CS 205 Project Instructor: Dr Eamonn Keogh

**Feature Selection with Nearest Neighbor**

Yuan Yao [y](mailto:bertie@cs.ucr.edu)yao009@ucr.edu

SID: 861242929 12/06/2016

In completing this project, I consulted:

* <https://docs.python.org/2/library/queue.html> for the priority queue data structure in python
* <https://docs.scipy.org/doc/numpy/reference/generated/numpy.std.html>, <https://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html>

for the mean and standard deviation function.

* <https://docs.scipy.org/doc/numpy/reference/generated/numpy.argmax.html>

for the argmax function

**All the important code is original.**

|  |
| --- |
| **import** **time**  **import** **numpy**  **import** **Queue**  **class** **KNN\_classifier**:  **def** Euclidean\_distance(self, p1, p2, feature\_set):  distance = 0.0;  **for** i **in** feature\_set:  d = (p1[i] - p2[i])  distance += numpy.dot(d, d)  distance = numpy.sqrt(distance);  **return** distance  **def** k\_nearst\_neighbor(self, test, p, feature\_set, k):  res = []  **for** t **in** test:  neighbors = Queue.PriorityQueue()  **for** pp **in** p:  neighbors.put((self.Euclidean\_distance(t, pp, feature\_set), pp))  *# vote for test data using k nearest neighbors*  vote = [0]\*10  **for** i **in** range(k):  n = neighbors.get()[1];  vote[int(n[0])] += 1  res.append(numpy.argmax(vote))  **return** res  **def** k\_fold\_cross\_validation(self, data, feature\_set, k\_fold):  correct = 0  k = len(data) - k\_fold + 1  **for** i **in** range(0, len(data), k):  test = data[i:i+k]  train = data[:i]+data[i+k:]  predict\_class = self.k\_nearst\_neighbor(test, train, feature\_set, 1)    **for** j **in** range(len(test)):  **if** test[j][0] == predict\_class[j]:  correct += 1  **return** float(correct)/len(data)  **class** **feature\_selection**:  **def** \_\_init\_\_(self):  **print** "Welcome to Yuan Yao Feature Selection Algorithm."  file = raw\_input("Type in the name of the file to test: ")  *# while True:*  **print** "**\n**Type the number of the algorithm you want to run."  **print** "**\t**1) Forward Selection"  **print** "**\t**2) Backward Elimination"  **print** "**\t**3) Yuan's Special Algorithm."  method = raw\_input()  method = int(method)  *# file = "cs\_205\_NN\_datasets/cs\_205\_small65.txt"*  data = extract\_data(file)  start\_time = time.time()  **if** method == 1:  accuracy = self.forward\_selection(data)  **elif** method == 2:  accuracy = self.backward\_selection(data)  **elif** method == 3:  accuracy = self.special\_selection(data)  **else**:  **print** "Please choose a method"    end\_time = time.time()  time\_elapsed = end\_time - start\_time  **print** "Time cost: **%f**s" %time\_elapsed  **def** forward\_selection(self, data):  knn = KNN\_classifier();  default = self.default\_rate(data)  **print** "Using no feature, the default rate is **%.1f%%**" %(default\*100)  **print** "**\n**Beginning search."  *# foward search*  remain\_features = [i **for** i **in** range(1, len(data[0]))]  best\_features = []  feature\_set = []  fs = ()  local\_maxima\_count = 0  irrelevant\_count = 0  **while** remain\_features:  **if** local\_maxima\_count > 1:  **print** "Break searching since accuracy has decreased many times"  **break**  **if** irrelevant\_count > 1:  **print** "Stop searching since accuracy only changes a little due to irrelevant feature"  best\_features.pop()  **break**  **for** feature **in** remain\_features:  temp = []  **if** fs:  temp += fs[1]    temp.append(feature)  accuracy = knn.k\_fold\_cross\_validation(data, temp, len(data))  feature\_set.append((accuracy, temp))  **print** "**\t**Using feature(s)", temp, "accuracy is **%.1f%%**" %(accuracy\*100)  **if** remain\_features.index(feature) == len(remain\_features) - 1:  **print**  *# store the best result in best\_features*  fs = self.maxSet(feature\_set)  **if** best\_features:  prev\_accurate = best\_features[-1][0]  **if** fs[0] < prev\_accurate:  local\_maxima\_count += 1  **print** "(Warning, Accuracy has decreased! Continuing search in case of local maxima)"    **elif** fs[0] - prev\_accurate < 0.02:  irrelevant\_count += 1  **print** "Feature set", fs[1], "was best, accuracy is **%.1f%%**" %(fs[0]\*100)  best\_features.append(fs)  feature\_set = []  remain\_features.remove(fs[1][-1])  *# report the final result*  res = self.maxSet(best\_features)  **print** "**\n**Finished search! The best feature subset is", res[1], ", which has an accuracy of **%.1f%%**" %(res[0]\*100)    **def** backward\_selection(self, data):  knn = KNN\_classifier()  default = self.default\_rate(data)  **print** "Using no feature, the default rate is **%.1f%%**" %(default\*100)  **print** "**\n**Beginning search."  *# foward search*  curr\_feature = [i **for** i **in** range(1, len(data[0]))]  feature\_set = []  best\_features = []  fs = ()  accuracy = knn.k\_fold\_cross\_validation(data, curr\_feature, len(data))  **print** "**\t**Using feature(s)", curr\_feature, "accuracy is **%.1f%%**" %(accuracy\*100)  **while** len(curr\_feature) > 1:  **for** feature\_to\_remove **in** curr\_feature:  temp = []  temp += curr\_feature  temp.remove(feature\_to\_remove)  accuracy = knn.k\_fold\_cross\_validation(data, temp, len(data))  feature\_set.append((accuracy, temp))  **print** "**\t**Using feature(s)", temp, "accuracy is **%.1f%%**" %(accuracy\*100)  **if** curr\_feature.index(feature\_to\_remove) == len(curr\_feature) - 1:  **print**  fs = self.maxSet(feature\_set)  **if** best\_features:  prev\_accurate = best\_features[-1][0]  **if** fs[0] < prev\_accurate:  **print** "(Warning, Accuracy has decreased! Continuing search in case of local maxima)"    **print** "Feature set", fs[1], "was best, accuracy is **%.1f%%**" %(fs[0]\*100)  best\_features.append(fs)  feature\_set = []  curr\_feature = fs[1]  *# report the final result*  res = self.maxSet(best\_features)  **print** "**\n**Finished search! The best feature subset is", res[1], ", which has an accuracy of **%.1f%%**" %(res[0]\*100)    *# This search method uses forward search after searching for every pair of the two features*  *# to give a much better result*  **def** special\_selection(self, data):  knn = KNN\_classifier();  default = self.default\_rate(data)  **print** "Using no feature, the default rate is **%.1f%%**" %(default\*100)  **print** "**\n**Beginning search."    remain\_features = [i **for** i **in** range(1, len(data[0]))]  best\_features = []  feature\_set = []  fs = ()  local\_maxima\_count = 0  irrelevant\_count = 0  *# search for each pair of the two features*  **for** i **in** range(len(remain\_features)):  **for** j **in** range(i+1, len(remain\_features)):  pair = [remain\_features[i], remain\_features[j]]  accuracy = knn.k\_fold\_cross\_validation(data, pair, 10)  feature\_set.append((accuracy, pair))  **print** "**\t**Using feature(s)", pair, "accuracy is **%.1f%%**" %(accuracy\*100)    fs = self.maxSet(feature\_set)  **print** "Feature set", fs[1], "was best, accuracy is **%.1f%%**" %(fs[0]\*100)  best\_features.append(fs)  feature\_set = []  **for** p **in** fs[1]:  remain\_features.remove(p)  **while** remain\_features:  **if** local\_maxima\_count > 1:  **print** "Break searching since accuracy has decreased many times"  **break**  **if** irrelevant\_count > 1:  **print** "Stop searching since accuracy only changes a little due to irrelevant feature"  best\_features.pop()  **break**  **for** feature **in** remain\_features:  temp = []  **if** fs:  temp += fs[1]    temp.append(feature)  accuracy = knn.k\_fold\_cross\_validation(data, temp, len(data))  feature\_set.append((accuracy, temp))  **print** "**\t**Using feature(s)", temp, "accuracy is **%.1f%%**" %(accuracy\*100)  **if** remain\_features.index(feature) == len(remain\_features) - 1:  **print**  *# store the best result in best\_features*  fs = self.maxSet(feature\_set)  **if** best\_features:  prev\_accurate = best\_features[-1][0]  **if** fs[0] < prev\_accurate:  local\_maxima\_count += 1  **print** "(Warning, Accuracy has decreased! Continuing search in case of local maxima)"    **elif** fs[0] - prev\_accurate < 0.02:  irrelevant\_count += 1  **print** "Feature set", fs[1], "was best, accuracy is **%.1f%%**" %(fs[0]\*100)  best\_features.append(fs)  feature\_set = []  remain\_features.remove(fs[1][-1])  *# report the final result*  res = self.maxSet(best\_features)  **print** "**\n**Finished search! The best feature subset is", res[1], ", which has an accuracy of **%.1f%%**" %(res[0]\*100)    **def** default\_rate(self, data):  counters = [0]\*10  **for** i **in** data:  counters[int(i[0])] += 1  **return** float(max(counters))/len(data)  **def** maxSet(self, feature\_set):  fs = (0, [])  **for** m **in** feature\_set:  **if** fs[0] < m[0]:  fs = m  **return** fs  **def** z\_normalized(data):  means = numpy.mean(data, axis=0, dtype=numpy.float64)  stds = numpy.std(data, axis=0, dtype=numpy.float64)  **for** i **in** range(len(data)):  **for** j **in** range(1, len(data[i])):  data[i][j] = (data[i][j] - means[j])/stds[j]  **def** extract\_data(file):  f = open(file, 'r')  data = []    line = f.readline()  **while** line:  data.append([float(x) **for** x **in** line.split()])  line = f.readline()  features = len(data[0])-1  instances = len(data)  **print** "This dataset has **%d** features (not including the class attribute), with **%d** instances." %(features, instances)  **print** "Please wait while I normalize the data...",  z\_normalized(data);  **print** "Done!"  **return** data  **if** \_\_name\_\_ == "\_\_main\_\_":  feature\_set = feature\_selection() |

