ML project review checklist

MUST-ASK QUESTIONS IN BOLD

P High-level questions about the project

- P.1 ▢ **What question were you trying to answer? How did you frame it as an ML task?**
- P.2 ▢ What is human-level performance on that task? What level of performance is needed?
- P.3 ▢ Is it possible to approach this problem without machine learning?
- P.4 ▢ If the analysis focused on deep learning methods, did you try shallow learning methods?
- P.5 ▢ What are the ethical and legal aspects of this project?
- P.6 ▢ **Which domain experts were involved in this analysis?**
- P.7 ▢ Which data scientists were involved in this analysis?
- P.8 ▢ Which tools or framework did you use? (How much of a known quantity is it?)
- P.9 ▢ Where is the pipeline published? (E.g. public or internal git repositories.)
- P.10 ▢ How thorough is the testing and documentation?

D Questions about the data preparation

- D.1 ▢ Where did the (a) feature data and (b) labels come from?
- D.2 ▢ What kind of data exploration did you do?
- D.3 ▢ What did you do about missing data? E.g. what kind of imputation did you do?
- D.4 ▢ How did you clean the data? How long did this take?
- D.5 ▢ What kind of normalization or standardization did you do?
- D.6 ▢ **Are the classes balanced? How did the distribution change your workflow?**
- D.7 ▢ What kind of data augmentation (new rows) or feature engineering (columns) did you do?
- D.8 ▢ **How did you split the data into train, validate and test? Or did you use some form of CV?**
- D.9 ▢ How do the feature and label distributions compare across datasets or folds?

T Questions about training and evaluation

- T.1 ▢ Which models did you explore and why? Did you also try the simplest models that fit the problem?
- T.2 ▢ How did you tune the hyperparameters of the model? Did you try grid search or other methods?
- T.3 ▢ How much regularization was applied, and of what kind?
- T.4 ▢ Did you do any dimensionality reduction, or check feature importances?
- T.5 ▢ **What kind of validation did you do? Did you use cross-validation?**
- T.6 ▢ What evaluation metric are you using? Why is it the most appropriate one? What is its distribution?
- T.7 ▢ **How do training, validation, and test metrics compare?**
- T.8 ▢ How does a dummy model score on this task?
- T.9 ▢ What is the state of the art, or human-level performance, on this task?
- T.10 ▢ If this was a classification task, are probabilities available in your model and did you use them?
- T.11 ▢ If this was a regression task, have you checked the residuals for normality and homoscedasticity?
- T.12 ▢ How explainable is your model? That is, do the learned parameters mean anything?
- T.13 ▢ Are there benchmarks for this task, and how well does your model do on them?

N Next steps for the project

- N.1 ▢ How will you deploy the model?
- N.2 ▢ How will you improve the model?
- N.3 ▢ Would collecting more data help? Can we address any imbalance with more data?
- N.4 ▢ Are there human or computing resources you need access to?

Checklist commentary

High-level questions about the project

P.1 What question were you trying to answer? How did you frame it as an ML task?

It's a good idea to articulate the task clearly, framing it explicitly in machine learning terms. We need to know:

- What will the model predict (**y**), what are the features (columns of **X**), and where did the data come from? How many data instances (rows of **X**) are there in the training set?
- How many targets per data instance? Specifically, are there single or multiple labels for classification, or one or more numbers for regression?
- If it's a classification problem, how many classes are there in total? In particular, are there two (e.g. pay or not-pay) or more (e.g. longitudinal, transverse, conjugate, oblique, orthogonal fractures).
- Whether it's a supervised, unsupervised or semi-supervised problem, or something else such as image segmentation, language parsing, etc.

For example, this is a reasonable response:

> *Our model predicts the lithology of a sample based on 5 features (4 wireline logs and 1 seismic trace). The task is framed as an 8-class, single-label, semi-supervised classification. For training, we have human-generated labels for 16 200 out of 142 000 samples from the Lambda Basin.*

This one is not adequate:

> *We predict reservoir quality from seismic using deep learning.*

P.2 What is human-level performance on the task? What level of performance is needed?

