Jupyter Notebooks in der Lehre und Forschung

WAW Software Engineering VI:
Software Engineering für Data Science,
Jena, 14.-15. Mai 2019

Daniel Speicher (dsp@bit.uni-bonn.de)
Bonn-Aachen International Center for Information Technology, Universität Bonn
Message: Quality for Jupyter Notebooks is different but possible

• Jupyter notebooks combine text, code and results.
  • “Calculation as a linear narrative”
  • Exploration, Explanation, Exercises

• The surprise of the Software Engineer:
  • Global variables, top level statements, few functions, less objects, no information hiding.

• Note on code quality of Jupyter notebooks
  • Communicative code
  • Design Pattern
  • Let’s continue the conversation ...

Acknowledgments: This material was prepared within the project P3ML which is funded by the Ministry of Education and Research of Germany (BMBF) under grant number 01/S17064. The authors gratefully acknowledge this support.
Observational basis for this talk


- Own notebooks at: [https://p3ml.github.io/](https://p3ml.github.io/)
  - Prototypical implementations for programming lab
  - Elaborating numerical recipes

- Students notebooks and our thorough review
- Notebooks of a course on Deep Learning (Coursera)
Jupyter Notebooks: Wow!

• A ``Jupyter Notebook is [a] web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.’’ (https://jupyter.org/)

• Consists of text and code cells.

• The content of code cells is sent on demand to a Python session, executed and the output inserted below the cell.
Jupyter Notebooks: Wow!

Text and executable code combined in one document.
Jupyter Notebooks: Wow!

### Lagrangian Dual of the Minimum Enclosing Ball Problem

For efficiency we calculate two constants upfront:

\[ X^T X \text{ and } Z = (x_j^T x_j)_{j=1,...,n} \]

```python
def init_ingredients(X):
    XtX = X.T.dot(X)
    return XtX, np.diag(XtX)
```

The Lagrangian Dual of the Minimum Enclosing Ball Problem and its negative gradient:

\[ D(\mu) = \mu^T Z - \mu^T X^T X \mu \]

\[ \nabla (-D(\mu)) = -\nabla D(\mu) = Z - 2X^T X \mu \]

```python
def lagrangian_dual(XtX, Z, mu):
    return mu.T.dot(Z) - mu.T.dot(XtX.dot(mu))

def neg_lagrangian_dual_gradient(XtX, Z, w):
    return -Z + 2 * XtX.dot(w)
```
Visualization of the Duality (3 Points only)

```
X = np.array([[[0.2, 0.4, 1.2],
               [0.2, 0.9, 0.1]]])
XtX, Z = init_ingredients(X)
mus = np.array([[[0.98, 0.01, 0.01], [0.5, 0.5, 0], [1/3, 1/3, 1/3], [0.186, 0.367, 0.447]]])
Ls = values_over_simplex(partial(lagrangian_dual, XtX, Z))

axs = init_plots_3D(len(mus), azim=125, elev=15)
for a, mu in zip(axs, mus):
    plot_flat_simplex(a)
    plot_over_simplex(a, Ls)
    plot_point_over_simplex(a, [mu[0], mu[1], lagrangian_dual(XtX, Z, mu)])
    done_plots()

axs = init_plots(len(mus), lims=[0, 1.5, 0, 1.1])
for a, mu in zip(axs, mus):
    a.set_title('\$\text{mus} = ({}))\$'.format(','.join(['{:2.2f}'.format(m) for m in mu]))
    plot_points(a, X)
    plot_circle(a, center(X, mu), radius(XtX, Z, mu))
    done_plots()
```

You may create visualizations.
Here: Duality of finding a minimum enclosing ball and a convex optimization problem.

The search for a minimum enclosing ball around three points is dual to the search for weights $\mu = (\mu_1, \mu_2, \mu_3)$ that maximize $D(\mu) = \mu^T z - \mu^T X^T X \mu$. 

© D. Speicher, Licensed under a CC BY-NC 4.0.
We are on another Planet

A calculation presented as a linear narrative

Global variables
Top level statements
Few functions (only in 37% of the notebooks)
Less objects (only in 12% of the notebooks)

No Information Hiding
Good, because we want to share our calculation.
Bad as far as the code is only a minor detail.
Another Caveat

A SIDE NOTE inspired by
Joel Grus: *I Don’t Like Notebooks*, JupyterCon 2018
Video: [https://youtu.be/7jiPelFXb6U](https://youtu.be/7jiPelFXb6U),
Slides: [https://twitter.com/joelgrus/status/1033035196428378113](https://twitter.com/joelgrus/status/1033035196428378113)
Jupyter Notebooks: Wow!

```python
one = 1
two = 2

one + one

2

one + one == two

True
```

The notebook shows results. Sometimes true ...
Jupyter Notebooks: Arrrrrgh!