Trace of forward selection on cs\_205\_small56.txt:

|  |
| --- |
| Welcome to Yuan Yao Feature Selection Algorithm.  Type in the name of the file to test: cs\_205\_NN\_datasets/cs\_205\_small56.txt  Type the number of the algorithm you want to run.  1) Forward Selection  2) Backward Elimination  3) Yuan's Special Algorithm.  1  This dataset has 10 features (not including the class attribute), with 100 instances.  Please wait while I normalize the data... Done!  Using no feature, the default rate is 79.0%  Beginning search.  Using feature(s) [1] accuracy is 56.0%  Using feature(s) [2] accuracy is 68.0%  Using feature(s) [3] accuracy is 71.0%  Using feature(s) [4] accuracy is 74.0%  Using feature(s) [5] accuracy is 69.0%  Using feature(s) [6] accuracy is 61.0%  Using feature(s) [7] accuracy is 69.0%  Using feature(s) [8] accuracy is 88.0%  Using feature(s) [9] accuracy is 70.0%  Using feature(s) [10] accuracy is 67.0%  Feature set [8] was best, accuracy is 88.0%  Using feature(s) [8, 1] accuracy is 84.0%  Using feature(s) [8, 2] accuracy is 84.0%  Using feature(s) [8, 3] accuracy is 83.0%  Using feature(s) [8, 4] accuracy is 82.0%  Using feature(s) [8, 5] accuracy is 95.0%  Using feature(s) [8, 6] accuracy is 81.0%  Using feature(s) [8, 7] accuracy is 85.0%  Using feature(s) [8, 9] accuracy is 80.0%  Using feature(s) [8, 10] accuracy is 84.0%  Feature set [8, 5] was best, accuracy is 95.0%  Using feature(s) [8, 5, 1] accuracy is 93.0%  Using feature(s) [8, 5, 2] accuracy is 85.0%  Using feature(s) [8, 5, 3] accuracy is 96.0%  Using feature(s) [8, 5, 4] accuracy is 90.0%  Using feature(s) [8, 5, 6] accuracy is 86.0%  Using feature(s) [8, 5, 7] accuracy is 91.0%  Using feature(s) [8, 5, 9] accuracy is 90.0%  Using feature(s) [8, 5, 10] accuracy is 88.0%  Feature set [8, 5, 3] was best, accuracy is 96.0%  Using feature(s) [8, 5, 3, 1] accuracy is 88.0%  Using feature(s) [8, 5, 3, 2] accuracy is 93.0%  Using feature(s) [8, 5, 3, 4] accuracy is 86.0%  Using feature(s) [8, 5, 3, 6] accuracy is 81.0%  Using feature(s) [8, 5, 3, 7] accuracy is 93.0%  Using feature(s) [8, 5, 3, 9] accuracy is 88.0%  Using feature(s) [8, 5, 3, 10] accuracy is 83.0%  (Warning, Accuracy has decreased! Continuing search in case of local maxima)  Feature set [8, 5, 3, 2] was best, accuracy is 93.0%  Using feature(s) [8, 5, 3, 2, 1] accuracy is 85.0%  Using feature(s) [8, 5, 3, 2, 4] accuracy is 80.0%  Using feature(s) [8, 5, 3, 2, 6] accuracy is 81.0%  Using feature(s) [8, 5, 3, 2, 7] accuracy is 89.0%  Using feature(s) [8, 5, 3, 2, 9] accuracy is 90.0%  Using feature(s) [8, 5, 3, 2, 10] accuracy is 84.0%  (Warning, Accuracy has decreased! Continuing search in case of local maxima)  Feature set [8, 5, 3, 2, 9] was best, accuracy is 90.0%  Break searching since accuracy has decreased many times  Finished search! The best feature subset is [8, 5, 3] , which has an accuracy of 96.0%  Time cost: 6.874294s |

I got the best feature set of [8, 5, 3] with an accuracy of 96%. The correct result I got from Prof. Eamonn is: On small dataset 56 the error rate can be 0.95 when using only features 8  3  5. I got a good result. I make my program stops when: 1. accuracy has decreased two or more times; 2. accuracy increased < 0.2 two or more times because the features added can be considered as irrelevant features. I use k-fold cross validation with k=n and that is actually the leave one out cross validation. The time cost for forward selection is 6.87s.

