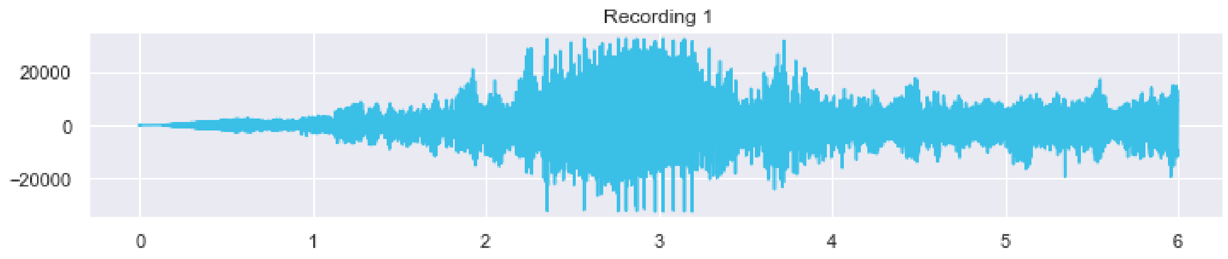
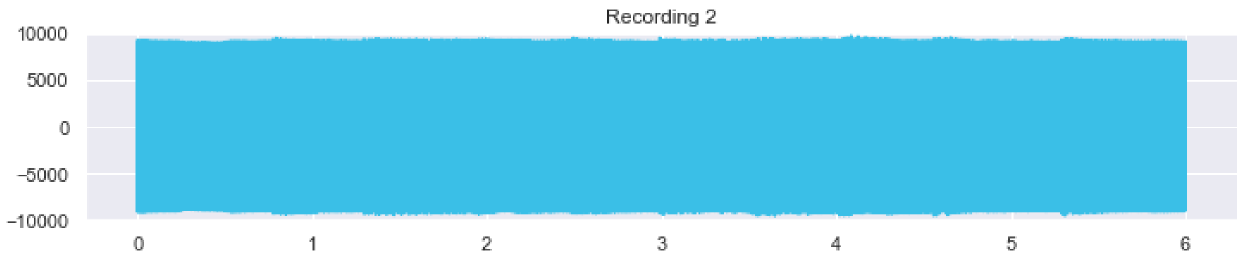
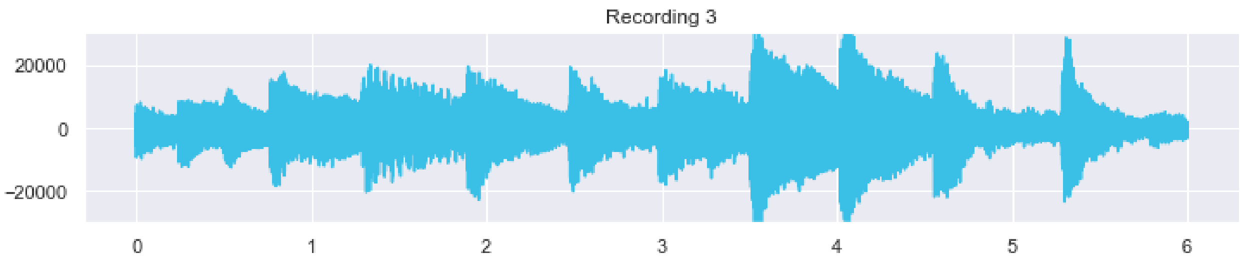
Pattern Recognition and Machine Learning
Lab 8

Question 1 - Implementing ICA from scratch

1. '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:
   Recording 1:
   

   Recording 2:
   

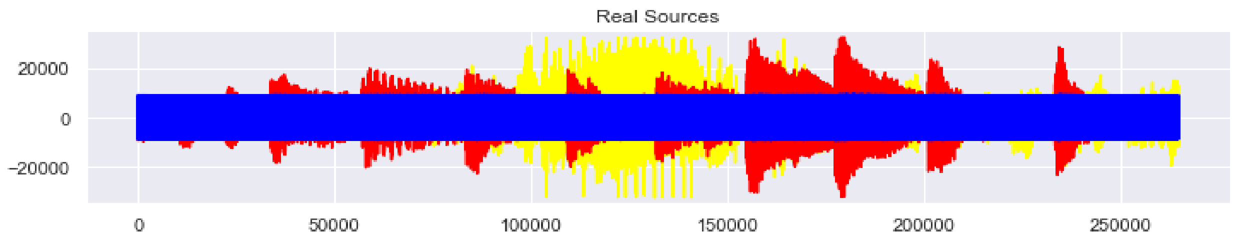
   Recording 3:
   

