[1]:
```
import h5py
import matplotlib.pyplot as plt
import numpy as np
import tensorflow.keras as K
import pandas as pd
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
import pandas as pd
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R)
SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math Kernel
Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(R) AVX)
instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R)
SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math Kernel
Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(R) AVX)
instructions.
```

```
2024-04-14 01:18:28.522266: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: SSE4.1 SSE4.2, in other operations,
rebuild TensorFlow with the appropriate compiler flags.
```

1 Loading The Data

[2]:
```
with h5py.File("info/data/output_signal.h5", "r") as file:
    signal_data = file["events"][:]

with h5py.File("info/data/output_bg.h5", "r") as file:
    bg_data = file["events"][:]
```

2 Subset of the Data

[3]:
```
# storing signal and background data in panda DataFrame
signal = pd.DataFrame(signal_data)
background = pd.DataFrame(bg_data)

# concatenating the data frames to be part of one big set
df = pd.concat([signal, background])

# reseting the indicies
df = df.reset_index()

# creating the labels for the data sets i.e; signal = 1, background = 0 for␣
↪classification
labels = np.concatenate([np.ones(signal.shape[0]), np.zeros(background.shape[0])])
labels = pd.DataFrame({'ttZ': labels})

# adding labels as a column at the end of the DataFrame
df = df.join(labels)
```

```
# shuffling the DataFrame
df_shuffled = df.sample(frac=1, random_state=42)  # 'random_state' for reproducibility
df_shuffled.head()
```

[3]:

	index	jet_1_pt	jet_2_pt	jet_3_pt	jet_1_eta	jet_2_eta
469349	469349	110.017174	89.244843	42.692078	-1.638234	-1.114363
8248	8248	183.934067	125.170509	93.043922	0.185653	-1.264277
699594	190559	195.334396	155.822617	39.977455	-0.860027	1.231296
149760	149760	165.566879	136.009689	124.258598	0.903266	-0.343321
72006	72006	128.334076	54.510422	45.616039	-0.975434	-2.489865

	jet_3_eta	jet_1_twb	jet_2_twb	jet_3_twb	bjet_1_pt	lep_1_pt
469349	0.334825	3	1	1	110.017174	165.685593
8248	1.190622	1	4	1	125.170509	68.110207
699594	2.197089	1	1	1	35.366051	184.187210
149760	0.041760	5	5	1	165.566879	113.711166
72006	-1.147104	1	1	1	32.256153	109.551361

	lep_2_pt	lep_3_pt	n_jets	n_bjets	n_leptons	met_met
469349	81.492973	32.393501	3	1	3	64.430573
8248	35.741417	12.237628	4	1	3	104.903961
699594	82.709946	36.546856	5	1	3	48.988052
149760	112.550560	34.406342	6	2	3	172.116821
72006	100.125435	48.681091	6	2	3	29.693062

	H_T	ttZ
469349	585.956726	1.0
8248	654.210388	1.0
699594	807.437683	0.0
149760	1007.604309	1.0
72006	606.260437	1.0

[4]:
```
# taking the first 30 000 rows from the shuffled DataFrame
subset_df = df_shuffled.iloc[:30000]

# splitting the labels from the rest of the dataset
X_sub = subset_df[['jet_1_pt', 'jet_2_pt', 'jet_3_pt', 'jet_1_eta', 'jet_2_eta',
       'jet_3_eta', 'jet_1_twb', 'jet_2_twb', 'jet_3_twb', 'bjet_1_pt',
       'lep_1_pt', 'lep_2_pt', 'lep_3_pt', 'n_jets', 'n_bjets', 'n_leptons',
       'met_met', 'H_T']]
y_sub = subset_df['ttZ']

# we can check the class distribution in the subset
print('ttZ events: {:.2f}%'.format(np.sum(y_sub)/len(y_sub) * 100))
print('WZ events: {:.2f}%'.format((1 - np.sum(y_sub)/len(y_sub)) * 100))
```

```
ttZ events: 68.69%
WZ events: 31.31%
```

3 Sequential Feature Selector

https://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/#example-9-selecting-the-best-feature-combination-in-a-k-range

[5]:
```
# model for classification
def build_model(input_dim=None):
    model = K.Sequential([
        K.layers.Normalization(),
        K.layers.Dense(50, activation="relu", input_dim=input_dim),
        K.layers.Dense(25, activation="relu"),
        K.layers.Dense(10, activation="relu"),
        K.layers.Dense(1, activation="sigmoid")
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# wrapper function
model = KerasClassifier(build_fn=lambda: build_model(input_dim=X_sub.shape[1]),␣
 ↪epochs=10, batch_size=32, verbose=0)
```

```
/var/folders/3x/lv7sddxn2gg8mq0dwdcn1wg40000gn/T/ipykernel_29259/2265007354.py:1
3: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras
(https://github.com/adriangb/scikeras) instead. See
https://www.adriangb.com/scikeras/stable/migration.html for help migrating.
  model = KerasClassifier(build_fn=lambda:
build_model(input_dim=X_sub.shape[1]), epochs=10, batch_size=32, verbose=0)
```

