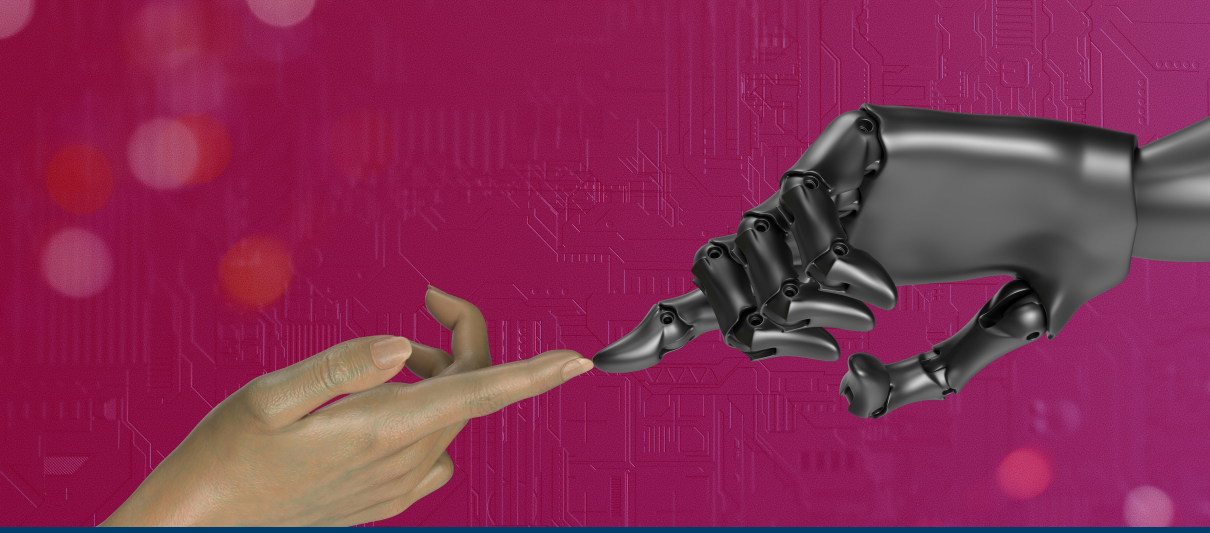


Hands-On Machine Learning

Dr. Derek Bridge | 2025/2026

UCC
Coláiste na hOllscoile Corcaigh
University College Cork, Ireland

School of Computer Science and Information Technology
Scoil na Ríomheolaíochta agus na Teicneolaíochta Faisnéise

1. Introduction

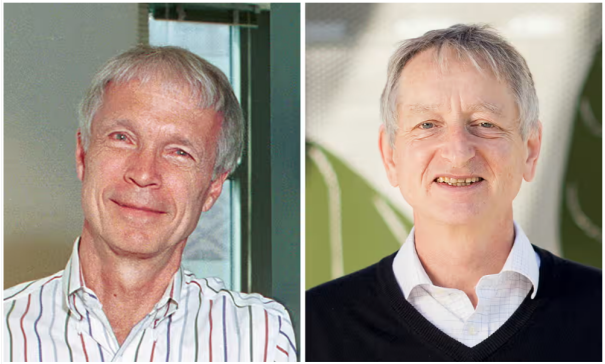
Dr. Derek Bridge

Position: Senior Lecturer
Location: Room 2.64, WGB
WWW: www.cs.ucc.ie/dbridge.html
Email: d.bridge@cs.ucc.ie

The Nobel Prize for Physics 2024

Machine learning pioneers win Nobel prize in physics

Geoffrey Hinton, 'godfather of AI', and John Hopfield honoured for work on artificial neural networks



The Nobel Prize for Chemistry 2024

"David Baker has succeeded [in] building entirely new kinds of proteins. Demis Hassabis and John Jumper have developed an AI model to solve a 50-year-old problem: predicting proteins' complex structures."

David Baker

"for computational protein design"

Demis Hassabis

"for protein structure prediction"

John Jumper

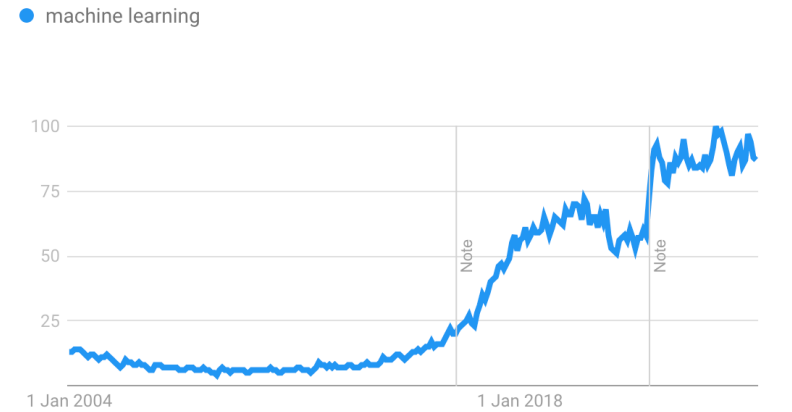
"for protein structure prediction"

Turing Award 2018

Yoshua Bengio, Geoffrey Hinton, and Yann LeCun

Turing Award 2024

Google Trends



Worldwide. 01/01/2004 - 16/05/2025. Web Search.

Drivers of Machine Learning

Algorithmic advances

- Core ideas have been around a long time:
  - Mathematical model of neurons (1940s).
  - Perceptrons (1950s).
  - k-nearest neighbours (1950s).
  - "Self-learning" game playing (1950s).
  - Decision trees (1960s or earlier).
  - Backpropagation (1980s or earlier).
  - Convolutional networks (1980s).
  - LSTMs (1990s).
  - Deep Networks (2000s/2010s)
- But a burst of new ideas from 2020 onwards.

Data

- Sensor data.
- Web data (text, images, audio, video, clicks).

Hardware

- Faster CPUs, then GPUs, now TPUs.
- HDDs of 40TB, SSDs of 100TB.

Freeware

- Software such as toolkits, libraries and APIs.
- Educational resources.

Money!

Machine Learning

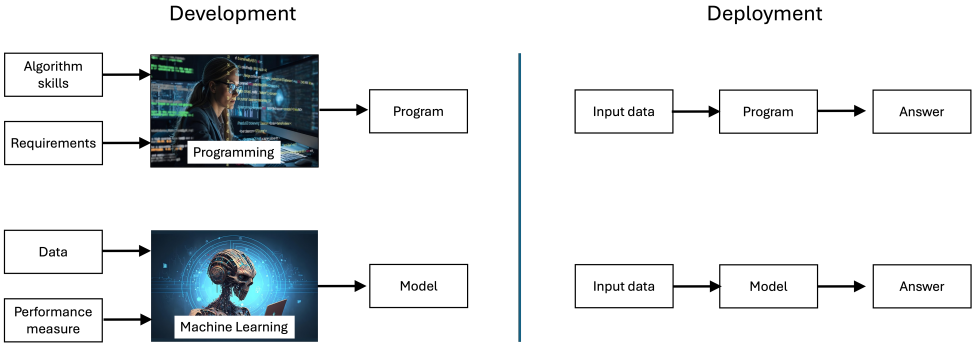
An early definition

the "field of study that gives computers the ability to learn without being explicitly programmed"
— Arthur Samuels, 1959

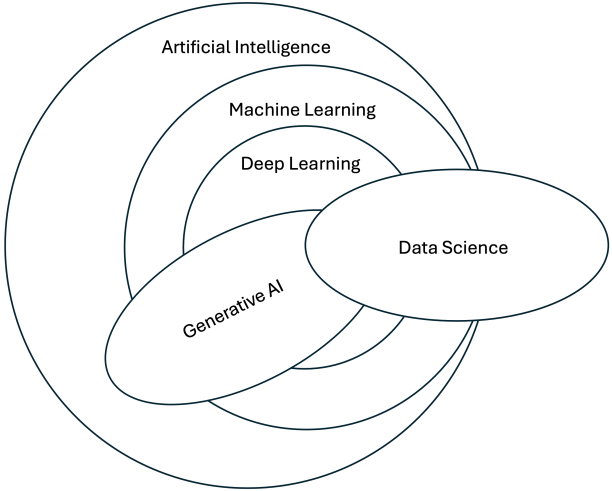
My definition

the development, analysis and deployment of algorithms that uncover patterns in training data which generalise to unseen data

Software Engineering vs. Machine Learning



Relationships with Other Fields



Types of Machine Learning

Unsupervised learning

- Given an (unlabeled) dataset
- Finds structure within the data, e.g.:
  - Clustering
  - Dimensionality reduction
  - Anomaly detection
  - Association rule mining
- Does this without the guidance of labels or rewards

Supervised learning

- Given a labeled dataset
- Learns to make predictions
  - Regression: predicting numeric values
  - Classification: predicting class membership
- Guided by the labels

Reinforcement learning

- Given an agent situated in an environment
- Learns the agent's action policy, e.g.:
  - for game-playing agents
  - for robot control
- Guided by action rewards

Applications of Machine Learning: A Snapshot from CIST @ UCC

Derek Bridge

Product recommendation for small-scale retailers
M Kaminskas, D Bridge, F Foping, D Roche
E-Commerce and Web Technologies: 16th International Conference on Electronic …

Textual case-based reasoning for spam filtering: A comparison of feature-based and feature-free approaches
SJ Delany, D Bridge
Artificial intelligence review 26, 75-87

Case-based support for forestry decisions: How to see the wood from the trees
C Nugent, D Bridge, G Murphy, BH Øyen
Case-Based Reasoning Research and Development: 8th International Conference …

Cathal Hoare

Urban building energy performance prediction and retrofit analysis using data-driven machine learning approach
U Ali, S Bano, MH Shamsi, D Sood, C Hoare, W Zuo, N Hewitt, …
Energy and Buildings 303, 113768

Building occupancy detection and localization using CCTV camera and deep learning
S Hu, P Wang, C Hoare, J O'Donnell
IEEE Internet of Things Journal 10 (1), 597-608

Multiclass sentiment classification of online health forums using both domain-independent and domain-specific features
R Alnashwan, H Sorensen, A O'Riordan, C Hoare
proceedings of the Fourth IEEE/ACM International Conference on Big Data …

Ken Brown

Forecasting workload in cloud computing: towards uncertainty-aware predictions and transfer learning
A Rossi, A Visentin, D Carraro, S Prestwich, KN Brown
Cluster Computing 28

Rapid Quantification of NaDCC for Water Purification Tablets in Commercial Production Using ATR-FTIR Spectroscopy Based on Machine Learning Techniques
H Asadi, T O'Mahony, J Lambert, KN Brown
Irish Conference on Artificial Intelligence and Cognitive Science, 106-120

Motion Sensors-Based Machine Learning Approach for the Identification of Anterior Cruciate Ligament Gait Patterns in On-the-Field Activities in Rugby Players
S Tedesco, C Crowe, A Ryan, M Sica, S Scheurer, AM Clifford, KN Brown, …
Sensors 20 (11), 3029

Luca Longo

Assessment of Mental Workload: A Comparison of Machine Learning Methods and Subjective Assessment Techniques
K Moustafa, S Luz, L Longo
Longo, L., & Leva, M. C. (Eds.). (2017). Human Mental Workload: Models and …

A Review on Machine Learning Methods for Customer Churn Prediction and Recommendations for Business Practitioners
A Manzoor, MA Qureshi, E Kidney, L Longo
IEEE Access

Schizo-Net: A *novel* Schizophrenia Diagnosis Framework Using Late Fusion Multimodal Deep Learning on Electroencephalogram-Based Brain Connectivity Indices
N Grover, A Chharia, R Upadhyay, L Longo
IEEE Transactions on Neural Systems and Rehabilitation Engineering 31, 464-473

Applications of Machine Learning: A Snapshot from CIST @ UCC

Hoang D. Nguyen

A Cough-based deep learning framework for detecting COVID-19
T Hoang, L Pham, D Ngo, HD Nguyen
2022 44th Annual International Conference of the IEEE Engineering in …

Federated Artificial Intelligence for Unified Credit Assessment
MD Hoang, L Le, AT Nguyen, T Le, HD Nguyen
International Conference on Human-Computer Interaction, 44-56

Carbon stock estimation at scale from aerial and satellite imagery
A To, HQV Pham, QH Nguyen, JG Davis, B O'Sullivan, SL Pan, ...
IEEE Conference on Artificial Intelligence (IEEE CAI 2024)

Barry O'Sullivan

The potential for artificial intelligence to predict clinical outcomes in patients who have acquired acute kidney injury during the perioperative period
BJ Kelly, J Chevarria, B O'Sullivan, G Shorten
Perioperative Medicine 10, 1-7

Predicting judicial decisions: A statistically rigorous approach and a new ensemble classifier
A Visentin, A Nardotto, B O'Sullivan
2019 IEEE 31st International Conference on Tools with Artificial …

Explainable machine learning for the multiclass classification of diffuse reflectance spectroscopy signals in orthopaedic applications
N Rossberg, CL Li, S Andersson-Engels, B O'Sullivan, K Komolibus, ...
Data Science for Photonics and Biophotonics 13011, 83-94

Salvatore Tedesco

Predicting Three-Dimensional Ground Reaction Forces in Running by Using Artificial Neural Networks and Lower Body Kinematics
DS Komaris, E Pérez-Valero, L Jordan, J Barton, L Hennessy, B O'Flynn, ...
IEEE Access 7, 156779-156786

Comparison of Machine Learning Techniques for Mortality Prediction in a Prospective Cohort of Older Adults
S Tedesco, M Andrulli, MÅ Larsson, D Kelly, A Alamäki, S Timmons, ...
International Journal of Environmental Research and Public Health 18 (23), 12806

Machine learning based canine posture estimation using inertial data
M Marcato, S Tedesco, C O'Mahony, B O'Flynn, P Galvin
Plos one 18 (6), e0286311

Andrea Visentin

A Review of Electricity Price Forecasting Models in the Day-Ahead, Intra-Day, and Balancing Markets
C O'Connor, M Bahloul, S Prestwich, A Visentin
Energies 18 (12), 3097

Road Pavement Health Monitoring System Using Smartphone Sensing with a Two-Stage Machine Learning Model
K Zhao, S Xu, J Loney, A Visentin, Z Li
Automation in Construction

Deep learning pipeline for blood cell segmentation, classification and counting
A Rao, KW Kho, V Mykytiv, A Visentin
32nd Irish Conference on Artificial Intelligence and Cognitive Science

Machine Learning is not magic.

Machine Learning is just the detection of patterns in data.

Machine Learning is just algorithms, data structures and maths.

"Machine Learning is statistics minus any checking of models and assumptions."

— Brian D. Ripley

Dangers

Bias

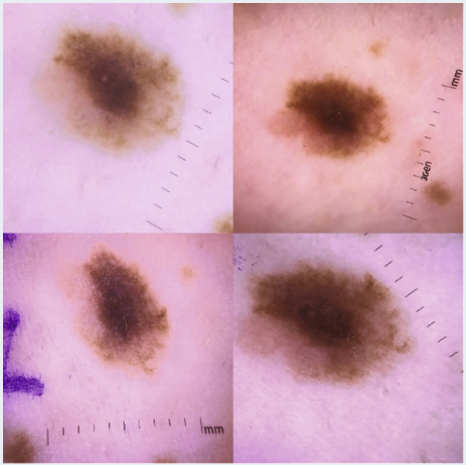
Insight - Amazon scraps secret AI recruiting tool that showed bias against women

By **Jeffrey Dastin**

October 11, 2018 1:50 AM GMT+1 · Updated 7 years ago

SAN FRANCISCO (Reuters) - Amazon.com Inc's machine-learning specialists uncovered a big problem: their new recruiting engine did not like women.

Shortcuts



Dangers

Surveillance

We're headed for big problems if gardaí get facial recognition technology

Ireland risks introducing a technology that scientific evidence has shown to be ineffective, inherently flawed and discriminatory. It's also a threat to fundamental rights

Expand

An Garda Síochána say they do not have a reference database to match images against, but they want access facial recognition technology anyway. Photograph: iStock

Privacy Violations

Netflix Spilled Your Brokeback Mountain Secret, Lawsuit Claims

An in-the-closet lesbian mother is suing Netflix for privacy invasion, alleging the movie rental company made it possible for her to be outed when it disclosed insufficiently anonymous information about nearly half-a-million customers as part of its $1 million contest to improve its recommendation system. The suit known as Doe v. Netflix (.pdf) was filed [...]

Environmental Impact

Figure 2. Direct and indirect environmental impacts of AI compute and applications

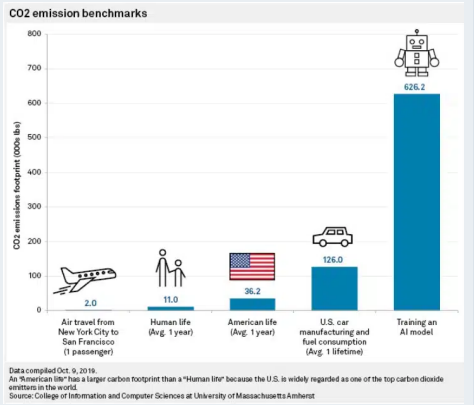
Direct environmental impacts AI compute resources lifecycle

Production	Transport	Operations	End-of-life
• Raw material extraction • Assembly • Manufacturing	• Distribution • Freight transportation • Handling & storage	• Energy consumption • Water consumption • Carbon footprint	• Collection & shipping • Dismantling & recycling • Waste disposal

Indirect environmental impacts AI applications

Positive impacts	Negative impacts
• Beneficial sectoral applications • Climate mitigation and adaptation • Environmental modelling and forecasting	• Harmful sectoral applications • Carbon leakage (net increase in emissions) • Consumption patterns and rebound effects

Source: OECD.AI Expert Group on AI Compute and Climate, literature review, expert interviews



CS3515 HOML

Contact time

Lectures

2×1 hour per week

Labs

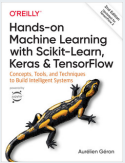
1×2 hours per week

Resources

Slides/Jupyter Notebooks

https://github.com/derekgbridge/homl_2025_2026

Book



(if you are skipping lectures/labs)

Software

scikit-learn
keras

Assessment

CA1, 50 marks

Classic ML
on a tabular dataset

CA2, 50 marks

Deep Learning
on an unstructured/ multimodal dataset

Academic Integrity

Academic misconduct includes

- **Poor Academic Practice**: Actions that include, but are not limited to, poor academic writing skills (e.g., poor referencing or passing off someone else's idea as your own), or small errors made through carelessness or misunderstanding.
- **Cheating**: Actions that attempt to get advantage by means that undermine values of integrity.
- **Contract Cheating / Essay Mills**: A form of academic misconduct where a person uses an undeclared and/or unauthorised third party, online or directly, to assist them in dishonestly producing work for academic credit or progression, whether or not payment or other favour is involved.
- **Cumulation effect**: Where continued poor academic practice and repeated minor instances of academic misconduct are treated as a case of major academic misconduct.
- **Fabrication/Falsification**: Making up data, experiments, or other significant information in proposing, conducting, or reporting research. This includes the fabrication or falsification of official University documents regarding credit and/or academic achievement.
- **Impersonation**: Undertaking in whole or in part any work required as part of a programme in the place of an enrolled learner, without permission from the provider.
- **Unethical Use of Generative Artificial Intelligence (GenAI)**: Generative Artificial Intelligence (GenAI) refers to a subset of artificial intelligence (AI) that uses algorithms and models trained on massive datasets to generate new text, audio, video, code, and more. Academic integrity is breached if students submit the products of GenAI as their own work without acknowledgement and without authorisation to use GenAI in fulfilling the task.
- **Plagiarism**: Presenting work or ideas taken from other sources without proper acknowledgement, whether done deliberately, carelessly, or inadvertently.
- Other acts that dishonestly use information to gain academic credit.

Plagiarism

Presenting work or ideas taken from other sources without proper acknowledgement, whether done deliberately, carelessly, or inadvertently.

It is a violation of UCC Policy and there are strict and severe penalties.

Types of Plagiarism include but are not limited to:

- **Collusion**: A joint effort of work is presented by an individual without due recognition of the input of others. Collusion also applies to both parties when an individual student provides their work to another student and allows them to present it as their own.
- **Self-plagiarism**: The use of one's own previous submitted/presented work in another context without appropriate citation.
- **Verbatim plagiarism**: Word-for-word copy of work from another source without providing acknowledgement.

You must read and comply with the Academic Integrity for Examinations and Assessments Policy 2025-2026.

The Policy applies to all work submitted, including software.

You can expect that your work will be checked for evidence of plagiarism or collusion.

In some circumstances it may be acceptable to reuse a small amount of work by others, but only if you provide explicit acknowledgement and justification.

If in doubt ask your module lecturer prior to submission. Better safe than sorry!

Reasonable Accommodations for Students Registered with DSS

- The university has a policy to assist students with disabilities.
- A Reasonable Accommodation does not refer to living arrangements. It is a professional term used to explain the range of supports and services available in University College Cork ("UCC") to assist students with disabilities/learning differences/significant ongoing health conditions to fully participate in their studies. In the education sector, Reasonable Accommodations are described as any action that helps alleviate a substantial disadvantage due to a disability/learning difference/significant ongoing health condition. Examples of Reasonable Accommodations can be found in Appendix 3 of the Reasonable Accommodations Policy.
- It is important to point out that the university does not oblige lecturers to record lectures and many do not regard it as the best option to enhance student learning.
- If you require reasonable accommodations, please register with DSS and discuss your situation with your lecturers.

