How to avoid machine learning pitfalls: a guide for academic researchers

Michael A. Lones[*]

Abstract

Mistakes in machine learning practice are commonplace, and can result in a loss of confidence in the findings and products of machine learning. This guide outlines common mistakes that occur when using machine learning, and what can be done to avoid them. Whilst it should be accessible to anyone with a basic understanding of machine learning techniques, it focuses on issues that are of particular concern within academic research, such as the need to do rigorous comparisons and reach valid conclusions. It covers five stages of the machine learning process: what to do before model building, how to reliably build models, how to robustly evaluate models, how to compare models fairly, and how to report results.

1 Introduction

It's easy to make mistakes when applying machine learning (ML), and these mistakes can result in ML models that fail to work as expected when applied to data not seen during training and testing [Liao et al., 2021]. This is a problem for practitioners, since it leads to the failure of ML projects. However, it is also a problem for society, since it erodes trust in the findings and products of ML [Gibney, 2022].

This guide aims to help newcomers avoid some of these mistakes. It's written by an academic, and focuses on lessons learnt whilst doing ML research in academia. Whilst primarily aimed at students and scientific researchers, it should be accessible to anyone getting started in ML, and only assumes a basic knowledge of ML techniques. However, unlike similar guides aimed at a more general audience, it includes topics that are of a particular concern to academia, such as the need to rigorously evaluate and compare models in order to get work published.

To make it more readable, the guidance is written informally, in a *Dos and Don'ts* style. It's not intended to be exhaustive, and references (with publicly-accessible URLs where available) are provided for further reading. Since it doesn't cover issues specific to particular academic subjects, it's recommended that readers also consult subject-specific guidance where available, e.g. in clinical medicine [Stevens et al., 2020], genomics [Whalen et al., 2022], environmental research [Zhu et al., 2023], materials science [Karande et al.,

[*]School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, Scotland, UK, Email: m.lones@hw.ac.uk, Web: www.macs.hw.ac.uk/∼ml355, Substack: Fetch Decode Execute.

2022], business and marketing [Van Giffen et al., 2022], computer security [Arp et al., 2022] and social science [Malik, 2020].

The review is divided into five sections. Before you start to build models covers issues that can occur early in the ML process, and focuses on the correct use of data and adequate consideration of the context in which ML is being applied. How to reliably build models then covers pitfalls that occur during the selection and training of models and their components. How to robustly evaluate models presents pitfalls that can lead to an incorrect understanding of model performance. How to compare models fairly then extends this to the situation where models are being compared, discussing how common pitfalls can lead to misleading findings. How to report your results focuses on reproducibility and factors that can lead to incomplete or deceptive reporting.

Changes

ML pitfalls are not static, and continue to evolve as ML develops. To address this, this guide has been updated annually since it was first released in 2021, and it will continue to be updated in the future. Feedback is welcome. If you cite it, please include the arXiv version number (currently v5[1]).

Changes from v4 Added Do use meaningful baselines, Do clean your data and Do consider model fairness. Extended Do look at your models, Do make sure you have enough data, Do think about how your model will be deployed, Don't allow test data to leak into the training process and Do evaluate a model multiple times.

Changes from v3 Added Do use a machine learning checklist and Do think about how and where you will use data. Rewrote Do evaluate a model multiple times. Revised Do keep up with progress in deep learning (and its pitfalls), Do be careful where and how you do feature selection, Do avoid sequential overfitting, Do choose metrics carefully and Do combine models (carefully). Extended Do use an appropriate test set.

Changes from v2 Added illustrations. Added Do avoid learning spurious correlations, Don't ignore temporal dependencies in time series data and Do keep up with progress in deep learning (and its pitfalls).

Changes from v1 Added Don't do data augmentation *before* splitting your data and Don't assume deep learning will be the best approach. Rewrote Don't use inappropriate models. Expanded Don't allow test data to leak into the training process, Do be careful when reporting statistical significance and Do be transparent.

[1]This version is published as "Avoiding machine learning pitfalls" in *Patterns* (Cell Press) [Lones, 2024]

Contents

1 Introduction **1**

2 Before you start to build models **4**
- 2.1 Do think about how and where you will use data . . . 4
- 2.2 Do take the time to understand your data . . . 4
- 2.3 Don't look at *all* your data . . . 5
- 2.4 Do clean your data . . . 5
- 2.5 Do make sure you have enough data . . . 6
- 2.6 Do talk to domain experts . . . 6
- 2.7 Do survey the literature . . . 7
- 2.8 Do think about how your model will be deployed . . . 7

3 How to reliably build models **7**
- 3.1 Don't allow test data to leak into the training process . . . 8
- 3.2 Do try out a range of different models . . . 9
- 3.3 Don't use inappropriate models . . . 9
- 3.4 Do keep up with progress in deep learning (and its pitfalls) . . . 9
- 3.5 Don't assume deep learning will be the best approach . . . 10
- 3.6 Do be careful where and how you do feature selection . . . 11
- 3.7 Do optimise your model's hyperparameters . . . 11
- 3.8 Do avoid learning spurious correlations . . . 13

4 How to robustly evaluate models **14**
- 4.1 Do use an appropriate test set . . . 14
- 4.2 Don't do data augmentation *before* splitting your data . . . 15
- 4.3 Do avoid sequential overfitting . . . 15
- 4.4 Do evaluate a model multiple times . . . 15
- 4.5 Do save some data to evaluate your final model instance . . . 17
- 4.6 Do choose metrics carefully . . . 17
- 4.7 Do consider model fairness . . . 18
- 4.8 Don't ignore temporal dependencies in time series data . . . 19

5 How to compare models fairly **19**
- 5.1 Don't assume a bigger number means a better model . . . 19
- 5.2 Do use meaningful baselines . . . 21
- 5.3 Do use statistical tests when comparing models . . . 21
- 5.4 Do correct for multiple comparisons . . . 22
- 5.5 Don't always believe results from community benchmarks . . . 22
- 5.6 Do combine models (carefully) . . . 23

6 How to report your results **23**
- 6.1 Do be transparent . . . 23
- 6.2 Do report performance in multiple ways . . . 24
- 6.3 Don't generalise beyond the data . . . 24
- 6.4 Do be careful when reporting statistical significance . . . 25
- 6.5 Do look at your models . . . 25
- 6.6 Do use a machine learning checklist . . . 26

7 Final thoughts **27**

2 Before you start to build models

It's normal to want to rush into training and evaluating models, but it's important to take the time to think about the goals of a project, to fully understand the data that will be used to support these goals, to consider any limitations of the data that need to be addressed, and to understand what's already been done in your field. If you don't do these things, then you may end up with results that are hard to publish, or models that are not appropriate for their intended purpose.

2.1 Do think about how and where you will use data

Data is central to most ML projects, but is often in short supply. Therefore it's important to think carefully about what data you need and how and where you will use it. Abstractly, you need data for two things, training models and testing models. However, for various reasons, this does not necessarily translate into using a single dataset divided into two parts. To begin with, model development often involves a period of experimentation: trying out different models with different hyperparameters, and preprocessing the data in different ways. To avoid overfitting (see Do avoid sequential overfitting), this process requires a separate validation set, i.e. an additional set of training data that's not used directly in training or testing models. If you have no prior idea of what modelling approach you're going to use, then this experimentation phase could potentially involve a lot of comparisons. Due to the multiplicity effect (see Do correct for multiple comparisons), the more comparisons you do, the more likely you are to overfit the validation data, and so the less useful the validation set will become in guiding your modelling decisions. So, in practice you might want to set aside multiple validation sets for this. Then there's the question of how you adequately test your selected model. Because it has the same biases as the training data, a test set taken from the same dataset as the training data may not be sufficient to measure the model's generality — see Do use an appropriate test set and Do report performance in multiple ways for more on this — meaning that, in practice, you may need more than one test dataset to robustly evaluate your model. Also be aware that you will often need additional test data when using cross-validation; see Do save some data to evaluate your final model instance.

2.2 Do take the time to understand your data

Eventually you will want to publish your work. This is a lot easier to do if your data is from a reliable source, has been collected using a reliable methodology, and is of good quality. For instance, if you are using data collected from an internet resource, make sure you know where it came from. Is it described in a paper? If so, take a look at the paper; make sure it was published somewhere reputable, and check whether the authors mention any limitations of the data. Do not assume that, because a data set has been used by a number of papers, it is of good quality — sometimes data is used just because it is easy to get hold of, and some widely used data sets are known to have significant limitations (see Paullada et al. [2021] for a discussion of this). If you train your model using bad data, then you will most likely generate a bad model: a process known as **garbage in garbage out**. One way to avoid bad data sets is to build a direct relationship with people who generate data, since this increases the likelihood of obtaining a good-quality dataset that meets your needs. It also avoids problems of overfitting community benchmarks; see Don't always believe results from community benchmarks. Yet regardless of where your data comes from, always begin by making sure that your data makes sense. Do some **exploratory data analysis** (see Cox [2017] for suggestions). Look for missing or inconsistent records. It is much easier to do this now, before you train a model, rather than later, when you're trying to explain to reviewers why you used bad data.

2.3 Don't look at all your data

As you look at data, it is quite likely that you will spot patterns and make insights that guide your modelling. This is another good reason to look at data. However, it is important that you do not make untestable assumptions that will later feed into your model. The "untestable" bit is important here; it's fine to make assumptions, but these should only feed into the training of the model, not the testing. So, to ensure this is the case, you should avoid looking closely at any test data in the initial exploratory analysis stage. Otherwise you might, consciously or unconsciously, make assumptions that limit the generality of your model in an untestable way. This is a theme I will return to several times, since the leakage of information from the test set into the training process is a common reason why ML models fail to generalise. See Don't allow test data to leak into the training process for more on this.

2.4 Do clean your data

Even good-quality datasets will have issues. Some of these come from unavoidable noise or omissions in the data collection process, others are due to human error during collection or collation. Whatever the cause, it's important to identify any issues, and do this before you start to build models. One common problem to look out for is data duplication, i.e. the unintentional inclusion of multiple copies of a data point. This can cause serious problems when a model is evaluated (see Don't do data augmentation *before* splitting your data for an example), so should be identified and removed early on. Another common problem is missing values. Some models can cope with these, but many can't, and so you'll have to replace missing values with something else before they can be trained. There are various forms of **imputation** that can be used to achieve this; see Emmanuel et al. [2021] for a review. If you do imputation, be careful to avoid data leaks during imputation — see Don't allow test data to leak into the training process. You should also check for outliers in your data, but only remove these if they are likely to be the result of noise or error rather than being natural extremes of the underlying data-generating process. For example, if a person's age is greater than 150, then it's probably an error; if it's 110, then it could be a natural outlier. A related issue is meaningless or inconsistent data, for instance a person with a negative age. Data cleaning can be a time-consuming process, and becomes more challenging as the complexity of data increases. For this reason, many people have explored automating data cleaning using ML approaches; see Côté et al. [2024] for a review.