Trace of backward Elimination on cs\_205\_small56.txt

|  |
| --- |
| Welcome to Yuan Yao Feature Selection Algorithm.  Type in the name of the file to test: cs\_205\_NN\_datasets/cs\_205\_small56.txt  Type the number of the algorithm you want to run.  1) Forward Selection  2) Backward Elimination  3) Yuan's Special Algorithm.  2  This dataset has 10 features (not including the class attribute), with 100 instances.  Please wait while I normalize the data... Done!  Using no feature, the default rate is 79.0%  Beginning search.  Using feature(s) [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] accuracy is 69.0%  Using feature(s) [2, 3, 4, 5, 6, 7, 8, 9, 10] accuracy is 68.0%  Using feature(s) [1, 3, 4, 5, 6, 7, 8, 9, 10] accuracy is 75.0%  Using feature(s) [1, 2, 4, 5, 6, 7, 8, 9, 10] accuracy is 70.0%  Using feature(s) [1, 2, 3, 5, 6, 7, 8, 9, 10] accuracy is 70.0%  Using feature(s) [1, 2, 3, 4, 6, 7, 8, 9, 10] accuracy is 66.0%  Using feature(s) [1, 2, 3, 4, 5, 7, 8, 9, 10] accuracy is 74.0%  Using feature(s) [1, 2, 3, 4, 5, 6, 8, 9, 10] accuracy is 70.0%  Using feature(s) [1, 2, 3, 4, 5, 6, 7, 9, 10] accuracy is 69.0%  Using feature(s) [1, 2, 3, 4, 5, 6, 7, 8, 10] accuracy is 65.0%  Using feature(s) [1, 2, 3, 4, 5, 6, 7, 8, 9] accuracy is 73.0%  Feature set [1, 3, 4, 5, 6, 7, 8, 9, 10] was best, accuracy is 75.0%  Using feature(s) [3, 4, 5, 6, 7, 8, 9, 10] accuracy is 75.0%  Using feature(s) [1, 4, 5, 6, 7, 8, 9, 10] accuracy is 75.0%  Using feature(s) [1, 3, 5, 6, 7, 8, 9, 10] accuracy is 69.0%  Using feature(s) [1, 3, 4, 6, 7, 8, 9, 10] accuracy is 70.0%  Using feature(s) [1, 3, 4, 5, 7, 8, 9, 10] accuracy is 73.0%  Using feature(s) [1, 3, 4, 5, 6, 8, 9, 10] accuracy is 79.0%  Using feature(s) [1, 3, 4, 5, 6, 7, 9, 10] accuracy is 64.0%  Using feature(s) [1, 3, 4, 5, 6, 7, 8, 10] accuracy is 73.0%  Using feature(s) [1, 3, 4, 5, 6, 7, 8, 9] accuracy is 76.0%  Feature set [1, 3, 4, 5, 6, 8, 9, 10] was best, accuracy is 79.0%  Using feature(s) [3, 4, 5, 6, 8, 9, 10] accuracy is 73.0%  Using feature(s) [1, 4, 5, 6, 8, 9, 10] accuracy is 78.0%  Using feature(s) [1, 3, 5, 6, 8, 9, 10] accuracy is 79.0%  Using feature(s) [1, 3, 4, 6, 8, 9, 