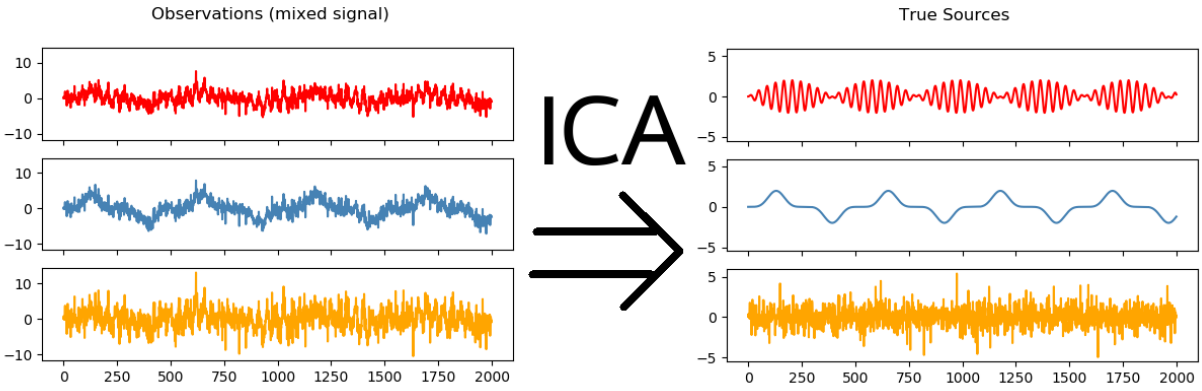
Pattern Recognition and Machine Learning

ICA & Sequential Feature Selector

Kartik Choudhary [B20CS025]

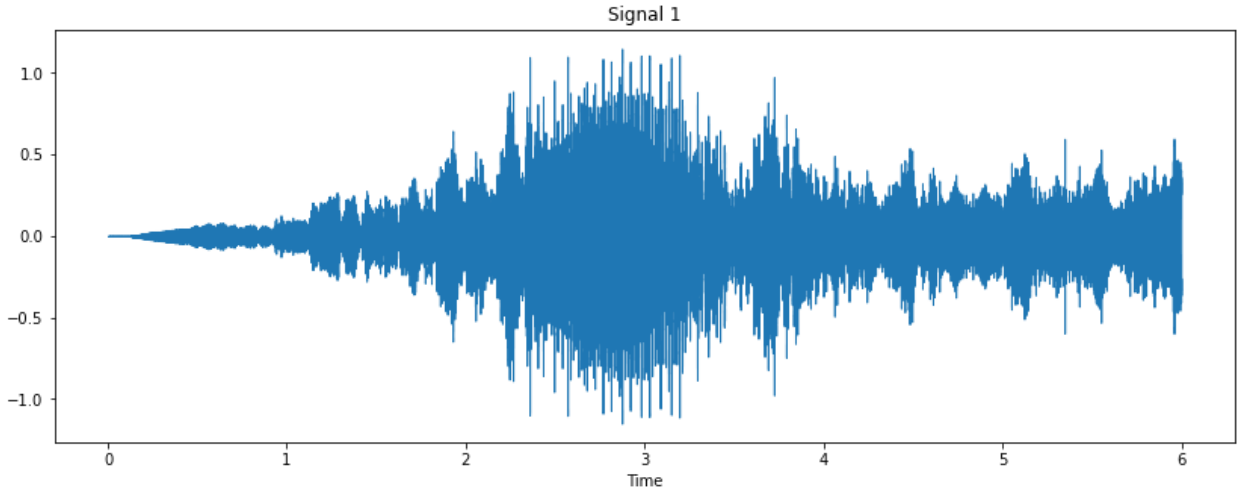


Question 1: Independent Component Analysis(ICA)

