**----------------------------SBS ----------------EXERCISE#1-----------------------------------------**

from sklearn.base import clone

from itertools import combinations

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

class SBS():

def \_\_init\_\_(self, estimator, k\_features,scoring=accuracy\_score,

test\_size=0.25, random\_state=1):

self.scoring = scoring

self.estimator = clone(estimator)

self.k\_features = k\_features

self.test\_size = test\_size

self.random\_state = random\_state

def fit(self, X, y):

X\_train, X\_test, y\_train, y\_test = \

train\_test\_split(X, y, test\_size=self.test\_size,

random\_state=self.random\_state)

dim = X\_train.shape[1]

self.indices\_ = tuple(range(dim))

self.subsets\_ = [self.indices\_]

score = self.\_calc\_score(X\_train, y\_train,

X\_test, y\_test, self.indices\_)

self.scores\_ = [score]

while dim > self.k\_features:

scores = []

subsets = []

for p in combinations(self.indices\_, r=dim-1):

score = self.\_calc\_score(X\_train, y\_train,

X\_test, y\_test, p)

scores.append(score)

subsets.append(p)

best = np.argmax(scores)

self.indices\_ = subsets[best]

self.subsets\_.append(self.indices\_)

dim -= 1

self.scores\_.append(scores[best])

self.k\_score\_ = self.scores\_[-1]

return self

def transform(self, X):

return X[:, self.indices\_]

def \_calc\_score(self, X\_train, y\_train,

X\_test, y\_test, indices):

self.estimator.fit(X\_train[:, indices], y\_train)

y\_pred = self.estimator.predict(X\_test[:, indices])

score = self.scoring(y\_test, y\_pred)

return score

## LOAD Data SET

import numpy as np

import pandas as pd

df\_wine = pd.read\_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data', header=None)

df\_wine.columns = ['Class label', 'Alcohol',

'Malic acid', 'Ash',

'Alcalinity of ash', 'Magnesium',

'Total phenols', 'Flavanoids',

'Nonflavanoid phenols',

'Proanthocyanins',

'Color intensity', 'Hue',

'OD280/OD315 of diluted wines',

'Proline']

from sklearn.model\_selection import train\_test\_split

X, y = df\_wine.iloc[:, 1:].values, df\_wine.iloc[:, 0].values

X\_train, X\_test, y\_train, y\_test = \

train\_test\_split(X, y, test\_size=0.3, random\_state=0)

#

from sklearn.preprocessing import StandardScaler

stdsc = StandardScaler()

X\_train\_std = stdsc.fit\_transform(X\_train)

X\_test\_std = stdsc.transform(X\_test)

## KNN

from sklearn.neighbors import KNeighborsClassifier

import matplotlib.pyplot as plt

knn = KNeighborsClassifier(n\_neighbors=2)

sbs = SBS(knn, k\_features=1)

sbs.fit(X\_train\_std, y\_train)

##

k\_feat = [len(k) for k in sbs.subsets\_]

plt.plot(k\_feat, sbs.scores\_, marker='o')

plt.ylim([0.7, 1.1])

plt.ylabel('Accuracy')

plt.xlabel('Number of features')

plt.grid()

plt.show()

##

print(sbs.subsets\_)

print(sbs.scores\_)

**---------------------------- ----------------EXERCISE#2-----------------------------------------**

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

##

df\_wine = pd.read\_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data', header=None)

df\_wine.columns = ['Class label', 'Alcohol',

'Malic acid', 'Ash',

'Alcalinity of ash', 'Magnesium',

'Total phenols', 'Flavanoids',

'Nonflavanoid phenols',

'Proanthocyanins',

'Color intensity', 'Hue',

'OD280/OD315 of diluted wines',

'Proline']

##

from sklearn.model\_selection import train\_test\_split

X, y = df\_wine.iloc[:, 1:].values, df\_wine.iloc[:, 0].values

X\_train, X\_test, y\_train, y\_test = \

train\_test\_split(X, y, test\_size=0.3, random\_state=0)

##

from sklearn.ensemble import RandomForestClassifier

feat\_labels = df\_wine.columns[1:]

forest = RandomForestClassifier(n\_estimators=10000,

random\_state=0,

n\_jobs=-1)

forest.fit(X\_train, y\_train)

importances = forest.feature\_importances\_

indices = np.argsort(importances)[::-1]

for f in range(X\_train.shape[1]):

print("%2d) %-\*s %f" % (f + 1, 30,

feat\_labels[f],

importances[indices[f]]))

plt.title('Feature Importances')

plt.bar(range(X\_train.shape[1]),

importances[indices],

color='lightblue',

align='center')

plt.xticks(range(X\_train.shape[1]),

feat\_labels, rotation=90)

plt.xlim([-1, X\_train.shape[1]])

plt.tight\_layout()

plt.show()

---------------------------- ----------------EXERCISE#3-----------------------------------------