   The signals are 'listened' via 'IPython.display.Audio'.
2. Extracting raw audio(implemented in Subpart - 1) from the three-wave files and merging them to create a dataset is implemented in the following way: Raw data is extracted using '.readframes(-1)' and converting it into NumPy arrays. Mixed-signal is constructed using the following formula:

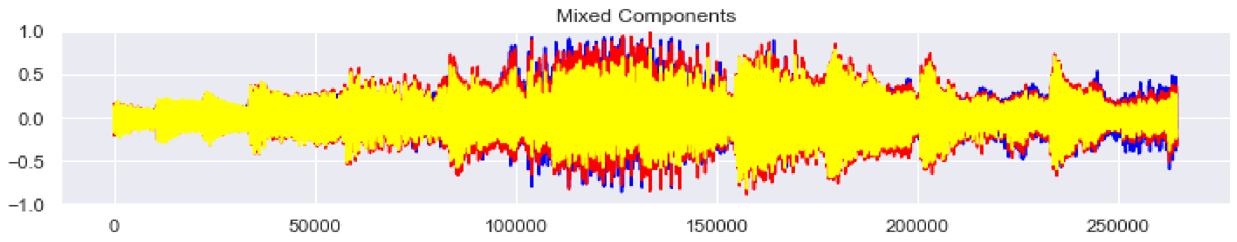
   $$X = A * S$$

   X - Mixed-signal
   A - Mixing Matrix (A random matrix of size (3,3))
   S - Source-Signal

Real Signals can be seen below:

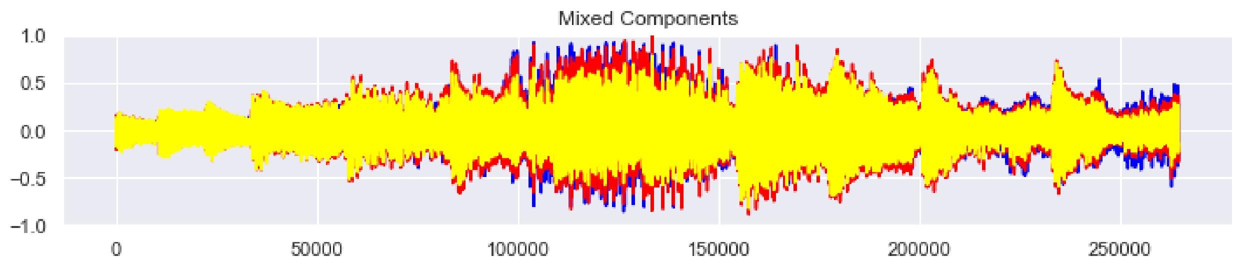


Mixed Signals can be seen below:

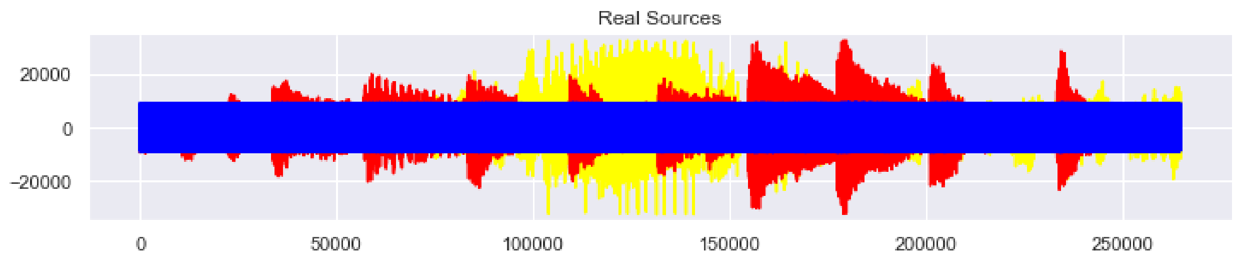


3. ICA is implemented (as a class) from scratch. The fit function takes as input the following:
   Iterations (default = 1000): Number of iterations for convergence.
   Threshold (default = 1e-10): If error < threshold, loop would break.
   X: Mixed Signals
   The functionalities are mentioned below:
   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: 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 above constructed ICA is implemented with mixed-signal X. The output is plotted as follows:

