CS4619: Artificial Intelligence 2

Introduction

Derek Bridge
School of Computer Science and Information Technology
University College Cork

Initialization

In [1]: %reload_ext autoreload
%autoreload 2
%matplotlib inline

In [2]:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

About this Module

| Lecturer:        | Derek Bridge, Room G-61, Western Gateway Building
d stop bridge amphora cs plip ucc plop ie
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Credit weighting:</td>
<td>5 credit optional module</td>
</tr>
</tbody>
</table>
| Prerequisites:   | Precision of thought
We will use: matrices, vectors, calculus, probabilities, Python |
| Lectures:        | 2 \times 1 \text{ hr per week}                                    |
| Labs:            | 1 \times 2 \text{ hr per week}                                    |
| Private study:   | At least 2 hrs per week                                           |
| Examination:     | 1.5 hr written exam (80\% of the marks)                           |
| Continuous
assessment: | Open-book programming class test (20\% of the marks)              |
| How to fail:     | Skip lectures & labs; avoid private study; cram at Easter; expect
the exam to be a memory test                                      |
| How to pass:     | Attend lectures & labs; summarize the notes; tackle the lab
activities properly; expect a problem-solving exam                 |
Prediction

We want to create programs that make predictions. Generically, we will refer to such programs as estimators.

There are two main types of prediction, and hence two types of estimator:

- **Regression** means predicting a continuous value
- **Classification** means predicting a discrete value

Regression

An example of regression is predicting the market value of your house.

- **Features:**
  - Features describe the houses, e.g.
    - `flarea`: the total floor area (in square metres)
    - `bdrms`: the number of bedrooms
    - `bthrms`: the number of bathrooms
  - A particular house has values for each of these features
    - e.g. your house might have a floor area of 114 square metres, 3 bedrooms and 2 bathrooms
    - i.e. `flarea = 114, bdrms = 3, bthrms = 2`
- **The dependent variable** (usually designated `y`):
  - the value we are predicting (the 'outcome', the 'target')
    - e.g. the predicted selling price of a house

Building a model: ask an expert
So how do we predict the price of your house?

We ask Ann. She's an auctioneer — an expert at predicting Cork city house prices.

But we don't ask her to predict your house price. We ask her for a general method for predicting Cork house prices.

She tells us that her rule-of-thumb is that prices start at 25k€ and increase by 1.5k€ for every extra square metre of floor area:

\[ y = 25 + 1.5 \times \text{flarea} \]

So, she predicts your house will sell for \( y = 25 + 1.5 \times 114 = 196 \text{k€} \).

Ann has given us a model.

- In very abstract terms, a model is an approximation of some part of reality that enables us to make predictions about that reality.
- In very concrete terms, a model is a formula that expresses the relationship between the dependent variable (the thing being predicted) and the features.

Ann's is a linear model: it takes the form of the equation of a straight line.

### Ann's Model

In [3]:

```python
def f_ann(flarea):
    return 25 + 1.5 * flarea

fig = plt.figure()
plt.title("Ann's model")
plt.xlabel("Floor area (sq metres)"
plt.xlim(0, 500)
plt.ylabel("Price (000 euros)"
plt.ylim(0, 1000)
xvals = np.linspace(0, 500, 2)
plt.plot(xvals, f_ann(xvals))
plt.show()
```

![Ann's model graph](image_url)
Ben's Model

We might also ask Ben, another Cork auctioneer, and he might give us a different model, e.g.
\[ y = 60bdrms + 80bthrms \]

Ben's model is also linear, so we can plot it:

```python
In [4]: def f_ben(bdrms, bthrms):
    return 60 * bdrms + 80 * bthrms

from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = Axes3D(fig)
ax.set_title("Ben's model")
ax.set_xlabel("Bedrooms")
ax.set_ylabel("Bathrooms")
ax.set_zlabel("Price (000 euros)")
xvals = np.linspace(0, 8, 2)
yvals = np.linspace(0, 5, 2)
xxvals, yyvals = np.meshgrid(xvals, yvals)
ax.plot_surface(xxvals, yyvals, f_ben(xxvals, yyvals))
plt.show()
```

Which Model is Better?
Ann’s and Ben’s models make different predictions. So we might ask: which is better?

- Ann predicts your house will sell for \( y = 25 + 1.5 \times 114 = 196 \text{k€} \).
- Ben predicts your house will sell for \( y = 60 \times 3 + 80 \times 2 = 178 \text{k€} \).
- Their models make different predictions. So we might ask: which is better?
- This is a complicated question, which we will explore in this module.
- For now, suppose your house sells for 210000€. Ann's prediction (196000€) is closer than Ben's (178000€), so we have some evidence that Ann’s model is better.

**Building a model: learn from data**

Rather than ask an expert, we want to learn a model from data.