10] accuracy is 74.0%  Using feature(s) [1, 3, 4, 5, 8, 9, 10] accuracy is 79.0%  Using feature(s) [1, 3, 4, 5, 6, 9, 10] accuracy is 74.0%  Using feature(s) [1, 3, 4, 5, 6, 8, 10] accuracy is 80.0%  Using feature(s) [1, 3, 4, 5, 6, 8, 9] accuracy is 79.0%  Feature set [1, 3, 4, 5, 6, 8, 10] was best, accuracy is 80.0%  Using feature(s) [3, 4, 5, 6, 8, 10] accuracy is 83.0%  Using feature(s) [1, 4, 5, 6, 8, 10] accuracy is 79.0%  Using feature(s) [1, 3, 5, 6, 8, 10] accuracy is 81.0%  Using feature(s) [1, 3, 4, 6, 8, 10] accuracy is 76.0%  Using feature(s) [1, 3, 4, 5, 8, 10] accuracy is 83.0%  Using feature(s) [1, 3, 4, 5, 6, 10] accuracy is 73.0%  Using feature(s) [1, 3, 4, 5, 6, 8] accuracy is 81.0%  Feature set [3, 4, 5, 6, 8, 10] was best, accuracy is 83.0%  Using feature(s) [4, 5, 6, 8, 10] accuracy is 86.0%  Using feature(s) [3, 5, 6, 8, 10] accuracy is 79.0%  Using feature(s) [3, 4, 6, 8, 10] accuracy is 78.0%  Using feature(s) [3, 4, 5, 8, 10] accuracy is 82.0%  Using feature(s) [3, 4, 5, 6, 10] accuracy is 73.0%  Using feature(s) [3, 4, 5, 6, 8] accuracy is 80.0%  Feature set [4, 5, 6, 8, 10] was best, accuracy is 86.0%  Using feature(s) [5, 6, 8, 10] accuracy is 86.0%  Using feature(s) [4, 6, 8, 10] accuracy is 85.0%  Using feature(s) [4, 5, 8, 10] accuracy is 80.0%  Using feature(s) [4, 5, 6, 10] accuracy is 75.0%  Using feature(s) [4, 5, 6, 8] accuracy is 81.0%  Feature set [5, 6, 8, 10] was best, accuracy is 86.0%  Using feature(s) [6, 8, 10] accuracy is 76.0%  Using feature(s) [5, 8, 10] accuracy is 88.0%  Using feature(s) [5, 6, 10] accuracy is 67.0%  Using feature(s) [5, 6, 8] accuracy is 86.0%  Feature set [5, 8, 10] was best, accuracy is 88.0%  Using feature(s) [8, 10] accuracy is 84.0%  Using feature(s) [5, 10] accuracy is 70.0%  Using feature(s) [5, 8] accuracy is 95.0%  Feature set [5, 8] was best, accuracy is 95.0%  Using feature(s) [8] accuracy is 88.0%  Using feature(s) [5] accuracy is 69.0%  (Warning, Accuracy has decreased! Continuing search in case of local maxima)  Feature set [8] was best, accuracy is 88.0%  Finished search! The best feature subset is [5, 8] , which has an accuracy of 95.0%  Time cost: 15.563661s |

