

Introduction to Machine Learning

» Admin

- * Me: Doug Leith www.scss.tcd.ie/doug.leith/
- * Module material: on Blackboard
- * Lots of online resources, but do be *careful*, esp. of blogs etc.
Coursera:
 - * www.coursera.org/learn/python-machine-learning
 - * www.coursera.org/learn/neural-networks-deep-learning
- * We'll use Python and sklearn package.
- * 40% marks for weekly assignments, 60% for final assignment
- * Prerequisites: python programming.

» Honour Code

- * Its ok (good in fact) to discuss the weekly assignments with other students, but you *must*:
 - (i) Write the answers yourself in your own words (no chatGPT etc)
 - (ii) Write your own code (no chatGPT etc), and don't share your code with others
 - (iii) You must explain/justify how you obtained your answer. So if you write code to carry out a calculation you need to discuss/explain what that code does, and if you present numerical results you need to discuss their interpretation. Saying "see code", simply giving numbers or plots without comment → low mark.
- * Final assignment is individual work and must be done entirely yourself - no discussion with other students. Write the answers yourself in your own words, write your own code. No chatGPT etc.
- * All submissions will be checked by anti-plagiarism software and each week a random sample will also be checked manually.
- * *Zero tolerance for plagiarism = referral to college authorities for disciplinary action.*

» Module Structure

- * First 3 weeks: quite intense, aim is to quickly get to the point where you can really apply machine learning techniques to real data
- * Then start drilling down on:
 - * Models (kNN, decision trees, neural nets, kernel methods) → will take up us to mid-term break
 - * Feature engineering and deep learning after mid-term
- * Weekly assignments are closely linked to lectures, be careful not to fall behind → remember they're worth a good chunk of the final mark





» Supervised Machine Learning

We'll mainly look at *supervised machine learning*, although we'll also touch on *unsupervised machine learning* towards end of module.

- * Aim of supervised learning to predict output/target value from labelled data
- * *Classification* e.g. predict whether a credit card transaction is fraudulent or not. Target values are discrete classes (fraudulent, not fraudulent).
- * *Regression* e.g. predict distance between two people based on received signal strength of Bluetooth beacons. Target values are continuous values (received signal strength e.g. -60dB)

» Classification Example

Training set

X Sample	Y Target Value (Label)
 x_1	Apple y_1
 x_2	Lemon y_2
 x_3	Apple y_3
 x_4	Orange y_4

- * Using the *training data* the classifier learns to predict the label Apple, Orange etc from input x_1, x_2, \dots
- * Input x is a numeric vector e.g. pixels in image. Also called a *feature* vector.
- * Output y is also a number e.g. 1 for Apple, 2 for Orange.
- * *Classifier* is a function that maps from x to prediction \hat{y} of output y :

$$\hat{y} = h(x)$$

- * Now given a new input x the classifier predicts its label
- * So we need to: (i) map from real input (image etc) to numeric feature vector x , (ii) learn function $h(x)$

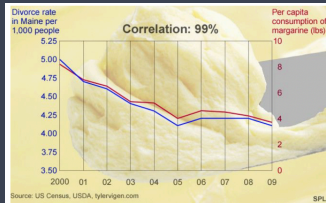
» Where Do We Get Training Data?

- * We need labelled data e.g. images of fruit plus the labels (whether apple, orange etc).
- * Usually easy to get the raw data (the images) but often much harder to get the labels.
- * **Get humans to label the data** e.g. use Amazon Mechanical Turk, Captchas, or just employ people directly.
 - * Its boring work, can be error prone
 - * Low paid human labour is the basis for “sophisticated” machines
- * **Leverage past human labour**, e.g.
 - * Wikipedia text, documents and their manual classification
 - * Academic papers and their manual keywords/classification
- * **In online services we might be able to extract the label by logging outcomes**, e.g. if flag a transaction as potentially fraudulent and later analysis confirms/denies this.
 - * But often hard to log the outcome of actual interest e.g. if show an ad its hard to link a later purchase to display of the ad.
 - * Instead can use indirect outcomes e.g. show an ad and user clicks on it. Many issues with this though, need to be careful

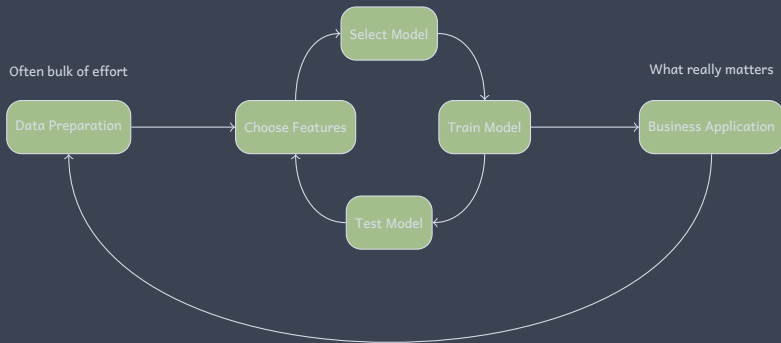
» Where Do We Get Training Data?

What can go wrong?