Suppose we collect a set of examples.

- Each example describes a house by giving values for the various features (e.g. floor area, number of bedrooms and number of bathrooms).
- But each example also gives the *actual* selling price of the house.

We take some or all of these examples, call them the training set, and give them to the learning algorithm. As best it can, it finds a model based on the examples in the training set.
**Terminology**

- We will say that the algorithm **learns** a model.
- We could also say that we are **training** the algorithm on the data.
- We could also say that the algorithm **fits** a model to the training set.
- We could also call it **function approximation**.
- We call this **supervised learning** because the examples include the actual outcomes (in this case, the actual selling prices). In **unsupervised learning**, the examples provide only the feature values.

**Cork Property Prices Dataset**

At the beginning of November 2014, I scraped a dataset of property prices for Cork city from www.daft.ie. They are in a CSV file. Each line in the file is an example, representing the sale of one house.

- Each line contains the feature-values (floor area, number of bedrooms, number of bathrooms).
- The last value on each line is the **actual** selling price. (In fac, it's the asking price, but we'll pretend that it's the selling price.)

Without going into any details, we'll learn a model from this data, using all of it as the training set.

(N.B. We will see later that using all the data for training is not necessarily the right thing to do.)

**Using scikit-learn to Build a Model for the Cork Property Prices Dataset**

```python
In [5]: from sklearn.linear_model import LinearRegression

# Use pandas to read the CSV file
def = pd.read_csv("dataset-corkA.csv")

# Get the feature-values and the target values into separate numpy arrays
X = df[['flarea', 'bdrms', 'bthrms']].values
y = df['price'].values

# Create linear regression object
estimator = LinearRegression()

# Train the model using the data
estimator.fit(X, y)

# Print the parameters that it learns
print('Intercept: ', estimator.intercept_)
print('Coefficients: ', estimator.coef_)

('Intercept: ', 62.502619092765826)
('Coefficients: ', array([-4.6900952, -72.28169508, -57.05464555]))
```
We have learned a linear model, like Ann's and Ben's. Specifically, the model is roughly

\[ y = 62.50 + 4.69\text{flarea} - 72.28\text{bdmins} - 57.05\text{bhrs} \]

What does this model predict that your house will sell for?

```python
In [6]: estimator.predict([[114, 3, 2]])
Out[6]: array([ 266.21909577])
```

It predicts your house will sell for \( y = 62.50 + 4.69 \times 114 - 72.28 \times 3 - 57.05 \times 2 = 266 \text{k€} \).

It's a linear model. But, unfortunately, we can't plot it. Why?

So, let's learn a model like Ann's: using just the floor area.

**Building Another Model**

```python
In [7]: # This time only use the floor area
X = df[['flarea']].values

# Re-train the model using the smaller dataset
estimator.fit(X, y)

# Print the parameters that it learns
print('Intercept: ', estimator.intercept_)
print('Coefficients: ', estimator.coef_)

('Intercept: ', -137.18973502289174)
('Coefficients: ', array([ 3.38381748]))
```

We can show the training examples and the model on the same plot:
Classification

In classification, we are given an object and we predict to which of a finite (and usually small) set of classes
the object belongs. So we formulate it as follows:

- We assume we have a finite set of labels, $C$.
- Given an object, our task is to assign one of the labels $y \in C$ to the object.

An example of classification is predicting whether a student will fail or pass his/her programming module. Features and their values describe the students, e.g. Craig's lecture attendance, $lect$, is 60%, his lab attendance, $lab$, is 45%, and his leaving certificate points, $cao$, are 500. We want a method to predict label $y \in \{\text{fail, pass}\}$.

Again, we could ask an expert for a model, e.g. Derek's model might make predictions as follows:

$$y = \begin{cases} 
\text{fail} & \text{if } lect \leq 80 \land lab \leq 80 \\
\text{pass} & \text{otherwise}
\end{cases}$$

But in this module, again, we are more interested to learn the model from data.
I have collected a dataset of student performances in a programming module. Without going into any details, we'll learn a model from this data, using all of it as the training set.

```python
In [9]: from sklearn.linear_model import LogisticRegression

# Read CSV file
df = pd.read_csv("dataset-cs1109.csv")

# Get the feature-values into a separate numpy arrays of numbers and
# the target values into a separate
# numpy arrays of ints
X = df[['lect', 'lab', 'cao']].values
y = df['outcome'].values

# Create logistic regression object
estimator = LogisticRegression()

# Train the model using the data
estimator.fit(X, y)

Out[9]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                           intercept_scaling=1, penalty='l2', random_state=None, tol=0.0001)
```

And let's see what it predicts for Craig, where a prediction of 0 means *fail* and 1 means *pass*:

```python
In [10]: estimator.predict([[60, 45, 500]])

Out[10]: array([0])
```

He fails!