2.5 Do make sure you have enough data

If you don't have enough data, then it may not be possible to train a model that generalises. Working out whether this is the case can be challenging, and may not be evident until you start building models: it all depends on the signal to noise ratio in the data set. If the signal is strong, then you can get away with less data; if it's weak, then you need more data. If you can't get more data — and this is a common issue in many research fields — then you can try using **data augmentation** techniques (see Wang et al. [2024], and for time series data, Iglesias et al. [2023]). These can be quite effective for boosting small data sets, though Don't do data augmentation *before* splitting your data. Data augmentation is also useful in situations where you have limited data in certain parts of your data set, e.g. in classification problems where you have less samples in some classes than others, a situation known as **class imbalance**. See Haixiang et al. [2017] for a review of methods for dealing with this; also see Do choose metrics carefully. Another option for dealing with small data sets is to use transfer learning — see Do keep up with progress in deep learning (and its pitfalls). A danger when using small datasets is that different data partitions may be biased, for instance in terms of the quality or difficulty of data they contain. For this reason, it is advisable to consider frequent repartitioning. Cross-validation (see Do evaluate a model multiple times) is an efficient way of achieving this in small data sets. If you have limited data, then it's also likely that you will have to limit the complexity of the ML models you use, since models with many parameters, like deep neural networks, can easily overfit small data sets (see Don't assume deep learning will be the best approach). Regardless of how you handle the problem of limited data, it's important to identify this issue early on, and come up with a suitable strategy to mitigate against it.

2.6 Do talk to domain experts

Domain experts can be very valuable. They can help you to understand which problems are useful to solve, they can help you choose the most appropriate feature set and ML model to use, and they can help you publish to the most appropriate audience. Failing to consider the opinion of domain experts can lead to projects which don't solve useful problems, or which solve useful problems in inappropriate ways. An example of the latter is using an opaque ML model to solve a problem where there is a strong need to understand how the model reaches an outcome, e.g. in making medical or financial decisions (see Rudin [2019]). At the beginning of a project, domain experts can help you to understand the data, and point you towards features that are likely to be predictive. At the end of a project, they can help you to publish in domain-specific journals, and hence reach an audience that is most likely to benefit from your research.

2.7 Do survey the literature

You're probably not the first person to throw ML at a particular problem domain, so it's important to understand what has and hasn't been done previously. Other people having worked on the same problem isn't a bad thing; academic progress is typically an iterative process, with each study providing information that can guide the next. It may be discouraging to find that someone has already explored your great idea, but they most likely left plenty of avenues of investigation still open, and their previous work can be used as justification for your work. To ignore previous studies is to potentially miss out on valuable information. For example, someone may have tried your proposed approach before and found fundamental reasons why it won't work (and therefore saved you a few years of frustration), or they may have partially solved the problem in a way that you can build on. So, it's important to do a literature review before you start work; leaving it too late may mean that you are left scrambling to explain why you are covering the same ground or not building on existing knowledge when you come to write a paper.

2.8 Do think about how your model will be deployed

Why do you want to build an ML model? This is an important question, and the answer should influence the process you use to develop your model. Many academic studies are just that — studies — and not really intended to produce models that will be used in the real world. This is fair enough, since the process of building and analysing models can itself give very useful insights into a problem. However, for many academic studies, the eventual goal is to produce an ML model that can be deployed in a real world situation. If this is the case, then it's worth thinking early on about how it is going to be deployed. For instance, if it's going to be deployed in a resource-limited environment, such as a sensor or a robot, this may place limitations on the complexity of the model. If there are time constraints, e.g. a classification of a signal is required within milliseconds, then this also needs to be taken into account when selecting a model. If using deep learning, then energy costs and carbon footprint may be a consideration, and if using LLMs, there may be further operational costs for hosting or accessing foundation models. Another consideration is how the model is going to be tied into the broader software system within which it is deployed; this procedure is often far from simple (see Sculley et al. [2015]). However, emerging approaches such as **ML Ops** aim to address some of the difficulties; see Kreuzberger et al. [2023] for a review, and Shankar et al. [2022] for a discussion of common challenges when operationalising ML models.

3 How to reliably build models

Building models is one of the more enjoyable parts of ML. With modern ML frameworks, it's easy to throw all manner of approaches at your data and see what sticks. However, this can lead to a disorganised mess of experiments that's hard to justify and hard to write up. So, it's important to approach model building in an organised manner, making sure you use data correctly, and putting adequate consideration into the choice of models.

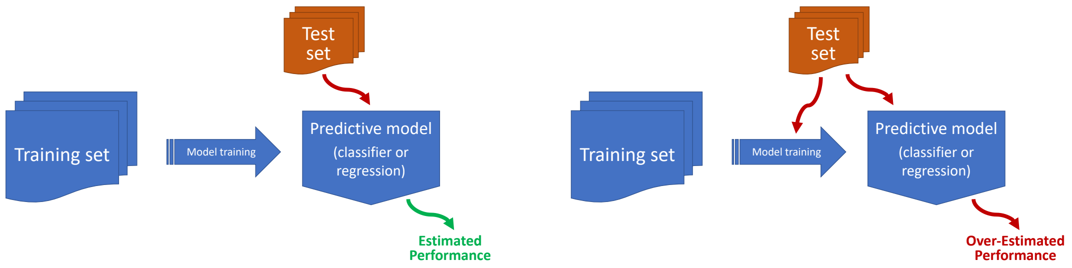

Test set
Training set
Model training
Predictive model (classifier or regression)
Estimated Performance
Test set
Training set
Model training
Predictive model (classifier or regression)
Over-Estimated Performance

Figure 1: See Don't allow test data to leak into the training process. [left] How things should be, with the training set used to train the model, and the test set used to measure its generality. [right] When there's a data leak, the test set can implicitly become part of the training process, meaning that it no longer provides a reliable measure of generality.

3.1 Don't allow test data to leak into the training process

It's essential to have data that you can use to measure how well your model generalises. A common problem is allowing information about this data to leak into the configuration, training or selection of models (see Figure 1). When this happens, the data no longer provides a reliable measure of generality, and this is a common reason why published ML models often fail to generalise to real world data. There are a number of ways that information can leak from a test set. Some of these seem quite innocuous. For instance, during data preparation, using information about the means and ranges of variables within the whole data set to carry out variable scaling or imputation — in order to prevent information leakage, these statistics should be calculated using only the training data. Other common examples of information leakage are carrying out feature selection before partitioning the data (see Do be careful where and how you do feature selection), using the same test data to evaluate the generality of multiple models (see Do avoid sequential overfitting and Don't always believe results from community benchmarks), and applying data augmentation before splitting off the test data (see Don't do data augmentation *before* splitting your data). The best thing you can do to prevent these issues is to partition off a subset of your data right at the start of your project, and only use this independent test set once to measure the generality of a single model at the end of the project (see Do save some data to evaluate your final model instance). There are also forms of data leakage which are specific to certain types of data. Time series data is particularly problematic, since the order of samples is significant, and random splits can easily cause leakage and overfitting — see Don't ignore temporal dependencies in time series data for more on this. Even for non-time series data, the experimental conditions used to generate data sets may lead to temporal dependencies, or other problematic conditions such as duplicated or similar samples — see Do use an appropriate test set for an example. In order to prevent leakage, these kinds of issues need to be identified and taken into account when splitting data. For a broader discussion of data leakage, see Kapoor and Narayanan [2023].

3.2 Do try out a range of different models

Generally speaking, there's no such thing as a single best ML model. In fact, there's a proof of this, in the form of the No Free Lunch theorem, which shows that no ML approach is any better than any other when considered over every possible problem [Wolpert, 2002]. So, your job is to find the ML model that works well for your particular problem. There is some guidance on this. For example, you can consider the **inductive biases** of ML models; that is, the kind of relationships they are capable of modelling. For instance, linear models, such as linear regression and logistic regression, are a good choice if you know there are no important non-linear relationships between the features in your data, but a bad choice otherwise. Good quality research on closely related problems may also be able to point you towards models that work particularly well. However, a lot of the time you're still left with quite a few choices, and the only way to work out which model is best is to try them all. Fortunately, modern ML libraries, such as scikit-learn [Varoquaux et al., 2015] in Python, tidymodels [Kuhn and Wickham, 2020] in R, and MLJ [Blaom et al., 2020] in Julia, allow you to try out multiple models with only small changes to your code, so there's no reason not to try them all out and find out for yourself which one works best. However, Don't use inappropriate models, and use a validation set, rather than the test set, to evaluate them (see Do avoid sequential overfitting). When comparing models, Do optimise your model's hyperparameters and Do evaluate a model multiple times to make sure you're giving them all a fair chance, and Do correct for multiple comparisons when you publish your results.

3.3 Don't use inappropriate models

By lowering the barrier to implementation, modern ML libraries also make it easy to apply inappropriate models to your data. This, in turn, could look bad when you try to publish your results. A simple example of this is applying models that expect categorical features to a dataset containing numerical features, or vice versa. Some ML libraries allow you to do this, but it may result in a poor model due to loss of information. If you really want to use such a model, then you should transform the features first; there are various ways of doing this, ranging from simple one-hot encodings to complex learned embeddings. Other examples of inappropriate model choice include using a classification model where a regression model would make more sense (or vice versa), attempting to apply a model that assumes no dependencies between variables to time series data, or using a model that is unnecessarily complex (see Don't assume deep learning will be the best approach). Also, if you're planning to use your model in practice, Do think about how your model will be deployed, and don't use models that aren't appropriate for your use case.

