06_feature_extraction

November 22, 2021

1 Data Mining

Activity performed for the discipline "Data Mining"
Matheus Pimenta

1.0.1 Material

Training dataset was provided by Rfam9 and two small non-coding RNA families are selected. The intron small non-coding RNA Rfam family is composed of two subclasses, which are Intron-gpI and Intron-gpII.

Group I and group II introns are found in genes encoding proteins (messenger RNA), transfer RNA, and ribosomal RNA in a very wide range of living organisms. Following transcription into RNA, group I and group II introns also make extensive internal interactions that allow them to fold into a specific, complex three-dimensional architecture. These complex architectures allow some group I and group II introns to be self-splicing, that is, the intron-containing RNA molecule can rearrange its own covalent structure so as to precisely remove the intron and link the exons together in the correct order. Group I and group II introns are distinguished by different sets of internal conserved sequences and folded structures, and by the fact that splicing of RNA molecules containing group II introns generates branched introns (like those of spliceosomal RNAs), while group I introns use a non-encoded guanosine nucleotide (typically GTP) to initiate splicing, adding it on to the 5'-end of the excised intron.

1.0.2 Methods

Two methods extract complex network features from the fasta sequence.
The first is BASiNET and the second is RNAcon.

1.1 Processing the RAW datasets

1.1.1 Using the seqkit to pre-process the FASTA files.

Counting the sequences

[ ]:
```
!grep ">" intron_gpI.fasta | wc -l
!grep ">" intron_gpII.fasta | wc -l
```

```
7295
1464
```

Removing the duplicated sequences

```
[ ]: !seqkit rmdup -s < intron_gpI.fasta > i_gpI_nodup.fa
     !seqkit rmdup -s < intron_gpII.fasta > i_gpII_nodup.fa
```

```
[INFO] 0 duplicated records removed
[INFO] 0 duplicated records removed
```

The sequences will be considered pre-processed in this case.
But, if necessary, the pre-processing step can be applied to homogenize the sequences.

Sampling the sequences

```
[ ]: !seqkit sample -s 23 -2 -n 1000 i_gpI_nodup.fa > i_gpI_sample.fasta
     !seqkit sample -s 23 -2 -n 1000 i_gpII_nodup.fa > i_gpII_sample.fasta
```

```
[INFO] sample by number
[INFO] first pass: counting seq number
[INFO] seq number: 7295
[INFO] second pass: reading and sampling
[INFO] 1000 sequences outputted
[INFO] sample by number
[INFO] first pass: counting seq number
[INFO] seq number: 1464
[INFO] second pass: reading and sampling
[INFO] 1000 sequences outputted
```

Rename the header *This step is optional.*

```
[ ]: !seqkit replace -p '.+' -r 'Intron_GPI_{nr}' i_gpI_sample.fasta > i_gpI.fasta
     !seqkit replace -p '.+' -r 'Intron_GPII_{nr}' i_gpII_sample.fasta > i_gpII.fasta
```

Recounting the sequences

```
[ ]: !grep ">" i_gpI.fasta | wc -l
     !grep ">" i_gpII.fasta | wc -l
```

```
1000
1000
```

The feature extraction step will be performed on the terminal using the two methods shown above.
After that, two datasets will be created.

1.2 Processing the Extracted Features

```
[ ]: import pandas as pd
```

```
[ ]: # BASiNET
     basinet_gpI = pd.read_csv("basinet_i_gpI.csv",
                               sep = ",",
                               index_col = 0)
     print("BASiNET Intron-gpI shape:", basinet_gpI.shape)
     basinet_gpII = pd.read_csv("basinet_i_gpII.csv",
                                sep = ",",
                                index_col = 0)
     print("BASiNET Intron-gpII shape:", basinet_gpII.shape)
     # Concatenate the dataframes - BASiNET
     basinet = pd.concat([basinet_gpI, basinet_gpII],
                         ignore_index = True)
     class_names = basinet["CLASS"]
     print("BASiNET - Full Dataset shape:", basinet.shape)
     # RNAcon
     rnacon_gpI = pd.read_csv("rnacon_i_gpI.csv",
                              sep = ",")
     #Drop the last NULL column
     rnacon_gpI = rnacon_gpI.iloc[:,:-1]
     print("RNAcon Intron-gpI shape:", rnacon_gpI.shape)
     rnacon_gpII = pd.read_csv("rnacon_i_gpII.csv",
                               sep = ",")
     #Drop the last NULL column
     rnacon_gpII = rnacon_gpII.iloc[:,:-1]
     print("RNAcon Intron-gpII shape:", rnacon_gpII.shape)
     # Concatenate the dataframes - RNAcon
     rnacon = pd.concat([rnacon_gpI, rnacon_gpII],
                        ignore_index = True)
     columns_name = {'## Articulation points' : 'ART.POINTS',
                     'Average path length' : 'MEAN.PATH.LEN',
                     'Average node betweenness' : 'MEAN.BET',
                     'Variance of node betweenness' : 'VAR.BET',
                     'Average edge betweenness' : 'MEAN.EDGE.BET',
                     'Variance of edge betweenness' : 'VAR.EDGE.BET',
                     'Average co-citation coupling' : 'MEAN.CIT.COUP',
                     'Average bibliographic coupling' : 'MEAN.BIB.COUP',
                     'Average closeness centrality' : 'MEAN.CLOSE.CENT',
                     'Variance of closeness centrality' : 'VAR.CLOSE.CENT',
                     'Average Burts constraint' : 'MEAN.BURT',
                     'Variance of Burts constraint' : 'VAR.BURT',
                     'Average degree' : 'MEAN.DEG',
                     'Diameter' : 'DIAMETER',
                     'Girth' : 'GIRTH',
                     'Average coreness' : 'MEAN.CORE',
                     'Variance of coreness' : 'VAR.CORE',
                     'Maximum coreness' : 'MAX.CORE',
                     'Graph density' : 'DENSITY',
                     'Transitivity' : 'TRANSITIVITY'}
     # Rename the columns
```

```
rnacon.rename(columns = columns_name,
              inplace = True)
print("RNAcon - Full Dataset shape:", rnacon.shape)
# Full Dataframe
df = pd.concat([rnacon,basinet],
               axis = 1)
print("Full Dataframe shape:", df.shape)
# Include the "CLASS" column in the RNAcon dataset
rnacon["CLASS"] = class_names
print("RNAcon with CLASS column shape:", rnacon.shape)
```

```
BASiNET Intron-gpI shape: (1000, 21)
BASiNET Intron-gpII shape: (1000, 21)
BASiNET - Full Dataset shape: (2000, 21)
RNAcon Intron-gpI shape: (1000, 20)
RNAcon Intron-gpII shape: (1000, 20)
RNAcon - Full Dataset shape: (2000, 20)
Full Dataframe shape: (2000, 41)
RNAcon with CLASS column shape: (2000, 21)
```

1.3 Basic measures

No features were removed in this step.
The Feature Selection step will be executed later.

[ ]:
```
import missingno as msno
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

1.3.1 BASiNET

[ ]:
```
basinet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 21 columns):
ASS.1     2000 non-null float64
ASS.2     2000 non-null float64
BET.1     2000 non-null float64
BET.2     2000 non-null float64
ASPL.1    2000 non-null float64
ASPL.2    2000 non-null float64
CC.1      2000 non-null float64
CC.2      2000 non-null float64
DEG.1     2000 non-null float64
DEG.2     2000 non-null float64
MIN.1     2000 non-null int64
MIN.2     2000 non-null int64
MAX.1     2000 non-null int64
MAX.2     2000 non-null int64
SD.1      2000 non-null float64
SD.2      2000 non-null float64
MT3.1     2000 non-null int64
MT3.2     2000 non-null int64
MT4.1     2000 non-null int64
MT4.2     2000 non-null int64
CLASS     2000 non-null object
dtypes: float64(12), int64(8), object(1)
memory usage: 328.2+ KB
```

```
[ ]: basinet.isna().sum().sort_values(ascending=False)
```

```
[ ]: CLASS     0
     DEG.2     0
     ASS.2     0
     BET.1     0
     BET.2     0
     ASPL.1    0
     ASPL.2    0
     CC.1      0
     CC.2      0
     DEG.1     0
     MIN.1     0
     MT4.2     0
     MIN.2     0
     MAX.1     0
     MAX.2     0
     SD.1      0
     SD.2      0
     MT3.1     0
     MT3.2     0
     MT4.1     0
     ASS.1     0
     dtype: int64
```

```
[ ]: basinet.isnull().sum().sort_values(ascending=False)
[ ]: CLASS     0
     DEG.2     0
     ASS.2     0
     BET.1     0
     BET.2     0
     ASPL.1    0
     ASPL.2    0
     CC.1      0
     CC.2      0
     DEG.1     0
     MIN.1     0
     MT4.2     0
     MIN.2     0
     MAX.1     0
     MAX.2     0
     SD.1      0
     SD.2      0
     MT3.1     0
     MT3.2     0
     MT4.1     0
     ASS.1     0
     dtype: int64
```

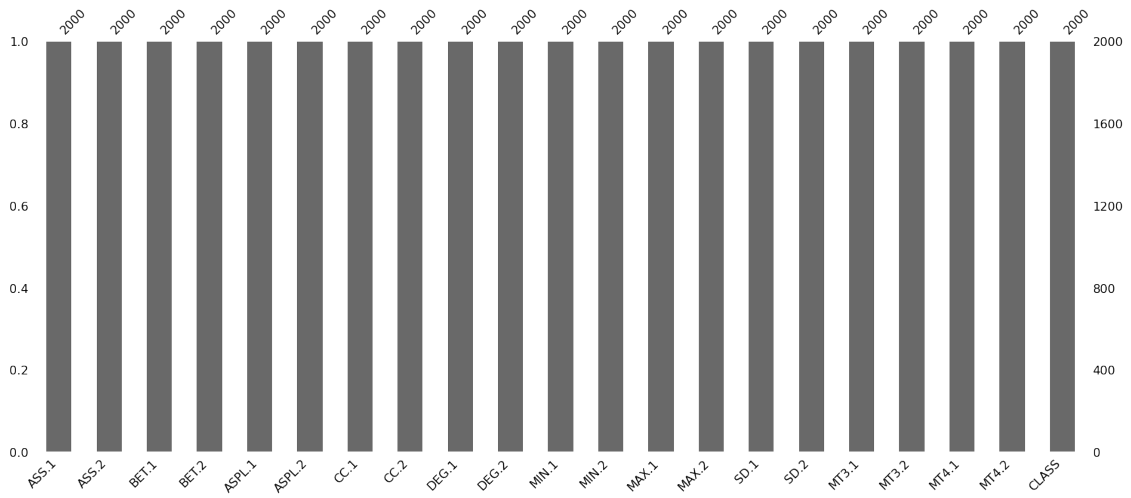
```
[ ]: pd.options.display.float_format = "{:.2f}".format
     print("Describe:\n",basinet.describe().T)
```

```
Describe:
           count     mean      std    min    25%     50%      75%       max
ASS.1   2000.00     0.03     0.13  -0.38  -0.05    0.02     0.10      0.83
ASS.2   2000.00    -0.05     0.32  -1.00  -0.17    0.00     0.00      1.00
BET.1   2000.00    40.84    11.99  14.56  30.59   37.59    49.36    128.16
BET.2   2000.00     9.72    11.88   0.00   0.00    0.55    20.54     52.13
ASPL.1  2000.00     2.69     0.70   1.67   1.99    2.51     3.28      5.50
ASPL.2  2000.00    40.01    12.36   0.00  36.12   41.62    46.34     72.37
CC.1    2000.00     0.18     0.11   0.00   0.08    0.15     0.29      0.49
CC.2    2000.00     0.08     0.10   0.00   0.00    0.00     0.16      1.00
DEG.1   2000.00     8.53     5.67   2.54   3.36    6.04    14.34     31.97
DEG.2   2000.00     2.73     3.02   0.00   0.24    1.00     5.26     19.08
MIN.1   2000.00    26.62    20.87   1.00   3.00   32.00    42.25     75.00
MIN.2   2000.00     2.81     3.69   0.00   1.00    1.00     3.00     33.00
MAX.1   2000.00    14.79     9.88   1.00   8.00   14.00    20.00     60.00
MAX.2   2000.00    14.29    11.18   0.00   5.00   14.00    21.00     60.00
SD.1    2000.00     6.64     5.29   1.02   1.98    3.94    11.47     43.70
SD.2    2000.00     4.34     4.41   0.00   0.73    1.92     7.87     41.27
MT3.1   2000.00  1905.66  1921.03  88.00 199.00  827.00  4001.50  12255.00
MT3.2   2000.00   149.86   216.26   0.00   0.00    9.00   291.00   2230.00
MT4.1   2000.00 21686.98 24969.11 171.00 717.00 5285.00 48058.50 196141.00
```

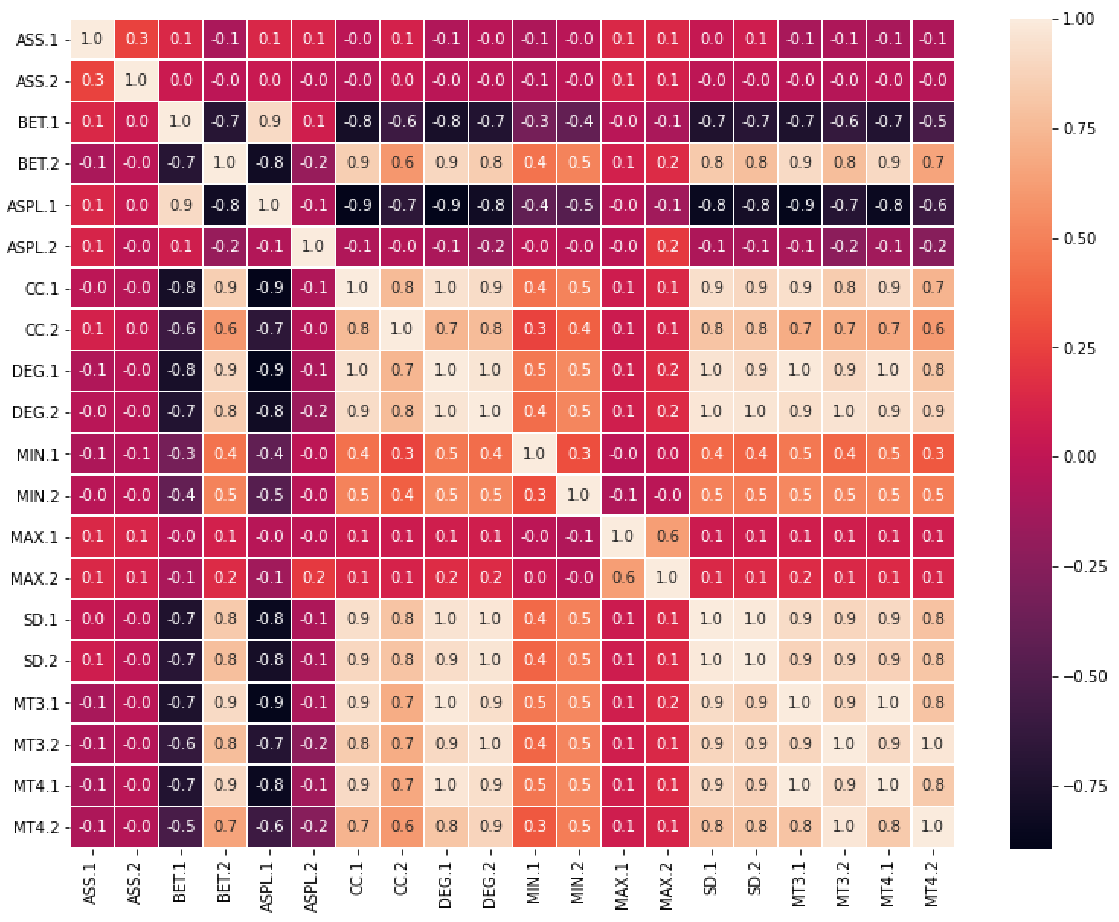
```
MT4.2  2000.00   838.57  1497.32   0.00   0.00    9.00  1426.75  22132.00
```

[ ]:
```
msno.bar(basinet)
```

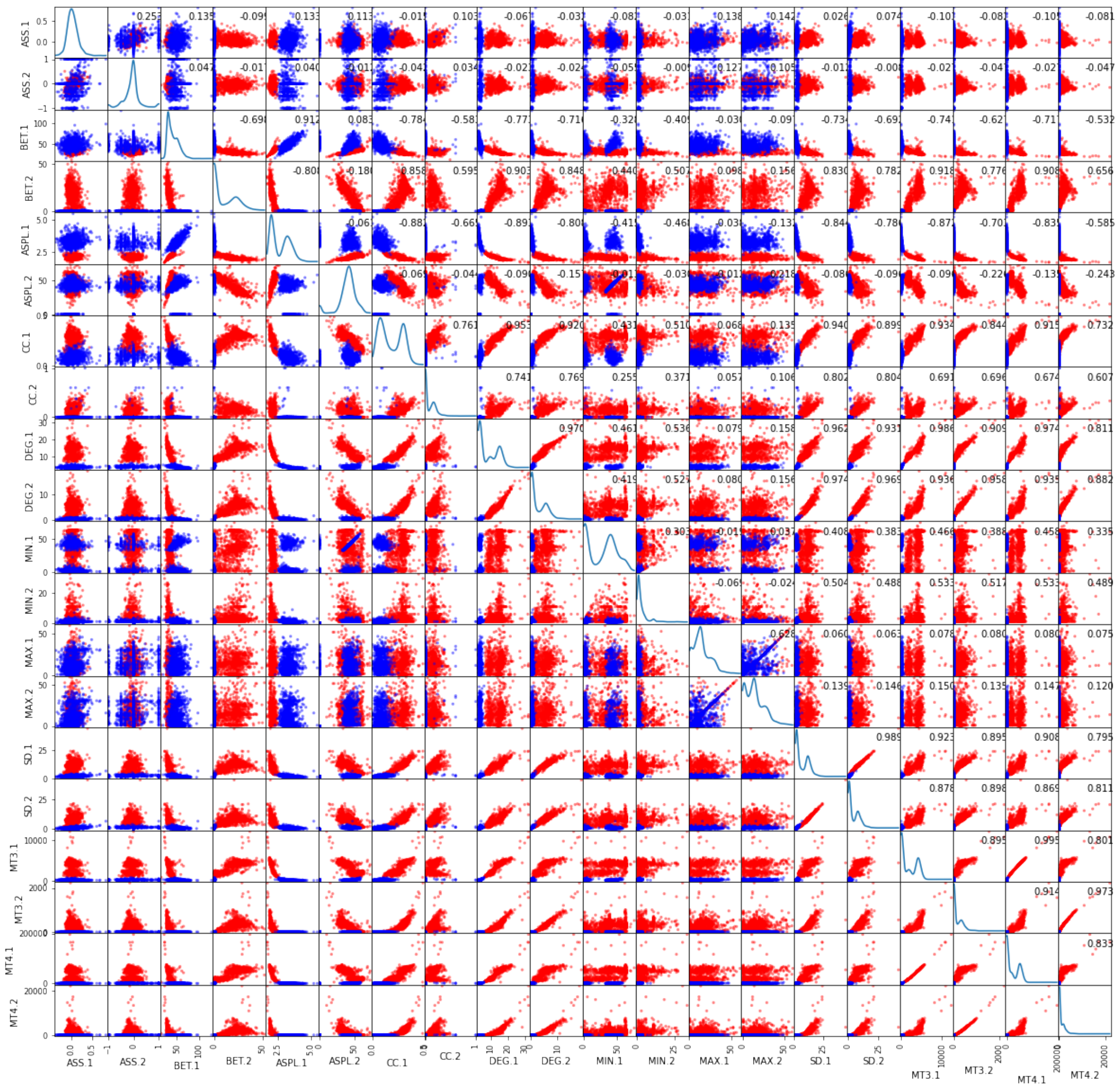
[ ]:
```
<AxesSubplot:>
```



[ ]:
```
f,ax = plt.subplots(figsize=(13, 10))
sns.heatmap(basinet.corr(),
            annot = True,
            linewidths=.5,
            fmt= '.1f',
            ax = ax)
plt.show()
```



[ ]:
```python
class_color = {"i_gpI" : "red",
               "i_gpII" : "blue"}
colors = [class_color[name] for name in basinet.CLASS]
ax = pd.plotting.scatter_matrix(basinet,
                                diagonal = "kde",
                                color = colors,
                                figsize=(20,20))
corr = np.asmatrix(basinet.corr())
for i, j in zip(*plt.np.triu_indices_from(ax, k=1)):
    ax[i, j].annotate("%.3f" %corr[i,j], (0.8, 0.8), xycoords='axes fraction', ha='center', va='center')
plt.show()
```



1.3.2 RNAcon

