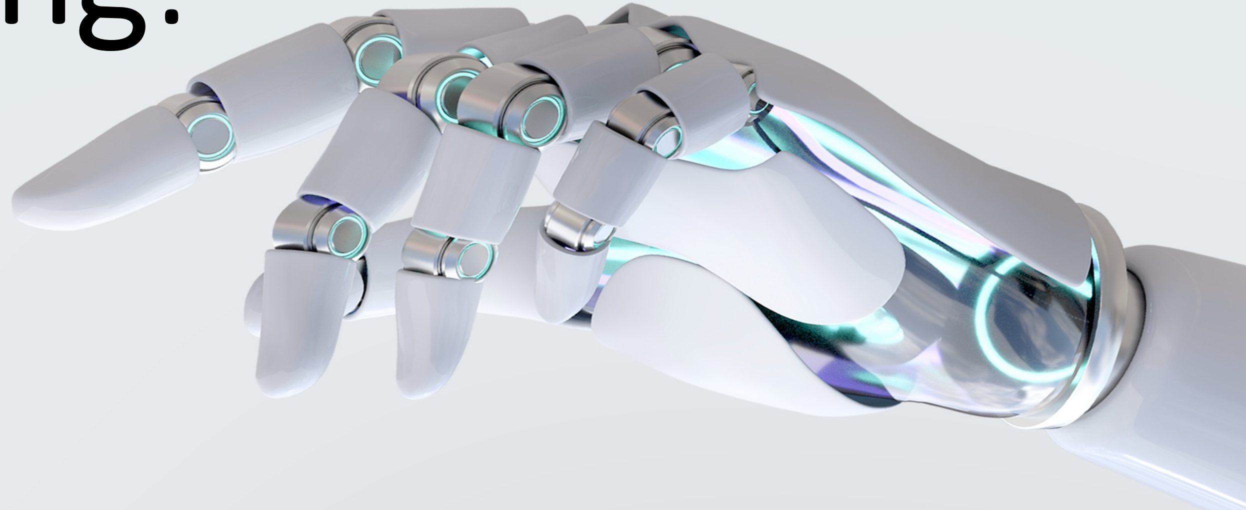
Information Science Research with **Machine Learning**:

Best Practices and Pitfalls



Andreas Vogelsang
University of Cologne
@andivogelsang

Tutorial @ RCIS 2022, Barcelona, Spain

Designed by rawpixel.com / Freepik

My research background

2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)

Requirements Engineering for Machine Learning: Perspectives from Data Scientists

Andreas Vogelsang
Technische Universität Berlin
Berlin, Germany
andreas.vogelsang@tu-berlin.de

Markus Borg
RISE Research Institutes of Sweden AB
Lund, Sweden
markus.borg@ri.se

***Abstract*—Machine learning (ML) is used increasingly in real-world applications. In this paper, we describe our ongoing endeavor to define characteristics and challenges unique to**

decisions in the development of ML systems are made by data scientists. These decisions include the definition of the fitness functions, the selection and preparation of data, and the

Abstract

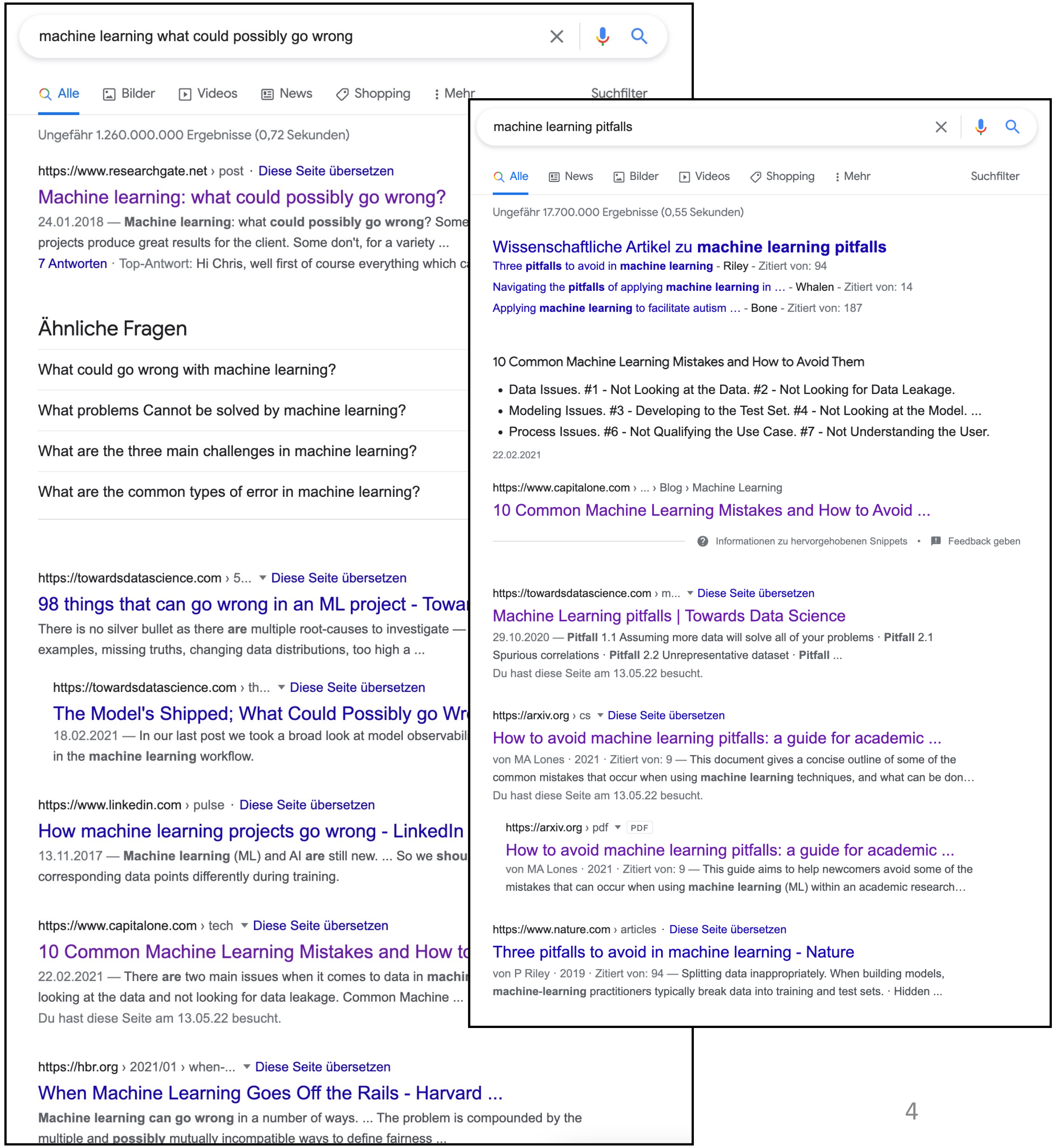
Causal relations in natural language (NL) requirements convey strong, semantic information. Automatically extracting such

Main Take-Aways from This Tutorial

- Think before you do ML
- ML is a tool, not a magic solution to everything
- If you do ML, do it properly and question your solution
- Evaluate the solution in its context; not only the model
- There is a lot that you can practically do wrong;
  code quality assurance is essential for your research

Scope

- Applying ML is easy; applying it reasonably is hard!
- There are general ML issues and specific issues for IS/SE research
- Focus of this tutorial: **Specific issues I see in SE/IS research**
- Target group:
  - Interested in applying ML in research
  - Basic knowledge about ML



What is Machine Learning?

Machine Learning (Simplified)

- Learn a function (called *model*)

  $$f(x_1, x_2, x_3, \ldots, x_n) \to y$$

  by observing data

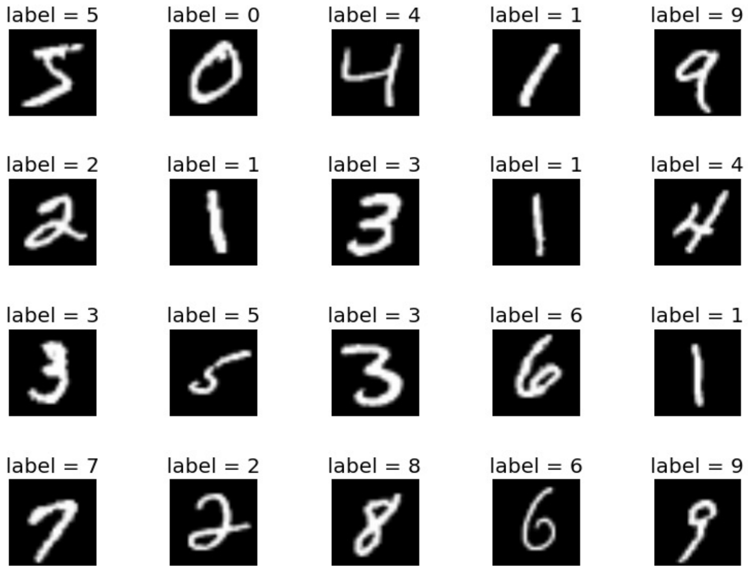
- Examples:
  - Detecting cancer in an image
  - Transcribing an audio file
  - Detecting spam
  - Detect suspicious activities for a credit card

- Typically used when writing that function manually is hard because the problem is hard or complex.

