Project: Feature Selection

Akshay Kumar, 10 March 2023

Background: In this project, we were tasked with completing the Forward Selection and Backward Elimination searching algorithms. These two algorithms use the nearest neighbor classifier which we were also required to code and wrap inside the searching functions. Files of small and large datasets were given that contained two classes and continuous features. The two searching algorithms were run on both of the assigned datasets and the best feature subset was selected based on the highest accuracy.

Results: In this project, I was tasked with finding the best feature subset for small set 83 and large set 20. Below is Figure 1 which shows the results for running Forward Selection on set 83.

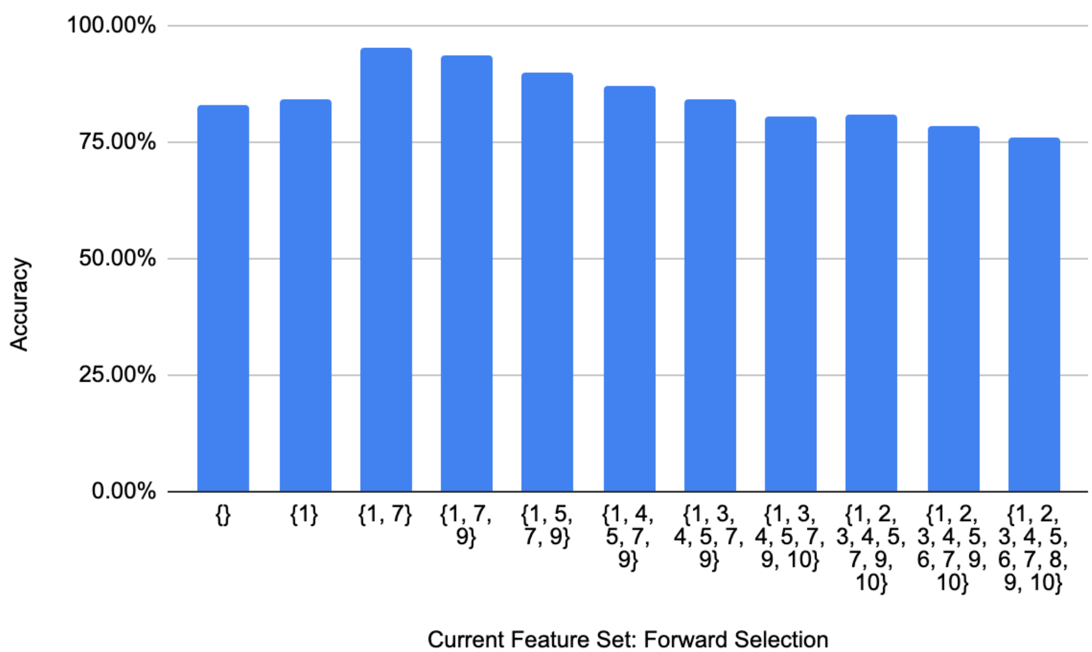


Figure 1: Accuracy of all feature subsets discovered by Forward Selection.

In Figure 1, at the beginning of the search, we have an empty set of features which gives an accuracy of 83.0%. Adding feature “1” improved the accuracy to 84.2%.

After adding feature “7” the accuracy jumped to 95.4%. After this point, the accuracy started to decrease as we added each further feature. For example, when the weak feature “9” was added next, the accuracy decreased to 93.6% for {1, 7, 9}. After all the features were added, the recorded accuracy for the full set was 76.2%.

Next, below is Figure 2 which shows the results for running Backward Elimination on set 83.

