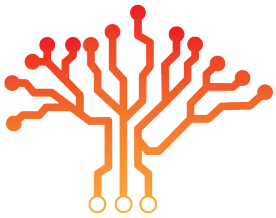


AI SINGAPORE®

AI Practitioner Handbook

Contributed by Engineers from AI Singapore

Edited by Kenny WJ Chua, Ryzal Kamis, Siavash Sakhavi, Anand Natarajan, Kevin Oh, Najib Ninaba, Kim Hock Ng and Laurence Liew

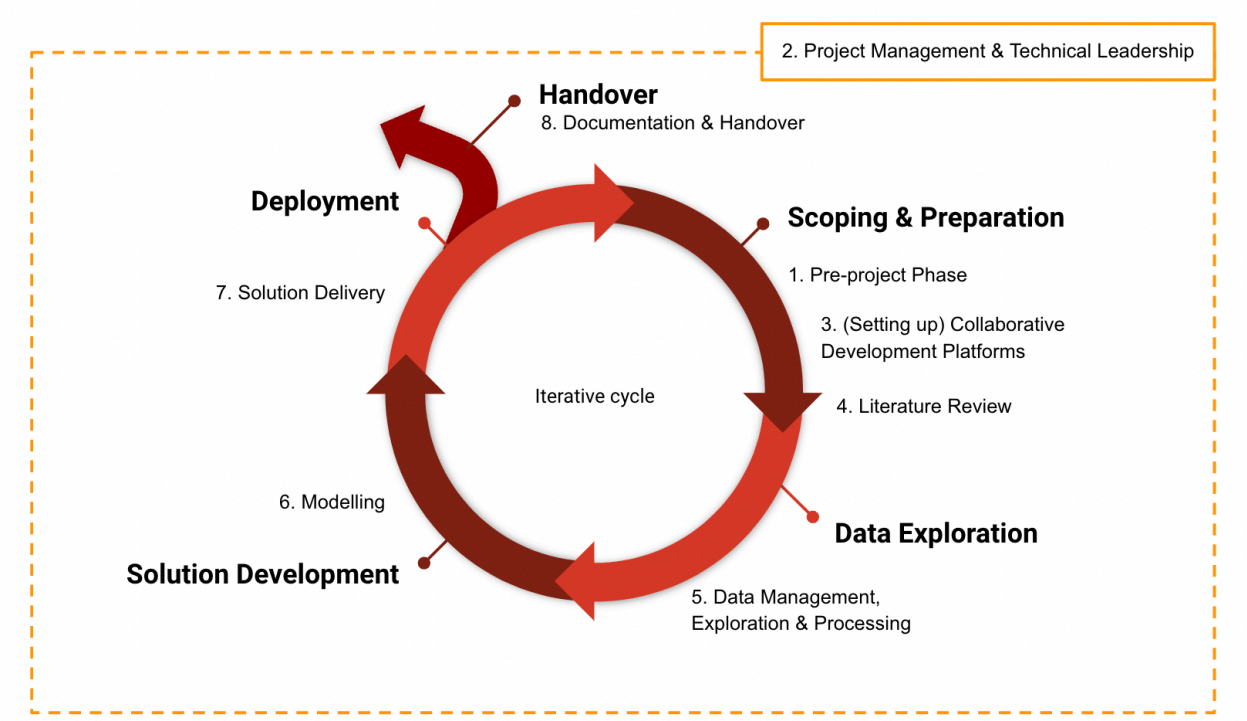
Mar 24, 2023

CONTENTS

AI SINGAPORE® | AI Practitioner Handbook

Introduction

This handbook is an accumulation of AI Singapore's Innovation and Platforms Engineering team's experience in delivering more than 40 AI Minimum Viable products (MVP) under the 100E programme over the last 5 years. It is an edited volume containing individual original articles written by our AI Engineers, themed around common topics encountered over a typical AI development lifecycle.



We envisioned this handbook as a useful guide for new AI Engineers joining AI Singapore and to quickly come up to speed on how we execute AI projects. However, the information contained here would also appeal to any new AI Engineers and Managers deploying their first AI project into production.

Delivering production AI models goes beyond building AI models in Jupyter notebooks. It includes preliminary data identification, cleaning and curation, followed by building, training and testing the models and finally deploying the model. Throughout the process, AI and ML Ops play a fundamental role to enable the whole end-to-end process.

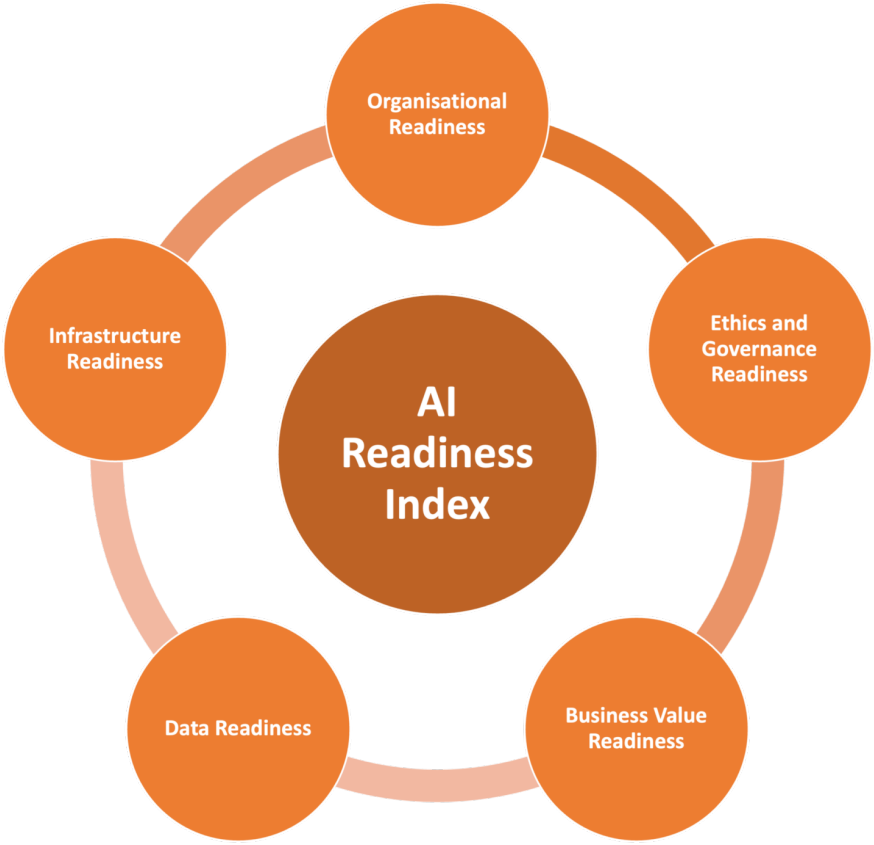
There are many excellent books and resources on various AI algorithms, and we will not replicate them here. Instead, we will focus on the best practices and knowledge required to deliver an AI project from end to end.

Identifying AI Ready Projects

AI projects executed by AI Singapore in the 100E programme are real-world problem statements from the industry and go through a rigorous review process before it gets approved and onboarded before any engineering work is done.

Leveraging the AI Readiness Index (AIRI) developed by AI Singapore, we are able to quickly identify organizations that are AI Unaware, AI Aware, AI Ready or AI Competent.

In the 100E programme, team's work focuses on AI Ready companies only[1].

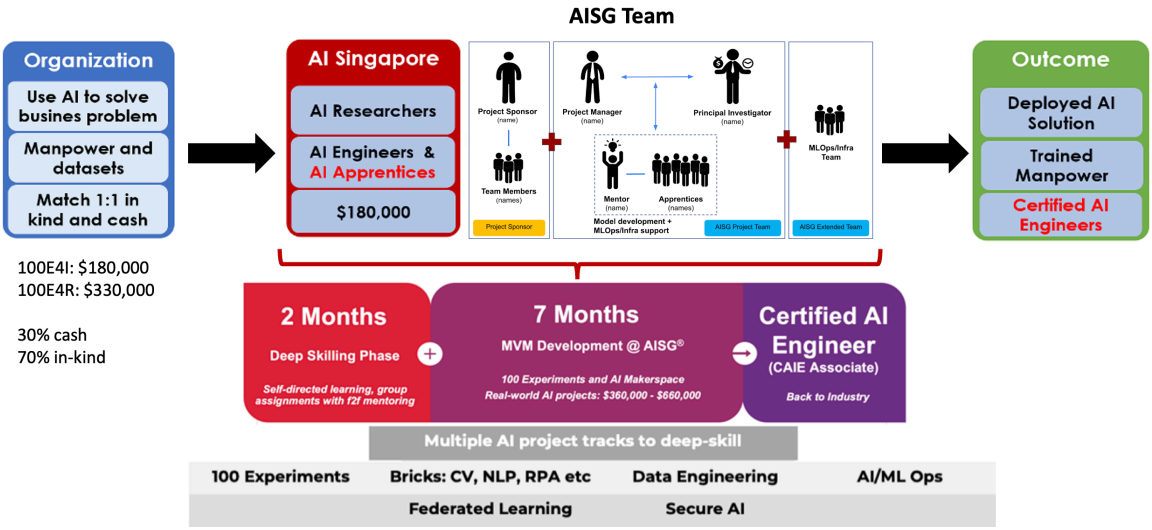


Delivering AI Ready Projects

Projects approved under the 100E programme are delivered in 7 months following an Agile methodology, typically over 10-15 sprints.

[1] AI Singapore and other Singapore agencies have other programmes which assist companies that are AI Unaware and AI Aware.

100E + AIAP



The project team will consist of AI Singapore's AI Researchers, Engineers, and AI Apprentices[2] (from the AI Apprenticeship Programme) working on the project full-time for the duration of the project. They are supported by Project Managers and the AI and ML Ops teams to deliver the project on time, and with the right infrastructure and architecture powered by established CI/CD pipelines.

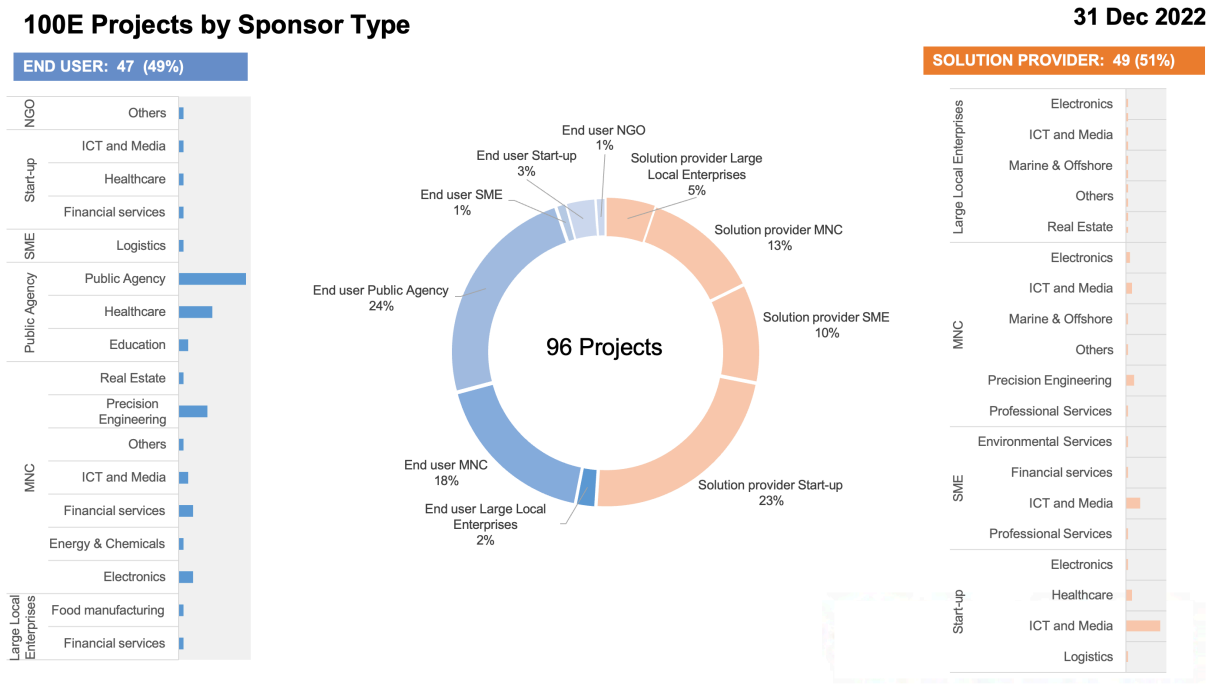
Our AI Engineers come from diverse academic disciplines and industry experience. In fact, many of our AI Engineers are hired from AI Apprentice cohorts. This handbook will provide new AI Engineers with a quick guide on our best practices to help them quickly become productive.

Conclusion

Since 2017, we have engaged more than 600 companies, approved more than 100[3] projects and completed more than 60 of these AI projects under the 100E programme, and trained more than 250 AI Apprentices.

[2] Who are the AI Apprentices? See How Did AI Singapore Build a "200" Strong All-Singaporean AI Engineering Team With the Blue Ocean Strategy?

[3] Information to date



The Handbook is a live project and will continue to evolve and be updated as we do more projects and learn more.

Let us get started on your first AI project!

–

Laurence Liew, Director (AI Innovation)

What Our Reviewers Say

"*Whether your role in an AI project is that of a technical lead, AI model implementor, data manager, domain or business function expert, or business-side project manager, this handbook will accelerate your learning curve for understanding the end-to-end aspects of the AI project.*"- Steven Miller, Professor Emeritus of Information Systems, Singapore Management University and co-author of *Working with AI*, MIT Press

"*This is a fantastic book because it focuses on an often overlooked aspect of ML education—the actual problems, people and teams you deploy it for. It's a great resource for anyone who wants to successfully put the theory of ML into practice. BAM!*"- Josh Starmer, Founder and CEO at StatQuest

"*AISG's release of the AI Practitioner Handbook as a practical and credible guide to accelerate the learning curve of incoming AI scientists and engineers is a generous service to Singapore's growing AI community.*"- Jason Tamara Widjaja, Global AI Lead at a multinational biopharmaceutical company

CHAPTER

ONE

PRE-PROJECT PHASE

1.1 Overview

This chapter covers sections pertaining to activities that happen before the commencement of development works in a project. Such activities include project scoping, data readiness assessment, and considerations for initial tech stack selection.

1.2 How can business challenges be translated into AI problems?

Contributor: Tan Kwan Chet, Lead AI Technical Consultant

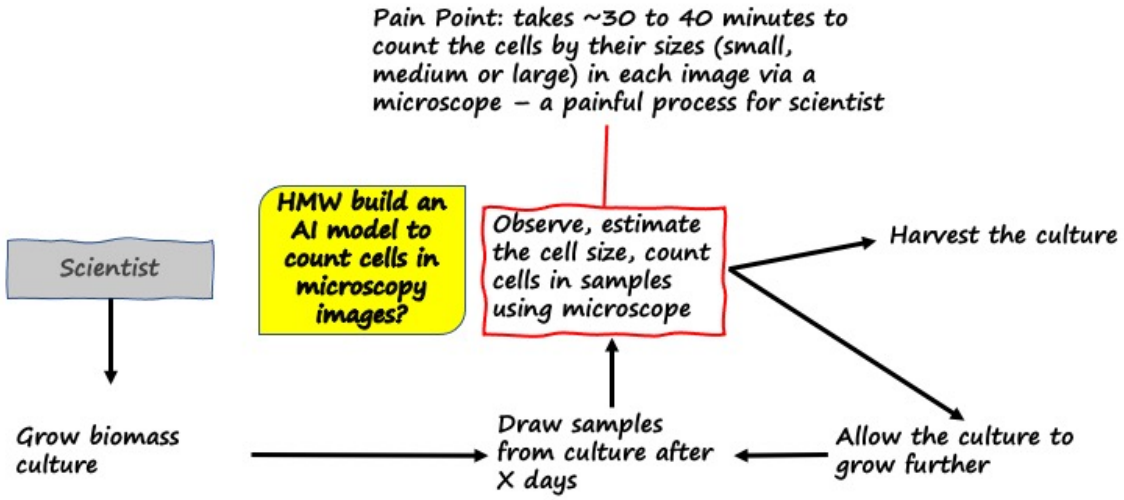
1.2.1 Framing Problem Statement from the Business Challenge

Most AI projects begin when a sponsor has 1 or a few business challenges to solve. During early stages of an AI project, the AI engineer needs to explore the data and methologies that can best tackle the AI problem present in business challenges. Understanding business context and process is a great starting point to frame the AI problem. It allows you to understand the sponsor's long term goal and identify the problem statement that affects the sponsor's business or represents a pain point that the sponsor wants to solve.

It takes a lot of creativity to formulate the problem statement. Sponsor tends to speak about their business problems at a high level. So your job is to filter the high level information shared by sponsor to a low level where the problem may relate to their employees who face the pain point on a day-to-day basis. This is important since your AI solution will likely be replacing a repetitive task that their employees may be facing. To form the problem statement, AI Singapore uses concepts from Design Thinking methodology and Design Sprint to help in this could use the following steps (illustrated as a map diagram below):

1. Identify the target audience (e.g. employee)
2. Define the problem from target audience's perspective (i.e. understand their pain point)
3. Understand when and where the problem is ocurring
4. Discover the benefit for the target audience and value for the business
5. Convert the information into a HMW (How Might We) problem statement

Example - Map Diagram



Note: the above-mentioned example is hypothetical and it is not representative of any company

1.2.2 Translation of Problem Statement into AI Problem

Next, it is wise to decompose the problem statement into different technical requirements of an AI problem. This can be done by asking a series of questions to prompt the sponsor to better describe their technical requirements.

Example - Translation from Problem Statement to Technical Requirements

Sponsor's Problem Statement: "How might we build an AI model to count cells on microscopy images?"

Question	Rationale	Technical Requirement
What is the main task that you are keen to replace?	This gives you an idea on what is the main task that AI needs to automate	AI model to detect cells by size
What is the type of the data involved?	By knowing if the data is image/video, text, tabular or multimodal, it will help you narrow down to the appropriate AI domain/models to focus on	Images of cells taken under microscope
What is the output required?	By knowing specifically how the business needs the outputs returned to them, it will help you select the correct algorithm and design the appropriate annotation format	Masks drawn around detected cells and size estimated -> Computer Vision instance segmentation model
Is the main task achievable by AI model alone?	You can check if there is a need to use heuristic method or rule-based methods in conjunction with the AI model in order to reach the required outcome	Apply pixel:nm size conversion to estimate actual sizes of detected objects and group them into classes
How will the AI model fit into your business processes?	This helps you to understand how the sponsor will use the AI model, in terms of what applications it will connect to, how often inference will be run, and how much human involvement there will be	Description of high level application architecture diagram and infrastructure requirements

With these, you would have gained a better understanding of the business challenge and how it translates into the AI problem. This will give you a coverage of what the sponsor hopes to achieve from its envisioned AI model.

1.2.3 References

- Sprint: How to Solve Big Problems and Test New Ideas in Just Five Days
- Design, Together
- Data Science for Business

1.3 What are some data considerations when framing an AI project?

Contributor: Tan Kwan Chet, Lead AI Technical Consultant

1.3.1 Key Areas in Data

Quote from Andrew Ng

> AI systems need both code and data, and "all that progress in algorithms means it's actually time to spend more time on the data," Ng said at the recent EmTech Digital conference hosted by MIT Technology Review.

Data Centricity

With the rise of Data-Centric AI (DCAI) since 2021, there is a switch in focus from code that builds the model to the data that model is trained on. The shift arises because improving the quality of the data could help model to perform better, and in a fairer and more robust manner, than simply improving the code alone.

4 Key Data Questions

Having understood the business challenge and AI problem at hand, this is where you need to delve into understanding the data the sponsor will be providing. You need to assess if there are risks/uncertainties posed within the data collected and annotated for solving the AI problem. Because a good AI model not only relies on algorithms but also quality data.

In practice, there are many considerations and checks for data quality. In this article, we focus on 4 key questions that are highly relevant at the pre-project phase to ensure that the AI problem is feasible and well-scoped. Note that while this is written with a supervised learning approach in mind, most of the considerations are also applicable to other learning approaches.

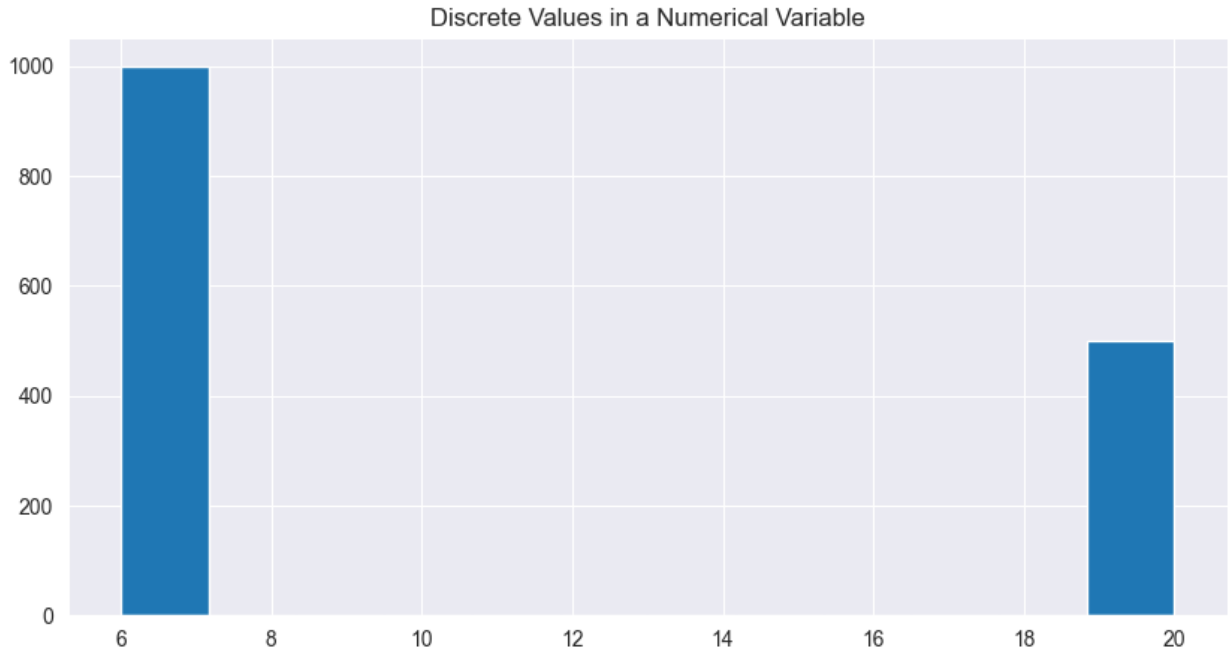
1. Is the target clearly defined and labels available?

Frequently, sponsor may not be able to articulate the target variable of interest. This is where the translation of the business problem in this chapter becomes important. This target variable must be related to a business outcome (e.g. reducing employee's repetitive tasks, prioritising patients for treatment based on mortality rate) that the sponsor wants to achieve.

Based on the translated AI task, this is where you ascertain whether the sponsor has a clear definition for the class/value/object/entity/text etc that is to be inferred. This determines whether quality labels can be defined.

If the target is of classification nature, it will be ideal to observe the current label proportion (e.g. binary label: "yes", "no") to ascertain if there is presence of label data imbalance. If the label data is highly imbalanced (e.g. 5% positive label and 95% negative label), an alternative would be to reframe the AI problem as an anomaly detection problem. More importantly, you would be keen to confirm if the label data corresponds to the definition of the target and the availability of the label data.

If the target belongs to a regression type, the question is more about ensuring the target variable is truly numeric/continuous. If upon checking you find that there is only a limited set of discrete values that the target variable can take on, it is better to reframe it as a classification problem. This is a possible scenario in businesses that rely on domain experts to run product experiments by testing different combinations of inputs with varying levels.



2. Does the data include features that can predict the target?

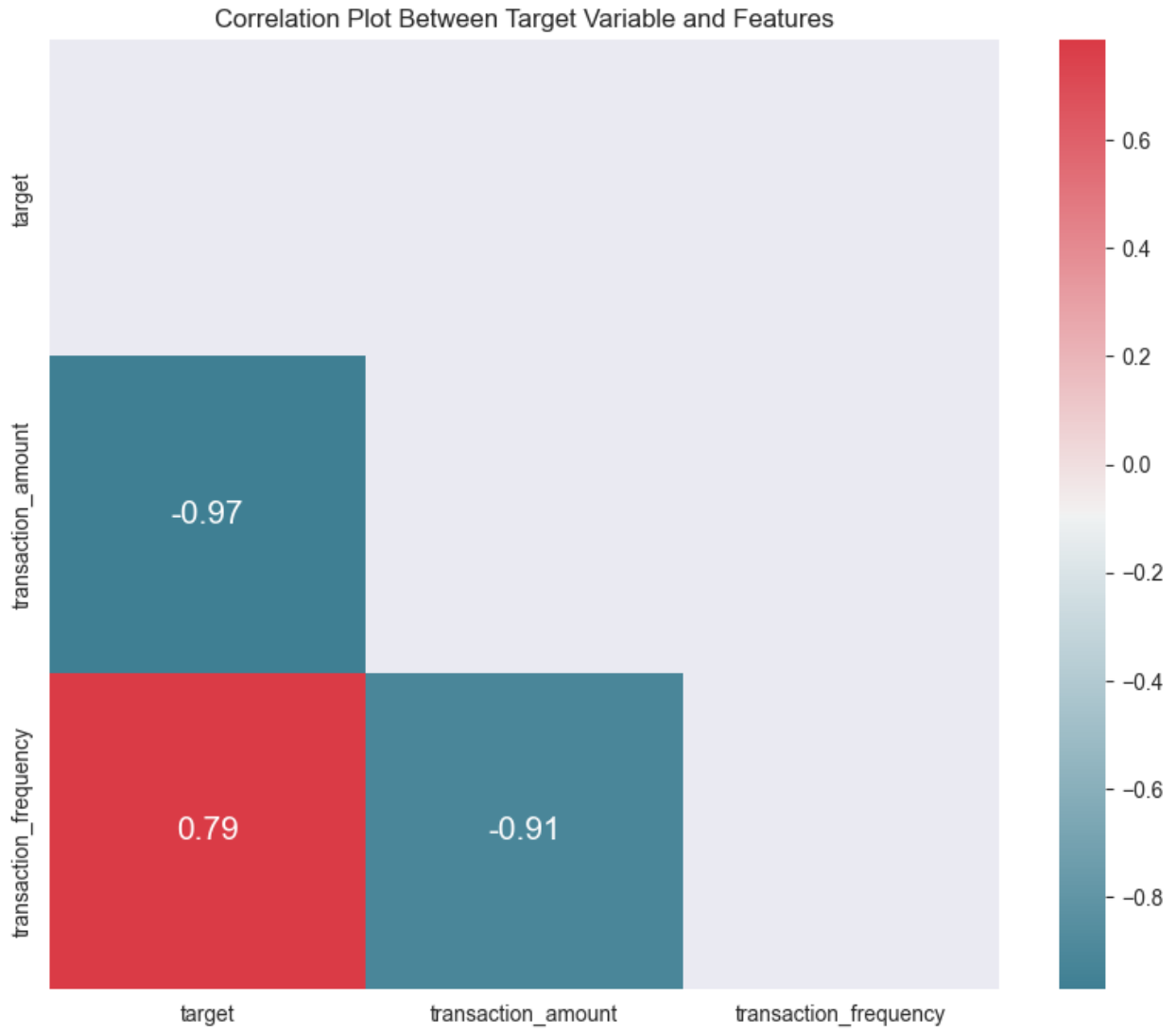
An ideal way to estimate if the relevant features are present would be to ask the sponsor the following questions based on a priori domain knowledge:

- If a human were to perform the task, what information would the person rely on to make the prediction?
- Is this information available for training the model and during inference?

