Machine Learning & AI

Daniel T. McGuiness, PhD

Version: η.2024.3

MCI

B.Sc - Machine Learning and Data Science II

Table of Contents

Introduction

Table of Contents

Introduction

- The goal of this lecture is to give you the fundamentals of ML and understanding of mathematical and programming principles.
- This lecture is a total of 2 SWS with a total of sixty (30) hours.
- There is a written exam at the end of the lecture series.
- There is one (1) assignment for this course.

  1$^{\text{st}}$ will be a pre-defined work which is individual based.

Introduction

- The individual assignment focuses on understanding ML principles.
- The assignment is uploaded to SAKAI for you to work on along with what is required of you for submission.
  - The assignment contains questions where applications of ML will be needed.
- The deadline is the **last lecture before the examination**.

Introduction

- The final exam will be done on the last session where questions covering the entire lecture will be asked.
- The duration of the exam will be ninety (90) and will be done written.
- You are able to bring a calculator to the exam but no personal reference sheets are allowed.
- Any reference documents (if needed) will be provided for you during the beginning of the exam.

Assessment Type	Overall Points	Breakdown	%
Homework	40		
		Report	20
		Solution(s)	60
		Code Analysis	20
Final Examination	60		

Table 1: Assessment Grade breakdown for the lecture.

Introduction

Covered Topic	Appointment
Machine Learning Landscape	1
End-to-End Machine Learning Project	2
Classification	3
Training Models	4
Support Vector Machines	5
Decision Trees	5
Ensemble Learning and Random Forests	5
Dimensional Reduction	6
Unsupervised Learning	6
Introduction to Artificial Neural Networks	7

Table 2: Distribution of materials across the semester.

Machine Learning Landscape

- Covers the methods used in ML
  - Example applications
  - Types of ML Systems
  - Challenges of ML
  - Testing and Validations

End-to-End Machine Learning Project

- A ML project to work from beginning to end
    - Working with real data
    - Visualising the data
    - Select and train the data
    - Testing the model

Classification

- Focusing on how to work with data
  - MNIST,
  - Performance Measures,
  - Error Analysis
  - Multi-label Classification

Training Models

- Understanding how to get models to explain data
  - Linear Regression
  - Gradient Descent,
  - Polynomial Regression,
  - Logistic Regression.

Support Vector Machines

- Focusing on Vector machines
  - Linear & non-linear SVM
  - SVM Regression

Introduction

Decision Trees

- Focus on building decision trees
  - Training a decision tree
  - Making predictions
  - Estimating probabilities

Ensemble Learning and Random Forests

- Focusing on random forests
  - Voting Classifiers
  - Bagging and Pasting
  - Random Forests
  - Stacking

Dimensional Reduction

- Focusing on how to work with high-dimension data
  - Approaches to reduce dimensions,
  - Manifold learning,
  - PCA,
  - Kernel PCA.

Unsupervised Learning

- How to work with models with no clear training.
  - K-means,
  - DBSCAN,
  - Gaussian Mixtures.

Introduction to Artificial Neural Networks

- A deep dive into artificial neural networks (ANNs)
  - The Perceptron,
  - Backpropagation,
  - Regression multi-layer Perceptrons,
  - Classification multi-layer perceptrons.

Books

- Aggarwal S. "*Neural Networks and Deep Learning*" Springer, 2023.
- Raschka S., et. al "*Python Machine Learning*" Packt 2017.
- Geron A. "*Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow*" O'Reilly 2022.
- Albon C. "*Machine Learning with Python Cookbook*" O'Reilly 2018.

Introduction

Lecture Notes

- Ng A., et.al "*CS229 Lecture Notes*",
- Migel A., et.al "*Lecture Notes on Machine Learning*"

Web Resources

- Scikit-learn documentation
- OpenCV documentation
- Pillow (fork of PIL) documentation

Machine Learning Landscape

Table of Contents

Learning Outcomes

(LO1) An Introduction to ML,

(LO2) Overview of Learning Methods,

(LO3) Application of ML Algorithms,

(LO4) General Problems of ML Training/Evaluations.

Machine Learning Landscape

- Spam filters are a great place to start showing use of ML.
- A spam filter is a ML program where, given examples of spam emails (flagged by users) and examples of regular emails (non-spam, also called `ham`), can learn to flag spam.
- Examples the system uses to learn are called the training set.
- Each training example is called a training instance (i.e., sample).
- Part of the ML system learning and making predictions is called a model.
  - Neural networks and random forests are examples of models.

Figure 1: "Spam, Spam, Spam, Spam... Lovely Spam! Wonderful Spam!"

- Consider how you would write a spam filter using traditional method;

1. Define spam.
   - Some words or phrases (such as `4U`, `credit card`, `free`, `amazing`) tend to come up a lot in subject.
   - You also notice sender email also looks fishy.
2. Write an `if/else` statement for each case, and the program would flag emails as `spam` if a number of these patterns were detected.
3. Test program and release it to the public.
4. Profit!!

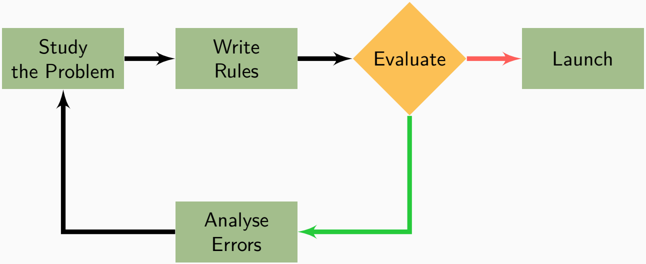


Figure 2: A block diagram on how to structure a spam filter using the traditional programming methods.

- As the problem is difficult, the traditional program will become a long list of complex rules.
  - This would be pretty hard to maintain.
- Whereas, a spam filter using ML automatically learns which words and phrases are good predictors of spam by detecting unusually frequent patterns of words in the spam examples compared to the ham examples [8].

Machine Learning Landscape

- As the problem is difficult, the traditional program will become a long list of complex rules.
  - This would be pretty hard to maintain.
- Whereas, a spam filter using ML automatically learns which words and phrases are good predictors of spam by detecting unusually frequent patterns of words in the spam examples compared to the ham examples [8].

Machine Learning Landscape

- As the problem is difficult, the traditional program will become a long list of complex rules.
  - This would be pretty hard to maintain.
- Whereas, a spam filter using ML automatically learns which words and phrases are good predictors of spam by detecting unusually frequent patterns of words in the spam examples compared to the ham examples [8].

- As the problem is difficult, the traditional program will become a long list of complex rules.
  - This would be pretty hard to maintain.
- Whereas, a spam filter using ML automatically learns which words and phrases are good predictors of spam by detecting unusually

The program is much shorter, easier to maintain, and most likely more accurate.

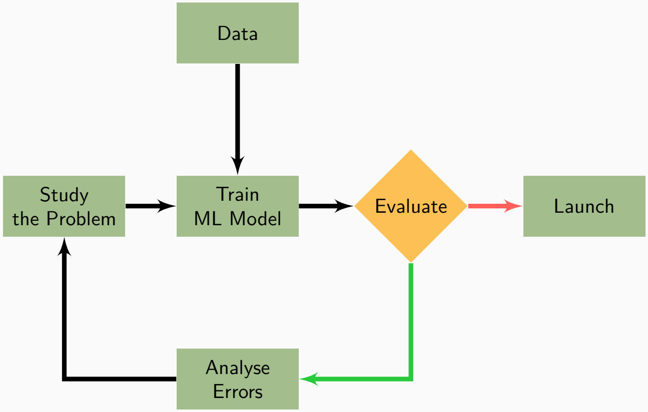


Figure 3: A block diagram on how to structure a spam filter using the ML approach.

- What if spammers notice that all their emails containing `4U` are blocked?
  - They might start writing `For U` instead.
- The traditional method needs to be updated to flag `For U` emails.
- If spammers keep working around your spam filter, you will need to keep writing new rules forever.

> A spam filter based on ML automatically notices that `For U` has become unusually frequent in spam flagged by users, and it starts flagging them without your intervention.

> ML works best for problems too complex for traditional approaches or have no known algorithm.

- Consider speech recognition [5]:
- Say you want to write a program capable of distinguishing the words `one` and `two`.
- You notice the word `two` starts with a high-pitch sound `T`.
- You could hard-code an algorithm measuring high-pitch sound intensity and use it to distinguish ones and twos.
  - This technique will not scale to thousands of words spoken by millions of very different people in noisy environments and in dozens of languages [2].
- The best solution is to write an algorithm that learns by itself.

Machine Learning Landscape

- ML can also help humans learn.
- ML models can be inspected to see what they have learned.
  - Although for some models this can be tricky.
    - such as black-box models.
- i.e., once a spam filter has been trained on enough spam, it can easily be inspected to reveal the list of words and combinations of words that it believes are the best predictors of spam.
- Sometimes this will reveal unsuspected correlations or new trends, and thereby lead to a better understanding of the problem.
- Digging into large amounts of data to discover hidden patterns is called data mining, and ML excels at it [12].

Machine Learning Landscape

- To summarise, ML is great for:
  - Problems where existing solutions require significant fine-tuning or long lists of rules.
  - Complex problems for which using a traditional approach gives no good solution.
  - Fluctuating environments.
  - Getting insights about complex problems and large amounts of data.

Machine Learning Landscape

- Analysing images on production line items classify [10]:
  - Called image classification, performed using convolutional neural networks or sometimes transformers.
- Detecting tumours in scans [11]:
  - Called semantic image segmentation, where each pixel in the image is classified, to determine the exact location and shape of tumours, typically using CNNs or transformers.
- Automatically classifying news articles [1]:
  - Called natural language processing (NLP), more specifically text classification, tackled using recurrent neural networks (RNNs) and CNNs, but transformers work even better.

Machine Learning Landscape

- Automatically flagging offensive comments on discussion forums [6]
  - Called text classification, using the same NLP tools.
- Summarising long documents automatically [15]:
  - Called text summarising, a sub-field of NLP.
- Creating a chat-bot or a personal assistant [17, 3]
  - Involves many NLP components, including:
    - Natural language understanding (NLU),
    - Question-Answering modules.

Machine Learning Landscape

- Forecasting a company's revenue next year [21], based on many metrics.
  - Called regression using any regression model, such as:
    - linear regression,
    - polynomial regression,
    - regression support vector machine,
    - regression random forest,
    - artificial neural network.
  - If you want to take into account sequences of past performance metrics, RNNs, CNNs, or transformers may prove useful [20].

Machine Learning Landscape

- Making your app react to voice commands.
  - Called speech recognition, requiring processing audio samples:
    - As they are long and complex sequences, they are typically processed using RNNs, CNNs, or transformers.
- Detecting credit card fraud:
  - Called anomaly detection, tackled using:
    - isolation forests,
    - Gaussian mixture models,
    - Auto-encoders.

Machine Learning Landscape

- Classifying clients on their purchases to design a marketing strategy based on class [13].
  - Called clustering, which can be achieved using k-means, DBSCAN, and more.
- Representing a complex, high-dimensional dataset in a clear and insightful diagram.
  - Called data visualisation, which involves dimensionality reduction techniques.

Machine Learning Landscape

- Recommending a product a client may be interested in, based on past purchases [16]:
  - Called a recommender system.
  - One approach is to feed past purchases to an ANN, and get it to output the most likely next purchase.
  - This ANN would typically be trained on past sequences of purchases across all clients.
- Building an intelligent bot for a game.
  - Tackled using reinforcement learning.
    - A branch of machine learning that trains agents to pick the actions that will maximise their rewards over time.
  - The famous AlphaGo program that beat the world champion at the game of Go was built using RL.