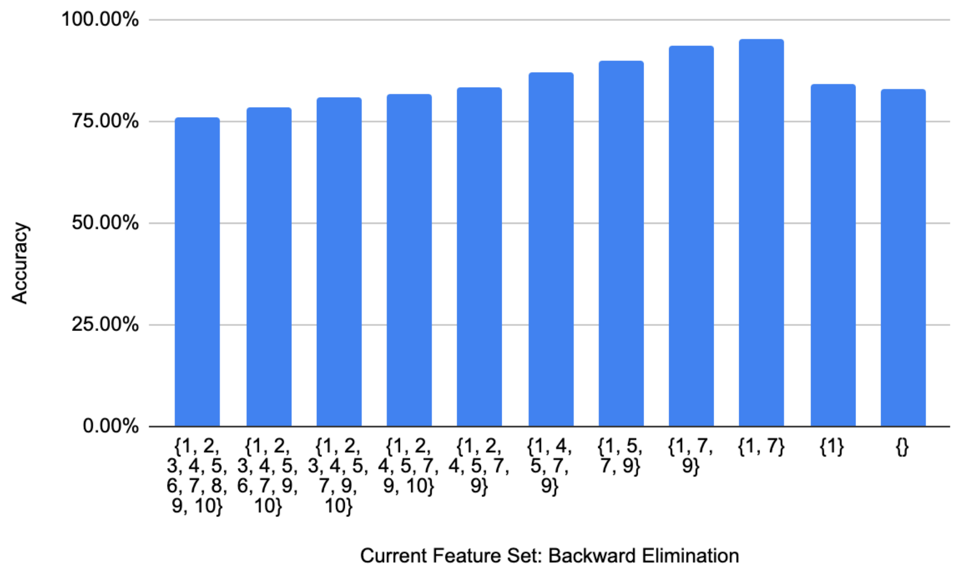


Figure 2: Accuracy of all feature subsets discovered by Backward Elimination.

In Figure 2, at the beginning of the search, we have {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} which gives an accuracy of 76.2%. This matches the accuracy of the last tested subset from performing Forward Selection. Similarly, the empty set at the end of our search gives an accuracy of 83.0% which matches the accuracy derived from performing Forward Selection. Removing each subsequent feature increased the accuracy all the way up to 95.4% when the set became {1, 7}. After this point, removing more features caused the accuracy to dip again with {1} giving 84.2% accuracy.

Small Set Conclusion: It is apparent that the feature subset {1, 7} is the best set for this particular problem. This set gives the highest possible accuracy of 95.4% with the next best set {1, 7, 9} giving an accuracy of 93.6%. It took about 11 seconds to run Forward Selection on this set and about 17 seconds to run Backward Elimination.

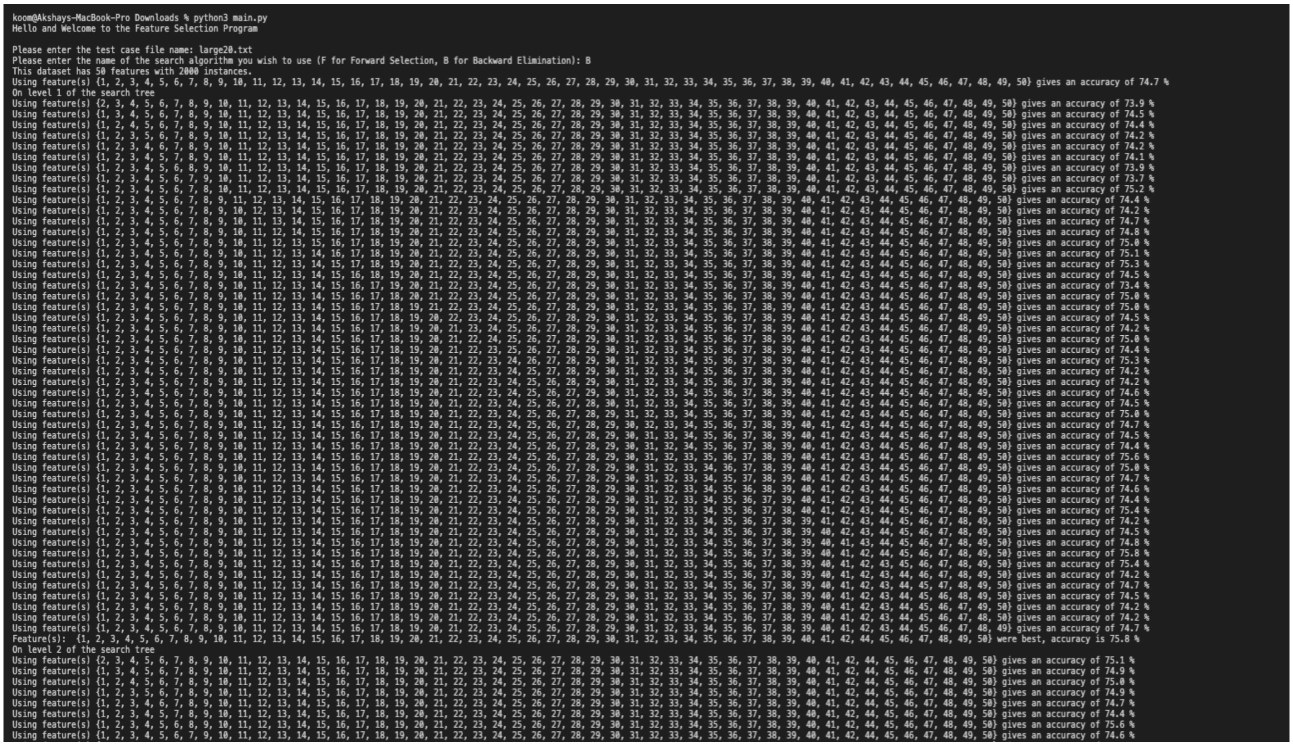
Following small set 83, we now look at large set 20 for which both searches were performed. Below are screenshots which show the truncated search result for running Forward Selection on set 20.

