

Supervised and Unsupervised Machine Learning for Bio-image Analysis

Robert Haase

Reusing materials from Johannes Soltwedel, Till Korten, Johannes Müller, Laura Žigutytė (TU Dresden), Ryan Savill (MPI-CBG), Matthias Täschner (ScaDS.AI/Uni Leipzig) and the Scikit-learn community.

Funded by



Bundesministerium
für Bildung
und Forschung

SACHSEN



Diese Maßnahme wird gefördert durch die Bundesregierung aufgrund eines Beschlusses des Deutschen Bundestages. Diese Maßnahme wird mitfinanziert durch Steuermittel auf der Grundlage des von den Abgeordneten des Sächsischen Landtags beschlossenen Haushaltes.

Follow-up: cupy

cupy was hard to make work. Consider using a GPU-runtime in Google Colab

Install packages

Select GPU runtime

```
try:
    import cupy
except:
    import numpy as np

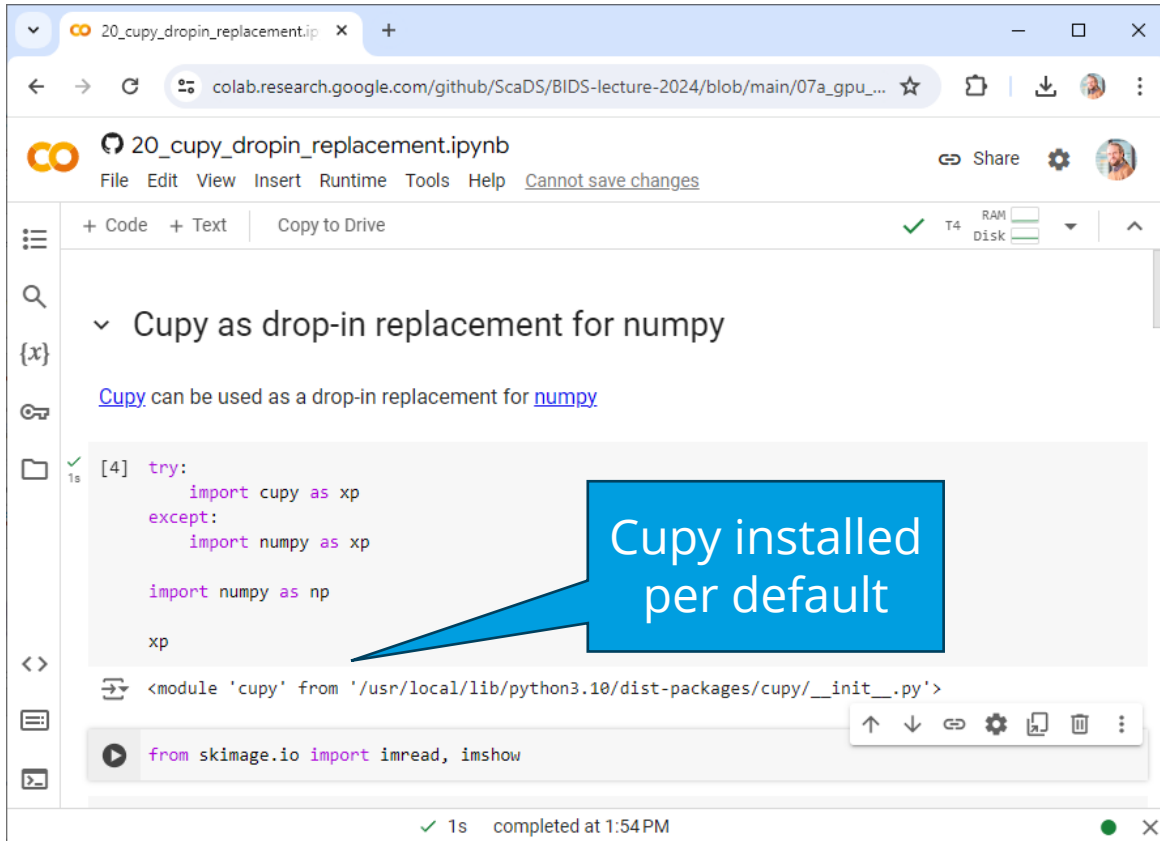
import numpy as np
```

```
!pip install stackview
```

```
Collecting jedi<=0.16 (from ipython>=4.0.0->ipywidgets->stackview)
  Downloading jedi-0.19.1-py2.py3-none-any.whl (1.6 MB)
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from ipython)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from ipyth
Requirement already satisfied: prompt-toolkit!=3.0.0,!<3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/pyt
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from ipython)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from ipython)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (from ipyth
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-d
Requirement already satisfied: notebook>=4.4.1 in /usr/local/lib/python3.10/dist-packages (from w
Requirement already satisfied: jupyter-core>=4.6.0 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.10/dist-packages (from jupyter
✓ Connected to Python 3 Google Compute Engine backend (GPU)
```

Follow-up: cupy

cupy was hard to make work. Consider using a GPU-kernel in Google Colab



```
[4] try:
    import cupy as xp
except:
    import numpy as xp

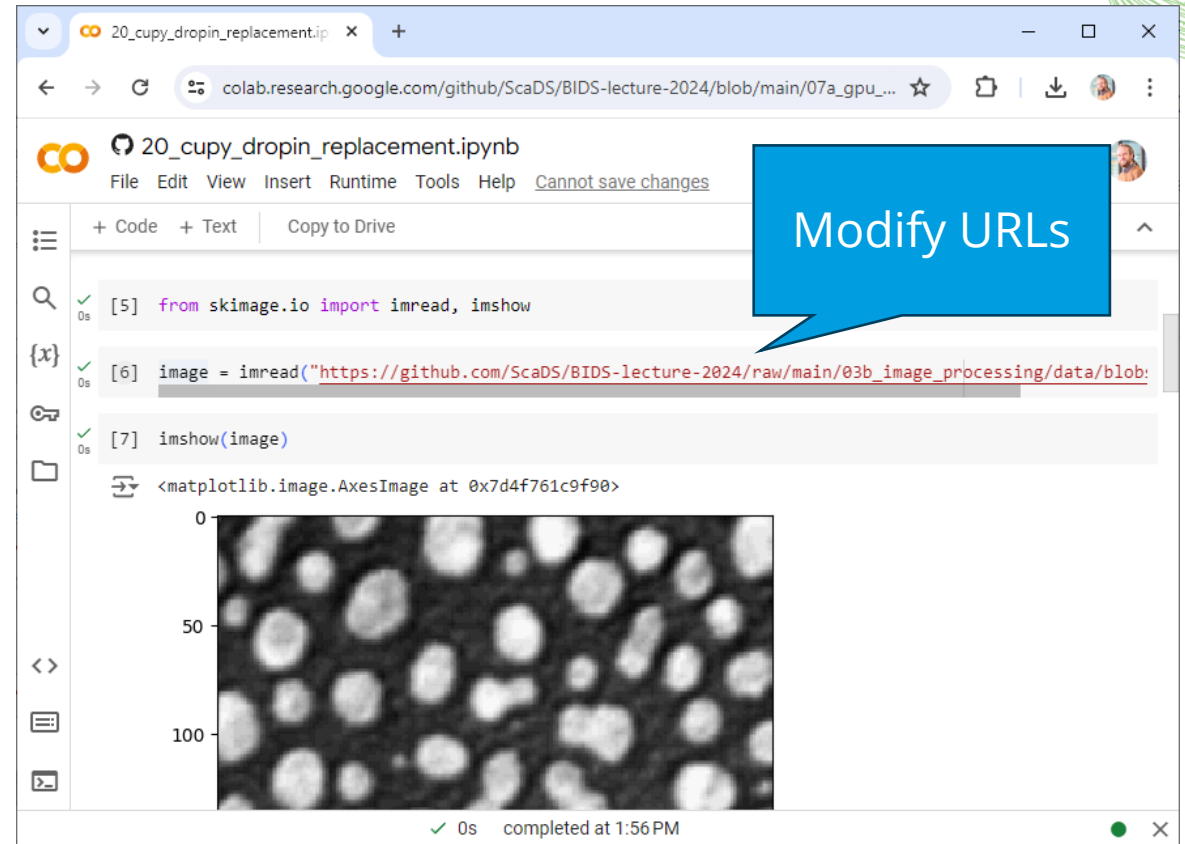
import numpy as np

xp
```

<module 'cupy' from '/usr/local/lib/python3.10/dist-packages/cupy/__init__.py'>

```
from skimage.io import imread, imshow
```

✓ 1s completed at 1:54 PM

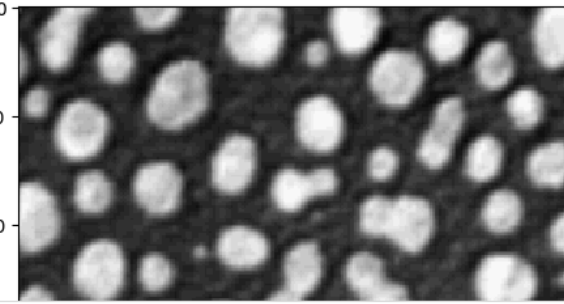


```
[5] from skimage.io import imread, imshow

[6] image = imread("https://github.com/ScaDS/BIDS-lecture-2024/raw/main/03b_image_processing/data/blob:

[7] imshow(image)
```

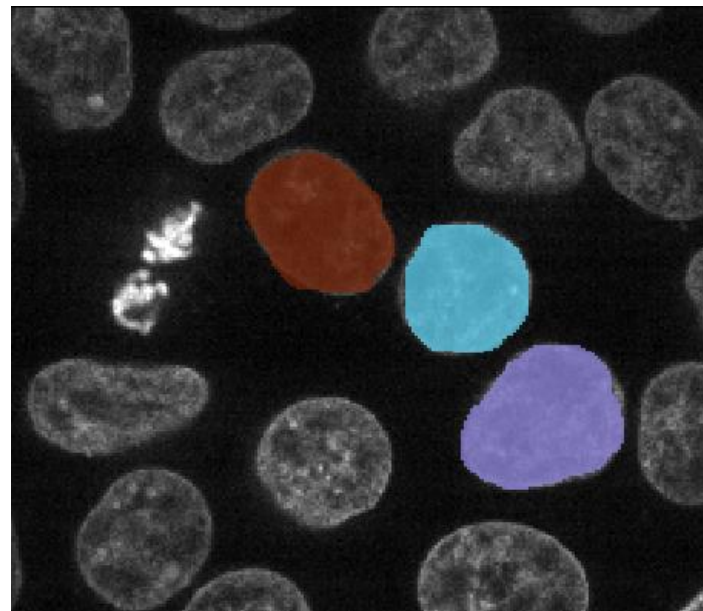
<matplotlib.image.AxesImage at 0x7d4f761c9f90>



✓ 0s completed at 1:56 PM

Quiz: Recap

What kind of label image is this?



Instance segmentation



Semantic segmentation



Sparse instance segmentation



Sparse semantic segmentation



Terminology

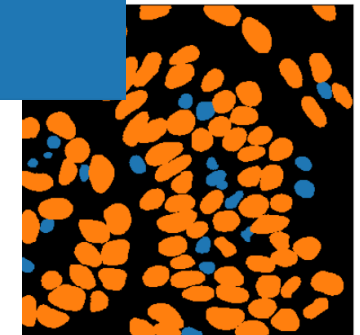
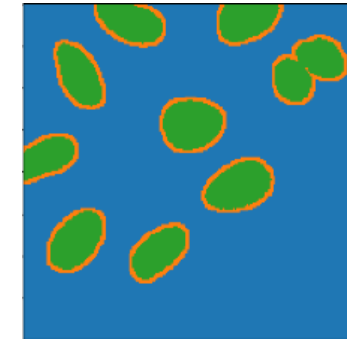
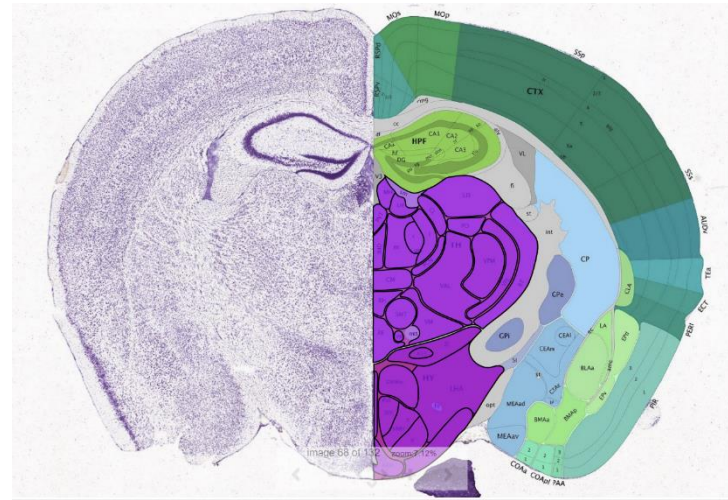
Instance segmentation



Instances:

- Cells, nuclei, cats, dogs, cars, trees

Semantic segmentation

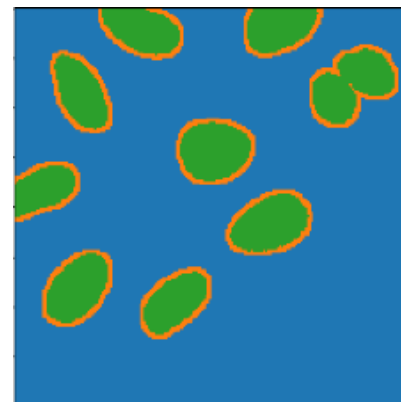


Regions:

- Anatomical, geographical
- All pixels belonging to the same type of object have the same value

Terminology

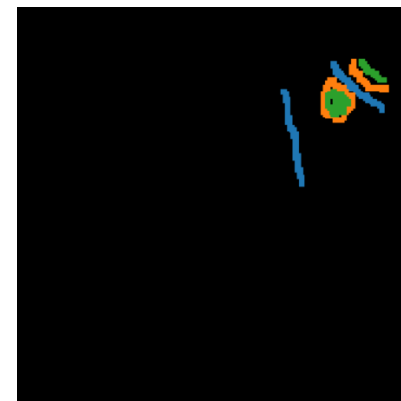
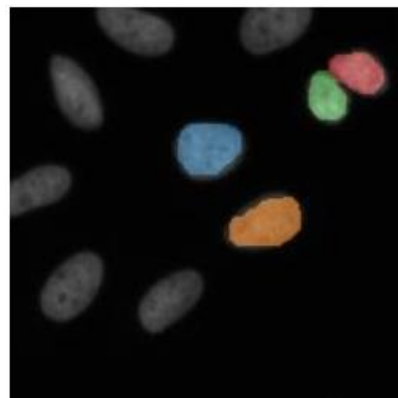
Instance segmentation



Semantic segmentation

Annotations are typically drawn by humans (e.g. to train machine learning models)

Sparse instance annotation



Sparse semantic annotation

Image segmentation using thresholding

Recap: Finding the right workflow towards a good segmentation takes time

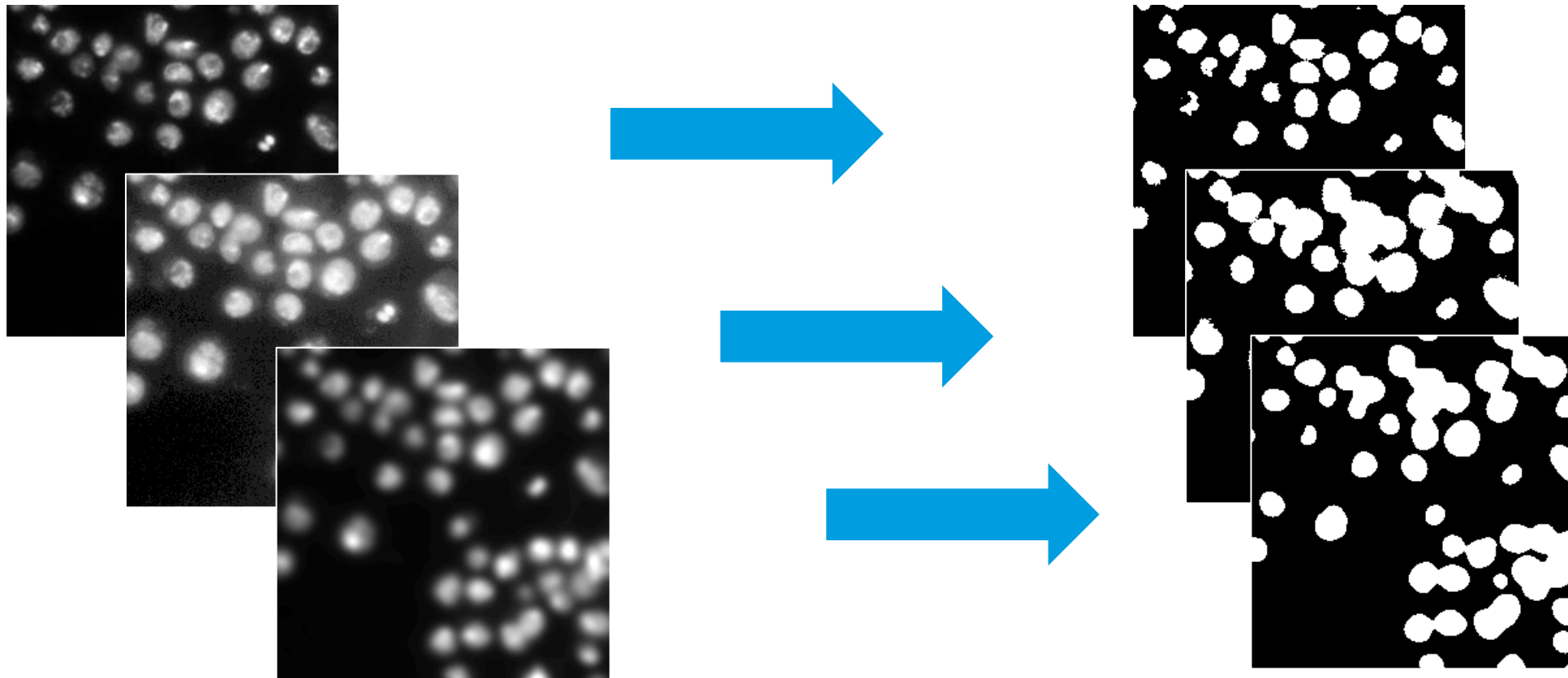


Image segmentation using thresholding

Recap: Combining images, e.g. using Difference of Gaussian (DoG)

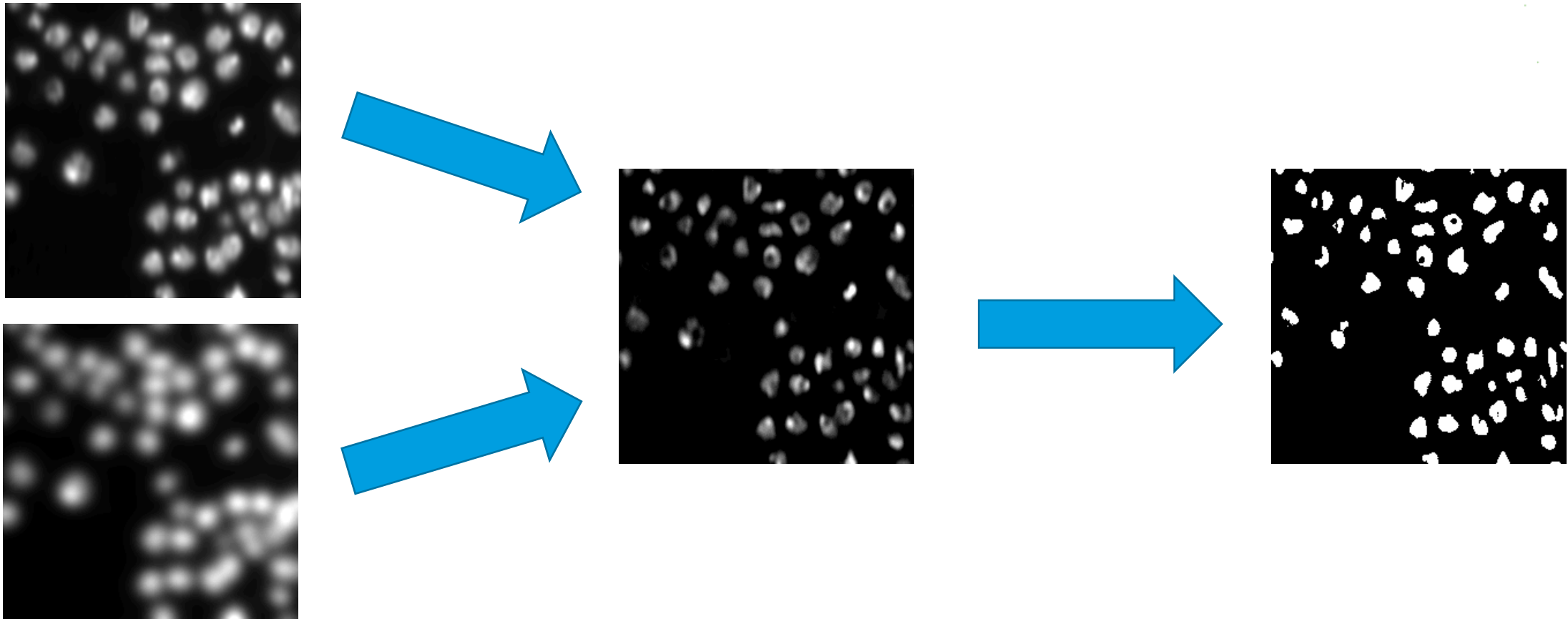
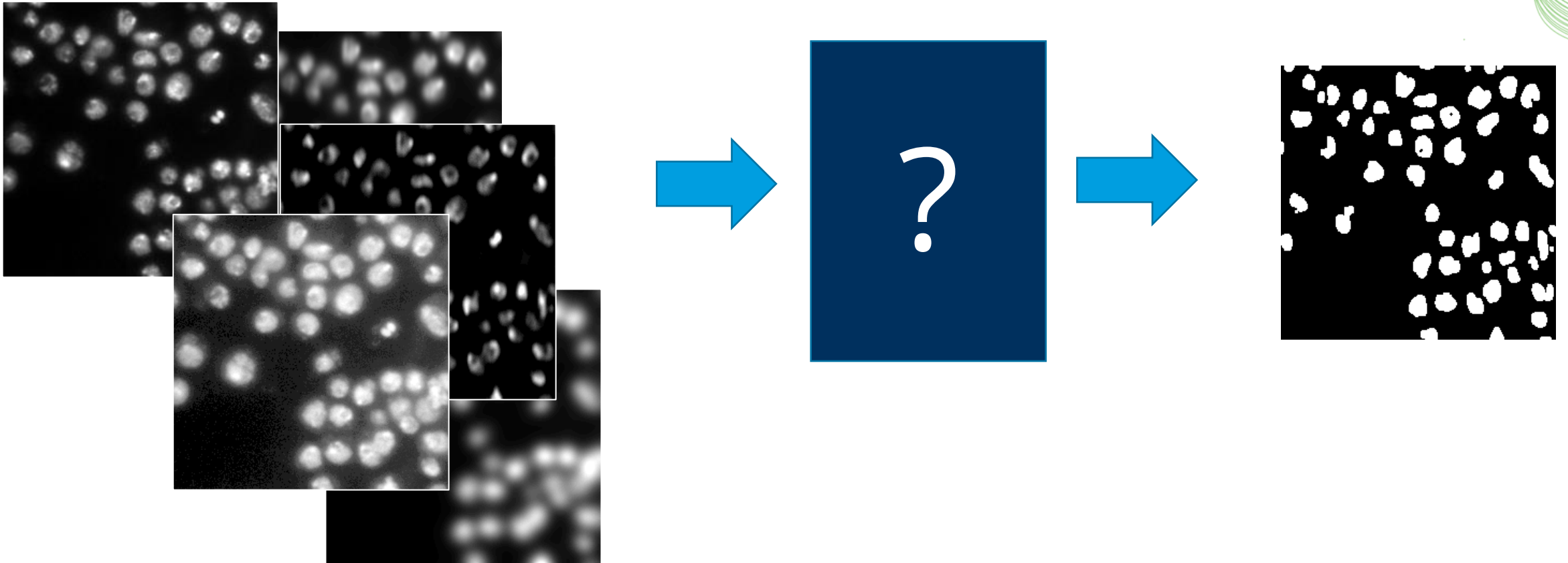


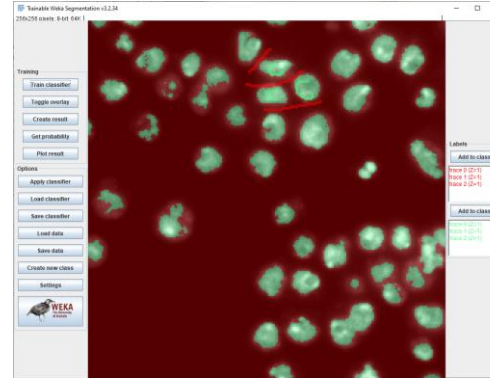
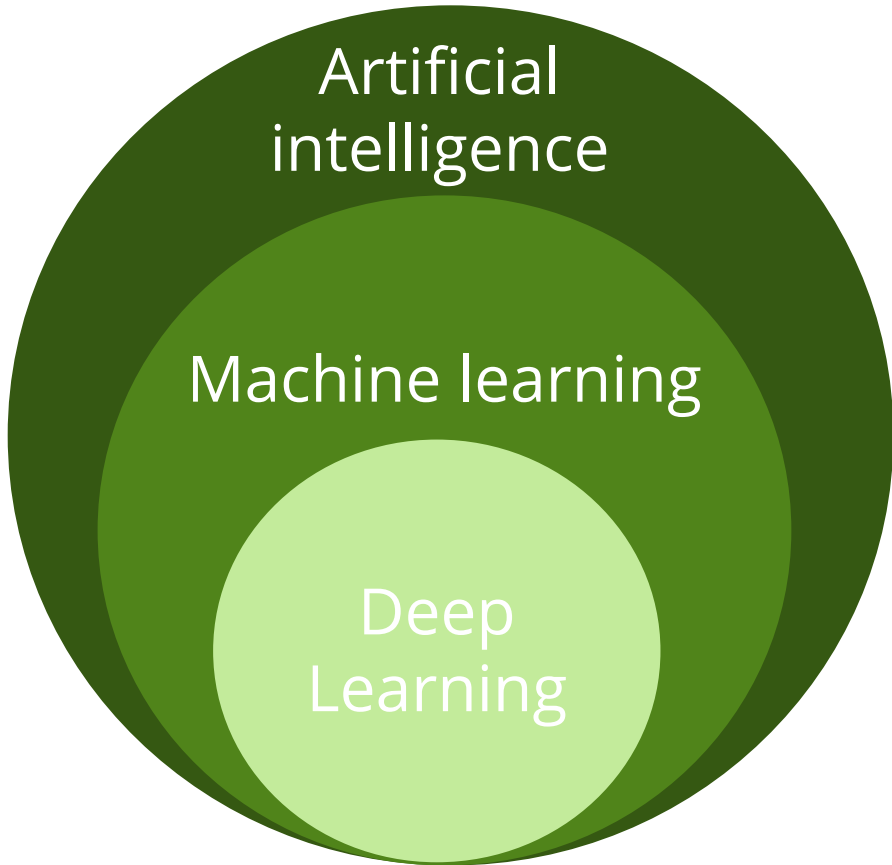
Image segmentation using thresholding

Might there be a technology for optimization which combination of images can be used to get the best segmentation result?

