Trung Lam
SID #: 861270734
Email: tlam036@ucr.edu
December 10, 2021

CS170 - Project 2: Feature Selection with Nearest Neighbor with Dr. Eamonn Keogh

All code is original except:

- Numpy library used for manipulation of data and computing Euclidean distance for nearest neighbor
- Copy library used to assist in copying variables

Consultation during the completion of the project:

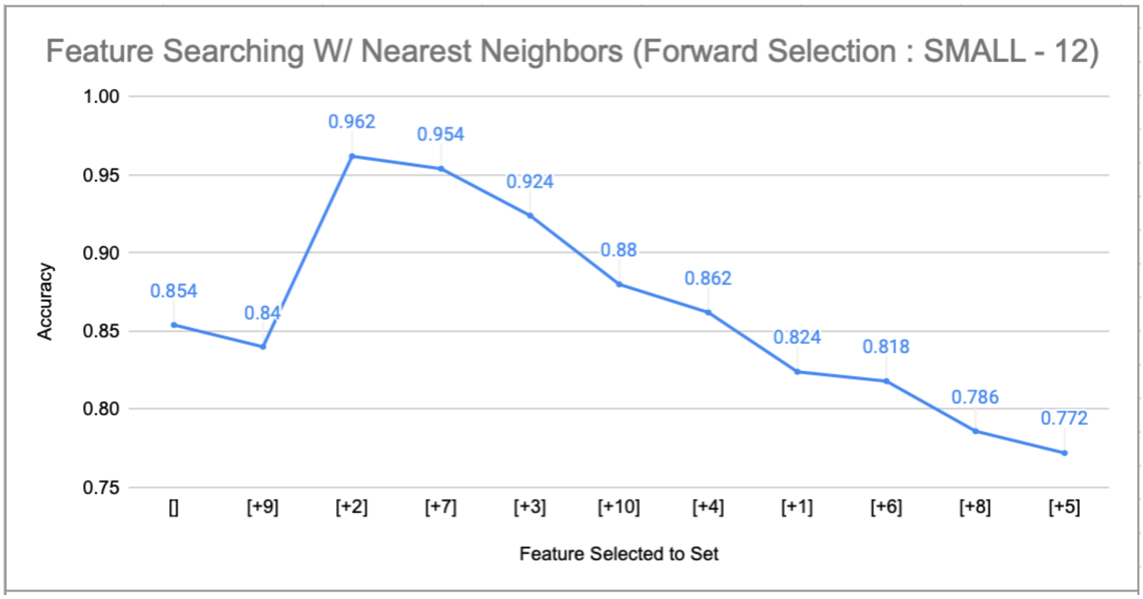
- Lecture video recording & slides of the project 2 briefing
- Numpy Documentations: https://numpy.org/doc/stable/reference/

Outline:

Project Description:

In this project, the objective is to select features in a given data set that gives the highest accuracy, indicating the best features to use for prediction purposes of a data set. This project will use the Nearest Neighbor algorithm with the Euclidean Distance to find the nearest neighbor, along with two search algorithms: Forward Selection, where the search is initialized as an empty set and continuously adds features with the highest accuracy until all possible features are exhausted. And Backward Elimination, where the search is initialized with all possible features, and continuously eliminates features which results in a set with the highest accuracy, until there are no features left to remove from the set.