Example: Handwritten Digits

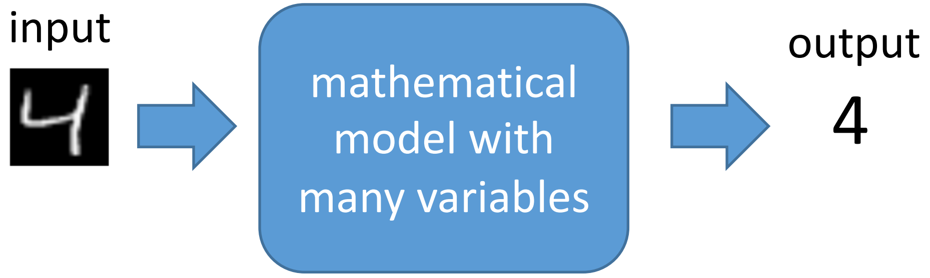
- Task:  Which digit is that? Easy: 4
- Not so easy: program a computer to solve this!

- The Machine Learning Approach:
- “Train“ a mathematical model to solve this task
- Training Data:
  - Many digits with labels (“classes“)



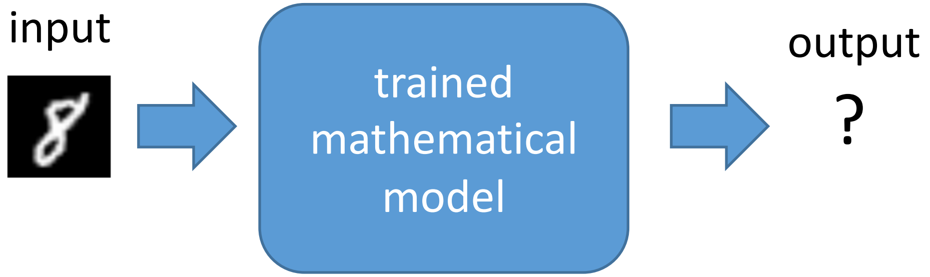
Example: Handwritten Digits

- Training Phase:



  - Repeatedly adjust the variables so that the model will compute output based on input on as many training examples as possible
  - Typically, an error function is minimized

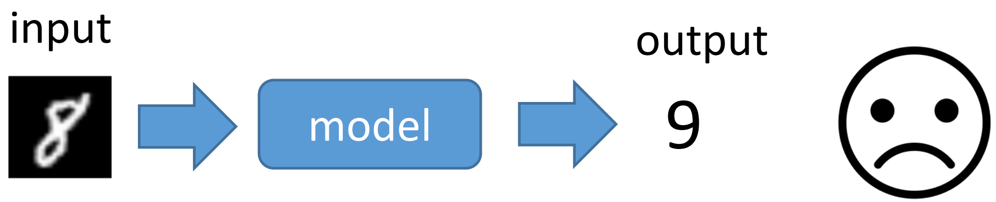
- Prediction Phase:



  - Use trained model to calculate output based on input

How good is our model?

- Idea: Given labeled data, how well can the function predict the outcome labels?



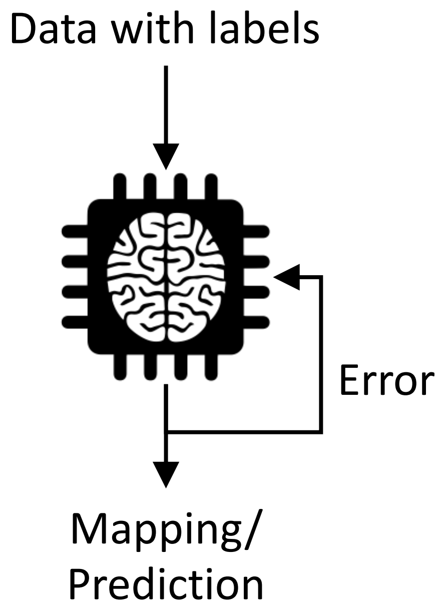
- Approach: Split dataset into training (90%) and test (10%) set
- Train on the training set, evaluate using the test set.
- Metrics
  - Accuracy: How many test examples were correctly classified?
  - Precision(class): How many of the examples predicted as class were correct?
  - Recall(class): How many examples of class did we classify correctly?

accuracy_train >> accuracy_test = sign of overfitting

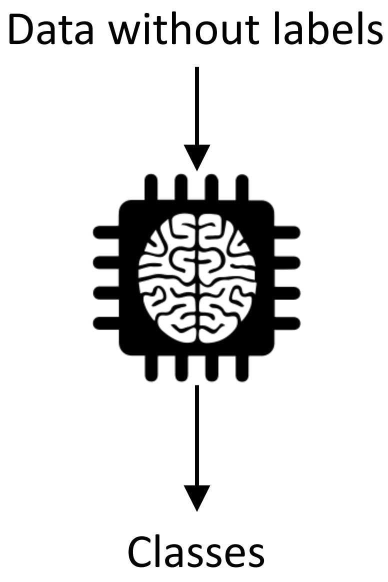
- Advanced: 10-fold cross validation

Types of Machine Learning

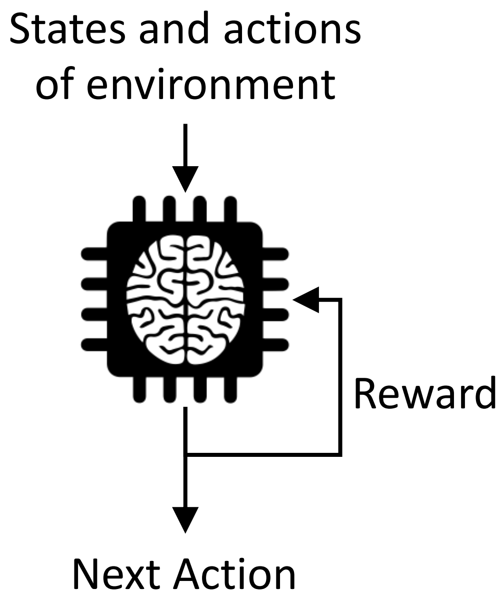
Supervised Learning



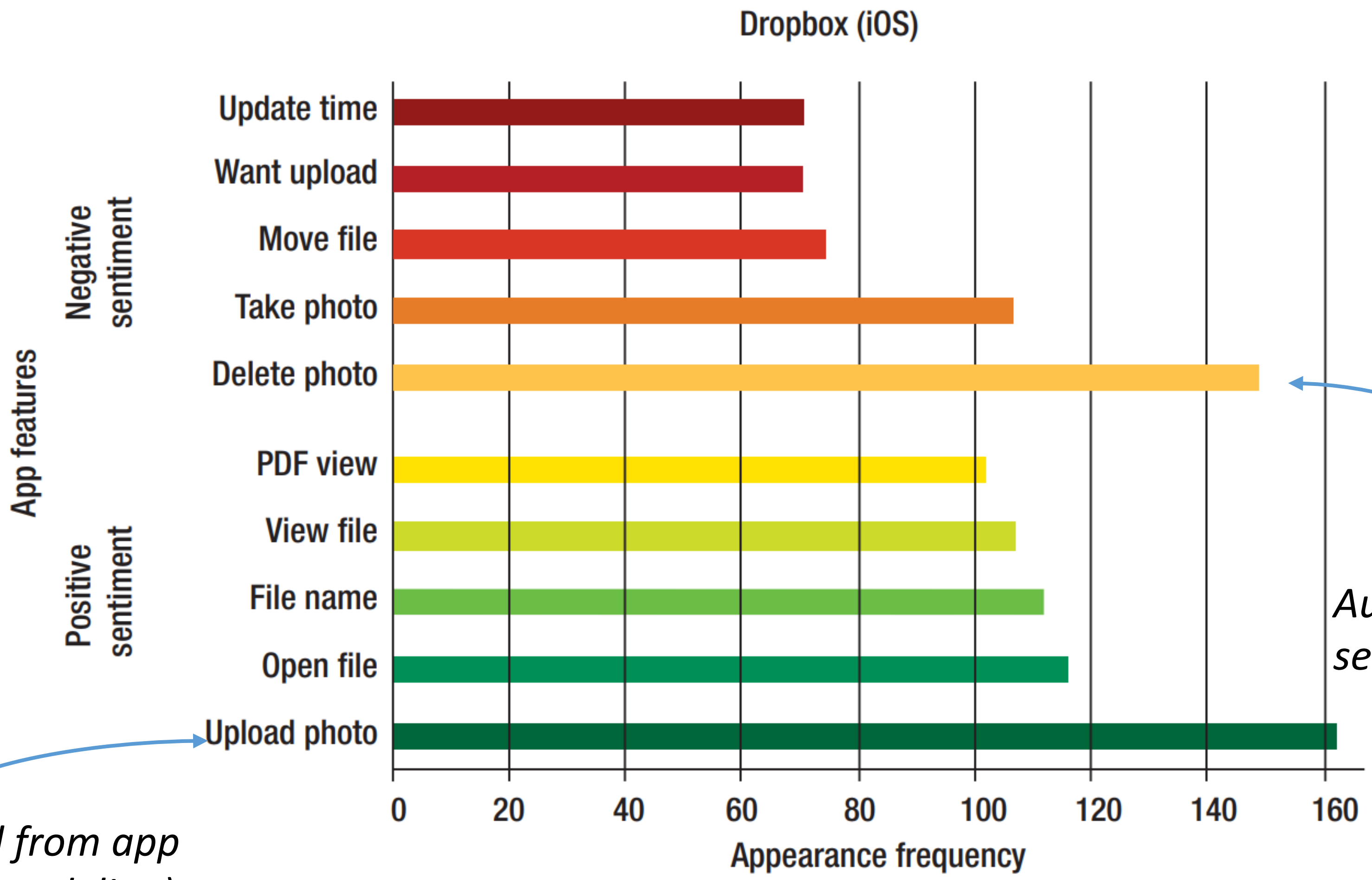
Unsupervised Learning



Reinforcement Learning



Applications: User Feedback Analysis



Automatically determined sentiment (ML or rule-based)