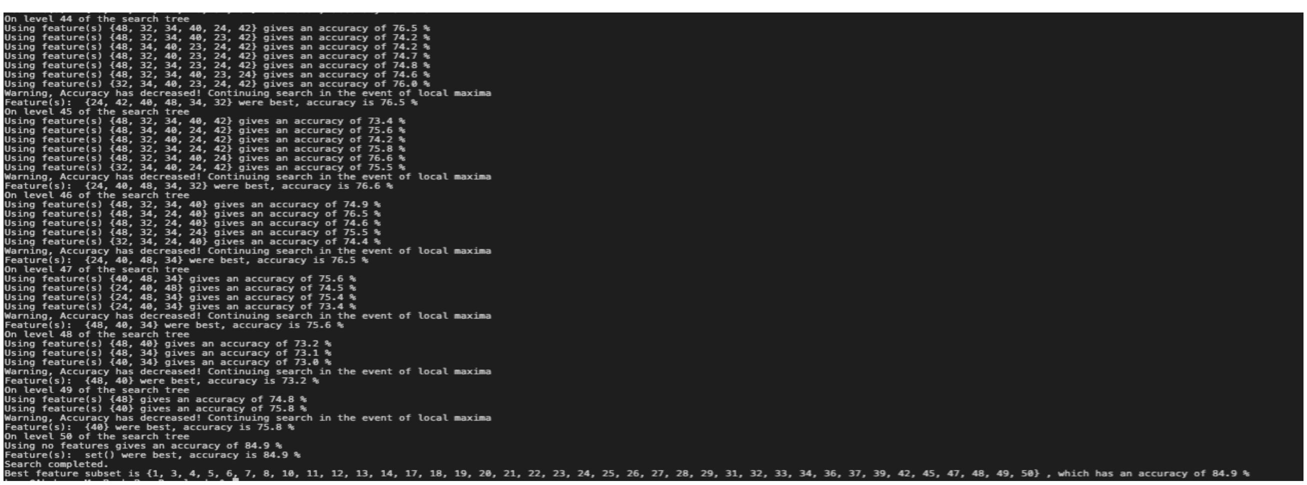
```
Hello and Welcome to the Feature Selection Program

Please enter the test case file name: large20.txt
Please enter the name of the search algorithm you wish to use (F for Forward Selection, B for Backward Elimination): F
This dataset has 50 features with 2000 instances.
Using no features gives an accuracy of 84.9 %
On level 1 of the search tree
Using feature(s) {1} gives an accuracy of 73.8 %
Using feature(s) {2} gives an accuracy of 72.8 %
Using feature(s) {3} gives an accuracy of 73.6 %
Using feature(s) {4} gives an accuracy of 75.3 %
Using feature(s) {5} gives an accuracy of 74.6 %
Using feature(s) {6} gives an accuracy of 74.4 %
Using feature(s) {7} gives an accuracy of 72.5 %
Using feature(s) {8} gives an accuracy of 73.8 %
Using feature(s) {9} gives an accuracy of 74.1 %
Using feature(s) {10} gives an accuracy of 74.2 %
Using feature(s) {11} gives an accuracy of 74.8 %
Using feature(s) {12} gives an accuracy of 73.4 %
Using feature(s) {13} gives an accuracy of 72.5 %
Using feature(s) {14} gives an accuracy of 75.1 %
Using feature(s) {15} gives an accuracy of 86.1 %
Using feature(s) {16} gives an accuracy of 74.1 %
Using feature(s) {17} gives an accuracy of 74.6 %
Using feature(s) {18} gives an accuracy of 74.7 %
Using feature(s) {19} gives an accuracy of 75.6 %
Using feature(s) {20} gives an accuracy of 74.8 %
Using feature(s) {21} gives an accuracy of 73.6 %
Using feature(s) {22} gives an accuracy of 72.8 %
Using feature(s) {23} gives an accuracy of 73.3 %
Using feature(s) {24} gives an accuracy of 72.6 %
Using feature(s) {25} gives an accuracy of 74.6 %
Using feature(s) {26} gives an accuracy of 75.4 %
Using feature(s) {27} gives an accuracy of 75.0 %
Using feature(s) {28} gives an accuracy of 74.9 %
Using feature(s) {29} gives an accuracy of 73.9 %
Using feature(s) {30} gives an accuracy of 75.8 %
Using feature(s) {31} gives an accuracy of 74.7 %
Using feature(s) {32} gives an accuracy of 75.2 %
Using feature(s) {33} gives an accuracy of 76.1 %
Using feature(s) {34} gives an accuracy of 74.5 %
Using feature(s) {35} gives an accuracy of 73.3 %
Using feature(s) {36} gives an accuracy of 74.2 %
Using feature(s) {37} gives an accuracy of 75.0 %
Using feature(s) {38} gives an accuracy of 75.8 %
Using feature(s) {39} gives an accuracy of 76.4 %
Using feature(s) {40} gives an accuracy of 75.8 %
Using feature(s) {41} gives an accuracy of 73.8 %
Using feature(s) {42} gives an accuracy of 75.4 %
Using feature(s) {43} gives an accuracy of 74.2 %
Using feature(s) {44} gives an accuracy of 74.0 %
Using feature(s) {45} gives an accuracy of 72.9 %
Using feature(s) {46} gives an accuracy of 74.5 %
Using feature(s) {47} gives an accuracy of 75.0 %
Using feature(s) {48} gives an accuracy of 74.8 %
Using feature(s) {49} gives an accuracy of 75.1 %
Using feature(s) {50} gives an accuracy of 73.6 %
Feature(s):  {15} were best, accuracy is 86.1 %
On level 2 of the search tree
Using feature(s) {1, 15} gives an accuracy of 86.0 %
Using feature(s) {2, 15} gives an accuracy of 85.1 %
Using feature(s) {3, 15} gives an accuracy of 86.1 %
```

```
racy of 75.1 %
Warning, Accuracy has decreased! Continuing search in the event of local maxima
Feature(s):  {1, 2, 3, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 39, 40, 41, 42, 44, 45, 46, 47, 48, 49, 50} were best, accura
cy is 76.1 %
On level 47 of the search tree
Using feature(s) {1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 39, 40, 41, 42, 44, 45, 46, 47, 48, 49, 50} gives an a
ccuracy of 75.0 %
Using feature(s) {1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 39, 40, 41, 42, 44, 45, 46, 47, 48, 49, 50} gives an a
ccuracy of 75.8 %
Using feature(s) {1, 2, 3, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 44, 45, 46, 47, 48, 49, 50} gives an
accuracy of 75.4 %
Using feature(s) {1, 2, 3, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50} gives an
accuracy of 75.3 %
Warning, Accuracy has decreased! Continuing search in the event of local maxima
Feature(s):  {1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 39, 40, 41, 42, 44, 45, 46, 47, 48, 49, 50} were best, acc
uracy is 75.8 %
On level 48 of the search tree
Using feature(s) {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 39, 40, 41, 42, 44, 45, 46, 47, 48, 49, 50} gives a
n accuracy of 75.5 %
Using feature(s) {1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 44, 45, 46, 47, 48, 49, 50} gives
an accuracy of 74.9 %
Using feature(s) {1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50} gives
an accuracy of 74.9 %
Warning, Accuracy has decreased! Continuing search in the event of local maxima
Feature(s):  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 39, 40, 41, 42, 44, 45, 46, 47, 48, 49, 50} were best,
accuracy is 75.5 %
On level 49 of the search tree
Using feature(s) {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 44, 45, 46, 47, 48, 49, 50} giv
es an accuracy of 75.8 %
Using feature(s) {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50} giv
es an accuracy of 74.4 %
Warning, Accuracy has decreased! Continuing search in the event of local maxima
Feature(s):  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 44, 45, 46, 47, 48, 49, 50} were be
st, accuracy is 75.8 %
On level 50 of the search tree
Using feature(s) {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50}
 gives an accuracy of 74.7 %
Warning, Accuracy has decreased! Continuing search in the event of local maxima
Feature(s):  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50} wer
e best, accuracy is 74.7 %
Search completed.
Best feature subset is {39, 15} , which has an accuracy of 97.2 %
```

