␣
↪round(sf_accuracy,2), round(sb_accuracy,2), round(sfloat_accuracy,2),␣
↪round(sfloatbw_accuracy,2)]
def addlabels(x_labels,y_labels):
    for i in range(len(x_labels)):
        plt.text(i,y_labels[i],y_labels[i], ha = 'center')
addlabels(x_labels,y_labels)
plt.bar(x_labels,y_labels, width=0.5, align='center',␣
↪color=['gray','lightgreen', 'plum', 'lavender', 'salmon', 'lightblue',
                    'purple', 'seagreen', 'skyblue', 'tomato'])
plt.xlabel('Feature Selection Techniques')
plt.ylabel('Accuracy')
plt.title('Accuracy Comparison')
plt.xticks(rotation = 90)
plt.show()
```



1.8.12 Comparisons

AUC Value

[47]:
```
auc = metrics.roc_auc_score(y_test,predict)
auc_pcc = metrics.roc_auc_score(y_test,pcc_predict)
```

```python
auc_src = metrics.roc_auc_score(y_test,src_predict)
auc_krc = metrics.roc_auc_score(y_test,krc_predict)
auc_mic = metrics.roc_auc_score(mic_y_test,mic_predict)
auc_annova = metrics.roc_auc_score(y_test,annova_predict)
auc_sf = metrics.roc_auc_score(y_test_wrap,sf_predict)
auc_sb = metrics.roc_auc_score(y_test_wrap,sb_predict)
auc_sfloat = metrics.roc_auc_score(y_test_wrap,sfloat_predict)
auc_sfloatbw = metrics.roc_auc_score(y_test_wrap,sfloatbw_predict)

print('AUC Value for Without Feature Selection:',auc)
print('AUC Value for Pearson:',auc_pcc)
print('AUC Value for Spearman:',auc_src)
print('AUC Value for Kendall:',auc_krc)
print('AUC Value for Mutual Information:',auc_mic)
print('AUC Value for Annova:',auc_annova)
print('AUC Value for Step Forward Feature Selection:',auc_sf)
print('AUC Value for Step Backward Feature Selection:',auc_sb)
print('AUC Value for Step Floating Forward Feature Selection:',auc_sfloat)
print('AUC Value for Step Floating Backward Feature Selection:',auc_sfloatbw)
```

```
AUC Value for Without Feature Selection: 0.9601708074534161
AUC Value for Pearson: 0.9466614906832297
AUC Value for Spearman: 0.9466614906832297
AUC Value for Kendall: 0.9512422360248447
AUC Value for Mutual Information: 0.9194968553459120
AUC Value for Annova: 0.9601708074534161
AUC Value for Step Forward Feature Selection: 0.9772727272727273
AUC Value for Step Backward Feature Selection: 0.9659090909090908
AUC Value for Step Floating Forward Feature Selection: 0.9772727272727273
AUC Value for Step Floating Backward Feature Selection: 0.9814935064935065
```