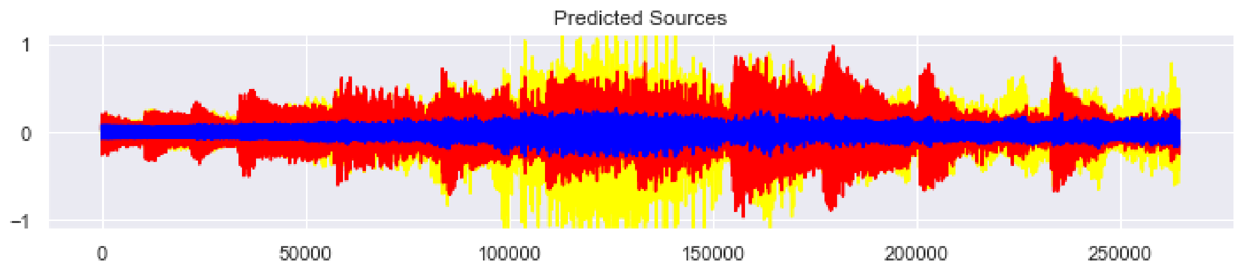
Mixed Components:



Real Components:

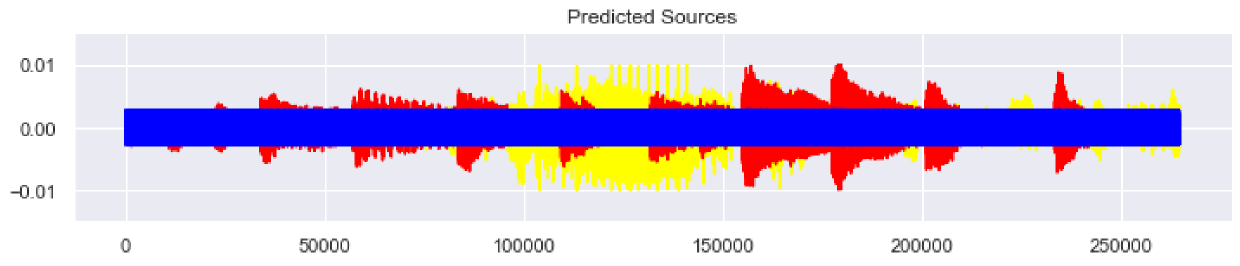


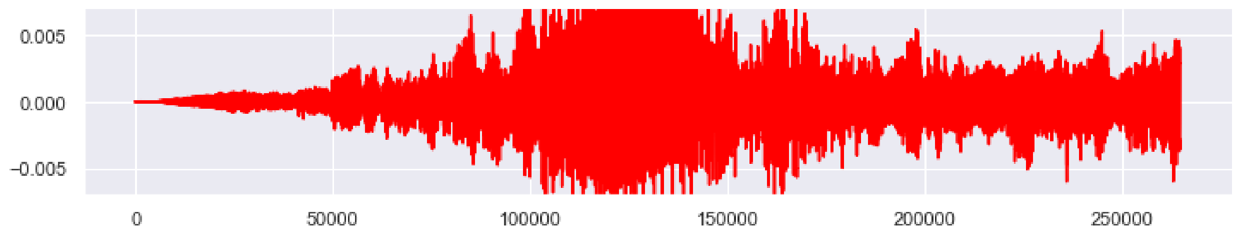
Predicted Components (ICA):