Automatically extracted from app reviews (NLP and topic modeling)

W. Maalej et al.: “Toward Data-Driven Requirements Engineering”, *IEEE Software*, 2016

Applications: Automatic Quality Assurance

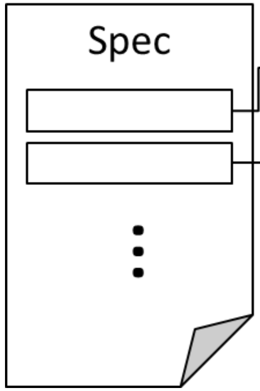
Detecting Domain-specific Ambiguities (based on word embeddings) [1]

Domain

Electronic Engineering (EEN)	Mechanical Engineering (MEN)	Medicine (MED)	Literature (LIT)	Sports (SPO)	Average
part 0.660	text 0.200	code 0.183	database 0.049	code 0.058	**code** 0.394
interface 0.664	support 0.430	**support** 0.351	support 0.170	**programming** 0.079	**database** 0.412
type 0.690	work 0.433	example 0.380	**set** 0.231	**system** 0.155	support 0.413
text 0.699	**part** 0.4473	**machine** 0.389	source 0.260	window 0.173	programming 0.474
version 0.703	type 0.458	form 0.390	code 0.275	machine 0.220	window 0.479
database 0.723	application 0.463	program 0.419	memory 0.310	source 0.243	text 0.484

Terms with lowest similarity in comparison to CS domain

Automatic Requirements Classification [2]



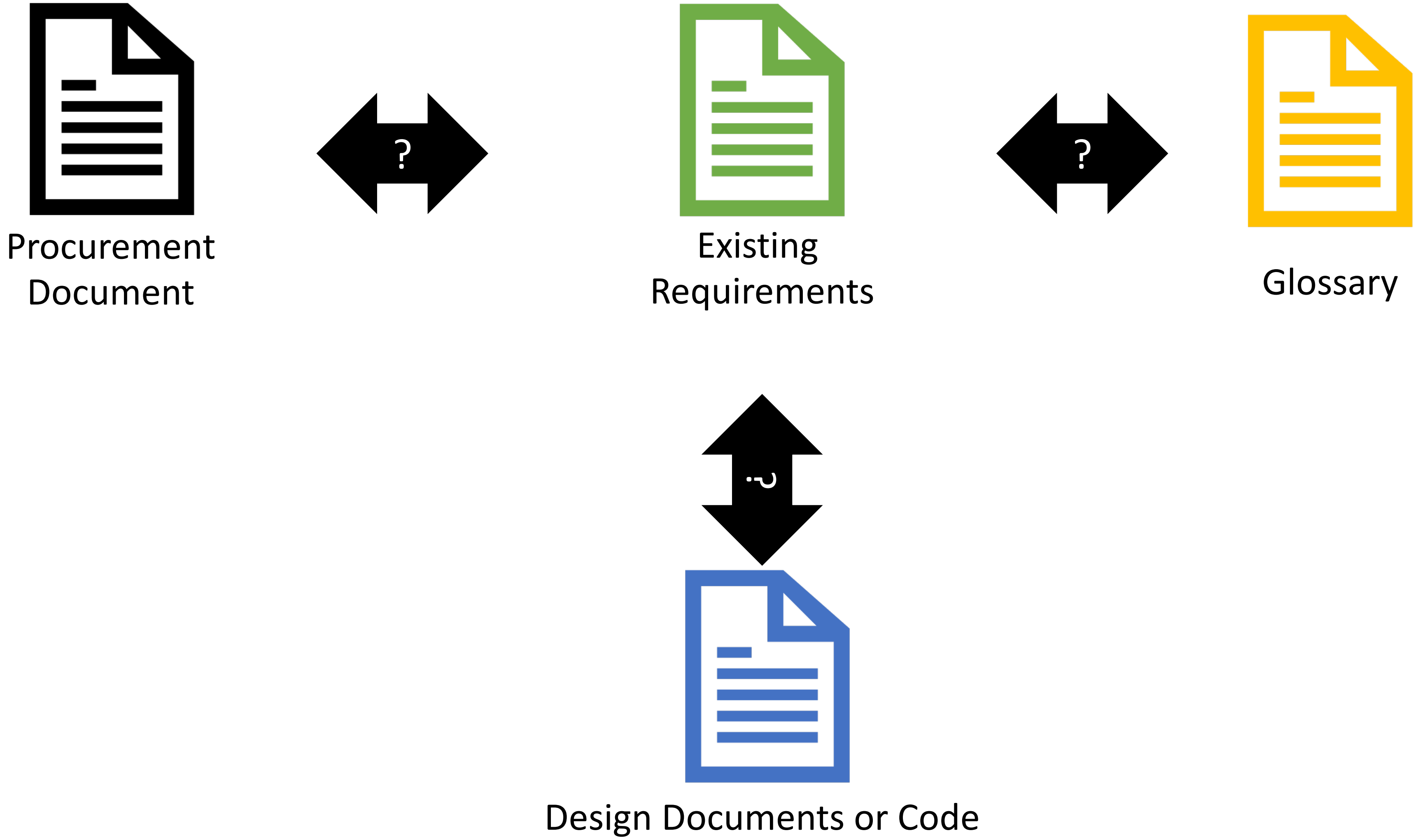
The duration until the switch is recognized as hanging must be a configurable parameter.

The component conditionally drives an external fan. This fan is required for active ventilation of the headlight.

Classification and feedback with neural networks

[1] A. Ferrari et al.: “Identification of Cross-Domain Ambiguity with Language Models”, *AIRE’18*
[2] J. Winkler, A. Vogelsang: “Automatic Classification of Requirements Based on Convolutional Neural Networks”, *AIRE’16*

Applications: Information Retrieval



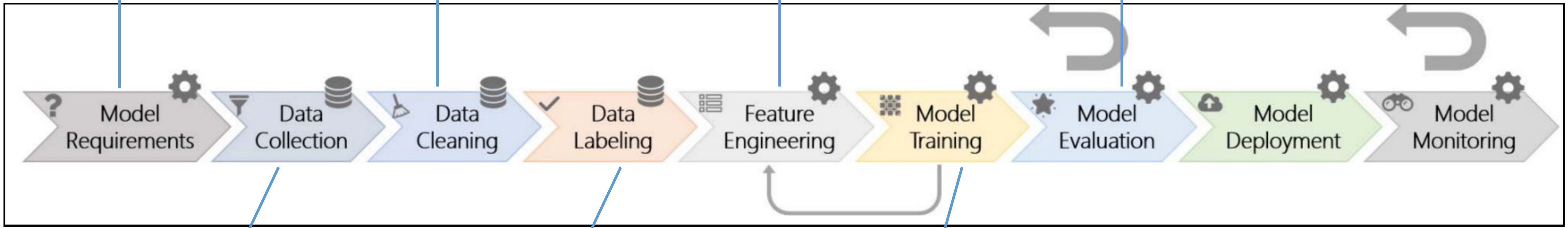
ML Pipeline

Think about the problem and its requirements

Remove wrong data, outliers, merge data from multiple sources

Convert raw data into a form suitable for learning, identifying features, encoding, normalizing

Determine fitness for purpose



Identify training data, often many sources

Identify labels on training data, possibly crowdsourced or (semi-)automated

Build the model, tune hyperparameters

Amershi et al.: Software Engineering for Machine Learning: A Case Study. *ICSE'19*

Machine Learning

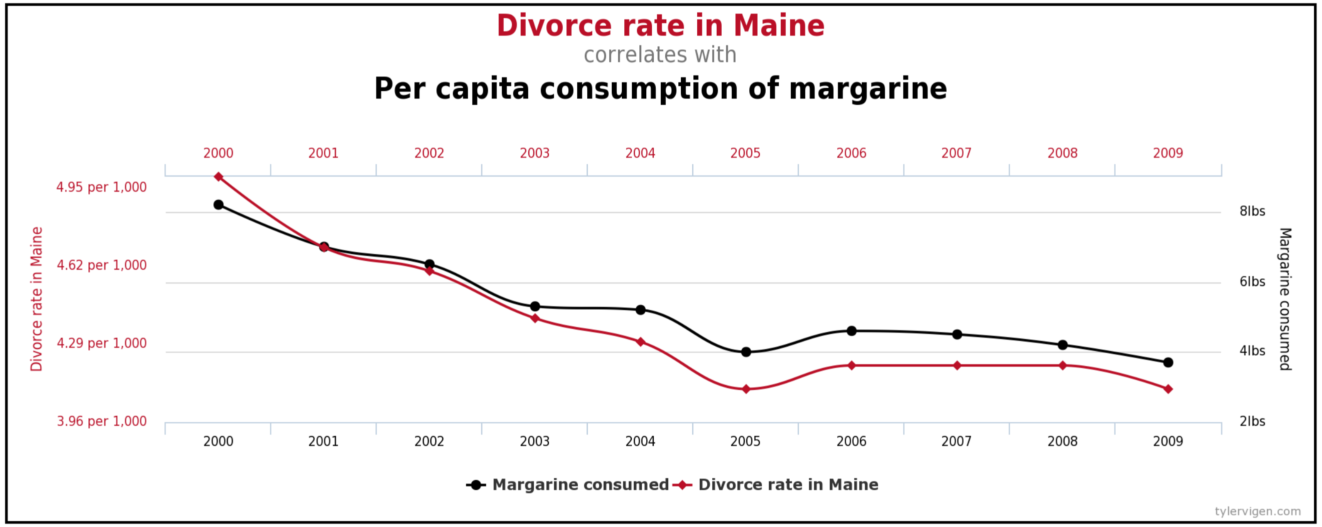
What could possibly go wrong?