Image credits

Photo by Igor Omilaev on Unsplash

Published in the Guardian

The Nobel Prize web site

Image from Silicon Republic based on images taken from the ACM

Image from the ACM

Image by Gerd Altmann from Pixabay

Image by fszalai from Pixabay

Image credits

Screenshot from reuters.com

Image taken from Akhila Narla, Brett Kuprel, Kavita Sarin, Roberto Novoa, Justin Ko, *Automated Classification of Skin Lesions: From Pixels to Practice*, Journal of Investigative Dermatology, vol.138(10), 2018, pp.2108-2110

Screenshot from www.irishtimes.com

Screenshot from wired.com

Taken from OECD Digital Economy Papers, No 341: *Measuring the Environmental Impacts of Artificial Intelligence Compute and Applications*

Image taken from earth.org but credited to spglobal.com using data from Emma Strubell, Ananya Ganesh and Andrew McCallum, *Energy and Policy Considerations for Deep Learning in NLP*, Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 2019, pp.3645-3650

From the O'Reilly web site

2. Supervised Learning

- We introduce some of the topics but using **synthetic data** — data that has been generated by a program that I wrote. For added realism the program injects some random noise into the data.
- The original idea for this data-generation program and its use in the later parts of the lecture come from Jake VanderPlas.

Prediction

- A lot of the time in Machine Learning, we want to create programs that make **predictions**.
- These programs are called **predictors** or **estimators**.
- (In everyday use, prediction is about the future; but we use the word more generally in Machine Learning.)

Regression

- **Regressors** predict continuous *numeric values*.
- E.g. given a description of a house, predict the selling price of the house.
- (Compare with **classification**.)

Dataset

A **dataset** is a set of **examples**.

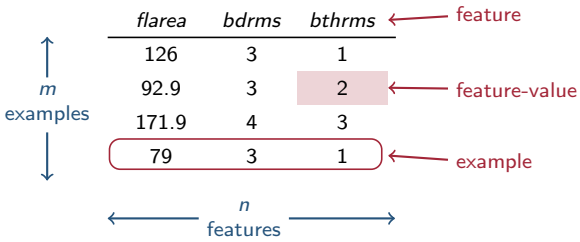
Examples

- An **example** is a single observation, representing an entity, event or situation.
- (An **example** may also be called an **instance** or a **sample**.)

Structured Dataset

- Each example fits a predefined schema.
- Most common is rows and columns, hence also called **tabular datasets**.
- (Compare with **unstructured dataset** and **mulitmodal dataset**.)

Tabular Dataset



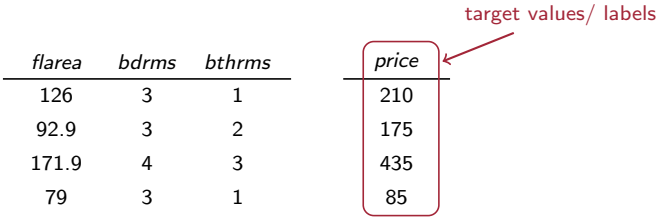
Labeled Dataset

- To *learn* how to make predictions, we need a **labeled dataset**.
- A labeled dataset is a set of **labeled examples**.

Labeled Example

- A **labeled example** is an example accompanied by its corresponding **label** or **target value**, i.e. the *actual* value we are trying to predict.
- E.g. a description of a house, but also its actual selling price.

Example Labeled Dataset



Notation

- We will refer to the two parts of this labeled dataset as $\boldsymbol{X}$ (uppercase, bold) and $\boldsymbol{y}$ (lowercase, bold).
- We will refer to an example (row) as $\boldsymbol{x}$ (lowercase, bold) and its corresponding target value/ label as y (lowercase, not bold).

Supervised Learning

- Given a labeled dataset;
- Learn to make predictions;
- Guided by the labels.

Model

In Supervised Learning, a **model** is a function (a formula, a set of rules, a procedure,...) that expresses the relationship between an object's features and the thing that is being predicted (the target value/label).

Training vs. Inference

- **Training**: Generalize from the training examples — learn a function (model) that captures the relationship between features and targets.
- **Inference**: Apply the learned function (model) to unseen examples — to make predictions.
- Training is also called **fitting a model** and inference is also called **prediction**.
- In `scikit-learn`, we have methods `fit` and `predict`.

Error Estimation

- We've learnt a model from a dataset.
- We want to know how well it will do in practice, once we start to use it for inference — to make predictions.
- This is called **error estimation**.
- We need one or more **evaluation metrics**.

Evaluation Metrics for Regression

Notation

- $\boldsymbol{X}$ is a dataset of examples.
- $\boldsymbol{x}$ is an example, $\boldsymbol{x} \in \boldsymbol{X}$.
- y is the *actual* target value for $\boldsymbol{x}$.
- $\hat{y}$ is the *predicted* target value for $\boldsymbol{x}$.
- $\overline{\boldsymbol{y}}$ is the mean of the actual target values.
- For regression, we measure the **error** (difference) between them: $\hat{y} - y$
- But we need to take the absolute value or square the error. Why?
- Why do some people prefer to square?
- We do this for each example in $\boldsymbol{X}$ and take the mean.

Mean absolute error (MAE)

$$MAE = \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}, y \in \boldsymbol{y}} \text{abs}(\hat{y} - y)$$

Mean squared error (MSE)

$$MSE = \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}, y \in \boldsymbol{y}} (\hat{y} - y)^2$$

Root mean squared error (RMSE)

$$RMSE = \sqrt{\frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}, y \in \boldsymbol{y}} (\hat{y} - y)^2}$$

Coefficient of determination (R^2-score)

$$R^2\text{-}score = 1 - \frac{\sum_{\boldsymbol{x} \in \boldsymbol{X}, y \in \boldsymbol{y}} (\hat{y} - y)^2}{\sum_{\boldsymbol{x} \in \boldsymbol{X}, y \in \boldsymbol{y}} (\overline{\boldsymbol{y}} - y)^2}$$

Holdout

- **Holdout** is a method for *parititioning* a dataset:
  - Shuffle the dataset — to ensure a random split.
  - Partition the dataset into two:
    - **training set** (e.g. the first 80% of the full dataset);
    - **test set** (the rest of the full dataset).
  - Train the model on the training set.
  - Test the model on the test set.
- This is called **holdout**, because the test set is withheld (held-out) during training.
- We need the training and test sets to be representative of the full dataset — otherwise, informally speaking, you might get 'lucky' or 'unlucky'.
- Holdout ideally needs a large dataset. With a small dataset, there is a greater chance that they will be unrepresentative.

Training Error and Test Error

- We learn a model by training on the training set (of course).
- **Training error**: This is where we evaluate also on the training set.
- **Test error**: This is where we evaluate on the test set.
- Generally, we are more interested in the test error — this shows how the model performs on **unseen data** and therefore how it is likely to perform when we deploy it.

Leakage

- It is essential that the test set is not used in any way to create the model. *Don't even look at it!*
- 'Cheating' is called **leakage**.
- Leakage will mean that you will probably underestimate the error of your model when it faces unseen data.
- The most obvious example of leakage is using the same data for training and testing.
- But watch out for more subtle examples too!

Underfitting and Overfitting

- **Underfitting**: the model is not complex enough.
- **Overfitting**: the model is too complex.

3. Binary Classification

The data we use here was collected by me when I was teaching a module (CS1109) between 2008 and 2014.

Classification

- **Classifiers** predict an object's **class** from a finite set of classes.
- E.g. given an email, predict whether the email is spam or ham.
- E.g. given a photo, predict whether it depicts a cat, dog or rabbit.
- (Compare with **regression**.)

Binary Classification

- In **binary classification**, there are just two classes.
- E.g. fail/pass, ham/spam, benign/malignant.
- (Contrast with **multiclass classification**.)

Positive/Negative

- In binary classification, it is common to refer to one class as the **negative class** and the other the **positive class**.
- It doesn't really matter which is which. But, usually, we treat the class that requires special action as the positive class.
- E.g. in spam filtering, ham is the negative class; spam is the positive class.
- What about tumour classification?
- This terminology is extended to other things too, e.g. we can refer to **negative examples** and **positive examples**.

Class exercise

- Consider:
  - Predicting tomorrow's rainfall.
  - Predicting whether we will have a white Christmas.
  - Predicting the sentiment of a tweet (negative, neutral or positive).
  - Predicting a person's sexual orientation.
  - Predicting a person's opinion of a movie on a rating scale of 1 star (rotten) to 5 stars (fab).
- Answer the following:
  - Which are regression and which classification?
  - If classification, which are binary and which are multiclass?
  - If binary, which is the positive class and which the negative?

Classes and Class Labels

- We assume we have a finite set of **labels**, $\mathcal{C}$, one per class.
- Given an object $\boldsymbol{x}$, a classifier will assign one of the labels $\hat{y} \in \mathcal{C}$ to the object.
- Software libraries often use integers for the labels (**label encoding**).
- E.g. given an email, a spam filter predicts $\hat{y} \in \{0, 1\}$, where 0 means ham and 1 means spam.
- But a classifier should not treat these as continuous, e.g. it should never output 0.5.
- Furthermore, where there are more than two labels, we should not assume a relationship between the labels.

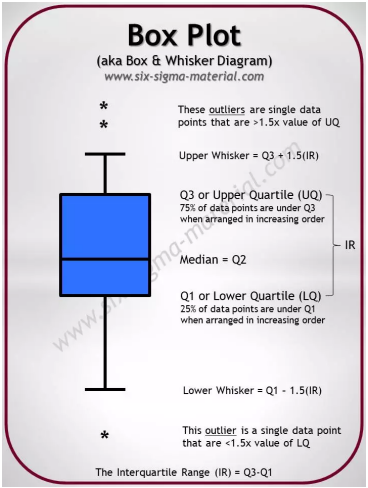
Question

- Suppose there are three classes $\{1, 2, 3\}$.
- Suppose we are classifying object $\boldsymbol{x}$ and we happen to know that its actual class label $y = 3$.
- One classifier predicts $\hat{y} = 1$.
- Another classifier predicts $\hat{y} = 2$.
- Which classifier has done better?

Stratified Holdout

- Splitting a dataset using holdout might result in partitions that do not reflect the distribution of examples within the classes:
  - Examples of one class might be under-represented in the training set or test set.
  - Examples of one class might even be completely absent from the training set or test set.
- **Stratified holdout**: the proportion of examples of each class in the overall dataset is respected in the partitioning into training and test sets.
- Although this fixes the distribution with respect to the classes, you may still get 'lucky' or 'unlucky' in other ways. So we will still want a large dataset for holdout.

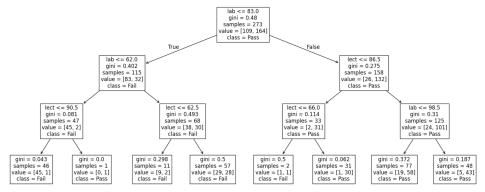
How to Read a Box-and-Whisker Plot



Decision Trees for Classification

A **Decision Tree** is a model, comprising a hierarchy of simple rules.

Inference: start at the root and keep going left or right, as appropriate.



- `gini`: is the **Gini impurity** of a node (with values in $[0.0, 0.5]$).
- `samples`: how many training examples the node applies to.
- `value`: how many training examples of each class this node applies to.
- `class`: the prediction if we stop at this node (the majority class of the training examples that this node applies to)

Gini Impurity

Assume binary classification for simplicity: classes $\mathcal{C} = \{c, c'\}$.

Then at a particular node in the tree,

$$Gini = 1 - (Prob(c)^2 + Prob(c')^2)$$

- Gini measures how often an example would be incorrectly labeled if it was randomly labeled according to the distribution of labels for this node.
- A Gini of 0 means no impurity: all examples that this node applies to belong to the same class.
- Values closer to 0.5 imply a lot more impurity.
- (If you're Googling this, make sure you're reading about Gini Impurity or the Gini index, not the Gini coefficient!).

Learning a Decision Tree

The CART Algorithm (Classification and Regression Tree)

- CART can learn trees for classification and for regression.
- For regression, it uses MSE in place of Gini.
- For classification, it can use entropy in place of Gini — but it doesn't usually make much difference!
- In the case of classification, it can learn trees for binary classification ($|\mathcal{C}| = 2$) and multiclass classification ($|\mathcal{C}| > 2$).
- It only produces binary trees (hence yes/no questions in non-leaf nodes).
- It can handle both numeric-valued and nominal-valued features and even missing values — although this may not be true of the scikit-learn implementation.
- It is recursive.
- It is greedy.

The CART Algorithm

for *each feature* x_i *and each value* v, **do**
split the dataset into two;

X_{left} are the examples in X for which $x_i \leq v$;

X_{right} are the examples in X for which $x_i > v$;

Then calculate the CART score: $\frac{|X_{left}|}{|X|}\text{Gini}(X_{left}) + \frac{|X_{right}|}{|X|}\text{Gini}(X_{right})$;

end

From the above, choose the feature x_i and value v with lowest score;

if *a stopping criterion has been reached (e.g. maximum depth or if no split reduces impurity)* **then**
return;

end

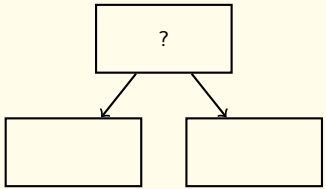
else
Recursively call CART on X_{left};

Recursively call CART on X_{right};

end

Example

What will go in the root?



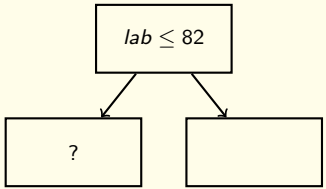
For each feature and each value, we compute the CART score.

$lect \leq 1$	0.477
$lect \leq 2$	0.472
$lect \leq 3$	0.469
$lect \leq 4$	0.469
$lect \leq 5$	0.466
⋮	⋮
$lab \leq 1$	0.472
$lab \leq 2$	0.472
$lab \leq 3$	0.472
$lab \leq 4$	0.472
$lab \leq 5$	0.469
⋮	⋮

The lowest CART score is $lab \leq 82 : 0.328$.

Example

What will go in the left-hand child?



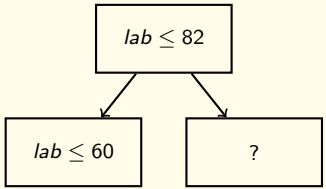
For each feature and each value, we compute the CART score.

$lect \leq 1$	0.4
$lect \leq 2$	0.398
$lect \leq 3$	0.396
$lect \leq 4$	0.396
$lect \leq 5$	0.395
⋮	⋮
$lab \leq 1$	0.398
$lab \leq 2$	0.398
$lab \leq 3$	0.398
$lab \leq 4$	0.398
$lab \leq 5$	0.396
⋮	⋮

The lowest CART score is $lab \leq 60 : 0.325$.

Example

What will go in the right-hand child?



For each feature and each value, we compute the CART score.

$lect \leq 57$	0.275
$lect \leq 58$	0.275
$lect \leq 59$	0.275
$lect \leq 60$	0.275
$lect \leq 61$	0.275
⋮	⋮
$lab \leq 84$	0.275
$lab \leq 85$	0.275
$lab \leq 86$	0.275
$lab \leq 87$	0.275
$lab \leq 88$	0.275
⋮	⋮

The lowest CART score is $lect \leq 84 : 0.275$.

A Variation of Classification

- Given an object $\boldsymbol{x}$, a classifier outputs a label, $\hat{y} \in \mathcal{C}$.
- Instead, a classifier could output a probability distribution over the labels $\mathcal{C}$.
- E.g. given an email $\boldsymbol{x}$, a spam filter might output $\langle 0.2, 0.8 \rangle$ meaning the probability that $\boldsymbol{x}$ is *ham* is 0.2, and the probability that it is *spam* is 0.8.
- The probabilities must sum to 1.

Questions

- In fact, a binary classifier can output just one probability, rather than two. Why?
- How do you think a Decision Tree calculates these probabilities?

Explainable Artificial Intelligence (XAI)

Driven mostly by ethical and legal concerns, **XAI** refers to research into making models and their predictions more understandable by stakeholders such as end-users, domain experts, developers, managers and auditors.

Interpretability

A model is **interpretable** to the extent that humans can gain insight into what has been learned.

Explainability

A prediction is **explainable** to the extent that humans can understand how or why a particular outcome has been predicted by a model.

Error Estimation for Classifiers

The main evaluation metric is **accuracy** (ACC).

(If you want an error estimate, then compute $1 - ACC$.)

Accuracy (ACC)

$ACC = \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}, y \in \boldsymbol{y}} int(\hat{y} = y)$

Alternatives to Accuracy (for Private Study) — Binary Classification

Confusion Matrix

		Predicted Class: Negative	Predicted Class: Positive
Actual Class	**Negative**	True Negatives (TN)	False Positives (FP)
	Positive	False Negatives (FN)	True Positives (TP)

Alternatives to Accuracy (for Private Study) — Binary Classification

TNR, FPR, FNR, TPR, NPV, FDR, FOR, PPV

		Predicted Class: Negative	Predicted Class: Positive		
Actual Class	**Negative**	True Negatives (TN)	False Positives (FP)	*True Negative Rate* $TNR = \frac{TN}{TN+FP}$	*False Positive Rate* $FPR = \frac{FP}{TN+FP}$
	Positive	False Negatives (FN)	True Positives (TP)	*False Negative Rate* $FNR = \frac{FN}{FN+TP}$	*True Positive Rate* $TPR = \frac{TP}{FN+TP}$

Negative Predicted Value $NPV = \frac{TN}{TN+FN}$	*False Discovery Rate* $FDR = \frac{FP}{FP+TP}$
False Omission Rate $FOR = \frac{FN}{TN+FN}$	*Positive Prediction Value* $PPV = \frac{TP}{FP+TP}$

Alternatives to Accuracy (for Private Study) — Binary Classification

		Predicted Class: Negative	Predicted Class: Positive
Actual Class	Negative	True Negatives (TN)	False Positives (FP)
	Positive	False Negatives (FN)	True Positives (TP)

Accuracy

$$ACC = \frac{TP + TN}{FP + FN + TP + TN}$$

Precision and Recall

$$Precision = \frac{TP}{FP + TP} = PPV$$

$$Recall = \frac{TP}{FN + TP} = TPR$$

F1 score

$$F1 = 2\frac{Prec \times Rec}{Prec + Rec}$$

Alternatives to Accuracy (for Private Study) — Multiclass Classification

Suppose $|\mathcal{C}| > 2$. Take each class $c \in \mathcal{C}$. Treat c as the positive class and all other classes as the negative class. Then we can calculate TP_c — the True Positive for class c, and similarly TN_c, FP_c, FN_c.

Micro-Averaged Precision

$$PRE_{micro} = \frac{\sum_{c \in \mathcal{C}} TP_c}{\sum_{c \in \mathcal{C}} FP_c + \sum_{c \in \mathcal{C}} TP_c}$$

Macro-Averaged Precision

$$PRE_{macro} = \frac{\sum_{c \in \mathcal{C}} PRE_c}{|\mathcal{C}|}$$

Hyperparameter

The maximum depth of a Decision Tree is an example of a **hyperparameter** — an input to the training algorithm.

Underfitting and Overfitting

- **Underfitting**: the model is not complex enough.
- **Overfitting**: the model is too complex.

The maximum depth is one factor influencing whether a Decision Tree underfits or overfits.

Image credits

	Photo from myirelandtour.com
	Taken from https://www.six-sigma-material.com

4. Regression

The data we use was collected by Dean De Cock in 2011 and made available on the web site of the Journal of Statistics Education, now more easily available on Kaggle. Each of the 2929 examples is a residential property in Ames, Iowa with around 80 features. The idea to use it (and to use a subset of it) comes from Sebastian Raschka: Machine Learning with PyTorch and Scikit-Learn, Packt Publishing, 2022.

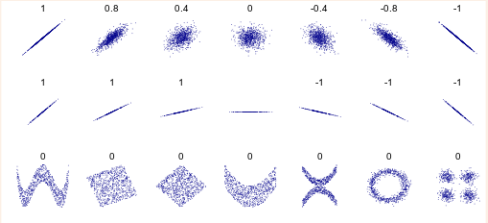
Correlation

- **Correlation coefficients** measure the strength and direction of the relationship between two variables (e.g. two features or a feature and the target)
- E.g. ice-cream sales and temperature are positively correlated.
- E.g. altitude and temperature are negatively correlated.

Pearson correlation coefficient

$$r = \frac{\sum_{i=1}^{m}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{m}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{m}(y_i - \overline{y})^2}}$$

where m is the number of values, x_i & y_i are individual values, and $\overline{x}$ & $\overline{y}$ are their means. r will lie between -1 and 1.



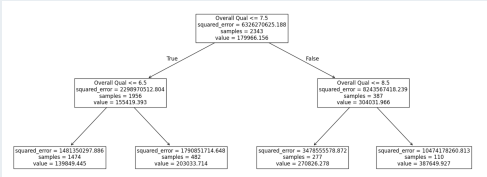
Some Uses of Correlation Coefficients in ML

- A feature that is not predictive of the target may be a candidate for removal.
- A feature that is strongly correlated with another (multicolinearity) may be a candidate for removal.
- Some Machine Learning algorithms assume that features are not correlated. If they are correlated, then either we should not use that algorithm or we should use dimensionality reduction techniques.
- In some cases, multicolinearity can reduce interpretability.
- If a feature is correlated with a sensitive feature (e.g. sex, gender, race, ...), then your model may be biased (even if it does not use the sensitive feature).

Warnings

- "Correlation does not imply causation." E.g. ice-cream sales and sunstroke are positively correlated.
- Correlation could be due to chance — especially in small datasets.
- Pearson correlation measures whether the two variables are linearly related — do they change at a constant rate. But their relationship might be curvilinear.

Decision Tree Regression



- `squared_error`: is the *MSE* of a node — this is used in place of Gini.
- `samples`: how many training examples the node applies to.
- `value`: the prediction if we stop at this node (the mean target value of the training examples that this node applies to)

Similarity & Distance

- In AI, we often want to know how *similar* one object is to another.
  - E.g. how similar is my house to yours?
  - E.g. which house in our dataset is most similar to yours?
- In fact, here we are instead going to measure how *different* they are using a **distance function**.

This is not about geographical distance.

