Standardizing the ML Experimentation Process

Why You Need a Standard Approach to Achieve Reproducibility in ML and How to Get There

It's Time to Standardize Machine Learning Experimentation

This eBook is focused on defining the process and requirements for developing production-ready machine learning models. These recommendations are based on the insights from working with enterprise customers of every shape and size, and from our peers in the industry.

THE CACE PRINCIPAL: WHY YOU NEED A STANDARD PROCESS FOR RELIABLE MACHINE LEARNING

As we all know, machine learning is used to address problems that cannot be well specified programmatically. Traditional software engineering allows strong abstraction boundaries between different components of a system in order to isolate the effects of changes[1]. Machine Learning systems on the other hand, are entangled with a host of upstream dependencies, such as the size of the dataset, the distribution of features within the dataset, data scaling and splitting techniques, the type of optimizer being used, etc.

As a result, machine learning is highly susceptible to the CACE principle[1]:

The CACE Principle

Changing Anything Changes Everything. In other words, a change anywhere in the machine learning process - especially those furthest upstream - will have an impact on your experiment and results.

Machine Learning is inherently an experimental and iterative science that requires the diligent tracking of multiple sources of variability. It is well known that reproducibility is an issue in many machine learning papers, and while steps are being taken to address these issues, as humans, we are often prone to oversight.

WHY DOES THIS MATTER?

Imagine a team of ten data scientists, each working on a single modeling task with a target metric of accuracy, however:

- **Three researchers label the target metric 'val_acc'**
- **Three label it 'accuracy'**
- **Four label it 'score'**

How would you compare the results across experiments? It may be feasible to work this way for a week, but what about in six months?

These seemingly innocuous problems become real headaches down the road, and can result in tremendous amounts of time lost simply trying to retrace steps.

Because ML systems lack a clear specification, because data collection is an imperfect science, and because effective machine learning models can be incredibly complex, experimentation is necessary.

The goal of the experimentation process is to understand how incremental changes affect the system. Rapid experimentation over different model types, data transformations, feature engineering choices, and optimization methods allows us to discern an image of what is and isn't working.

Andrew Ng's team at Landing AI suggest using 1 Day Sprints when developing ML Systems[3]. These Sprints are organized in the following way:

1. **Build models and write code each day**
2. **Set up training and run experiments overnight**
3. **Analyze results in the morning**
4. **Repeat**

Given the pace of these iterations, the need to automate the logging process starts to become more clear. Automated logging is less prone to clerical error, such as mistakenly writing down the wrong random seed or the wrong CUDA version, and serves as a reliable source of truth further down the line. Debugging models in production is much easier when we have a reliable traceback of the training process that created them.

Adoption of an experimental paradigm involves two key behaviors. Behaviors that are critical to delivering consistent, repeatable, and production-ready machine learning models:

- **Keeping track of all experimental metadata—code, metrics, hyperparameters, datasets, samples, predictions, visualizations, models, etc.,**
- **Logging everything to a central repository in a systemized way that enables comparative analysis.**

Standardizing Experimentation

Keeping track of experimental metadata is simply the starting point for adopting a consistent and repeatable experiment process. There are other critical steps that, when adopted, enable data scientists and data science teams to achieve this consistency as part of the process, rather than an afterthought. These steps include:

- **Defining and Documenting the scope and success criteria of your project**
- **Creating dumb baselines**
- **Understanding and Validating data**
- **Data pre-processing & feature engineering**
- **Iterating over different types of models**
- **Hyperparameter search & fine tuning**
- **Serving the model**
- **Monitoring model performance in the wild**

STEP 1: DEFINE AND DOCUMENT THE SCOPE AND SUCCESS CRITERIA FOR YOUR PROJECT

This is the most crucial step in the process. Before writing any code or looking at any data, you must be able to clearly state and quantify the scope of success for your project.