There should be adequate signal and variance within the relevant features. Signal refers to the presence of sufficient information relevant to predicting the target. Variance refers to having sufficient range and diversity of data.

This can be checked via a small, representative sample dataset from the sponsor.

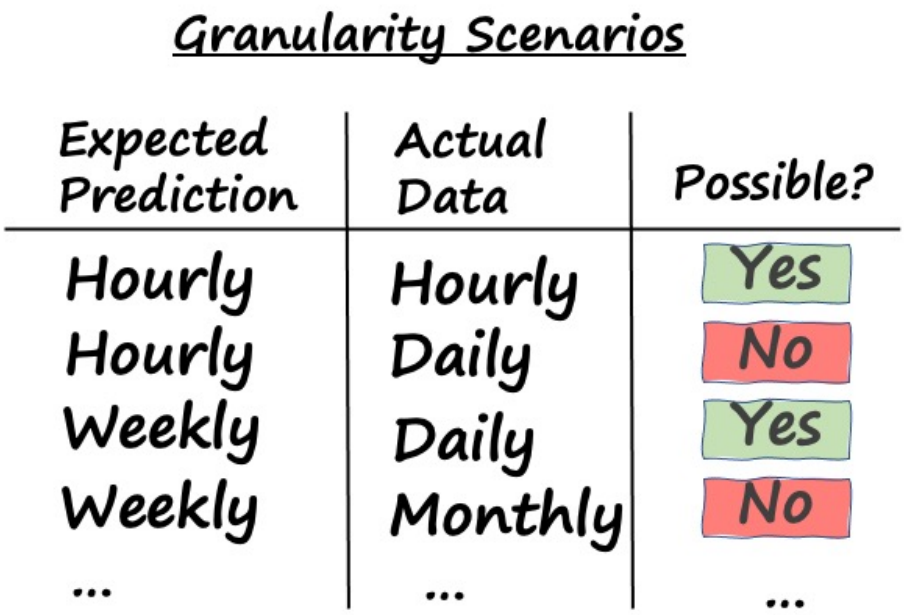
- for CV and NLP problems, using a pretrained model to run inference on sample data is a great way to assess the 'information-ness'. Sometimes, you can also 'eyeball' the data, since for images and text, WYSIWYG!
- for tabular problems, you can create a plot of mutual information/correlation between features and the target variable. A simple example is shown below for numeric features.



At times, you will face a situation in which sponsor claims that their domain knowledge is accurate in determining the predictive features while our exploratory data analysis (EDA) lends limited support for it. You could attempt some denoising techniques (e.g. outlier removal, averaging/binning) and check if the correlation improves. If it does not improve, this might be due to some confounding effect of a third feature that may influence the relationship between target and feature of interest.

3. Is the data at the correct granularity?

It is critical to ensure that the training data and production data share the same level of granularity (e.g. hourly, daily, weekly, monthly or yearly) as what your sponsor expects in prediction. So if the sponsor expects the prediction on an hourly basis and the sponsor collects the data daily, then it will not be possible to build such an AI model to produce a granular prediction based on aggregated data. It is also worthy to note that if expected prediction is at a higher or similar granularity as the training or production data, then it is possible to build an AI model.



Besides matching granularity between training and production data, understanding the correct granularity allows you to estimate the volume of data available for training. For example, if the goal is to aggregate a sensor's readings from seconds to hours for prediction, because the per-second readings contain mostly noise, then having "86,400 data points" may sound like a large dataset, but in actual fact, it is insufficient as there are only 24 hourly aggregates for you to work with.

4. Is the training data representative of the production data?

It is critical that training data is representative of the production data to ensure the trained model will be fair and robust:

- Avoid unintended bias. If the model is trained on training data that is dominated by a specific label (i.e. majority label), then it is very likely that the model is robust in predicting a majority label while it will unlikely predict a minority label.
- Improve robustness and generalisability. The distribution of training data will be what the model will learn from. Assuming that the distribution between training and production data is highly similar, this would mean that the trained model is able to generalise and make robust predictions in varying contexts that are often present in production data. It also means the model is less prone to immediately drift the moment it is applied in production.

So how we check for representativeness? To do so, you would first need to discuss and identify a few key attributes that capture the defining characteristics of the data. Here is an example for a salient object detection problem, where 'size of objects' and 'whether there are multiple objects' are 2 attributes. For problems that involve human subjects like customers and patients, defining characteristics could include key demographics like age, gender and ethnicity.

Data Definition

Attribute	Definition
Object size	Object size is required to be between 20% to 50% of an image
Multi-object	Multi-objects refer to 2 objects or more within an image while non-multi-objects refer to a single object per image
...	...

Once you are able to define these, you could request the sponsor to estimate the proportion of their **production** data broken down by different attributes. For example, below is a breakdown of the distribution of object size for the salient object detection dataset. The goal is then to ensure that the training data matches this production distribution:

Object Size Attribute

Categories	Number of Images
20% and below	100
20% to 50%	50
50% and above	70

Using these 4 key questions will allow you to have a gauge of the quality of the data for building the model. This is an important starting point for building a model that has good predictive power, and it is also fair and robust.

1.3.2 References

- Data-Centric AI
- Landing AI
- Why it's time for 'data-centric artificial intelligence'
- Is My Data Any Good? A Pre-ML Checklist
- What to do After Deploying your Model to Production

1.4 What are the considerations for reducing technical debt?

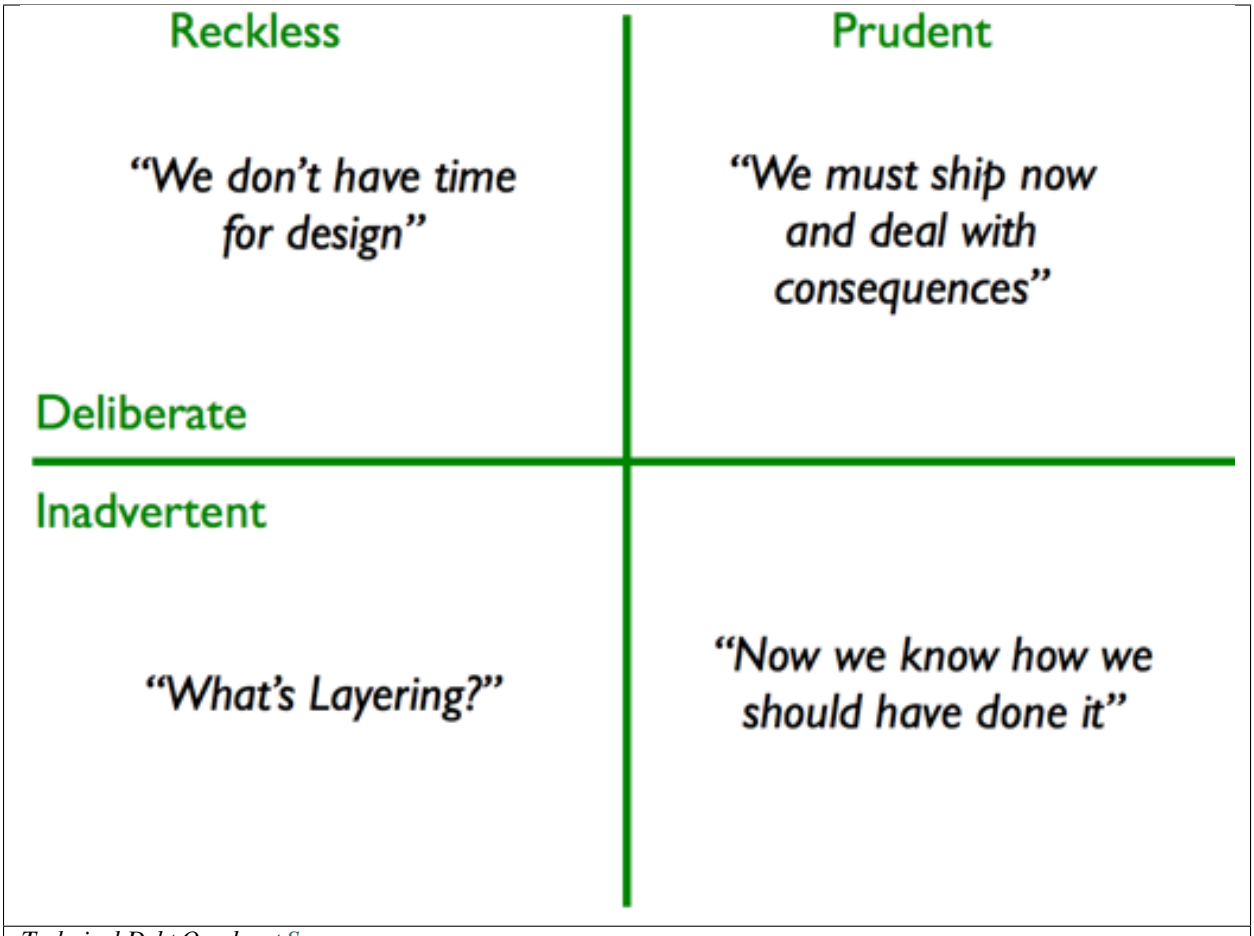
Contributor(s): Dylan Poh Guan Kiong, AI Engineer (GitHub)

1.4.1 What is technical debt?

In software development, technical debt is the accumulation of continuous expenses. It occurs when software engineers prioritise speed of deployment over all other development factors. Fast builds can cause problems that can be very difficult to resolve in the future.

Similar to financial debt, technical debt that is not repaid can accrue "interest," which makes it more expensive to make modifications in future. However, it might not always be a bad thing as developers might intentionally cut corners to produce a proof of concept to advance initiatives.

Martin Fowler offers a more detailed analyses of technical debt quadrant:



Technical Debt Quadrant Source

1.4.2 Deliberate and prudent technical debt

Deliberate vs inadvertent

AI engineers may deliberately choose to cut corners, knowing that it would cost them in the long run. For instance, the developer may choose to build and deploy a model without investing in a framework or pipeline, choosing quick release over maintainability. In other words, they consciously trade short-term gains for future expenses.

Junior AI engineers might not fully appreciate the benefit of using version control system and inadvertently decide to develop without the relevant tools. They may later find it impossible to roll back the code when the pipelines are broken.

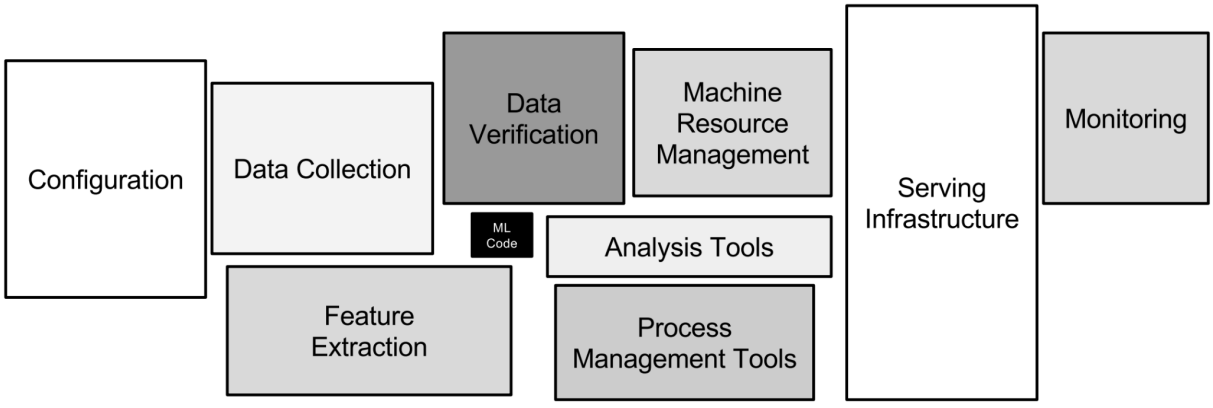
Prudent vs reckless

Some engineering choices, like rapidly prototyping a large number of models to choose the best algorithm, hence overlooking a maintainable pipeline could be a prudent decision since they are aware that they are incurring debt and consider the pros and drawbacks of using the best algorithm versus paying it off sooner.

However, knowing sound software development principles but writing spaghetti code because of a tight schedule could result in reckless debt.

1.4.3 Technical Debt in Machine Learning

The maintenance/infrastructure expenses associated with machine learning project go well beyond those technical debt associated with standard software projects. Jumping onto the hype train and using machine learning to tackle problems that already have a solution or that can be solved quickly using heuristics may be costly and reckless, but using machine learning to replace repetitious work with labor shortage (such as object detection on defects) will be a deliberate choice with long-term benefits.



ML Code takes up only a small portion of the whole pipeline, and technical debt can accumulate quickly in other processes. Source

1.4.4 The 25 Best Practices

Technical debt has already been the subject of extensive research. A list of best practices as recommended by Matthew McAteer is provided here:

- Use interpretability tools like SHAP values
- Use explainable model types if possible
- Always re-train downstream models
- Set up access keys, directory permissions, and service-level-agreements.
- Use a data versioning tool.
- Drop unused files, extraneous correlated features, and maybe use a causal inference toolkit.
- Use any of the countless DevOps tools that track data dependencies.
- Check independence assumptions behind models (and work closely with security engineers.
- Use regular code-reviews (and/or use automatic code-sniffing tools).
- Repackage general-purpose dependencies into specific APIs.

- Get rid of pipeline jungles with top-down redesign/reimplementation.
- Set regular checks and criteria for removing code, or put the code in a directory or on a disk far-removed from the business-critical stuff.
- Stay up-to-date on abstractions that are becoming more solidified with time
- Use packages like Typing and Decimal, and don't use 'float32' for all data objects
- Do not leave all works-in-progress in the same directory. Clean it up or toss it out.
- Make sure endpoints are accounted, and use frameworks that have similar abstractions between languages
- Make it so you can set your file paths, hyperparameters, layer type and layer order, and other settings from one location
- Monitor the models' real-world performance and decision boundaries constantly
- Make sure distribution of predicted labels is similar to distribution of observed labels
- Put limits on real-world decisions that can be made by machine learning systems
- Check assumptions behind input data
- Make sure your data isn't all noise and no signal by making sure your model is at least capable of overfitting
- Use reproducibility checklists when releasing research code
- Make a habit of checking and comparing runtimes for machine learning models
- Set aside regular, non-negotiable time for dealing with technical debt (whatever form it might take)

1.4.5 Measuring technical debt

Before delving into specifics, this article includes a list of pertinent questions to consider, allowing the reader to get a general idea of the amount of technical debt present in a system:

- How easily can an entirely new algorithmic approach be tested at full scale?
- What is the transitive closure of all data dependencies?
- How precisely can the impact of a new change to the system be measured?
- Does improving one model or signal degrade others?
- How quickly can new members of the team be brought up to speed?

And here is a paper that goes into greater detail and provides additional information about the rubric used to evaluate technical debt.

1.4.6 References

- Machine Learning: The High-Interest Credit Card of Technical Debt
- Hidden Technical Debt in Machine Learning Systems
- The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction

1.5 How can an engineer assess a client's AI readiness?

Contributor: Joy Lin, Senior AI Technical Consultant

1.5.1 Who is this for?

This article is aimed at AI engineers in an organisation like AI Singapore, who are building a solution for another company (henceforth referred to as the client) to take over and implement. In AI Singapore, the business development/presales team, together with the AI engineer, scope for feasible AI projects. In doing so, they assess whether the client has the capability to take over the final solution, integrate and maintain it, as the goal is to enable companies to build their own AI capabilities in the long run. Assuming that the client's proposed AI solution has an established business value and is ethical, this article narrows the focus to the technical aspects of AI readiness.

1.5.2 Why does AI engineer need to assess client's AI capabilities?

When participating in pre-project scoping, it is beneficial for you to gauge if the client is capable of taking over your solution for deployment. Most AI projects end up in failures during deployment phase due to three main reasons:

1. Technical hurdles in implementing/integrating model into existing operations
2. Decision makers unwilling to approve change to existing operations
3. Model performance not considered strong enough by decision makers

1.5.3 What can the AI engineer look out for?

To ensure a successful project, let us focus on three broad areas to help you assess the client's AI readiness level.

1. Organisational readiness

> Is there an existing technical team who is able to integrate and maintain the AI models? If not, what are the client's plans to hire the necessary resources?
>
> Is the client's management supportive of AI projects and team expansion if necessary?
>
> Do the client's management and key stakeholders allow room for experimentation and development?

As an organisation looking to use AI solutions, the client's management should understand that model building and maintenance requires iterative experimentation and continuous re-training, in turn supporting the investment of right resources.

2. Infrastructure readiness

> Is there appropriate infrastructure to hold data in a centralised and standardised manner (e.g. data warehouse), or is the client relying on disparate file systems?
>
> Are there sufficient computational resources (e.g. CPUs, GPUs, memory) to support model deployment and maintenance for this project? If not, are there plans to acquire more or re-allocate resources?

Preparing for the above strengthens the client's infrastructure capabilities, easing their transition from deployment to integration with minimal disruption to their existing activities.

3. Data readiness

> Does the centralised data warehouse provide consistent data format, up-to-date metadata and a single source of truth?
>
> Is there accurate and complete data to be used in this project?

It is important for the client to maintain high quality and consistent data that can be used for re-training when model drifts over time.

Note: Click *here* to read about data readiness on a *project* level.

More details available below to help in your assessment:

1. AI Readiness Index (AIRI) developed by AI Singapore and increasingly used by organisations in Singapore
2. Oxford's Government AI readiness index

CHAPTER

TWO

PROJECT MANAGEMENT & TECHNICAL LEADERSHIP

2.1 Overview

This chapter includes sections on both project management and technical leadership. Project management matters include insights on project lifecycle, managing interactions end-users of the ML solutions, etc. Technical leadership matters relate to managing a team of AI developers, as well as typical activities performed by a technical lead over the lifecycle of a project.

2.2 How can I build an effective AI development team?

Contributor: Kenny WJ Chua, Senior AI Engineer

2.2.1 Introduction

The responsbilities of a technical lead are different from those of an individual contributor. Engineers, especially those who are who are unfamiliar with leadership responsibilities, may face a learning curve in fulfilling these responsibilities.

A technical lead indirectly delivers an AI project on time and on target by building an **effective** AI development team. This article provides suggestions on promoting effective teams by contextualising general leadership principles.

In AISG, an engineer serves as technical lead for a small team of apprentices (i.e., 'developers'). Therefore, the contents of this article would be useful to any AI practitioner who is in a similar leadership role.

This article requires some basic knowledge of a AI project life cycle and typical activities that occur in each stage. The article on *building a* cohesive *AI development team* may also be useful for engineers in the role of technical lead.

2.2.2 Model the way

A technical lead may have a specific mental picture of the deliverables and ideal approach for a given task, but the end result may differ when delegated to developers. One possible reason is imperfect communication, such that developers do not entirely understand the specifications that have been laid out (hence the need for architecture diagrams). Alternatively, less experienced developers may not understand the need to adhere strictly to certain best practices.

In such a context, it is important for the technical lead to be able to **demonstrate** some of these values and practices that he/she espouses. For example, a technical lead can cultivate a team habit of writing well-structured and documented code by taking on a few of such tasks himself/herself. This may be particularly fruitful when onboarding a new team of junior developers who are not yet familiar with the required standards.

A technical lead can cultivate an environment for innovation by enforcing a **no-blame culture** for failed experiments. This involves focusing on and rectifying the work process that led to the failure, rather than paying attention to the personnel involved. There is also a need to choose the *correct* measurement of success. For example, developers should not be judged in terms of model evaluation metrics, which are strongly dependent on data quality which is largely beyond the team's control. It would perhaps be fairer and more fruitful to assess the developers' adherence to best practices (e.g., judicious data splits, systematic model experimentation).

As the technical lead, take responsibility for the team's failures, while giving full credit to the team for success. While there is a need to balance practical considerations of the project, the aim of these activities is to build a safe space that encourages developers to experiment.

2.2.3 Challenge the process

Developers within the team can come from different backgrounds, and can therefore contribute diverse, innovative ideas. Taken discerningly, this can present a significant advantage for the project.

Complementing an innovative culture, an effective team also strives for **iterative improvement** that brings the project closer to established best practices. This requires the technical lead to keep up to date by spending a significant portion of his/her time **reading broadly about and evaluating best practices** for each stage of the AI project lifecycle. Such information can be obtained by discerningly following tech blogs and influential thought leaders on social media platforms. Reading/research need not be confined to the specific subdomain of the current project, as ideas in other areas may be transferrable or can provide inspiration.

Iterative improvements come at the inter- and intra-project levels. At the inter-project level, an experienced technical lead can evaluate past projects to identify areas of workflow improvement. The technical lead can also network and share regularly with other technical leads in order to facilitate inter-project transfer of ideas. At an intra-project level, Agile philosophy stresses the importance of iterative development. When time allows, the project team can revisit previously written code in order to further improve it to more closely align with best practices.

2.2.4 Enable others to act

The primary responsiblity of the technical lead is to be an **enabler** rather than a "super contributor". While a technical lead can contribute directly to the codebase at times, he/she needs to be aware that this comes with a trade-off of spending less time on enablement activities. Enabling activities are those that amplify/multiply the effectiveness of the team. These include researching and evaluating potential approaches, planning and delegation, as well as removing technical blockers.

Technical enablements include **boilerplate code and abstract classes**, which can help developers onboard more quickly and with greater consistency. Additionally, a robust *MLOps workflow* provides automation that enables to developers to concentrate of feature development and modelling activities rather than infrastructural work.

Pay particular attention during **stand-up meetings** and **sprint retrospectives**. These are platforms during which developers are likely to raise their concerns regarding **technical blockers**. The technical lead can triage the list in order to identify and resolve the most urgent and important blockers. Relatedly, regular **one-on-one meetings** with developers can also help encounter personal/interpersonal blockers that may not surface during meetings with the whole team.

2.2.5 References

- The Leadership Challenge: How to Make Extraordinary Things Happen in Organizations
- AI project life cycle
- 1-on-1s
- Your team needs a tech lead, not a lead techie

2.3 How can I cultivate a cohesive AI development team?

Contributor: Kenny WJ Chua, Senior AI Engineer

2.3.1 Introduction

Engineers in charge of a development team take on the role of a technical lead. This entails a set of responsibilities that are distinct from those of an individual contributor. Engineers, especially those who are inexperienced with leadership responsibilities, may face challenges in rallying their team.

A **cohesive** team is one that is well-aligned towards a single, common goal. This article provides suggestions on promoting cohesive teams by operationalising general leadership principles within the context of an AI development team.

In the context of AISG, an engineer plays the role of technical lead for a small team of apprentices (henceforth referred to as 'developers'). Hence, the contents of this article would be relevant to any AI practitioner who is in a similar role as team lead.

The article assumes that the reader has some (but not necessarily extensive) knowledge of a AI project life cycle and typical activities that occur in each stage. You may also wish to consult the complementary article on *building an* effective *AI development team*.

2.3.2 Importance of a cohesive team

During early stages of a project, developers can come onboard with divergent expectations of project deliverables. Even when project deliverables have been specified in writing, there can be varying interpretations regarding the scope of said deliverables and the approaches to achieving them. Developers will also come with their own set of specific expectations on what they wish to learn from a project, e.g., specific tech stacks.

Divergent expectations present a risk to the cohesiveness of the development team. This can make it challenging to come to a consensus on technical decisions. Correspondingly, the risk of miscommunication is also increased, which can lead to time-consuming rework of project components. It is therefore in the team's best interest to mitigate such risks by promoting alignment on project requirements and individual expectations.

2.3.3 Promoting a shared vision

The technical lead can do so by promoting a shared vision for the team. This vision includes the technical direction, scope and expected quality of work. The early stages of the project represents a critical window to align expectations on the project vision, so as to minimise the accumulation of downstream technical debt.

As much as possible, the project vision ought to be practically and tangibly laid out in the form of system ***architecture diagrams***, rather than left as potentially ambiguous textual/verbal descriptions. Areas that are **out-of-scope** also need to be made explicit as much as possible, such that the team can come to a common understanding of the work that needs to be prioritised for project delivery.

An effective vision requires **buy-in** from the project team. One way to facilitate this is to articulate how developers can benefit from the project. This frequently comes in the form of opportunities for professional/technical growth, e.g., by learning new technologies. However, it should also be carefully hedged that the choice of technologies used in the project need to be guided by project requirements rather than learning needs. While it is critical to foster technical growth amongst junior developers, their learning should occur in the form of on-job-training while keeping project needs as a priority.

