Nearest Neighbor Project                        CS 170 Intro to Artificial Intelligence

Salman Bana                        Instructor: Dr Eamonn Keogh

ID 861164246

[sbana002@ucr.edu](mailto:sbana002@ucr.edu)

15 December 2017

In completing this assignment, I consulted…

* Dr. Eamonn Keogh’s slides for understanding how to implement the Nearest Neighbor algorithm, and for ideas on a custom algorithm.
* Various mathworks pages in understand how to randomize arrays, ask for user input, and print the accuracy and features used at each step of the way. <https://www.mathworks.com/help/matlab/ref/randperm.html>  
  <https://www.mathworks.com/help/matlab/ref/fprintf.html>

function [features, prev\_accuracy] = backward\_elimination(X, Y, cond, not)

%Backwards Elimination

warning = ['Warning, Accuracy has decreased! ' ...

'Continuing search in case of local maxima \n'];

fprintf('Beginning search. \n');

n = size(X, 2);

if isempty(not)

worst = [];

else

worst = not;

end

prev\_accuracy = 0;

for remove = 1:n

index = 0; best\_accuracy = 0;

for j = 1:n

flag = true;

if ~isempty(worst)

for k = 1:length(worst)

if isequal(j, worst(k))

%disp(j)

%disp(best(k))

flag = false;

end

end

end

tempX = X;

tempX(:, [worst, j]) = [];

if flag

accuracy = cross\_validation(tempX, Y);

%temp\_X = X(error\_vector, :); temp\_Y = Y(error\_vector, :);

%X(error\_vector, :) = []; Y(error\_vector, :) = [];

%X = [temp\_X; X]; Y = [temp\_Y; Y];

fprintf('Removing features ');

fprintf('%i,', [worst, j]);

fprintf(' the accuracy is %f \n', accuracy);

if accuracy > best\_accuracy

best\_accuracy = accuracy;

index = j;

end

end

end

if cond

if prev\_accuracy > best\_accuracy + 3

break;

elseif (prev\_accuracy <= best\_accuracy + 3 ...

&& prev\_accuracy > best\_accuracy)

%Add worst to index

worst = [worst, index];

fprintf(warning);

else

prev\_accuracy = best\_accuracy;

worst = [worst, index];

true\_worst = worst;

end

else

if prev\_accuracy > best\_accuracy

break;

else

prev\_accuracy = best\_accuracy;

worst = [worst, index];

true\_worst = worst;

end

end

end

features = 1:n;

features(true\_worst) = [];

end

function accuracy = cross\_validation(X, Y)

error\_vector = []; m = size(X, 1);

for i = 1:m

min\_distance = 100000; index = 0;

for j = 1:m

if i ~= j

current\_distance = sqrt(sum((X(j, :) - X(i, :)) .^ 2));

if current\_distance <= min\_distance

min\_distance = current\_distance;

index = j;

end

end

end

if ~isequal(Y(i), Y(index))

error\_vector = [error\_vector index];

%if accuracy < base

%break

%end

end

end

accuracy = (m - length(error\_vector)) \* 100 / m;

end

%Salman Bana’s Search Algorithm

n = size(X, 2);

best\_features = []; best\_accuracy = 0;

for i = 1:floor(n / 5)

%Shuffle the features

idx = randperm(n);

temp\_X = X(:, idx);

%Create two arrays of size (n / 5). One for the features

%One for accuracy

total\_features = []; update = 1; not = [];

%feature\_cell = cell(1, floor(n / 5));

%accuracy\_array = zeros(1, 10);

for k = 1:floor(n / 5)

features = backward\_elimination(...

temp\_X(:, update:(5 \* k)), Y, true, not);

total\_features = [total\_features, features + (5 \* (k - 1))];

%feature\_cell(1, k) = features + ((k - 1) \* 5);

%accuracy\_array(1, k) = accuracy;

update = update + 5;

end

not = 1:n;

not(total\_features) = [];

[features, accuracy] = backward\_elimination(...

temp\_X, Y, true, not);

for siz = 1:length(features)

features(siz) = idx(features(siz));

end

if accuracy >= best\_accuracy

best\_accuracy = accuracy;

best\_features = features;

end

end

Welcome to Salman Bana Feature Selection Algorithm.

Type in the name of the file to test: **Small6.txt**

Type the number of the algorithm you want to run.

1) Forward Selection

2) Backward Elimination

3) Salman Special Algorithm

**1**

This dataset has 10 features (not including the class attribute), with 100 instances.

Please wait while I normalize the data… Done!

Running nearest neighbor with all 10 features,using “leaving-one-out” evaluation, I get an accuracy of 65.0%

Beginning search.

