SFS WRAPPER, STRATIFIED 5 FOLD USING KNN CLASSIFIER

In [1]:

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
data = pd.read_csv('mushrooms1.csv')
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.naive_bayes import MultinomialNB
from sklearn import svm
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
```

In [2]:
```
1 data#----------INITIAL DATA
```

Out[2]:

	cap-shape	Cap surface	cap-color	bruises	gill-attachment	gill-spacing	gill-size	gill-color	stalk-shape	stalk-root	...	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	p
0	Convex	Smooth	n	Bruises	f	c	n	k	e	e	...	w	w	p	w	o	p	k	
1	Convex	Smooth	y	Bruises	f	c	b	k	e	c	...	w	w	p	w	o	p	n	
2	Bell	Smooth	w	Bruises	f	c	b	n	e	c	...	w	w	p	w	o	p	n	
3	Convex	Scaly	w	Bruises	f	c	n	n	e	e	...	w	w	p	w	o	p	k	
4	Convex	Smooth	g	NoBruises	f	w	b	k	t	e	...	w	w	p	w	o	e	n	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
8119	Knobbed	Smooth	n	NoBruises	a	c	b	y	e	?	...	o	o	p	o	o	p	b	
8120	Convex	Smooth	n	NoBruises	a	c	b	y	e	?	...	o	o	p	n	o	p	b	
8121	Flat	Smooth	n	NoBruises	a	c	b	n	e	?	...	o	o	p	o	o	p	b	
8122	Knobbed	Scaly	n	NoBruises	f	c	n	b	t	?	...	w	w	p	w	o	e	w	
8123	Convex	Smooth	n	NoBruises	a	c	b	y	e	?	...	o	o	p	o	o	p	o	

8124 rows × 22 columns

In [3]:
```
1 data_n = data.to_numpy()
2 print(data_n[:,0])
```

```
['Convex' 'Convex' 'Bell' ... 'Flat' 'Knobbed' 'Convex']
```

In [4]:
```
cols = ['cap-shape', 'Cap surface','cap-color','bruises','gill-attachment', 'gill-spacing','gill-size','gill-color',
data[cols] = data[cols].apply(lambda x: pd.factorize(x)[0] + 1) #--------FACTORIZING THE DATA

```

In [5]:
```
Data_preprocessed = pd.DataFrame(data)#---------CONVERTING TO NUMPY DATASET
data_nnp = data.to_numpy()
```

In [6]:
```
data_nnp[1,:]
```

Out[6]:
```
array([1, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2],
      dtype=int64)
```

In [ ]:
```

```

STARTING THE STRATIFIED K FOLD FROM HERE

In [7]:
```
data_c1 = []#-------------LIST WITH POSITIVE OUTPUT CLASS
data_c2 = []#--------------------------- LIST WITH NEGATIVE OUTPUT CLASS
data_c1.clear()
data_c2.clear()
for i in range(len(data_nnp)):
    if (data_nnp[i,21] == 1):#-------------------COUNTING NUMBER OF POSITIVE AND NEGATIVE CLASEES
        data_c1.append(data_nnp[i,:])

    if(data_nnp[i,21] == 2):
        data_c2.append(data_nnp[i,:])


```

In [8]:
```
data_c2 = np.asarray(data_c2)
data_c1 = np.asarray(data_c1)

```

In [9]:
```
print(data_c1.shape)
print(data_c2.shape)

data_c1 = np.delete(data_c1, (0), axis=0)
data_c2 = np.delete(data_c2, (0,1,2,3,4,5,6,7), axis=0)

print(data_c1.shape)#------------resizing the data to be divisiable by 5, BECAUSE DATA WILL BE DEVIDED IN 5 PARTS I
print(data_c2.shape)
```

```
(3916, 22)
(4208, 22)
(3915, 22)
(4200, 22)
```

In [10]:
```
data_c1_split=np.split(data_c1, 5)
data_c2_split = np.split(data_c2, 5)#---- SPLITTING BOTH CLASSES INTO 5 PARTS
```

In [11]:
```
x = data_c2_split + data_c1_split
len(x)
```

Out[11]:
```
10
```

In [12]:
```
print(data_c2_split[0].shape)
print(data_c1_split[0].shape)

```

```
(840, 22)
(783, 22)
```

In [13]:
```
data_merged = []
data_merged.clear()#-----------merging both pos and neg classes
for i in range(5):
    data_merged.append(np.concatenate((data_c1_split[i], data_c2_split[i]), axis=0) )
```

In [14]:
```
# len(data_merged)
print(data_merged[0].shape)#----------merged classes which is 840+783
data_merged[2].shape
a=np.concatenate((data_merged[0], data_merged[1],data_merged[2]), axis=0) # concatinating all 5 parts of folds

```

```
(1623, 22)
```

In [15]:
```
#X_train1, X_test1, y_train, y_test = train_test_split(data_nnp, y,test_size=0.5)
```

In [16]:
```

for j in range(5):
    np.random.shuffle(data_merged[j])#------now shuffling the values within 5 parts
```

SFS FOR SINGLE FEATURE AT A TIME AND CALCULATING ACCURACY WITH EACH FEATURE SEPERATELY

In [17]:

```
all_list = []
all_list.clear()
def SFS_single(x_train, y_train, x_test, y_test):#---- starting sfs from here, it will apply classifier for each fea

    output=0
    old_output=0
    ave_output=0

    for m in range(len(testing_set[1,:])):

        d = x_train[:,m].reshape(-1,1)
        d_t = x_test[:,m].reshape(-1,1)




#         clf = MultinomialNB()
#         clf.fit(d, y_train)

#         predicted = clf.predict(X_train)
#         # print(predicted[5000:6000])
#         print(clf.score( X_test, y_test))


        #clf = LogisticRegression(random_state=0, solver='lbfgs',multi_class='multinomial').

        clf = KNeighborsClassifier(n_neighbors=5)
        clf.fit(d, y_train)
        clf.predict(d_t)
        output = clf.score(d_t,y_test)




        print("Accuracy with KNNclassifier with features number", + m ,output)
        ave_output = (old_output+output)/2
        #print("Average",+ave_output)
        old_output = output
        ave_output = (old_output+output)/2
        #print("Average",+ave_output)
        all_list.append(output)
```

```
42
43
44
45
46
```

CALLING SFS SINGLE FUNCTION WITHIN STRATIFIED 5 FOLDS.

In [18]:

```
def concat_list(training_set,size=4):
    my_arr = training_set[0]
    for i in range(1,size):
        my_arr = np.concatenate((my_arr, training_set[i]), axis=0)
    return my_arr
    #-----------------5 Stratifieid K fold
l= 0

for i in range(5):#-------------a loop for 5 stratified k fold
    training_set = []
    #training_set.clear()
    testing_set = []
    testing_set =data_merged[i]
    for j in range(5):
        if i !=j:
            training_set.append(data_merged[i])
    training_set = concat_list(training_set,size=4)

    training_set_labels = training_set[:,21]
    training_set = training_set[:,0:21]
    testing_set_labels = testing_set[:,21]
    testing_set = testing_set[:,0:21]
    print("when test at in k fold-----------------------------------", +i)




    SFS_single(training_set,training_set_labels,testing_set,testing_set_labels)#----------calling sfs fucntion
```

```
Accuracy with KNNclassifier with features number 0 0.7017868145409735
Accuracy with KNNclassifier with features number 1 0.6266173752310537
Accuracy with KNNclassifier with features number 2 0.7270486752926679
Accuracy with KNNclassifier with features number 3 0.6370918052988294
Accuracy with KNNclassifier with features number 4 0.5902649414664202
```

```
Accuracy with KNNclassifier with features number 5 0.6833025261860751
Accuracy with KNNclassifier with features number 6 0.9217498459642637
Accuracy with KNNclassifier with features number 7 0.9741219963031423
Accuracy with KNNclassifier with features number 8 0.9568699938385705
Accuracy with KNNclassifier with features number 9 0.5576093653727665
Accuracy with KNNclassifier with features number 10 0.5009242144177449
Accuracy with KNNclassifier with features number 11 0.5335797905113987
Accuracy with KNNclassifier with features number 12 0.7652495378927912
Accuracy with KNNclassifier with features number 13 0.7677141096734442
Accuracy with KNNclassifier with features number 14 0.4824399260628466
Accuracy with KNNclassifier with features number 15 0.5224892174984597
Accuracy with KNNclassifier with features number 16 0.807147258163894
Accuracy with KNNclassifier with features number 17 0.8539741219963032
Accuracy with KNNclassifier with features number 18 0.5175600739371534
Accuracy with KNNclassifier with features number 19 0.8539741219963032
```

In [19]:

```
print(len(all_list))
average_list = []
average_list.clear()#-----------calculating average of all features after 5 k folds
for i in range(len(testing_set[1,:])):
    a = (all_list[i] + all_list[i+21] + all_list[i+42] + all_list[i+63] + all_list[i+84])/5
    average_list.append([i,a])

```

```
105
```

In [20]:

```
print(average_list) #--------------average list of features
sorted_average_list = sorted(average_list,reverse=True, key=lambda x : x[1])#
```

```
[[0, 0.5704251386321626], [1, 0.6373382624768947], [2, 0.7133703019100432], [3, 0.6723351817621689], [4, 0.525200246457
178], [5, 0.622550831792976], [6, 0.7122612446087493], [7, 0.8353666050523723], [8, 0.736290819470117], [9, 0.788170055
4528651], [10, 0.7466420209488601], [11, 0.7519408502772643], [12, 0.6833025261860752], [13, 0.6660505237215034], [14,
0.5035120147874307], [15, 0.5115218730745532], [16, 0.577449168207024 1], [17, 0.8521256931608134], [18, 0.8480591497227
357], [19, 0.7567467652495379], [20, 0.7426987060998151]]
```

In [21]:
```
#print(sorted_average_list)#-------sorted average list
d = np.asarray(sorted_average_list)#--------sorted list of features, 17th feature has the greateast accuracy
print(d)
```