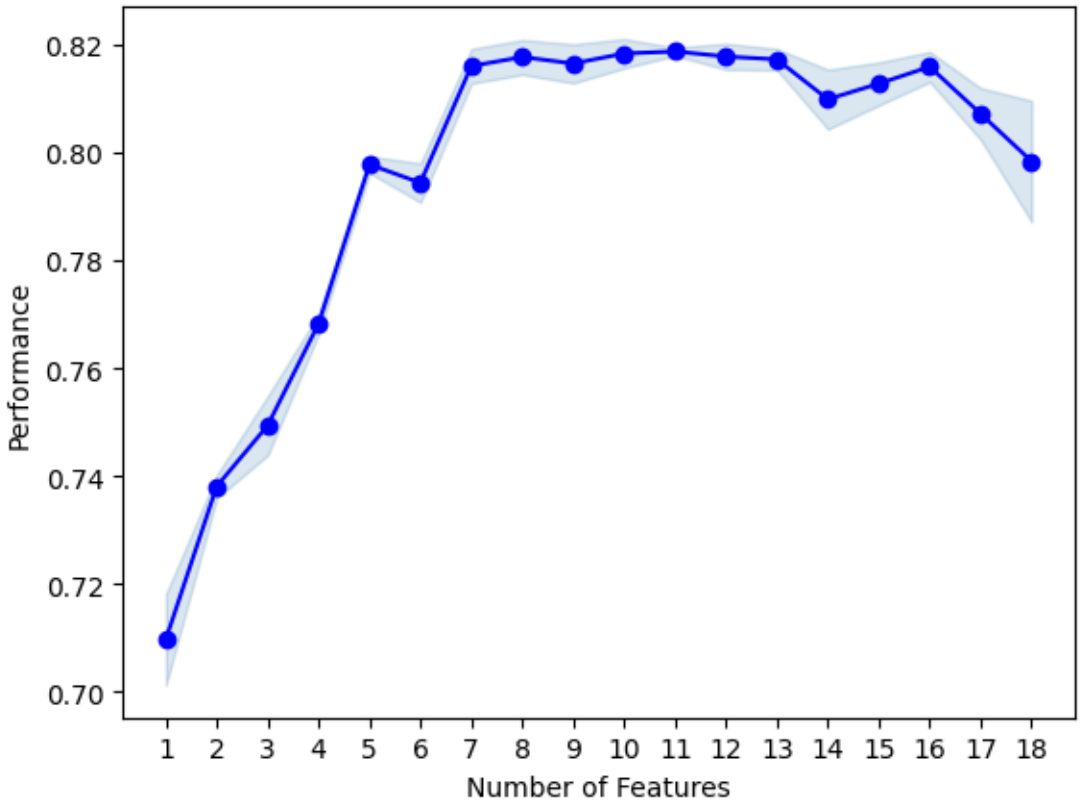
[6]:
```
# SFS model, will check performance from 1 feature to all 18
sfs = SFS(model,
          k_features = (1,18),
          forward=True,
          floating=False,
          scoring='accuracy',
          cv=5,
          verbose=0)

sfs = sfs.fit(X_sub, y_sub)
```

[7]:
```
# plotting performance improvement with each additional feature

print('best combination (ACC: %.3f): %s\n' % (sfs.k_score_, sfs.k_feature_idx_))
plot_sfs(sfs.get_metric_dict(), kind='std_err');
plt.savefig('feature_num_performance_30k.pdf')
```

```
best combination (ACC: 0.819): (0, 2, 3, 5, 6, 7, 8, 9, 13, 14, 15)
```



[8]:
```
# saving performance of all feature sets

feature_performance_df = pd.DataFrame.from_dict(sfs.get_metric_dict()).T
feature_performance_df.to_csv('performance_30k.csv', index=False)
feature_performance_df
```

