Imperial College London

Ethics of ML/DS Part I

Week 6 : safety and security

Safety and security

- **Documentation and Guardrails**
- Testing and Stress-testing
- Versioning and Maintaining
- Monitoring and Retraining

Safe in the hands of others

“A knife is a tool – it is all about how you use it.”

That might be true but:

- A knife is harmful in a very obvious way.
- Even so, the knife maker has to say, for example, “not suitable for kids”.
- More complex products need to inform their user about their risks.
- They also need to minimize the risk to the extent possible.

Who is the user of a model?

End user: the person who is using the service or product enabled by the model.

Developer: the person who developed the model.

Maintainer: the person who is responsible for maintaining the model.

Product owner: the person responsible for the product/service using the model.

In many situations, the developer and product owner will work hand in hand. But:

- ML-as-a-service often involves serving pre-trained ML models behind an API
- A model might be used / repurposed by others.
- Models can be used as baselines or transfer learning starting points.

Interlude: what is an API?

Application Programming Interface. is a term used often in many different settings in software engineering. A **web API** in particular usually refers to a “restful service” over http: i.e., the ability to ask for a prediction from a model by making an http request over the internet. This is a scalable way to use a model without moving the model itself. It also means the developer is not ”in the room”.

Example of word2vec served over an API: *http://bionlp-www.utu.fi/wv_demo/*

The role of documentation

***Documentation** offers information intended to:*

- Allow developers to collaborate efficiently by communicating the intended purpose of each programming module, and its exact way of working.
- Allow users of code assets to use them correctly.

Example: The call *model.fit(., .)* from sklearn needs to receive the feature matrix *X* first, and the target variable *y* second. If, say, X is univariate, doing it the other way around won’t throw an error, but it *will give you the wrong answer.*

Documenting models: model cards

We can take a very expansive interpretation of what it means to "use a model correctly" and create appropriate guardrails to ensure its *ethical* use.

Standardizing this process is beneficial as it:

- Makes it easier for even novice developers to know what to report in the docs
- Makes it easier for users to understand the limitations of a model at a glance
- Incentivizes good practices by making certain checks necessary, not optional

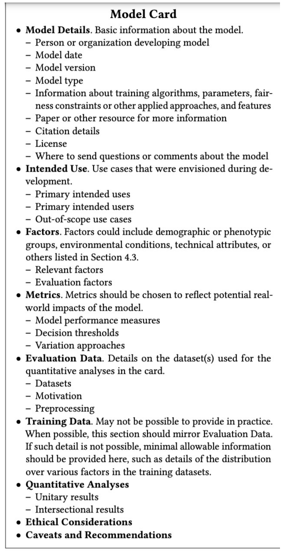
Imperial College
London

Model cards for model reporting

Consider an image classification problem – detect smiling faces (CelebA dataset)

Main findings:

- False discovery rate on older men is much higher than that for other groups.
- Men (in aggregate) have a higher false negative rate
- May be the most useful when detecting presence is more important than its absence
- Additional finetuning, for example, with images of older men, may be needed



Model Card

- **Model Details**. Basic information about the model.
  - Person or organization developing model
  - Model date
  - Model version
  - Model type
  - Information about training algorithms, parameters, fairness constraints or other applied approaches, and features
  - Paper or other resource for more information
  - Citation details
  - License
  - Where to send questions or comments about the model
- **Intended Use**. Use cases that were envisioned during development.
  - Primary intended uses
  - Primary intended users
  - Out-of-scope use cases
- **Factors**. Factors could include demographic or phenotypic groups, environmental conditions, technical attributes, or others listed in Section 4.3.
  - Relevant factors
  - Evaluation factors
- **Metrics**. Metrics should be chosen to reflect potential real-world impacts of the model.
  - Model performance measures
  - Decision thresholds
  - Variation approaches
- **Evaluation Data**. Details on the dataset(s) used for the quantitative analyses in the card.
  - Datasets
  - Motivation
  - Preprocessing
- **Training Data**. May not be possible to provide in practice. When possible, this section should mirror Evaluation Data. If such detail is not possible, minimal allowable information should be provided here, such as details of the distribution over various factors in the training datasets.
- **Quantitative Analyses**
  - Unitary results
  - Intersectional results
- **Ethical Considerations**
- **Caveats and Recommendations**