```
[ ]: rnacon.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 21 columns):
ART.POINTS          2000 non-null object
MEAN.PATH.LEN       2000 non-null float64
MEAN.BET            2000 non-null float64
VAR.BET             2000 non-null float64
MEAN.EDGE.BET       2000 non-null float64
VAR.EDGE.BET        2000 non-null float64
MEAN.CIT.COUP       2000 non-null float64
MEAN.BIB.COUP      2000 non-null float64
MEAN.CLOSE.CENT    2000 non-null float64
VAR.CLOSE.CENT     2000 non-null float64
MEAN.BURT          2000 non-null float64
VAR.BURT           2000 non-null float64
MEAN.DEG           2000 non-null float64
DIAMETER           2000 non-null int64
GIRTH              2000 non-null int64
MEAN.CORE          2000 non-null float64
VAR.CORE           2000 non-null float64
MAX.CORE           2000 non-null int64
DENSITY            2000 non-null float64
TRANSITIVITY       2000 non-null int64
CLASS              2000 non-null object
dtypes: float64(15), int64(4), object(2)
memory usage: 328.2+ KB
```

[ ]:
```
rnacon.isna().sum().sort_values(ascending=False)
```

[ ]:
```
CLASS              0
VAR.CLOSE.CENT     0
MEAN.PATH.LEN      0
MEAN.BET           0
VAR.BET            0
MEAN.EDGE.BET      0
VAR.EDGE.BET       0
MEAN.CIT.COUP      0
MEAN.BIB.COUP      0
MEAN.CLOSE.CENT    0
MEAN.BURT          0
TRANSITIVITY       0
VAR.BURT           0
MEAN.DEG           0
DIAMETER           0
GIRTH              0
MEAN.CORE          0
VAR.CORE           0
MAX.CORE           0
DENSITY            0
ART.POINTS         0
dtype: int64
```

[ ]:
```
rnacon.isnull().sum().sort_values(ascending=False)
```

[ ]:
```
CLASS              0
VAR.CLOSE.CENT     0
MEAN.PATH.LEN      0
MEAN.BET            0
VAR.BET             0
MEAN.EDGE.BET       0
VAR.EDGE.BET        0
MEAN.CIT.COUP       0
MEAN.BIB.COUP       0
MEAN.CLOSE.CENT     0
MEAN.BURT           0
TRANSITIVITY        0
VAR.BURT            0
MEAN.DEG            0
DIAMETER            0
GIRTH               0
MEAN.CORE           0
VAR.CORE            0
MAX.CORE            0
DENSITY             0
ART.POINTS          0
dtype: int64
```

```
[ ]: pd.options.display.float_format = "{:.3f}".format
     print("Describe:\n",rnacon.describe().T)
```

```
Describe:
                     count          mean           std       min        25%  \
MEAN.PATH.LEN    2000.000        33.001        23.131     8.224     13.662
MEAN.BET         2000.000      6057.844      7233.527   242.821    486.731
VAR.BET          2000.000 96302031.579 210802591.309 31211.461 182463.320
MEAN.EDGE.BET    2000.000      4597.436      5676.915   170.090    323.053
VAR.EDGE.BET     2000.000 83150848.327 178688922.809 22935.150 149754.096
MEAN.CIT.COUP    2000.000         0.058         0.045     0.006      0.012
MEAN.BIB.COUP    2000.000         0.058         0.045     0.006      0.012
MEAN.CLOSE.CENT 2000.000         0.001         0.000     0.000      0.000
VAR.CLOSE.CENT  2000.000         0.000         0.000     0.000      0.000
MEAN.BURT        2000.000         0.446         0.017     0.407      0.432
VAR.BURT         2000.000         0.008         0.004     0.003      0.005
MEAN.DEG         2000.000         2.967         0.310     2.212      2.710
DIAMETER         2000.000        89.701        62.962    19.000     39.000
GIRTH            2000.000         4.000         0.000     4.000      4.000
MEAN.CORE        2000.000         2.403         0.213     1.479      2.267
VAR.CORE         2000.000         0.375         0.126     0.174      0.303
MAX.CORE         2000.000         3.000         0.000     3.000      3.000
DENSITY          2000.000         0.024         0.018     0.003      0.006
TRANSITIVITY     2000.000         0.000         0.000     0.000      0.000

                     50%           75%           max
MEAN.PATH.LEN     23.476        49.750       142.700
MEAN.BET           1815.108      11151.197      46273.049
VAR.BET         2726241.122 140454975.447 3571253588.880
MEAN.EDGE.BET      1233.772       8330.002      36848.632
VAR.EDGE.BET    2102285.013 119159230.744 2771242958.713
MEAN.CIT.COUP         0.040          0.103          0.152
MEAN.BIB.COUP         0.040          0.103          0.152
MEAN.CLOSE.CENT       0.000          0.001          0.002
VAR.CLOSE.CENT        0.000          0.000          0.000
MEAN.BURT             0.446          0.459          0.497
VAR.BURT              0.007          0.012          0.015
MEAN.DEG              2.951          3.241          3.620
DIAMETER             63.500        130.000        419.000
GIRTH                 4.000          4.000          4.000
MEAN.CORE             2.419          2.580          2.789
VAR.CORE              0.349          0.397          0.923
MAX.CORE              3.000          3.000          3.000
DENSITY               0.017          0.042          0.059
TRANSITIVITY          0.000          0.000          0.000
```

There are some features with null standard deviation.
These features have been removed.

```
[ ]: col_to_drop = ["MEAN.CLOSE.CENT", "VAR.CLOSE.CENT", "GIRTH", "MAX.CORE",␣
      ↪"TRANSITIVITY"]

     rnacon.drop(columns = col_to_drop,
                 axis = 1,
                 inplace = True)
```

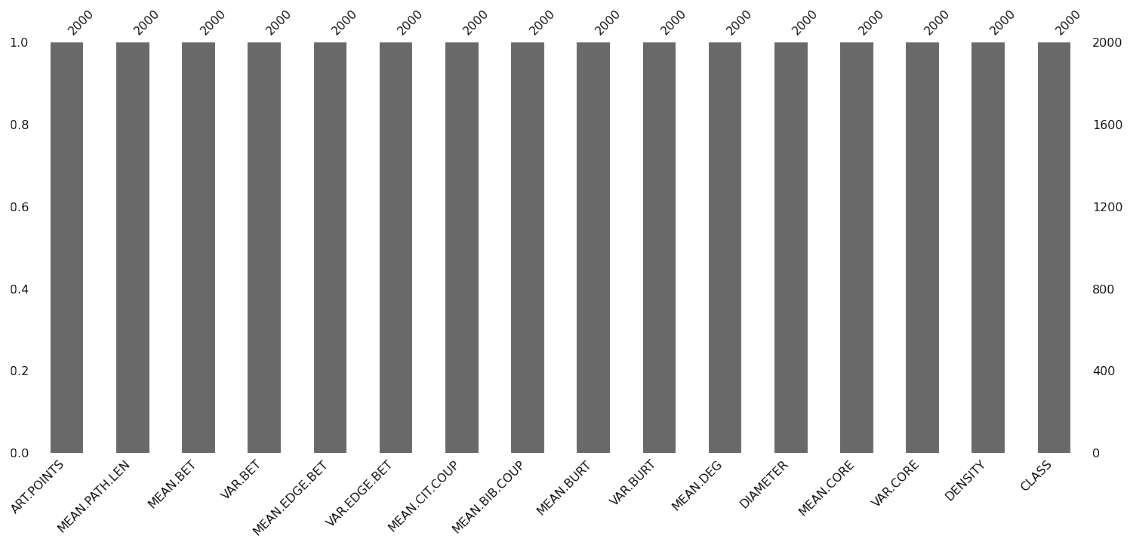
```
[ ]: pd.options.display.float_format = "{:.3f}".format
     print("Describe:\n",rnacon.describe().T)
```

```
Describe:
                 count          mean           std       min        25%  \
MEAN.PATH.LEN 2000.000        33.001        23.131     8.224     13.662
MEAN.BET      2000.000      6057.844      7233.527   242.821    486.731
VAR.BET       2000.000 96302031.579 210802591.309 31211.461 182463.320
MEAN.EDGE.BET 2000.000      4597.436      5676.915   170.090    323.053
VAR.EDGE.BET  2000.000 83150848.327 178688922.809 22935.150 149754.096
MEAN.CIT.COUP 2000.000         0.058         0.045     0.006      0.012
MEAN.BIB.COUP 2000.000         0.058         0.045     0.006      0.012
MEAN.BURT     2000.000         0.446         0.017     0.407      0.432
VAR.BURT      2000.000         0.008         0.004     0.003      0.005
MEAN.DEG      2000.000         2.967         0.310     2.212      2.710
DIAMETER      2000.000        89.701        62.962    19.000     39.000
MEAN.CORE     2000.000         2.403         0.213     1.479      2.267
VAR.CORE      2000.000         0.375         0.126     0.174      0.303
DENSITY       2000.000         0.024         0.018     0.003      0.006
```

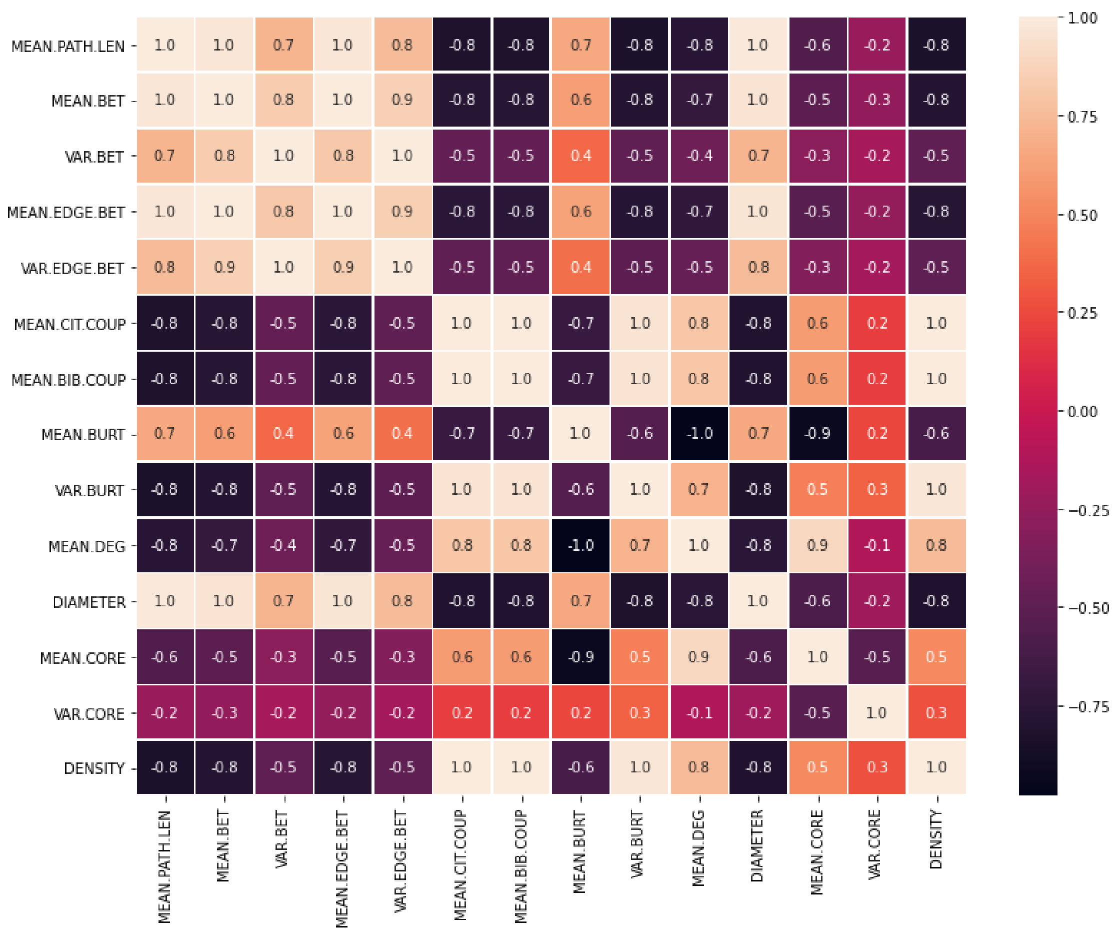
```
                        50%          75%            max
MEAN.PATH.LEN        23.476       49.750        142.700
MEAN.BET           1815.108    11151.197      46273.049
VAR.BET         2726241.122 140454975.447 3571253588.880
MEAN.EDGE.BET      1233.772     8330.002      36848.632
VAR.EDGE.BET    2102285.013 119159230.744 2771242958.713
MEAN.CIT.COUP         0.040        0.103          0.152
MEAN.BIB.COUP         0.040        0.103          0.152
MEAN.BURT             0.446        0.459          0.497
VAR.BURT              0.007        0.012          0.015
MEAN.DEG              2.951        3.241          3.620
DIAMETER             63.500      130.000        419.000
MEAN.CORE             2.419        2.580          2.789
VAR.CORE              0.349        0.397          0.923
DENSITY               0.017        0.042          0.059
```

[ ]:
```
msno.bar(rnacon)
```

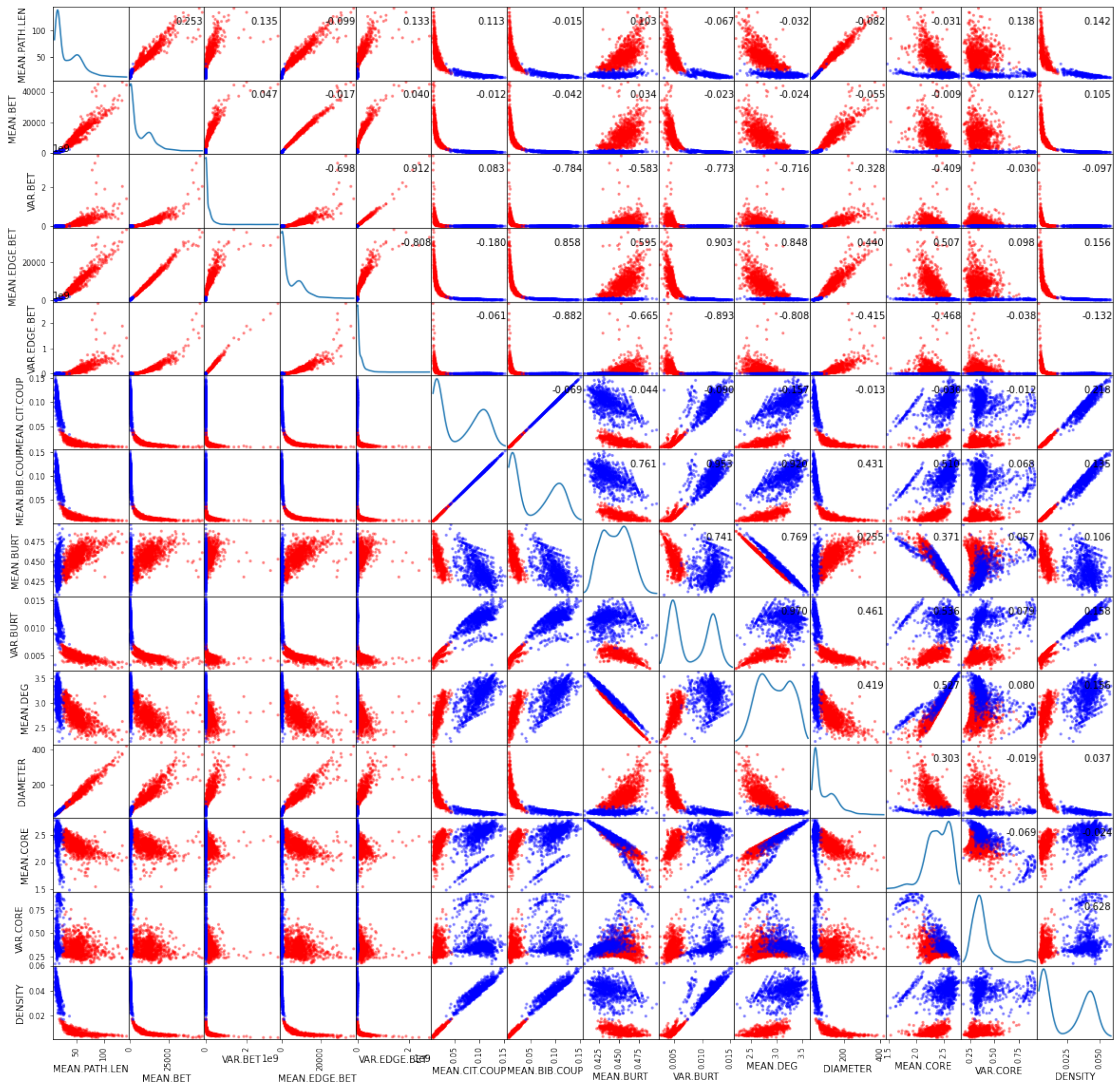
[ ]: `<AxesSubplot:>`



[ ]:
```
f,ax = plt.subplots(figsize=(13, 10))
sns.heatmap(rnacon.corr(),
            annot = True,
            linewidths=.5,
            fmt= '.1f',
            ax = ax)
plt.show()
```



```
[ ]: class_color = {"i_gpI" : "red",
                    "i_gpII" : "blue"}
     colors = [class_color[name] for name in rnacon.CLASS]

     ax = pd.plotting.scatter_matrix(rnacon,
                                     diagonal = "kde",
                                     color = colors,
                                     figsize=(20,20))
     corr = np.asmatrix(basinet.corr())
     for i, j in zip(*plt.np.triu_indices_from(ax, k=1)):
         ax[i, j].annotate("%.3f" %corr[i,j], (0.8, 0.8), xycoords='axes fraction',␣
      ↪ha='center', va='center')
     plt.show()
```



1.3.3 Full Dataset

```
[ ]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 41 columns):
ART.POINTS         2000 non-null object
MEAN.PATH.LEN      2000 non-null float64
MEAN.BET           2000 non-null float64
VAR.BET            2000 non-null float64
MEAN.EDGE.BET      2000 non-null float64
VAR.EDGE.BET       2000 non-null float64
MEAN.CIT.COUP      2000 non-null float64
MEAN.BIB.COUP      2000 non-null float64
MEAN.CLOSE.CENT    2000 non-null float64
VAR.CLOSE.CENT     2000 non-null float64
MEAN.BURT          2000 non-null float64
VAR.BURT           2000 non-null float64
MEAN.DEG           2000 non-null float64
DIAMETER           2000 non-null int64
GIRTH              2000 non-null int64
MEAN.CORE          2000 non-null float64
VAR.CORE           2000 non-null float64
MAX.CORE           2000 non-null int64
DENSITY            2000 non-null float64
TRANSITIVITY       2000 non-null int64
ASS.1              2000 non-null float64
ASS.2              2000 non-null float64
BET.1              2000 non-null float64
BET.2              2000 non-null float64
ASPL.1             2000 non-null float64
ASPL.2             2000 non-null float64
CC.1               2000 non-null float64
CC.2               2000 non-null float64
DEG.1              2000 non-null float64
DEG.2              2000 non-null float64
MIN.1              2000 non-null int64
MIN.2              2000 non-null int64
MAX.1              2000 non-null int64
MAX.2              2000 non-null int64
SD.1               2000 non-null float64
SD.2               2000 non-null float64
MT3.1              2000 non-null int64
MT3.2              2000 non-null int64
MT4.1              2000 non-null int64
MT4.2              2000 non-null int64
CLASS              2000 non-null object
dtypes: float64(27), int64(12), object(2)
memory usage: 640.8+ KB
```

[ ]:
```
df.isna().sum().sort_values(ascending=False)
```

[ ]:
```
CLASS              0
TRANSITIVITY       0
MAX.CORE           0
VAR.CORE           0
MEAN.CORE          0
GIRTH              0
DIAMETER           0
MEAN.DEG           0
```

```
VAR.BURT           0
MEAN.BURT          0
VAR.CLOSE.CENT     0
MEAN.CLOSE.CENT    0
MEAN.BIB.COUP      0
MEAN.CIT.COUP      0
VAR.EDGE.BET       0
MEAN.EDGE.BET      0
VAR.BET            0
MEAN.BET           0
MEAN.PATH.LEN      0
DENSITY            0
ASS.1              0
MT4.2              0
ASS.2              0
MT4.1              0
MT3.2              0
MT3.1              0
SD.2               0
SD.1               0
MAX.2              0
MAX.1              0
MIN.2              0
MIN.1              0
DEG.2              0
DEG.1              0
CC.2               0
CC.1               0
ASPL.2             0
ASPL.1             0
BET.2              0
BET.1              0
ART.POINTS         0
dtype: int64
```

```
[ ]: df.isnull().sum().sort_values(ascending=False)
```

```
[ ]: CLASS              0
     TRANSITIVITY       0
     MAX.CORE           0
     VAR.CORE           0
     MEAN.CORE          0
     GIRTH              0
     DIAMETER           0
     MEAN.DEG           0
     VAR.BURT           0
     MEAN.BURT          0
```

```
VAR.CLOSE.CENT     0
MEAN.CLOSE.CENT    0
MEAN.BIB.COUP      0
MEAN.CIT.COUP      0
VAR.EDGE.BET       0
MEAN.EDGE.BET      0
VAR.BET            0
MEAN.BET           0
MEAN.PATH.LEN      0
DENSITY            0
ASS.1              0
MT4.2              0
ASS.2              0
MT4.1              0
MT3.2              0
MT3.1              0
SD.2               0
SD.1               0
MAX.2              0
MAX.1              0
MIN.2              0
MIN.1              0
DEG.2              0
DEG.1              0
CC.2               0
CC.1               0
ASPL.2             0
ASPL.1             0
BET.2              0
BET.1              0
ART.POINTS         0
dtype: int64
```

```
[ ]: pd.options.display.float_format = "{:.3f}".format
     print("Describe:\n",df.describe().T)
```

```
Describe:
                    count         mean          std      min       25%  \
MEAN.PATH.LEN   2000.000       33.001       23.131    8.224    13.662
MEAN.BET        2000.000     6057.844     7233.527  242.821   486.731
VAR.BET         2000.000 96302031.579 210802591.309 31211.461 182463.320
MEAN.EDGE.BET   2000.000     4597.436     5676.915  170.090   323.053
VAR.EDGE.BET    2000.000 83150848.327 178688922.809 22935.150 149754.096
MEAN.CIT.COUP   2000.000        0.058        0.045    0.006     0.012
MEAN.BIB.COUP   2000.000        0.058        0.045    0.006     0.012
MEAN.CLOSE.CENT 2000.000        0.001        0.000    0.000     0.000
VAR.CLOSE.CENT  2000.000        0.000        0.000    0.000     0.000
```