[8]:

	feature_idx	\
1	(9,)	
2	(9, 13)	
3	(9, 13, 14)	
4	(6, 9, 13, 14)	
5	(6, 7, 9, 13, 14)	
6	(2, 6, 7, 9, 13, 14)	
7	(2, 6, 7, 8, 9, 13, 14)	
8	(2, 5, 6, 7, 8, 9, 13, 14)	
9	(2, 3, 5, 6, 7, 8, 9, 13, 14)	
10	(2, 3, 5, 6, 7, 8, 9, 13, 14, 15)	
11	(0, 2, 3, 5, 6, 7, 8, 9, 13, 14, 15)	
12	(0, 2, 3, 5, 6, 7, 8, 9, 10, 13, 14, 15)	
13	(0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 14, 15)	
14	(0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 14, 15, 17)	
15	(0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 14, 15, 16...	
16	(0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15...	
17	(0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	

```
18  (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...

                                              cv_scores avg_score  \
1   [0.7248333333333333, 0.6886666666666666, 0.721...  0.709633
2   [0.7351666666666666, 0.7353333333333333, 0.735...  0.738067
3   [0.7548333333333334, 0.765, 0.7485, 0.747, 0.7...  0.749367
4   [0.765, 0.7721666666666667, 0.7733333333333333...  0.768233
5   [0.7918333333333333, 0.8003333333333333, 0.8, ...    0.7978
6   [0.7828333333333334, 0.8046666666666666, 0.794...    0.7944
7   [0.8116666666666666, 0.8283333333333334, 0.810...  0.816067
8   [0.8115, 0.8275, 0.8211666666666667, 0.81, 0.8...  0.817733
9   [0.815, 0.8263333333333334, 0.812, 0.8065, 0.8...  0.816533
10  [0.8146666666666667, 0.8246666666666667, 0.811...    0.8184
11  [0.8196666666666667, 0.8185, 0.817, 0.8175, 0...  0.818767
12  [0.8168333333333333, 0.8203333333333334, 0.816...  0.817833
13  [0.8201666666666667, 0.8226666666666667, 0.811...    0.8173
14  [0.8206666666666667, 0.8223333333333334, 0.810...    0.8099
15             [0.8215, 0.82, 0.8055, 0.8155, 0.8015]    0.8128
16  [0.817, 0.8243333333333334, 0.811, 0.808666666...  0.815933
17  [0.8131666666666667, 0.789, 0.806, 0.815333333...    0.8072
18  [0.7715, 0.8163333333333334, 0.8115, 0.8218333...    0.7984

                                       feature_names  ci_bound   std_dev  \
1                                        (bjet_1_pt,)  0.021713  0.016894
2                                  (bjet_1_pt, n_jets)  0.005534  0.004306
3                            (bjet_1_pt, n_jets, n_bjets)  0.014078  0.010953
4                 (jet_1_twb, bjet_1_pt, n_jets, n_bjets)  0.004797  0.003732
5   (jet_1_twb, jet_2_twb, bjet_1_pt, n_jets, n_bj...  0.003981  0.003097
6   (jet_3_pt, jet_1_twb, jet_2_twb, bjet_1_pt, n_...  0.009342  0.007268
7   (jet_3_pt, jet_1_twb, jet_2_twb, jet_3_twb, bj...  0.008231  0.006404
8   (jet_3_pt, jet_3_eta, jet_1_twb, jet_2_twb, je...  0.008258  0.006425
9   (jet_3_pt, jet_1_eta, jet_3_eta, jet_1_twb, je...  0.009256  0.007201
10  (jet_3_pt, jet_1_eta, jet_3_eta, jet_1_twb, je...  0.006957  0.005413
11  (jet_1_pt, jet_3_pt, jet_1_eta, jet_3_eta, jet...  0.001938  0.001508
12  (jet_1_pt, jet_3_pt, jet_1_eta, jet_3_eta, jet...  0.006169  0.004799
13  (jet_1_pt, jet_3_pt, jet_1_eta, jet_2_eta, jet...  0.005066  0.003942
14  (jet_1_pt, jet_3_pt, jet_1_eta, jet_2_eta, jet...  0.014196  0.011045
15  (jet_1_pt, jet_3_pt, jet_1_eta, jet_2_eta, jet...  0.010215  0.007947
16  (jet_1_pt, jet_3_pt, jet_1_eta, jet_2_eta, jet...  0.007186  0.005591
17  (jet_1_pt, jet_3_pt, jet_1_eta, jet_2_eta, jet...  0.012362  0.009618
18  (jet_1_pt, jet_2_pt, jet_3_pt, jet_1_eta, jet_...  0.028888  0.022476

     std_err
1   0.008447
2   0.002153
3   0.005477
4   0.001866
5   0.001549
6   0.003634
7   0.003202
8   0.003213
9   0.003601
10  0.002706
```

```
11  0.000754
12    0.0024
13  0.001971
14  0.005523
15  0.003974
16  0.002796
17  0.004809
18  0.011238
```