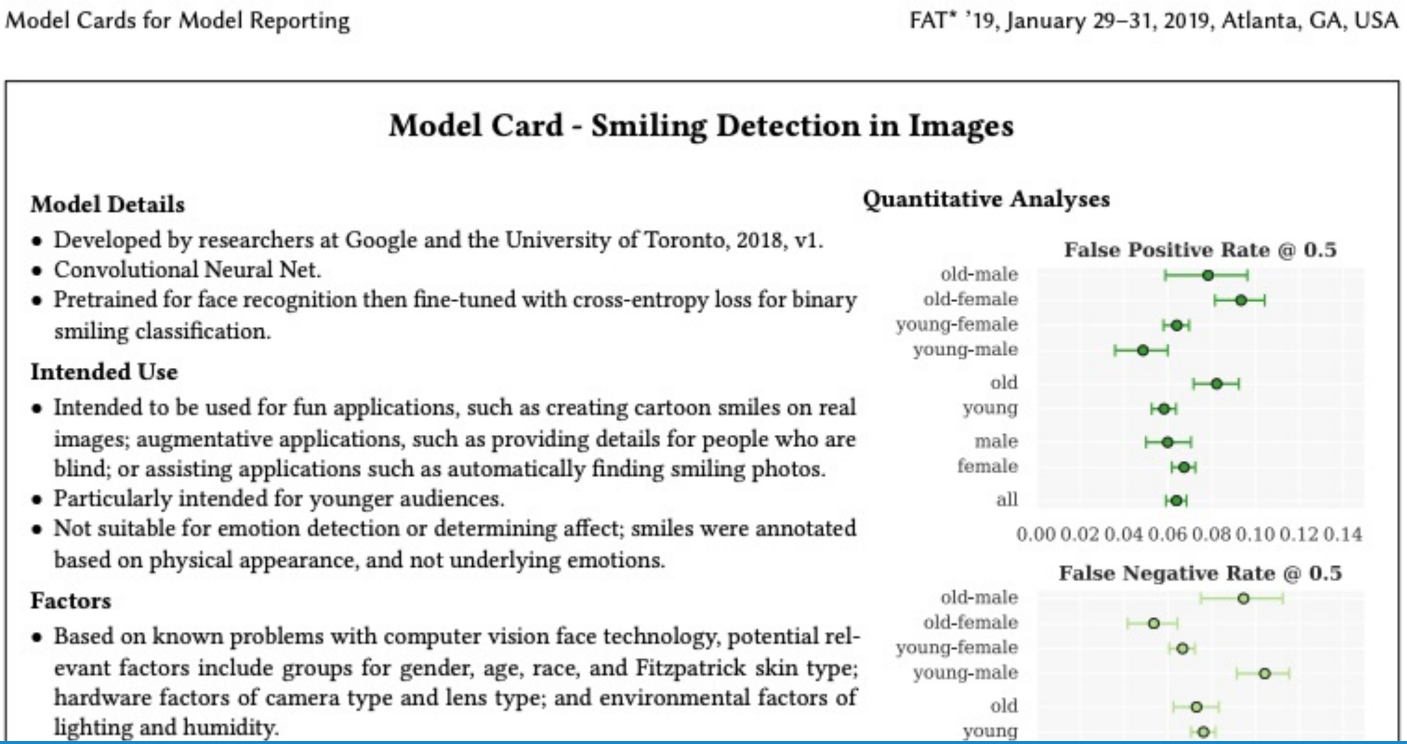
Mitchell, Margaret, et al. "Model cards for model reporting." *Proceedings of the conference on fairness, accountability, and transparency*. 2019. https://arxiv.org/abs/1810.03993

Imperial College London

Model cards for model reporting

- **Model details** (developer/maintainer, version, model type, paper/citation, license)
- **Intended use** (primary intended uses and users, out-of-scope uses)
- **Factors** (demographic groups, technical attributes etc)
- **Metrics** (out of sample performance, decision thresholds, fairness)
- **Evaluation data** (details about the data used for the metric evaluation)
- **Training data** (details about the training data, incl. preprocessing)
- **Ethical considerations, caveats and recommendations**

Mitchell, Margaret, et al. "Model cards for model reporting." *Proceedings of the conference on fairness, accountability, and transparency*. 2019. https://arxiv.org/abs/1810.03993



Model Cards for Model Reporting FAT* '19, January 29–31, 2019, Atlanta, GA, USA

Model Card - Smiling Detection in Images

Model Details

- Developed by researchers at Google and the University of Toronto, 2018, v1.
- Convolutional Neural Net.
- Pretrained for face recognition then fine-tuned with cross-entropy loss for binary smiling classification.

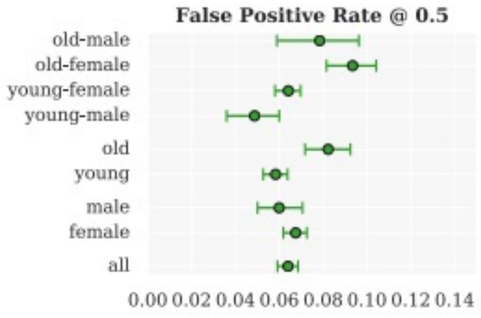
Intended Use

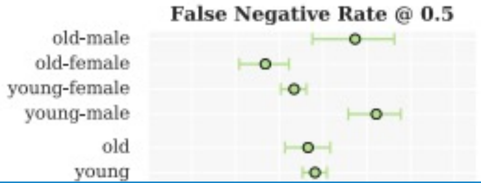
- Intended to be used for fun applications, such as creating cartoon smiles on real images; augmentative applications, such as providing details for people who are blind; or assisting applications such as automatically finding smiling photos.
- Particularly intended for younger audiences.
- Not suitable for emotion detection or determining affect; smiles were annotated based on physical appearance, and not underlying emotions.

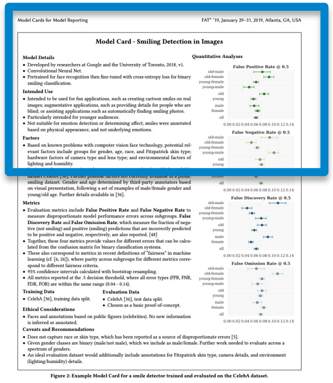
Factors

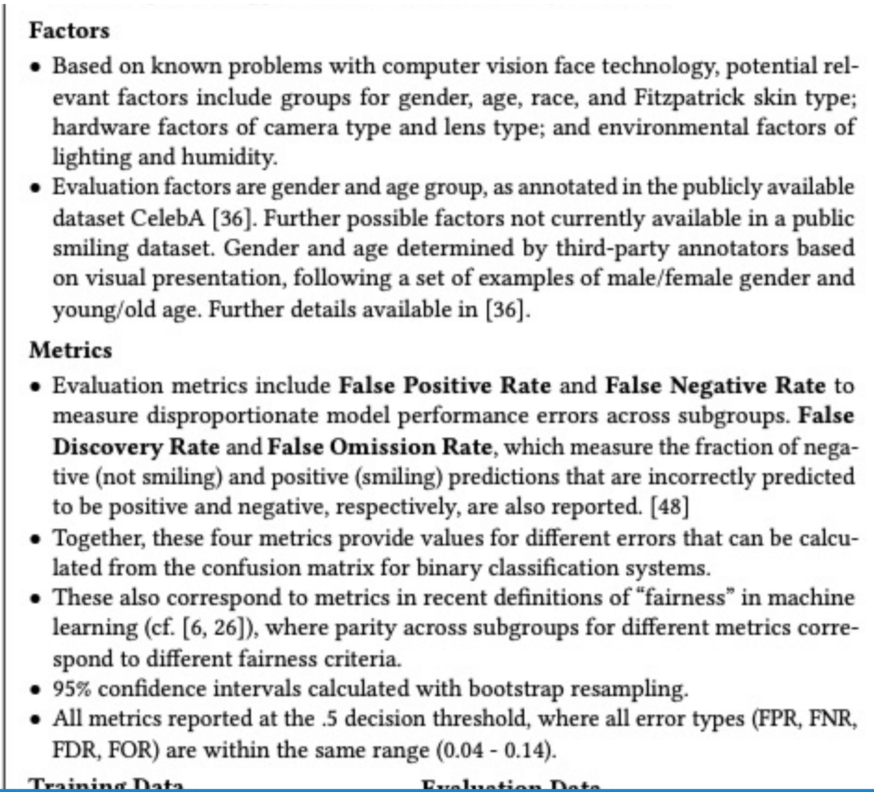
- Based on known problems with computer vision face technology, potential relevant factors include groups for gender, age, race, and Fitzpatrick skin type; hardware factors of camera type and lens type; and environmental factors of lighting and humidity.

