Lecture 1: Introduction to machine learning

Course overview

Description and objectives

This module covers how to apply machine learning techniques to large data-sets -- so-called *big-data*.

An introduction to machine learning (ML) is presented to provide a general understanding of the concepts of machine learning, common machine learning techniques, and how to apply these methods to data-sets of moderate sizes.

Deep learning and computing frameworks to scale machine learning techniques to big-data are then presented.

Scientific data formats and data curation methods are also discussed.

Syllabus

Foundations of ML (e.g. overview of ML, training, data wrangling, scikit-learn, performance analysis, gradient descent), data formats and curation (e.g. serialisation, deserialisation, domain specific languages, markup languages, controlled vocabularies, semantic models), ML methods (e.g. logistic regression, SVMs, ANNs, decision trees, ensemble learning and random forests, dimensionality reduction), deep learning and scaling to big-data (e.g. TensorFlow, Deep ANNs, CNNs, RNNs, Autoencoders, Cloud computing) and applications of ML in astrophysics, high-energy physics and industry.

Prerequisites

Students should have a reasonable working knowledge of Python, some familiarity with working in the command line environment in Linux/Unix based operating systems, and a general understanding of elementary mathematics, including linear algebra and calculus.

No previous familiarity with machine learning is required.

Resources

Textbooks

- VanderPlas, "*Python data science handbook*", O'Reilly, 2017, ISBN 9781491912058 (Example code (https://github.com/jakevdp/PythonDataScienceHandbook))
- Geron, "*Hands-on machine learning with Scikit-Learn and TensorFlow*", O'Reilly, 2017, ISBN 9781491962299 (Example code (https://github.com/ageron/handson-ml))
- Goodfellow, Bengio, Courville (GBC), "*Deep learning*", MIT Press, 2016, ISBN 9780262035613 (Website (http://www.deeplearningbook.org))

Code frameworks and libraries

- Scikit-Learn (http://scikit-learn.org/stable/)
- TensorFlow (https://www.tensorflow.org/)

Tutorials

- Scikit-Learn tutorial (https://github.com/jakevdp/sklearn_tutorial), VanderPlas

Jupyter notebooks

Each lecture has an accompaning Jupyter notebook, with executable code.

Jupyter notebooks

Each lecture has an accompaning Jupyter notebook, with executable code.

These slides are a Jupyter notebook.

Notebooks can be viewed in slide mode using RISE (https://github.com/damianavila/RISE).

Jupyter notebooks

Each lecture has an accompaning Jupyter notebook, with executable code.

These slides are a Jupyter notebook.

Notebooks can be viewed in slide mode using RISE (https://github.com/damianavila/RISE).

The supporting Jupyter notebooks thus serve as the course *slides*, *lecture notes*, and *examples*.

Course philosophy

This is a practical, hands-on course. While we will cover basic concepts and background theory (but not in great mathematical depth or rigor), a large component of the course will focus on implementing and running machine learning algorithms.

The lectures will include many code examples and exercises to be completed in class.

Versions of the course Jupyter notebooks -- without solutions to exercises -- will be made available before each lecture. Students can then follow examples by running code live in class (and inspecting variables and making modifications). Exercises will be completed by adding to these notebooks during class.

Schedule

We will have one 3 hour session of three back-to-back lectures each week, with a short break between lectures.

Assessment

- Coursework: 30%
- Exam: 70%

Coursework

Coursework will involve downloading a Jupyter notebook, which you will need to complete.

Throughout the notebook you will need to complete code, analytic exercises and descriptive answers. Much of the grading of the coursework will be performed automatically.

The coursework will be issued after the first 9 lectures, which contain the material required to complete the coursework.

Exam

Answer FOUR questions of the FIVE questions provided.

Exam duration is *TWO hours*.

Each question has equal mark (20 marks per question).

Markers place importance on clarity and a portion of the marks are awarded for clear descriptions, answers, drawings, and diagrams, and attention to precision in quantitative answers.

Computing setup

Students are expected to bring their own laptops to class in order to run notebooks and complete examples.

All examples are implemented in Python 3.

The main Python libraries that are required include the following:

- `numpy`
- `scipy`
- `matplotlib`
- `scikit-learn`
- `ipython/jupyter`
- `seaborn`
- `tensorflow`

A python environment is required to run the notebooks.

Instructions for using Anaconda are included here, although other Python setups could be used. An environment to run the notebooks can be set up with the versions of the libraries in `requirements.txt` (included below), following the steps below in terminal (MacOS) or anaconda prompt (Windows):

1. Create an environment named MLBDenv with the following libraries in the `requirements.txt` file installed. (conda-forge is needed for the rdflib library) `conda config --append channels conda-forge conda create --name MLBDenv python=3.7 --file requirements.txt`

2. Go into the `MLBDenv` environment and then create a kernel which can run the notebook in this environment. `conda activate MLBDenv python -m ipykernel install --user --name MLBDenv --display-name "MLBDenv" conda deactivate`

3. When running a jupyter notebook, we can switch to the MLBDenv environment by clicking on the tab Kernel -> Change kernel -> MLBDenv.

Content of requirements.txt:

```
matplotlib=3.1.1
scipy=1.3.1
pandas=0.25.3
scikit-learn=0.21.3
seaborn=0.9.0
pillow=6.2.1
ipykernel=5.1.3
tensorflow=1.14.0
mako=1.1.0
sqlalchemy=1.3.12
pyyaml=5.2
ply=3.11
rdflib=4.2.2
lxml=4.4.2
```

What is machine learning?

Artifical intelligence (AI)

Ironically...

- Solving "computational problems" that are difficult for humans is straightforward for machines (i.e. problems described by list of formal mathematical rules).
- Solving "intuitive problems" that are easy for humans is difficult for machines (i.e. problems difficult to describe formally).

This is often known as Moravec's paradox (https://en.wikipedia.org/wiki/Moravec%27s_paradox) (although formal definition is a little more specific).

Artifical intelligence (AI)

Ironically...

