Feature Selection with Nearest Neighbor

Jiaqi Xiong jiaqixiong01137@gmail.com
04- December -2018

Contents

Feature Selection with Nearest Neighbor 1
I. Introduction 2
II. Algorithm 2
1. Prerequisite 2
1) K-Nearest Neighbor Classifier 2
2) K-Fold Cross Validation 3
2. Search Algorithm 3
1) Greedy Forward Selection (GFS) 3
2) Greedy Backward Elimination (GBE) 4
3) My Original Special Search Algorithm based on GFS 4
III. Comparison of Results from Three Search Algorithms on Two Sample Datasets 6
1. Results from GFS and GBE 6
1) On sample datasets 6
2) On small data 92 7
3) On large data 12 9
2. The Superiority of My Original Algorithm 10
1) Predictive accuracy 10
2) Speed 10
IV. Conclusion 11
V. Reference 11
VI. Appendix: A trace of the Forward Selection on a small dataset 12

Instructed by: Dr Eamonn Keogh eamonn@cs.ucr.edu

I. Introduction

The classification problem can be expressed as: Given a training database, predict the class label of a previously unseen instance. The nearest neighbor algorithm is a very simple, yet very competitive classification algorithm. But nearest-neighbor is hopelessly confused by irrelevant attributes because they all contribute to the similarity between examples. With enough irrelevant attributes, accidental similarity in the irrelevant dimensions swamps out meaningful similarity in the important ones, and nearest-neighbor becomes no better than random guessing. How do we mitigate the nearest neighbor algorithms sensitivity to irrelevant features?
We may consider using more training instances or asking an expert what features are relevant to the task, but a more expensive but smarter option is to "wrap" the attribute selection around the learner itself, with a hill-climbing search over the subsets of features. Using K-fold cross validation is a good way to choose the subset of features we need to adjust in the classifier. But we may find it hard to find the weak feature, and also may find spurious features.

Therefore, I generate a new algorithm that not only solves these problems but also speed up search. The following report presents my findings about feature search algorithms with k-nearest neighbor classifier through the process of project completion. It explores greedy forward selection search, greedy backward elimination search and my original search algorithm. To compare these 3 algorithms, they were tested on datasets SMALLtestdata__92.txt and LARGEtestdata__12 in Matlab R2019a, and the full code are available on my Github.

II. Algorithm

The whole algorithm can be thought as a wrapper that takes a k-nearest neighbor algorithm as an argument. The wrapper enumerates k-nearest neighbor classifiers according to the set of features selected. The sequence it visits the search tree each of whose nodes is a set of features according to 3 different greedy search algorithms below. For each node it visits, it uses k fold cross validation on the classifier to compute accuracy. This accuracy for each node is the heuristic function for greedy search using to estimate the merit of the state. Where there is a tie, we choose the simpler model.

1. Prerequisite

1) K-Nearest Neighbor Classifier

```
Nearest Neighbor
if the nearest instance to the previously unseen instance is class 1
    class is 1
else
    class is 2
```

Observe the pseud-code of the nearest neighbor algorithm above: It just consists of predicting the new instance's class based on the class of its nearest neighbor. However, nearest-neighbor

tends to overfitting: if we have the wrong class for a data point, it spreads to its entire metro area. Thus, we consider generalizing the nearest neighbor algorithm to the k- nearest neighbor (KNN) algorithm: a previously unseen instance is classified by finding its k, where k is typically chosen to be an odd number to avoid ties, nearest neighbors and letting them vote. Because it only goes wrong if most of the k nearest neighbors is noisy, k-nearest-neighbor is more robust than nearest-neighbor. The price, of course, is that its vision is blurrier: -Fine details of the frontier get washed away by the voting. When k goes up, variance decreases, but bias increases.