Using features 1, the accuracy is 68.0%

Using features 2, the accuracy is 72.0%

Using features 3, the accuracy is 60.0%

Using features 4, the accuracy is 88.0%

Using features 5, the accuracy is 78.0%

Using features 6, the accuracy is 66.0%

Using features 7, the accuracy is 69.0%

Using features 8, the accuracy is 65.0%

Using features 9, the accuracy is 62.0%

Using features 10, the accuracy is 57.0%

Feature set 4 was best, accuracy is 88.0%

Using features 4,1, the accuracy is 81.0% Using features 4,2, the accuracy is 79.0%

Using features 4,3, the accuracy is 81.0%

Using features 4,5, the accuracy is 95.0%

Using features 4,6, the accuracy is 85.0%

Using features 4,7, the accuracy is 79.0%

Using features 4,8, the accuracy is 76.0%

Using features 4,9, the accuracy is 77.0%

Using features 4,10, the accuracy is 81.0%

Feature set 4,5 was best, accuracy is 95%

Using features 4,5,1, the accuracy is 82.0%

Using features 4,5,2, the accuracy is 85.0%

Using features 4,5,3, the accuracy is 88.0%

Using features 4,5,6, the accuracy is 90.0%

Using features 4,5,7, the accuracy is 85.0%

Using features 4,5,8, the accuracy is 88.0%

Using features 4,5,9, the accuracy is 86.0%

Using features 4,5,10, the accuracy is 90.0%

Feature set 4,5,10 was best, accuracy is 90%

Finished search!! The best feature subset is 4, 5 which has an accuracy of 95.0%

In class, we discussed one machine learning algorithm called k nearest neighbor which determines the class a test point using the class of the nearest training point to it. We learned that it is important to select relevant and useful features because extraneous and irrelevant features significantly lower the accuracy of the algorithm. That begs the question, how do we choose what features to choose?

We discussed two basic algorithms in class: forward selection and backwards elimination. Forward selection is a greedy algorithm that starts with an empty set (no features), finds the best single feature (highest accuracy on cross-validation set), and adds it to the set until the accuracy has reached some maxima. Forward selection is good when the greedy selection algorithm is optimal for that problem, however, in some datasets, features can be highly correlated and a combination of two or more weaker features leads to higher accuracy.

On the other hand, backward elimination starts with a set of all features, finds the feature for which, removed, the set has the highest accuracy, and then continues to remove in that fashion until the accuracy has reached some maxima. Some of the shortcomings of backwards elimination are that with the inclusion of all of the features, strong features found in forward selection do not perform as well when muddled with irrelevant features.

In addition, backwards elimination also usually more computationally costly than forwards selection. For example, if there are four relevant features in a dataset of fifty features, forward selection will run through look through fifty features and add one, run through forty-nine features and add one … until it adds the fourth feature, looks through forty-six features and then stops. This equates to sum (46:50) = 240 cross-validation passes. In contrast, backwards elimination would remove features until there are only four features in the set, equating to sum (4:50) = 1269 passes which is over five times as large.

In this project, we were tasked to create our own custom search algorithm that would either improve the runtime or improve the optimality of the forward selection or backward elimination algorithms. I set out to improve the optimality of the backward elimination algorithm.

Starting with how and why I designed this algorithm as such, we must first look at the results of the two optimization algorithms on my assigned small data set, Small6.txt. Testing on 100 instances by 10 features dataset, both the forward selection and backward elimination algorithm found that the best feature set was 4, 5 with a 95.0% accuracy. However, on the larger dataset with 100 instances and 50 features, Large30.txt, the two algorithms yielded different results; the forward selection algorithm selected 24, 46, 17 to be the best feature set with a 91% accuracy while the backwards elimination algorithm narrowed the feature set down to 4, 5, 8, 18, 19, 22, 23, 25, 27, 36, 40, 47, 48 with an 83% accuracy.

From comparing the results of the two algorithms on both datasets, I came to a conclusion: On smaller datasets with fewer features, backwards elimination performs comparably to the greedy forward selection algorithm while still conferring the benefits of finding highly correlated features. This is what led me to come up with my custom algorithm which attempts to optimize the backwards elimination algorithm so that the algorithm has the benefits/capabilities of both the forward selection and backwards elimination algorithm.

My custom search algorithm is as follows:

1. Separate the dataset into random groups of 5 features and perform backwards elimination one each group
2. Consolidate all of the features returned from running backwards elimination on the groups of 5 into one feature set and perform backwards elimination on the combined feature set.
3. Record the accuracy and feature set
4. Re-run steps 1-3 a few times (I chose n / 5 times where n is the number of features in the data set) using different random groupings of 5
5. Report the feature set with the highest accuracy among the multiple runs.

While 5 is an arbitrary number that I chose for the size of the groups, the small dataset showed that even with 10 features, the backwards elimination algorithm got the same result as the forward selection algorithm. After step 1, the number of features was consistently reduced to about 40% of the original dataset. In the case of the large dataset, approximately 20 features out of the original 50 remained. For very large datasets, one could repeat step 1 with the remaining features in order to further reduce the number of features, but I elected not to since the large data set only consists of 50 features. Finally, step 4 is necessary because it minimizes the random chance that a strong feature is eliminated because it is paired with other strong features.

I ran my custom algorithm 3 times which resulted in feature sets [17, 46, 24], [36, 46,17], and [46, 24] with an accuracy of 91.0%, 84%, and 88%. This coincides with my results from the forward selection algorithm, and vastly improves upon the backwards elimination algorithm.

As an added bonus, while my aim was to optimize the usage of the backwards elimination algorithm, breaking up the features into n / 5 groups of 5 and running backwards elimination on those groups has a faster runtime than performing backwards elimination on a dataset with n features. The maximum amount of cross-validation passes the large dataset needs to make is sum(1:n) = n2 / 2 = O(n2), while the maximum for the groups is (n / 5) \* sum(1:5) = 15n / 5 = 3n = O(n). If the number of reruns in step 4 is kept constant rather than variable like I used, this algorithm scales very well and has the potential to be faster than the original backwards elimination algorithm.

**Large Dataset 30:**

Strong features: 24, 46

Weaker features: 17

**Small Dataset 6:**

Strong features: 4, 5