Here are some of the types of questions you might want to answer at this stage of the process:

- **What business objective is your model trying to achieve?**
- **What are your KPIs for this task?**
- **What is the specific task that you are trying to automate with Machine Learning?**
- **How will your users interact with this model?**
- **What is the input to this system?**
- **What is the expected output?**
- **How will you measure that your model is actually working?**
- **How do we know if we can trust these predictions?**
- **How important is it for the model to be interpretable?**
- **Where will this model be deployed? On-premises? Cloud? Mobile?**
- **What information will be exposed to the user?**
- **How much variance in predicted output can be tolerated?**
- **What are the proxy metrics will we use to evaluate the system's performance?**
- **What are the latency requirements for the predictions?**
- **Is there a risk for our model or data to be biased? If so, how can we detect this?**
- **What are the proxy metrics will we use to evaluate the system's performance?**
- **What are the latency requirements for the predictions?**
- **Is there a risk for our model or data to be biased? If so, how can we detect this?**

Answering these questions will involve talking to the various stakeholders for your project and developing a shared understanding of the objective of your model, and the evaluation process for the model.

Pro Tip: Preparing Machine Learning for Production

One critical consideration and set of parameters that must be documented are the restrictions your model will have if and when it goes to production. These restrictions can be CPU, storage, etc. Defining and documenting these restrictions, then building your model with that context in mind, is a major factor in if your experiment will be successful in the wild.

If these considerations are not taken into account, you might end up with an excellent model that can't be deployed anywhere.

STEP 2: SET UP DUMB BASELINES

Once you have your task and evaluation metrics defined, it may be tempting to hit the ground running and start trying different models on the data.

This is a recipe for endless suffering[3]. Yes, that's right. Endless suffering.

The model is a small part of a much larger system, and is often the easiest part to update. Before creating a model, we must ensure that our evaluation framework is sound and behaves in the way that we expect it to.

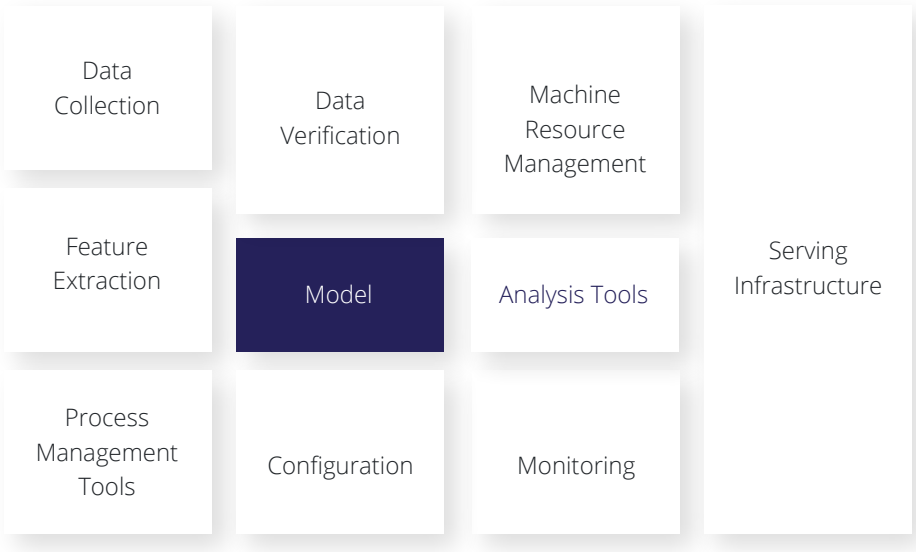

Figure 1. A schematic of a typical machine learning pipeline
(Source: https://developers.google.com/machine-learning/testing-debugging/pipeline/overview)

We can think of this phase of the process as developing the scaffolding for our entire training and evaluation pipeline. The scaffolding should allow us to validate our hypotheses quickly, identify blind spots in the metrics we initially selected in Step 1, and come up with reasonable points of comparisons for our future models.

In this phase, we need to run multiple experiments that are computationally cheap to evaluate and do not take much time.