As seen from the screenshots, at the beginning of the search, we have an empty set of features which gives an accuracy of 84.9%. Adding feature “15” improved the accuracy to 86.1%. After adding feature “39” the accuracy jumped to 97.2%. After this point, the accuracy started to decrease as we added each further feature. For example, when the weak feature “21” was added next, the accuracy decreased to 95.7% for {15, 21, 39}. The accuracy further decreased to 93.3% for {15, 17, 21, 39}. After all the features were added, the recorded accuracy for the full set was 74.7%.

Next, below are two more screenshots which show the truncated search result for running Backward Elimination on set 20.





As seen from the screenshots, at the beginning of the search, we have {1, 2, 3, 4, …50} which gives an accuracy of 74.7%. This matches the accuracy of the last tested subset from performing Forward Selection. Similarly, the empty set at the end of our search gives an accuracy of 84.9% which matches the accuracy derived from performing Forward Selection. Performing Backward Elimination gave a best feature subset of {1, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 31, 32, 33, 34, 36, 37, 39, 42, 45, 47, 48, 49, 50} with an accuracy of 84.9%. This makes sense since it's usually not possible to achieve as high of an accuracy as Forward Selection. This is because it is easier for Backward Elimination to go down the wrong path of a higher local maxima than for Forward Selection, especially with a lot of features. The empty set also had an accuracy of 84.9%, but since the highest accuracy only gets updated if we have a higher accuracy, the best set returned is the one that came earlier.