- Solving "computational problems" that are difficult for humans is straightforward for machines (i.e. problems described by list of formal mathematical rules).
- Solving "intuitive problems" that are easy for humans is difficult for machines (i.e. problems difficult to describe formally).

This is often known as Moravec's paradox (https://en.wikipedia.org/wiki/Moravec%27s_paradox) (although formal definition is a little more specific).

Solution is to allow computers to learn from experience and to build an understanding of the world through a hierarchy of concepts.

Knowledge base approach

Hard-code knowledge about world in formal set of rules and use logical inference.

Very difficult to capture complexity of intuitive problems in this manner.

Machine learning (ML)

Arthur Samuel (1959):

> *"[Machine learning is the] field of study that gives computers the ability to learn without being explicitly programmed."*

Tom Mitchell (1997):

> *"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."*

Uses of machine learning

1. **Prediction:** Predict outcome given data.
2. **Inference:** Better understand data (and their distribution).

Data representations

Performance of machine learning depends on representation of data given.

Data represented to learning algorithm as "*features*".

Traditional approach to machine learning involved *"feature engineering"*, where a practitioner with domain expertise would develop techniques to extract informative features from raw data.

Examples of features

- Computer visions: edges and corners
- Spam: frequency of words
- Character recognition: histograms of black pixels along rows/columns, number of holes, number of strokes

Learning representations

Alternative is to learn features.

- Can discover informative features from data.
- Minimal human intervention.

Learning representations

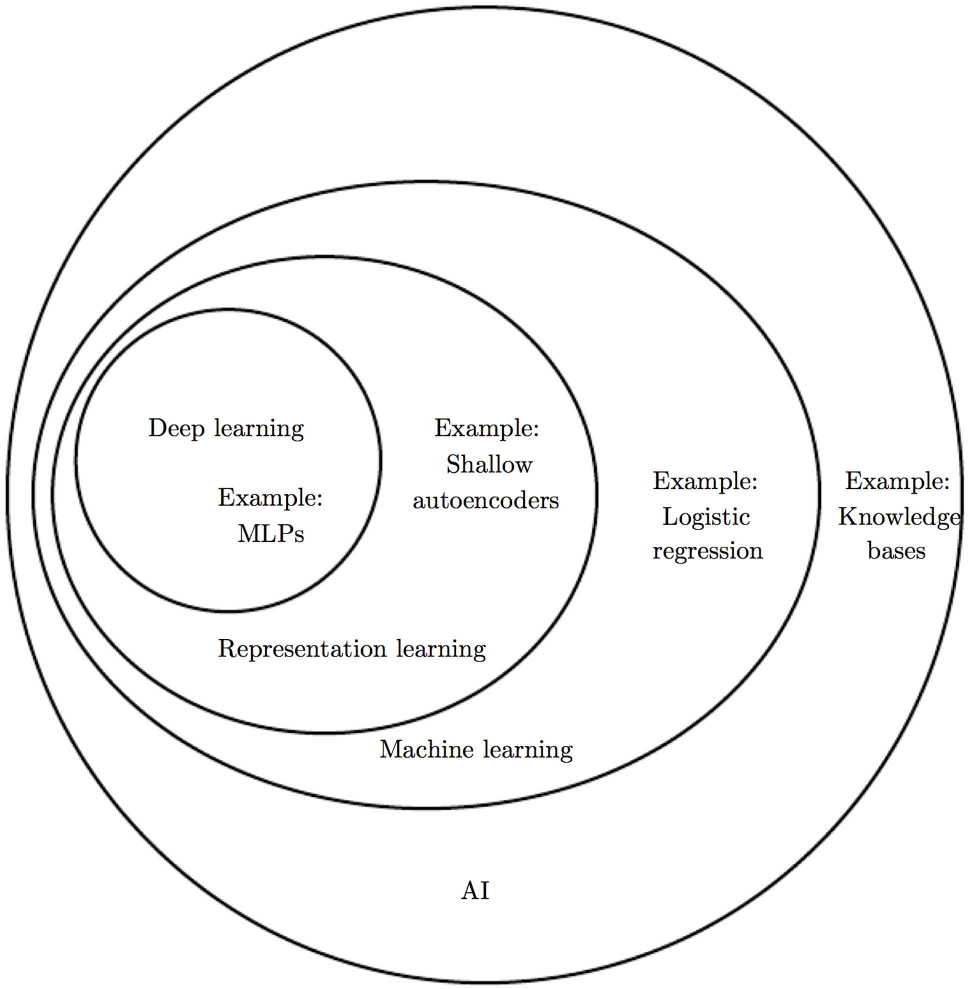
Alternative is to learn features.

- Can discover informative features from data.
- Minimal human intervention.

Approaches to representation learning

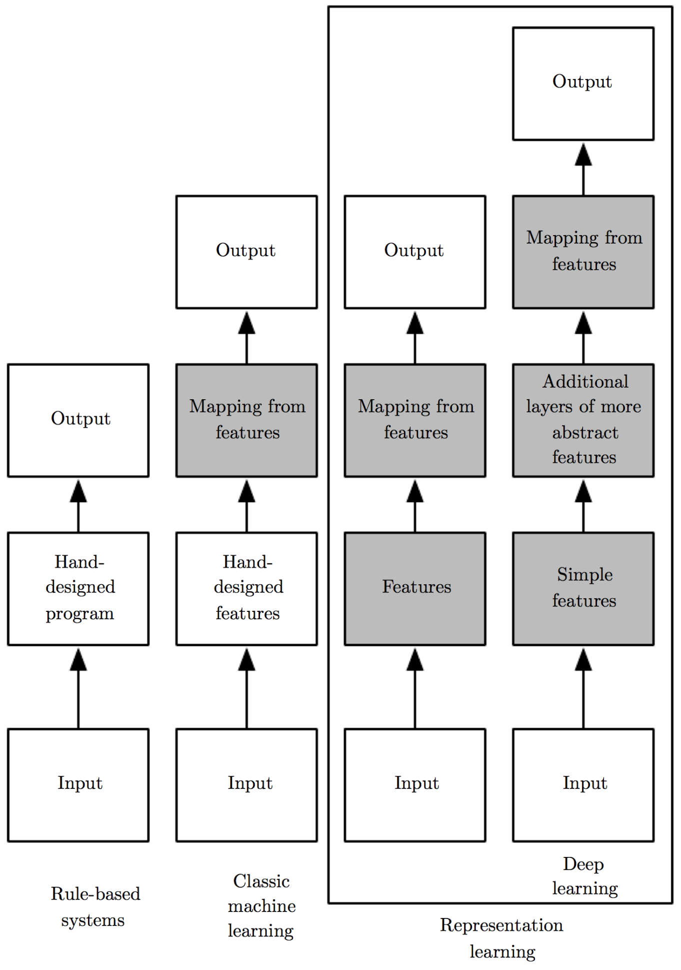
- Dedicated feature learning, e.g. autoencoder combining encoder and decoder.
- Representation learning integral to overall machine learning technique, e.g. deep learning.