Some of the types of experiments that you can run in this phase include:

- **Simulating the model predictions with a heuristic**
- **Simulating the model predictions using random sampling**
- **Using an Oracle Model (a mock of the model that makes perfect predictions)**

These experiments will help us understand how our model's predictions affect the KPI that we care about. The Oracle approach in particular, will help clarify whether a model can improve your system at all. You may find that even a perfect set of predictions does not affect your KPI in any meaningful way.

Some other questions to consider at this stage are:

- **What do the metrics look like when using a heuristic?**
- **How close does the heuristic get us to our KPI? Is the gap big enough that we can justify using machine learning to make up the difference?**
- **How close must our model performance be to the Oracle performance in order to positively affect our KPI?**
- **If the Oracle model is unable to affect our KPI, do we need to change our approach? Do we need to rethink the target variable that we are trying to predict?**
- **Do we need to change how our model interacts with the rest of the system or with our users?**

Answers to these questions will give us an indicator for the kind of performance that we are expecting from our model.

Why does this matter?

Suppose our heuristic is the ability to accurately predict our target value 65% of the time, which produces a 2% increase in our KPI. The predictions from our Oracle are 100% accurate and provide a 10% increase in our KPI.

We now know that increasing model accuracy positively affects our KPI. We also know that we need a model that is accurate more than 65% of the time in order to justify putting this system into production.

In this phase of development, we are verifying whether an ML solution is necessary, and establishing trust in our evaluation framework. We want to make sure that optimizing our proxy metrics positively affects the KPIs we care about.

If possible, deploy a proxy for your model, e.g. a heuristic-based approach, on production data in shadow mode (i.e. the predictions are not actually used). This is a way to characterize the types of errors that your simple model will make, which can inform further model improvements [6].

This is also an opportunity to monitor the type of data you are expecting your model to consume. Understanding how this data changes over time will give an idea of how often this model will have to be retrained.