Therefore, although I programmed k-nearest neighbor classifier for generality, I was only willing to use k=1, that is, nearest neighbor classifier to test in this project to save time. (Fortunately, it seems that I made a good decision, since by trial and error the results with k=1 seems to be the most prospective.) Also, although this need not be the case, I assumed that the k-nearest neighbor algorithm uses the Euclidean Distance for, that is,

$$D(Q,C) \equiv \sqrt{\sum_{i=1}^{n} (q_i - c_i)^2}$$

2) K-Fold Cross Validation

The idea is that each instance serves double duty—as training instance and test instance. First, we split the dataset into k equal subsets. We then perform k rounds of learning; on each round one of k subsets is held out as a test set and the remaining instances are used as training data. The average test set score of the k rounds, which then be a better estimate than a single score, should be defined as accuracy shown below

$$\text{Accuracy} = \frac{\text{Number of correct classifications}}{\text{Number of instances in our database}}$$

The extreme is k = I, where I is the number of instances in the dataset, also known as leave-one-out cross-validation. Popular values for k are 5 and 10—enough to give an estimate that is statistically likely to be accurate, at a cost of 5 to 10 times longer computation time. However, although I programmed k-fold cross validation for generality, I was only willing to use k=I to test in this project for accuracy. (Don't worry. I will speed up by my original search algorithm.)

2. Search Algorithm

1) Greedy Forward Selection (GFS)

Greedy Forward Selection
Initial state: Empty Set-No features
Operators: Add a new feature.
Evaluation Function: K-fold cross validation.

As depicted above, we start with an empty set, and iterate, each time considering adding a feature with highest accuracy to the set, always record the best set of features, until all features are in the set.

2) Greedy Backward Elimination (GBE)

Greedy Backward Elimination
Initial state: Universal Set-All features
Operators: Add a new feature.
Evaluation Function: K-fold cross validation.

Likewise, as shown above, this is a hill-climbing search that keeps deleting attributes as long as that doesn't hurt k-nearest-neighbor's accuracy.

3) My Original Special Search Algorithm based on GFS

Through GFS and GBE discussed previously, we can easily find the two strong features, however, the small size of the dataset, means I might have missed the weak feature. Meanwhile, with so many extra features to search through, some random features will look good by chance. That's to say, I might have added a random feature that adds a little bit of spurious accuracy.

Additionally, although my computer using Matlab can do feature search by GFS and GBE on the large datasets given by instructor in under 3 minutes, however, for some real problems, we might have millions of instances, and thousands of features. Then the same code might take a long time which we cannot bear.

Thus, my original search algorithm attempt be better in either of two ways:
It can be faster through faster search.
It can give better results through better search.

Then I simply combine faster search and better search to let my original search algorithm improved in both two ways. And we will see that my original special search algorithm should have been better even if the results do not support this.

i. Faster Search Algorithm

```
xjq_validation(best-so-far-accuracy)
mistakes=0;
for each round
    train classifier through training set and predict instances in test
set, and keep track of how many mistakes I have made so far
    if mistakes > 100- best-so-far- accuracy,
    break out of loop, and return 0;
return 100-mistakes
```

The faster search algorithm is almost the same as the GFS except the evaluation function as shown above. This idea is similar to the Alpha-Beta pruning: If a possibility is bad, we don't need to find out exactly how bad it is. That's to say, once we can make sure the mistakes of this node is more than that of the best so far node, we immediately assume the accuracy for this

node is 0. In this way, our time will be saved and the validation of our search won't be hurt. Indeed, from the results discussed later, we will see this algorithm does work.

ii. Better Search Algorithm

In GFS and GBE we will add and keep a new feature, even if it only gets one more instance correct. However, we have dozens of irrelevant features. It is very likely that one or two of them will classify one or two extra data points by random chance. This is not what I expect to happen. While the spurious features happened to help a little bit on these 200 instances, they will hurt unseen data we will see in the future. This idea to solve this problem is called resampling as shown below.

```
Better Search Algorithm
making k copies of the dataset
for each copy
    randomly delete d% of the data
    run GFS(copy)
choose strong features that look good in all copies
for each copy
    delete those strong features temporarily
    run GFS(copy)
choose weak features that look good in most copies
```