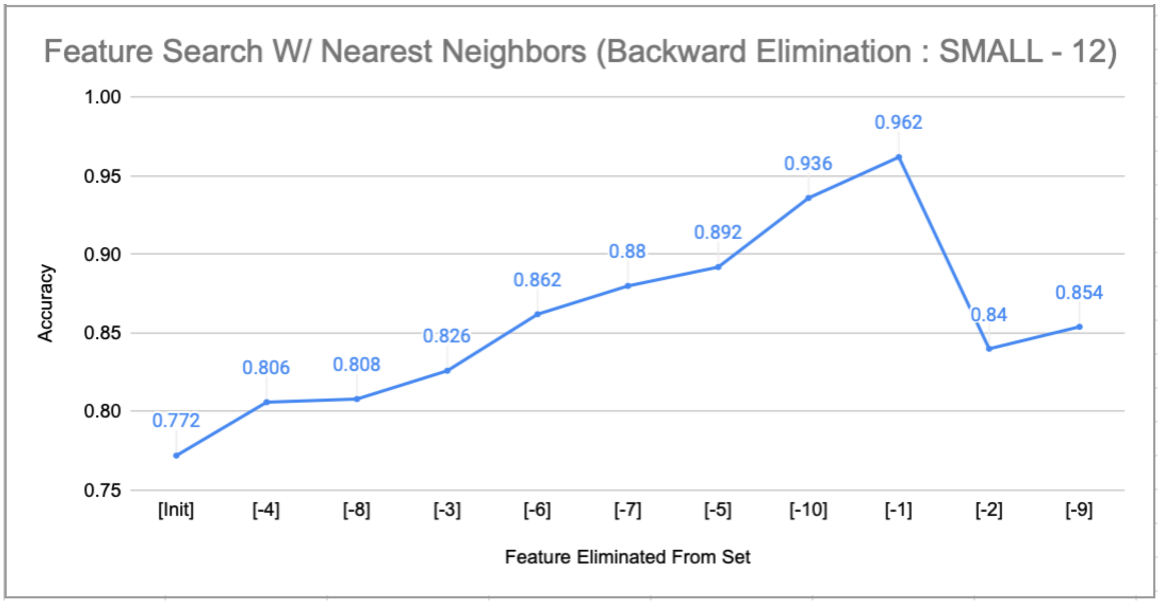
Small Dataset Result (Feature Selection):



In the figure above, using forward selection, the search was initialized as the default rate of the dataset which is the percentage of the highest occurring class in that dataset, which is approximately 85%. As the search continues and adds features to the set, the highest accuracy from the entire search is approximately 96% using the features 9 and 2. Afterwards, as expected, the accuracy continues to get worse as more features are added to the set.

Small Dataset Result (Backward Elimination):

To compare, let's now run the dataset using backward elimination.

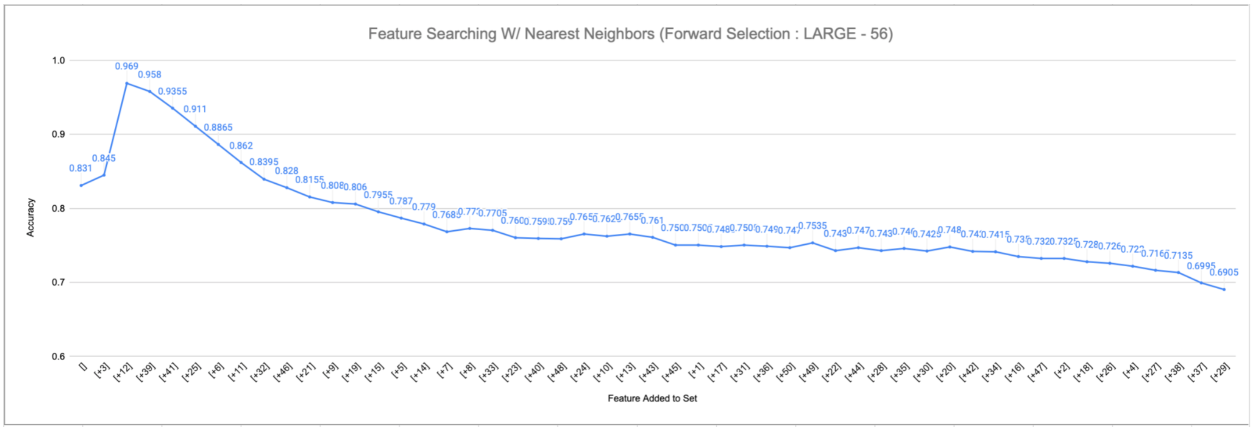


In the figure above, unlike forward selection, the search was initialized with all possible features, label [init] for space. Over time, the search eliminates a feature from the set, eventually leading to the highest accuracy achieved after removing feature 1, giving the best features to be 2 and 9, the same features identified by the forward selection search algorithm.

Small Dataset Conclusion (Features 2 and 9):

Based on both the forward selection and backward elimination search algorithm, it can be concluded with high confidence that features 2 and 9 are the best features of the dataset.