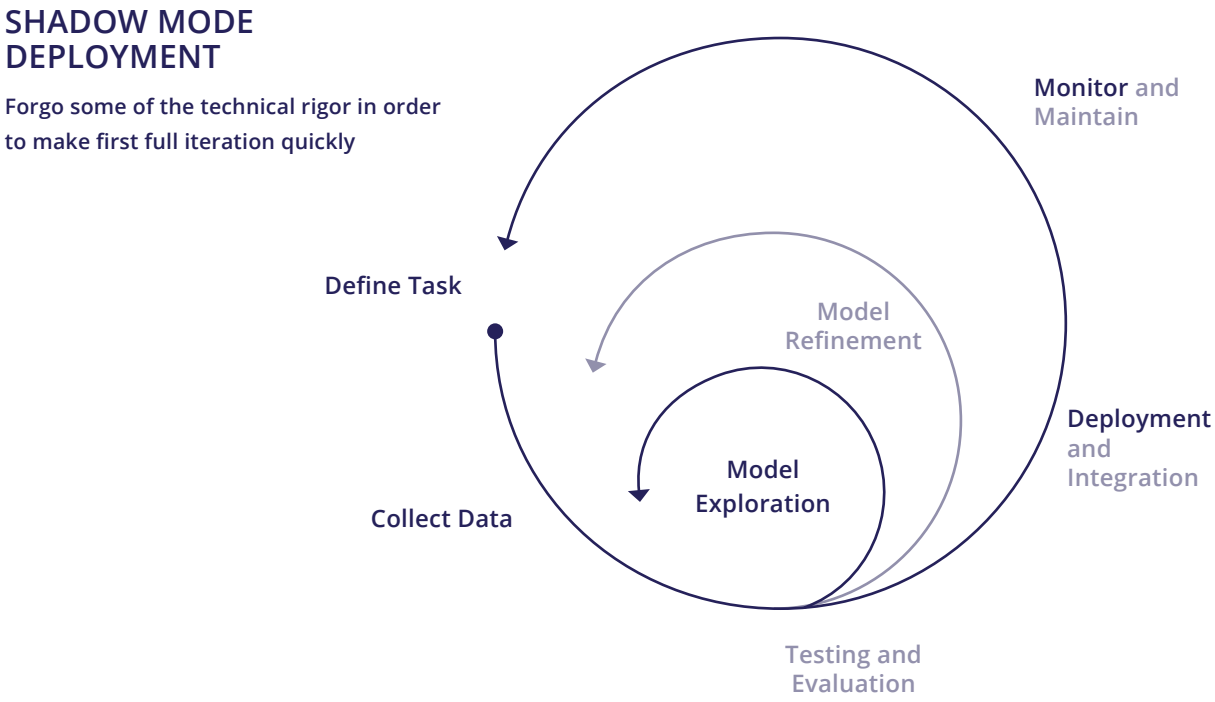

Figure 2. Using shadow deployments for further model understanding
(Source: https://www.jeremyjordan.me/ml-requirements/)

STEP 3: UNDERSTAND AND VALIDATE YOUR DATA

This step assumes that you already have data available for your model.

During this phase of the development process, it is important to familiarize yourself with the data. It is well worth the time to scan through a hundred individual examples to identify outliers, corrupted samples, and other irregularities that may not be easily detectable through statistical tests.

The types of information you might want to extract from these experiments are:

- **The number of examples in the dataset**
- **The number of features in the unprocessed dataset**
- **Statistics about the initial features in your data**
- **The number of categorical features**
- **The number of numerical features**
- **Identify missing data**

- **Identify incorrectly labelled data**
- **If it is a supervised learning problem, profile your target distribution.**
  - **In the case of a regression problem, log summary statistics of the target distribution.**
  - **For a classification problem look at the frequency distribution over the number of classes to check for imbalances**
- **Check if your target distribution needs to be transformed (log scaling, min-max scaling, etc)**
- **Check if your features need to be scaled**
- **Check for outliers in the target.**
- **Are there special considerations to take when creating splits so that there isn't any data leakage between training and validation sets.**
- **Are there any underlying biases in this data that the model might inadvertently pick up? For example, classify an object as a boat based on the presence of water in the image.**
- **Validate the assumptions about your target value.**
  - **Does it fall within a certain range? If not, identify why that is**
  - **Do the class distributions make sense?**
- **Is the data meeting privacy guidelines? Does it contain personally identifiable information?**
- **What was the process used to collect this data? Are there ethical issues with using this data?**

Once you have performed these experiments on the data, you are ready to build a schema for your dataset. A schema will encode these extracted intuitions about the data so that they can be automatically checked when new data becomes available [2].

Another crucial step is to develop a datasheet for your data in order to address aspects of the data that cannot be conveyed through a statistical summary. A datasheet documents the motivation, composition, collection process, recommended uses, and applicability of the data [6]

Why does this matter?

Let's say a feature in your unprocessed dataset might only contain NaN values, so as part of your feature engineering pipeline, you remove it. A few months later, new data comes in and this feature is no longer filled with NaNs. However, since your pipeline is dropping this feature, you never detect the change.

By testing against a data schema, we provide our system with a way to signal that your pipeline might have to be revisited.

To construct the schema, start with calculating statistics from the training data, and then adjust them as appropriate based on the information extracted from the experiments mentioned above. [2]

Pro Tip: Use Automation Where Possible

We recommend automating the dataset profiling and schema construction steps. Your profiling script should capture and log the relevant metadata and use it to build the schema. We will consider a single execution of this script over the data as an experiment. Make sure your experiments have unique identifiers. It is best to automatically generate an identifier when executing the script, rather than manually creating one for the experiment after a run.

We can now reference this experiment id within the schema and track changes to it using git. Rerunning the profiling script on fresh data should create a new schema that references the new experiment.

For tracking changes to the data pipeline we suggest using tools like DVC or Pachyderm.

STEP 4: DATA PREPROCESSING / FEATURE ENGINEERING

Based on the information gathered from Step 3, we are now ready to get started with preparing the data for our model.

Model performance is tightly coupled to the data and our selection of preprocessing steps will have a huge impact on the final results. There may be numerous transformations that are beneficial to the model, some of which might be known beforehand, while others might only reveal themselves after a few iterations.