In this project, for simplifying the test, I made 3 different versions for datasets for time limitation by randomly deleting 5% of the data. Now each of the three copies is very similar to the true dataset, but if a spurious feature happens to look good in one copy, it is very unlikely to look good in the other two copies, but the weak feature will tend to show up a lot more than we might expect by chance. Therefor, if we choose features that tend to look good in all copies, we can find better results.

iii. Special Search Algorithm

```
Special Search Algorithm
making k copies of the dataset
for each copy
    randomly delete d% of the data
    run Faster Search(copy)
choose strong features that look good in all copies
for each copy
    delete those strong features temporarily
    run Faster Search(copy)
choose weak features that look good in most copies
```

III. Comparison of Results from Three Search Algorithms on Two Sample Datasets

I got two different random personal datasets "SMALLtestdata__92.txt" and "small LARGEtestdata__12.txt" to work with. The data files are in the following format: ASCII Text, IEEE standard for 8 place floating numbers (space delimited). Class labels are in the first column either a "1" or "2". The second column up to the last column are the features. Small is 200 instances, and 10 features. Large is 200 instances, and 100 features. I got the hint that all the features are random and irrelevant, except that two are strongly related to the class (and to each other), and one is weakly related to the class.

And from now on, I will compare three search algorithms discussed above in terms of their predictive accuracy and speed.

1. Results from GFS and GBE

1) On sample datasets

Comparing the real values given by instructor and my own results on public datasets as shown in Table 1, we can find while there exists some slightly differences, it can be accepted since at least two features are founded through GFS and at least one feature through GBE. Thus, I am confident to do GFS and GBE on my personal datasets.

Best features/Accuracy		GFS	GBE	Real Value
SMALLtestdata	108	{6 4}/94%	{4 6}/94%	{6 5 4}/91%
	109	{7 2}/97%	{2 7}/97%	{7 9 2}/89%
	110	{6 9}/96%	{6 9}/96%	{6,4,9} /92.5%
	SAMPLE	{4 10 5}/95%	{4 10}/93.5%	{4 5 10}/95%
LARGEtestdata	108	{46 95 94}/95.5%	{32 46}/89%	{46 26 95}/93%
	109	{72 2}/96%	{1 2 3 4 6 7 8 10 11 12 13 15 17 18 20 24 26 30 31 32 33 34 37 38 44 45 46 47 50 61 63 64 67 68 71 73 78 79 81 84 88 89 92 93 96 99 100}/88.5%	{72 19 2}/92.5
	110	{1 6} 94%	{5 6 7 10 11 13 16 20 21 24 26 27 29 37 40 44 47 56 59 61 63 65 74 77 82 88 91 93 94}/90.5%	{1 66 6}/93.5%

Table 1: The Real Values and My Test Results on Public Datasets

2) On small data 92

The search procedures by GFS and GBE are shown in Figure 1 and Figure 2, respectively. And the table 2 summarize my results.