3.4 Do keep up with progress in deep learning (and its pitfalls)

Whilst deep learning may not always be the best solution (see Don't assume deep learning will be the best approach), if you are going to use deep learning, then it's advisable to try and keep up with recent developments in this fast-moving field. Figure 2 summarises

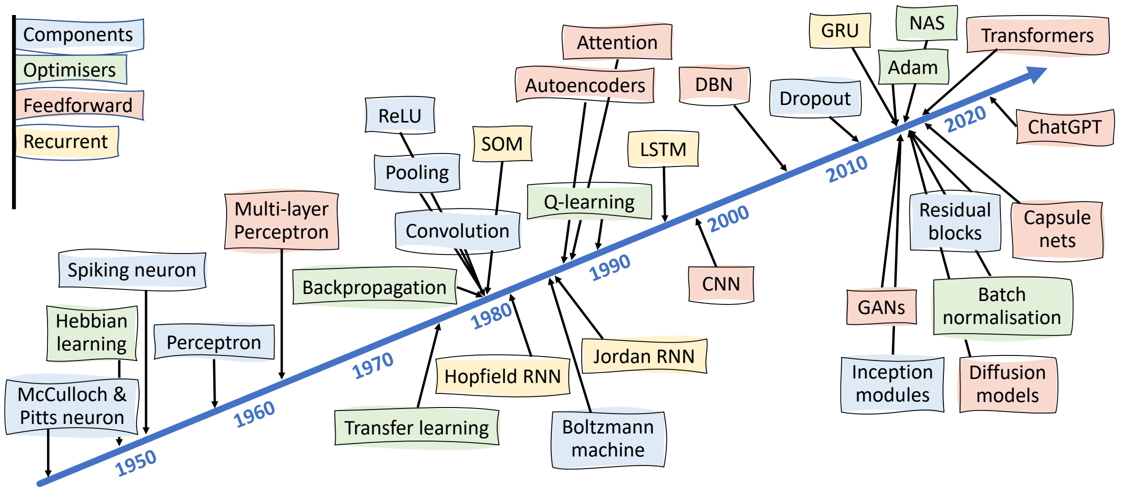


Figure 2: See Do keep up with progress in deep learning (and its pitfalls). A rough history of neural networks and deep learning, showing what I consider to be the milestones in their development. For a far more thorough account of the field's historical development, take a look at Schmidhuber [2015].

some of the important developments over time. Multilayer perceptrons (MLP) and recurrent neural networks (particularly LSTM) have been around for some time, but have largely been subsumed by newer models such as convolutional neural networks (CNN) [Li et al., 2021] and **transformers** [Lin et al., 2022]. For example, transformers have become the go-to model for processing sequential data (e.g. natural language), and are increasingly being applied to other data types too, such as images [Khan et al., 2022]. A prominent downside of both transformers and deep CNNs is that they have many parameters and therefore require a lot of data to train them. However, an option for small data sets is to use **transfer learning**, where a model is pre-trained on a large generic data set and then fine-tuned on the data set of interest [Han et al., 2021]. Larger pre-trained models, many of which are freely shared on websites such as Hugging Face, are known as **foundation models**; see Zhou et al. [2023] for a survey. Whilst powerful, these come with their own set of pitfalls. For example, their ability to fully memorise input data is the cause of data security and privacy concerns [Li et al., 2023]. The use of opaque, often poorly documented, training datasets also leads to pitfalls when fitting them into broader ML pipelines (see Do combine models (carefully) for more info) and comparing them fairly with other ML models (see Don't assume a bigger number means a better model and Don't always believe results from community benchmarks). For an extensive, yet accessible, guide to deep learning, see Zhang et al. [2023].

3.5 Don't assume deep learning will be the best approach

A common pitfall is to assume that deep neural networks will provide the best solution to any problem, and consequently fail to try out other, possibly more appropriate, models.

Whilst deep learning is great for certain tasks, it is not good at everything; there are plenty of examples of it being out-performed by "old fashioned" machine learning models such as random forests and SVMs. See, for instance, Grinsztajn et al. [2022], who show that tree-based models often outperform deep learners on tabular data. Certain kinds of deep neural network architecture may also be ill-suited to certain kinds of data: see, for example, Zeng et al. [2023], who argue that transformers are not well-suited to time series forecasting. There are also theoretical reasons why any one kind of model won't always be the best choice (see Do try out a range of different models). In particular, a deep neural network is unlikely to be a good choice if you have limited data, if domain knowledge suggests that the underlying pattern is quite simple, or if the model needs to be interpretable. This last point is particularly worth considering: a deep neural network is essentially a very complex piece of decision making that emerges from interactions between a large number of non-linear functions. Non-linear functions are hard to follow at the best of times, but when you start joining them together, their behaviour gets very complicated very fast. Whilst explainable AI methods (see Do look at your models) can shine some light on the workings of deep neural networks, they can also mislead you by ironing out the true complexities of the decision space (see Molnar et al. [2020]). For this reason, you should take care when using either deep learning or explainable AI for models that are going to make high stakes or safety critical decisions; see Rudin [2019] for more on this.

3.6 Do be careful where and how you do feature selection

A common stage of training a model is to carry out **feature selection** (surveyed by Cai et al. [2018]). When doing this, it is important to treat it as part of model training, and not something more general that you do before model training. A particularly common error is to do feature selection on the whole data set before splitting off the test set, something that will result in information leaking from the test set into the training process (see Don't allow test data to leak into the training process). Instead, you should only use the training set to select the features which are used in both the training set and the test set (see Figure 3). The same is true when doing **dimensionality reduction**. For example, if you're using principal component analysis (PCA), the component weightings should be determined by looking only at the training data; the same weightings should then be applied to the test set. Special care should be taken when using autoencoders for dimensionality reduction — see Do combine models (carefully). If you're doing cross-validation (see Do evaluate a model multiple times) then it's important to carry out feature selection or dimensionality reduction independently within each iteration, each time using just the training folds (see Figure 3, bottom).

3.7 Do optimise your model's hyperparameters

Many models have **hyperparameters** — that is, numbers or settings that affect the configuration of the model. Examples include the kernel function used in an SVM, the number of trees in a random forest, and the architecture of a neural network. Many

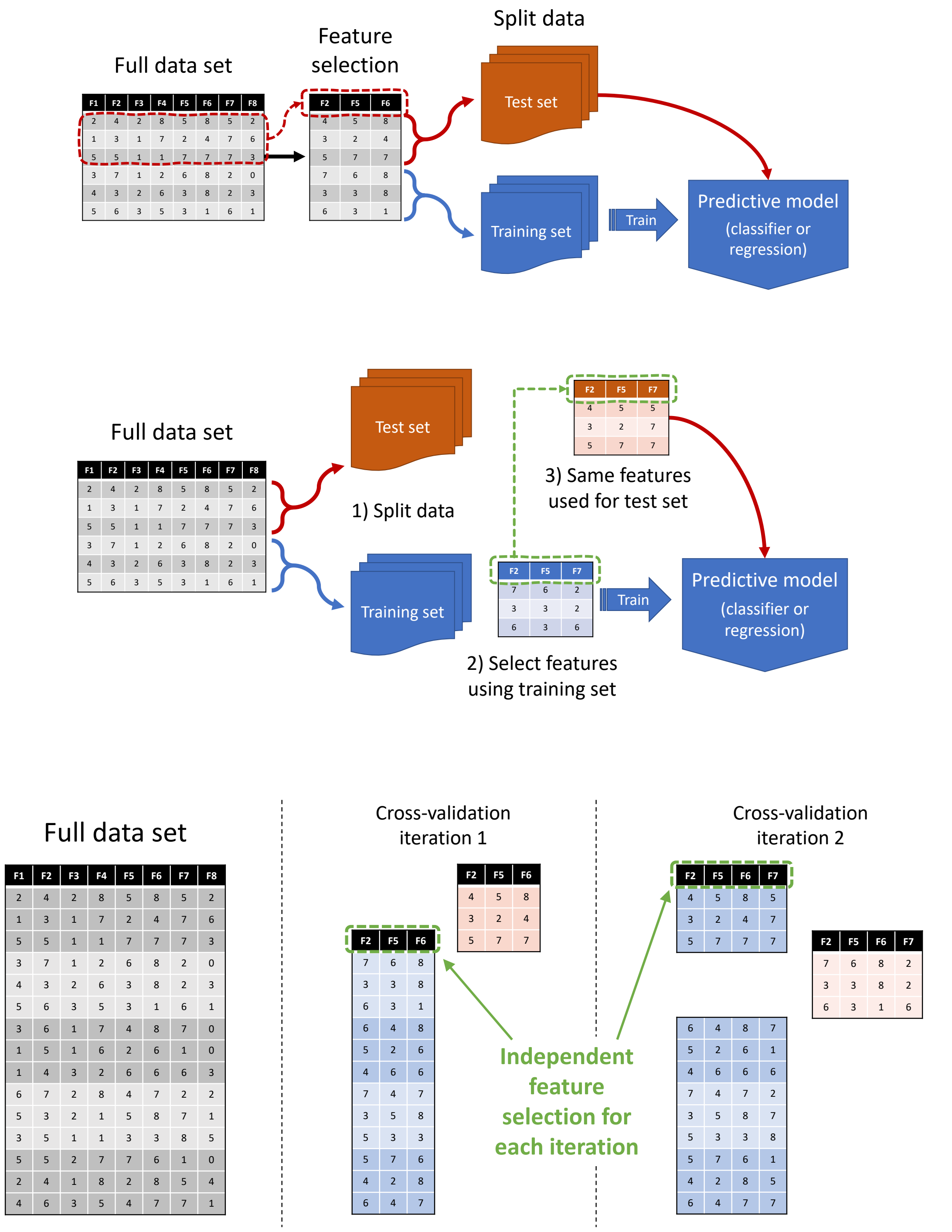

Full data set
Feature selection
Split data
Test set
Training set
Train
Predictive model
(classifier or regression)
Full data set
Test set
1) Split data
3) Same features used for test set
Training set
Train
Predictive model
(classifier or regression)
2) Select features using training set
Full data set
Cross-validation iteration 1
Cross-validation iteration 2
Independent feature selection for each iteration

Figure 3: See Do be careful where and how you do feature selection. [top] Data leakage due to carrying out feature selection before splitting off the test data (outlined in red), causing the test set to become an implicit part of model training. [centre] How it should be done. [bottom] When using cross-validation, it's important to carry out feature selection independently for each iteration, based only on the subset of data (shown in blue) used for training during that iteration.

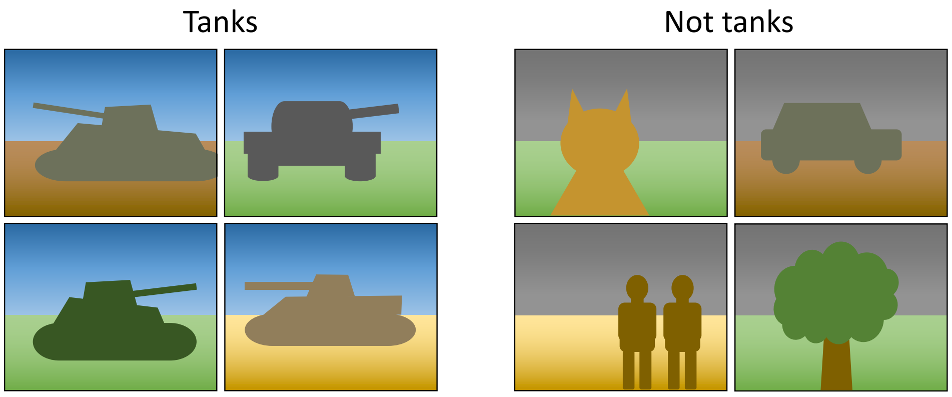


Figure 4: See **Do avoid learning spurious correlations**. The problem of spurious correlations in images, as illustrated by the tank problem. The images on the left are tanks, and those on the right are not tanks. However, the consistent background (blue for tanks, grey for others) means that these images can be classified by merely looking at the colours of pixels towards the top of the images, rather than having to recognise the objects in the images, resulting in a poor model.

of these hyperparameters significantly effect the performance of the model, and there is generally no one-size-fits-all. That is, they need to be fitted to your particular data set in order to get the most out of the model. Whilst it may be tempting to fiddle around with hyperparameters until you find something that works, this is not likely to be an optimal approach. It's much better to use some kind of **hyperparameter optimisation** strategy, and this is much easier to justify when you write it up. Basic strategies include random search and grid search, but these don't scale well to large numbers of hyperparameters or to models that are expensive to train, so it's worth using tools that search for optimal configurations in a more intelligent manner. See Bischl et al. [2023] for further guidance. It is also possible to use **AutoML** techniques to optimise both the choice of model and its hyperparameters, in addition to other parts of the machine learning pipeline — see Barbudo et al. [2023] for a review.