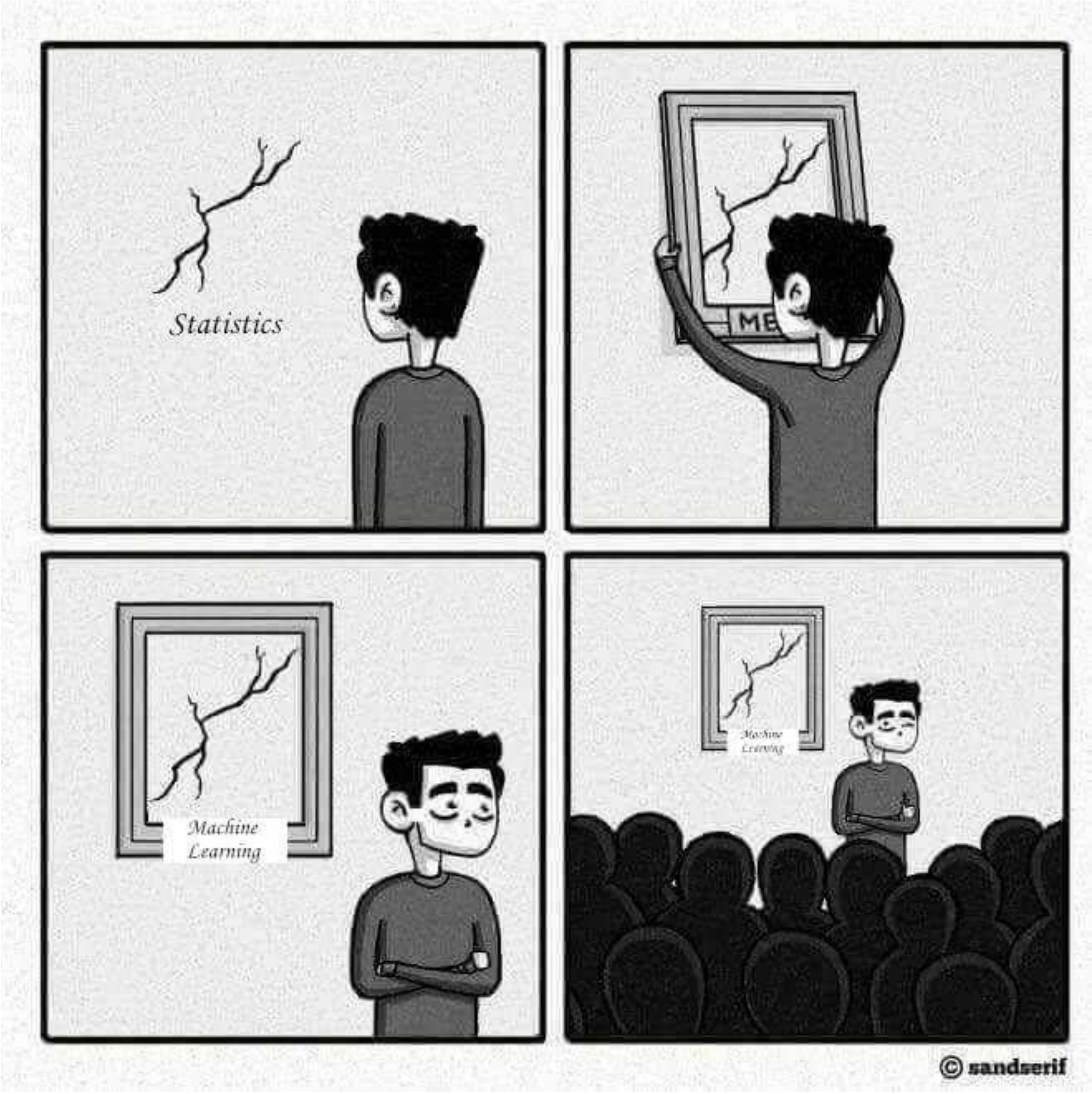
Pitfall #1: Problem? What problem?

- Theories/Hypotheses
  - Do you have any reason to believe that there is a relation between input and output?
  - Is this reason explicit enough in your data to compensate for noise?
- What is the achievable performance?
  - Is there actually a clear and correct answer for every input example? Would a human be able to identify this answer for all cases?
- What are reasonable performance metrics and requirements?

Think before you do ML

On Theories and Hypotheses





On Achievable Performance

Dog or Muffin?

Trace or no trace?

The DPU-BOOT CSC shall provide a monitor which accepts commands over the RS-232 interface.

Bootstrap Monitor: The Bootstrap Monitor checks entered commands for syntax and number of arguments, and displays an error message to the RS-232 interface if an invalid command or argument is entered. A complete listing of these messages is given in document 7384-BSPS-01.

Hardware Exceptions: The Bootstrap ignores any hardware exceptions that might occur while it is running. If an exception occurs, the Bootstrap simply resumes execution with the next instruction following the one at which the exception occurred.

On Performance Measures for ML

The truth is

The ML application predicts

	Case A	not case A
Case A	True Positives (TP)	False Positives (FP)
not case A	False Negatives (FN)	True Negatives (TN)

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{Recall} = \frac{TP}{TP+FN}$$

"The customers don't understand the performance measures."

-- Data science practitioner

Vogelsang, Borg: "Requirements Engineering for Machine Learning: Perspectives from Data Scientists", AIRE'19

Performance Measures for ML

- Example: Identify cancer in X-ray images
- Requirement:
  “The app shall have an accuracy of > 90%”

- Warning: Imbalanced training data
- What if the training data consists of
  - 95% images without cancer
  - 5% images with cancer
- A (trivial) algorithm that always predicts “no cancer” has an accuracy of 95%

The truth is

The ML application predicts	Case A	not case A
Case A	True Positives (TP)	False Positives (FP)
not case A	False Negatives (FN)	True Negatives (TN)

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

Change the requirement:
„The app shall have an accuracy of > 90% on a balanced training set“

Change the requirement:
„The app shall have a recall for detecting cancer of 100%“

Performance Measures for ML

- Example: Identify cancer in X-ray images
- Requirement:
  “The app shall have a recall for detecting cancer of 100%”

- Warning: Precision vs. Recall Trade-off
- A (trivial) algorithm that always predicts “cancer” has a recall of 100%
- Precision is only 5%. Does that algorithm help?

The truth is

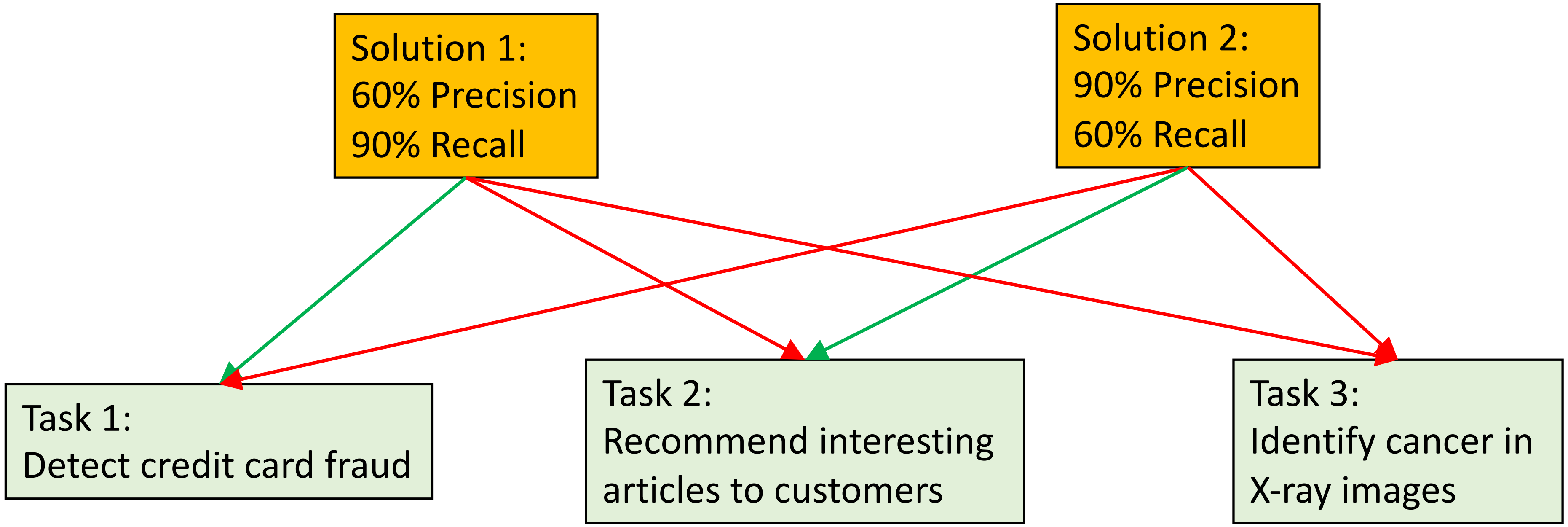
		Case A	not case A
The ML application predicts	Case A	True Positives (TP)	False Positives (FP)
	not case A	False Negatives (FN)	True Negatives (TN)