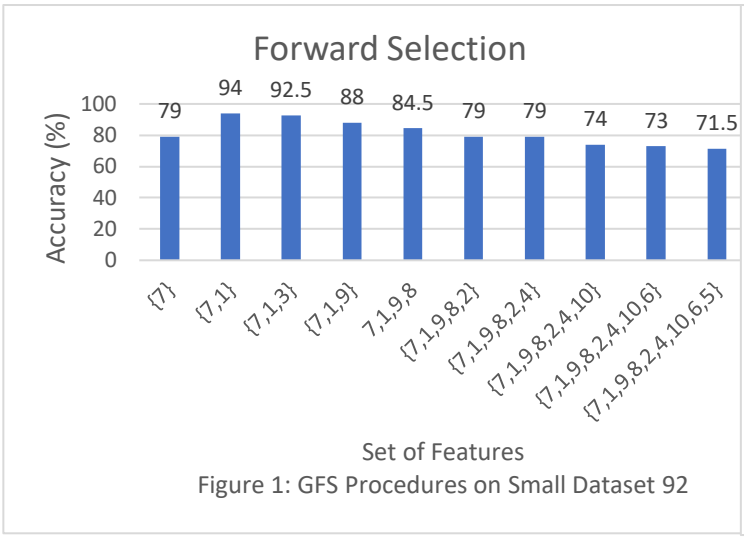


Figure 1: GFS Procedures on Small Dataset 92

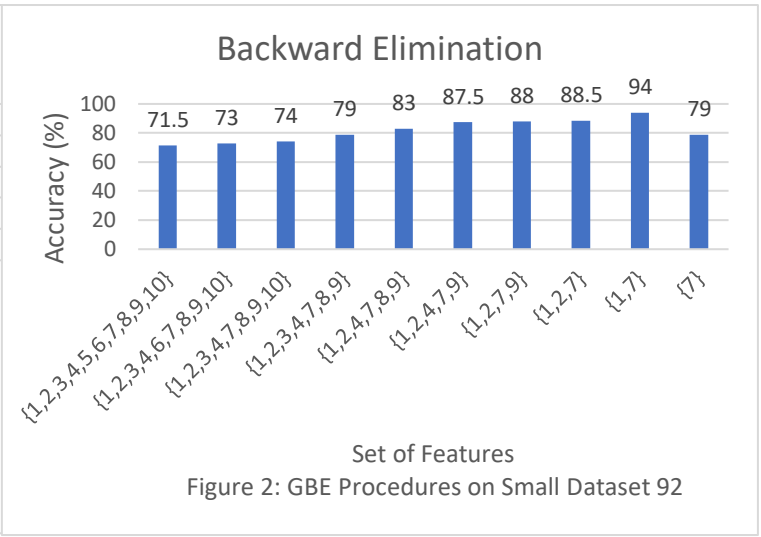


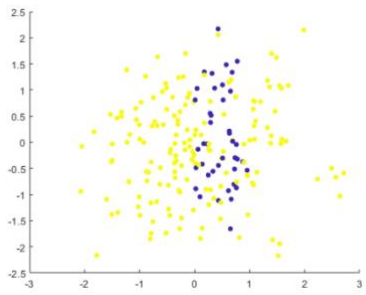
Figure 2: GBE Procedures on Small Dataset 92

Search Algorithm	GFS	GBE	Special Search			Final Result
Best features/ Accuracy	{7 1}/94%	{1 7}/94%	Version 1	Version 2	Version 3	{7 1 3}/92.5%
			{7 1}/92.6316%	{7 1}/94.2105%%	{7 1}/93.6842%	
			After Deleting Feature {7 1}			
			Version 4	Version 5	Version 6	
			{8 3 5 2 10}/74.2105%	{8 3 5 2 9}/76.3158%	{3}/73.1579%	

Table 2: My Test Results on Small Dataset 92

The plot of two features are shown in Figure 3-6, after analysis of the plots, I am confident of my final result for small dataset 92 and that later I have checked with instructor guaranteeing my success.

Figure 3: Plot of the Two Strong Features 7 and 1



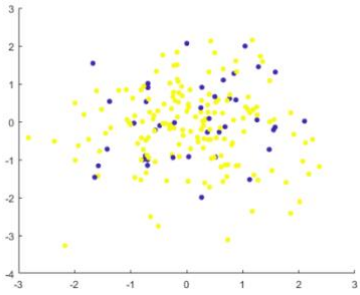
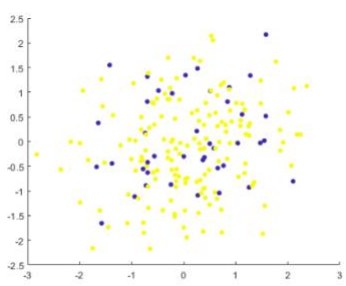