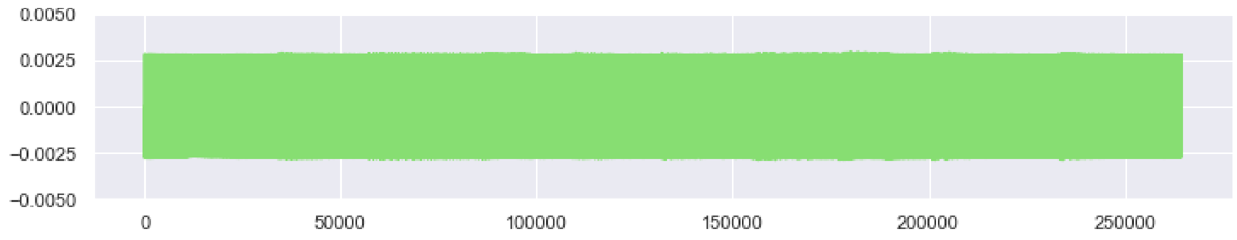


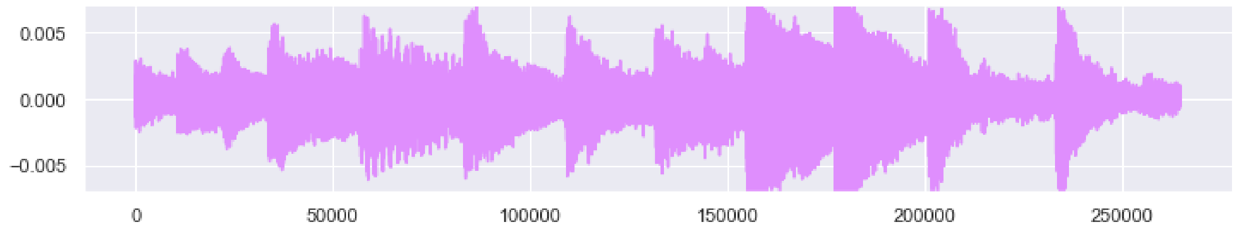
Observations:
Yellow Singal : ICA is able to extract it with considerable accuracy.
Red Signal : ICA is able to extract it with considerable accuracy.
Blue Signal : ICA is able to extract it partially woth some noise.

5. FastICA is implemented (import from sklearn.decomposition) selecting n_components = 3 on the mixed-signal X. The predicted source, along with the mixture and source signal are plotted as follows:
Predicted Components:



6. The individual components were extracted and visualised. The plots can be seen below:
   Recording 1:
   

   Recording 2:
   

   Recording 3:
   

   The predicted recordings are also listened to in the Google Colab Notebook via 'IPython.display.Audio'.
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 woth 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 - Sequential feature selection

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. Handling Missing Values: Considering only a small percentage of ‘NaN’ values are present in the ‘Arrival Delay in Minutes’ feature, hence it is simply dropped without loss of information. ‘Unamed’ column was dropped as it had no great information. Columns like ‘Gender’, ‘Staisfaction’, and ‘Class’ were encoded to make the dtaa ready for the SFS. The dataset is split into X (features) and y (target) NumPy arrays.
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	0.786
4, 12	0.850
4, 7, 12	0.890
4, 7, 10, 12	0.921
2, 4, 7, 10, 12	0.928
2, 4, 5, 7, 10, 12	0.937
2, 4, 5, 7, 10, 12, 19	0.944
2, 4, 5, 7, 10, 12, 17, 19	0.944
2, 4, 5, 7, 9, 10, 12, 17, 19	0.943
1, 2, 4, 5, 7, 9, 10, 12, 17, 19	0.940

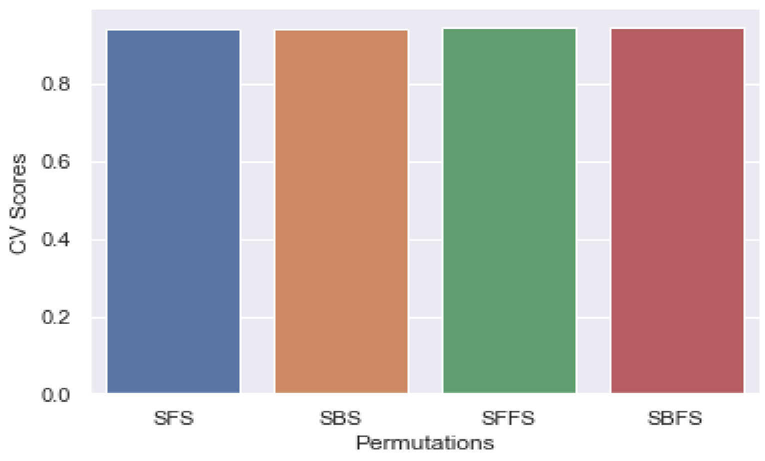
The 10 best features selected were:
1 -> Gender
2 -> Customer Type
4 -> Type of Travel
5 -> Class
7 -> Inflight wifi service
9 -> Ease of Online booking
10 -> Gate location
12 -> Online boarding
17 -> Baggage handling
19 -> Inflight service