Figure 4: Deep Blue, computer chess-playing system designed by IBM in the early 1990s, playing against then current grand-master Garry Kasparov. It became the first computer winning against a world champion under tournament conditions [4].

Figure 5: AlphaGo was designed to play GO (a very complex game for computers to tackle) and was able to win a master which was deemed a milestone in ML.

Machine Learning Landscape

- There types of ML useful to classify data in broad categories:
  - Supervision during training:
    - Supervised,
    - Unsupervised,
    - Semi-supervised,
    - Self-supervised.
  - Whether or not they can learn incrementally on the fly:
    - Online v. Batch Learning,
  - Whether they work by simply comparing new data points to known data points, or instead by detecting patterns in the training data and building a predictive model:
    - Instance v. Model based learning.

Machine Learning Landscape

- There types of ML useful to classify data in broad categories:
  - Supervision during training:
    - Supervised,
    - Unsupervised,
    - Semi-supervised,
    - Self-supervised.
  - Whether or not they can learn incrementally on the fly:
    - Online v. Batch Learning,
  - Whether they work by simply comparing new data points to known data points, or instead by detecting patterns in the training data and building a predictive model:
    - Instance v. Model based learning.

Machine Learning Landscape

- There types of ML useful to classify data in broad categories:
  - Supervision during training:
    - Supervised,
    - Unsupervised,
    - Semi-supervised,
    - Self-supervised.
  - Whether or not they can learn incrementally on the fly:
    - Online v. Batch Learning,
  - Whether they work by simply comparing new data points to known data points, or instead by detecting patterns in the training data and building a predictive model:
    - Instance v. Model based learning.

Machine Learning Landscape

- There types of ML useful to classify data in broad categories:
  - Supervision during training:
    - Supervised,
    - Unsupervised,
    - Semi-supervised,
    - Self-supervised.
  - Whether or not they can learn incrementally on the fly:
    - Online v. Batch Learning,
  - Whether they work by simply comparing new data points to known data points, or instead by detecting patterns in the training data and building a predictive model:
    - Instance v. Model based learning.

Machine Learning Landscape

- There types of ML useful to classify data in broad categories:
  - Supervision during training:
    - Supervised,
    - Unsupervised,
    - Semi-supervised,
    - Self-supervised.
  - Whether or not they can learn incrementally on the fly:
    - Online v. Batch Learning,
  - Whether they work by simply comparing new data points to known data points, or instead by detecting patterns in the training data and building a predictive model:
    - Instance v. Model based learning.

Machine Learning Landscape

- There types of ML useful to classify data in broad categories:
  - Supervision during training:
    - Supervised,
    - Unsupervised,
    - Semi-supervised,
    - Self-supervised.
  - Whether or not they can learn incrementally on the fly:
    - Online v. Batch Learning,
  - Whether they work by simply comparing new data points to known data points, or instead by detecting patterns in the training data and building a predictive model:
    - Instance v. Model based learning.

Machine Learning Landscape

- There types of ML useful to classify data in broad categories:
  - Supervision during training:
    - Supervised,
    - Unsupervised,
    - Semi-supervised,
    - Self-supervised.
  - Whether or not they can learn incrementally on the fly:
    - Online v. Batch Learning,
  - Whether they work by simply comparing new data points to known data points, or instead by detecting patterns in the training data and building a predictive model:
    - Instance v. Model based learning.

- There types of ML useful to classify data in broad categories:
  - Supervision during training:
    - Supervised,
    - Unsupervised,
    - Semi-supervised,
    - Self-supervised.
  - Whether or not they can learn incrementally on the fly:
    - Online v. Batch Learning,
  - Whether they work by simply comparing new data points to known data points, or instead by detecting patterns in the training data and building a predictive model:
    - Instance v. Model based learning.

Machine Learning Landscape

- There types of ML useful to classify data in broad categories:
  - Supervision during training:
    - Supervised,
    - Unsupervised,
    - Semi-supervised,
    - Self-supervised.
  - Whether or not they can learn incrementally on the fly:
    - Online v. Batch Learning,
  - Whether they work by simply comparing new data points to known data points, or instead by detecting patterns in the training data and building a predictive model:
    - Instance v. Model based learning.

Machine Learning Landscape

- There types of ML useful to classify data in broad categories:
  - Supervision during training:
    - Supervised,
    - Unsupervised,
    - Semi-supervised,
    - Self-supervised.
  - Whether or not they can learn incrementally on the fly:
    - Online v. Batch Learning,
  - Whether they work by simply comparing new data points to known data points, or instead by detecting patterns in the training data and building a predictive model:
    - Instance v. Model based learning.

Machine Learning Landscape

- There types of ML useful to classify data in broad categories:
  - Supervision during training:
    - Supervised,

> These criteria are not exclusive. It is possible to combine them. For example, a state-of-the-art spam filter may learn on the fly using a DNN model trained using human-provided examples of spam and ham; this makes it an online, model-based, supervised learning system.

  - Whether they work by simply comparing new data points to known data points, or instead by detecting patterns in the training data and building a predictive model:
    - Instance v. Model based learning.

Machine Learning Landscape

- ML can be classified based on the amount and type of supervision they get during training.
- There are many categories, but we'll discuss the main ones:
  - supervised learning,
  - unsupervised learning,
  - self-supervised learning,
  - semi-supervised learning,
  - reinforcement learning.

Supervised learning

- The fed training set includes the desired solutions, called labels.
- A typical supervised learning task is classification.
- The spam filter is a good example of this:
  - Trained with many example emails along with their class, and it must learn how to classify new emails.
- Another typical task is to predict a target numeric value, such as the price of a car, given a set of features.
  - Called regression and to train it you need many examples of cars, including both their features and their targets.

Supervised learning

- The fed training set includes the desired solutions, called labels.
- A typical supervised learning task is classification.
- The spam filter is a good example of this:
  - Trained with many example emails along with their class, and it must learn how to classify new emails.
- Another typical task is to predict a target numeric value, such as the price of a car, given a set of features.
  - Called regression and to train it you need many examples of cars, including both their features and their targets.

Machine Learning Landscape

Supervised learning

- The fed training set includes the desired solutions, called labels.
- A typical supervised learning task is classification.
- The spam filter is a good example of this:
  - Trained with many example emails along with their class, and it must learn how to classify new emails.
- Another typical task is to predict a target numeric value, such as the price of a car, given a set of features.
  - Called regression and to train it you need many examples of cars, including both their features and their targets.

Supervised learning

- The fed training set includes the desired solutions, called labels.
- A typical supervised learning task is classification.
- The spam filter is a good example of this:
  - Trained with many example emails along with their class, and it must learn how to classify new emails.
- Another typical task is to predict a target numeric value, such as the price of a car, given a set of features.
  - Called regression and to train it you need many examples of cars, including both their features and their targets.

Supervised learning

- The fed training set includes the desired solutions, called labels.
- A typical supervised learning task is classification.
- The spam filter is a good example of this:
  - Trained with many example emails along with their class, and it must learn how to classify new emails.
- Another typical task is to predict a target numeric value, such as the price of a car, given a set of features.
  - Called regression and to train it you need many examples of cars, including both their features and their targets.

Supervised learning

- The fed training set includes the desired solutions, called labels.
- A typical supervised learning task is classification.
- The spam filter is a good example of this:
  - Trained with many example emails along with their class, and it must learn how to classify new emails.
- Another typical task is to predict a target numeric value, such as the price of a car, given a set of features.
  - Called regression and to train it you need many examples of cars, including both their features and their targets.

Supervised learning

- The fed training set includes the desired solutions, called labels.
- A typical supervised learning task is classification.

> Some regression models can be used for classification as well, and vice versa. i.e., logistic regression is commonly used for classification [14], as it can output a value that corresponds to the probability of belonging to a given class (e.g., 20% chance of being spam).

- Called regression and to train it you need many examples of cars, including both their features and their targets.

Supervised learning

- Target and label are generally treated as synonyms in supervised learning.
- But target is more common in regression tasks and label is more common in classification tasks.
- Features are sometimes called predictors or attributes.
- These terms may refer to individual samples, i.e.,

individual	this car's mileage feature is equal to 15,000,
all	the mileage feature is strongly correlated with price

Unsupervised learning

- Training data is unlabelled where it tries to learn without a teacher.
- For example say you have a lot of data about your blog's visitors.
  - Run a clustering algorithm to try to detect groups of similar visitors.
  - At no point algorithm knows which group a visitor belongs to,
    - it finds those connections without your help.
  - i.e., it notices % of visitors are teenagers who love comic books and read your blog after school while % are adults who enjoy sci-fi and who visit during the weekends.
  - If you use a hierarchical clustering algorithm it may also subdivide each group into smaller groups.
  - This may help you target your posts for each group.

Unsupervised learning

- Visualisation algorithms are also good examples.
- Feed them a lot of complex and unlabelled data and they output a 2D or 3D representation of your data that can easily be plotted.
- These algorithms try to preserve as much structure as they can.
  - Trying to keep separate clusters in the input space from overlapping in the visualisation.
- This would show how the data is organised and perhaps identify unsuspected patterns.

Unsupervised learning

- A related task is dimensionality reduction where the goal is to simplify the data without losing too much information.
- A way is to merge several correlated features into one.
- i.e., a car's mileage may be strongly correlated with its age so the dimensionality reduction algorithm will merge them into one feature that represents the car's wear and tear.
- This is called feature extraction.

Unsupervised learning

- A related task is dimensionality reduction where the goal is to simplify the data without losing too much information.
- A way is to merge several correlated features into one.
- i.e., a car's mileage may be strongly correlated with its age so the dimensionality reduction algorithm will merge them into one feature that represents the car's wear and tear.
- This is called feature extraction.

Unsupervised learning

- A related task is dimensionality reduction where the goal is to simplify the data without losing too much information.
- A way is to merge several correlated features into one.
- i.e., a car's mileage may be strongly correlated with its age so the dimensionality reduction algorithm will merge them into one feature that represents the car's wear and tear.
- This is called feature extraction.

Unsupervised learning

- A related task is dimensionality reduction where the goal is to simplify the data without losing too much information.
- A way is to merge several correlated features into one.
- i.e., a car's mileage may be strongly correlated with its age so the dimensionality reduction algorithm will merge them into one feature that represents the car's wear and tear.
- This is called feature extraction.

Unsupervised learning

- A related task is dimensionality reduction where the goal is to

It is a good idea to reduce the number of dimensions in your training data using a dimensionality reduction algorithm before feeding it to another ML algorithm (such as a supervised learning algorithm)

It will run much faster the data will take up less disk and memory space and in some cases it may also perform better [18].

Unsupervised learning

- Another task is anomaly detection.
  - Detecting unusual credit card transactions to prevent fraud,
  - Catching manufacturing defects,
  - Automatically removing outliers before feeding to another ML.

  Shown normal instances during training so it can recognise.

  When it sees a new instance it can tell whether it looks like a normal one or whether it is likely an anomaly.
- Another task is novelty detection.
  - Aims to detect new instances looking different from all instances in the training set.

> This requires having a very clean training set devoid of any instance that you would like the algorithm to detect.

Unsupervised learning

- For example if you have thousands of pictures of dogs:
  - % of these pictures are Chihuahuas then novelty detection should not treat new pictures of Chihuahuas as novelties.
  - Whereas anomaly detection may consider these dogs as so rare and so different from other dogs and classify them as anomalies.
- Finally another common unsupervised task is association rule learning digging into big data and discover interesting relations between attributes.
- For example suppose you own a supermarket.
  - Running association rule on logs reveal people who purchase barbecue sauce and potato chips also tend to buy steak,
  - Thus you may want to place these items close to one another.