For this step, we recommend first splitting all your available data into training, validation, and test sets, and log the indices of the examples in each split. You can check for the quality of your splits against your data schema in order to confirm that each split is representative of your entire dataset.

We also recommend sampling from these splits to create smaller versions of your dataset. Remember, we want to rapidly test as many feature engineering ideas as possible, so we don't want to wait around for our experiments to run over all the available data.

In Step 3, we identified the features that need further processing. In this step, we will develop multiple versions of features for the model using different preprocessing techniques. Depending on the scale of your pipeline, it might be worth investing in a dedicated feature store so that computationally heavy features can be stored and reused across experiments and teams.

The types of experiments at this stage will depend on the task. For example, if we were building a topic model for a large corpus of news articles, we might want to investigate the effect of vocabulary size on the model's ability to separate topics.

To do this, we would run multiple experiments with our feature extraction pipeline to turn the raw text data into numerical features, vary the size of the allowed vocabulary between experiments, and store the results from each experiment run. This would result in multiple sets of features that represent different vocabulary sizes. Structuring feature engineering in this way will allow us to treat each transformed data as a hyperparameter when running modelling experiments.

At this stage of development, we will run multiple experiments that:

- **Perform outlier removal on the target**
- **Perform scaling on the target**
- **Perform scaling on the features**
  - **Min-Max Scaling**
  - **Standard Scaling**
- **Encode categorical variables**
  - **Word2Vec**
  - **GLoVE**
  - **One Hot Encoding**
- **Create new features from existing features**
- **Combine features from other datasets**
- **Resizing images to a fixed size**
- **In the case of images, apply different augmentation techniques, such as flipping, rotating, gray scaling etc.**

STEP 5: ITERATE OVER DIFFERENTS TYPES OF MODELS

1. **A target value for our proxy metric (accuracy, roc score, etc) and KPI**
2. **An idea of the performance of simple heuristic methods on our task**
3. **An idea of any irregularities in the raw dataset or in the computed features**
4. **An idea of any irregularities in the target**
5. **A small representative sample of the train, validation, and test data to develop the model against**
6. **A set of precomputed features created by running different versions of a feature extraction pipeline on the sampled data from Step 5.**

Machine Learning development is iterative, and the results from later stages in the pipeline are informed by decisions in the earlier stages.

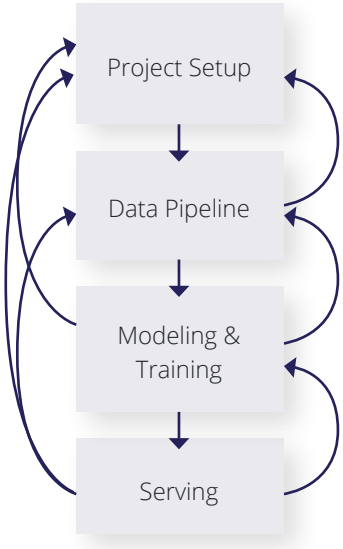

Figure 3. Machine Learning Development Workflow

(Source: https://docs.google.com/presentation/d/1mvmJ1PnCe7lWGmSoL80CjLe7N2QpEwkU8x7l62BawME/edit#slide=id.p4)

Start this process by trying the simplest model for your task, e.g. a Linear or Logistic Regression model. Run an initial set of experiments through this simple model in order to verify that your training and evaluation setup is working as expected. Ensure that you set and log a random seed for your experiments, as this will help with reproducing the results. Through these experiments, we are trying to verify the following:

1. **Is the data being loaded in the correct way for the model to consume?**
2. **Are all the relevant metrics for this modelling task being logged? Do new ones have to be defined?**
3. **Are the splits in the datasets valid? Is there any leakage between the training and validation sets?**
4. **Can the results be reproduced when rerunning the same experiment? Are there any discrepancies caused by incorrectly setting the random seed somewhere?**
5. **Do the label and prediction match up perfectly**

Once we are confident that the training and evaluation framework is trustworthy, we can start slowly ramping up the complexity of the model.