It's a good idea for a team to think about performance levels before starting to look for solutions. The purpose of machine learning is usually to add business value, and machine learning follows the usual law of diminishing returns, so it's important to know when the model is good enough (both so you know when you're ahead, and when to stop running!).

'Performance' might mean, 'how many mistakes can you make?' or 'how close does the answer have to be?', or something else. It depends on the task.

'Good enough' can also mean different things. Here are some ways to look at it:

- What is the performance of the current workflow? Don't be surprised if this is difficult to answer. And note that not everyone will be able to give you an unbiased answer.
- What is the performance of the most naive approach that works? There are statistical tools to help answer this question, e.g. in **`sklearn.dummy`**.

- What performance is needed by the 'customer' of the prediction, in order for the prediction to be useful? Often, you can just ask them.

Performance isn't everything. There are other reasons to use computers to solve problems: they can be fast, they can be repeatable, they can emit probabilities, they can provide alternative answers, and they can be embedded in mobile devices. It's possible you will gladly lose some accuracy to gain something else. Just make sure you know what the transaction looks like before you finish the project.

P.3 Is it possible to approach this problem without machine learning?

For the sake of argument, let's say machine learning includes any data-driven decision model with learned parameters (so I'm including linear regression), or with inference from data (e.g. the k-nearest neighbours classifier).

Even with such a generous definition of machine learning, we can often still avoid machine learning. For example:

- People with deep expertise in a specific domain may already have the knowledge you need. For example, a rig operator may already know the threshold pressure of the pumping equipment — no need for that anomaly detection model.
- Some problems in physics are extremely well-understood and can be solved analytically. In other words, there's an equation for it.
- Predicting the future is hard, and machine learning models that try to do this in a robust and ethical way are difficult to train and use. For example, be wary of hiring models in HR — they have been shown to be flawed (e.g. see Raghavan et al. 2019).

Even if we're convinced of the need to learn a model from data, we should continue to apply Occam's razor, so to speak, and prefer simple models over complex ones. With careful application, you can use the machine learning process to discover the one predictive feature that turns a complex model into a simple gradient or threshold. Smart practitioners always want more data (rows) and fewer features (columns).

One of the key questions here is: can the problem be simplified? Start by tackling the simplest version of the problem (e.g. casting it as a binary classification problem, or a one-dimensional regression problem). Similarly, start with simple models. It will be much easier to iterate on the number of features, model type, etc, once the pipeline is established.

P.4 If the analysis focused on deep learning methods, did you try shallow learning methods?

This is nothing more than a logical extension of the previous point. Most 'shallow' models have a couple of learnable parameters per feature. Deep neural networks may have thousands (or millions, or billions) of parameters. This complexity comes with a commensurate increase in the amount of data and time (both data scientist and CPU time) needed to train the model.

For various reasons, many data scientists are drawn to deep learning solutions, and may have skipped the exploration of shallow solutions completely. In some cases, e.g. image segmentation problems, this may be justified.

P.5 What are the ethical and legal aspects of this project?

Some projects have obvious ethical or legal aspects. For example, be wary of any project that uses data about people. You will need to ask questions about identifiability, where the data came from, what those people have agreed to, and so on.

Many subsurface projects will not have obvious issues, but the questions still need to be asked. For example:

- Are you entitled to use the training data and labels? Even open data can have non-commercial licensing, or may be of uncertain provenance (and therefore a risk).
- Are you entitled to use the software you are using? For example, some open software has separate licensing for commercial use.
- What are the consequences of the model making poor decisions? For example, a model predicting formation pressure may raise safety concerns.

Some of the risks might be mitigated with smart source code control, quality checking, and software testing (e.g. integration testing, sanity testing, user testing, and so on).

Remember there are other risks besides legal and ethical ones — for example, what will industry regulators or business partners make of the application?

P.6 Which domain experts were involved in this analysis?

Anecdotally, practitioners agree that machine learning technical tasks in science and engineering must involve domain experts. Even if the task itself is well-formulated, most data science teams do not have the required understanding of the provenance, units, calibration, corrections, uncertainty, and error implicit in subsurface data.

Some domain knowledge can also help solve the problem, for example with feature engineering (e.g. computing VSH from GR, or predicting velocity with Gardner's relationship).

P.7 Which data scientists were involved in this analysis?