The notebook shows results. Sometimes true ...

... sometimes not.
Jupyter Notebooks: Arrrrgh!

```
In [1]: one = 1
two = 2

In [2]: one + one
Out[2]: 2

In [3]: one + one == two
Out[3]: True

In [1]: one = 1
two = 2

In [2]: one + one
Out[2]: 2

In [4]: one + one == two
Out[4]: False
```
Jupyter Notebooks: Arrrrgh!

Just one of many ways how execution order might spoil results.
Jupyter Notebooks: Arrrrgh!

```python
jupiter = 'Jupiter is '
emphasis = ', '.join(7 * ['so']); jupiter += emphasis
jupiter += ' wonderful!
print(jupiter)
```
Jupyter Notebooks: Arrrrgh!

```python
jupiter = 'Jupiter is ' + ', '.join(7 * ['so']) + ' wonderful!
print(jupiter)
```

Jupiter is so, so, so, so, so, so, so, so wonderful!
Jupyter Notebooks: Arrrrgh!

```python
jupyter = 'Jupyter is '
emphasis = ','.join(7 * ['so']); jupiter += emphasis
jupyter += ' wonderful!
print(jupyter)
```
Jupyter Notebooks: Arrrrgh!

```python
jupyter = 'Jupyter is '
emphasis = ', '.join(7 * ['so']); jupyter += emphasis
jupyter += ' wonderful!
print(jupyter)
```

```
Jupyter is wonderful!
```
Jupyter Notebooks: Arrrrgh!

```python
jupyter = 'Jupyter is ' emphasis = ', '.join(7 * ['so']); jupyter += emphasis jupyter += ' wonderful!' print(jupyter)
```

```
Jupyter is wonderful!
```

```
print(jupyter)
```

```
Jupiter is so, so, so, so, so, so, so, so wonderful!so, so, so, so, so, so, so, so
```
Out of sync:
Mind – Notebook – Python Kernel

(missed one case)
jupyter

(jupyter (not initialized above))
jupyter

(jupyter (was initialized before))
jupyter
We are on another Planet

• Manual Execution Order Matters

• Partial renaming “refactoring”
  => Old variable with old state still in the process
  => Silent errors or difficult to find errors

• Developer needs to maintain a mental model of the state of the calculation.
Quotes on Communicative Code

“[O]ur intellectual powers are rather geared to master static relations and [...] our powers to visualize processes evolving in time are relatively poorly developed. For that reason we should do [...] our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.”

Edsger W. Dijkstra, Letters to the Editor: Go To Statement Considered Harmful, 1968
Practice: Use „Find and Replace“

... or even an IDE that offer debugging and refactoring.
More Helpful Practices

• Restart the kernel and rerun the calculation regularly
  Resyncs notebook and Python kernel
• Use assertions to verify implicit assumptions
  Resyncs mind, notebook and Python kernel

• For reusable code:
  • You may prototype in a notebook.
  • You must test your code.
  • Probably better extracted into regular code soon.

END OF SIDE NOTE
Communicative Code

A slightly adapted account on:
Late Imports – Universal Language – Identifier Length
Quotes on Communicative Code

“A good code should read like a story, not like a puzzle.”

Venkat Subramaniam, 2018
Late imports

Accompanying notebook for the recipe:

- Abstract of the paper: In this note, we study least squares optimization for parameter estimation. By means of the basic example of a linear regression task, we explore different formulations of the ordinary least squares problem, show how to solve it using NumPy or SciPy, and provide suggestions for practical applications.

In this notebook you find now in addition to the technical report as lsg_solution_Y0(X, Y) a direct implementation of the formula \((X'X)^{-1}X'Y\) based on numpy matrices reading \((X'X)^{-1}X'Y\). It comes with a slight performance penalty, especially for small matrices: 200% for \(n = 100\), but only 3% for \(n = 100000\). \(X\) and \(Y\) in this case have to be instantiated with np.matrix(...) not np.array(...).

