CT4101

Machine Learning

Name: Andrew Hayes
E-mail: a.hayes18@universityofgalway.ie
Student ID: 21321503

2024–09–18

Contents

1 Introduction

1.1 Lecturer Contact Details

- Dr. Frank Glavin.
- `frank.glavin@universityofgalway.ie`

1.2 Grading

- Continuous Assessment: 30% (2 assignments, worth 15% each).
- Written Exam: 70% (Last 2 year's exam papers most relevant).

1.3 Module Overview

Machine Learning (ML) allows computer programs to improve their performance with experience (i.e., data). This module is targeted at learners with no prior ML experience, but with university experience of mathematics & statistics and **strong** programming skills. The focus of this module is on practical applications of commonly used ML algorithms, including deep learning applied to computer vision. Students will learn to use modern ML frameworks (e.g., scikit-learn, Tensorflow / Keras) to train & evaluate models for common categories of ML task including classification, clustering, & regression.

1.3.1 Learning Objectives

On successful completion, a student should be able to:

1. Explain the details of commonly used Machine Learning algorithms.
2. Apply modern frameworks to develop models for common categories of Machine Learning task, including classification, clustering, & regression.
3. Understand how Deep Learning can be applied to computer vision tasks.
4. Pre-process datasets for Machine Learning tasks using techniques such as normalisation & feature selection.
5. Select appropriate algorithms & evaluation metrics for a given dataset & task.
6. Choose appropriate hyperparameters for a range of Machine Learning algorithms.
7. Evaluate & interpret the results produced by Machine Learning models.
8. Diagnose & address commonly encountered problems with Machine Learning models.
9. Discuss ethical issues & emerging trends in Machine Learning.

2 What is Machine Learning?

There are many possible definitions for "machine learning":

- Samuel, 1959: "Field of study that gives computers the ability to learn without being explicitly programmed".
- Witten & Frank, 1999: "Learning is changing behaviour in a way that makes *performance* better in the future".
- Mitchelll, 1997: "Improvement with experience at some task". A well-defined ML problem will improve over task T with regards to **performance** measure P, based on experience E.
- Artificial Intelligence $\neq$ Machine Learning $\neq$ Deep Learning.
- Artificial Intelligence $\not\supseteq$ Machine Learning $\not\supseteq$ Deep Learning.

Machine Learning techniques include:

- Supervised learning.
- Unsupervised learning.
- Semi-Supervised learning.
- Reinforcement learning.

Major types of ML task include:

1. Classification.
2. Regression.
3. Clustering.
4. Co-Training.
5. Relationship discovery.
6. Reinforcement learning.

Techniques for these tasks include:

1. **Supervised learning:**
    - **Classification:** decision trees, SVMs.
    - **Regression:** linear regression, neural nets, k-NN (good for classification too).
2. **Unsupervised learning:**
    - **Clustering:** k-Means, EM-clustering.
    - **Relationship discovery:** association rules, bayesian nets.
3. **Semi-Supervised learning:**
    - **Learning from part-labelled data:** co-training, transductive learning (combines ideas from clustering & classification).
4. **Reward-Based:**
    - **Reinforcement learning:** Q-learning, SARSA.

In all cases, the machine searches for a **hypothesis** that best describes the data presented to it. Choices to be made include:

- How is the hypothesis expressed? e.g., mathematical equation, logic rules, diagrammatic form, table, parameters of a model (e.g. weights of an ANN), etc.
- How is search carried out? e.g., systematic (breadth-first or depth-first) or heuristic (most promising first).
- How do we measure the quality of a hypothesis?
- What is an appropriate format for the data?
- How much data is required?

To apply ML, we need to know:

- How to formulate a problem.

- How to prepare the data.
- How to select an appropriate algorithm.
- How to interpret the results.

To evaluate results & compare methods, we need to know:

- The separation between training, testing, & validation.
- Performance measures such as simple metrics, statistical tests, & graphical methods.
- How to improve performance.
- Ensemble methods.
- Theoretical bounds on performance.

2.1 Data Mining

Data Mining is the process of extracting interesting knowledge from large, unstructured datasets. This knowledge is typically non-obvious, comprehensible, meaningful, & useful.

The storage "law" states that storage capacity doubles every year, faster than Moore's "law", which may results in write-only "data tombs". Therefore, developments in ML may be essential to be able to process & exploit this lost data.

2.2 Big Data

Big Data consists of datasets of scale & complexity such that they can be difficult to process using current standard methods. The data scale dimensions are affected by one or more of the "3 Vs":

- **Volume:** terabytes & up.
- **Velocity:** from batch to streaming data.
- **Variety:** numeric, video, sensor, unstructured text, etc.

It is also fashionable to add more "Vs" that are not key:

- **Veracity:** quality & uncertainty associated with items.
- **Variability:** change / inconsistency over time.
- **Value:** for the organisation.

Key techniques for handling big data include: sampling, inductive learning, clustering, associations, & distributed programming methods.

3 Introduction to Python

Python is a general-purpose high-level programming language, first created by Guido van Rossum in 1991. Python programs are interpreted by an *interpreter*, e.g. **CPython** – the reference implementation supported by the Python Software Foundation. CPython is both a compiler and an interpreter as it first compiles Python code into bytecode before interpreting it.