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

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

Performance Measures for ML

Specifying performance requirements for ML applications demands a rigorous analysis of the problem to be solved



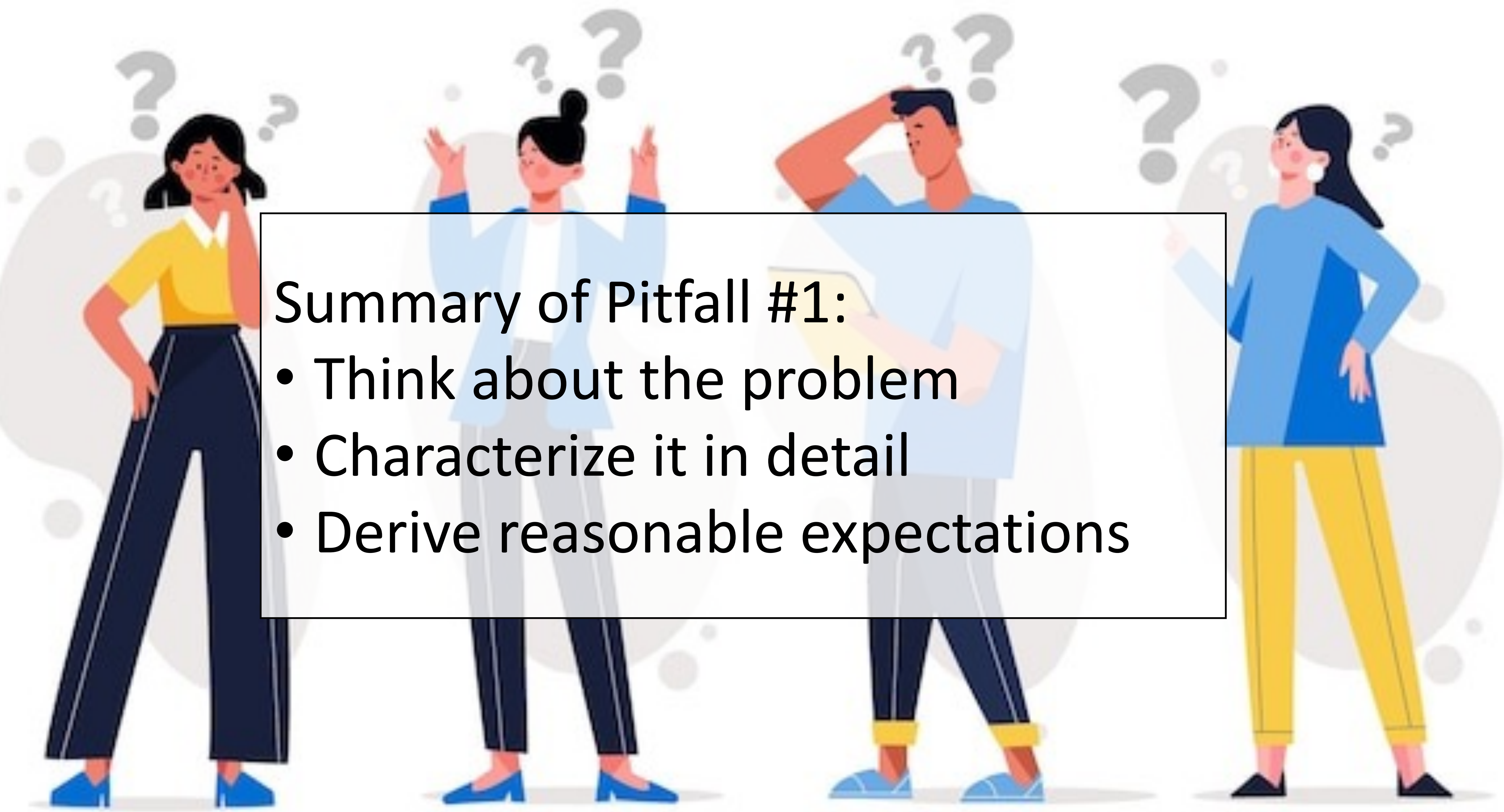
On Performance Measures

„[…] With an accuracy of 0.8 in our evaluation, our approach works quite well […]“

-- every ML for SE/IS paper

- So an accuracy of 0.8 is good? How do you know?
- Would an accuracy of 0.75 still be good or already bad?
- Is an accuracy of 0.9 possible or realistic?

- Also:
  - Is a false positive similarly bad than a false negative?



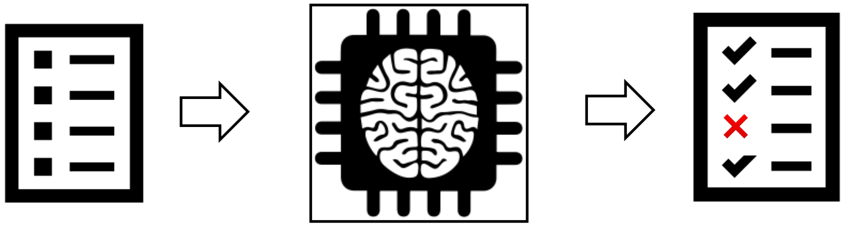
Summary of Pitfall #1:

- Think about the problem
- Characterize it in detail
- Derive reasonable expectations

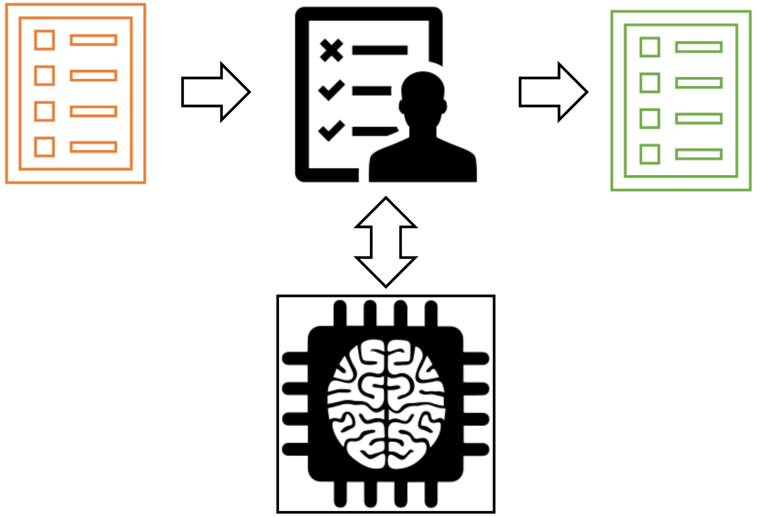
Pitfall #2: ML solutions and their context

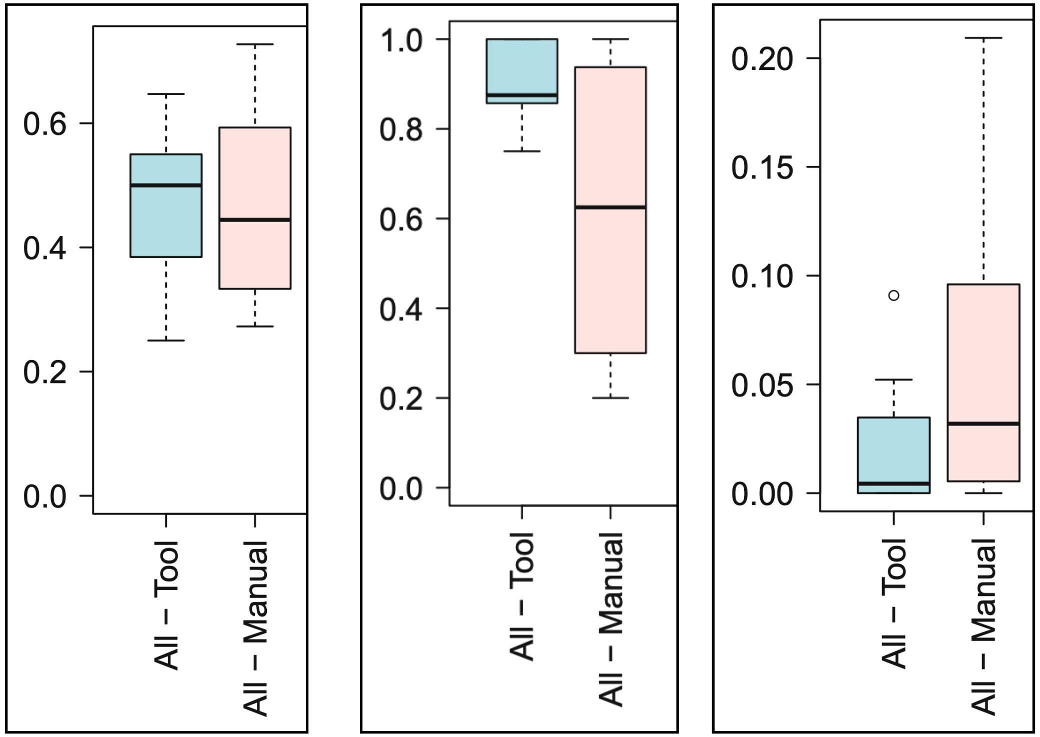
Most ML solutions in IS/SE research are used to assist the user. Therefore, understanding the problem **in its context** is crucial