Euclidean Distance

- Let $\boldsymbol{x}$ and $\boldsymbol{x}'$ be two examples with n features.

$$d(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{(\boldsymbol{x}_1 - \boldsymbol{x}'_1)^2 + (\boldsymbol{x}_2 - \boldsymbol{x}'_2)^2 + \cdots + (\boldsymbol{x}_n - \boldsymbol{x}'_n)^2}$$
$$= \sqrt{\sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{x}'_i)^2}$$

- Its minimum value is 0 (meaning identical) but there's no maximum value (depends on your data).
- Class exercise. What is the Euclidean distance between $\boldsymbol{x} = \begin{bmatrix} 100 \\ 1 \\ 4 \end{bmatrix}$ and $\boldsymbol{x}' = \begin{bmatrix} 100 \\ 5 \\ 1 \end{bmatrix}$?

Nearest-Neighbours Regressor

- To predict the target value $\hat{y}$ for unseen example $\boldsymbol{x}'$,
  - we find the example $\boldsymbol{x}$ in the labeled training set whose distance from $\boldsymbol{x}'$ is smallest;
  - we use the y-value for $\boldsymbol{x}$ as our prediction.
- We also refer to this as a **1-nearest-neighbour regressor** or **1NN** or **kNN for** $k = 1$.
- It has the problem that it can be incorrectly influenced by noisy examples (incorrect feature-values or incorrect labels).

k-Nearest-Neighbours Regressor

- To predict the target value $\hat{y}$ for unseen example $\boldsymbol{x}'$,
  - we find the k examples in the labeled training set whose distance from $\boldsymbol{x}'$ is smallest;
  - we use the *mean* of their y-values as our prediction.
- We abbreviate the name of this to **kNN**, e.g. 3NN.
- It is less influenced by noisy examples.
- k is a **hyperparameter** and we need a way of choosing its value — see later.

Question

- k is a **hyperparameter** – we'll soon learn how to choose a good value.
- Why are we unlikely to use $k = m$, where m is the number of examples in the training set?

Question

- One variant of kNN is **weighted kNN**, where we compute a distance-weighted mean.
- Now we could reasonably use $k = m$. Why? (Doing so is sometimes called Shepard's method.)

Questions

- Which is more likely to result in **overfitting**? Small k or large k?
- Hence, which is more likely to result in **underfitting**?

Questions

- **Explainability**: how would we explain a kNN prediction to someone?
- **Interpretability**: Not applicable to kNN. Why?

Questions

- How would we modify the algorithm to make a **kNN classifier?**
- Some classifiers output probabilities, one per class. How would we modify the algorithm to make a kNN classifier that outputs these **class probabilities**?

Problems with Euclidean distance

Problem

Features with different scales.

Solution

Scaling

Problem

Features that are correlated with each other.

Problem

The curse of dimensionality, where $n \gg m$.

Solution

Dimensionality reduction (e.g. Principal Components Analysis)

(These are problems with many other distance measures too.)

Scaling Numeric Values

- Different numeric-valued features often have very different ranges.
  - E.g. floor area ranges from a few tens to a few hundreds of square metres;
  - But the number of bedrooms and bathrooms ranges from 0 to a dozen or so.
- When computing the Euclidean distance, features with large ranges will dominate the distance calculations, thus giving features with small ranges negligible influence.
- E.g., consider your house $\boldsymbol{x} = \begin{bmatrix} 126 \\ 3 \\ 1 \end{bmatrix}$ and two others, $\boldsymbol{x}' = \begin{bmatrix} 131 \\ 3 \\ 1 \end{bmatrix}$ and $\boldsymbol{x}'' = \begin{bmatrix} 126 \\ 7 \\ 1 \end{bmatrix}$

  Intuitively, which is most similar to yours? According to Euclidean distance, which is most similar to yours?

Scaling

Scaling makes features more comparable.

Min-max scaling

$$x \leftarrow \frac{x - min}{max - min}$$

Robust scaling

$$x \leftarrow \frac{x - median}{inter\text{-}quartile\text{-}range}$$

Standardization

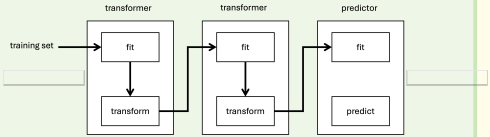
$$x \leftarrow \frac{x - mean}{standard\text{-}deviation}$$

Question

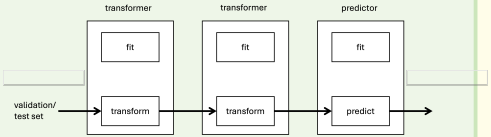
Consider a feature's *min*, *max*, *median*, *inter-quartile-range*, *mean* and *standard-deviation* when used for scaling.
Are these computed on (a) the whole dataset, (b) the training set and separately also the test set, or (c) just the training set?

scikit-learn pipelines

Training (`fit`)



Inference (`predict`)



Two types of Supervised Learning

Supervised Learning

↙ ↘

Instance-Based Learning
e.g. kNN

Model-Based Learning
e.g. Decision Trees

Instance-Based Learning and Model-Based Learning

Differ in *when* they generalize

	Instance-Based	**Model-Based**
Train (fit)	Memorize the training examples	Generalize from the training examples – learn a function that captures the relationship between features and targets
Infer (predict)	Generalize from the training examples — use distance from training examples to predict unseen example's target value	Apply the learned function to the unseen example

Compare and Contrast

Compare model-based learning and instance-based learning.
Think about training time, inference time, incrementality and re-training.

Speeding-up kNN inference

There are data structures that support faster retrieval of neighbours (e.g. kd-trees and ball-trees).
During training (fit), we could insert the training examples into one of these data structures.

Image credits

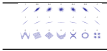


Image by Denis Boigelot, taken from Wikipedia

5. Model Selection

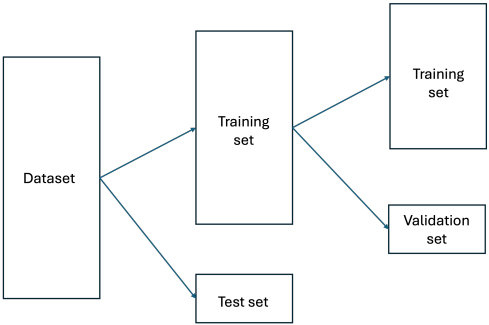
In this lecture, we use the same housing data that we used in the previous lecture.

Model Selection

- **Model Selection** involves evaluating multiple algorithms and hyperparameter configurations to identify the best-performing model(s) for your data.
- If you (like many people) use the test set for model selection, then information about the test set is being used to develop the model — **leakage**! Your final error estimate will likely be over optimistic.
- We need another test set — one that we use during model selection. This is called the **validation set**.

Model Selection using Holdout

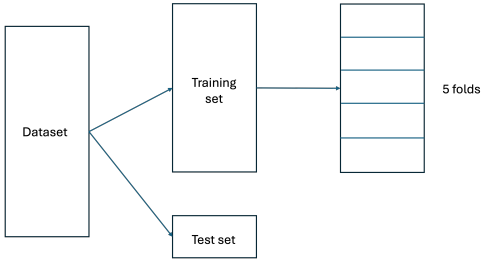
- The simplest approach is to shuffle then randomly partition the dataset into three, e.g. 60%-20%-20%:
  - training set;
  - validation set;
  - test set.
- Train the various configurations on the training set.
- Test them on the validation set.
- Choose the configuration with the lowest validation error.
- Train the winning configuration on the union of the training set and validation set.
- Test this model on the test set.



- This method requires an even bigger dataset: one we can split into three large enough partitions!
- When we have a smaller dataset, we use a **resampling** method, where the examples get used and re-used for training and validation.

Model Selection using k-Fold Cross-Validation

- A common resampling method is k-**fold cross-validation**.
- Shuffle, then randomly partition the dataset into two: training set and test set.
- Then partition the training set into k equal-sized partitions.
- For $i \in [1, \ldots, k]$:
  - Train the various configurations on all folds except fold k.
  - Test them on fold k, acting as the validation set.
- Choose the configuration with the lowest *mean* validation error.
- Train the winning configuration on the original training set.
- Test this model on the test set.



Holdout

Advantage

- The validation error is independent of the training set.

Disadvantages

- Results can vary quite a lot across different runs. Informally, you might get lucky — or unlucky. In other words, in any one split, the data used for training or testing might not be representative.
- We are training on only a subset of the available dataset, perhaps as little as 60% of it. From so little data, we may learn a worse model and so our error measurement may be pessimistic.

k-Fold CV

Advantage

- The validation errors of the folds are independent — because examples are included in only one validation set.
- Better use is made of the dataset: for $k = 10$, for example, we train using $9/10$ of the dataset.
- You now have k validation errors. So, not only can you compute their mean, you could also compute their standard deviation, or you could even use tests for statistical significance when comparing different models.

Disadvantages

- While the validation sets are independent of each other, the training sets are not. They will overlap with each other to some degree.
- The number of folds is constrained by the size of the dataset and the desire sometimes on the part of statisticians to have folds of at least 30 examples.
- It can be costly to train the learning algorithm k times.
- There may still be some variability in the results due to 'lucky'/'unlucky' splits.

Important

- In the past, students have tried holdout and k-fold CV as if they were in competition with each other. This betrays a misunderstanding. You do not try them both and see which one gives the lower error. You pick one of them — the one that makes most sense for your data — and use it.
- In practice, we only use the holdout method when we have a very large dataset, e.g. 10s of 1000s. The size of the dataset mitigates the disadvantages.
- We use k-fold CV when we have a smaller dataset, e.g. several 100s.
- (There are some alternatives, e.g. repeated holdout, leave-one-out CV,...)
- Whichever you choose (holdout of k-fold CV), remember to **stratify** for classification.

Grid Search

- Suppose you have multiple hyperparameters or other ways of configuring an algorithm.
- **Grid Search** will try all combinations.

Randomized Search

- When the number of combinations of hyperparameter values/configurations is high, Grid Search's exhaustive approach may take too long.
- We can instead use **Randomized Search**, which tries a certain number of them, chosen at random.

There are yet further alternatives to these two, e.g. Halving Grid Search. And there are advanced Bayesian methods.

6. Preprocessing

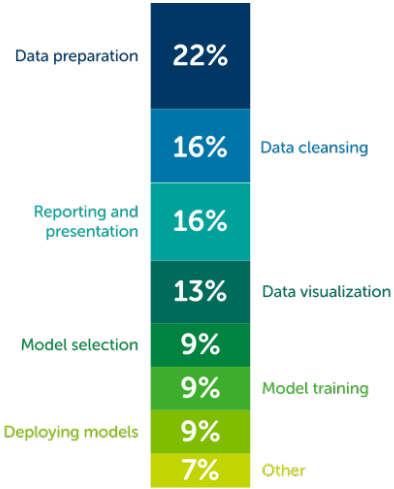
In this lecture, we predict whether to approve a loan or not. We use a slightly modified version of a dataset from Kaggle.

Machine Learning Projects (simplified!)

1. Understand the business problem.
2. Select performance measures.
3. Acquire a dataset.
4. Take a cheeky look.
5. Cleanup anything that is simply invalid.
6. Split into training and test sets using holdout.
7. Exploratory Data Analysis.
8. Feature Engineering.
9. Preprocess the data.
10. Model Selection.
11. Repeat steps 7–10 until satisfied.
12. Error Estimation: test on the test set.
13. Decide whether to deploy.
14. If yes, then train on the whole dataset.
15. Once deployed, monitor and retrain regularly.

How do data scientists spend their time?

(Survey of $m = 1,966$ respondents.)



Step 1: Understand the business problem.

We are a small company, making loans in a competitive market. We want to speed-up loan decisions and reduce the number of people who default on their loans.

- Notice that the business problem is not expressed in terms of Machine Learning, and ML will likely be just one component of the solution (assuming it gets used at all).
- Assemble a team: Computer Scientist(s), Statistician(s) but also Domain Expert(s).

Step 2: Select performance measures.

- This is Supervised Learning of a Binary Classifier.
- We will measure accuracy.
- We will compare with a **majority-class classifier** (a **dummy classifier**).

We might want a 'reference point' to know whether we have succeeded.

- E.g. maybe we want accuracy to reach some threshold, or to be some amount higher than human performance at this task or an existing computer system or a dummy classifier.
- E.g. maybe we must not only be better than current performance but **statistically significantly** better.

Sometimes there is more than one performance measure.

- E.g. maybe in regression we also want to measure overestimates separately from underestimates; in binary classification maybe we also want to measure false positives and false negatives.
- E.g. maybe we want to measure time/memory for training.
- E.g. maybe we want to measure time/memory for inference.
- E.g. maybe we want an incremental learning algorithm that can easily accommodate new training examples when they arise.
- E.g. maybe we care about learning an interpretable model or being able to explain the system's predictions.

Step 3: Acquire a dataset.

Dataset acquisition is more easily said than done — especially since we need a *labeled dataset*. In the real world, datasets don't come from your lecturer or from Kaggle. More on this in a later lecture.

Step 4: Take a cheeky look.

Enough to get an understanding of the kind of data but not too much — avoid leakage.

Step 5: Cleanup anything that is simply invalid.

Historical datasets may contain values that your domain expert says are invalid. Occasionally, your domain expert may be able to correct the invalid values. More usually, we delete rows and perhaps columns to get rid of these.

- E.g. if a feature value is required but there are examples where it is missing.
- E.g. if there is a maximum value for a feature but examples where this maximum is exceeded.
- E.g. if there are examples where a nominal-valued (non-numeric) feature has a value that falls outside of the finite set of allowed values.
- E.g. if there are examples where the target value is missing.
- E.g. if there are duplicate examples – these may or may not be invalid.

Only do this to data that you will henceforth disallow.

Step 6: Split into training and test sets using holdout.

Stratified holdout in this case!

Step 7: Exploratory Data Analysis.

- Best to perform EDA and try Feature Engineering on a copy of the training set.
- Visualizations: histograms, scatter plots, strip plots, box plots, ...
- Statistics including correlation coefficients.
- Some people drop features at this point, e.g. where there is multicolinearity or where features are not predictive of the target. But be wary of this.

Step 8: Feature Engineering.

In **feature engineering**, we invent new features that are computed from the existing features — hopefully, ones that are predictive of the target.

- E.g. products or ratios of existing features.
- E.g. the result of applying functions to existing features, such as squaring, taking the square root, taking the log, ...

We would then do some more EDA to see whether the new features are promising or not.

Step 9: Preprocess the data.

Among other things, we must consider:

Outliers

See next slides.

Missing values

See next slides.

Scaling numeric-features

See previous lecture.

Converting non-numeric features to numeric

See next slides.

Feature selection/dimensionality reduction

See later lecture.

Create a pipeline of scikit-learn transformers that puts all your preprocessing code in one place.

Also insert any promising features from Feature Engineering.

Outliers

- Earlier, we deleted examples that had extreme values which our domain expert regarded as invalid.
- But we may still have **outliers**. If used in training, they may skew what gets learned — not so much Decision Trees, but many other models.

Decide which are outliers

- Speak with the domain expert.
- Some people use the first and third quartiles (Q1 and Q2). Anything below 1.5 x Q1 or above 1.5 x Q2 is an outlier. But (a) this looks at features individually, instead of together, and (b) it may result in too many outliers.
- There are ML methods —often using unsupervised learning– for identifying outliers/anomalies, e.g. Isolation Forests.

Decide what to do about them

- We can do nothing — leave them.
- We can delete examples that have outlier values.
- We can clip the values to some maximum if they're too big or minimum if they're too small.
- We can apply some mathematical function (e.g. log and sqrt will reduce the effect of large values).

It is hard to say whether these solutions should be applied only to the training set or also applied to validation/test sets.

Missing values

- Most Machine Learning algorithms cannot handle **missing values**.
- We may have removed some of them during data cleaning in discussion with our domain expert. But some may remain.

Detecting missing values

- They may appear in a dataset as NaN or None. (E.g. when missing from a csv file, Pandas reads them in as NaN.)
- But they may appear as zero or as "?" or "N/A" or "unknown" or...

What to do about them

- We can delete examples that have missing values, or we can delete features if they are plagued by missing values.
- Or we can **impute** a value:
  - We can replace missing values by the mean (for numeric-valued features), or the mode (for nominal-valued features), or by some constant.
  - Or we could learn a model that predicts the missing value from the other features in the example.
- The mean/mode/etc. must be computed from the training data only.

Converting non-numeric features to numeric

Boolean-valued

- Two values. E.g. Education has values “Graduate” and “Not graduate”.
- Binary encoding: e.g. “Graduate” becomes 0 and “Not Graduate” becomes 1.

Unordered nominal values

- More than two values.
- **Ordinal encoding**: assign integers, e.g. “Semiurban” 0, “Urban” 1, “Rural”’ 2.
- But ML algorithms might assume that the integers themselves are meaningful, when they’re actually arbitrary. E.g. an algorithm might assume that “Semiurban” (0) is more similar to ”Urban” (1) than it is to “Rural” (3).
- Warning: in many datasets, unordered nominal values have already been encoded in this way.
- Often, **one-hot encoding** is preferable.

One-Hot Encoding

- If the original nominal-valued feature has, e.g., 3 values, then we replace the feature by 3 *binary-valued* features.
- In each example, exactly one of the new features is set to 1 and the rest are zero.

Loan_Amount	Loan_Term	Property_Area
71.0	360.0	Rural
40.0	180.0	Rural
253.0	360.0	Urban
187.0	360.0	Urban
133.0	360.0	Semiurban

becomes

Loan_Amount	Loan_Term	Semiurban	Rural	Urban
71.0	360.0	0	1	0
40.0	180.0	0	1	0
253.0	360.0	0	0	1
187.0	360.0	0	0	1
133.0	360.0	1	0	0

- One-hot encoding turns each non-numeric feature into multiple binary-valued features. Thus, you can end up with a lot of new features. Maybe you need to do some dimensionality reduction afterwards.

Converting non-numeric features to numeric

Ordered nominal values

- More than two values but on some sort of scale, e.g. Dependents has values "0", "1", "2" and "3+".
- Ordinal encoding: "0" becomes 0, "1" becomes 1, "2" becomes 2 and "3+"' becomes 3.
- Why might it be wrong? What assumption does it make?
- Instead of guessing, we could compare One-Hot Encoding and Ordinal Encoding in our grid search.

Step 10: Model Selection.

- Choose between holdout and k-fold CV.
- Choose candidate models, Decision Trees, kNN,...
- For each model, create grids.
- Run.
- Analyse validation errors; check for underfitting/overfitting.

Step 11: Repeat steps 7–10 until satisfied.

Step 12: Error Estimation — Test on the test set.

Step 13: Decide whether to deploy.

Step 14: If yes, then train on the whole dataset.

Step 15: Once deployed, monitor and retrain regularly.

Image credits

Image from Anaconda 2022 State of Data Science Report, 2022

Photo by The New York Public Library on Unsplash

7. Linear Regression

In this lecture, we use the same housing data that we used in two of the previous lectures.

Parametric learning

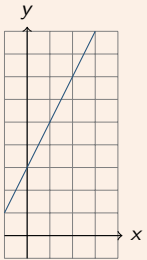
- In **parametric learning**, the model that we will learn has a structure that is known in advance — a function of a particular form.
- But the function contains **parameters** (numeric variables).
- During training, it is the job of the learning algorithm to choose values for the parameters.

Examples

- Parametric learning: Linear Regression, Logistic Regression, most Neural Networks.
- Non-parametric learning: Decision Trees, kNN.

Equation of a Straight Line

- E.g. $y = 3 + 2x$
- Question: From the point of view of plotting this line, what's 3? What's 2?
- In general, $y = b + wx$
- In maths, b and w are called **coefficients**.
- In ML, they are the **parameters**.
- In ML, b might be called the **bias** and w is a **weight**.



Linear Equations

In general, a **linear equation** is of the form:

$$\begin{aligned} y &= b + \boldsymbol{w}_1\boldsymbol{x}_1 + \cdots + \boldsymbol{w}_n\boldsymbol{x}_n \\ &= b + \sum_{i=1}^{n} \boldsymbol{w}_i\boldsymbol{x}_i \\ &= \boldsymbol{wx} \text{ (explained on next two slides)} \end{aligned}$$

Vectors

- In HOML, an n-dimensional **vector** is just an array of n numbers, referred to as the **elements** of the vector.
- We use bold, lowercase letters for vectors, e.g.

$$\boldsymbol{x} = [2, 4, 3]$$

Dot Product

- Suppose we have two n-dimensional vectors, $\boldsymbol{u}$ and $\boldsymbol{v}$.
- Their **dot product**, $\boldsymbol{uv}$, is computed by summing the products of corresponding elements.

$$\begin{aligned}\boldsymbol{uv} &= \boldsymbol{u}_1\boldsymbol{v}_1 + \boldsymbol{u}_2\boldsymbol{v}_2 + \cdots + \boldsymbol{u}_n\boldsymbol{v}_n \\ &= \sum_{i=1}^{n} \boldsymbol{u}_i\boldsymbol{v}_i\end{aligned}$$

Exercise

- Let $\boldsymbol{u} = [2, 4, 3]$ and $\boldsymbol{v} = [5, 0, 1]$.
- Compute $\boldsymbol{uv}$.