As an experiment we write \(\hat{y}\) (Unicode 0177) instead of \(\hat{\text{y}}\). This is not yet a recommendation. Try for yourself. There are different strongly developed mathematical formula competes with the difference.

```python
import numpy as np
import numpy.linalg as la
import numpy.random as rnd
import matplotlib.pyplot as plt
```
As an experiment we write \( y = \beta_0 \) instead of \( y = \beta_0 + \varepsilon \). This is not yet a recommendation. Try for yourself. There are different strongly diverging options on this topic on the net. The benefit of close resemblance to the mathematical formula competes with the difficulty of typing the character.

```python
In [1]:
import numpy as np
import numpy.linalg as la
import numpy.random as rnd
import matplotlib.pyplot as plt

In [2]:
import warnings
warnings.simplefilter('ignore', category=FutureWarning)

Example data

In [3]:
def create_data(n, xmin=-2, xmax=2, a=1.1, b=2.0):
    x = np.random.random(n) * (xmax - xmin) + xmin
```

... much further down ...

```
plt.show()
```

Performance comparison

```python
In [12]:
import timeit

for i in range(1, 5):
    for v in [0, 1, 2, 3]:
        print(f'\( \beta_0 = \{v\} \)', end='')
    print()
```

© D. Speicher, Licensed under a CC BY-NC 4.0.
Late imports

Performance comparison

```python
import timeit

print(9*' ' + 'n', end='')
for v in [0, 1, 2, 3]:
    print(8*' ' + 'V{}'.format(v), end='')
print()

for n in [100, 1_000, 10_000, 100_000, 1_000_000, 10_000_000, 100_000_000, 1_000_000_000]:
    print('{:10}'.format(n), end=')
    for v in [0, 1, 2, 3]:
        if v == 0:
            setup = 'x, y = create_data(n); X = np.matrix(np.vander(x, 2)); y = np.matrix(y).T'
        else:
            setup = 'x, y = create_data(n); X = np.vander(x, 2)'

        t = timeit.timeit(
            stmt = 'lsq_solution_V{}(X, y)'.format(v),
            setup = setup,
            number = 100,
            globals = globals()
        )

        print('{:10.3f}'.format(t), end='')
print()
```

<table>
<thead>
<tr>
<th>n</th>
<th>V0</th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.007</td>
<td>0.003</td>
<td>0.010</td>
<td>0.008</td>
</tr>
<tr>
<td>1000</td>
<td>0.004</td>
<td>0.002</td>
<td>0.009</td>
<td>0.008</td>
</tr>
</tbody>
</table>
Late imports

Performance comparison

```python
import timeit

print(9*' '+ 'n', end='')
for v in [0, 1, 2, 3]:
    print(8*' '+ 'V{}'.format(v), end='')
print()

for n in [100, 1_000, 10_000, 100_000]:
    print('{:10}'.format(n), end='')
    for v in [0, 1, 2, 3]:
        if v == 0:
            setup = 'x, y = create_data(n, v)
        else:
            setup = 'x, y = create_data(n, v)
        t = timeit.timeit(
            stmt = 'lsq_solution(x, y, n, v, v0, v1)
            setup = setup,
            number = 100,
            globals = globals()
        )
        print('{:10.3f}'.format(t), end='')
    print()
```

What does the coder want to tell us?

Suggestion: “This section covers a separate concern that I still want to share together with the rest of the notebook.”

Goals: Separate concerns – Share together – Know dependencies early
Universal Language: Code ~ Domain

• Statistics, Ordinary Least Squares solution:

\[ w = (X^T X)^{-1} X^T y \]

• Implementation:

```python
# X and y created with numpy.array(..)
w = np.dot(np.dot(la.inv(np.dot(X.T, X)), (X.T)), y)
```
Universal Language: Code ~ Domain

• Statistics, Ordinary Least Squares solution:

\[ w = (X^TX)^{-1}X^Ty \]

• Implementation:

```python
# X and y created with numpy.array(..)
w = la.inv(X.T.dot(X)).dot(X.T).dot(y)
```
Universal Language: Code ~ Domain

• Statistics, Ordinary Least Squares solution:

\[ w = (X^T X)^{-1} X^T y \]

• Implementation:

```python
# X and y created with numpy.matrix(..)
w = (X.T * X).I * X.T * y
```
Quotes on Communicative Code

“To communicate effectively, the code must be based on the same language used to write the requirements - the same language that the developers speak with each other and with domain experts.”