3.8 Do avoid learning spurious correlations

Spurious correlations are features within data which are correlated with the target variable, but which have no semantic meaning. They are basically red herrings, and it's not uncommon for ML models to pick up on them in training, and consequently fail to generalise well. A classic example is the tank problem. Legend[2] has it that the US military were looking to train an ML model that could recognise tanks. However, because the tank pictures used in training were taken during different weather conditions to the non-tank pictures, the model ended up discriminating based on features such as the number of blue pixels in the sky, rather than the presence of a tank (see Figure 4 for

[2]There is some debate about whether this actually happened: see `https://www.gwern.net/Tanks`.

an illustration). An ML model that uses such spurious correlations to perform classification would appear to be very good, in terms of its metric scores, but would not work in practice. More complex data tends to contain more of these spurious correlations, and more complex models have more capacity to overfit spurious correlations. This means that spurious correlations are a particular issue for deep learning, where approaches such as regularisation (see Do keep up with progress in deep learning (and its pitfalls)) and data augmentation (see Do make sure you have enough data) can help mitigate against this. However, spurious correlations can occur in all data sets and models, so it is always worth looking at your trained model to see whether it's responding to appropriate features within your data — see Do look at your models.

4 How to robustly evaluate models

In order to contribute to progress in your field, you need to have valid results that you can draw reliable conclusions from. Unfortunately it's really easy to evaluate ML models unfairly, and, by doing so, muddy the waters of academic progress. So, think carefully about how you are going to use data in your experiments, how you are going to measure the true performance of your models, and how you are going to report this performance in a meaningful and informative way.

4.1 Do use an appropriate test set

First of all, always use a test set to measure the generality of an ML model. How well a model performs on the training set is almost meaningless, and a sufficiently complex model can entirely learn a training set yet capture no generalisable knowledge. It's also important to make sure the data in the test set is appropriate. That is, it should not overlap with the training set and it should be representative of the wider population. For example, consider a photographic data set of objects where the images in the training and test set were collected outdoors on a sunny day. The presence of the same weather conditions means that the test set will not be independent, and by not capturing a broader variety of weather conditions, it will also not be representative. Similar situations can occur when a single piece of equipment is used to collect both the training and test data; if the model overlearns characteristics of the equipment, it will likely not generalise to other pieces of equipment, and this will not be detectable by evaluating it on the test set. If using public datasets to test a model, be wary of **Frankenstein datasets**, which are assembled from other public datasets and risk overlap with training data. Also be careful when handling datasets that contain multiple data points for each subject; if using these, it's important to make sure that each subject's data points are kept together when splitting off the test set or when doing cross-validation. See Roberts et al. [2021] for a revealing account of how a number of these pitfalls led to the failure of the vast majority of Covid-19 detection models to generalise beyond their test sets.

4.2 Don't do data augmentation before splitting your data

Data augmentation (see Do make sure you have enough data) can be a useful technique for balancing datasets and boosting the generality and robustness of ML models. However, it's important to do data augmentation only on the training set, and not on data that's going to be used for testing. Including augmented data in the test set can lead to a number of problems. One problem is that the model may overfit the characteristics of the augmented data, rather than the original samples, and you won't be able to detect this if your test set also contains augmented data. A more critical problem occurs when data augmentation is applied to the entire data set before it is split into training and test sets. In this scenario, augmented versions of training samples may end up in the test set, which in the worst case can lead to a particularly nefarious form of data leakage in which the test samples are mostly variants of the training samples. For an interesting study of how this problem affected an entire field of research, see Vandewiele et al. [2021].

4.3 Do avoid sequential overfitting

Oddly, one of the most pernicious forms of data leakage doesn't have a commonly-agreed name[3], so I'm going to refer to it as sequential overfitting. This occurs when you train multiple models in succession, using knowledge gained about each model's performance to guide the configuration of the next one, and you use the same test set to evaluate each model. Often this is done as an informal process, trying out different models and different hyperparameters until you get good performance on the test set. As such, it is rarely documented, which is one reason why it's so pernicious. Specifically, the problem lies in using the test set throughout this process, since using the test set to choose between models means that information about the test set implicitly leaks into the training process. See Figure 5 for an illustration of this idea. The consequence is that models gradually overfit the test set; the more times you use the test set, the more the overfitting that occurs. The solution is to either use a **validation set** (i.e. a separate set of samples that are not directly used in training, but which are used to guide training) or use a holdout data set to test the final model. See Cawley and Talbot [2010] and Hosseini et al. [2020] for more on this.

4.4 Do evaluate a model multiple times

Many ML models are stochastic or unstable. That is, if you train them multiple times, or if you make small changes to the training data, then their performance varies significantly. The same is true of using LLMs at inference time. This means that a single evaluation of a model can be unreliable, and may either underestimate or overestimate the model's true potential. For this reason, it is common to carry out multiple evaluations. At training time, there are numerous ways of doing this. For stochastic models, the simplest is to train the same model multiple times using different random seeds and then look at the average performance. A more robust approach is to also vary the

[3]Though Hosseini et al. [2020] suggested "over-hyping", from *over*fitting of *hyp*er-parameters.

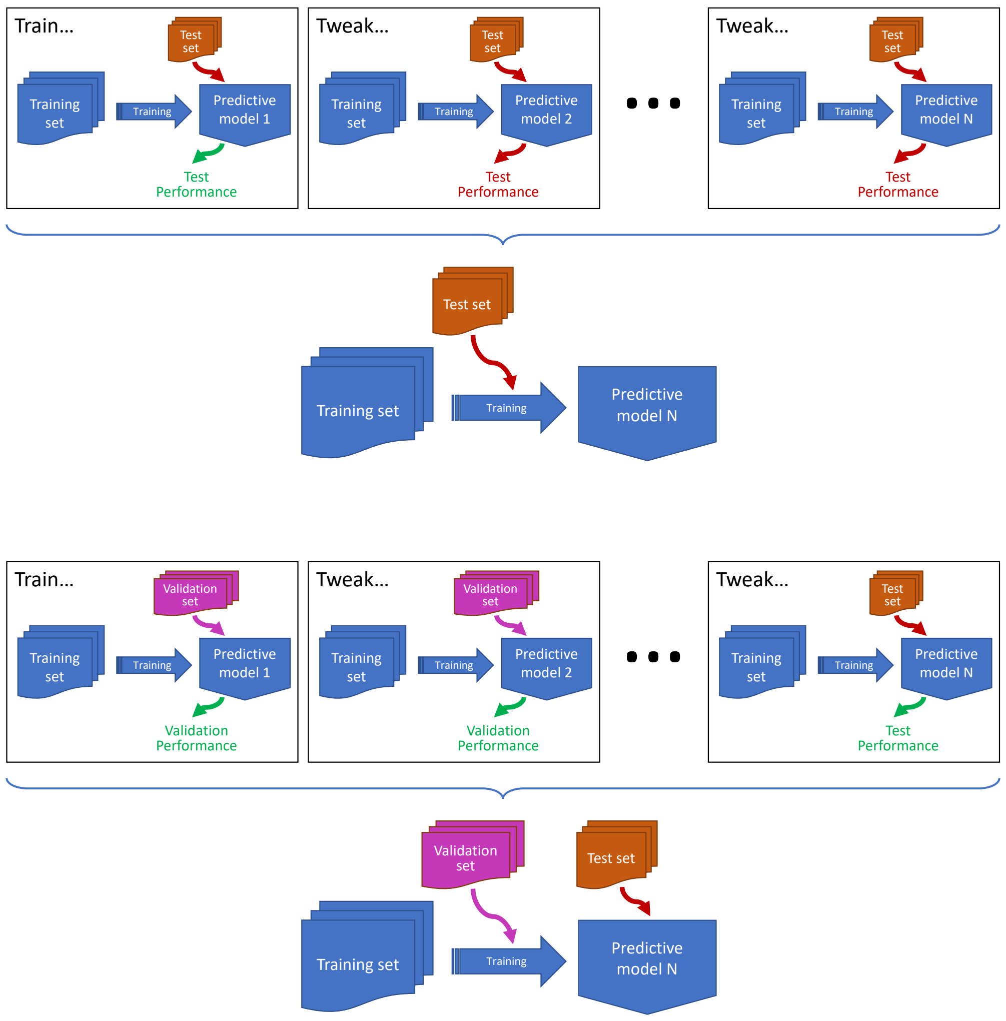


Figure 5: See Do avoid sequential overfitting. [top] Using the test set repeatedly during model selection results in the test set becoming an implicit part of the training process. [bottom] A validation set should be used instead during model selection, and the test set should only be used once to measure the generality of the final model.

data for each model trained. **Cross-validation** (CV) is a particularly popular way of doing this, and comes in numerous flavours [Arlot et al., 2010], most of which involve splitting the data into a number of folds. When doing CV, it is important to be aware of any dependencies within the data and take these into account. Failure to do so can result in data leakage. For instance, in medical datasets, it is commonplace to have multiple data points for a single subject; to avoid data leakage, these should be kept together within the same fold. Time series data is particularly problematic for CV; see Don't ignore temporal dependencies in time series data for a discussion of how to handle this. If you're carrying out hyperparameter optimisation, then you should use

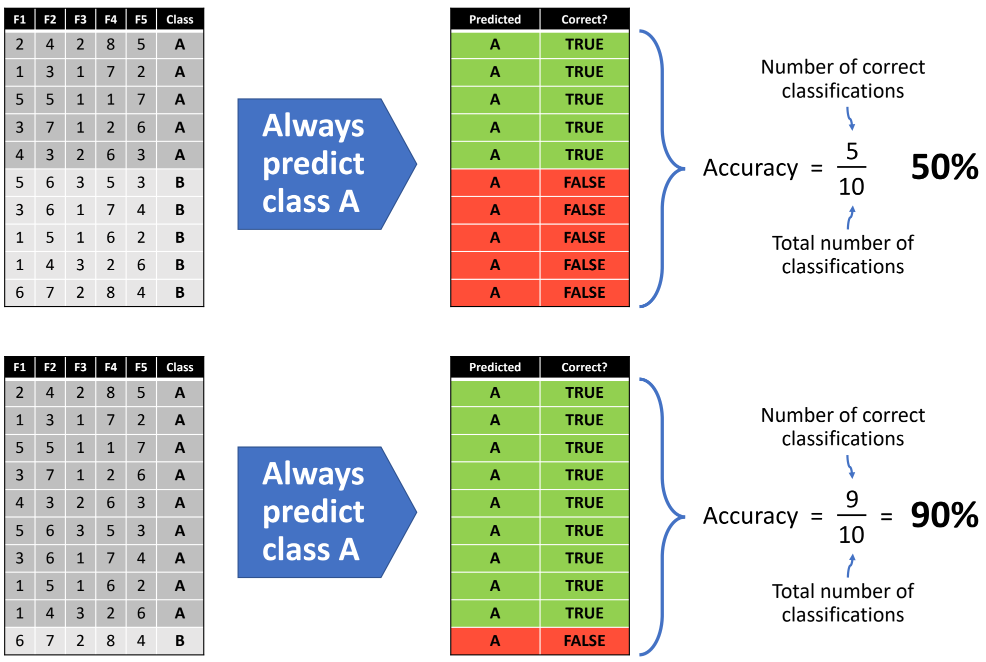
nested cross-validation (also known as double cross-validation), which uses an extra loop inside the main cross-validation loop to avoid overfitting the test folds. If some of your data classes are small, then you may need to do **stratification**, which ensures each class is adequately represented in each fold. In addition to looking at average performance across multiple evaluations, it is also standard practice to provide some measure of spread or confidence, such as the standard deviation or the 95% confidence interval.