Start trying boosted trees, Gaussian Processes, Neural Nets, etc. Try using off the shelf configurations for these models, and do not use any sort of regularization at this stage. The first set of experiments you should run with a new model type should be to see if the model can overfit the training data.

Let us say that you have sampled 1000 examples from your dataset, which contains a total of 10000 training samples. Run your feature engineering pipeline on these samples and then try to overfit your model on these 1000 samples. Iterate over the different versions of your features to see which ones are most significant to each model.

Compare the training metrics across model types and features to verify which types of models have the capacity to learn from the data at all.

Why does this matter?

Let's take the case of a Neural Network. You may want to try to train it on a single datapoint, or a single batch. Ensure that you are able to get an error close to 0, and check if the prediction and labels align perfectly.[4]

If this isn't the case, there might be something wrong with your model architecture. You can debug this by logging the weights, gradients, and activations in the model, and inspecting whether or not these are changing over the training process.

Once you are able to find a set of models that can overfit the training data, start trying regularization approaches to drive up your validation metrics. For your regularization experiments, you should first investigate the effects of scaling the amount of data on your model. Track how the training and validation metrics for each model type change as more samples are added to the training set.

Some other approaches that can be taken here:

1. **L1 and L2 regularization**
2. **Early Stopping**
3. **Dropout**

Select the best set of model and feature combinations from this initial set of experiments, and confirm once again that they meet the requirements defined in Step 1. For example, you may find that a Neural Network is best suited for your task, but the size of the model is too big to deploy on a mobile application, so you might want to rethink using this model, or consider some sort of pruning strategy.

These initial set of models will now move onto further development. At this stage, we should have an idea of which set of features work best for our task and selected model types.

STEP 6: HYPERPARAMETER SEARCH AND FINE TUNING

At this stage, you should have an idea of the types of models that are able to fit your data, the features that work best for each model type, and the effects of scaling the data on each candidate model type.

We are now ready to start tuning our model hyperparameters. One way to speed up this process is to use the sampled dataset to quickly iterate over different hyperparameters configurations. This is highly dependent on whether you can get a representative sample of your dataset. If the data is complex, this can be much harder.

One workaround is to investigate the effect of how adding more data affects the performance of the top N hyperparameters determined from the sampled data. N is determined based on the compute resources that are available. In order to avoid overfitting to the sampled dataset, we also suggest building multiple validation sets by sampling from the full validation set. We can use the average of the results from each set as an indicator of the performance of the hyperparameters on the entire validation set.

Start with a simple search algorithm like Random Search. In general, Random Search is a hard to beat baseline, although depending on the problem, you may opt to use Bayesian Optimization methods further down the line when you have narrowed down your search space.

An iterative search strategy is usually the most effective way to tune your hyperparameters. Start with a wide range for each hyperparameter, and sweep a coarser search space. Use discrete values for the parameter options, even if your hyperparameter is a continuous variable, such as the learning rate, or momentum, it is much more efficient to narrow down the range of values before trying to search a continuous parameter space.

Once you have a set of hyperparameters that work well with your data, you may choose to further fine tune your approach by trying ensembles using the top performing models, letting your model train for an extended period of time, further optimize the hyperparameters of your feature engineering pipeline, etc.

STEP 7: SERVING

The final step of our experimentation involves deploying the model to the end user. At this stage, you should already have an evaluation framework in place for assessing the models performance, and should have already run some baseline heuristic models through the framework.

Opt to use a canary deployment, or a shadow deployment initially, and monitor your model's performance metrics. Some of the things that you may find at this stage are:

1. **The model needs further refinement in order to work with production data.**
2. **The data is experiencing drift at a much faster rate than initially anticipated, so the training pipeline might have to be retriggered more frequently.**
3. **How does the model handle edge cases, or outliers?**

Achieving Reproducibility

WHAT IS REPRODUCIBILITY?

We define reproducibility to mean recreating the exact results reported by the original author [8]. Previous works have proposed formal criteria to distinguish between terms such as replicability, and robustness, that are often used interchangeably with reproducibility [9, 10].

