```
'''it returns an array of objects containing group and feature list'''
def getDataFromFile(fileName):
    data = []
    file_in = open(fileName, 'r')

    # read every line
    for instance in file_in.readlines():
        features = [] # create a feature list
        obj = {} # create an object to store group and feature information
        instance = instance.strip() # remove leading and trailing spaces

        values = instance.split(' ') # split row based on spaces
        obj['group'] = float(values[0]) # store first value as group

        # read the remainging line and stores every feature in features list
        for feature in values[1:]:
            if(feature == ''):# omit white space
                continue
            features.append(float(feature))

        obj['features'] = features # add feature list to the object
        data.append(obj)

    return data

'''Normalization using Z-Score'''
import statistics
def normalization(data):
    # making deep copy of data
    normalizedData = []
    for instance in data:
        tempObject = {}
        tempObject['group'] = instance.get('group')
        tempObject['features'] = [feature for feature in instance.get('features')]
        normalizedData.append(tempObject)

    # loop through each features and normalize it
    for i in range(len(normalizedData[0].get('features'))):
        featureList = []

        for instance in normalizedData:
            featureList.append(instance.get('features')[i])
```

```
43
44          mean = statistics.mean(featureList)
45          stdev = statistics.stdev(featureList)
46
47          for instance in normalizedData:
48              feature = instance.get('features')[i]
49              feature = (feature - mean) / stdev
50              instance['features'][i] = feature
51
52      return normalizedData
53
54  import math
55  '''it accepts array of objects as training data and just
    list of features for test data'''
56  def nearestNeighbourClassifier(trainingDataSet,
    testInstance):
57      distances = []
58      for trainingInstance in trainingDataSet: #get object
    from array
59          distanceBetweenFeature = float(0)
60          # loop through each feature and calculate distance
61          for trainingFeature, testingFeature in zip(
    trainingInstance.get('features'), testInstance):
62              distanceBetweenFeature += (trainingFeature -
    testingFeature)**2
63
64          # store value of euclidean distance with group
    number as tuples
65          euclideanDistance = math.sqrt(
    distanceBetweenFeature)
66          distances.append((euclideanDistance,
    trainingInstance.get('group')))
67
68      # sort distances and returns the group number
69      distances = sorted(distances)
70      return distances[0][1]
71
72
73  def leaveOneOutValidator(data):
74      correctGuess = 0
75      # select each index to leave as test data
76      for leaveIndex in range(len(data)):
77          testData = data[leaveIndex]
78          if(leaveIndex == 0):
79              trainingDataSet = data[1:]
80          else:
81              # add the remaining instances to the training
    set
82              firstSubset = data[0:leaveIndex]
83              secondSubset = data[leaveIndex+1:]
```

```
 84             trainingDataSet = firstSubset + secondSubset
 85
 86         # call nearest neighbour algo for each set and
    check if it has detected the group correctly
 87         guessedGroup = nearestNeighbourClassifier(
    trainingDataSet, testData.get('features'))
 88         if(guessedGroup == testData.get('group')):
 89             correctGuess = correctGuess+1
 90
 91     # returns the overall percentage
 92     validationScore = correctGuess/len(data)
 93     return  validationScore * 100
 94
 95 '''it returns a set of features for a given indices'''
 96 def extractFeatures(data, featureIndices):
 97     newDataSet = []
 98     for i in range(len(data)):
 99         obj = {}
100         extractedFeatures = []
101         for index in featureIndices:
102             extractedFeatures.append(data[i].get('features
    ')[index])
103
104         obj['group'] = data[i].get('group')
105         obj['features'] = extractedFeatures
106         newDataSet.append(obj)
107
108     return newDataSet
109
110 def forwardSelection(data):
111     print('Beginning Search.\n')
112     featureCount = len(data[0].get('features'))
113
114     # indices of features which improves the model
115     indices = []
116     bestScores = [] # best scores on each iteration
117
118     checkedLocalMaxima = False
119     # maximum number of iteration
120     for i in range(featureCount):
121         best = 0.0
122         newIndex = 0
123
124         # add new feature on each iteration for evaluation
125         for j in range(featureCount):
126             if(j in indices):
127                 continue
128
129             # copy previously selected indices
130             temp = [val for val in indices]
```

```
131             # add new index
132             if(len(temp) != 0):
133                 if(j < temp[0]):
134                     temp.insert(0, j)
135                 else:
136                     temp.append(j)
137             else:
138                 temp.append(j)
139 
140             # extract subset of features and groups