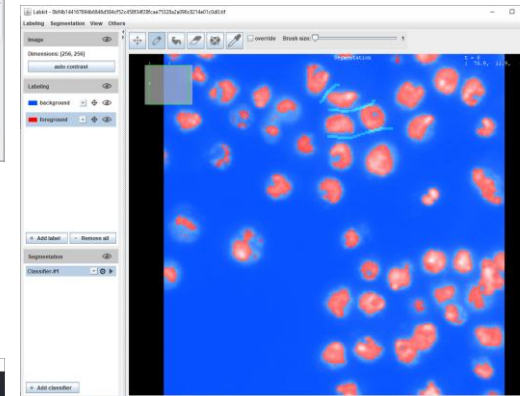


Machine learning

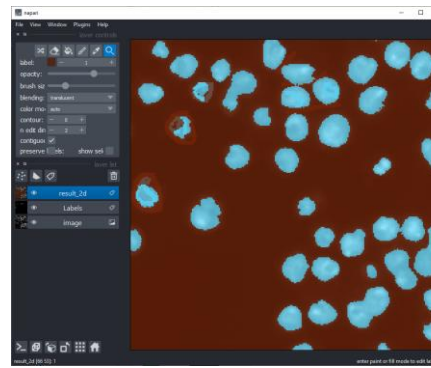
Finds more and more applications, also in life sciences.



Trainable Weka Segmentation
<https://imagej.net/plugins/tws/>



LabKit
<https://imagej.net/plugins/labkit/>

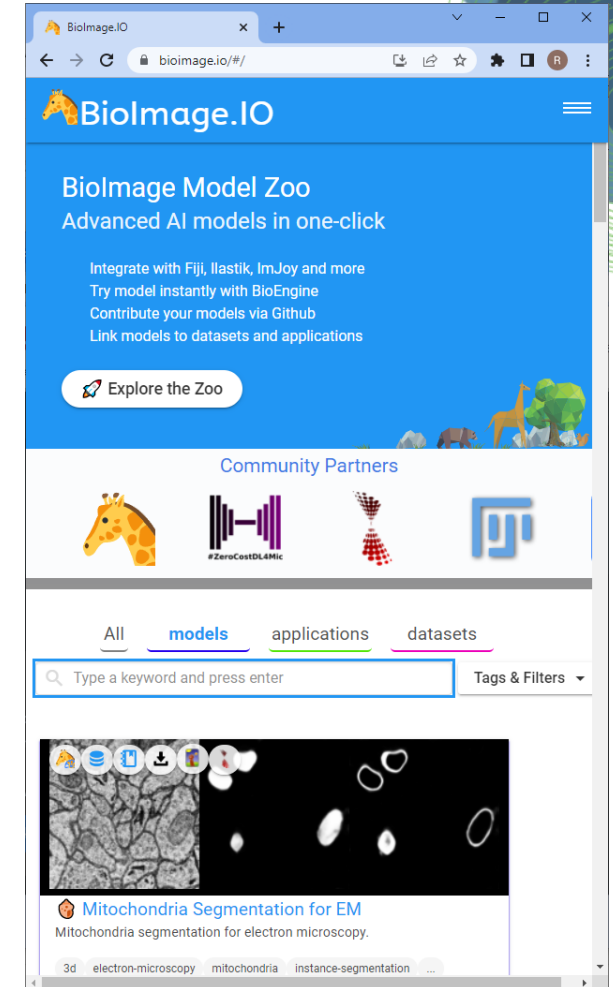
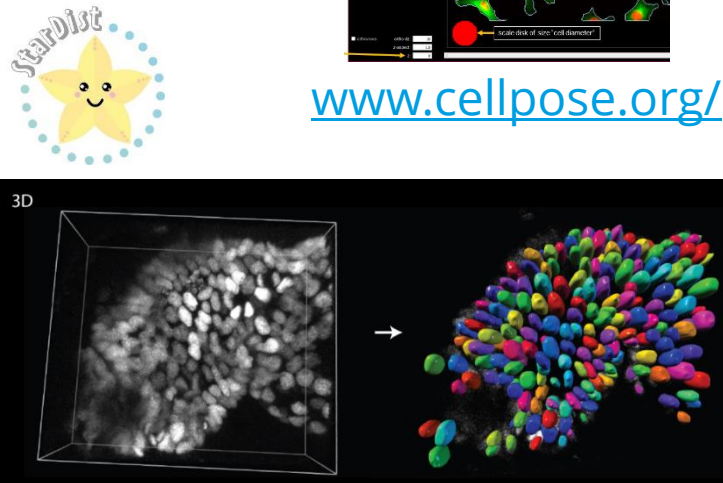
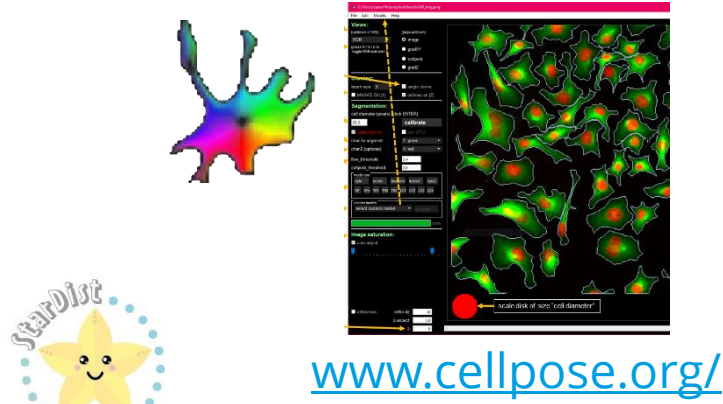
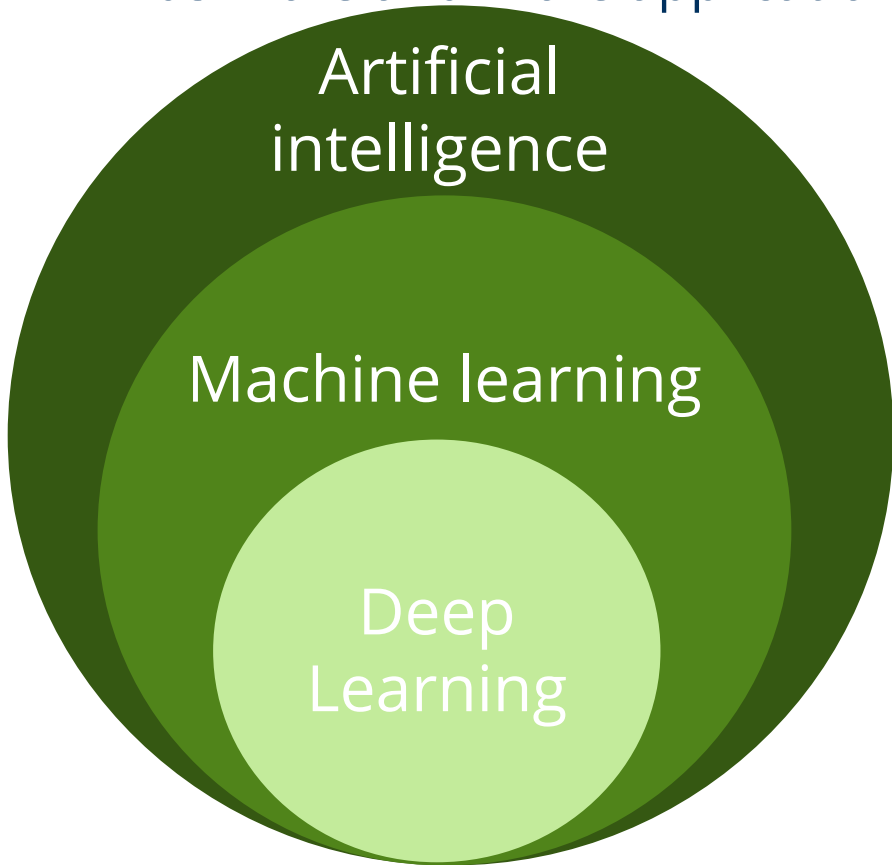


Python / scikit-learn /
napari / apoc

Machine learning

A research field in computer science

Finds more and more applications, also in life sciences.

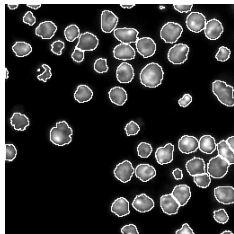
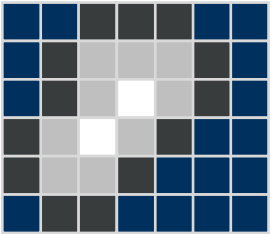


<https://bioimage.io/>

Machine learning

Automatic construction of predictive models from given data

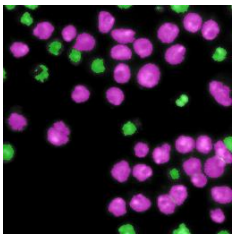
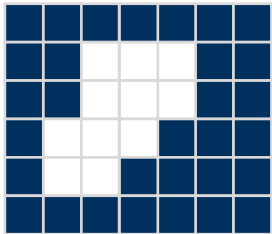
Pixels, Objects, Images, Audio, Text, Measurements, ...



Dense Segmentation / Binarization

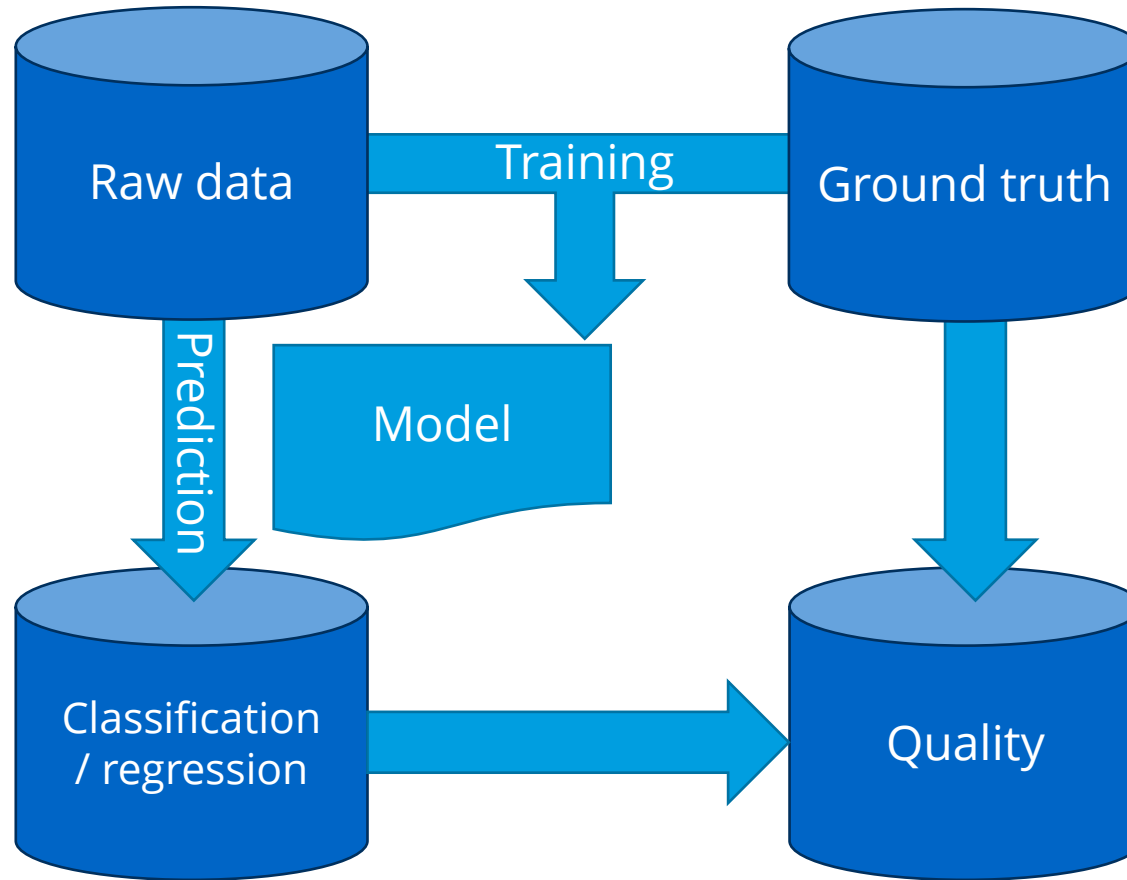
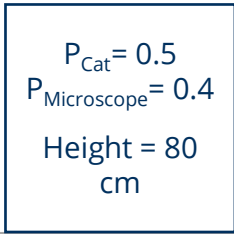
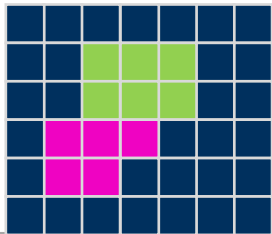
Object classification

Image classification



Instance segmentation

Cont. quantity

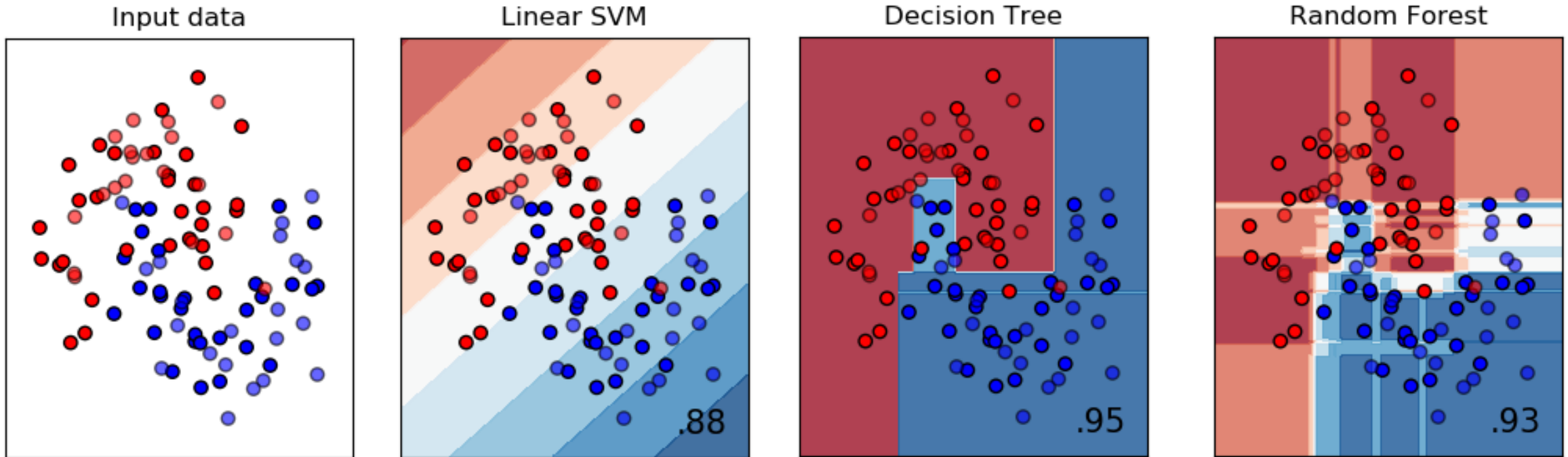


Annotated raw data, usually generated by humans

Precision, Recall

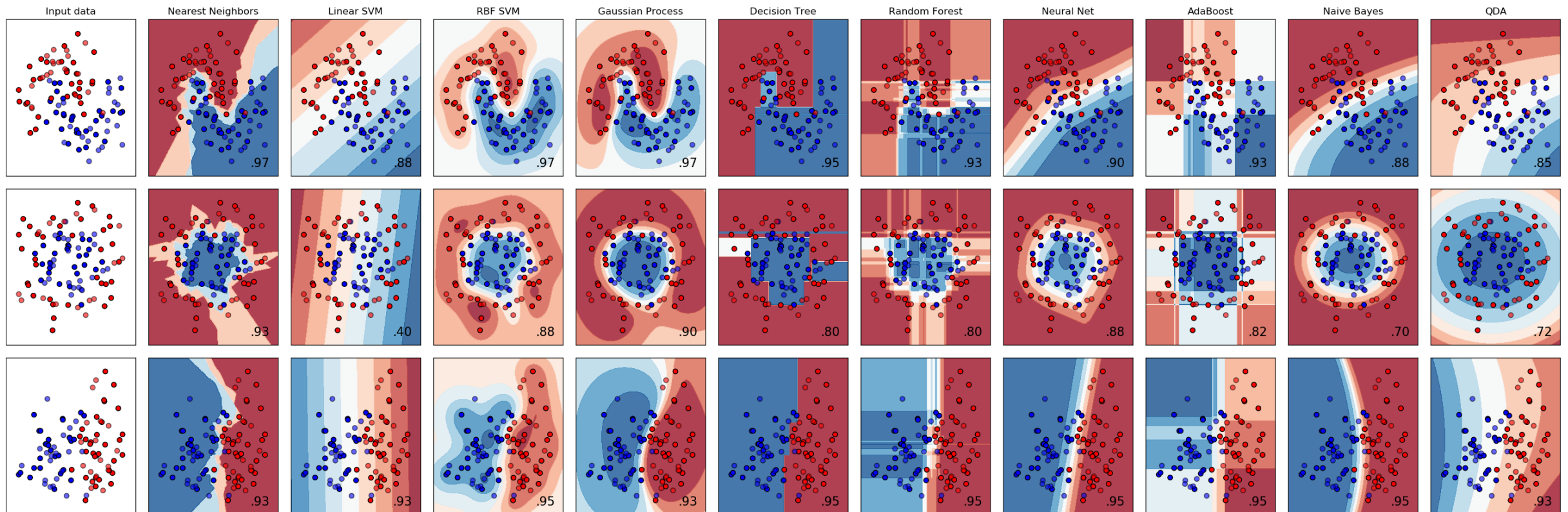
Goal

Guess classification (color) from position of a sample in parameter space.



Approaches

The right approach depends on data, computational resources and desired quality

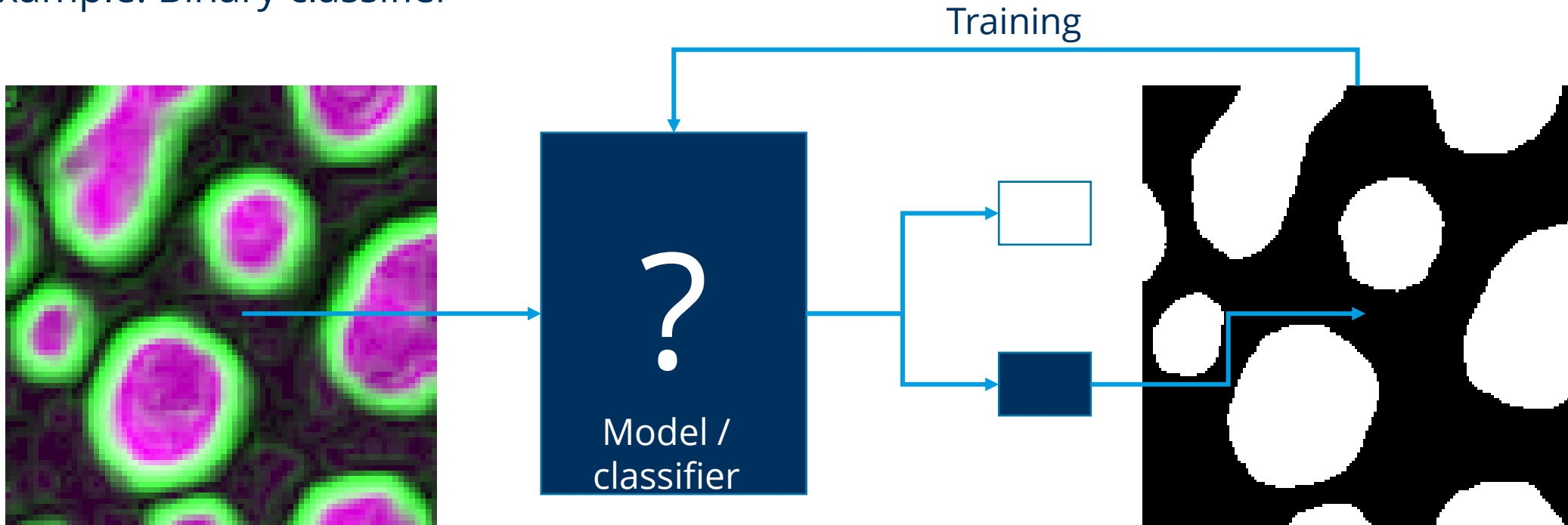


Machine learning for image segmentation

Supervised machine learning: We give the computer some ground truth to learn from

The computer derives a *model* or a *classifier* which can judge if a pixel should be foreground (white) or background (black)

Example: Binary classifier



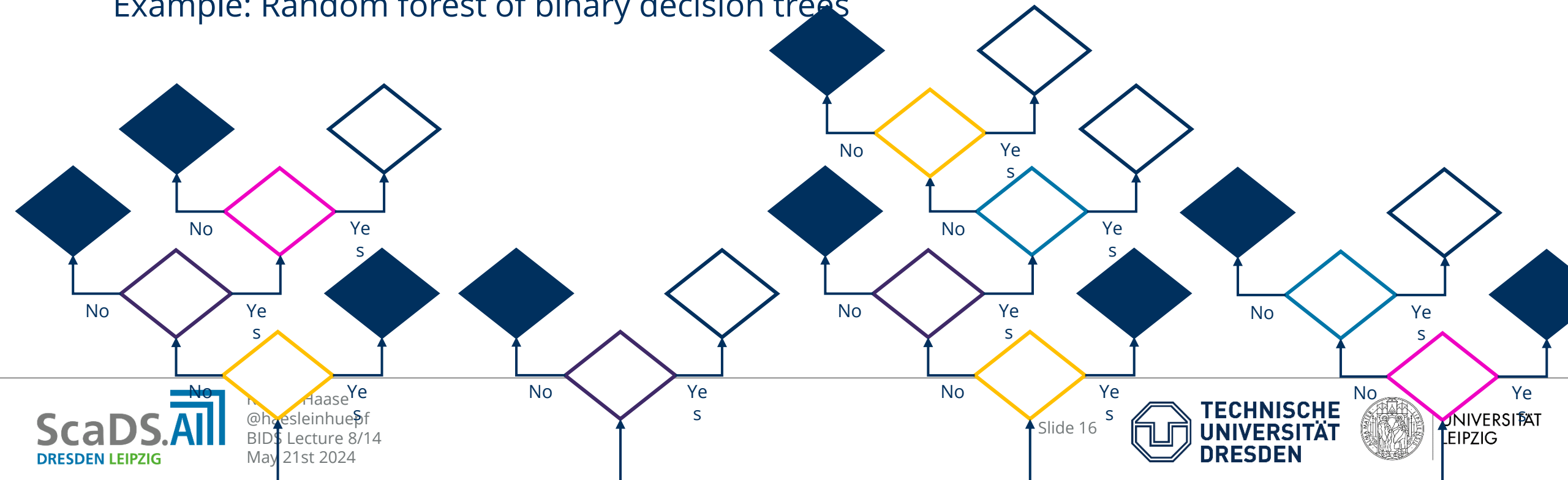
Random forest based image segmentation

Decision trees are classifiers, they decide if a pixel should be white or black

Random decision trees are randomly initialized, afterwards evaluated and selected