In-person or synchronous remote **code review** sessions during the initial phase of a project present a practical opportunity to align expectations regarding code quality. These initial sessions can be conducted with the entire development team, such that any clarifications regarding coding and documetation styles can be immediately clarified. These initial code reviews will then set the tone for the expected code quality for the rest fo the project.

2.3.4 Encourage the heart

Morale is a key driver of team success. Maintaining high morale within the team can provide developers with encouragement and motivation to perform at their best. However, developing under tight deadlines and high expectations is stressful, so it takes a deliberate effort to keep the morale up.

One way to do so is by regularly **celebrating small project milestones** with the team. Beyond providing words of affirmation, demonstrate your appreciation in tangible ways that will actually help/be appreciated by the developers. For example, encourage the team to slow down after a particularly challenging sprint, or gather the team for a nice lunch after a milestone deployment. While some of these activities can incur some short-term costs in terms of time, the idea is to provide some time and space for developers to recharge so that can be greater long-term gains.

2.3.5 References

- The Leadership Challenge: How to Make Extraordinary Things Happen in Organizations
- AI project life cycle

2.4 What kind of engineering principles can I set for my development team?

Contributor(s): Siavash Sakhavi, Senior AI Engineer, Co-written with https://rytr.me/

Principles and values are two important concepts that should be set for any software team. All teams should set certain principles and values for themselves in order to be productive, successful, and happy. These two concepts are what will help guide your projects and keep everyone on the same page.

2.4.1 What are the differences between principles and values?

The first step in setting the values of a team is to identify the principles that will guide their behavior. These principles are the underlying beliefs that drive their behavior and decisions.

A principle is something that is true for all members of a team, no matter what. These principles are typically agreed upon by the team members at the start of their project or business.

Values on the other hand are more personal to each individual and reflect what they believe in. In order to set your values, you need to first identify your principles and then list what those principles mean for you and how they will affect your actions as an individual on the team.

2.4.2 Examples of Engineering Principles

The following are some engineering values that you can consider for your team:

- **Learn with curiosity**: Learning is often about following the rules, but this can limit your mind. When you are curious, you have a spirit of exploration that can lead to creative solutions. Curiosity leads to questions, which lead to experiments and discovery.
- **Strive for simplicity**: This includes the use of simple software design to ensure the user is able to understand it with ease. The engineer should also create documentation that is clear and concise. Simplicity is key in developing software that has a low bug rate as well as being easy to update.
- **Articulate your mental model**: A software engineer's mental model is their internal representation of the system being developed. It is the most important part of their job. They should be able to articulate this mental model to other developers in order for them to be able to understand what needs to be done, as well as what has been done and why it was done that way.
- **Teach with compassion**: It is essential that software engineers are willing to teach their team members who may not know as much about the given task. Not only will this ensure the success of the project, but it will also provide a better work environment for all.
- **Ship fast, sustainably**: Software developers often focus on the idea of "shipping" a product. But that's not the end goal. The goal is to have sustainable software engineering practices. This means shipping software in small iterations, and ensuring that the team has a good work-life balance because nobody can be productive if they're exhausted or burnt out.
- **Fix problems, even when they're not yours**: Ther is an epidemic of engineers who feel the responsibility for their own code but not the code that they don't touch. This leads to uncoordinated, confusing and unmaintainable code. It is important for developers to take responsibility for the whole codebase and not just their own piece of it.
- **Communicate early and often**: Communication is one of the most important skills in any team, and software development teams are no exception. From simple voice conversations to a more complicated Google chat discussion, it is essential that all members understand what is going on at any given moment. Without good communication, chaos can ensue and deadlines can be missed.
- **Company (i.e. Client) over team over individual**: The software development team should prioritize the client's requirements over the team's ambitions and individual's professional progress. It is easy to get distracted with new "shiny" software and frameworks. However, when the client's needs are not a priority, this can lead to poor code quality and even missed deadlines.
- **Outcome over effort (Don't reinvent the wheel)**: Most of the code required to process data, define a model and the given model is available online. There is no need to write everything from scratch.

Reference(s):

- You can find more engineering values and principles in this GitHub Repo

2.5 How might we simplify and translate technical jargon?

Contributor: Shafir Ahmad, Senior AI Technical Consultant

Communicating with people who are not actively involved in your project can present a challenge. Technical staff such as engineers and developers find it difficult to communicate complex technical concepts to clients, senior management and customers alike. It might even look like you are speaking different languages.

The fact is that many of these stakeholders you communicate with will understand the big picture and high level concepts, but not the intricate engineering details. It is better to keep these details for a more technically inclined audience.

Here are some tips you can use to reach an understanding between yourself and non-technical stakeholders.

2.5.1 1. Know your audience

Learn about your audience. Find out what their knowledge is on the subject matter beforehand so that you can tailor the information to their needs. During early engagements, it is better to communicate at a level below what you reasonably expect them to know, while being careful not to dumb it down too much such as to cause offence. When in doubt, you could always let them know "If this is something you already know, let us know and we can skip to the next section."

Additionally, if the level of understanding is different between participants, it would be helpful to target the "least technical person" and while pre-empting the rest of the audience to "bear with me if you already know some of this information". Do continually observe the stakeholder's gestures, facial expressions and body language to ensure everyone is following along.

2.5.2 2. Reduce or Eliminate Tech Jargon

Start eliminating technical jargon and acronyms from your conversations with the stakeholders. Certain terms may not make any sense to the stakeholder, and others may hold a different meaning from the one you commonly use (e.g. NLP = Neuro Linguistic Programming vs NLP = Natural Language Processing).

Try translating technical terms into their equivalents in everyday language. Pro tip: it helps if you understand the technicalities deeply.

Here is an example of communicating with a non-technical decision-maker:

```
Technical jargon: "We will ship an end-to-end pipeline for you to retrain the model␣
↪upon data drift in your production system."

Everyday language: "We will share the code and instructions with your technical team␣
↪to enable them to refresh the model after the solution has been implemented and you␣
↪have gathered enough new data."
```

Consider the use of metaphors, analogies and storytelling techniques that will help your audience understand the material.

2.5.3 3. Provide background information, context and education

Do not assume that the stakeholder knows or does not know certain concepts or terms, or that they understand the context. However, do assume that they are intelligent and will be able to understand if given an adequate explanation. Do translate key jargon and acronyms that are important to the discussion in a way that makes it simple for the stakeholder to understand. Metaphors and analogies are a good way to translate a technical concept into a more familiar context. It could be useful to give a glossary and definitions of some necessary terms.

2.5.4 4. Use visuals to illustrate technical concepts

While written words and verbal speech are effective ways of communication, working to make the concepts visual can make it more effective. You may have heard the adage "A picture is worth a thousand words". Research suggest that adding visuals can help increase memory recall to 65% versus 10% from just hearing it. With the use of creative illustrations such as flowcharts, process diagrams, demos, models and architecture diagrams, stakeholders can learn and understand the most technical concepts from these visual aids.

Take care not to overload the visuals, and keep it as simple and uncluttered as possible. Remember to tailor it to your audience as a high level executive may not need an in depth understanding of an architecture diagram, but needs an overview.

2.5.5 Conclusion

Pro Tip: An excellent way to practice is to try explaining some of your work to your spouse or friend. As the saying goes, "If you can't explain it simply, you don't understand it well enough."

With just a bit of thought and planning, you can make the discussion more engaging and understandable for the participants. You may need to arrange for additional meetings to provide the participants with the level of depth of understanding they require.

2.5.6 References

- Power of Visuals
- Know your Audience
- How to give a technical presentation
- Cut Technical Jargon

CHAPTER

THREE

COLLABORATIVE DEVELOPMENT PLATFORMS

3.1 Overview

This chapter consolidates sections on practices that facilitate a team of developers who are contributing jointly to a single codebase. This includes considerations on maintaining code quality, minimising code conflicts, and continuous integration.

3.2 What are the key platforms required for collaborative ML development?

Contributor(s): Dylan Poh Guan Kiong, AI Engineer (GitHub)

With the increasing complexity of machine learning projects, it is highly likely that a team of developers is required to complete the task. As a result, it is critical to have the ability to collaborate efficiently.

Fundamentally, there are two ways that teams collaborate.

Luke Marsden: When you boil it down to basics, there are really two fundamental modes of collaborating between different people doing work: synchronous and asynchronous.

Synchronous: Communications are scheduled, real-time interactions by phone, video, or in-person, sometimes interrupting one another for urgent tasks or simply asking a question. In terms of development, the upstream have to complete their tasks before the downstream team can continue with their work (Dependencies issues)

Asynchronous: Communication happens on your own time, with a different team or people concurrently working on a different task. An efficient mechanism is required to resolve merge conflicts.

With this in mind, we seek to introduce some tools and practices for a team to get started in the hope that stumbling blocks will be reduced to the minimum. The practices introduced here assume development using Python as the primary language.

3.2.1 Communication tools

Establishing effective communication right from the beginning of any project will allow all team members to align toward the same goal and reduce any possible friction. A quintessential example of miscommunication is an unclear problem statement leading to team members having different views on the outcome of the project.

The following tools (by no means exhaustive) may be useful for remote, office, and hybrid working arrangements:

- A big whiteboard often produces the most efficient channel to share ideas. (Physical Meetup)
- Google Workspace, a full suite of communication tools such as Google meet, shared Google calendar, and Gmail to see the complete list of participants and scheduled meetings. Other option includes the Microsoft Team.
- Miro is an easy to use, intuitive, collobration tool that uses little effort to setup quickly. Other option includes Coggle.

3.2.2 Code versioning

Jupyter notebooks (.ipynb) are powerful, but also have many limitations. For example, it is impractical to use pure git for versioning Jupyter notebooks (because of large amounts of embedded and frequently changing HTML). Jupyter notebooks are also unsuitable for deployment or production. Naming and renaming files or folders with comments are also undesirable and difficult to keep track of.

Instead, using git for distributed version control and tracking changes helps coordinate work among team members. Developing code in Python script (.py) is highly preferred and using GitLab or GitHub to manage your code online with an advanced interface to resolve merged conflict that occurs frequently for asynchronous work.

Merge conflicts occur when the source and target branches of a merge request (as termed by GitLab) or pull request (as termed by GitHub) contain different changes or commits. As mentioned, this is especially common during collaboration when more than two engineers are contributing to the code. In this case, engineers must choose which change to accept. Once the changes are made, you can assign reviewer to inspect the changes which ensure the correctness of the merge and prevent breaking the code. Examples of the occurrence of and tools to handle conflicts are linked here for both GitHub and GitLab.

Keeping a standardised environment for development keeps Python package dependencies consistent, and together with git, allows different members to easily switch between different machines such as local machines or machines in the cloud. One popular option that manages the environment is Conda (environment.yml)

More in-depth information on setting up a machine learning project repository can be found here.

3.2.3 Model/weight/data versioning

Shared storage such as cloud storage ensures the team is assessing the correct dataset and having access to a shared transformed dataset cuts down data preprocessing/transformation time allowing the engineer to focus on modeling. Also, weights and datasets should not be pushed to the version control system, as the repository size might become too large for the local machine.

Having a system to keep track of the trained model and the corresponding dataset early on will prove to be fruitful, as more experiments will make it difficult to trace back. One good practice will be to save the checkpoints and the models together and keep track of the corresponding dataset used for the training. Recommended tools include MLflow and Weights & Biases for tracking experiments and versioning models. DVC and neptune.ai are tools to consider for data version control. Even a simple shared spreadsheet keeping track of model/weights/dataset versioning will pay off eventually.

It is also possible to set up a cloud-based compute/development environment where all collaborators can share the same environment and tools (VScode, Jupyter Lab) and shared storage. Some of the examples are Amazon EC2 Instance Types, Data Science Virtual Machine and Google Compute Engine.

3.3 What are some considerations in setting up a project repository?

Contributor(s): Siti Nuruljannah Baharudin, AI Engineer

3.3.1 Use of existing project templates

Prior to setting up a repository, you would want to plan the structure of your code base. If your organisation provides an existing template for machine learning projects, that would be a good place to start. Otherwise, you may want to consider using one of the many open-source frameworks available, which come with boilerplate code for the various components in a typical ML project. The scaffolding provided by such frameworks would not only save you some effort in setting up the project, but also help to establish a standard for developers to create modular and maintainable code.

3.3.2 Configuration management

Regardless of the framework or template used, management of configuration files is an important area of consideration in any ML project, since ML pipelines designed to be flexible and extensible would rely heavily on configuration files as inputs. You are recommended to keep all configuration files in a single directory, separate from the source code files. It is also good practice to maintain separate configuration files for each environment developers work in. For instance, configuration files for the local development environment could be placed in a subdirectory `/config/local`, files for the test server environment in `/config/test`, and so on.

3.3.3 General file management

In a Git repository, a .gitignore file is crucial in helping you filter out files and directories that should not be pushed to the repository, such as configuration files for a developer's local environment, which would likely contain settings unique to the individual's workspace. Other examples of files that should not end up in the repository are authentication keys, data files, log files and trained model weights.

Typically, individual developers would create their own notebooks containing EDA-related code or prototype code for the models being explored. Although these may end up as throwaway code, they serve the useful purpose of knowledge sharing at the beginning of a project. It is recommended that these files are placed in a `notebooks` subdirectory and pushed to the repository within each developer's personal branch, which do not need to be merged to the main branch.

3.3.4 Dependency management

Another area to consider is the management of code dependencies. At the start of a new project, initialising a single set of initial dependencies would suffice. In a Python project, this could mean maintaining a single `requirements.txt` file. As the code increases in complexity, or if separate components of the application are to be deployed in different environments, it is common for conflicting dependencies to occur. In this case, developers would need to maintain multiple sets of dependencies. These dependency files could be named with suffixes indicating the environment or module it is for, e.g. `requirements-train.txt`.

3.3.5 Developer productivity tools

Besides software-related components, your repository may contain other files such as issue templates and hook scripts. Issue templates are used to pre-fill issue descriptions with custom fields or sections, which guides developers to include all necessary information when creating an issue. For instance, an issue template for a bug could include sections like "Steps to reproduce", "Expected result" and "Actual result".

In a Git repository, hooks are scripts that Git executes locally before or after events such as commit and push. For example, you could set up a pre-commit script to run code through a linter, ensuring it meets the standards before it is allowed to be committed. With the numerous hook scripts and frameworks available, hooks do not require a lot of effort to set up, yet can greatly increase developer productivity when used appropriately.

Reference(s):

- AI Singapore's Cookiecutter Template for End-to-end ML Projects
- Kedro - A Python framework for creating reproducible, maintainable and modular data science code
- Git Branch tutorial

CHAPTER

FOUR

LITERATURE REVIEW

4.1 Overview

This chapter contains sections that aim to systematise the process of reviewing literature relating to a problem domain, as well as go-to references that provide a starting point for typical AI problems.

4.2 What are some of the factors to consider during literature review?

Contributor(s): Andy Ong, AI Engineer

Literature reviews are necessary at the start of every AI project to understand and explore available solutions in the market. Business problems require different solutions, and below are some factors that you can consider while performing your literature review.

- Business needs
- Development time
- Existing code respositories
- Pre-trained models
- Reported results

4.2.1 Business needs

Business considerations must be taken into account during any literature review. These considerations tend to set restrictions on the project. Understanding these considerations can ensure that you are researching relevant solutions. For example, a medical AI project requires model explainability, and this will require a search for solutions involving explainable models. Alternatively, if the use case requires edge deployment in mobile devices, this would instead direct the selection of literature toward those that support smaller and quicker models.

The literature search should not be limited to your project's domain. An example is using autoencoders for anomaly detection even though they are typically used for computer vision. Such literature can introduce different approaches and highlight possible challenges that you might face in your project. You can understand the various approaches and their benefits, and how you can integrate them into your solution.

4.2.2 Development time

All projects have a set of timelines to adhere to, and development time may not be sufficient to build pipelines from scratch. You will have to allocate time for integration tests, project handover and other ad-hoc tasks such as troubleshooting. You can research and identify potential libraries to aid in your pipeline to avoid building it from scratch. You can also attempt to inherit certain classes from suitable libraries to customise them to your project with minimal code. The following sections are specific examples of strategies that can aid in shortening development time.

4.2.3 Existing code repositories

You should be reading up on literature that include training or inference scripts in their code repository. These solutions can be incorporated into your codebase with ease as compared to solutions without any scripts. Papers with Code contains papers with open-sourced code repositories.

If the literature is supported by code repositories, you should evaluate the readability of the code and complexity of integration. Supplied code repositories can often be difficult to understand, and integration may take longer than anticipated. Additionally, some repositories/implementations may be more comprehensive than others, such as those that contain class weights as configurable hyperparameters or include an evaluation function.

Consider the copyright agreements and licenses for the papers and codes in light of this. It's possible that your project will not be able to use commercial IP software or certain licenses (like the GPL). Examples of permissive licenses with few limitations on use include the MIT and BSD licenses.

4.2.4 Pre-trained models

For NLP or CV projects, it may be advantageous to search for solutions with pre-trained, open-sourced models. This will cut down on training time by ensuring that you can use transfer learning to such tasks. Long training times are needed for language or computer vision models, which can add time and cost to a project. It is advantageous if the pre-trained model was developed using datasets from a field related to your use case. For instance, if your use case involves evaluating financial accounts, FinBERT, a pre-trained NLP model, may be more immediately applicable than BERT-base.

4.2.5 Reported results

The results reported in the study should be interpreted in a discerning manner. Reported results may entail some cherry-picking. More crucially, the data used is likely different from your project's use-case. Consequently, expect to see a difference in accuracy, since data is typically the main element affecting how well any model performs. Rather than focusing on the reported results, you should instead concentrate on the evaluation techniques and algorithms employed in the literature.

CHAPTER

FIVE

DATA MANAGEMENT, EXPLORATION & PROCESSING

5.1 Overview

This chapters encompasses sections with a focus on data engineering, data lineage, data versioning, exploratory data analysis and feature engineering.

5.2 Which data storage options are suitable for the project?

Contributor(s): Syakyr Surani, Platforms Engineer

There are many data storage options that one can implement for their project, but to choose one (or several) of these options can be confusing to some. This guide hopes to give you some guidance in navigating various data storage options that are suitable for your project by honing in on three main considerations when choosing the right options for you.

5.2.1 Requirements and Competency of Project Sponsor

Data competency of the Project Sponsor cannot be understated when choosing a suitable data storage option, especially when they are not prepped up to be part of an AI-powered project. The skillsets of analysts in any given organisation can vary a lot from simple Excel data entry ones to highly automated processes employed by big tech companies. Therefore, it is important to understand the competency as well as the readiness of the Project Sponsor in committing on a set of data storage options. You can read more about this in our AI Readiness Index article as one of the ways to ascertain the Project Sponsor's data competency.

To give an example, you have a team from the Project sponsor that has only started going through AI tutorials, whether from YouTube or a reputable course provider. The data you need for the project is relatively small and only require the basic skills needed to build a model and to be used internally and sparingly. Then, using CSV or Excel files may be sufficient for the project.

On the other hand, if the project takes in a lot of data from multiple data sources, then they may want to implement a data lake to store those data before being processed and transformed into data warehouse(s). An article from Guru99 gives more insight regarding about data lakes and warehouses here.

We would also need to take the willingness of the Project Sponsor to accept a solution that is outside their comfort zone into consideration as well. In some cases, they might request something that is more sustainable than their current solution, while other Project Sponsors may not be ready to handle databases and are only comfortable with what they have been doing.

5.2.2 Reliability and Consistency

Reliability and consistency may be important in your project, especially when the latest dataset may be needed to reduce model drifting or other similar issues. Flat files such as CSV and Excel files may not be a viable solution as different members in your team may be using different versions of data, which may result in unreliable and inconsistent metrics when it comes to evaluation.

Therefore, it would be recommended to look at a more reliable solution such as database management systems (examples include SQL, MongoDB, etc.) if that is to be the case. A section provided by Microsoft Azure can give more information regarding these types of systems here, relational or non-relational.

5.2.3 Data Management (Simplicity vs Complexity)

The topic of data management coincides with the competency of your team as well as how reliable you need the data to be in the manner of cost efficiency. This cost could be in financial terms, in terms of your team's valuable time to pick up a new skillset, in terms of potential security and privacy issues surrounding the use of self-hosted or managed cloud solutions, or a mixture of all three.

Does your organisation have the capability to resolve operational issues to any data store solutions proposed? Is the data sensitive enough to warrant an in-house solution, or is it more cost-effective to have it run in one of the many managed cloud providers instead? Or is it less hassle to just share flat files such as CSV and Excel files in a cloud storage? These questions could be used to shape your current project's requirements, but you should also project those requirements towards the long term, and discuss within your team to reduce the times needed to refactor your project.

5.2.4 Final Thoughts

There are more considerations you may need to take note of other than the three that is discussed in this guide, but we believe that these three are the main considerations to take note of. However, if you feel that you need more tips on navigating your way on finding a suitable data storage solution, there is a Microsoft Azure page you might find more insights here.

5.2.5 References

- AI Readiness Index | AI Singapore
- Data Lake vs Data Warehouse: What's the Difference? | Guru99
- Understanding Data Store Models | Microsoft
- Criteria for Choosing a Data Store | Microsoft

5.3 Is there a systematic structure for performing exploratory data analysis?

Contributor: Er YuYang, Senior AI Engineer

The purpose of this article is to have a guide to performing exploratory analysis (EDA) in a more systematic manner. This guide is designed for a generic EDA purpose and primarily focuses on tabular data. However, parts of this workflow are still applicable for unstructured data types.

5.3.1 Overview

Most EDA activities should contain these blocks:

Structure investigation

This section should contains the most basic checks on the overall completeness of and level of information in the dataset. These include checking the data size and type, as well as granularity/cardinality of each table. It should also examine if the data contains any private or sensitive data.

Content investigation

This section looks into the quality, distribution and relationships in the dataset, and need not be in this particular order.

Quality checks

This sub-section should cover aspects of data quality such as missing and erroneous values. The main aim is to identify potential data to drop, repair or impute.

Distribution checks

This sub-section should cover the understanding of each individual feature. The main aim is to find out the chracteristics of each feature, and also to uncover trends or patterns in the data.

Relationship checks

This sub-section should cover more in-depth analyses such as bivariate relationships between features, and between features and target. The main aim is to generate ideas for feature engineering and selection based on observed relationships.

5.3.2 Structure investigation

This section should be short (about less than 10 notebook cells), as it only aims to gain some insights on the size and datatype of the data. This sections does not go into the content of each variable/feature.

When checking the datatype, organise features according to whether they are numeric/continuous or categorical. This will help to decide on suitable visualisations when performing content investigation later.

When dealing with private or sensitive data, discuss with stakeholder whether masking or removing them is a better option. When duplicate rows are present, highlight them and discuss with stakeholders on how to deal with them. Inappropriate handling of duplicate rows can results in data leakage when performing data split for model training/evaluation.

When performing granularity/cardinality checks, it is important to understand how fine or coarse each row and column is and verify whether the table is of the correct granularity per the database schema

Expected ouputs of structure investigation:

- Raw version of data loaded in the notebook
- Size of data, whether parallel processing is needed
- Datatype of features, which features are numeric/continuous, or categorical

- (Optional) Further discussion with stakeholders on handling private/sensitive data and duplicate rows

5.3.3 Content investigation

Quality checks

This sub-section should cover matters relating to data cleaning and imputation. If there are too many missing values, consider dropping the column. However if dropping is not an option, a more sophisticated method for imputing the missing value is to use the other avaiable features to predict the value of that particular column (model-based imputation/multiple imputation).

The choice of approach for handling missing data is also influenced by the type of "missingness". There are mainly three kinds of "missingness": MCAR, MAR, MNAR (see article link at reference section). Before performing any imputation, it is highly recommended to first consult the stakeholder and understand possible reasons behind the "missingness".

Generally, there are 4 ways to impute:

- Assign a fixed value(s) based on domain knowledge
- Simple imputation based on statistical summaries
- Model-based imputation
- Multiple imputation (see video link at reference section)

Expected ouputs:

- Cleaned version of data ready for plotting charts
- Identify data/column(s) that requires preprocessing in the datapipeline

Distribution checks

This sub-section should cover the generation of distribution and summary statistics. Understanding properties like central tendency (mean), spread (variance) and skewness help you to spot erroneous values and potential outliers easily.

If there are district groups in the data, try grouping them and perform individual analysis for each group, in addition to the analysis for the entire dataset. Try to use looping to cover the entire possible combinations when plotting charts instead of selecting particular features combination that you preceive is useful. Investigate further if certain combination of features have interesting results.

Perform cardinality checks on categorical features, as there could be similar category due to spelling mistake or variations. Cleaning these erroneous categories will help to reduce unnecessary noise in the analysis. It also helps to assess whether some categories are too sparse and may need to be merged for greater statistical power.

Try to link insights from the analysis to feature engineering. For example, in a parcel delivery context, if you find that number of deliveries (the target) is much higher on Mondays compared to the rest of the week, you could engineer a separate feature `is_monday` to boost the model's performance.

Highlight the findings on the notebook regardless of whether they are useful. This allows the reader to understand the purpose of the charts.

The analysis should cover the following:

- Summary statistics of the features
- Distribution of the features
- Feature patterns (trend)

Expected ouputs:

- Characteristics of each feature
- Spotting potential outliers and erroneous values
- Summary of the findings
- Potential feature characteristics that can be used for feature transformation in the data pipeline

Relationship checks

This section aims to find out the relationship between feature and target so as to generate ideas for feature engineering and selection (feature importance). The goal is to identify correlated features or categories. Correlated features distort interpretability and feature importance. As such, it is worth exploring dropping or merging collinear features based on domain knowledge.

In the scenario if there are too few features in the data, feature engineering should be explored. Conversely if there are too many features, feature importance should be explored and then dropping less important features.

If certain features have a very similar relationship to the target and are sparse, you could merge them during feature engineering to create a stronger feature. For instance, `capital_gains` and `capital_loss` can be merged into `is_investor` since if there are not many individuals who invest.

The analysis should cover the following:

- Relationships between each feature and the target
- Relationships between features

Expected ouputs:

- Identify features that are positively/negatively correlated to each other
- Identify features that are positively/negatively correlated to the target label
- Summary of the findings
- Ideas on feature engineering/selection

5.3.4 Automated EDA tools

Pandas Profiling and Google Facets are two excellent tools for performing EDA. They are useful to provide initial insights before drilling down into areas of interest through manual EDA.