Eric Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software, 2003
Universal Language:
Code ~ Domain

• Statistics, Ordinary Least Squares solution:

\[ w = (X^T X)^{-1} X^T y \]

• Implementation:

```python
# X and y created with numpy.array(..)
w = la.inv(X.T @ X) @ X.T @ y
```

As numpy arrays behave differently than numpy matrices some recommend for consistency using @ for matrix multiplication of arrays. Compare:

```python
a = np.array([[1, 2], [2, 1]])
a ** 2
array([[1, 4],
       [4, 1]])
```

```python
m = np.matrix([[1, 2], [2, 1]])
m ** 2
matrix([[5, 4],
        [4, 5]])
```
Universal Language: Code ~ Domain

• Statistics, Ordinary Least Squares solution:

\[ w = (X^T X)^{-1} X^T y \]

• Implementation:

```python
# X and y created with numpy.array(..)
w, _, _, _ = la.lstsq(X, y)
```

Identifier Length

• **Shorter identifier names take longer to comprehend**
  (See [Hofmeister 2019] and related work)

• For longer identifiers:
  • Observation: Bugs are found faster.
  • Hypothesis: Identifier meaning easier recognized.

• **But,** in mathematical contexts there are some short identifiers that have well established meaning:

  Established short  >>  longer unfamiliar
Length has still its value

```python
def MacQueen(X, k):
    n = X.shape[0]
    M = np.copy(X[:k])
    N = np.ones(k)

    for j in range(k, n):
        i = np.argmin(np.sum((M - X[j])**2, axis=1))
        N[i] += 1
        M[i] += 1./N[i] * (X[j] - M[i])

    return M
```

```python
def MacQueen(points, k):
    means = np.copy(points[:k])
    sizes = np.ones(k)

    for point in points[k:]
        i = np.argmin(np.sum((means - point)**2, axis=1))
        sizes[i] += 1
        means[i] += 1./sizes[i] * (point - means[i])

    return means
```

Variables

- X
- k
- n
- M
- N
- j
- i
- points
- k
- means
- sizes
- point
- i
Design Patterns

Solutions to conflicting forces in the context of “calculations as a linear narrative”: Function Exemplification – Updated Progress Line – Visualization Callback
Design Patterns

• Solution to conflicting forces in a context

• See e.g. Section 1.1 in [Gamma 1995]

• The context of a calculation presented as a linear narrative leads to solutions that differ substantially from solutions for other kinds of software.
Function Exemplification – Forces

• Notebooks present code and its result in a linear sequence.

• Result of a function definition is a defined function and no immediate output.

• Self defined functions (let alone objects) are therefore used much less frequently in notebooks than in other software.

• Still, functions are helpful for internal reuse and to give structure to a longer calculation.
Function Exemplification – Solution

• **Illustrate the use** of the function in the next cell.
• (for functions without side effects, short runtime and easy to provide parameters)
Utility function guessing the schema from given JSON data

def print_json_schema(json, indentation=-1):
    if isinstance(json, dict):
        print()
        indentation += 1
        for key, value in json.items():
            print(indentation * ' ', key, end=': ')
            print_json_schema(value, indentation)
        indentation -= 1
        print()
    elif isinstance(json, list):
        length = len(json)
        if length:
            print(length, end='x ')
            print_json_schema(json[0], indentation)
        else:
            print('0x ???')
    else:
        print(type(json))

print_json_schema({'countries': ['Germany', 'France'], 'politicians': [{'first': 'Angela', 'last': 'Merkel'}, {}, {}])

countries: 2x <class 'str'>
politicians: 3x
    first: <class 'str'>
    last: <class 'str'>
Function Exemplification – Ex. 2

Utility functions for naming files and extracting text from HTML

```python
def deputy_file_name_part(deputy):
    return '_'.join([deputy['meta']['username'], deputy['party'].lower().replace(' ', '-')])

def question_file_name_parts(q, question):
    question_nr = 'Q{:04}'.format(q + 1) # Maximum in 12/18: 344 questions (Andrea Nahles)
    question_id = '_'.join([question_nr, question['date']])
    category = question['category'].lower().replace(' ', '-')
    return question_id, category

def answer_file_name_part(a, answer):
    answer_nr = 'A{:02}'.format(a + 1) # Maximum in 12/18: 2 answer for one question (often)
    return '_'.join([answer_nr, answer['date']])

print(deputy_file_name_part(deputy))

if questions:
    oldest = len(questions) - 1
    print(question_file_name_parts(oldest, questions[oldest]))

    answers = questions[oldest]['answers']
    if answers:
        print(answer_file_name_part(0, answers[0]))

ulrich-wolfgang-kelber_spd
('Q0037_2017-07-24', 'inneres-und-justiz')
A01_2017-07-25
```
def deputy_file_name_part(deputy):
    return '_'.join([deputy['meta']['username'], deputy['party'].lower().replace(' ', '-')])