Random forests consist of many random decision trees

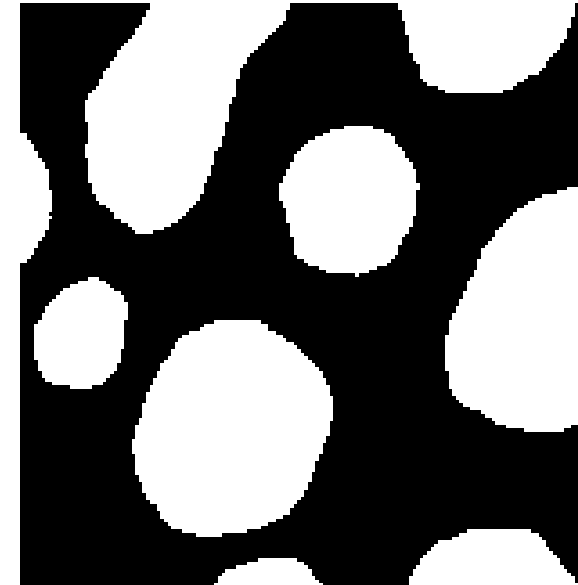
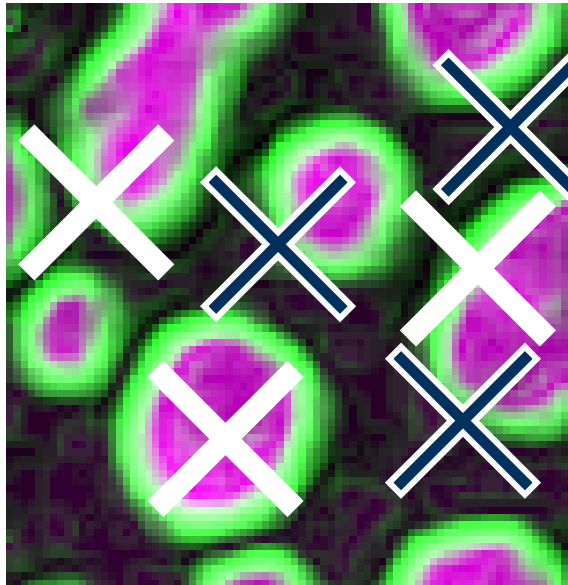
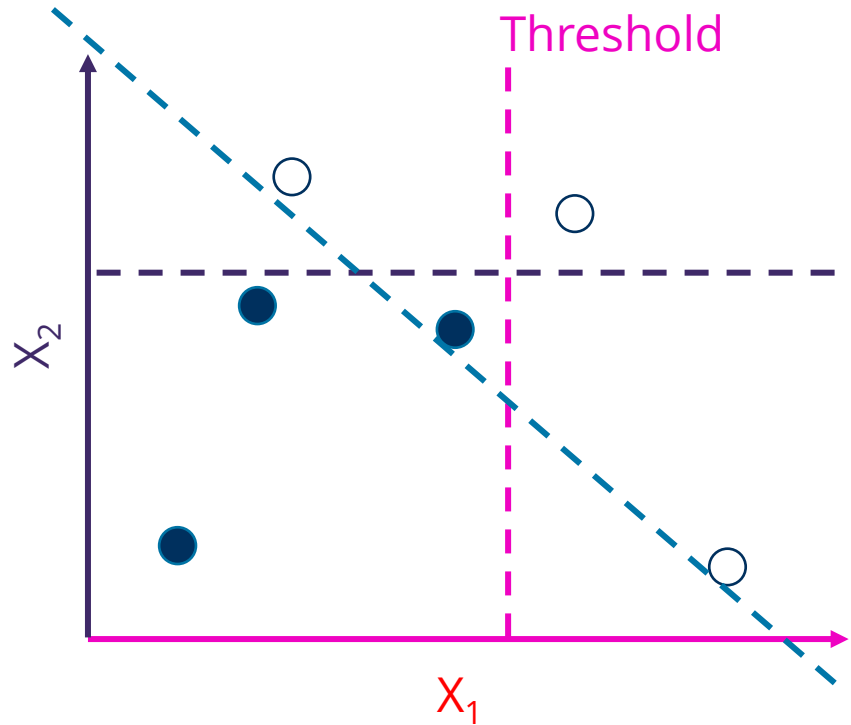
Example: Random forest of binary decision trees



Deriving random decision trees

For efficient processing, we randomly *sample* our data set

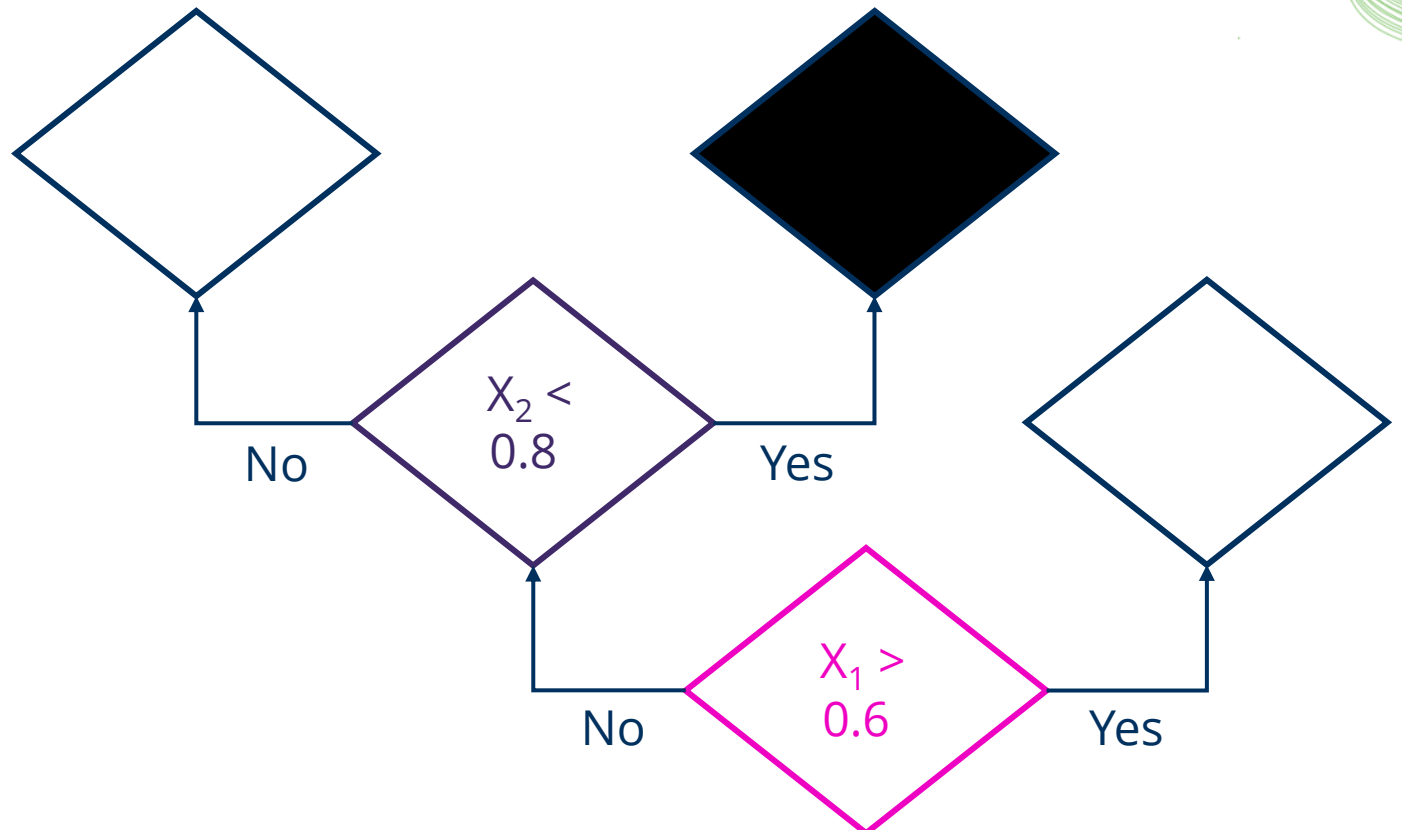
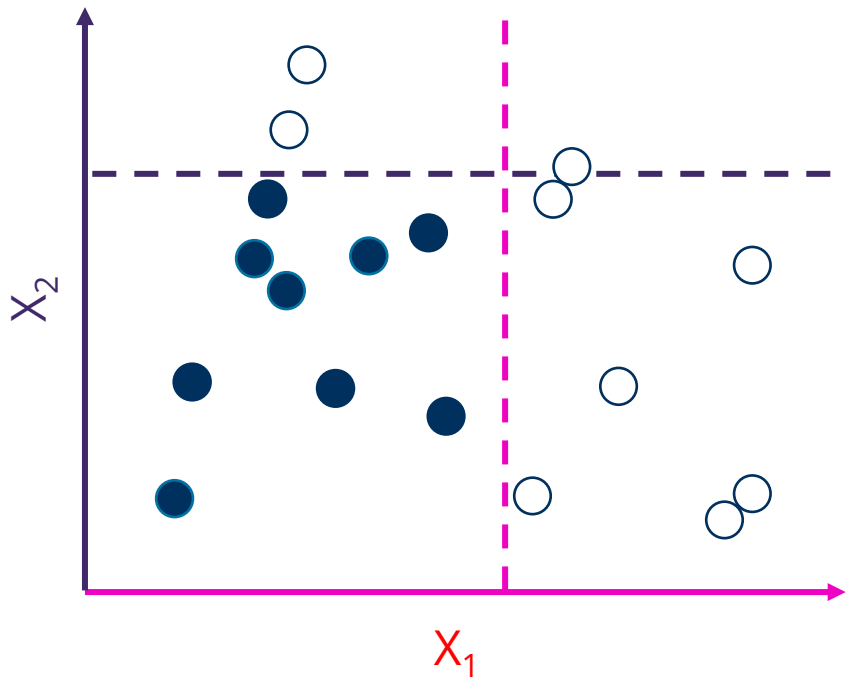
- Individual pixels, their intensity and their classification



Note: You cannot use a single threshold to make the decision

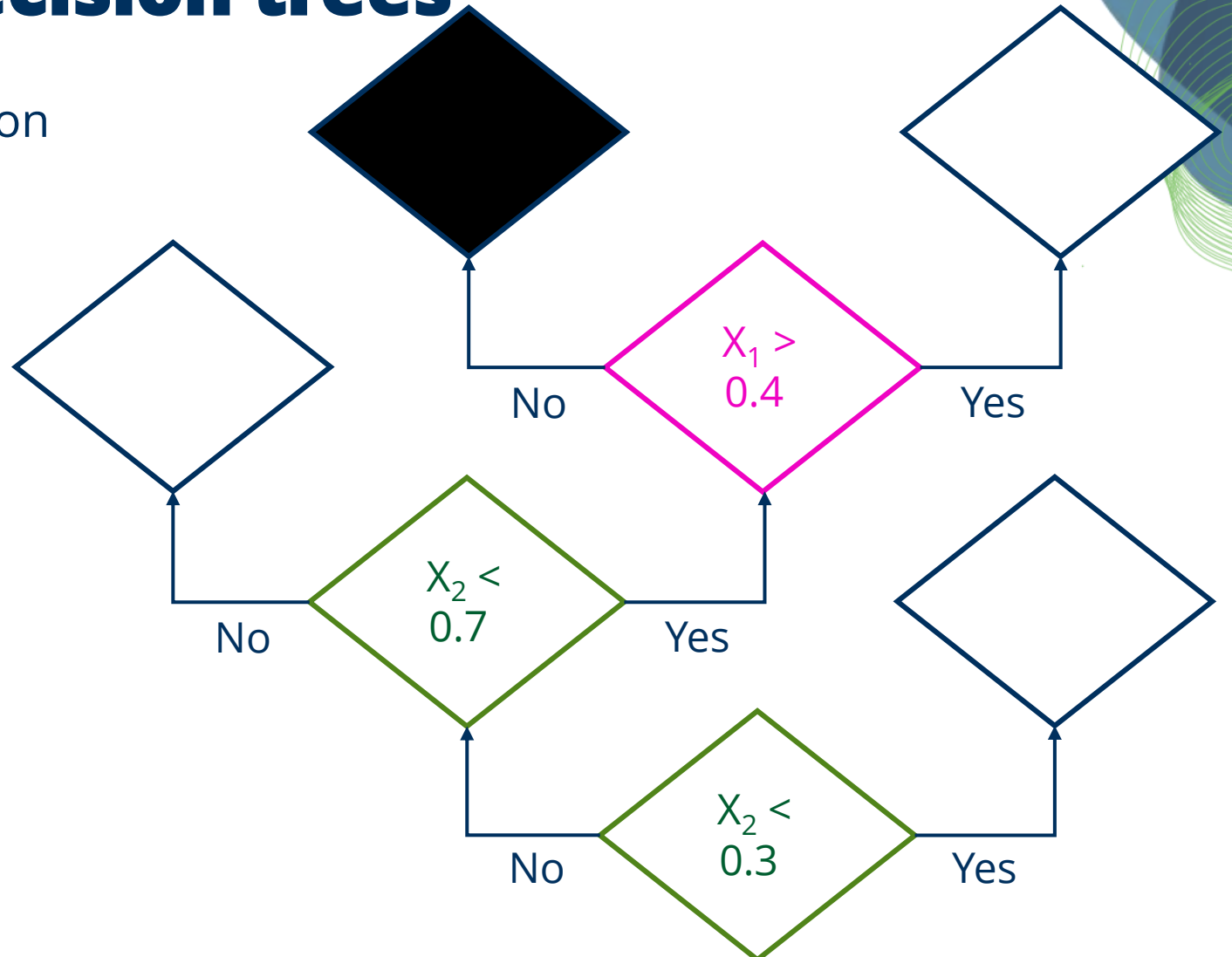
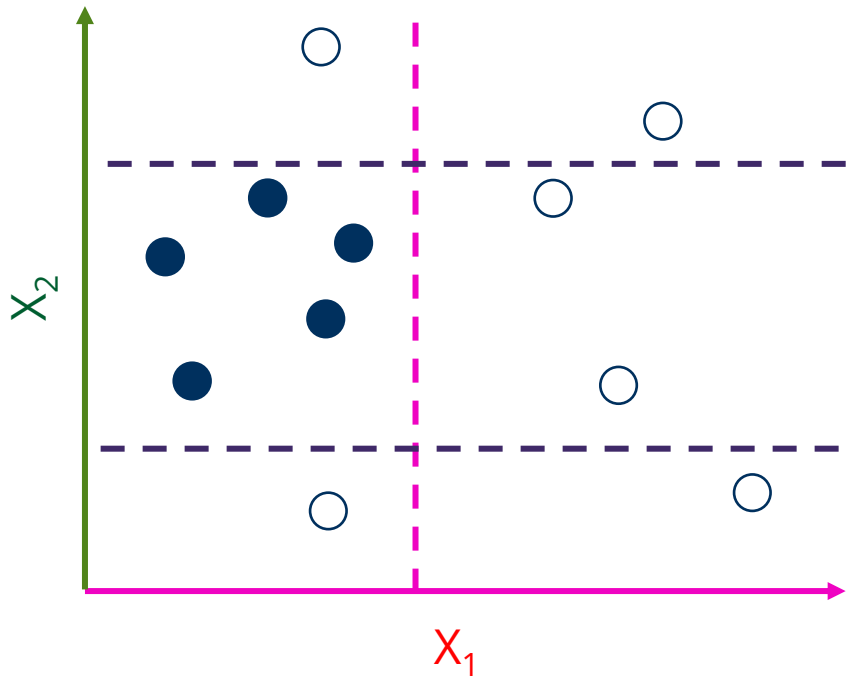
Deriving random decision trees

Decision trees combine several thresholds on several parameters



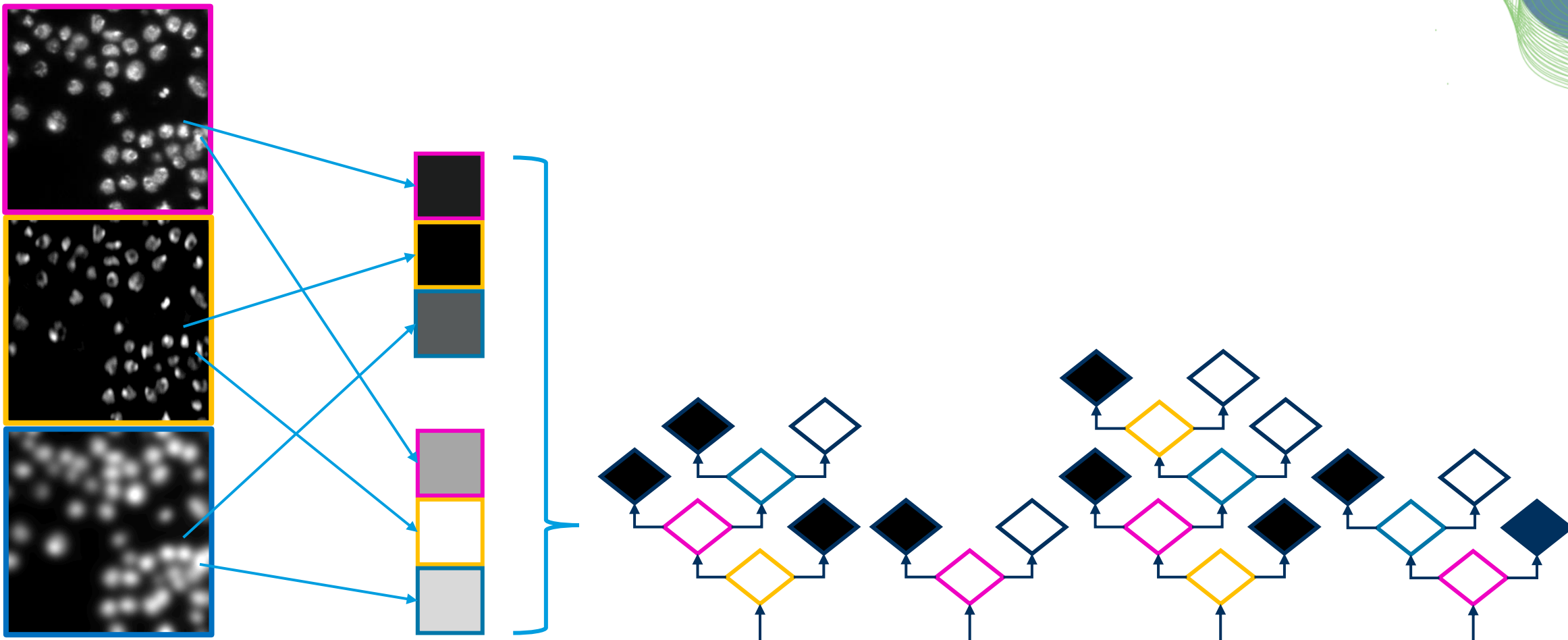
Deriving random decision trees

Depending on sampling, the decision trees are different



Random Forest Pixel Classifiers

By training many decision trees, errors are equilibrated

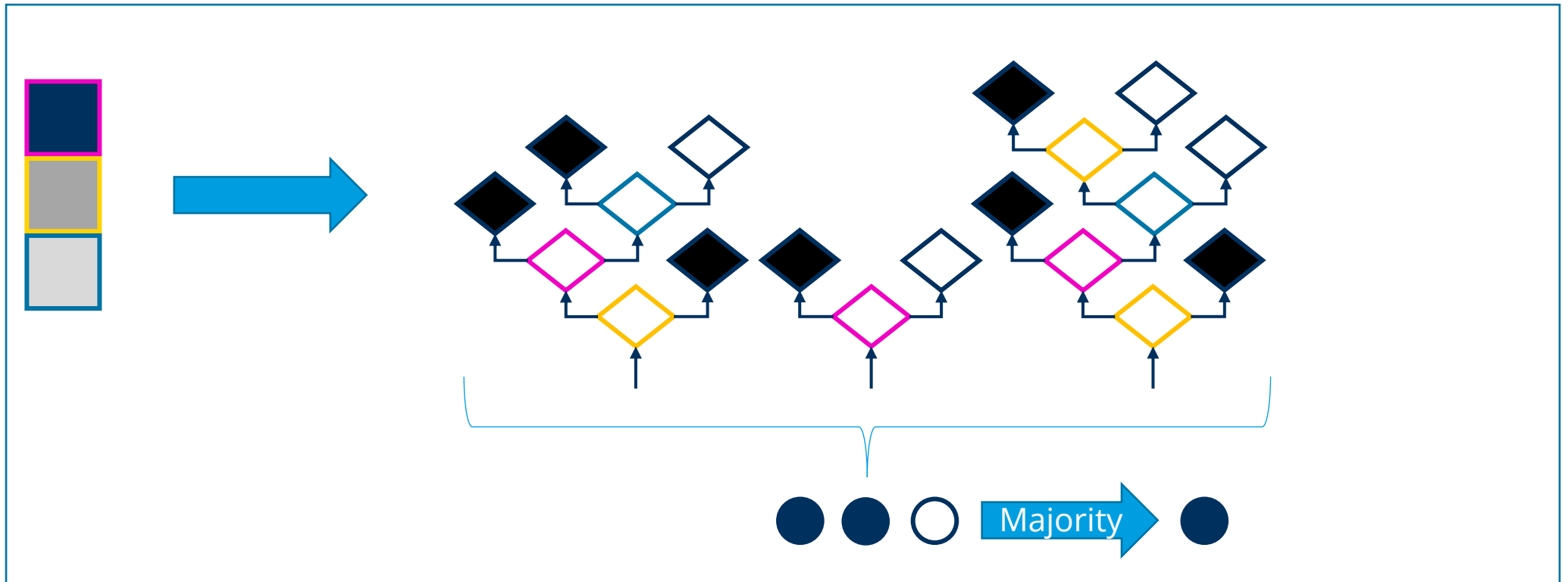


Sampling

Random Forest Pixel Classifiers

Combination of individual tree decisions by voting or max / mean

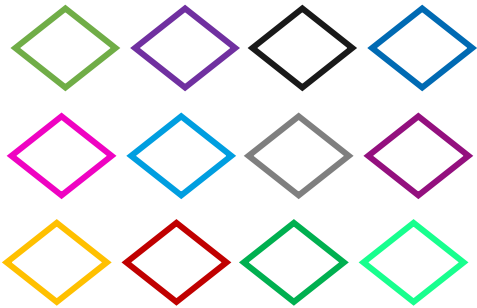
Prediction



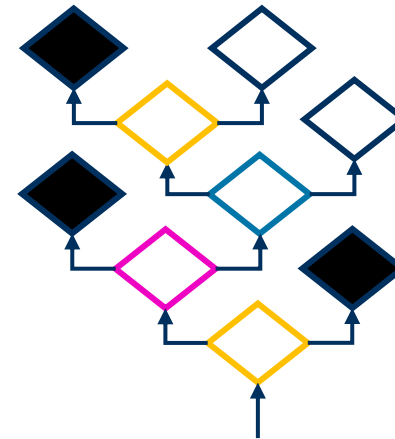
Random Forest Pixel Classifiers

Typical numbers for pixel classifiers in microscopy

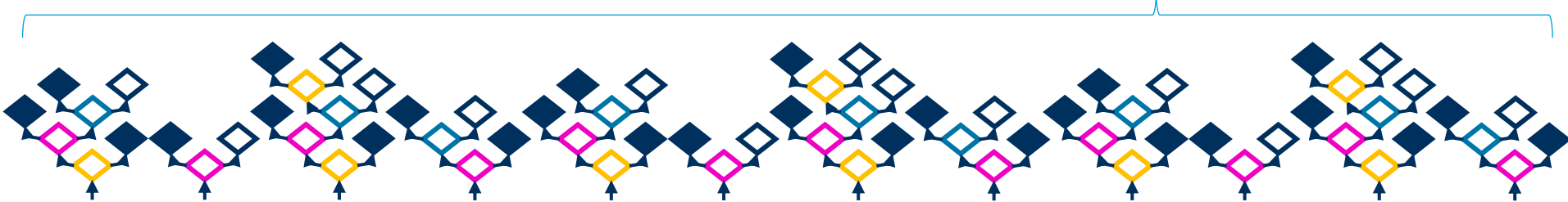
Available features:



- Gaussian blur image
- DoG image
- LoG image
- **Hessian**
-



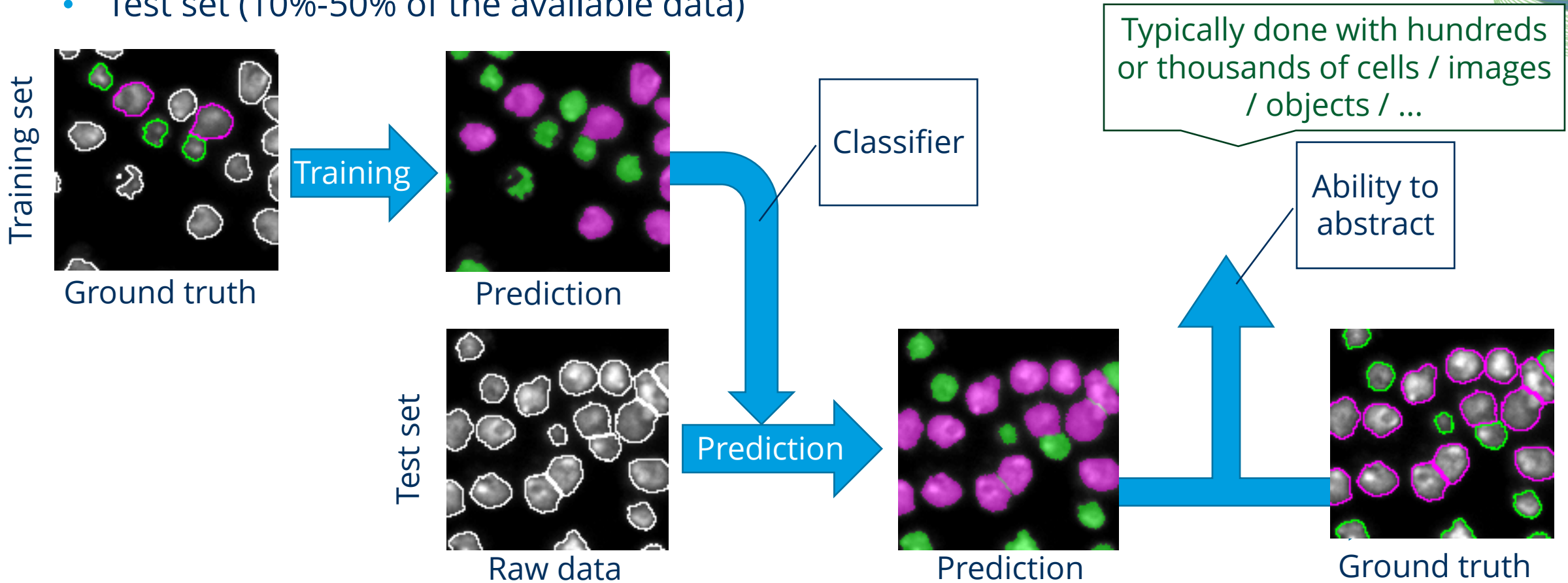
Number of trees: > 100



Model validation

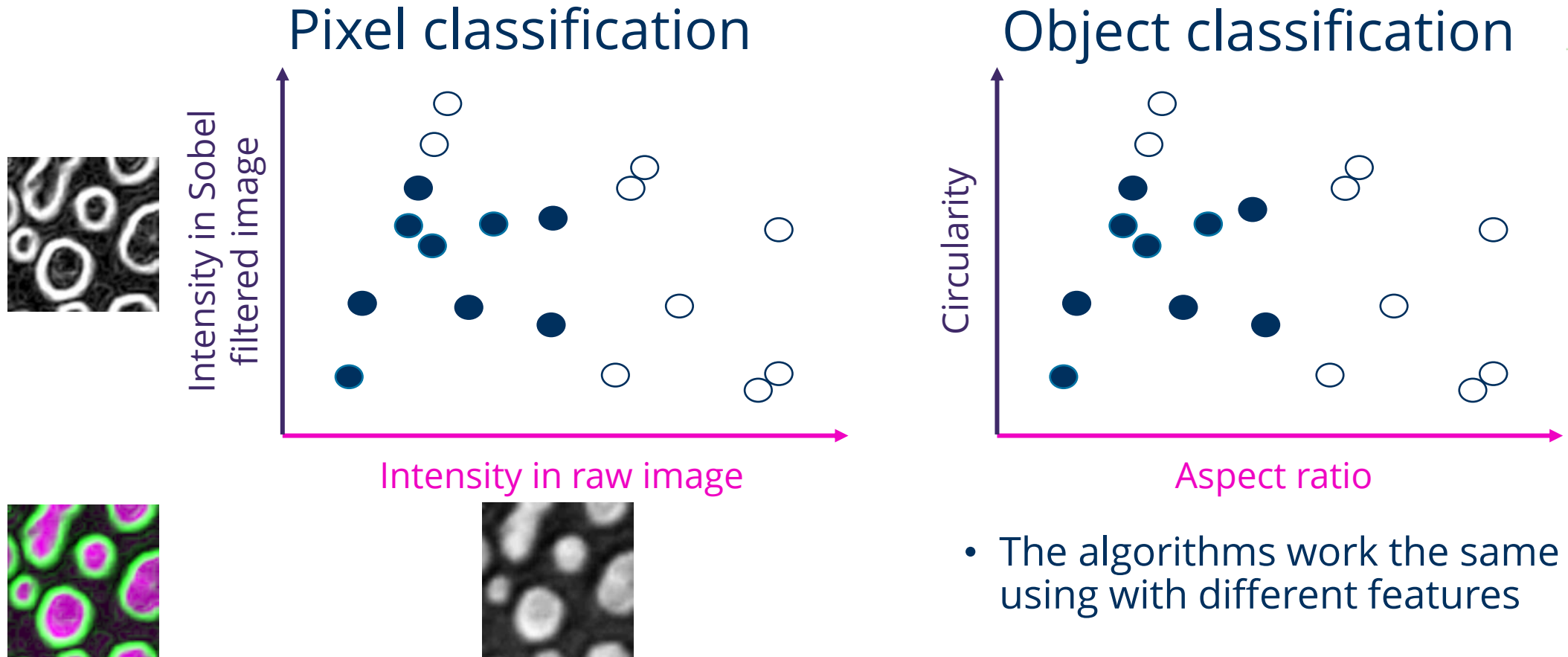
In order to assess model quality, we split the ground truth into two set

- Training set (50%-90% of the available data)
- Test set (10%-50% of the available data)



Object classification

What if we exchange pixel features with object features?