5.3.5 References

- EDA structure
- Types of missingness
- MissForest
- Visualising distributions in seaborn
- Bivariate analysis in Python
- Phi K correlation coefficient
- Multiple imputation (video)
- Pandas Profiling
- Generate reports using Pandas Profiling

- Visualising ML datasets using Google Facets

5.4 What are some ways to do EDA for CV tasks?

Contributor: Calvin Neo, AI Engineer

The purpose of this article is to guide new AI Engineers on performing Exploratory Data Analysis of Computer Vision tasks in a systematic manner. It assumes a basic understanding of image format, and how bounding boxes and masks are drawn.

5.4.1 Image EDA

Basic Image Analysis

The basic image analysis considers the broad characteristics of the images that have been obtained.

The guiding questions are as such:

1. What is the number of images provided? Is this the number of images promised to be delivered for experiments?
2. What is the format of images eg JPG/PNG/JPEG? If there are multiple formats given, what are the counts for each image format?
3. What are the image sizes/resolution (eg 1080x960)? If there are multiple dimensions, what are the counts for each dimension?
4. How similar are the images to each other? You can do an EDA to check on this by performing image hashing to discover groups which are similar to each other. This also helps to uncover whether some of the images might have been curated from a continuous video stream of a scene that is changing very slowly - in such a case, these images may be near-duplicates.

Colour Intensity of Images

The colour intensity of images can be explored by either mean or mode of the channel values of each image. The goal for analysing annotations' colour channels is to understand what the dominant colours are. For example, images in forest settings may have more green, and images along water bodies are expected to be blue-dominated. This also ties in with annotations, where if the annotated objects are similar in colour values, the model would need to rely on feature other than colours such as edges.

Brightness of Images

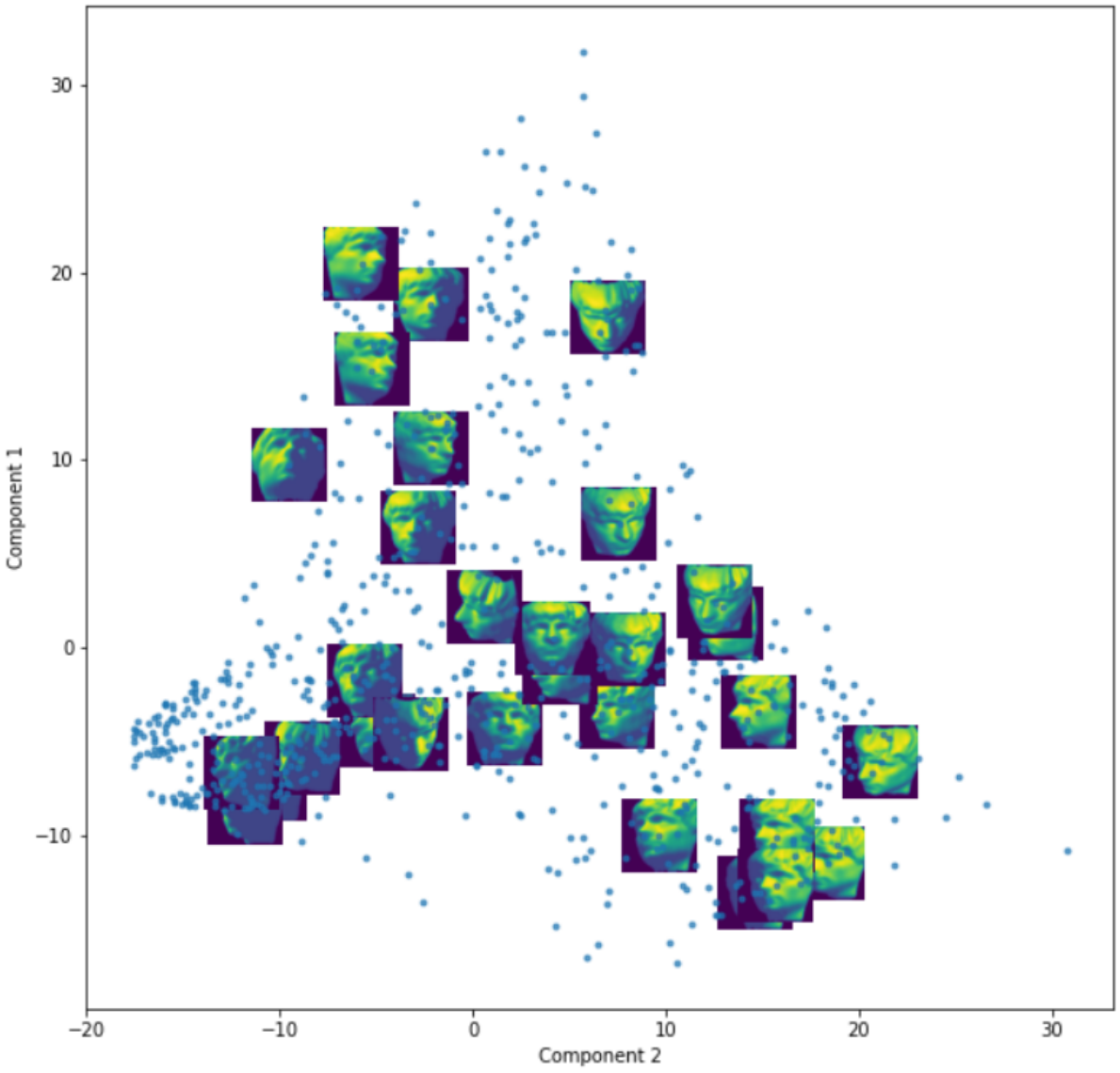
Brightness values are closely related to colour intensity. If the colour intensity for all channels are higher, then the images may appear to be brighter than normal and vice versa. It is thus the job of the AI Engineer to explore the data to ensure the brightness values of the images are of the normal range. Otherwise, they would need to scale up or down the brightness levels to ensure that the images for training are consistent.

Contrast of Images

Contrast is the pixel to pixel difference in channel values. For example, a green frog in Bukit Timah Nature Reserve would be said to have low contrast. This makes edge detection more important for the model.The onus is thus on the AI Engineer to explore the images to ensure the contrast levels are of the normal range. Otherwise, they would need to employ contrast adjustment techniques to ensure contrast values are normalised.

Image Similarity

1. PCA or ISOMAP per channel can be performed on the images to see how the images are related to each other.



5.4.2 Annotations (Basic Overview)

Bounding Boxes Formats

1. COCO - [x_min, y_min, width, height]
2. YOLO - [x_center, y_center, width, height]
3. Pascal VOC - [x_min, y_min, x_max, y_max]

For example, if the given format is in Pascal VOC, and the model requires it in YOLO format, we would need to convert the annotations to YOLO format for training.

An example for during inference: Read images using CV2, run YOLOv3 model, convert and return in the desired format according to what is required to be delivered.

Masks Format

Masks annotations gives the pixel-wise positions of the mask.

Masks are provided in an array of x-y pairs. It will typically be:

```
masks_per_instance = [x_1, y_1, x_2, y_2, ... x_n, y_n]
```

Masks may also come in the form of PNG files, where the PNG's image size would correspond to the original image size.

5.4.3 Annotations EDA

Basic Annotations Analysis

The basic annotations analysis attempts to find characteristics of the annotation such as number (and proportion) of classes as well as the values of the bounding boxes/ masks.

This is useful because it helps determine what type of metrics should be used, how the data (images) should be split, and what parameters could be tuned. For metrics, the usual case for this is error analysis of model performance on bounding box sizes. For example, if the bboxes/masks are mostly large objects among all the images, assuming images are sufficiently clear, it is clear that the model will do well. However, if the annotations are small, then the model will inevitably struggle. Hence, the AI Engineer will pay attention to certain metrics more, such as mAP with Area=Large.

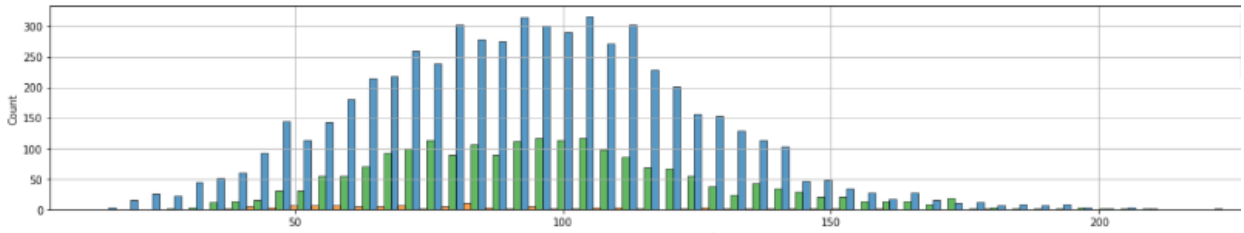
The data split refers to how one might partition the data for training/validating/testing the model. Generally, all sets would need similar proportions of the bbox size (small/medium/large) and class distribution. In the ideal scenario, these proportions are easy to split at random, but it is extremely rare. Thus, by performing EDA on the annotations, the Engineer will have a better idea on how the data split can be done.

Tuning parameters refers to configuring parts of model training/inference that could be useful for better performance. An example of this is anchor box size(s). By default, the anchor box sizes in the models were set to be tuned to the dataset that they were benchmarked against, for example MS COCO Dataset. However, the bbox/mask of real data that is received would not fit nicely to these anchor box sizes. Hence, by understanding a particular range of typical bbox/mark sizes, the anchor box sizes can be tuned to accomodate the range. This has been observed to be beneficial to model training and inference performance.

With these considerations in mind, these are some guiding questions when performing EDA on annotations:

1. How many instances of each classes are there?
2. Bounding Box - what is width and height distribution overall and per class?
3. Bounding Box - what is the overall size (W*H) overall and per class?

4. Bounding Box - What are the top few Width/Height combination? This can be found using clustering methods.
5. Segmentation - what is distribution of the mask pixel counts overall and per class?



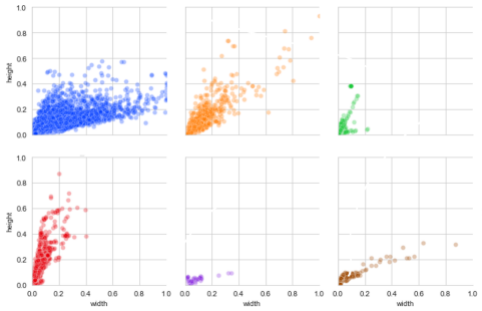
Colours of Annotations

As with images, the annotations have 1 or more channels per pixel, with the most popular format being Red, Green and Blue (RGB). The utility for this mainly lets the researchers know if the model would be better served trying to use certain channels when making inference.

Average Intensity per channel

To explore this for all classes, we can firstly average the colour intensity of the annotation's pixels per channel.

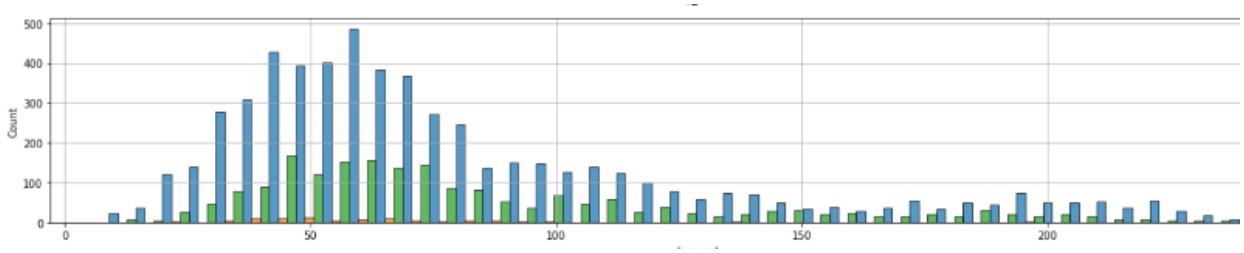
Secondly, countplot of the average colour intensity values could be generated. The image below shows the counts of each annotation with the average values of the red channels separated by classes. In this case, there are three classes, with one of the class having very spread out red channel intensity, and the other two classes having colour intensity being normally distributed.



Example of a Height-Width comparison among bboxes separated by class.

Mode Intensity per channel

Besides averaging the intensity per annotation, the highest occurring value (mode) of the intensity of the annotation can also be used to analyse the colour of the annotation.



There are many other ways to visualise this. The main goal is to check on the colour distribution among the annotations.

5.4.4 Annotations Random Sample Checks

Bounding Box and Mask Random Sample Checks

Bounding Boxes Random Sample Check refers to performing a check by sampling images and analysing their annotations. These try to account to account for human error that very often occurs.

1. Are the bboxes/masks drawn tight enough?
2. Were the bboxes/masks drawn accurately?
3. Are the objects being occluded? In other words, were parts of the object blocked by another object? In general, if the object of interest is being occluded, its annotation should still be drawn such that it includes the occluded portion.
4. Are the objects truncated? In other words, were parts of the objects "spilling" out of the image?

For occlusion and truncation, it is recommended to include "occlusion" and "truncation" in the metadata of the annotation.

5.4.5 Conclusion

This was a short survey of some of the areas in which the AI Engineer can begin in exploratory data analysis for images as well as their annotations. This is by no means exhaustive, and the AI Engineer should explore further methods for EDA in this space.

5.4.6 References

Perceptual Hashing - https://www.phash.org/

VOC annotation guide - http://host.robots.ox.ac.uk/pascal/VOC/voc2011/guidelines.htmlcd The Scientist and Engineer's Guide to Digital Signal Processing - https://www.dspguide.com/ch23/5.htm

Further EDA readings

Histograms:

https://homepages.inf.ed.ac.uk/rbf/HIPR2/histgram.htm

http://web.cs.wpi.edu/~emmanuel/courses/cs545/S14/slides/lecture02.pdf

https://www.allaboutcircuits.com/technical-articles/image-histogram-characteristics-machine-learning-image-processing/

Anchor Boxes:

https://d2l.ai/chapter_computer-vision/anchor.html

Other Data Exploration Ideas:

https://neptune.ai/blog/data-exploration-for-image-segmentation-and-object-detection

5.5 What are the various data split strategies?

Contributor(s): Lee Xin Jie, Senior AI Engineer (100E)

5.5.1 Static train-test Split

The most common train test split strategy is to create static training, validation and test data sets.

Train set

The train set is the subset of the data that will be used to train the ML model. The model's performance on the train set will only provide an indication of how well the model will perform on previously seen data.

Validation set

The validation set, also commonly referred to as the development set, consists of data that is unseen during model training. It provides an estimation of the model's performance during production. The validation set is often used during model hyperparameter tuning and algorithm selection. When performing algorithm selection, it is important to ensure that the validation set used is the same across all algorithms to ensure a fair comparison.

Test set

The test set is the second hold-out set that will not be used during training. Since the validation set is commonly used to influence the modelling decisions, our ML model pipeline is likely to be biased in performing well on the validation set. Hence, we will need a completely unseen set of data to provide a final benchmark of our model's performance on unseen data. It is important to not use this test set to influence your modelling decision. Otherwise you will get an inflated sense of how well the model will perform in real life. You should only evaluate your model on the test set during the final step of the pipeline. As such, it is common for the train set performance to be higher than the validation performance, which in turn, should be higher than the test performance.

Tip: After finding the best model from validation, it is also good practice to re-run this model (with the best hyperparameters) on the full dataset (train + validation + test) and save the resulting weights for further use.

Split ratio

There is no single 'golden' data splitting proportion. In general, the larger your dataset, the lower the proportion of data you have to set aside for the validation and test sets. If you have a dataset with a billion data points, you could theoretically set aside 1% of the data for the test set, and still have a sufficiently large test set of 10 million data points. Conversely, the smaller your dataset, the larger the proportion of data that should be set aside for your validation and test sets. This will allow you to have a more confident estimation of model performance during production, as your model's performance will be determined on a larger and more diverse validation/test set.

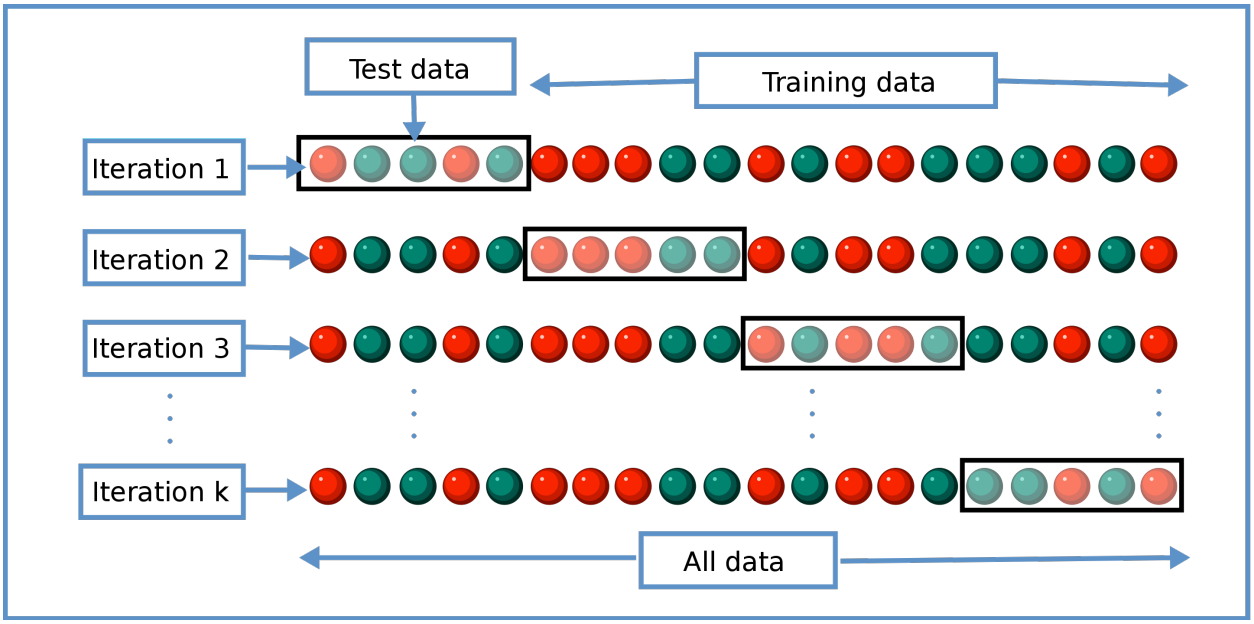
One commonly used rule of thumb will be to adopt a 70-20-10 split for your train, validation and test sets respectively. Ultimately, you should prioritise selecting a ratio catered to your needs.

5.5.2 Cross-validation

A single, static validation set could potentially present a biased assessment of model performance. This is particularly the case with smaller datasets where favourable validation performance may arise by chance.

Cross-validation is an alternative to having a static validation set. In cross-validation, you will conduct multiple rounds of model training, each time with a different section of your dataset serving as the validation set. This will minimise the variance associated with choosing any particular choice of validation set. However, cross-validation does come with the penalty of additional training time. The difference could be significant especially when training large models. So it is usually recommended when you are dealing with a smaller dataset where there could be insufficient samples to form a reliable validation set. In such cases, favourable validation performance may arise by chance due to a higher proportion of 'easier' examples appearing in the validation set, hence the need for cross-validation.

An illustration of this is as follows.



Source: Cross-validation (statistics)

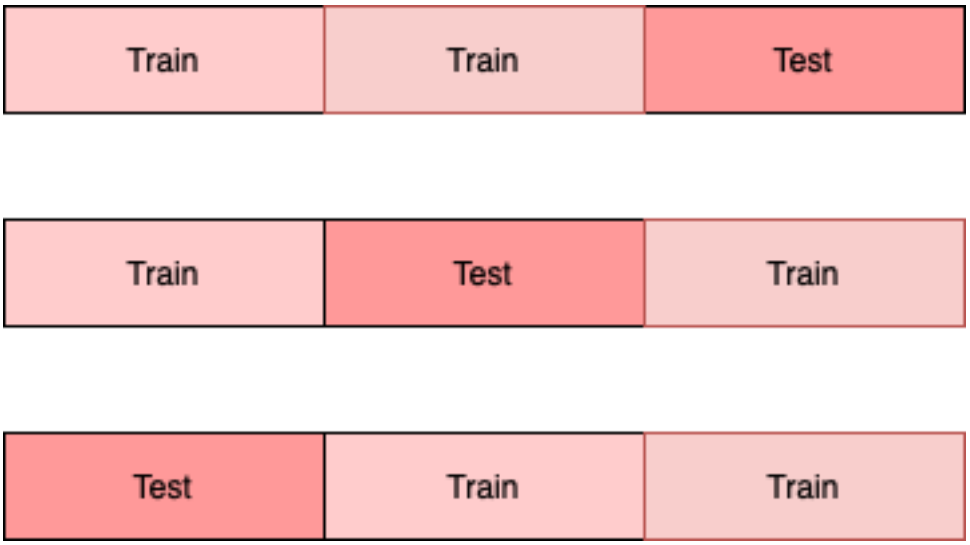
To obtain the final evaluation result, you can take the average of all splits' test set results. If you are satisfied with this result, you can train the final "champion" model by fitting the model to the entire dataset. This will be the model used for inference.

5.5.3 3. Nested cross-validation

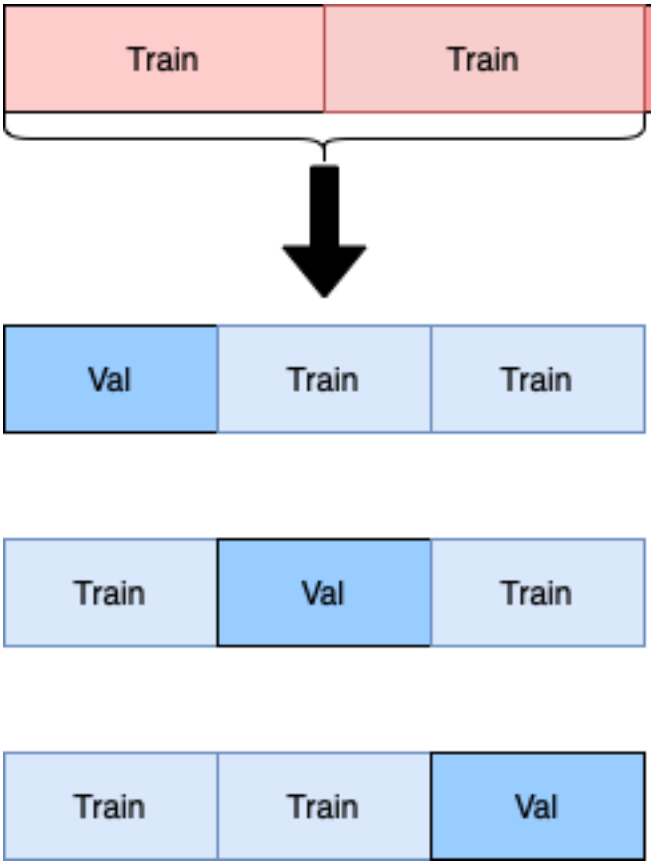
While cross-validation reduces the amount of overfitting as compared to the use of a static train test split, it does not reduce it completely. This is because the same score is used to select the best model and to evaluate the model.

To overcome it, you may choose to use nested cross-validation, which will separate the score used to select the best model and to evaluate the model. In this approach, the outer loop is used mainly for model evaluation, while the inner loop is used for hyperparameter tuning.

To illustrate this, let us assume you are using an outer fold of 3, and an inner fold of 3, and you are tuning and evaluating for the parameter n for a model. For the outer loop, you will split the dataset into 3 folds, and rotate the test fold for each set.



Taking the first training set, you will further split it into 3 folds with 1 fold serving as the validation set.



You will train a model with a value of n, say $n=1$, on the training folds, and evaluate the model on the validation fold. You repeat the model training with the same $n=1$ value each time you rotate the validation fold. The mean of the scores across all validation folds will be the validation score for the parameter $n=1$. You will repeat this procedure for all other n values, and then select the best n value.

Train	Test	n	Val Acc
Inn2,Inn3	Inn1	1	92
Inn1,Inn3	Inn2	1	90
Inn1,Inn2	Inn3	1	88
Inn2,Inn3	Inn1	2	80
Inn1,Inn3	Inn2	2	83
Inn1,Inn2	Inn3	2	77

n	Mean Val Acc
1	90
2	80

Finally, you will train on the full training and validation set with this best *n* parameter, and evaluate on the test set that was set aside initially in the outer fold.

You repeat the same procedure for all iterations of the outer loop, to obtain the final evaluation of the model.

Train	Test	Best n	Test Acc
Out2,Out3	Out1	1	92
Out1,Out3	Out2	2	84
Out1,Out2	Out3	1	88
Mean test accuracy: 88			

When you are satisfied with the results, you can train the final "champion" model by fitting the model to the entire dataset.

The drawback of the nested cross-validation approach is that the training time will even be longer than the standard cross-validation.

Tip: In cases of extremely small datasets, you could consider *nested cross-validation* to maximise the usage of the whole dataset for training, validation and testing.

5.5.4 Stratified Split

For datasets with unbalanced distribution of targets and/or features, you may want to consider stratified splitting. Stratified splitting aims to split your dataset, while maintaining similar proportions of any desired features/targets across your train, validation and test sets.

An example of a dataset with features that are skewed in their distribution will be a credit worthiness classification problem, where you may have fewer individuals in the dataset with ages of less than 20 years old. A random split could result in insufficient numbers of them getting assigned to the validation and test sets. If it is critical to model the behaviour of this age group, you can stratify the dataset based on age buckets. Doing so would fix the proportion of each age group getting assigned to each set.

An example of a dataset with imbalanced targets will be a fraud detection dataset, where fraudulent examples are typically the minority. In this case, you will stratify the dataset based on the target.

In general, the larger the dataset, the less likely features and targets will be unevenly distributed across the sets. You should still check your dataset for any uneven distribution in features and targets.

5.5.5 Temporal Split

When you are dealing with problems related to forecasting future values, you may want to consider temporal splitting. As an example, assume you have data from January to April. You may want to set aside data from January to February for your training dataset, March for your validation dataset, and April for your test dataset.

