Practical Aspects of Data Science

Data Science Retreat - 2019/B20

Patrick Baier

Course Goal

Prepare you to be ready for the daily work in a data science job

- ➔ **Model Learning (Day 1)**
  Model training, Evaluation metrics, imbalanced data problems
- ➔ **Model Operation (Day 2)**
  Probability calibration, Model deployment, missing features, monitoring

Schedule - Day 2

09:30 - 11:00	Probability Calibration
11:00 - 13:00	Web Service Deployment
14:00 - 15:40	Data Imputation
16:00 - 17:00	Monitoring + Data Biases
17:00 - 17:30	Wrap Up

Probability Calibration

Problem description

In binary classification (class zero and class one), a model gives us the probability that a data point belongs to class one (i.e. p = 0.7).

How accurate/well-calibrated is this probability?

Example:
Assume an ML predictor outputs p_{return} = 0.7 for an order. If the predictor is well-calibrated, we can assume that 70% of the orders that have the same features as this orders are going to be returned.

Calibration

Why is this important?

Many real-world problems require a realistic probability rather than a (binary) decision.
→ Estimation of money at risk (= p_{return} `* basketValue`).
→ Estimate click probability of a banner.

Different machine learning models are better or less suited:
→ Logistic regression: Naturally gives well-calibrated predictions.
→ Tree ensembles: Not so good calibrated predictions.
→ Oversampling leads to unrealistic probabilities.

Calibration Error (Brier Score)

$$BS = \frac{1}{N}\sum_{t=1}^{N}(f_t - o_t)^2$$

f_t predicted probability

o_t observed probability

N number of observations

Calibration Error (Brier Score)

How do we measure the calibration of our predictions?

1. Predict on test data.
2. Sort by raw probabilities in ascending order.
3. Create buckets for every x data points in ascending order.
4. For every bucket calculate:
   a. p_{real} : real probability (#ones / #all)
   b. p_{pred} : average predicted probability

5. RMSE(p_{real}, p_{pred}) over all buckets

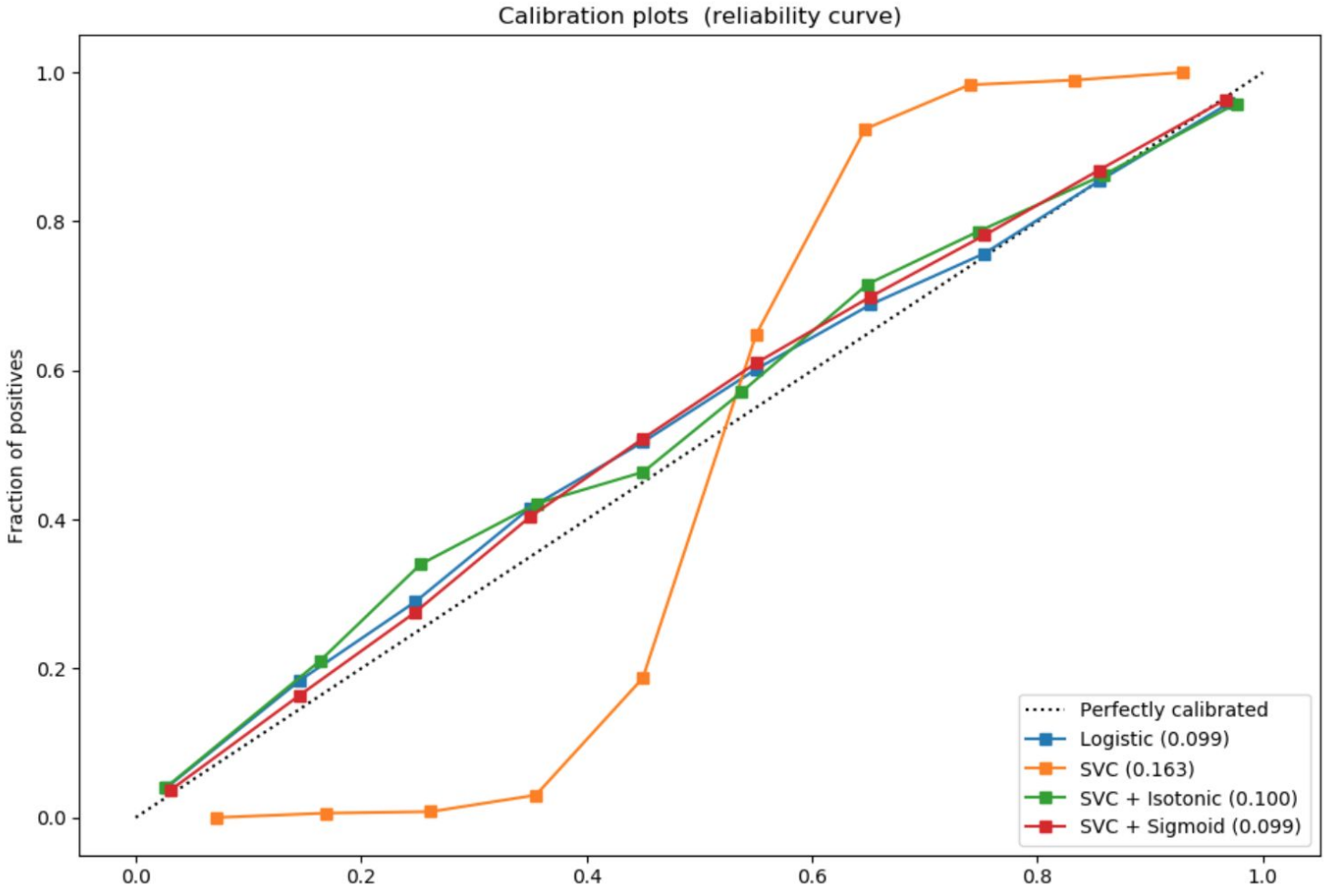
prediction	real label
...	...
0.51	0
0.53	0
0.54	1
0.56	1
0.58	1
...	...