```
MEAN.BURT        2000.000         0.446          0.017     0.407       0.432
VAR.BURT         2000.000         0.008          0.004     0.003       0.005
MEAN.DEG         2000.000         2.967          0.310     2.212       2.710
DIAMETER         2000.000        89.701         62.962    19.000      39.000
GIRTH            2000.000         4.000          0.000     4.000       4.000
MEAN.CORE        2000.000         2.403          0.213     1.479       2.267
VAR.CORE         2000.000         0.375          0.126     0.174       0.303
MAX.CORE         2000.000         3.000          0.000     3.000       3.000
DENSITY          2000.000         0.024          0.018     0.003       0.006
TRANSITIVITY     2000.000         0.000          0.000     0.000       0.000
ASS.1            2000.000         0.029          0.125    -0.382      -0.053
ASS.2            2000.000        -0.051          0.317    -1.000      -0.165
BET.1            2000.000        40.843         11.991    14.562      30.594
BET.2            2000.000         9.717         11.880     0.000       0.000
ASPL.1           2000.000         2.687          0.700     1.669       1.992
ASPL.2           2000.000        40.009         12.363     0.000      36.122
CC.1             2000.000         0.177          0.109     0.000       0.076
CC.2             2000.000         0.078          0.104     0.000       0.000
DEG.1            2000.000         8.534          5.671     2.537       3.362
DEG.2            2000.000         2.732          3.018     0.000       0.240
MIN.1            2000.000        26.616         20.872     1.000       3.000
MIN.2            2000.000         2.814          3.686     0.000       1.000
MAX.1            2000.000        14.790          9.881     1.000       8.000
MAX.2            2000.000        14.290         11.180     0.000       5.000
SD.1             2000.000         6.641          5.293     1.018       1.981
SD.2             2000.000         4.338          4.408     0.000       0.731
MT3.1            2000.000      1905.659       1921.029    88.000     199.000
MT3.2            2000.000       149.862        216.261     0.000       0.000
MT4.1            2000.000     21686.983      24969.108   171.000     717.000
MT4.2            2000.000       838.574       1497.315     0.000       0.000

                         50%           75%            max
MEAN.PATH.LEN         23.476        49.750        142.700
MEAN.BET            1815.108     11151.197      46273.049
VAR.BET          2726241.122 140454975.447 3571253588.880
MEAN.EDGE.BET       1233.772      8330.002      36848.632
VAR.EDGE.BET     2102285.013 119159230.744 2771242958.713
MEAN.CIT.COUP          0.040         0.103          0.152
MEAN.BIB.COUP          0.040         0.103          0.152
MEAN.CLOSE.CENT        0.000         0.001          0.002
VAR.CLOSE.CENT         0.000         0.000          0.000
MEAN.BURT              0.446         0.459          0.497
VAR.BURT               0.007         0.012          0.015
MEAN.DEG               2.951         3.241          3.620
DIAMETER              63.500       130.000        419.000
GIRTH                  4.000         4.000          4.000
MEAN.CORE              2.419         2.580          2.789
VAR.CORE               0.349         0.397          0.923
```

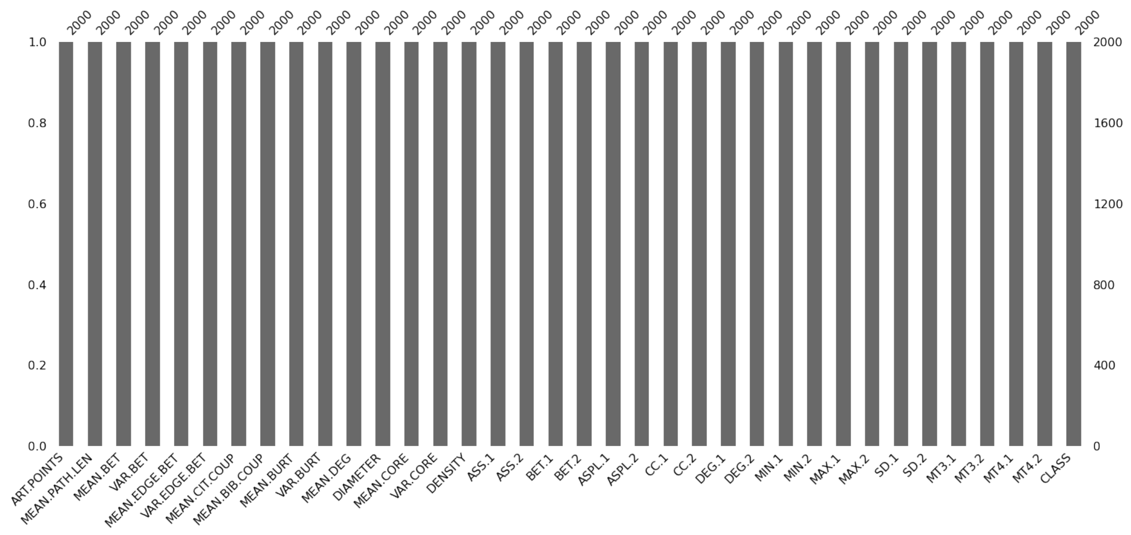
```
MAX.CORE                3.000        3.000         3.000
DENSITY                 0.017        0.042         0.059
TRANSITIVITY            0.000        0.000         0.000
ASS.1                   0.019        0.099         0.834
ASS.2                   0.000        0.000         1.000
BET.1                  37.595       49.356       128.155
BET.2                   0.549       20.543        52.127
ASPL.1                  2.505        3.278         5.497
ASPL.2                 41.623       46.343        72.374
CC.1                    0.150        0.288         0.488
CC.2                    0.000        0.155         1.000
DEG.1                   6.036       14.344        31.969
DEG.2                   1.000        5.261        19.082
MIN.1                  32.000       42.250        75.000
MIN.2                   1.000        3.000        33.000
MAX.1                  14.000       20.000        60.000
MAX.2                  14.000       21.000        60.000
SD.1                    3.943       11.475        43.699
SD.2                    1.924        7.870        41.267
MT3.1                 827.000     4001.500     12255.000
MT3.2                   9.000      291.000      2230.000
MT4.1                5285.000    48058.500    196141.000
MT4.2                   9.000     1426.750     22132.000
```

The same columns that were dropped in the RNAcon dataset.

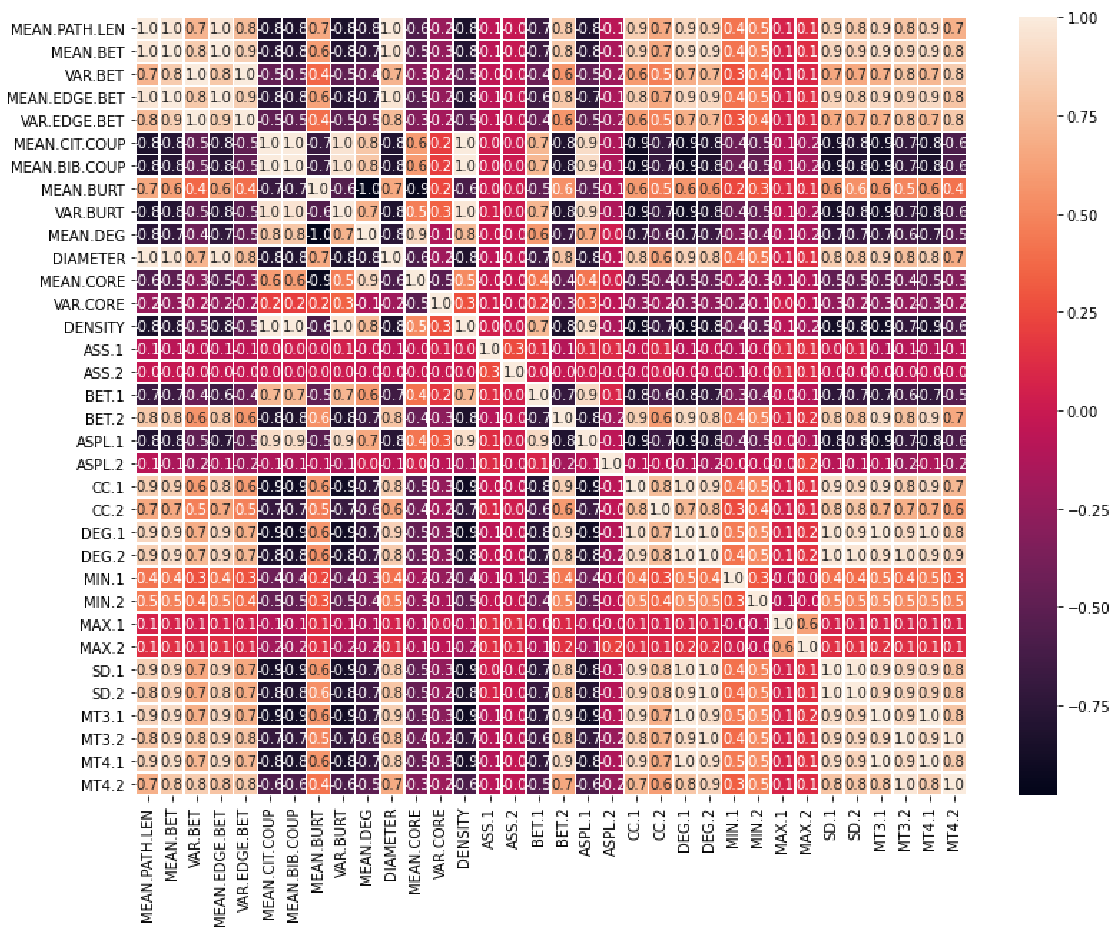
```
[ ]: col_to_drop = ["MEAN.CLOSE.CENT", "VAR.CLOSE.CENT", "GIRTH", "MAX.CORE",␣
      ↪"TRANSITIVITY"]

     df.drop(columns = col_to_drop,
                 axis = 1,
                 inplace = True)
```

```
[ ]: msno.bar(df)
```

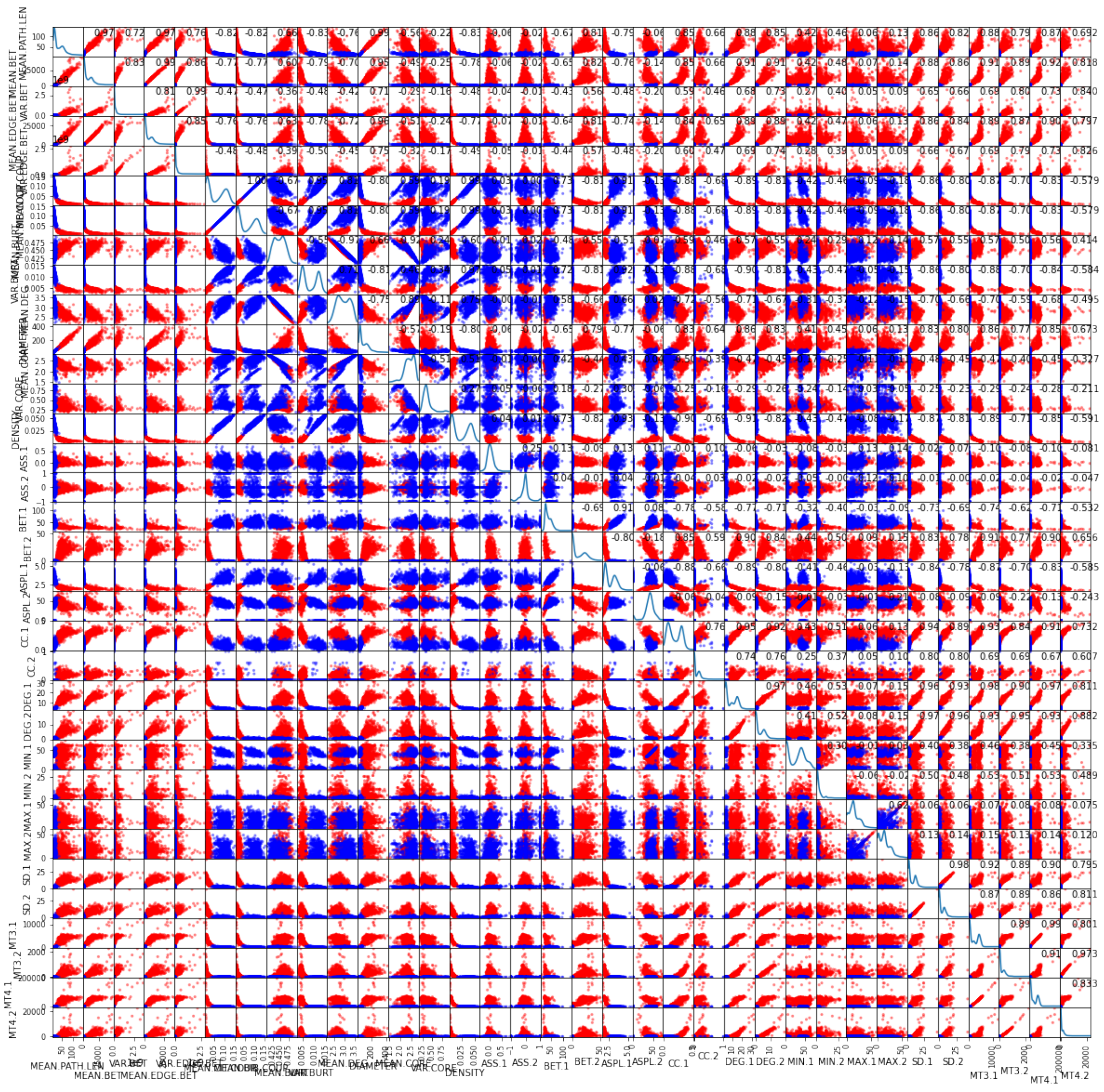
```
[ ]: <AxesSubplot:>
```



```
[ ]: f,ax = plt.subplots(figsize=(13, 10))
     sns.heatmap(df.corr(),
                 annot = True,
                 linewidths=.5,
                 fmt= '.1f',
                 ax = ax)
     plt.show()
```



```
[ ]: class_color = {"i_gpI" : "red",
                    "i_gpII" : "blue"}
     colors = [class_color[name] for name in df.CLASS]

     ax = pd.plotting.scatter_matrix(df,
                                     diagonal = "kde",
                                     color = colors,
                                     figsize=(20,20))
     corr = np.asmatrix(df.corr())
     for i, j in zip(*plt.np.triu_indices_from(ax, k=1)):
         ax[i, j].annotate("%.3f" %corr[i,j], (0.8, 0.8), xycoords='axes fraction',␣
      ↪ha='center', va='center')
     plt.show()
```



1.4 Rescale the datasets

```
[ ]: from sklearn import preprocessing
```

1.4.1 BASiNET

```
[ ]: class_intron = basinet['CLASS']
     basinet.drop(columns=["CLASS"],
                  axis = 1,
                  inplace = True)
     basinet_col_names = basinet.columns
```

[ ]:
```
min_max_scaler = preprocessing.MinMaxScaler()
basinet_minmax = min_max_scaler.fit_transform(basinet)
basinet = pd.DataFrame(basinet_minmax)
le = preprocessing.LabelEncoder()
class_intron = le.fit_transform(class_intron)
```

[ ]:
```
basinet.columns = basinet_col_names
basinet["CLASS"] = class_intron
```

[ ]:
```
pd.options.display.float_format = "{:.3f}".format
print("Describe:\n",basinet.describe().T)
```

```
Describe:
           count  mean   std   min   25%   50%   75%   max
ASS.1  2000.000 0.338 0.103 0.000 0.271 0.329 0.395 1.000
ASS.2  2000.000 0.474 0.158 0.000 0.417 0.500 0.500 1.000
BET.1  2000.000 0.231 0.106 0.000 0.141 0.203 0.306 1.000
BET.2  2000.000 0.186 0.228 0.000 0.000 0.011 0.394 1.000
ASPL.1 2000.000 0.266 0.183 0.000 0.084 0.218 0.420 1.000
ASPL.2 2000.000 0.553 0.171 0.000 0.499 0.575 0.640 1.000
CC.1   2000.000 0.363 0.224 0.000 0.157 0.308 0.591 1.000
CC.2   2000.000 0.078 0.104 0.000 0.000 0.000 0.155 1.000
DEG.1  2000.000 0.204 0.193 0.000 0.028 0.119 0.401 1.000
DEG.2  2000.000 0.143 0.158 0.000 0.013 0.052 0.276 1.000
MIN.1  2000.000 0.346 0.282 0.000 0.027 0.419 0.557 1.000
MIN.2  2000.000 0.085 0.112 0.000 0.030 0.030 0.091 1.000
MAX.1  2000.000 0.234 0.167 0.000 0.119 0.220 0.322 1.000
MAX.2  2000.000 0.238 0.186 0.000 0.083 0.233 0.350 1.000
SD.1   2000.000 0.132 0.124 0.000 0.023 0.069 0.245 1.000
SD.2   2000.000 0.105 0.107 0.000 0.018 0.047 0.191 1.000
MT3.1  2000.000 0.149 0.158 0.000 0.009 0.061 0.322 1.000
MT3.2  2000.000 0.067 0.097 0.000 0.000 0.004 0.130 1.000
MT4.1  2000.000 0.110 0.127 0.000 0.003 0.026 0.244 1.000
MT4.2  2000.000 0.038 0.068 0.000 0.000 0.000 0.064 1.000
CLASS  2000.000 0.500 0.500 0.000 0.000 0.500 1.000 1.000
```

1.4.2 RNAcon

[ ]:
```
class_intron = rnacon['CLASS']
rnacon.drop(columns=["CLASS"],
             axis = 1,
             inplace = True)
rnacon_col_names = rnacon.columns
```

[ ]:
```
min_max_scaler = preprocessing.MinMaxScaler()
rnacon_minmax = min_max_scaler.fit_transform(rnacon)
rnacon = pd.DataFrame(rnacon_minmax)
```

```
le = preprocessing.LabelEncoder()
class_intron = le.fit_transform(class_intron)
```

[ ]:
```
rnacon.columns = rnacon_col_names
rnacon["CLASS"] = class_intron
```

[ ]:
```
pd.options.display.float_format = "{:.3f}".format
print("Describe:\n",rnacon.describe().T)
```

```
Describe:
                  count  mean   std   min   25%   50%   75%   max
ART.POINTS    1996.000 0.171 0.146 0.000 0.055 0.077 0.276 1.000
MEAN.PATH.LEN 2000.000 0.184 0.172 0.000 0.040 0.113 0.309 1.000
MEAN.BET      2000.000 0.126 0.157 0.000 0.005 0.034 0.237 1.000
VAR.BET       2000.000 0.027 0.059 0.000 0.000 0.001 0.039 1.000
MEAN.EDGE.BET 2000.000 0.121 0.155 0.000 0.004 0.029 0.222 1.000
VAR.EDGE.BET  2000.000 0.030 0.064 0.000 0.000 0.001 0.043 1.000
MEAN.CIT.COUP 2000.000 0.355 0.310 0.000 0.045 0.237 0.666 1.000
MEAN.BIB.COUP 2000.000 0.355 0.310 0.000 0.045 0.237 0.666 1.000
MEAN.BURT     2000.000 0.433 0.187 0.000 0.274 0.433 0.579 1.000
VAR.BURT      2000.000 0.442 0.276 0.000 0.179 0.336 0.728 1.000
MEAN.DEG      2000.000 0.537 0.220 0.000 0.354 0.525 0.731 1.000
DIAMETER      2000.000 0.177 0.157 0.000 0.050 0.111 0.278 1.000
MEAN.CORE     2000.000 0.705 0.162 0.000 0.602 0.717 0.840 1.000
VAR.CORE      2000.000 0.267 0.169 0.000 0.171 0.233 0.297 1.000
DENSITY       2000.000 0.378 0.313 0.000 0.057 0.263 0.698 1.000
CLASS         2000.000 0.500 0.500 0.000 0.000 0.500 1.000 1.000
```

[ ]:
```
rnacon.isna().sum().sort_values(ascending=False)
```

[ ]:
```
ART.POINTS       4
CLASS            0
DENSITY          0
VAR.CORE         0
MEAN.CORE        0
DIAMETER         0
MEAN.DEG         0
VAR.BURT         0
MEAN.BURT        0
MEAN.BIB.COUP    0
MEAN.CIT.COUP    0
VAR.EDGE.BET     0
MEAN.EDGE.BET    0
VAR.BET          0
MEAN.BET         0
MEAN.PATH.LEN    0
dtype: int64
```

```
[ ]: rnacon.isnull().sum().sort_values(ascending=False)
```

```
[ ]: ART.POINTS       4
     CLASS            0
     DENSITY          0
     VAR.CORE         0
     MEAN.CORE        0
     DIAMETER         0
     MEAN.DEG         0
     VAR.BURT         0
     MEAN.BURT        0
     MEAN.BIB.COUP    0
     MEAN.CIT.COUP    0
     VAR.EDGE.BET     0
     MEAN.EDGE.BET    0
     VAR.BET          0
     MEAN.BET         0
     MEAN.PATH.LEN    0
     dtype: int64
```

```
[ ]: rnacon['ART.POINTS'] = rnacon['ART.POINTS'].fillna(0)
```

1.4.3 Full Dataset