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	CV Scores
SFS	0.94799176, 0.94670443, 0.94052523, 0.94515963
SBS	0.94418126, 0.93944387, 0.94397528, 0.94004944
SFFS	0.94624099, 0.94665294, 0.94438723, 0.94561187
SBFS	0.94562307, 0.94521112, 0.94356334, 0.94355171

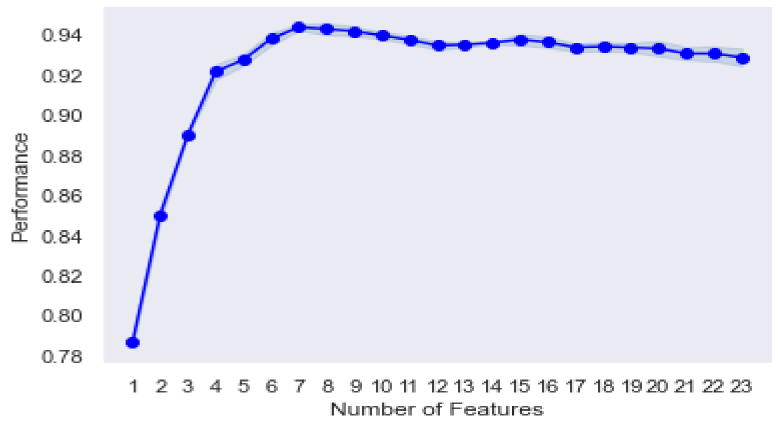
Average CV scores:
'SFS': 0.9439722810282707,
'SBS': 0.9419124639260432,
'SFFS': 0.9457232549879764,
'SBFS': 0.9444873095628182

A plot for the average CV scores was plotted below:

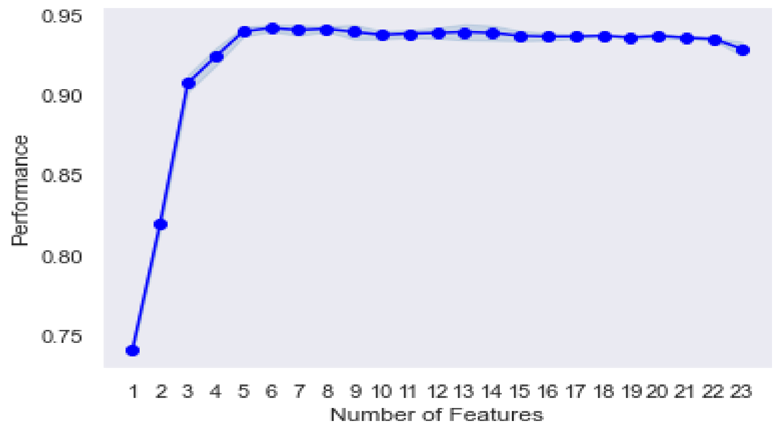


5. The output from the feature selection is visualized in a pandas DataFrame format using the get_metric_dict for all four configurations in the Google Colab Notebook.
6. Plots of the results for each configuration (from mlxtend.plotting import plot_sequential_feature_selection)