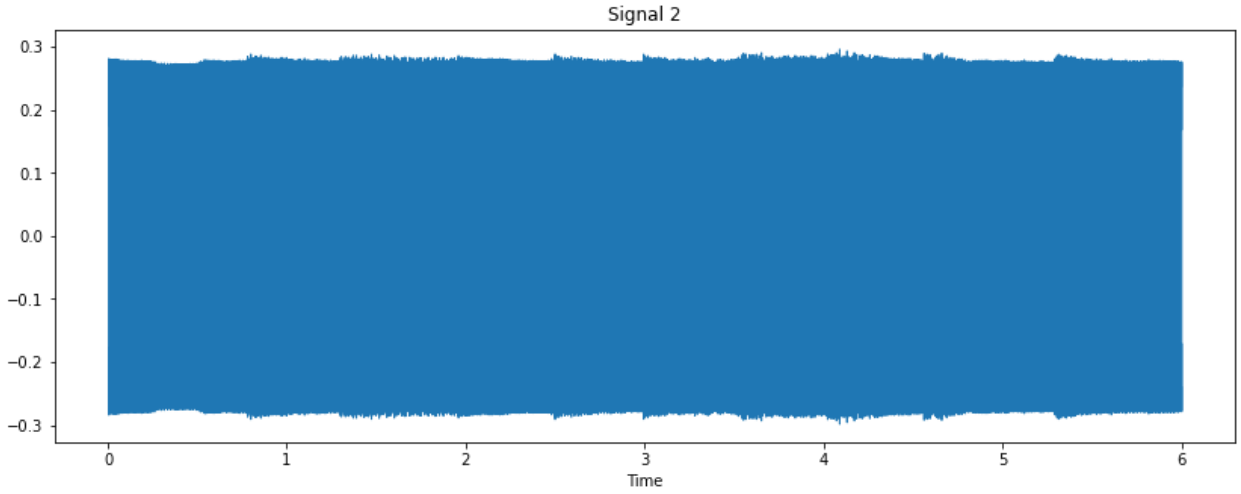
1) For reading visualizing and listening to the audio files the librosa library is imported. ‘Wave’ and ‘IPython’ modules were imported in order to read, listen and

visualize the audio files. Audio files were visualized using the matplotlib library and the plots can be seen below:

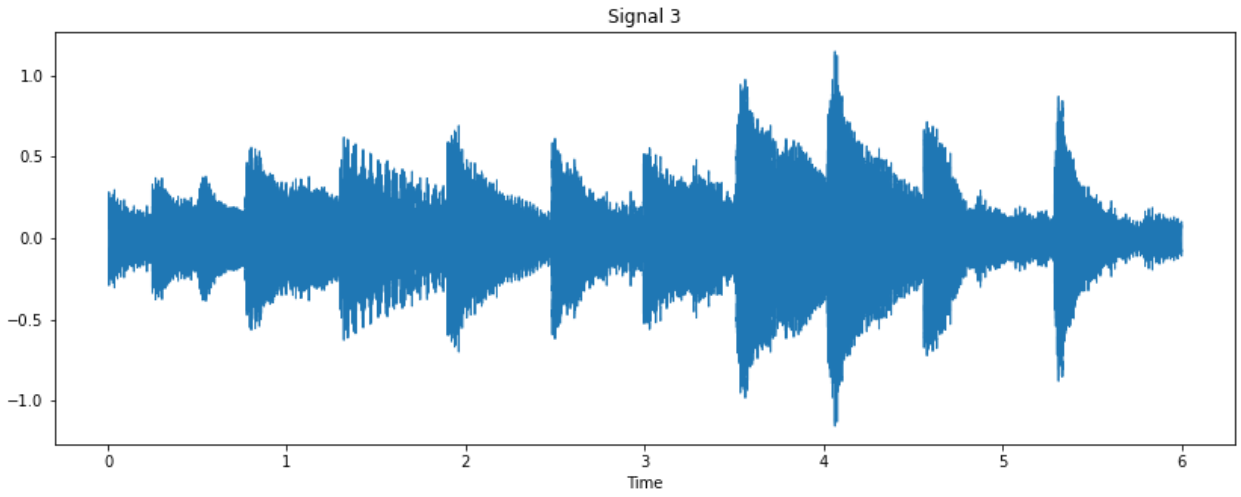
SIGNAL 1:



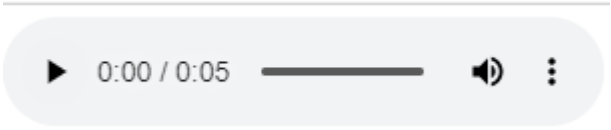
SIGNAL 2:



SIGNAL 3:



Listening to the audio files: The signals are 'listened' via 'IPython.display.Audio'



2) The three signals are then zipped together and converted into the NumPy array for further analysis. The X dataset is then created.
3) Implementation of ICA from scratch is then done and the unmixed signal is finally obtained. The fit function takes as input the following:

   Iterations (default = 1000): Number of iterations for convergence.