The Human-in-the-Loop



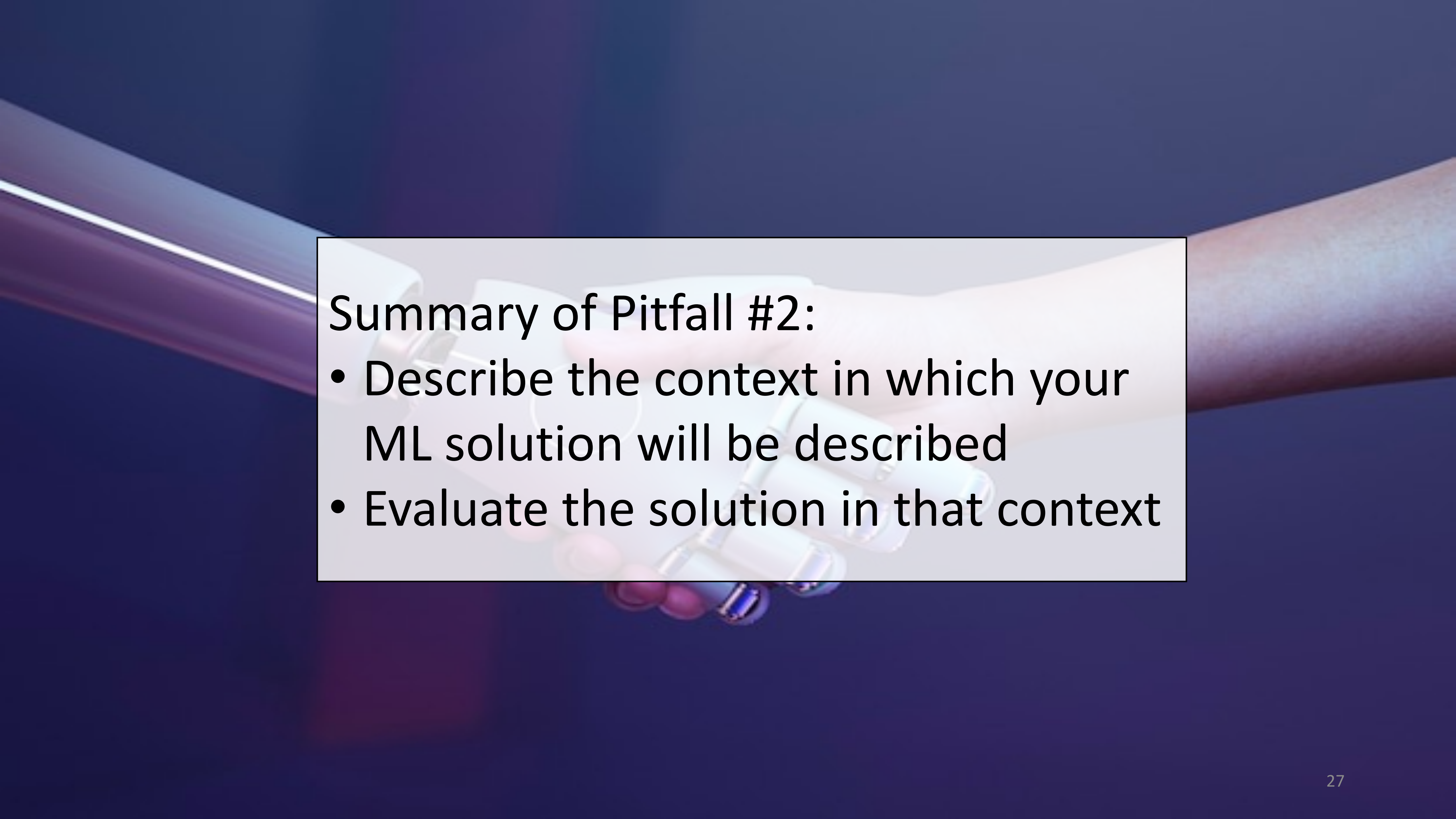
ML finds defects in requirements specifications:
Accuracy > 0.9





Defect correction rate | Unwarned defect miss rate | Requirements rephrase rate

Winkler, Vogelsang: “Using Tools to Assist Identification of Non-requirements in Requirements Specifications – A Controlled Experiment”, REFSQ’18



Summary of Pitfall #2:

- Describe the context in which your ML solution will be described
- Evaluate the solution in that context

Pitfall #3: Data Quantity and Quality

- A lot of data that we use is labeled by humans
- Human-labeled data should be treated with extra care
  - Humans make mistakes (obvious and maybe easy to fix)
  - Humans may have different labeling schemes (remember the tracing example)

- If you use human-labeled data
  - Label by at least two independent labelers and consider the inter-rater-agreement
  - Label iteratively and refine labeling criteria if necessary
  - Make labeling criteria explicit and write about them in the paper

On Data Quantity

- There is no rule or criterion for how much data you need to solve your problem with ML
- BUT:
  - In general, you need thousands of data points
  - For deep learning, you need at least tens of thousands

- Get as **much** and as **diverse** data as you can
- Evaluate model performance w.r.t. data size → does the performance still increase if you increase the dataset?

Summary of Pitfall #3:

- Double-check human-labeled data
- Discuss the amount of data w.r.t. the selected ML solutions

Pitfall #4: See.... It works!

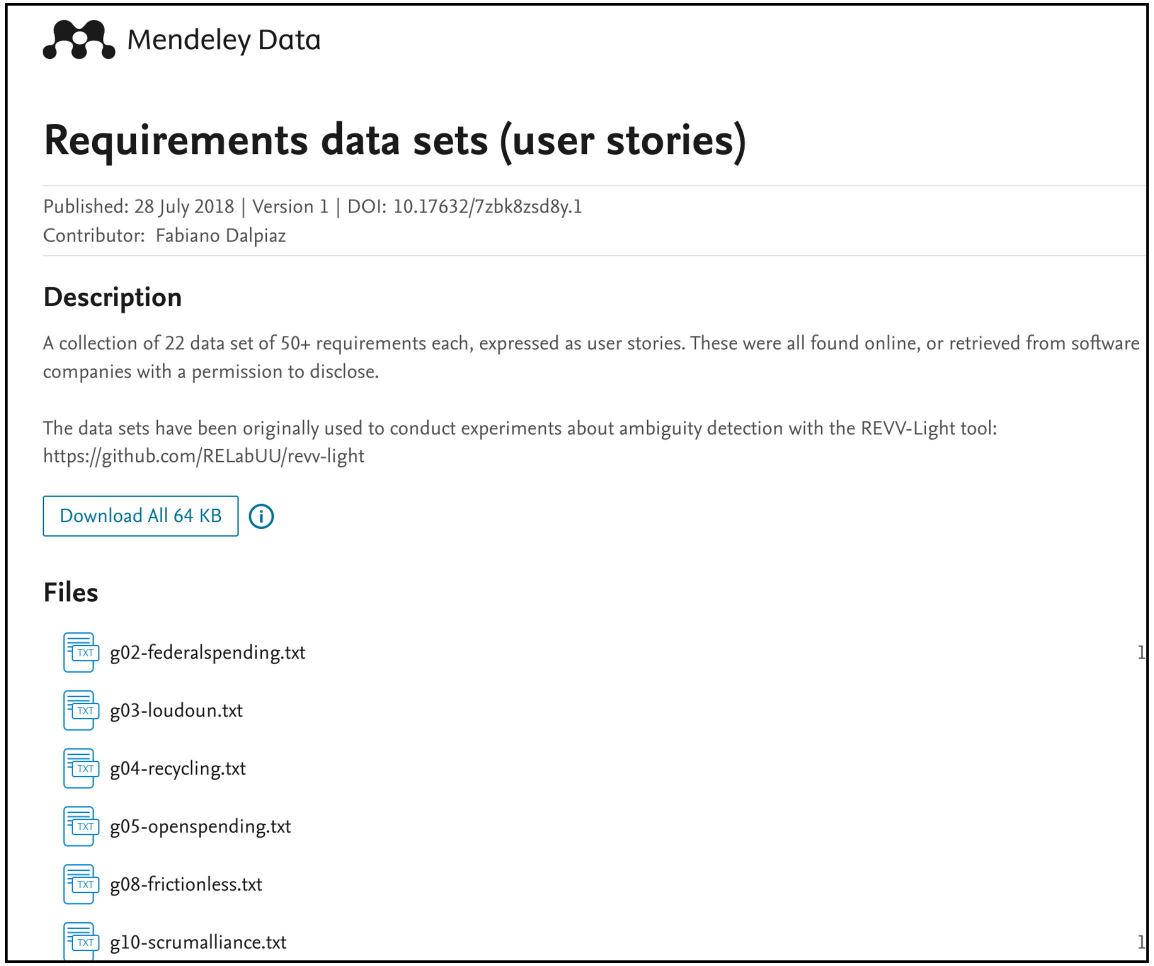
- It is not enough to present just the performance of your approach

- You should compare with
  - A trivial baseline approach (e.g., ZeroR classifier)
  - A simple (and interpretable!) ML approach (e.g., decision tree)
  - Other alternative approaches

- Use statistical tests to compare the classifiers (e.g., randomization test, t-test, Wilcoxon Signed Rank,…)

Train-Test Leakage

- Many datasets in SE/IS have some (hierarchical) structure
  - E.g., data points gathered from several projects
- Standard cross-validation splits data randomly
  - Potentially unique characteristics of single projects are part of the training and test set (train-test leakage)


Mendeley Data

Requirements data sets (user stories)

Published: 28 July 2018 | Version 1 | DOI: 10.17632/7zbk8zsd8y.1
Contributor: Fabiano Dalpiaz

Description

A collection of 22 data set of 50+ requirements each, expressed as user stories. These were all found online, or retrieved from software companies with a permission to disclose.