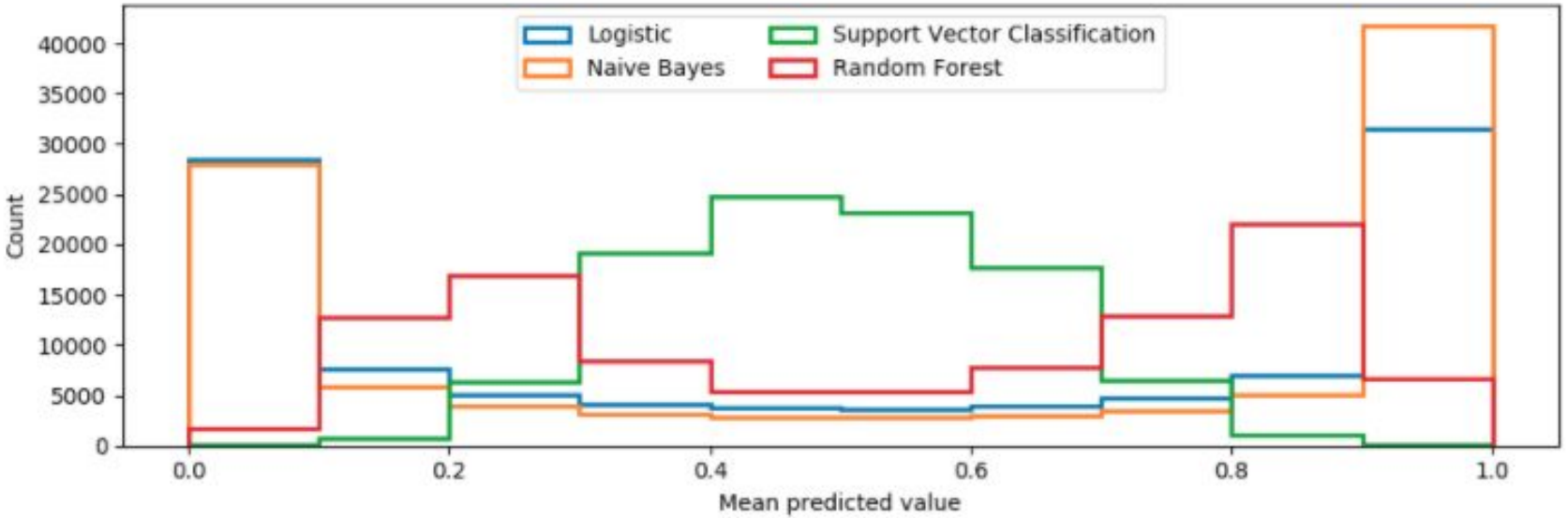
bucketSize = 5

$$p_{real} = 3 / 5 = 0.6$$

$$p_{pred} = (0.51+0.53+0.54+0.56+0.58)/5 = 0.544$$

$$err = 0.6-0.544$$


Calibration plots (reliability curve)
Fraction of positives
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Perfectly calibrated
Logistic (0.099)
SVC (0.163)
SVC + Isotonic (0.100)
SVC + Sigmoid (0.099)



Very distribution depending on algorithm:

- Logistic regression has predictions all over the range
- Random Forest has only few predictions close to zero and one

(Simple) Calibration Function

Learn calibration function on top of existing ML model on training data. This function maps raw prediction to calibrated prediction.

Learning:

1. Create buckets (as on previous slide) based on training predictions.
2. For every bucket: Calculate real return-probability (using true labels) and remember mapping: bucket → calibrated return-probability

Prediction:

1. Predict on data point with ML model.
2. Lookup bucket for prediction and output calibrated return-probability.

Learn Calibration Function

prediction	real label
...	...
0.51	0
0.53	0
0.54	1
0.56	1
0.58	1
...	...

`bucketSize = 5`

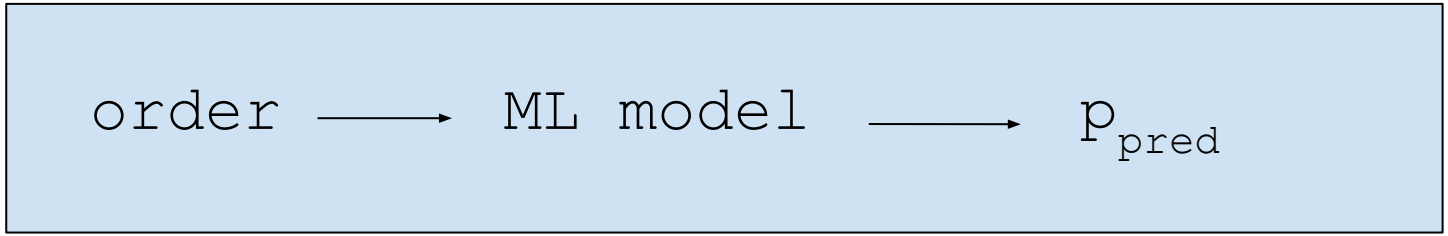
Learn function:

$$f(p_{pred}) \rightarrow p_{real}$$

$$f([0.51...0.58]) \rightarrow 3/5$$

Apply Calibration Function

1. Prediction step



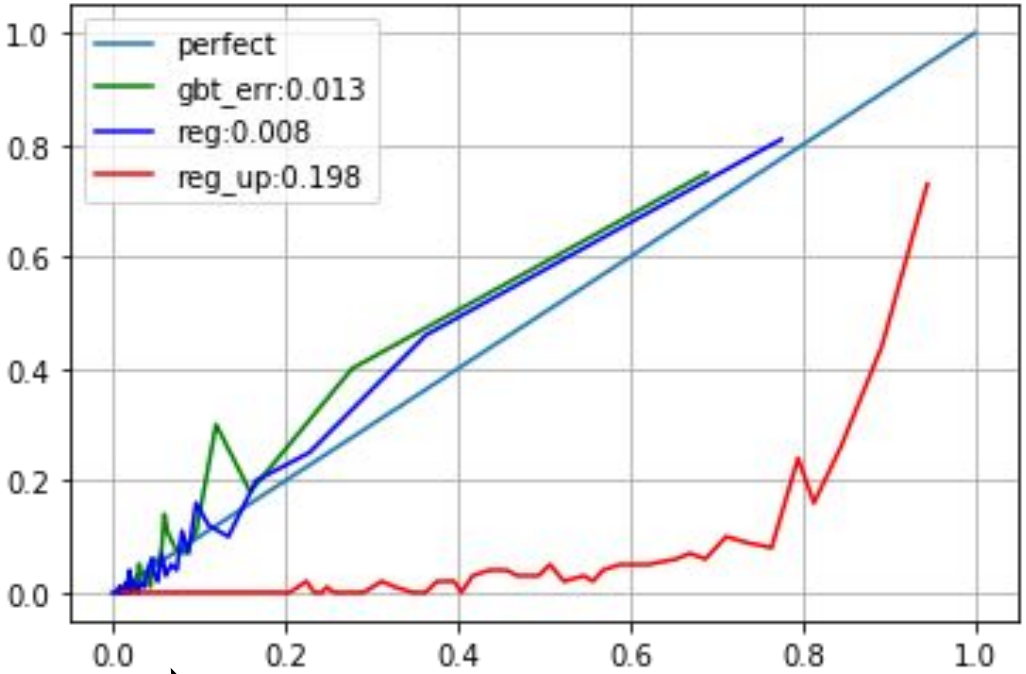
2. Calibration step

$$f_{calb}(p_{pred}) \rightarrow p_{real}$$

Task 4

1. Check the calibration error of the models from task 1:
   - Form buckets of 100 consecutive predictions (starting from lowest probability).
   - Plot the error over the probability range (see next slide for example)
   - Calculate the mean error over all buckets (Brier score).
2. Learn a calibration algorithm to calibrate your model:
   - Form buckets of 100 consecutive predictions on the training data.
   - Learn the mapping from buckets to real probability.
3. Compare the outcome to the uncalibrated case.

Task 4



Real probability
(fraction of
positives per
bucket)

Predicted probability
(Avg. prediction per bucket)

Task 4

4. How does this classifier behave in terms of:
   - Discriminative power? (area under the roc)
   - Calibration?

5. What would be an classifier that has the opposite characteristics?

p prediction	real label
0.3	1
0.2	1
0.10	1
0.09	1
0.06	0
0.05	0
0.04	0
0.03	0
0.02	0
0.01	0

More Advanced Algorithms

There are more advanced algorithms for probability calibration (which in most cases do not turn out to be much better).

- Sigmoid calibration: Learn a logistic regression on top of the prediction

  $p_{real} = 1 / (1 + e^{(a*p_{pred} + b)})$

- Isotonic calibration:
  - Finds a non-decreasing approximation of a function
  - Results in a function that is piecewise linear (see next slide)
  - http://fa.bianp.net/blog/2013/isotonic-regression/

http://scikit-learn.org/stable/modules/calibration.html

MIMIC Calibration Algorithm

One calibrated that works good in practice proposed by Magnetic.

Self study:
Read the following blog post and note down the characteristics of the calibration algorithm (one person needs to present the details).
How is it different from the one we implemented before?

http://tech.magnetic.com/2015/06/click-prediction-with-vowpal-wabbit.html

Calibration Resources

Some interesting reads:

- https://www.svds.com/classifiers2/
- http://scikit-learn.org/stable/modules/calibration.html
- https://jmetzen.github.io/2015-04-14/calibration.html

Model Deployment

Real-time Predictions

Remember our requirement:

Built a binary classification model that predicts **in real-time** the probability if a customer returns the order.