   Threshold (default = 1e-10): If error < threshold, loop would break.

   X: Mixed Signals

   The functionalities are mentioned below:

Centralize: The function centralizes X(mixed-signal) by taking the mean of individual signals and subtracting it from the individual signals.

Whiten: The function whitens X(centralized mixed-signal), implying that the covariance matrix of the whitened signal will be equal to the identity matrix.

This requires the eigenvalue decomposition of its covariance matrix. The corresponding mathematical equation can be described as follows:

x = ED^(-0.5)ETx

E: orthogonal matrix of eigenvectors

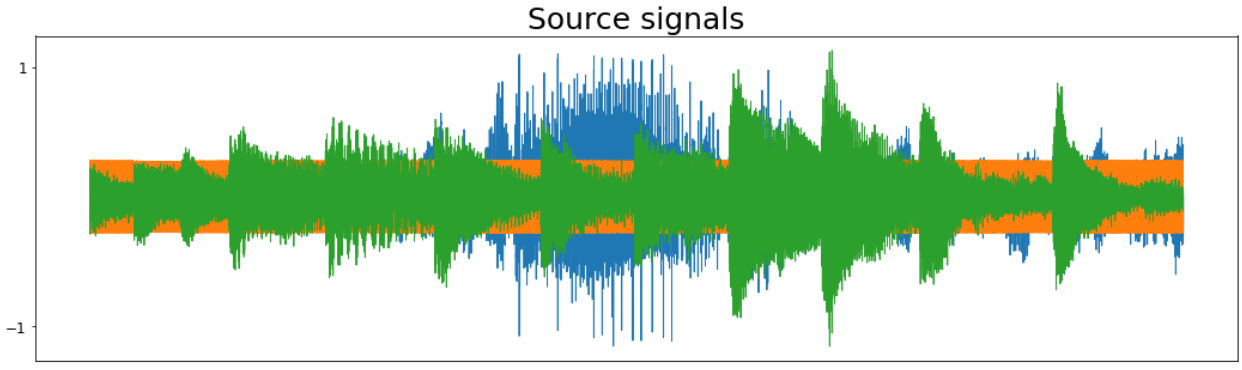
D: diagonal matrix of eigenvalues

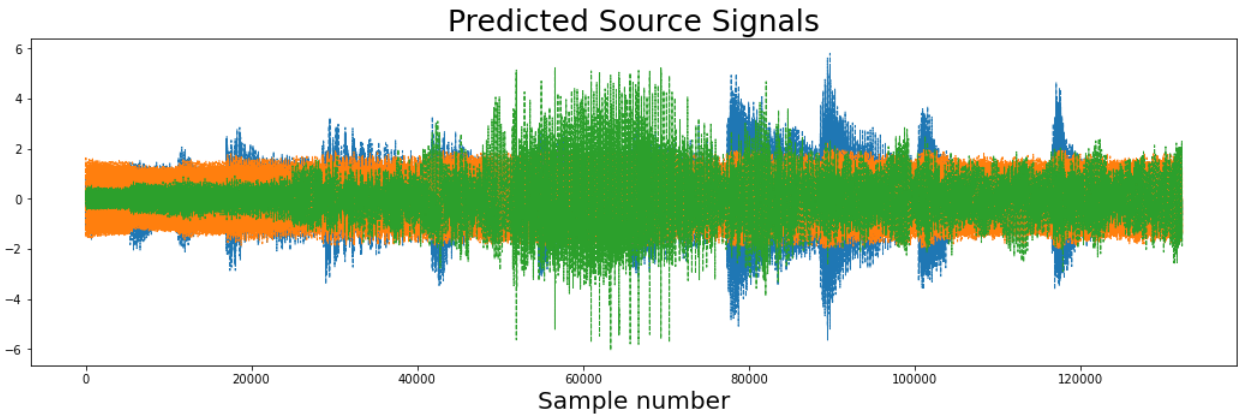
Fit: The function first centralizes and then whitens the data using the above-mentioned functionalities. Then, it updates the values of the de-mixing matrix w until the algorithm is converged or the maximum number of iterations has been reached.

Transform: The transform function does the matrix multiplication of the demixing matrix and the mixed-signal and returns the independent signals.

