**(This file may get updated)**

# Part II: Implementing the actual evaluation function and NN-classifier

In this part of the project, you are going to implement the Nearest Neighbor Classifier and the actual evaluation function (leave one out validator).

## The Classifier Class

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. You may want to use specialized data structures, sorting, indexing, etc. to speed things up, but this is not mandatory.

> **Note:** You are going to use the euclidean distance measure to compute the distance between two points in an n-dimensional space (n is the number of features). See the intro of this page if you need the equation.

Please try to implement the classifier in a modular way so that you have the methods **Train** and **Test**.

> **Train:** The input to the *Train* method is the set of training instances (or their IDs), no output for this method. Note that the class label for each instance is provided along with the feature vector..
>
> **Test:** The input to the *Test* method is a test instance (or its ID) and the output is the predicted class label.

## The Validator Class

**What it does:** Given a feature subset as input, it returns a score (classifier's accuracy) as output. It needs to use the training data and the classifier to do its job.

**Why we need it:** We are going to use the evaluation module (validator) later in Part III when we want to evaluate different feature subsets (nodes in the search tree) by calculating accuracies. Remember that our greedy algorithm from part I did not use a real evaluation function; instead the dummy evaluation function assigned a random value (accuracy) to each feature subset (node).

**In this part, however, we are not concerned with feature search. We just want to calculate the accuracy of the NN-classifier given a specific feature subset (as input).**

**The algorithm:** To calculate the accuracy, we use the leave-one-out validation method which is a simpler version of the k-fold cross validation method. Here is an example of how it works: Assume we have 5 training instances and we are using the feature subset {feature1,feature3, feature7}:

| Instance ID | Class Label | Feature1 | Feature3 | Feature7 |
|---|---|---|---|---|
| 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 representing 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 instance#1, use the rest of the instances (instances# {0,2,3,4}) as training set, and ask the classifier to predict the class label for instance#1. If it says class 2, it predicted the class correctly. Otherwise it made a mistake. Next, 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 the feature subset {feature1,feature3, feature7} is 3/5=0.6 or 60%[1].

## Testing your classifier and validator

**Please test that your NN algorithm and validator are working correctly by checking your code on the following datasets (Click on the names. These are links to the datasets):**

Small Dataset (Has 100 instances and 10 features)

- If you use only features {3, 5, 7}, accuracy should be about 0.89

Large Dataset (Has 1000 instances, and 40 features)

- If you use only features {1, 15, 27}, accuracy should be about 0.949

## Dataset Format

The datasets that you are going to test your system on will only have two classes. Also, they only 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
```

## Submission for Part II

For this part, you are going to submit

1) The code for the NN classifier and the validator.
2) The trace that shows the steps and the time spent on each step. There is no set format for this trace but make sure it includes the results on the sample datasets provided.

[1] These numbers are just for demonstration. Please do not use the above sample table for testing your code. Use the provided datasets instead.