4.5 Do save some data to evaluate your final model instance

I've used the term *model* quite loosely, but there is an important distinction between evaluating the potential of a general model (e.g. how well a neural network can solve your problem), and the performance of a particular model instance (e.g. a specific neural network produced by one run of back-propagation). Cross-validation is good at the former, but it's less useful for the latter. Say, for instance, that you carried out ten-fold cross-validation. This would result in ten model instances. Say you then select the instance with the highest test fold score as the model which you will use in practice. How do you report its performance? Well, you might think that its test fold score is a reliable measure of its performance, but it probably isn't. First, the amount of data in a single fold is relatively small. Second, the instance with the highest score could well be the one with the easiest test fold, so the evaluation data it contains may not be representative. Consequently, the only way of getting a reliable estimate of the model instance's generality may be to use another test set. This is also true in situations where the independence of the existing test set may have been compromised, e.g. by using it more than once (see Do avoid sequential overfitting). So, if you have enough data, it's better to keep some aside and only use it once to provide an unbiased estimate of the final selected model instance. However, it's worth noting one other option when using cross-validation, and that is to ensemble the model instances (see Do combine models (carefully)). The resulting ensemble will have performance in line with the average as measured through cross-validation, so another test set is not required to measure its performance. On the downside, it will likely have poorer inference time, efficiency and interpretability than a single model instance, so this approach is generally only worth considering if you have very little data.

4.6 Do choose metrics carefully

Be careful which metrics you use to evaluate your ML models. For instance, in the case of classification models, the most commonly used metric is accuracy, which is the proportion of samples in the data set that were correctly classified by the model. This works fine if your classes are balanced, i.e. if each class is represented by a similar number of samples within the data set. But many data sets are not balanced, and in this case accuracy can be a very misleading metric. Consider, for example, a data set in which 90% of the samples represent one class, and 10% of the samples represent another class. A binary classifier which always outputs the first class, regardless of its input, would have an accuracy of 90%, despite being completely useless (see Figure 6). In this kind



F1	F2	F3	F4	F5	Class
2	4	2	8	5	A
1	3	1	7	2	A
5	5	1	1	7	A
3	7	1	2	6	A
4	3	2	6	3	A
5	6	3	5	3	B
3	6	1	7	4	B
1	5	1	6	2	B
1	4	3	2	6	B
6	7	2	8	4	B

Predicted	Correct?
A	TRUE
A	TRUE
A	TRUE
A	TRUE
A	TRUE
A	FALSE
A	FALSE
A	FALSE
A	FALSE
A	FALSE

F1	F2	F3	F4	F5	Class
2	4	2	8	5	A
1	3	1	7	2	A
5	5	1	1	7	A
3	7	1	2	6	A
4	3	2	6	3	A
5	6	3	5	3	A
3	6	1	7	4	A
1	5	1	6	2	A
1	4	3	2	6	A
6	7	2	8	4	B

Predicted	Correct?
A	TRUE
A	TRUE
A	TRUE
A	TRUE
A	TRUE
A	TRUE
A	TRUE
A	TRUE
A	TRUE
A	FALSE

Always predict class A
Number of correct classifications
Accuracy = 5/10
50%
Total number of classifications
Always predict class A
Number of correct classifications
Accuracy = 9/10 = 90%
Total number of classifications

Figure 6: See **Do choose metrics carefully**. The problem with using accuracy as a performance metric on imbalanced data. Here, a dummy model which always predicts the same class label has an accuracy of 50% or 90% depending on the distribution of class labels within the data.

of situation, it would be preferable to use a metric such as F_1 score, Cohen's kappa coefficient (κ) or Matthews Correlation Coefficient (MCC), all of which are relatively insensitive to class size imbalance. For a broader review of methods for dealing with imbalanced data, see Haixiang et al. [2017]. There are also various pitfalls associated with regression metrics, particularly within the context of time series forecasting; see Hewamalage et al. [2023] for a discussion of these. A well-known example is relying only on the RMSE, which (a bit like accuracy) is susceptible to assigning high value to models which always predict no change. Also see Do report performance in multiple ways.

4.7 Do consider model fairness

Overall performance metrics are not the only important measures of how good a model is. If a model is to be deployed within the real world, another important measure is fairness. There are various definitions of fairness, but in a nutshell, it's about making sure that the model doesn't treat its human subjects unequally with regard to characteristics such as gender, ethnicity, income or personal politics. This is also referred to as **algorithmic bias**, and there are many examples of models being biased towards or against particular groups of people. A common source of unfairness is using an unrepresentative dataset to train an ML model. For instance, if a medical diagnosis model is trained on data from a single country, then the data may be biased towards the majority ethnicity, and the model may not operate fairly when exposed to users from other ethnicities. However, unfairness can also come from other sources, including subconscious bias during data preparation and the inductive biases of the model. Regardless of the source, it is important to understand any resulting biases, and ideally take steps to mitigate against them (e.g. applying data augmentation to minority samples — see Do make sure you have enough data). There are many different fairness metrics, so part of the puzzle is working out which are most relevant to your modelling context; see Caton and Haas [2024] for a review.

4.8 Don't ignore temporal dependencies in time series data

Time series data is unlike many other kinds of data in that the order of the data points is important. Many of the pitfalls in handling time series data are a result of ignoring this fact. Most notably, time series data are subject to a particular kind of data leakage (see Don't allow test data to leak into the training process) known as **look ahead bias**. This occurs when some or all of the data points used to train the model occur later in the time series than those used to test the model. In effect, this can allow knowledge of the future to leak into training, and this can then bias the test performance. A situation where this commonly occurs is when standard cross-validation (see Do evaluate a model multiple times) is applied to time series data, since it results in the training folds in all but one of the cross-validation iterations containing data that is in the future relative to the test fold. This can be avoided by using special forms of cross-validation that respect temporal dependencies, such as **blocked cross-validation**, though whether this is necessary depends to some extent on the nature of the time series data, e.g. whether it is stationary or non-stationary. See Cerqueira et al. [2020] and Wang and Ruf [2022] for more on this. Look ahead bias can also result from carrying out data-dependent preprocessing operations before splitting off the test data; see Figure 7 for a simple example of this, but also see Do be careful where and how you do feature selection.

5 How to compare models fairly

Comparing models is the basis of academic research, but it's surprisingly difficult to get it right. If you carry out a comparison unfairly, and publish it, then other researchers may subsequently be led astray. So, do make sure that you evaluate different models within the same context, do explore multiple perspectives, and do use make correct use of statistical tests.

5.1 Don't assume a bigger number means a better model

It's not uncommon for a paper to state something like "In previous research, accuracies of up to 94% were reported. Our model achieved 95%, and is therefore better." There are various reasons why a higher figure does not imply a better model. For instance, if the models were trained or evaluated on different partitions of the same data set, then small

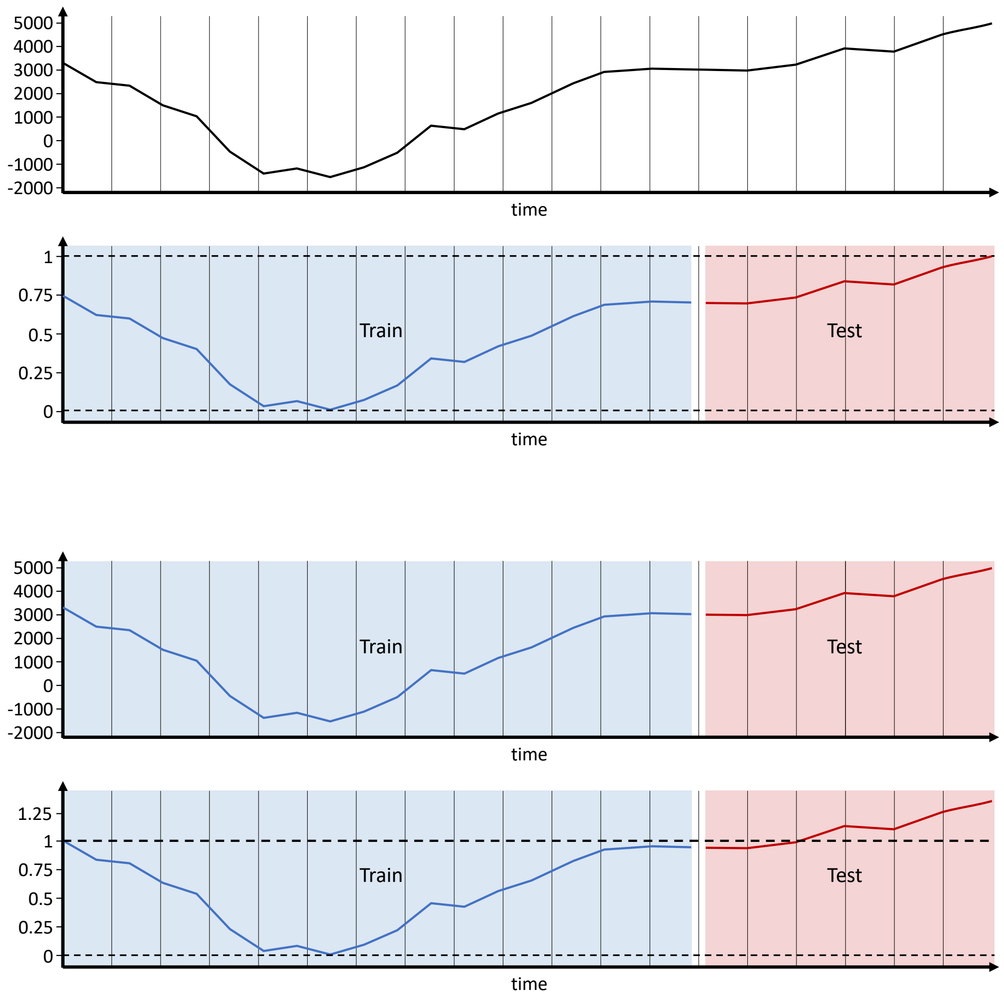


Figure 7: See Don't ignore temporal dependencies in time series data. [top] A time series is scaled to the interval [0, 1] before splitting off the test data (shown in red). This could allow the model to infer that values will increase in the future, causing a potential look ahead bias. [bottom] Instead, the data should be split before doing scaling, so that information about the range of the test data can't leak into the training data.