Writing a Linear Equation as the Dot Product of Two Vectors

- Suppose we have a **linear equation**

$$y = b + \boldsymbol{w}_1 \boldsymbol{x}_1 + \cdots + \boldsymbol{w}_n \boldsymbol{x}_n$$

- Collect all the $\boldsymbol{x}$ values into an n-dimensional vector:

$$\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]$$

Collect all the weights into an n-dimensional vector:

$$\boldsymbol{w} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n]$$

Then the linear equation can be written as a dot product:

$$y = b + \boldsymbol{w}\boldsymbol{x}$$

- Better, insert a cheeky 1 into vector $\boldsymbol{x}$ (so that it is now $n+1$-dimensional):

$$\boldsymbol{x} = [1, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]$$

Insert b into vector $\boldsymbol{w}$ (so it too is $n+1$-dimensional):

$$\boldsymbol{w} = [b, \boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n]$$

Then the linear equation can be written as this dot product:

$$y = \boldsymbol{w}\boldsymbol{x}$$

Linear Regression with one feature

- If there's just one feature $\boldsymbol{x}_1$, the model we learn will be of the form $\hat{y} = b + \boldsymbol{w}_1 \boldsymbol{x}_1$
- In training, the goal is to find values for the parameters b and $\boldsymbol{w}_1$.
- In other words, we're fitting a *line* to the training data: the line that 'goes through the middle of the data'.

Linear Regression with two features

- If there are two features $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, the model we learn will be of the form $\hat{y} = b + \boldsymbol{w}_1 \boldsymbol{x}_1 + \boldsymbol{w}_2 \boldsymbol{x}_2$
- In training, the goal is to find values for the parameters b, $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$.
- In other words, we're fitting a *plane* to the training data, again 'going through the middle of the data'.

Linear Regression with more than two features

- If there are n features $\boldsymbol{x}_1 \dots \boldsymbol{x}_n$, the model we learn will be of the form $\hat{y} = b + \boldsymbol{w}_1 \boldsymbol{x}_1 + \cdots \boldsymbol{w}_n \boldsymbol{x}_n$
- In training, the goal is to find values for the parameters b, $\boldsymbol{w}_1 \dots \boldsymbol{w}_n$.
- In other words, we're fitting a *hyperplane* to the training data, again 'going through the middle of the data'.

Why Linear Models?

Real-world problems are not usually linear but they might be nearly linear.

- A simple, possibly *interpretable* model (but multicolinearity can spoil this, as can large n unless the model is sparse).
- Training is fast; inference is fast.
- A good baseline: if a fancy model's error is not significantly better, then maybe it's not worth the bother.

- Suppose we have n features. Then there are $n + 1$ parameters. Why?
- For concreteness, assume two features, hence three parameters.
- During training, Linear Regression must find the best values for these parameters. Let's refer to each choice as a **hypothesis**, h.
- E.g. one hypothesis: $b = 800$, $\boldsymbol{w}_1 = -5$, $\boldsymbol{w}_2 = 3.7$.
- E.g. another hypothesis: $b = -10.34$, $\boldsymbol{w}_1 = 0$, $\boldsymbol{w}_2 = 3.1459$
- Which of these is better? Which 'goes through the training data' best?
- We need to *score* the hypotheses so we can pick the one with the best score.
- The score of each hypothesis is computed by a **loss function**.

Loss Function

- The loss function, designated J, takes in a hypothesis h and says how good it is.
- For each example $\boldsymbol{x}$ in the training set, it computes the difference between h's predicted value for $\boldsymbol{x}$ (i.e. $h(\boldsymbol{x})$) and $\boldsymbol{x}$'s actual value, y.
- N.B. Low scores are better!

Ordinary Least-Squares Regression (OLS)

- The most common loss function for regression is the mean squared error (MSE):

$$J(\boldsymbol{X}, \boldsymbol{y}, h) = \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}, y \in \boldsymbol{y}} (h(\boldsymbol{x}) - y)^2$$

  where $\boldsymbol{X}$ is the training set with $\boldsymbol{y}$ its labels.
- Linear Regression using this loss function is referred to as **Ordinary Least-Squares Regression (OLS)**:
  - "least" because we are minimizing the loss;
  - "squares" because the loss is mean squared error;
  - "ordinary" to distinguish it from more exotic variants.
- Question: why are we squaring? (Two answers.)
- In fact, we often divide by 2:

$$J(\boldsymbol{X}, \boldsymbol{y}, h) = \frac{1}{2m} \sum_{\boldsymbol{x} \in \boldsymbol{X}, y \in \boldsymbol{y}} (h(\boldsymbol{x}) - y)^2$$

Why use MSE for the Loss Function?

- MSE is continuously differentiable.
- MSE is **convex**.
  - Informally, this means that it has a unique minimum.
  - When there are two features, loss is bowl-shaped.
  - (To be 100% accurate: there is a unique minimum provided no feature in $\boldsymbol{X}$ is linearly dependent on any of the others or, equivalently, provided $\boldsymbol{X}$ has an inverse.)

Finding the Best Hypothesis

Why does this not work?

for *each possible hypothesis* h **do**
    compute h's loss, $J(\boldsymbol{X}, \boldsymbol{y}, h)$;

end

return the h that had the lowest loss

Normal Equation

Given the training data $\boldsymbol{X}$ and $\boldsymbol{y}$, the **Normal Equation** computes the winning hypothesis:

$$(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

Gradient Descent

start with an initial guess for the values of the parameters;

repeat
    update the guess, changing the parameter values in a way that reduces the loss;

until *convergence*;

Deriving the Normal Equation

- We need the **gradient** of the loss function with regard to each $\boldsymbol{w}_i$:

$$\frac{\partial J(\boldsymbol{X}, \boldsymbol{y}, h)}{\partial \boldsymbol{w}_i}$$

  - In other words, how much the loss will change if we change $\boldsymbol{w}_i$ a little.
  - With respect to a particular $\boldsymbol{w}_i$, it is called the **partial derivative**.
  - The **gradient vector** contains each of these partial derivatives.

$$\nabla J(\boldsymbol{X}, \boldsymbol{y}, h) = \begin{bmatrix} \frac{\partial J(\boldsymbol{X}, \boldsymbol{y}, h)}{\partial b} \\ \frac{\partial J(\boldsymbol{X}, \boldsymbol{y}, h)}{\partial \boldsymbol{w}_1} \\ \vdots \\ \frac{\partial J(\boldsymbol{X}, \boldsymbol{y}, h)}{\partial \boldsymbol{w}_n} \end{bmatrix} = \begin{bmatrix} \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}} (b - y) \\ \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}} (\boldsymbol{w}_1 \boldsymbol{x}_1 - y) \times \boldsymbol{x}_1 \\ \vdots \\ \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}} (\boldsymbol{w}_n \boldsymbol{x}_n - y) \times \boldsymbol{x}_n \end{bmatrix}$$

- Set $\nabla J(\boldsymbol{X}, \boldsymbol{y}, h) = \boldsymbol{0}$.
- Do some algebraic manipulation to get $b, w_1 \dots w_n$ on the left-hand side, and we get

$$\boldsymbol{w} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

8. Gradient Descent

In this lecture, we use the same housing data that we used in several of the previous lectures.

Gradient Descent

- **Gradient Descent** is a *generic* method for finding optimal solutions to problems that involve minimizing a loss function.
- It is a search in the model's parameter space for values of the parameters that minimize the loss function.

> start with an initial guess for the values of the parameters;
>
> **repeat**
>
> update the guess, changing the parameter values in a way that reduces the loss;
>
> **until** *convergence*;

- Ideally, it keeps searching until convergence — changes to the parameter values do not result in lower loss.
- The key to this algorithm is how to update the parameter values.

How to Update the Parameters

- To update the parameter values to reduce the loss:
  - Compute the **gradient vector**, $\nabla J(\boldsymbol{X}, \boldsymbol{y}, h)$.
    - But this points 'uphill' and we want to go 'downhill'.
    - And we want to make 'baby steps', so we use a **learning rate**, α, which is between 0 and 1.
  - So subtract α times the gradient vector from vector $\boldsymbol{w}$:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla J(\boldsymbol{X}, \boldsymbol{y}, h)$$

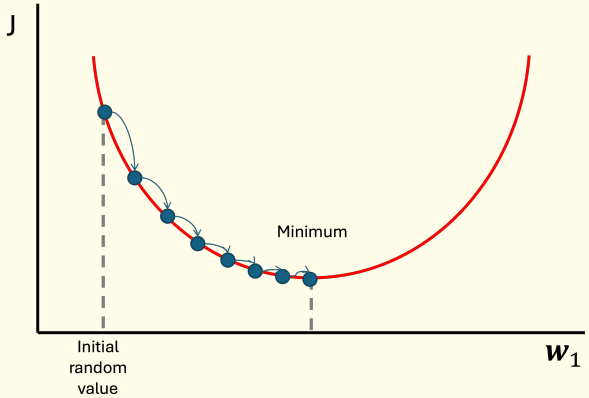
  Or

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{\alpha}{m} \boldsymbol{X}^T (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$$

- (Note how this simultaneously updates the parameters, $\boldsymbol{w}$.)

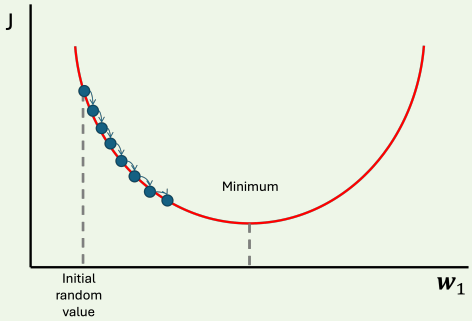
Baby Steps

- We'll use an example with a single feature/single parameter $\boldsymbol{w}_1$ in order to visualise.
- We update $\boldsymbol{w}_1$ gradually, one baby step at a time, until the algorithm converges on minimum loss.

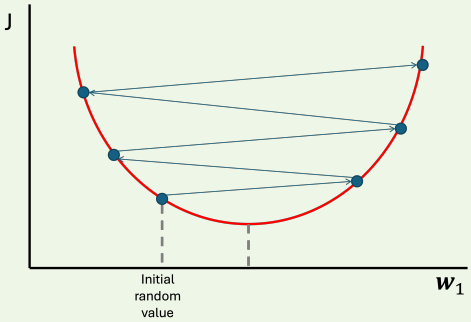


- The size of the steps is determined by the learning rate.

- If the learning rate is too small, it will take many updates until convergence:



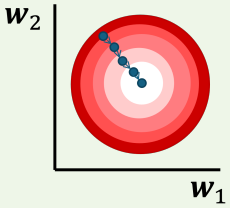
- If the learning rate is too big, the algorithm might jump across the valley — it may even end up with higher loss than before, making the next step bigger.
- This might make the algorithm **diverge**:



Scaling for Gradient Descent

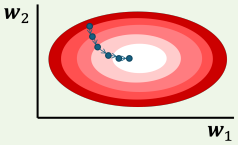
- If we are doing OLS regression using the Normal Equation, we do not need to scale the features.
- But if we are using Gradient Descent, we do need to scale the features.

- E.g. features 1 and 2 have similar ranges of values — a 'bowl':



- The algorithm goes straight towards the minimum.

- E.g. feature 1 has smaller values than feature 2 — an elongated 'bowl':



- Since feature 1 has smaller values, it takes a larger change in w_1 to affect the loss function, which is why it is elongated.
- It takes more steps to get to the minimum — steeply down but not really towards the goal, followed by a long march down a nearly flat valley.
- It makes it more difficult to choose a value for the learning rate that avoids divergence: a value that suits one feature may not suit another.

Batch Gradient Descent

In **Batch Gradient Descent**, the updates involve a calculation over the entire training set $\boldsymbol{X}$ on every iteration.

initialize $\boldsymbol{w}$ randomly;

repeat

$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{\alpha}{m}\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$;

until *convergence*;

- The algorithm **converges** when the parameters do not change between iterations. Equivalently, it converges when the loss doesn't change. In practice, we introduce a hyperparameter: the maximum. number of iterations.

Question

- Some people suggest a variant of Batch Gradient Descent in which the value of α is decreased over time, i.e. its value in later iterations is smaller.
- Why do they suggest this?
- And why isn't it necessary?

- Batch Gradient Descent is more suitable for smaller training sets.
- If the problem is convex, BGD is guaranteed to find the minimum (assuming the learning rate is small enough and the maximum number of iterations is high enough).

Stochastic Gradient Descent

In **Stochastic Gradient Descent**, the updates involve a calculation using a single randomly-selected training example $\boldsymbol{x}$.

initialize $\boldsymbol{w}$ randomly;

for *each epoch* **do**
  shuffle the training set;
  for *each example* $\boldsymbol{x}$ **do**
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \boldsymbol{x}(\boldsymbol{w}\boldsymbol{x} - y)$;
  end
end

- Stochastic Gradient Descent works even on huge datasets since only one example needs to be in memory in each iteration.
- But, because it is stochastic, the loss will not necessarily decrease on each iteration:
  - *On average*, the loss decreases, but in any one iteration, loss may go up or down.
  - Eventually, it will get close to the minimum (assuming a convex problem), but it will continue to go up and down a bit.
  - So, once you stop it, the parameters it finds will be close to the best, but not necessarily optimal.

Simulated Annealing

- Stochastic Gradient Descent may 'bounce' around the minimum loss.
- One solution to this is **Simulated Annealing**, where we gradually reduce the learning rate α.
  - Updates start out 'large' so you make progress.
  - But, over time, updates get smaller, allowing SGD to settle at or near the global minimum.
- The function that determines how to reduce the learning rate is called the **learning schedule**.
  - Reduce it too quickly and you may not converge on or near to the global minimum.
  - Reduce it too slowly and you may still bounce around a lot and, if stopped after too few iterations, may end up with a suboptimal solution.

Mini-Batch Gradient Descent

In **MiniBatch Gradient Descent**, the updates involve a calculation over randomly-selected subsets of the training set $\boldsymbol{X}$.

initialize $\boldsymbol{w}$ randomly;

for *each epoch* **do**
  shuffle the training set and split into mini-batches;
  for *each mini-batch* $\boldsymbol{X}_i, \boldsymbol{y}_i$ **do**
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{\alpha}{|\boldsymbol{X}_i|} \boldsymbol{X}_i^T (\boldsymbol{X}_i \boldsymbol{w} - \boldsymbol{y}_i)$;
  end
end

- Mini-Batch Gradient Descent is stochastic, therefore similar to SGD.
- But by working on subsets of the training set, rather than single examples, 'bouncing' may be gentler.
- It is very common when training Neural Networks.

Efficiency/scaling-up to large training sets

- Normal Equation:
  - This is linear in m, so can handle large training sets efficiently if they fit into main memory.
  - But it has to compute the inverse (or pseudo-inverse) of a $n \times n$ matrix, which takes time between quadratic and cubic in n, and so is only feasible for smallish n (up to a few thousand).
- Gradient Descent:
  - SGD scales really well to huge m.
  - All three Gradient Descent methods can handle huge n (even 100s of 1000s).

Finding the global minimum for OLS regression

- Normal Equation: guaranteed to find the global minimum.
- Gradient Descent: all a bit dependent on number of iterations, learning rate, learning schedule.

Feature scaling

- Normal Equation: scaling is not needed.
- Gradient Descent: scaling is needed.

Generality

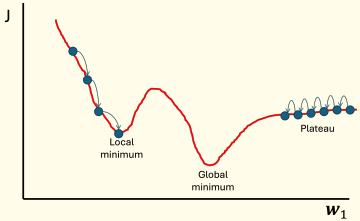
Gradient Descent is a general method, whereas the Normal Equation is only for OLS regression.

Convex Functions

- The loss function for OLS regression is convex and it has a slope that never changes abruptly.
- This gives us good 'guarantees' about reaching the minimum (depending on such things as running for long enough, using a learning rate that isn't too high, and whether we are using Batch, Mini-Batch or Stochastic Gradient Descent).
- But Gradient Descent is a generic method: you can use it to find the minima of other loss functions.

Non-Convex Functions

- Not all loss functions are convex, which can cause problems for Gradient Descent.

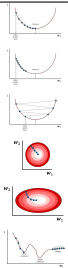


- The algorithm might converge to a local minimum, instead of the global minimum.
- It may take a long time to cross a plateau.

Gradient Descent for Non-Convex Problems

- One thing is to prefer Stochastic Gradient Descent (or Mini-Batch Gradient Descent): because of the way they 'bounce around', they might even escape a local minimum, and might even get to the global minimum.
- In this context, simulated annealing is also useful: updates start out 'large' allowing these algorithms to make progress and even escape local minima; but, over time, updates get smaller, allowing these algorithms to settle at or near the global minimum.
- But, if using simulated annealing, if you reduce the learning rate too quickly, you may still get stuck in a local minimum.

Image credits



I based these diagrams on ones to be found in A. Géron: Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow (2nd edn), O'Reilly, 2019

9. Logistic Regression

In this lecture, for binary classification we use the same student data that we used in a previous lecture. For multiclass classification, we use the Iris dataset. This widely-used dataset was created by the (in)famous statistician, Ronald Fisher (Fisher, R.A. "The use of multiple measurements in taxonomic problems", Annual Eugenics, 7, Part II, 179-188, 1936). The dataset is available all over the web now, especially the following dataset repository: https://archive.ics.uci.edu/ml/datasets/iris

	Regression	Classification
Decision Trees	Use MSE as the splitting criterion	Use, e.g., Gini as the splitting criterion
kNN	Take the mean of the neighbours' y-values	Take a majority-vote of the neighbours' y-values
Linear Models	Linear Regression, e.g. OLS regression which uses MSE as the loss function	Logistic Regression using cross-entropy as the loss function

Linear Regression

- In **Linear Regression** we want to find the line/plane/hyperplane that best fits the training examples — it 'goes through the middle of them'.

Logistic Regression

- Despite its name, **Logistic Regression** is a classifier, not a regressor.
- In Logistic Regression for binary classification, we want to find the line/ plane/ hyperplane that best *separates* training examples of different classes.
- Ideally, we want positive examples on one side and negatives on the other side. If we get this, we say that the dataset is **linearly separable**.

Revision

- Given an object $\boldsymbol{x}$, a classifier outputs a label, $\hat{y} \in \mathcal{C}$.
- Instead, a classifier could output a probability distribution over the labels $\mathcal{C}$.
- E.g. given an email $\boldsymbol{x}$, a spam filter might output $\langle 0.2, 0.8 \rangle$ meaning the probability that $\boldsymbol{x}$ is *ham* is 0.2, and the probability that it is *spam* is 0.8.
- The probabilities must sum to 1.
- In fact, a binary classifier can output just one probability, rather than two. Why?

Logistic Regression

- Given an example $\boldsymbol{x}$, Logistic Regression predicts the probability that $\boldsymbol{x}$ belongs to the positive class, $Prob(\hat{y} = 1 \mid \boldsymbol{x})$.

Logistic Regression

- Logistic Regression is a linear model, so we are trying to find values for b and $w_1, \ldots, w_n$:

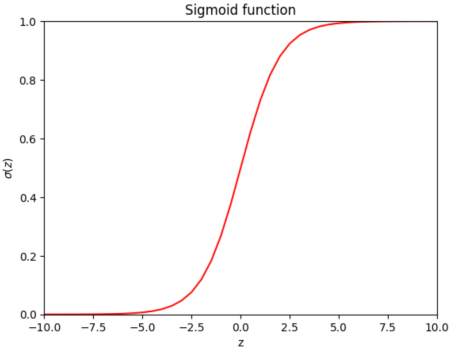
$$z = b + \boldsymbol{w}_1 \boldsymbol{x}_1 + \cdots + \boldsymbol{w}_n \boldsymbol{x}_n$$

- But we need it to produce a probability.
- To 'squash' z to $[0, 1]$, we use the **sigmoid function** (also called the **logistic function** or the **logit**):

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- Putting it all together:

$$Prob(\hat{y} = 1 \mid \boldsymbol{x}) = \sigma(b + \boldsymbol{w}_1 \boldsymbol{x}_1 + \cdots + \boldsymbol{w}_n \boldsymbol{x}_n)$$

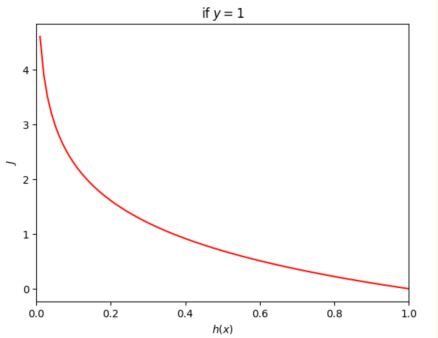


The Loss Function for Logistic Regression

- During training, we must find values for the parameters, $b, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_n$.
- Recall that we refer to each choice as a **hypothesis**, h.
- We need a **loss function** to score the hypotheses — we pick the hypothesis with lowest loss.
- We need a loss function, which penalizes a hypothesis if
  - it outputs high probabilities for negative examples; and
  - it outputs low probabilities for positive examples.
- The loss function that we will use is called **binary cross-entropy** and is based on logarithms. (It is also called the **log loss function**.)

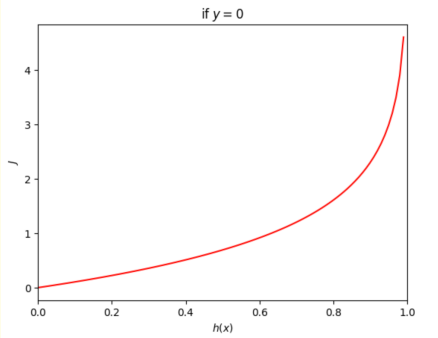
Positive Examples

- Consider a single training example $\boldsymbol{x}$ whose actual class $y = 1$.
- $-\log(h(\boldsymbol{x}))$ is a useful loss function:



Negative Examples

- Consider a single training example $\boldsymbol{x}$ whose actual class $y = 0$.
- $-\log(1 - h(\boldsymbol{x}))$ is a useful loss function:



Binary Cross-Entropy

- In summary, for a given example $\boldsymbol{x}$, we propose the following as the loss function

$$\begin{cases} -\log(h(\boldsymbol{x})) & \text{if } y = 1 \\ -\log(1 - h(\boldsymbol{x})) & \text{if } y = 0 \end{cases}$$

  Or, equivalently:

$$-y \log(h(\boldsymbol{x})) + -(1 - y) \log(1 - h(\boldsymbol{x}))$$

  Or

$$-[y \log(h(\boldsymbol{x})) + (1 - y) \log(1 - h(\boldsymbol{x}))]$$

- The overall loss function, J, is simply the average of this over all the training examples:

$$J(\boldsymbol{X}, \boldsymbol{y}, h) = -\frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}, y \in \boldsymbol{y}} [y \log(h(\boldsymbol{x})) + (1 - y) \log(1 - h(\boldsymbol{x}))]$$

Finding the Hypotheses that Minimizes Binary Cross-Entropy

- Happily, this loss function is convex. What does that mean?
- But there is no equivalent to the Normal Equation.
- We must use Gradient Descent.
- Since we are using Gradient Descent, we must remember to scale the data.