Figure 4: Plot of Two Irrelative Features 5 and 8

Figure 5: Plot of Weak Feature 3 with a Irrelative Feature 5



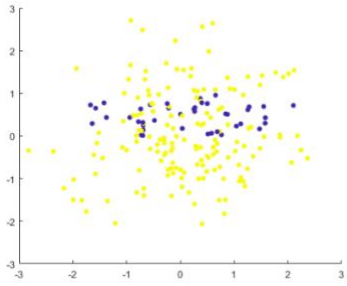


Figure 6: Plot of a Strong Feature 7 with a Irrelative Feature 5

3) On large data 12

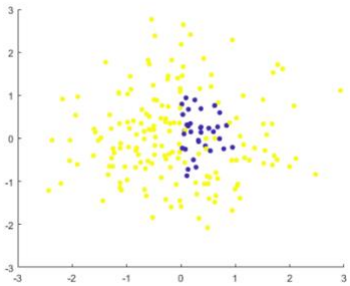
Search Algorithm	GFS	GBE	Special Search			Final Result
			Version 1	Version 2	Version 3	
Best features/ Accuracy	{76 49 62}/91%	{1 2 3 4 5 6 7 8 9 10 13 15 18 19 20 21 23 24 25 26 27 28 29 32 35 36 39 40 41 42 44 46 48 50 52 53 56 64 66 68 69 74 75 78 80 81 82 84 92 96 97 98 99}/90.5%	{76 2 49 15 25 39 32 62 34}/92.1053%	{76 49 62 15}/92.1053%	{76 49 62 15 8}/91.0526%	{76 49 62}/91%

Table 3: My Test Results on Large Dataset 12

From the table 3, GFS and GBE give different answers, the one with the highest accuracy is most likely to be correct. Thus, we choose the result by GFS. Since after doing GFS, we got 3 features with high accuracy, we need not search for weak feature or eliminate spurious features anymore according to the hint that only three relative features.

The plot of two features are shown in Figure 7-10, after analysis of the plots, I am confident of my final result for large dataset 12 and that later I have checked with instructor guaranteesing my success.

Figure 7: Plot of the Two Strong Features 76 and 49



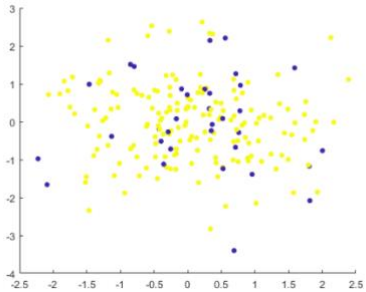
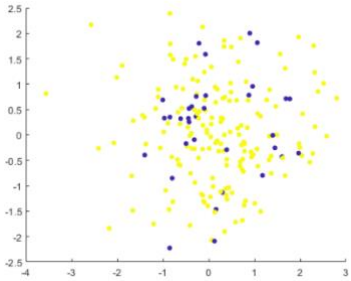