Continuing the previous point, we can similarly say that most scientists and engineers do not have the statistical, mathematical or software engineering skills to effectively implement production models.

This is not to say that scientists with some programming skills cannot explore and experiment with a machine learning task. But they will produce better results as part of integrated data science teams with broad skills.

P.8 Which tools or framework did you use? (How much of a known quantity is it?)

There is a core collection of standard tools in machine learning. These include:

- Scikit-learn, TensorFlow, and PyTorch for Python, among many others.
- Caret, dplyr and others for R, among many others.
- Weka for Java, among others.

There is also a universe of tools that build on these. Some of them, for example keras and openai in the Python world, are open source and well-trusted. Others may not be — perhaps because they are new or obscure, or proprietary and closed-source, or perhaps because they are poor. The machine learning practitioners in your organization should regularly discuss tooling, and advise each other on making the best use of what is available. It's important to be able to experiment, but it can be hard to tell what to trust.

P.9 Where is the pipeline published? (E.g. public or internal git repositories.)

Maintaining the code in an open repository (public or corporate) has several advantages:

- Encourages the use of a version control system such as git.
- Facilitates peer-review of code.
- Provides a way to maintain documentation and collect issues (aka tickets) from users.
- Flag some versions as official, supported releases.
- Build automated, so-called 'continuous integration' pipelines.

P.10 How thorough is the testing and documentation?

"Untested code is broken code." Without tests, it's impossible to know for certain that the code works correctly. Once a project moves beyond basic proof of concept, organizations should insist on tests being part of the project.

Likewise, documentation is essential. The level should depend on the degree to which a team or organization depends on the code. Documentation can include:

0. Docstrings (non-negotiable).
1. Usage examples.
2. Basic explanations and advice.
3. In-depth tutorials.
4. Peer-reviewed and citable (e.g. a paper or book).

People sometimes talk about code being 'self-documenting'. This might allow a developer to avoid writing code comments, but clean, readable code — while important — is almost never a substitute for proper documentation.

Questions about the data preparation

D.1 Where did the (a) feature data and (b) labels come from?

What's the provenance of the data used to train the model? This should lead to more questions about reliability, legal issues, other data sources, error and uncertainty, and so on.

Are the labels objective data or subjective interpretations? If the latter, who made them?

In order to be properly reproducible, the provenance of the data, and how another person might acquire the same dataset, should be documented with the project. The data should be subject to change control, just like the code.

D.2 What kind of data exploration did you do?

Most data science practitioners spend some time exploring the data and creating summaries and visualizations, to get a feel for:

- The amount of data.
- Missing data.
- Improper datum or unit conversions.
- The statistical distributions of the features and labels, and presence of outliers.
- Cross-correlation between features.
- Possible issues with error and uncertainty.
- What kinds of models might be appropriate for the task.

Being able to perform basic multivariate analysis also depends on being able to parse all of the data, so it's a way to see that the data ingestion pipeline is functioning as expected.

D.3 What did you do about missing data? E.g. what kind of imputation did you do?

Most datasets have some missing data. Beware of the many ways of expressing missing values, which could include otherwise valid numbers like 0, –1, or –999.25 (missing date-times may be scrambled in yet other ways).

If most of the rows in a large dataset are complete, and if the data instances are independent (often they aren't), then the best thing to do might be to simply remove them.

Other approaches include 'imputing' missing values, e.g. by replacing with zeros, using the feature mean, or with a predictive model (e.g. predicting S-wave velocity from P-wave velocity and density).

D.4 How did you clean the data? How long did this take?

Depending on the data source, and the intended application of the model, it may be necessary to preprocess the data. In subsurface applications, it usually is required. Data cleaning migt involve:

- Removing, clipping or otherwise scaling outliers.

- Reconciling units, or measurements from different tools.
- Correcting for measurement conditions.

Bear in mind that if trianing data must be cleaned to produce an acceptable model, then similar processing may have to be applied to future data before inference (model application).

This part of the workflow can take a substantial amount of time. Auditing where time is spent is a good way to get better at planning machine learning projects.

D.5 What kind of normalization or standardization did you do?

Make sure that any scalers were fit to the training data only, then applied to the validation and test data separately. It is important not to fit the scaler to all the data, or to re-fit it to the validation or test data.