Stochastic Gradient Descent

initialize $\boldsymbol{w}$ randomly;

for *each epoch* **do**
  shuffle the training set;
  for *each example* $\boldsymbol{x}$ **do**
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \boldsymbol{x}(\sigma(\boldsymbol{w}\boldsymbol{x}) - y)$;
  end
end

Types of Classification

- **Binary Classification**: there are just two classes, i.e. $|\mathcal{C}| = 2$, e.g. fail/pass, ham/spam, benign/malignant.
- **Multiclass Classification**: there are more than two classes, i.e. $|\mathcal{C}| > 2$
  - E.g. a post to a forum or discussion board can be a question, an answer, a clarification or an irrelevance.
  - E.g. an iris can be Iris setosa, Iris versicolor or Iris virginica

Even More Types of Classification

- **Multilabel Classification**: the classifier can assign $\boldsymbol{x}$ to more than one class.
  - The classifier outputs a *set* of labels, $\hat{y} \subseteq \mathcal{C}$.
  - E.g. consider a movie classifier where the classes are genres, e.g. $\mathcal{C} = \{comedy, action, horror, musical, romance\}$ The classifier's output for *The Blues Brothers* should be $\{comedy, action, musical\}$.

  Do *not* confuse this with **multiclass classification**.
- **Ordered classification**: there is an *ordering* defined on the classes.
  - The ordering matters in measuring the performance of the classifier.
  - E.g. consider a classifier that predicts a student's degree class, i.e. $\mathcal{C} = \{Ordinary, 3rd, 2ii, 2i, 1st\}$
    - Suppose for student $\boldsymbol{x}$, the actual class $y = 1st$
    - One classifier predicts $\hat{y} = 2ii$
    - Another classifier predicts $\hat{y} = 2i$
    - Which classifier has done better?

Mutliclass Classifiers

- Decision Trees can be used for multiclass classification without change.
- kNN classifiers also.
- But Logistic Regression is for binary classification.

Multinomial Logistic Regression

- **Multinomial Logistic Regression** is for multiclass classification.
- Instead of learning one set of parameters, it learns one per class — in other words, it is learning a line/ plane/ hyperplane for each class.
- Instead of outputting one probability, it outputs $|\mathcal{C}|$ probabilities — one per class.
- 'Squashing' is more complicated because not only must each output be squashed to $[0, 1]$, they must also sum to 1. So we do not use the sigmoid function. We use the **softmax function**.
- In place of binary cross-entropy, we use a generalization called **categorical cross-entropy**.

For those who want all the maths

Multinomial Logistic Regression

- Multinomial Logistic Regression learns a set of parameters $\boldsymbol{w}_c$ for each class $c \in \mathcal{C}$
- Then, the probability that $\boldsymbol{x}$ belongs to class c will use $\boldsymbol{w}_c$ with the **softmax** function:

$$Prob(\hat{y} = c \mid \boldsymbol{x}) = \sigma(\boldsymbol{w}_c \boldsymbol{x}) = \frac{e^{\boldsymbol{w}_c \boldsymbol{x}}}{\sum_{\kappa \in \mathcal{C}} e^{\boldsymbol{w}_\kappa \boldsymbol{x}}}$$

Categorical Cross-Entropy

- The loss function for Multinomial Logistic Regression generalises binary cross-entropy:

$$J(\boldsymbol{X}, \boldsymbol{y}, h) = \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}, y \in \boldsymbol{y}} \sum_{\kappa \in \mathcal{C}} int(y = \kappa) log(\sigma(\boldsymbol{w}_\kappa \boldsymbol{x}))$$

10. Ensembles

The wine dataset that we use in this lecture is available from the following dataset repository: https://archive.ics.uci.edu/dataset/186/wine+quality. However, we are using a version that I have modified slightly: (a) I merged separate CSV files for red and white wines into a single file, and (b) I reduced class imbalance a little.

The Wisdom of the Crowd

- The aggregated opinions of a diverse group of independent individuals yields the best judgment.
- E.g. trial-by-jury, committees, democracies, social network ratings.

Ensembles

- We can *combine* models into what is called an **ensemble**.
- The ensemble often out-performs the individual models that it contains.
- Ensembles can turn **weak learners** into **strong learners**.
  - Weak learners: one that are doing only slightly better than chance.
  - Strong learners: ones with much higher accuracy/lower error than chance.
- If this is to happen, there needs to be diversity among the individual models.

Voting/Averaging

- The simplest approach.
- E.g. for classification:
  - Training: Train several different classifiers (e.g. Decision Tree, Logistic).
  - Inference: each model classifies the example; they vote on the result.
- Question: what do we do for regression?

Bagging

See next slides.

Boosting

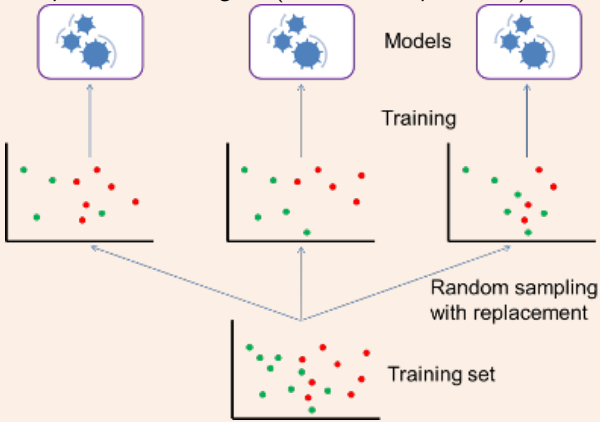
See next slides.

Stacking

Not covered in CS3315.

Bagging ('bootstrap aggregating')

- Training: train several models *of the same kind* (e.g. several Logistic Regression classifiers) *but on different random subsamples of the training set* (chosen *with replacement*).



- Inference: each model makes a prediction and these predictions are combined using voting/averaging.

Aggregation in Bagging

- In bagging, the aggregation (voting or averaging) is *not* usually weighted, i.e. all the models in the ensemble are treated equally.
- Contrast with boosting.

How many models should we use?

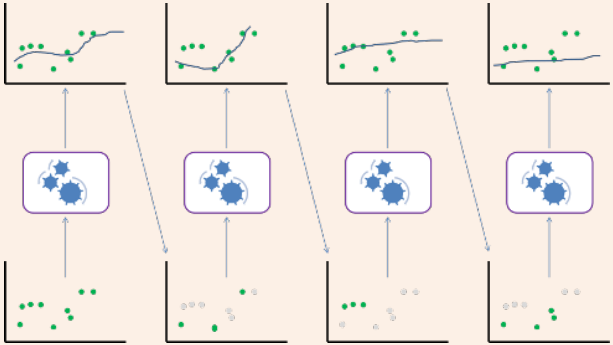
- To some extent, 'the more, the merrier': predictions typically become more reliable as the ensemble gets larger.
- Seldom is the accuracy of the ensemble lower than the accuracy of any of its members — but this is *not guaranteed*.
- The number of models is a **hyperparameter**: we can either guess how many models to use or choose its value using, e.g., grid search.

Random Forests

- A **Random Forest** uses **bagging** to create an ensemble of **Decision Trees**.
- Very popular and very successful.
- There are ways of increasing the diversity of the Decision Trees in a Random Forest — in addition to the random subsampling of the training set.
- For example, when learning the trees, instead of selecting the best feature for splitting, each tree might select the best among a randomly-chosen subset of the features.

Boosting

- Training: train several models *of the same kind* but *sequentially*.
  - In the first model, all training examples are treated equally.
  - In subsequent models, training examples on which the previous model made an error receive greater weight.



- Inference: each model makes a prediction and these predictions are combined using voting/averaging. However, we use a weighted vote/weighted mean, where a model's weight is based on its performance on the training set.

AdaBoost for Classification

- Training:

> Assign equal weights to each example in the training set;
>
> **for** $i \in 1 \ldots n\_estimators$ **do**
>
> Train classifier i on the weighted training set;
>
> Compute training accuracy for classifier i;
>
> Misclassified examples are given new, higher weights;
>
> **end**

- Inference:
  - Each classifier in the ensemble classifies the example.
  - Take a weighted vote using the training accuracies as the weights.

Weighted Training Examples

- AdaBoost assumes that, when we train the classifiers in the ensemble, they are sensitive to the weights of the training examples.
- So far, none of the classifiers we have studied does this — although it is easy to adapt kNN.
- Question: Suppose a classifier cannot handle weighted training examples. What can we do to give the same effect?

Gradient Boosting for Regression

- **Gradient Boosting** is similar to AdaBoost: it trains several models of the same kind, sequentially
- But, it does *not* use a weighted training set.
- Instead, each regressor is trained on the **errors** of the previous regressor.

Gradient Boosting: Training

- Train regressor 1 on $\boldsymbol{X}$ and $\boldsymbol{y}_1 = \boldsymbol{y}$.
- Use regressor 1 to predict $\hat{\boldsymbol{y}_1}$ for $\boldsymbol{X}$.
- Train regressor 2 on $\boldsymbol{X}$ and $\boldsymbol{y}_2 = \hat{\boldsymbol{y}_1} - \boldsymbol{y}_1$.
- Use regressor 2 to predict $\hat{\boldsymbol{y}_2}$ for $\boldsymbol{X}$.
- Train regressor 3 on $\boldsymbol{X}$ and $\boldsymbol{y}_3 = \hat{\boldsymbol{y}_2} - \boldsymbol{y}_2$.

Gradient Boosting: Inference

- Each regressor in the ensemble makes a prediction for the example.
- But the final prediction is the *sum* of these predictions.
- Question: Why does this make sense?

- An ensemble can be more accurate than the individual models within it.
- But there are problems too.

Computational costs

- Training time increases.
- Inference time increases.

XAI

- Do you think an ensemble is an *interpretable* model?
- Do you think that a prediction from an ensemble is easily *explainable*?

Image credits

Taken from tiktok.com

I based these diagrams on ones to be found in A. Géron: Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow (2nd edn), O'Reilly, 2019

11. Model Complexity

In this lecture, we use several of the datasets that we used previously.

Non-Linear Models

- Suppose we have three features, $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{x}_3$.
- A linear model will learn values for parameters $b, \boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3$:

$$y = b + \boldsymbol{w}_1\boldsymbol{x}_1 + \boldsymbol{w}_2\boldsymbol{x}_2 + \boldsymbol{w}_3\boldsymbol{x}_3$$

- But what if the true relationship between the features and the target values is non-linear?
- Then you can use non-linear approaches such as a Decision Trees, kNN or even neural networks.

Polynomial Models

- But there's a really neat trick for using a *linear model* to get some of the same effect!
- We systematically add extra features to our dataset.
- Some of the new features will be *powers* of the original features, e.g. a new feature $\boldsymbol{x}_4 = \boldsymbol{x}_1^2$.
- Others will be *products* of the original features, e.g. a new feature $\boldsymbol{x}_7 = \boldsymbol{x}_1\boldsymbol{x}_2$. (These are often called *interaction features*).
- (This should be reminiscent of *feature engineering* — where we derived new features from existing features.)
- Then learn a linear model on the new dataset.

Example: Quadratic Model

- We want to learn models of this form:

$$y = b + \boldsymbol{w}_1\boldsymbol{x}_1 + \boldsymbol{w}_2\boldsymbol{x}_2 + \boldsymbol{w}_3\boldsymbol{x}_3 + \boldsymbol{w}_4\boldsymbol{x}_1^2 + \boldsymbol{w}_5\boldsymbol{x}_2^2 + \boldsymbol{w}_6\boldsymbol{x}_3^2 + \boldsymbol{w}_7\boldsymbol{x}_1\boldsymbol{x}_2 + \boldsymbol{w}_8\boldsymbol{x}_1\boldsymbol{x}_3 + \boldsymbol{w}_9\boldsymbol{x}_2\boldsymbol{x}_3$$

- But instead we learn linear models of this form:

$$y = b + \boldsymbol{w}_1\boldsymbol{x}_1 + \boldsymbol{w}_2\boldsymbol{x}_2 + \boldsymbol{w}_3\boldsymbol{x}_3 + \boldsymbol{w}_4\boldsymbol{x}_4 + \boldsymbol{w}_5\boldsymbol{x}_5 + \boldsymbol{w}_6\boldsymbol{x}_6 + \boldsymbol{w}_7\boldsymbol{x}_7 + \boldsymbol{w}_8\boldsymbol{x}_8 + \boldsymbol{w}_9\boldsymbol{x}_9$$

  but where

$$\boldsymbol{x}_4 = \boldsymbol{x}_1^2 \quad \boldsymbol{x}_7 = \boldsymbol{x}_1\boldsymbol{x}_2$$

$$\boldsymbol{x}_5 = \boldsymbol{x}_2^2 \quad \boldsymbol{x}_8 = \boldsymbol{x}_1\boldsymbol{x}_3$$

$$\boldsymbol{x}_6 = \boldsymbol{x}_3^2 \quad \boldsymbol{x}_9 = \boldsymbol{x}_2\boldsymbol{x}_3$$

- Question: What would a Cubic Model look like?

Warning

- Polynomial Regression of degree d transforms a dataset that had n features into one that has $\frac{(n+d)!}{d!n!}$ features.

Model Complexity

- The **complexity of a model** is its capacity to capture the underlying patterns in the data.
- Do not confuse with **time complexity**, **space complexity**, **sample complexity**, ...

Examples

- Decision trees: complexity increases with depth.
- kNN: complexity decreases with k. (Why?)
- In parametric learning: complexity increases with the number of parameters.
- Polynomial models: complexity increases with the degree. (Why is this saying the same as the previous bullet point?)

Underfitting and Overfitting

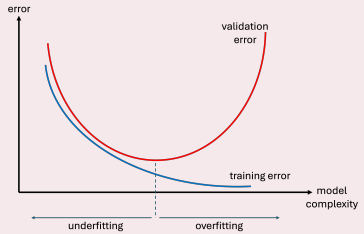
- **Underfitting**: the model is not complex enough.
- **Overfitting**: the model is too complex.

Recap

- We train on the training set.
- But we can evaluate on the training set, validation set or test set.
- This gives us the **training error**, **validation error** or **test error**.

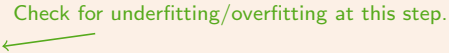
Detecting underfitting/overfitting

- Underfitting: the validation error is high and even the training error is high.
- Overfitting: the training error is low but the validation error is high.



Machine Learning Projects (from previous lecture)

1. Understand the business problem.
2. Select performance measures.
3. Acquire a dataset.
4. Take a cheeky look.
5. Cleanup anything that is simply invalid.
6. Split into training and test sets using holdout.
7. Exploratory Data Analysis.
8. Feature Engineering.
9. Preprocess the data.
10. Model Selection.
11. Repeat steps 7–10 until satisfied.
12. Error Estimation: test on the test set.
13. Decide whether to deploy.
14. If yes, then train on the whole dataset.
15. Once deployed, monitor and retrain regularly.



Remedies for underfitting

- Gathering more training examples **will not help**.

- Change to a more complex model:
  - Polynomial regression: increase the degree.
  - Decision Trees: increase the depth.
  - kNN: reduce k.
- Collect data for additional features that you hope will be more predictive.
- Feature engineering: create new features which you hope will be predictive.

Remedies for overfitting

- Gathering more training examples may help. (Why?)
- If we cannot get more genuine examples, we may synthesize additional examples (see **data augmentation** in a later lecture).

- Change to a less complex model:
  - Polynomial regression: reduce the degree.
  - Decision Trees: reduce the depth.
  - kNN: increase k.
- Clean the training examples to remove noise.
- Dimensionality reduction/feature selection: to reduce the number of features.
- Parametric models: use **Regularization**.
- Decision Trees: use post-pruning.
- Neural Networks: use dropout; use batch-normalisation (see later lectures).
- Gradient Descent: use early-stopping.
- Use an ensemble (although the relationship between ensembles and overfitting is not well-understood).

Regularization for Parametric Models

- **Regularization** constrains the range of values that the parameters can take.
- We 'encourage' the learning algorithm to only choose small values (close to zero).
- This makes the distribution of the values of the parameters more regular.
- We do this by modifying the loss function, e.g.

$$J(\boldsymbol{X}, \boldsymbol{y}, h) = \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}, y \in \boldsymbol{y}} (h(\boldsymbol{x}) - y)^2 + \text{penalty}$$

- This modified loss function prefers a hypothesis that (a) fits the data, while at the same time (b) uses small parameter values.

Lasso Regression

- In **Lasso Regression**, we penalize by the l_1-norm of the weights, which is simply the sum of their absolute values, i.e.:

$$J(\boldsymbol{X}, \boldsymbol{y}, h) = \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}, y \in \boldsymbol{y}} (h(\boldsymbol{x}) - y)^2 + \lambda \sum_{j=1}^{n} |\boldsymbol{w}_j|$$

(Minor point: we don't penalize b.)

The Regularization Parameter

- λ is called the **regularization parameter**.
- It controls how much penalization we want and this determines the balance between the two parts of the modified loss function: fitting the data versus shrinking the parameters.
- When $\lambda = 0$, there is no regularization; but as λ increases, the penalties will get ever greater.
- As λ grows, some of the $\boldsymbol{w}_j$ will be driven to zero. This means that the model learns to treat some features as irrelevant. Hence, Lasso Regression performs some **feature selection** too. (Compare this with Ridge Regression.)
- Lasso Regression can be more interpretable than Linear Regression. Why?
- λ isn't actually a 'parameter'. What should we call it and how could we choose its value?

Ridge Regression

- In **Ridge Regression**, we penalize by the l_2-norm, which is simply the sum of the squares of the weights, i.e.:

$$J(\boldsymbol{X}, \boldsymbol{y}, h) = \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}, y \in \boldsymbol{y}} (h(\boldsymbol{x}) - y)^2 + \lambda \sum_{j=1}^{n} \boldsymbol{w}_j^2$$

(Strictly speaking, the l_2-norm would be the square root of the sum of squares.)

- Both Lasso and Ridge Regression shrink the values of the weights
- But we saw Lasso Regression may additionally result in coefficients being set to zero.
- This does not happen with Ridge Regression.
- Optionally, consult section 3.4.3 of The Elements of Statistical Learning by Hastie, Friedman & Tibshirani (available online) for an explanation.
- One observation from the book is that, roughly speaking, Lasso Regression shrinks the coefficients by approximately the same constant amount (unless they are so small that they get shrunk to zero), whereas, again roughly speaking, Ridge Regression shrinks the coefficients by approximately the same proportion.

12. Feature Selection

In this lecture, we use a dataset about glass and its uses. It was collected by the *USA Forensic Science Service*. It is publicy-available from the UCI Machine Learning Repository.

Recap

- **Overfitting**: the model is too complex.
- One remedy, especially, for parametric models, is to reduce the number of features (and hence the number of parameters).
- Regularization of a parametric model using the $l_1$norm (Lasso) can set weights to zero, which is a form of feature selection.

Feature Selection and Dimensionality Reduction

- **Feature Selection**: select a subset of the features.
  - Sequential Feature Selection Algorithms.
  - Filter Methods that use a scoring function.
  - Filter Methods that use a separate model.
- **Dimensionality Reduction**: map the existing features to a new feature space and retain only some of the new features.

Sequential Feature Selection Algorithms

- There are *many* approaches to feature selection.
- Among the simplest are **Sequential Feature Selection Algorithms**.
- Greedy algorithms choosing a feature to remove/add one at a time.
- Stop when you have the desired number of features — a hyperparameter, so could choose it by, e.g., grid search.
- But, if there are many features, then these algorithms train and evaluate many models — expensive!

Sequential Backward Selection

$\mathcal{F} \leftarrow$ all the features;

repeat
  Determine which feature f has least impact on validation error;

  Remove f from $\mathcal{F}$;

until *until stopping criterion*;

Sequential Forward Selection

$\mathcal{F} \leftarrow \{\}$;

repeat
  Determine which feature f has most impact on validation error;

  Insert f into $\mathcal{F}$;

until *until stopping criterion*;

Filter Methods for Feature Selection

- Sequential Feature Selection Algorithms are costly to run.
- The alternative is to use a **filter method**.
- In a filter method, we compute **feature importances** in some separate way.
- Then we retain the features with highest importance.

Examples

- There are numerous filter methods available.
- For regression, feature importance can be the F-value — based on the Pearson correlation between the feature and the target.
- For classification, feature importance can be the ANOVA F-value — a version of the F-value adapted for classification.

Disadvantages

- Being based on Pearson, these scores measure linear relationships only.
- This approach measures importance for each feature in isolation — ignoring whether they are redundant when other features are present or more useful jointly with other features.

Using a Separate Model

- Some ML models can output feature importances, e.g. Decision Trees, Random Forests.
- Instead of using, e.g., F-values, we can train a Decision Tree or Random Forest, use its feature importances to filter the features, then train the model we ultimately want.
- This can overcome the disadvantages above.
- It may be more accurate than the Decision Tree or Random Forest on its own.

Principal Components Analysis

Principal Components Analysis (PCA) gives a method for **dimensionality reduction**.

- PCA creates new features —called **components**— based on the existing ones. The components are linear combinations of the existing features. There is one component per existing feature.
- But these components are ordered — meaning we can discard the least important ones.

Mathematically speaking...