Quantitative Analyses









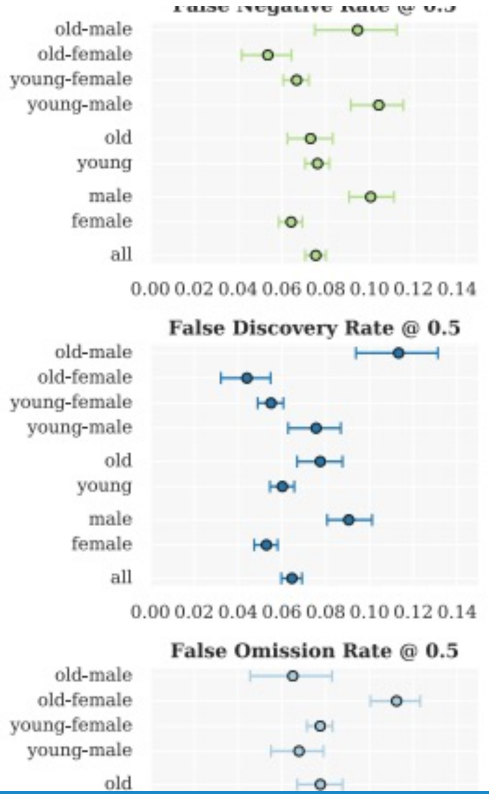
Factors

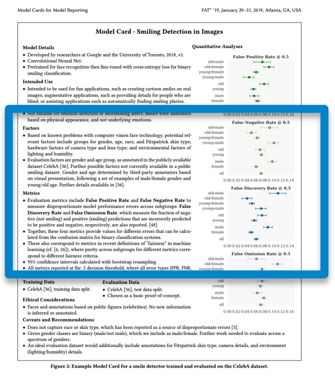
- Based on known problems with computer vision face technology, potential relevant factors include groups for gender, age, race, and Fitzpatrick skin type; hardware factors of camera type and lens type; and environmental factors of lighting and humidity.
- Evaluation factors are gender and age group, as annotated in the publicly available dataset CelebA [36]. Further possible factors not currently available in a public smiling dataset. Gender and age determined by third-party annotators based on visual presentation, following a set of examples of male/female gender and young/old age. Further details available in [36].