The data sets have been originally used to conduct experiments about ambiguity detection with the REVV-Light tool: https://github.com/RELabUU/revv-light

Download All 64 KB

Files

- g02-federalspending.txt
- g03-loudoun.txt
- g04-recycling.txt
- g05-openspending.txt
- g08-frictionless.txt
- g10-scrumalliance.txt

Train-Test Leakage

- Splitting datasets by structural properties may give a more realistic performance estimation

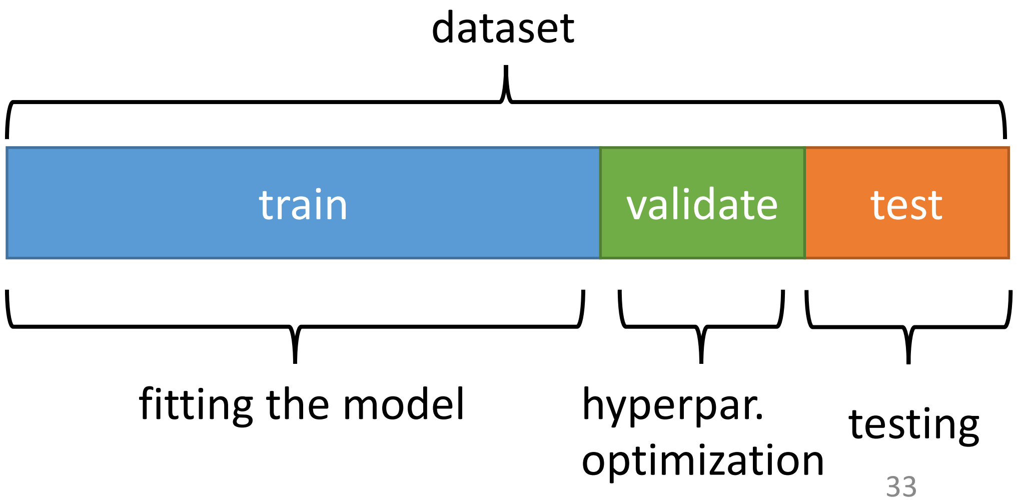
- If your dataset is composed of several similar “data sources” (e.g., projects), split the dataset into training, validation, and test by projects.

Standard randomized 10-fold cross-validation

PVM value	Support	Precision	Recall	F_1
Test	23,529	0.997	0.969	0.983
No Test	3,437	0.833	0.981	0.901

Leave-one-out 10-fold cross-validation

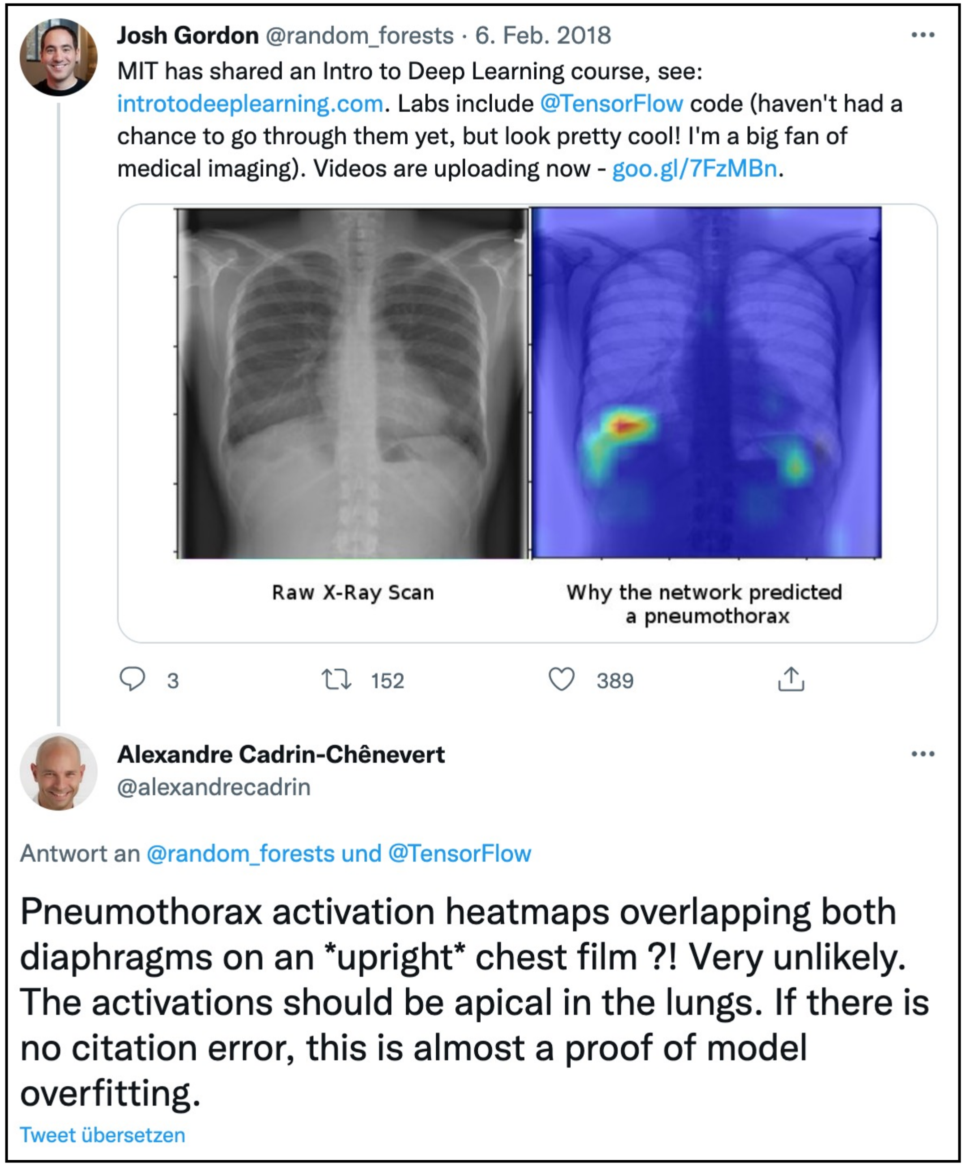
PVM value	Support	Precision	Recall	F_1
Test	23,529	0.948	0.962	0.955
No Test	3,437	0.437	0.366	0.399



No Qualitative Evaluation

- Show and analyze on which examples the model fails

- Involve domain experts and validate results with them


Josh Gordon @random_forests · 6. Feb. 2018

MIT has shared an Intro to Deep Learning course, see: introtodeeplearning.com. Labs include @TensorFlow code (haven't had a chance to go through them yet, but look pretty cool! I'm a big fan of medical imaging). Videos are uploading now - goo.gl/7FzMBn.