- The algorithms work the same using with different features

Supervised and Unsupervised Machine Learning for Bio-image Analysis

Using
Python

Robert Haase

Reusing materials from Johannes Soltwedel, Till Korten, Johannes Müller, Laura Žigutytė (TU Dresden), Ryan Savill (MPI-CBG), Matthias Täschner (ScaDS.AI/Uni Leipzig) and the Scikit-learn community.

Funded by



Bundesministerium
für Bildung
und Forschung

SACHSEN



Diese Maßnahme wird gefördert durch die Bundesregierung aufgrund eines Beschlusses des Deutschen Bundestages. Diese Maßnahme wird mitfinanziert durch Steuermittel auf der Grundlage des von den Abgeordneten des Sächsischen Landtags beschlossenen Haushaltes.

Tabular object classification

Classify objects starting from feature vectors (table columns)

Raw data

	area	elongation
0	3.950088	2.848643
1	4.955912	3.390093
2	7.469852	5.575289
3	2.544467	3.017479
4	3.465662	1.463756
5	3.156507	3.232181
6	9.978705	6.676372
7	6.001683	5.047063
8	2.457139	3.416050
9	3.672295	3.407462
10	9.413702	7.598608

“Ground truth” annotation

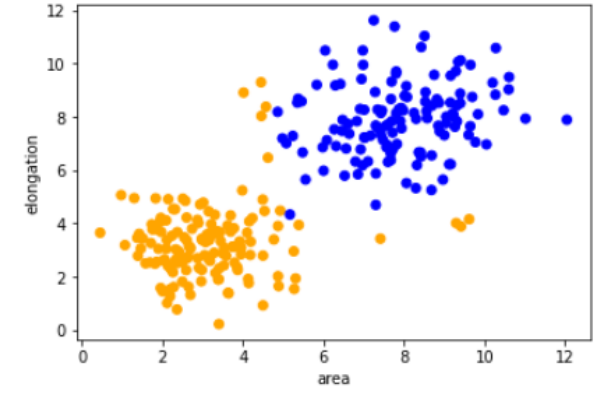
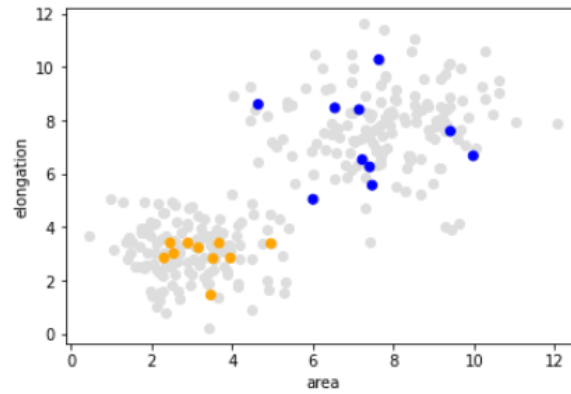
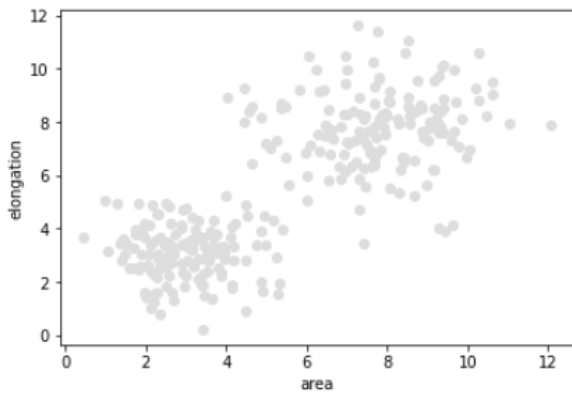
```
annotation = [1, 1, 2, 1, 1, 1,
```

Classifier training

```
classifier = RandomForestClassifier()  
classifier.fit(train_data, train_annotation)
```

Classifier prediction

```
result = classifier.predict(validation_data)
```



Interactive pixel classification

Prepare an empty layer for annotations and keep a `reference`

```
labels = viewer.add_labels(  
    np.zeros(image.shape).astype(int))
```

Read annotations

```
manual_annotations = labels.data
```

```
from skimage.io import imshow
```

```
imshow>manual_annotations,  
      vmin=0, vmax=2)
```

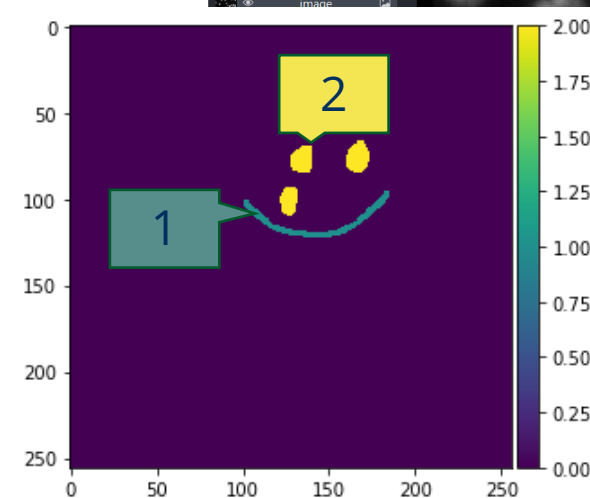
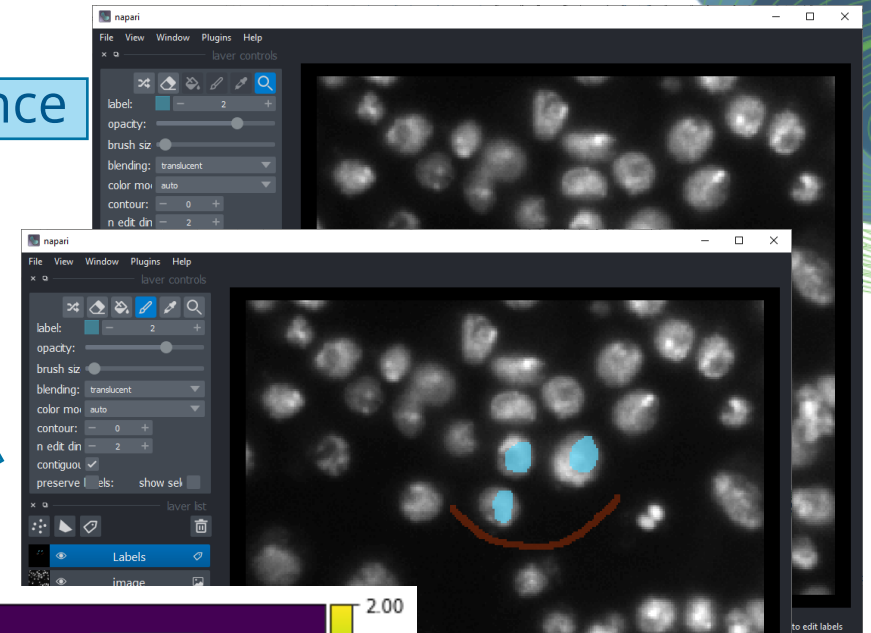
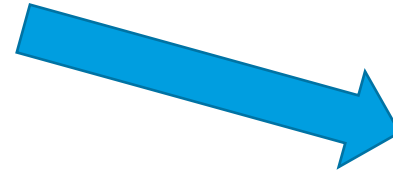
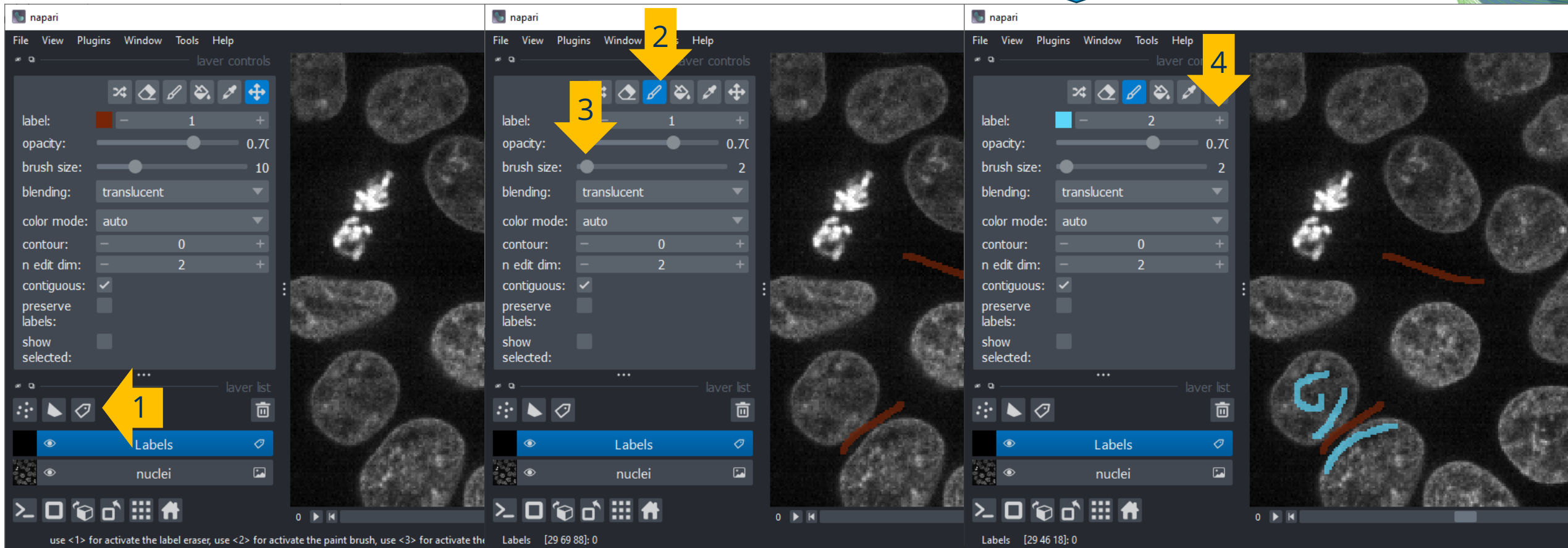


Image data source: [BBBC038v1](https://doi.org/10.26434/chemrxiv-2019-08-01), available from the Broad Bioimage Benchmark Collection (Caicedo et al., Nature Methods, 2019).

Napari – common workflows

Pixel / object annotation drawing

- [1: Create empty labels layer]
- 2: Select paint brush tool
- 3: Decrease brush size
- 4: Increase label



Interactive pixel classification

Pixel classification using scikit-learn

- Expects one-dimensional arrays for features and ground truth

```
# train classifier
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(max_depth=2, random_state=0)
classifier.fit(X, y)
```

```
y_ = classifier.predict(X)
```

Image data

Ground truth /
annotation

prediction

Image data

Interactive pixel classification

Pixel classification using scikit-learn

- Expects one-dimensional arrays for features and ground truth

```
# for training, we need to generate features
```

```
feature_stack = generate_feature_stack(image)
```

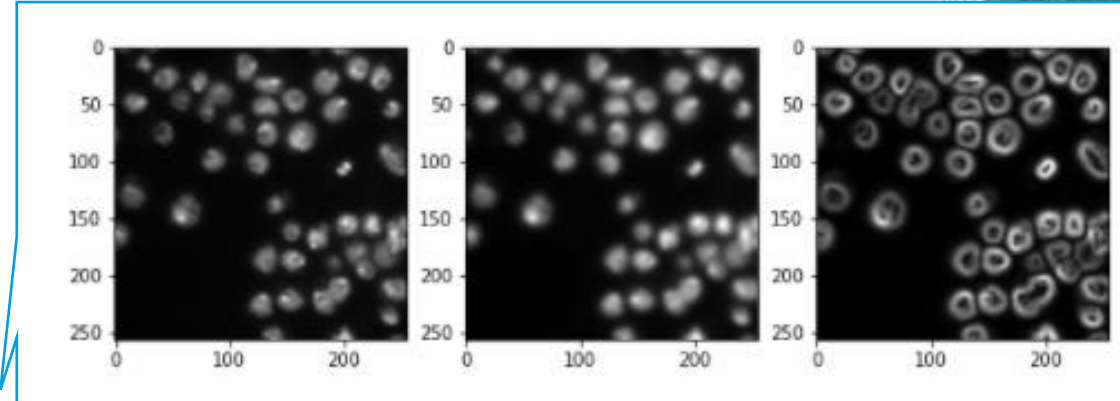
```
X, y = format_data(feature_stack, manual_annotations)
```

```
# train classifier
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier(max_depth=2, random_state=0)
```

```
classifier.fit(X, y)
```



Interactive pixel classification

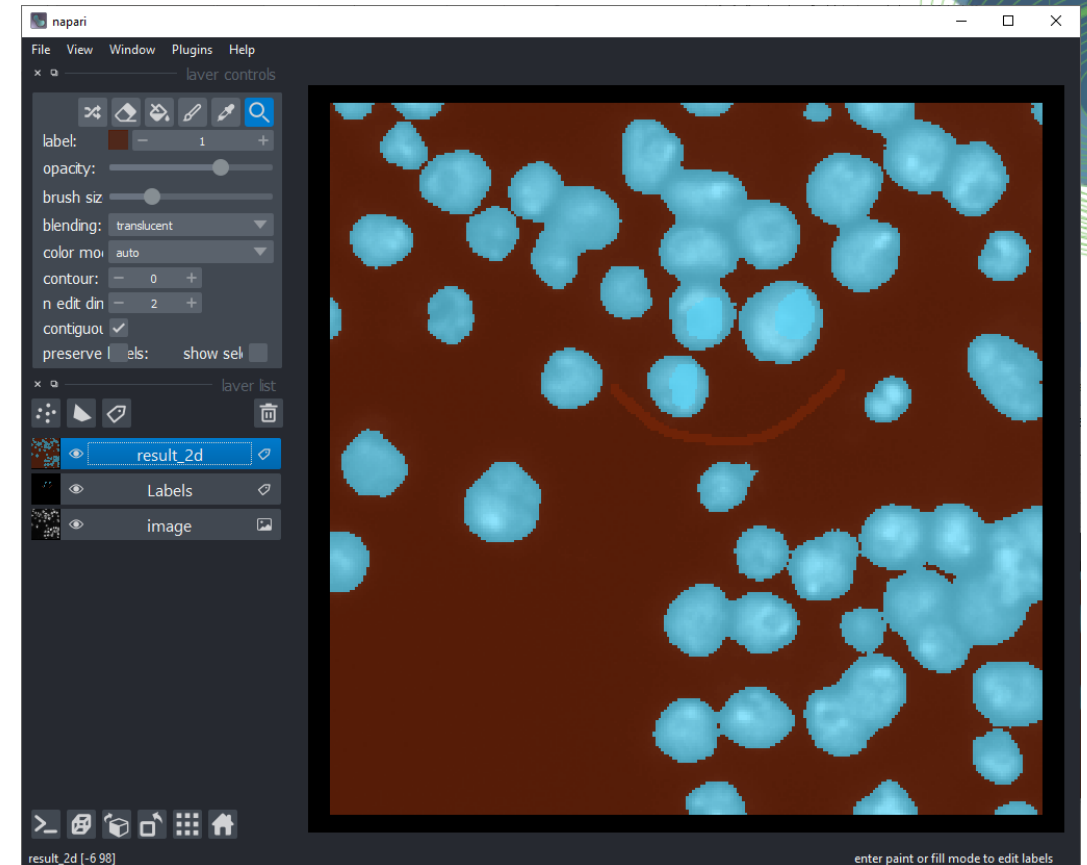
Pixel classification using scikit-learn

- Expects one-dimensional arrays for features and ground truth

```
# process the whole image and show result  
result_1d = classifier.predict(feature_stack.T)  
result_2d = result_1d.reshape(image.shape)
```

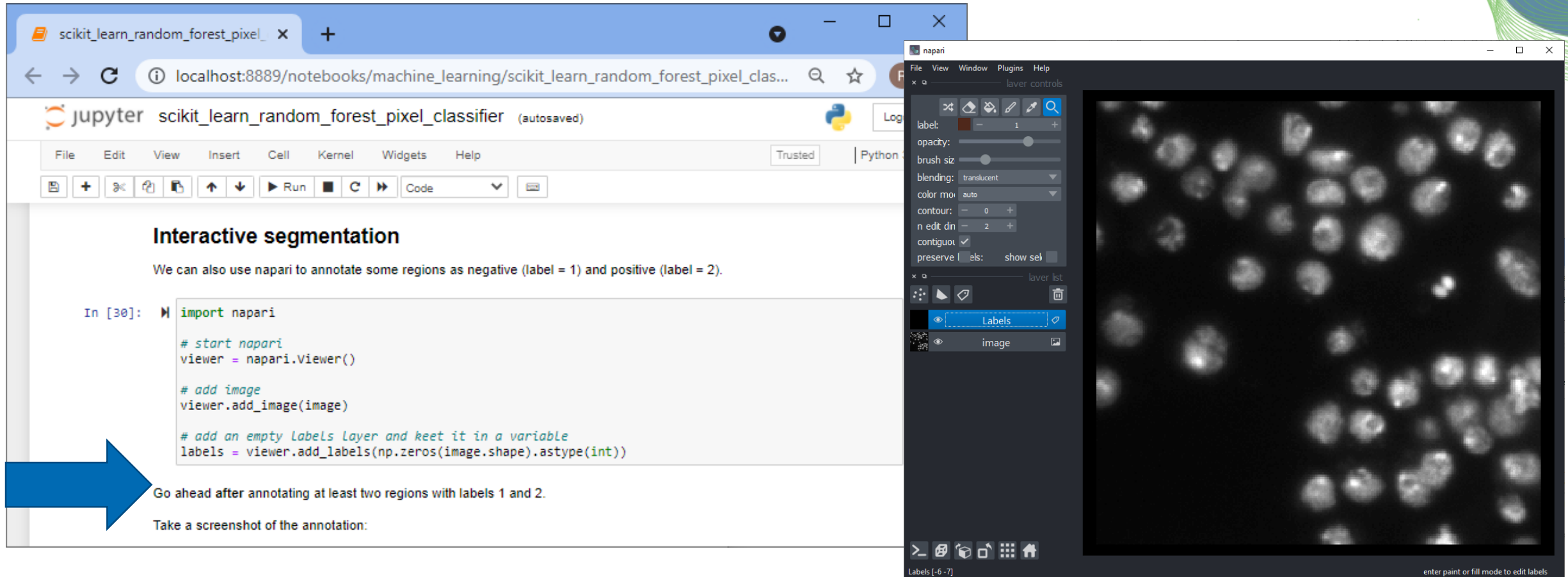
```
viewer.add_labels(result_2d)
```

Convert 1D
result back to 2D



Interactive pixel classification

Jupyter notebooks and napari side-by-side



Interactive segmentation

We can also use napari to annotate some regions as negative (label = 1) and positive (label = 2).

```
In [30]: import napari

# start napari
viewer = napari.Viewer()

# add image
viewer.add_image(image)

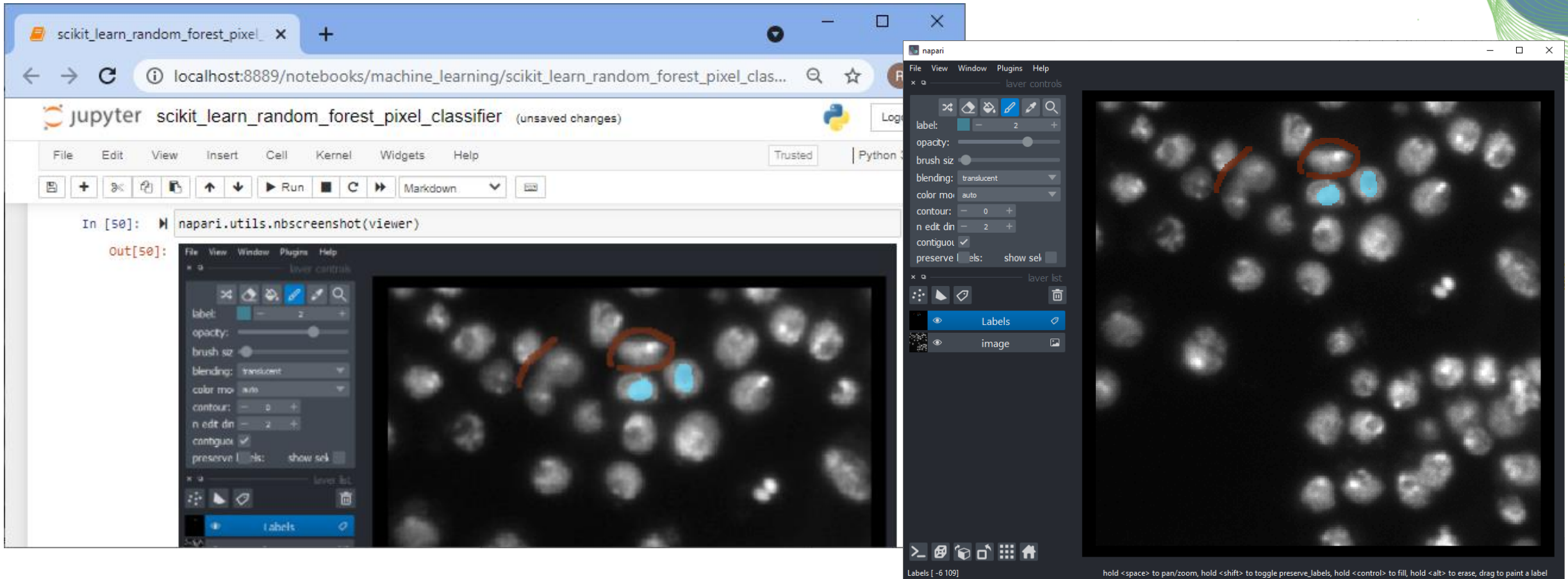
# add an empty Labels Layer and keep it in a variable
labels = viewer.add_labels(np.zeros(image.shape).astype(int))
```

Go ahead after annotating at least two regions with labels 1 and 2.

Take a screenshot of the annotation:

Interactive pixel classification

Jupyter notebooks and napari side-by-side



Interactive pixel classification

Jupyter notebooks and napari side-by-side

The image shows a side-by-side view of a Jupyter notebook and the napari application. The Jupyter notebook on the left contains Python code for training a Random Forest classifier and processing an image. The napari application on the right displays the resulting image with a layer list containing 'result_2d', 'Labels', and 'image'. The 'result_2d' layer is selected, and its properties are visible in the layer controls panel.

```
In [52]: # generate features (that's actually not necessary,  
# as the variable is still there and the image is the same.  
# but we do it for completeness)  
feature_stack = generate_feature_stack(image)  
X, y = format_data(feature_stack, manual_annotations)  
  
# train classifier  
classifier = RandomForestClassifier(max_depth=2, random_state=0)  
classifier.fit(X, y)  
  
# process the whole image and show result  
result_1d = classifier.predict(feature_stack.T)  
result_2d = result_1d.reshape(image.shape)  
imshow(result_2d)  
  
matplotlib_plugin.py (150): Low image data range; displaying image with stretched contrast.  
Out[52]: <matplotlib.image.AxesImage at 0x244550ec130>
```

Supervised and Unsupervised Machine Learning for Bio-image Analysis

Robert Haase

Reusing materials from Johannes Soltwedel, Till Korten, Johannes Müller, Laura Žigutytė (TU Dresden), Ryan Savill (MPI-CBG), Matthias Täschner (ScaDS.AI/Uni Leipzig) and the Scikit-learn community.