print(deputy_file_name_part(deputy))

ulrich-wolfgang-kelber_spd
Function Exemplification – Ex. 2

def question_file_name_parts(q, question):
    question_nr = '{:04}'.format(q + 1)  # Maximum in 12/18: 344 questions (Andrea Nahles)
    question_id = '_'.join([question_nr, question['date']])
    category = question['category'].lower().replace(' ', '-')
    return question_id, category

if questions:
    oldest = len(questions) - 1
    print(question_file_name_parts(oldest, questions[oldest]))

('Q0037 2017-07-24', 'inneres-und-justiz')
def answer_file_name_part(a, answer):
    answer_nr = 'A{:02}'.format(a + 1)  # Maximum in 12/18: 2 answer for one question (often)
    return '_'.join([answer_nr, answer['date']])

answers = questions[oldest]['answers']
if answers:
    print(answer_file_name_part(0, answers[0]))

A01_2017-07-25
Updated Progress Line - Forces

Latent Dirichlet Allocation

We instantiate the LDA algorithm as follows:

```python
In [7]:
lda_start_time = time.perf_counter()
lda_algorithm = LatentDirichletAllocation(n_components=n, max_iter=200, learning_method='online', learning_offset=100., random_state=0)

In [8]:
lda_algorithm.fit_transform(word)

lda_end_time = time.perf_counter()
```

We instantiate the LDA algorithm and pass the configuration parameters: results after stopping for at least 100 iterations (in the case of LDA). The topic learning should be reduced to parameter tuning. In this section, we perform topic transformation based on the same, but throw away the topic within document or distribution. The latter would not even be used for previously unseen or

```
iteration: 1 of max_iter: 200
iteration: 2 of max_iter: 200
iteration: 3 of max_iter: 200
iteration: 4 of max_iter: 200
iteration: 5 of max_iter: 200
iteration: 6 of max_iter: 200
iteration: 7 of max_iter: 200
iteration: 8 of max_iter: 200
iteration: 9 of max_iter: 200
iteration: 10 of max_iter: 200
iteration: 11 of max_iter: 200
iteration: 12 of max_iter: 200
iteration: 13 of max_iter: 200
iteration: 14 of max_iter: 200
iteration: 15 of max_iter: 200
iteration: 16 of max_iter: 200
iteration: 17 of max_iter: 200
```
Updated Progress Line – Forces

- During exploration: Provide feedback.
- Later on: Have high information/space rate.
Updated Progress Line – Solution

• Let the calculation repeatedly overwrite only temporarily interesting progress information in the same line.

```python
print('Done: {} of {}.'.format(i, n), end='\r')
```
Updated Progress Line – Example

Create or update deputy files (JSON) and question files (URL)

```python
success = []
failure = []

for d, deputy in enumerate(deputies):
    deputy_prefix = deputy_file_name_part(deputy)
    deputy_file = corpus_dir / (deputy_prefix + '.json')

    try:
        if update_only_missing_deputies and deputy_file.exists(): continue

    except Exception as exception:
        failure.append((deputy_file.name, exception))

    finally:
        print(\'\r{} of {}. {} files successfully created. {} files failed. Latest: {}'.format(\n            d+1, len(deputies), len(success), len(failure), deputy_file.stem), end='')
```

716 of 716. 73 files successfully created. 0 files failed. Latest: gyde-jensen_fdp

© D. Speicher, Licensed under a CC BY-NC 4.0.
Visualization Callback – Forces

• Algorithm, implementation should be influenced by no other concerns

• We often want to show intermediate state of the algorithm.

• It is often interesting to visualize algorithms in varying detail and with respect to different aspects.

• Same implementation should be usable with or without visualization. (If it is not visualized it should be fast.)
Visualization Callback – Solution

• We pass a visualization function as a parameter to the function that implements the algorithm.

• Default value this parameter gets an anonymous function doing nothing

• The algorithm function calls the parameter function passing all potentially interesting information in.

• Visualization functions that actually show something may have additional parameters that can be ``frozen'' by creating a partial function.