Semi-supervised Learning

- Labelling data is usually time consuming and costly.
  - Often have plenty of unlabelled and few labelled instances.
- Some algorithms can deal with data that's partially labelled.
  - This is called semi-supervised learning [19].
- Some photo-hosting services are good examples of this.
  - Once images are uploaded it recognises `person A` appears on many photos and will categories them as a class.
  - This is the unsupervised part of the algorithm (clustering).
  - All the system needs is for you to tell it who these people are.

Semi-supervised Learning

- Labelling data is usually time consuming and costly.
  - Often have plenty of unlabelled and few labelled instances.
- Some algorithms can deal with data that's partially labelled.
  - This is called semi-supervised learning [19].
- Some photo-hosting services are good examples of this.
  - Once images are uploaded it recognises `person A` appears on many photos and will categories them as a class.
  - This is the unsupervised part of the algorithm (clustering).
  - All the system needs is for you to tell it who these people are.

Semi-supervised Learning

- Labelling data is usually time consuming and costly.
  - Often have plenty of unlabelled and few labelled instances.
- Some algorithms can deal with data that's partially labelled.
  - This is called semi-supervised learning [19].
- Some photo-hosting services are good examples of this.
  - Once images are uploaded it recognises `person A` appears on many photos and will categories them as a class.
  - This is the unsupervised part of the algorithm (clustering).
  - All the system needs is for you to tell it who these people are.

Semi-supervised Learning

- Labelling data is usually time consuming and costly.
  - Often have plenty of unlabelled and few labelled instances.
- Some algorithms can deal with data that's partially labelled.
  - This is called semi-supervised learning [19].
- Some photo-hosting services are good examples of this.
  - Once images are uploaded it recognises `person A` appears on many photos and will categories them as a class.
  - This is the unsupervised part of the algorithm (clustering).
  - All the system needs is for you to tell it who these people are.

Semi-supervised Learning

- Labelling data is usually time consuming and costly.
  - Often have plenty of unlabelled and few labelled instances.
- Some algorithms can deal with data that's partially labelled.
  - This is called semi-supervised learning [19].
- Some photo-hosting services are good examples of this.
  - Once images are uploaded it recognises `person A` appears on many photos and will categories them as a class.
  - This is the unsupervised part of the algorithm (clustering).
  - All the system needs is for you to tell it who these people are.

Semi-supervised Learning

- Labelling data is usually time consuming and costly.
  - Often have plenty of unlabelled and few labelled instances.
- Some algorithms can deal with data that's partially labelled.
  - This is called semi-supervised learning [19].
- Some photo-hosting services are good examples of this.
  - Once images are uploaded it recognises `person A` appears on many photos and will categories them as a class.
  - This is the unsupervised part of the algorithm (clustering).
  - All the system needs is for you to tell it who these people are.

Semi-supervised Learning

- Labelling data is usually time consuming and costly.
  - Often have plenty of unlabelled and few labelled instances.
- Some algorithms can deal with data that's partially labelled.
  - This is called semi-supervised learning [19].
- Some photo-hosting services are good examples of this.
  - Once images are uploaded it recognises `person A` appears on many photos and will categories them as a class.
  - This is the unsupervised part of the algorithm (clustering).
  - All the system needs is for you to tell it who these people are.

Semi-supervised Learning

- Labelling data is usually time consuming and costly.
  - Often have plenty of unlabelled and few labelled instances.
- Some algorithms can deal with data that's partially labelled.
  - This is called semi-supervised learning [19].
- Some photo-hosting services are good examples of this.
  - Once images are uploaded it recognises `person A` appears on many photos and will categories them as a class.
  - This is the unsupervised part of the algorithm (clustering).
  - All the system needs is for you to tell it who these people are.

Semi-supervised Learning

Most semi-supervised learning algorithms are combinations of unsupervised and supervised,

i.e., a clustering algorithm may be used to group similar instances together and then every unlabelled instance can be labelled with the most common label in its cluster,

Once the whole dataset is labelled it is possible to use any supervised learning algorithm.

Self-Supervised Learning

- Involves actually generating a fully labelled dataset from a fully unlabelled one.
- Once the whole dataset is labelled any supervised learning algorithm can be used.
- i.e., if you have a large dataset of unlabelled images you can randomly mask a small part of each image and then train a model to recover the original image.

Self-Supervised Learning

- During training the masked images are used as the inputs to the model and the original images are used as the labels.
- The resulting model may be quite useful in itself.
    - i.e., to repair damaged images or to erase unwanted objects from pictures
- Generally a model trained using self-supervised learning is not the final goal,
- You'll usually want to tweak and fine-tune the model for a slightly different task.
    - One that you actually care about.

Self-Supervised Learning

- For example suppose you need a pet classification model.
  - Given a picture of a pet it will tell you what species it belongs.
- If you have a dataset of unlabelled photos you can start by training an image repairing model using self-supervised learning.
- Once performing, it should be able to distinguish different pet species.
  - Repairing masked cat image it must know not to add a dog.
- If model's architecture allows, it is possible to tweak the model so that it predicts pet species instead of repairing images.
- Final is to fine-tune the model on a labelled dataset,
  - Model already knows what cats and dogs look like.
  - Only needed so the model can learn the mapping between the species it already knows and the labels we expect from it.

Self-Supervised Learning

- For example suppose you need a pet classification model.
  - Given a picture of a pet it will tell you what species it belongs.
- If you have a dataset of unlabelled photos you can start by training an image repairing model using self-supervised learning.
- Once performing, it should be able to distinguish different pet species.
  - Repairing masked cat image it must know not to add a dog.
- If model's architecture allows, it is possible to tweak the model so that it predicts pet species instead of repairing images.
- Final is to fine-tune the model on a labelled dataset,
  - Model already knows what cats and dogs look like.
  - Only needed so the model can learn the mapping between the species it already knows and the labels we expect from it.

Self-Supervised Learning

- For example suppose you need a pet classification model.
  - Given a picture of a pet it will tell you what species it belongs.
- If you have a dataset of unlabelled photos you can start by training an image repairing model using self-supervised learning.
- Once performing, it should be able to distinguish different pet species.
  - Repairing masked cat image it must know not to add a dog.
- If model's architecture allows, it is possible to tweak the model so that it predicts pet species instead of repairing images.
- Final is to fine-tune the model on a labelled dataset,
  - Model already knows what cats and dogs look like.
  - Only needed so the model can learn the mapping between the species it already knows and the labels we expect from it.

Self-Supervised Learning

- For example suppose you need a pet classification model.
  - Given a picture of a pet it will tell you what species it belongs.
- If you have a dataset of unlabelled photos you can start by training an image repairing model using self-supervised learning.
- Once performing, it should be able to distinguish different pet species.
  - Repairing masked cat image it must know not to add a dog.
- If model's architecture allows, it is possible to tweak the model so that it predicts pet species instead of repairing images.
- Final is to fine-tune the model on a labelled dataset,
  - Model already knows what cats and dogs look like.
  - Only needed so the model can learn the mapping between the species it already knows and the labels we expect from it.

Self-Supervised Learning

- For example suppose you need a pet classification model.
  - Given a picture of a pet it will tell you what species it belongs.
- If you have a dataset of unlabelled photos you can start by training an image repairing model using self-supervised learning.
- Once performing, it should be able to distinguish different pet species.
  - Repairing masked cat image it must know not to add a dog.
- If model's architecture allows, it is possible to tweak the model so that it predicts pet species instead of repairing images.
- Final is to fine-tune the model on a labelled dataset,
  - Model already knows what cats and dogs look like.
  - Only needed so the model can learn the mapping between the species it already knows and the labels we expect from it.

Self-Supervised Learning

- For example suppose you need a pet classification model.
  - Given a picture of a pet it will tell you what species it belongs.
- If you have a dataset of unlabelled photos you can start by training an image repairing model using self-supervised learning.
- Once performing, it should be able to distinguish different pet species.
  - Repairing masked cat image it must know not to add a dog.
- If model's architecture allows, it is possible to tweak the model so that it predicts pet species instead of repairing images.
- Final is to fine-tune the model on a labelled dataset,
  - Model already knows what cats and dogs look like.
  - Only needed so the model can learn the mapping between the species it already knows and the labels we expect from it.

Self-Supervised Learning

- For example suppose you need a pet classification model.
  - Given a picture of a pet it will tell you what species it belongs.
- If you have a dataset of unlabelled photos you can start by training an image repairing model using self-supervised learning.
- Once performing, it should be able to distinguish different pet species.
  - Repairing masked cat image it must know not to add a dog.
- If model's architecture allows, it is possible to tweak the model so that it predicts pet species instead of repairing images.
- Final is to fine-tune the model on a labelled dataset,
  - Model already knows what cats and dogs look like.
  - Only needed so the model can learn the mapping between the species it already knows and the labels we expect from it.

Self-Supervised Learning

- For example suppose you need a pet classification model.
  - Given a picture of a pet it will tell you what species it belongs.
- If you have a dataset of unlabelled photos you can start by training an image repairing model using self-supervised learning.
- Once performing, it should be able to distinguish different pet species.
  - Repairing masked cat image it must know not to add a dog.
- If model's architecture allows, it is possible to tweak the model so that it predicts pet species instead of repairing images.
- Final is to fine-tune the model on a labelled dataset,
  - Model already knows what cats and dogs look like.
  - Only needed so the model can learn the mapping between the species it already knows and the labels we expect from it.

Self-Supervised Learning

- For example suppose you need a pet classification model.
  - Given a picture of a pet it will tell you what species it belongs.
- If you have a dataset of unlabelled photos you can start by training an image repairing model using self-supervised learning.
- Once performing, it should be able to distinguish different pet species.
  - Repairing masked cat image it must know not to add a dog.
- If model's architecture allows, it is possible to tweak the model so that it predicts pet species instead of repairing images.
- Final is to fine-tune the model on a labelled dataset,
  - Model already knows what cats and dogs look like.
  - Only needed so the model can learn the mapping between the species it already knows and the labels we expect from it.

Self-Supervised Learning

- For example suppose you need a pet classification model.
  - Given a picture of a pet it will tell you what species it belongs.
- If you have a dataset of unlabelled photos you can start by training

> Transferring knowledge from one task to another is called **transfer learning** and it's one of the most important techniques in machine learning today especially when using deep neural networks

- If model's architecture allows, it is possible to tweak the model so that it predicts pet species instead of repairing images.
- Final is to fine-tune the model on a labelled dataset,
  - Model already knows what cats and dogs look like.
  - Only needed so the model can learn the mapping between the species it already knows and the labels we expect from it.

Self-Supervised Learning

- Some consider self-supervised learning to be a part of unsupervised learning as it deals with fully unlabelled datasets,
- But self-supervised learning uses generated labels during training so it's closer to supervised learning.
- And the term unsupervised learning is generally used when dealing with tasks like clustering dimensionality reduction or anomaly detection.
- whereas self-supervised learning focuses on the same tasks as supervised learning mainly classification and regression.
- In short it's best to treat self-supervised learning as its own category.

Reinforcement learning

- Reinforcement learning is a very different problem
- The learning method called an agent in this context
  - can observe the environment,
  - select and perform actions,
  - get rewards in return.
    - or penalties in the form of negative rewards
- It must then learn by itself what is the best strategy called a policy to get the most reward over time

Reinforcement learning

- A policy defines what action the agent should choose when given a situation.
- i.e., robots implement reinforcement learning to learn how to walk.
- AlphaGo is also a good example of reinforcement learning.
  - it made the headlines in when it beat Ke Jie the number one ranked player in the world at the time at the game of Go.
  - It learned its winning policy by analysing millions of games and then playing many games against itself.