GPU-
accelerated

Using
Python

Funded by



Bundesministerium
für Bildung
und Forschung

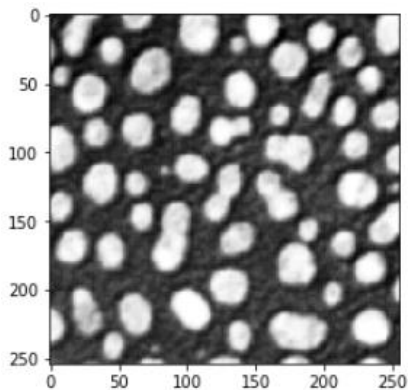
SACHSEN



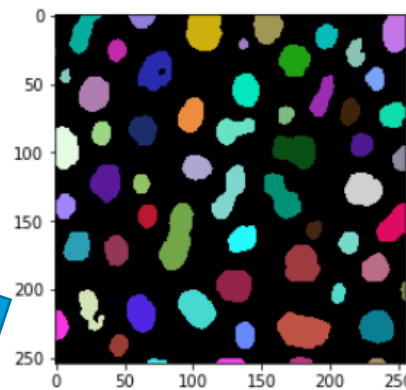
Diese Maßnahme wird gefördert durch die Bundesregierung aufgrund eines Beschlusses des Deutschen Bundestages. Diese Maßnahme wird mitfinanziert durch Steuermittel auf der Grundlage des von den Abgeordneten des Sächsischen Landtags beschlossenen Haushaltes.

Accelerated pixel and object classification

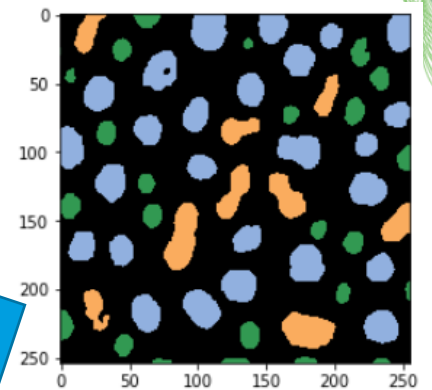
APOC is a python library that makes use of OpenCL-compatible Graphics Cards to accelerate pixel and object classification



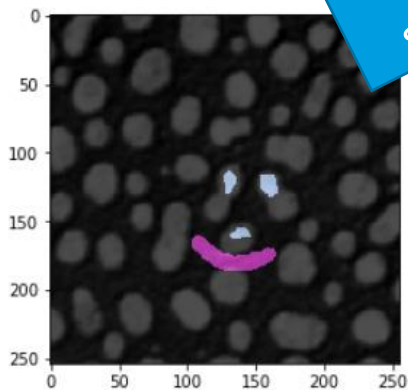
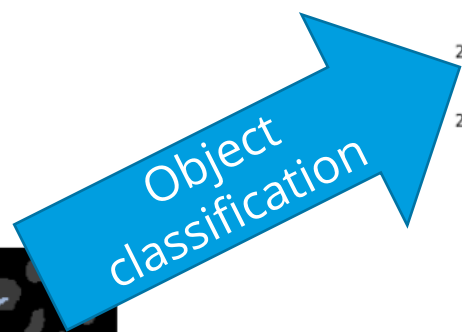
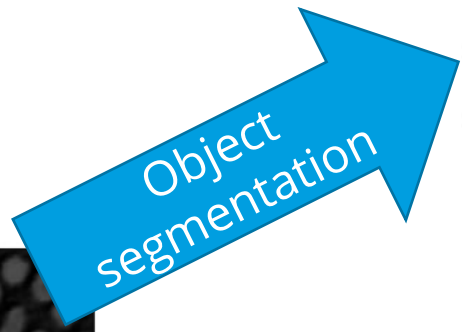
Raw image



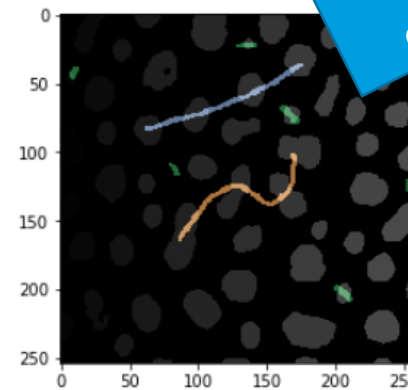
Object label image



Class label image



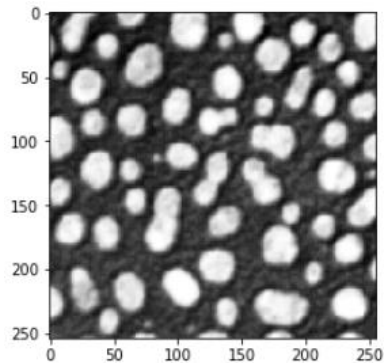
Pixel annotation



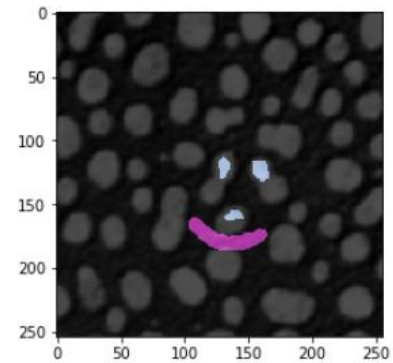
Object annotation

Object segmentation

Pixel classification + connected component labeling



Raw image



Pixel annotation

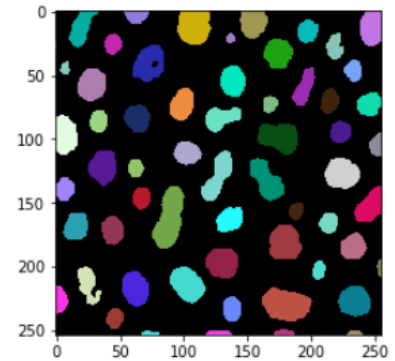
```
# define features
features = "gaussian_blur=1 gaussian_blur=5 sobel_of_gaussian_blur=1"

# this is where the model will be saved
cl_filename = 'my_object_segmenter.cl'

# delete classifier in case the file exists already
apoc.erase_classifier(cl_filename)

# train classifier
clf = apoc.ObjectSegmenter(opencl_filename=cl_filename, positive_class_identifier=2)
clf.train(features, manual_annotations, image)

segmentation_result = clf.predict(features=features, image=image)
cle.imshow(segmentation_result, labels=True)
```



Object label image

Training on folders of annotated images

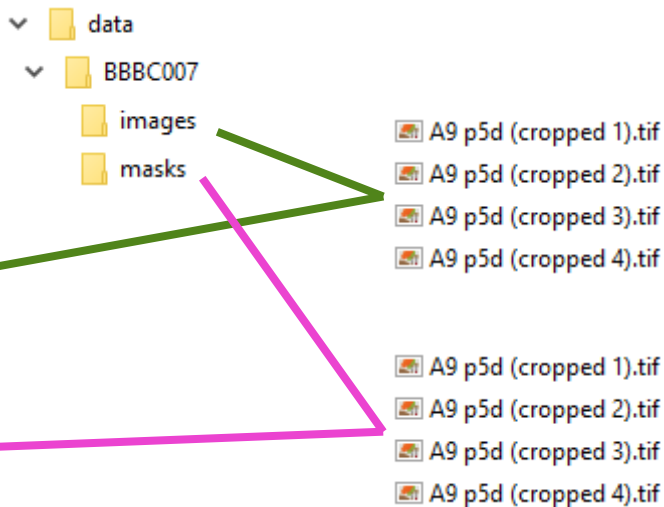
Training

If the folders are setup properly, we can pass the folders to the training.

```
[4]: # setup classifier and where it should be saved
segmenter = apoc.ObjectSegmenter(openc1_filename="data/object_segmen...

# setup feature set used for training
features = apoc.PredefinedFeatureSet.object_size_1_to_5_px.value

# train classifier on folders
apoc.train_classifier_from_image_folders(
    segmenter,
    features,
    image = image_folder,
    ground_truth = masks_folder)
```

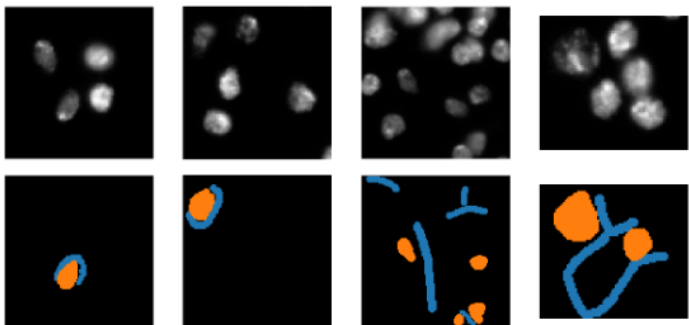


```
[2]: image_folder = "data/BBBC007/images/"
masks_folder = "data/BBBC007/masks/"

[3]: file_list = os.listdir(image_folder)

# show all images
fig, axs = plt.subplots(1, 4, figsize=(15,15))
for i, filename in enumerate(file_list):
    image = imread(image_folder + filename)
    stackview.imshow(image, plot=axs[i])
plt.show()

# show corresponding Label images
fig, axs = plt.subplots(1, 4, figsize=(15,15))
for i, filename in enumerate(file_list):
    masks = imread(masks_folder + filename)
    stackview.imshow(masks, plot=axs[i], labels=True)
plt.show()
```



Prediction

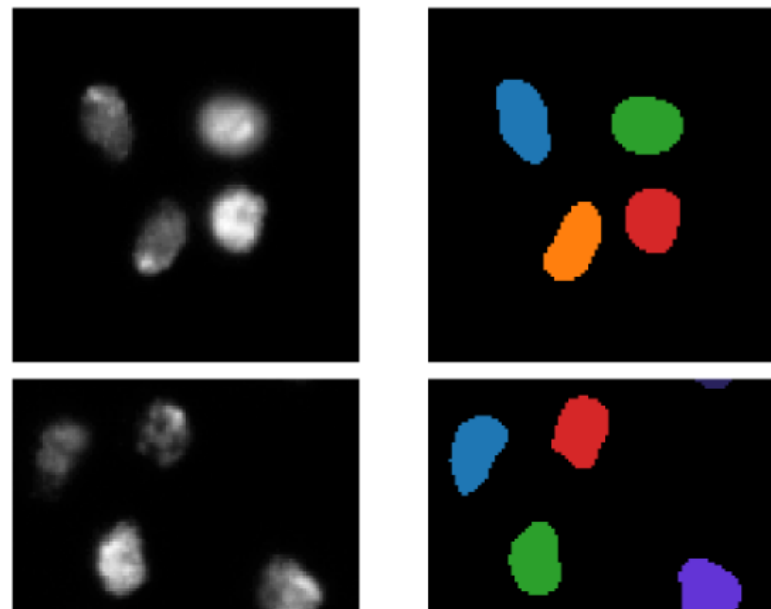
After the training, we can apply the classifier to all images in the image folder. The following line reloads the classifier from disk. In that way we can ensure that it was stored correctly.

```
[5]: segmenter = apoc.ObjectSegmenter(openc1_filename="data/object_segmen...

[6]: # show all images
for i, filename in enumerate(file_list):
    fig, axs = plt.subplots(1, 2, figsize=(15,15))

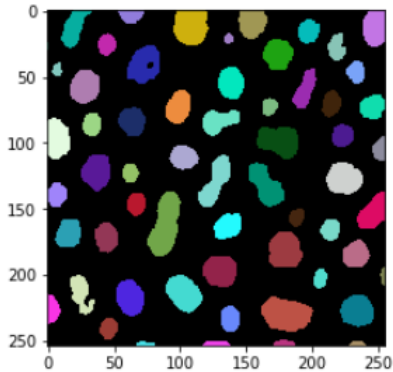
    image = imread(image_folder + filename)
    stackview.imshow(image, plot=axs[0])

    labels = segmenter.predict(image)
    stackview.imshow(labels, plot=axs[1], labels=True)
```

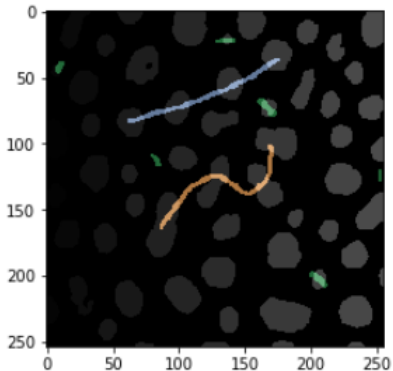


Object classification

Feature extraction + tabular classification



Object label image



Object annotation

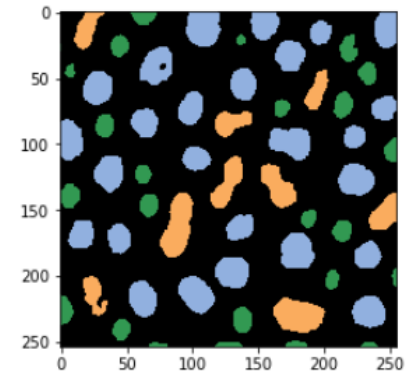
```
# for the classification we define size and shape as criteria  
features = 'area mean_max_distance_to_centroid_ratio'
```

```
# This is where the model will be saved  
cl_filename_object_classifier = "my_object_classifier.cl"
```

```
# delete classifier in case the file exists already  
apoc.erase_classifier(cl_filename_object_classifier)
```

```
# train the classifier  
classifier = apoc.ObjectClassifier(cl_filename_object_classifier)  
classifier.train(features, segmentation_result, annotation, image)
```

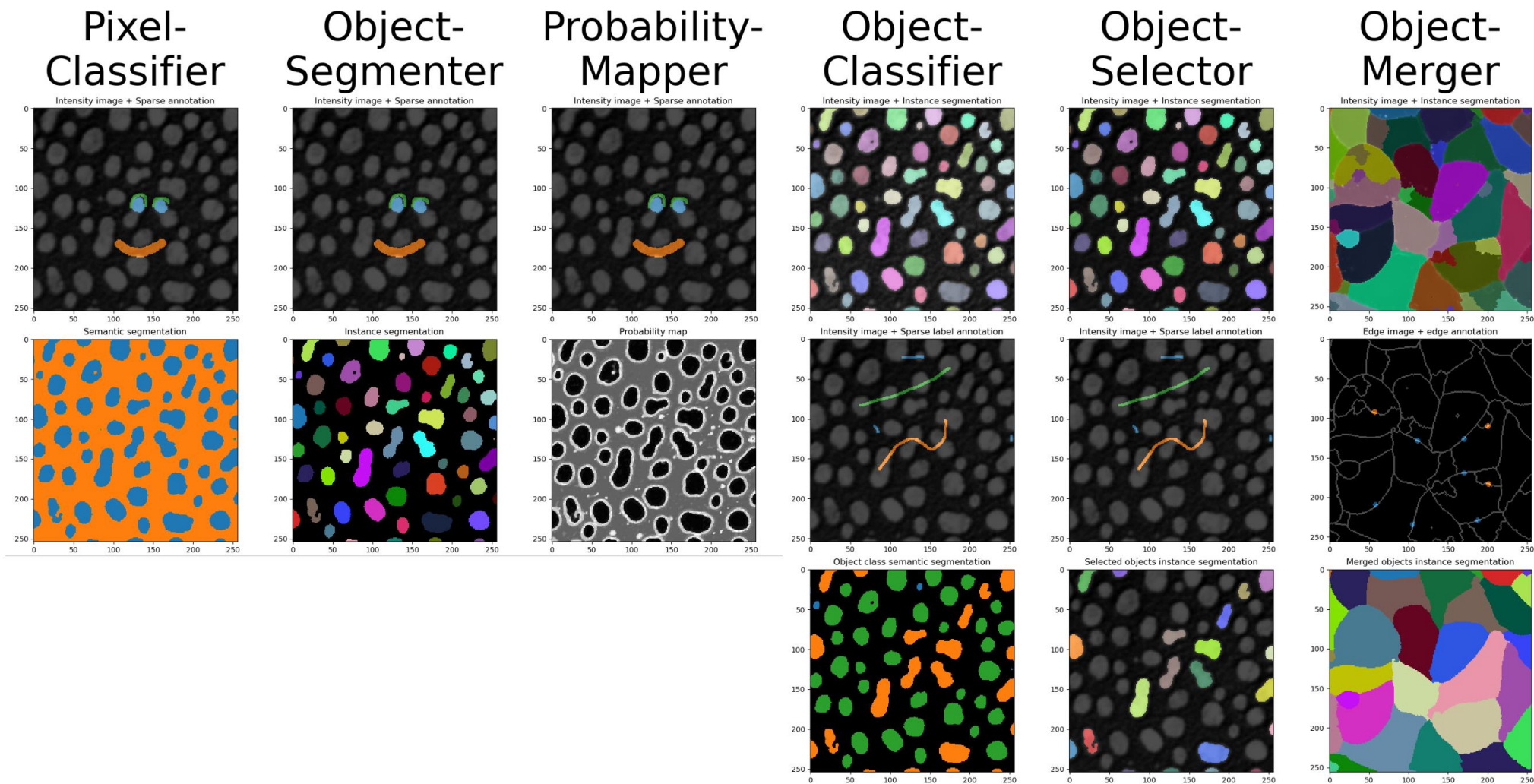
```
# determine object classification  
classification_result = classifier.predict(segmentation_result, image)  
cle.imshow(classification_result, labels=True)
```



Class label image

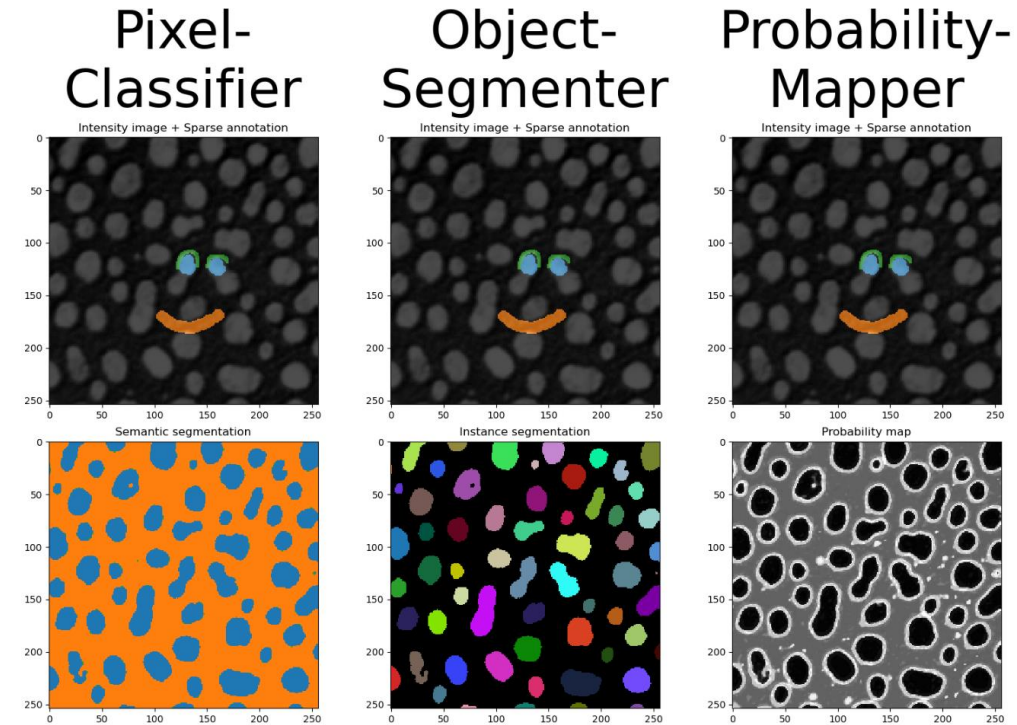
Object classification

Other classification / regression tasks



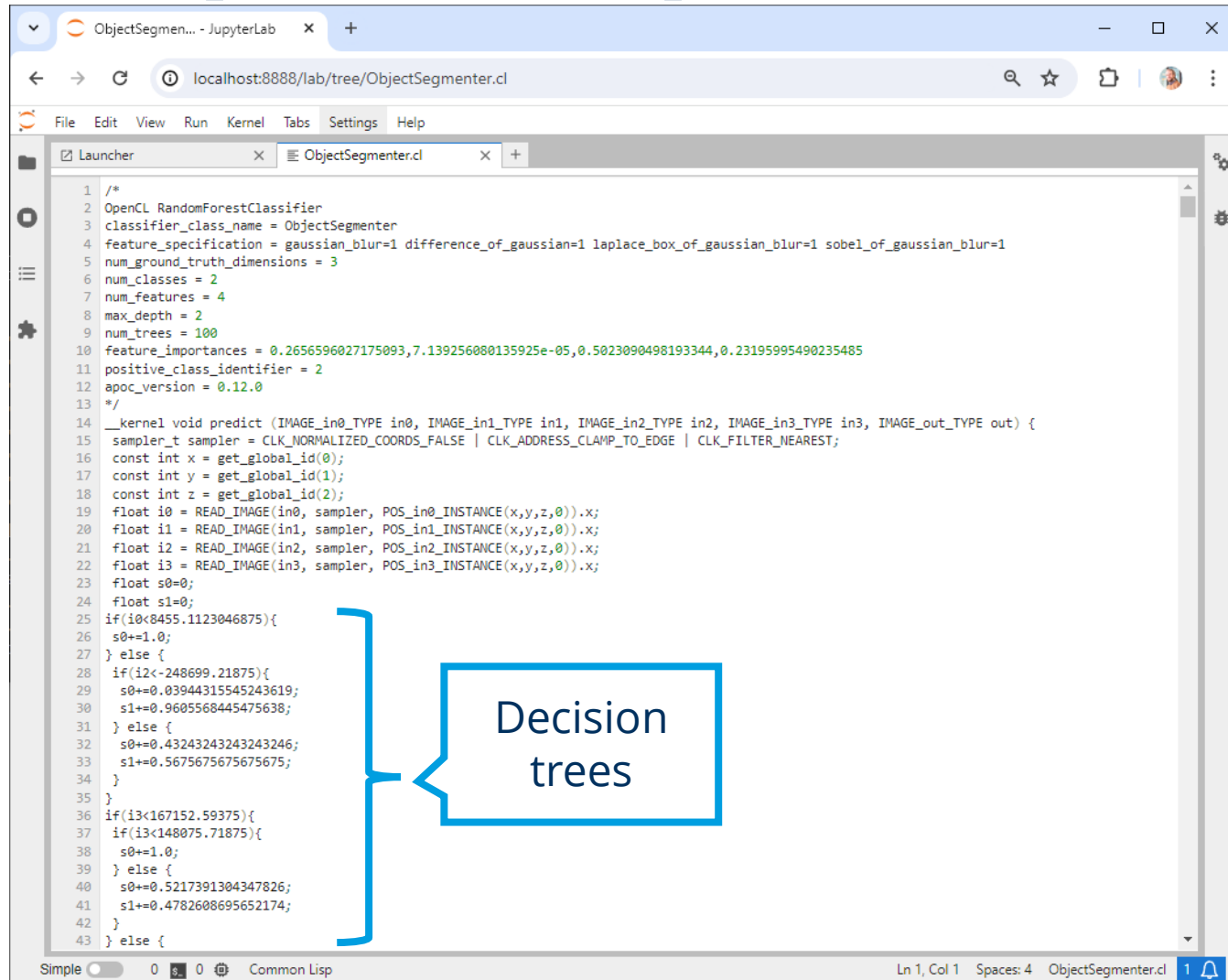
Quiz: Classification versus Regression

Which of these three solves a regression task?