→ We have to build a webservice that contains our model:

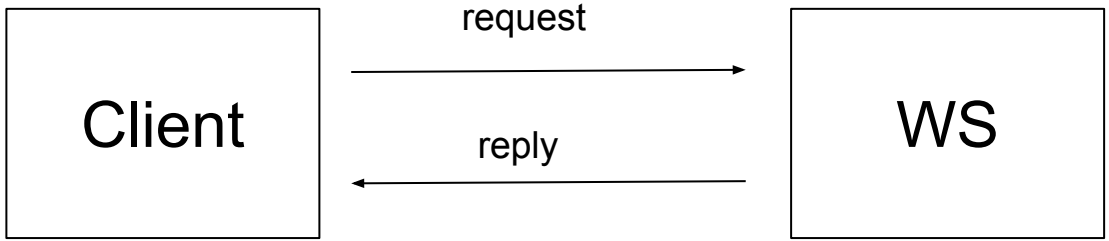
Input: Feature vector

Output: Return probability

Webservice

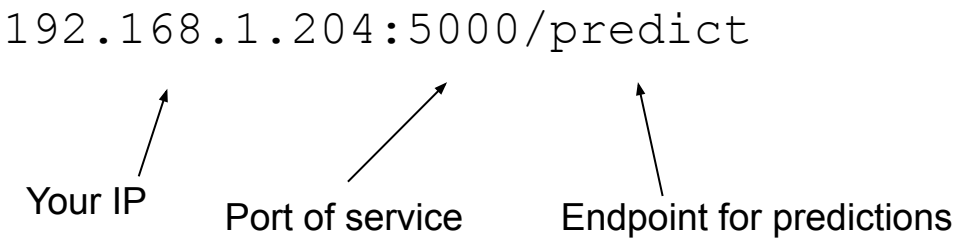
A webservice is a program that receives requests and answers them.

Most famous: web server, i.e. receive requests for HTML pages (from a browser), sends back HTML pages.



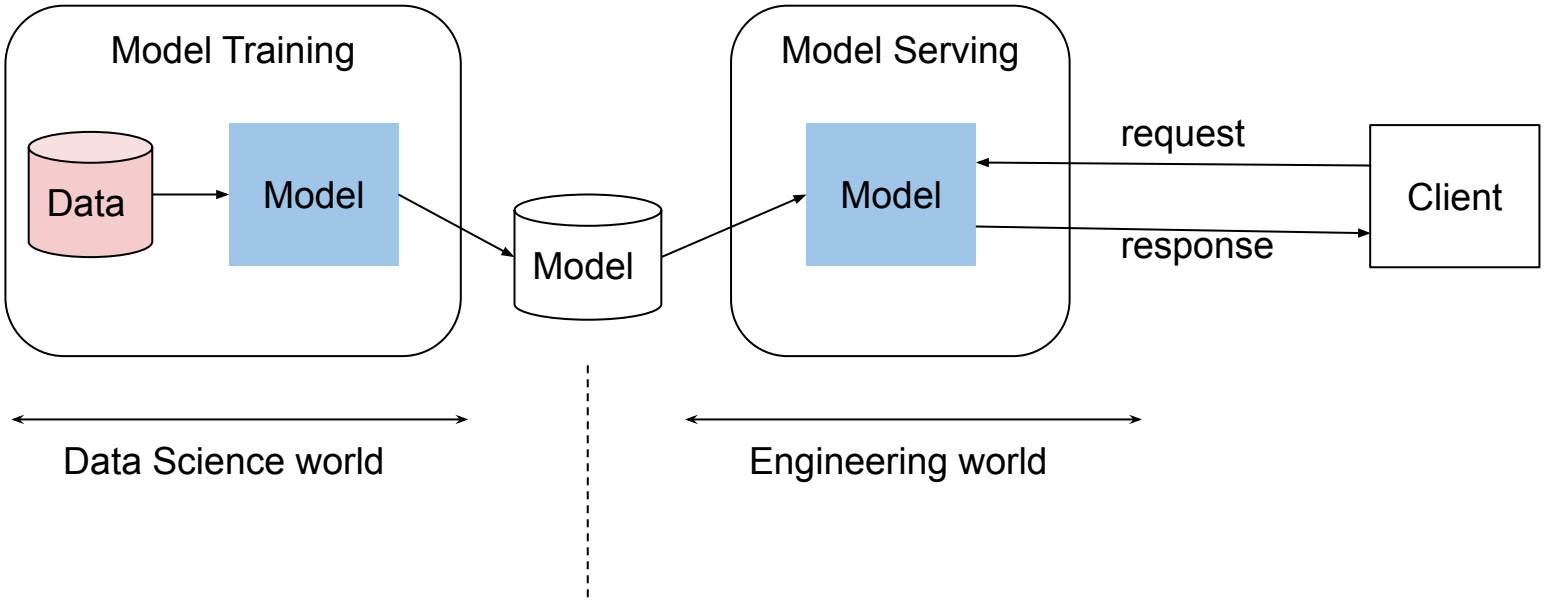
Webservice

We provide our model to the outside world through such a service.
i.e. other people send a request in the form:



For this, we have to encapsulate our prediction model into a such a webservice.

Model in Production



Flask Webservice

Let's build a REST service for predictions:

```
@app.route('/predict', methods=['POST'])
def predict():
    basket = request.json['basket']
    zipCode = request.json['zipCode']
    totalAmount = request.json['totalAmount']
    p = probability(basket, zipCode, totalAmount)
    return jsonify({'probability': p}), 201
```

Bind method to `predict` endpoint of server

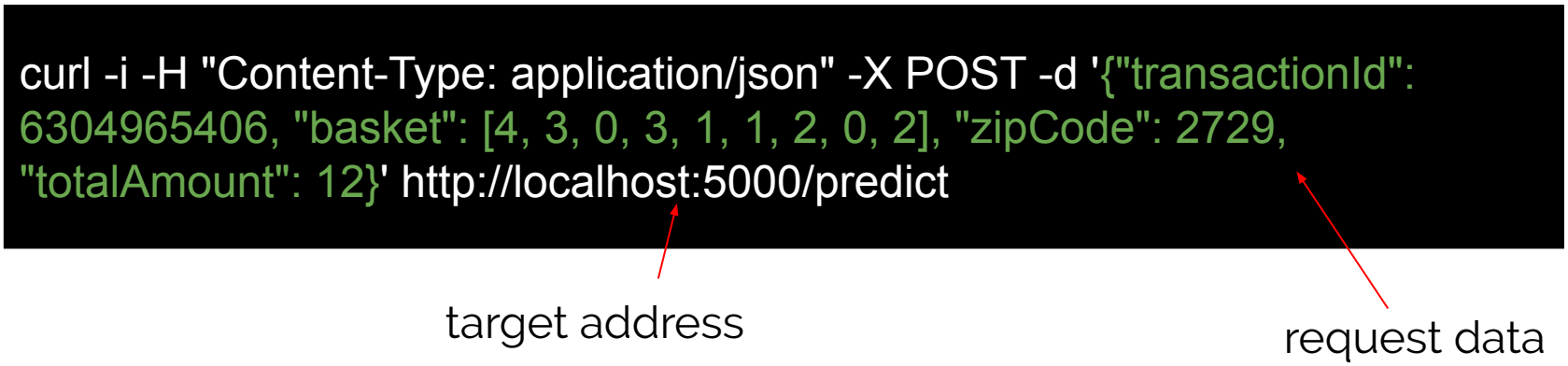
Extract fields from request that was sent by client

Return return probability to client (as json)

Test Request

Send a request to the server (bash script *request.test*):

```
curl -i -H "Content-Type: application/json" -X POST -d '{"transactionId":
6304965406, "basket": [4, 3, 0, 3, 1, 1, 2, 0, 2], "zipCode": 2729,
"totalAmount": 12}' http://localhost:5000/predict
```



Flask will make the request data available in Python as request object.

