MSE 446 Lecture Notes

Jacob Bodera

Winter 2025

Module 1: Introduction to Machine Learning

1.1 What is Machine Learning

- ML is a branch of AI which assists humans by using data and algorithms and gradually improving

Artificial Intelligence: exceeding or matching the capabilities of a human. Ability to discover (find new information), infer (read in other info from sources that not explicitly stated), and reason (figure things out)

- natural language processing, vision, text to speech, motion

Machine Learning: something that learns rather than having to be programmed, uses large amounts of information to do this. A subset of AI. There are two main types: supervised (has more human involved learning) and unsupervised learning

- medical treatments (NN used to read genes of viruses and detect potentially dangerous mutations), self-driving cars (locating landmarks on the fly to make prediction more accurate), banking automation (better offers for customers, fraud detection)

Deep Learning: a subset of ML and an example includes neural networks which have many different layers. Often don't know how reliable the results are or how they were derived.

1.2 Type of Machine Learning

Supervised Learning

Machines are trained on labelled data - contains observations with:

- Input data, consisting of the properties of observations (e.g. shape, age, height, etc.)
- Output data or the label of the observation (e.g. cat or dog, diabetes or non-diabetes)

The properties of data are also called *features*. The label is also called the *outcome*. Supervised learning find a mapping from features to outcome. Applications include image classification, risk assessment, patient classification, fraud detection.

Pros:

- can predict output based on previous experiences
- have exact idea about the classes of an object
- widely used for solving real-world problems

Cons:

- not suitable for complex tasks
- not able to predict with high accuracy if test data is different from training data
- execution time of training could be very high
- need sufficient knowledge about classes of data

Unsupervised Learning

Do not have labelled data and so we cannot supervise the ML models with the relationship between data features and target variable. Finds hidden patterns and groups data based on this.

Pros:

- can be used for more complex tasks due to not requiring labelled data
- easier to use due to no labelled data

Cons:

- more difficult than supervised as there are no state feature or target variable
- may have less accuracy as algorithms don't know exact output

Reinforcement Learning (RL)

A type of algorithm based on an agent learning to behave properly in an environment in which it is rewarded for good outcomes and punished for bad ones. So, it is mainly a feedback based ML technique, learns by it's own experiences. There is no labelled data. Useful when decision making is sequential and goal is achieved in the long-term (such as ChatGPT).

Machine Learning Workflow

The success of an ML project relies on designing a valid research questions and then following these steps:

1. **Gather data**: using real0time data from internet-of-things (IoT) -based devices or collecting it from other valid databases.
2. **Prepare data**: one of the most important requirements for a successful ML project. Common to spend a lot of time on this step.
3. **Select the model**: that best fits data at hand - identifying proper supervised, unsupervised, or reinforcement learning model
4. **Train and Test**: choose proper ratio of data for training ML models and testing their performance
5. **Evaluate**: select best parameters for the model to achieve highest accuracy

1.3 Python and Virtual Environment Installation

Python was introduced in 1991 and is often called a scripting language as you can write quick codes (scripts) to automate tasks. Very popular for data analysis and visualization, interactive computing, and machine learning.

Python Packages or Libraries

Python doesn't include everything you need right away. Can install developer written software for specific purposes.

Library: a collection of pre-written code that provides specific functions or capabilities

Package: way to organize related modules to create a modular and structured codebase

Virtual Environment

Can use a browser to run python and runs code snippets in the backend. Keeps a copy of current Python and package versions on the disk so that code runs properly.

Module 2: Python Language Basics

2.1 Basic Semantics

Some core semantics of Python include:

Object Model: everything (numbers, strings, data structures, functions, etc.) are considered as an object

Comments: text preceeded with a # are ignored by the interpreter

Functions: call functions using parentheses using zero or more arguments. Almost all object have functions and call them by `object.method(x, y, z)`.

Variables and argument passing: when assigning a variable, we create a reference to the object on the right side of the equality

```
# Assign x to a new variable
x = [1,2,3]
# y and x refer to the same object
y = x
y
```

Dynamic References: object references have no inherent types. Types are important to know how to handle variables and so can see type of object using `type(variable)`.

Attributes and methods: objects have attributes (other object stored inside this one) and methods (functions related to an object to access internal data).