> Learning was turned off during the games against the champion AlphaGo was just applying the policy it had learned.

- This is called offline learning.

Machine Learning Landscape

- Another criterion used to classify machine learning systems is whether or not the system can learn incrementally from a stream of incoming data.
- The methods are:
  - Batch learning,
  - Online learning.

Batch Learning

- The system is incapable of learning incrementally.
  - it must be trained using all the available data.
- This takes a lot of time and computing resources so it is typically done offline.
- First the system is trained and then it is launched into production and runs without learning anymore.
  - It just applies what it has learned.
- This is also called offline learning.

Batch Learning

- Unfortunately a model's performance tends to decay slowly over time as the world continues to evolve while the model remains unchanged.
  - This phenomenon is often called model rot or data drift.
- The solution is to regularly retrain the model on up-to-date data.
- How often you need to do that depends on the use case.
  - if model classifies pictures of cats and dogs performance will decay very slowly.
  - if model deals classifies financial market then it is likely to decay quite fast.

Batch Learning

- Unfortunately a model's performance tends to decay slowly over time as the world continues to evolve while the model remains unchanged.
  - This phenomenon is often called model rot or data drift.
- The solution is to regularly retrain the model on up-to-date data.
- How often you need to do that depends on the use case.
  - if model classifies pictures of cats and dogs performance will decay very slowly.
  - if model deals classifies financial market then it is likely to decay quite fast.

Batch Learning

- Unfortunately a model's performance tends to decay slowly over time as the world continues to evolve while the model remains unchanged.
  - This phenomenon is often called model rot or data drift.
- The solution is to regularly retrain the model on up-to-date data.
- How often you need to do that depends on the use case.
  - if model classifies pictures of cats and dogs performance will decay very slowly.
  - if model deals classifies financial market then it is likely to decay quite fast.

Batch Learning

- Unfortunately a model's performance tends to decay slowly over time as the world continues to evolve while the model remains unchanged.
  - This phenomenon is often called model rot or data drift.
- The solution is to regularly retrain the model on up-to-date data.
- How often you need to do that depends on the use case.
  - if model classifies pictures of cats and dogs performance will decay very slowly.
  - if model deals classifies financial market then it is likely to decay quite fast.

Batch Learning

- Unfortunately a model's performance tends to decay slowly over time as the world continues to evolve while the model remains unchanged.
  - This phenomenon is often called model rot or data drift.
- The solution is to regularly retrain the model on up-to-date data.
- How often you need to do that depends on the use case.
  - if model classifies pictures of cats and dogs performance will decay very slowly.
  - if model deals classifies financial market then it is likely to decay quite fast.

Batch Learning

- Unfortunately a model's performance tends to decay slowly over time as the world continues to evolve while the model remains unchanged.
  - This phenomenon is often called model rot or data drift.

Even a model trained to classify pictures of cats and dogs may need to be retrained regularly due to cameras keep changing along with image formats sharpness brightness and size ratios

decay very slowly.
  - if model deals classifies financial market then it is likely to decay quite fast.

Batch Learning

- If you want a batch learning to know about new data you need to train a new version of the system from scratch on the full dataset.
  - not just the new data but also the old data.
- Finally replacing the old model with the new one.
- Fortunately the whole process of training evaluating and launching a machine learning system can be automated fairly easily so even a batch learning system can adapt to change.

> Training using the full set of data can take many hours so you would typically train a new system only every few hours or even just weekly If your system needs to adapt to rapidly changing data then you need a more reactive solution.

Batch Learning

- Also training needs significant computing resources.
- If you have a lot of data and you automate your system to train from scratch every day it will end up costing you a lot of money.
- If the amount of data is huge it may even be impossible to use a batch learning algorithm.
- Finally if your system needs to be able to learn autonomously and it has limited resources then carrying around large amounts of training data and taking up a lot of resources to train for hours every day is a showstopper.
- A better option in all these cases is to use algorithms that are capable of learning incrementally.
    - Called online learning.

Online Learning

- Trains incrementally by feeding data instances sequentially.
  - Either individually or in small groups called mini batches.
- Each learning step is fast and cheap so the system can learn about new data on the fly as it arrives.
- Useful for systems needing to adapt to change extremely rapidly.
  - such as patterns in the stock market.
- It is also a good option if you have limited computing resources.
  - i.e., if the model is trained on a mobile device.
- Additionally online learning can be used to train models on huge datasets that cannot fit in one machine's main memory.
  - this is called out-of-core learning.
- The algorithm loads part of the data runs a training step on that data and repeats the process until it has run on all of the data.

Online Learning

- One important parameter of online learning systems is how fast they should adapt to changing data.
  - This is called the learning rate.
- Setting a high learning rate then your system will rapidly adapt to new data but it will also tend to quickly forget the old data.
- Setting a low learning rate the system will have more inertia.
  - It will learn more slowly but it will also be less sensitive to noise in the new data or to non-representative data points (outliers).

Online Learning

- A big problem is that if bad data is fed to the system the system's performance will decline possibly quickly.
    - For example, bad data could come from a bug.
    - It could come from someone trying to game the system.
- To reduce this risk, monitor the system closely and promptly switch learning off (and possibly revert) if you detect a drop in performance.
- You may also want to monitor the input data and react to abnormal data for example using an anomaly detection algorithm.

Machine Learning Landscape

- A way to categorise ML systems is by how they generalise.
- Most ML tasks are about making predictions.
  - This means that given a number of training examples the system needs to be able to make good predictions for examples it has never seen before.
- Having a good performance measure on the training data is good but insufficient.
  - The true goal is to perform well on new instances.
- There are two (2) main approaches to generalisation:
  1. Instance-based learning.
  2. Model-based learning.

Instance-Based Learning

- The system learns the examples by heart then generalises to new cases by using a similarity measure to compare them to the learned examples.
- For example, flagging emails that are identical to known spam emails very similar to known spam emails.
- This requires a measure of similarity between two emails.
- Similarity measure between two emails could be to count the number of words they have in common.

Model-Based Learning

- Generalise from a set of examples is to build a model of examples and then use that model to make predictions.

Machine Learning Landscape

- For example you want to know if money makes people happy.
- You look at the graph below.

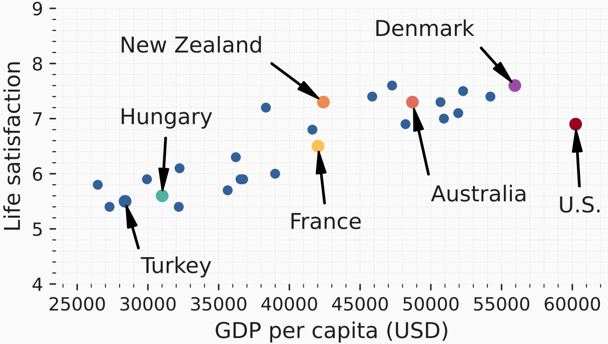


Figure 6: There seems to be something here.

- Looks like life satisfaction goes up more or less linearly as the country's GDP per capita.
- We can model life satisfaction as a linear function of GDP per capita.
- This step is called model selection you selected a linear model of life satisfaction with just one attribute, GDP per capita.

$$\text{Life Satisfaction} = \theta_0 + \theta_1 \text{GDP per Capita.}$$

- Before modelling, define the parameter values θ_0, θ_1.
- Which values will make your model perform best?

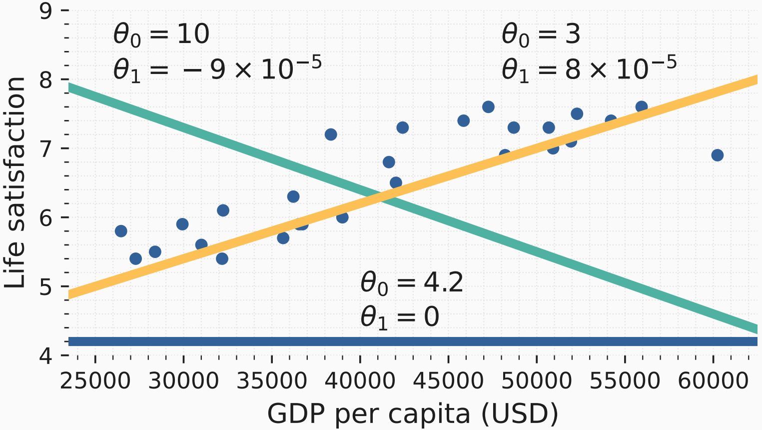


Figure 7: Possible linear models.

- To answer this question, specify a performance measure.
- Either define a utility function (or fitness function) measuring how good your model is or define a cost function measures how bad it is.
- For linear regression problems people typically use a cost function that measures the distance between the linear model's predictions and the training example.

> The goal is to minimise this distance.

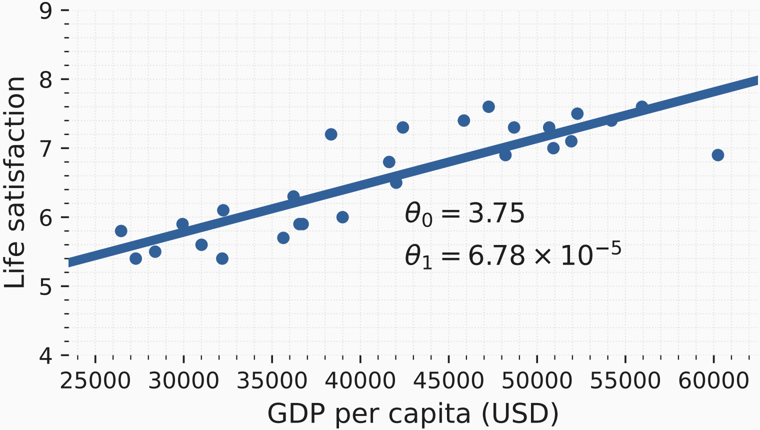


Figure 8: Best fit to the training set.

- As the main task is to select a model and train it on some data the two (2) things that can go wrong are:
  - bad model,
  - bad data.
- Let's start with examples of bad data.

Machine Learning Landscape

- For a toddler to learn an apple all it takes is for you to point to an apple and say “apple”.
- Now the child is able to recognise apples in all sorts of colours and shapes.
- ML is not quite there yet.
  - It takes a lot of data for most machine learning algorithms to work properly.
- For very simple problems you need thousands of examples.
- For complex problems such as image or speech recognition you may need millions of examples.

Machine Learning Landscape

- To generalise well, it is crucial the training data be representative of the new cases you want to generalise.
- This is true whether you use instance-based learning or model-based learning
- For example the set of countries earlier for training the linear model was not perfectly representative.
  - It did not contain any country with a GDP per capita lower than $ 23.500,00 or higher than 62.500,00 $

Machine Learning Landscape

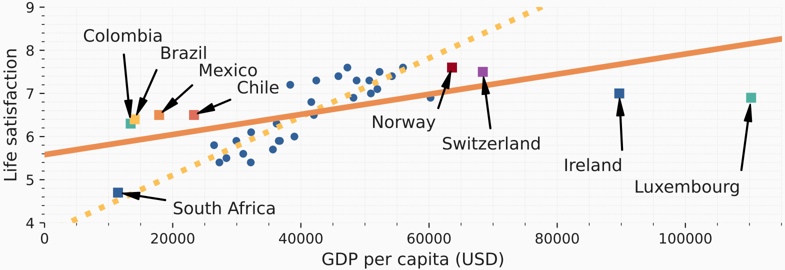


Figure 9: A more representative training sample.