Whitaker et al. adopt the terminology where reproducible work means to recreate results using the same data and tools. Replicable work means to arrive at similar results with different data, for example, a proposed image classification model achieves the best test accuracy on a baseline dataset as well as other image datasets with similar class distributions. Robust work means to achieve similar results using the same data, but changing aspects of the tools (changing an implementation of a model, or changing the model framework). Generalizable work means to arrive at similar conclusions based on different data and different tools, for example the Adam Optimizer appears to work well across model architectures, and datasets.

Tatman et al. create a taxonomy to quantify the degree to which a project is reproducible [8]. Each type of reproducibility can be characterized by a score of high, medium, or low based on the provided artifacts. We expand on this work and provide a checklist of artifacts that allow us to determine the degree of reproducibility of a machine learning project.

	Data		
Code, Parameters, Environment, Analysis		**Same**	**Different**
	Same	Reproducible	Replicable
	Different	Robust	Generalizable

Table 1. Reproducible Research. Adapted from: https://github.com/WhitakerLab/ReproducibleResearch

	Low	Medium	High
Artifact	✓	✓	✓
Description of the Model	✓	✓	✓
Description of Model Hyperparameters	✓	✓	✓
Description of the Dataset	✓	✓	✓
Description of Preprocessing Steps	✓	✓	✓
Description of Training Steps	✓	✓	✓
Documented Experimentation Methodology	✓	✓	✓
Uncertainty Quantification of Reported Metrics	✓	✓	✓
Random Seed	✗	✓	✓
Source Code	✗	✓	✓
Dataset	✗	✓	✓
Software Environment Information is Provided (Dependencies, OS, etc)	✗	✗	✓
Compute Requirements in CPU/GPU days and Cost are defined	✗	✗	✓
Hardware Requirements are provided (CPU/GPU Type, GPU Memory)	✗	✗	✓

Table 2: Checklist of artifacts that allow us to determine the degree of reproducibility of a machine learning project.

WHY IS IT IMPORTANT?

Reproducibility is the linchpin to all machine learning success. It is a prerequisite when it comes to establishing trust in a model and its predictions. The more people that can run your algorithm and verify the results, the more confident you can be that your methods are sound. More importantly, having others validate your approach brings transparency and credibility to your work.

When other people are able to replicate and understand your work, they can extend it and build new things. Practices such as version control, continuous integration and unit testing allow developers to build upon each other's work, while maintaining visibility into the entire process.

Reproducibility allows us to build better baselines. In order to progress on any task, we need to be able to compare our approach to established methods. If we cannot reproduce the results of existing techniques, then we have no way of knowing whether we are actually moving forward. There are also times when it is not possible to rerun someone else's work due to factors like private datasets, extensive training time, or non-standard computing infrastructure [9]. In these cases, we can only evaluate the work by comparing it to a known reproducible baseline. Unreliable baselines can result in too much time spent on explorations of unpromising or incorrect techniques. This represents wasted resources that could be better put to use elsewhere.

WHY IS IT DIFFICULT?

Machine Learning is inherently non-deterministic. Below, we list a few factors that inject non-determinism into the experimentation process. These factors tend to change over the course of the model development process and their effects are subtle and hard to detect unless properly logged.

1. **Random Seeds used for things such as parameter initialization, data splitting, data shuffling, dropout regularization, etc.**
2. **Changes in Machine Learning frameworks, and default values in source code.**
3. **Non determinism of floating point calculations in GPUs.**
4. **Differences in hardware (CPU type, GPU type)**
5. **Difference in software dependency versions (sklearn 0.19.0 vs 0.21.0)**

TOWARDS REPRODUCIBLE WORKFLOWS

Machine learning is an iterative process involving many variables. Manually tracking the entire process is an unreasonable and error prone burden. Most of the hidden technical debt [1] surrounding machine learning systems involves some form of tracking, whether it be metrics, dataset distributions, or hardware details, and it is in our best interest to automate this process as much as possible.