- PCA mean-centers the training examples — by subtracting each feature's mean from the feature-values.
- Calculates the covariance matrix of the mean-centered training examples.
- Calculates the eigenvectors of the covariance matrix. These are the axes of a transformed space — the new features. But there are still n of them.
- Retains a subset of the new features — the ones with the largest eigenvalues.

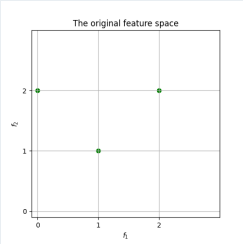
There follows a very informal presentation of PCA.

Example

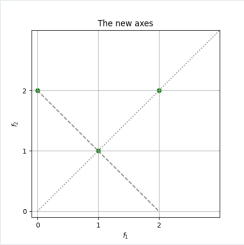
Suppose a dataset describes objects using just two features, f_1 and f_2.

f_1	f_2
1	1
2	2
0	2

Since there are only two features, we can plot — a 2D visualization:



- But the co-ordinate system we use is arbitrary.
- A different pair of axes would work just as well.
- E.g. we could draw one axis diagonally.
- By convention, the other will be perpendicular (or orthogonal) to the first — in other words, at right angles.
- But, even then, we can decide where along the first axis to place it.



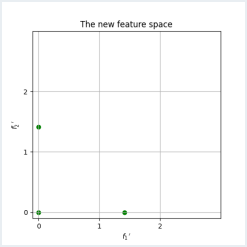
- On this new 'graph paper', the co-ordinates of the examples are now different.

f_1	f_2
1	1
2	2
0	2

becomes

f_1'	f_2'
0	0
$\sqrt{2}$	0
0	$\sqrt{2}$

- We can see this if we rotate the graph paper, so that the diagonal is horizontal:



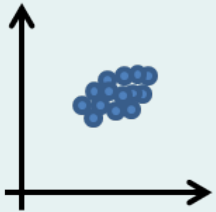
- These new features convey exactly the same information as the originals — they're just on a different co-ordinate system.

Choosing the new axes

- There is an infinite choice of axes!
- In PCA, the orientation of the first axis is based on covering the greatest variance (spread).
- Each subsequent axis must cover as much remaining spread as possible but with the constraint that they must be perpendicular to one another.
- Making these decisions does require that the features be comparable. This is why PCA mean-centers their values first.

Question

The plot contains examples described by two features, which you can assume have been standardized.



PCA will choose two new axes. Where will they be?

Principal Components

- At this stage, we've gained very little: we had n features (axes), we still have n features (new axes, components).
- But examples in real-world datasets often lie close to a lower-dimensional subspace of the high-dimensional space.
- So, we can re-express our data ('project it') to just the first n' axes ($n' < n$), hopefully, with very little loss of information.

Explained Variance Ratio

- How do we choose n' — the number of components to retain?
- We can treat n' as a hyperparameter and set it using, e.g., grid search.
- Or we can plot the **explained variance ratio** of each component — tells us how much of the training sets' variance lies along that axis.
- There is often an 'elbow' in the plot — this gives the value of n'

Limitations of PCA

- It can only be used on the numeric-valued features.
- Its new features are always linear combinations of the existing ones: in essence, we're fitting straight-line axes through the values.
- Being a form of unsupervised learning, it ignores the target values/labels. (This is also an advantage: it can be used for both regression and classification.)

Alternatives to PCA

- There are many techniques that work in a similar way but with more flexibility.
- For non-numeric data, e.g. Canonical Correspondence Analysis (CCA).
- For bag-of-words (next lecture), e.g. Singular Value Decomposition (SVD).
- To allow non-linear combination, e.g. Kernel PCA, Isomap, Locally Linear Embedding, Multidimensional Scaling (MDS).
- To take the classes into account in classification tasks, e.g. Linear Discriminant Analysis (LDA).

XAI

Feature Selection

- Feature Selection (including l_1 regularization of parametric models) can reduce the number of features.
- This may improve explainability.

Dimensionality Reduction

- Dimensionality Reduction, e.g. PCA, can also be used to reduce the number of features.
- But it worsens explainability. Why?

13. Text Preprocessing

In this lecture, our dataset comprises just three tweets.

Structured Dataset

- Each example fits a predefined schema.
- Most common is rows and columns, hence also called **tabular datasets**.

Unstructured datasets

- Examples are in their native format.
- E.g. a dataset of texts, or images, or audio, or video.

Text Datasets

- A **text dataset** is also sometime called a **corpus**.
- Each *example* is sometimes referred to as a **document** — whether it be as short as a tweet/ SMS/ email or as long as a student essay.

Text Preprocessing

- **Text Preprocessing** converts examples from raw text into a more structured, numeric format that can be handled by Machine Learning algorithms.
- We study **Bag of Words**, where each document becomes a vector of numbers.

Sets

- A **set** is a collection of objects with two properties:
  - Order is not important, e.g. $\{a, c, f\} = \{c, f, a\}$
  - Duplicates are not allowed, e.g. $\{a, c, a, f\} = \{a, c, f\}$
- The set of all possible objects U is called the **universal set**, e.g. $U = \{a, b, c, d, e, f, g\}$

Data Structures for Sets

- There are many data structures we can use to store sets, e.g. linked lists, binary search trees, ...
- But we can describe a set by a binary-valued vector.
- E.g. if $U = \{a, b, c, d, e, f, g\}$, then we can represent the set $\{a, c, f\}$ as follows:

a	b	c	d	e	f	g
1	0	1	0	0	1	0

- Then, we can store the vector as, e.g., an array.
- In fact, if U is large but the sets are much smaller, then our vectors will be **sparse** (mostly zero).
- It may be more efficient to use a data structure that only stores the non-zero elements.

Bags

- A **bag** is a collection of objects with one property:
  - Order is not important, e.g. $\{a, c, f\} = \{c, f, a\}$
  - This time, duplicates *are* allowed, e.g. $\{a, c, a, f\} \neq \{a, c, f\}$

Data Structures for Bags

- We can describe a bag by an integer-valued vector, where the numbers are the frequencies with which the elements occur.
- E.g. if $U = \{a, b, c, d, e, f, g\}$, then we can represent the bag $\{a, c, a, f\}$ as follows:

a	b	c	d	e	f	g
2	0	1	0	0	1	0

- We can store these as arrays or sparse arrays.

Bag Of Words

- We will represent each document in our corpus (each example in our text dataset) as a **bag of words**.
- This will lose lots of information. Start thinking about what we will lose!
- Nevertheless, bag-of-words is very competitive on text classification tasks such as spam filtering and sentiment analysis.

Running Example

A dataset comprising three Barack Obama tweets (quoting Nelson Mandela)

"No one is born hating another person because of the color of his skin or his background or his religion."
"People must learn to hate, and if they can learn to hate, they can be taught to love."
"For love comes more naturally to the human heart than its opposite."

Text Preprocessing

- Tokenization.
- Optionally, discard stop-words.
- Optionally, apply stemming or lemmatization.
- Either count vectorization or TF-IDF vectorization.

Tokenization

- First, we must **tokenize** each document. This means splitting it into **tokens** (sometimes also called **terms**).
- Simplest approach: change uppercase to lowercase, treat punctuation symbols as spaces, split at the spaces.
- Roughly, this mean one token = one word.

Discussion

- In reality, tokenization is surprisingly complicated.
- E.g. should we treat "People" and "people"' as different tokens?
- E.g. should we keep the punctuation as separate tokens?
- E.g. is "don't" one token or two or three?
- E.g. should **bigrams** (pairs of consecutive words) become tokens?

Running Example

{"no", "one", "is", "born", "hating", "another", "person", "because", "of", "the", "color", "of", "his", "skin", "or", "his", "background", "or", "his", "religion"}
{"people", "must", "learn", "to", "hate", "and", "if", "they", "can", "learn", "to", "hate", "they", "can", "be", "taught", "to", "love"}
{"for", "love", "comes", "more", "naturally", "to", "the", "human", "heart", "than", "its", "opposite"}

Stop-words

- Optionally, we can discard **stop-words**.
- Typically, stop-words are common words such as "a", "the", "in", "on", "is", "are",...
- (Linguists call these the closed-class words or function words or grammatical words —categories of words, such as articles and prepositions, that do not readily accept new members— as opposed to the open-class words or content words or lexical words —categories of words, such as nouns, verbs, adjectives and adverbs, where it is easier to add new words.)

Discussion

- For simple classifiers (e.g. spam filters, sentiment analysers), discarding stop-words does no harm.
- Other times, we would lose too much, e.g. consider discarding them from queries to a web search engine where the query is "to be or not to be".

Running Example

{"born", "hating", "person", "color", "skin", "background", "religion"}
{"people", "learn", "hate", "learn", "hate", "taught", "love"}
{"love", "comes", "naturally", "human", "heart", "opposite"}

Stemming

- Optionally, we can **stem** the words.
- Stemming uses rules-of-thumb to remove or replace prefixes or suffixes, so that variants of a word are reduced to the same stem form.
- E.g. drop "ing", "s", "ly", "un", ...

Lemmatization

- Also optional, **lemmatization** is an alternative to stemming.
- It reduces variants of a word to the base or root form of the word, as it appears in a dictionary.

Discussion

- Stemming/lemmatization reduce dimensionality and may reveal commonalities among texts that were originally different.
- Lemmatization is more expensive than stemming.

Running Example: Stemming (by NLTK's rules)

{"born", "hate", "person", "color", "skin", "background", "religion"}
{"peopl", "learn", "hate", "learn", "hate", "taught", "love"}
{"love", "come", "natur", "human", "heart", "opposit"}

Running Example: Lemmatization

{"born", "hate", "person", "color", "skin", "background", "religion"}
{"people", "learn", "hate", "learn", "hate", "teach", "love"}
{"love", "come", "natural", "human", "heart", "opposite"}

Count Vectorization

- Each document (example) becomes a vector.
- Each token becomes a feature.
- Feature-values are **frequencies** — how many times that token appears in that document.

Running Example

back-ground	born	color	come	hate	heart	human	learn	love	natur	opposit	peopl	person	religion	skin	taught
1	1	1	0	1	0	0	0	0	0	0	0	1	1	1	0
0	0	0	0	2	0	0	2	1	0	0	1	0	0	0	1
0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0

TF-IDF Vectorization

- **TF-IDF Vectorization** is an alternative to Count Vectorization.
- The feature-values are now **tf-idf scores** — they penalize words that recur across multiple documents on the principle that they are less discriminative.
- E.g. in emails, words such as "hi", "best", "regards" will be penalized.

Running Example

back-ground	born	color	come	hate	heart	human	learn	love	natur	opposit	peopl	person	religion	skin	taught
0.39	0.39	0.39	0	0.30	0	0	0	0	0	0	0	0.39	0.39	0.39	0
0	0	0	0	0.51	0	0	0.67	0.26	0	0	0.34	0	0	0	0.34
0	0	0	0.42	0	0.42	0.42	0	0.32	0.42	0.42	0	0	0	0	0

TF-IDF

- Simplest version:

$$tf\_idf(t, d) = tf(t, d) \times idf(t, D)$$

where $tf(t, d)$ is **term frequency**

$$tf(t, d) = \frac{\text{number of times token } t \text{ appears in document } d}{\text{total number of tokens in document } d}$$

and $idf(t, D)$ is **inverse document frequency**

$$idf(t, D) = \log \frac{\text{total number of documents in corpus } D}{\text{number of documents containing token } t}$$

- Variants of the formula might: logarithmically scale tf to avoid biases towards long documents; add 1 to parts of the formula to avoid edge cases; normalize the results, e.g. divide by the l_2-norm.

Unigrams

- We have assumed **unigrams**: one token = one word.
- E.g. "No one is born hating another person because of the color of his skin or his background or his religion."
- The tokens are "no", "one", "is", "born", ...

Bigrams

- We could additionally use **bigrams**: one token = a pair of consecutive words.
- E.g. in addition to the unigrams, the tokens are "no one", "one is", "is born"', "born hating", ...

Trigrams

- We could additionally use **trigrams**: one token = sequences of three consecutive words.
- E.g. in addition to the unigrams and bigrams, the tokens are "no one is", "one is born", "is born hating"', ...
- In general, we refer to *n*-**grams**.

- The cost is many more features (tokens) — vectors of even higher dimensionality and sparsity.
- The advantage is now we are making available to the classifier some of the grammatical structure (word order, the functions of the stop-words, etc) — we lose it if we use only unigrams.

Task-Specific Tokens (Hacks!)

Spam Filtering

- Words with punctuation inside them, e.g. “loan$”, “mort.gage”, “vi@gra”, can all be mapped to a single token – `SPECIAL_TOKEN`.
- Tell-tale phrases can be treated as single tokens, e.g., “submit your manuscript”, “urgent reply” — even when you are not using *n*-grams more generally.

Sentiment Analysis

- Words preceded by “not” can be replaced by a single token, e.g. “not like” and “not recommend” become `NOT_like` and `NOT_recommend` — even when you are not using *n*-grams more generally.
- Emoticons can be left as tokens — even when other punctuation is being stripped. E.g. `:)` and `:=(`

Language Identification

- Use character bigrams and trigrams as tokens — instead of words.
- E.g. tokens from English: “goo”, “ood”, “od ”, “d l”, “ lu”, “luc”, “uck”.
- E.g. tokens from mystery language: “pow”, “owo”, “wod”, “odz”, “dze”, “zen”, “eni”, “nia”.

14. Text Classification

Stanford University researchers have taken 50,000 movie reviews from IMDB, labelled them as either positive or negative and made them available. I've taken 25,000 of them to use as a dataset to illustrate this lecture.

Possible Uses of Bag-Of-Words

Bag-Of-Words might be a competitive approach on tasks such as:

- **spam filtering**
- **email sorting**, e.g. announcements, personal,...
- **sentiment analysis**, e.g. positive, negative, neutral
- **language identification**
- **topic classification**, e.g. sports, politics, travel
- **inappropriate content filtering**, e.g. racism, sexism, sexually explicit content, extremism,...

Discussion

Why might a classifier of social media posts incorrectly classify some of them as inappropriate when they are not?

Advantages of Bag-Of-Words

- Simple.
- Efficient and scalable.
- Surprisingly effective at many text classification tasks.

Disadvantages of Bag-Of-Words

- Does not try to represent the semantics of the text.
- In particular, it ignores word order.
- In particular, it is unaware of synonyms.
- In particular, if we discard stop-words, then we lose the relationships that the stop-words convey.
- We may have to deal with its high dimensionality.
- To accommodate new vocabulary (e.g. from new training examples) requires that we run Text Preprocessing afresh — we end up with a different set of features (tokens).

Text Preprocessing (from previous lecture)

- Tokenization.
- Optionally, discard stop-words.
- Optionally, apply stemming or lemmatization.
- Either count vectorization or TF-IDF vectorization.

In reality...

We may need lots of additional preprocesssing, especially if the dataset has been scraped from the Web. For example:

- Some examples may use different character encodings, not UTF-8.
- They may not all be written in the same language, e.g. some in English, others in Hindi.
- There may be speling errors and tpyos.
- There may be markup (e.g. HTML) or other 'formatting' (e.g. SMTP headers, MIME headers).

High Dimensionality

- After Text Preprocessing, we can use standard Machine Learning algorithms to train classifiers, regressors, etc.
- However, n will be high – the number of features (the number of distinct tokens) — and the vectors will often be sparse.
- Reduce the number of features by:
  - discarding tokens that appear in too few documents (set a minimum df);
  - discarding tokens that appear in too many documents (set a maximum df);
  - keeping only the most frequent tokens (ordered by tf).
- Use dimensionality reduction: e.g. Singular Value Decomposition (SVD) is suitable for bag-of-words, rather than PCA.

Naive Bayes Classifier

- **Naive Bayes** is a linear classifier — simple, fast, often accurate — very popular, especially for simple text classification.
- Naive Bayes is a multiclass classifier but we will present the maths for the binary case, $\mathcal{C} = \{c, c'\}$.
- Let's assume we want to classify a document, *doc*. To simplify notation, assume *doc* is a bag-of-words (bag of tokens).
- Compute the **likelihood** that the class is c given *doc*, $\mathit{likelihood}(c \mid \mathit{doc})$, as follows:

$$\mathit{likelihood}(c \mid \mathit{doc}) = \mathit{Prob}(c) \prod_{t \in \mathit{doc}} \mathit{Prob}(t \mid c)$$

- Similarly, compute $\mathit{likelihood}(c' \mid \mathit{doc})$
- Choose the class with the higher likelihood.

$likelihood(c \mid doc) = Prob(c) \prod_{t \in doc} Prob(t \mid c)$

- $Prob(c)$ is the **class prior** — the probability of encountering this class.
- Estimate it from the training set:

$$Prob(c) = \frac{m_c}{m}$$

where m_c is the number of training examples whose class label is c, and m is the number of training examples.

$likelihood(c \mid doc) = Prob(c) \prod_{t \in doc} Prob(t \mid c)$

- $Prob(t \mid c)$ is the **conditional probability** that token t occurs in documents of class c.
- Estimate it from the training set:

$$Prob(t \mid c) = \frac{freq_{t,c}}{freq_c}$$

where $freq_{t,c}$ is the number of occurrences of token t in training examples whose class label is c, and $freq_c$ is the number of occurrences of tokens of any kind in training examples whose class label is c.

- But for some token-class combinations, this will be zero. This will make the final likelihood values also zero.
- **Laplace smoothing**: to eliminate zeros, we add 1 to each count

$$Prob(t \mid c) = \frac{freq_{t,c} + 1}{freq_c + n}$$

where n is the size of the vocabulary (number of distinct tokens).

Example

Training set

brill	funny	lame	mint	short	tame	class label
1	2	0	0	0	0	pos
0	2	0	0	1	0	pos
0	1	0	1	0	0	pos
0	1	1	0	0	1	neg

Test example (doc)

brill	funny	lame	mint	short	tame	class
0	3	1	0	0	1	?

Is it pos?

$$Prob(\text{pos}) = 3/4$$

$$\begin{aligned} Prob(\text{funny} \mid \text{pos}) &= (5+1)/(8+6) &&= 6/14 \\ Prob(\text{lame} \mid \text{pos}) &= (0+1)/(8+6) &&= 1/14 \\ Prob(\text{tame} \mid \text{pos}) &= (0+1)/(8+6) &&= 1/14 \end{aligned}$$

$$likelihood(\text{pos} \mid doc) = 3/4 \times (6/14)^3 \times 1/14 \times 1/14 = 0.0003$$

Is it neg?

$$Prob(\text{neg}) = 1/4$$

$$\begin{aligned} Prob(\text{funny} \mid \text{neg}) &= (1+1)/(3+6) &&= 2/9 \\ Prob(\text{lame} \mid \text{neg}) &= (1+1)/(3+6) &&= 2/9 \\ Prob(\text{tame} \mid \text{neg}) &= (1+1)/(3+6) &&= 2/9 \end{aligned}$$

$$likelihood(\text{neg} \mid doc) = 1/4 \times (2/9)^3 \times 2/9 \times 2/9 = 0.0001$$

Probabilities

If you want the classifier to output probabilities, then divide the likelihood by $Prob(doc)$:

$$Prob(c \mid doc) = \frac{Prob(c) \prod_{t \in doc} Prob(t \mid c)}{Prob(doc)}$$
$$= \frac{likelihood(c \mid doc)}{likelihood(c \mid doc) + likelihood(c' \mid doc)}$$

Example

- $Prob(c \mid doc) = \frac{0.0003}{0.0003+0.0001} = 0.75$
- $Prob(c' \mid doc) = \frac{0.0001}{0.0003+0.0001} = 0.25$

Why "Bayes"?

- **Bayes' theorem** states that the posterior probability $Prob(H \mid E)$ of a hypothesis H given evidence E is given by

$$Prob(H \mid E) = \frac{Prob(H)Prob(E \mid H)}{Prob(E)}$$

- Schematically,

$$posterior\ \ probability = \frac{prior\ \ probability \times conditional\ \ probability}{evidence}$$

- In our case,

$$Prob(c \mid doc) = \frac{Prob(c)Prob(doc \mid c)}{Prob(doc)}$$

Why "Naive"?

- The Naive Bayes classifier makes the naive simplifying assumption that $Prob(doc \mid c)$ can be calculated as follows:

$$Prob(doc \mid c) = \prod_{t \in doc} Prob(t \mid c)$$

  e.g. $Prob(\{\text{funny}, \text{funny}, \text{mint}\} \mid c)$ is given by $Prob(\text{funny} \mid c) \times Prob(\text{funny} \mid c) \times Prob(\text{mint} \mid c)$.
- In other words, it assumes **conditional independence** — hence we can multiply these probabilities.
- In truth, this assumption does not hold.
- E.g. if a movie review contains the word "car", it will be more likely that it contains the word "chase".
- But, in practice, Naive Bayes is effective even though the assumption is violated.

Underflow

- The formula requires that we multiply a lot of probabilities.
- This can result in **floating point underflow** — a result so small that it cannot be represented.
- But we can 'demote' multiplication to addition by using **logarithms**:

$$\log(xy) = \log(x) + \log(y)$$

- So we use

$$loglikelihood(c \mid doc) = \log Prob(c) + \sum_{t \in doc} \log Prob(t \mid c)$$

— the class with highest log-likelihood is still the most probable.

Training

During training, we can estimate all the probabilities we might ever need from the training set and cache them.

foreach $c \in \mathcal{C}$ **do**
    logprior[c] $= \log(m_c/m)$;
    foreach $t \in$ vocabulary **do**
        logcondprob[t][c] $= \log(\frac{freq_{t,c}+1}{freq_c+n})$
    end
end

Inference

To predict the class of *doc* (which is a bag):

foreach $c \in \mathcal{C}$ **do**
    loglikelihood[c] $\leftarrow$ logprior[c];
    foreach $t \in doc$ **do**
        loglikelihood[c] += logcondprob[t][c]
    end
end
return c for which loglikelihood[c] is highest.