In fast moving environments such as fraud detection and cyber attack classification, bad actors might develop new fraud and cyber attacks techniques. It may be necessary to continuously train the model on the latest data to predict future frauds and cyber attacks, even though the task does not involve forecasting. Evaluating the model on historical frauds and attacks may be insufficient. Hence, you may choose to consider temporal splitting in this scenario.

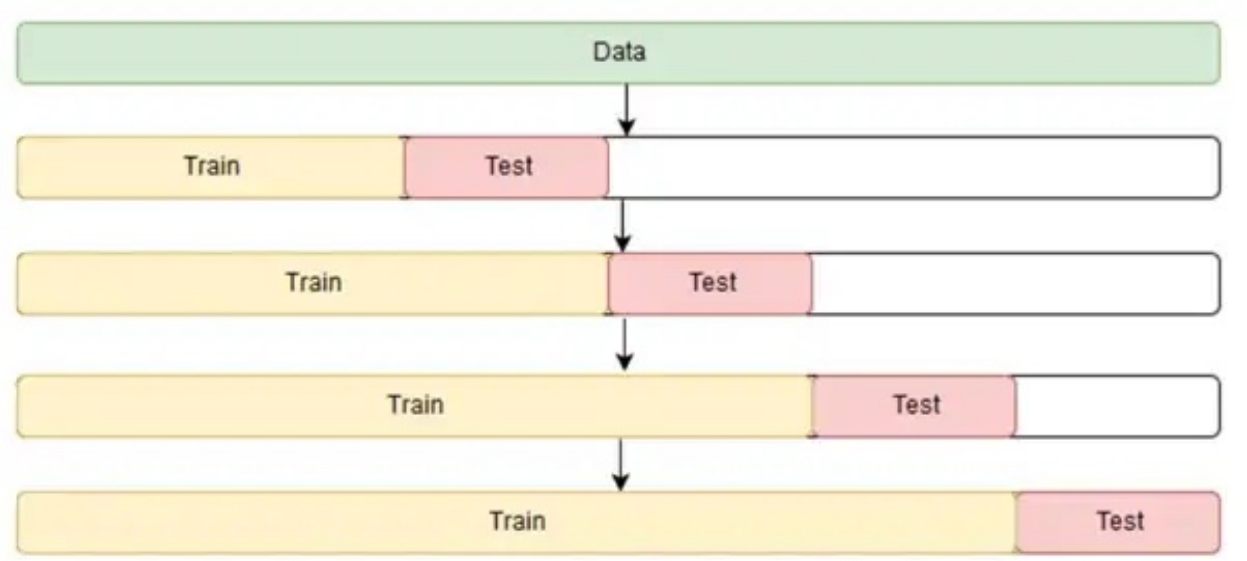
In scenarios where there are high correlations between successive times, such as in weather forecasting, you will also want to consider temporal splits and avoid placing February 2 in the training set and February 1 in the validation set to minimise data leakage.

For seasonal data, you may want to take seasonality into account when performing temporal splits. As an example, you can place the first 20 days of every month into the training set, the next 5 days into the validation set, and the last 5 days in the test set.

When performing cross validation for temporal data, you cannot simply use random data sampling to assign data points to the training and test sets. This is because there are temporal dependencies between each observation, and you generally want to avoid using future data to predict past data.

For each kth split in temporal cross validation, the first k folds will be assigned to the train set, while the k+1^{th} fold will be assigned to the test set. Hence, all successive training sets are supersets of previous sets.

An illustration of this is as follows.



Source: Cross Validation in Time Series

The steps to obtain the final evaluation results are the same as cross validation. You will likewise obtain the final result by taking the average of all k splits' tests results.

Similarly, when you are satisfied with the results, you can train the final "champion" model by fitting the model to the entire dataset.

5.5.6 References

- Building Machine Learning Powered Applications: Going from Idea to Product
- Machine Learning Design Patterns: Solutions to Common Challenges in Data Preparation, Model Building, and MLOps
- Machine Learning Yearning: Technical Strategy for AI Engineers, In the Era of Deep Learning

5.6 How do we make data splits repeatable?

Contributor(s): Lee Xin Jie, Senior AI Engineer (100E)

It is common for ML beginners to adopt naive random data splitting. In this approach, you may randomly select 80%, 10%, 10% of the data for your training, validation and testing data each time you run the algorithm. However, naive random data splitting is often not repeatable. While you may be able to fix random seeds to ensure some repeatability in data splitting on your machine, this repeatability is not guaranteed when the dataset has changed, or when your colleagues run the same data split algorithm on a different machine.

Having a repeatable train test split ensures that your experiments are reproducible not just by yourself, but by others.

Model reproducibility is discussed in detail later in the section How can I maximise model reproducibility.

Firstly, let's introduce the concept of hashing.

Hashing is a one way transformation of any given string or key into a hash value, which is often an output of a fixed length. A hash function is always deterministic, and the same input will always generate the same output. Since it is a one way transformation, you are not able to retrieve the original string or key from the hash value.

One approach to ensure the data splits are repeatable is to take a selected set of column/columns and hash it. You can then make use of the hash to split your data. This hash value will always be the same every time you run the algorithm.

Columns that are candidates for hashing should:

1. Not be features or labels that are used for training
2. Not be correlated to label
3. Be granular enough

Features and labels should not be used for hashing, as doing so may inadvertently introduce bias into the data splits. This is because data with certain features or labels may appear more in some data splits, and less in the other data splits. Columns selected for hashing should also be granular enough, hence columns such as year should not be selected.

Examples of columns that could be used are ID columns. Alternatively, you can concatenate all columns as a string and hash on that.

Once the column/columns are hashed, a simple approach for performing train-test split will be as follows. As an example, if you want 80% of your dataset to be allocated into the training set, 10% to the validation set and 10% to the test set, you can apply the modulo operator to the hashed value. Data points with remainders of less than 8 can be assigned to the training set. Those with remainders of 8 can be assigned to the validation set and those with remainders of 9 can be assigned to the test set.

Date	Store	Hash(Date,Store) % 10	Data Split
10 November 2022	ABC	6	Train
10 November 2022	XYZ	8	Validation
11 November 2022	ABC	3	Train
12 November 2022	DEF	1	Train

To do this in python, an example will be:

```
s = "10november2022abc"
hashed_value = hash(s)
remainder = hashed_value % 10
```

If there is a combination of columns that determine if 2 data points are correlated, and you want these data points to be grouped together in the same split, you can concatenate these columns together in a string and hash it.

Repeatable data splitting is not just limited to the above mentioned method. The key thing is to ensure the repeatability of the splitting process by any users on any machines.

For time series problems, you may want to adopt temporal data splits. You can adopt various data splitting approaches as long as you version control the data split logic to ensure its reproducibility.

Examples of applicable data splits are:

1. Select cutoff datetimes for data split, e.g. data before 2022-12-15 00:00:00.000000 to be in the training set, data after 2022-12-15 00:00:00.000000 to be in the validation set, etc.
2. Selecting the first 10 months of every year to be in the training set, 11th month to be in the validation set, and the 12th month to be in the test set.
3. Selecting the first 20 days of every month to be in the training set, the next 5 days to be in the validation set, and the last 5 days to be in the test set.

5.6.1 References

- Machine Learning Design Patterns: Solutions to Common Challenges in Data Preparation, Model Building, and MLOps

5.7 What are some scenarios of bias, unfairness, data leakage in data splits?

Contributor(s): Lee Xin Jie, Senior AI Engineer (100E)

5.7.1 Potential scenarios of bias and unfairness in data splits and suggestions to remedy

Having biases in datasets may lead to unfairness in our models. We want to ensure that model predictions are fair for different groups of users and scenarios.

There are 2 sources of such bias and unfairness:

- i) The full dataset is skewed to certain subgroups (eg. we are training a model to predict voting behaviour in Singapore but the dataset contains 95% Chinese).
- ii) The splitting of the full dataset into train, validation and test sets introduces skewness (eg. the train set contains 30% aged above 60 while the validation set contains 15% aged above 60).

This article discusses point ii).

Point i) is pertains to curating a representative dataset at the start of the project. This is partly discussed in the section What are the key areas to look out for in the data when framing the AI project?.

Certain problems are inherently imbalanced. For example, fraud detection datasets usually have fewer fraudulent examples than normal examples. In such a scenario, you should favour a stratified data split over a random split to ensure that there are equal proportions of fraudulent to normal examples in each data split. You will want to avoid a situation where your train set is predominantly normal, and your development set has a higher proportion of fraudulent data than in reality.

When dealing with datasets involving geographical and demographical slices, you may also want to consider stratified data split to ensure that each split contains a balanced set of examples across geographical and demographical attributes. This will help reduce any potential unfairness in model predictions between users from different geographical and demographical slices.

5.7.2 Potential scenarios of data leakage in data splits and suggestions to remedy

Data leakage is the scenario where during training, the model has access to information that will not be available during production. Often, if your model performs surprisingly well on the validation/test set (e.g. 100% test set performance), you should perform a check for data leakages. Data leakages should be avoided, as they allow the model to leverage on leaked information during training and achieve inflated performance on the validation/test set.

A common example of data leakage is **temporal data leakage**, which may occur when you are building a model to forecast future events based on past events. For example, you are building a model to predict the price of a stock in future, and you have past daily data of the stock's prices. If you perform a random data split, it is likely that the train set may contain future stock prices, while the test set may contain past stock prices. This will allow the model to leverage on future stock prices it encounters during training to predict past prices on the test set, thereby inflating the model's performance.

In temporal problems involving high correlations between successive times, you should also aim to group successive days together in the same data split to minimise data leakage.

Datasets with **duplicated data points** are also likely to cause data leakages. If a random data split is adopted, you may have the same data points in both the train and test set. This will result in inflated test performance.

Datasets with **near duplicates** can also subtly introduce data leakage. One example will be in computer vision, where it is possible to have multiple images that are near duplicates but not perfect duplicates. This can easily occur when the data comes from video streams made up of continuous sequences of images. For example, if you are training a model to classify car models, you may have images of the exact same car taken 1 second apart. In this case the images are likely to have very similar characteristics. If you split the dataset randomly, you are likely to split these images into each of the train, validation and test set. Your model will then likely evaluate well on the images assigned to the test set, since it has already encountered similar images during training. In this scenario, you should consider a stratified splitting of your dataset, where images of the same car should be assigned to a single dataset split. You could use timestamps or image hashing to achieve this.

When **engineering features**, ensure that you do not inject data leakage during the process. If you are building a model to predict how many items a user will purchase at a store on a particular day, you will want to only engineer features that are available during inference in production. Examples of features leveraging information that won't be available are the number of items sold in the store during the same day, or a moving average of the number of items purchased by the user up to and including the day itself, etc.

Having a **repeatable data split algorithm** is important to avoid introducing unforeseen data leakages during model retraining. Imagine the scenario where an unrepeatable random data split algorithm is adopted. When you have new datasets, you will likely generate new data splits and trigger model retraining. After retraining, you will often have to compare the performance of the old model vs the new model, to choose which model to deploy. An unrepeatable data split algorithm may mean that some of the old training data may fall into the new validation dataset. Hence, your old

model may appear to perform better on the new validation dataset and you will deploy that model. However you may soon realise that the actual model performance in production is lower after deployment.

5.7.3 References

- Building Machine Learning Powered Applications: Going from Idea to Product
- Machine Learning Design Patterns: Solutions to Common Challenges in Data Preparation, Model Building, and MLOps

5.8 How do I build a basic end-to-end workflow?

Contributor(s): Syakyr Surani, Platforms Engineer & Ryzal Kamis, Assistant Head (MLOps)

This guide assumes that you know basic data engineering and ML/DL modelling, and you want to start to build a workflow so that you have a working AI-powered product at the end of the project.

5.8.1 What Does It Mean to Have an End-to-End Workflow

When you started learning about ML, most likely you were only taught to train a model or perhaps to clean the data before training. Having an end-to-end workflow means you have a birds-eye view on the whole project from conception to deployment. This means you can delegate tasks better and diagnose issues much more effectively rather than overhauling the whole project when something unexpected happens and renders the project undeployable or unusable.

The flowchart below describes a typical end-to-end workflow of an ML project:

However, if you're just starting out in building an end-to-end workflow for your project, this simplified flowchart would be more suitable:

This guide will focus more on the simplified flowchart. If you are looking for what the components represent in the typical end-to-end workflow but not present in the simplified one, you can refer to *this section instead*.

5.8.2 Data Storage

This pertains to storing the lifeblood of all ML/DL projects, of which it is paramount that you know how and where to store data, and that you are able to identify raw and processed data. Raw data is defined as data that is obtained from primary sources such as temperature sensors, while processed data is defined as data that is augmented by algorithm(s) or from another ML/DL model.

It is also important that you know how the product is to be designed, such as how data is being fed into the model, or through a pre-processing algorithm first. You could ingest the data either locally or remotely, and remote options differ between using object storage for blob-like data such as image and audio files, database storage for text data, of which it may be structured (SQL, etc.) or unstructured (NoSQL, JSON, etc.).

Understanding different types of data storage is important since the data that you may be given may not be suitable to be fed into the model directly. For example, you might be given a compressed folder of Excel sheets, but directly processing them before feeding into the model may be suboptimal when you could parallise both processes and possibly reducing training time. This may also make resuming work easier since there are more checkpoints between the raw data and the model itself.

You can check *this section* for more information on data storage options.

5.8.3 Version Control

Most, if not all, codebases benefit from version control to reduce errors by rolling back changes to a known working source. You may already have been exposed to it through GitHub or GitLab, which are the leading platforms for source code management especially for open source projects. Understanding how to use version control effectively would make the end-to-end workflow easier.

For AI practitioners such as ourselves, there is also a subset of version control that targets toward datasets, which you may read this article for more information.

5.8.4 Developer Workspace

A workspace is the easiest to understand within the workflow, but can also be quite complex, especially pertaining to authentication and authorisation protocols that your organisation, or client, may have. Some reasons may be due to data being sensitive, the machine being airgapped, or that the code must be compatible with legacy hardware that you cannot interact directly with. Therefore, you may need to check with your project objectives and guidelines to see whether you may need to deploy a remote developer workspace as part of your project infrastructure depending on those protocols.

5.8.5 Model Experimentation

This is the main part, of which the workflow is designed around on. A well-designed workflow would make this process run more smoothly since you would not have the need to change the codebase drastically when any one of the other components have to change due to external events beyond your control.

5.8.6 Model Serving

This may well be the end goal for most, if not all, ML/DL projects. Typical tutorials would usually just end it with the model to be served locally through script execution, more mature projects would require a remote callable API since processes are usually interacted between machines rather than only within one machine. These APIs may usually use either the REST or gRPC protocol. You would have to decide which method is more suitable for your project.

5.8.7 Final Thoughts

There are many components and processes that can be part of your project's end-to-end workflow, but if you are starting out on the development process, this guide has recommended the processes you would want to focus on before expanding on other quality-of-life improvements that are described in *this section instead*.

5.8.8 References

- The Guide to Data Versioning | LakeFS

5.9 How do I enhance my end-to-end workflow?

Contributor(s): Syakyr Surani, Platforms Engineer & Ryzal Kamis, Assistant Head (MLOps)

This guide assumes that you have built a basic end-to-end workflow and want to scale up your workflow while increasing robustness and reliability. If you want to know more about building a basic workflow instead, then you can go to *this section instead*.

As per from that section, this flowchart describes the typical end-to-end workflow of an AI project:

This section deals with the other components and processes that are present in this workflow, but not the simplified version in the other section.

5.9.1 Artifact Storage & Governance

Artifacts pertain to data that are not used in training the models in your project. This can be your configuration settings, logs, etc. On the other hand, data and artifact governance refers to the authentication and authorisation protocols that are discussed previously in the Developer Workspace section.

5.9.2 CI/CD

Continuous integration and continuous delivery/deployment processes pertain to the automation of various processes within the workflow itself. While this would benefit end-to-end workflows greatly, it is not implemented within many ML/DL projects due to its complexity and time needed to design an effective CI/CD process. If you do want to know more about CI/CD, you can refer to *this section* instead.

5.9.3 Model Registry

This could be seen as a version control for models to manage problems such as model drifting and retraining. You may need this component as your project scales up, and you need to manage multiple models within the project as well as across multiple projects that may use the same model(s). This component would also make A/B testing easier since you could reference multiple versions of the same model to gauge any improvements a new version may provide.

5.9.4 Container Registry

Containers such as Docker abstracts the codebase while providing a sandbox to reduce dependency conflicts while making component management easier. This allows better scaling and modularity by setting manageable blocks according to their use case instead of by just the products themselves. However, you should take note whether your project requirements allow containerisation as some policies within your organisation/client may be against this process due to the overheads it may have, especially if the project does not require to be designed for scaling.

5.9.5 Experiment, Pipeline & Model Monitoring/Tracking

The experiment and pipeline tracking as well as model monitoring can be seen as similar components that target different processes. Experiment and pipeline tracking focuses more towards monitoring the models as they are being trained, and from there, decisions are being made through controlling the hyperparameters of the model, or use a different model architecture altogether.

On the other hand, model monitoring concerns overseeing the trained models and see whether issues are present while it is used for inference such as model drifting. These metrics would be collected and processed to make decisions such as controlling data input requirements or retraining the model.

5.9.6 Final Thoughts

There are many more components and processes that could be part of your end-to-end workflow that are not discussed in this as well as the basic workflow sections, but the ones that are discussed should be sufficient for you to build a robust and reliable AI end-to-end workflow. If you want to know more about this and MLOps in general, you can take a look at ml-ops.org for more insight.

5.9.7 Reference

- MLOps: Machine Learning Operations | InnoQ

5.10 How can I reduce the risks of data poisoning and data extraction?

Contributor(s): Kew Wai Marn, AI Engineer

5.10.1 Data poisoning

Ideally, data should be collected and labeled in a controlled and safe environment. Practically, this is time-consuming, and thus expensive, which not everyone can afford. Therefore, data is sometimes collected from the Internet or other untrusted sources. This poses a huge risk as an adversary can intentionally manipulate the data, which causes the ML system to be compromised. For example, training data can be injected with specific features that causes the model to fail during inference when inference data with such features are used; i.e. model "backdoors".

5.10.2 Data extraction (inference/inversion attacks)

Other than manipulating the data, an adversary may extract details of the training data by querying the model or inspecting it directly. When the adversary has unlimited query access to the model, they can use that to predict whether or not a particular example was contained in the model's training dataset.

5.10.3 How to check your data sources

In order to reduce the risk of data poisoning and extraction, you should check the following in your ML system:

1. Proper access control to protect the raw and processed data from unauthorized users.

To prevent data manipulation, the first thing to do is to secure it (e.g. only authorized personnel are allowed to access within a specified time frame).

2. Ensure the integrity of the data sources for your ML system

Other than securing your data, it is important to know about the data's history ie. data generation, collection, and assembly process, especially if the source is public. In addition, list down any potential considerations for data quality (e.g. trustworthiness, reliability, etc), so that it is easy to backtrack when necessary.

Examples of considerations and measures:

- How do you ensure that the raw data was not manipulated or poisoned, even by authorized personnel?
- Have you implemented measures in the data collection process to ensure the reliability of the data collected? (e.g. if your raw data is collected from sensors, possible measures to ensure reliability include regular re-calibration, and/or redundant streams with multiple correlated and overlapping sensors)
- How do you assure the quality of your labeling process, and validate the resulting labels?

The main idea is to not use data blindly. Take time to understand the data source,. You can also analyse the data to look for inconsistencies i.e. look at feature value range, statistics ie. mean for numerical data, data frequencies for categorical data.

3. Ensure that the model output is not part of the input

This is important for models using data crawled from the internet. For example, when translations of pages done by the model were used as training data for the model.

4. Model only outputs what the user requires (e.g. no confidence score unless necessary)

This is to prevent an adversary in trying to extract information from your model. For example, for a image classifier, knowing confidence score can allow the adversary to predict whether or not a particular image was contained in the model's training dataset.

5. Limit queries by users

As mentioned earlier, if an adversary were given unlimited query access to a model, they can bombard the model with potential data candidates, in order to predict the distribution of the training data.

6. Anonymize the dataset

As the last line of defense, annoymizing the dataset would prevent any personal identifiable information (PII) from leaking, as per Singapore's Personal Data Protection Act (PDPA). However, by itself this is not enough as it does not protect the proprietary data.

Reference(s):

- BIML Architectural Risk Analysis
- Machine Learning Security against Data Poisoning: Are We There Yet?
- Just How Toxic is Data Poisoning? A Unified Benchmark for Backdoor and Data Poisoning Attacks
- Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures
- PDPA Overview

CHAPTER

SIX

MODELLING

6.1 Overview

This chapter includes sections discussing model training, evaluation and error analysis. Additionally, articles on model explanability and interpretability, and machine learning risks are also included in this chapter.

6.2 What are some internal and external considerations when selecting evaluation metrics?

Contributor(s): Er YuYang, Senior AI Engineer

This guide assumes that you have a fair understanding of the business value of the AI/ML project, and have an appreciation of the implementation difficulty of the evaluation metrics. Internal considerations refer to those concerning the development team, while external considerations refer to those concerning business users/stakeholders.

6.2.1 1. Understanding the problem space

Problem statement

Understanding how the AI/ML solves the problem statement helps to identify the correct optimising metrics (explained in section 2) for assessing the performance of the model. Complementarily, understanding the deployment environment helps in choosing the satisficing metrics (explained in section 3). In most cases, optimising metrics tend to be ML metrics, while satisficing metrics tend to be non-ML metrics.

There is often a trade-off between ML and non-ML metrics. For instance, a more complex model can return better ML metrics, but can also come with longer training and inference times (non-ML metrics). As such, developers and business users are faced with a choice as to which metrics to prioritise based on the business value of the AI solution.

Technical ability of stakeholders

It is important to understand the level of technical ability possessed by eventual end-users and/or maintainers of the AI/ML solution. This understanding can help identify metrics that these stakeholders are *comfortable* with.

Comfortable metrics are important for for ease of communicating with the stakeholders. Importantly, they allow future technical teams to understand and improve the AI/ML model in later stages of its life cycle.

6.2.2 2. Optimising metrics

The goal here is to establish a single-number evaluation metric, as multiple evaluation metrics make it hard to compare between different models.

For instance, Classifier A has precision of 95% with recall of 90% and Classifier B has precision of 98% with recall of 85%. Neither classifier is obviously superior and does not immediately guide you towards picking one. In this case, you may consider taking the F1 score, which is the harmonic mean of precision and recall.

To obtain a single-number metric for multi-class classification problems, you can combine metrics of each class into single metric using average or weighted average. However, keep the metrics separated if you are evaluating intraclass performance instead of overall model performance.

6.2.3 3. Satisficing metrics

In theory, optimising metrics should be maximised as much as possible. In practice, however, practical trade-offs/compromises have to be made. These take the form of satisficing metrics, which are often non-ML related. Examples of satisficing metrics include:

- Running/inference time
- Training time
- Memory/storage usage (particularly applicable for edge deployment)

It is not practical or cost-efficient to increase optimising metrics by a minuscule percentage while making disproportionate trade-offs on satisficing metrics. For example, increasing model accuracy by 0.0001 may required significantly longer and more expensive computation time. Deployment considerations (eg. size of model, runtime) may also come into play here.

6.2.4 4. Combining optimising and satisficing metrics

Consideration of both optimising and satisficing metrics means that there are at least two sets of numbers to optimise. How can we adhere more closely to the more elegant single-number metric approach?

One approach is to derive a single metric through a forumla. For instance:

```
metrics = accuracy of model - 0.5 * runtime
```

Another approach is to first define an "acceptable" threshold for the satisficing metric through consultation with stakeholders. The development team can then work to maximise the optimising metrics within acceptable bounds of the satisficing metric.

Once your team is aligned on the evaluation metric to optimise, they will be able to make faster progress.

6.2.5 References

Machine Learning Yearning, Chapers 8 and 9

6.3 How can I maximise model reproducibility?

Contributor(s): Kew Wai Marn, AI Engineer

6.3.1 Model reproducibilty

Reproducible models are important because they allow independent practitioners to achieve the same results, which improves communication and collaboration. Furthermore, reproducible models are less error-prone and more reliable.

Apart from ensuring reproducibilty in the data and the code of the ML system, you would need to check your model logic as well. Assuming you have the same training code, and the same training data, you should be able to produce the same model.

6.3.2 Model reproducibility checklist

If you have the same training code, and the same training data, but you are not able to produce the same model, below are several things you should check to ensure reproducibility of your model. The main goal is to reduce non-deterministic behaviour in ML modelling as much as possible.

1. Model tests

Apart from testing code, you should test your model as well. Below are several things to test your model for.

a. API calls

One of the things you can test is to test the API calls. As ML frameworks and libraries are constantly getting upgraded, you need to make sure the API calls are working as expected. Write unit tests with random input data running through the API call (ie. a single step of gradient descent).

To reduce the risk of API calls breaking, you can version the libraries used; eg. create a software enviroment with specific and static dependency versions recorded in a requirements file.

b. Algorithmic correctness

Other than testing the API calls, you should make sure your model's performance is not due to luck.

To check for algorithmic correctness you can:

- Verify that the loss decreases with increasing training iterations.
- Verify that without regularization, the training loss is low enough. If your model is complex enough, it will capture information from the training data.
- Test specific subcomputations of your algorithm. (e.g. test that neural network weights are updated with every pass).

c. Test that model is stable

A stable model would be reproducible as it produces consistent results. When training a neural network, your weights and layer outputs should not be `NaN` or `Inf`. A given layer should also not return zeroes for more than half of its outputs. Write tests to check for `NaN` and `Inf` values in your weights and layer outputs.

Furthermore, perform sensitivity analysis on the hyperparameters of your ML system to ensure system hyperparameter stability.

2. Deterministically seed the random number generator (RNG)

Apart from fixing the seed value for data splits, you should use a fixed seed value across all the libraries in the code base in order to reduce non-deterministic behaviours eg. weight initialization, regularization, and optimization in neural networks, split points in random forest algorithm.

Seeding the RNG from Python (i.e. `random.seed()`) is not enough as some packages have their own implementation of a pseudo-random number generator (e.g. NumPy). Do check their documentation on how they generate the random numbers.