- * Data is unrepresentative E.g.
 - * Collected only from students or males, so not representative of the whole population
 - * Was collected in the past and things have changed
- * Data is too “noisy” or unreliable E.g.
 - * Clicking on an ad may only be weakly correlated to an eventual purchase, and hard to make the link after a delay
- * Data fails to capture useful relationships
 - * Correlation vs causation



» Machine Learning Workflow



» Example: Movie Review Sentiment Analysis

- * IMDb movie reviews
<http://www.cs.cornell.edu/people/pabo/movie-review-data/>
- * E.g. “that’s exactly how long the movie felt to me. there weren’t even nine laughs in nine months. it’s a terrible mess of a movie starring a terrible mess of a man, mr. hugh grant ...”
- * Our task: Given the text of a review our task is to say whether it is a positive or negative review
- * We have *training data* i.e. a set of reviews which have already been labelled as positive or negative.
- * So how should we start ?

» Example: Movie Review Sentiment Analysis

- * Some words are positive, some negative e.g. wonderful, great, terrible, awful
- * Count the number of positive words and the number of negative words. Predict review is positive if more positive words, negative if more negative words
- * How to automate this idea?
 - * We have labelled reviews (training data), so use these to learn the positive and negative words ...

» Bag of Words

- * Delete uninteresting words from reviews (and, of, the), called *stop words*.
- * Truncate word endings e.g. happening, happened, happens → happen, called *stemming*
- * Collect resulting set of truncated words or *tokens* into a *dictionary* (just a big list of N words)
- * Map text for a review to an array of size N , entry for word i is the number of times it appears in review. E.g. “a terrible mess of a movie starring a terrible mess of a man , mr. hugh grant” has non-zero entries:

grant	10287,	1
hugh	11485,	1
mess	14967,	2
mr	15553,	1
starring	22491,	1
terrible	23718,	2

» Linear Model

- * Assign weight θ_i to word i in dictionary (we'll come back to how to choose θ_i). So we have N weights, one for each word in the dictionary.
- * Given text for a review map it to an array x of size N and calculate

$$z_i = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_N x_N = \sum_{i=1}^n \theta_i x_i = \theta^T x$$

where x_i is the i 'th element of array x (corresponding to i 'th word in dictionary) and θ is array with i 'th element θ_i .

- * If $z_i > 0$ predict review is positive, otherwise predict its negative i.e. prediction is

$$\hat{y}_i = \text{sign}(\theta^T x)$$

where $\text{sign}(z) = +1$ when $z > 0$, -1 when $z < 0$ and 0 when $z = 0$.

» Linear Algebra Notation

We'll just be using this for notation:

- * Vector $x = \begin{bmatrix} 230.1 \\ 37.8 \end{bmatrix}$, element $x_1 = 230.1$
- * Matrix $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, element $A_{11} = 1$
- * Transpose $x^T = [230.1, 37.8]$
- * Inner product $x^T y = \sum_{i=1}^n x_i y_i$ for two vectors with n elements

If you want a refresh, there is lots of revision material online e.g.

- * <https://www.coursera.org/lecture/machine-learning/matrices-and-vectors-38jIT>
- * <https://www.khanacademy.org/math/linear-algebra>

» Training Our Model

- * We still need to select values for the weights θ . Lots of ways to do that ...
- * We have training data so select θ to make predictions as accurate as possible for that data - *training* our model
 - * We need to decide how to measure errors i.e. select a *cost function*
 - * We need an *optimization algorithm* that minimises errors
- * Its also an option to use our insight to select θ , e.g. select the weight for word i to be higher when that word is more frequent in positive reviews.

» Example: Movie Review Sentiment Analysis

- * Using Logistic Regression cost function (come back to this later), three words with most negative weights are:

worst -1.220056

supposed -0.871509

boring -0.817451

and three most positive are:

family 0.935607

truman 0.877639

excellent 0.775348

- * Words seem a bit strange ?
- * These weights predict review sentiment with 100% accuracy for training data.
 - * Is that what we expect?
 - * Is this a good test of prediction performance?

» Try For Yourself

- * Download dataset from *http://www.cs.cornell.edu/people/pabo/movie-review-data/review_polarity.tar.gz*
- * Unzip file. This creates folder *txt_sentoken* with subfolders *pos* and *neg*.
- * Python code:

```
from sklearn.datasets import load_files
d = load_files("txt_sentoken", shuffle=False)
x=d.data; y=d.target
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_df=0.2)
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2)
Xtrain = vectorizer.fit_transform(xtrain)
Xtest = vectorizer.transform(xtest)
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(Xtrain, ytrain)
preds = model.predict(Xtest)
from sklearn.metrics import classification_report
print(classification_report(ytest, preds))
```

» Key Ideas

- * *Feature engineering.* We mapped from input text to an array of numbers. Lots of ways we could do this.
- * *Model selection.* Our prediction was $\text{sign}(\theta^T x)$, an example of a linear model. Lots of other models are possible.
- * *Cost function selection.* We used Logistic cost function
- * *Optimisation.* We selected weights to minimise this cost function
- * *Evaluating performance.*