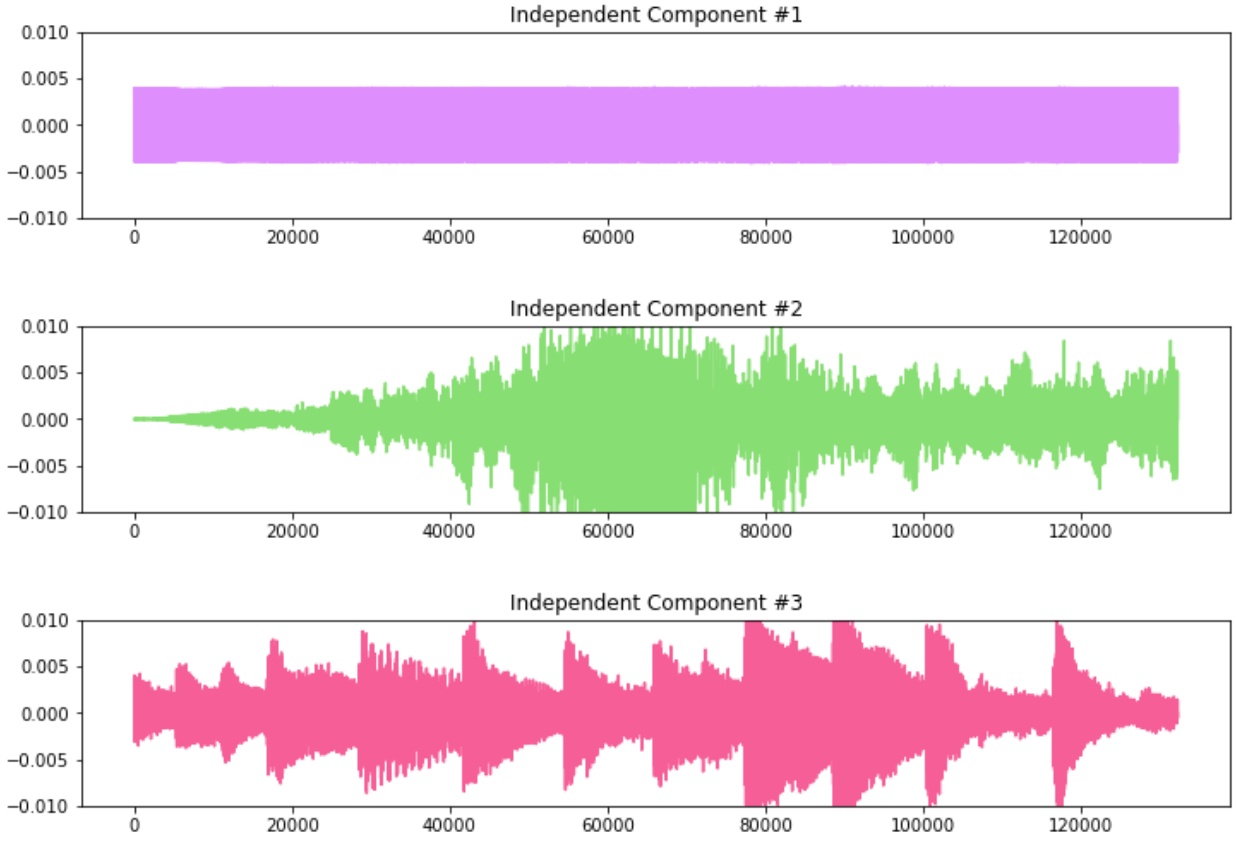
Other functionalities can be found in the notebook

4) The plotting mixture, real source, and predicted source from the output of ICA.





5) FastICA is implemented (import from sklearn.decomposition) selecting n_components = 3 on the mixed-signal X.
6) The individual components were extracted and visualized. The plots can be seen below:



7) From the graphs plotted in Subparts - 4 and 5, the following observations can be made: Recording 1: ICA is able to extract it with considerable accuracy. Recording 2: ICA is able to extract it partially with some noise. Recording 3: ICA is able to extract it with considerable accuracy. Recording 1: Fast ICA is able to extract it with considerable accuracy. Recording 2: Fast ICA is able to extract it with considerable accuracy. Recording 3: Fast ICA is able to extract it with considerable accuracy. From the above-mentioned observations, it can be concluded that FastICA performs better than ICA in extracting the source signal from the mixed-signal with considerable accuracy. Fast ICA is faster, computationally less expensive than ICA, and more stable