Large Set Conclusion: I believe that the feature subset {15, 39} is the best set for this particular problem. This set gives the highest possible accuracy of 97.2% with the next best set {15, 21, 39} giving an accuracy of 95.7%. It took about 12 hours to run Forward Selection on this set and about 24 hours to run Backward Elimination.

Resources: To complete this assignment, I used:

- The Project 2 Briefing slides/video for the search and nearest neighbor algorithm framework, and for concepts to help me understand the assignment
- https://www.python.org/ for python documentation on various functions and expressions
- The sample project posted in dropbox for report format

Screenshots for Forward Selection and Backward Elimination on set 83

```
Hello and Welcome to the Feature Selection Program

Please enter the test case file name: small83.txt
Please enter the name of the search algorithm you wish to use (F for Forward Selection, B for Backward Elimination): F
This dataset has 10 features with 500 instances.
Using no features gives an accuracy of 83.0 %
On level 1 of the search tree
Using feature(s) {1} gives an accuracy of 84.2 %
Using feature(s) {2} gives an accuracy of 70.4 %
Using feature(s) {3} gives an accuracy of 70.0 %
Using feature(s) {4} gives an accuracy of 73.4 %
Using feature(s) {5} gives an accuracy of 73.0 %
Using feature(s) {6} gives an accuracy of 69.8 %
Using feature(s) {7} gives an accuracy of 77.8 %
Using feature(s) {8} gives an accuracy of 71.0 %
Using feature(s) {9} gives an accuracy of 69.4 %
Using feature(s) {10} gives an accuracy of 70.4 %
Feature(s):  {1} were best, accuracy is 84.2 %
On level 2 of the search tree
Using feature(s) {1, 2} gives an accuracy of 87.0 %
Using feature(s) {1, 3} gives an accuracy of 87.4 %
Using feature(s) {1, 4} gives an accuracy of 83.0 %
Using feature(s) {1, 5} gives an accuracy of 83.2 %
Using feature(s) {1, 6} gives an accuracy of 85.6 %
Using feature(s) {1, 7} gives an accuracy of 95.4 %
Using feature(s) {8, 1} gives an accuracy of 83.8 %
Using feature(s) {1, 9} gives an accuracy of 86.0 %
Using feature(s) {1, 10} gives an accuracy of 86.6 %
Feature(s):  {1, 7} were best, accuracy is 95.4 %
On level 3 of the search tree
Using feature(s) {1, 2, 7} gives an accuracy of 93.2 %
```

```
On level 9 of the search tree
Using feature(s) {1, 2, 3, 4, 5, 6, 7, 9, 10} gives an accuracy of 78.4 %
Using feature(s) {1, 2, 3, 4, 5, 7, 8, 9, 10} gives an accuracy of 77.6 %
Warning, Accuracy has decreased! Continuing search in the event of local maxima
Feature(s):  {1, 2, 3, 4, 5, 6, 7, 9, 10} were best, accuracy is 78.4 %
On level 10 of the search tree
Using feature(s) {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} gives an accuracy of 76.2 %
Warning, Accuracy has decreased! Continuing search in the event of local maxima
Feature(s):  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} were best, accuracy is 76.2 %
Search completed.
Best feature subset is {1, 7} , which has an accuracy of 95.4 %
```

```
Hello and Welcome to the Feature Selection Program

Please enter the test case file name: small83.txt
Please enter the name of the search algorithm you wish to use (F for Forward Selection, B for Backward Elimination): B
This dataset has 10 features with 500 instances.
Using feature(s) {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} gives an accuracy of 76.2 %
On level 1 of the search tree
Using feature(s) {2, 3, 4, 5, 6, 7, 8, 9, 10} gives an accuracy of 72.8 %
Using feature(s) {1, 3, 4, 5, 6, 7, 8, 9, 10} gives an accuracy of 77.0 %
Using feature(s) {1, 2, 4, 5, 6, 7, 8, 9, 10} gives an accuracy of 75.8 %
Using feature(s) {1, 2, 3, 5, 6, 7, 8, 9, 10} gives an accuracy of 77.8 %
Using feature(s) {1, 2, 3, 4, 6, 7, 8, 9, 10} gives an accuracy of 74.6 %
Using feature(s) {1, 2, 3, 4, 5, 7, 8, 9, 10} gives an accuracy of 77.6 %
Using feature(s) {1, 2, 3, 4, 5, 6, 8, 9, 10} gives an accuracy of 75.2 %
Using feature(s) {1, 2, 3, 4, 5, 6, 7, 9, 10} gives an accuracy of 78.4 %
Using feature(s) {1, 2, 3, 4, 5, 6, 7, 8, 10} gives an accuracy of 77.2 %
Using feature(s) {1, 2, 3, 4, 5, 6, 7, 8, 9} gives an accuracy of 75.0 %
Feature(s):  {1, 2, 3, 4, 5, 6, 7, 9, 10} were best, accuracy is 78.4 %
On level 2 of the search tree
Using feature(s) {2, 3, 4, 5, 6, 7, 9, 10} gives an accuracy of 71.4 %
Using feature(s) {1, 3, 4, 5, 6, 7, 9, 10} gives an accuracy of 79.4 %
Using feature(s) {1, 2, 4, 5, 6, 7, 9, 10} gives an accuracy of 79.4 %
Using feature(s) {1, 2, 3, 5, 6, 7, 9, 10} gives an accuracy of 79.4 %
Using feature(s) {1, 2, 3, 4, 6, 7, 9, 10} gives an accuracy of 78.4 %
Using feature(s) {1, 2, 3, 4, 5, 7, 9, 10} gives an accuracy of 80.8 %
Using feature(s) {1, 2, 3, 4, 5, 6, 9, 10} gives an accuracy of 76.0 %
Using feature(s) {1, 2, 3, 4, 5, 6, 7, 10} gives an accuracy of 77.6 %
Using feature(s) {1, 2, 3, 4, 5, 6, 7, 9} gives an accuracy of 79.2 %
Feature(s):  {2, 3, 4, 5, 7, 1, 9, 10} were best, accuracy is 80.8 %
On level 3 of the search tree
Using feature(s) {2, 3, 4, 5, 7, 9, 10} gives an accuracy of 75.2 %
```

```
On level 9 of the search tree
Using feature(s) {7} gives an accuracy of 77.8 %
Using feature(s) {1} gives an accuracy of 84.2 %
Warning, Accuracy has decreased! Continuing search in the event of local maxima
Feature(s):  {1} were best, accuracy is 84.2 %
On level 10 of the search tree
Using no features gives an accuracy of 83.0 %
Warning, Accuracy has decreased! Continuing search in the event of local maxima
Feature(s):  set() were best, accuracy is 83.0 %
Search completed.
Best feature subset is {1, 7} , which has an accuracy of 95.4 %
```