differences in performance may be due to this. If the datasets had different degrees of class imbalance, then the difference in accuracy could merely reflect this (see Do choose metrics carefully). If they used different data sets entirely, then this may account for even large differences in performance. Another reason for unfair comparisons is the failure to carry out the same amount of hyperparameter optimisation (see Do optimise your model's hyperparameters) when comparing models; for instance, if one model has default settings and the other has been optimised, then the comparison won't be fair. For these reasons, and others, comparisons based on published figures should always be treated with caution. To be sure of a fair comparison between two approaches, you should freshly implement all the models you're comparing, optimise each one to the same degree, carry out multiple evaluations (see Do evaluate a model multiple times), and then use statistical tests (see Do use statistical tests when comparing models) to determine whether the differences in performance are significant. A further complication when comparing foundation models (see Do keep up with progress in deep learning (and its pitfalls)) it that the original training data is often unknown; consequently it may be impossible to ensure that the test set is independent of the training data, and therefore a fair basis for comparison.

5.2 Do use meaningful baselines

When introducing a new modelling approach, it is essential to compare against established approaches. These are commonly referred to as **baseline models**, or just baselines. It is important that these baselines are selected so that they provide a meaningful basis for comparison. Baselines are often simpler than the new approach, and are chosen to demonstrate that any complexity in the new model is necessary. For example, if you're extending model X, then it makes sense to use model X as a baseline. However, it also makes sense to use other simpler models. For instance, if you're developing a deep learning approach that uses tabular data, then you should also compare against simpler models like decision trees and SVMs to show that a more complex approach is justified. If you're solving a regression problem, then you should also consider using simple baselines like logistic regression. The simplest baselines are known as **naïve baselines** and are used to show that your model is not doing something trivial. An illustrative example of why these are necessary is described in Hewamalage et al. [2023], where a complex transformer model designed for time series forecasting is shown to perform worse than a naïve baseline that always forecasts the next value in a time series to be the same as the previous value. This kind of naïve baseline, in which there is no real decision making process, is also known as a **dummy model**. Another example is a classifier that always outputs the most frequent class label (as described in Do choose metrics carefully). In addition to simple baselines, it's also important to compare against state-of-the-art (**SOTA**) models. Otherwise you may be asked something like "Why are you extending model X when model Y is known to be better than model X?"

5.3 Do use statistical tests when comparing models

If you want to convince people that your model is better than someone else's, then a statistical test can be a useful tool. Broadly speaking, there are two categories of tests for comparing individual ML models. The first is used to compare individual model instances, e.g. two trained decision trees. For example, McNemar's test is a fairly common choice for comparing two classifiers, and works by comparing the classifiers' output labels for each sample in the test set (so do remember to record these). The second category of tests are used to compare two models more generally, e.g. whether a decision tree or a neural network is a better fit for the data. These require multiple evaluations of each model, which you can get by using cross-validation or repeated resampling (or, if your training algorithm is stochastic, multiple repeats using the same data). The test then compares the two resulting distributions. Student's T test is a common choice for this kind of comparison, but it's only reliable when the distributions are normally distributed, which is often not the case. A safer bet is Mann-Whitney's U test, since this does not assume that the distributions are normal. For more information, see Raschka [2020] and Carrasco et al. [2020]. Also see Do correct for multiple comparisons and Do be careful when reporting statistical significance.

5.4 Do correct for multiple comparisons

Things get a bit more complicated when you want to use statistical tests to compare more than two models, since doing multiple pairwise tests is a bit like using the test set multiple times — it can lead to overly-optimistic interpretations of significance. Basically, each time you carry out a comparison between two models using a statistical test, there's a probability that it will discover significant differences where there aren't any. This is represented by the confidence level of the test, usually set at 95%: meaning that 1 in 20 times it will give you a false positive. For a single comparison, this may be a level of uncertainty you can live with. However, it accumulates. That is, if you do 20 pairwise tests with a confidence level of 95%, one of them is likely to give you the wrong answer. This is known as the **multiplicity effect**, and is an example of a broader issue in data science known (at least when done intentionally) as **data dredging** or **p-hacking** — see Stefan and Schönbrodt [2023]. To address this problem, you can apply a correction for multiple tests. The most common approach is the Bonferroni correction, a very simple method that lowers the significance threshold based on the number of tests that are being carried out; see Salzberg [1997] for a gentle introduction. However, there are numerous other approaches, and there is also some debate about when and where these corrections should be applied; for an accessible overview, see Streiner [2015].

5.5 Don't always believe results from community benchmarks

In certain problem domains, it has become commonplace to use benchmark data sets to evaluate new ML models. The idea is that, because everyone is using the same data to train and test their models, then comparisons will be more transparent. Unfortunately this approach has some major drawbacks. First, if access to the test set is unrestricted, then you can't assume that people haven't used it as part of the training process. This is known as "training to the test set", and leads to results that are heavily over-optimistic. A more subtle problem is that, even if everyone who uses the data only uses the test set once, collectively the test set is being used many times by the community. In effect, by comparing lots of models on the same test set, it becomes increasingly likely that the best model just happens to over-fit the test set, and doesn't necessarily generalise any better than the other models (see Do correct for multiple comparisons and Do avoid sequential overfitting). For these, and other reasons, you should be careful how much you read into results from a benchmark data set, and don't assume that a small increase in performance is significant. This is particularly the case where foundation models (see Do keep up with progress in deep learning (and its pitfalls)) are used, since it is possible that their training data included the test sets from community benchmarks. See Paullada et al. [2021] for a wider discussion of issues surrounding the use of shared datasets. Also see Do report performance in multiple ways.

5.6 Do combine models (carefully)

Whilst this section focuses on comparing models, it's good to be aware that ML is not always about choosing between models. Often it makes sense to use combinations of models. Different ML models explore different trade-offs; by combining them, you can sometimes compensate for the weaknesses of one model by using the strengths of another model, and vice versa. **Ensembles** are a well-established group of composite models. There are lots of ensemble learning approaches — see Dong et al. [2020] for a review — but they can be roughly divided into those that form ensembles out of the same base model type (examples include random forests, bagging and boosting) and those that combine different types of ML model. An example of the latter is **stacked generalisation** (or stacking), where a model is trained to aggregate the outputs of a group of base models. However, ensembles are not the only kind of composition. Another, increasingly common, form of composition occurs when **embedding models** (such as autoencoders or foundation models such as BERT) are used to provide input to other models. When using stacking or embedding, it's important to ensure that no data leaks (see Don't allow test data to leak into the training process) occur, i.e. that the test data used to measure the performance of the composite model is not used in the training of any of its components. This is a common pitfall, especially when the model components are trained on overlapping data. To reduce the likelihood of sequential overfitting (see Do avoid sequential overfitting), it is also advisable to use a separate test set to evaluate the composite model.

6 How to report your results

The aim of academic research is not self-aggrandisement, but rather an opportunity to contribute to knowledge. In order to effectively contribute to knowledge, you need to provide a complete picture of your work, covering both what worked and what didn't. ML is often about trade-offs — it's very rare that one model is better than another in every way that matters — and you should try to reflect this with a nuanced and considered approach to reporting results and conclusions.

6.1 Do be transparent

First of all, always try to be transparent about what you've done, and what you've discovered, since this will make it easier for other people to build upon your work. In particular, it's good practice to share your models in an accessible way. For instance, if you used a script to implement all your experiments, then share the script when you publish the results. This means that other people can easily repeat your experiments, which adds confidence to your work. It also makes it a lot easier for people to compare models, since they no longer have to reimplement everything from scratch in order to ensure a fair comparison. Knowing that you will be sharing your work also encourages you to be more careful, document your experiments well, and write clean code, which benefits you as much as anyone else. It's also worth noting that issues surrounding reproducibility are gaining prominence in the ML community, so in the future you may not be able to publish work unless your workflow is adequately documented and shared — for example, see Pineau et al. [2021]. Checklists (Do use a machine learning checklist) are useful for knowing what to include in your workflow. You might also find **experiment tracking** frameworks, such as MLflow [Chen et al., 2020], useful for recording your workflow.

6.2 Do report performance in multiple ways

One way to achieve better rigour when evaluating and comparing models is to use multiple data sets. This helps to overcome any deficiencies associated with individual data sets (see Don't always believe results from community benchmarks) and allows you to present a more complete picture of your model's performance. It's also good practice to report multiple metrics for each data set, since different metrics can present different perspectives on the results, and increase the transparency of your work. For example, if you use accuracy, it's also a good idea to include metrics that are less sensitive to class imbalances (see Do choose metrics carefully). In domains such as medicine and security, it's important to know where errors are being made; for example, when your model gets things wrong, is it more inclined to false positives or false negatives? Metrics that summarise everything in one number, such as accuracy, give no insight into this. So, it's important to also include partial metrics such as precision and recall, or sensitivity and specificity, since these do provide insight into the types of errors your model produces. And make sure it's clear which metrics you are using. For instance, if you report F-scores, be clear whether this is F_1, or some other balance between precision and recall. If you report AUC, indicate whether this is the area under the ROC curve or the PR curve. For a broader discussion, see Blagec et al. [2020].

6.3 Don't generalise beyond the data

It's important not to present invalid conclusions, since this can lead other researchers astray. A common mistake is to make general statements that are not supported by the data used to train and evaluate models. For instance, if your model does really well on one data set, this does not mean that it will do well on other data sets. Whilst you can get more robust insights by using multiple data sets (see Do report performance in multiple ways), there will always be a limit to what you can infer from any experimental study. There are numerous reasons for this (see Paullada et al. [2021]), many of which are to do with how datasets are curated. One common issue is bias, or **sampling error**: that the data is not sufficiently representative of the real world. Another is overlap:

multiple data sets may not be independent, and may have similar biases. There's also the issue of quality: and this is a particular issue in deep learning datasets, where the need for quantity of data limits the amount of quality checking that can be done. So, in short, don't overplay your findings, and be aware of their limitations.