Approaches to artifical intelligence



[Image credit: GBC (http://www.deeplearningbook.org/)]

AI pipelines



[Image credit: GBC (http://www.deeplearningbook.org/)]

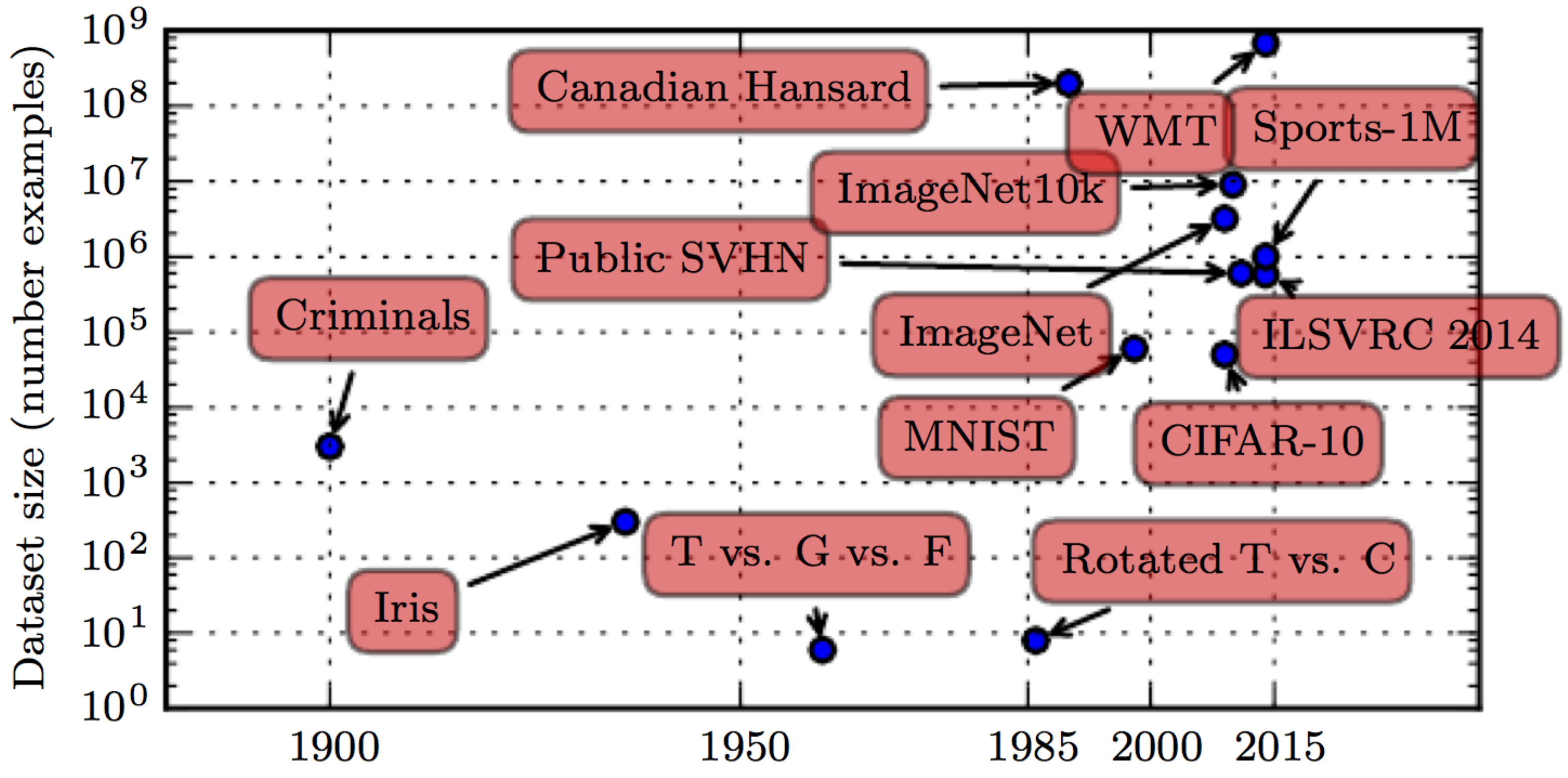
The unreasonable effectiveness of data

As society becomes increasing digitised, the volume of available data is exploding.

A significant increase in the volume of data can lead to dramatic increases in the performance of machine learning techniques.