main.py

```
import math
import copy
import validationAndNeighbor as nearestNeighbor

numInstances = 0  #Number of instances, global because used in multiple functions
numFeatures = 0    #Number of features, global because used in multiple functions

def normalizeInstances(myInstances):
   global numFeatures #Because this global variable is used in this function
   meanList = []   #Includes the mean of each column as an entry in the list
   stDevList = []  #Includes the std of each column as an entry in the list
   for i in range(1, numFeatures + 1): #Starting from 1 to exclude class column
       meanSum = 0 #Used to collect sum of each column
       for row in myInstances: #For each line
           meanSum += row[i] #Sum all the values in the column
       meanList.append(meanSum / numInstances) #Divide by number of instances to get
mean for that particular column and add to mean list

   for i in range(1, numFeatures + 1):
       stDevSum = 0
       for row in myInstances:
           stDevSum += pow((row[i] - meanList[i - 1]), 2) #Subtract value from mean
and square result for each row
       stDevList.append(math.sqrt(stDevSum / numInstances)) #Take square root of sum
divided by number of instances to get std for that column and add to stDev list

   for i in range(0, numInstances):
       for j in range(1, numFeatures + 1):
           myInstances[i][j] = ((myInstances[i][j] - meanList[j-1]) / stDevList[j-1])
#Change each feature value to normalized feature value
   return myInstances #return newly normalized instances

def forwardSelection(normInstances):
   global numFeatures
   currentFeatures = set() #new set for current features
   finalFeatures = set() #new set for final features
   highestAccuracy = 0.0 #Used to note the highest accuracy
   myAccuracy = nearestNeighbor.leaveOneCrossValidation(numInstances, currentFeatures,
normInstances, 0) #Calculate the accuracy of the feature set using leave one out cross
validation
```

```
    print("Using no features gives an accuracy of", myAccuracy, "%") #Print accuracy
for empty set
    for i in range(numFeatures): #for each feature
        addFeature = -1 #new feature to be added initially set to -1 for default
        print("On level %d of the search tree" % (i+ 1))
        localAddFeature = -1 #Local feature to be added initially set to -1 for default
        localAccuracy = 0.0 #Highest local accuracy initially set to 0.0
        for j in range(1, numFeatures + 1): #j will take on the value of each feature
            if(j not in currentFeatures): #meaning we have not yet selected this
feature
                whatIfSet = copy.copy(currentFeatures) #shallow copy the current
feature set in order to use for printing purposes
                whatIfSet.add(j) #Add the new potential feature
                myAccuracy = nearestNeighbor.leaveOneCrossValidation(numInstances,
currentFeatures, normInstances, j)
                if(len(whatIfSet) == 0): #If we have an empty set
                    print("Using no features gives an accuracy of", myAccuracy, "%")
#Nicer syntax when printing for empty set
                else:
                    print("Using feature(s)", whatIfSet, "gives an accuracy of",
myAccuracy, "%") #Prints accuracy in the case of adding the feature
                if(myAccuracy > highestAccuracy): #if we have a new all time high
accuracy
                    highestAccuracy = myAccuracy #Set highest to the accuracy we just
calculcated since it is higher
                    addFeature = j #feature to be added is j
                elif(myAccuracy > localAccuracy): #if we have a new local maxima
                    localAccuracy = myAccuracy #Set highest local to the accuracy we
just calculcated since it is higher than current local accuracy
                    localAddFeature = j #local feature to be added is j
        if(addFeature >= 0): #if we have a new feature to be added that is going to
bring the all time accuracy higher
            currentFeatures.add(addFeature) #Add the feature into from the current
feature set
            finalFeatures.add(addFeature) #Add the feature into the final feature set
            print("Feature(s): ", currentFeatures, "were best, accuracy is",
highestAccuracy, "%")
        else: #All time high accuracy was not surpassed in this round but keep
searching in case this is a local maxima
            print("Warning, Accuracy has decreased! Continuing search in the event of
local maxima")
```

```
            currentFeatures.add(localAddFeature) #Add the feature into the current
feature set
            print("Feature(s): ", currentFeatures, "were best, accuracy is",
localAccuracy, "%") #Feature subset that gives highest local accuracy
    print("Search completed.")
    print("Best feature subset is", finalFeatures, ", which has an accuracy of",
highestAccuracy, "%") #Feature subset that gives highest accuracy

def backwardElimination(normInstances):
    global numFeatures
    currentFeatures = set()
    finalFeatures = set()
    highestAccuracy = 0.0
    myAccuracy = 0.0
    for i in range(numFeatures):
        currentFeatures.add(i+1) #Starting with all the features
        finalFeatures.add(i+1) #Starting with all the features