Figure 8: Plot of Two Irrelative Features 27 and 89

Figure 9: Plot of Weak Feature 62 with a Irrelative Feature 27



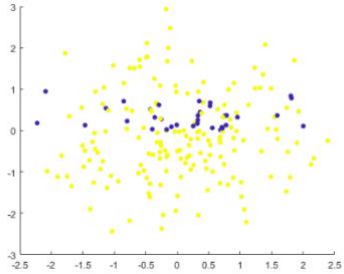


Figure 10: Plot of a Strong Feature 76 with a Irrelative Feature 27

2. The Superiority of My Original Algorithm

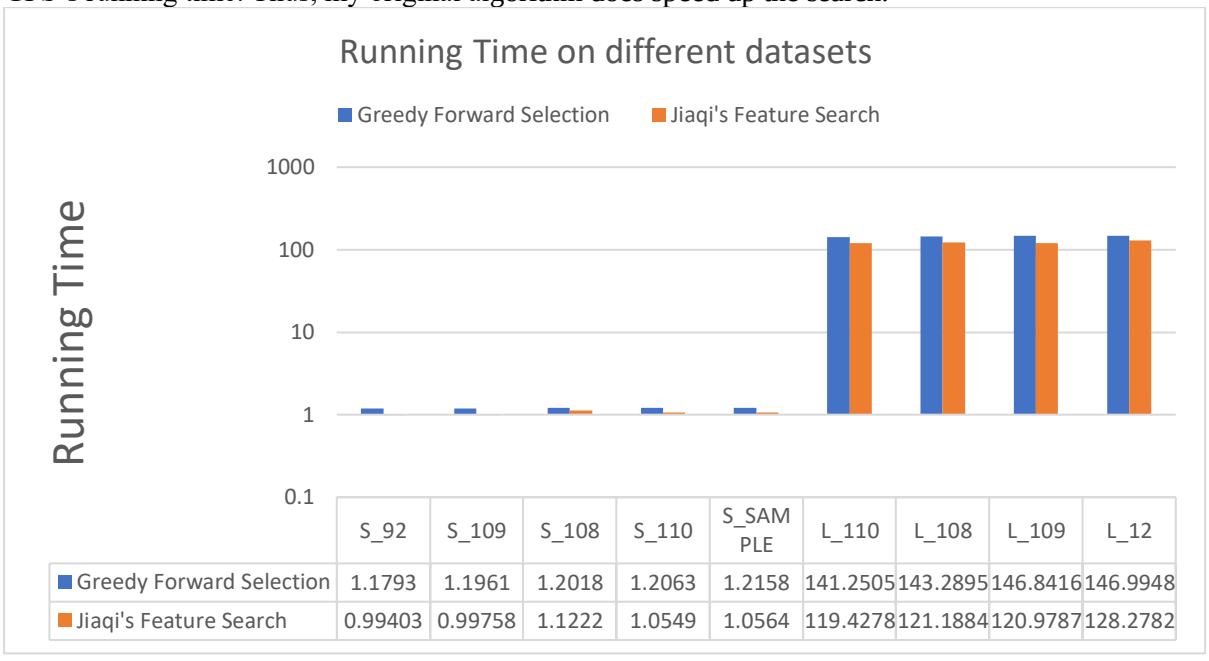
We can show the advantages of my original algorithm by estimating the accuracy of our classifier and considering the time requirements

1) Predictive accuracy

From the table 2, we can see that either by GFS or GBE we cannot find the weak feature 3. But through my special search algorithm, we can see that the two good features 7 and 1 show up in all three runs. Thus, we know 7 and 1 are strong features. After delete strong features, features 3 shows up in all three runs. Thus, we know 3 is weak features. And by Figure 3-6, I make sure that my special search algorithm did better job than GFS and GBE.

2) Speed

Figure 10 below provide algorithms running time computed by Matlab automatically. From the table, we can know that my faster search algorithm's running time is apparently shorter than GFS's running time. Thus, my original algorithm does speed up the search.



	S_92	S_109	S_108	S_110	S_SAMPLE	L_110	L_108	L_109	L_12
Greedy Forward Selection	1.1793	1.1961	1.2018	1.2063	1.2158	141.2505	143.2895	146.8416	146.9948
Jiaqi's Feature Search	0.99403	0.99758	1.1222	1.0549	1.0564	119.4278	121.1884	120.9787	128.2782

Figure 11: GFS and Jiaqi's Feature Search(FS) Running Time on different datasets

IV. Conclusion

From what has been discussed and tested above, we conclude the result that

A. The best set of features for small dataset 92 is {7 3 1} with the accuracy 92.5% and for large dataset 12 is {76 49 62} which has an accuracy of 91%.
B. Forward and backward selection can give different answers and for large dataset 12 forward selection is most likely to be best, which may mean this dataset with lowly correlated features.
C. My expectation that my own search algorithm are faster and better than GFS and GBE are consistent with practical results.

V. Reference

1. Russell, S. J., & Norvig, P. (2016). Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited,. (for exact definition of KNN and k-fold cross validation)
2. Domingos, Pedro. The master algorithm: How the quest for the ultimate learning machine will remake our world. Basic Books, 2015. (for more information about features selection)
3. Slides “Heuristic Search”, “Machine Learning 1” and “Machine Learning 2” by Dr Eamonn Keogh (for review of the KNN algorithms, standards to estimate a classifier)