```
[ ]: class_intron = df['CLASS']
     df.drop(columns=["CLASS"],
                  axis = 1,
                  inplace = True)
     df_col_names = df.columns
```

```
[ ]: min_max_scaler = preprocessing.MinMaxScaler()
     df_minmax = min_max_scaler.fit_transform(df)
     df = pd.DataFrame(df_minmax)
     le = preprocessing.LabelEncoder()
     class_intron = le.fit_transform(class_intron)
```

```
[ ]: df.columns = df_col_names
     df["CLASS"] = class_intron
```

```
[ ]: pd.options.display.float_format = "{:.3f}".format
     print("Describe:\n",df.describe().T)
```

```
Describe:
                   count  mean   std   min   25%   50%   75%   max
ART.POINTS    1996.000 0.171 0.146 0.000 0.055 0.077 0.276 1.000
MEAN.PATH.LEN 2000.000 0.184 0.172 0.000 0.040 0.113 0.309 1.000
MEAN.BET      2000.000 0.126 0.157 0.000 0.005 0.034 0.237 1.000
```

```
VAR.BET       2000.000 0.027 0.059 0.000 0.000 0.001 0.039 1.000
MEAN.EDGE.BET 2000.000 0.121 0.155 0.000 0.004 0.029 0.222 1.000
VAR.EDGE.BET  2000.000 0.030 0.064 0.000 0.000 0.001 0.043 1.000
MEAN.CIT.COUP 2000.000 0.355 0.310 0.000 0.045 0.237 0.666 1.000
MEAN.BIB.COUP 2000.000 0.355 0.310 0.000 0.045 0.237 0.666 1.000
MEAN.BURT     2000.000 0.433 0.187 0.000 0.274 0.433 0.579 1.000
VAR.BURT      2000.000 0.442 0.276 0.000 0.179 0.336 0.728 1.000
MEAN.DEG      2000.000 0.537 0.220 0.000 0.354 0.525 0.731 1.000
DIAMETER      2000.000 0.177 0.157 0.000 0.050 0.111 0.278 1.000
MEAN.CORE     2000.000 0.705 0.162 0.000 0.602 0.717 0.840 1.000
VAR.CORE      2000.000 0.267 0.169 0.000 0.171 0.233 0.297 1.000
DENSITY       2000.000 0.378 0.313 0.000 0.057 0.263 0.698 1.000
ASS.1         2000.000 0.338 0.103 0.000 0.271 0.329 0.395 1.000
ASS.2         2000.000 0.474 0.158 0.000 0.417 0.500 0.500 1.000
BET.1         2000.000 0.231 0.106 0.000 0.141 0.203 0.306 1.000
BET.2         2000.000 0.186 0.228 0.000 0.000 0.011 0.394 1.000
ASPL.1        2000.000 0.266 0.183 0.000 0.084 0.218 0.420 1.000
ASPL.2        2000.000 0.553 0.171 0.000 0.499 0.575 0.640 1.000
CC.1          2000.000 0.363 0.224 0.000 0.157 0.308 0.591 1.000
CC.2          2000.000 0.078 0.104 0.000 0.000 0.000 0.155 1.000
DEG.1         2000.000 0.204 0.193 0.000 0.028 0.119 0.401 1.000
DEG.2         2000.000 0.143 0.158 0.000 0.013 0.052 0.276 1.000
MIN.1         2000.000 0.346 0.282 0.000 0.027 0.419 0.557 1.000
MIN.2         2000.000 0.085 0.112 0.000 0.030 0.030 0.091 1.000
MAX.1         2000.000 0.234 0.167 0.000 0.119 0.220 0.322 1.000
MAX.2         2000.000 0.238 0.186 0.000 0.083 0.233 0.350 1.000
SD.1          2000.000 0.132 0.124 0.000 0.023 0.069 0.245 1.000
SD.2          2000.000 0.105 0.107 0.000 0.018 0.047 0.191 1.000
MT3.1         2000.000 0.149 0.158 0.000 0.009 0.061 0.322 1.000
MT3.2         2000.000 0.067 0.097 0.000 0.000 0.004 0.130 1.000
MT4.1         2000.000 0.110 0.127 0.000 0.003 0.026 0.244 1.000
MT4.2         2000.000 0.038 0.068 0.000 0.000 0.000 0.064 1.000
CLASS         2000.000 0.500 0.500 0.000 0.000 0.500 1.000 1.000
```

```
[ ]: df.isna().sum().sort_values(ascending=False)
```

```
[ ]: ART.POINTS       4
     MT4.2            0
     ASS.1            0
     DENSITY          0
     VAR.CORE         0
     MEAN.CORE        0
     DIAMETER         0
     MEAN.DEG         0
     VAR.BURT         0
     MEAN.BURT        0
     MEAN.BIB.COUP    0
     MEAN.CIT.COUP    0
     VAR.EDGE.BET     0
     MEAN.EDGE.BET    0
     VAR.BET          0
     MEAN.BET         0
     MEAN.PATH.LEN    0
     ASS.2            0
     CLASS            0
     BET.2            0
     MAX.1            0
     MT4.1            0
     MT3.2            0
     MT3.1            0
     SD.2             0
     SD.1             0
     MAX.2            0
     MIN.2            0
     ASPL.1           0
     MIN.1            0
     DEG.2            0
     DEG.1            0
     CC.2             0
     CC.1             0
     ASPL.2           0
     BET.1            0
     dtype: int64
```

[ ]:
```
df.isnull().sum().sort_values(ascending=False)
```

[ ]:
```
ART.POINTS       4
MT4.2            0
ASS.1            0
DENSITY          0
VAR.CORE         0
MEAN.CORE        0
DIAMETER         0
MEAN.DEG         0
VAR.BURT         0
MEAN.BURT        0
MEAN.BIB.COUP    0
MEAN.CIT.COUP    0
VAR.EDGE.BET     0
MEAN.EDGE.BET    0
VAR.BET          0
MEAN.BET         0
MEAN.PATH.LEN    0
ASS.2            0
```

```
CLASS              0
BET.2              0
MAX.1              0
MT4.1              0
MT3.2              0
MT3.1              0
SD.2               0
SD.1               0
MAX.2              0
MIN.2              0
ASPL.1             0
MIN.1              0
DEG.2              0
DEG.1              0
CC.2               0
CC.1               0
ASPL.2             0
BET.1              0
dtype: int64
```

Filling the NaN values.

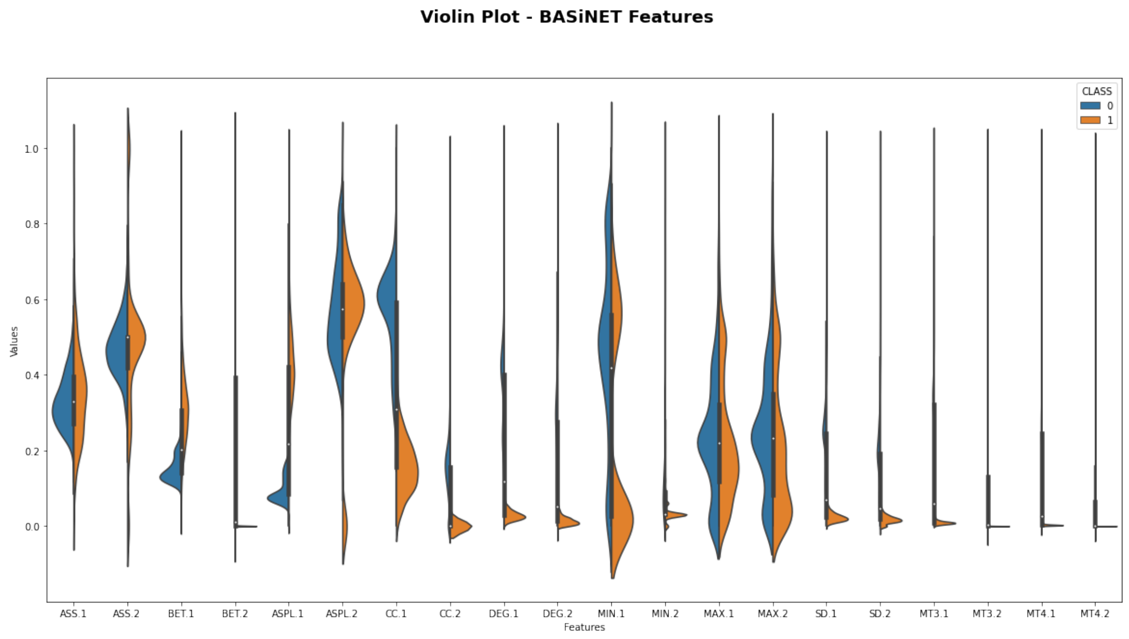
```
[ ]: df["ART.POINTS"] = df["ART.POINTS"].fillna(0)
```

1.5 Data Visualization

Other visualizations can be done.
This is **only** an idea for data visualization with violin plots.

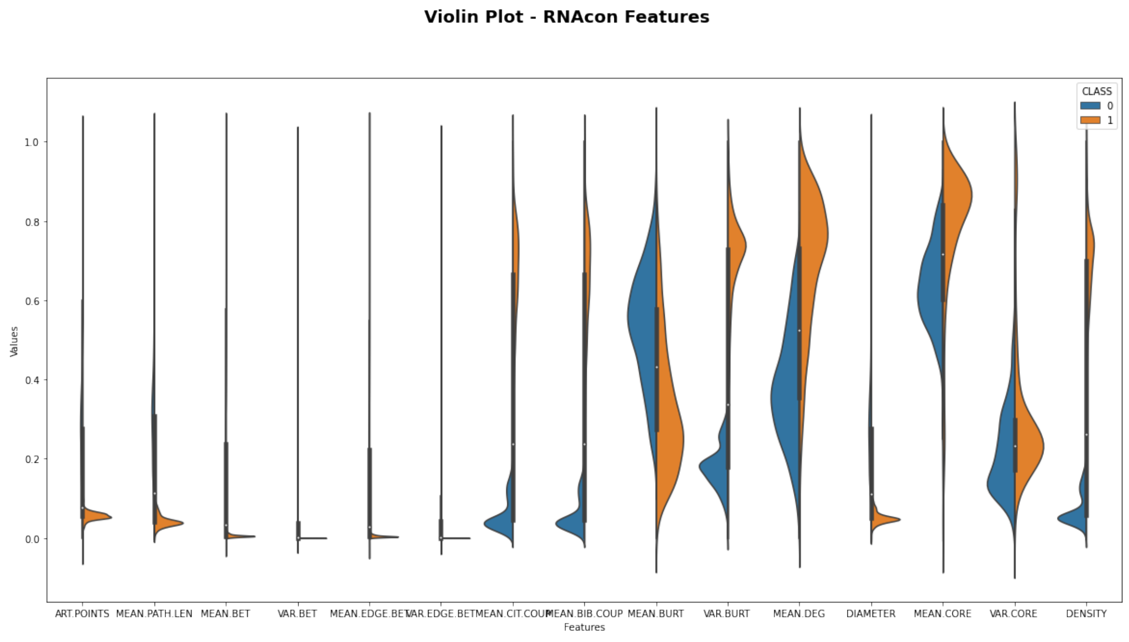
1.5.1 BASiNET

```
[ ]: f, ax = plt.subplots(figsize=(20, 10))
     data = pd.melt(basinet,
                    id_vars="CLASS",
                    var_name="Features",
                    value_name="Values")
     sns.violinplot(x="Features",
                    y = "Values",
                    data=data,
                    split=True,
                    hue="CLASS")
     f.suptitle('Violin Plot - BASiNET Features', fontsize=18, fontweight='bold')
     plt.show()
```



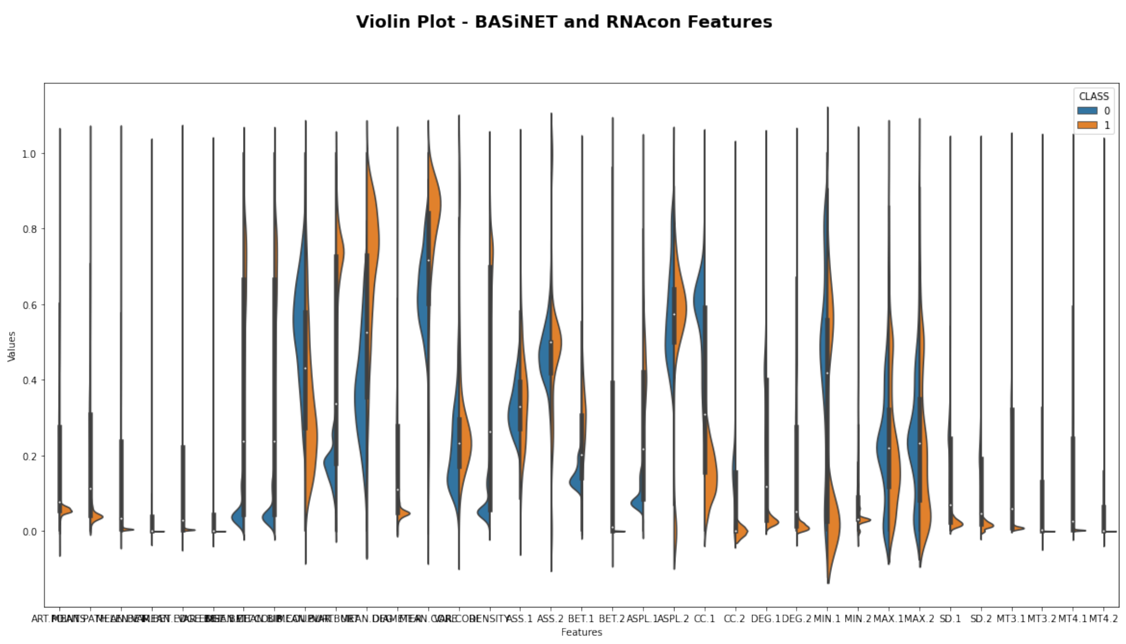
1.5.2 RNAcon

```
[ ]: f, ax = plt.subplots(figsize=(20, 10))
data = pd.melt(rnacon,
               id_vars="CLASS",
               var_name="Features",
               value_name="Values")
sns.violinplot(x="Features",
               y = "Values",
               data=data,
               split=True,
               hue="CLASS")
f.suptitle('Violin Plot - RNAcon Features', fontsize=18, fontweight='bold')
plt.show()
```



1.5.3 Full Dataset

[ ]:
```
f, ax = plt.subplots(figsize=(20, 10))
data = pd.melt(df,
               id_vars="CLASS",
               var_name="Features",
               value_name="Values")
sns.violinplot(x="Features",
               y = "Values",
               data=data,
               split=True,
               hue="CLASS")
f.suptitle('Violin Plot - BASiNET and RNAcon Features', fontsize=18,␣
 ↪fontweight='bold')
plt.show()
```



1.6 Classification - All Features case

```
[ ]: from sklearn.ensemble import RandomForestClassifier
     from sklearn.model_selection import StratifiedKFold, RepeatedStratifiedKFold,␣
      ↪cross_validate, train_test_split
     from sklearn.metrics import classification_report, accuracy_score,␣
      ↪confusion_matrix, matthews_corrcoef
     from sklearn.metrics import roc_curve, auc, roc_auc_score
     import math
```

1.6.1 BASiNET

```
[ ]: X_basinet = basinet.drop(columns = 'CLASS',
                               axis = 1)
     Y_basinet = basinet['CLASS']
     X_basinet_train, X_basinet_test, Y_basinet_train, Y_basinet_test =␣
      ↪train_test_split(X_basinet,
                                                                             ␣
      ↪    Y_basinet,
                                                                             ␣
      ↪    test_size = 0.2,
                                                                             ␣
      ↪    stratify = Y_basinet)
```

```
[ ]: RF_basinet = RandomForestClassifier(n_estimators = 100)
     RF_basinet.fit(X_basinet_train,Y_basinet_train)
     Y_basinet_pred = RF_basinet.predict(X_basinet_test)
     Y_basinet_pred_proba = RF_basinet.predict_proba(X_basinet_test)[::,1]
```

```
[ ]: n_splits = 10
     n_repeats = 10
     seed = 7
     cv = RepeatedStratifiedKFold(n_splits=n_splits,
                                  n_repeats=n_repeats,
                                  random_state=seed)
```

```
[ ]: n_scores_basinet = cross_validate(RF_basinet,
                                       X_basinet,
                                       Y_basinet,
                                       scoring = 'accuracy',
                                       cv = cv,
                                       n_jobs=-1,
                                       error_score = 'raise')
     print("Mean:", np.mean(n_scores_basinet['test_score']))
```

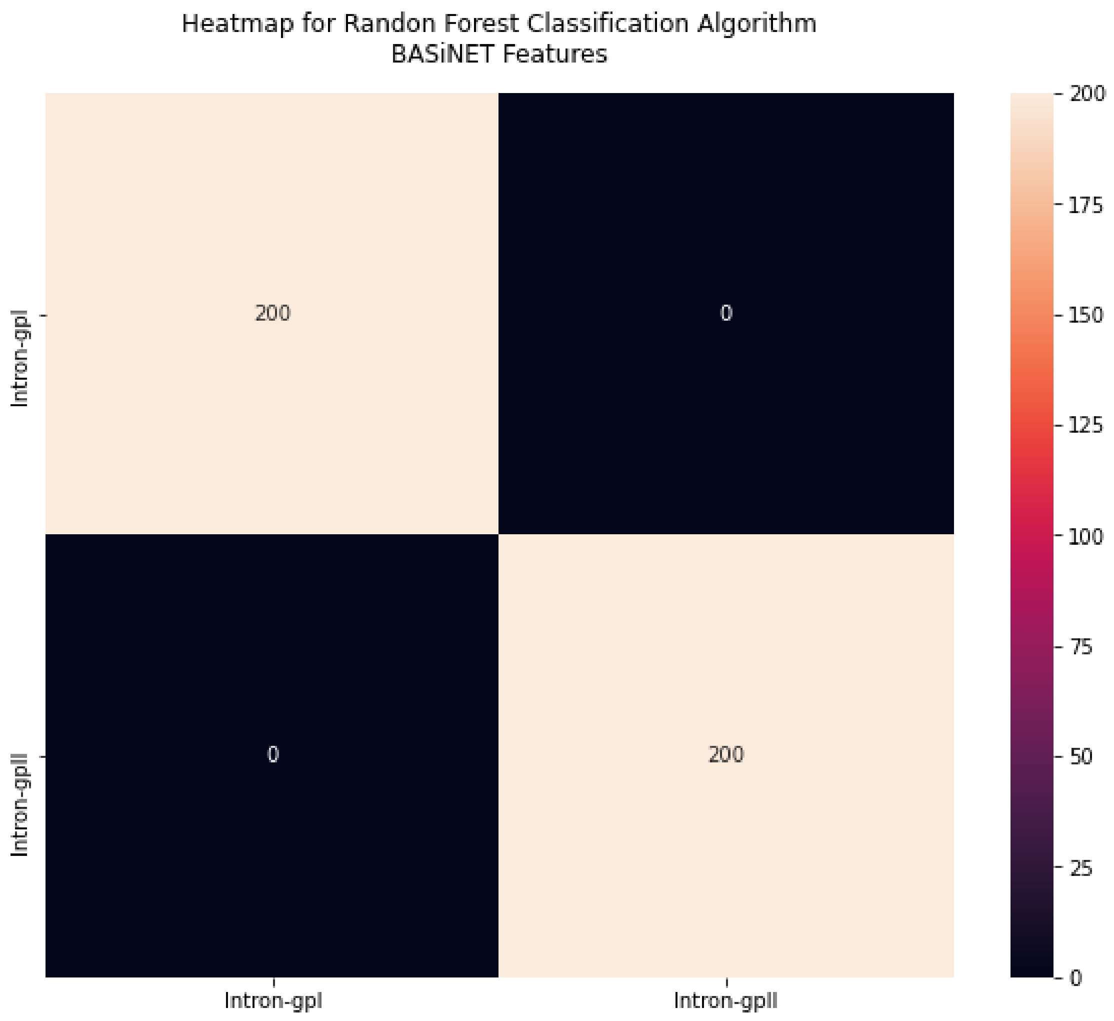
```
Mean: 0.9997000000000001
```

1.7 Metrics

```
[ ]: cm_basinet = confusion_matrix(y_true = Y_basinet_test,
                                   y_pred = Y_basinet_pred)
```

```
[ ]: plt.figure(figsize = (10,8))
     ax = plt.axes()
     x_axis_labels = ['Intron-gpI', 'Intron-gpII'] # labels for x-axis
     y_axis_labels = ['Intron-gpI', 'Intron-gpII'] # labels for y-axis
     sns.heatmap(cm_basinet,
                 vmin=0,
                 vmax=200,
                 annot=True,
                 fmt="d",
                 ax = ax,
                 xticklabels=x_axis_labels,
                 yticklabels=y_axis_labels)
     ax.set_title('Heatmap for Randon Forest Classification Algorithm\nBASiNET␣
      ↪Features',pad=15)
```