In this regard, the machine learning community can benefit tremendously from best practices employed in Software Engineering. Rapid iterative and reproducible workflows in software engineering are achieved by leveraging automated tools such as Github, and CI. We should think of our experiments as software engineering, and adopt these best practices that are designed to facilitate collaboration [14]. Use version control with your source code. Make use of pull requests, CI tools and containers, if possible, to run your experiments. Automate the process of tracking your experiment parameters, metrics, models, and compute environment.

The reproducibility crisis in machine learning is well documented and is a serious barrier to future progress [11,12,13,15]. It stifles collaborative experimentation since it prevents anyone from being certain about the claims made, and makes new ideas harder to try out. It also makes it difficult to transition models from development to production, because no one wants to use a model they can't understand or rebuild.

Reproducibility is a kindness to your future self and everyone else who might want to build upon your work [14] -- and that is a great goal all on it's own.

References

1] Sculley, David, et al. "Machine learning: The high interest credit card of technical debt." (2014).

[2] Breck, Eric, et al. "The ml test score: A rubric for ml production readiness and technical debt reduction." 2017 IEEE International Conference on Big Data (Big Data). IEEE, 2017.

[3] Ng, Andrew. "Andrew Ng at Amazon Re:MARS 2019." YouTube, YouTube, 9 Sept. 2019, www.youtube.com/watch?v=j2nGxw8sKYU.

[4] Karpathy, Andrej. A Recipe for Training Neural Networks. karpathy.github.io/2019/04/25/recipe/.

[5] Jordan, Jeremy. Organizing Machine Learning Projects: Project Management Guidelines. 1 Dec. 2020, www.jeremyjordan.me/ml-projects-guide/.

[6] Jordan, J. (2020, July 24). Building machine learning products: A problem well-defined is a problem half-solved. Retrieved January 11, 2021, from https://www.jeremyjordan.me/ml-requirements/

[7] Gebru, Timnit, et al. "Datasheets for datasets." arXiv preprint arXiv:1803.09010 (2018).

[8] Tatman, Rachael, Jake VanderPlas, and Sohier Dane. "A practical taxonomy of reproducibility for machine learning research." (2018).
[9] Pineau, Joelle, et al. "ICLR Reproducibility Challenge 2019." ReScience C 5.2 (2019): 5.

[10] The Turing Way Community, Becky Arnold, Louise Bowler, Sarah Gibson, Patricia Herterich, Rosie Higman, ... Kirstie Whitaker. (2019, March 25). The Turing Way: A Handbook for Reproducible Data Science (Version v0.0.4). Zenodo. http://doi.org/10.5281/zenodo.3233986

[11] Pineau, Joelle, et al. "Improving Reproducibility in Machine Learning Research (A Report from the NeurIPS 2019 Reproducibility Program)." arXiv preprint arXiv:2003.12206 (2020).

[12] Beam, Andrew L., Arjun K. Manrai, and Marzyeh Ghassemi. "Challenges to the reproducibility of machine learning models in health care." Jama 323.4 (2020): 305-306.

[13] Ding, Z., Reddy, A., & Joshi, A. (2020, August 24). Reproducibility. Retrieved January 12, 2021, from https://blog.ml.cmu.edu/2020/08/31/5-reproducibility/

[14] Grus, J. (2019, May). Reproducibility @ ICLR 2019. Retrieved January 12, 2021, from https://docs.google.com/presentation/d/1yHLPvPhUs2KGI5ZWo0sU-PKU3GimAk3iTsI38Z-B5Gw/edit#slide=id.g5921051688_0_68

[15] Heaven, W. (2020, November 12). AI is wrestling with a replication crisis. Retrieved January 12, 2021, from https://www.technologyreview.com/2020/11/12/1011944/artificial-intelligence-replication-crisis-science-big-tech-google-deepmind-facebook-openai/

Comet provides a self-hosted and cloud-based meta machine learning platform allowing data scientists and teams to track, compare, explain and optimize experiments and models.

Backed by thousands of users and multiple Fortune 100 companies, Comet provides insights and data to build better, more accurate AI models while improving productivity, collaboration and visibility across teams.

Learn more at www.comet.ml