- If you train a linear model on this data you get the solid line,
- while the old model is represented by the dotted line.
- Adding missing countries significantly alter the model and shows a simple linear model would not work well.

> By using a non-representative training set you trained a model that is unlikely to make accurate predictions especially for very poor and very rich countries.

- It is crucial to use a training set that is representative.
- If the method is flawed, it is called sampling bias.

Machine Learning Landscape

- If your training data is full of errors outliers and noise, it will make it harder for the system to detect the underlying patterns.
- It is worth cleaning up the training data.
  - If some instances are clearly outliers it may help to simply discard them or try to fix the errors manually
  - If some instances are missing a few features decide whether to:
    - ignore this attribute altogether,
    - ignore these instances,
    - fill in the missing values,
    - train one model with the feature and one model without it.

- System will only be capable of learning if the training data contains enough relevant features and not too many irrelevant ones.
- A critical part of the success of a ML project is coming up with a good set of features to train on.
- This process called feature engineering involves the following steps:
  1. **Feature selection** selecting the most useful features to train
  2. **Feature extraction** combining existing features to produce a more useful one
  3. **Creating new features** by gathering new data

- Say you are visiting a foreign country and the taxi driver rips you off.
- You might be tempted to say that all taxi drivers in that country are thieves.
- Overgeneralising is something that we humans do all too often. and unfortunately machines can fall into the same trap if we are not careful.
- In ML this is called over-fitting.
    - The model performs well on the training data but it does not generalise well.

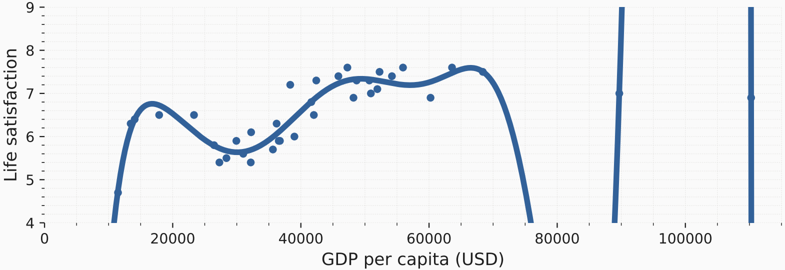


Figure 10: Overfitting the training data.

- The figure shows an example of a high-degree polynomial life satisfaction model that strongly over-fits the training data.
- Even though it performs much better on the training data than the simple linear model would you really trust its predictions?

- Complex models such as DNNs can detect subtle patterns.
  - If the training set is noisy or has sampling noise then the model is likely to detect patterns in the noise itself.
- Obviously these patterns will not generalise to new instances.
- For example feeding life satisfaction model with attributes.
  - including uninformative ones such as the country's name.
- A complex model may detect patterns like the fact that all countries in the training data with a w in their name have a life satisfaction greater than 7.
  - New Zealand (`7.3`)
  - Norway (`7.6`)
  - Sweden (`7.3`)
  - Switzerland (`7.5`)

- Constraining a model to make it simpler and reduce the risk of over-fitting is called regularisation.
- For example, the linear model has two parameters θ_0, θ_1.
- This gives the learning algorithm two degrees of freedom.
  - Forcing $\theta_1 = 0$ make the model have a line that can only go up or down.
  - Limiting θ_1 will keep the DoF between 1 and 2.
- The goal is to find the right balance between fitting the training data perfectly and keeping the model simple enough to ensure that it will generalise well.

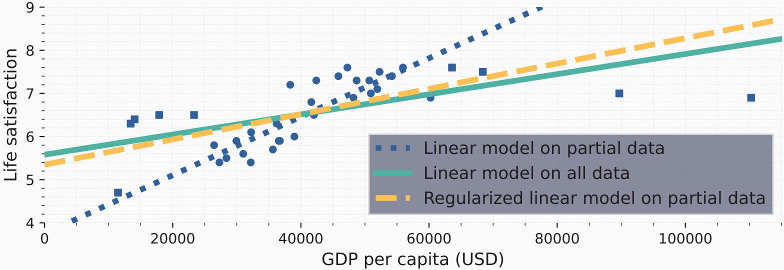


Figure 11: Regularisation reduces the risk of over-fitting.

- The amount of regularisation to apply during learning can be controlled by a hyper-parameter.
- A hyper-parameter is a parameter of a learning algorithm (not of the model).
- It is not affected by the learning algorithm itself.
- It must be set prior to training and remains constant during training.
- If you set the regularisation hyper-parameter to a very large value you will get an almost flat model (a slope close to zero) the learning algorithm will almost certainly not over-fit the training data but it will be less likely to find a good solution.

- Under-fitting is the opposite of over-fitting.
  - Occurs when your model is too simple to learn the underlying structure of the data.
- For example a linear model of life satisfaction is prone to under-fit.
  - Reality is just more complex than the model so its predictions are bound to be inaccurate even on the training examples.
- Here are the main options for fixing this problem [9]:
  - Select a more powerful model with more parameters,
  - Feed better features to the learning algorithm,
  - Reduce the constraints on the model.

- To know how well a model will generalise to new cases is to actually try it out on new cases.
- One way to do that is to put your model in production and monitor how well it performs.
- This works well but if the model is bad, users will complain.
- A better option is to split your data into two sets:
  - Training set,
  - Test set.
- Train the model using the training set and test it using the test set.
- The error rate on new cases is called the generalisation error.
- Evaluating the model on test set gives an estimate of this error.
- This value tells how well the model will perform on instances it has never seen before.
- If the training error is low, but the generalisation error is high it means that your model is over-fitting the training data.

Machine Learning Landscape

- Evaluate a model by using a test set.
- Suppose there is hesitation between two types of models.
  - linear v. poly.
  - How can you decide between them?
- An option is to train both and compare how well they generalise using the test set

- Suppose the linear model generalises better but you want to apply some regularisation to avoid over-fitting.
- How do you choose the value of the regularisation hyper-parameter?
- An option is to train different models using different values for this hyper-parameter.
- Suppose you find the best hyper-parameter value that produces a model with the lowest generalisation error.
- You launch this model into production but unfortunately it does not perform as well as expected and produces more errors.
- What just happened?

- The problem is the generalisation error was measured multiple times on the test set and adapted the model and hyper-parameters to produce the best model for that particular set.
- This means the model is unlikely to perform as well on new data.
- A solution to this problem is called holdout validation.
- Simply hold out part of the training set to evaluate several candidate models and select the best one
- The new held-out set is called the validation set.
  1. Train multiple models with various hyper-parameters on the reduced training set.
  2. Select the model that performs best on the validation set
  3. After this holdout validation process Train the best model on the full training set.

Machine Learning Landscape

- Having a large amount of data for training does not making better if the data does not represent the application.
- For example, a mobile app to take pictures of flowers and automatically determine their species.
- Download millions of pictures of flowers on the web but wont be representative pictures (i.e., actually taken with the app)
- The most important rule to remember is that both the validation set and the test set must be as representative as possible.

End to End ML Project

Table of Contents

End to End ML Project

Learning Outcomes

(LO1) Following a practical ML Project,

(LO2) Analysing Data,

(LO3) Training Models,

(LO4) Fine-tuning Models.

End to End ML Project

- Let's start learning ML by beginning with a project.
- We will conduct the following steps:
  1. See the big picture,
  2. Retrieve the data,
  3. Discover and visualise data,
  4. Prepare the data for analysis,
  5. Selecting a model an train it,
  6. Fine tuning the model,
  7. Present solution,

End to End ML Project

- Popular open data repositories:
  - UC Irvine Machine Learning Repository,
  - Kaggle datasets,
  - Amazon AWS datasets.
- Meta Portals:
  - Data Portals,
  - OpenDataMonitor,
  - Quandl,
- Other Sources:
  - Wikipedia's list of ML datasets.

- For this exercise we will work with California Housing Prices dataset from the StatLib repository.
- This is based on data from 1990 California census.

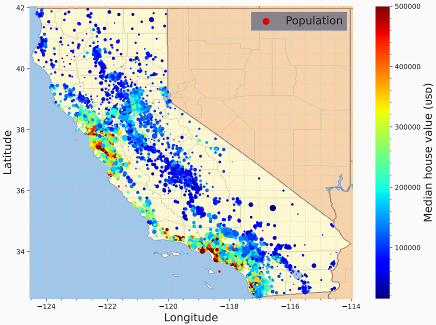


Figure 12: Housing density of the state of California.

End to End ML Project

- Our task will be to use the census data to build a model of housing prices within the state.
- The data we have includes numerous metrics such as:
  - population,
  - median income,
  - median housing price for each block group.

> Block group is a term to describe the smallest geographical unit US Census Bureau publishes data (typical range is 600-3000 people.)

End to End ML Project

- Our first task should be to define our goal.
- Building the model is not the endgame.

> We need to answer why we are building the model.

- You are told the model's output (a district's median housing price predictions) will be fed to another ML system, with other signals.
- This downstream system will determine whether it is worth investing in a given area of the problem or not.

> Getting this right is critical, as it directly affects revenue.

- The next question is what the current solution looks like (if any).

> The current situation will often give you a reference for performance, as well as insights on how to solve the problem.

- Your are answered that the district housing prices are currently estimated manually by experts.
  - A team gathers up-to-date information, and when there is no median housing price, they estimate it using complex rules.
- This is costly and time-consuming, and their estimates are not great.
- In cases of finding out the actual median housing price, they often realise that their estimates were off by more than 30%.

End to End ML Project

- First task is to determine the model of training.
- Let's analyse our tasks and data:
  - It is **supervised learning**, as we have data with labels
  - It is **regression task**, as we are trying to predict a value.
  - It is **batch learning**, as there is no continuous flow of data coming into the system, there is no particular need to adjust to changing data rapidly, and the data is small enough to fit in memory.

- First step is to select a performance measure.
- A standard in regression is Root Mean Square Error (RMSE).
    - It gives and idea on error a system makes in predictions.
        - This also gives a higher weight for large errors.

$$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2},$$

where m is the number of instances, $\mathbf{x}^{(i)}$ is a vector of all the feature values of the i^{th} instance in the datasets, and $y^{(i)}$ is its label.

- While RMSE is the preferred performance measure for regression tasks, in some you may prefer to use another.
- For example, suppose that there are many outlier districts.
- In that case, consider using the Mean Absolute Error (MAE).

> It also called the average absolute deviation.

$$\mathrm{MAE}(\mathbf{X}, h) = \frac{1}{m} \sum_{i=1}^{m} \left| h\left(\mathbf{x}^{(i)}\right) - y^{(i)} \right|$$

- Both RMSE and MAE are ways to measure the distance between two vectors: the vector of predictions and the vector of target values.

Generalised Performance measure

- RMSE corresponds to the Euclidean norm:
  - It is also called the ℓ_2 norm.
- Computing the sum of absolutes MAE corresponds to the ℓ_1 norm.
- This is sometimes called the Manhattan norm because it measures the distance between two points in a city if you can only travel along orthogonal city blocks [7].

Higher norm index focuses on large values and neglects small ones.

This is why the RMSE is more sensitive to outliers than the MAE. But when outliers are exponentially rare (like in a bell-shaped curve), the RMSE performs very well and is generally preferred.

- Our final pre-code checklist is to list and verify assumptions.
- Let's list and verify the assumptions that have been made so far.
- This can help you catch serious issues early on.
- For example, district prices your system outputs are going to be fed into a downstream machine learning system, and you assume that these prices are going to be used as such.