```
[ ]: Text(0.5, 1.0, 'Heatmap for Randon Forest Classification Algorithm\nBASiNET
     Features')
```



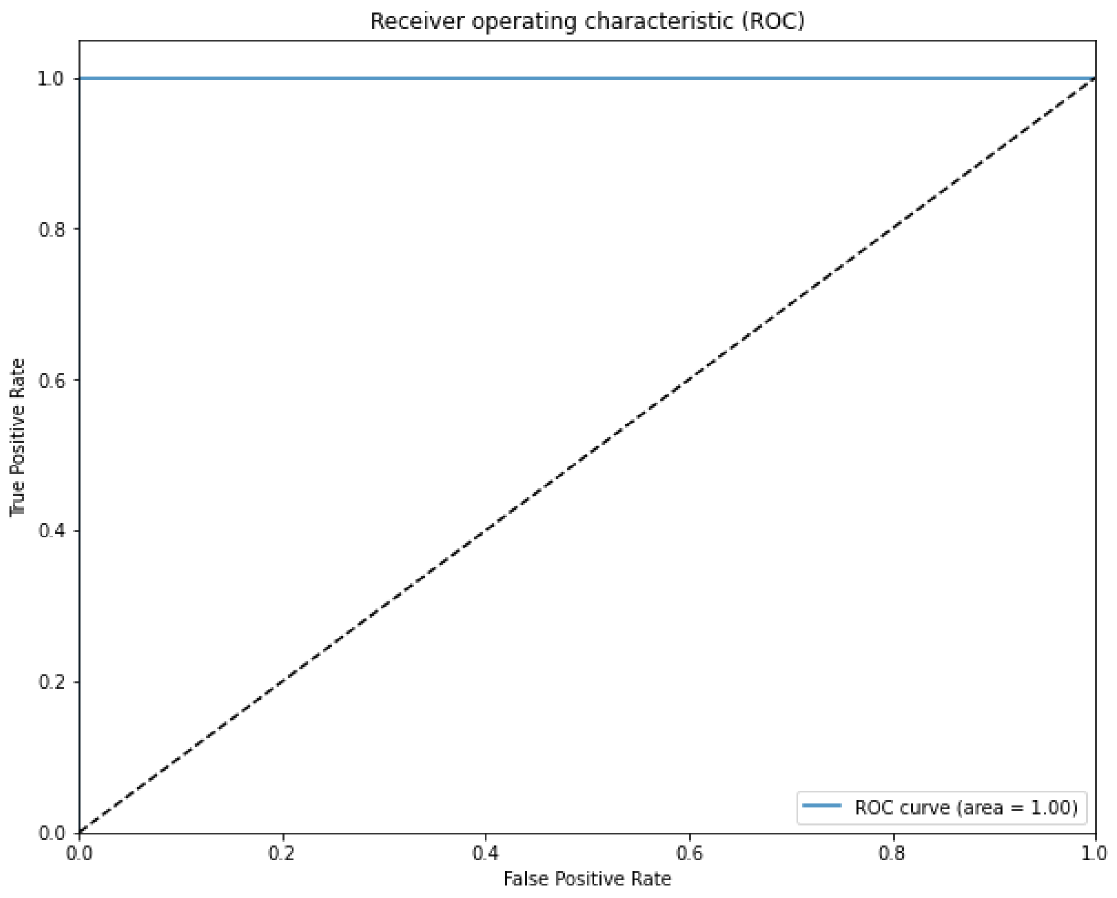
```
[ ]: print(classification_report(Y_basinet_test, Y_basinet_pred,␣
      ↪target_names=['Intron-gpI','Intron-gpII']))
     print("Accuracy:",accuracy_score(Y_basinet_test, Y_basinet_pred))
     print("Matthews correlation coefficient (MCC):
      ↪",matthews_corrcoef(Y_basinet_test,Y_basinet_pred))
```

```
              precision    recall  f1-score   support

  Intron-gpI       1.00      1.00      1.00       200
 Intron-gpII       1.00      1.00      1.00       200

    accuracy                           1.00       400
   macro avg       1.00      1.00      1.00       400
weighted avg       1.00      1.00      1.00       400

Accuracy: 1.0
```

```
Matthews correlation coefficient (MCC): 1.0
```

```
[ ]: fpr_RF_basinet, tpr_RF_basinet, _ = roc_curve(Y_basinet_test,␣
      ↪Y_basinet_pred_proba)
     auc_RF_basinet = auc(fpr_RF_basinet, tpr_RF_basinet)
```

```
[ ]: plt.figure(figsize = (10,8))
     plt.plot(fpr_RF_basinet, tpr_RF_basinet, label='ROC curve (area = %0.2f)' %␣
      ↪auc_RF_basinet)
     plt.plot([0, 1], [0, 1], 'k--')
     plt.xlim([0.0, 1.0])
     plt.ylim([0.0, 1.05])
     plt.xlabel('False Positive Rate')
     plt.ylabel('True Positive Rate')
     plt.title('Receiver operating characteristic (ROC)')
     plt.legend(loc="lower right")
     plt.show()
```



[ ]:
```python
FP = cm_basinet.sum(axis=0) - np.diag(cm_basinet)
FN = cm_basinet.sum(axis=1) - np.diag(cm_basinet)
TP = np.diag(cm_basinet)
TN = cm_basinet.sum() - (FP + FN + TP)
FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)

# Sensibility, recall or true positive rate
TPR = TP/(TP+FN)
m_TPR = np.mean(TPR)
print("Recall:\t\t", m_TPR)
# Specificity or true negative rate
TNR = TN/(TN+FP)
m_TNR = np.mean(TNR)
print("True Negative:\t", m_TNR)
# Precision
PPV = TP/(TP+FP)
m_PPV = np.mean(PPV)
print("Precision:\t", m_PPV)
# Negative pred
NPV = TN/(TN+FN)
m_NPV = np.mean(NPV)
print("Pred. Negative:\t", m_NPV)
# False positive
FPR = FP/(FP+TN)
m_FPR = np.mean(FPR)
print("False Positive: ", m_FPR)
# False negative
FNR = FN/(TP+FN)
m_FNR = np.mean(FNR)
print("False Negative:\t", m_FNR)
# False discovery
FDR = FP/(TP+FP)
m_FDR = np.mean(FDR)
print("F Discovery:\t", m_FDR)
# Accuracy for each class
ACC = (TP+TN)/(TP+FP+FN+TN)
m_ACC = np.mean(ACC)
print("Accuracy:\t", m_ACC)

x = (m_TPR + m_FPR) * (m_TPR + m_FNR) * (m_TNR + m_FPR) * (m_TNR + m_FNR)
mcc = ((m_TPR * m_TNR) - (m_FPR * m_FNR)) / math.sqrt(x)

f1 = 2 * ((m_PPV * m_TPR) / (m_PPV + m_TPR))
print("F1:\t\t", f1)
print("MCC:\t\t", mcc)
```

```
Recall:          1.0
True Negative:   1.0
Precision:       1.0
Pred. Negative:  1.0
False Positive:  0.0
False Negative:  0.0
F Discovery:     0.0
Accuracy:        1.0
F1:              1.0
MCC:             1.0
```

1.7.1 RNAcon

```
[ ]: X_rnacon = rnacon.drop(columns = 'CLASS',
                            axis = 1)
     Y_rnacon = rnacon['CLASS']
     X_rnacon_train, X_rnacon_test, Y_rnacon_train, Y_rnacon_test =␣
      ↪train_test_split(X_rnacon,
                                                                               ␣
      ↪Y_rnacon,
                                                                               ␣
      ↪test_size = 0.2,
                                                                               ␣
      ↪stratify = Y_rnacon)
```

```
[ ]: RF_rnacon = RandomForestClassifier(n_estimators = 100)
     RF_rnacon.fit(X_rnacon_train,Y_rnacon_train)
     Y_rnacon_pred = RF_rnacon.predict(X_rnacon_test)
     Y_rnacon_pred_proba = RF_rnacon.predict_proba(X_rnacon_test)[::,1]
```

```
[ ]: n_splits = 10
     n_repeats = 10
     seed = 7
     cv = RepeatedStratifiedKFold(n_splits=n_splits,
                                  n_repeats=n_repeats,
                                  random_state=seed)
```

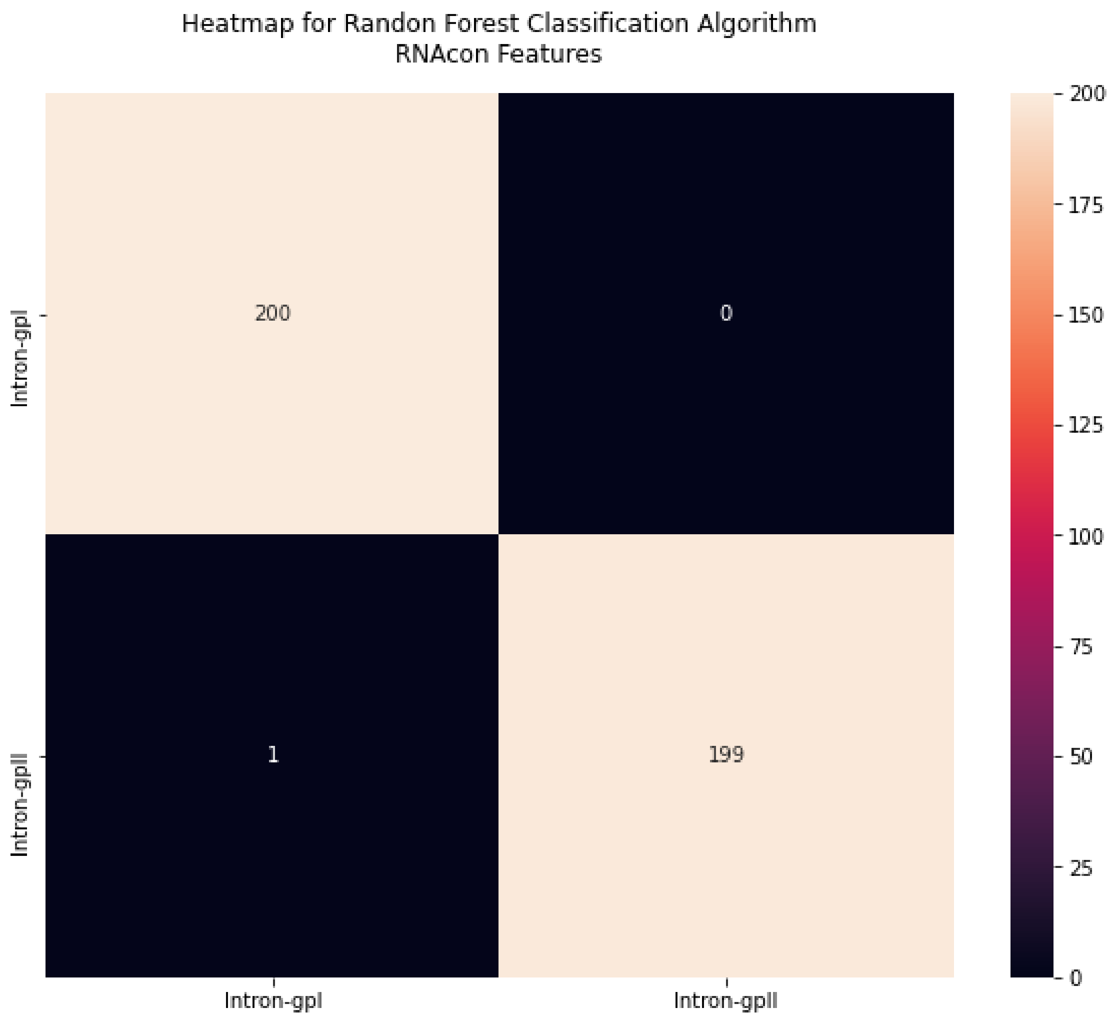
```
[ ]: n_scores_rnacon = cross_validate(RF_rnacon,
                                      X_rnacon,
                                      Y_rnacon,
                                      scoring = 'accuracy',
                                      cv = cv,
                                      n_jobs=-1,
                                      error_score = 'raise')
     print("Mean:", np.mean(n_scores_rnacon['test_score']))
```

```
Mean: 0.9979500000000001
```

```
[ ]: cm_rnacon = confusion_matrix(y_true = Y_rnacon_test,
                                  y_pred = Y_rnacon_pred)
```

```
[ ]: plt.figure(figsize = (10,8))
     ax = plt.axes()
     x_axis_labels = ['Intron-gpI', 'Intron-gpII'] # labels for x-axis
     y_axis_labels = ['Intron-gpI', 'Intron-gpII'] # labels for y-axis
     sns.heatmap(cm_rnacon,
                 vmin=0,
                 vmax=200,
                 annot=True,
                 fmt="d",
                 ax = ax,
                 xticklabels=x_axis_labels,
                 yticklabels=y_axis_labels)
     ax.set_title('Heatmap for Randon Forest Classification Algorithm\nRNAcon␣
      ↪Features',pad=15)
```

```
[ ]: Text(0.5, 1.0, 'Heatmap for Randon Forest Classification Algorithm\nRNAcon
     Features')
```



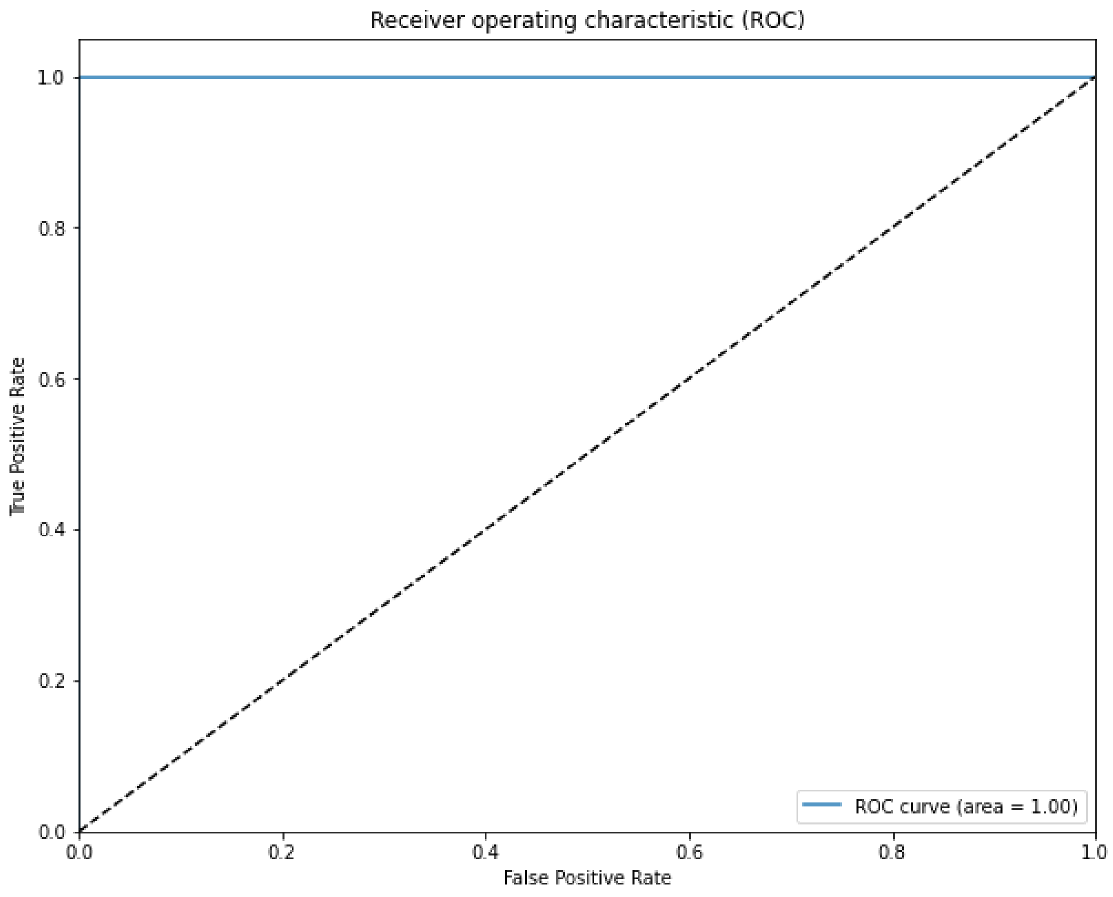
```
[ ]: print(classification_report(Y_rnacon_test, Y_rnacon_pred,␣
      ↪target_names=['Intron-gpI','Intron-gpII']))
     print("Accuracy:",accuracy_score(Y_rnacon_test, Y_rnacon_pred))
     print("Matthews correlation coefficient (MCC):
      ↪",matthews_corrcoef(Y_rnacon_test,Y_rnacon_pred))
```

```
              precision    recall  f1-score   support

   Intron-gpI       1.00      1.00      1.00       200
  Intron-gpII       1.00      0.99      1.00       200

     accuracy                           1.00       400
    macro avg       1.00      1.00      1.00       400
 weighted avg       1.00      1.00      1.00       400

Accuracy: 0.9975
```

```
Matthews correlation coefficient (MCC): 0.9950124377332079
```

```
[ ]: fpr_RF_rnacon, tpr_RF_rnacon, _ = roc_curve(Y_rnacon_test, Y_rnacon_pred_proba)
     auc_RF_rnacon = auc(fpr_RF_rnacon, tpr_RF_rnacon)
```

```
[ ]: plt.figure(figsize = (10,8))
     plt.plot(fpr_RF_rnacon, tpr_RF_rnacon, label='ROC curve (area = %0.2f)' %␣
      ↪auc_RF_rnacon)
     plt.plot([0, 1], [0, 1], 'k--')
     plt.xlim([0.0, 1.0])
     plt.ylim([0.0, 1.05])
     plt.xlabel('False Positive Rate')
     plt.ylabel('True Positive Rate')
     plt.title('Receiver operating characteristic (ROC)')
     plt.legend(loc="lower right")
     plt.show()
```



```
[ ]: FP = cm_rnacon.sum(axis=0) - np.diag(cm_rnacon)
     FN = cm_rnacon.sum(axis=1) - np.diag(cm_rnacon)
```

```python
TP = np.diag(cm_rnacon)
TN = cm_rnacon.sum() - (FP + FN + TP)
FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)

# Sensibility, recall or true positive rate
TPR = TP/(TP+FN)
m_TPR = np.mean(TPR)
print("Recall:\t\t", m_TPR)
# Specificity or true negative rate
TNR = TN/(TN+FP)
m_TNR = np.mean(TNR)
print("True Negative:\t", m_TNR)
# Precision
PPV = TP/(TP+FP)
m_PPV = np.mean(PPV)
print("Precision:\t", m_PPV)
# Negative pred
NPV = TN/(TN+FN)
m_NPV = np.mean(NPV)
print("Pred. Negative:\t", m_NPV)
# False positive
FPR = FP/(FP+TN)
m_FPR = np.mean(FPR)
print("False Positive: ", m_FPR)
# False negative
FNR = FN/(TP+FN)
m_FNR = np.mean(FNR)
print("False Negative:\t", m_FNR)
# False discovery
FDR = FP/(TP+FP)
m_FDR = np.mean(FDR)
print("F Discovery:\t", m_FDR)
# Accuracy for each class
ACC = (TP+TN)/(TP+FP+FN+TN)
m_ACC = np.mean(ACC)
print("Accuracy:\t", m_ACC)

x = (m_TPR + m_FPR) * (m_TPR + m_FNR) * (m_TNR + m_FPR) * (m_TNR + m_FNR)
mcc = ((m_TPR * m_TNR) - (m_FPR * m_FNR)) / math.sqrt(x)

f1 = 2 * ((m_PPV * m_TPR) / (m_PPV + m_TPR))
print("F1:\t\t", f1)
print("MCC:\t\t", mcc)
```