Question 2:

1) From the given link, the author downloaded 'train.csv'. The following functions were performed on the dataset to gain further insight into the dataset: '.head()', '.info()', and '.describe()'. The following preprocessing techniques have been implemented for the given dataset. The columns are encoded using the LabelEncoder.
2) An object of SFS by embedding Decision Tree classifier object, providing 10 features, forward as True, floating as False, and scoring = accuracy is created.
3) The above constructed SFS object is trained. Accuracy for all 10 features was calculated and the results are tabulated below:

Features Selected	Accuracy
12	0.7903833960456308
4, 12	0.8496881632686332
4, 7, 12	0.8912649918655335
4, 7, 10, 12	0.9217162073206818
4, 7, 10, 12	0.9291750066542516
2, 4, 5, 7, 10, 12	0.9414363341541672

2, 4, 5, 7, 10, 12, 19	0.9486449100081658
2, 4, 5, 7, 10, 12, 17, 19	0.9513878339578626
2, 4, 5, 7, 9, 10, 12, 17, 19	0.9521481429300355
2, 4, 5, 7, 10, 12, 13, 14, 17, 19	0.95074300073932

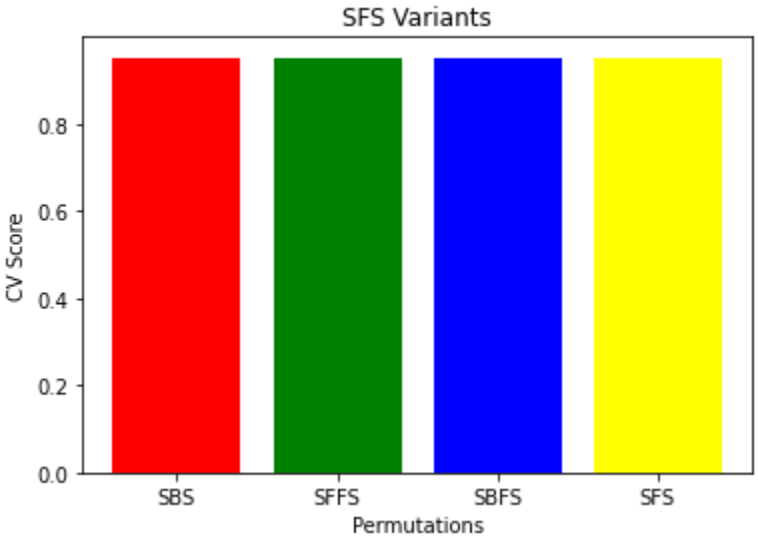
The best features selected are:

```
'feature_names': ('Customer Type',
 'Type of Travel',
 'Class',
 'Inflight wifi service',
 'Gate location',
 'Online boarding',
 'Seat comfort',
 'Inflight entertainment',
 'Baggage handling',
 'Inflight service')}
```

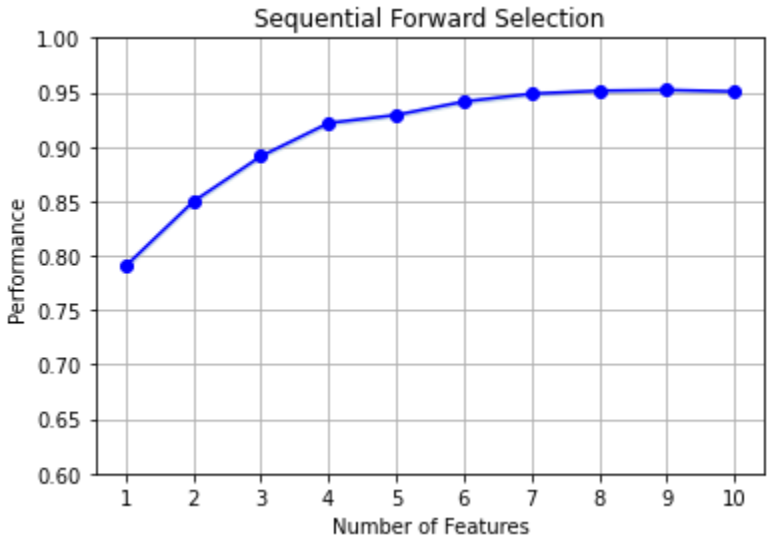
4) Using the forward and Floating parameters toggle between SFS(forward True, floating False), SBS (forward False, floating False), SFFS (forward True, floating True), SBFS (forward False, floating True), and choose cross-validation = 4 for each configuration. The cv scores for each configuration are tabulated and visualized below:

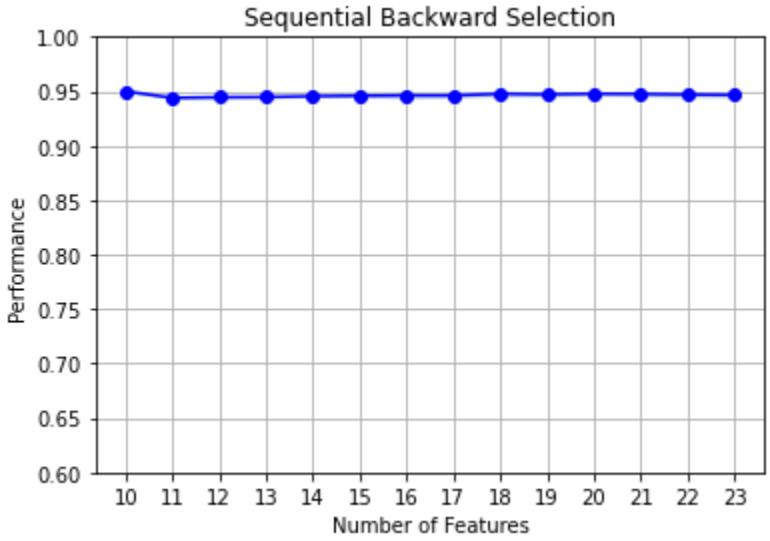
Technique	Average CV=4 Score
SBS	0.9499826763165999
sffs	0.9514744379427165
sbfs	0.9482695565137048
sfs	0.9499826763165999

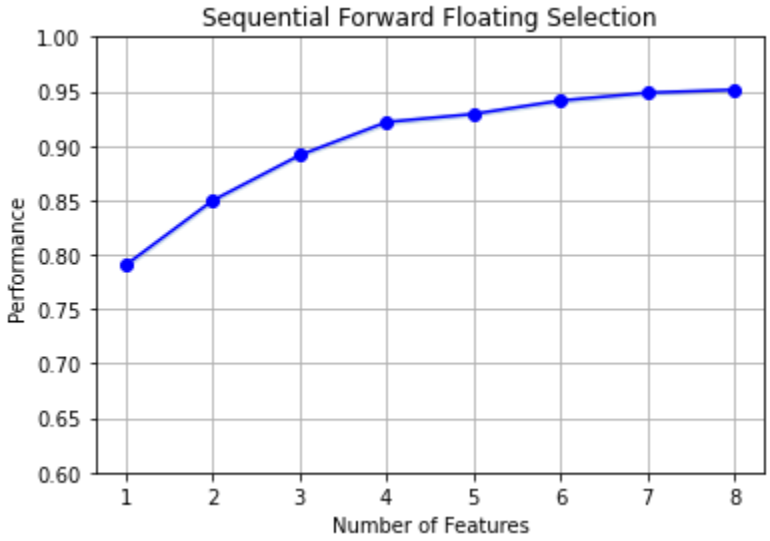
A plot for the average CV scores was plotted below:

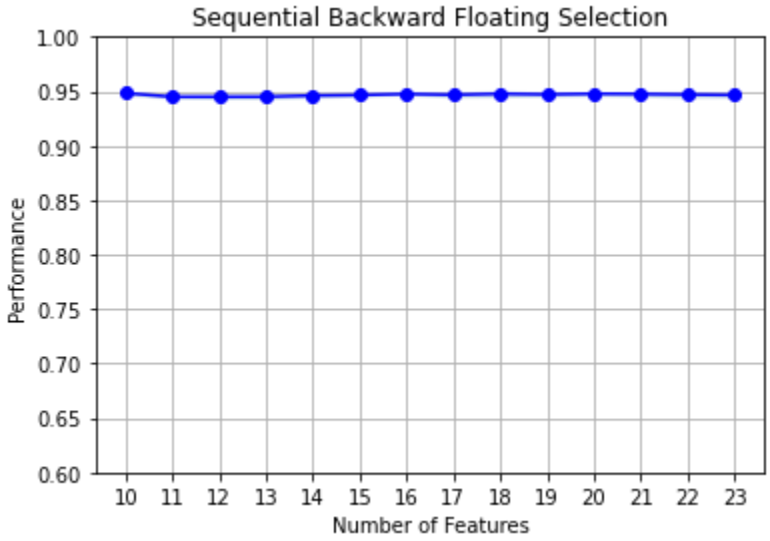


5) The output from the feature selection is visualized in a pandas DataFrame format using the get_metric_dict for all four configurations in the Google Colab Notebook.
6) Plots of the results for each configuration (from mlxtend.plotting import plot_sequential_feature_selection)