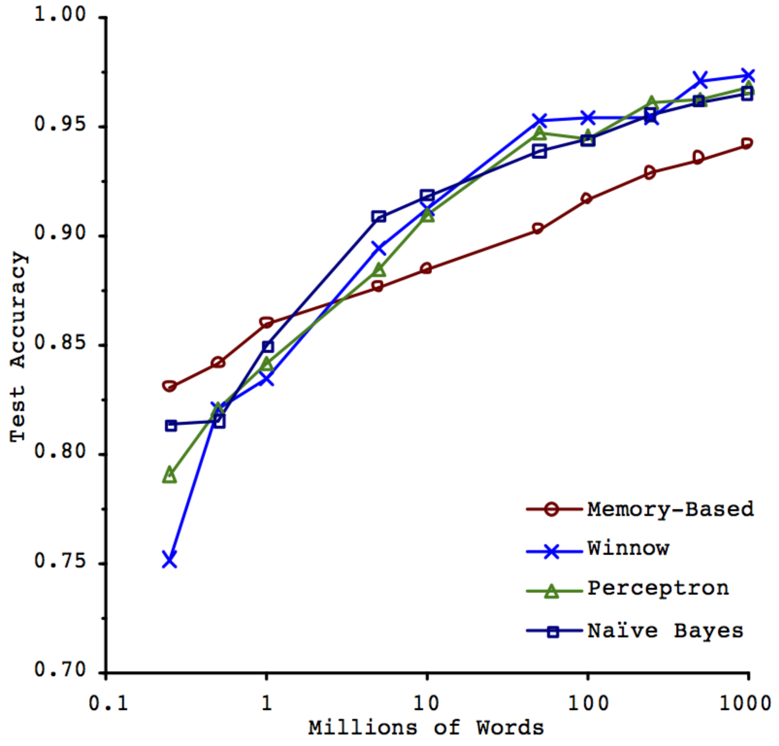
(Term coined in Halevy, Norbig & Pereira, 2009, *The unreasonable effectiveness of data* (http://goo.gl/q6LaZ8).)

Size of benchmark data-sets



[Image credit: GBC (http://www.deeplearningbook.org/)]

Size of data can have a larger impact than algorihm



Source: Banko & Brill, 2001, *Scaling to very very large corpora for natural language disambiguation* (http://goo.gl/R5enIE)

As a rule of thumb, a supervised deep learning algorithm will perform reasonably well with around 5,000 labelled samples.

With 10 million samples, it will match or exceed human performance.

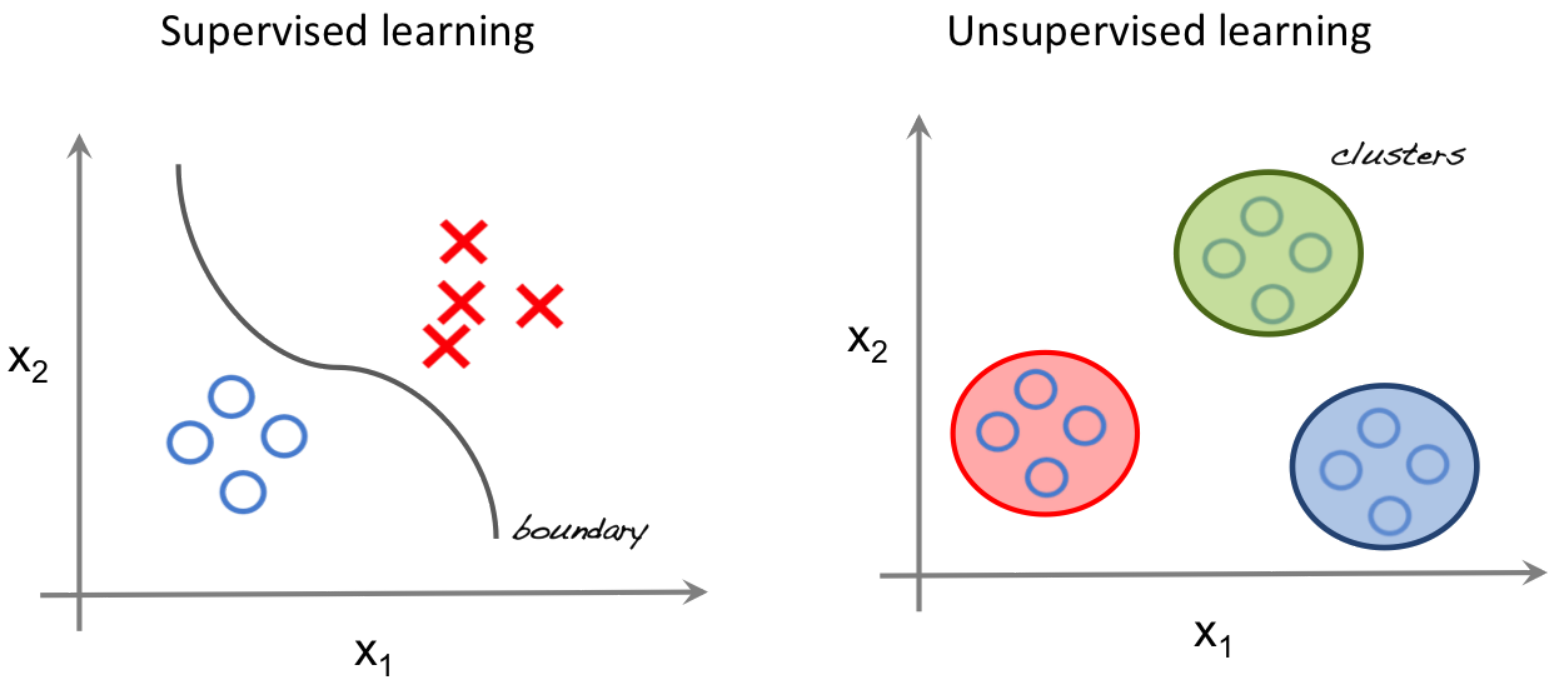
[Source: GBC (http://www.deeplearningbook.org/)]

However, in many cases very large datasets are not available and in some cases not possible.

Hence, developing effective algorithms remains critical.

Classes of machine learning

1. **Supervised:** Learn to predict output given input (given labelled training data).
2. **Unsupervised:** Discover internal representation of input.
3. **Reinforcement:** Learn action to maximise payoff.



[Image source (http://beta.cambridgespark.com/courses/jpm/01-module.html)]

Supervised learning

Learn to predict output given input (given labelled training data).

1. **Regression:** Target output is a (real) number,
   e.g. estimate flux intensity.

2. **Classification:** Target output is a class label,
   e.g. classify galaxy morphology.