However, there will still be areas of randomness that are irreducible, such as randomness in third party libraries and in GPUs configurations. Please refer to this *section* for more information.

3. Model version control

Other than versioning the data, model versioning is important to ensure reproducibility. By taking snapshots of the entire ML pipeline, it is possible to reproduce the same output again, with the trained weights. Each model version should also be tagged to it's corresponding metadata (ie. hyperparameters), training data, and code. Model registries and feature stores help in mainining model versions.

4. Integration tests

As a ML pipeline consiste of several components, tests that runs the entire pipeline end-to-end should be written. To run integration tests more quickly, train on a subset of the data or with a simpler model.

5. Logging

It is important to log everything used in ML modelling (e.g. model parameters, hyperparameters, feature transformations, order of features, method to select them, structure of the ensemble (if applicable), hardware specifications). This would help to recreate the environment the model was created in. Furthermore, dependency changes (e.g. changes in API calls) should be recorded.

6.3.3 Inherent stochasticity

As mentioned earlier, the ML pipeline is not always entirely reproducible. Inherent stochasticity exists in some components, such as when using GPUs, randomness in backend libraries, and stochastic ML algorithms. In order to improve reproducibility, you can report the average performance of your model using multiple runs or even build an ensemble of models, each trained with a different random number seed. The last resort is to record down any potential stochasticity that exist.

Reference(s):

- Testing for Deploying Machine Learning Models

- Testing Pipelines in Production
- AWS Whitepaper Reproducibility
- Building a Reproducible Machine Learning Pipeline
- Reproducible AI: What it is, Why it Matters & How to Improve it?
- Introduction to Random Number Generators for Machine Learning in Python
- Embrace Randomness in Machine Learning

6.4 How do I assess model robustness?

Contributor(s): Kew Wai Marn, AI Engineer

6.4.1 Overview of Machine Learning Risks

Risks of machine learning (ML) systems can come from data, model, and/or infrastructure. According to the architectural risk analysis by the Berryville Institute of Machine Learning, the top risk is adversarial examples. This is when an attacker tries to fool the system by giving malicious input in the form of small perturbations that cause mispredictions. These perturbations are derived from a model's vulnerabilities, which are areas where the model fails to perform; ie. predict incorrectly.

Apart from intentional attacks, risks can also come from intrinsic design flaws which can similarly lead to incorrect predictions.

6.4.2 What is ML Robustness?

According to Singapore's Model AI Governance Framework, ML robustness is defined as whether the system can function with unexpected inputs. The ability of the model to perform with unexpected inputs affects the reliability of the ML system. Unexpected inputs can be from adversarial or non-adversarial origin.

It is important to consider ML robustness, because non-robust models can fail unexpectedly. For example, consider a road sign detection model deployed in an autonomous vehicle. Misclassify a stop sign as a speed limit sign could result in life-threatening consequences.

Adversarial Robustness

It is important to consider adversarial robustness, because adversarial examples are the top risk to ML systems (as mentioned earlier). Adverserial robustness is defined in terms of the minimal perturbation value that the attacker must introduce for a successful attack (ie. model fails to infer correctly with adversarial input). If a model is adversarially robust, it will require more pertubations for a sucessful attack. It is important to note that there are many other metrics that can be used to measure adversarial robustness (CLEVER, loss sensitivity, etc.).

Non-adversarial Robustness

On the other hand, apart from attacks, the ML model itself could be flawed (not able to perform as it should be). Unexpected inputs, mentioned in the definition above, can be quite ambiguous. One type of unexpected inputs can be out-of-distribution (OOD) data. OOD data are data that has a different distribution from your training data but is still within the problem scope where it might appear in production. For example, images from a different camera of the same brand.

This risk can be reduced if data representativeness was considered in the data collection phase, as mentioned in Chapter 1.

6.4.3 Robustness Testing

One way to ensure robust ML systems is to do robustness testing, where we can anticipate the robustness of the models using adversarial examples and OOD data.

For adversarial robustness testing, the goal is to test how resilient the model is to adversarial examples, or how easily adversarial examples can be created from the model.

For non-adversarial robustness testing, the goal is to test where the model fails using OOD data.

There are two ways to obtain OOD data: either collect from another source, or to generate it by applying metamorphic mutation. Metamorphic mutation comes from software testing, where we mutate data that was inferred correctly by the model in a way that does not change the label (label-preserving mutations); eg. horizontally flip an image.

For text, it is slightly trickier as valid mutations depend on your model's objective. For example, adding typos would be a valid mutation if your model is supposed to be invariant on typos ie. not a spell checker.

For tabular data, mutation strategies are currently still under active research.

Robustness Testing Tools

Here are some tools you can try for specific types of data, model and task:

Tool	Data Type	Model Type	Task Type (Nature of model output)
IBM ART, Adversarial	Generally arrays of numeric data (i.e. Images, Audio), but is attack specific	Any for black box attacks, specific model type for specific white box attacks	Attack-specific, Mainly classification
TextAttack, Adversarial	Text	Any for black box attacks. specific model type for specific white box attacks	Classification, Sequence-to-sequence
Microsoft Check-List. Non-adversarial	Any arbitrary format (as the mutation functions can be user-defined). Only some text mutations supported in tool.	Any	Any (as the expectation functions can be user-defined)

When To Perform Robustness Testing?

Using the robustness testing tools, you can test your model in the CI/CD pipeline, at the same level as model evaluation, to get a good evaluation of your model. Just like unit testing or integration testing, you only need to set it up once. With every new retraining, you can compare the robustness between model versions.

6.4.4 Words of Caution

The goal of robustness testing is to find areas where your model does not perform, and improve on them, rather than deciding whether the model is robust or not robust. There is always a need for comparison between models versions (especially for adversarial robustness testing). Testing the robustness of one model does not tell you anything other than where it fails. Lastly, you might find that standard accuracy might have a trade off with robustness, this is still (as of writing) an area under research.

Reference(s):

- BIML Architectural Risk Analysis
- Singapore's Model AI Governance Framework
- DeepFool
- IBM-ART
- Microsoft CheckList
- TextAttack
- Test ML like software
- Metamorphic Testing

6.5 How do I select classification metrics?

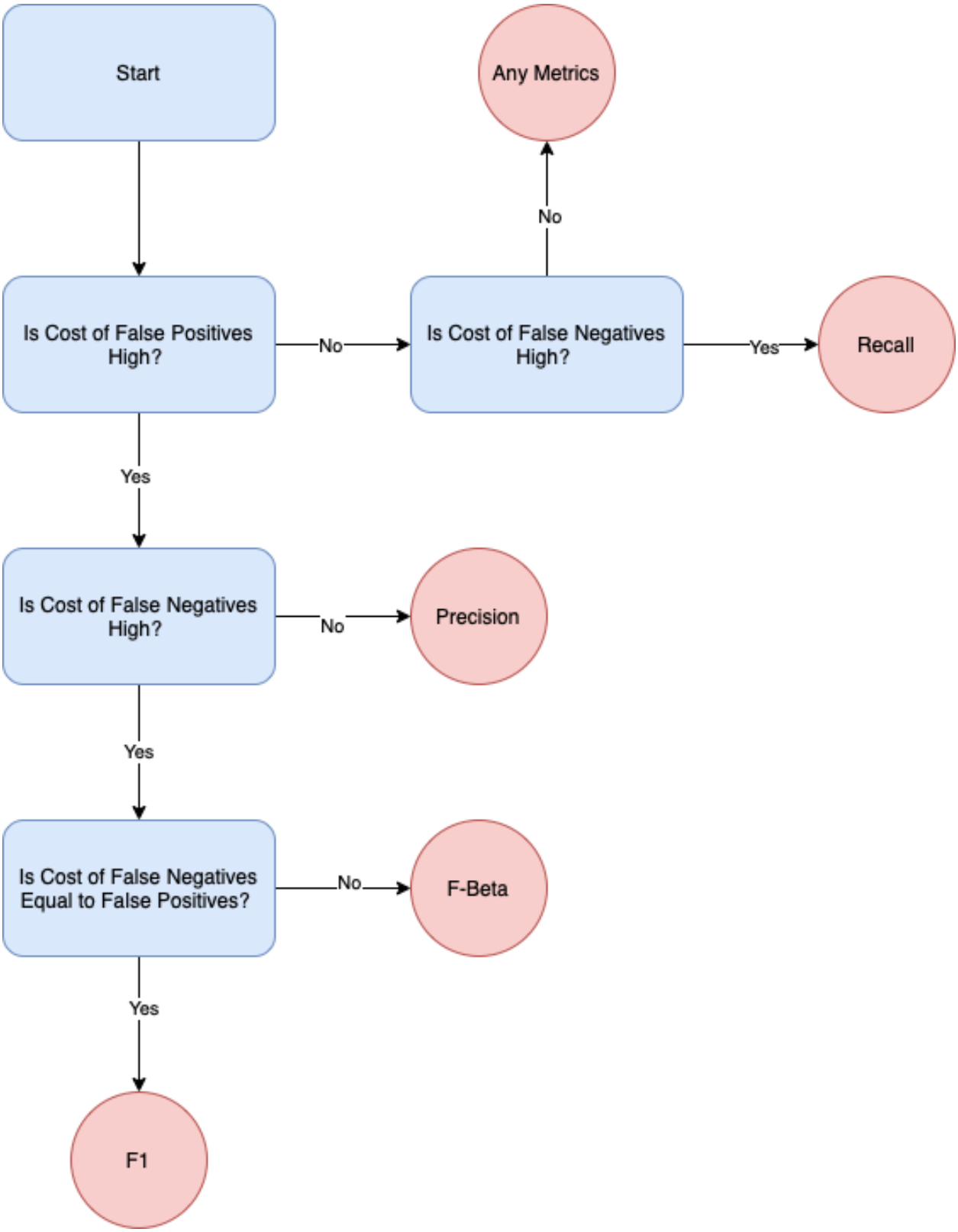
Contributor(s): Lee Xin Jie, Senior AI Engineer (100E), Er YuYang, Senior AI Engineer (100E)

6.5.1 Introduction

The common classification evaluation metrics includes accuracy, confusion matrix, precision, recall, F1, ROC & AUC, precision-recall curve.

This chapter will not elaborate on the definitions of the above mentioned evaluation metrics, as there are numerous resources available for this topic. Instead, this chapter will introduce best practices regarding the selection of classification metrics. This will serve as an extension of the previous chapter *What are some internal and external considerations when selecting evaluation metrics*, and the concepts discussed here can extend through various domains, be it Natural Language Processing, Computer Vision, etc.

6.5.2 Imbalanced datasets



While accuracy is the most common classification metrics, it can be misleading for imbalanced datasets. For example,

consider a fraud detection problem where only 5% of the labels are fraudulent. If the model predicts every example to be non fraudulent, it will achieve an accuracy of 95%, which can be very misleading.

In such situations, you may want to consider using precision, recall and F1. Precision measures of all your model's positive predictions, how many of them are correct. Recall measures of all the actual positive examples, how many of them were identified correctly by the model.

Precision is more important than recall when you cannot afford to have any False Positives as compared to False Negatives. Do take cost into consideration when deciding if precision or recall is more important. When the cost of acting is high and the cost of not acting is low, then precision is preferred. Recall is more important than precision when you cannot afford to have any False Negatives as compared to False Positives. It is more important when the opportunity cost of passing up is high.

The F1 score, also known as F-score, is the harmonic mean of precision and recall. F1 can be computed with the formula. You should choose this metric if you want to balance both precision and recall.

$F_1 = 2 \cdot \frac{precision \dot{} recall}{precision + recall}$

Since there is often an inverse relationship between precision and recall, you might want to weigh one more over the other. This will likely depend on the client's needs, hence do check with them on their needs before proceeding. You may also want to consider the F_β score:

$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$

where β is chosen such that recall is considered β times as important as precision.

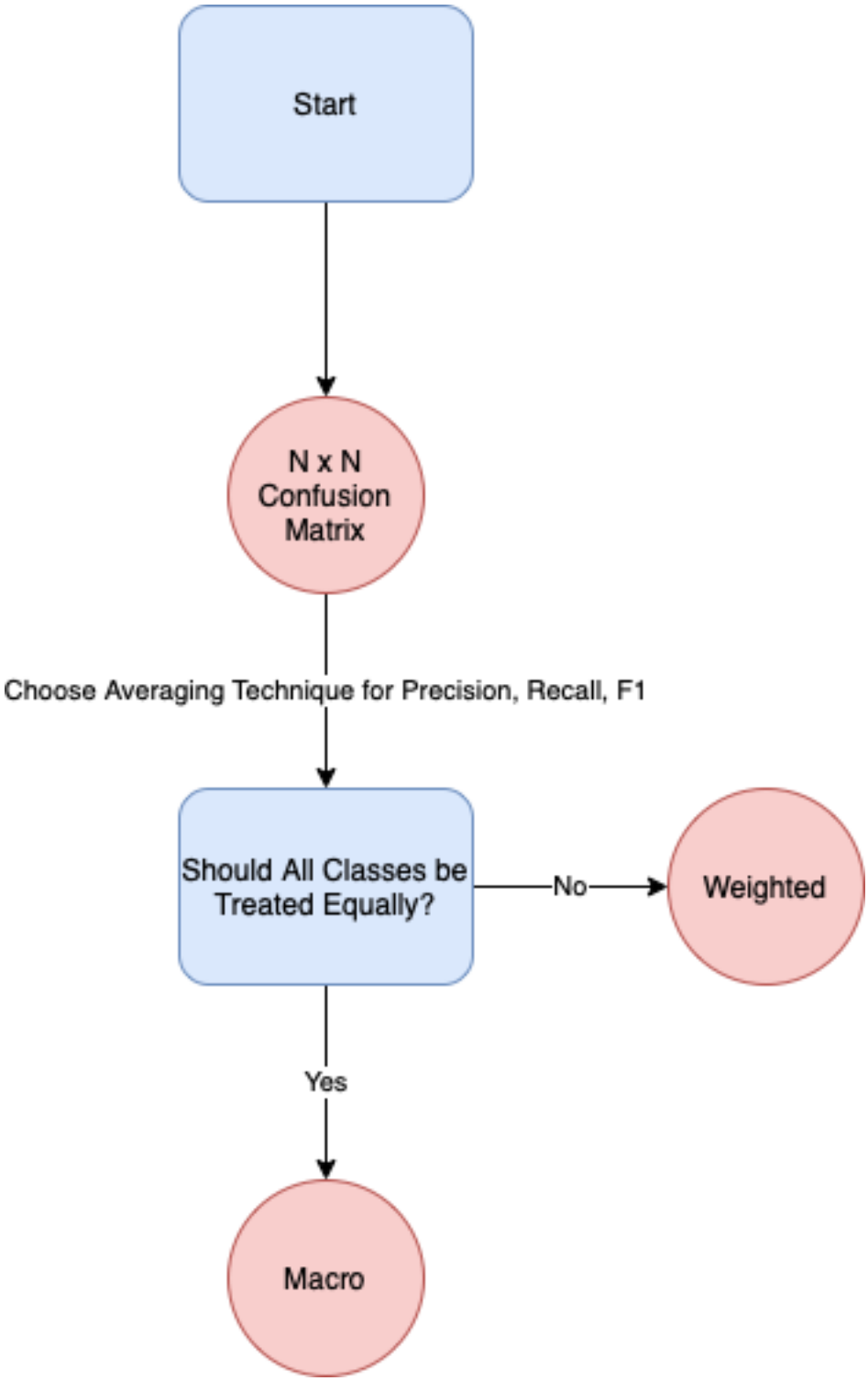
Why not ROC?

The ROC curve, also known as the receiver operating curve, plots the true positive rate (TPR) vs the false positive rate (FPR) over all classification thresholds from 0 to 1. The AUC, also known as the area under the curve, measures the aggregated performance of your model over all classification thresholds.

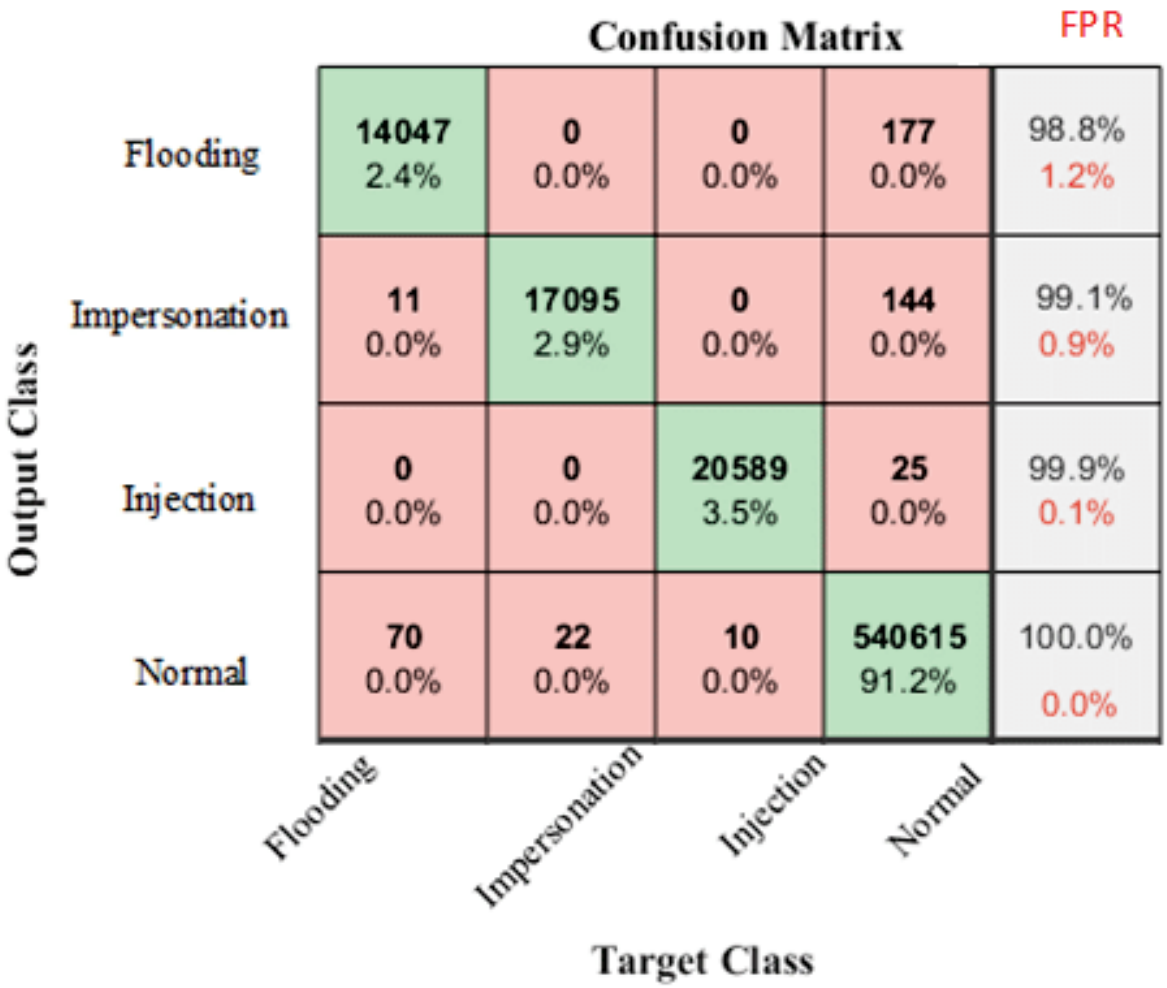
The ROC curve, as well as the AUC score, can be misleading in imbalanced datasets. As an example, assume that the positive class is the minority class in an imbalanced dataset. Because the negative class has a lot of examples for the model to learn, the True Negatives are typically very high, especially when compared to the number of false positives. This means that the FPR will likely be low even when there are alot of false positives. Hence this implies that it is possible for a low precision model to achieve a good ROC curve and a high AUC score.

To illustrate this, assume there are 10 positive examples and 1000 negative examples. There are 9 True Positives, 1 False Negative, 100 False Positives and 900 True Negatives. The TPR is 0.9 while the FPR is 0.1. Since the TPR is high and the FPR is low, the AUC is likely high. This does not mean that the model is good, as the model's precision is relatively low. In such a scenario, it is preferable to look at the precision recall curve which plots the precision and recall across all thresholds. Likewise, you are also able to compute the AUC of the precision-recall curve, which you can use to compare the performance of various models.

6.5.3 Multi-class datasets



In multi-class classification, the first evaluation metric you may want to consider is an N $\times$ N confusion matrix. An example of which is shown below.



Source: Artificial Intelligence Technique for Gene Expression by Tumor RNA-Seq Data: A Novel Optimized Deep Learning Approach

With an N $\times$ N confusion matrix, you can quickly identify which classes are performing well, or which classes are commonly misclassified as another class, etc.

To extend precision, recall and F1 to the multi-class domains, we can first compute these scores on a per class basis by using the one-vs-rest approach. This is akin to treating the evaluation of multi-class problems as the evaluation of multiple binary classes.

If you need a single score to compare models, you can compute the averages of precision, recall and F1 using three different averaging techniques.

1. Micro: Computes metrics globally by counting the true positives, false positives and false negatives.
2. Macro: Computes the unweighted mean of the metrics for all classes, disregarding class imbalance.
3. Weighted: Computes the metrics for each class, then finds their average weighted by support (the number of true instances for each label).

The choice of which averaging techniques to use depends on the business objective. If you prefer to treat all classes equally, regardless of their sample size, you can opt for macro-averaging. This could be for situations where the minority class performance is very important to the project sponsor.

On the other hand, if it is acceptable to treat each class's importance based on the class size, you can opt for weighted- or micro-averaging. Do note that for multi-class problems, the micro-averaged F1 score will be the same as the global accuracy.

6.5.4 Multi-label datasets

For multi-label problems, you can extend the binary classification metrics by computing each metric per class. This is no different from a binary classification problem. For example, you can compute precision, recall, F1, confusion matrix for each of the classes in a one vs rest approach. Once you have obtained each class's metrics, you can compute the averages across all classes with the same averaging techniques as covered for multi-class problems: micro, macro, weighted averaging. Note that for multi-label problems, the micro-averaged F1 score is not the same as accuracy.

6.5.5 References

- Comprehensive Guide to Multiclass Classification Metrics

6.6 What are some common CV evaluation metrics?

Contributor(s): Calvin Neo, AI Engineer

This guide assumes a fair understanding of the computer vision-related tasks specifically image classification, object detection, and instance segmentation and their accompanying terms such as bounding boxes (bboxes) and masks. Additionally, it assumes an understanding of classification metrics such as F1 score, precision and recall.

The purpose of this article is to guide AI engineers on key considerations when evaluating metrics for object detection and instance segmentation tasks. For the most part, conventional classification metrics can be used to evaluate image classification tasks. Hence, an understanding of image classification would be useful here. In this case, refer to the article on *classification metrics*.

6.6.1 Mean Average Precision (mAP)

There are many articles on how to calculate mAP available. It is thus instructive to instead discuss considerations when reading mAP.

In discussing mAP evaluation, consider the output below:

Average Precision	IOU	Area	Max Detections	mAP
Average Precision	0.5	all	1000	0.743
Average Precision	0.75	all	1000	0.213
Average Precision	0.50:0.95	all	100	0.326
Average Precision	0.50:0.95	small	1000	0.127
Average Precision	0.50:0.95	medium	1000	0.349
Average Precision	0.50:0.95	large	1000	0.406

Where mAP for IOU = 0.50:0.95 measures the AP values at IOUs 0.50 to 0.95, with a step of 0.05, and then averaging the values.

Firstly, it is useful to evaluate the Intersection over Union (IOU) threshold value to understand how "tight" a bbox/mask. A higher IOU threshold value indicates that the model's inference closely overlaps with the ground truth, and is hence

a more stringent evaluation criteria. Generally, the mAP suffers as the IOU threshold value increases. Thus, evaluating how much these values fall would be valuable clues on how well the model may produce tight bboxes/masks.

Using the outputs above, the second row shows that at IOU threshold of 0.50, the mAP is given by 0.743. Similarly, the third row showing IOU=0.75 threshold shows 0.213, a 0.53 fall in value. This indicates that the bboxes/masks produced by the model were not tight enough, and more training may be required.

Secondly, the objects being detected can be broadly separated to small/medium/large objects based on the bbox/mask size. Here, it is assumed the small/medium/large are area of the bbox/mask, and that area threshold for these sizes are clearly defined. For more information on how to determine small/medium/large objects, see article on *Exploratory Data Analysis for Compter Vision Tasks*.

A model generally performs the worst on small objects and gets better as the size increases. Thus, evaluating this is also instructive. Considering the above outputs, the mAP is 0.127/0.349/0.406 on small/medium/large objects. Thus, it would be good to have the model be trained on more small objects in order to potentially improve its performance.

6.6.2 Frames Per Second (FPS)

FPS are generally measured by running multiple images through the pipeline and obtaining the amount of time taken to process each image, before converting it to frames per second. The way to do it is:

$$FPS = \frac{1}{\text{Average Time Taken Per Image}}$$

Here, the AI Engineer would need to consider the following:

FPS should be calculated based off either model's or the software architecture's inference speed. For model's FPS, it is simply the inference speed solely on the model, without taking into account other processes like data ingestion/pre-processing and inference post-processing. The software architecture's inference speed considers all the different software components that is related to the inference such as the above mentioned processing steps.

In most cases, though, the FPS of concern would be solely on the model since the main concern is to deliver a model. This is especially so when the concern of FPS is complicated by other factors outside of the model and beyond the AI Engineer's cotrol, namely internet connections and hardware. Whichever way, the onus is on the AI Engineer to communicate how the FPS was measured to manage expectations.

It is also important to consider what affects FPS. The first factor would be model size. A model that is deep, and hence tend to be larger in size, requires a longer processing time. Another factor is image size. Larger image size will take longer for inference. For example, a 416x416 image will be faster than 1080x960 image. While again this is obvious, it is also often overlooked by AI Engineers when reading papers that purport higher inference speeds, but downplays the role that smaller images have on inference speed.

6.6.3 Trade-off Between mAP and FPS

When a model has been identified based on mAP and FPS, further refinements can be made that may involve trading FPS for mAP or vice versa. For example, for certain models, the images that is passed in can have its size further reduced. The advantage here would be that inference speed is higher, and hence the FPS will be higher as well. However, since the image is reduced, the model may not have enough features to extract and map, further reducing its ability to accurately infer from the image. This drives down mAP. Increasing the image size will have the opposite effect.

