# CMPT 310 - Artificial Intelligence Survey

**Bonus Assignment**

Due date: April 25, 2021 — J. Classen
5 bonus marks — April 12, 2021

**Important Note:** Students must work individually on this, and other CMPT 310, assignments. You may not discuss the specific questions in this assignment, nor their solutions with any other student. You may not provide or use any solution, in whole or in part, to or by another student.

You are encouraged to discuss the general concepts involved in the questions in the context of completely different problems. If you are in doubt as to what constitutes acceptable discussion, please ask!

## Cross-Validation

In the lecture we discussed that in order to evaluate a hypothesis generated by a learning algorithm, it is important to keep the training data and test data separate. It is not surprising that the learnt model exhibits a low error rate on the set of examples it was trained on. If it really generalizes well only becomes apparent when being tested on previously unseen examples, similar as to how a professor knows that an exam will not accurately evaluate students if they have already seen the exam questions.

When we have $N$ examples in total, a straightforward approach is to split them into a training set and test set, use the training set to train, and then test using the test set. A disadvantage of this is that we don't make full use of all available data: if we use half the data for the test set, then we are only training on half the data, and we may get a poor hypothesis. On the other hand, if we reserve only 10% of the data for the test set, then it may happen that we get a very poor estimate of the actual accuracy.

One approach is to "squeeze" more out of the data and still get an accurate estimate using a technique called **$k$-fold cross-validation**. The idea is that each example serves double duty as training data and test data. First we split the data into $k$ equal subsets. We then perform $k$ rounds of learning; on each round $\frac{1}{k}$ of the data is held out as a test set and the remaining examples are used as training data. The average test set score of the $k$ rounds should then be a better estimate than a single score. 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. (The extreme is $k = N$, also known as leave-one-out cross-validation or LOOCV.)

The neural network for the restaurant scenario whose learning performance is measured on slide "Neural Networks 27" has four hidden nodes, as shown on slide "Neural Networks 22". This number was chosen somewhat arbitrarily. In this assignment, we want to use cross-validation to find the best number of hidden nodes (using a single hidden layer).

## Software

Similar as in Assignment 1, we will use Python 3[1], and code from the `aima-python` repository. However, note that this time we provide you with a ZIP file that contains

- a template Python file `abonus.py` for your solution and
- (modified) module files `learning.py` and `utils.py` from the repository needed to solve this assignment, as well as
- files `restaurant.csv` and `restaurant_numeric.csv` with the data for the 12 examples for the restaurant scenario from the lecture slides for illustration purposes.

There is no need to check out code from the repository through git. Simply extract the contents of `abonus.zip` into a separate directory and work on your solution there. Initially, when running `abonus.py`, you should see the following output:

```
Size 1: 0
Size 2: 0
Size 3: 0
Size 4: 0
Size 5: 0
Size 6: 0
Size 7: 0
Size 8: 0
Size 9: 0
Size 10: 0
Size 11: 0
Size 12: 0
Size 13: 0
Size 14: 0
Size 15: 0
```

Take some time to read and understand the files being provided in `abonus.zip`. For your solution, fill in the parts of the template that say "`### YOUR CODE HERE ###`". Do not use any modules or code except from the files provided in the ZIP and the standard Python 3 library. Do not modify anything else. In particular, leave `learning.py` and `utils.py` unchanged (you will be only submitting your modified `abonus.py` file).

[1] `https://www.python.org/`

## Question 1: Generate a Data Set (2 bonus marks)

Implement the function `generate_restaurant_dataset(size)` that takes an integer `size` as argument, and generates a data set with that many new, random examples for the restaurant scenario.

Note that the module `learning.py` already contains code for generating example data from the "real" restaurant decision tree, which you are free to use for this purpose. The problem is that the data being generated there contains non-numerical attributes, and so as a next step, it has to be converted into numerical form so it can be processed by a neural network. Specifically, convert the data so that

- all Boolean attributes are represented in the obvious fashion, representing `'No'` by `0` and `'Yes'` by `1`;
- for the `Patrons` attribute, use local encoding and represent `'None'` by `0`, `'Some'` by `1`, and `'Full'` by `2`;
- for the `Price` attribute, use local encoding and represent `'$'` by `0`, `'$$'` by `1`, and `'$$$'` by `2`;
- for the `WaitEstimate` attribute, use local encoding and represent `'0-10'` by `0`, `'10-30'` by `1`, `'30-60'` by `2`, and `'>60'` by `3`;
- for the `Type` attribute, use distributed encoding, where each value of `Burger`, `French`, `Italian` and `Thai` is represented by an attribute of the same name that takes value `1` just in case the restaurant is of that particular type, and where all others are `0`.

The output of the function should be an instance of the `DataSet` class that represents a set of examples generated from the restaurant decision tree, but using this numerical representation.

**Note:** For illustration, the CSV files in the ZIP contain the 12 restaurant examples from the slides, once in their original representation, and once converted into numeric form. Your function should not return these examples, but newly created, random ones.

## Question 2: Do Cross-Validation (2 bonus marks)

Next, implement the function

```
nn_cross_validation(dataset,hidden_units, epochs, k)
```

whose arguments are a data set, the number of hidden units to be used for the neural network, the maximal number of epochs to be used during training, and the $k$ parameter, respectively.

The function should perform one $k$-fold cross-validation on the given data set, where in each round, a (new) neural network is trained whose (single) hidden layer consist of a number of nodes as specified by `hidden_units`, and where the maximal number of epochs being performed during training is as specified by `epochs`. The return value should be the overall error, averaged over all $k$ rounds.

**Note:** The original `aima-python` repository version of module `learning.py` contains code for doing a cross-validation. Do not use this functionality (it will not work "out of the box"), but implement your own! However, you may use all other code from the `learning.py` module provided in the ZIP file. In particular, you do not have to implement the Back-propagation algorithm yourself.

## Question 3: Run Experiments (1 bonus mark)

Finally, run experiments to determine the optimal network structure. Create a PDF document that shows an example run of your program, and briefly explains what the optimal number of hidden units is according to the results of your experiments. (You are invited to also include plots or other visual representations of your data, but this is optional and not required for receiving marks.)

For your experiments, you may use the values for `N`, `k`, `epochs`, and `size_limit` that are given in the provided template, or higher ones. Note that it will take a certain amount of time for the program to run all experiments, so while working on your solution, it may be helpful to set these variables to smaller values, and only increase them again once you are ready to run the actual experiments.

## What to Submit

Your submission should consist of

- a file `abonus.py` that is the modified template file containing your own implementation and
- a PDF file called `abonus.pdf`, containing the example run and your (brief) analysis of the results.

In particular, there is no need to submit the other Python module files from the ZIP (you should not modify these for your solution).