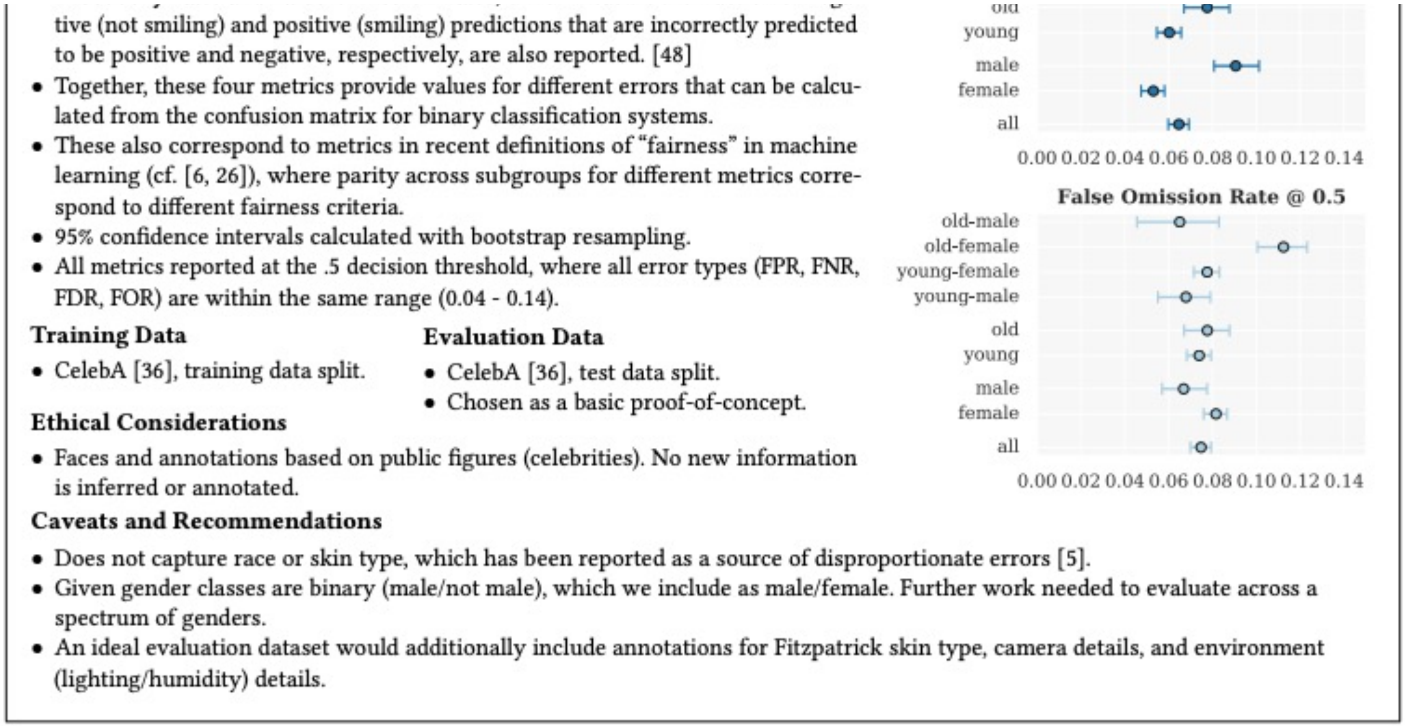
Metrics

- Evaluation metrics include **False Positive Rate** and **False Negative Rate** to measure disproportionate model performance errors across subgroups. **False Discovery Rate** and **False Omission Rate**, which measure the fraction of negative (not smiling) and positive (smiling) predictions that are incorrectly predicted to be positive and negative, respectively, are also reported. [48]
- Together, these four metrics provide values for different errors that can be calculated from the confusion matrix for binary classification systems.
- These also correspond to metrics in recent definitions of "fairness" in machine learning (cf. [6, 26]), where parity across subgroups for different metrics correspond to different fairness criteria.
- 95% confidence intervals calculated with bootstrap resampling.
- All metrics reported at the .5 decision threshold, where all error types (FPR, FNR, FDR, FOR) are within the same range (0.04 - 0.14).

Training Data **Evaluation Data**







tive (not smiling) and positive (smiling) predictions that are incorrectly predicted to be positive and negative, respectively, are also reported. [48]

- Together, these four metrics provide values for different errors that can be calculated from the confusion matrix for binary classification systems.
- These also correspond to metrics in recent definitions of "fairness" in machine learning (cf. [6, 26]), where parity across subgroups for different metrics correspond to different fairness criteria.
- 95% confidence intervals calculated with bootstrap resampling.
- All metrics reported at the .5 decision threshold, where all error types (FPR, FNR, FDR, FOR) are within the same range (0.04 - 0.14).

Training Data

- CelebA [36], training data split.

Evaluation Data

- CelebA [36], test data split.
- Chosen as a basic proof-of-concept.

Ethical Considerations

- Faces and annotations based on public figures (celebrities). No new information is inferred or annotated.

Caveats and Recommendations

- Does not capture race or skin type, which has been reported as a source of disproportionate errors [5].
- Given gender classes are binary (male/not male), which we include as male/female. Further work needed to evaluate across a spectrum of genders.
- An ideal evaluation dataset would additionally include annotations for Fitzpatrick skin type, camera details, and environment (lighting/humidity) details.

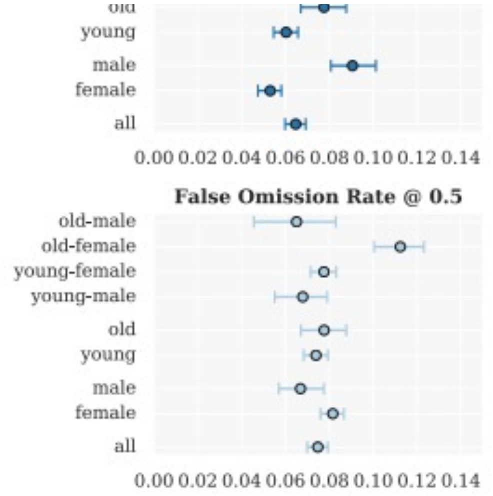
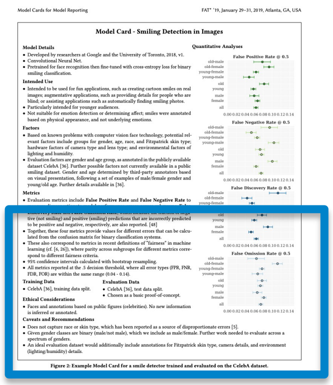


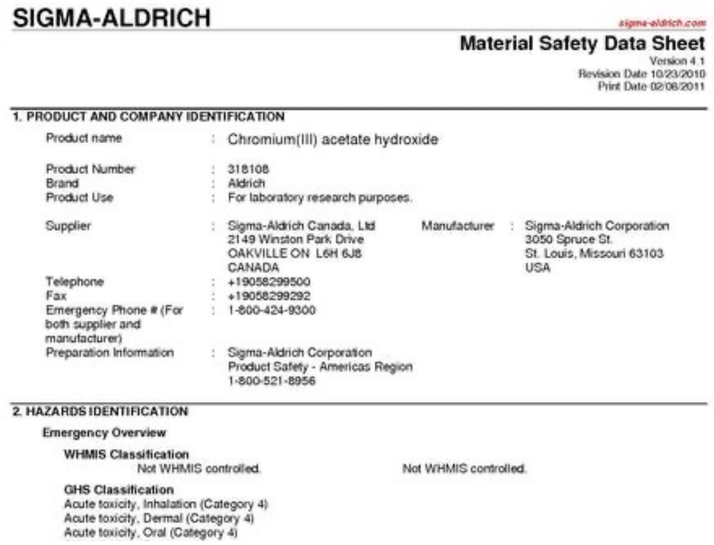
Figure 2: Example Model Card for a smile detector trained and evaluated on the CelebA dataset.



Imperial College London

Safety sheets are commonplace in other industries

- More than documentation, model cards and ”datasheets” capture limitations and results from safety analysis in a concise way.
- Similar cards can be produced for each dataset released in the wild, to avoid issues.