> What if downstream system converts the prices into categories (`cheap`, `medium`, or `expensive`) and then uses those categories instead of the prices themselves?

- In this case, getting the price perfectly right is not important at all.
- your system just needs to get the category right (i.e., classification).

- Below is our function called `load_housing_data` which downloads our data and turns it into a `pandas` Dataframe.

```
def load_housing_data():
    # path to save the file
    tarball_path = Path("datasets/housing.tgz")
    # check if the path exists, if not create one

    if not tarball_path.is_file():
        Path("datasets").mkdir(parents=True, exist_ok=True)
        url =
        ↪  "https://github.com/dTmC0945/L-MCI-BSc-Data-Science-II/raw/main/da
        urllib.request.urlretrieve(url, tarball_path)
        with tarfile.open(tarball_path) as housing_tarball:
            housing_tarball.extractall(path="datasets")
    return pd.read_csv(Path("datasets/housing/housing.csv"))
```

- With this function present load the data:

```
housing = load_housing_data()
```

- We can see the data below:

```
print(housing.head())
```

```
   longitude  latitude  ...  median_house_value  ocean_proximity
0    -122.23     37.88  ...            452600.0         NEAR BAY
1    -122.22     37.86  ...            358500.0         NEAR BAY
2    -122.24     37.85  ...            352100.0         NEAR BAY
3    -122.25     37.85  ...            341300.0         NEAR BAY
4    -122.25     37.85  ...            342200.0         NEAR BAY
[5 rows x 10 columns]
```

- Each row represents one district which have ten (10) attributes:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
None
```

- As seen previously, `.info()` method from `pandas` is useful to get quick description of data.

- There are `20,640` instances in the dataset.
- This is fairly small by ML standards, but it's perfect to get started.
- Notice that the `total_bedrooms` attribute has only `20,433` non-null values, meaning that `207` districts are missing this feature.

> We need to take care of this later.

End to End ML Project

- All attributes are numerical, except the `ocean_proximity` field.
- It's an object, so it could hold any kind of Python object.
- But since you loaded this data from a `.csv` file, it must text.
- When you looked at the top five rows, you probably noticed that the values in the `ocean_proximity` column were repetitive, which means it is probably a categorical attribute.
- You can find out what categories exist and how many districts belong to each category by using the `value_counts()` method:

```
print(housing["ocean_proximity"].value_counts())
```

- All attributes are numerical, except the `ocean_proximity` field.
- It's an object, so it could hold any kind of Python object.
- But since you loaded this data from a `.csv` file, it must text.

```
ocean_proximity
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: count, dtype: int64
```

```
print(housing["ocean_proximity"].value_counts())
```

- Have a look at other fields.
  - The `.describe()` method shows a summary of numerical methods.

```
print(housing.describe)
```

The null values (`NaN`) are ignored.

- Another method of understanding the data is to plot a histogram for each numerical attribute.
- Either plot one attribute at a time, or call `.hist()` method, and it will plot a histogram for each numerical attribute.

```
import matplotlib.pyplot as plt

housing.hist(bins=50,  figsize=(12, 8))

cp.store_fig("attribute-histogram-plots", close=True)
```

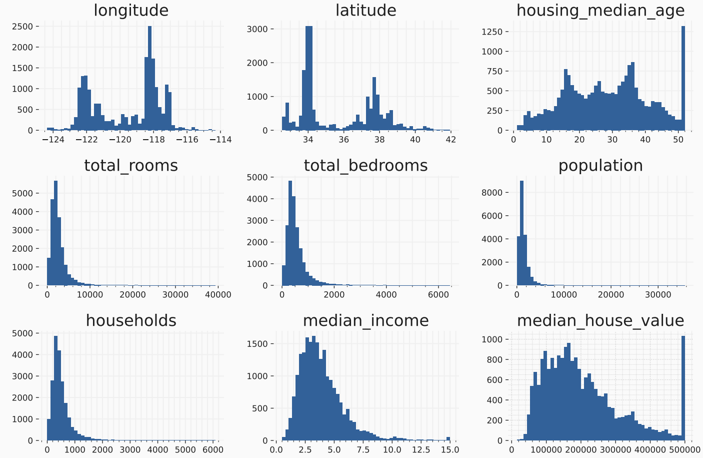


Figure 13: A histogram for each numerical attribute.

There are a two (2) things of interest:

1. The median income is not expressed in US dollars.
   - After checking with the team, you are told that the data has been scaled and capped at 15 (actually, 15.0001) for higher median incomes, and at 0.5 (actually, 0.4999) for lower median incomes.
   - The numbers represent roughly tens of thousands of dollars.
     - i.e., 3 means roughly 30,000 United States Dollar (USD).

> Working with preprocessed attributes is common in ML, and it is not necessarily a problem, but you should try to understand how the data was computed.

There are a few things of interest:

2. The housing median age and value were also capped.
   - value may be a serious problem as it is the target attribute.
   - The ML algorithm may learn prices never go beyond that limit.
   - You need to check with your client to see if this is a problem or not.

If they tell you that they need precise predictions even beyond 500,000 USD, you have two (2) options:

- Collect proper labels for the districts whose labels were capped.
- Remove those districts from the training set

- Creating a test set is theoretically simple; pick some instances randomly, typically 20% of the dataset, and set them aside:

```python
import numpy as np

# Define a function to shuffle and split data
def shuffle_and_split_data(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

- Afterwards use the function as follows:

```python
train_set, test_set = shuffle_and_split_data(housing, 0.2)
```

End to End ML Project

- While this works, it's not perfect.
- If you run the program again, it will generate a different test set.
- Over time, the ML algorithm will get to see the whole dataset, which is something to avoid.
- A solution is to save the test set on the first run and then load it in subsequent runs.
- Another option is to set the random number generator's seed.

```
np.random.seed(42)
```

- `sklearn` has few functions to split datasets into multiple subsets.
- The simplest function is `train_test_split()`, which does pretty much the same thing as the `shuffle_and_split_data()` function defined earlier, with a couple of additional features.
- First, there is a `random_state parameter` that allows you to set the random generator seed.
- Second, you can pass it multiple datasets with an identical number of rows, and it will split them on the same indices.

> This is very useful if you have a separate DataFrame for labels.

```python
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2,
↪ random_state=42)
```

End to End ML Project

- We taken a quick glance at the data to get a general understanding of the kind of data to manipulate.
- Now the goal is to go into a little more depth.
- First, put the test set aside to only explore the training set.
- If training set is large, sample an exploration set, to make manipulations easy and fast during the exploration phase.
- Now, the training set is quite small, so work directly on the full set.
- Since you're going to experiment with various transformations of the full training set, make a copy of the original so you can revert to it afterwards.

```
housing = strat_train_set.copy()
```

- As the dataset includes geographical information (latitude and longitude), it is a good idea to create a scatterplot of all the districts to visualise the data:

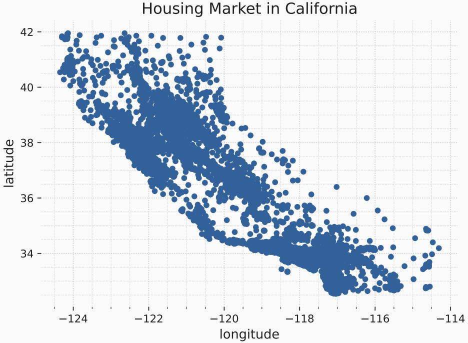


Figure 14: A geographical scatterplot of the data.

- It is hard to see any particular pattern.
- Setting the alpha option to `0.2` makes it much easier to visualise the places where there is a high density of data points.

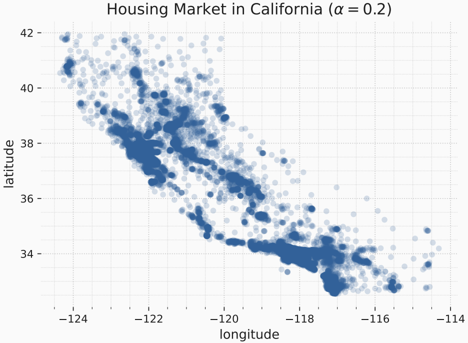


Figure 15: A better visualisation that highlights high-density areas.

- Next, look at the housing prices. The radius of each circle represents the district's population, and the color represents the price.

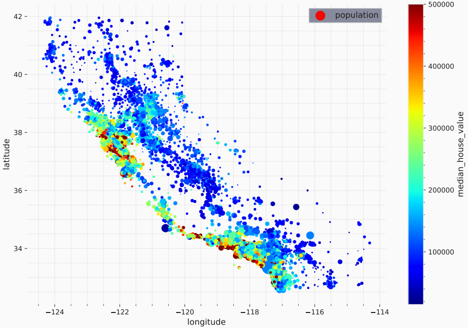


Figure 16: California housing prices: red is expensive, blue is cheap, larger circles indicate areas with a larger population.

- It tells you the housing prices are very much related to the location (e.g., close to the ocean) and to the population density,
  - This makes sense.
- A clustering algorithm should be useful for detecting the main cluster and for adding new features that measure the proximity to the cluster centres.

> The ocean proximity attribute may be useful as well, although in Northern California the housing prices in coastal districts are not too high, so it is not a simple rule.

- As the dataset is large, the standard correlation coefficient (i.e., Pearson's) between every pair of attributes using the `.corr()` method:

```
corr_matrix = housing.corr(numeric_only=True)
corr_matrix["median_house_value"].sort_values(ascending=False)
```

- As the dataset is large, the standard correlation coefficient (i.e., Pearson's) between every pair of attributes using the `.corr()` method:

```
median_house_value    1.000000
median_income         0.688380
total_rooms           0.137455
housing_median_age    0.102175
households            0.071426
total_bedrooms        0.054635
population           -0.020153
longitude            -0.050859
latitude             -0.139584
Name: median_house_value, dtype: float64
```

```
corr_matrix = housing.corr(numeric_only=True)
corr_matrix["median_house_value"].sort_values(ascending=False)
```

- The correlation coefficient ranges from `-1` to `1`.
- Close to `1` means a strong positive correlation.
- Close to `-1`, it means a strong negative correlation.
- Close to `0` mean no linear correlation.

- Another way to check is to use the Pandas `scatter_matrix()` function.
  - Plots every numerical attribute against every other numerical attribute.
- As there are 11 numerical attributes, you would get 121 plots.
- Focus on a few promising attributes most correlated with the median housing value.

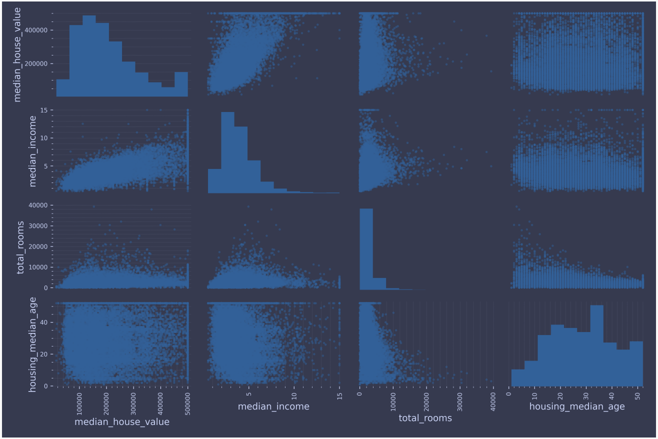
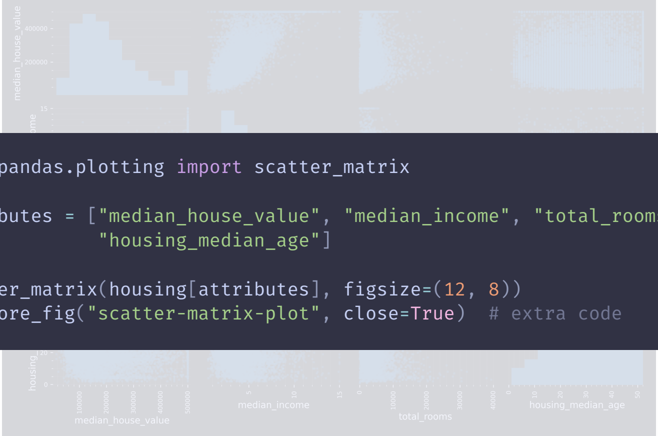


Figure 17: Correlation results of the `scatter_matrix()` function.




```python
from pandas.plotting import scatter_matrix

attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]

scatter_matrix(housing[attributes], figsize=(12, 8))
cp.store_fig("scatter-matrix-plot", close=True)  # extra code
```

Figure 17: Correlation results of the `scatter_matrix()` function.