Hence, it is important to establish "good enough". That is, the engineer would need to work with the sponsor to come up with the metric threshold that would satisfy their use case. For example, if a model currently has mAP = 0.50, and FPS = 60, and the engineer knows that inference speed, for various reasons, would only need to be around 30, then they would be able to increase image size further to potentially increase mAP while accepting the FPS reduction.

It is also good to bear in mind diminishing returns. In the case above, if the FPS went down to 30 and the mAP went up to 0.52, would the trade-off be worth it? No in most cases. Thus, running through various parameters to find a good trade-off is important in such cases.

6.6.4 Adjusting Thresholds for Inference - Confidence and Non-max Suppression (NMS)

When a model makes an inference, it will give a confidence level of the object's class. If the confidence does not meet or exceed the threshold, then it will not be included.

Similarly, when a model makes duplicative inference on the same object, meaning it has two or more bounding boxes on the same object, there has to be a mechanism to suppress these duplicative boxes.

Thus, the engineer may use the Confidence and/or NMS threshold to ensure to adjust the precision and recall of the model.

To show this, consider the output below. Such a table can be generated by running the model on a combination of possible threshold values.

Confidence	NMS	IOU	Precision	Recall	F1 Score
0.3	0.1	0.5	0.697	0.752	0.724
0.5	0.1	0.5	0.750	0.732	0.741
0.7	0.1	0.5	0.811	0.697	0.750
0.3	0.4	0.5	0.640	0.789	0.707
0.5	0.4	0.5	0.718	0.759	0.738
0.7	0.4	0.5	0.794	0.717	0.754
0.3	0.7	0.5	0.579	0.800	0.672
0.5	0.7	0.5	0.682	0.768	0.722
0.7	0.7	0.5	0.774	0.723	0.747

It is clear here that Confidence and NMS threshold can appear to be at odds with each other. As Confidence Threshold is increases, the Precision increases, but Recall is lower. Conversely, a higher NMS threshold gives better recall.

6.6.5 Prioritising Precision or Recall

As stated above, the AI Engineer has to continually engage with the product owner in determining which to prioritise. Obviously, they can also rely on F1-Score to harmonise the two. However, do keep in mind the following:

1. Choose Precision if quality is the the priority of the model. That is, it needs to identify objects and their classes.
2. Choose Recall if quantity is the priority of the model. That is, it needs to identify many more objects without caring too much about whether the class was correctly identified.

6.6.6 Conclusion

This article addresses the conventional metrics for object detection and instance segmentation in the context of pushing a product. Beyond simply calculating these metrics, the AI Engineer would do well look out for granular details when evaluating the metrics in order to give a more accurate picture of model performance.

6.7 What are some metrics for Named Entity Recognition?

Contributor(s): Lee Xin Jie, Senior AI Engineer (100E)

6.7.1 Named Entity Recognition

Named Entity Recognition refers to the task for locating and classifying named entities in text documents.

This article assumes prior knowledge of the IOB (also known as BIO) format for entity tagging. IOB stands for inside, outside and beginning, where outside (O) indicates a token not belonging to an entity, beginning (B) indicates a token is the start of an entity and inside (I) indicates a token is inside an entity. You may refer to this article for more information on the IOB format.

NER can be thought of as a token level classification problem. Hence, NER models are often tuned using metrics such as F1, precision, recall at a token level during model training. When it comes to evaluating NER models on their downstream tasks' performance, it may be more beneficial to aggregate the individual tokens into full named-entities, and evaluate NER models at a full named-entity level against golden standard annotations. This also makes the evaluation more relatable for the client.

There are multiple evaluation schemes for evaluating NER models at a full named-entity level. This article will focus on the SemEval (International Workshop on Semantic Evaluation) scheme, which is one of the most popular schemes. The SemEval scheme is built off the Message Understanding Conferene (MUC) scheme.

Before we dive deeper into the SemEval scheme, ley us take a brief look into the MUC scheme. The MUC scheme introduces 5 categories to reflect the correctness of the full named-entities, and the classes are as follows. For a further read up on the MUC scheme, refer to this link

Evaluation Scheme	Explanation
Correct (COR)	The output of a system and the golden annotation are the same
Incorrect (INC)	The output of a system and the golden annotation don't match
Partial (PAR)	System and the golden annotation are somewhat "similar" but not the same
Missing (MIS)	A golden annotation is not captured by a system
Spurius (SPU)	System produces a response which doesn't exist in the golden annotation
Source	

In the SemEval scheme, there are 4 options of measuring the overall performance. They differ in terms of what is considered to be correct, incorrect, partial, missed and spurious predictions. Specifically, these 4 options indicate the correctness of the full named-entities with respect to their boundaries and entity classes. All 4 options can be applied to the computation of F1, precision and recall. A detailed explanation can be found in this blog.

Evaluation Scheme	Explanation
Strict	Exact boundary surface string and entity type
Exact	Exact boundary match over the surface string, regardless of the type
Partial	Partial boundary match over the surface string, regardless of the type
Type	Some overlap between the system tagged entity and the gold annotation is required
Source	

To illustrate the SemEval scheme with an example, consider the same example as before.

Token	Gold Labels
The	Other
news	Org
agency	Org
is	Other
here	Other

Scenario	Gold		Prediction		Evaluation Schema			
	Entity Type	Surface String	Entity Type	Surface String	Strict	Exact	Type	Partial
1	org	news agency	org	news agency	COR	COR	COR	COR
2			org	The	SPU	SPU	SPU	SPU
3	org	news agency			MIS	MIS	MIS	MIS
4	org	news agency	org	The news agency	INC	INC	COR	PAR
5	org	news agency	per	news agency	INC	COR	INC	COR
6	org	news agency	per	The news agency	INC	INC	INC	PAR

Next, we compute the total number of gold annotations with the formula:

$Possible(POS) = COR + INC + PAR + MIS = TP + FN$

The total annotations produced by the system is:

$ACTUAL(ACT) = COR + INC + PAR + SPU = TP + FP$

To compute the precision and recall for Strict and Exact:

$Precision = \frac{COR}{ACT} = \frac{TP}{TP+FP}$

$Recall = \frac{COR}{POS} = \frac{TP}{TP+FN}$

To compute the precision and recall for Partial and Type:

$Precision = \frac{COR+0.5\times PAR}{ACT} = \frac{TP}{TP+FP}$

$Recall = \frac{COR+0.5\times PAR}{POS} = \frac{COR}{ACT} = \frac{TP}{TP+FP}$

To compute the F1, precision and recall for the example above:

Measure	Partial	Type	Exact	Strict
Correct	2	2	2	1
Incorrect	0	2	2	3
Partial	2	0	0	0
Missed	1	1	1	1
Spurius	1	1	1	1
Precision	0.6 (a1)	0.4 (b1)	0.4 (c1)	0.2 (d1)
Recall	0.6 (2)	0.4 (b2)	0.4 (c2)	0.2 (d2)
F1	0.6	0.4	0.4	0.2

Reference for calculation:

a1) $\frac{2+0.5\times2}{2+0+2+1} = \frac{3}{5}$

a2) $\frac{2+0.5\times2}{2+0+2+1} = \frac{3}{5}$

b1) $\frac{2+0.5\times0}{2+2+0+1} = \frac{2}{5}$

b2) $\frac{2+0.5\times0}{2+2+0+1} = \frac{2}{5}$

c1) $\frac{2}{2+2+0+1} = \frac{2}{5}$

c2) $\frac{2}{2+2+0+1} = \frac{2}{5}$

d1) $\frac{1}{1+3+0+1} = \frac{1}{5}$

d2) $\frac{1}{1+3+0+1} = \frac{1}{5}$

In general, partial will be the most lenient evaluation, while strict will be the most stringent evaluation. The scores for type and exact will usually fall in between partial and strict.

If getting both the boundaries and the classes of the full named-entities are important, then strict will be the measure that you should prioritise. If it is only critical to get the boundaries correct, then exact will be the more appropriate measure to optimise for. Likewise, if it is only critical to get the classes correct and it is acceptable to get partial boundary overlaps, then type will be the more appropriate measure. Lastly, if only partial boundary overlaps are required and it is not important to get the classes correct, then partial can be the preferred measure. Generally, it is more common for type and exact to be preferred. You should check what the client's requirements are before making a decision as to which to prioritise.

6.7.2 References

- Named-Entity evaluation metrics based on entity-level
- Inside–outside–beginning (tagging)

6.8 How can we evaluate time-series classification models?

Contributor(s): Andy Ong, AI Engineer

This guide assumes that you have a fair understanding of typical classification evaluation metrics. This article provides a high-level discussion of the various evaluation metrics used in most projects.

6.9 Time-series classification

Time-series models can be evaluated using conventional regression and classification evaluation metrics. For regression problems, we can use Mean Squared Error (MSE) or Root Mean Square Error (RMSE). This article focuses on time-series classification problems, where the outcome to be predicted/forecasted is categorical. Recall, precision or F1 can be used to evaluate classification models, depending on business needs.

For a time-series classification project, a simple approach is to view every timepoints individually, and evaluate the model by looking at its F1 Score or precision and recall (similar to a non-time-series classification problem).

However, these evaluation metrics might not be adequate to fairly assess time-series models. Firstly, this approach is susceptible to noise. Moreover, it also ignores the autocorrelated nature of time-series data. This autocorrelatio tends to result in sequences of points falling into the same class, rather than one-off occurrences.

6.9.1 Time segments

Consider the following anomaly detection example:

Image credit

Here we have 3 ground truth anomalies, and 2 predicted anomalies.

In this approach, we will view timepoints as segments. In the graph below, there are 7 segments, which consist of 2 True Positives (TP), 1 False Positive (FP), 1 False Negatives (FN) and 3 True Negatives (TN). Note that each segment is equally treated as a single instance regardless of its length. This prevents excessive emphasis on long segments.

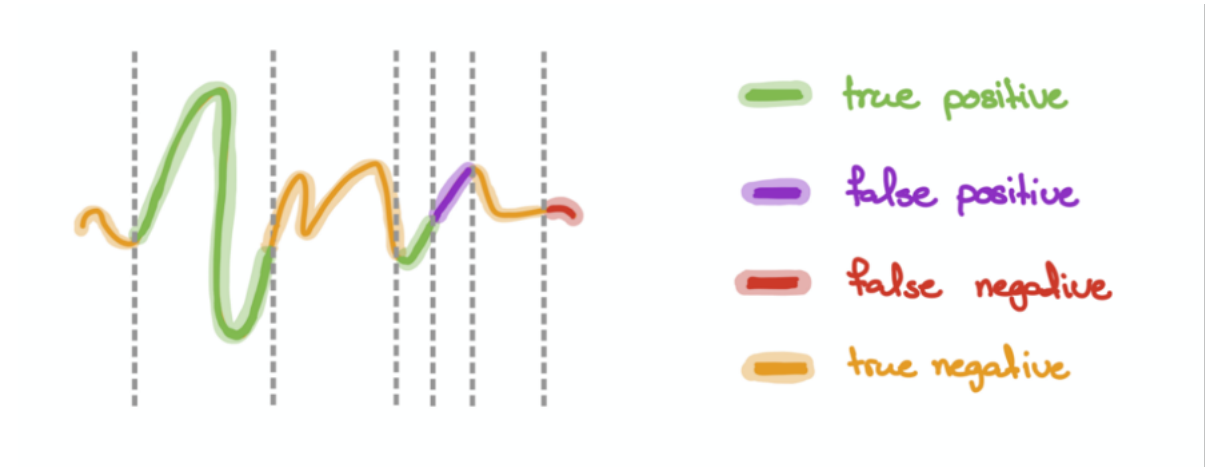


Image credit

6.9.2 Overlapping segment

Another way would be to view any "overlapping" predictions between the ground truth and predictions as 1 continuous True Positive.

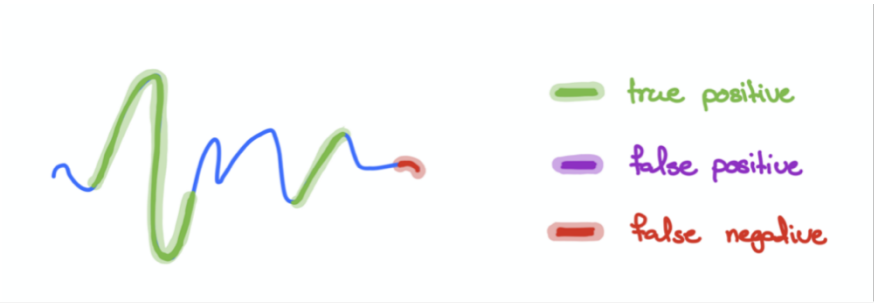


Image credit

Taking the above diagram as an example, we can see that there are 2 True Positives. Unlike the earlier approach, no False Positives are recorded here as the relevant ground truth and predicted segments are overlapping. This approach is more lenient, and it rewards the model for *any* successful detection of anomalies.

6.9.3 Case study

Consider the following case study to contextualise the two approaches described earlier

Company A is attempting to build an anomaly detection model to predict machinery failures. The ground truth and model prediction values for their model are as follows:

	T = 0	T = 1	T = 2	T = 3	T = 4	T = 5	T = 6	T = 7
Ground truth	0	0	1	1	0	0	0	1
Detected	0	0	0	1	0	1	0	1
1's denote anomalies								

The confusion matrices below are calculated using the time segment and overlapping segment approaches.

<table>
<tr><th>Approach</th><th>Confusion Matrix</th><th>Recall</th></tr>
<tr><td>Time Segment</td><td>
<table>
<tr><td></td><td colspan="3">Predicted</td></tr>
<tr><td rowspan="3">Actual</td><td>Total = 7</td><td>Anomaly</td><td>Normal</td></tr>
<tr><td>Anomaly</td><td>2</td><td>1</td></tr>
<tr><td>Normal</td><td>1</td><td>3</td></tr>
</table>
TP = 2, FP = 1, TN = 3, FN = 1
</td><td>Recall = 2 / 3 (0.66)</td></tr>
<tr><td>Overlapping Segment</td><td>
<table>
<tr><td></td><td colspan="3">Predicted</td></tr>
<tr><td rowspan="3">Actual</td><td>Total = 6</td><td>Anomaly</td><td>Normal</td></tr>
<tr><td>Anomaly</td><td>2</td><td>0</td></tr>
<tr><td>Normal</td><td>1</td><td>3</td></tr>
</table>
TP = 2, FP = 1, TN = 3, FN = 0
</td><td>Recall = 2 / 2 (1.00)</td></tr>
</table>

From the table above, we can see that the overlapping approach resulted in a recall of 1.00, which is more lenient as compared to the weighted segment (0.66). As Company A is planning to use the model to detect critical and costly machinery failures, it is not wise to use such a lenient approach like overlapping segment. Instead, the more stringent time segment approach is more appropriate.

This case study demonstrates how business needs must be taken into consideration when deciding the most suitable evaluation approach is suitable for your time series classification project.

6.10 References

Time series anomaly detection—-in the era of deep learning

6.11 How can we provide post-hoc explanations for black-box AI models?

Contributor: Joy Lin, Senior AI Technical Consultant

6.11.1 Introduction

Interpretability and explainability have been the recent buzzwords in explainable AI as an attempt to decipher how models derive predictions. As AI is gaining wider acceptance and adoption, stakeholders are increasingly requesting, sometimes under legal requirements, for an explanation on how the model reaches its decision, in order to generate trust in the predictions. Scenarios include: providing a valid reason for rejecting a bank loan, or how various features help in categorising a medical disease.

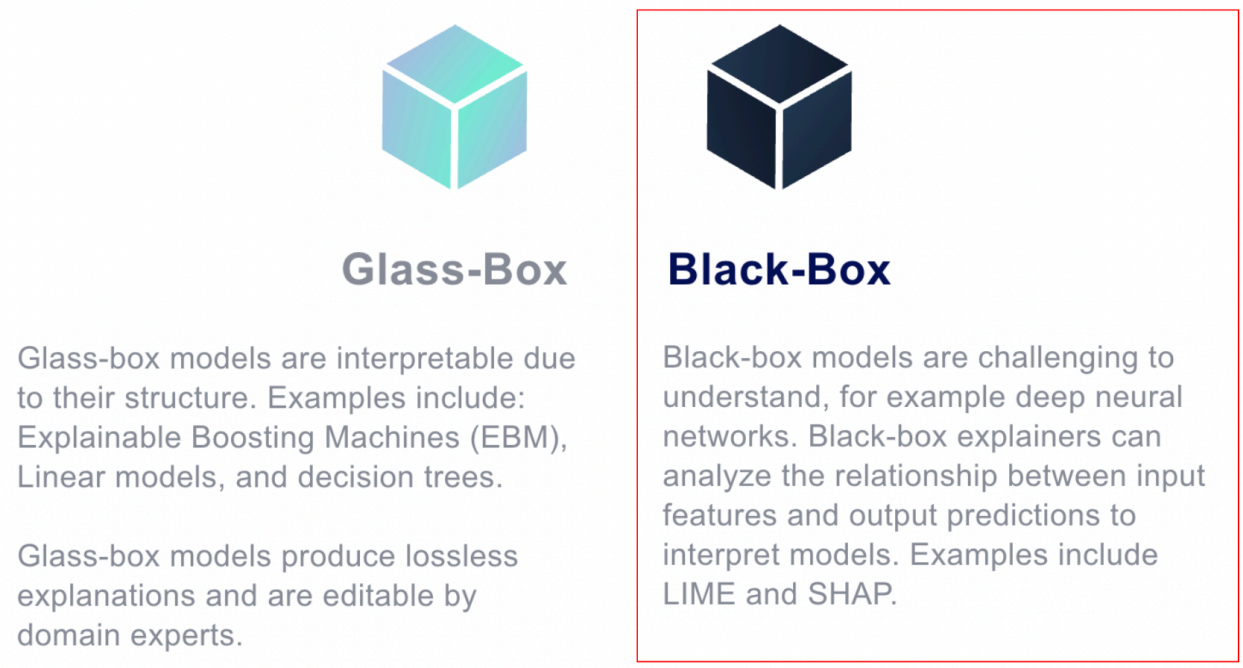
Explanations come in useful:

- during model development for error analysis and sense-checking
- after model deployment to increase stakeholder and end user trust, avoiding significant social or financial impact
- during model maintenance to detect drift, bias, or performance changes to guide re-training

6.11.2 Interpretable (Glass-box) VS Explainable (Black-box) Models

There are two broad approaches to deriving explanations, which is tied to the type of model used: interpretability (using glass-box models) or explainability (using black-box models). The table and diagram below shows the distinction between these approaches.

	Interpretability (Glass-box)	Explainability (Black-box)
Transparency	Highly translucent	Opaque
Model performance	Less powerful	More powerful
Relationship to model	Model-specific	Model-agnostic
Derivation of explanations	Via model's functional form	Via post-hoc analysis of model inputs and outputs
Outcome	Exact explanation	Approximate explanation
Examples	Linear regression model coefficients	Permutation feature importance ranking



[Source]

With glass-box models, explanations are inherent to the model, and exact.

With black-box models, explanations require additional post-hoc analysis of the trained model, and explanations are approximate only. Due to this, they need to be used with caution as explanations may vary between runs or when input data changes slightly.

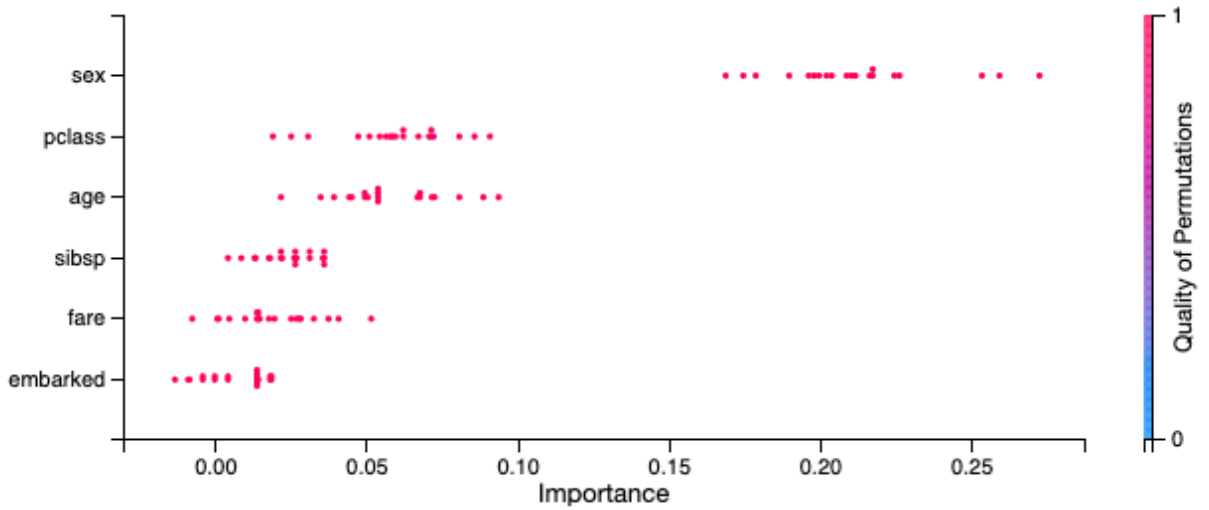
In this article, we focus only on explainability methods for black-box models.

A. Understanding feature importance and relationship between inputs and outputs of the model

Global model-agnostic methods describe how input features affect the output prediction *on average*. In particular, these methods tell us i) the importance of each feature, and ii) how it impacts the prediction (positive, neutral, or negative). They can be applied to black-box models like tree-based models and neural networks.

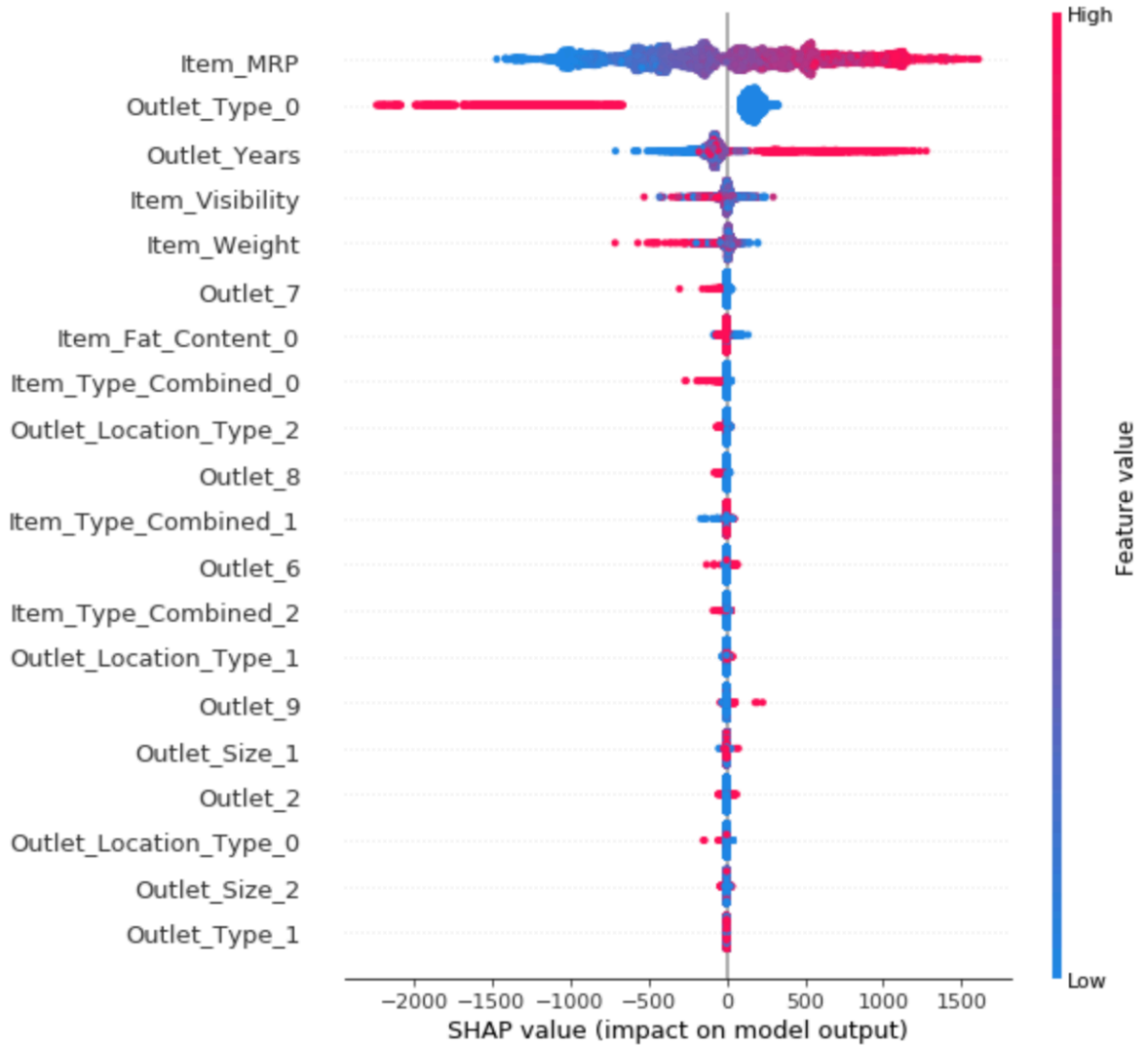
Examples:

1. Permutation feature importance: measures the importance of a feature as an increase in loss when the feature is permuted.