3 152 389

Alexandre Cadrin-Chênevert
@alexandrecadrin

Antwort an @random_forests und @TensorFlow

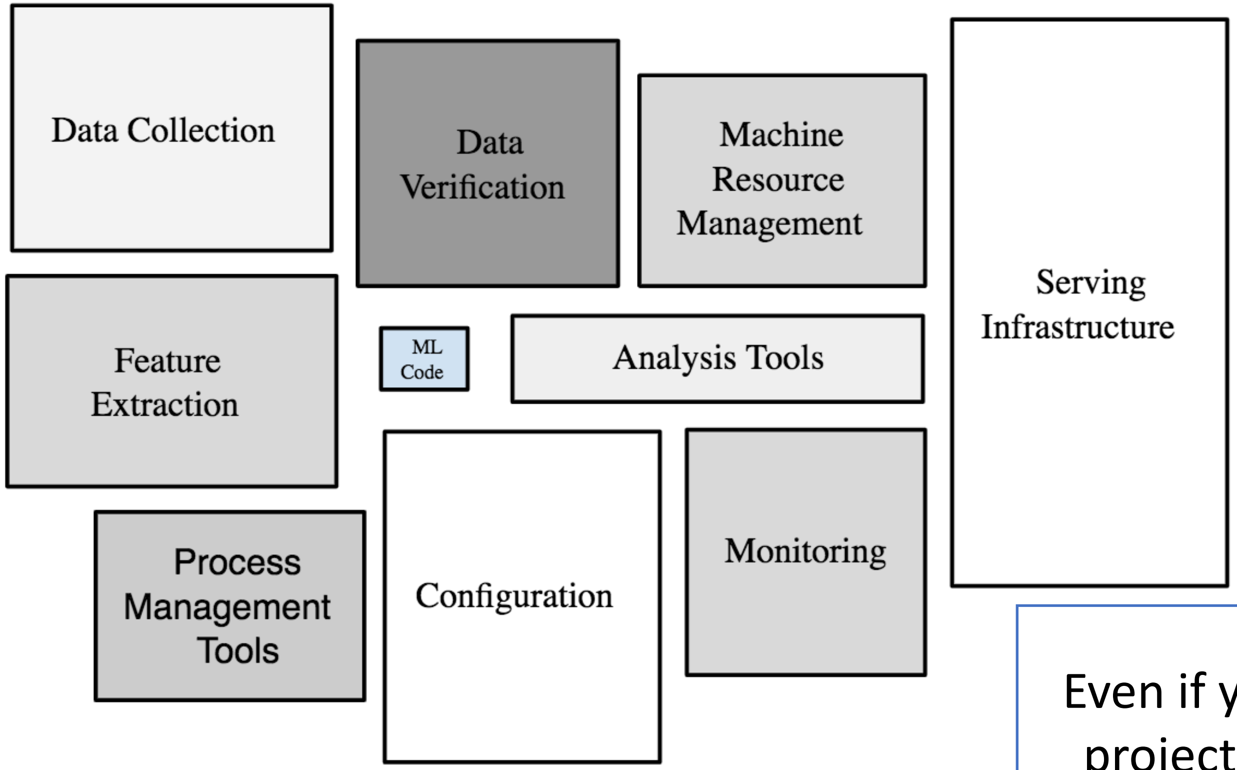
Pneumothorax activation heatmaps overlapping both diaphragms on an *upright* chest film ?! Very unlikely. The activations should be apical in the lungs. If there is no citation error, this is almost a proof of model overfitting.

Tweet übersetzen

Summary of Pitfall #4:

- Provide detailed quantitative and qualitative evaluations
- Check with domain experts

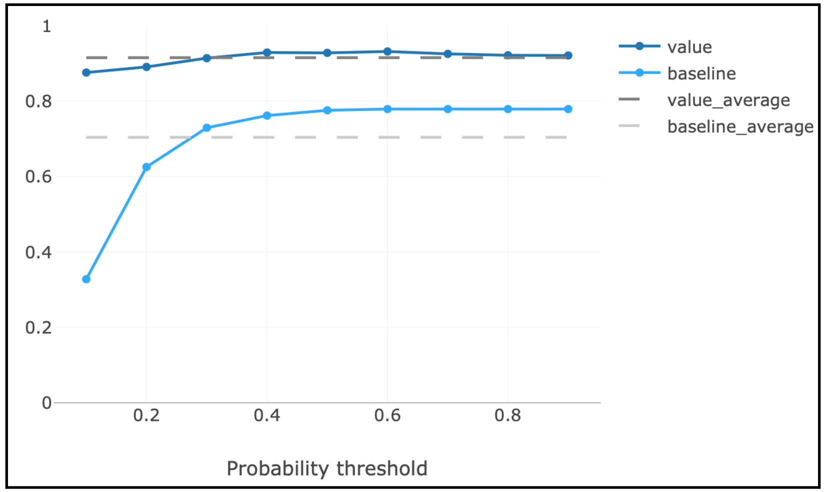
Pitfall #5: Research depends on complex code

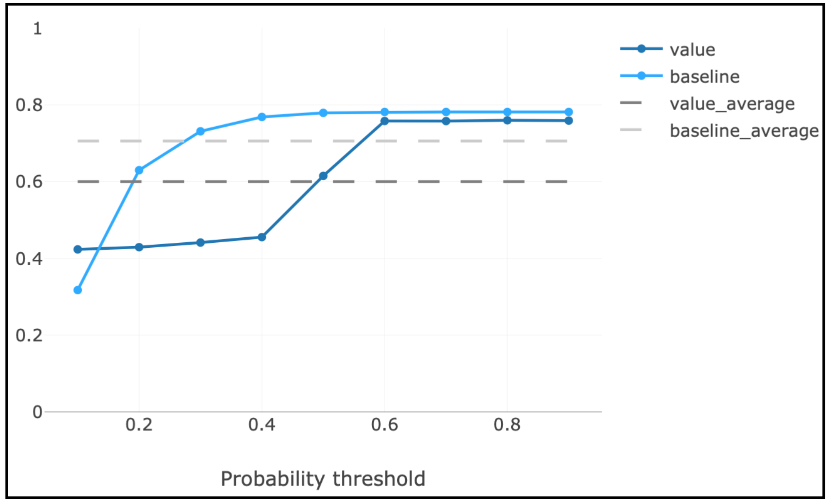


Even if you don't have all the components in a research project, it is still very likely that you have bugs in your pipeline

D. Sculley et al.: Hidden technical debt in machine learning systems. *NeurIPS'15*

From one of our own papers…






```
function train(data, model::Val{:bayesnet}, subsample = nothing)
    # extract graph layout
    graph_layout = Tuple(keys(data.edges))
    graph_data = subsample != nothing ? data.data[subsample,:] : data.data

    if size(graph_data, 2) > 0
        # remove completely empty lines, BayesNets does not like them
        graph_data = graph_data[sum(convert(Matrix, graph_data), dims = 2)[:] .> 0, :]
    end

    # add one, BayesNets expects state labelling to be 1-based
    graph_data = DataFrame(colwise(x -> convert(Array{Int64}, x) .+ 1, data.data), names(data.data))

    return BayesNets.fit(BayesNets.DiscreteBayesNet, graph_data, graph_layout)
end
export bayesian_train
```

subsample um Validierungs und Trainigsdata zu trennen

Daten ohne Validierungs-daten

Überschreibung durch Daten mit Validierungsdaten

alte Daten

Your research heavily relies on the correctness of your code.

Therefore,

- Do code reviews
- Write sanity checks and test cases along your ML pipeline
- Apply other basic SE practices (e.g., version control)

Publish your Data and Code

- There are several reasons why publishing code and data becomes even more important for data-driven research
  - Others are able to reproduce and check your research (see previous slide)
  - More importantly: Others are able to build upon your work
- If you can't publish data (e.g., because of an NDA), you may still be able to publish your data processing and learning pipeline

A hint especially for junior researchers:

- The benefit from someone "using" your research is much larger than the risk of someone "stealing" your idea, code, or data.
- Publishing messy code is better than publishing no code.

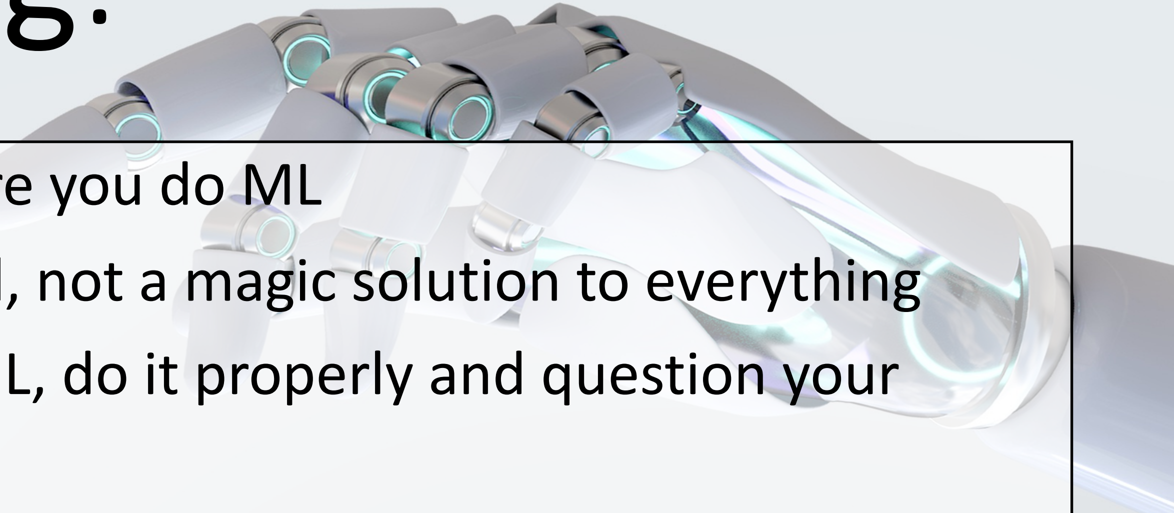
Summary of Pitfall #5:

- Check your code carefully
- Publish your code and data

Information Science Research with **Machine Learning**:

Best Practices and Pitfalls

Andreas Vogelsang
University of Cologne
@andivogelsang



- Think before you do ML
- ML is a tool, not a magic solution to everything
- If you do ML, do it properly and question your solution
- Evaluate the solution in its context; not only the model
- There is a lot that you can practically do wrong; code quality assurance is essential for your research

Tutorial @ RCIS 2022, Barcelona, Spain