- The Pandas displays a histogram of each attribute.
- Looking at the correlation scatterplots, it seems like the most promising attribute to predict the `median_house_value` is the `median_income`, so you zoom in on their scatterplot.

```
housing.plot(kind="scatter",
             x="median_income",
             y="median_house_value",
             alpha=0.1, grid=True)

cp.store_fig("income-vs-house-value-scatterplot", close=True)
```

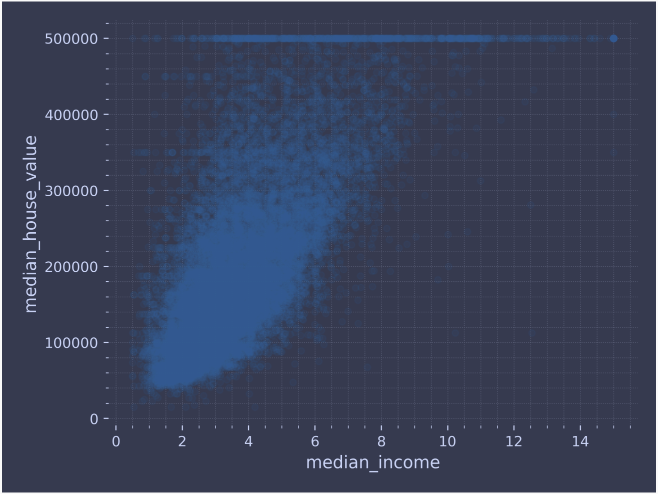


Figure 18: Median income versus median house value.

- This plot reveals a few things.
- The correlation is indeed quite strong;
  - see the upward trend, and the points are not too dispersed.
- Second, the price cap you noticed earlier is clearly visible as a horizontal line at 500,000 USD.
- But the plot also reveals other less obvious straight lines: a horizontal line around 450,000 USD, another around 350,000 USD, perhaps one around 280,000 USD, and a few more below that.

> You may want to try removing the corresponding districts to prevent your algorithms from learning to reproduce these data quirks.

End to End ML Project

- Time to prepare the data for your ML algorithms.
- Instead of doing this manually, write functions for this purpose, for several good reasons:
  1. Allows you to reproduce these transformations easily on any dataset.
  2. Gradually build a library of transformation functions you can reuse in future projects.
  3. Use these functions in your live system to transform the new data before feeding it to your algorithms.
  4. Makes it possible to easily try various transformations and see which combination of transformations works best.

- First revert to a clean training set.
- Also separate the predictors and the labels, as you don't necessarily want to apply the same transformations to the predictors and the target values.

> note that `.drop()` creates a copy of the data and does not affect `strat_train_set`.

```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

End to End ML Project

- Most ML algorithms cannot work with missing features, so we need to clean.
- For example, you noticed earlier the `total_bedrooms` attribute has some missing values.
- You have three options to fix this:
  1. Get rid of the corresponding districts,
  2. Get rid of the whole attribute,
  3. Set the missing value to some value (zero, mean, median)
     - This is called imputation.

- You can accomplish these easily using the Pandas DataFrame's methods:
    - `.dropna()`,
    - `.drop()`,
    - `.fillna()`,

```
housing_option1 = housing.copy()
housing_option1.dropna(subset=["total_bedrooms"], inplace=True)
housing_option1.loc[null_rows_idx].head()
```

```
housing_option2 = housing.copy()
housing_option2.drop("total_bedrooms", axis=1, inplace=True)
housing_option2.loc[null_rows_idx].head()
```

```
housing_option3 = housing.copy()
median = housing["total_bedrooms"].median()
housing_option3["total_bedrooms"].fillna(median, inplace=True)

housing_option3.loc[null_rows_idx].head()
```

- Go for option 3 since it is the least destructive.
- Instead of the preceding code, use `sklearn` class: `SimpleImputer`.
- The benefit is it will store the median value of each feature:
  - Makes it possible to impute missing values not only on the training set, but also on the validation set, the test set, and any new data fed to the model.
- To use it, first create a `SimpleImputer` instance, specifying to replace each attribute's missing values with the median of that attribute:

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")
```

- As the median can only be computed on numerical attributes, create a copy of the data with only numerical attributes.

> This will exclude the text attribute `ocean_proximity`.

```
housing_num = housing.select_dtypes(include=[np.number])
```

- Now fit the imputer instance to the training data using `.fit()` method:

```
imputer.fit(housing_num)
```

- The `imputer` computed the median of each attribute and stored the result in its `statistics_` instance variable.
- Only the `total_bedrooms` attribute had missing values.
- To make sure there are no missing values in new data after the system goes live, apply the imputer to all the numerical attributes:

```
print(imputer.statistics_)
```

```
print(housing_num.median().values)
```

- The `imputer` computed the median of each attribute and stored the result in its `statistics_` instance variable.
- Only the `total_bedrooms` attribute had missing values.
- To make sure there are no missing values in new data after the system goes live, apply the imputer to all the numerical attributes:

```
print(imputer.statistics_)
```

```
print(housing_num.median().values)
```

- The `imputer` computed the median of each attribute and stored the result in its `statistics_` instance variable.
- Only the `total_bedrooms` attribute had missing values.
- To make sure there are no missing values in new data after the system goes live, apply the imputer to all the numerical attributes:

```
[-118.51     34.26     29.      2125.      434.      1167.      408.
    3.5385]
```

```
print(housing_num.median().values)
```

- The `imputer` computed the median of each attribute and stored the result in its `statistics_` instance variable.
- Only the `total_bedrooms` attribute had missing values.
- To make sure there are no missing values in new data after the system goes live, apply the imputer to all the numerical attributes:

```
print(imputer.statistics_)
```

```
print(housing_num.median().values)
```

- The `imputer` computed the median of each attribute and stored the result in its `statistics_` instance variable.
- Only the `total_bedrooms` attribute had missing values.
- To make sure there are no missing values in new data after the system goes live, apply the imputer to all the numerical attributes:

```
[-118.51     34.26      29.       2125.       434.       1167.       408.
    3.5385]
```

```
print(housing_num.median().values)
```

- You can use this trained `imputer` to transform the training set by replacing missing values with the learned medians:

```
X = imputer.transform(housing_num)
# to see the name of the columns
print(imputer.feature_names_in_)
```

- Missing values can also be replaced with:
  - mean value `strategy="mean"`,
  - most frequent value `strategy="most_frequent"`,
  - a constant value `strategy="constant", fill_value=...`.

The last two strategies support non-numerical data.

End to End ML Project

- Till now we only dealt with numerical attributes, but data may also contain text attributes.
- In this dataset, there is just one: the `ocean_proximity` attribute.
- Let's look at its value for the first few instances:

```
housing_cat = housing[["ocean_proximity"]]
housing_cat.head(8)
```

- Till now we only dealt with numerical attributes, but data may also contain text attributes.
- In this dataset, there is just one: the `ocean_proximity` attribute

```
      ocean_proximity
13096        NEAR BAY
14973       <1H OCEAN
3785           INLAND
14689          INLAND
20507      NEAR OCEAN
1286           INLAND
18078       <1H OCEAN
4396         NEAR BAY
```

```
housing_cat = housing[["ocean_proximity"]]
housing_cat.head(8)
```

- It's not arbitrary text:
- There are a limited number of possible values.
  - As each represents a category.
- So this attribute is a categorical attribute.
- ML learning algorithms prefer to work with numbers, so let's convert these categories from text to numbers.
- For this, we can use `sklearn`'s `OrdinalEncoder` class:

> Ordinal encoding works by mapping each unique category value to a different integer. Typically, integers start at 0 and increase by 1 for each additional category.

End to End ML Project

```
from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
```

- Here's what the first few encoded values in `housing_cat_encoded` look like:

```
[[3.]
 [0.]
 [1.]
 [1.]
 [4.]
 [1.]
 [0.]
 [3.]]
```

- You can get the list of categories using the `categories_` instance variable.
- It is a list containing a 1D array of categories for each categorical attribute.
- In this case, a list containing a single array since there is just one categorical attribute.

```
print(ordinal_encoder.categories_)
```

- You can get the list of categories using the `categories_` instance variable.
- It is a list containing a 1D array of categories for each categorical attribute.
- In this case, a list containing a single array since there is just one

```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
      dtype=object)]
```

```
print(ordinal_encoder.categories_)
```

- An issue is ML algorithms will assume that two nearby values are more similar than two distant values.
- This may be fine in some cases
  - i.e., for ordered categories such as `bad`, `average`, `good`, and `excellent`
- but it is obviously not the case for the `ocean_proximity` column.
  - i.e., categories 0 and 4 are clearly more similar than categories 0 and 1.

End to End ML Project

- To fix this, create one binary attribute per category:
  - One attribute equal to 1 when the category is `<1H OCEAN` (and 0 otherwise), another attribute equal to `1` when category is `INLAND` (and `0` otherwise), and so on.
- This is called one-hot encoding, because only one attribute will be equal to `1` (hot), while the others will be `0` (cold).
- The new attributes are sometimes called dummy attributes.
- `sklearn` provides a `OneHotEncoder` class to convert categorical values into one-hot vectors.

- To fix this, create one binary attribute per category:
  - One attribute equal to 1 when the category is `<1H OCEAN` (and 0 otherwise), another attribute equal to `1` when category is `INLAND` (and `0` otherwise), and so on.

```python
from sklearn.preprocessing import OneHotEncoder

cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
```

- `sklearn` provides a `OneHotEncoder` class to convert categorical values into one-hot vectors.

- By default, the output of a `OneHotEncoder` is a SciPy sparse matrix, instead of a `numpy` array.

> Sparse matrix is an efficient representation which contains mostly zeros. Internally it only stores the nonzero values and positions.

- When a categorical attribute has thousand categories, one-hot encoding results in a matrix full of `0`s except for a single `1` per row.
- A sparse matrix is useful for this as it will save plenty of memory and speed up computations.
- You can use a sparse matrix like a normal 2D array, but to convert it to a (dense) NumPy array, call the `toarray()` method:

```
print(housing_cat_1hot.toarray())
```

End to End ML Project

- One of the most important transformations you need to apply to your data is feature scaling.
- With few exceptions, ML algorithms don't perform well when the input numerical attributes have different scales.
- This is the case for the housing data:
  - The total number of rooms ranges from about `6` to `39,320`, while the median incomes only range from `0` to `15`.

> Without any scaling, most models will be biased toward ignoring the median income and focusing more on the number of rooms.