Likewise, however, you should refit the scaler to all of the data when training the final model.

D.6 Are the classes balanced? How did the distribution change your workflow?

In a classification problem, having substantially different numbers of examples (the 'support') in each class can make it difficult to train and evaluate a model.

Here are some ways you can deal with the issue:

- Collect more data, if it's an option.
- Choose correct evaluation metrics (some attempt to handle imbalance, eg F1 vs accuracy).
- Choose models that handle imbalance well, e.g. tree-based models. (Some implementations, like XGBoost, try to implicitly handle imbalance.)
- If using cross-validation, use the stratified k-fold variant.
- Split the abundant classes into groups, and mix with the rare class (which you repeat in each group)
- Under-sample the abundant class(es) -- but this means losing the learning effect from those samples.
- Cluster the abundant class into r clusters (where r is the number of instances in the rare class). Nice idea -- sort of undersampling without losing the information.
- Over-sample the rare class, e.g. by repeating or fuzzing.
- Simulate synthetic samples for the rare class (e.g. with SMOTE).
- Use a custom cost function, or a cost matrix, that penalizes errors on the rare class(es).

D.7 What kind of data augmentation (new rows) or feature engineering (columns) did you do?

It is common to add new features to the training and validation data. (And these features will have to be added to future data before predictions can be made.) Here are some examples:

- Grouping features to ensure there is only one row per 'observation'. E.g. if two analyses were performed on a sample, it is usually preferable to combine them into a single row with multiple features.

- Encoding, especially one-hot encoding of categorical features.
- Linear or nonlinear combinations of existing features.
- Nonlinear transformations of existing features (e.g. log transformation).
- Spatially filtered or shifted versions of existing features (e.g. smoothed well-logs). Be wary of including data from the future if you are performing time-series prediction.
- Analytic attributes of one or more columns.
- Binning noisy features.

Data augmentation is one way to cope with class imbalance (see D.6). Using synthetic data might also be a good way to get a lot of data very cheaply, with the added advantage that the labels are known perfectly in synthetic data.

Augmentation is also very commonly performed on image datasets before deep learning, for which large amounts of data with high variance are usually advantageous. Various techniques can be employed, and are usually applied randomly to images in the training set:

- Flipping or rotating to change the orientation and position of objects.
- Zooming, usually in, to change the sizes of objects.
- Cropping, so that objects move around in the image.
- Shearing, to change the perspective on objects in the image.
- Changing brightness, contrast, and other properties of the histogram.
- Creating completely new instances, e.g. with a forward model or a GAN.

Not all augmentation methods are appropriate to all image sets — ask yourself if the augmented images are plausible.

D.8 How did you split the data into train, validate and test? Or did you use some form of CV?

If you have sufficient data, you should use hold-out sets for model evaluation. If you don't have enough data to spare, cross validation (CV) is a good option.

If you use hold-out sets, you should usually have at least two: one for validation during your model selection workflow, and one for checking the generalization of the model at the end of the workflow. Many standard machine learning workflows omit this important step.

In selecting the hold-out sets, or CV folds, you must ensure that there is no correlation between samples in the training set and samples in the hold-out sets or folds (e.g. samples from nearby points in the earth should not be split across sets). It is very easy to accidentally leak information from the training set to the validation set or *vice versa*. (Also see D.9 and T.5.)

D.9 How do the feature and label distributions compare across datasets or folds?

In selecting the hold-out sets, or CV folds, you must also ensure that the training and hold-out sets or folds have similar distributions with respect to the features, and with respect to the target(s). (Also see D.8 and T.5.)

Questions about training and evaluation

T.1 Which models did you explore and why? Did you also try the simplest models that fit the problem?

Keep a record of the model types you try to fit, with a summary of the outcome. For example, what hyperparameters you used, and the overall quality of the fit. If you reject a model, give a reason, e.g. poor fit, unstable results, counterintuitive results.

For classification problems, make sure to try K-nearest neighbours, a nonlinear support vector machine, and random forest or boosted trees. For regression problems, try ridge regression with interactions and 2nd order polynomials. Resist the temptation to go straight to deep learning.