All the important code is original. Subroutines that are not completely original are

```
function  feature_search_demo(data)
current_set_of_features = []; % Initialize an empty set
for i = 1 : size(data,2)-1
    disp(['On the ',num2str(i),'th level of the search tree'])
    feature_to_add_at_this_level = [];
    best_so_far_accuracy    = 0;
     for k = 1 : size(data,2)-1
       if isempty(intersect(current_set_of_features,k)) % Only consider adding, if not
already added.
        disp(['--Considering adding the ', num2str(k),' feature'])
        accuracy = leave_one_out_cross_validation(data,current_set_of_features,k+1);

        if accuracy > best_so_far_accuracy
            best_so_far_accuracy = accuracy;
            feature_to_add_at_this_level = k;
        end
      end
     end
    current_set_of_features(i) =  feature_to_add_at_this_level;
    disp(['On level ', num2str(i),' i added feature ', num2str(feature_to_add_at_this_level), ' to
current set'])
 end
end
```

They are provided Dr Eamonn Keogh.

VI. Appendix: A trace of the Forward Selection on a small dataset

Here is the first and last page of my trace on the small dataset-'SMALLtestdata__92':

```
>> Feature_Selection_with_Nearest_Neighbor
Welcome to Jiaqi Xiong Feature Selection Algorithm.
Type the number of the algorithm you want to run:
1)	Forward Selection
2)	Backward Elimination
3)	Jiaqi's Faster Algorithm.
4)	Jiaqi's Better Algorithm.
5)	Jiaqi's Special Algorithm.1
This dataset has 10 features (not including the class attribute), with 200 instances.
Type integer k1, the number of equal sized sections into which you want to divide the dataset for k fold cross validation:200
Type odd k2, the number of neighbors who should vote for the class of unseen instance: 1
Running1nearest neighbor with all 10 features, using 200 fold cross validation, I get an accuracy of 71.5%
Beginning Forward search.
On the 1th level of the search tree
--Considering adding the 1 feature, accuracy is 67.5%
--Considering adding the 2 feature, accuracy is 67%
--Considering adding the 3 feature, accuracy is 70%
--Considering adding the 4 feature, accuracy is 70.5%
--Considering adding the 5 feature, accuracy is 65%
--Considering adding the 6 feature, accuracy is 67%
--Considering adding the 7 feature, accuracy is 79%
--Considering adding the 8 feature, accuracy is 75%
--Considering adding the 9 feature, accuracy is 63%
--Considering adding the 10 feature, accuracy is 70%
On level 1 I added feature 7 to current set
On the 2th level of the search tree
--Considering adding the 1 feature, accuracy is 94%
--Considering adding the 2 feature, accuracy is 73.5%
……
--Considering adding the 6 feature, accuracy is 77%
--Considering adding the 10 feature, accuracy is 77%
(Warning, Accuracy has decreased! Continuing search in case of local maxima)
On level 6 I added feature 2 to current set
On the 7th level of the search tree
--Considering adding the 4 feature, accuracy is 79%
--Considering adding the 5 feature, accuracy is 77.5%
--Considering adding the 6 feature, accuracy is 75.5%
--Considering adding the 10 feature, accuracy is 77%
(Warning, Accuracy has decreased! Continuing search in case of local maxima)
On level 7 I added feature 4 to current set
On the 8th level of the search tree
--Considering adding the 5 feature, accuracy is 72.5%
--Considering adding the 6 feature, accuracy is 71.5%
--Considering adding the 10 feature, accuracy is 74%
(Warning, Accuracy has decreased! Continuing search in case of local maxima)
On level 8 I added feature 10 to current set
On the 9th level of the search tree
--Considering adding the 5 feature, accuracy is 71.5%
--Considering adding the 6 feature, accuracy is 73%
(Warning, Accuracy has decreased! Continuing search in case of local maxima)
On level 9 I added feature 6 to current set
On the 10th level of the search tree
--Considering adding the 5 feature, accuracy is 71.5%
(Warning, Accuracy has decreased! Continuing search in case of local maxima)
On level 10 I added feature 5 to current set
Finished Forward search!! The best feature subset is {2 1 } which has an accuracy of 94%
running time = 1.1793
```