How supervised learning works

- Select model defined by function f, and model target y from inputs x by
  $$y = f(x, \theta),$$
  where θ are the parameters of the model that are learnt during training.

How supervised learning works

- Select model defined by function f, and model target y from inputs x by
  $$y = f(x, \theta),$$
  where θ are the parameters of the model that are learnt during training.

- Learning typically involves minimising the difference between the inputs and outputs for the model, given a training data-set (more on training, validation and test data-sets later).

Unsupervised learning

Discover internal representation of input.

1. **Cluster finding:** Learn cluster of similar structure in data.
2. **Density estimation:** Learn representations of data (probability distributions).
3. **Dimensionality reduction:** Provides compact, low-dimensional representation of data.

Unsupervised learning

Discover internal representation of input.

1. **Cluster finding:** Learn cluster of similar structure in data.
2. **Density estimation:** Learn representations of data (probability distributions).
3. **Dimensionality reduction:** Provides compact, low-dimensional representation of data.

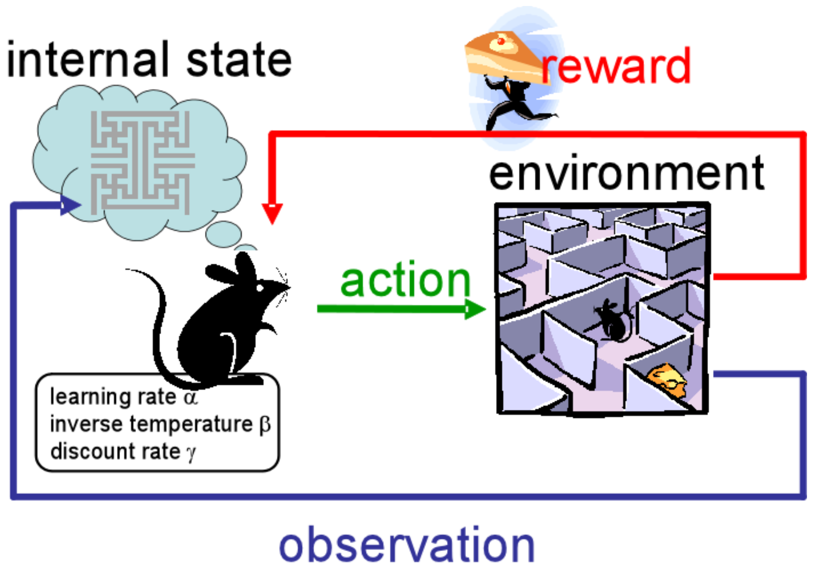
Unsupervised learning examples

Anomaly detection, clustering groups of similar objects, visualising high-dimensional data in 2D or 3D plots are examples of unsupervised learning.

Reinforcement learning

Learn action to maximise payoff.

- Output is an action or sequence of actions and the only supervisory signal is an occasional numerical (scalar) reward.
- Difficult since rewards are delayed.
- Not covered in this course.



[Image credit (https://www.analyticsvidhya.com/blog/2016/12/getting-ready-for-ai-based-gaming-agents-overview-of-open-source-reinforcement-learning-platforms/)]

Reinforcement learning examples

Go, playing computer games, driverless cars, self navigating vaccum cleaners, scheduling of elevators are all applications of reinforcement learning.

E.g. Google [DeepMind] machine learns to master video games (http://www.bbc.co.uk/news/science-environment-31623427)

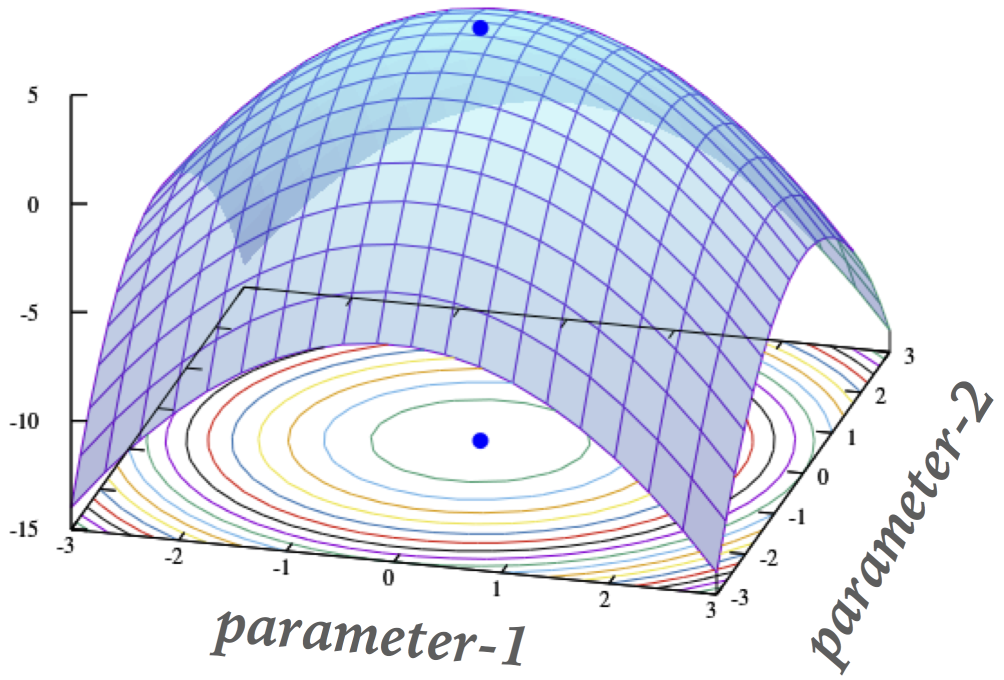
Training

Machine *learning* often involves solving an *optimization* problem, i.e. finding the parameters θ of the model f to best represent the training data (for supervised learning).