If your model uses a non-standard cost function, be sure to document this. If it uses a custom cost function, you also need to document how it was implemented.

T.2 How did you tune the hyperparameters of the model? Did you try grid search or other methods?

A model's hyperparameters affect the structure and learning process of the model, e.g. the depth of a decision tree, or the amount of regularization to use in a support vector machine. There are various approaches to tuning these values:

- Grid search: exhaustive search of the parameter space, e.g. with `sklearn.model_selection.GridSearchCV`.
- Random search: sampling the parameter space is usually more efficient, e.g. with `sklearn.model_selection.RandomizedSearchCV`.
- Smart search: for example, searching with genetic algorithms, as in TPOT.
- Optimization: using a model to optimize the search is also an option, e.g. with `scikit-optimize`.

T.3 How much regularization was applied, and of what kind?

Regularization is a set of methods to train 'smoother' models that generalize better, thereby reducing the tendency to overfit. There are many approaches, one common one being to add a parameter penalty, such as the norm of the learned parameters, to the cost function.

Many models implement some kind of regularization by default — read the documentation! Vary the amount or type of regularization and observe the effect, especially on the model variance (the difference between the training score and the validation score).

T.4 Did you do any dimensionality reduction, or check feature importances?

The curse of dimensionality affects all models. High-dimensional features are hard to visualize, hard to interpret, slow to train on, probably contain colinearity and redundancy, and require large amounts of data to fully sample.

It is therefore sensible to look for ways to reduce the dimensionality of a task, e.g. by selection, or with feature engineering (e.g. SIFT for images), or with dimensionality reduction (e.g. principal components).

Feature importance is available in some models (e.g. tree-based methods, or linear regression) and can provide insight into the usefulness of each feature. There are more tools in `sklearn.feature_selection`.

T.5 What kind of validation did you do? Did you use cross-validation?

The standard approach is to use validation and testing hold-out datasets. Cross-validation is one way to cope with not being able to hold out data, and is also a way to explore model variance.

See also D.8 and D.9.

T.6 What evaluation metric are you using? Why is it the most appropriate one? What is its distribution?

If you are using a non-standard evaluation metric, you should document this. If you are using a custom evaluation metric, it must be carefully documented.

If you're using a non-standard or custom metric, and the model is using a substantially different cost function for optimization, the argument for this should be documented.

For most tasks it is useful to report multiple evaluation metrics. For example, you might give precision, recall and F1 for a classifier, or R^2, RMSE, and standard error for a regressor. (See also T.11.)

The distribution of the evaluation metric should be considered. Have you selected a train–test split with anomalously good validation metrics?

T.7 How do training, validation, and test metrics compare?

It may be useful to show how training and validation metrics vary with model complexity, or across your hyperparameter space, or during training iterations (e.g. neural network epochs). The relationship between the two scores can indicate whether the model is underfit or overfit.

If and only if you have reached the end of the workflow, you may also provide the test evaluation. This should be similar to the validation metric. If it is better, you should check that the datasets have comparable feature and target distributions. If it worse, but there was no indication of overfitting, you should look for signs of leakage (from validation to training) or consider the possibility of overfitting to the validation data. In either case, you should consider starting the model selection again with new training, validation and test data.

T.8 How does a dummy model score on this task?

Dummy models, such as those in the `sklearn.dummy` module, are a good way to quantify the minimum useful score for an estimator, which helps to calibrate expectations and set goals for training a model.

T.9 What is the state of the art, or human-level performance, on this task?

How are people currently solving this task? For example, are they using numerical or analytical methods, or human judgment and interpretation? What are the objective mesures of quality of those methods? Answering these questions will help set goals for training a model.

T.10 If this was a classification task, are probabilities available in your model and did you use them?

Most classifiers can produce the probability of each instance being a member of a given class. You may be able to use these probabilities to find problematic classes, or noisy instances, or simply to make a more interesting data visualization.

T.11 If this was a regression task, have you checked the residuals for normality and homoscedasticity?

As well as reporting the validation metric(s) for a regression It is essential to report some basic properties of the residuals. In particular:

- You want the distribution to be approximately normal; there are statistical tests for this.
- You do not want the residuals to be correlated with the features or the target.
- You do not want the variance of the residuals to be correlated with the features or the target.