Using backward elimination, I find the best feature set is [5, 8] with an accuracy of 95%. That is also a good result. I also do a leave one out cross validation on the dataset. The time cost for backward elimination is 15.56s

Trace of my original algorithm on cs\_205\_small56.txt

|  |
| --- |
| Welcome to Yuan Yao Feature Selection Algorithm.  Type in the name of the file to test: cs\_205\_NN\_datasets/cs\_205\_small56.txt  Type the number of the algorithm you want to run.  1) Forward Selection  2) Backward Elimination  3) Yuan's Special Algorithm.  3  This dataset has 10 features (not including the class attribute), with 100 instances.  Please wait while I normalize the data... Done!  Using no feature, the default rate is 79.0%  Beginning search.  Using feature(s) [1, 2] accuracy is 76.0%  Using feature(s) [1, 3] accuracy is 73.0%  Using feature(s) [1, 4] accuracy is 72.0%  Using feature(s) [1, 5] accuracy is 71.0%  Using feature(s) [1, 6] accuracy is 70.0%  Using feature(s) [1, 7] accuracy is 66.0%  Using feature(s) [1, 8] accuracy is 75.0%  Using feature(s) [1, 9] accuracy is 72.0%  Using feature(s) [1, 10] accuracy is 74.0%  Using feature(s) [2, 3] accuracy is 67.0%  Using feature(s) [2, 4] accuracy is 67.0%  Using feature(s) [2, 5] accuracy is 80.0%  Using feature(s) [2, 6] accuracy is 62.0%  Using feature(s) [2, 7] accuracy is 58.0%  Using feature(s) [2, 8] accuracy is 80.0%  Using feature(s) [2, 9] accuracy is 73.0%  Using feature(s) [2, 10] accuracy is 64.0%  Using feature(s) [3, 4] accuracy is 69.0%  Using feature(s) [3, 5] accuracy is 64.0%  Using feature(s) [3, 6] accuracy is 64.0%  Using feature(s) [3, 7] accuracy is 62.0%  Using feature(s) [3, 8] accuracy is 66.0%  Using feature(s) [3, 9] accuracy is 73.0%  Using feature(s) [3, 10] accuracy is 58.0%  Using feature(s) [4, 5] accuracy is 57.0%  Using feature(s) [4, 6] accuracy is 69.0%  Using feature(s) [4, 7] accuracy is 55.0%  Using feature(s) [4, 8] accuracy is 62.0%  Using feature(s) [4, 9] accuracy is 76.0%  Using feature(s) [4, 10] accuracy is 76.0%  Using feature(s) [5, 6] accuracy is 66.0%  Using feature(s) [5, 7] accuracy is 62.0%  Using feature(s) [5, 8] accuracy is 81.0%  Using feature(s) [5, 9] accuracy is 76.0%  Using feature(s) [5, 10] accuracy is 60.0%  Using feature(s) [6, 7] accuracy is 65.0%  Using feature(s) [6, 8] accuracy is 74.0%  Using feature(s) [6, 9] accuracy is 72.0%  Using feature(s) [6, 10] accuracy is 60.0%  Using feature(s) [7, 8] accuracy is 75.0%  Using feature(s) [7, 9] accuracy is 70.0%  Using feature(s) [7, 10] accuracy is 60.0%  Using feature(s) [8, 9] accuracy is 75.0%  Using feature(s) [8, 10] accuracy is 68.0%  Using feature(s) [9, 10] accuracy is 69.0%  Feature set [5, 8] was best, accuracy is 81.0%  Using feature(s) [5, 8, 1] accuracy is 93.0%  Using feature(s) [5, 8, 2] accuracy is 85.0%  Using feature(s) [5, 8, 3] accuracy is 96.0%  Using feature(s) [5, 8, 4] accuracy is 90.0%  Using feature(s) [5, 8, 6] accuracy is 86.0%  Using feature(s) [5, 8, 7] accuracy is 91.0%  Using feature(s) [5, 8, 9] accuracy is 90.0%  Using feature(s) [5, 8, 10] accuracy is 88.0%  Feature set [5, 8, 3] was best, accuracy is 96.0%  Using feature(s) [5, 8, 3, 1] accuracy is 88.0%  Using feature(s) [5, 8, 3, 2] accuracy is 93.0%  Using feature(s) [5, 8, 3, 4] accuracy is 86.0%  Using feature(s) [5, 8, 3, 6] accuracy is 81.0%  Using feature(s) [5, 8, 3, 7] accuracy is 93.0%  Using feature(s) [5, 8, 3, 9] accuracy is 88.0%  Using feature(s) [5, 8, 3, 10] accuracy is 83.0%  (Warning, Accuracy has decreased! Continuing search in case of local maxima)  Feature set [5, 8, 3, 2] was best, accuracy is 93.0%  Using feature(s) [5, 8, 3, 2, 1] accuracy is 85.0%  Using feature(s) [5, 8, 3, 2, 4] accuracy is 80.0%  Using feature(s) [5, 8, 3, 2, 6] accuracy is 81.0%  Using feature(s) [5, 8, 3, 2, 7] accuracy is 89.0%  Using feature(s) [5, 8, 3, 2, 9] accuracy is 90.0%  Using feature(s) [5, 8, 3, 2, 10] accuracy is 84.0%  (Warning, Accuracy has decreased! Continuing search in case of local maxima)  Feature set [5, 8, 3, 2, 9] was best, accuracy is 90.0%  Break searching since accuracy has decreased many times  Finished search! The best feature subset is [5, 8, 3] , which has an accuracy of 96.0%  Time cost: 5.846906s |

