# Project 2 (Feature Selection with Nearest Neighbor)
# iLearn submission of code and report due on Tuesday, June 2nd by 11:59 pm

As we have seen in class this quarter, the nearest neighbor algorithm is a very simple, yet very competitive classification algorithm. It does have one major drawback however; it is very sensitive to irrelevant features. With this in mind you will code up the nearest neighbor classifier, and then use it inside a "wrapper" which searches for the best subset of features.

For feature search, you can choose any of the following search methods:

1) Forward Selection
2) Backward Elimination

Don't be scared by the phrase "search algorithm". We discussed forward-selection in class (see the slides for MachineLearning2_featureSelection). Backward-elimination is very similar: it starts with the full set of features and removes one feature at a time. Both forward-selection and backward-elimination are greedy.

*Optional for extra points: You can also implement your original search algorithm that is better than the above two. It could be better in either (or both) of two ways:*

1) *It could be faster.*
2) *It could give better results.*

*So you need to have also implemented both forward selection and backward elimination to be able to make comparisons. In your report, you must clearly explain why your original algorithm is better. Without a clear explanation of this, you won't get the extra points.*

The criteria for evaluating a feature subset is the validation score. Instead of k-fold cross validation, we use the leave-one-out validation method. Here is an example of how it works: Assume we have 5 training instances and we are using 3 features:

| Instance ID | Class Label | Feature1 | Feature2 | Feature3 |
|---|---|---|---|---|
| 0 | 1 | 0.01 | 0.02 | 0.02 |
| 1 | 2 | 0.01 | 0.01 | 0.03 |
| 2 | 1 | 0.02 | 0.03 | 0.02 |
| 3 | 1 | 0.03 | 0.02 | 0.02 |
| 4 | 2 | 0.05 | 0.01 | 0.05 |

First we leave out instance 0 and use instances 1-4 as training data for our nearest neighbor classifier. Then we test our classifier (trained on instances 1-4) on instance 0: As input, we give it the feature set associated with instance 0, which is (0.01,0.02,0.02) and ask it to classify it (output is the predicted class which is either 1 or 2). We already know the correct answer (1) and can check whether the classifier predicted the class correctly or made a mistake. If it says class 1, it predicted the class correctly. Otherwise it made a mistake.

We repeat the above procedure for all instances. i.e., we leave out instance1, train the classifier on instances {0,2,3,4} and ask it to predict the class label for instance1. If it says class 2, it predicted the class correctly. Otherwise it made a mistake. We leave out instance 2, train on instances {0,1,3,4} and ask the classifier to predict class label for instance 2. If it says class 1, it predicted the class correctly. Otherwise it made a mistake. etc….

At the end, we count how many times the classifier was correct. Let's say it was correct 3 out of 5 times. Therefore, the accuracy of our classifier by using these 3 features is 3/5=0.6 or 60%

The goal of feature search is to find the set of features that maximize the accuracy. Ideally, we would like to test the accuracy of our classifier for all possible subsets of features and choose the subset that results in the highest accuracy. i.e., if our data set comes with 4 possible features (f1,f2,f3,f4), we run the above procedure 15 times, once for each of the following subset of features: {f1}, {f2},{f3},{f4}, {f1,f2}, {f1,f3},{f1,f4},{f2,f3},{f2,f4},{f3,f4},{f1,f2,f3}, {f1,f3,f4}, {f1,f2,f4},{f2,f3,f4}{f1,f2,f3,f4}. However, this kind of exhaustive search becomes infeasible for larger number of features ($2^n$-1 possible feature subsets). Therefore, we need to use a search algorithm.

So you need to implement:

1. The nearest neighbor classifier (All you need to do is to keep the training instances in memory and when a new data point is given for classification, compute its distance to the training points and return the class label of the nearest training point)

2. The Leave-one-out-validator: It uses the training data and the classifier. Given a subset of features as input, it returns an accuracy score as output.

3. The search algorithm: It starts from an initial state (in forward selection, it is an empty set and in backward elimination, it is the set of all features), adds/removes features to/from the previous set and evaluates the resulting feature subsets using the validator (assigns an accuracy score to them) and chooses the highest-accuracy option and continues until no more features can be added/removed. At the end, it reports the feature subset with highest accuracy as the best feature subset.

