SYSC 3110 Project – Simple Machine Learning Framework

The goal of this team project is to come up with a very simple environment for performing machine learning on simple problems. Machine learning is a set of techniques where you use past examples to make predictions about new ones. An example is described by an *input* (described by a set of *features*), and an *output*. A *training* example has values for each feature and for the output, whereas a *testing* example only has values for the features, but the output is the job of the machine learning algorithm to figure out. It does so by looking for patterns and correlations between features and outputs in the training examples.

The particular technique we're going to use is very simple (yet quite effective!) and is called "k-Nearest Neighbours", or kNN. For a given test example, the kNN algorithm finds, as its name indicates, the k "nearest" examples by using a combination of *distance functions* for the features.

Here's an example: suppose we want to predict the price of houses in Ottawa, based on a set of features such as location (in GPS Cartesian coordinates), square footage (an integer) and age of the house (an enum or a string). We have the following 3 training examples:

name	**coordinates**	**sq. ft.**	**age**	**price**
h1	(12, 25)	1200	new	500,000
h2	(10, 50)	1000	old	300,000
h3	(30, 100)	800	new	400,000

We want to predict the price of the following test example:

h4	(15, 20)	1000	new	???

To do this, we first need to decide how to calculate the distance of the test example from the training examples. And so we need to come up with distance functions for each feature.

For square footage, which is a simple ordered value, we can simply use the absolute difference in square footage.

For coordinates, which is a complex feature containing two continuous values, we can use the euclidean distance between the two points.

For age, which is a discrete value, we can say the distance is 0 if the two values match (e.g. both are "new" or both are "old"), and 1 otherwise.

If we simply apply the sum of distances between the test example and each of the testing examples, we obtain:

```
distance(h1,h4) ~= 206
distance(h2,h4) ~= 30
distance(h3,h4) ~= 282
```

and if we use k (the number of nearest neighbors) = 1, h2 is selected as the least distant and therefore nearest neighbour, and so the predicted value for the price of h4 is 300,000. If we use k = 2, h1 and h2 are the most similar, and we can average the price between the two for our prediction (or use a weighted average if we want to be more precise). If the output was discrete (an enum or a string for example), we could have used a vote between the outputs of the k nearest neighbours to determine the ouput.

Note that the distance metric for the age feature does not have much influence in the overall similarity (it is never more than a value of 1), whereas square footage, due to its high values, can overwhelm the other distances. This means you may want to "normalize" the distances (i.e., divide them by some max value to make them all fit between 0 and 1), and/or modify the overall distance by adding some weight to the value contributed by each feature. This means that you don't want to hard-code your distance metrics, but be able to pick and choose the ones you want and associate them to the given features.

Also, note that some features are simple (such as square footage in our examples), while others may be complex, potentially made up of other features. For example, you could have a feature called "rooms", which contains a set of square footages for each room. One should be able to compose these more complex features (and their distance metrics) out of simpler ones.

How do we know if our prediction is any good? One way to do this is to withhold some of the training examples and use them for testing instead. Since we know the actual output of those testing examples, we can calculate the *error* as the distance between the predicted output and the actual output, and the sum of the errors for all the test examples can be used as an estimate of the quality of the prediction.

The project is divided into 4 iterations, each ending with a milestone corresponding to deliverables that will be graded. You will be able to use the TA feedback from iteration i for iteration i+1.

Milestone 1: The “No GUI” version. Come up with features and distance functions for the features of the learning problem, programmatically instantiate examples and create test examples, perform predictions on the test examples, calculate errors. simple console-based version. One should be able to associate among many possible distance metrics for a given feature, and one should be able to compose complex features out of simple ones (like in the “rooms” example given earlier). Demonstrate that you can apply your solution to at least two different kNN problems, using different metrics. One of the problems should be the housing pricing example provided above

- Deliverables: code (source + executable in a jar file), documentation complete with UML diagrams (including with complete variable and method signatures),

and a readme file (see explanation below), all in one zip file. The code and the design are allowed to "smell" at this point.
- Deadline: Monday Oct 30th. Weight: 15% of the overall project grade.

Milestone 2: GUI and unit tests. The GUI should allow the user to create, edit and display the training and testing examples, perform kNN (with various values of k), and evaluate the quality of the prediction.
- Deliverables: source (including tests) + updated documentation and diagrams + readme file, all in one zip file. In particular, document the changes you made to your design and explain why.
- Deadline: Monday Nov 13th. Weight: 20% of the overall project grade.

Milestone 3: More flexibility! The GUI should now allow the user to choose distance metrics for features from a predefined set of distance metrics. We will also provide, right before the start of Milestone 3, a kNN problem requiring a set of distance metrics you may not have already implemented yet, and your framework should be able to accommodate it.
- Deliverables: code + tests + documentation + readme file. Make sure that you document the changes since the last iteration and the reasons for those changes. Your code and design should be "smell-free" at this point.
- Deadline: Monday Nov 27rd. Weight: 30% of the overall project grade.

Milestone 4: Two more features! Allow the tool to load and save data and features settings from/to files.
- Deliverables: code + design + documentation + readme file.
- Deadline: Friday Dec 8th. Weight: 35% of the overall project grade.

Milestones must contain all necessary files and documentation, even those items that are unchanged from previous milestones. Missing files cannot be submitted after the deadline, no exceptions. Verify your submission contains all necessary files (in particular, don't forget to include your source code!) before submitting on cuLearn.
The "readme" file, listed as a deliverable for each iteration, is typically a short text file that lists and describes: the rest of the deliverables, the authors, the changes that were made since the previous deliverable, the known issues (known issues are graded less severely than undocumented ones!) and the roadmap ahead.
"Documentation" includes up-to-date UML diagrams, detailed descriptions of design decisions made, complete user manuals, and javadoc documentation.

Note that nobody is stopping you from working ahead of schedule! In fact, iteration i+1 will very often give you good insight into iteration i.

This is a **team** project. Each team should have 4 members (or exceptionally 3). A project's success obviously depends on contributions from each member! So divide the work and cooperate. **Each contribution (source code, documentation, etc.) must contain the name of its author**: we will use this to determine whether there is any significant difference in the quality and quantity of the contributions of the team

members. If any such difference is detected, the individual grades will be adjusted accordingly.
You are expected to use Github to manage your project (version control, issue-tracking, etc...), but the deliverables for each milestone should also be submitted for marking on cuLearn. Each team will be given a private Github team repository. A TA and/or the instructor will also be members of each of the Github teams, so that they can track your use of the tool, verify that all group members are contributing, and their feedback will be given by opening new issues.

The marking scheme for each iteration will be comprised of: completeness of the deliverables, the quality of the deliverables, and, for iterations 2 to 4, we also look at whether you took into account the feedback you received from the TA in the previous iteration.