Task 1

1. Get the webserver up and running on your machine.
   Execute in a shell: `python server.py`

2. Execute `request.test` to send a requests to your server:
   Execute in a different shell: `source request.test`

For Windows users:
> set FLASK_APP=server.py
> python -m flask run
> curl -i -H "Content-Type: application/json" -X POST -d "{\"transactionId\": 6304965406, \"basket\": [4, 3, 0, 3, 1, 1, 2, 0, 2], \"zipCode\": 2729, \"totalAmount\": 12}" http://localhost:5000/predict

Task 2

1. Extend the webserver to handle requests using the ML model.
   - Save your model in the jupyter notebook, example: http://scikit-learn.org/stable/modules/model_persistence.html

   - Load the model into your server
   - Predict with the model on the incoming request

2. Run the request simulation tool to test your server:
   Execute in a shell: `python3 loadSimulator.py`
   Why is it failing for some requests?

3. Bonus: Return a calibrated probability.

Missing Features

Problem

- We often have data points where one or more features are missing/nulls: $\langle x_1, x_2, \texttt{null}, x_4 \rangle$
- We cannot train or predict on only a subset of features

Why are values missing?

- network problems
- timeouts
- optional fields
- unavailability
- ...

What can we do?

Feature is missing in live system:

- Value Imputation
- Multiple models

Feature is missing in training:

- Drop data point
- Value imputation
- Model imputation
- Mixture model

Live - Imputation

Fill up a feature not available at runtime with:

- median values of training data
- mean value of training data

} Continuous feature

- median values of training data
- majority value of training data

} Discrete feature

Live - Multi models

Learn mode with reduced features:

Normal model: $f(x_1, x_2, x_3, x_4)$

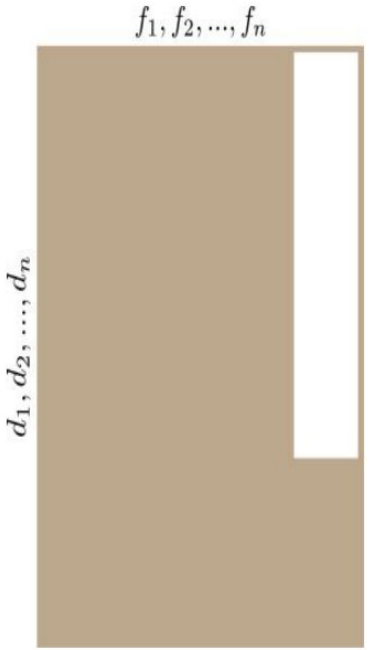
Reduced models: $f(x_2, x_3, x_4)$, $f(x_1, x_3, x_4)$, …
$f(x_3, x_4)$, $f(x_2, x_4)$, $f(x_2, x_3)$, …
$f(x_1)$, $f(x_2)$, …

Choose at runtime maximal model with available feature set

→ Huge overhead (2^n different models), but: best performance.

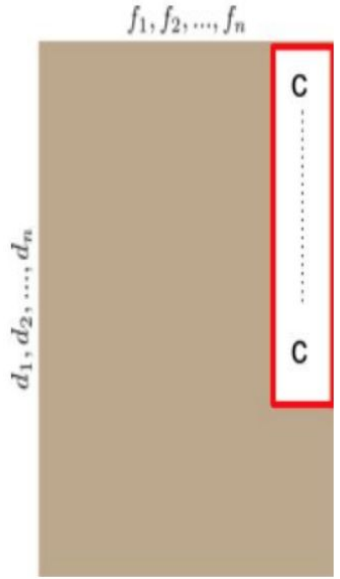
→ Tradeoff: Only learn reduced models for often failing features.

Training - Drop data



dropping incomplete features
→ drastically reduces data

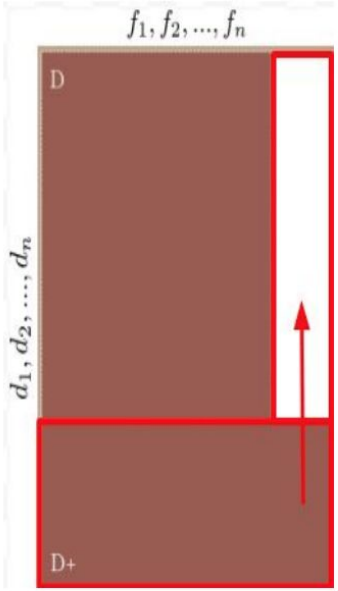
Training - Value imputation



Value imputation:

- mean
- median
- value out of definition scope

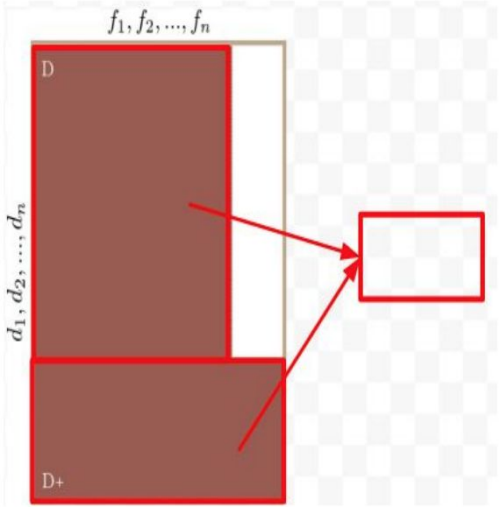
Training - Model imputation



Model imputation:

- train model on complete data points
- target variable is the missing feature
- predict missing values

Training - Mixture models



Mixture models:

- 2 submodels:
  1. complete data points
  2. reduced feature set
- use their output as input for decision model

Task

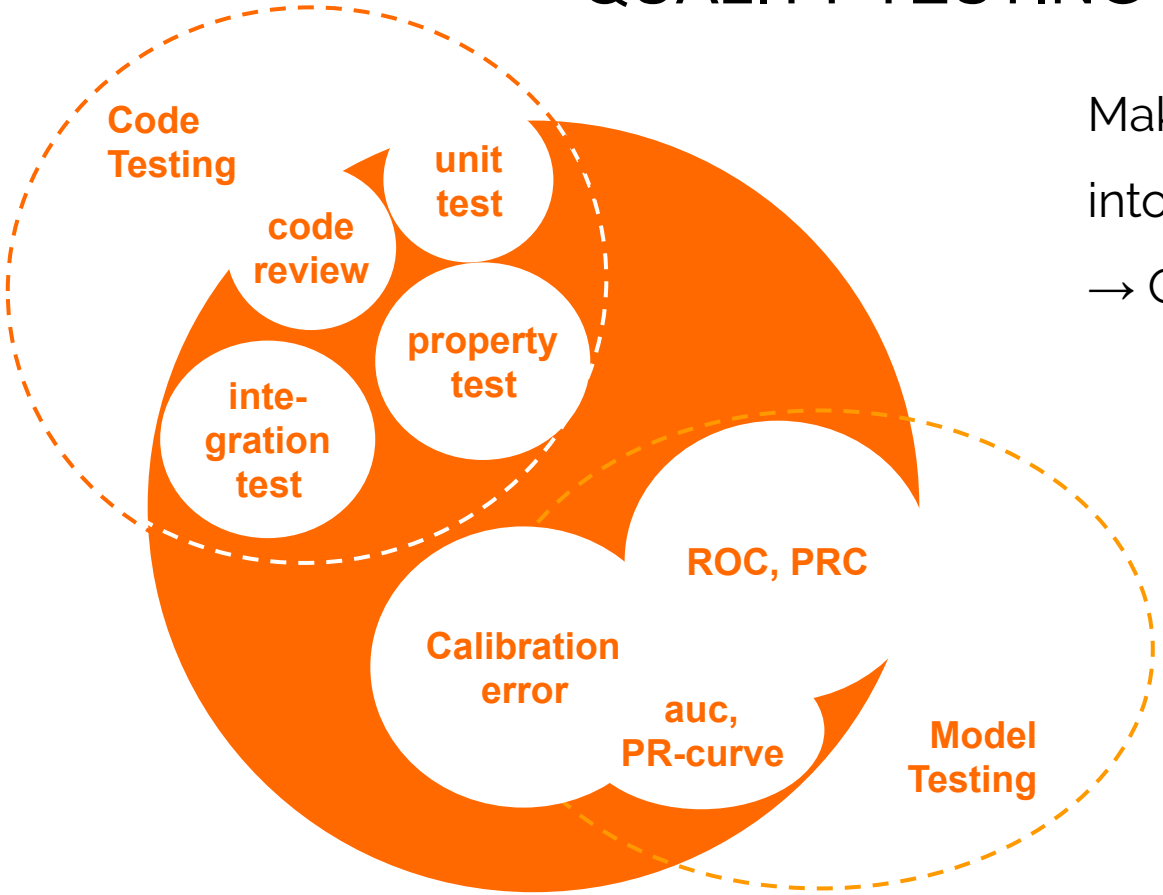
1. Extend the web server to be able to deal with missing features:
   - Create an imputer that is built on top of the training data.
   - Use this imputer in the flask server.
2. Run the request simulation tool again.