Objective function

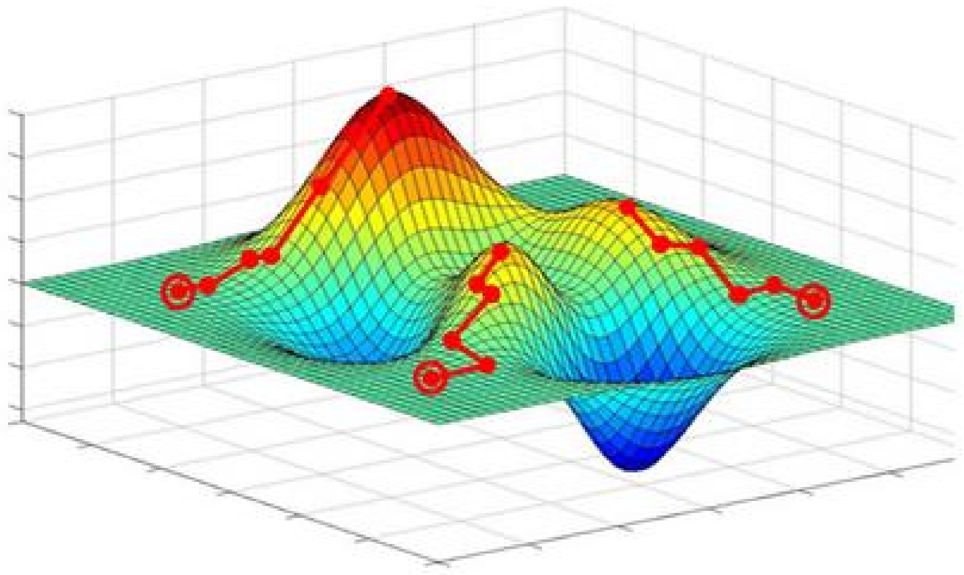
Typically maximise/minimise some goodness-of-fit/cost function.

Example of convex objective function



[Image credit: Kirkby, UC Irvine, LSST Dark Energy Summer School 2017]

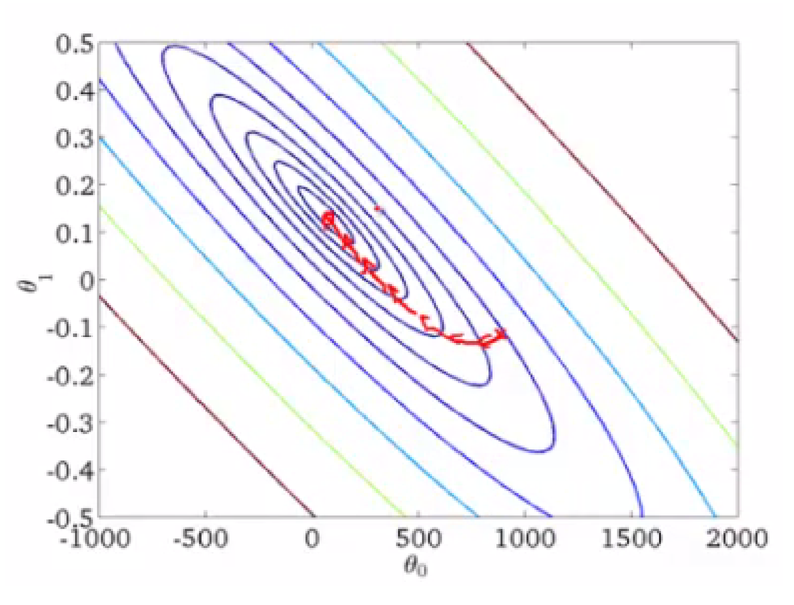
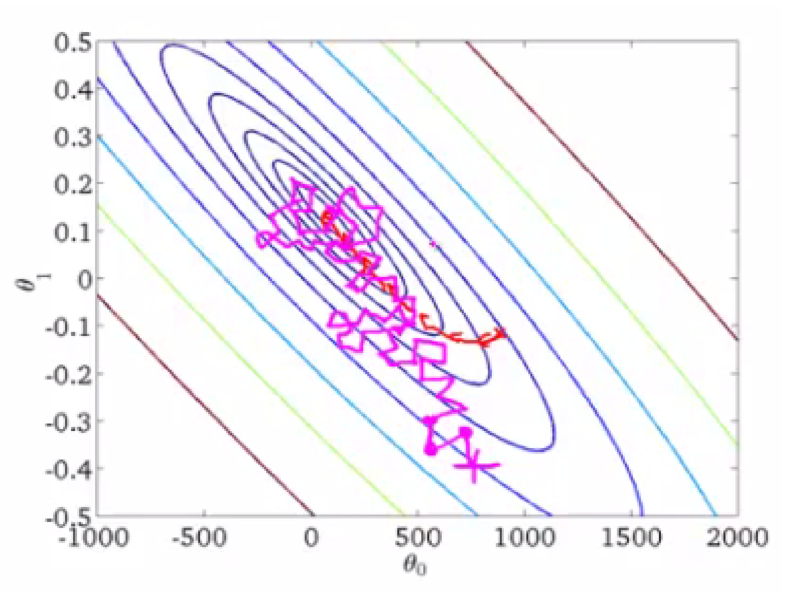
Example of non-convex objective function



[Image source (https://cs.hse.ru/data/2016/08/26/1121363361/moml.jpg)]

Using gradients to optimize objective function (i.e. perform training)

- **(Batch) Gradient descent:** Use all data at each iteration (full dimension).
- **Stochastic gradient descent:** Use a random data-point at each iteration (1 dimension).
- **Backpropagation:** propagate errors backwards through networks.

	
Batch gradient descent	Stochastic gradient descent

[Image source (http://www.holehouse.org/mlclass)]

Batch and online learning

Batch learning

Algorithm is trained using all available training data at once.

Also called *offline learning*.

Batch learning

Algorithm is trained using all available training data at once.

Also called *offline learning*.

- Requires substantial resources (CPU, memory space, disk space).
- If want to add new training data, must re-train from scratch on new full set of data (i.e. not just the new data but also the old data).

Online learning

Algorithm is trained using a sub-set of the training data.