```
Recall:           0.9975
True Negative:    0.9975
Precision:        0.9975124378109452
Pred. Negative:   0.9975124378109452
False Positive:   0.0025
False Negative:   0.0025
F Discovery:      0.0024875621890547263
Accuracy:         0.9975
F1:               0.9975062188667011
MCC:              0.9950000000000001
```

1.7.2 Full Dataset

[ ]:
```
X_df = df.drop(columns = 'CLASS',
                        axis = 1)
Y_df = df['CLASS']
X_df_train, X_df_test, Y_df_train, Y_df_test = train_test_split(X_df,
                                                                Y_df,
                                                                test_size = 0.2,
                                                                stratify = Y_df)
```

[ ]:
```
RF_df = RandomForestClassifier(n_estimators = 100)
RF_df.fit(X_df_train,Y_df_train)
Y_df_pred = RF_df.predict(X_df_test)
Y_df_pred_proba = RF_df.predict_proba(X_df_test)[::,1]
```

[ ]:
```
n_splits = 10
n_repeats = 10
seed = 7
cv = RepeatedStratifiedKFold(n_splits=n_splits,
                             n_repeats=n_repeats,
                             random_state=seed)
```

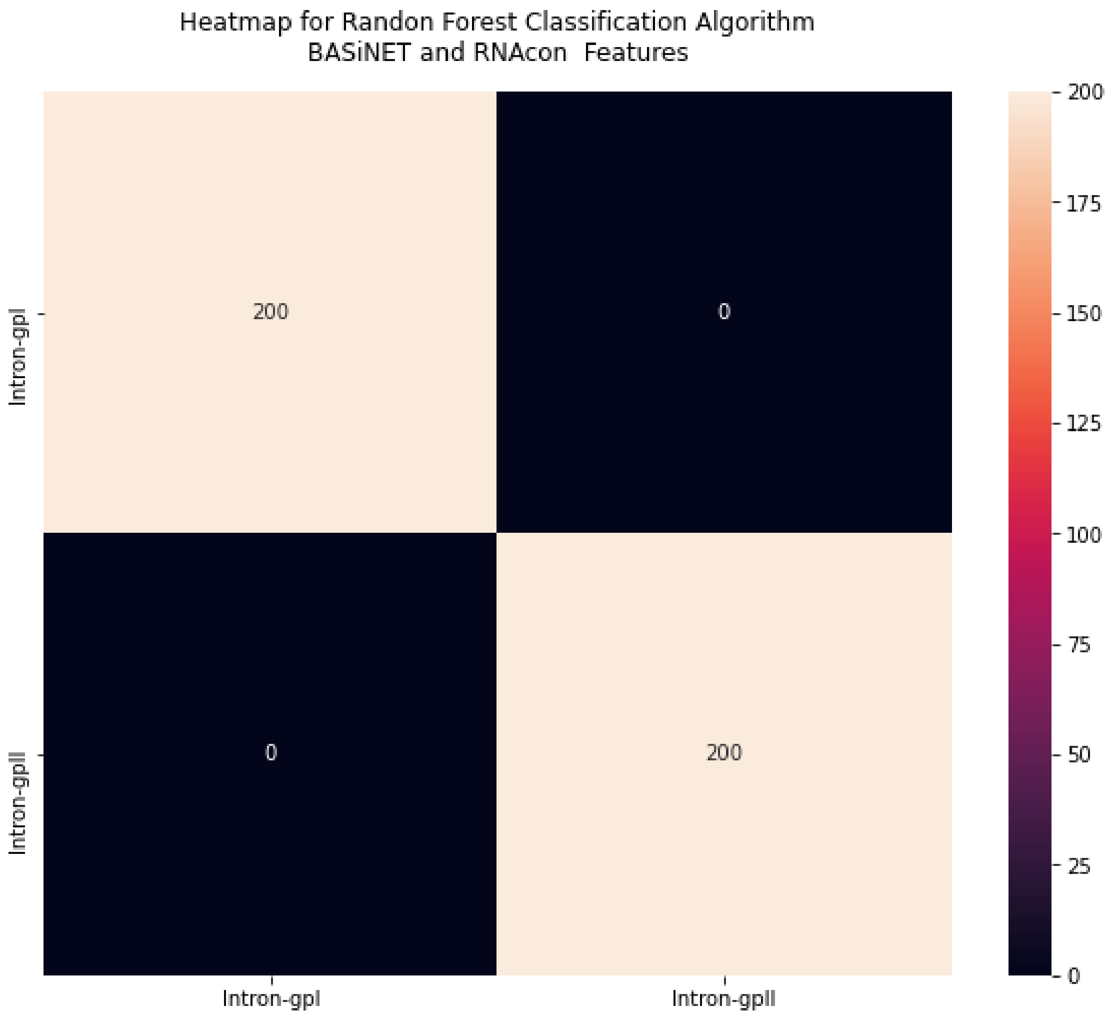
[ ]:
```
n_scores_df = cross_validate(RF_df,
                             X_df,
                             Y_df,
                             scoring = 'accuracy',
                             cv = cv,
                             n_jobs=-1,
                             error_score = 'raise')
print("Mean:", np.mean(n_scores_df['test_score']))
```

```
Mean: 0.9994
```

[ ]:
```
cm_df = confusion_matrix(y_true = Y_df_test,
                         y_pred = Y_df_pred)
```

```
[ ]: plt.figure(figsize = (10,8))
     ax = plt.axes()
     x_axis_labels = ['Intron-gpI', 'Intron-gpII'] # labels for x-axis
     y_axis_labels = ['Intron-gpI', 'Intron-gpII'] # labels for y-axis
     sns.heatmap(cm_df,
                 vmin=0,
                 vmax=200,
                 annot=True,
                 fmt="d",
                 ax = ax,
                 xticklabels=x_axis_labels,
                 yticklabels=y_axis_labels)
     ax.set_title('Heatmap for Randon Forest Classification Algorithm\nBASiNET and␣
      ↪RNAcon  Features',pad=15)
```

```
[ ]: Text(0.5, 1.0, 'Heatmap for Randon Forest Classification Algorithm\nBASiNET and
     RNAcon  Features')
```

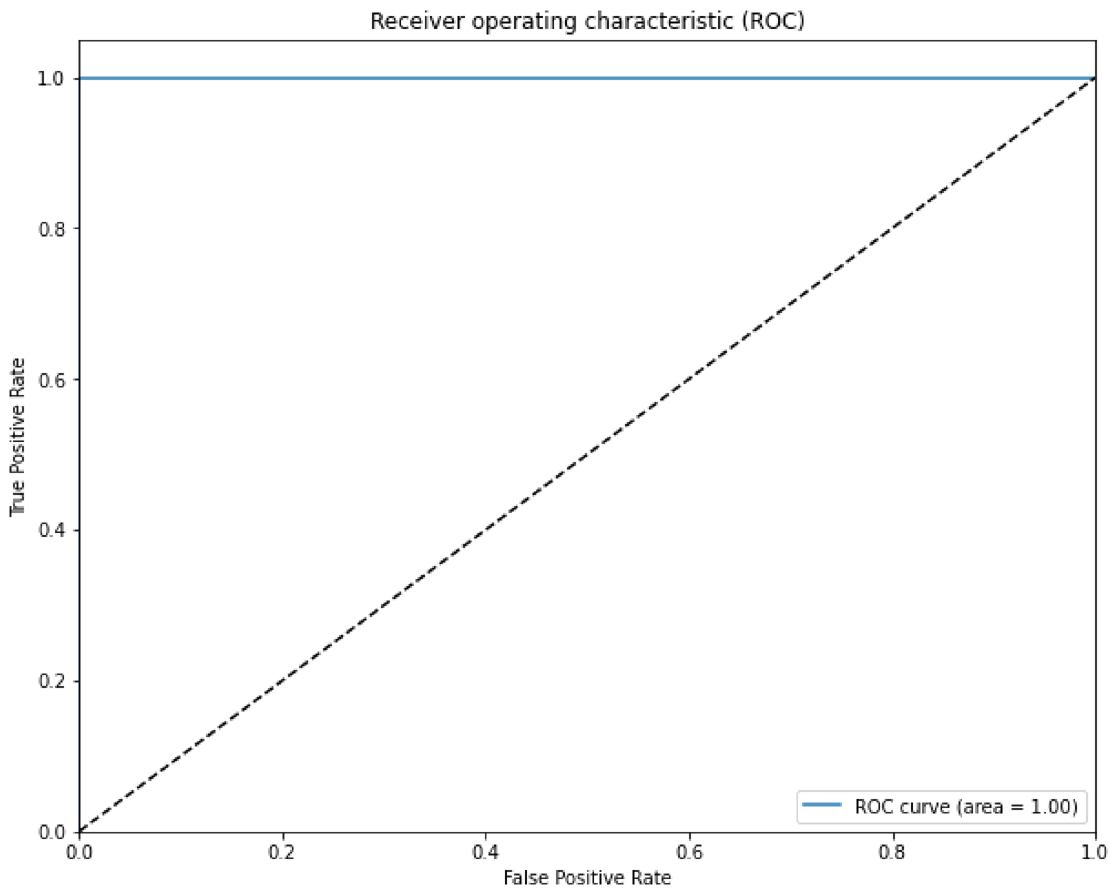


[ ]:
```
print(classification_report(Y_df_test, Y_df_pred,␣
 ↪target_names=['Intron-gpI','Intron-gpII']))
print("Accuracy:",accuracy_score(Y_df_test, Y_df_pred))
print("Matthews correlation coefficient (MCC):
 ↪",matthews_corrcoef(Y_df_test,Y_df_pred))
```

```
              precision    recall  f1-score   support

   Intron-gpI       1.00      1.00      1.00       200
  Intron-gpII       1.00      1.00      1.00       200

     accuracy                           1.00       400
    macro avg       1.00      1.00      1.00       400
 weighted avg       1.00      1.00      1.00       400

Accuracy: 1.0
Matthews correlation coefficient (MCC): 1.0
```

[ ]:
```
fpr_RF_df, tpr_RF_df, _ = roc_curve(Y_df_test, Y_df_pred_proba)
auc_RF_df = auc(fpr_RF_df, tpr_RF_df)
```

[ ]:
```
plt.figure(figsize = (10,8))
plt.plot(fpr_RF_df, tpr_RF_df, label='ROC curve (area = %0.2f)' % auc_RF_df)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```



```
[ ]: FP = cm_df.sum(axis=0) - np.diag(cm_df)
     FN = cm_df.sum(axis=1) - np.diag(cm_df)
     TP = np.diag(cm_df)
     TN = cm_df.sum() - (FP + FN + TP)
     FP = FP.astype(float)
     FN = FN.astype(float)
     TP = TP.astype(float)
     TN = TN.astype(float)

     # Sensibility, recall or true positive rate
     TPR = TP/(TP+FN)
     m_TPR = np.mean(TPR)
     print("Recall:\t\t", m_TPR)
     # Specificity or true negative rate
     TNR = TN/(TN+FP)
     m_TNR = np.mean(TNR)
     print("True Negative:\t", m_TNR)
     # Precision
     PPV = TP/(TP+FP)
```

```
m_PPV = np.mean(PPV)
print("Precision:\t", m_PPV)
# Negative pred
NPV = TN/(TN+FN)
m_NPV = np.mean(NPV)
print("Pred. Negative:\t", m_NPV)
# False positive
FPR = FP/(FP+TN)
m_FPR = np.mean(FPR)
print("False Positive: ", m_FPR)
# False negative
FNR = FN/(TP+FN)
m_FNR = np.mean(FNR)
print("False Negative:\t", m_FNR)
# False discovery
FDR = FP/(TP+FP)
m_FDR = np.mean(FDR)
print("F Discovery:\t", m_FDR)
# Accuracy for each class
ACC = (TP+TN)/(TP+FP+FN+TN)
m_ACC = np.mean(ACC)
print("Accuracy:\t", m_ACC)

x = (m_TPR + m_FPR) * (m_TPR + m_FNR) * (m_TNR + m_FPR) * (m_TNR + m_FNR)
mcc = ((m_TPR * m_TNR) - (m_FPR * m_FNR)) / math.sqrt(x)

f1 = 2 * ((m_PPV * m_TPR) / (m_PPV + m_TPR))
print("F1:\t\t", f1)
print("MCC:\t\t", mcc)
```

```
Recall:          1.0
True Negative:   1.0
Precision:       1.0
Pred. Negative:  1.0
False Positive:  0.0
False Negative:  0.0
F Discovery:     0.0
Accuracy:        1.0
F1:              1.0
MCC:             1.0
```

1.8 Feature Selection

1.9 Using SFS - Wrapper approach

Sequential feature selection (SFS) algorithms are a family of greedy search algorithms that are used to reduce an initial d-dimensional feature space to a k-dimensional feature subspace where $k < d$.

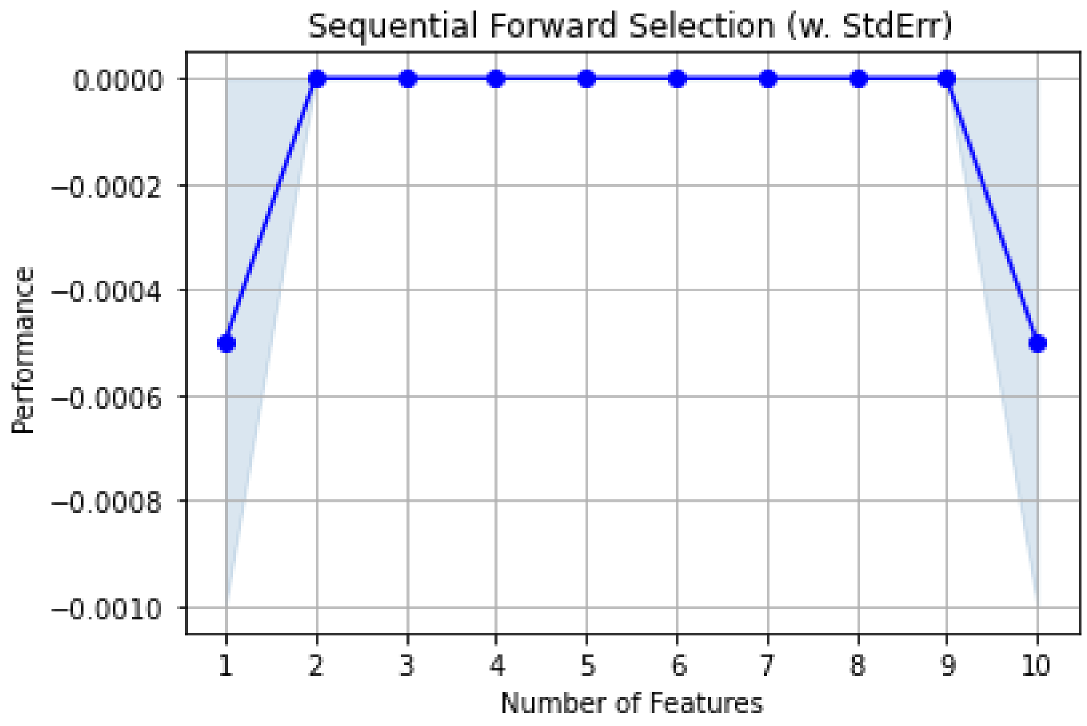
The motivation behind feature selection algorithms is to automatically select a subset of features that is most relevant to the problem.
Other wrapper approaches can be applied. The SFS approach is the simplest approach.

1.9.1 BASiNET

```
[ ]: from mlxtend.feature_selection import SequentialFeatureSelector as SFS
     from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
```

```
[ ]: sfs_basinet = SFS(RF_basinet,
                       k_features=10, # half of the features
                       forward=True,
                       floating=False,
                       scoring='neg_mean_squared_error',
                       cv=10,
                       n_jobs=-1)
```

```
[ ]: sfs_basinet = sfs_basinet.fit(X_basinet, Y_basinet)
     fig = plot_sfs(sfs_basinet.get_metric_dict(), kind='std_err')
     plt.title('Sequential Forward Selection (w. StdErr)')
     plt.grid()
     plt.show()
```



```
[ ]: sfs_basinet_metrics = pd.DataFrame.from_dict(sfs_basinet.get_metric_dict()).T
     # Two features have better metrics.
     sfs_basinet_features = list(sfs_metrics["feature_names"][2])
     print("Features Selected:", sfs_basinet_features)
```

```
Features Selected: ['ASS.2', 'DEG.1']
```

Random Forest with the selected features.

```
[ ]: X_basinet_sfs = basinet[sfs_features]
     Y_basinet_sfs = basinet['CLASS']
     X_basinet_sfs_train, X_basinet_sfs_test, Y_basinet_sfs_train,␣
      ↪Y_basinet_sfs_test = train_test_split(X_basinet_sfs,
                                                                                ␣
      ↪                    Y_basinet_sfs,
                                                                                ␣
      ↪                    test_size = 0.2,
                                                                                ␣
      ↪                    stratify = Y_basinet_sfs)
```

```
[ ]: RF_basinet_sfs = RandomForestClassifier(n_estimators = 100)
     RF_basinet_sfs.fit(X_basinet_sfs_train,Y_basinet_sfs_train)
     Y_basinet_sfs_pred = RF_basinet_sfs.predict(X_basinet_sfs_test)
     Y_basinet_sfs_pred_proba = RF_basinet_sfs.predict_proba(X_basinet_sfs_test)[::
      ↪,1]
```

```
[ ]: n_splits = 10
     n_repeats = 10
     seed = 7
     cv = RepeatedStratifiedKFold(n_splits=n_splits,
                                  n_repeats=n_repeats,
                                  random_state=seed)
```

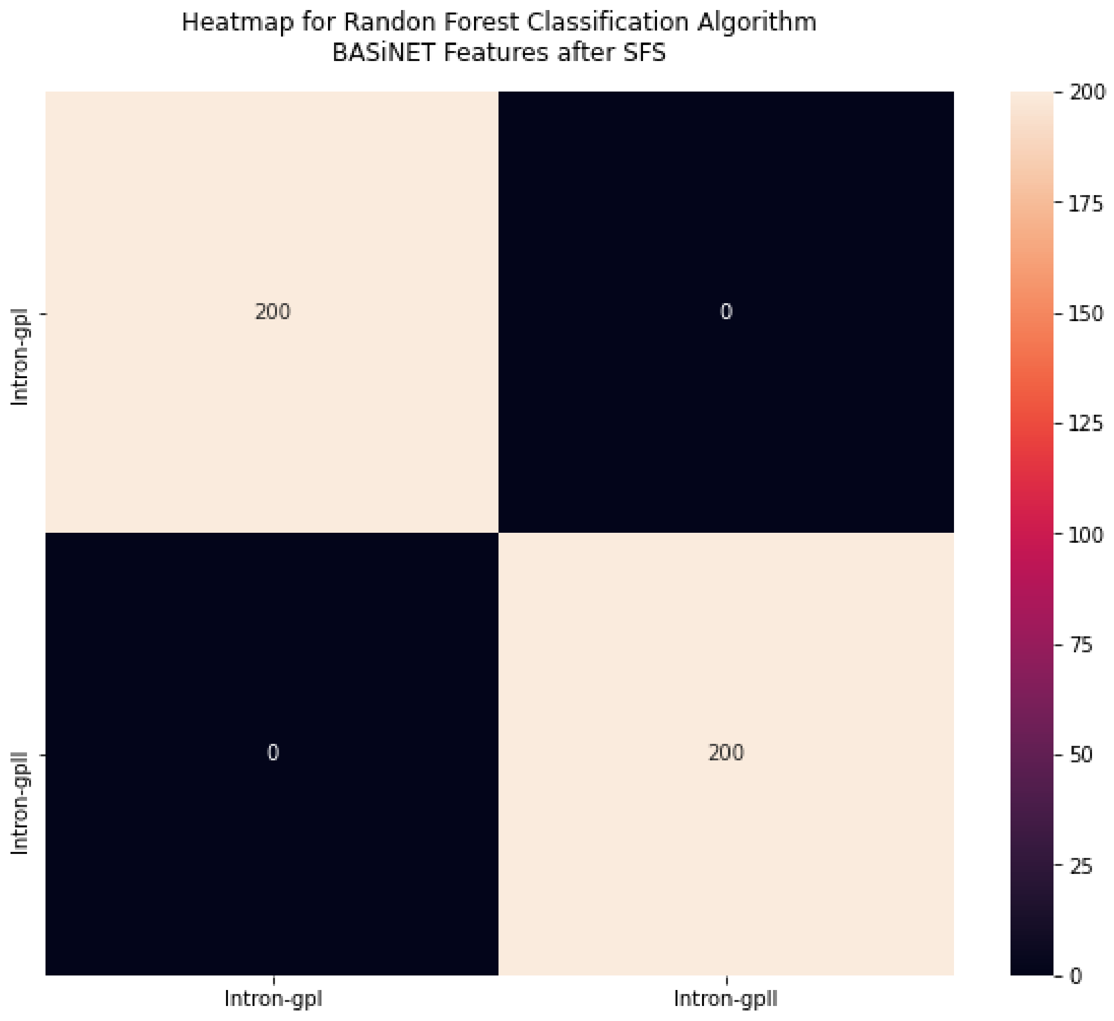
```
[ ]: n_scores_basinet_sfs = cross_validate(RF_basinet_sfs,
                                           X_basinet_sfs,
                                           Y_basinet_sfs,
                                           scoring = 'accuracy',
                                           cv = cv,
                                           n_jobs=-1,
                                           error_score = 'raise')
     print("Mean:", np.mean(n_scores_basinet_sfs['test_score']))
```

```
Mean: 1.0
```

```
[ ]: cm_basinet_sfs = confusion_matrix(y_true = Y_basinet_sfs_test,
                                       y_pred = Y_basinet_sfs_pred)
```

```
[ ]: plt.figure(figsize = (10,8))
     ax = plt.axes()
     x_axis_labels = ['Intron-gpI', 'Intron-gpII'] # labels for x-axis
     y_axis_labels = ['Intron-gpI', 'Intron-gpII'] # labels for y-axis
     sns.heatmap(cm_basinet_sfs,
                 vmin=0,
                 vmax=200,
                 annot=True,
                 fmt="d",
                 ax = ax,
                 xticklabels=x_axis_labels,
                 yticklabels=y_axis_labels)
     ax.set_title('Heatmap for Randon Forest Classification Algorithm\nBASiNET␣
      ↪Features after SFS',pad=15)
```

```
[ ]: Text(0.5, 1.0, 'Heatmap for Randon Forest Classification Algorithm\nBASiNET
     Features after SFS')
```