3. Bonus: Implement a `/metrics` endpoint in Flask that returns how often each feature failed in the last 100 requests.

Note: Do not hard-code the imputation values in the flask server (use an dict-object that is loaded on startup).

Monitoring

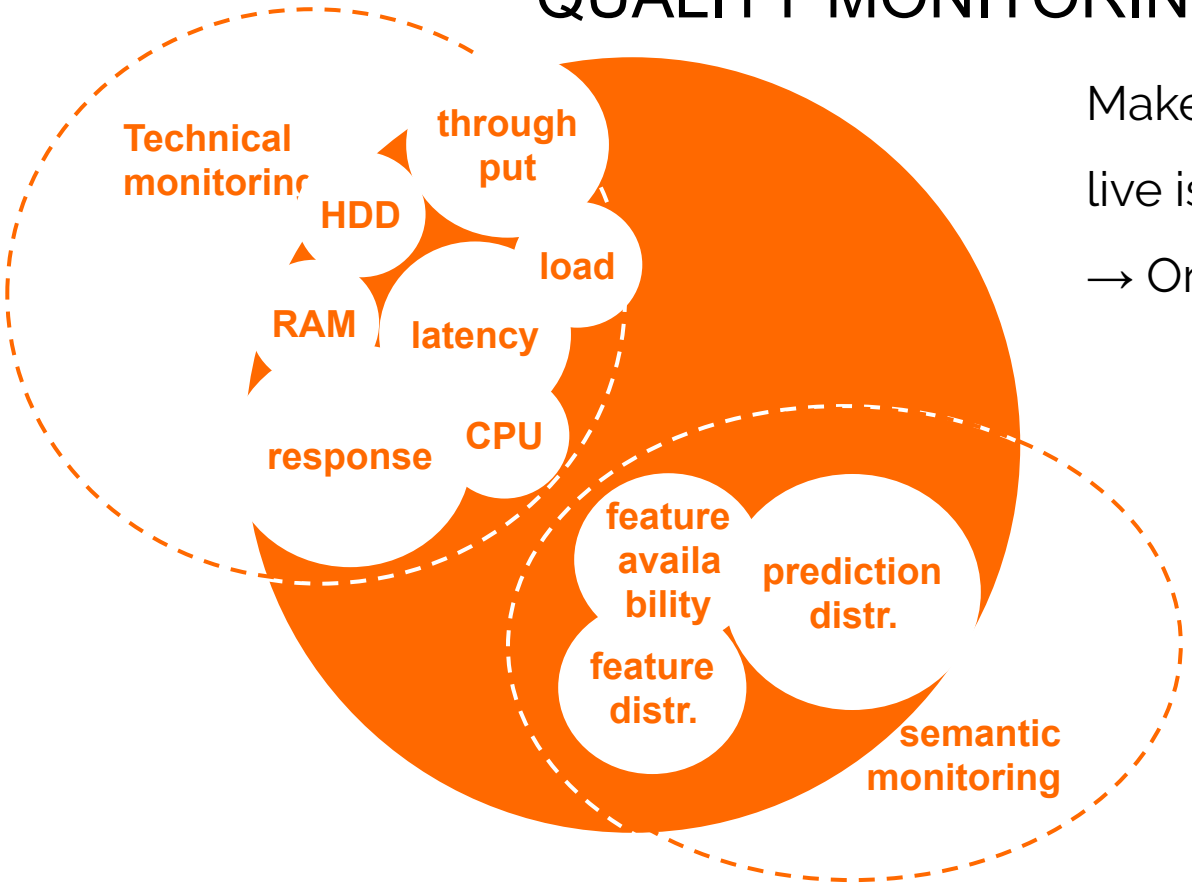
QUALITY TESTING



Make sure that what goes into production is good.

→ Offline world

QUALITY MONITORING



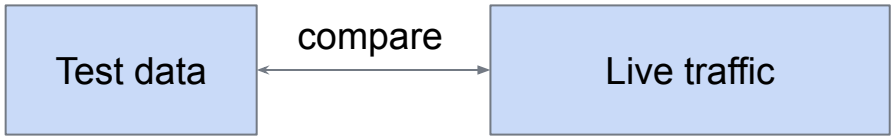
Make sure everything that runs live is working as expected

→ Online world

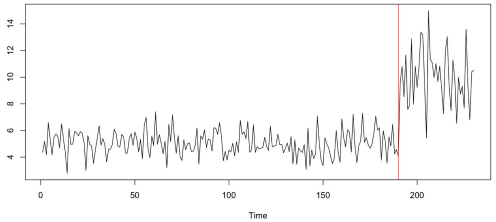
QUALITY MONITORING

There are two stages for model monitoring:

1. Check if model is working when it goes live.



2. Check if model is working continuously in live.



Monitor model with change point detection

QUALITY MONITORING

1. Check if model is working before it goes live

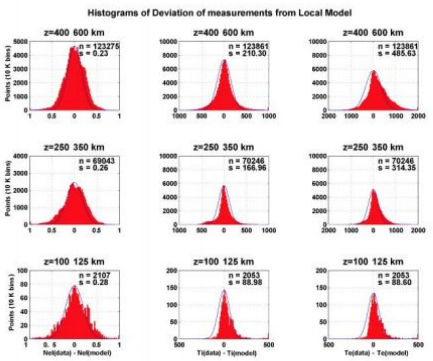
   Compare live data distribution to test data:
   - Failing features
   - Feature distribution
   - Probability distribution

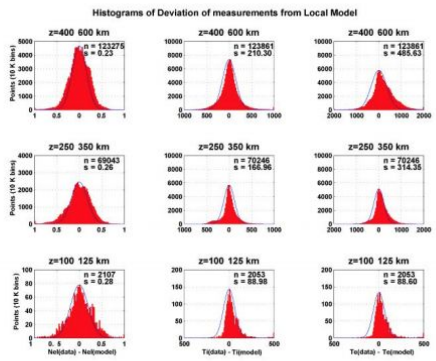
   When everything looks good, put model live.