Online learning

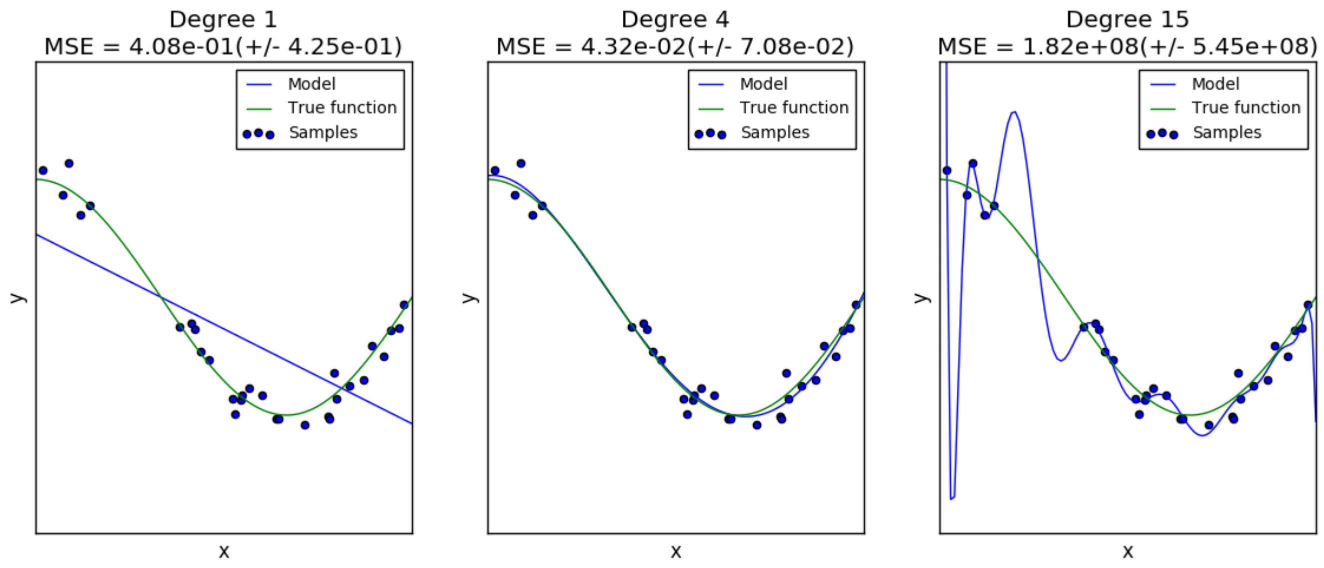
Algorithm is trained using a sub-set of the training data.

- Each learning step does *not* require substantial resources.
- Can integate new training data on the fly.
- May be able to throw away data once used it (although might not want to).
- If fed bad data, performance will decline.
- Noisy training.

Overfitting and underfitting

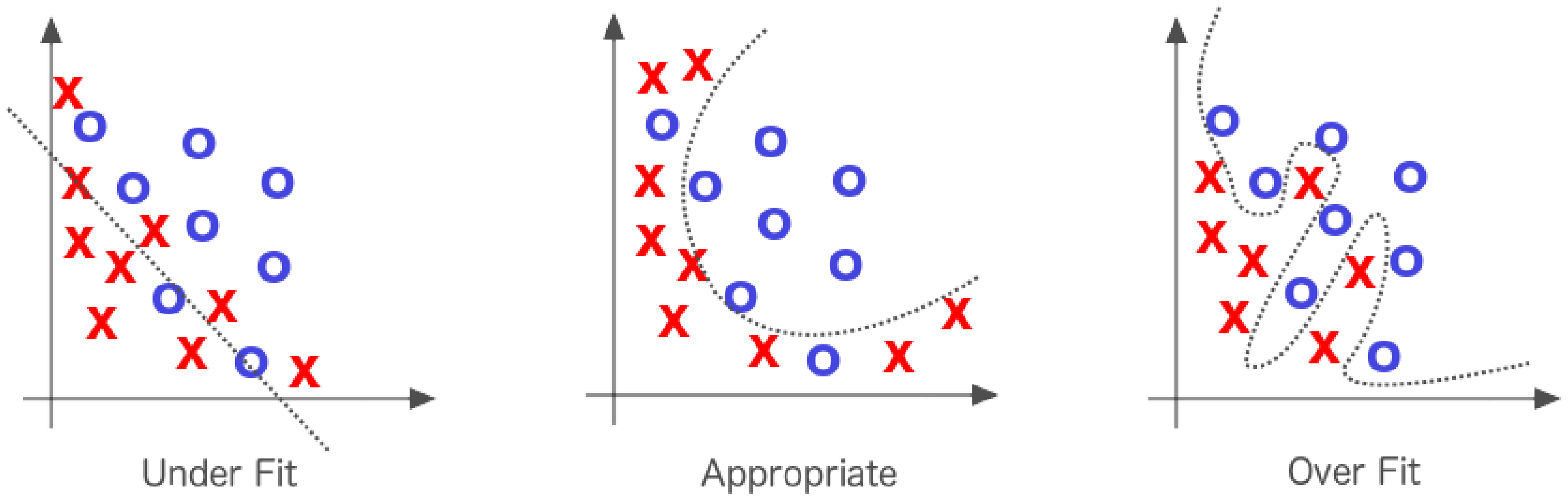
- **Problem:** The learned model may fit the training set extremely well but fail to generalise to new examples.

1D example



[Image source (http://scikit-learn.org/stable/_images/sphx_glr_plot_underfitting_overfitting_001.png)]

2D example



[Image source (https://www.safaribooksonline.com/library/view/deep-learning/9781491924570/assets/dpln_0107.png)]

Techniques to avoid overfitting

- Reduce complexity of model.
- Regularization:
  - Place additional constraints (priors) on features/parameters.
  - E.g. smoothness of parameters, sparsity of model (i.e. limit complexity).
- Split data into training, validation and test sets (e.g. cross-validation).

Testing and validation

No free lunch theorem

Essentially, all algorithms are equivalent when performance is averaged over all possible problems.

Consequently, there is no a priori model that is guaranteed to work best on all problems.