6.4 Do be careful when reporting statistical significance

I've already discussed statistical tests (see Do use statistical tests when comparing models), and how they can be used to determine differences between ML models. However, statistical tests are not perfect. Some are conservative, and tend to under-estimate significance; others are liberal, and tend to over-estimate significance. This means that a positive test doesn't always indicate that something is significant, and a negative test doesn't necessarily mean that something isn't significant. Then there's the issue of using a threshold to determine significance; for instance, a 95% confidence threshold (i.e. when the p-value < 0.05) means that 1 in 20 times a difference flagged as significant won't be significant. In fact, statisticians are increasingly arguing that it is better not to use thresholds, and instead just report p-values and leave it to the reader to interpret these [Betensky, 2019]. Beyond statistical significance, another thing to consider is whether the difference between two models is actually important. If you have enough samples, you can always find significant differences, even when the actual difference in performance is miniscule. To give a better indication of whether something is important, you can measure **effect size**. There are a range of approaches used for this: Cohen's d statistic is probably the most common, but more robust approaches, such as Kolmogorov-Smirnov, are preferable. For more on effect size and reporting statistical significance, see Aguinis et al. [2021]. You might also consider using Bayesian statistics; although there's less guidance and tools support available, these theoretically have a lot going for them, and they avoid many of the pitfalls associated with traditional statistical tests — see Benavoli et al. [2017] for more info.

6.5 Do look at your models

Trained models contain a lot of useful information. Unfortunately many authors just report the performance metrics of a trained model, without giving any insight into what it actually learnt. Remember that the aim of research is not to get a slightly higher accuracy than everyone else. Rather, it's to generate knowledge and understanding and share this with the research community. If you can do this, then you're much more likely to get a decent publication out of your work. So, do look inside your models and do try to understand how they reach a decision. For relatively simple models like decision trees, it can also be beneficial to provide visualisations of your models, and most libraries have functions that will do this for you. For more complex models, there are a range of *explainable AI* (XAI) techniques that can be used. Some of these are model-specific, and others are model-agnostic. Well-established examples of the latter are LIME and SHAP; both give insights into which features are important for a model. For CNNs and vision transformers, a common approach is to use **saliency maps**, which show

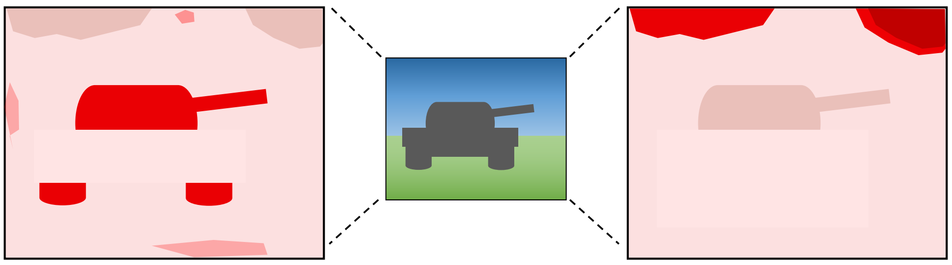


Figure 8: See Do look at your models. Using saliency maps to analyse vision-based deep learning models. Imagine these two maps (in red) were generated for the image shown in the centre, for two different deep learning models trained on the kind of tank recognition data mentioned in Do avoid learning spurious correlations. Darker colours indicate features that are of greater importance to the model, so the model on the left (which predominantly focuses on the components of the tank) is likely to generalise much better than the one on the right (which predominantly focuses on the background of the image).

the importance of different parts of an input image — see Figure 8 for an illustrative example. Grad-CAM is a popular technique for generating these, but there are plenty of other methods too. For non-vision transformers, a common approach is to visualise attention weights. See Dwivedi et al. [2023] for a survey of XAI techniques, and Ali et al. [2023] for a discussion of the limitations of current approaches. Whilst XAI techniques can give you useful insights into a model's behaviour, it's important to bear in mind that they are unlikely to tell you exactly what a model is doing. This is particularly the case for deep learning models (see Don't assume deep learning will be the best approach), whose complexity makes their behaviour inherently difficult to analyse. For complex models, ablation studies Meyes et al. [2019] can also be useful. This involves successively removing parts of the model to see what is important, and can result in a simpler model which is more amenable to analysis.

6.6 Do use a machine learning checklist

This guide aims to give an appreciation of the main things that can go wrong during machine learning, plus some guidance on how to avoid these things going wrong. Checklists, on the other hand, are designed to take you more formally through the ML pipeline and encourage you to document (and more importantly, think about) how your implementation decisions support a meaningful outcome. In some domains, e.g. certain fields of medicine, it is compulsory to complete a checklist before submitting a paper for publication. However, beyond their quality assurance role, checklists are arguably most useful at the start of a study when it comes to planning an ML pipeline. Since I'm one of the authors, I'd particularly encourage you to look at REFORMS [Kapoor et al., 2024], which is a combined checklist and set of consensus-based recommendations for doing ML-based science (although much of it is also applicable to ML practice more generally). Other, more domain-specific, checklists are also available.

7 Final thoughts

ML is becoming an important part of people's lives, yet the practice of ML is arguably in its infancy. There are many easy-to-make mistakes that can cause an ML model to appear to perform well, when in reality it does not. In turn, this has the potential to misinform when these models are published, and the potential to cause harm if these models are ever deployed. This guide describes the most common of these mistakes, and also touches upon more general issues of good practice in ML, such as fairness, transparency and the avoidance of bias. It also offers advice on avoiding these pitfalls. However, new threats continue to emerge as new approaches to ML are developed, and it is therefore important for users of ML to remain vigilant. This is the nature of a fast-moving research area — the theory of how to do ML almost always lags behind the practice, practitioners will always disagree about the best ways of doing things, and what we think is correct today may not be correct tomorrow.

You can find more on ML pitfalls at my Substack, Fetch Decode Execute.

Acknowledgements

Many thanks to everyone who gave me feedback on the draft manuscript, to everyone who has since sent me suggestions for revisions and new content, and to the editor and peer reviewers of the version published in *Patterns*.

References

Where available, preprint URLs are also included for papers that are not open access.

H. Aguinis, M. Vassar, and C. Wayant. On reporting and interpreting statistical significance and p values in medical research. *BMJ Evidence-Based Medicine*, 26(2):39–42, 2021. http://doi.org/10.1136/bmjebm-2019-111264.

S. Ali, T. Abuhmed, S. El-Sappagh, K. Muhammad, J. M. Alonso-Moral, R. Confalonieri, R. Guidotti, J. Del Ser, N. Díaz-Rodríguez, and F. Herrera. Explainable artificial intelligence (xai): What we know and what is left to attain trustworthy artificial intelligence. *Information fusion*, 99:101805, 2023. https://doi.org/10.1016/j.inffus.2023.101805.

S. Arlot, A. Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010. https://doi.org/10.1214/09-SS054.

D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck. Dos and don'ts of machine learning in computer security. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3971–3988, 2022. https://www.usenix.org/system/files/sec22-arp.pdf.

R. Barbudo, S. Ventura, and J. R. Romero. Eight years of automl: categorisation, review and trends. *Knowledge and Information Systems*, 65(12):5097–5149, 2023. https://doi.org/10.1007/s10115-023-01935-1.

A. Benavoli, G. Corani, J. Demšar, and M. Zaffalon. Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. *The Journal of Machine Learning Research*, 18(1):2653–2688, 2017. https://jmlr.org/papers/v18/16-305.html.

R. A. Betensky. The p-value requires context, not a threshold. *The American Statistician*, 73(sup1):115–117, 2019. https://doi.org/10.1080/00031305.2018.1529624.

B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix, et al. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(2):e1484, 2023. https://doi.org/10.1002/widm.1484.

K. Blagec, G. Dorffner, M. Moradi, and M. Samwald. A critical analysis of metrics used for measuring progress in artificial intelligence. *Preprint at arXiv*, 2020. https://arxiv.org/abs/2008.02577.

A. D. Blaom, F. Kiraly, T. Lienart, Y. Simillides, D. Arenas, and S. J. Vollmer. Mlj: A julia package for composable machine learning. *Journal of Open Source Software*, 5 (55):2704, 2020. https://doi.org/10.21105/joss.02704.

J. Cai, J. Luo, S. Wang, and S. Yang. Feature selection in machine learning: A new perspective. *Neurocomputing*, 300:70–79, 2018. https://doi.org/10.1016/j.neucom.2017.11.077.

J. Carrasco, S. García, M. Rueda, S. Das, and F. Herrera. Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: Practical guidelines and a critical review. *Swarm and Evolutionary Computation*, 54:100665, 2020. https://doi.org/10.1016/j.swevo.2020.100665.

S. Caton and C. Haas. Fairness in machine learning: A survey. *ACM Computing Surveys*, 56(7):1–38, 2024. https://doi.org/10.1145/3616865.

G. C. Cawley and N. L. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *The Journal of Machine Learning Research*, 11: 2079–2107, 2010. https://www.jmlr.org/papers/volume11/cawley10a/cawley10a.pdf.

V. Cerqueira, L. Torgo, and I. Mozetič. Evaluating time series forecasting models: An empirical study on performance estimation methods. *Machine Learning*, 109(11): 1997–2028, 2020. https://doi.org/10.1007/s10994-020-05910-7.

A. Chen, A. Chow, A. Davidson, A. DCunha, A. Ghodsi, S. A. Hong, A. Konwinski, C. Mewald, S. Murching, T. Nykodym, et al. Developments in mlflow: A system to accelerate the machine learning lifecycle. In *Proceedings of the fourth international* *workshop on data management for end-to-end machine learning*, pages 1–4, 2020. https://doi.org/10.1145/3399579.3399867.

P.-O. Côté, A. Nikanjam, N. Ahmed, D. Humeniuk, and F. Khomh. Data cleaning and machine learning: a systematic literature review. *Automated Software Engineering*, 31(2):54, 2024. https://doi.org/10.1007/s10515-024-00453-w (preprint: https://doi.org/10.48550/arXiv.2310.01765).

V. Cox. Exploratory data analysis. In *Translating Statistics to Make Decisions*, pages 47–74. Springer, 2017.

X. Dong, Z. Yu, W. Cao, Y. Shi, and Q. Ma. A survey on ensemble learning. *Frontiers of Computer Science*, 14(2):241–258, 2020. https://doi.org/10.1007/s11704-019-8208-z.

R. Dwivedi, D. Dave, H. Naik, S. Singhal, R. Omer, P. Patel, B. Qian, Z. Wen, T. Shah, G. Morgan, et al. Explainable ai (xai): Core ideas, techniques, and solutions. *ACM Computing Surveys*, 55(9):1–33, 2023. https://doi.org/10.1145/3561048.

T. Emmanuel, T. Maupong, D. Mpoeleng, T. Semong, B. Mphago, and O. Tabona. A survey on missing data in machine learning. *Journal of Big data*, 8:1–37, 2021. https://doi.org/10.1186/s40537-021-00516-9.