Some Variants of Naive Bayes

Multinomial Naive Bayes

- This is the one we have studied!
- It assumes features are frequencies.
- Hence, it works on text that has been Count Vectorized.
- (In practice, it also works on TF-IDF scores.)

Bernoulli Naive Bayes

- This assumes features are binary.
- Hence, it works on text that has been Count Vectorized with frequencies capped at 1.

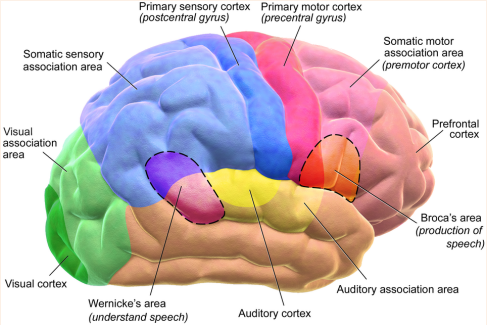
Gaussian Naive Bayes

- This works on tabular datasets whose features are all numeric.
- But it does assume that the feature values within each class follow a Gaussian (Normal) distribution — a bell curve.
- E.g. features such as a person's height and weight might follow this distribution.
- In practice, it is robust enough to perform well even if the distributions are slightly skewed.
- We might make features more Gaussian-like by transforming them using, e.g. log, square root.

15. Neural Networks

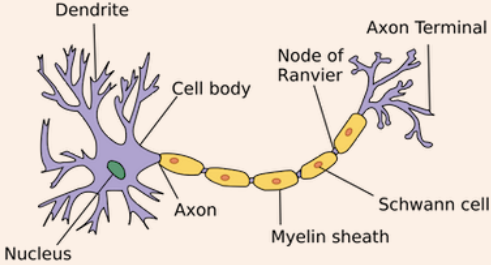
In this lecture, we use three datasets that we have used previously: housing (for regression), CS1109 (for binary classification) and Iris (for multiclass classification).

Brains



Your **brain** is a network of about 10^{11} **neurons**, each connected to about 10^4 others.

Neurons

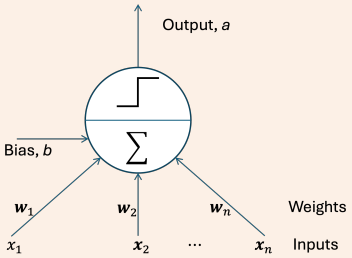


Sufficient electrical activity on a neuron's dendrites causes an electrical pulse to be sent down the axon, where it may activate other neurons.

Neural Networks

- **Neural Networks** are loosely inspired by what we know about our brains.
- However, they are *not* models of our brains — too many simplifications.

Artificial Neurons



- An artificial neuron has n real-valued **inputs**, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$
- The connections have real-valued **weights**, $w_1, \ldots, w_n$
- The neuron also has a number b called the **bias**.
- The neuron computes the weighted sum of its inputs and adds b:

$$z = b + \boldsymbol{w}_1\boldsymbol{x}_1 + \boldsymbol{w}_2\boldsymbol{x}_2 + \cdots + \boldsymbol{w}_n\boldsymbol{x}_n$$

- The neuron then applies an activation function, g, to the weighted sum to produce output a:

$$a = g(z)$$

Activation Functions

- **linear activation function**:

$$g(z) = z$$

- **step activation function**:

$$g(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

- **sigmoid activation function**:

$$g(z) = \frac{1}{1 + e^{-z}}$$

- **ReLU activation function**
(ReLU stands for Rectified Linear Unit):

$$g(z) = \max(0, z)$$

- **tanh activation function**
(tanh is the hyperbolic tangent):

$$g(z) = \tanh(z)$$

Questions

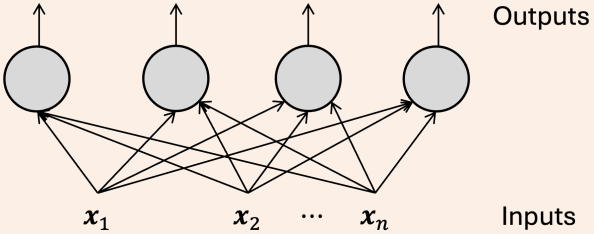
- If we create a model that comprises a single neuron with the linear activation function, then what kind of model do we have?
- If we create a model that comprises a single neuron with the sigmoid activation function, then what kind of model do we have?

Non-Linearity

- Apart from the linear activation function, these activation functions are non-linear, which is important to the power of neural networks.

Dense Layers

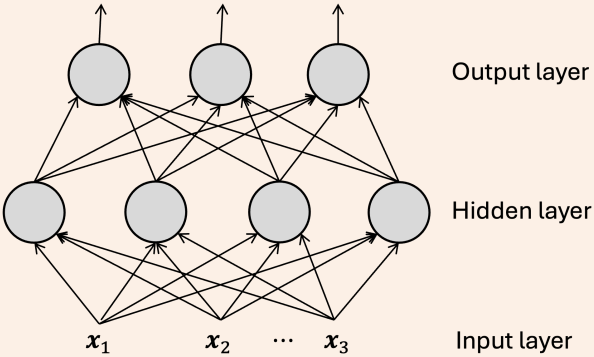
- We don't usually have just one neuron. We have a **layer**, containing several neurons.
- Below is a **dense layer** (also called a **fully-connected layer**) — every input is connected to every neuron in the layer.



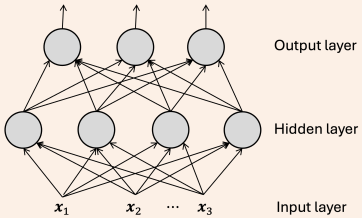
- So now we have more than one output, one per neuron. But each is calculated in the same way as before: compute a weighted sum of the inputs; apply the activation function to the weighted sum.

Dense Layers

- We don't usually have just one layer.
- Let's assume they are also **dense layers**, e.g.:



- The outputs of one layer are the inputs of the next layer.



These neural networks contain:

- an **input layer** (although this is not a layer of neurons);
- one or more **hidden layers**;
- an **output layer**.

Depth

- The depth of a multi-layer neural network is simply the number of layers of *neurons*.
- Question: What is the depth of the network in the diagram?

Deep Networks

- The word 'deep' in **deep network** does not mean profound.
- In deep networks, we have 'lots' of layers — tens or even hundreds.
- **Deep learning** means training a deep network.

Learning

- Brains learn by strengthening or weakening the connections between biological neurons.
- Analogously, neural networks learn by modifying the values of the parameters (weights and biases).

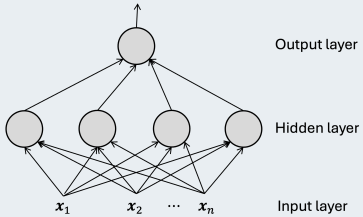
Parametric Learning

- A neural network is just a function — a parameterized function, where the parameters are the weights and biases of the neurons.
- We must choose the network architecture — this is not learned.

Supervised Learning

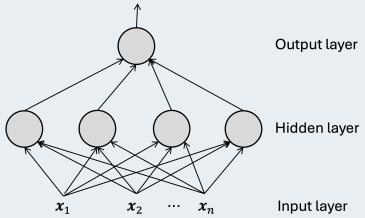
- We must obtain a labelled dataset.
- We must choose the values of numerous hyperparameters (perhaps using, e.g., grid search).
- We must choose a loss function (mean-squared error, binary cross-entropy or categorical cross-entropy, for example).
- Then we run a variant of Gradient Descent, knonw as **backpropagation**, to find the values of the network's parameters.

A Neural Network for Regression



- Input layer: one input per feature.
- Hidden layers: one or more hidden layers.
- Activation function for neurons in hidden layers can be the sigmoid function or ReLU.
- Output layer: just one output neuron.
- Activation function for the output neuron should be the linear function.

A Neural Network for Binary Classification

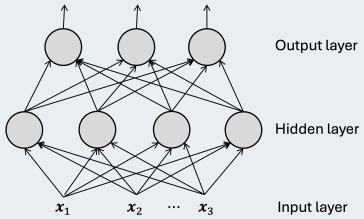


- Input layer: one input per feature.
- Hidden layers: one or more hidden layers.
- Activation function for neurons in hidden layers can be the sigmoid function or ReLU.
- Output layer: just one output neuron.
- Activation function for the output neuron should be the sigmoid function.

Question

Suppose the network outputs 0.7, what will this mean? (Hint: same as Logistic Regression.)

A Neural Network for Multi-Class Classification



(Here we're assuming three classes.)

- Input layer: one input per feature.
- Hidden layers: one or more hidden layers.
- Activation function for neurons in hidden layers can be the sigmoid function or ReLU.
- Output layer: one output neuron per class.
- Activation function for the output neurons should be the softmax function.

Questions

- Why softmax?
- Suppose the network outputs $\langle 0.2, 0.5, 0.3 \rangle$, what will this mean? (Hint: same as Multinomial Logistic Regression.)

Model Complexity

- More layers —and more neurons in each layer— increase model complexity.
- Obvious — because it increases the number of parameters (weights and biases).

Representation learning

- A neural network takes a vector of data into its input layer.
- The first layer in some sense transforms the input vector into a new vector — a different representation of the input vector.
- The second layer transforms the output of the previous layer into another new vector — another representation; and so on.
- So, learning in neural networks is about finding successive layers of representations.

Automated Feature Engineering

- In Feature Engineering, we used our ingenuity to derive new features from existing features.
- In a neural network, a neuron combines the outputs of previous neurons — a bit like computing a new feature from existing features.
- So, learning in neural networks automates feature engineering — however, these new features are not easily interpretable.

Image credits

	Taken from Wikipedia
	Taken from Wikipedia

16. Keras

In this lecture, we again work with three datasets that we used previously — housing, CS1109 and Iris. But we also work with a dataset of images — the MNIST dataset, which contains handwritten digits.

The Keras library

- scikit-learn has very limited support for neural networks.
- There are many software frameworks that do support tensor computation, neural neworks and deep learning.
- Python examples: TensorFlow, PyTorch and JAX.
- We will use **Keras**, which is a high-level API, first released in 2015 by François Chollet of Google (https://keras.io), which has done a lot to make Deep Learning accessible to people.

Advantages

- It is very high-level, making it easy to construct networks, fit models and make predictions.
- It is multi-backend, meaning it works atop of whichever framework you prefer (TensorFlow, PyTorch or JAX).

Disadvantage

- The downside is it gives less fine-grained control than the backend framework itself.
- But, when fine-grained control is needed, you can mix in functions, methods and classes from the backend.

Specifying a Neural Network Architecture

- Keras neural networks are built from **layers**.
- Consecutive layers must be compatible: the shape of the input to one layer is the shape of the output of the preceding layer.
- Question: how many neurons will we use in the output layer of a network for (a) regression, (b) binary classification and (c) multiclass classification?
- We can choose the activation function of each layer.
- Question: which activation function will we typically use in hidden layers?
- Question: which activation function will we use in the output layer of a network for (a) regression, (b) binary classification and (c) multiclass classification?

Compiling the Neural Network

- Once the network is specified, we compile it.
- We provide a loss function.
- We provide an optimizer.
- We provide a list of metrics to monitor during training and testing.
- Question: which loss function will we use for (a) regression, (b) binary classification and (c) multiclass classification?

Training and Inference

- After compiling, we can train (fit).
- Then we can do inference (make predictions), including error estimation (evaluation on the test set).

Gradient Descent for Neural Networks

- We train a neural network using **Gradient Descent**.
- The loss function for a neural network is not **convex** — there may be, e.g., **local minima**.
- We tend to us **Mini-Batch Gradient Descent** — its 'bounciness' might escape a local minimum.

Optimizer

- In Keras, the optimizer selects the rule for updating the weights and biases.

Simulated Annealing

- Instead of setting the learning rate to a numeric value, in Keras we can specify a **learning schedule**, which modifies the learning rate during training.

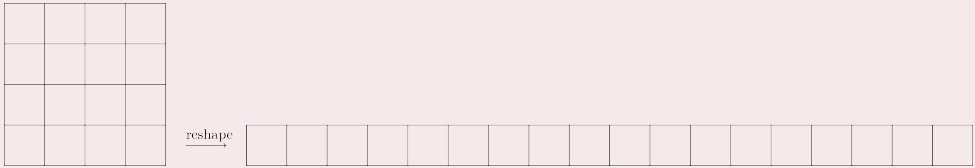
Plateau Detection

- Separately, we can create a class that, during training, decreases the learning rate if the best validation loss does not improve for several consecutive epochs.

MNIST Dataset

- A dataset created by the National Institute of Standards and Technology (NIST) in the USA.
- In the modified version of the dataset (MNIST):
  - 60,000 training examples, 10,000 test examples
  - Grayscale images of handwritten digits from employees of the American Census Bureau and from American high school students.
  - Each image is 28×28 pixels.

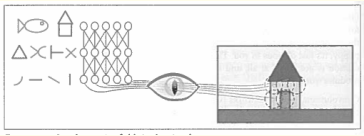
Reshaping the Images

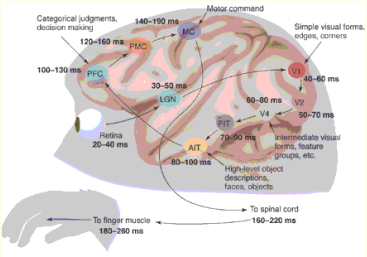


17. Convolutional Neural Networks

In this lecture, we use the MNIST dataset, which we saw in the previous lecture.

Primate Vision





- In the primate vision system, there seems to be a hierarchy of neurons within the visual cortex.
- In the lowest layers,
  - neurons have small local receptive fields, i.e. they respond to stimuli in a limited region of the visual field; and
  - they respond to, e.g., spots of light.
- In higher layers,
  - they combine the outputs of neurons in the lower layers;
  - they have larger receptive fields; and
  - they respond to, e.g., lines at particular orientations.
- In the highest layers,
  - they respond to ever more complex combinations, such as shapes and objects.
- There are perhaps as many as 8 layers in the visual cortex alone.

Convolutional Neural Networks

- **Convolutional Neural Networks** (**convnets** or **CNNs**) are widely used in computer vision and in other perceptual problems including speech recognition and natural language processing.
- We will use 2D convnets, which are widely used for datasets of images.
- They have nice properties, some of which resemble the visual cortex in primates.

The Properties of CNNs

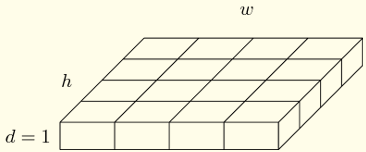
- They learn features that are **translation invariant**. In other words, a feature map in a convolutional layer will recognize that feature anywhere in the image: bottom-left, top-right, ...
- They learn **spatial hierarchies** of features: from small local features such as lines in lower layers up to larger shapes in higher layers.

Grayscale Images

- A **grayscale image** has a certain height h and width w.
- Therefore, it makes sense to represent each one as an h by w **matrix** of integers $[0, 255]$.
- However, up to now, we have reshaped them into **vectors**.

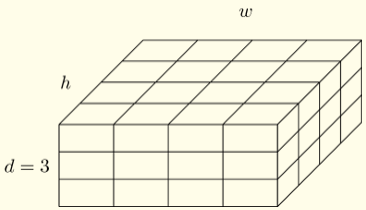
- Question: What is the disadvantage of this? What information gets destroyed?

- So, henceforth, we will not flatten them in this way.
- In fact, for consistency with colour images, we will treat grayscale images as three-dimensional objects with a height h, width w and depth d, but with $d = 1$.



Colour Images

- A **colour image** has a height h, width w and depth d, sometimes called the number of **channels**.
- Question: $d = 3$. Why?



Datasets of images

Datasets of images will have four dimensions: m, h, w, d. What is m?

Datasets of videos

Dataset of videos will have five dimensions. Why?

Tensors

Scalars

A quantity (a number), often referred to in this context as a **scalar**, has no dimensions.

Vectors

A **vector** has one dimension, n.

Matrices

A **matrix** has two dimensions, m and n.

Higher dimensions

We can also have objects that have three or more dimensions.

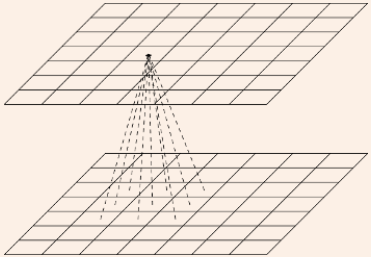
Tensors

We refer to all of these objects as **tensors** and we refer to the number of dimensions as the **rank** of the tensor.

- A scalar is a rank 0 tensor.
- A vector is a rank 1 tensor.
- A matrix is a rank 2 tensor.
- And we can have rank 3 tensors, rank 4 tensors, and so on.

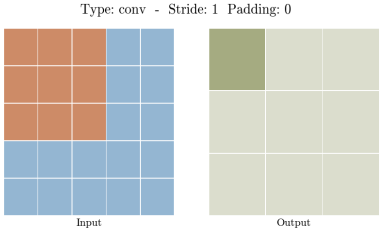
Convolutional Layers

- A **2D Convolutional Layer** is a rank 3 tensor of neurons, whose shape is (h, w, d), where d, the depth, is the number of **feature maps**.
- For simplicity to begin with, let's assume $d = 1$.
- Connections:
  - In the case of a *dense layer*, we saw that every neuron in that layer has connections from every neuron in the preceding layer.
  - But in the case of a *convolutional layer*, every neuron in that layer has connections from only a small rectangular **window** of neurons in the preceding layer, typically 3×3 or 5×5.



Height and Width of a Convolutional Layer

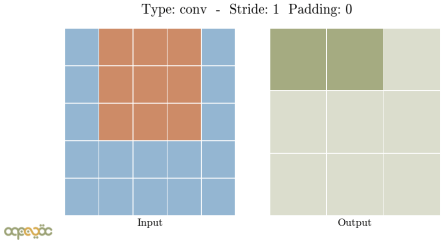
- Suppose the shape of the preceding layer is $(28, 28, 1)$ and the windows in the convolutional layer are (3×3).
- This gives a convolutional layer whose height is 26 and whose width is 26. Why?



- In fact, if we wish, we can make the convolutional layer have the same height and width as the preceding layer by using **padding**. Padding adds a border of zeros around the previous layer.
- And, if we wish, we can make the convolutional layer have even smaller height and width than the preceding layer by setting the **stride**. Instead of windows that overlap by 1, we can introduce a distance between successive windows.

Height and Width of a Convolutional Layer

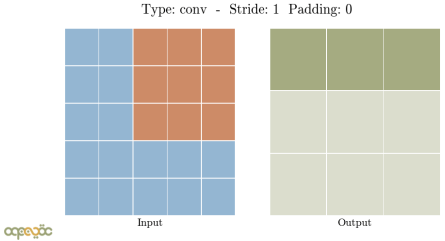
- Suppose the shape of the preceding layer is $(28, 28, 1)$ and the windows in the convolutional layer are (3×3).
- This gives a convolutional layer whose height is 26 and whose width is 26. Why?



- In fact, if we wish, we can make the convolutional layer have the same height and width as the preceding layer by using **padding**. Padding adds a border of zeros around the previous layer.
- And, if we wish, we can make the convolutional layer have even smaller height and width than the preceding layer by setting the **stride**. Instead of windows that overlap by 1, we can introduce a distance between successive windows.

Height and Width of a Convolutional Layer

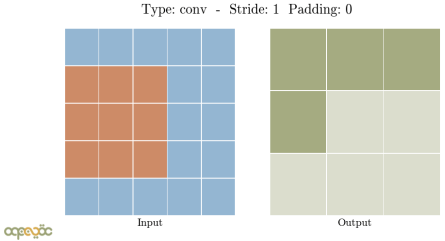
- Suppose the shape of the preceding layer is $(28, 28, 1)$ and the windows in the convolutional layer are (3×3).
- This gives a convolutional layer whose height is 26 and whose width is 26. Why?



- In fact, if we wish, we can make the convolutional layer have the same height and width as the preceding layer by using **padding**. Padding adds a border of zeros around the previous layer.
- And, if we wish, we can make the convolutional layer have even smaller height and width than the preceding layer by setting the **stride**. Instead of windows that overlap by 1, we can introduce a distance between successive windows.

Height and Width of a Convolutional Layer

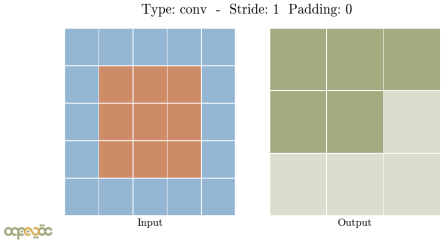
- Suppose the shape of the preceding layer is $(28, 28, 1)$ and the windows in the convolutional layer are (3×3).
- This gives a convolutional layer whose height is 26 and whose width is 26. Why?



- In fact, if we wish, we can make the convolutional layer have the same height and width as the preceding layer by using **padding**. Padding adds a border of zeros around the previous layer.
- And, if we wish, we can make the convolutional layer have even smaller height and width than the preceding layer by setting the **stride**. Instead of windows that overlap by 1, we can introduce a distance between successive windows.

Height and Width of a Convolutional Layer

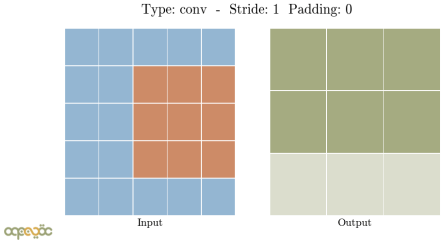
- Suppose the shape of the preceding layer is $(28, 28, 1)$ and the windows in the convolutional layer are (3×3).
- This gives a convolutional layer whose height is 26 and whose width is 26. Why?



- In fact, if we wish, we can make the convolutional layer have the same height and width as the preceding layer by using **padding**. Padding adds a border of zeros around the previous layer.
- And, if we wish, we can make the convolutional layer have even smaller height and width than the preceding layer by setting the **stride**. Instead of windows that overlap by 1, we can introduce a distance between successive windows.