My original search algorithm is: search for every pair of two features of all the features and then do a forward search with the guaranteed best two features. Because of all the features in the dataset, only 2 are strongly related to the class. Therefore, I choose to search every pair of two features in order to get the best 2 features that are strongly related to the class. Therefore, my algorithm can give better result than the other two algorithms.

The result I got from my original algorithm is feature set [5, 8, 3] with an accuracy of 96%. I use k-fold cross validation with k=10 to do search on each pair of two features in order to speedup classification and do a leave one out cross validation on the following forward selection. The time cost for my original algorithm is 5.84s.

Result for cs\_205\_large56.txt:

|  |
| --- |
| Forward Selection:  Finished search! The best feature subset is [72, 48] , which has an accuracy of 96.0%  Time cost: 90.888684s    Backward Elimination:  Finished search! The best feature subset is [7, 34, 48, 59, 60, 61, 63, 66, 67, 81, 82, 85, 91, 93, 97, 99, 100] , which has an accuracy of 91.0%  Time cost: 1976.174461s  My original search algorithm:  Finished search! The best feature subset is [48, 72] , which has an accuracy of 96.0%  Time cost: 805.740074s |

I reset k=n to do a leave one out cross validation on all the steps of my original algorithm in order to get a better result. From the results we can see that the forward selection is the fastest and give a good result. The backward elimination works fine on small datasets but becomes useless on large datasets. My original algorithm runs slow but can have a better result than forward selection if the dataset is very large.