    myAccuracy = nearestNeighbor.leaveOneCrossValidation(numInstances, currentFeatures,
normInstances, 0)
    print("Using feature(s)", currentFeatures, "gives an accuracy of", myAccuracy, "%")
#Prints accuracy for the full set of features before we remove anything
    for i in range(numFeatures):
        removeFeature = -1 #new feature to be removed initially set to -1 for default
        print("On level %d of the search tree" % (i+ 1))
        localRemoveFeature = -1
        localAccuracy = 0.0
        for j in range(1, numFeatures + 1):
            if(j in currentFeatures): #If we still have not removed this feature
                whatIfSet = copy.copy(currentFeatures)
                whatIfSet.remove(j) #remove feature in question for printing purpose
                myAccuracy = nearestNeighbor.leaveOneCrossValidation(numInstances,
currentFeatures, normInstances, (-1*j))
                if(len(whatIfSet) == 0):
                    print("Using no features gives an accuracy of", myAccuracy, "%")
                else:
                    print("Using feature(s)", whatIfSet, "gives an accuracy of",
myAccuracy, "%") #Prints the accuracy in the case of removing the feature
                if(myAccuracy > highestAccuracy):
                    highestAccuracy = myAccuracy
                    removeFeature = j #feature to be removed is j
                elif(myAccuracy > localAccuracy):
                    localAccuracy = myAccuracy
```

```
                    localRemoveFeature = j #local feature to be removed is j
       if(removeFeature >= 0): #if we have a new feature to be removed that is going to bring all time accuracy higher
           currentFeatures.remove(removeFeature) #Remove the feature from the current feature set
           finalFeatures.remove(removeFeature) #Remove the feature from the final feature set
           print("Feature(s): ", currentFeatures, "were best, accuracy is", highestAccuracy, "%")
       else:
           currentFeatures.remove(localRemoveFeature) #Remove the feature from the current feature set only since we still have a different better feature set somewhere else
           print("Warning, Accuracy has decreased! Continuing search in the event of local maxima")
           print("Feature(s): ", currentFeatures, "were best, accuracy is", localAccuracy, "%")
   print("Search completed.")
   print("Best feature subset is", finalFeatures, ", which has an accuracy of", highestAccuracy, "%")


def main():
   global numInstances
   global numFeatures
   myInstances = [] #Will be the whole collection of values in file
   print("Hello and Welcome to the Feature Selection Program\n")
   fileName = input("Please enter the test case file name: ") #Grab input from user for file name
   searchType = "" #Variable used for gathering user input on algorithm type
   while (searchType != "F" and searchType != "B"): #While the user entered a value algorithm type
       searchType = input("""Please enter the name of the search algorithm you wish to use (F for Forward Selection, B for Backward Elimination): """) #Grab input from user
   try:
       myFile = open(fileName, 'r') #Try to open the file
   except:
        raise IOError('The given file does not exist.') #If file is not found then raise an IO error
   firstInstance = myFile.readline() #Read first line of file
   numFeatures = len(firstInstance.split()) - 1 #Split line into elements by space and subtract 1 because of class column
```

```python
    myFile.seek(0) #Return cursor to start position of file to avoid calculating wrong number of instances
    fileContent = myFile.read() #Read all contents of file as string
    fileList = fileContent.split("\n") #Split up file content string by newline
    for row in fileList: #Each row represents a line in the file
        if(len(row) > 0): #If we have a row of actual values
            numInstances = numInstances + 1 #For each row we have a new instance
    for i in range(numInstances):
        myInstances.append([]) #Initializing an empty array of size num of instances
    incrementor = -1 #Used to assign values to each row; -1 because gets incremented before being used
    for line in fileList: #For each line in the file
        incrementor+=1 #Increment for each line to move down each row in our myInstances array
        valueList = line.split("  ") #Split up the row string by 2 character spaces to get just the value
        for value in valueList: #For each value in the row
            if(len(value) > 0): #If we have an actual number value and not a space
                myInstances[incrementor].append(float(value)) #Adding values to each row
    normInstances = normalizeInstances(myInstances) #Normalize instances and return new array
    print("This dataset has %d features with %d instances." % (numFeatures, numInstances))
    if (searchType == "F"): #If we have selected forward selection
        forwardSelection(normInstances)
    else: #We selected backward elimination
        backwardElimination(normInstances)
    myFile.close() #Close the file since we are done with it

if __name__ == '__main__': #Prevents main from being called when the code is imported as a module
    main()
```