Imports

Need to first import modules, libraries, or the packages to use methods or functions within them. If we have the following file:

```
PI = 3.14159

def f(x):
    return x+5

def g(a,b):
    return a*b
```

Now, from another file, we want to create two variables based on the scalar and functions in "some_module.py" so we can import it.

```
import some_module

result = some_module.f(10)
pi = some_module.PI
```

Can also import specific parts from some_module.

```
from some_module import g, PI

result = g(10, PI)
```

You can also give an import a short form using the `as` keyword.

```
import some_module as sm
from some_module import PI as pi

a1 = sm.f(pi)
a2 = sm.g(10,pi)
```

2.2 Types of Objects

Python Objects: scalar types (numbers, strings, booleans) and sequence types (lists, tuples, sets, dictionaries)

Numpy Objects: arrays

Pandas Objects: series, dataframes

Python Objects

Scalar Types

Are built-in types used to deal with numerical data, strings, or boolean values. Below is a list of main scalars:

- `None`: null or missing value
- `str`: string or text
- `int`: integer (whole number)
- `float`: floating-point number (decimal points)
- `bytes`: raw binary data
- `bool`: Boolean (`True` or `False`) value

Numbers: can be in different forms, mainly `int` and `float`. Integers take less memory so use them unless otherwise necessary.

Strings: are words or Unicode characters. Need to put them in single (’) or double (“) quotes to distinguish from variables. Put them in triple quotes (’ ’ ’ or”“ “) for multiline strings with line breaks.

- string are immutable but can be combined using `+`

You can specify the types of your variables by using the appropriate constructors such as, `str()`, `float()`, `int()`.

Lists and Tuples

A *list* stores a sequence of values in a single variable, which can include any data types and duplicates. Use square brackets. Can also use `list()` constructor.

- lists are mutable

```
l = [1, 2, 3]
# Note the double parentheses
l2 = list(("machine", 4, False))
```

A *tuple* is similar to a list but defined with round braces or none at all.

- requires very small amount of memory since it is immutable

```
t = (1, 2, 3)
t = 1, 2, 3
t = tuple((1, 2, 3))
```

A *set* also stores multiple elements of any data types in a single variable and uses curly braces.

- unordered and does not allow duplicates

```
s = {1, 2, 3}
```

A *dictionary* (or *dict*) store data in the form of key-value pairs within curly braces. Can think of `keys` as columns and retrieve values of a certain key by `dictName[key]`.

```
animals = {'fish': ['bass','cod','shark'],
           'amphibian': ['frog','salamander'],
          'insect': ['fly','spider'],
          'reptile': ['snake','crocodile'],
          'bird': ['eagle','jay'],
          'mammal': ['dolphin','dog']}
```

```
animals.keys()
# dict_keys(['fish', 'amphibian', 'insect', 'reptile', 'bird', 'mammal'])

animals.values()
# dict_values([['bass', 'cod', 'shark'], ['frog', 'salamander'], ['fly',
'spider'], ['snake', 'crocodile'], ['eagle', 'jay'], ['dolphin', 'dog']])

animals.items()
# dict_items([('fish', ['bass', 'cod', 'shark']), ('amphibian', ['frog',
'salamander']), ('insect', ['fly', 'spider']), ('reptile', ['snake',
'crocodile']), ('bird', ['eagle', 'jay']), ('mammal', ['dolphin', 'dog'])])
```

Numpy Objects

Used for handling large scale mathematical operations, such as vector calculus, matrix algebra, etc. Used by scikit-learn and pandas packages in the backend.

A Numpy *array* is a list of numbers and allows for more complex mathematical operations.

```
y = np.array([1, 2, 3])
y**2

array([1, 4, 9])
```

```
# Generate same psuedo-random numbers
np.random.seed(156)
# Generate random integers between -4 and 4
arr = np.random.randint(-4,4,(5,5))
arr

# array([[ 0,  0, -2,  1,  3],
        [-1, -2, -3,  0,  3],
        [-2,  3,  1,  3, -2],
        [-4,  2, -1, -3,  2],
        [ 2, -4,  1,  3,  0]])
```

Slice an Array

Inherits the same slicing method from a regular Python list, however, slices are applied separately as `[rows, cols]`.

- Slice from beginning (omitted) to row 3 (not included)
- Slice from second last column (-2) to the end (omitted)

```
arr[:3,-2:]

# array([[ 1, 3],
        [ 0, 3],
        [ 3, -2]])
```

Concatenate Arrays

There are many ways to combine to array, `a` and `b`, in Numpy. Two way include `vstack()` and `hstack()`.

```
np.vstack([a, b])
np.hstack([a, b])
```

Pandas Objects

Pandas is a Swiss army knife for data preparation and analysis in machine learning. Two main data structures of pandas are `Series` and `DataFrame`.