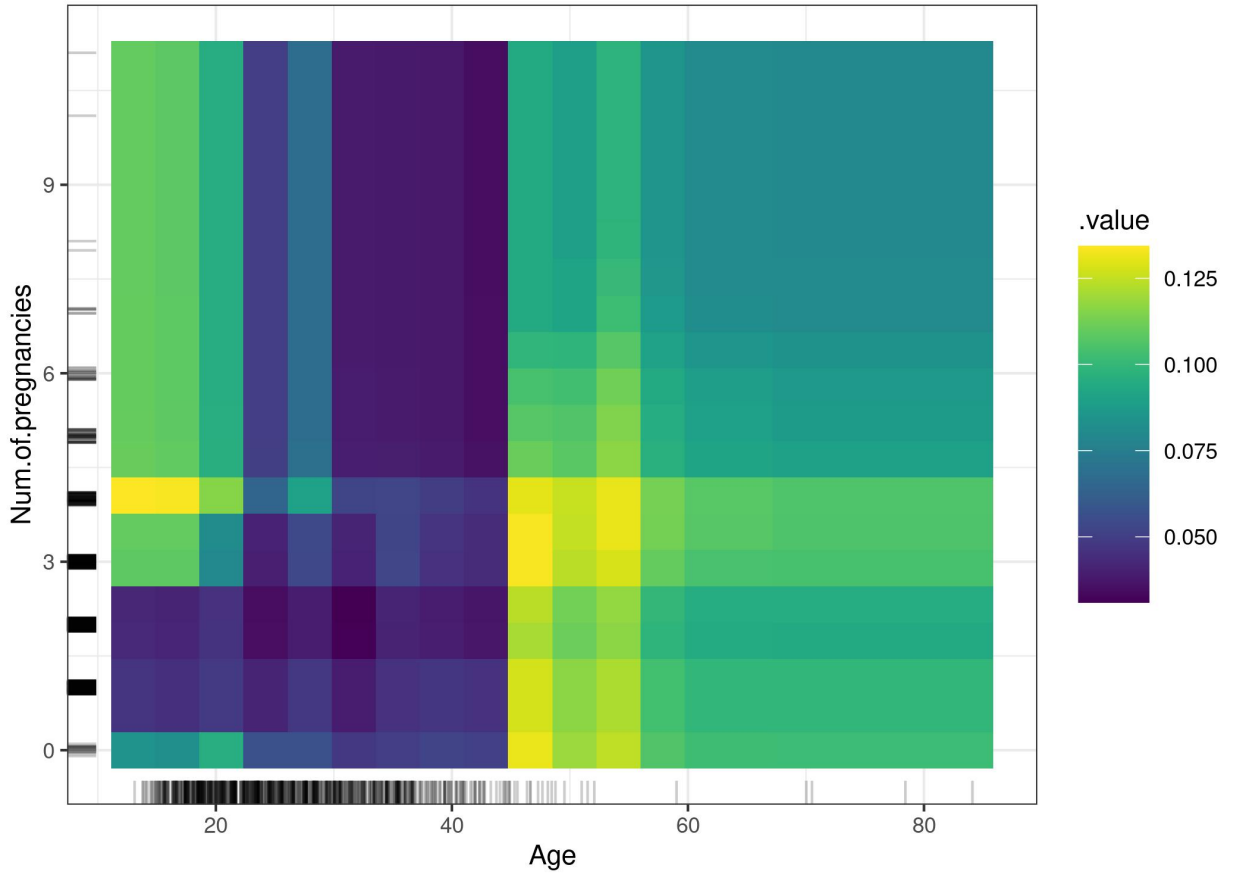
[Source] In the Titanic dataset, passenger gender is the most important feature to survival outcome, and about twice as important as passenger class and age.

1. SHAP (SHapley Additive exPlanations): globally estimates the contribution of every input feature with a combination of high/low and positive/negative Shapley value(s). Useful for sense-checking feature-target relationship.



[Source] Summary plot of feature importance and feature effects, where each dot represents an instance per feature.

1. Partial Dependence Plot (PDP): shows the marginal impact that one or two features have on the predictions. Useful to deep dive and understand the direction of the relationship between specific feature(s) and the target, although limited to easy visualisation of maximum two features.



[Source] PDP of cancer probability VS the interaction of age and number of pregnancies. The plot shows the increase in cancer probability at 45. For ages below 25, women who had 1 or 2 pregnancies have a lower predicted cancer risk, compared to women who had 0 or more than 2 pregnancies. Caution: These relationships should not be interpreted causally.

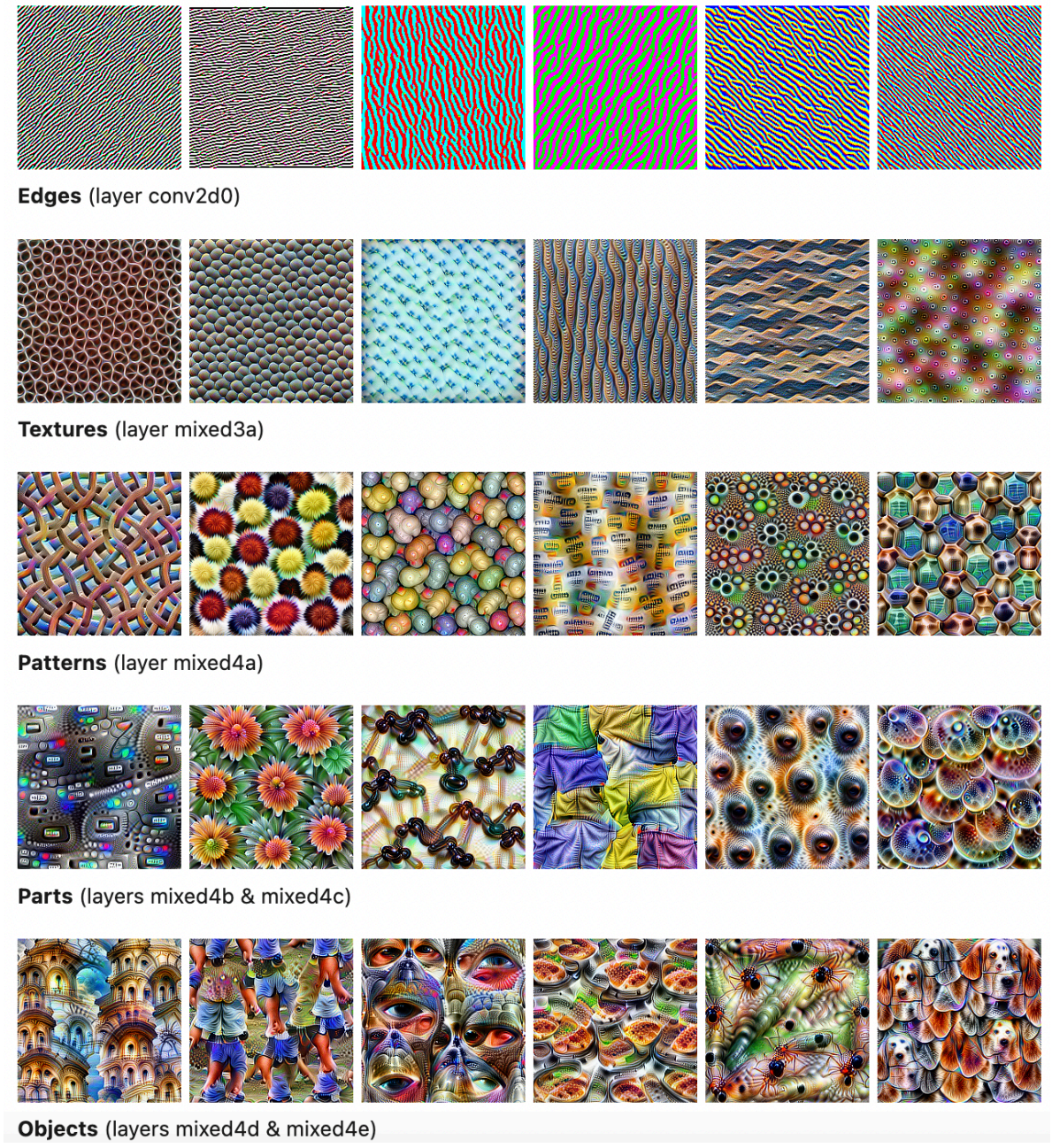
B. Explaining learned representations inside a neural network

A neural network can have multiple nodes and layers where input data undergo complex mathematical operations to output predictions. However increasing network density makes it nearly impossible to represent these complexities in a human-readable way. Instead, we use visual representations of data inside a neural network to understand the predictions.

Examples:

1. Learned features

   A model can learn - with increasing complexity across the network layers - the various edges, textures, patterns, parts, and eventually objects in images. Assessing these network layers helps you understand the features learned (or missed).

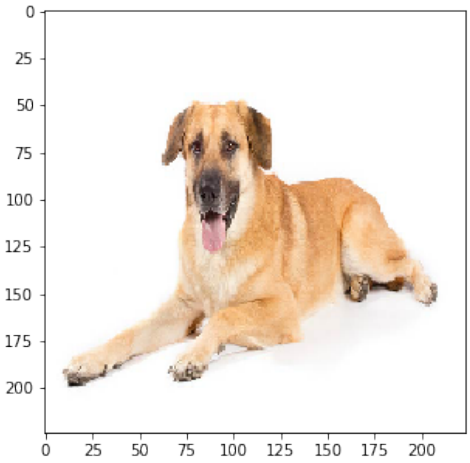


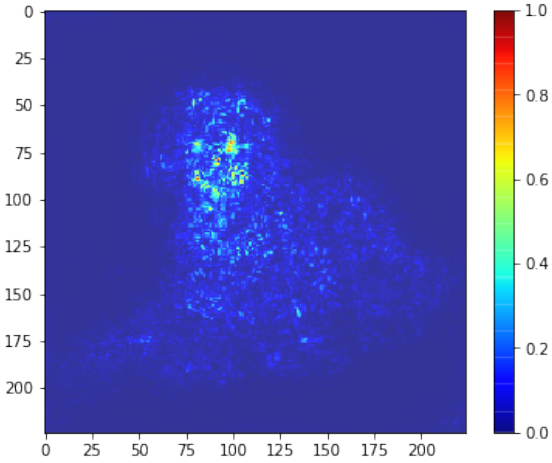
[Source]

Similarly, models can learn features from text or tabular data.

2. Pixel attribution using saliency maps

   Saliency maps provide another visualisation using ranked coloured pixels to indicate their contribution to the model





prediction.

[Source] Input image (left) and corresponding saliency map (right).

6.11.3 References

- Interpretable Machine Learning
- Glass-box models (model-specific)
- Black-box models (model-agnostic)

CHAPTER

SEVEN

7. SOLUTION DELIVERY

7.1 Overview

This chapter covers matters relating to the deployment of an end-to-end solution, including practices on incremental and continuous deployment.

7.2 How can I better understand the client's deployment requirements?

Contributor(s): Siti Nuruljannah Baharudin, AI Engineer

7.2.1 General

1. Where would the final solution be deployed?
    - Options may include a cloud service, an on-premise server, or an edge device. If the specifications are fixed, the client should provide them as soon as possible in order for the team to assess whether they are sufficient for the AI solution to operate on.
2. How should the application be packaged for deployment?
    - Although containerising the application is a common solution, some organisations may have restrictions on the containerisation technology allowed on their servers, or even disallow containerisation altogether. If so, it is recommended that you assess the impact on the deployment effort. For instance, the team may lack expertise in the containerisation technology used, or when containerisation is not used, more system integration testing may be required. These would result in additional effort to prepare the deployment package, which may impact the project schedule.
3. Are there any restrictions or preferences regarding the operating system on which the solution should be deployed?
4. Are there any restrictions on software licenses to be used?
    - Many open source libraries used in machine learning projects are licensed under Apache 2.0, MIT and BSD licenses. If your solution requires the use of libraries with less permissive licenses, it would be best to check with the client.
5. Would a staging environment or test device be provided for developers to test with prior to initial deployment?
    - If not provided, you would need to commission an internal environment that closely mirrors the specifications of the environment on which the solution would eventually be deployed.

7.2.2 Data & Model Outputs

1. Where would the data to be used for retraining and inference be located in the deployment environment?
2. Where would the model outputs be stored? Is there any storage size limit on this location?
3. If the data storage is located remotely, what is the authentication method to be used to access it?

7.2.3 Usage

1. What is the frequency at which the user intends to run the retraining and inference pipelines?
    - Generally, the requirements and considerations for inference are more critical compared to retraining, as the latter is usually performed less frequently and is less time-sensitive.
2. Is batch inference a requirement? If so, what is the expected size or volume of data in a single batch?
3. What are the requirements on inference speed and training time?
    - You would need to consider whether the limitations of the chosen deployment device would affect these requirements. If refinements can be made to the code to meet the requirements, the additional effort would also need to be considered.
4. What are the requirements on the size of the trained model?

7.2.4 Integration

1. Is the application required to interface with another system via an API?
2. If an API is to be supported, would the application function as the server or a client?
3. Which type of API should be supported?
    - If the client does not have specific requirements on the API, you may recommend them the most commonly used choice that can be easily supported by the tech stack you are using.
4. What is the required format of the API payload?
    - If the client has provided a specific format, you would need to consider how the size of the payload may affect the inference speed requirements. Performing some benchmarking tests using sample payload files would enable you to better determine this.
5. If no API is required, is there any preferred way for users to trigger the training or inference engine?

7.3 How can we build a minimum viable configuration for CI/CD automation?

Contributor(s): Syakyr Surani, Platforms Engineer

This guide assumes that you have some basic knowledge of CI/CD and an existing codebase. The codebase is in a working state and while currently you find it manageable, you anticipate that you need to move away from manual testing that can be unreliable even with a checklist to work with. You may have multiple projects to manage, or there are more contributors raising a pull/merge request than you can possibly handle just by manual testing. You have heard about CI/CD, but you do not know where to start.

This guide will not recommend specific CI/CD products or tools, but a general outline to adhere to for an effective minimum implementation, and where to go from there. The flowchart below gives the outline of the guide:

7.3.1 Identify Current Processes

Before anything can happen, it is best to plan your actions out. For that, you would need to list out the actions taken to run manual testing for your codebase. This is made easier if you already have a checklist of processes to test before pushing/merging to a branch. Here is a list to check against to see if you have missed anything:

- Linting
- Error checking (whether the code produces any unexpected errors)
- Packaging (Docker, PyPI, etc.)

Since we focus mainly on AI projects, there should also be a consideration to allow for fast prototyping of model training before packaging the project into a viable product to be interacted by the stakeholders.

7.3.2 Create Sample Data

Due to the nature of our organisation, we deal with data more often than other tech firms. As such, it is imperative that we create and build sample data to test out the functionality of our codebase before running it on the full dataset in order to reduce time taken bugfixing or adding new features.

7.3.3 Identify Stages of Deployment

This step is important in order to reduce time taken to test and have less resources used prior to the main events such as training and monitoring during deployment. Some stages of testing can be defined as follows:

- Tests made while committing to a bugfix/feature branch (linting, etc.)
- Tests made before doing a pull/merge request to dev (main) branch (error checking, etc.)
- Tests made from dev branch to release (production) (deployment builds, etc.)

7.3.4 Start Small

This final step is paramount as to not paralyse yourself in the vast overarching process of CI/CD automation. Start with a small subset of the codebase to familiarise yourself with the process before attempting to refactor an entire codebase, especially if you are working in a project with large teams or with critical infrastructure. Iterative development is key to managing a large undertaking, one step at a time.

If you need some guidance on where to start, you could refer to *this section of the handbook* under the simplified workflow for more information.

7.3.5 Final Thoughts

Every project is different, and therefore there is no one Minimum Viable Code/Configuration that can be proposed without first understanding the nature of such projects. Nevertheless, it is with hope that this guide can point out your first steps in building and integrating CI/CD automation into your codebase for a more robust and resilient one.

Below are some of the references you can look at to supplement your CI/CD journey.

7.3.6 References

- Gitlab - How to learn CI/CD fast
- Spiceworks - Top 10 CI/CD Best Practices for 2022

CHAPTER

EIGHT

DOCUMENTATION & HANDOVER

8.1 Overview

This chapter focuses on the process of knowledge transfer to technical or non-technical teams who are taking over an AI solution.

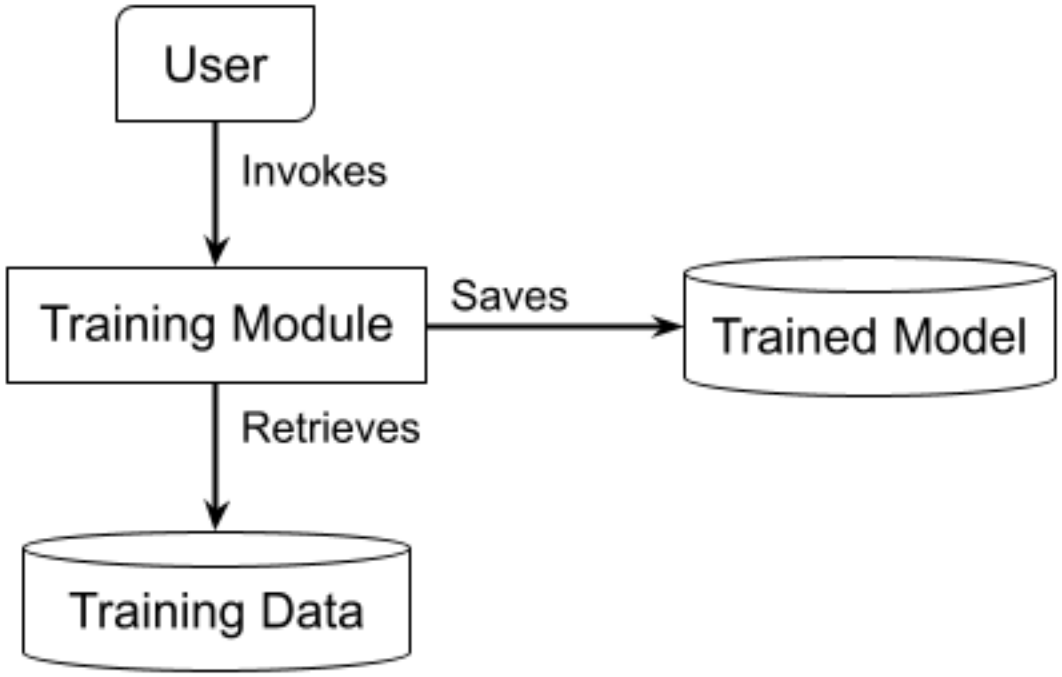
8.2 What are some good practices in documenting system architecture and processes?

Contributor(s): Siti Nuruljannah Baharudin, AI Engineer

In machine learning projects, documentation should be treated like a growing entity that is maintained the way we maintain code. At the same time, we do not want to spend excessive effort documenting details that may rapidly change over the course of the project. In the systems and software engineering fields, modelling languages such as UML are used to document system architecture and processes. Although they will not be discussed here, the concepts from these modelling languages can be adopted to suit the documentation requirements of your project.

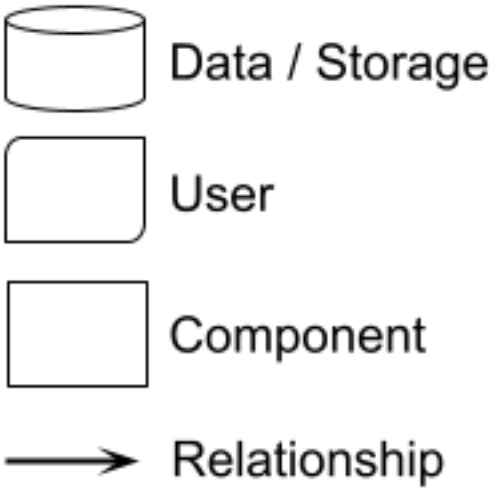
8.2.1 Use simple visuals

Box-and-arrow diagrams are a simple and easily maintainable visual method to document system architecture and processes. This example shows how a high-level process can be illustrated with a box-and-arrow diagram:



8.2.2 Consistency in notation

Choose a shape to denote each element and a line to denote each relationship or flow, and stick to these choices for all diagrams used in your documentation. This basic set of notations is a good place to start.



8.2.3 One diagram per process

Illustrate only one process per diagram. If there is a need to show lower-level detail of a process, create a separate diagram for that. I would recommend using the C4 model as a guide. It defines four levels of diagrams - Context, Container, Component and Code - based on a hierarchy of abstractions.

8.2.4 Consider your audience

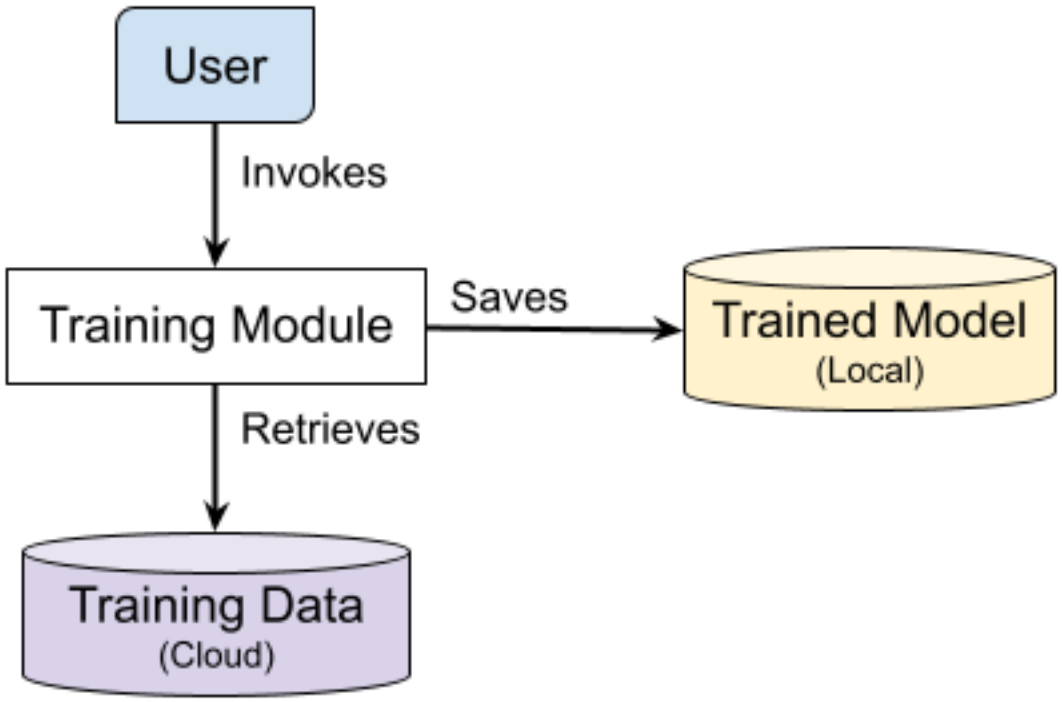
Within the development team, it may be worth the while to create up to level 3 (Component) diagrams. However, such diagrams may not be useful to non-technical people outside the development team. Furthermore, as they contain implementation details, these diagrams likely get outdated quickly.

In documentation delivered to end-users, Context diagrams would typically suffice for the general audience, to whom the solution is a black box. On the other hand, if the users are technical people and require understanding the inner workings of the system, including the lower-level diagrams would be a good idea. It is recommended that you automate the generation of Component and Code diagrams to minimise the effort spent on maintaining them.

8.2.5 Design considerations

Colour

Besides shapes, colours may also be used to visually differentiate between element types. For instance, in the diagram below, purple is used to denote a remote storage location while yellow is used to denote a local storage location.

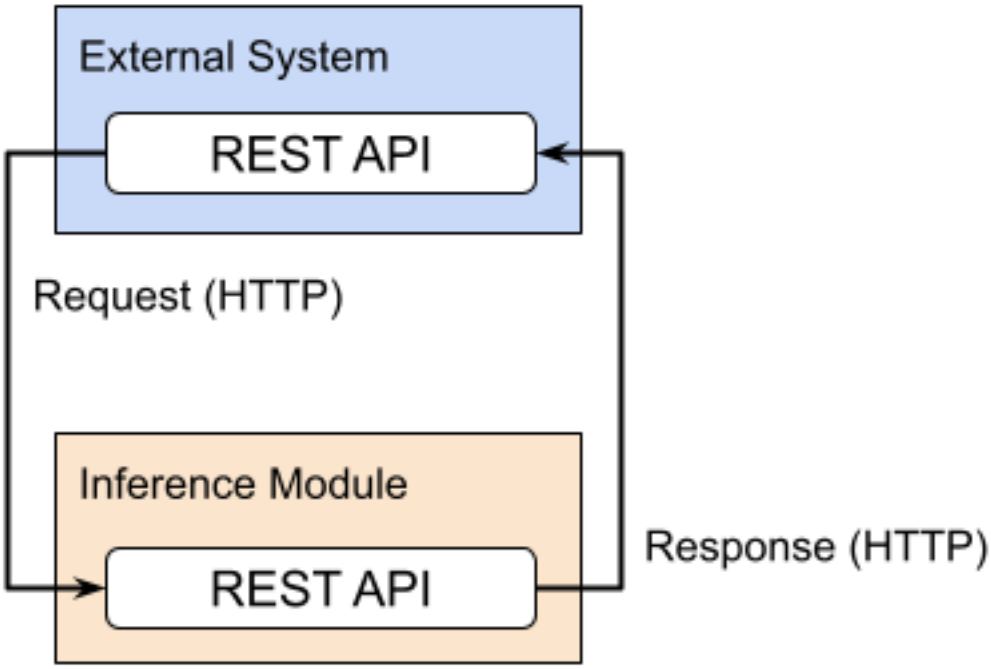


It is best that you refrain from relying only on colours to relay information in your diagram as there could be readers who have difficulty distinguishing colours. The use of colours should serve to improve understanding of the diagram and increase its visual appeal for the average reader.

Labels

As seen in the earlier examples, every element and relationship in your diagram should be labelled. A label should be specific yet concise, in a legible font.

In a system architecture diagram, the relationships between elements should have the protocol or technology explicitly labelled, as seen in the example below.



Reference(s):

- The C4 model for visualising software architecture
- Bhatti et al. (2021) Docs for Developers: An Engineer's Field Guide to Technical Writing.

CHAPTER

NINE

CITE AI SINGAPORE'S AI PRACTITIONER HANDBOOK

This handbook is developed by AI Singapore, with the main purpose of benefiting the in-house engineering staff. However, we hope that it would benefit the public as well and if you would like to cite this content, you may do so with the following snippet for BibTeX:

```
@book{aisg_aiprac_hbook,
    author       = {AI Singapore},
    title        = {AI Practitioner Handbook},
    howpublished = {\url{https://aisingapore.github.io/ai-practitioner-handbook/}},
    year         = {2023}
}
```