Machine Learning *NextAI, 2025 Cohort*

Assignment 2

Due Date: 1:00pm Thursday, March 27, 2025
Assignments should be submitted as a single zip file
to stephania.stefanakou@nextai.com for Toronto teams
or maud.razafindramboa@nextai.com for Montreal teams.

Late submissions will not be accepted.

Deliverables

This is an individual assignment. If a team has multiple technical co-founders, then each technical co-founder should submit an assignment.

Submissions will consist of the following two files:

- An assignment report **in PDF format**.
- A completed version of `dt.py` (you are provided with skeleton code and are asked to complete some functions).

Reports should not exceed six (6) pages exclusive of cover page, including figures, tables and code. Material in excess of six pages will not be marked.

Technical Presentation Requirements

Include a cover page indicating your name, title of work, course, instructor's name and date. Assignments will be judged on the basis of visual appearance, the grammatical correctness and quality of writing, and the visual appearance and readability of any graphics, as well as their contents. Please make sure that the text of your report is well-structured, using paragraphs, full sentences, and other features of a well-written presentation. Use itemized lists of points where appropriate. Text font size should be between 10 and 12 point.

1 Implement a Decision Tree Classifier

In this assignment you will be required to implement a decision tree classifier. Download the file **assignment2_files.zip** from Google Drive. This file contains some skeleton code that you will need to complete and submit, as well as a variety of functions that you may find useful:

- **dt.py:** Partially completed code for a decision tree classifier. You will need to fill in the missing pieces to get it working!

- **binary.py:** Generic interface for binary classifiers.
- **datasets.py:** Where a handful of test data sets are stored.
- **util.py:** A handful of useful utility functions: these will undoubtedly be helpful to you, so take a look!
- **runClassifier.py:** A few wrappers for doing useful things with classifiers, like training them, generating learning curves, etc.
- **data/*:** some datasets which we can use.

1.1 Training function

To get started, start up a Python shell (IPython or Jupyter Notebook recommended) and import the functions that you will need to implement the decision tree:

```
>>> import dt
>>> import datasets
>>> import runClassifier
```

Next, you should implement the training procedure for decision trees (the `trainDT` function in `dt.py`). We've provided a fair amount of the code, which should help you guard against corner cases. (Hint: take a look at **util.py** for some useful functions for implementing training). Once you've implemented the training function, we can test it on the included tennis dataset. You should get the same output as the following code snippet:

```
>>> h = dt.DT({'maxDepth': 1})
>>> h
Leaf 1

>>> h.train(datasets.TennisData.X, datasets.TennisData.Y)
>>> h
Branch 6
  Leaf 1.0
  Leaf -1.0
```

If you get a "Method not implemented" error, or if your output looks different from the above snippet, you may have skipped a step in your implementation. It may help to look at Algorithm 1 (DecisionTreeTrain) on page 13 of the textbook to make sure you're not missing anything.

If your output is correct, try making your decision tree deeper. You should get something like:

```
>>> h = dt.DT({'maxDepth': 5})
>>> h.train(datasets.TennisData.X, datasets.TennisData.Y)
```

```
>>> h
Branch 6
  Branch 7
    Leaf 1.0
    Branch 2
      Leaf 1.0
      Leaf -1.0
  Branch 1
    Branch 7
      Branch 2
        Leaf -1.0
        Leaf 1.0
      Leaf -1.0
    Leaf 1.0
```

1.2 Prediction function

Once you have the training function you will be able to fit your model on the training data, but we still need a way to make predictions. For this you will need to implement the `predict` function in `dt.py`.

To see if your prediction function is working properly, train and evaluate your decision tree on the sentiment analysis dataset. You can use the runClassifier convenience function to both train, and test your function. You should get the following output (this may take a few seconds):

```
>>> h = dt.DT({'maxDepth': 1})
>>> runClassifier.trainTestSet(h, datasets.SentimentData)
Training accuracy 0.630833, Test accuracy 0.595
```

Question 1-1. (10 points) Include your `trainDT` function as well as the output of running the above in your report. Your implementation will be graded on correctness, efficiency, and readability. Make sure you also submit your entire `dt.py` file.

1.3 Plotting a learning curve

Now we can use the runClassifier function to generate a learning curve, which will show us the training and test accuracy (y-axis) as a function of the number of data points (x-axis) used for training:

```
>>> h = dt.DT({'maxDepth': 9})
>>> curve = runClassifier.learningCurveSet(h, datasets.SentimentData)
[training progress shows here]
>>> runClassifier.plotCurve('Learning Curve', curve)
```

Question 1-2. (5 points) Include your curve in your report. We should see training accuracy (roughly) going down and test accuracy (roughly) going up. Why does training accuracy tend to go down? Why is test accuracy not monotonically increasing? You should also see jaggedness in the test curve toward the left. Why?

1.4 Plotting a hyperparameter curve

We can also generate similar curves by changing the maximum depth hyperparameter (this will probably take several minutes):

```
>>> curve = runClassifier.hyperparamCurveSet(dt.DT({}),
  'maxDepth', [1,2,4,6,8,12,16], datasets.SentimentData)
[training progress shows here]
>>> runClassifier.plotCurve('Hyperparameter Curve', curve)
```

Question 1-3. (5 points) Include your curve in your report. You should see training accuracy monotonically increasing and test accuracy making something like a hill. Which of these is guaranteed to happen and which is just something we might expect to happen? Why?

1.5 Decision tree with no depth limit

Question 1-4. (2 points) Until now we've been limiting the depth of our decision trees with the "maxDepth" hyperparameter. Without changing your `trainDT` function, how could you modify the code to train a decision tree that has no limit to its depth?