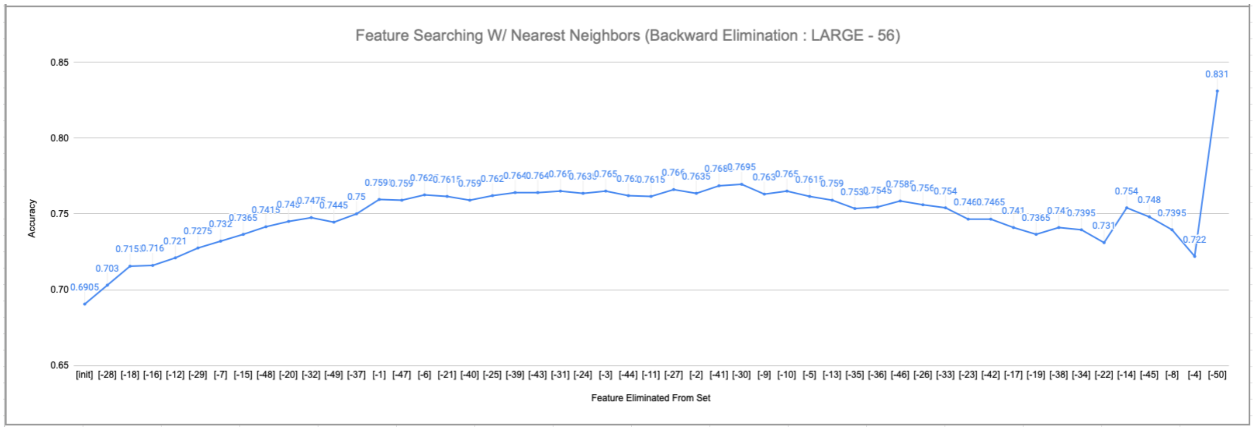
Large Dataset Result (Forward Selection):



Similar to the small dataset, in the figure above, the trajectory is almost identical to that of the small dataset using forward selection, such that the best features can be found early on and over time, adding more features would only reduce the accuracy. In this dataset, the best features using forward selection were determined to be features 3 and 12 with the accuracy of approximately 97%.

Large Dataset Result (Backward Elimination):

To compare, let's now run the dataset using backward elimination.



In the figure above, the backward elimination search algorithm for the large dataset does not give a definite answer unlike the small dataset, as the accuracy of the data hovers between 70% and

80% as more features are removed from the set, with the exception of last iteration which indicates the default rate of the data with the accuracy of approximately 83%.

Large Dataset Conclusion (Features 3 and 12):

Given the result from the forward selection algorithm, the best features of the large dataset can be, with average confidence, features 3 and 12 with the accuracy of approximately 97% due to the backward elimination algorithm returning essentially no best features. But this can be the result of coding errors.

Example Outputs (Only Small Dataset for time)

Forward Selection:

```
Feature Selection Using Nearest Neighbors
Pick Search Algorithm (1: Forward Selection, 2: Backward Elimination) : 1
Pick Type of Data (1: Small Data, 2: Large Data) : 1
Level [ 0 ] : Default Rate with accuracy:  0.854
Considering selecting feature:  1
Considering selecting feature:  2
Considering selecting feature:  3
Considering selecting feature:  4
Considering selecting feature:  5
Considering selecting feature:  6
Considering selecting feature:  7
Considering selecting feature:  8
Considering selecting feature:  9
Considering selecting feature:  10
Level [ 1 ] : Select features: 9 , with accuracy:  0.84
Current Set: [9]
Considering selecting feature:  1
Considering selecting feature:  2
Considering selecting feature:  3
Considering selecting feature:  4
Considering selecting feature:  5
Considering selecting feature:  6
Considering selecting feature:  7
Considering selecting feature:  8
Considering selecting feature:  10
Level [ 2 ] : Select features: 2 , with accuracy:  0.962
Current Set: [9 2]
Considering selecting feature:  1
Considering selecting feature:  3
Considering selecting feature:  4
Considering selecting feature:  5
Considering selecting feature:  6
Considering selecting feature:  7
Considering selecting feature:  8
Considering selecting feature:  10
Level [ 3 ] : Select features: 7 , with accuracy:  0.954
Current Set: [9 2 7]
.
. (Omitting Outputs for Space)
.
Considering selecting feature:  5
Level [ 10 ] : Select features: 5 , with accuracy:  0.772
Current Set: [ 9  2  7  3 10  4  1  6  8  5]
Best Feature Set: [array([9, 2]), 0.962]
```