Under the hood: clesperanto / OpenCL

classifier.cl files
can be *read*

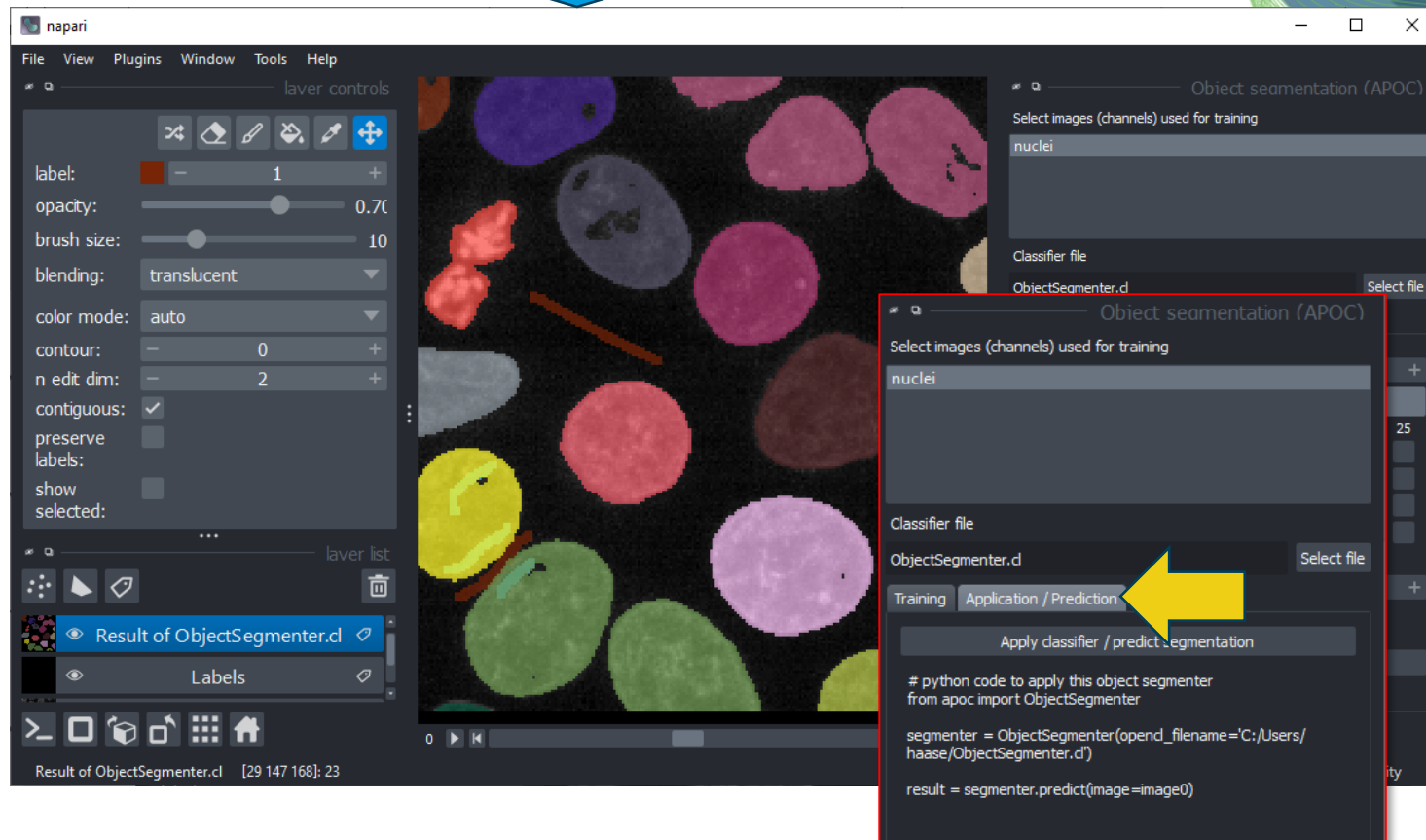
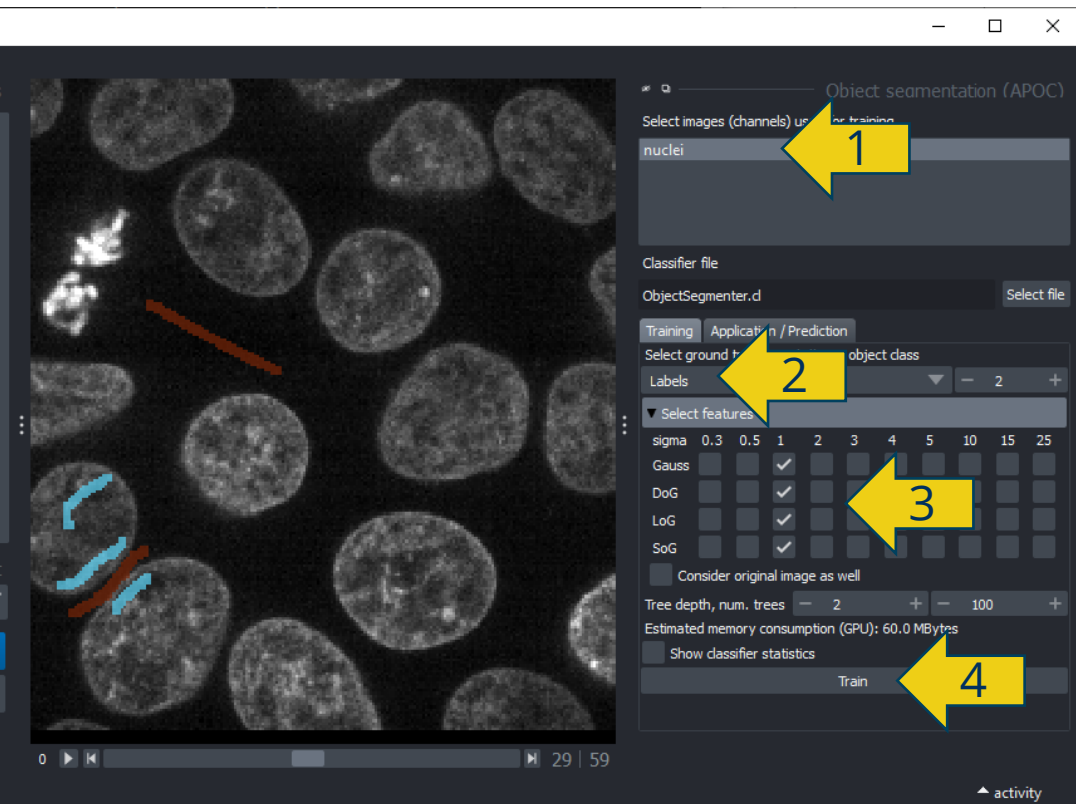


```
1 /*
2 OpenCL RandomForestClassifier
3 classifier_class_name = ObjectSegmener
4 feature_specification = gaussian_blur=1 difference_of_gaussian=1 laplace_box_of_gaussian_blur=1 sobel_of_gaussian_blur=1
5 num_ground_truth_dimensions = 3
6 num_classes = 2
7 num_features = 4
8 max_depth = 2
9 num_trees = 100
10 feature_importances = 0.2656596027175093,7.139256080135925e-05,0.5023090498193344,0.23195995490235485
11 positive_class_identifier = 2
12 apoc_version = 0.12.0
13 */
14 __kernel void predict (IMAGE_in0_TYPE in0, IMAGE_in1_TYPE in1, IMAGE_in2_TYPE in2, IMAGE_in3_TYPE in3, IMAGE_out_TYPE out) {
15     sampler_t sampler = CLK_NORMALIZED_COORDS_FALSE | CLK_ADDRESS_CLAMP_TO_EDGE | CLK_FILTER_NEAREST;
16     const int x = get_global_id(0);
17     const int y = get_global_id(1);
18     const int z = get_global_id(2);
19     float i0 = READ_IMAGE(in0, sampler, POS_in0_INSTANCE(x,y,z,0)).x;
20     float i1 = READ_IMAGE(in1, sampler, POS_in1_INSTANCE(x,y,z,0)).x;
21     float i2 = READ_IMAGE(in2, sampler, POS_in2_INSTANCE(x,y,z,0)).x;
22     float i3 = READ_IMAGE(in3, sampler, POS_in3_INSTANCE(x,y,z,0)).x;
23     float s0=0;
24     float s1=0;
25     if(i0<0.8455.1123046875){
26         s0+=1.0;
27     } else {
28         if(i2<-248699.21875){
29             s0+=0.03944315545243619;
30             s1+=0.9605568445475638;
31         } else {
32             s0+=0.4324324324324246;
33             s1+=0.5675675675675675;
34         }
35     }
36     if(i3<167152.59375){
37         if(i3<148075.71875){
38             s0+=1.0;
39         } else {
40             s0+=0.5217391304347826;
41             s1+=0.4782608695652174;
42         }
43     } else {
```

Decision trees

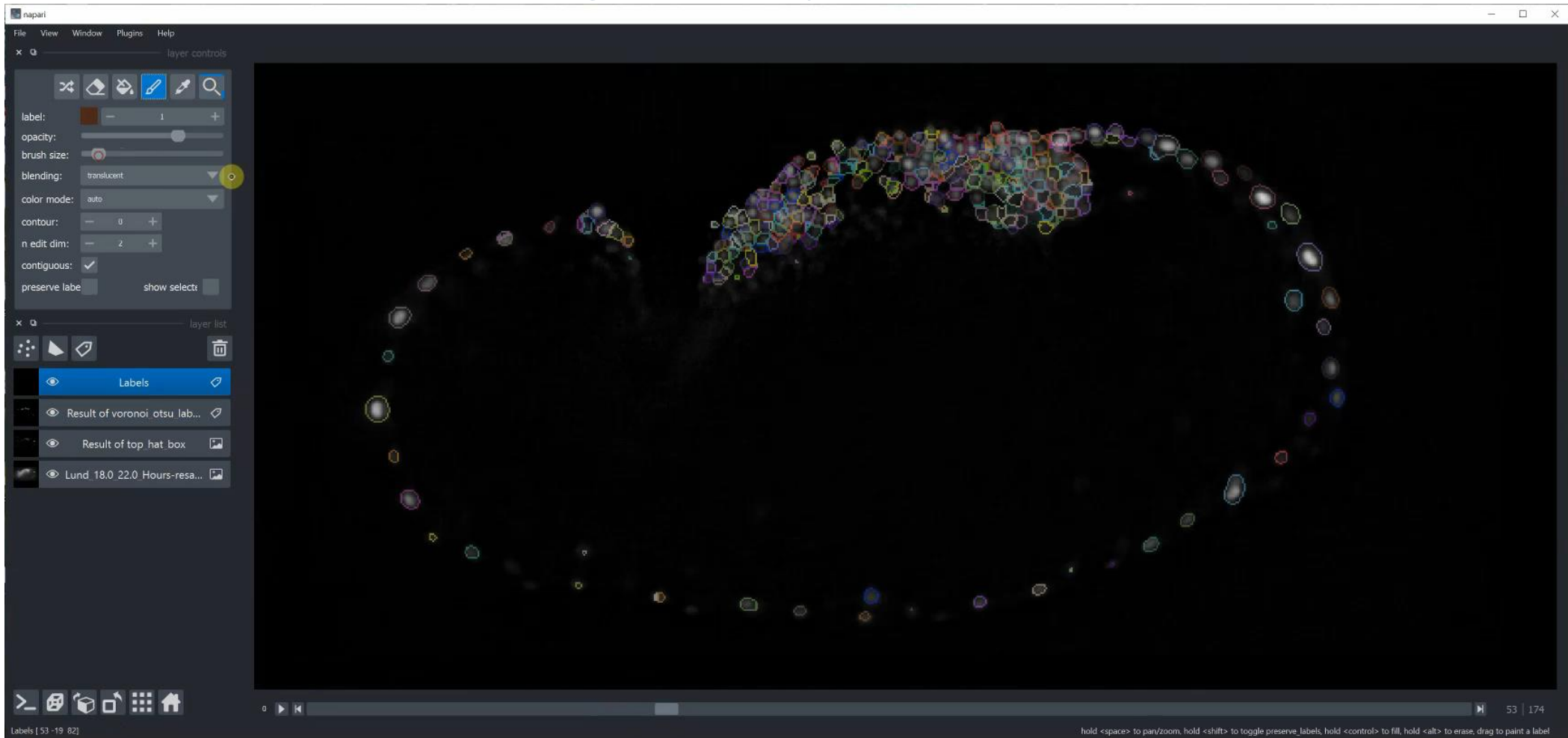
Graphical user interface: Object segmentation

- 1: Select image[s]
- 2: Select ground truth annotation
- 3: Select features
- 4: Train / predict



Graphical user interface: Object classification

Annotation / classification of segmented objects



Graphical user interface: Object classification

Inspect how the random forest classifier makes decisions

Note: Beware of correlated parameters!

The screenshot shows the napari GUI with the following components:

- Layer Controls (Left):** Includes tools for selection, pan, zoom, and layer properties like label, opacity, brush size, blending, rendering, color mode, and contour.
- Viewer (Center):** Displays three views of a kidney: the original grayscale image, a segmented view with white dots, and a classified view with multi-colored dots.
- Object Classification Panel (Right):** Titled 'Object classification (APOC)', it shows settings for image, labels, and model filename. The 'max depth' is set to 3 and 'num ensemble' is set to 100, both highlighted with a green box. Other features like mean intensity, standard deviation intensity, and pixel count are checked. A 'Run' button is at the bottom.
- Table (Bottom Left):** A table showing classifier statistics for three different models (1, 2, 3).

Dock widget 1

	1	2	3
area	0.010	0.056	0.000
mean_intensity	0.200	0.278	1.000
standard_deviation_intensity	0.030	0.000	0.000
mean_max_distance_to_centroid_ratio	0.270	0.222	0.000
average_distance_of_n_nearest_neighbors=1	0.120	0.111	0.000
average_distance_of_n_nearest_neighbors=6	0.170	0.111	0.000
average_distance_of_n_nearest_neighbors=10	0.200	0.222	0.000

Graphical user interface: Object classification

Inspect how the random forest classifier makes decisions

Note: Beware of correlated parameters!

Dock widget 2

	1	2	3
area	0.060	0.000	0.000
mean_intensity	0.330	0.167	0.000
standard_deviation_intensity	0.040	0.111	0.000
mean_max_distance_to_centroid_ratio	0.260	0.444	1.000
average_distance_of_n_nearest_neighbors=6	0.310	0.278	0.000

Graphical user interface: Object classification

Inspect how the random forest classifier makes decisions

Note: Beware of correlated parameters!

Pixel count

Mean intensity

Standard deviation of intensity

Average distance to 6 nearest neighbors

Mean/max distance-to-centroid ratio

	1	2	3
area	0.060	0.000	0.000
mean_intensity	0.330	0.167	0.000
standard_deviation_intensity	0.040	0.111	0.000
mean_max_distance_to_centroid_ratio	0.260	0.444	1.000
average_distance_of_n_nearest_neighbors=6	0.310	0.278	0.000

Supervised and Unsupervised Machine Learning for Bio-image Analysis

Robert Haase

Reusing materials from Johannes Soltwedel, Till Korten, Johannes Müller, Laura Žigutytė (TU Dresden), Ryan Savill (MPI-CBG), Matthias Täschner (ScaDS.AI/Uni Leipzig) and the Scikit-learn community.

Funded by



Bundesministerium
für Bildung
und Forschung

SACHSEN

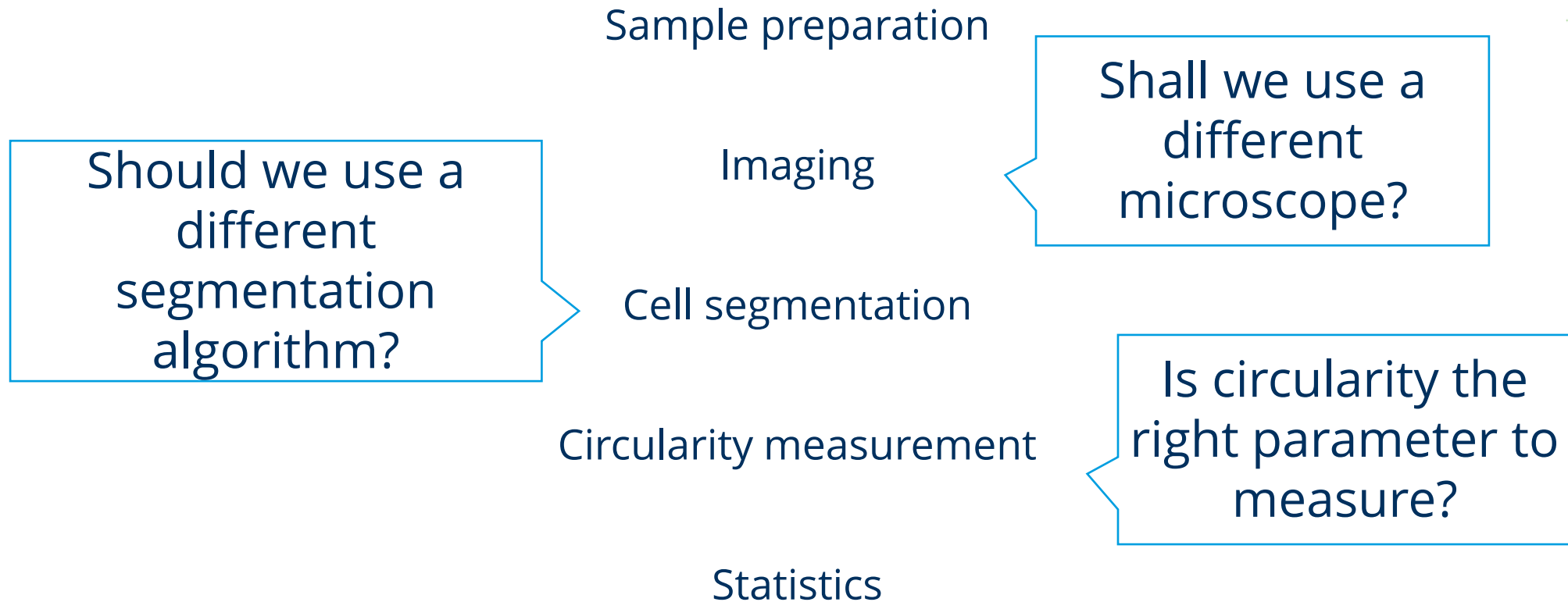


Diese Maßnahme wird gefördert durch die Bundesregierung aufgrund eines Beschlusses des Deutschen Bundestages. Diese Maßnahme wird mitfinanziert durch Steuermittel auf der Grundlage des von den Abgeordneten des Sächsischen Landtags beschlossenen Haushaltes.

Hypothesis-driven quantitative biology

Hypothesis: Cell shape can be influenced by modifying X.

Null-Hypothesis: Circularity of modified cells is similar to cells in the control group.



Hypothesis *generating* quantitative biology

Hypothesis: ~~Cell shape can be influenced by modifying X.~~

Question: Which image-derived parameter is influenced when modifying X?

Sample preparation

Imaging

Which segmentation algorithms allow measurements that show a relationship with X?

Cell segmentation algorithm A, algorithm B, algorithm C

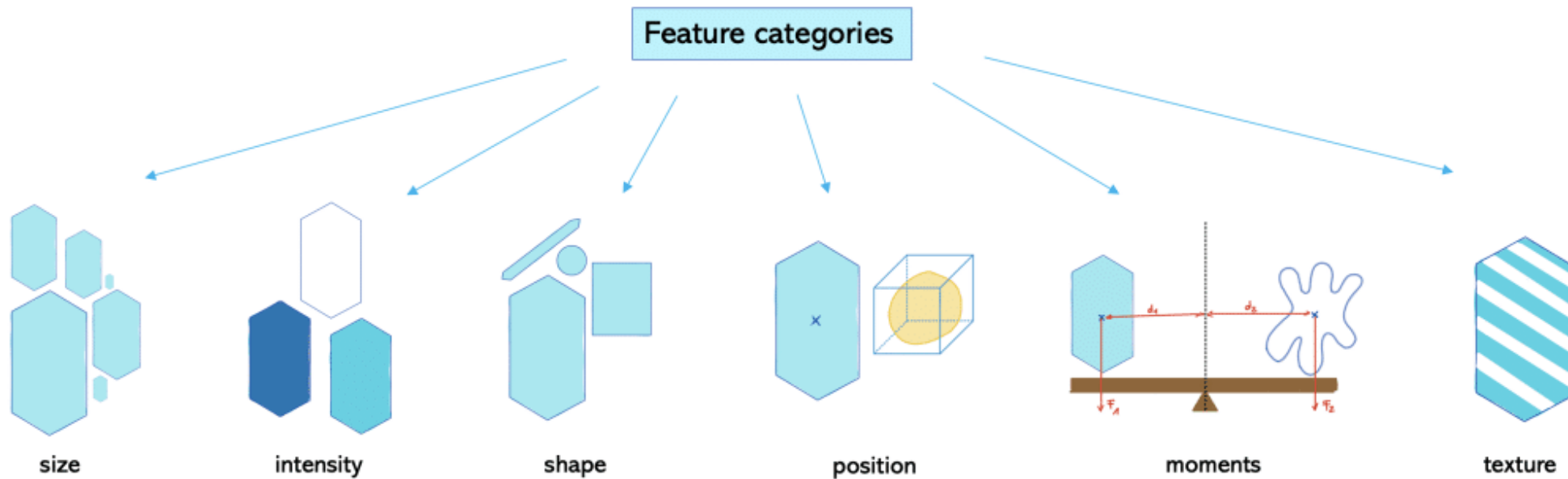
Why?

Measurement of circularity, solidity, elongation, extend, texture, intensity, topology ...

Statistics

Which parameter shows any relationship with X?

Feature selection



Which of these features reflect the phenotype we are perceiving?

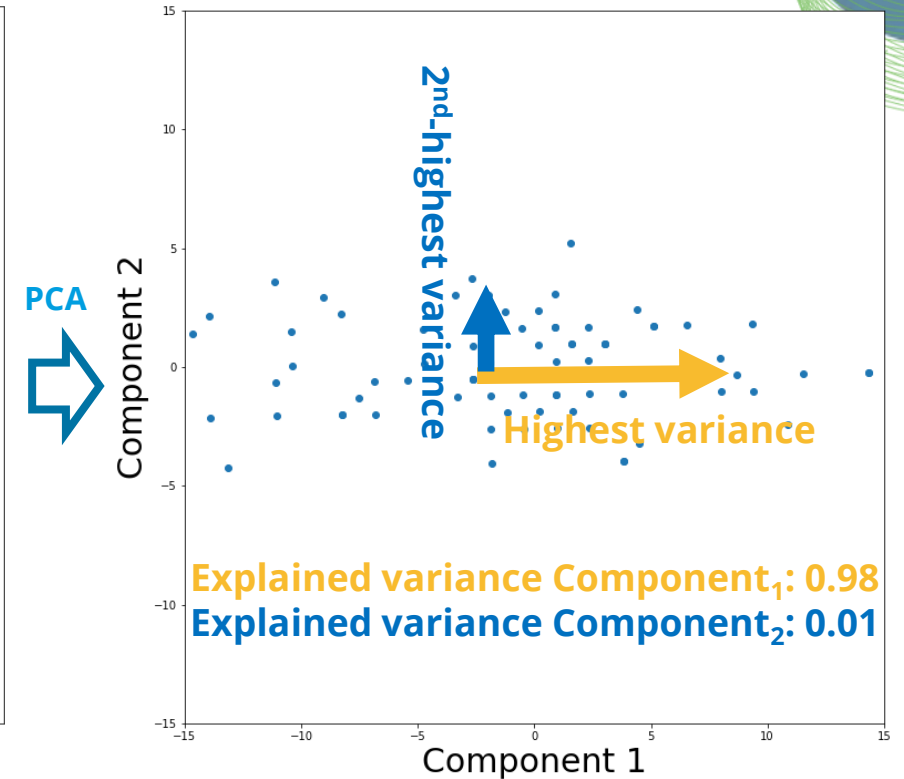
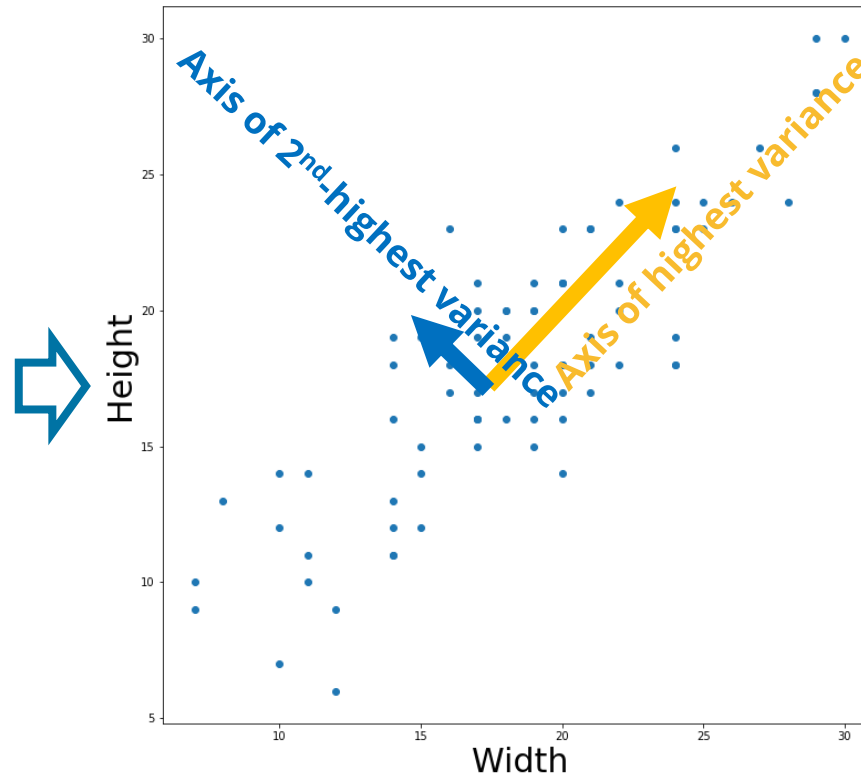
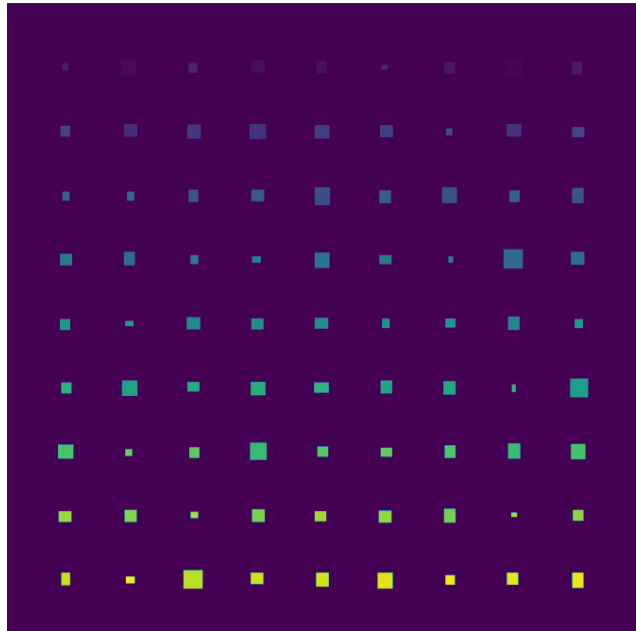
Feature selection: challenges

- Features are not independent
 - Area and diameter
 - Roundness, circularity, solidity, extent, aspect ratio, elongation, Feret's diameter, ...
- Best classification most likely involves multiple features
- Vast amount of features can hardly be visualized
- Need for dimensionality reduction
 - Principal component analysis (PCA)
 - t-Distributed Stochastic Neighbour Embedding (t-SNE)
 - Uniform Manifold Approximation and Projection (UMAP)
- Grouping objects (clustering)

PCA: Principal Component Analysis

Decomposes data into linear combinations of features that explain the highest variance