validationAndNeighbor.py

```
import math

def nearestNeighborAlgorithm(numInstances, oneOutInstance, features, myInstances):
   nearestNeighbor = -1 #Nearest neighbor initially set to -1
   nearestNeighborDistance = float("inf") #distance of nearest neighbor initially set to infinity
   for i in range(numInstances):
       if(i == oneOutInstance): #If the one out instance is the ith iteration then don't do anything this iteration
           pass
       else:
           myDistance = 0
           for k in range(len(features)):
               myDistance += pow((myInstances[i][features[k]] - myInstances[oneOutInstance][features[k]]), 2) #Calculcate distance given distance = √(sum((x-y)^2)) where x and y are two values
           myDistance = math.sqrt(myDistance) #Take the square root of the distance
           if(myDistance < nearestNeighborDistance): #If the newly calculated distance is lower than the nearest neighbor distance then its the new nearest neighbor
               nearestNeighborDistance = myDistance #Set the nearest neighbor distance to this distance
               nearestNeighbor = i #Nearest neighbor is this iteration
   return nearestNeighbor #We return our nearest neighbor


def leaveOneCrossValidation(numInstances, features, myInstances, feature):
   if(feature > 0): #If we are performing forward selection
       featuresList = list(features) #Make a list version of the set because sets are not subscriptable
       featuresList.append(feature) #Add the feature to the list
   elif(feature < 0): #If we are performing backward elimination
       feature = feature * -1 #Change the feature back to positive
       features.remove(feature) #Remove the feature from the set
       featuresList = list(features)
       features.add(feature) #Add the feature back to the set since we don't want to change the original set that was passed in
   else: #If we are not appending or removing an item
       featuresList = list(features)
   numCorrectlyClassified = 0 #Used to keep track for number of instances correctly classified
   for i in range(numInstances):
```

```
       oneOutInstance = i #For each fold we take an instance
       nearestNeighbor = nearestNeighborAlgorithm(numInstances, oneOutInstance,
featuresList, myInstances) #Calculate the nearest neighbor to this one out instance
       if(myInstances[nearestNeighbor][0] == myInstances[oneOutInstance][0]): #If the
nearest neighbor class label matches the one out class label
           numCorrectlyClassified += 1 #Increment the number of correctly classified
labels
   accuracy = (numCorrectlyClassified / numInstances) #Accuracy is the number of
correctly classified labels divided by the number of total instances
   accuracy *= 100 #Multiply by 100 to get percentage number
   return round(accuracy, 1) #Round accuracy to the tenths place
```