Series

Similar to a list, can contain any data type or Python objects.

```
s1 = pd.Series([5, 13, -9, 21])
s1

# 0     5
  1    13
  2    -9
  3    21
  dtype: int64
```

We can customize the index to label our data:

```
s2 = pd.Series([5,13,-9,21], index=['a','b','c','d'])
s2

# a     5
  b    13
  c    -9
  d    21
  dtype: int64
```

Can index in several ways such as using the label or by using a Boolean expression:

```
s2[['b', 'c']]

# b    13
  c    -9
  dtype: int64

s2[s2 > 8]

# b    13
  d    21
  dtype: int64
```

DataFrame

A *data frame* is a collection of series and is represented in a tabular format consisting of rows and columns. Can also create a data frame from a dictionary using the `pandas.DataFrame` method.

Can specify the order of the columns by passing the `columns` parameter:

```
pd.DataFrame(data, columns=['year', 'province', 'population'])
```

Can extract a columns as a series by using the column name:

```
df.province
# OR
```

```
df['province']
```

Can extract a row by passing a `loc` attribute:

```
# Extract the third row
df.loc[3]
```

Can add a new column by an assignment statement. When adding a new column with fewer entries than the number of rows, can specify which rows the data is added to using the `index` parameter. The unspecified rows are assigned to `NaN`.

```
df['debt'] = [314.1, 380.4, 185.0, 185.6, 37.7, 55.8]

col = pd.Series([0.4, -1.9], index=[1, 3])
df['new'] = col
```

Can delete a column using the `del` keyword:

```
del df['new']
```

Dates and time are very important in pandas. A column can be converted to a data using `pd.to_datetime()`. By default, the function expects ISO format (YYY-MM-DD) but you can change this with the `format` parameter.

```
pd.to_datetime('30-09-2016', format='%d-%m-%Y')

# Timestamp('2016-09-30 00:00:00')
```

2.3 Common Data Manipulation

A crucial step for the data processing pipeline for ML. It involves the process of cleaning, transforming, and preparing raw data to make sure its suitable for modeling.

String and List Methods

`split()`: to separate a string by some indicators

`strip()`: used to remove white space from a string

`in`: to determine whether an element is part of a string or a list object

`count()`: find the number of occurrences of a value

`replace()`: used to replace a string with another value

`str.upper()`: used to capitalize the first letter of a string. Can be applied on the column of a DataFrame

`str.lower()`: used to lower case a string

The `Map()` Method

Mapping is another way to add columns to a data frame. If we want to add a new column to the grade data, we can create a dictionary and use `map()` to manually match student names to grades.

```
report_to_grade = {'Ariel': 75, 'Chris': 75, 'David': 85, 'Emily': 90, 'Isaac': 60,
                   'Jordan': 90, 'Morgan': 75, 'Olivia': 85, 'Ryan': 80, 'William': 95}

grade['report'] = grade['name'].map(report_to_grade)
```

Replacing Values

There are many ways to replace values and this is often done when there are `NaN` values for some data points.

```
grade['quiz'].replace(0, np.nan)

# Replace multiple values with same value
grade['quiz'].replace([0, 100], np.nan)

# Replace multiple values with different values
grade['quiz'].replace([0, 100], [np.nan, 99])

# Can do the same with a dictionary
grade['quiz'].replace({0: np.nan, 100: 99})
```

Rename Columns and Rows

The `replace()` method can be used to change the name of the rows and columns.

```
grade.rename(index=grade['name'], columns=str.title)
```

Binning Methods

Useful for when you need to sort continuous values into bins for further analysis.

```
# Defining our own bins sizes with labels
bin_names = ['unacceptable', 'good', 'excellent']
bins = [0, 60, 80, 100]
group_n = pd.cut(report, bins, labels=bin_names)

# Equal sized cut with 2 decimal precision
pd.cut(report, 3, precision=2)

# Quantile-based cuts
pd.qcut(report, 3, precision=2)
```

Can count how many items are in each bin by using `pandas.value_counts` method and passing the cut data.

Random Sampling

Random sampling is an efficient way to downsize a data set and is useful for bootstrapping. This is done using the `sample()` method. The axis can be specified using the `axis` parameter with `axis=1` corresponding to columns and the default set to rows. Replacement is specified through the `replace` parameter.

```
grade.sample(num_samples, axis, replace)
```