Height and Width of a Convolutional Layer

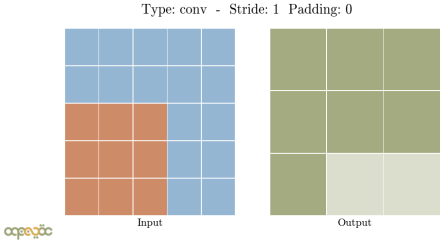
- Suppose the shape of the preceding layer is $(28, 28, 1)$ and the windows in the convolutional layer are (3×3).
- This gives a convolutional layer whose height is 26 and whose width is 26. Why?



- In fact, if we wish, we can make the convolutional layer have the same height and width as the preceding layer by using **padding**. Padding adds a border of zeros around the previous layer.
- And, if we wish, we can make the convolutional layer have even smaller height and width than the preceding layer by setting the **stride**. Instead of windows that overlap by 1, we can introduce a distance between successive windows.

Height and Width of a Convolutional Layer

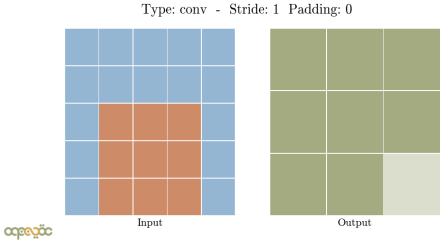
- Suppose the shape of the preceding layer is $(28, 28, 1)$ and the windows in the convolutional layer are (3×3).
- This gives a convolutional layer whose height is 26 and whose width is 26. Why?



- In fact, if we wish, we can make the convolutional layer have the same height and width as the preceding layer by using **padding**. Padding adds a border of zeros around the previous layer.
- And, if we wish, we can make the convolutional layer have even smaller height and width than the preceding layer by setting the **stride**. Instead of windows that overlap by 1, we can introduce a distance between successive windows.

Height and Width of a Convolutional Layer

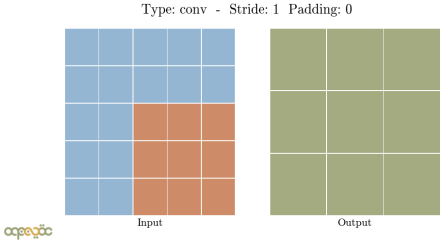
- Suppose the shape of the preceding layer is $(28, 28, 1)$ and the windows in the convolutional layer are (3×3).
- This gives a convolutional layer whose height is 26 and whose width is 26. Why?



- In fact, if we wish, we can make the convolutional layer have the same height and width as the preceding layer by using **padding**. Padding adds a border of zeros around the previous layer.
- And, if we wish, we can make the convolutional layer have even smaller height and width than the preceding layer by setting the **stride**. Instead of windows that overlap by 1, we can introduce a distance between successive windows.

Height and Width of a Convolutional Layer

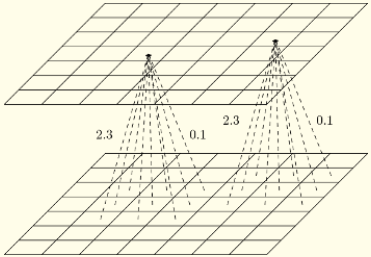
- Suppose the shape of the preceding layer is $(28, 28, 1)$ and the windows in the convolutional layer are (3×3).
- This gives a convolutional layer whose height is 26 and whose width is 26. Why?



- In fact, if we wish, we can make the convolutional layer have the same height and width as the preceding layer by using **padding**. Padding adds a border of zeros around the previous layer.
- And, if we wish, we can make the convolutional layer have even smaller height and width than the preceding layer by setting the **stride**. Instead of windows that overlap by 1, we can introduce a distance between successive windows.

The Weights of a Feature Map in a Convolutional Layer

- The idea of a feature map is that it will learn a specific aspect (feature) of its input:
  - e.g. the presence of a vertical line;
  - e.g. the presence of a pair of eyes.
- Within one feature map, all neurons share the same weights!

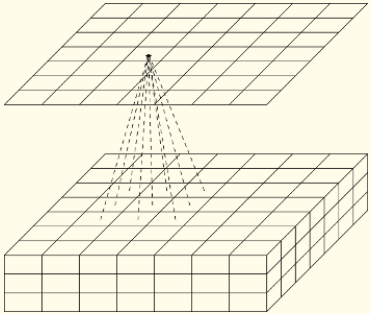


Advantages

- This reduces the number of parameters that must be learned.
- More importantly, it means that the feature map will respond to the presence of that feature no matter where it is in the input (the translational invariance, mentioned earlier).
- (You may see the word **filter** to refer to the weights of the neurons in a feature map.)

Convolutional Layers are Stacks of Feature Maps

- Now consider the case where $d > 1$: the convolutional layer comprises a stack of d feature maps.
- A neuron in a feature map in a convolutional layer is connected to a window of neurons in each of the feature maps of the previous layer (or, in the case of the first layer, in each of the channels of the input).
- Note how this means that a feature map in one layer combines several feature maps (or channels) of the previous layer (the spatial hierarchy, mentioned earlier).

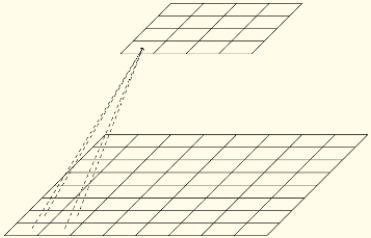


Pooling Layers

- Convolutional Layers are often followed by **Pooling Layers**.
- The goal is to have a layer that shrinks the number of neurons in higher layers:
  - to reduce the amount of computation;
  - to reduce memory usage;
  - to reduce the number of parameters to be learned, thus reducing the risk of overfitting; and
  - to create a hierarchy in which higher convolutional layers contain information about the totality of the original input image.

Height and Width of a Pooling Layer

- Like Convolutional Layers, a Pooling Layer works on rectangular windows: neurons in the pooling layer are connected to windows of neurone in the previous layer
  - typically 2×2;
  - typically adjacent rather than overlapping.
- E.g. if the previous layer has height h and width w, and the pooling layer uses adjacent 2×2 pooling windows, then the pooling layer will have height $h/2$ and width $w/2$.
- The depth of the pooling layer is the same as the depth of the previous layer.



Max Pooling Layers

- Pooling layers have no weights: nothing to learn.
- In a **Max Pooling Layer**, a neuron in the pooling layer receives the outputs of the neurons in the window in the previous layer and outputs only the *largest* of them.
- Pooling layers work on the feature maps independently, which is why they have the same depth as the previous layer.

Example

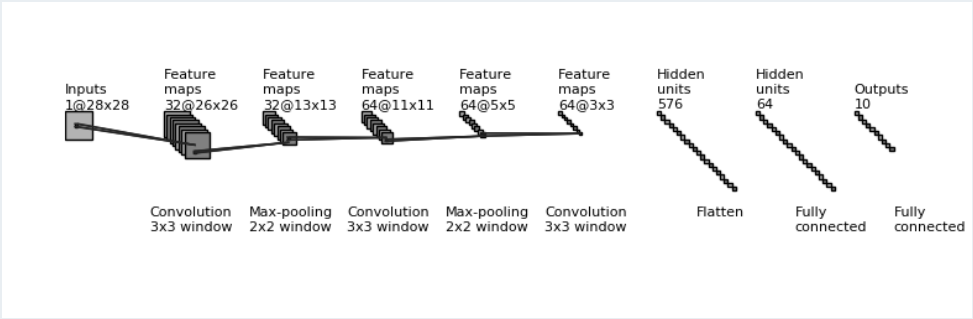

Inputs
1@28x28
Feature maps
32@26x26
Feature maps
32@13x13
Feature maps
64@11x11
Feature maps
64@5x5
Feature maps
64@3x3
Hidden units
576
Hidden units
64
Outputs
10
Convolution
3x3 window
Max-pooling
2x2 window
Convolution
3x3 window
Max-pooling
2x2 window
Convolution
3x3 window
Flatten
Fully connected
Fully connected

Image credits

Diagram taken from A. Géron: Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow (2nd edn), O'Reilly, 2019

Image from Simon J. Thorpe & Michèle Fabre-Thorpe: Seeking Categories in the Brain. Science, vol.291, pp.260–263 (2001).

Image from Machine Learning Animations

18. Training Neural Networks

In this lecture, we use MNIST again, but briefly. Then, the main part of the lecture uses a dataset of images of cats and dogs. It comes from Microsoft researchers, for a Kaggle competition: https://www.kaggle.com/c/dogs-vs-cats.. It contains 12,500 medium-resolution JPEGs depicting cats and 12,500 depicting dogs. We use a subset of the full dataset.

Gradient Descent for Neural Networks

- At the output layer, we can straightforwardly compute loss: we can get the network's output (its prediction) and we have the target value, because the training set is a labeled dataset.
- But we cannot straightforwardly compute loss at the hidden layers: we know what outputs their neurons produce but we do not know what they should produce (target values).
- The solution is to assume that each neuron in layer l is responsible for some part of the loss of the neurons it is connected to in layer $l + 1$.

The Backpropagation Algorithm

- The **Backpropagation algorithm** repeatedly makes two passes through the network:
  - a **forward pass** to make predictions: we feed training examples into the network and then work forwards through the network, computing activations layer by layer until we reach the output layer;
  - a **backward pass** to compute the gradients: we calculate error at the output layer and then work backwards computing shares of the error layer by layer until we reach the input layer.
- With these two steps completed, we have all the gradients, so we can update all the weights and biases.

Demos

- http://experiments.mostafa.io/public/ffbpann/
- https://mlu-explain.github.io/neural-networks/

The Vanishing Gradient Problem

- Each weight is updated by an amount proportional to the gradient of the loss function with respect to that weight.
- But if the gradient is very small, the weight doesn't change much.
- This may prevent the network from converging to a good approximation of the target function.
- This is worse for deeper networks. The shares of the error become ever smaller as they are backpropagated.

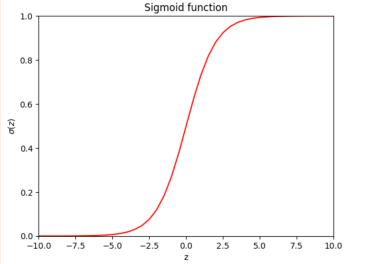
Non-saturating activation functions

Better initialization

Batch normalization

Saturating Activation Functions

- Certain activation functions, including sigmoid, are one cause of the vanishing gradient problem.
- When the input to this function becomes large (positive or negative), the function **saturates** (i.e. becomes very flat).
- When it saturates, its derivative is extremely close to 0 so there's not much gradient to propagate back to earlier layers.
- Even when the gradient is at its greatest (when input z is 0 and $\sigma(z) = 0.5$), it is only 0.25. So in the back propagation, gradients always diminish by a quarter or more.
- This is why we rarely use the sigmoid function as the activation function in the hidden layers of deep networks.

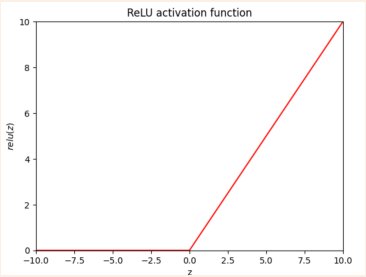


Non-Saturating Activation Functions

- Alternatives to sigmoid have been proposed, including the **rectified linear unit activation function**, ReLU

$$ReLU(z) = \max(0, z)$$

- Neurons that use ReLU have obvious problems too:
  - If their input (weighted sum) is negative, the output is zero and the gradient is zero — and if this is true for all examples in the training set then, in effect, the neuron dies.
  - The gradient changes abruptly at $z = 0$, which can make Gradient Descent bounce around.
- Alternatives to ReLU such as **Leaky ReLU**, **Exponential Linear Unit** (ELU) and **Scaled ELU** have been proposed, having at least some non-zero gradient for negative inputs but they are slower to compute and they introduce further hyperparameters.

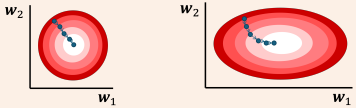


Better Random Initialization

- It turns out that vanishing gradients are more likely for certain ways of initializing the weights and biases.
- Perhaps the most typical method was to use a normal distribution with mean of 0 and standard deviation of, e.g., 0.05.
- Better methods have been proposed, e.g. **Glorot uniform initialization** (also called Xavier uniform initialization).

Batch Normalization

- We know feature scaling is useful for Gradient Descent.



- But, if this is a good idea for the inputs to the first hidden layer, why not use the same idea for the inputs to subsequent layers?
- In essence, in **Batch Normalization**, we normalize the activations (outputs) of layer l prior to them being used as inputs to layer $l + 1$.
- This will control the distribution of the values throughout the training process.

Other Benefits of Batch Normalization

- Batch Normalization reduces the vanishing gradients problem so much, we can even use saturating activation functions.
- Training becomes less sensitive to the method used for randomly initializing weights.
- Much larger learning rates work (faster convergence) with less risk of divergence.
- It acts like a regularizer.

Pretrained Convolutional Neural Networks

- A **pretrained network** is a saved network that was trained, usually on a large dataset.
- Remarkably, these are increasingly being made available for image classification, speech recognition and other tasks.
- Consider the ImageNet dataset (http://www.image-net.org/):
  - 1.4 million images, each manually labeled with one class per image;
  - thousands of classes, mostly animals and everyday objects;
  - annual competitions (ImageNet Large Scale Visual Recognition Challenges, ILSVRC), now hosted on Kaggle.
- There are several pretrained neural networks for ImageNet, half a dozen of which are made available directly in Keras, including
  - VGG16 and VGG19;
  - ResNet50 and ResNet101;
  - Inception V3.
- These networks incorporate a number of innovations, which we don't have time to study in depth, including local response normalization, skip connections, inception modules, and depthwise separable convolutional layers.

Transfer Learning

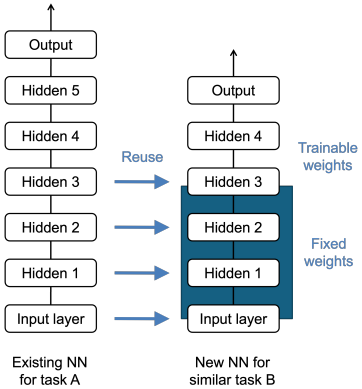
- In **Transfer Learning**, we take a model that was learned when solving one problem and re-use it for solving a different but related problem.
- Specifically,
  - take a pre-trained network;
  - re-use its lower layers, even freezing their weights.
- E.g. re-use the lower layers of ResNet in a new network that is trained to classify images of just cats and dogs, or different types of vehicles, or for face recognition, or perhaps even facial expression recognition.
- Of course, this will only work well if the original and new tasks share similar low-level features.

Advantages of Transfer Learning

- It speeds-up training for the new problem.
- It means that less training data may be needed for the new problem (see later).

Re-using the Convolutional Base of a Pretrained ConvNet

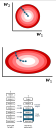
- Convolutional Neural Networks typically comprise two parts:
  - the **convolutional base**: the convolutional and pooling layers;
  - the densely-connected top layers for, e.g. classification.
- We want to reuse the convolutional base:
  - the features learned by these layers are likely to be more generic;
  - the features learned by the top layers will be more specific to the original task.



Transfer Learning Reduces Data Needs

- It's often said that we need lots of data for deep learning.
- But transfer learning helps us in cases where we have more limited data.
- E.g. we want to do face recognition but we have only a few pictures of each person, not enough to train a good classifier:
  - Collect loads of pictures of faces of random people from the web.
  - Train a network to detect whether or not two different pictures portray the same person.
  - This network is presumably a good feature detector for faces.
  - Use the lower layers of this network as the lower layers of your face recognition classifier.
- E.g. we want to do some natural language processing but we don't have a large dataset.
  - Collect loads of sentences from the web, label them 'good'.
  - Corrupt these sentences in various ways, and label them 'bad'.
  - Train a network to classify sentences and non-sentences ('good' vs. 'bad').
  - Use the lower layers of this network as the lower layers of your natural language processing system.

Image credits



I based these diagrams on ones to be found in A. Géron: Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow (2nd edn), O'Reilly, 2019

19. Overfitting in Neural Networks

In this lecture, we use the MNIST dataset again.

Overfitting

- One of the central problems of deep learning is overfitting.
- Reminder. If your model overfits, your main options are:
  - Gather more training examples;
  - Change to a less complex model;
  - Clean the training examples to remove noise.
  - Dimensionality reduction/feature selection: to reduce the number of features.
  - Parametric models: use regularization.
  - Decision Trees: use post-pruning.
  - Neural Networks: use dropout; use batch-normalisation.
  - Gradient Descent: use early-stopping.
  - Use an ensemble (although the relationship between ensembles and overfitting is not well-understood).

What we will look at

- reducing the network's size to give a less complex model;
- weight regularization;
- dropout;
- early stopping;
- data augmentation — creating synthetic training examples, rather than gathering more real training examples.

Reducing Network Size

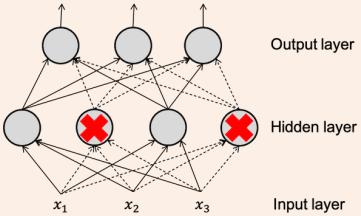
- We can make the model (neural network) less complex by reducing the number of parameters.
- Obviously enough, this is achieved by:
  - reducing the number of hidden layers, and/or
  - reducing the number of neurons within the hidden layers.

Weight Regularization

- For models that have parameters, we used regularization to ensure that the parameters took only small values by penalizing large values in the loss function.
  - Lasso: we penalized by the l_1-norm (the sum of their absolute values).
  - Ridge: we penalized by the l_2-norm (the sum of their squares).
  - A hyperparameter, λ, called the 'regularization parameter', controlled the balance between fitting the data versus shrinking the parameters.
- Weight Regularization in neural networks is the same idea, but applied to the weights in the layers of a network.
- Weight regularization can work well when the network is small but has little effect on larger networks.
- For larger networks, a better option can be dropout.

Dropout

- Imagine we have a layer that uses **dropout** with **dropout rate** p, e.g. $p = 0.5$.
- Then, in a given step of the backprop algorithm, each neuron in the layer has probability p of being ignored — treated as if it were not there.



One way of doing dropout

- Training. For any given mini-batch:
  - In the forward propagation,
    - decide which neurons will be dropped (chosen with probability p);
    - set the activations of the dropped neurons to zero;
    - multiply the activations of the kept neurons by $1/(1-p)$.
  - In the backpropagation, ignore the dropped out neurons.

  Note that different neurons will get dropped for each mini-batch.
- Inference. No change.

But why did we multiply activations by $1/(1-p)$?

- In inference, for $p = 0.5$, a neuron in the next layer will receive input from on average twice as many neurons as it did in training.
- The multiplication by $1/(1-p)$ compensates for this.

Why does dropout reduce overfitting?

- Consider a company whose employees were told to toss a coin every monring to decide whether to go to work or not.
- The organization would need to become more resilient. It could not rely on any one employee to perform critical tasks: the expertise would need to be spread across many employees. They must become more like generalists, less like specialists.
- Similarly, in dropout layers, neurons learn more robust features.

- Since a neuron can be present or absent, it's like training on a different neural network at each step.
- The final result is a bit like an ensemble of these many different virtual neural networks.
- However, it typically increases the number of epochs needed for convergence (roughly double when $p = 0.5$).

Early Stopping

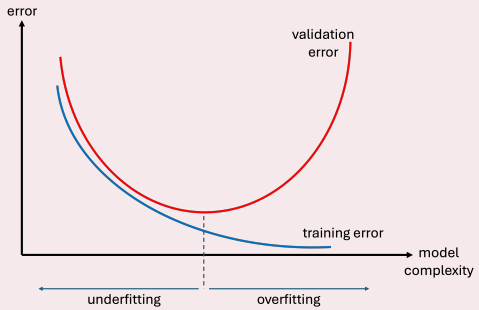
- We know that a sign of overfitting is that validation error stops getting lower and starts getting larger.
- We can exploit this during Gradient Descent as another way of avoiding overfitting, known as **early stopping**.
- During Gradient Descent, monitor validation error (or loss).
- Interrupt training when the validation error has stopped improving for a certain number of epochs.

Data Augmentation

- Another way to reduce overfitting is to get more examples but here we do something similar: **data augmentation**.
- We augment the training set with examples that we synthesize from the existing training set.
- It's relatively easy when the dataset consists of images to use transformations on existing images to synthesize believable-looking new images.
- Be aware that this is not as good as additional real examples. These synthesized examples are correlated with each other and the originals from which they were generated.
- Augmentation layers in a neural network will only augment the training data, not the validation or test data.

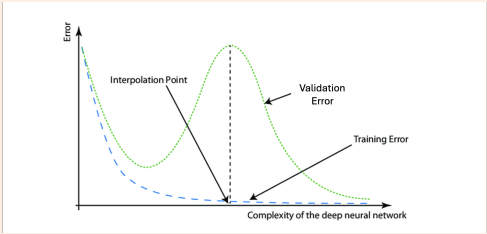
The Conventional Wisdom

- For low complexity models (e.g. small neural networks), training error and validation error are both high (underfitting).
- But as we increase complexity (e.g. more network layers), both decrease. But, after a certain point, validation error starts to rise again (overfitting).



Double-Descent

- In fact, with deep learning, we are finding something counter-intuitive. If we keep increasing the complexity, then validation error decreases again, falling to a new minimum. This is called **double descent**.



Over-Paramtereization and the Interpolation Threshold

- We are finding this happens with **over-parameterized** models — ones that are excessively complex.
- The current theory, in high-level terms, is that the over-parameterised model makes smoother interpolations between the training examples, and this is what gives the better validation error.

If you're interested

- A visualization
- An explanation

Image credits

Image based on one from A. Géron: Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow (2nd edn), O'Reilly, 2019. The analogy between dropout and a company whose employees are told to toss a coin to decide whether to go to work each morning comes from the same book.