Backward Elimination:

```
Feature Selection Using Nearest Neighbors
Pick Search Algorithm (1: Forward Selection, 2: Backward Elimination) : 2
Pick Type of Data (1: Small Data, 2: Large Data) : 1
Level [ 0 ] : Initial features: [ 1  2  3  4  5  6  7  8  9 10] , with accuracy:  0.772
Considering eliminating feature:  1
Considering eliminating feature:  2
Considering eliminating feature:  3
Considering eliminating feature:  4
Considering eliminating feature:  5
Considering eliminating feature:  6
Considering eliminating feature:  7
Considering eliminating feature:  8
Considering eliminating feature:  9
Considering eliminating feature:  10
Level [ 1 ] : Eliminate feature: 4
Current Set: [ 1  2  3  5  6  7  8  9 10] , with accuracy:  0.806
Considering eliminating feature:  1
Considering eliminating feature:  2
Considering eliminating feature:  3
Considering eliminating feature:  5
Considering eliminating feature:  6
Considering eliminating feature:  7
Considering eliminating feature:  8
Considering eliminating feature:  9
Considering eliminating feature:  10
Level [ 2 ] : Eliminate feature: 8
Current Set: [ 1  2  3  5  6  7  9 10] , with accuracy:  0.808
.
. (Omitting Outputs for Space)
.
Considering eliminating feature:  1
Considering eliminating feature:  2
Considering eliminating feature:  5
Considering eliminating feature:  9
Considering eliminating feature:  10
Level [ 6 ] : Eliminate feature: 5
Current Set: [ 1  2  9 10] , with accuracy:  0.892
Considering eliminating feature:  1
Considering eliminating feature:  2
Considering eliminating feature:  9
Considering eliminating feature:  10
Level [ 7 ] : Eliminate feature: 10
Current Set: [1 2 9] , with accuracy:  0.936
Considering eliminating feature:  1
Considering eliminating feature:  2
Considering eliminating feature:  9
Level [ 8 ] : Eliminate feature: 1
Current Set: [2 9] , with accuracy:  0.962
Considering eliminating feature:  2
Considering eliminating feature:  9
Level [ 9 ] : Eliminate feature: 2
Current Set: [9] , with accuracy:  0.84
Considering eliminating feature:  9
Level [ 10 ] : Eliminate feature: 9
Current Set: [] , with accuracy:  0.854
Best Feature Set: [array([2, 9]), 0.962]
```