SIGMA-ALDRICH

sigma-aldrich.com

Material Safety Data Sheet

Version 4.1
Revision Date 10/23/2010
Print Date 02/08/2011

1. PRODUCT AND COMPANY IDENTIFICATION

Product name	Chromium(III) acetate hydroxide		
Product Number	318108		
Brand	Aldrich		
Product Use	For laboratory research purposes.		
Supplier	Sigma-Aldrich Canada, Ltd 2149 Winston Park Drive OAKVILLE ON L6H 6J8 CANADA	Manufacturer	Sigma-Aldrich Corporation 3050 Spruce St. St. Louis, Missouri 63103 USA
Telephone	+19058299500		
Fax	+19058299292		
Emergency Phone # (For both supplier and manufacturer)	1-800-424-9300		
Preparation Information	Sigma-Aldrich Corporation Product Safety - Americas Region 1-800-521-8956		

2. HAZARDS IDENTIFICATION

Emergency Overview

WHMIS Classification
Not WHMIS controlled. Not WHMIS controlled.

GHS Classification
Acute toxicity, Inhalation (Category 4)
Acute toxicity, Dermal (Category 4)
Acute toxicity, Oral (Category 4)

Sigma Aldrich, CC BY-SA 3.0 <https://creativecommons.org/licenses/by-sa/3.0>, via Wikimedia Commons

Summary

- **Safety** is about safety in the hands of others.
- These “others” might include model co-developers, product owners or developers of services that use the model, other teams that might make use of the model for comparison or to build upon in an open-source setting, etc.
- **Documentation** is a way to focus the attention of users on what matters, at different levels of detail depending on what is being documented.
- **At the top level,** standardized model cards are a good way to instill good practices and ensure that imperfect models can still be used safely.

Safety and security

- Documentation and Guardrails
- **Testing and Stress-testing**
- Versioning and Maintaining
- Monitoring and Retraining

Verification and validation (V&V)

Verification

Did we build the thing right?

Validation

Did we build the right thing?

Verification and validation (V&V)

Verification

Did we build the thing right?

Validation

Did we build the right thing?

- In consumer products, user acceptance criteria are used to **validate** the product.
- In technical products (e.g., in modelling and simulation), **validation** is often done by comparing the output of a simulator to the real world.
- In ML, **validation** is typically out-of-sample accuracy, but can go beyond that (e.g., response times, fairness and privacy guarantees, etc.)

Imperial College London

Verification and validation (V&V)

Verification

Did we build the thing right?

Validation

Did we build the right thing?

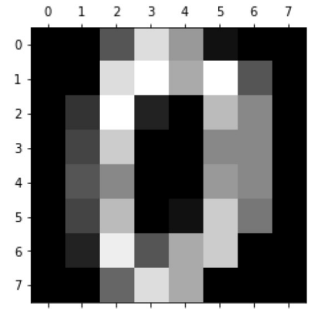
- **Verification** ensures that the product is capturing the intention of the developer.
- For example, the data engineering code that translates raw data into feature vectors must be carefully verified to ensure that the features are capturing what was intended by the data science team. **If not, no-one might notice!**

Example: verifying and validating a choice of feature

- Consider you are feature engineering on the classic “handwritten digits” dataset.
- Each example is an 8x8 image. You are extracting scalar features from these images so that each example is a feature vector.
- One of your features is meant to capture the maximum value of each column in the image.
- You write this code, and it seems right:

```
In [15]: pd.DataFrame(digits.images[0]).apply(np.max,1)
```

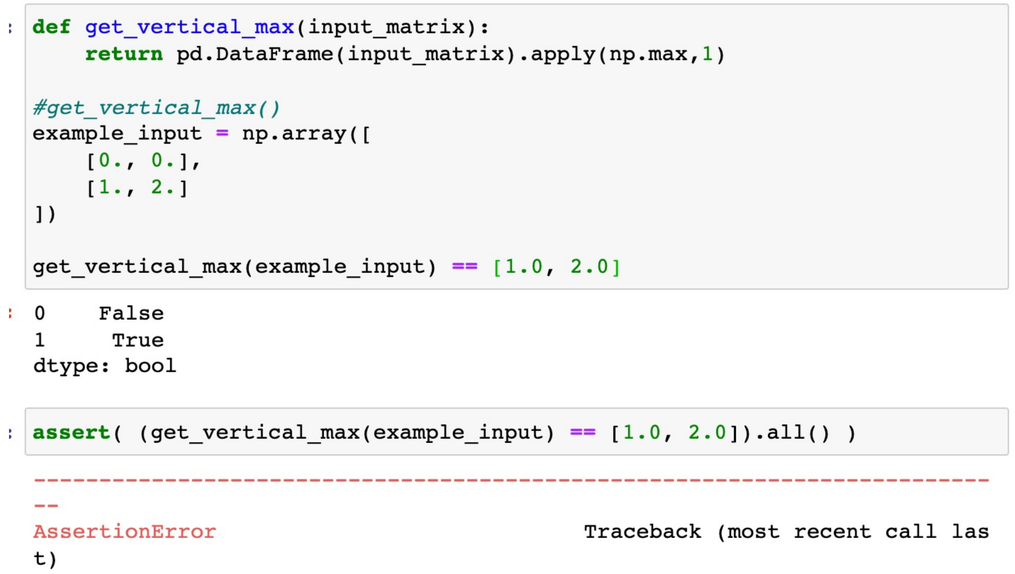
```
In [10]: from sklearn.datasets import load_digits
         digits = load_digits()
         print(digits.data.shape)
         (1797, 64)
         import matplotlib.pyplot as plt
         plt.gray()
         plt.matshow(digits.images[0])
         plt.show()

         (1797, 64)

         <Figure size 432x288 with 0 Axes>
```



Example: verifying and validating a choice of feature

- Turn it into a function
- Come up with a single, *non-trivial* example that tests its output.
- Convert it into an “assertion” (i.e., something that throws an error if it fails, and does nothing if it passes).