**Please make sure your code is as modular as possible so that one can plug-in different classifiers, validators and search algorithms.**

To make life simple, you can assume the following. I will only give you datasets that have two classes. I will only give you datasets that have continuous features (although you **must normalize** them).

The data files will be in the following format. ASCII Text, IEEE standard for 8 place floating numbers. This is a common format; you should be able to find some code to load the data into your program, rather that writing it from scratch (as always, document borrowed code); you can also share the code for this on Piazza. The first column is the class, these values will always be either "1"s or "2"s. The other columns contain the features, which are **not** normalized. There may be an arbitrary number of features (for simplicity I will cap the maximum number at 64). There may be an arbitrary number of instances (rows), for simplicity I will cap the maximum number at 2,048. Below is a trivial sample dataset. The first record is class "2", the second is class "1" etc. This example has just two features.

```
2.0000000e+000  1.2340000e+010  2.3440000e+000
1.0000000e+000  6.0668000e+000  5.0770000e+000
2.0000000e+000  2.3400000e+010  3.6460000e+000
1.0000000e+000  4.5645400e+010  3.0045000e+000
```

Think carefully before you start coding this. Students in the past seem to have made this more complicated than it need be. In particular, in Matlab one should be able to write the nearest neighbor algorithm in 8 lines of code, and the 3 search algorithms in another 17 lines of code. C++ and Java programs tend to be longer, but even so, I would be surprised if this took more than 100 lines of code (although you won't be penalized for this).

**Very important**: **Make sure your nearest neighbor algorithm is working correctly before you attempt the search algorithms.** We will provide some test datasets for this purpose.

You may use some predefined utility routines, for example sorting routines. However, I expect all the major code to be original. You must document any book, webpage, person or other resources you consult in doing this project (see the first day's handout).

You may consult colleagues at a high level, discussing ways to implement the tree data structure for example. But you may **not** share code. At most, you might illustrate something to a colleague with pseudocode.

On iLearn, you will submit:

1. A printout of your code.
2. A trace of your program on your own SMALL dataset with BOTH **forward** and **backward** algorithms.
3. A report which summaries **your findings**. You need to **compare the search algorithms** on 4 datasets:
    a. The initial small and large datasets that I gave to everyone along with the correct answer (to test their code) and
    b. Your own small and large datasets.

You must keep the evolving versions of your code, so that, if necessary you can demonstrate to the course staff how you went about solving this problem (in other words, we may ask you to prove that you did the work, rather than copy it from somewhere).

Your program should have a trace like the one below, so that it can be tested.

Welcome to Bertie Woosters (change this to your name) Feature Selection Algorithm.
Type in the name of the file to test : **Bertie_test_2.txt**

This dataset has 4 features (not including the class attribute), with 345 instances.

Please wait while I normalize the data… Done!

Type the number of the algorithm you want to run (if you have implemented more than one algorithm, otherwise, just print the name of the algorithm you have implemented).

1) Forward Selection
2) Backward Elimination
3) Bertie's Special Algorithm.

**1**

Running nearest neighbor with all 4 features, using "leaving-one-out" evaluation, I get an accuracy of 75.4%

Beginning search.

Using feature(s) {1} accuracy is 45.4%
Using feature(s) {2} accuracy is 63.7%
Using feature(s) {3} accuracy is 71.4%
Using feature(s) {4} accuracy is 48.1%

Feature set {3} was best, accuracy is 71.4%

Using feature(s) {1,3} accuracy is 48.9%
Using feature(s) {2,3} accuracy is 70.4%
Using feature(s) {4,3} accuracy is 78.1%

Feature set {4,3} was best, accuracy is 78.1%

Using feature(s) {1,4,3} accuracy is 56.9%
Using feature(s) {2,4,3} accuracy is 73.4%

(Warning, Accuracy has decreased! Continuing search in case of local maxima)
Feature set {2,4,3} was best, accuracy is 73.4%

Using feature(s) {1,2,4,3} accuracy is 75.4%

Finished search!! The best feature subset is {4,3}, which has an accuracy of 78.1%