Python interpreters are available for a wide variety of operating systems & platforms. Python supports multiple programming paradigms, including procedural programming, object-oriented programming, & functional programming. Python is **dynamically typed**, unlike languages such as C, C++, & Java which are *statically typed*, meaning that many common error checks are deferred until runtime in Python, whereas in a statically typed language like Java these checks are performed during compilation.

Python uses **garbage collection**, meaning that memory management is handled automatically and there is no need for the programmer to manually allocate & de-allocate chunks of memory.

Python is used for all kinds of computational tasks, including:

- Scientific computing.
- Data analytics.
- Artificial Intelligence & Machine Learning.
- Computer vision.
- Web development / web apps.
- Mobile applications.
- Desktop GUI applications.

While having relatively simple syntax and being easy to learn for beginners, Python also has very advanced functionality. It is one of the most widely used programming languages, being both open source & freely available. Python programs will run almost anywhere that there is an installation of the Python interpreter. In contrast, many languages such as C or C++ have separate binaries that must be compiled for each specific platform & operating system.

Python has a wide array of libraries available, most of which are free & open source. Python programs are usually much shorter than the equivalent Java or C++ code, meaning less code to write and faster development times for experienced Python developers. Its brevity also means that the code is easier to maintain, debug, & refactor as much less source code is required to be read for these tasks. Python code can also be run without the need for ahead-of-time compilation (as in C or C++), allowing for faster iterations over code versions & faster testing. Python can also be easily extended & integrated with software written in many other programming languages.

Drawbacks of using Python include:

- **Efficiency:** Program execution speed in Python is typically a lot slower than more low-level languages such as C or C++. The relative execution speed of Python compared to C or C++ depends a lot on coding practices and the specific application being considered.
- **Memory Management** in Python is less efficient than well-written C or C++ code although these efficiency concerns are not usually a major issues, as compute power & memory are now relatively cheap on desktop, laptop, & server systems. Python is used in the backend of large web services such as Spotify & Instagram, and performs adequately. However, these performance concerns may mean that Python is unsuitable for some performance-critical applications, e.g. resource-intensive scientific computing, embedded devices, automotive, etc. Faster alternative Python implementations such as **PyPy** are also available, with PyPy providing an average of a four-fold speedup by implementing advanced compilation techniques. It's also possible to call code that is implemented in C within Python to speed up performance-critical sections of your program.
- **Dynamic typing** can make code more difficult to write & debug compared to statically-typed languages, wherein the compiler checks that all variable types match before the code is executed.
- **Python2 vs Python3:** There are two major version of Python in widespread use that are not compatible with each other due to several changes that were made when Python3 was introduced. This means that some libraries that were originally written in Python2 have not been ported over to Python3. Python2 is now mostly used only in legacy business applications, while most new development is in Python3. Python2 is no longer supported or receives updates as of 2020.

3.1 Running Python Programs

Python programs can be executed in a variety of different ways:

- through the Python interactive shell on your local machine.
- through remote Python interactive shells that are accessible through web browsers.
- by using the console of your operating system to launch a standalone Python script (`.py` file).
- by using an IDE to launch a `.py` file.
- as GUI applications using libraries such as Tkinter PyQt.
- as web applications that provide services to other computers, e.g. by using the Flask framework to create a web server with content that can be accessed using web browsers.
- through Jupyter / JupyterLab notebooks, either hosted locally on your machine or cloud-based Jupyter notebook execution environments such as Google Colab, Microsoft Azure Notebooks, Binder, etc.

3.2 Hello World

The following programs writes "Hello World!" to the screen.

```
1 print("Hello World!")
```

Listing 1: `helloworld.py`

3.3 PEP 8 Style Guide

PEPs (Python Enhancement Proposals) describe & document the way in which the Python language evolves over time, e.g. addition of new features. Backwards compatibility policy etc. PEPSs can be proposed, then accepted or rejected. The full list is available at `https://www.python.org/dev/peps/`. **PEP 8** gives coding conventions for the Python code comprising the standard library in the main Python distribution. See: `https://www.python.org/dev/peps/pep-0008/`. It contains conventions for the user-defined names (e.g., variables, functions, packages), as well as code layout, line length, use of blank lines, style of comments, etc.

Many professional Python developers & companies adhere to (at least some of) the PEP8 conventions. It is important to learn to follow these conventions from the start, especially if you want to work with other programmers, as experienced Python developers will often flag violations of the PEP 8 conventions during code reviews. Of course, many companies & open-source software projects have defined their own internal coding style guidelines which take precedence over PEP 8 in the case of conflicts. Following PEP 8 conventions is relatively easy if you are using a good IDE, e.g. PyCharm automatically finds & alerts you to violations of the PEP 8 conventions.

3.3.1 Variable Naming Conventions

According to PEP 8, variable names "should be lowercase, with words separated by underscores as necessary to improve readability", i.e. `snake_case`. "Never use the characters `l`, `0`, or `I` as single-character variable names. In some fonts, these characters are indistinguishable from the numerals one & zero. When tempted to use `l`, use `L` instead". According to PEP 8, different naming conventions are used for different identifiers, e.g.: "Class names should normally use the CapWords convention". This helps programmers to quickly & easily distinguish which category an identifier name represents.

3.4 Dynamic Typing

In Python, variable names can point to objects of any type. Built-in data types in python include ***str***, ***int***, ***float***, etc.