- You notice your test fails.



```python
def get_vertical_max(input_matrix):
    return pd.DataFrame(input_matrix).apply(np.max,1)

#get_vertical_max()
example_input = np.array([
    [0., 0.],
    [1., 2.]
])

get_vertical_max(example_input) == [1.0, 2.0]
```

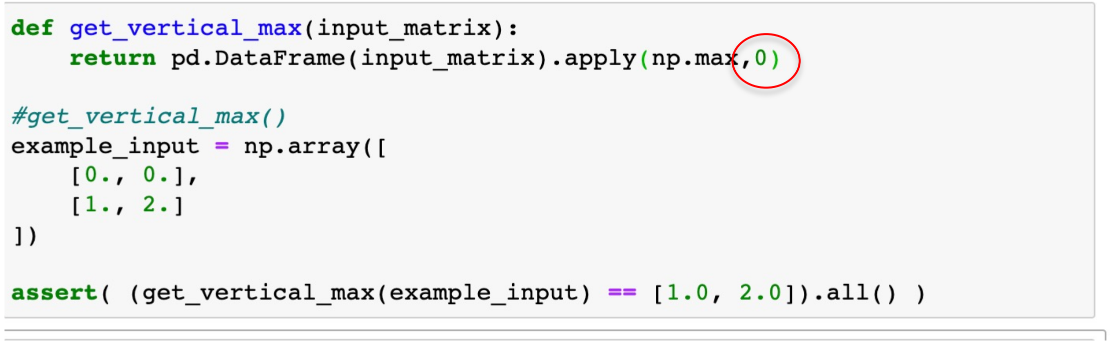
```
0    False
1     True
dtype: bool
```

```python
assert( (get_vertical_max(example_input) == [1.0, 2.0]).all() )
```

```
-------------------------------------------------------------------------
--
AssertionError                            Traceback (most recent call las
t)
```

Example: verifying and validating a choice of feature

- You change the index from 1 to 0.
- The assertion now passes.
- You have peace of mind. Moreover, if you later change the function (e.g., to make it more complex), tests continue to ensure you didn't break anything in the process.



```
def get_vertical_max(input_matrix):
    return pd.DataFrame(input_matrix).apply(np.max,0)

#get_vertical_max()
example_input = np.array([
    [0., 0.],
    [1., 2.]
])

assert( (get_vertical_max(example_input) == [1.0, 2.0]).all() )
```

- **Validating** the choice of feature can then involve checking whether its inclusion adds predictive accuracy, or whether a human expert agrees that the intended logic is worth including.
- For example, a feature built with mistaken logic might be rejected by the model because of the error in the logic, which might lead to the mistaken conclusion it was useless).

“Why test? I have checked my code.”

- Testing is something we all do anyway – but non-engineers do it in the command line / notebook. Testing automates this and keeps rerunning these “mental checks” every time you change your code.
- How do you pick examples that cover all the edge cases? It is a bit of an art, but there are guidelines, such as always thinking about boundaries, using random numbers, and keeping your functions small.
- Unit tests are also a great contribution to documentation: nothing like an example to help a description.

Experienced developers start by writing the tests, and then the function!

1. Simple assertions
2. Unit testing framework, triggered manually
3. Continuous integration framework, triggers tests every time code is changed*

* to be precise, every time code is *pushed to the development branch,* i.e., whenever the code changes significantly and is stable

Tests are also a (great) form of documentation

- There is nothing like an example to help a description.
- There are frameworks that unify documentation and testing

The module *docstring* in Python is a convenient way to simultaneously test and document in an elegant way.

```python
def square(x):
    """Return the square of x.

    >>> square(2)
    4
    >>> square(-2)
    4
    """

    return x * x
```

Unit testing is just the beginning

- Unit tests
- Feature tests
- Integration tests
- Performance tests
- Stress (load) tests
- Penetration tests

Summary

- Data science has matured out of the realm of “experimentation” and into that of production software
- Proper testing is therefore needed to ensure the intention of the developer was correctly captured.
- Unit testing lies at the basis of that pyramid, is easy to incorporate, saves time overall by catching mistakes early and sharpening thinking on edge cases, and can also be part of the documentation.
- At the other end of the pyramid, load testing gauges performance in extreme conditions and penetration testing resistance to adversarial attacks. These are only now starting to be part of ML development.

Safety and security

- Documentation and Guardrails
- Testing and Stress-testing
- **Versioning and Maintaining**
- Monitoring and Retraining

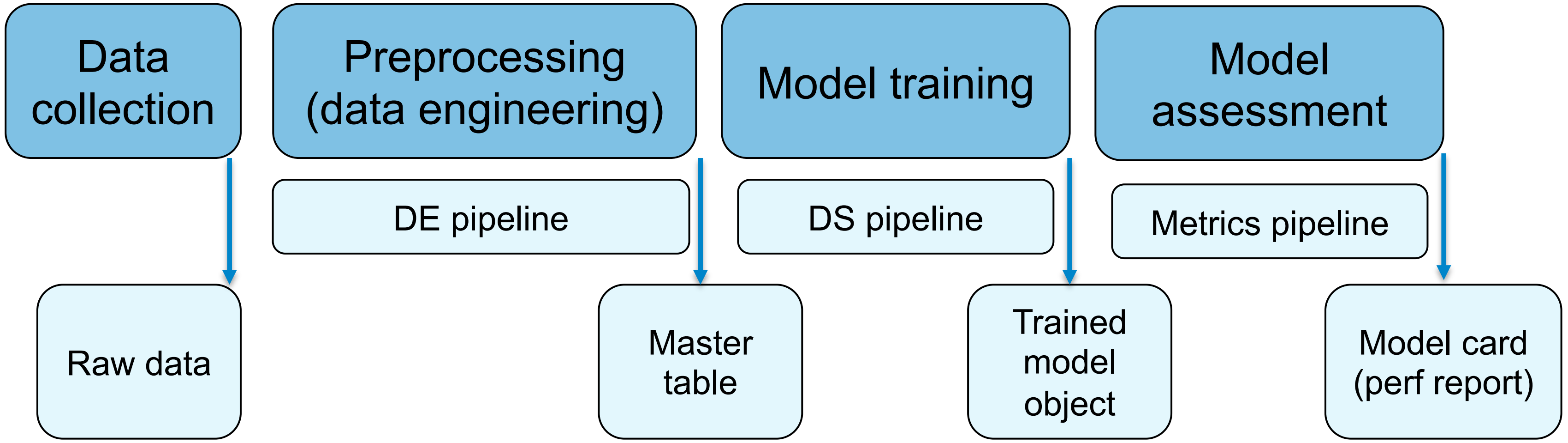
What is versioning?