QUALITY MONITORING

Compare feature distributions and output probability:

Feature distribution on test data

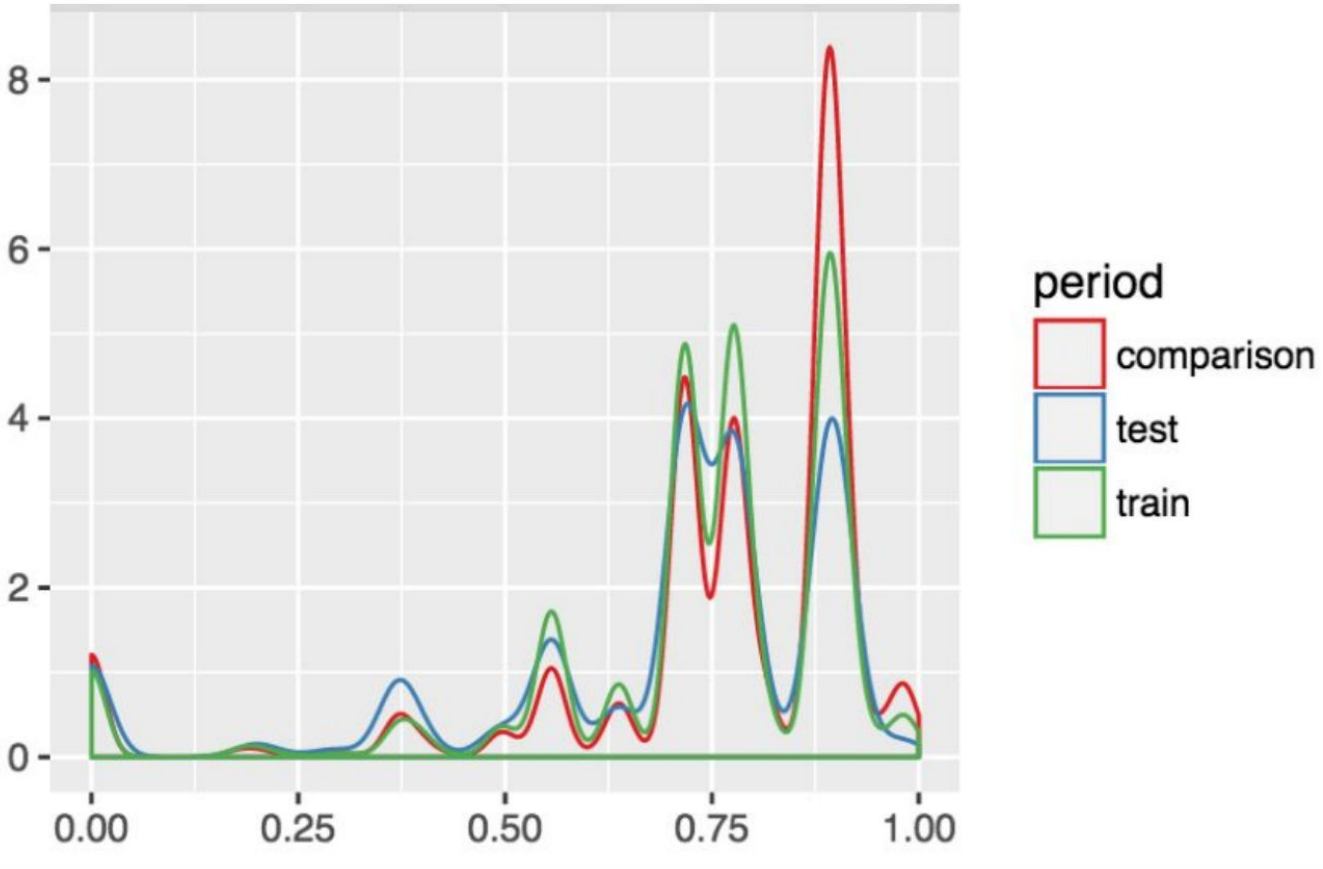




Feature distribution on live data

Quality Monitor

QUALITY MONITORING



QUALITY MONITORING

Compare distributions with some distance:

feature name	**this vs previous**	**this vs test**	**previous vs test**
feature one	0.000008	0.928806	0.928798
feature two	0.0009117	0.019504	0.020416
feature three	0.1075305	0.316970	0.313337
feature four	0.943896	0.943655	0.045654
...	...	...	...
prediction	6.606939e-02	0.255182	0.277325

QUALITY MONITORING

possible discoveries

technical problems,
change of behaviour (= date change)

Distribution Distance

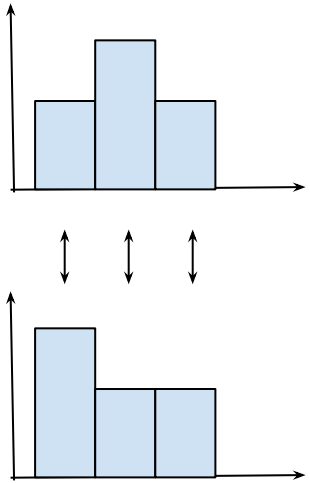
How do we measure the distance between two distributions?
Answer: There are statistical test for that!

1. Kolmogorow-Smirnow-Test
2. Kullback–Leibler divergence
3. Wasserstein distance

Distribution Distance

We can also use a simple distance measure:

- Build a normalized histogram (divide each bucket by number of values)
- Count differences for each bucket



Task

Let's build such a system!

1. Integrate a logging mechanism into the flask server:
   Write every request and the predicted probability to a logfile.

2. Test the logging with the loadSimulator. Increase the request speed of Line 59: `time.sleep(0.01)`

Task

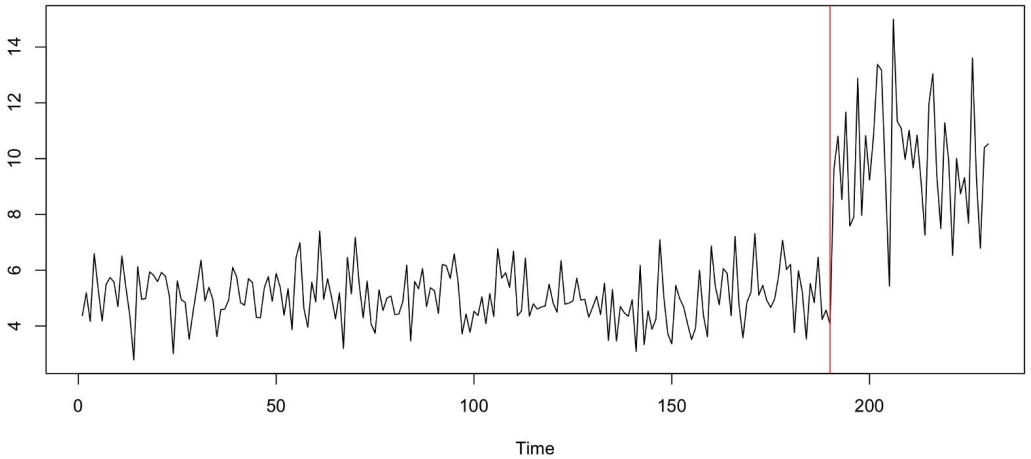
3. Built on a monitoring system on top of these logs.
   Requirements:.
   - Give out the share of missing features for every feature.
   - Compare the distribution of all features after 500 requests against the test data.
   - Give an alarm if it changes dramatically