E. Gibney. Is AI fuelling a reproducibility crisis in science? *Nature*, 608:250–251, 2022. https://doi.org/10.1038/d41586-022-02035-w.

L. Grinsztajn, E. Oyallon, and G. Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in neural information processing systems*, 35:507–520, 2022. https://openreview.net/pdf?id=Fp7__phQszn.

G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing. Learning from class-imbalanced data: Review of methods and applications. *Expert Systems with Applications*, 73:220–239, 2017. https://doi.org/10.1016/j.eswa.2016.12.035.

X. Han, Z. Zhang, N. Ding, Y. Gu, X. Liu, Y. Huo, J. Qiu, Y. Yao, A. Zhang, L. Zhang, et al. Pre-trained models: Past, present and future. *AI Open*, 2:225–250, 2021. https://doi.org/10.1016/j.aiopen.2021.08.002.

H. Hewamalage, K. Ackermann, and C. Bergmeir. Forecast evaluation for data scientists: common pitfalls and best practices. *Data Mining and Knowledge Discovery*, 37(2): 788–832, 2023. https://doi.org/10.1007/s10618-022-00894-5.

M. Hosseini, M. Powell, J. Collins, C. Callahan-Flintoft, W. Jones, H. Bowman, and B. Wyble. I tried a bunch of things: The dangers of unexpected overfitting in classification of brain data. *Neuroscience & Biobehavioral Reviews*, 119:456–467, 2020. https://doi.org/10.1016/j.neubiorev.2020.09.036 (preprint: https://www.biorxiv.org/content/10.1101/078816v2.abstract).

G. Iglesias, E. Talavera, Á. González-Prieto, A. Mozo, and S. Gómez-Canaval. Data augmentation techniques in time series domain: a survey and taxonomy. *Neural Computing and Applications*, 35(14):10123–10145, 2023. https://doi.org/10.1007/s00521-023-08459-3.

S. Kapoor and A. Narayanan. Leakage and the reproducibility crisis in machine-learning-based science. *Patterns*, 4(9):100804, 2023. ISSN 2666-3899. https://doi.org/10.1016/j.patter.2023.100804.

S. Kapoor, E. M. Cantrell, K. Peng, T. H. Pham, C. A. Bail, O. E. Gundersen, J. M. Hofman, J. Hullman, M. A. Lones, M. M. Malik, P. Nanayakkara, R. A. Poldrack, I. D. Raji, M. Roberts, M. J. Salganik, M. Serra-Garcia, B. M. Stewart, G. Vandewiele, and A. Narayanan. REFORMS: Consensus-based recommendations for machine-learning-based science. *Science Advances*, 10(18):eadk3452, 2024. https://doi.org/10.1126/sciadv.adk3452.

P. Karande, B. Gallagher, and T. Y.-J. Han. A strategic approach to machine learning for material science: How to tackle real-world challenges and avoid pitfalls. *Chemistry of Materials*, 34(17):7650–7665, 2022. https://doi.org/10.1021/acs.chemmater.2c01333.

S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022. https://doi.org/10.1145/3505244 (preprint: https://doi.org/10.48550/arXiv.2101.01169).

D. Kreuzberger, N. Kühl, and S. Hirschl. Machine learning operations (MLOps): Overview, definition, and architecture. *IEEE access*, 2023. https://doi.org/10.1109/ACCESS.2023.3262138.

M. Kuhn and H. Wickham. *Tidymodels: a collection of packages for modeling and machine learning using tidyverse principles*, 2020. https://www.tidymodels.org.

H. Li, Y. Chen, J. Luo, Y. Kang, X. Zhang, Q. Hu, C. Chan, and Y. Song. Privacy in large language models: Attacks, defenses and future directions. *Preprint at arXiv*, 2023. https://doi.org/10.48550/arXiv.2310.10383.

Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, 2021. https://doi.org/10.1109/TNNLS.2021.3084827 (preprint: https://doi.org/10.48550/arXiv.2004.02806).

T. Liao, R. Taori, I. D. Raji, and L. Schmidt. Are we learning yet? a meta review of evaluation failures across machine learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. https://openreview.net/forum?id=mPducS1MsEK.

T. Lin, Y. Wang, X. Liu, and X. Qiu. A survey of transformers. *AI Open*, 2022. https://doi.org/10.1016/j.aiopen.2022.10.001.

M. A. Lones. Avoiding machine learning pitfalls. *Patterns*, 2024. https://doi.org/10.1016/j.patter.2024.101046.

M. M. Malik. A hierarchy of limitations in machine learning. *Preprint at arXiv*, 2020. https://doi.org/10.48550/arXiv.2002.05193.

R. Meyes, M. Lu, C. W. de Puiseau, and T. Meisen. Ablation studies in artificial neural networks. *Preprint at arXiv*, 2019. https://doi.org/10.48550/arXiv.1901.08644.

C. Molnar, G. König, J. Herbinger, T. Freiesleben, S. Dandl, C. A. Scholbeck, G. Casalicchio, M. Grosse-Wentrup, and B. Bischl. General pitfalls of model-agnostic interpretation methods for machine learning models. In *International Workshop on Extending Explainable AI Beyond Deep Models and Classifiers*, pages 39–68. Springer, 2020. https://doi.org/10.1007/978-3-031-04083-2_4.

A. Paullada, I. D. Raji, E. M. Bender, E. Denton, and A. Hanna. Data and its (dis)contents: A survey of dataset development and use in machine learning research. *Patterns*, 2(11):100336, 2021. ISSN 2666-3899. https://doi.org/10.1016/j.patter.2021.100336.

J. Pineau, P. Vincent-Lamarre, K. Sinha, V. Larivière, A. Beygelzimer, F. d'Alché Buc, E. Fox, and H. Larochelle. Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program). *Journal of machine learning research*, 22(164):1–20, 2021. https://www.jmlr.org/papers/volume22/20-303/20-303.pdf.

S. Raschka. Model evaluation, model selection, and algorithm selection in machine learning. *Preprint at arXiv*, 2020. https://arxiv.org/abs/1811.12808.

M. Roberts, D. Driggs, M. Thorpe, J. Gilbey, M. Yeung, S. Ursprung, A. I. Aviles-Rivero, C. Etmann, C. McCague, L. Beer, et al. Common pitfalls and recommendations for using machine learning to detect and prognosticate for covid-19 using chest radiographs and ct scans. *Nature Machine Intelligence*, 3(3):199–217, 2021. https://doi.org/10.1038/s42256-021-00307-0.

C. Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019. https://doi.org/10.1038/s42256-019-0048-x (preprint: https://doi.org/10.48550/arXiv.1811.10154).

S. L. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data mining and knowledge discovery*, 1(3):317–328, 1997. https://doi.org/10.1023/A:1009752403260.

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015. https://doi.org/10.1016/j.neunet.2014.09.003 (preprint: https://doi.org/10.48550/arXiv.1404.7828).

D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, 28:2503–2511, 2015. https://papers.nips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf.

S. Shankar, R. Garcia, J. M. Hellerstein, and A. G. Parameswaran. Operationalizing machine learning: An interview study. *Preprint at arXiv*, 2022. https://doi.org/10.48550/arXiv.2209.09125.

A. M. Stefan and F. D. Schönbrodt. Big little lies: A compendium and simulation of p-hacking strategies. *Royal Society Open Science*, 10(2):220346, 2023. https://doi.org/10.1098/rsos.220346.

L. M. Stevens, B. J. Mortazavi, R. C. Deo, L. Curtis, and D. P. Kao. Recommendations for reporting machine learning analyses in clinical research. *Circulation: Cardiovascular Quality and Outcomes*, 13(10):e006556, 2020. https://doi.org/10.1161/CIRCOUTCOMES.120.006556.

D. L. Streiner. Best (but oft-forgotten) practices: the multiple problems of multiplicity—whether and how to correct for many statistical tests. *The American journal of clinical nutrition*, 102(4):721–728, 2015. https://doi.org/10.3945/ajcn.115.113548.

B. Van Giffen, D. Herhausen, and T. Fahse. Overcoming the pitfalls and perils of algorithms: A classification of machine learning biases and mitigation methods. *Journal of Business Research*, 144:93–106, 2022. https://doi.org/10.1016/j.jbusres.2022.01.076.

G. Vandewiele, I. Dehaene, G. Kovács, L. Sterckx, O. Janssens, F. Ongenae, F. De Backere, F. De Turck, K. Roelens, J. Decruyenaere, S. Van Hoecke, and T. Demeester. Overly optimistic prediction results on imbalanced data: a case study of flaws and benefits when applying over-sampling. *Artificial Intelligence in Medicine*, 111:101987, 2021. https://doi.org/10.1016/j.artmed.2020.101987 (preprint: https://arxiv.org/abs/2001.06296).

G. Varoquaux, L. Buitinck, G. Louppe, O. Grisel, F. Pedregosa, and A. Mueller. Scikit-learn: Machine learning without learning the machinery. *GetMobile: Mobile Computing and Communications*, 19(1):29–33, 2015. https://doi.org/10.1145/2786984.2786995.

W. Wang and J. Ruf. Information leakage in backtesting. *Available at SSRN 3836631*, 2022. https://doi.org/10.2139/ssrn.3836631.

Z. Wang, P. Wang, K. Liu, P. Wang, Y. Fu, C.-T. Lu, C. C. Aggarwal, J. Pei, and Y. Zhou. A comprehensive survey on data augmentation. *Preprint at arXiv*, 2024. https://doi.org/10.48550/arXiv.2405.09591.

S. Whalen, J. Schreiber, W. S. Noble, and K. S. Pollard. Navigating the pitfalls of applying machine learning in genomics. *Nature Reviews Genetics*, 23(3):169–181, 2022. https://doi.org/10.1038/s41576-021-00434-9.

D. H. Wolpert. The supervised learning no-free-lunch theorems. *Soft computing and industry*, pages 25–42, 2002. https://doi.org/10.1007/978-1-4471-0123-9_3.

A. Zeng, M. Chen, L. Zhang, and Q. Xu. Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 11121–11128, 2023. https://ojs.aaai.org/index.php/AAAI/article/view/26317/26089.

A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. *Dive into deep learning*. Cambridge University Press, 2023. https://d2l.ai.

C. Zhou, Q. Li, C. Li, J. Yu, Y. Liu, G. Wang, K. Zhang, C. Ji, Q. Yan, L. He, et al. A comprehensive survey on pretrained foundation models: A history from bert to chatgpt. *Preprint at arXiv*, 2023. https://doi.org/10.48550/arXiv.2302.09419.

J.-J. Zhu, M. Yang, and Z. J. Ren. Machine learning in environmental research: common pitfalls and best practices. *Environmental Science & Technology*, 57(46):17671–17689, 2023. https://doi.org/10.1021/acs.est.3c00026.