- Versioning is a way to track changes in your code in a linear manner
- Even when experimenting back and forth on a piece of code, you will usually commit to a certain version as the “next step” – that’s a “commit” in git-speak.
- This means you can fall back to older versions when needed or can have a “master” fully functional version while you’re still working on the next update.

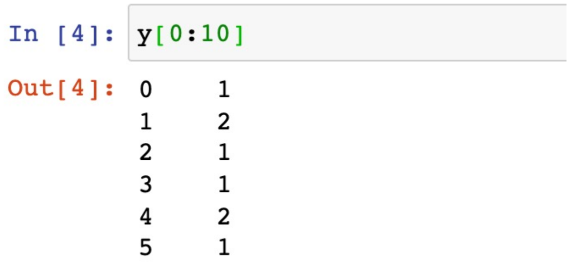
Github is the dominant, but not the only, version control system for software engineering.

Different git workflows

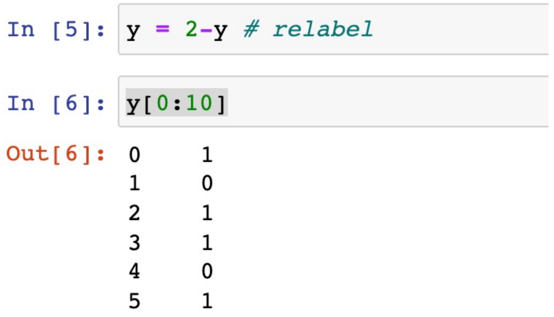
Data science pipelines contain more than code



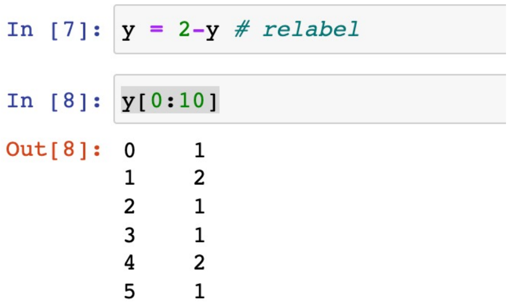
A Jupyter notebook is not always your friend



```
In [4]: y[0:10]

Out[4]: 0    1
        1    2
        2    1
        3    1
        4    2
        5    1
```



```
In [5]: y = 2-y # relabel

In [6]: y[0:10]

Out[6]: 0    1
        1    0
        2    1
        3    1
        4    0
        5    1
```

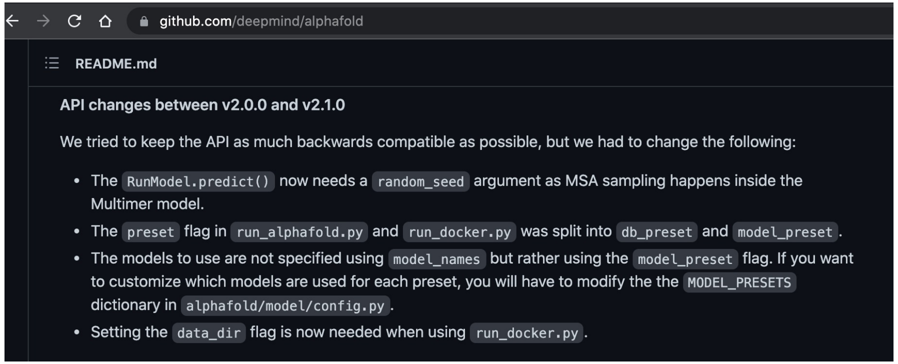


```
In [7]: y = 2-y # relabel

In [8]: y[0:10]

Out[8]: 0    1
        1    2
        2    1
        3    1
        4    2
        5    1
```

- Even without modifying the code, you can change the results of a notebook by modifying the order of execution. If all you are looking at is a final accuracy metric, you might fail to notice.
- After an initial phase of experimentation, every change to the code or order of execution should result in a new version of the model, whose accuracy metrics should be compared to other versions.

Maintaining

- Keeping a backlog of reported issues and making space to address them at regular intervals.
- Distinction between internal code versions (very frequent) and release versions (less frequent)
- **Changelogs** inform the users of what changed from one version to the next



API changes between v2.0.0 and v2.1.0

We tried to keep the API as much backwards compatible as possible, but we had to change the following:

- The `RunModel.predict()` now needs a `random_seed` argument as MSA sampling happens inside the Multimer model.
- The `preset` flag in `run_alphafold.py` and `run_docker.py` was split into `db_preset` and `model_preset`.
- The models to use are not specified using `model_names` but rather using the `model_preset` flag. If you want to customize which models are used for each preset, you will have to modify the the `MODEL_PRESETS` dictionary in `alphafold/model/config.py`.
- Setting the `data_dir` flag is now needed when using `run_docker.py`.

A segment of the README file of the open-source release of Alphafold by DeepMind

Summary

- Versioning your code makes you be more thoughtful and disciplined about your development.
- It also allows you to collaborate with others, and improve on a product already in use.
- Github is the primary tool for version control though there are others that work similarly.
- Data science pipelines are unique because they involve not only code, but also raw data, and data outputs from executing parts of the pipeline. This makes version control even more important.
- Continuing to improve a product or model after it has been released is common (good) practice. It involves tagging certain versions as “shippable” and carefully informing the user of any changes.