T.12 How explainable is your model? That is, do the learned parameters mean anything?

Most users of machine learning models would like the models to be understandable (this property is sometimes called 'interpretability' or 'explainability', and people talk about 'explainable AI', or XAI).

You can help the users of your model by looking at it from this perspective. How can you make the model more explainable?

T.13 Are there benchmarks for this task, and how well does your model do on them?

Some tasks, such as the classification of handwritten digits, have well-known benchmarks like the MNIST dataset. Most tasks, however, do not. You should research whether there is a benchmark for the task you are performing.

If there isn't one, could you publish one? It might be useful for others in your field who would like to improve upon your model.

Next steps for the project

N.1 How will you deploy the model?

If the model is almost ready for application in the real world, then deployment is the next challenge. But really it's never too early to think about how the model can be usefully deployed. In other words, how are the people that need it going to use it?

For example: will it be baked into an existing software application? Or will the team build a web or mobile application around it? Or is it part of a hardware project? Answers to these questions will affect various things from model choice to updating the model in future.

N.2 How will you improve the model?

If the model is not ready for deployment, then the team will still be focused on improving it. They will probably have plenty of ideas, but make sure they are not missing obvious things:

- Get more or better data.
- Improve data preprocessing, e.g. by dealing with missing and erroneous values.
- Try to simplify the model with feature selection.
- Improve the feature engineering.
- Try stacking, averaging, or otherwise combining models.

Bear in mind the inevitably diminshing returns.

Models are rarely 'finished'. Most of the time, models are not pushed into production and then never revisited. They are more like software — they can be improved over time. The updates might be regular, even continuous, or more sporadic. But the chances are high that the model can be improved in the future by training on more or cleaner data, using updated software, applying smarter feature engineering, or by improving some other part of the machine learning pipeline.

The only way to know which changes might be required is to pay attention to various things:

- Regularly, or even continuously, monitor the performance of the model on its task.
- Ask users of the model's results for feedback, and provide ways for them to give it.
- Know which versions of libraries projects use so that 3rd party updates can be monitored.

N.3 Would collecting more data help? Can we address any imbalance with more data?

Collecting more data almost always helps a model generalize. It might perform less well in validation, as the variance in the data goes up, but if some of that variance is learnable the model will perform better overall.

Class imbalance should always be addressed (see D.5), but it may not always be practical to address it with more data.

N.4 Are there human or computing resources you need access to?

The team may need some resources or help to do the best possible job; it makes sense to ask what could help. For example, some deep neural networks are prohibitively time-consuming to train without access to accelerated computing hardware, such as graphics cards or cloud computing resources. If training seems unstable and there is a lot of variance in the predictions, they may benefit from the advice of a statistician. If they have written a lot of code and it's becoming hard to maintain, a software engineer might be able to help refactor it.

Changelog

v0.4, May 2021

- Added commentary pages.
- Added the question numbers for easier reference.
- Added 'testing' to question P.10 about documentation.
- Pointed ageo.co/checklist shortlink at this document.
- Changed 'the' to 'any' to question N.2 about data and class imbalance.
- Switched positions of N.1 and N.4.
- Combined features and labels into D.1.
- Changed the order of D.3 to D.6.
- Added D.9 about data distributions.
- Added T.3 (regularization) and T.4 (feature selection).
- Removed a redundant question about residuals.
- Broadened T.8 about dummy models.
- Removed the part about CV from T.5 (it's covered by D.8).
- Changed T.9 to a question about state-of-the-art, because it was a repeat.

v0.3, July 2020

- Changed 'normalization' to 'normalization or standardization'.
- Added 'some form of CV' to validation options.

v0.2, April 2019

- Split 'feature engineering' into 'data augmentation (rows)' and 'feature engineering (columns)'.

v0.1, March 2019

- Added question about dummy classifier.

v0.0, March 2019

- Initial content.

Contributors

- Matt Hall (first draft and subsequent updates).
- Anton Biryukov (bolding some questions and emphasizing validation).
- Justin Gosses (point about ethics).
- Lukas Mosser (points about ensemble models and validation).
- Behzad Alaei (emphasizing data splitting).