Example: Squares of different size

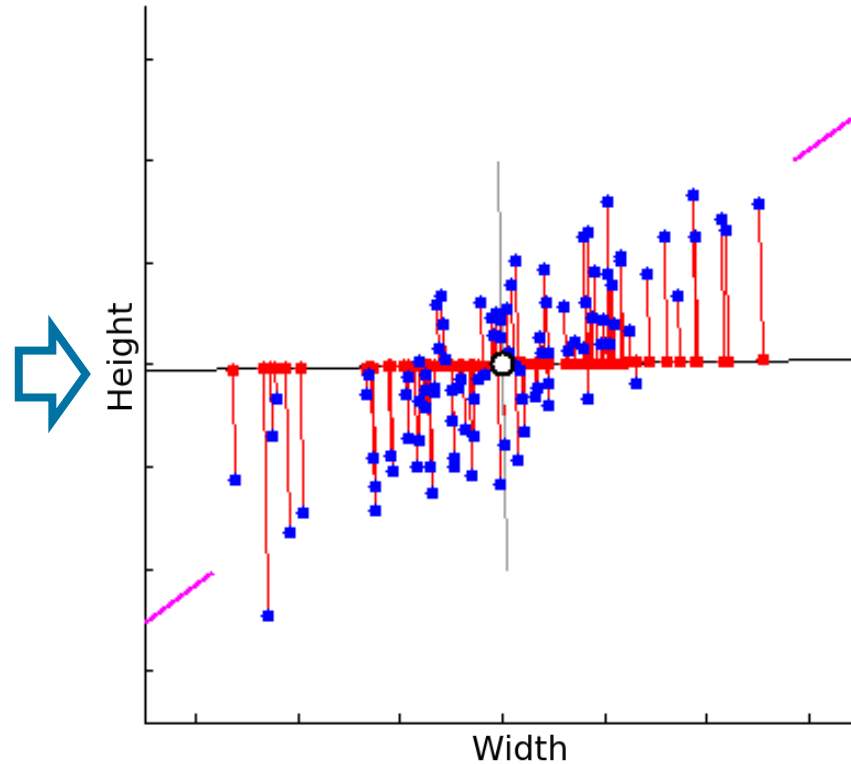
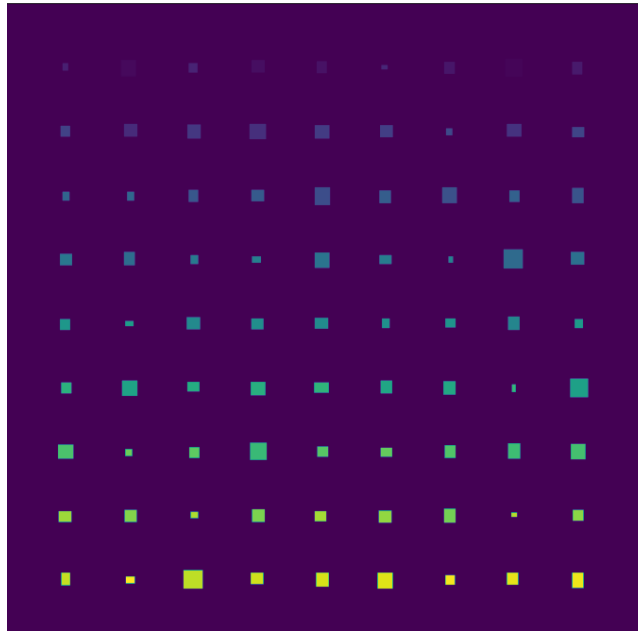


→ PCA transforms width/height measurements into a coordinate system that explains existing variance better

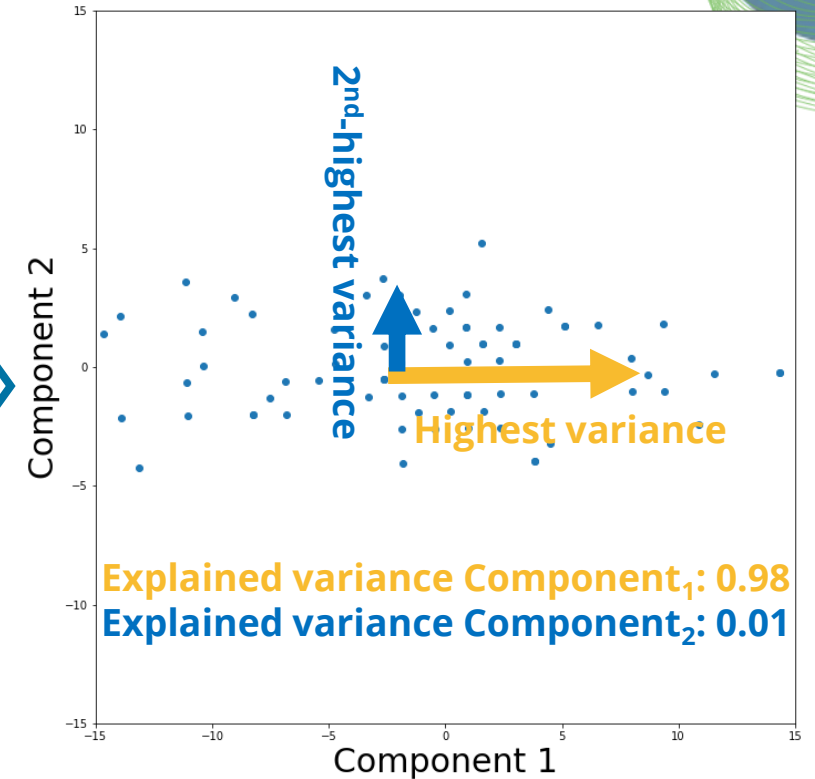
PCA: Principal Component Analysis

Decomposes data into linear combinations of features that explain the highest variance

Example: Squares of different size



PCA



→ PCA transforms width/height measurements into a coordinate system that explains existing variance better

PCA in Python: `sklearn.decomposition.PCA`

- Import package

```
from sklearn.decomposition import PCA
```

- Apply PCA

```
pca = PCA(n_components=2)  
pca.fit(standardized_data)
```

- Transform data into new coordinate system

```
transformed_data = pca.transform(data)
```

Important!

Always check the explained variance along the PCA component axes!

```
pca.explained_variance_ratio_
```

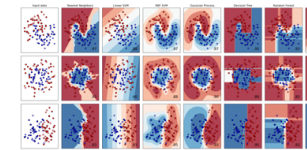
```
array([0.98773142, 0.01226858])
```

The screenshot shows the scikit-learn website homepage. At the top, there is a navigation bar with links for 'Install', 'User Guide', 'API', 'Examples', 'Community', and 'More'. Below the navigation bar, the 'scikit-learn' logo is prominently displayed, followed by the tagline 'Machine Learning in Python'. A list of key features is shown: 'Simple and efficient tools for predictive data analysis', 'Accessible to everybody, and reusable in various contexts', 'Built on NumPy, SciPy, and matplotlib', and 'Open source, commercially usable - BSD license'. There are also buttons for 'Getting Started', 'Release Highlights for 1.1', and 'GitHub'.

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition, and more...

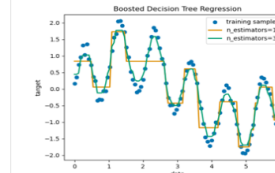


Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices, and more...

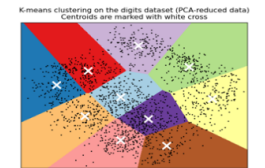


Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes, and more...



Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency

Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tun-

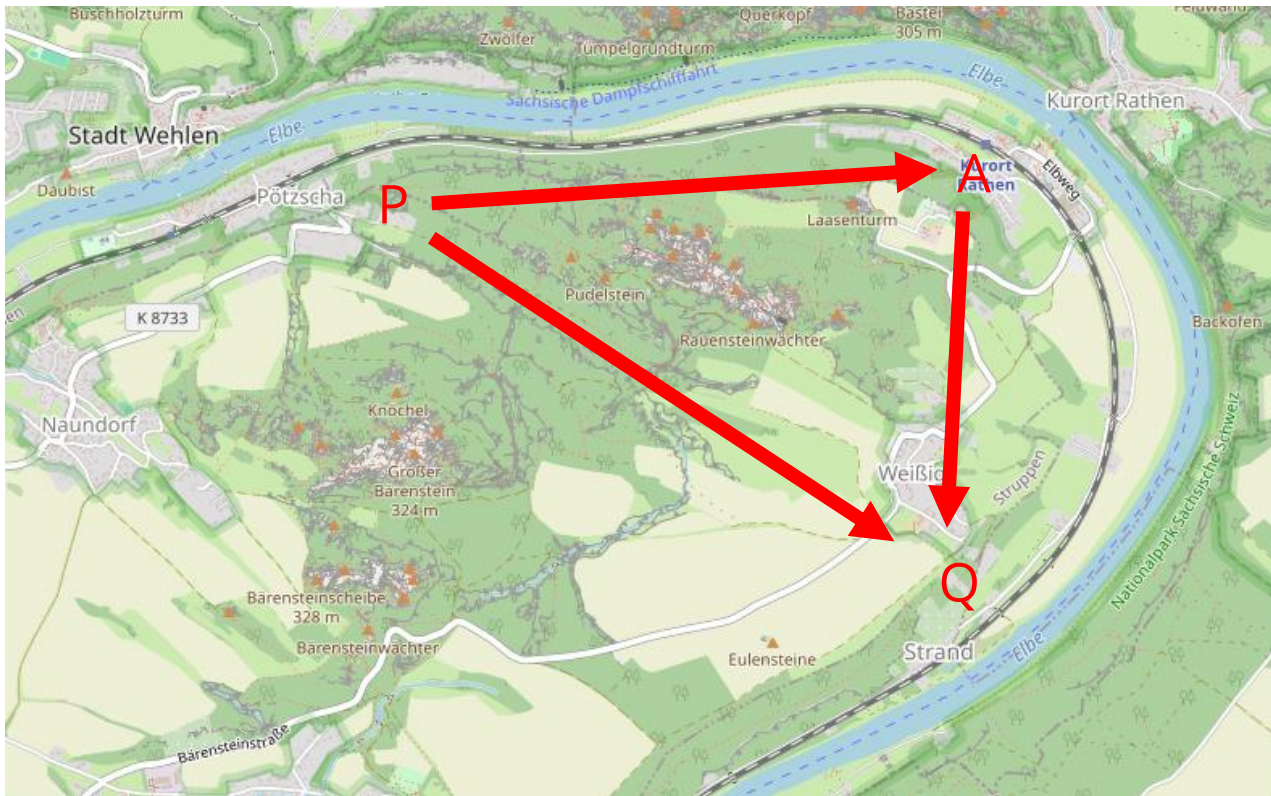
Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.

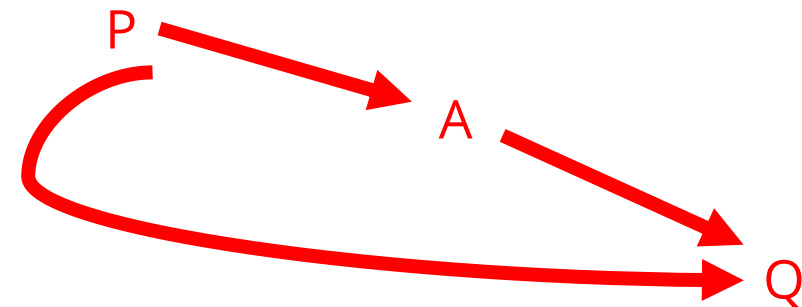
Non-Euclidian spaces

Not all dimensions (features) might be distances



Use travel time between P and Q as metric for distance

→ Travelling from Stadt Wehlen to Strand by bike is probably faster if you make a detour through Rathen



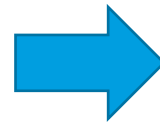
Dimensionality reduction: UMAP

Uniform Manifold Approximation Projection

Preserve local distances at the expense of global distortions

Many dimensions

	count	mean	std
label	44.0	22.500000	12.845233
area	44.0	401.863636	202.852288
bbox_area	44.0	542.750000	295.106376
equivalent_diameter	44.0	21.781085	6.174086
convex_area	44.0	423.295455	216.613747
max_intensity	44.0	234.909091	17.517856
mean_intensity	44.0	190.116971	15.034153
min_intensity	44.0	128.000000	0.000000
extent	44.0	0.758804	0.063276
local_centroid-0	44.0	11.439824	4.126230
local_centroid-1	44.0	10.138666	3.491815
solidity	44.0	0.953153	0.024749
feret_diameter_max	44.0	26.382434	8.915046
major_axis_length	44.0	25.876797	9.591558
minor_axis_length	44.0	18.872898	5.158791
orientation	44.0	0.053057	0.691430
eccentricity	44.0	0.600434	0.165688
standard_deviation_intensity	44.0	29.556705	5.507399
aspect_ratio	44.0	1.374342	0.397611
roundness	44.0	0.762889	0.156695
circularity	44.0	0.918858	0.133288



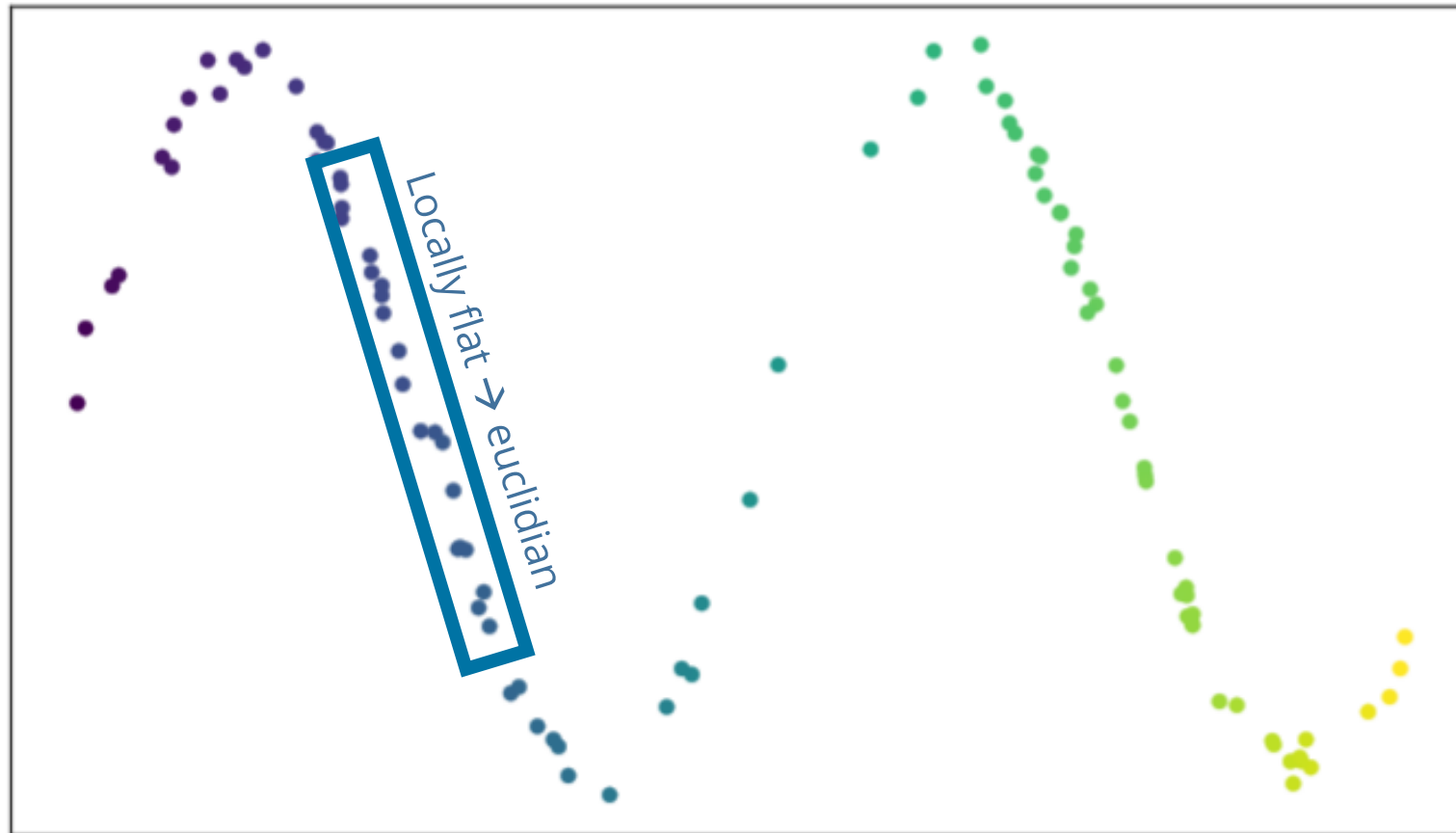
UMAP 2



UMAP 1

Dimensionality reduction: UMAP

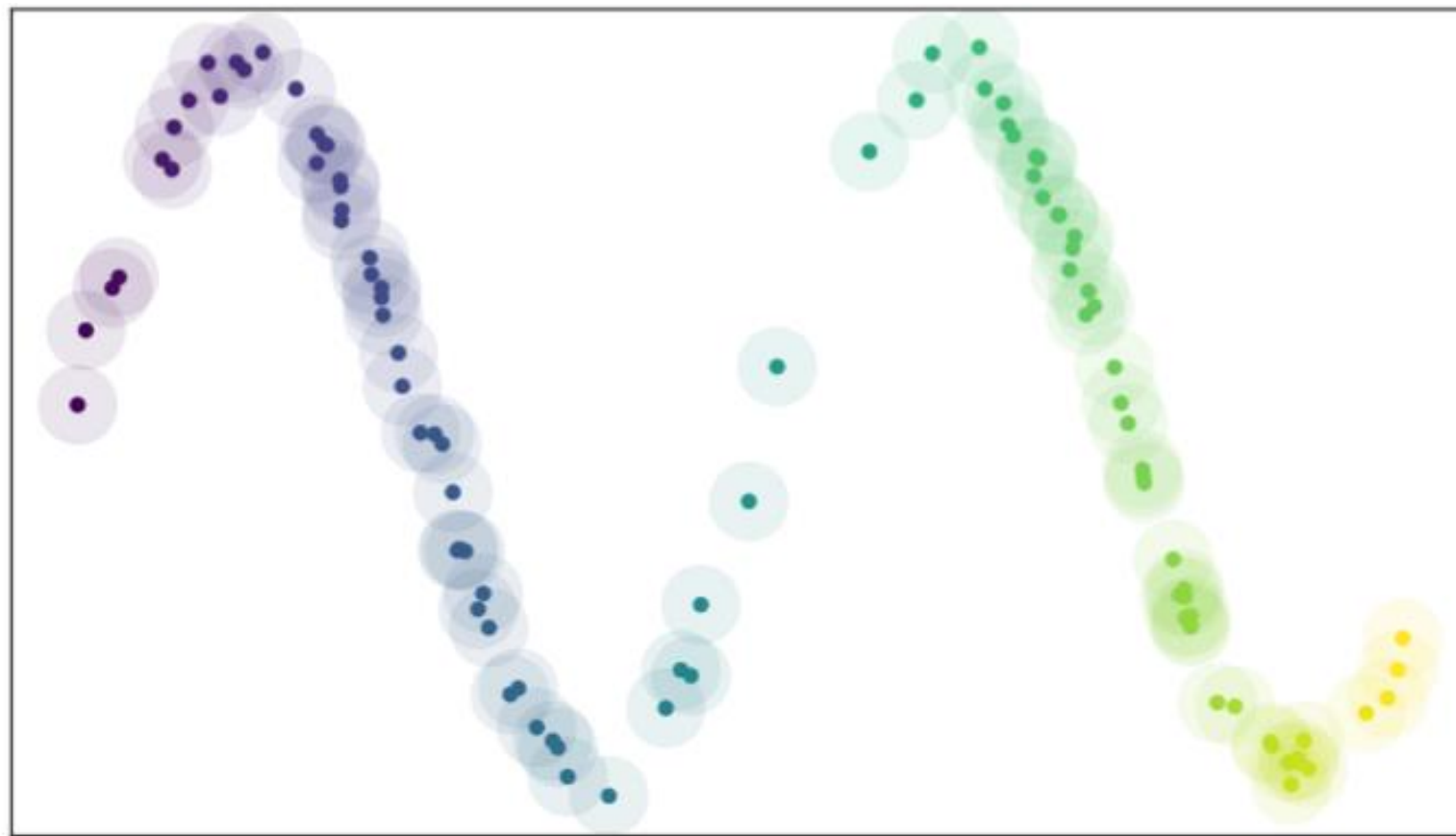
Initial situation: Our data suggests an underlying structure (“topology”)



Goal:

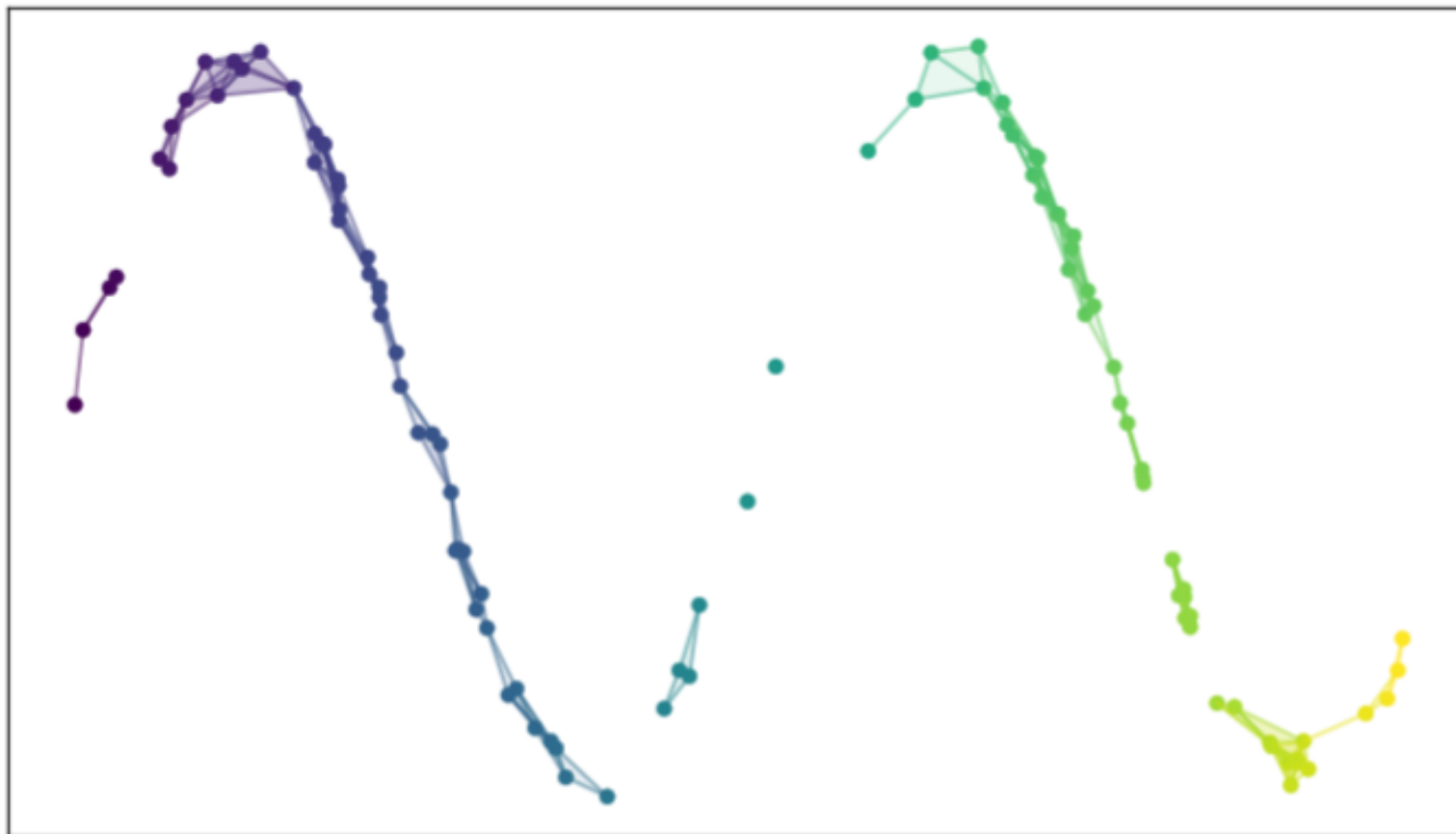
Reconstruct underlying topology to identify a space that best explains differences in our data