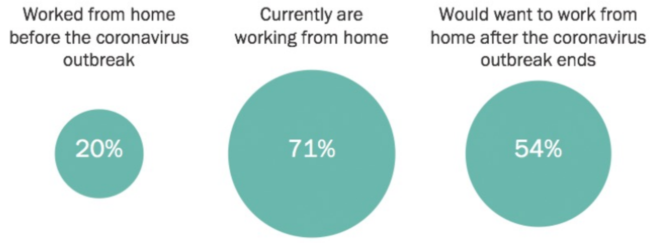
Safety and security

- Documentation and Guardrails
- Testing and Stress-testing
- Versioning and Maintaining
- **Monitoring and Retraining**

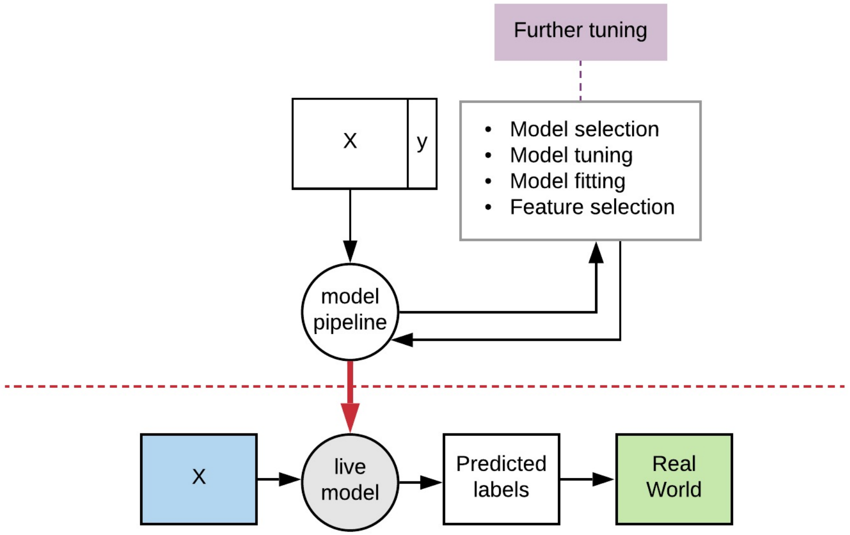
What is monitoring?

- Monitor usage
- Monitor dataset shift (changing distribution of input data)
- Monitor performance (requires access to labels)

Example: Covid pandemic has stressed many models trained on pre-Covid data, though the impact on performance depends on the use case



https://www.pewresearch.org/social-trends/2020/12/09/how-the-coronavirus-outbreak-has-and-hasnt-changed-the-way-americans-work/

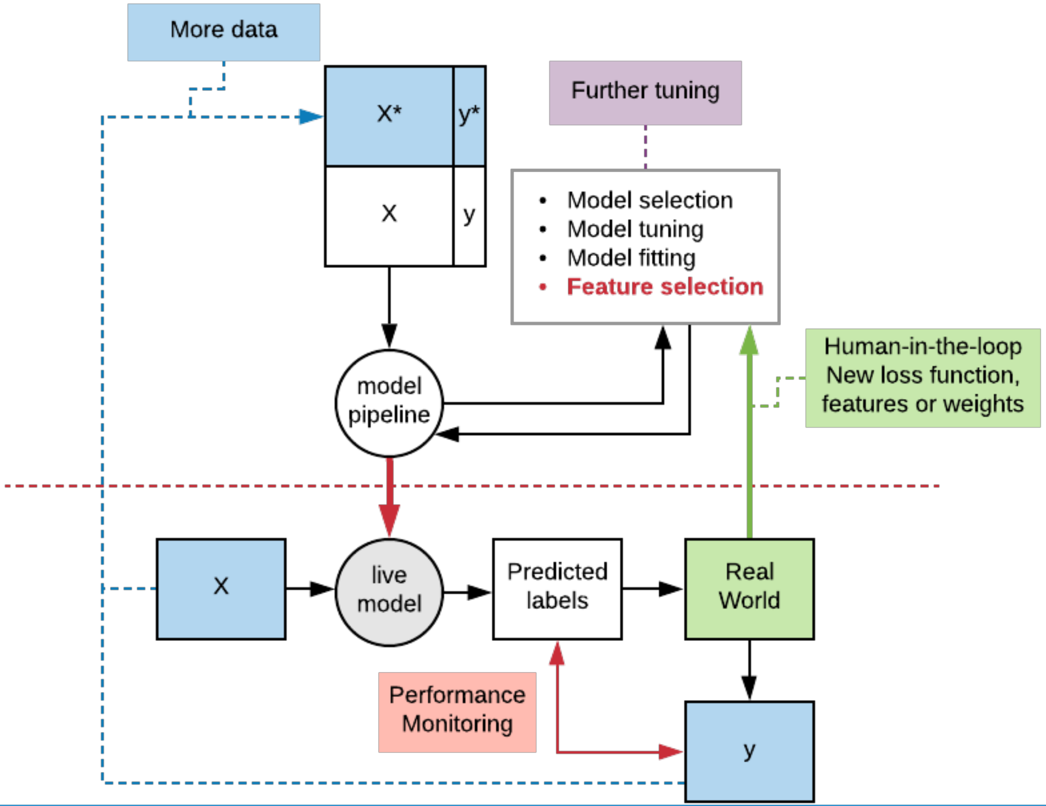

Further tuning
X
y
Model selection
Model tuning
Model fitting
Feature selection
model pipeline
X
live model
Predicted labels
Real World

🟦 More data

🟪 Further tuning

🟩 Human-in-the-loop New loss function, features or weights

🟥 Performance Monitoring



Safety and security - summary

- Documentation of code is necessary but not sufficient: the trained model object itself must be documented, via model cards or something similar.
- All the ethical considerations we have covered in the course can be summarily described in a consistent way in a model card.
- Safety is about safety in the hands of others.
- Best practices in software engineering have a lot to teach us.
- We still have a lot to discuss in later parts of the course on the topic of security, but we have already seen examples of “adversarial” privacy attacks.
- The completion of model training is the start, not the end, of its lifecycle.