```
[[17.         0.85212569]
 [18.         0.84805915]
 [ 7.         0.83536661]
 [ 9.         0.78817006]
 [19.         0.75674677]
 [11.         0.75194085]
 [10.         0.74664202]
 [20.         0.74269871]
 [ 8.         0.73629082]
 [ 2.         0.7133703 ]
 [ 6.         0.71226124]
 [12.         0.68330253]
 [ 3.         0.67233518]
 [13.         0.66605052]
 [ 1.         0.63733826]
 [ 5.         0.62255083]
 [16.         0.57744917]
 [ 0.         0.57042514]
 [ 4.         0.52520025]
 [15.         0.51152187]
 [14.         0.50351201]]
```

GOT THE INDEX OF MOST IMPORTANT FEATURES

In [22]:
```
sorted_average_list= np.asarray(sorted_average_list, dtype=np.int)#-------now this is the index of features which hi
print(sorted_average_list)

```

```
[[17  0]
 [18  0]
 [ 7  0]
 [ 9  0]
 [19  0]
 [11  0]
 [10  0]
 [20  0]
 [ 8  0]
 [ 2  0]
 [ 6  0]
 [12  0]
 [ 3  0]
 [13  0]
 [ 1  0]
 [ 5  0]
 [16  0]
 [ 0  0]
 [ 4  0]
 [15  0]
 [14  0]]
```

USING THESE INDEX OF FEATURES AND ADDING FEATAURES ONE BY ONE TO GET ACCCURACY. IF THE PERFORMANCE IS SAME OR DECREASE THE LOOP WILL BREAK

In [23]:

```
sfs_feature = []
sfs_feature_t = []
def SFS(X_train, y_train, X_test, y_test):#--------------in this fucntion each feature will added to the list contin
    i=1
    old_output = 0
    for i in range(len(sorted_average_list)):


        e = X_train[:,sorted_average_list[i,0]]#------for training data
        sfs_feature.append(e)
        b = np.array(sfs_feature)
        d = np.transpose(b)
       # print(d.shape)




        e_t = X_test[:,sorted_average_list[i,0]]
        sfs_feature_t.append(e_t)
        b_t = np.array(sfs_feature_t)
        d_t = np.transpose(b_t)
       # print(d_t.shape)


        #print("x_train_shape", + d.shape)

#         clf = RandomForestClassifier(n_estimators=400, max_depth=2)
#         clf.fit(d, y_train)
#         output = clf.score(d_t,y_test)
#         print("Accuracy with random forest adding features", + i ,output)

        #clf = LogisticRegression(random_state=0, solver='lbfgs',multi_class='multinomial')

        clf = KNeighborsClassifier(n_neighbors=5)
        clf.fit(d, y_train)
        clf.predict(d_t)
        output = clf.score(d_t,y_test)


        if(output<=old_output):
            print("Stopping because next accuracy is lower or same--------------------------")
```

```
            break

        else:
            print("Accuracy with KNN classifier adding features", + i ,output)
            old_output = output
```

PUTTING SFS FUNCTION IN STRATIFIED 5 FOLD LOOP

In [24]:

```
def concat_list(training_set,size=4):
    my_arr = training_set[0]
    for i in range(1,size):#---------------again using stratified k fold to calcualte accuracy with with each featur
        my_arr = np.concatenate((my_arr, training_set[i]), axis=0)
    return my_arr
    #-----------------5 Stratifieid K fold
l= 0

for i in range(5):#-------loop for stratifeid k fold
    training_set = []
    #training_set.clear()
    testing_set = []
    testing_set =data_merged[i]
    for j in range(5):
        if i !=j:
            training_set.append(data_merged[i])
    training_set = concat_list(training_set,size=4)

    training_set_labels = training_set[:,21]
    training_set = training_set[:,0:21]
    testing_set_labels = testing_set[:,21]
    testing_set = testing_set[:,0:21]
    print("WHEN K FOLD TEST AT", +i)


    SFS(training_set,training_set_labels,testing_set,testing_set_labels)#--------calling the sfs function
```

```
WHEN K FOLD TEST AT 0
Accuracy with KNN classifier adding features 0 0.7288971041281578
Stopping because next accuracy is lower or same----------------------------
WHEN K FOLD TEST AT 1
Accuracy with KNN classifier adding features 0 0.9470117067159581
```

```
Accuracy with KNN classifier adding features 1 0.9913739987677141
Stopping because next accuracy is lower or same--------------------------
WHEN K FOLD TEST AT 2
Accuracy with KNN classifier adding features 0 0.7375231053604436
Accuracy with KNN classifier adding features 1 0.9993838570548367
Stopping because next accuracy is lower or same--------------------------
WHEN K FOLD TEST AT 3
Accuracy with KNN classifier adding features 0 0.9963031423290203
Accuracy with KNN classifier adding features 1 0.9987677141096735
Accuracy with KNN classifier adding features 2 1.0
Stopping because next accuracy is lower or same--------------------------
WHEN K FOLD TEST AT 4
Accuracy with KNN classifier adding features 0 0.9772027110289587
Accuracy with KNN classifier adding features 1 0.977818853974122
Accuracy with KNN classifier adding features 2 1.0
Stopping because next accuracy is lower or same--------------------------
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```