- There are two (2) common ways to get all attributes to have the same scale

Min-Max Scaling (Normalisation)

- For each attribute, the values are shifted and rescaled so that they end up ranging from `0` to `1`.
- This is performed by subtracting the min value and dividing by the difference between the min and the max.
- `sklearn` provides a transformer called `MinMaxScaler` for this.
- It has a `feature_range` hyperparameter that lets you change the range if, for some reason, you don't want `0–1`.
- i.e., neural networks work best with zero-mean inputs, so a range of `–1 to 1` is preferable.

Standardisation

- First subtracts the mean value.
  - so standardised values have a zero mean.
- Then divides the result by the standard deviation.
- This makes the values have a standard deviation equal to 1.

> Standardisation does not restrict values to a specific range

- However, standardisation is much less affected by outliers.
- i.e., suppose a district has a median income equal to 100 (by mistake), instead of the usual 0–15. Min-max scaling to the 0–1 range would map this outlier down to 1 and it would crush all the other values down to 0–0.15,
- `sklearn` provides `StandardScaler` for standardisation:

- If a feature's distribution has a heavy tail both min-max scaling and standardisation will squash most values into a small range.
- ML models generally don't like this at all.
- So before you scale the feature, you should first transform it to shrink the heavy tail, and if possible to make the distribution roughly symmetrical.
- For example, a common way to do this for positive features with a heavy tail to the right is to replace the feature with its square root (or raise the feature to a power between 0 and 1).
- If the feature has a really long and heavy tail, such as a power law distribution, then replacing the feature with its logarithm may help.

End to End ML Project

- So far we've only looked at the input features, but the target values may also need to be transformed.
- For example, if the target distribution has a heavy tail, you may choose to replace the target with its logarithm.
- But if you do, the regression model will now predict the log of the median house value, not the median house value itself.
- You will need to compute the exponential of the model's prediction if you want the predicted median house value.
- Luckily, most of `sklearn`'s transformers have an `inverse_transform()` method, making it easy to compute the inverse of their transformations.

- As you can see, there are many data transformation steps that need to be executed in the right order.
- Fortunately, `sklearn` provides the `Pipeline` class to help with such sequences of transformations.
- Here is a small pipeline for numerical attributes, which will first impute then scale the input features:

```
from sklearn.pipeline import Pipeline

num_pipeline = Pipeline([
    ("impute", SimpleImputer(strategy="median")),
    ("standardize", StandardScaler()),
])
```

- `Pipeline` takes a list of name/estimator pairs (2-tuples) defining a sequence of steps.
- The names can be anything you like, as long as they are unique and don't contain double underscores (`__`).

> They will be useful later, when we discuss hyperparameter tuning.

- The estimators must all be transformers.
- i.e., they must have a `fit_transform()` method, except for the last one, which can be anything:
- a transformer, a predictor, or any other type of estimator.

End to End ML Project

- If you don't want to name the transformers, use the `make_pipeline()` instead
- It takes transformers as positional arguments and creates a `Pipeline` using the names of the transformers' classes, in lowercase and without underscores (e.g., "`simpleimputer`"):

```
from sklearn.pipeline import make_pipeline

num_pipeline = make_pipeline(SimpleImputer(strategy="median"),
↪  StandardScaler())
```

End to End ML Project

- We framed the problem, we got the data and explored it,
- Sampled a training set and a test set, and a preprocessing pipeline to
- Automatically clean up and prepare your data for machine learning algorithms.
- You are now ready to select and train a machine learning model.

- Due to all these previous steps, things are now going to be easy.
- You decide to train a very basic linear regression model to get started:

```
from sklearn.linear_model import LinearRegression

lin_reg = make_pipeline(preprocessing, LinearRegression())
print(lin_reg.fit(housing, housing_labels))
```

- Done! You now have a working linear regression model.
- The model could be of course improved, but that is a different topic for a different time.

End to End ML Project

- You are ready to evaluate the final model on the test set.
- There is nothing special about this process.
- Just get the predictors and the labels from your test set and run your `final_model` to transform the data and make predictions, then evaluate these predictions:

```
X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()

final_predictions = final_model.predict(X_test)

final_rmse = mean_squared_error(y_test, final_predictions,
↪  squared=False)
final_rmse
```

- Now need to get your solution ready for production.
  - i.e., cleaning up the code, writing tests and documentation.
- You can then deploy your model.
- A basic way to do this is to save the best model you trained, transfer the file to your production, and load it.
- To save the model, you can use the `joblib` library like this:

```
import joblib

joblib.dump(final_model, "my_california_housing_model.pkl")
```

- Once your model is in production, you can load it and use it.
- For this, first import any custom classes and functions the model relies on (which means transferring the code to production), then load the model using `joblib` and use it to make predictions:

```python
import joblib

# extra code – excluded for conciseness
from sklearn.cluster import KMeans
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.metrics.pairwise import rbf_kernel

def column_ratio(X):
    return X[:, [0]] / X[:, [1]]

#class ClusterSimilarity(BaseEstimator, TransformerMixin):
#    [...]

final_model_reloaded =
↪   joblib.load("my_california_housing_model.pkl")

new_data = housing.iloc[:5]  # pretend these are new districts
predictions = final_model_reloaded.predict(new_data)
print(predictions)
```

- i.e., perhaps the model will be used within a website:
  - User will type in some data about a district and click Estimate Price.
  - This sends a query containing the data to the web server,
  - Which will forward it to the web application.
  - Finally your code will call the model's `.predict()` method.

> You want to load the model upon server startup, rather than every time the model is used.

End to End ML Project

- Alternatively, you can wrap the model within a dedicated web service that your web application can query through.
- This makes it easier to upgrade the model to new versions without interrupting the main application.
- It also simplifies scaling, since you can start as many web services as needed and load-balance the requests coming from your web application across these web services.
- Moreover, it allows your web application to use any programming language, not just Python.

End to End ML Project

- But deployment is not the end of the story.
- You also need to write monitoring code to check your system's live performance at regular intervals and trigger alerts when it drops.
- It may drop very quickly, for example if a component breaks in your infrastructure, but be aware that it could also decay very slowly, which can easily go unnoticed for a long time. This is quite common because of model rot: if the model was trained with last year's data, it may not be adapted to today's data.

End to End ML Project

- So, you need to monitor your model's live performance.
- The action you take depends on the application.
- In some cases, the model's performance can be inferred from downstream metrics.
- For example, if your model is part of a recommender system and it suggests products that the users may be interested in, then it's easy to monitor the number of recommended products sold each day. If this number drops (compared to non-recommended products), then the prime suspect is the model. This may be because the data pipeline is broken, or perhaps the model needs to be retrained on fresh data (as we will discuss shortly).

End to End ML Project

- However, you may also need human analysis to assess the model's performance.
- For example, suppose you trained an image classification model to detect various product defects on a production line.
- How can you get an alert if the model's performance drops, before thousands of defective products get shipped to your clients? One solution is to send to human raters a sample of all the pictures that the model classified (especially pictures that the model wasn't so sure about).
- Depending on the task, the raters may need to be experts, or they could be nonspecialists, such as workers on a crowdsourcing platform (e.g., Amazon Mechanical Turk).
- In some applications they could even be the users themselves, responding, for example, via surveys or repurposed captchas.

Appendix

Table of Contents

Appendix

Go Back

A black-box ML refers to machine learning models that give you a result or reach a decision without explaining or showing how they did so.

The internal processes used and the various weighted factors remain unknown. In other words, there is a lack of transparency in this technology.

Appendix

Go Back

Data mining is the process of extracting and discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems.

Data mining is an interdisciplinary subfield of computer science and statistics with an overall goal of extracting information (with intelligent methods) from a data set and transforming the information into a comprehensible structure for further use.

[1] Jeelani Ahmed and Muqeem Ahmed. "Online news classification using machine learning techniques". In: *IIUM Engineering Journal* 22.2 (2021), pp. 210–225.

[2] MA Anusuya and Shriniwas K Katti. "Speech recognition by machine, a review". In: *arXiv preprint arXiv:1001.2267* (2010).

[3] Soufyane Ayanouz, Boudhir Anouar Abdelhakim, and Mohammed Benhmed. "A smart chatbot architecture based NLP and machine learning for health care assistance". In: *Proceedings of the 3rd international conference on networking, information systems & security*. 2020, pp. 1–6.

[4] The Editors of Encyclopaedia. Britannica. *Deep Blue*. 2024. URL: `https://www.britannica.com/topic/Deep-Blue`.

[5] Santosh K Gaikwad, Bharti W Gawali, and Pravin Yannawar. "A review on speech recognition technique". In: *International Journal of Computer Applications* 10.3 (2010), pp. 16–24.

[6] Aditya Gaydhani et al. "Detecting hate speech and offensive language on twitter using machine learning: An n-gram and tfidf based approach". In: *arXiv preprint arXiv:1809.08651* (2018).

[7] Latifa Greche et al. "Comparison between Euclidean and Manhattan distance measure for facial expressions classification". In: *2017 International conference on wireless technologies, embedded and intelligent systems (WITS)*. IEEE. 2017, pp. 1–4.

[8] MD Islam and Morshed Chowdhury. "Spam filtering using ML algorithms". In: (2005).

[9] H Jabbar and Rafiqul Zaman Khan. "Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study)". In: *Computer Science, Communication and Instrumentation Devices* 70.10.3850 (2015), pp. 978–981.

[10] Ziqiu Kang, Cagatay Catal, and Bedir Tekinerdogan. "Machine learning applications in production lines: A systematic literature review". In: *Computers & Industrial Engineering* 149 (2020), p. 106773.

[11] Ye-Jiao Mao et al. "Breast tumour classification using ultrasound elastography with machine learning: A systematic scoping review". In: *Cancers* 14.2 (2022), p. 367.

[12] What Is Data Mining. *Introduction to data mining*. Springer, 2006.

[13] Patel Monil et al. "Customer segmentation using machine learning". In: *International Journal for Research in Applied Science and Engineering Technology (IJRASET)* 8.6 (2020), pp. 2104–2108.

[14] Abdallah Bashir Musa. "Comparative study on classification performance between support vector machine and logistic regression". In: *International Journal of Machine Learning and Cybernetics* 4 (2013), pp. 13–24.

[15] Joel Larocca Neto, Alex A Freitas, and Celso AA Kaestner. "Automatic text summarization using a machine learning approach". In: *Advances in Artificial Intelligence: 16th Brazilian Symposium on Artificial Intelligence, SBIA 2002 Porto de Galinhas/Recife, Brazil, November 11–14, 2002 Proceedings 16*. Springer. 2002, pp. 205–215.

[16] Jatin Sharma et al. "Product recommendation system a comprehensive review". In: *IOP conference series: materials science and engineering*. Vol. 1022. 1. IOP Publishing. 2021, p. 012021.

[17] Prissadang Suta et al. "An overview of machine learning in chatbots". In: *International Journal of Mechanical Engineering and Robotics Research* 9.4 (2020), pp. 502–510.

[18] Laurens Van Der Maaten, Eric Postma, Jaap Van den Herik, et al. "Dimensionality reduction: a comparative". In: *J Mach Learn Res* 10.66-71 (2009).

[19] Jesper E Van Engelen and Holger H Hoos. "A survey on semi-supervised learning". In: *Machine learning* 109.2 (2020), pp. 373–440.

[20] Jiajing Wang et al. "Predicting stock market trends using lstm networks: overcoming RNN limitations for improved financial forecasting". In: *Journal of Computer Science and Software Applications* 4.3 (2024), pp. 1–7.

[21] Helmut Wasserbacher and Martin Spindler. "Machine learning for financial forecasting, planning and analysis: recent developments and pitfalls". In: *Digital Finance* 4.1 (2022), pp. 63–88.