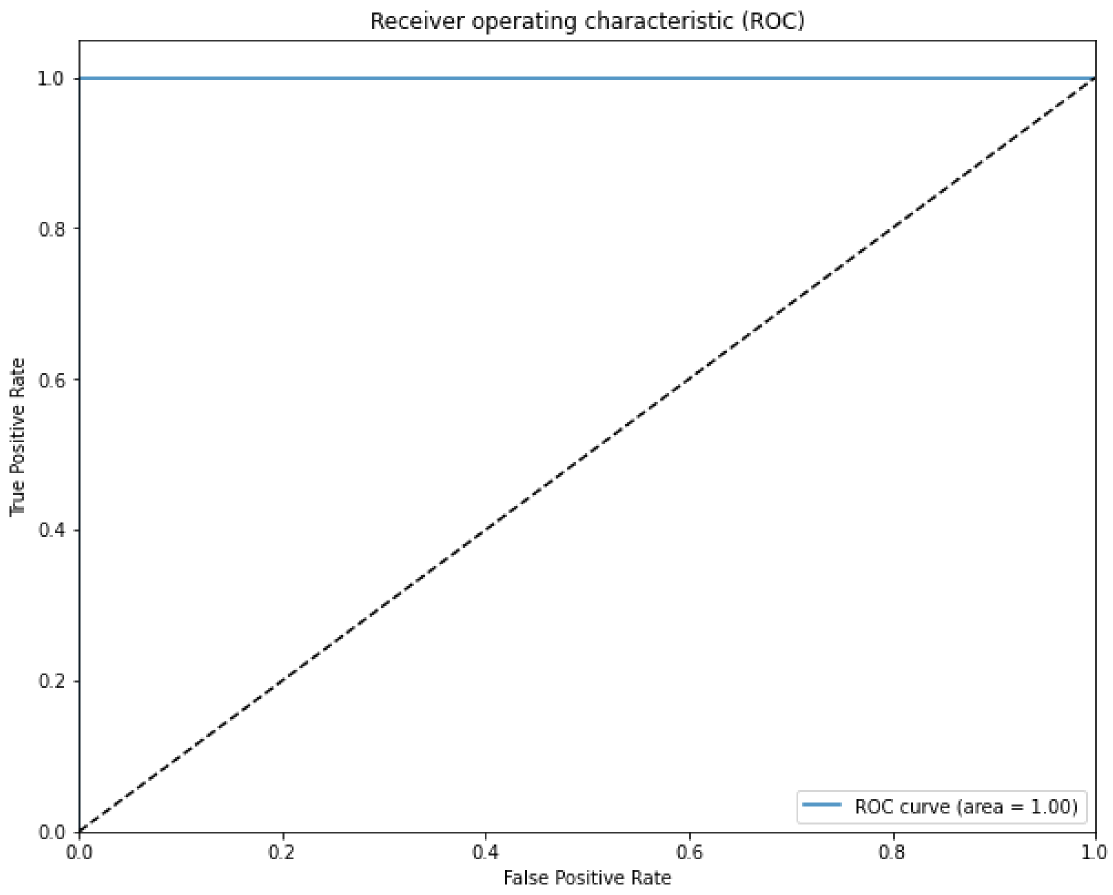


```
[ ]: print(classification_report(Y_basinet_sfs_test, Y_basinet_sfs_pred,␣
      ↪target_names=['Intron-gpI','Intron-gpII']))
     print("Accuracy:",accuracy_score(Y_basinet_sfs_test, Y_basinet_sfs_pred))
     print("Matthews correlation coefficient (MCC):
      ↪",matthews_corrcoef(Y_basinet_sfs_test,Y_basinet_sfs_pred))
```

```
                  precision    recall  f1-score   support

     Intron-gpI       1.00      1.00      1.00       200
    Intron-gpII       1.00      1.00      1.00       200

       accuracy                           1.00       400
      macro avg       1.00      1.00      1.00       400
   weighted avg       1.00      1.00      1.00       400

Accuracy: 1.0
Matthews correlation coefficient (MCC): 1.0
```

```
[ ]: fpr_RF_basinet_sfs, tpr_RF_basinet_sfs, _ = roc_curve(Y_basinet_sfs_test,␣
      ↪Y_basinet_sfs_pred_proba)
     auc_RF_basinet_sfs = auc(fpr_RF_basinet_sfs, tpr_RF_basinet_sfs)
```

```
[ ]: plt.figure(figsize = (10,8))
     plt.plot(fpr_RF_basinet_sfs, tpr_RF_basinet_sfs, label='ROC curve (area = %0.
      ↪2f)' % auc_RF_basinet_sfs)
     plt.plot([0, 1], [0, 1], 'k--')
     plt.xlim([0.0, 1.0])
     plt.ylim([0.0, 1.05])
     plt.xlabel('False Positive Rate')
     plt.ylabel('True Positive Rate')
     plt.title('Receiver operating characteristic (ROC)')
     plt.legend(loc="lower right")
     plt.show()
```



```
[ ]: FP = cm_basinet_sfs.sum(axis=0) - np.diag(cm_basinet_sfs)
     FN = cm_basinet_sfs.sum(axis=1) - np.diag(cm_basinet_sfs)
     TP = np.diag(cm_basinet_sfs)
     TN = cm_basinet_sfs.sum() - (FP + FN + TP)
     FP = FP.astype(float)
     FN = FN.astype(float)
     TP = TP.astype(float)
     TN = TN.astype(float)

     # Recall or true positive rate
     TPR = TP/(TP+FN)
     m_TPR = np.mean(TPR)
     print("Recall:\t\t", m_TPR)
     # True negative rate
     TNR = TN/(TN+FP)
     m_TNR = np.mean(TNR)
     print("True Negative:\t", m_TNR)
     # Precision
     PPV = TP/(TP+FP)
```

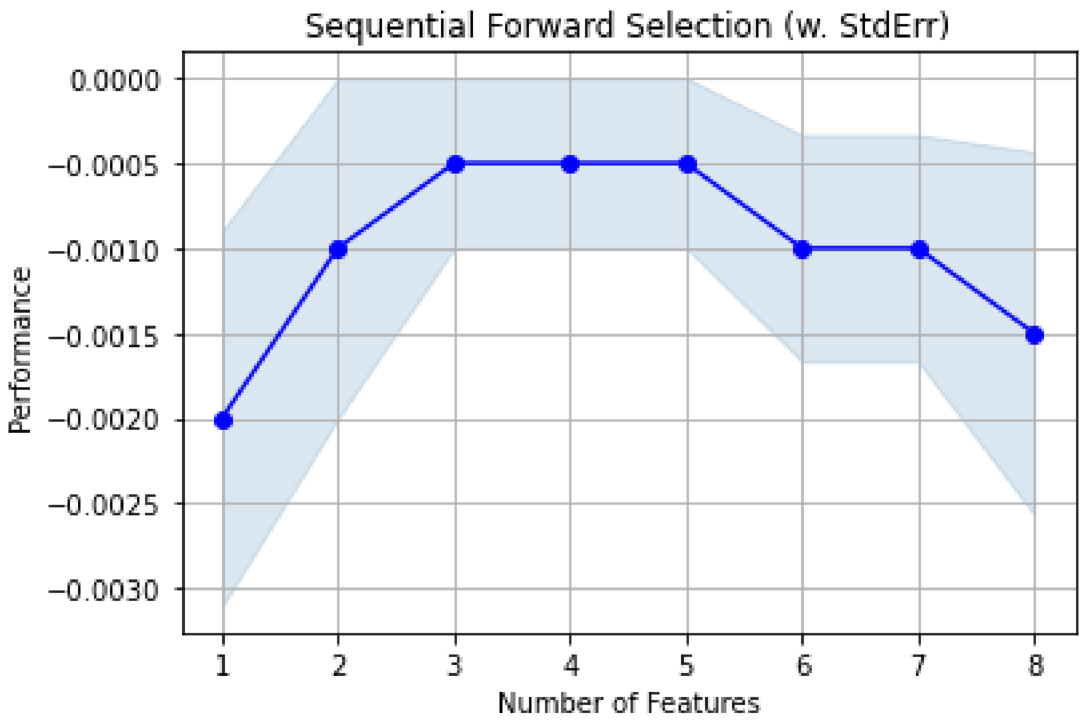
```
m_PPV = np.mean(PPV)
print("Precision:\t", m_PPV)
# Negative pred
NPV = TN/(TN+FN)
m_NPV = np.mean(NPV)
print("Pred. Negative:\t", m_NPV)
# False positive
FPR = FP/(FP+TN)
m_FPR = np.mean(FPR)
print("False Positive: ", m_FPR)
# False negative
FNR = FN/(TP+FN)
m_FNR = np.mean(FNR)
print("False Negative:\t", m_FNR)
# False discovery
FDR = FP/(TP+FP)
m_FDR = np.mean(FDR)
print("F Discovery:\t", m_FDR)
# Accuracy for each class
ACC = (TP+TN)/(TP+FP+FN+TN)
m_ACC = np.mean(ACC)
print("Accuracy:\t", m_ACC)

x = (m_TPR + m_FPR) * (m_TPR + m_FNR) * (m_TNR + m_FPR) * (m_TNR + m_FNR)
mcc = ((m_TPR * m_TNR) - (m_FPR * m_FNR)) / math.sqrt(x)

f1 = 2 * ((m_PPV * m_TPR) / (m_PPV + m_TPR))
print("F1:\t\t", f1)
print("MCC:\t\t", mcc)
```

```
Recall:          1.0
True Negative:   1.0
Precision:       1.0
Pred. Negative:  1.0
False Positive:  0.0
False Negative:  0.0
F Discovery:     0.0
Accuracy:        1.0
F1:              1.0
MCC:             1.0
```

1.9.2 RNAcon

```
[ ]: sfs_rnacon = SFS(RF_rnacon,
                      k_features=8, # half of the features
                      forward=True,
                      floating=False,
                      scoring='neg_mean_squared_error',
                      cv=10,
                      n_jobs=-1)
```

[ ]:
```
sfs_rnacon = sfs_rnacon.fit(X_rnacon, Y_rnacon)
fig = plot_sfs(sfs_rnacon.get_metric_dict(), kind='std_err')
plt.title('Sequential Forward Selection (w. StdErr)')
plt.grid()
plt.show()
```



[ ]:
```
sfs_rnacon_metrics = pd.DataFrame.from_dict(sfs_rnacon.get_metric_dict()).T
# Three features have better metrics.
sfs_rnacon_features = list(sfs_rnacon_metrics["feature_names"][3])
print("Features Selected:", sfs_rnacon_features)
```

```
Features Selected: ['VAR.BURT', 'VAR.CORE', 'DENSITY']
```

Random Forest with the selected features.

[ ]:
```
X_rnacon_sfs = rnacon[sfs_rnacon_features]
Y_rnacon_sfs = rnacon['CLASS']
X_rnacon_sfs_train, X_rnacon_sfs_test, Y_rnacon_sfs_train, Y_rnacon_sfs_test =␣
 ↪train_test_split(X_rnacon_sfs,
                                                                               ␣
 ↪                Y_rnacon_sfs,
                                                                               ␣
 ↪                test_size = 0.2,
                                                                               ␣
 ↪                stratify = Y_rnacon_sfs)
```

```
[ ]: RF_rnacon_sfs = RandomForestClassifier(n_estimators = 100)
     RF_rnacon_sfs.fit(X_rnacon_sfs_train,Y_rnacon_sfs_train)
     Y_rnacon_sfs_pred = RF_rnacon_sfs.predict(X_rnacon_sfs_test)
     Y_rnacon_sfs_pred_proba = RF_rnacon_sfs.predict_proba(X_rnacon_sfs_test)[::,1]
```

```
[ ]: n_splits = 10
     n_repeats = 10
     seed = 7
     cv = RepeatedStratifiedKFold(n_splits=n_splits,
                                  n_repeats=n_repeats,
                                  random_state=seed)
```

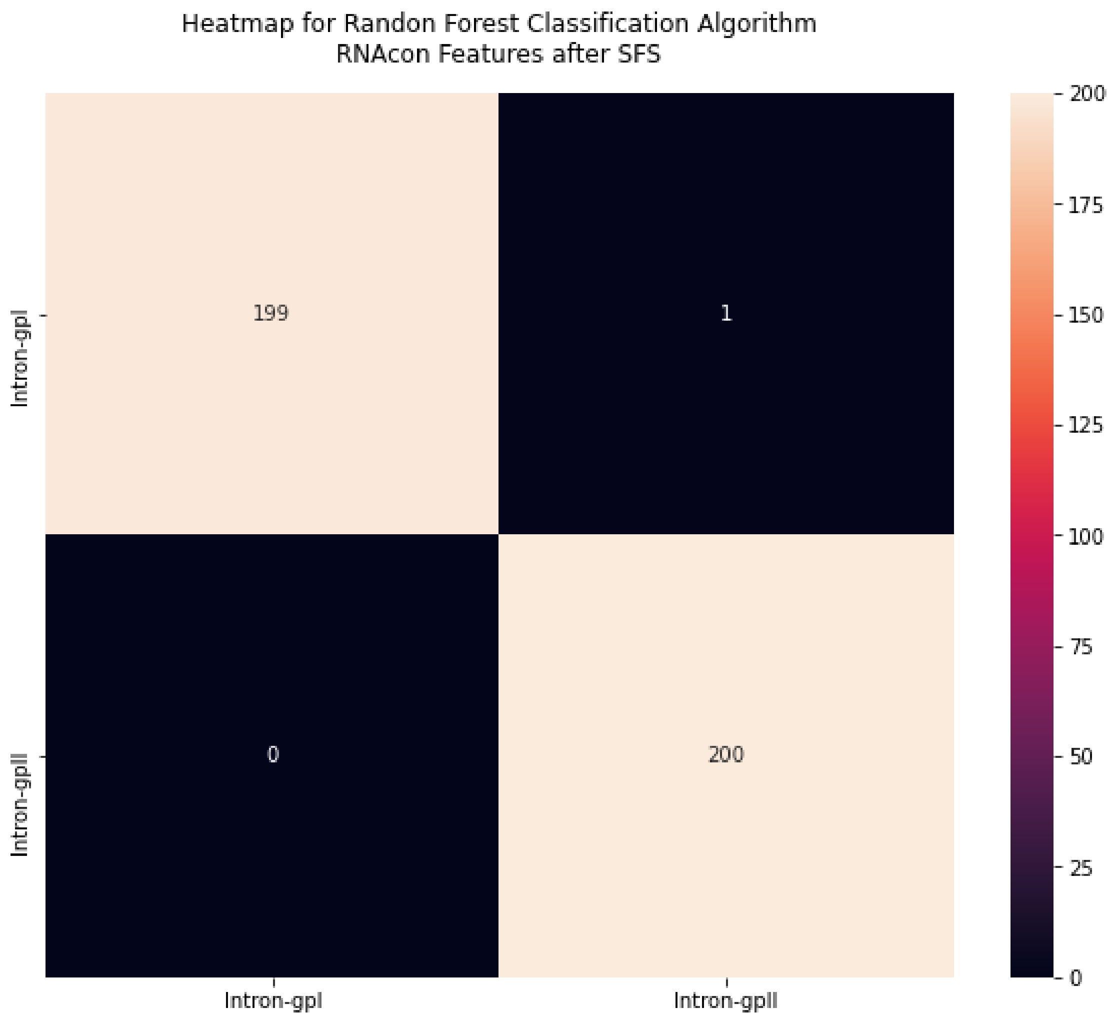
```
[ ]: n_scores_rnacon_sfs = cross_validate(RF_rnacon_sfs,
                                          X_rnacon_sfs,
                                          Y_rnacon_sfs,
                                          scoring = 'accuracy',
                                          cv = cv,
                                          n_jobs=-1,
                                          error_score = 'raise')
     print("Mean:", np.mean(n_scores_rnacon_sfs['test_score']))
```

```
Mean: 0.9993500000000001
```

```
[ ]: cm_rnacon_sfs = confusion_matrix(y_true = Y_rnacon_sfs_test,
                                      y_pred = Y_rnacon_sfs_pred)
```

```
[ ]: plt.figure(figsize = (10,8))
     ax = plt.axes()
     x_axis_labels = ['Intron-gpI', 'Intron-gpII'] # labels for x-axis
     y_axis_labels = ['Intron-gpI', 'Intron-gpII'] # labels for y-axis
     sns.heatmap(cm_rnacon_sfs,
                 vmin=0,
                 vmax=200,
                 annot=True,
                 fmt="d",
                 ax = ax,
                 xticklabels=x_axis_labels,
                 yticklabels=y_axis_labels)
     ax.set_title('Heatmap for Randon Forest Classification Algorithm\nRNAcon␣
      ↪Features after SFS',pad=15)
```

[ ]: Text(0.5, 1.0, 'Heatmap for Randon Forest Classification Algorithm\nRNAcon
     Features after SFS')



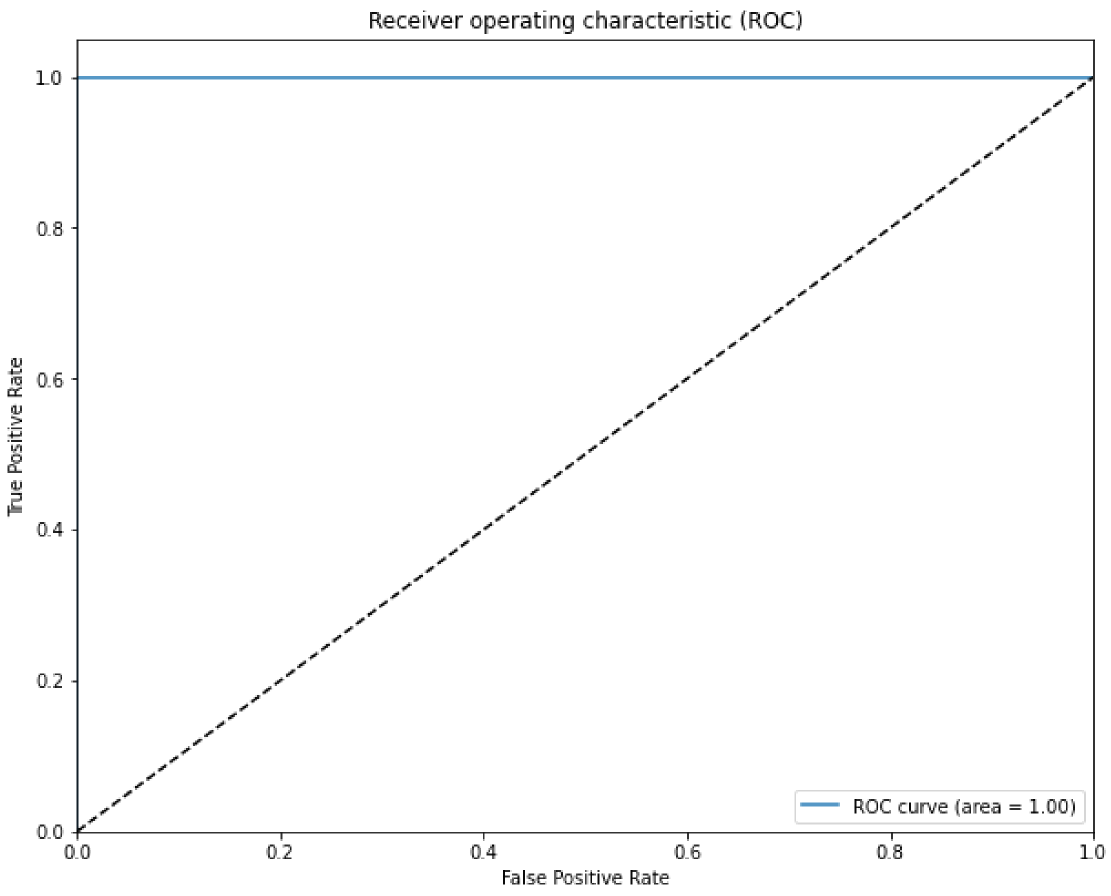
```
[ ]: print(classification_report(Y_rnacon_sfs_test, Y_rnacon_sfs_pred,␣
      ↪target_names=['Intron-gpI','Intron-gpII']))
     print("Accuracy:",accuracy_score(Y_rnacon_sfs_test, Y_rnacon_sfs_pred))
     print("Matthews correlation coefficient (MCC):
      ↪",matthews_corrcoef(Y_rnacon_sfs_test,Y_rnacon_sfs_pred))
```

```
              precision    recall  f1-score   support

  Intron-gpI       1.00      0.99      1.00       200
 Intron-gpII       1.00      1.00      1.00       200

    accuracy                           1.00       400
   macro avg       1.00      1.00      1.00       400
```

```
weighted avg       1.00      1.00      1.00       400

Accuracy: 0.9975
Matthews correlation coefficient (MCC): 0.9950124377332079
```

```
[ ]: fpr_RF_rnacon_sfs, tpr_RF_rnacon_sfs, _ = roc_curve(Y_rnacon_sfs_test,␣
      ↪Y_rnacon_sfs_pred_proba)
     auc_RF_rnacon_sfs = auc(fpr_RF_rnacon_sfs, tpr_RF_rnacon_sfs)
```

```
[ ]: plt.figure(figsize = (10,8))
     plt.plot(fpr_RF_rnacon_sfs, tpr_RF_rnacon_sfs, label='ROC curve (area = %0.2f)'␣
      ↪% auc_RF_rnacon_sfs)
     plt.plot([0, 1], [0, 1], 'k--')
     plt.xlim([0.0, 1.0])
     plt.ylim([0.0, 1.05])
     plt.xlabel('False Positive Rate')
     plt.ylabel('True Positive Rate')
     plt.title('Receiver operating characteristic (ROC)')
     plt.legend(loc="lower right")
     plt.show()
```



```
[ ]: FP = cm_rnacon_sfs.sum(axis=0) - np.diag(cm_rnacon_sfs)
     FN = cm_rnacon_sfs.sum(axis=1) - np.diag(cm_rnacon_sfs)
     TP = np.diag(cm_rnacon_sfs)
     TN = cm_rnacon_sfs.sum() - (FP + FN + TP)
     FP = FP.astype(float)
     FN = FN.astype(float)
     TP = TP.astype(float)
     TN = TN.astype(float)

     # Recall or true positive rate
     TPR = TP/(TP+FN)
     m_TPR = np.mean(TPR)
     print("Recall:\t\t", m_TPR)
     # True negative rate
     TNR = TN/(TN+FP)
     m_TNR = np.mean(TNR)
     print("True Negative:\t", m_TNR)
     # Precision
     PPV = TP/(TP+FP)
     m_PPV = np.mean(PPV)
     print("Precision:\t", m_PPV)
     # Negative pred
     NPV = TN/(TN+FN)
     m_NPV = np.mean(NPV)
     print("Pred. Negative:\t", m_NPV)
     # False positive
     FPR = FP/(FP+TN)
     m_FPR = np.mean(FPR)
     print("False Positive: ", m_FPR)
     # False negative
     FNR = FN/(TP+FN)
     m_FNR = np.mean(FNR)
     print("False Negative:\t", m_FNR)
     # False discovery
     FDR = FP/(TP+FP)
     m_FDR = np.mean(FDR)
     print("F Discovery:\t", m_FDR)
     # Accuracy for each class
     ACC = (TP+TN)/(TP+FP+FN+TN)
     m_ACC = np.mean(ACC)
     print("Accuracy:\t", m_ACC)

     x = (m_TPR + m_FPR) * (m_TPR + m_FNR) * (m_TNR + m_FPR) * (m_TNR + m_FNR)
     mcc = ((m_TPR * m_TNR) - (m_FPR * m_FNR)) / math.sqrt(x)

     f1 = 2 * ((m_PPV * m_TPR) / (m_PPV + m_TPR))
```

```
print("F1:\t\t", f1)
print("MCC:\t\t", mcc)
```

```
Recall:          0.9975
True Negative:   0.9975
Precision:       0.9975124378109452
Pred. Negative:  0.9975124378109452
False Positive:  0.0025
False Negative:  0.0025
F Discovery:     0.0024875621890547263
Accuracy:        0.9975
F1:              0.9975062188667011
MCC:             0.9950000000000001
```

For the complete dataset, the SFS feature selection step will not apply. This is not necessary as the SFS was applied to the two split datasets before and good results were observed using two or three features to classify non-coding RNA into intron-gpI and intron-gpII.