QUALITY MONITORING

2. Check if model is working continuously in live.

→ Change point detection

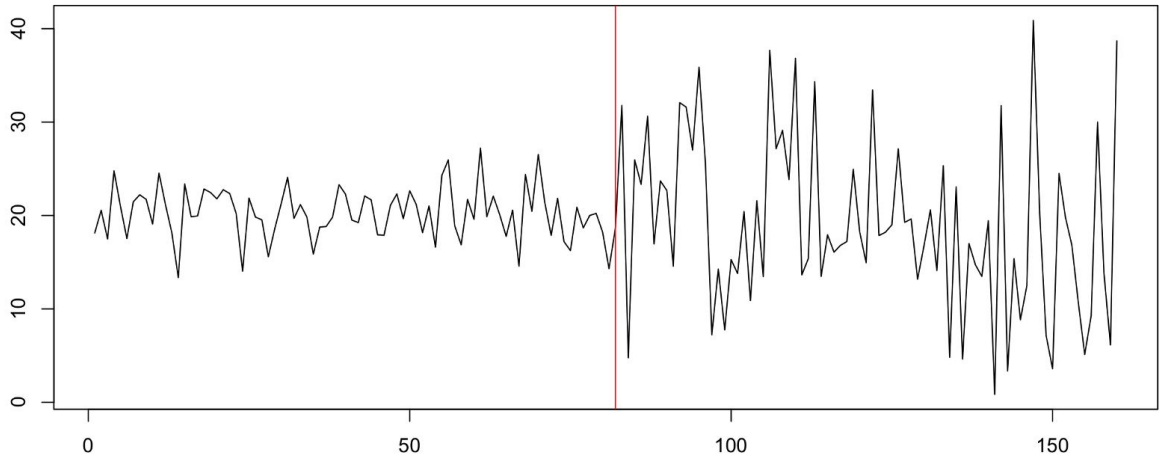
Idea: Check if a time series changes is behaviour at some point



Change in **mean** of time series

QUALITY MONITORING

Idea: Check if a time series changes is behaviour at some point



Change in **variance** of time series

QUALITY MONITORING

For every feature, we want to detect if such a thing happened.

There is a lot of research about this topic going on:

Reeves, Jaxk, et al. "A review and comparison of changepoint detection techniques for climate data." Journal of Applied Meteorology and Climatology 46.6 (2007): 900–915.

Task

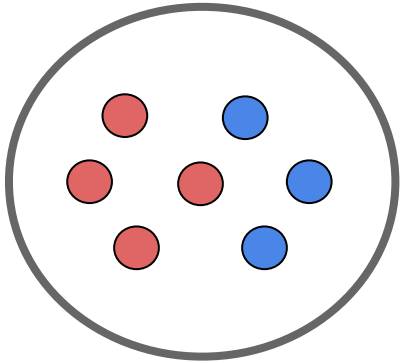
4. Implement a change point detector (use the provided amount.log):
   - The algorithm should work in an online-fashion (consuming one data point after another).
   - Use a sliding window should of size 100.
   - The alert should pop up if the mean does a major shift. (There is only one in the data)

Data Science Organisation

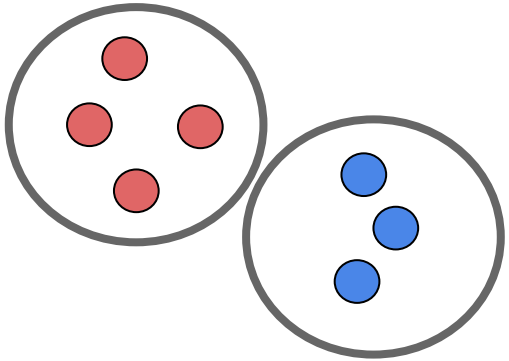
Disclaimer:
Unlike in more mature fields like Software Engineering (SE), there is no state-of-the-art DS organization form. Expect that people do not know how to handle your “ivory tower team” and try to apply the same methods as for SE. You need to educate them that DS is quite different.

DS Team Structure

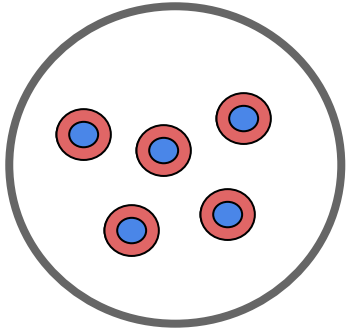
Mixed Team



Separate Team



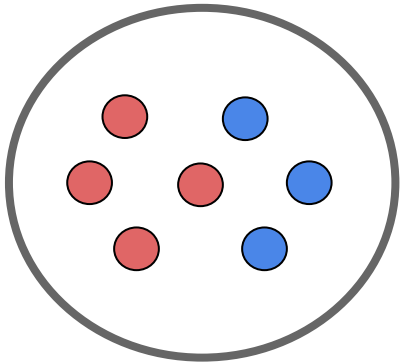
Hybrid Team



Data Scientist

Engineer

Mixed Team

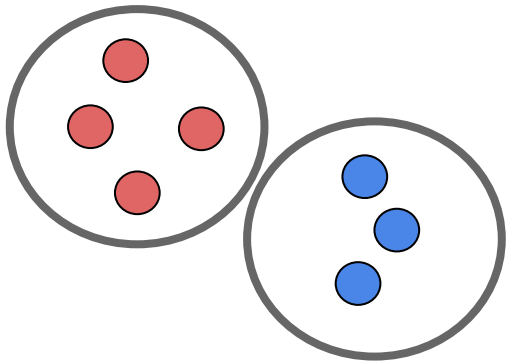


Data Scientist

Engineer

- All work together
- Engineers and DSs are equal
- Engineers:
  - Deployment
  - Automation
- Data Scientists:
  - Data Analysis
  - Experiments
- Both: Data Engineering

Separate Team

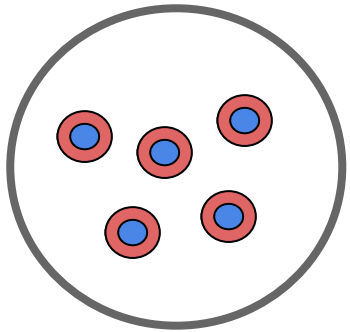


Data Scientist

Engineer

- “Throw over the fence” setup
- Working climate is often very difficult:
  - DSs see Engineers as code monkeys.
  - Engineers sees DSs as unproductive ivory tower people.
- If you work in such a setup: Make sure to talk and socialize with the engineers (show interest in their topic, go for lunch etc…)

Hybrid Team



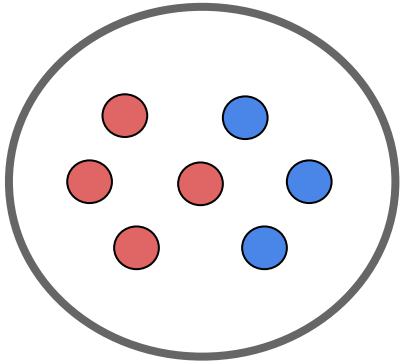
Data Scientist

Engineer

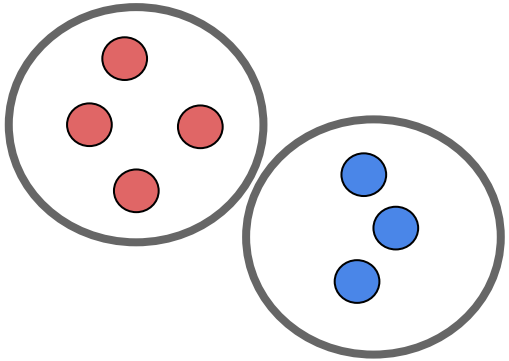
- People in this team do both: Engineering and DS.
- Often found in small start-ups.
- Gives you the chance to learn a lot about running a production system (a very valuable asset that will differentiate you from most other DSs).
- But: Make sure that you have still enough DS time as business grows.