(Wolpert, 1996, *The lack of a priori distinctions between learning algorithms* (http://goo.gl/q6LaZ8))

It is therefore a matter of validating models empirically.

Training and test datasets

Split data into training and test sets (e.g. 80% for training and 20% for testing).

The model is trained on the *training set* and then tested on the *test set*.

Training and test datasets

Split data into training and test sets (e.g. 80% for training and 20% for testing).

The model is trained on the *training set* and then tested on the *test set*.

No data used in training the method is then used to evaluate it.

Training and test datasets

Split data into training and test sets (e.g. 80% for training and 20% for testing).

The model is trained on the *training set* and then tested on the *test set*.

No data used in training the method is then used to evaluate it.

Error rate on the test set is called the *generalization error* or *out of sample error*.

If the training error is low but the generalization error is high, it suggests the model is overfitted.

Hyperparameters

Many machine learning algorithms contain hyperparameters to control the model.

One (**bad**) approach is to evaluate alternative models defined by different hyperparameters on test set and select the model that performs best.

However, this optimizes the model for the test set and may not generalise to other data well.

Validation

A better approach is to split the data into three sets:

1. Training set
2. Validation set
3. Test set

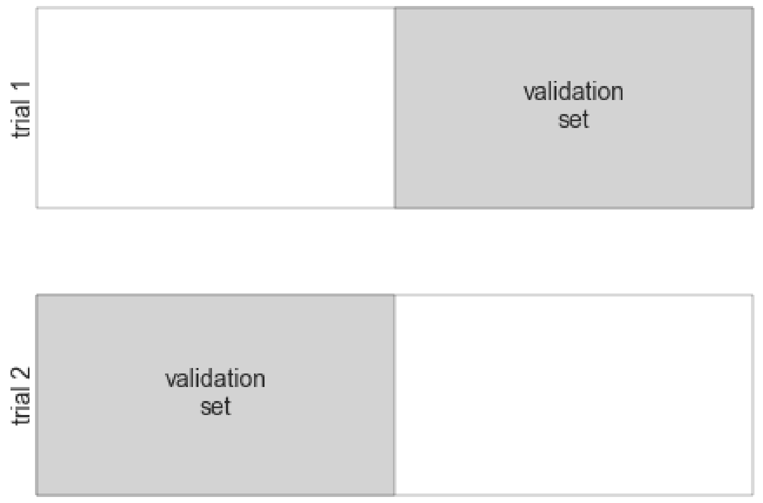
Train models on the training set and evaluate different models (with different hyperparameters) on the validation set.

Only once the final model to be used is fully specified should it be applied to the test set to estimate its generalization performance.

Cross-validation

A disadvantage of the previous approach is that less data are available for training.

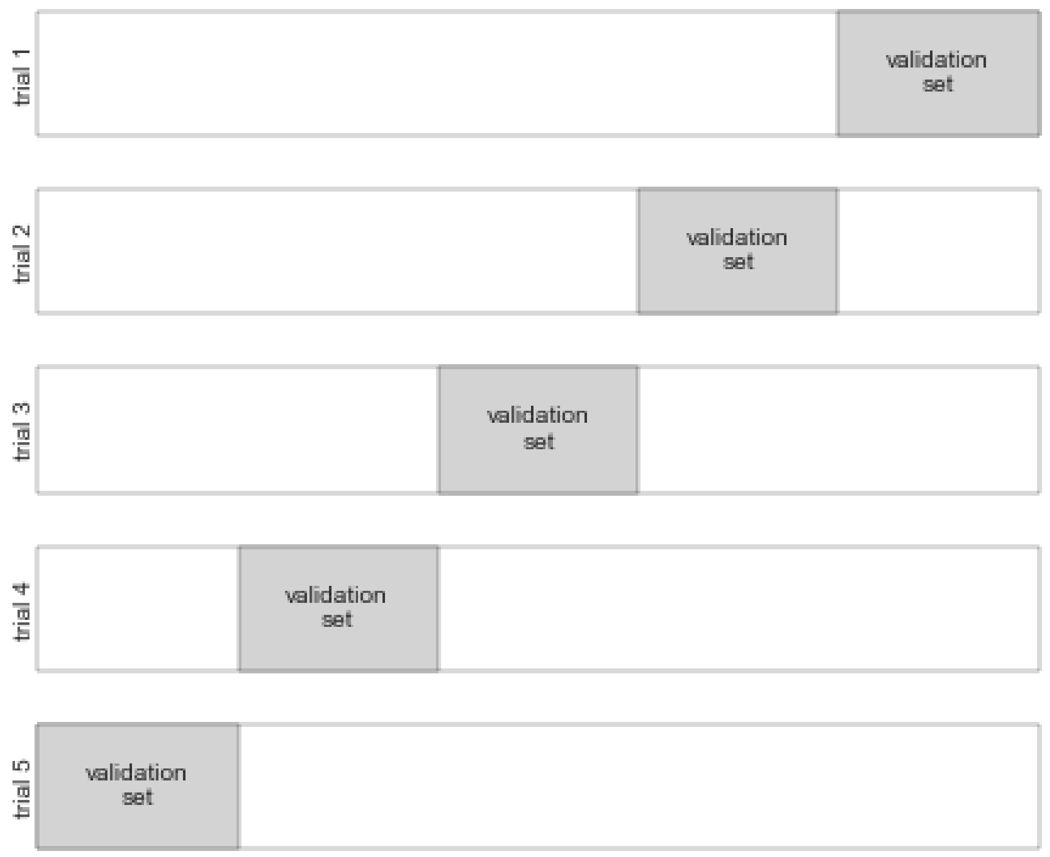
Cross-validation addresses this issue by performing a sequence of fits where each subset of the data is used both as a training set and a validation set.



[Image credit: VanderPlas (https://github.com/jakevdp/PythonDataScienceHandbook)]

Get validation accuracy scores for each trial, which could be combined.

Extension to n-fold cross-validation



[Image credit: VanderPlas (https://github.com/jakevdp/PythonDataScienceHandbook)]