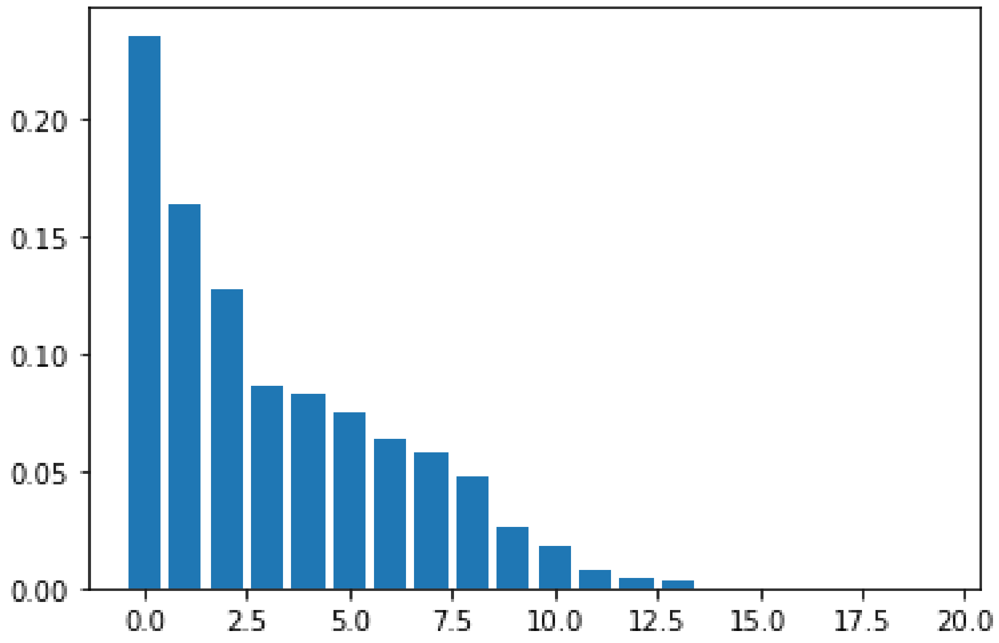
1.10 Feature Importance

1.10.1 BASiNET

```
[ ]: RF_basinet_importance = pd.Series(RF_basinet.feature_importances_,
                                  index=basinet.columns.drop('CLASS')).
      ↪sort_values(ascending=False)
     print(RF_basinet_importance)
     plt.bar([x for x in range(len(RF_basinet_importance))], RF_basinet_importance)
     plt.show()
```

```
MT3.1    0.236
DEG.1    0.163
MT4.1    0.127
MT3.2    0.086
ASPL.1   0.082
BET.2    0.075
SD.1     0.064
DEG.2    0.058
MT4.2    0.047
SD.2     0.026
CC.1     0.018
CC.2     0.008
MIN.2    0.004
ASPL.2   0.003
ASS.1    0.000
ASS.2    0.000
BET.1    0.000
MIN.1    0.000
```

```
MAX.2    0.000
MAX.1    0.000
dtype: float64
```

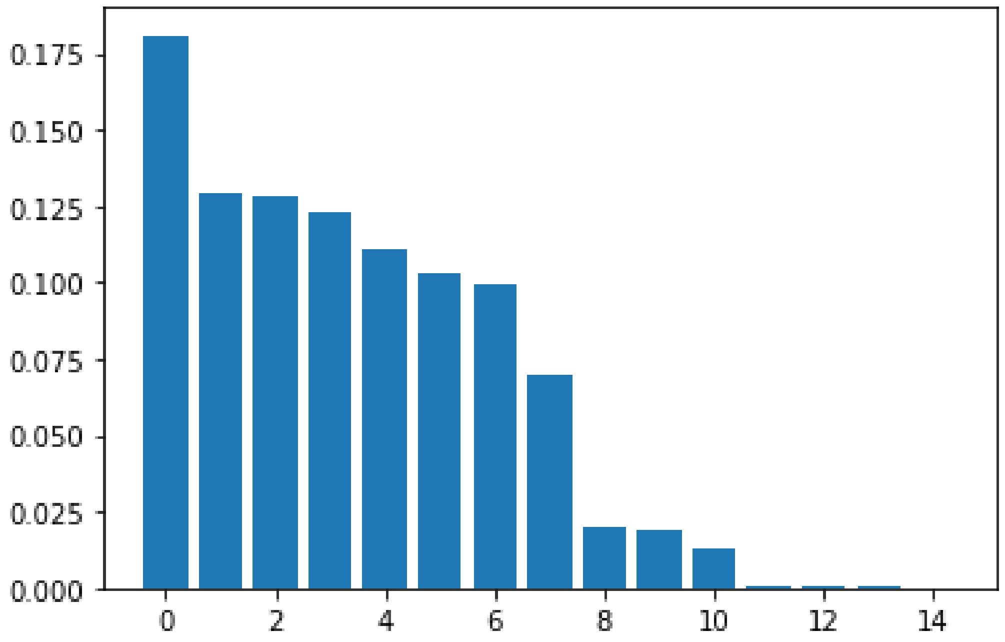


1.10.2 RNAcon

```
[ ]: RF_rnacon_importance = pd.Series(RF_rnacon.feature_importances_,
                                      index=rnacon.columns.drop('CLASS')).
      ↪sort_values(ascending=False)
     print(RF_rnacon_importance)
     plt.bar([x for x in range(len(RF_rnacon_importance))], RF_rnacon_importance)
     plt.show()
```

```
DENSITY          0.181
VAR.BURT         0.130
VAR.BET          0.128
VAR.EDGE.BET     0.123
MEAN.BET         0.111
MEAN.CIT.COUP    0.103
MEAN.BIB.COUP    0.100
MEAN.EDGE.BET    0.070
DIAMETER         0.020
MEAN.PATH.LEN    0.019
MEAN.DEG         0.013
MEAN.CORE        0.001
ART.POINTS       0.000
```

```
MEAN.BURT       0.000
VAR.CORE        0.000
dtype: float64
```



The most important features are different in the two approaches. In the SFS approach, the selected features from BASiNET are ASS.1 and DEG.1, while the two most important features from the Feature Importance are MT3.1 and DEG.1.
In the RNAcon dataset, two features are at the intersection of the two approaches, which are DENSITY and VAR.BURT features. But the VAR.CORE is not important to the Feature Importance approach. This is a common situation when the SFS approach is used, this situation is called "nested features" and is due to the forwarding action of the method.

1.11 Filter Approach

[ ]:
```python
from sklearn.feature_selection import SelectKBest, mutual_info_classif
```

1.12 Mutual Information

1.12.1 BASiNET

[ ]:
```python
mi_basinet = SelectKBest(score_func=mutual_info_classif,
                         k = 2)
mi_basinet.fit(X_basinet_train, Y_basinet_train)
cols = mi_basinet.get_support(indices=True)
colname_Filter = basinet.columns[cols]
```

```
print (colname_Filter)
```

```
Index(['DEG.1', 'MT3.1'], dtype='object')
```

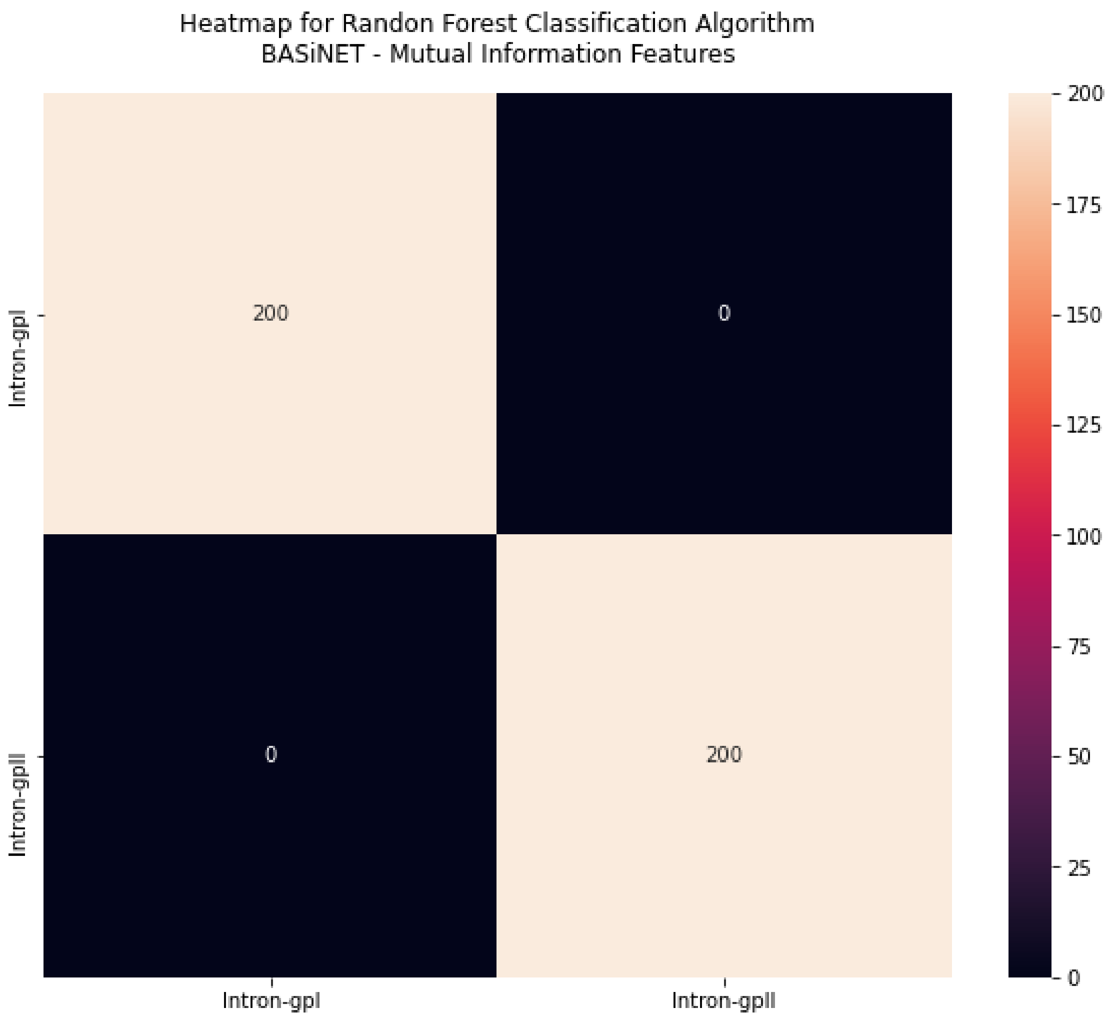
[ ]:
```
X_basinet_mi_train = mi_basinet.transform(X_basinet_train)
X_basinet_mi_test = mi_basinet.transform(X_basinet_test)
```

[ ]:
```
RF_basinet_mi = RandomForestClassifier(n_estimators=100)
RF_basinet_mi = RF_basinet_mi.fit(X_basinet_mi_train,Y_basinet_train)
Y_basinet_mi_pred = RF_basinet_mi.predict(X_basinet_mi_test)
Y_basinet_mi_pred_proba = RF_basinet_mi.predict_proba(X_basinet_mi_test)[::,1]
```

[ ]:
```
cm_basinet_mi = confusion_matrix(y_true = Y_basinet_test,
                                 y_pred = Y_basinet_mi_pred)
```

[ ]:
```
plt.figure(figsize = (10,8))
ax = plt.axes()
x_axis_labels = ['Intron-gpI', 'Intron-gpII'] # labels for x-axis
y_axis_labels = ['Intron-gpI', 'Intron-gpII'] # labels for y-axis
sns.heatmap(cm_basinet_mi,
            vmin=0,
            vmax=200,
            annot=True,
            fmt="d",
            ax = ax,
            xticklabels=x_axis_labels,
            yticklabels=y_axis_labels)
ax.set_title('Heatmap for Randon Forest Classification Algorithm\nBASiNET -␣
 ↪Mutual Information Features',pad=15)
```

[ ]:
```
Text(0.5, 1.0, 'Heatmap for Randon Forest Classification Algorithm\nBASiNET -
Mutual Information Features')
```



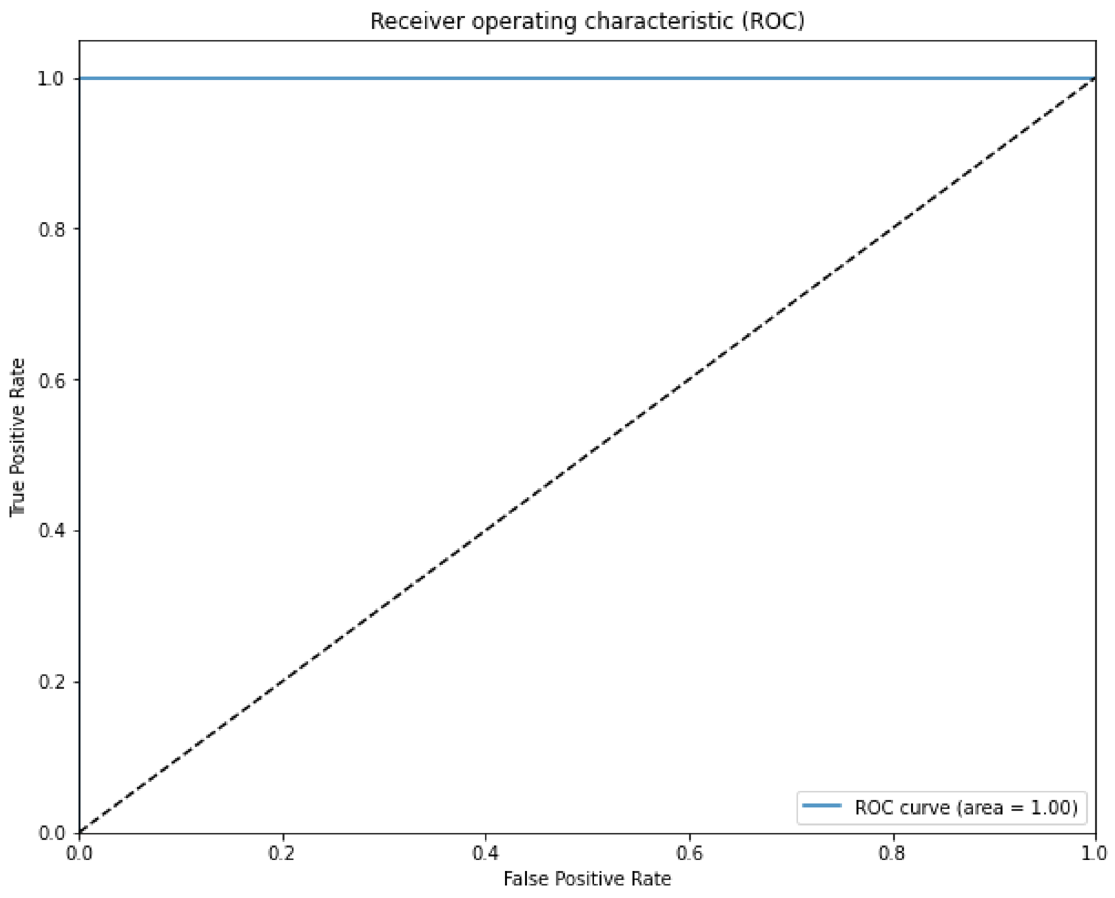
```
[ ]: print(classification_report(Y_basinet_test, Y_basinet_mi_pred,␣
      ↪target_names=['Intron-gpI','Intron-gpII']))
     print("Accuracy:",accuracy_score(Y_basinet_test, Y_basinet_mi_pred))
     print("Matthews correlation coefficient (MCC):
      ↪",matthews_corrcoef(Y_basinet_test,Y_basinet_mi_pred))
```

```
              precision    recall  f1-score   support

  Intron-gpI       1.00      1.00      1.00       200
 Intron-gpII       1.00      1.00      1.00       200

    accuracy                           1.00       400
   macro avg       1.00      1.00      1.00       400
weighted avg       1.00      1.00      1.00       400

Accuracy: 1.0
```

```
Matthews correlation coefficient (MCC): 1.0
```

[ ]:
```
fpr_RF_basinet_mi, tpr_RF_basinet_mi, _ = roc_curve(Y_basinet_test,␣
 ↪Y_basinet_mi_pred_proba)
auc_RF_basinet_mi = auc(fpr_RF_basinet_mi, tpr_RF_basinet_mi)
```

[ ]:
```
plt.figure(figsize = (10,8))
plt.plot(fpr_RF_basinet_mi, tpr_RF_basinet_mi, label='ROC curve (area = %0.2f)'␣
 ↪% auc_RF_basinet_mi)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```



```
[ ]: FP = cm_basinet_mi.sum(axis=0) - np.diag(cm_basinet_mi)
     FN = cm_basinet_mi.sum(axis=1) - np.diag(cm_basinet_mi)
     TP = np.diag(cm_basinet_mi)
     TN = cm_basinet_mi.sum() - (FP + FN + TP)
     FP = FP.astype(float)
     FN = FN.astype(float)
     TP = TP.astype(float)
     TN = TN.astype(float)

     # Recall or true positive rate
     TPR = TP/(TP+FN)
     m_TPR = np.mean(TPR)
     print("Recall:\t\t", m_TPR)
     # True negative rate
     TNR = TN/(TN+FP)
     m_TNR = np.mean(TNR)
     print("True Negative:\t", m_TNR)
     # Precision
     PPV = TP/(TP+FP)
     m_PPV = np.mean(PPV)
     print("Precision:\t", m_PPV)
     # Negative pred
     NPV = TN/(TN+FN)
     m_NPV = np.mean(NPV)
     print("Pred. Negative:\t", m_NPV)
     # False positive
     FPR = FP/(FP+TN)
     m_FPR = np.mean(FPR)
     print("False Positive: ", m_FPR)
     # False negative
     FNR = FN/(TP+FN)
     m_FNR = np.mean(FNR)
     print("False Negative:\t", m_FNR)
     # False discovery
     FDR = FP/(TP+FP)
     m_FDR = np.mean(FDR)
     print("F Discovery:\t", m_FDR)
     # Accuracy for each class
     ACC = (TP+TN)/(TP+FP+FN+TN)
     m_ACC = np.mean(ACC)
     print("Accuracy:\t", m_ACC)

     x = (m_TPR + m_FPR) * (m_TPR + m_FNR) * (m_TNR + m_FPR) * (m_TNR + m_FNR)
     mcc = ((m_TPR * m_TNR) - (m_FPR * m_FNR)) / math.sqrt(x)

     f1 = 2 * ((m_PPV * m_TPR) / (m_PPV + m_TPR))
     print("F1:\t\t", f1)
print("MCC:\t\t", mcc)
```

```
Recall:          1.0
True Negative:   1.0
Precision:       1.0
Pred. Negative:  1.0
False Positive:  0.0
False Negative:  0.0
F Discovery:     0.0
Accuracy:        1.0
F1:              1.0
MCC:             1.0
```

1.12.2 RNAcon

It's anologue

1.13 Conclusion

In this notebook, data mining techniques were applied to a Rfam9 dataset composed of two intron non-coding RNA families. Two methods were compared in order to analyze the extracted features.

Using all the features provided by the BASiNET and RNAcon methods excellent results were obtained using the Random Forest algorithm. The third dataset was built merging the RNAcon and BASiNET datasets and, again, excellent results were obtained.

In order to reduce the problem dimensionality, the Feature Selection step was performed. Two approaches were tested, in the SFS approach, two features were selected from the BASiNET dataset, while three features were selected from the RNAcon dataset. To compare the selected features the Feature Importance approach was made and different features from the BASiNET dataset were selected, in the RNAcon two features were at the intersection of SFS and Feature Importance, and one feature was selected in the SFS approach that is not important in the Feature Importance approach. This situation is called "Nested Features" and is due to the greedy algorithm greedily includes features and may include features that do not improve results.

In the filter approach, using Mutual Information features the two most important features of the BASiNET dataset are selected, that is, the same two features that are selected in Feature Importance are selected in the Mutual Information approach. Using these features in the Random Forest algorithm, excellent results were obtained.

Thus, using only two features provided by the BASiNET feature extraction approach, the Random Forest algorithm obtains optimal results for classifying two intron non-coding RNA families.

More analysis can be done in this scenario, such as trying to classify using only one feature.