Code

```
import numpy as np
import copy as cp

#Support Functions

def nearest_neighbors(x, data):

  #calculate Euclidean distance
  distance = np.linalg.norm(x - data)
  return distance

def cross_validation(labels, data):
  distance = []
  prediction = []

  #Begin Loop
  for x in range(len(data)):
    for i in range(len(data)):

      #If x is the data itself, skip
      if np.all(data[x] == data[i]):
        distance.append(float("inf"))
        continue

      #calculate distance
      distance.append(nearest_neighbors(data[x], data[i]))

    #retrieve the indice of the nearest data and fetch the label
    nearest_ind = np.argmin(distance)
    prediction.append(labels[nearest_ind])
    distance.clear()

  #calculate accuracy
  accuracy = (np.sum(labels == prediction) / len(data))
  return accuracy

def forward_selection(labels, features, default_rate):

  selected_features = []
  accuracy = []
  best_accuracy = 0
  best_features = 0

  print('Level [ 0 ] : Default Rate with accuracy: ', default_rate)

  #find first feature to add

  #calculate accuracies per feature
  for i in range(len(features[0])):
    print('Considering selecting feature: ', i + 1)
    accuracy.append(cross_validation(labels, features[:, i]))

  #retrieve max indice of the feature with greatest accuracy
  ind_max = np.argmax(accuracy)
  selected_features.append(ind_max)
  print('Level [ 1 ] : Select features:', ind_max + 1, ', with accuracy: ',
accuracy[ind_max])
```

```
  print('Current Set:', np.array(selected_features) + 1)
  accuracy.clear()

  counter = 2
  #Begin loop with set initialized with first feature found
  while (1):
    for i in range(len(features[0])):
      temp = cp.deepcopy(selected_features)
      if i in temp:

        #if feature is already in set, set accuracy to 0
        accuracy.append(0)
        continue
      temp.append(i)
      print('Considering selecting feature: ', i + 1)
      accuracy.append(cross_validation(labels, features[:, temp]))

    #retrieve max indice to feature with greatest accuracy
    ind_max = np.argmax(accuracy)
    selected_features.append(ind_max)
    print('Level [', counter, '] : Select features:', ind_max + 1, ', with accuracy: ',
accuracy[ind_max])
    print('Current Set:', np.array(selected_features) + 1)

    #compare current best feature to overall best feature
    if accuracy[ind_max] >= best_accuracy:
      best_accuracy = accuracy[ind_max]
      best_features = [np.array(cp.deepcopy(selected_features)) + 1, accuracy[ind_max]]
    accuracy.clear()

    #if search reaches the end, break
    if(len(selected_features) == len(features[0])):
      break
    counter += 1

  return best_features

def backward_elimination(labels, features, default_rate):

  accuracy = []
  feature = 0
  best_accuracy = 0
  best_features = 0

  #initialize first set with all possible features
  initial = np.arange(len(features[0]))

  print('Level [ 0 ] : Initial features:', initial + 1, ', with accuracy: ',
cross_validation(labels, features[:, initial]))

  temp = cp.deepcopy(initial)

  counter = 1
  while (1):

    #if search reaches end, break
    if len(temp) == 0:
      break
    #iterate through all elements in temp
    for i in temp:

      #delete an element one by one and calculate accuracy
      temp = np.delete(temp, np.argwhere(temp == i))
      print('Considering eliminating feature: ', i + 1)
      accuracy.append(cross_validation(labels, features[:, temp]))
      temp = cp.deepcopy(initial)

    #retrieve max indice of set with maximum accuracy
    ind_max = np.argmax(accuracy)
    feature = initial[ind_max]

    #delete feature that result in set with max accuracy from initial set
    initial = np.delete(initial, ind_max)
    print('Level [', counter, '] : Eliminate feature:', feature + 1)
    print('Current Set:', initial + 1, ', with accuracy: ', cross_validation(labels, 
features[:, initial]))

    #compare current best accuracy to overall best accuracy
    if cross_validation(labels, features[:, initial]) >= best_accuracy:
      best_accuracy = cross_validation(labels, features[:, initial])
      best_features = [initial + 1, best_accuracy]
    accuracy.clear()
    temp = cp.deepcopy(initial)
    counter += 1

  return best_features

# main program
def main():
  small_data = np.loadtxt('Ver_2_CS170_Fall_2021_Small_data__12.txt')
  small_labels = small_data[:, 0]
  small_features = small_data[:, 1:]
  small_default = np.sum(small_labels == max(small_labels)) / len(small_labels)

  large_data = np.loadtxt('Ver_2_CS170_Fall_2021_LARGE_data__56.txt')
  large_labels = large_data[:, 0]
  large_features = large_data[:, 1:]
  large_default = np.sum(large_labels == max(large_labels)) / len(large_labels)

  print('Feature Selection Using Nearest Neighbors')
  searchInput = input('Pick Search Algorithm (1: Forward Selection, 2: Backward 
Elimination) : ')
  dataInput = input('Pick Type of Data (1: Small Data, 2: Large Data) : ')

  if(searchInput == '1'):
    if(dataInput == '1'):
      best_feature = forward_selection(small_labels, small_features, small_default)
      print('Best Feature Set:', best_feature)
    elif(dataInput == '2'):
      best_feature = forward_selection(large_labels, large_features, large_default)
      print('Best Feature Set:', best_feature)
    else:
      print('Invalid Input')
      return
  elif(searchInput == '2'):
```

```
  if(dataInput == '1'):
    best_feature = backward_elimination(small_labels, small_features, small_default)
    print('Best Feature Set:', best_feature)
  elif(dataInput == '2'):
    best_feature = backward_elimination(large_labels, large_features, large_default)
    print('Best Feature Set:', best_feature)
  else:
    print('Invalid Data Input')
    return
else:
  print('Invalid Search Input')
  return
```