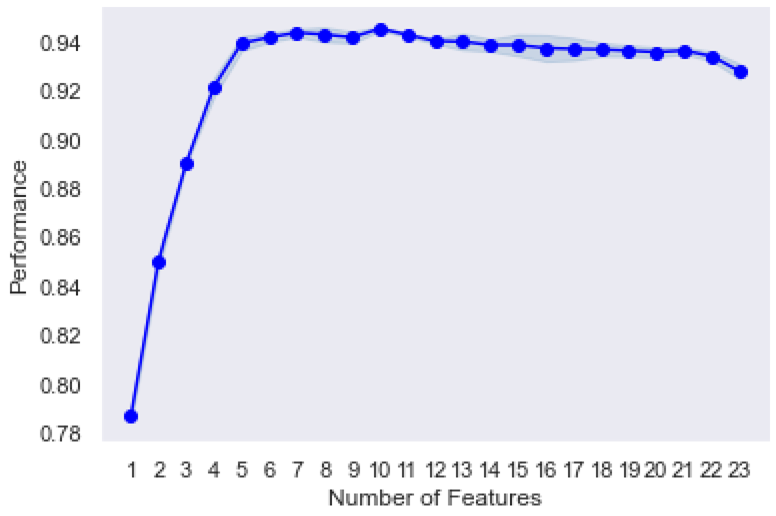
   SFS -



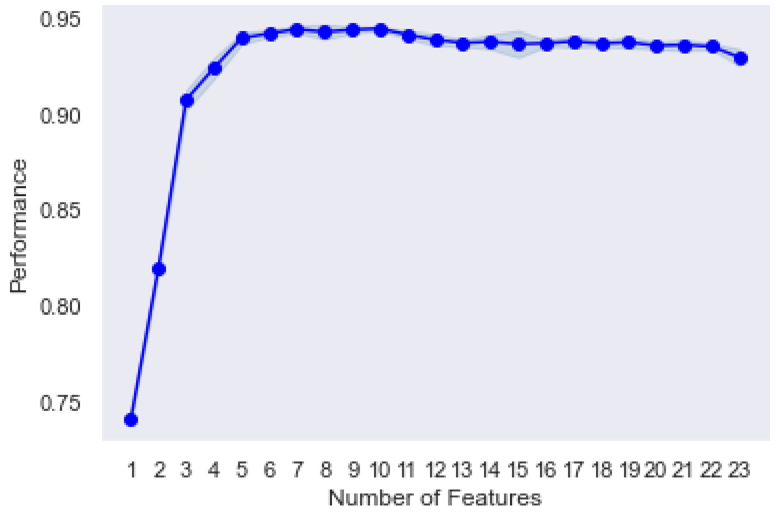
   SBS -



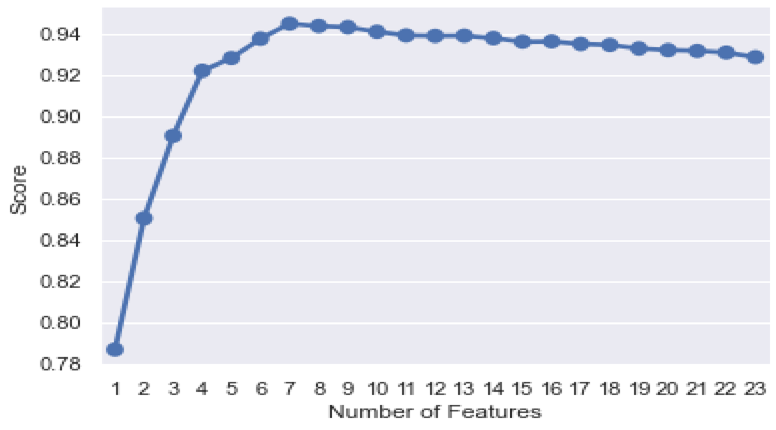
SFFS -



SBFS -



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

Number of features: 16
k_features: (1, 2, 4, 5, 7, 8, 9, 10, 12, 13, 14, 15, 17, 18, 19, 20)
Score: 0.9362482637751555

Number of features: 17
k_features: (1, 2, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 17, 18, 19, 20, 22)
Score: 0.9350121326874945

Number of features: 18
k_features: (2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 19, 20, 21, 22)
Score: 0.9345485603224916

Number of features: 19
```

```
k_features: (0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20)
Score: 0.9328488303473936

Number of features: 20
k_features: (1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
22)
Score: 0.9320766429388131

Number of features: 21
k_features: (1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22)
Score: 0.9316647097530681

Number of features: 22
k_features: (0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22)
Score: 0.9308404720675008

Number of features: 23
k_features: (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22)
Score: 0.9286262864375147

Best result was found for the following permutation:
Number of features: 7
k_features: (2, 4, 5, 7, 10, 12, 19)
Score: 0.9448992717735261
```

- ***Piyush Arora (B20CS086)***