• ~ Strategy + Null Object as Default Strategy
Visualization Callback – Ex. 1a

```python
def MacQueen(points, k, show=lambda state, i, sizes, means: None):
    means = np.copy(points[:k])
sizes = np.ones(k)

    for point in points[k:]:
        i = np.argmin(np.sum((means - point)**2, axis=1))
sizes[i] += 1
    means[i] += 1./sizes[i] * (point - means[i])

    return means
```

```python
def plot_some_macqueen_state(n_rows, n_cols, points, k,
                             state, i, sizes, means):
```

```python
plot_some = partial(plot_some_macqueen_state, 1, 6, points, 3)
means = MacQueen(points, 3, show=plot_some)
```

© D. Speicher, Licensed under a CC BY-NC 4.0.
Visualization Callback – Ex. 1a

```python
def MacQueen(points, k, show=lambda state, i, sizes, means: None):
    means = np.copy(points[:k])
    sizes = np.ones(k)
    for point in points[k:]:
        i = np.argmin(np.sum((means - point) ** 2, axis=1))
        sizes[i] += 1
        means[i] += 1./sizes[i] * (point - means[i])
    return means
```

Default: show nothing. Exemplifies signature.

Calls not part of the Gestalt of the algorithm

Visualization function with additional arguments

Partial function with "frozen" arguments

Call to the algorithm passing the visualization function

Progress visualization as “small multiples”
Visualization Callback – Ex. 1b

```python
def store_macqueen_state(k, cluster, means, state, j, i, N, M):
    k=3; cluster = list(range(k)); means = list()
    store_state = partial(store_macqueen_state, k, cluster, means)

M = MacQueen(X, k, show=store_state)
```

**Visualization function for storing state**

Call to the algorithm passing the visualization function

Actually visualize the stored state (clusters and history of means)

```
M = MacQueen(X, k, show=store_state)
```

Initialized. cluster = [0, 1, 2], means = []
Clustering 160 points. Cluster sizes: 47, 71, 42

```
plot_cluster_and_means(cluster, means)
```

© D. Speicher, Licensed under a CC BY-NC 4.0.
Further Recommendations:

• Volodymyr Kazantsev, Kateryna Nerush: *Clean Code in Jupyter notebooks, using Python*  
  PyData Berlin 2017  
  Video: [https://youtu.be/2QLgf2Yllus](https://youtu.be/2QLgf2Yllus)  
  Slides: [https://de.slideshare.net/katenerush/clean-code-in-jupyter-notebooks](https://de.slideshare.net/katenerush/clean-code-in-jupyter-notebooks)

• Joel Grus, Matt Gardner, Mark Neumann: *Writing Code for NLP Research*  
  EMNLP 2018 tutorial  
  Slides: [https://docs.google.com/presentation/d/17NoJY2SnC2UMbVegaRCWA70ca7UCZ3vHnMqBV4SUayc/](https://docs.google.com/presentation/d/17NoJY2SnC2UMbVegaRCWA70ca7UCZ3vHnMqBV4SUayc/)
Announcements of our Notebooks on ResearchGate

• Latent Dirichlet Allocation
  https://www.researchgate.net/project/P3ML-ML-Engineering-Knowledge/update/5c4f789c3843b0544e62df38

• Expectation Maximization for Gaussian Mixture Models
  https://www.researchgate.net/project/P3ML-ML-Engineering-Knowledge/update/5c61b19acfe4a781a57eea06

• Minimum Enclosing Balls
  https://www.researchgate.net/project/P3ML-ML-Engineering-Knowledge/update/5c73e079cfe4a781a58317e0

• List: https://p3ml.github.io/
• Jupyter notebooks combine text, code and results.

• Code quality guidelines need to be adapted for the context of “calculations as a linear narrative”. (M2)

• Searching for “solutions to conflicting forces in a context” is still a helpful practice. (M3)
  • Function Exemplification, Updated Progress Line, Visualization Callback
  • Be creative! Let’s share Jupyter notebook patterns!

• Usability Engineering & Software Engineering
Vielen Dank für Ihre Aufmerksamkeit
Code Quality Cultivation

M0
Definition of the functionality of the system...
...as seen from the outside.

M1
Tests
...as realized inside the system.

M2
Code quality knowledge...
...illustrated by example.
...defined as executable rules.

Bad Smell Detection raises too many false alarms
=>
Rules to analyze code quality belong into the hands of developers