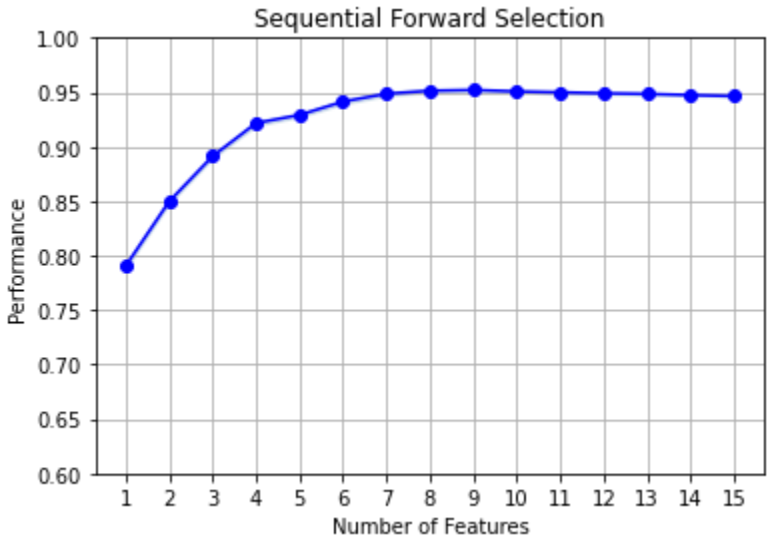



Sequential Backward Selection
Performance
Number of Features
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
10 11 12 13 14 15 16 17 18 19 20 21 22 23


Sequential Forward Floating Selection
Performance
Number of Features
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
1 2 3 4 5 6 7 8



7) Variations of features by increasing or decreasing with performance are visualized and can be seen below:



Results obtained for different features:

Number of features: 1

k_features: (12,)

Score: 0.786755293148486

Number of features: 2

k_features: (4, 12)

Score: 0.8504558543366698

Number of features: 3

k_features: (4, 7, 12)

Score: 0.8904681156579699

Number of features: 4

k_features: (4, 7, 10, 12)

Score: 0.9219834643232151

Number of features: 5

k_features: (2, 4, 7, 10, 12)

Score: 0.9281632179987603

Number of features: 6

k_features: (2, 4, 5, 7, 10, 12)

Score: 0.9375868775483086

Number of features: 7

k_features: (2, 4, 5, 7, 10, 12, 19)

Score: 0.9448992717735261

Number of features: 8

k_features: (2, 4, 5, 7, 10, 12, 17, 19)

Score: 0.9437147002978203

Number of features: 9

k_features: (2, 4, 5, 7, 9, 10, 12, 17, 19)

Score: 0.9431482739332478

Number of features: 10

k_features: (2, 4, 5, 7, 9, 10, 12, 14, 17, 19)

Score: 0.9409340617808276

Number of features: 11

k_features: (1, 2, 4, 5, 7, 9, 10, 12, 13, 17, 19)

Score: 0.939183355687325

Number of features: 12

k_features: (1, 2, 4, 5, 7, 9, 10, 12, 13, 17, 19, 20)

Score: 0.9389257698070201

Number of features: 13

k_features: (1, 2, 4, 5, 7, 9, 10, 12, 13, 14, 17, 19, 20)

Score: 0.9390287564187606

Number of features: 14

k_features: (2, 4, 5, 7, 9, 10, 11, 12, 13, 14, 15, 17, 19, 20)

Score: 0.9378442910327915

Number of features: 15

k_features: (1, 2, 4, 5, 7, 9, 10, 11, 12, 13, 14, 15, 17, 19, 20)

Score: 0.9360933860210329

Best result was found for the following permutation:

Number of features: 7

k_features: (2, 4, 5, 7, 10, 12, 19)

Score: 0.9448992717735261