DS Team Structure

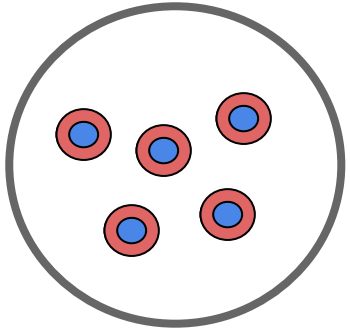
Mixed Team



Separate Team



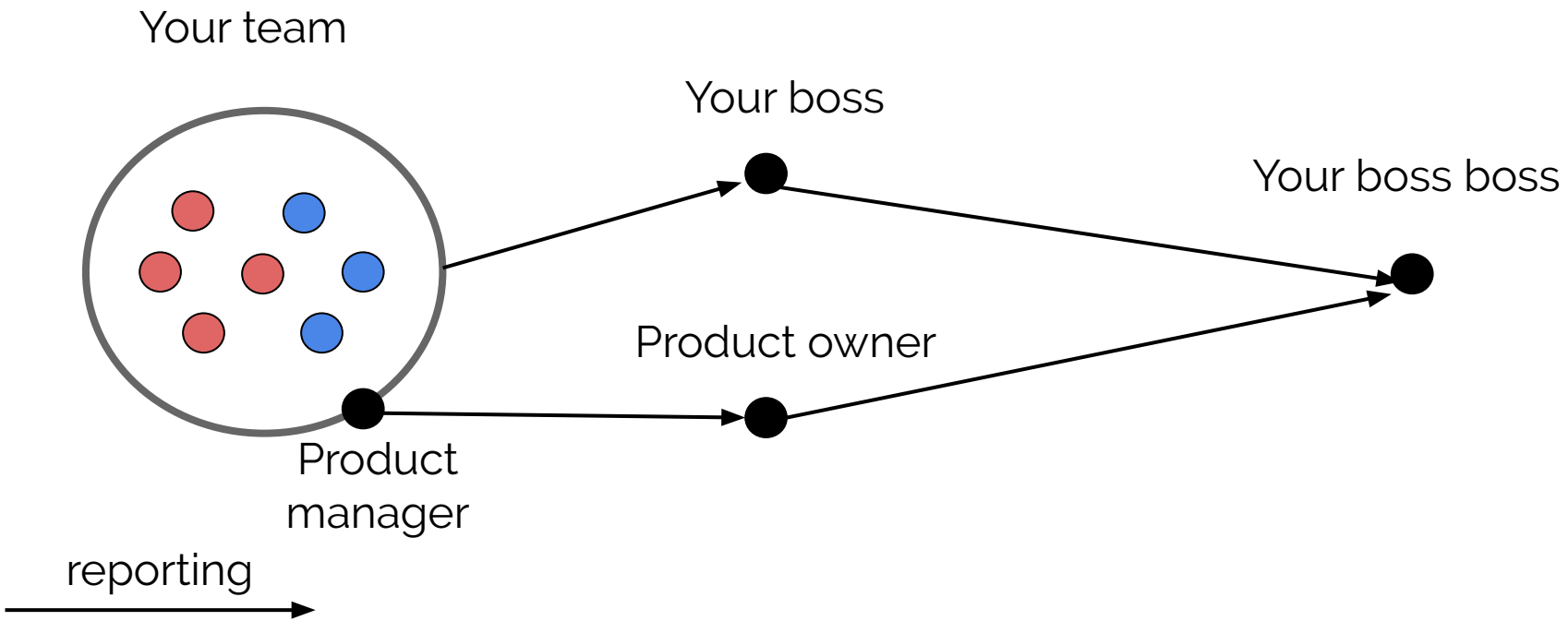
Hybrid Team



Data Scientist

Engineer

Your Stakeholders



Stakeholders - Product

Product manager:

- Maybe within your team or not, maybe sitting with you or not.
- Breaks down requirements from product owner (tries to tell you what. you do, may creates tickets, may want to know what you do).

Product owner:

- Creates the product vision and communicates it.
- Manages external stakeholders for the product.

AI Product Management

How should PMs and AI teams work together? Here's one default split of responsibilities:

Product Manager (PM) responsibility

- Provide dev/test sets, ideally drawn from same distribution.
- Provide evaluation metric for learning algorithm (accuracy, F1, etc.)

This is a way for the PM to express what ML task they think will make the biggest difference to users.

AI Scientist/Engineer responsibility

- Acquire training data
- Develop system that does well according to the provided metric on the dev/test data.

Stakeholders - Bosses

Your boss:

- Coordinates your work and manages you (people management).
- Creates technical vision and talks a lot to product owner.

Your boss boss:

- Manages your boss and the product owner and typically has not much to do with technical details.
- Measures outcome in dollars.

Stakeholders - You

Your team:

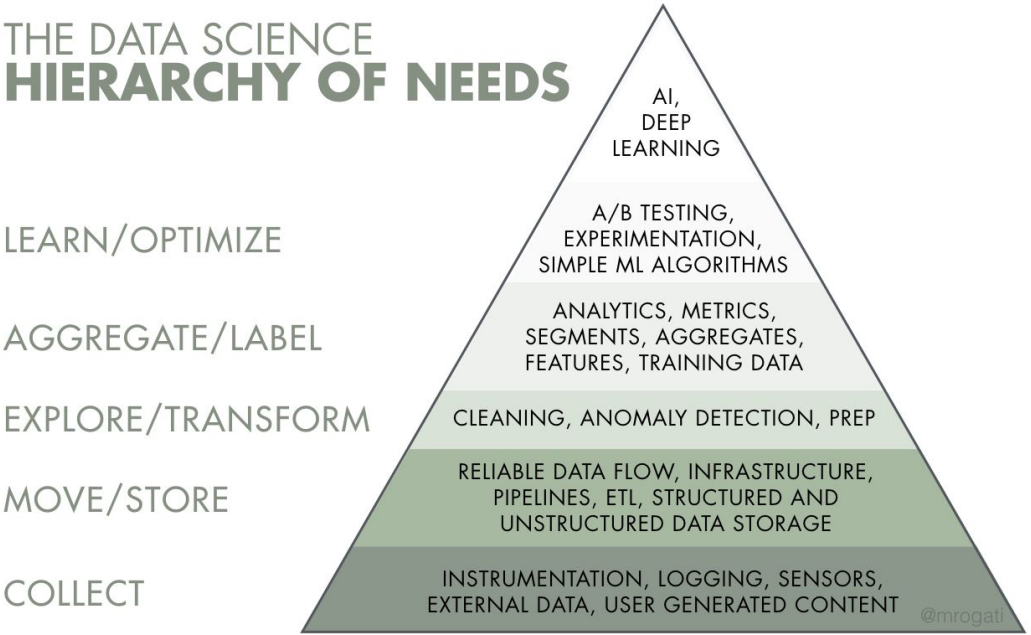
- Need to do the product work (could be defined more or less sharp).
  - Probably stuff like: relearn model, do analysis, boring stuff
  - This can also include research (but needs to be tracked as well).
- But: You can (and should) create proposals:
  - Come up with an idea how to improve things (new algorithms, new features, …)
  - Show clearly the business value of this (in terms of dollars!)
  - Help your boss to sell this to the product world

Data Science Manifesto

A set of rules of best practices for Data Science in the wild:

http://www.datasciencemanifesto.org/

The AI Hierarchy of Needs



https://hackernoon.com/the-ai-hierarchy-of-needs-18f111fcc007

What is your ML test score

Let's go through this paper and see what we covered in this workshop:

```
@inproceedings{45742,
title = {What's your ML test score? A rubric for ML production systems},
author  = {Eric Breck and Shanqing Cai and Eric Nielsen and Michael
Salib and D. Sculley},
year  = {2016}
}
```

Useful links about practical DS

- What is the Team Data Science Process?
  https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/overview

- Applying devops to data science
  http://www.hyperbi.co.uk/applying-devops-to-data-science/

- Andrew NG book about DS in practice (http://www.mlyearning.org)