    according to new list of features to evaluate
141             newDataSet = extractFeatures(data, temp)
142             accuracyRate = leaveOneOutValidator(newDataSet
    )
143 
144             # update accuracy rate on every addition of
    features
145             print('\t Using feature(s)' + str([index+1 for
     index in temp]) + ' accuracy is ' + str(accuracyRate))
146             if(accuracyRate > best):
147                 best = accuracyRate
148                 newIndex = j
149 
150         # add the index of selected best feature
151         indices.append(newIndex)
152         indices = sorted(indices)
153 
154         # for for local maxima
155         if(len(bestScores) != 0 and best < bestScores[len(
    bestScores) - 1][0]):
156             if(checkedLocalMaxima):
157                 print('\nAddition of features is not
    improving the model\n')
158                 bestScores = sorted(bestScores)
159                 break
160             else:
161                 print('\n(Warning, Accuracy has decreased
    ! Continuing search in case of local maxima)')
162                 checkedLocalMaxima = True
163         elif(checkedLocalMaxima):
164             checkedLocalMaxima = False
165 
166         tempBestScoreIndices = [index+1 for index in
    indices]
167         if(len(tempBestScoreIndices) != featureCount):
168             print('\nFeature(s) set' + str(
    tempBestScoreIndices) + ' was best, accuracy is ' + str(
    best) + '\n')
169         else:
170             print()
```

```
171         bestScores.append((best, tempBestScoreIndices))
172         bestScores = sorted(bestScores)
173 
174 
175     print('Finished Search!! The best feature subset is '
     + str(bestScores[len(bestScores) - 1][1]) + ', which has
    an accuracy of '+str(bestScores[len(bestScores) - 1][0]))
176 
177 
178 def backwardElimination(data):
179     print('Beginning Search.\n')
180     featureCount = len(data[0].get('features'))
181 
182     indices = [i for i in range(featureCount)] # indices
    of features which improves the model
183 
184     scoreOfAllFeatures = leaveOneOutValidator(data) #
    accuracy when all features are selected
185 
186     bestScores = scoreOfAllFeatures
187     checkedLocalMaxima = False
188 
189     # maximum number of iteration
190     for i in range(featureCount):
191         scoreList = []
192         accuracyImproved = False
193 
194         # remove a feature on each iteration
195         for j in range(len(indices)):
196             temp = [index for index in indices] # copy
    previously selected index list
197             temp.pop(j) # remove a feature
198 
199             # get features based on the indices of new
    features
200             newDataSet = extractFeatures(data, temp)
201             accuracyRate = leaveOneOutValidator(newDataSet
    )
202 
203             # evaluate accuracy and update if needed
204             print('\t Using feature(s)' + str([index + 1
    for index in temp]) + ' accuracy is ' + str(accuracyRate))
205             if (accuracyRate >= bestScores):
206                 bestScores = accuracyRate
207                 accuracyImproved = True
208 
209             scoreList.append((accuracyRate, temp))
210 
211         # check for local maxima
212         if(not accuracyImproved):
```

```
213             if(checkedLocalMaxima):
214                 print('\nAddition of features is not
    improving the model\n')
215                 break
216             else:
217                 print('\n(Warning, Accuracy has decreased
    ! Continuing search in case of local maxima)')
218                 checkedLocalMaxima = True
219 
220         scoreList = sorted(scoreList)
221         indices = scoreList[len(scoreList) - 1][1]
222         print('\nFeature(s) set' + str([index + 1 for
    index in indices]) + ' was best, accuracy is ' + str(
    bestScores) + '\n')
223 
224     print('Finished Search!! The best feature subset is '
     + str([index + 1 for index in indices]) + ', which has an
     accuracy of ' + str(bestScores))
225 
226 
227 
228 def main():
229     print('Welcome to Calvin Ng \'s Feature Selection
    Algorithm')
230     fileName = input('Type in the name of the file to test
     : ')
231 
232     print('\nType the number of the algorithm you want to
    run.\n')
233     print('1)Forward Selection')
234     print('2)Backward Elimination')
235 
236     algoType = input('\t\t\t')
237 
238     data = getDataFromFile(fileName)
239 
240     print('This dataset has '+str(len(data[0].get('
    features')))+' features (not including the class attribute
    ), with '+str(len(data))+' instances.')
241 
242     print('\nPlease wait while I normalize the data…', end
    =' ')
243     normalizedData = normalization(data)
244     print('Done!')
245 
246     accuracyOfAllFeatures = leaveOneOutValidator(
    normalizedData)
247     print('\nRunning nearest neighbor with all '+str(len(
    data[0].get('features')))+' features, using “leaving-one-
    out” evaluation, I get an accuracy of '+str(
```

```
247 accuracyOfAllFeatures)+'%\n')
248
249     if(algoType == '1'):
250         forwardSelection(normalizedData)
251     elif(algoType == '2'):
252         backwardElimination(normalizedData)
253     else:
254         print('Please input correct type')
255
256
257 if __name__ == '__main__':
258     main()
```