Dimensionality reduction



Naïve approach:
Points within a
defined radius are
considered
neighbors

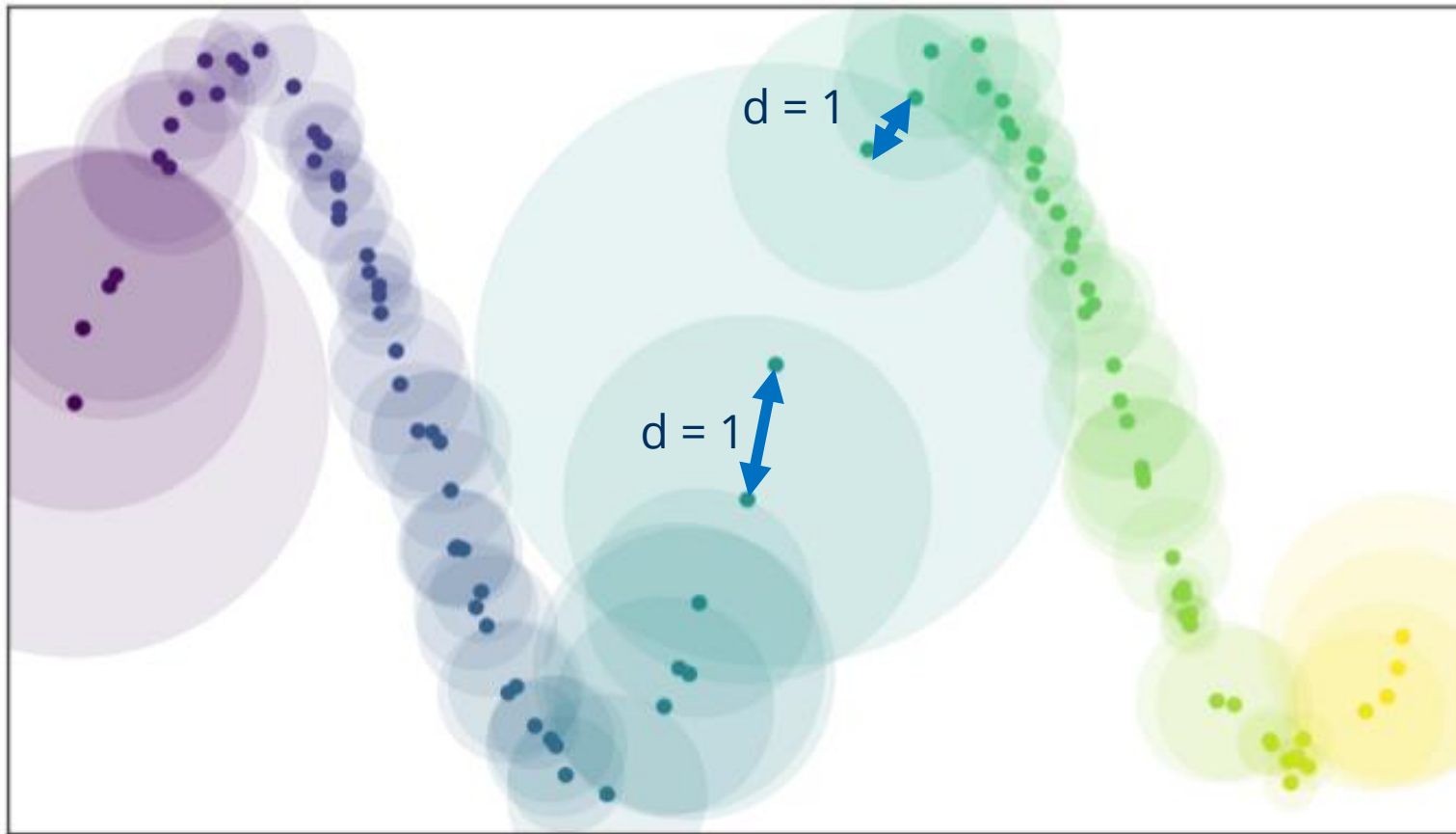
Dimensionality reduction



Naïve approach:
Points within a defined radius are considered neighbors

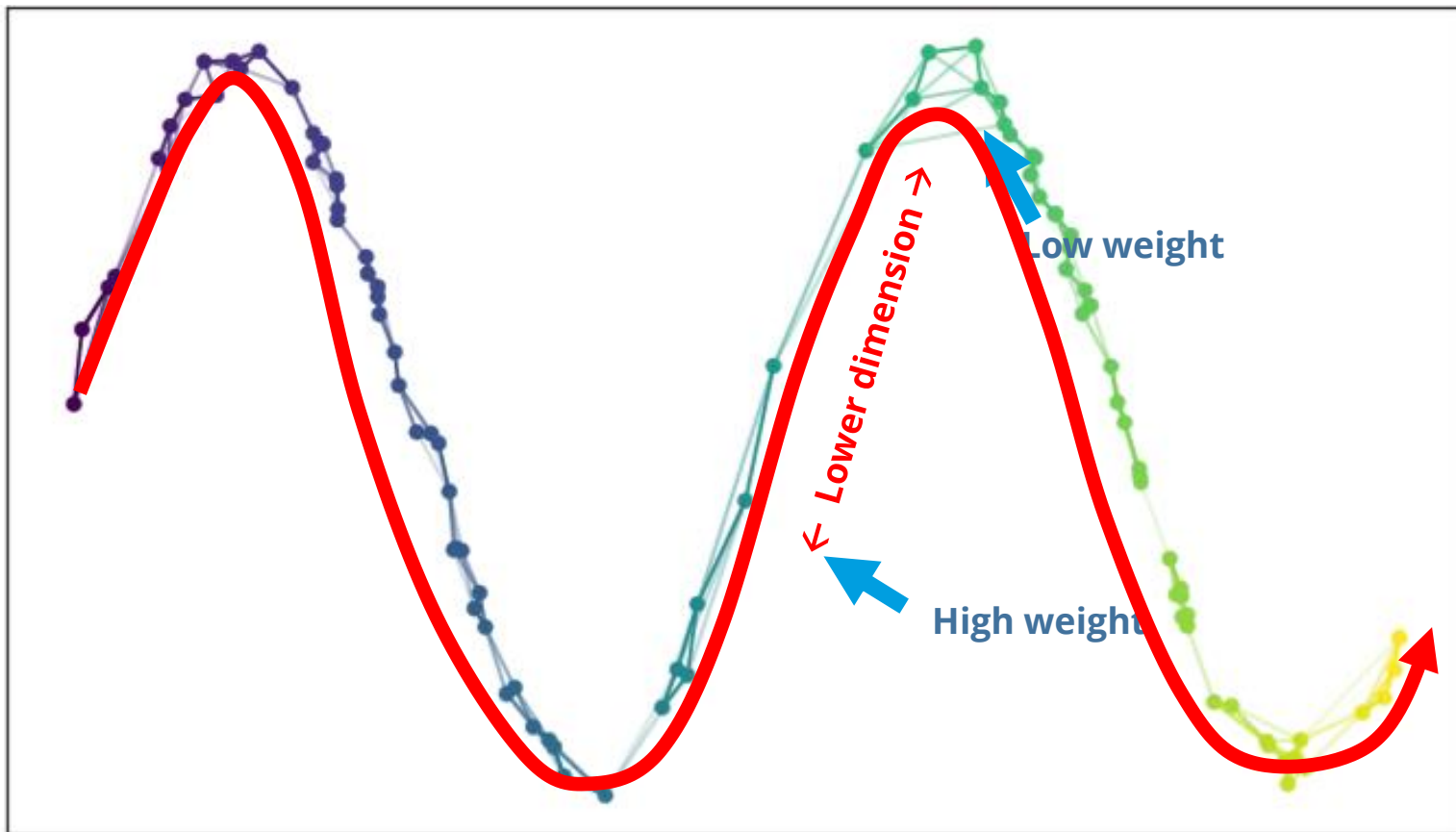
Result:
Neighborhood graph with interruptions

Dimensionality reduction: UMAP



Approach:
Normalize distances
by dividing by the
average distance to
 n nearest neighbors
(Example: $n=1$)

Reduce dimensionality preserving fuzzy topology



Approach:

Normalize distances by dividing by the average distance to n nearest neighbors

Build a graph considering normalized distances

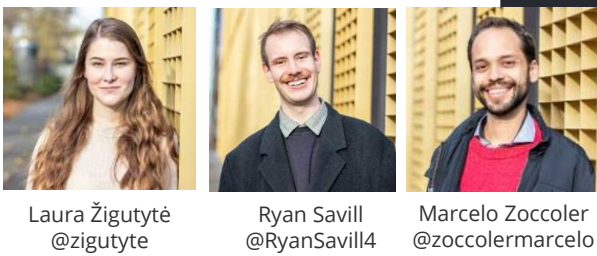
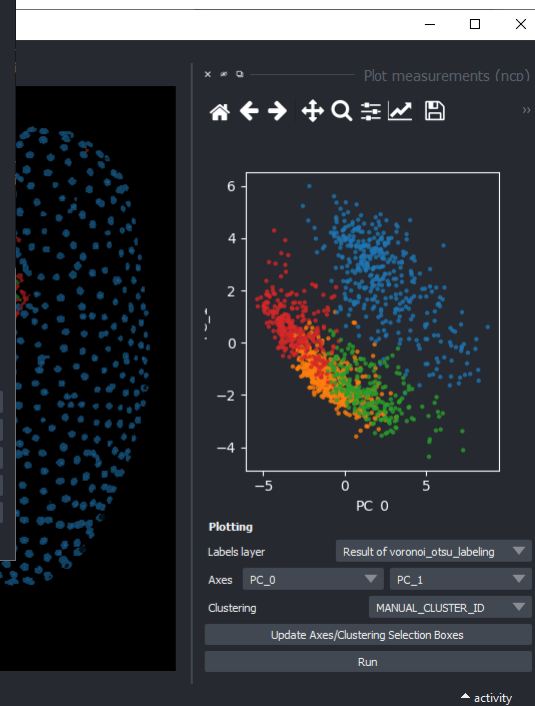
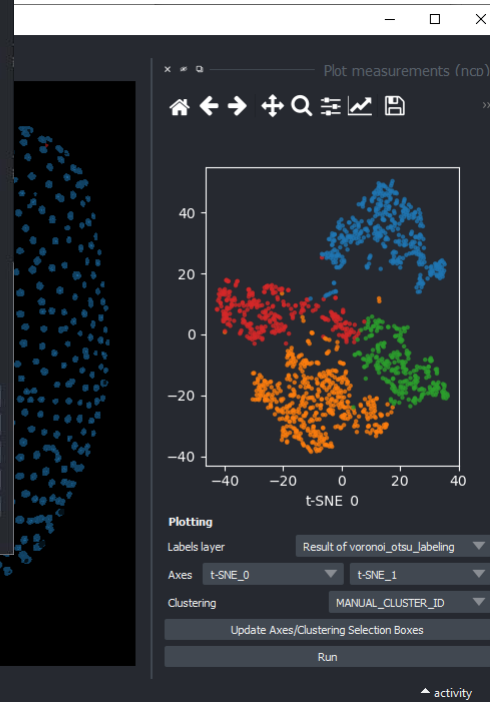
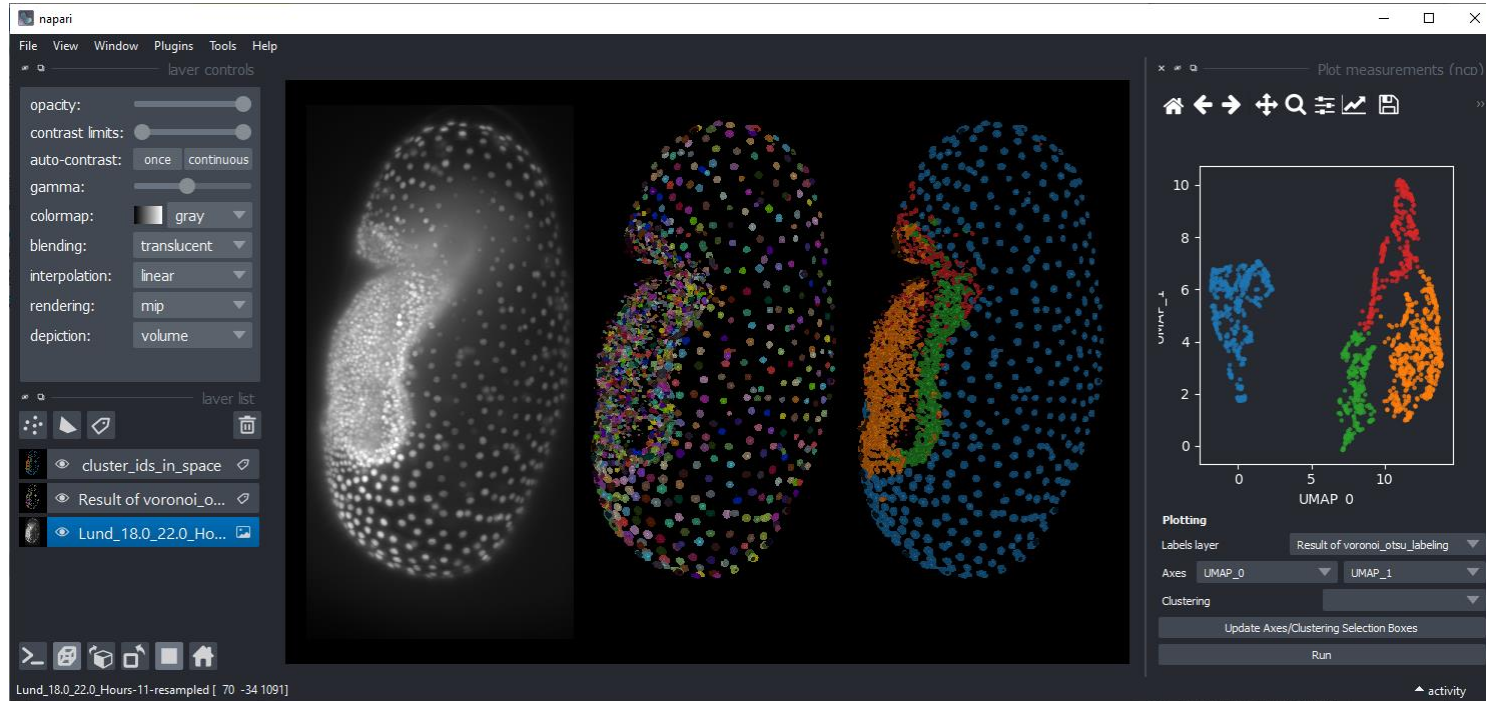
Project data into lower dimensional space

Dimensionality reduction

Uniform manifold approximation and projection (UMAP)

t-distributed stochastic neighbor embedding (t-SNE)

Principal component analysis (PCA)



burs-11-resampled [69 728 889]

Lund_18,0_22,0_Hours-11-resampled [70 -31 1003]

UMAP in Python

Selecting columns from a pandas DataFrame

```
[8]: measurements.describe().T
```

	count	mean	std
label	44.0	22.500000	12.845233
area	44.0	401.863636	202.852288
bbox_area	44.0	542.750000	295.106376
equivalent_diameter	44.0	21.781085	6.174086
convex_area	44.0	423.295455	216.613747
max_intensity	44.0	234.909091	17.517856
mean_intensity	44.0	190.116971	15.034153
min_intensity	44.0	128.000000	0.000000
extent	44.0	0.758804	0.063276
local_centroid-0	44.0	11.439824	4.126230
local_centroid-1	44.0	10.138666	3.491815
solidity	44.0	0.953153	0.024749
feret_diameter_max	44.0	26.382434	8.915046
major_axis_length	44.0	25.876797	9.591558
minor_axis_length	44.0	18.872898	5.158791
orientation	44.0	0.053057	0.691430
eccentricity	44.0	0.600434	0.165688
standard_deviation_intensity	44.0	29.556705	5.507399
aspect_ratio	44.0	1.374342	0.397611
roundness	44.0	0.762889	0.156695
circularity	44.0	0.918858	0.133288

```
[9]: selected_measurements = measurements[[  
      'area',  
      'equivalent_diameter',  
      'convex_area',  
      'max_intensity',  
      'mean_intensity',  
      'min_intensity',  
      'extent',  
      'solidity',  
      'feret_diameter_max',  
      'major_axis_length',  
      'minor_axis_length',  
      'eccentricity',  
      'standard_deviation_intensity',  
      'aspect_ratio',  
      'roundness',  
      'circularity']]  
selected_measurements.describe().T
```

Select *reasonable* features

```
[9]:
```

	count	mean	std
area	44.0	401.863636	202.852288
equivalent_diameter	44.0	21.781085	6.174086
convex_area	44.0	423.295455	216.613747
max_intensity	44.0	234.909091	17.517856
mean_intensity	44.0	190.116971	15.034153
min_intensity	44.0	128.000000	0.000000
extent	44.0	0.758804	0.063276
solidity	44.0	0.953153	0.024749
feret_diameter_max	44.0	26.382434	8.915046
major_axis_length	44.0	25.876797	9.591558
minor_axis_length	44.0	18.872898	5.158791
eccentricity	44.0	0.600434	0.165688
standard_deviation_intensity	44.0	29.556705	5.507399
aspect_ratio	44.0	1.374342	0.397611
roundness	44.0	0.762889	0.156695
circularity	44.0	0.918858	0.133288

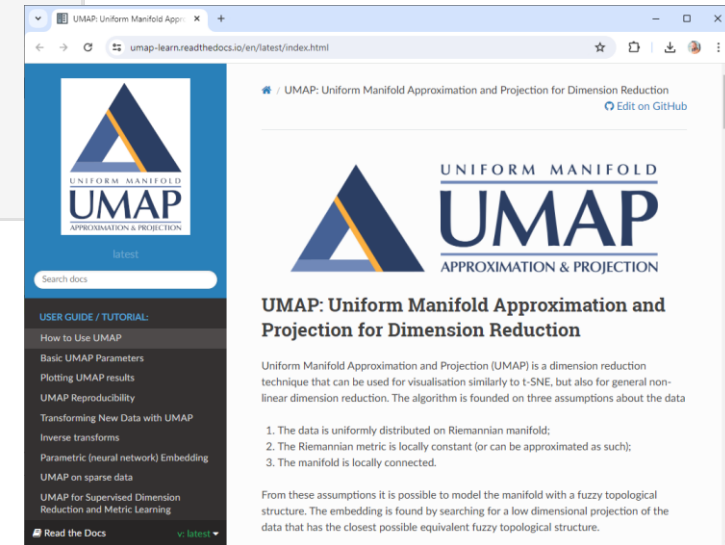
UMAP in Python

```
[10]: # configure UMAP algorithm
umap = UMAP(n_neighbors=5, n_components=2)

# apply algorithm
transformed_data = umap.fit_transform(selected_measurements.values.tolist())

# store results back in table
measurements['UMAP0'] = transformed_data[:,0]
measurements['UMAP1'] = transformed_data[:,1]
```

Data
conversion



Annotating UMAPs in Napari

Draw a lasso here to visualize which objects the data points correspond to

The screenshot displays the Napari interface. On the left, the 'layer controls' panel shows settings for the 'labels' layer, including label number (1), opacity (0.00), brush size (10), and blending mode (translucent). The 'layer list' at the bottom shows 'cluster_ids_in_space' and 'labels'. The central panel shows a microscopy image with orange and blue cell annotations. On the right, a UMAP plot shows the data points with axes labeled 'UMAP0' and 'UMAP1'. The 'Plotting' panel below the plot shows 'Labels layer' set to 'labels', 'Axes' set to 'UMAP0' and 'UMAP1', and 'Clustering' set to 'MANUAL_CLUSTER_ID'. A status bar at the bottom provides keyboard shortcuts: 'use <1> for activate the label eraser, use <2> for activate the paint brush, use <3> for activate the fill bucket, use <4> for pick mode'.

Interpreting annotations in Napari

The screenshot displays the Napari application window. On the left, the 'layer controls' panel is visible, showing settings for the 'labels' layer, including label, opacity, brush size, blending, color mode, contour, n edit dim, contiguous, preserve labels, and show selected. The central panel shows a microscopy image with segmented cells in orange and blue. On the right, the 'Plot measurements (ncv)' panel is open, showing a scatter plot of 'mean_intensity' (y-axis, 140-220) versus 'area' (x-axis, 0-600). The plot shows a positive correlation between area and mean intensity. Below the plot, the 'Plotting' panel is visible, showing the 'Labels layer' selected, with 'area' and 'mean_intensity' chosen as axes, and 'MANUAL_CLUSTER_ID' as the clustering method. A callout box points to the plot axes with the text: 'Switch plot axes to see relationships between annotation and features'. At the bottom of the plot panel, there are buttons for 'Update Axes/Clustering Options' and 'Plot'. A footer note at the bottom of the plot panel reads: 'use <1> for activate the label eraser, use <2> for activate the paint brush, use <3> for activate the fill bucket, use <4> for pick mode'.

Correlation statistics

```
[16]: def colorize(styler):
      styler.background_gradient(axis=None, cmap="PiYG")
      return styler

df = measurements.corr().T
df.style.pipe(colorize)
```

[16]:

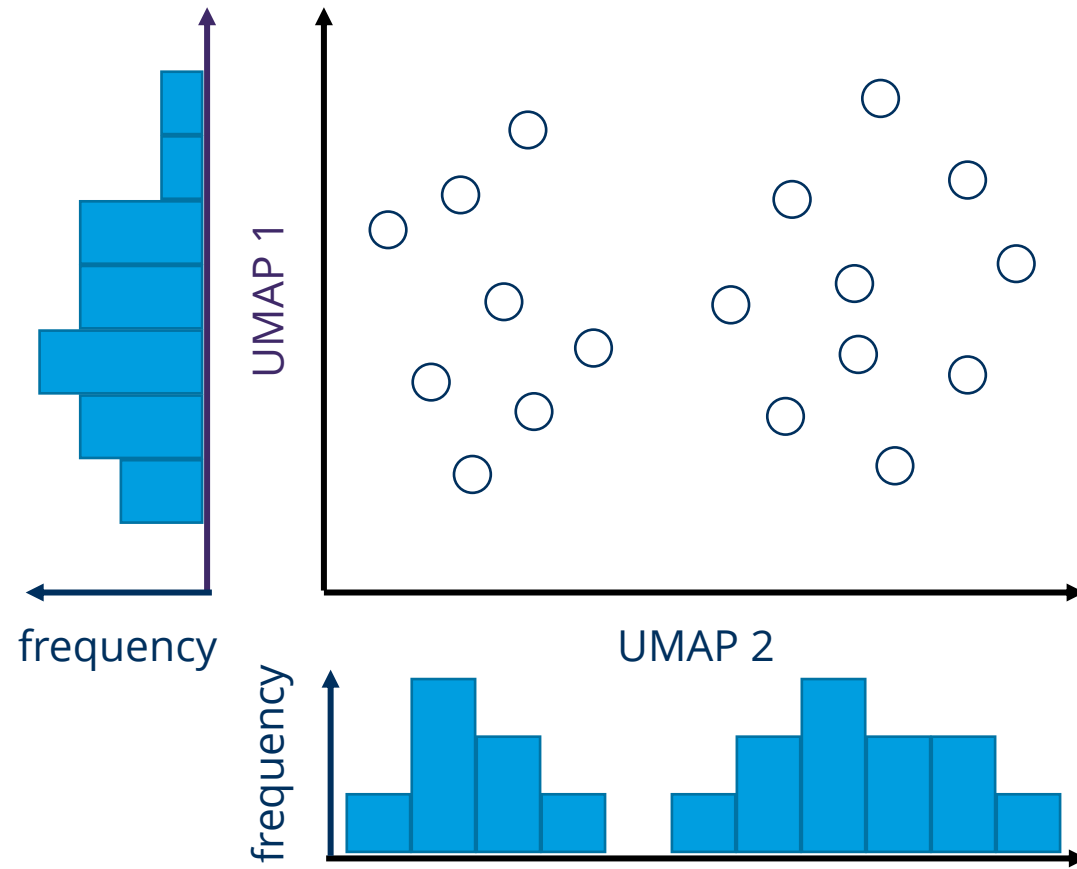
	label	area	bbox_area	equivalent_diameter	convex_area	max_intensity	mean_intensity	min_intensity	extent	local_centroid-0	local_centroid-1	solidity	feret_diameter_max	major_axis_length	minor_axis_length	orientation	eccentricity
label	1.000000	0.261682	0.223070	0.249249	0.250594	0.110791	0.235692	nan	0.031673	0.177363	0.227746	0.090163	0.208067	0.198908	0.237521	0.319053	0.059804
area	0.261682	1.000000	0.973718	0.978723	0.997560	0.511730	0.530250	nan	-0.362472	0.847281	0.935689	-0.243908	0.930981	0.911069	0.859240	0.280673	0.348585
bbox_area	0.223070	0.973718	1.000000	0.948328	0.985584	0.481524	0.476951	nan	-0.546728	0.902854	0.904551	-0.416707	0.973189	0.967337	0.752580	0.213080	0.479196
equivalent_diameter	0.249249	0.978723	0.948328	1.000000	0.974614	0.633984	0.618553	nan	-0.395696	0.858779	0.947036	-0.266587	0.931696	0.904412	0.904698	0.197456	0.363799
convex_area	0.250594	0.997560	0.985584	0.974614	1.000000	0.506730	0.517356	nan	-0.413323	0.862417	0.934090	-0.305706	0.948048	0.932682	0.832264	0.263176	0.389269
max_intensity	0.110791	0.511730	0.481524	0.633984	0.506730	1.000000	0.825115	nan	-0.324093	0.504879	0.603305	-0.253635	0.536089	0.502524	0.645600	-0.139025	0.246172
mean_intensity	0.235692	0.530250	0.476951	0.618553	0.517356	0.825115	1.000000	nan	-0.160940	0.412859	0.609264	-0.077797	0.458515	0.422638	0.707711	0.132754	0.017030
min_intensity	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
extent	0.031673	-0.362472	-0.546728	-0.395696	-0.413323	-0.324093	-0.160940	nan	1.000000	-0.631158	-0.375580	0.853431	-0.631776	-0.664733	-0.062873	0.252915	-0.756019
local_centroid-0	0.177363	0.847281	0.902854	0.858779	0.862417	0.504879	0.412859	nan	-0.631158	1.000000	0.706437	-0.439244	0.937673	0.932889	0.623186	0.003490	0.560853
local_centroid-1	0.227746	0.935689	0.904551	0.947036	0.934090	0.603305	0.609264	nan	-0.375580	0.706437	1.000000	-0.290177	0.863585	0.840724	0.875044	0.271191	0.318154
solidity	0.090163	-0.243908	-0.416707	-0.266587	-0.305706	-0.253635	-0.077797	nan	0.853431	-0.439244	-0.290177	1.000000	-0.512903	-0.556555	0.049965	0.279509	-0.723572
feret_diameter_max	0.208067	0.930981	0.973189	0.931696	0.948048	0.536089	0.458515	nan	-0.631776	0.937673	0.863585	-0.512903	1.000000	0.996744	0.690639	0.077145	0.614849
major_axis_length	0.198908	0.911069	0.967337	0.904412	0.932682	0.502524	0.422638	nan	-0.664733	0.932889	0.840724	-0.556555	0.996744	1.000000	0.639308	0.076773	0.647021
minor_axis_length	0.237521	0.859240	0.752580	0.904698	0.832264	0.645600	0.707711	nan	-0.062873	0.623186	0.875044	0.049965	0.690639	0.639308	1.000000	0.278107	-0.012148
orientation	0.319053	0.280673	0.213080	0.197456	0.263176	-0.139025	0.132754	nan	0.252915	0.003490	0.271191	0.279509	0.077145	0.076773	0.278107	1.000000	-0.305652
eccentricity	0.059804	0.348585	0.479196	0.363799	0.389269	0.246172	0.017030	nan	-0.756019	0.560853	0.318154	-0.723572	0.614849	0.647021	-0.012148	-0.305652	1.000000
standard_deviation_intensity	0.189165	0.288670	0.267528	0.402328	0.285105	0.867057	0.902001	nan	-0.216260	0.284331	0.379400	-0.169801	0.306228	0.280378	0.455324	-0.089349	0.107307
aspect_ratio	0.036433	0.411794	0.581132	0.386884	0.462720	0.121313	-0.044872	nan	-0.848271	0.678234	0.321805	-0.787587	0.690082	0.736200	-0.030443	-0.181927	0.853302
roundness	-0.055815	-0.415592	-0.569335	-0.406856	-0.464090	-0.191680	0.009002	nan	0.834550	-0.638667	-0.359961	0.801971	-0.690444	-0.732103	0.003699	0.224205	-0.955978
circularity	-0.054152	-0.626241	-0.718764	-0.701230	-0.659125	-0.636372	-0.411166	nan	0.808533	-0.785693	-0.644979	0.773934	-0.832660	-0.839196	-0.435236	0.242901	-0.779895
UMAP0	-0.065835	-0.442711	-0.413779	-0.509190	-0.435101	-0.324496	-0.387465	nan	0.168523	-0.391875	-0.488311	0.068021	-0.457079	-0.437340	-0.479807	0.025473	-0.204662
UMAP1	0.139702	0.819263	0.813951	0.793707	0.821940	0.391350	0.365621	nan	-0.375632	0.720004	0.753502	-0.260000	0.753713	0.736954	0.702828	0.277117	0.251959
MANUAL_CLUSTER_ID	0.080739	0.677335	0.719434	0.590973	0.700457	0.156570	0.074372	nan	-0.371454	0.582543	0.616873	-0.418390	0.671673	0.686248	0.387847	0.163152	0.424045

My annotation seems related to area

My annotation seems not related to intensity

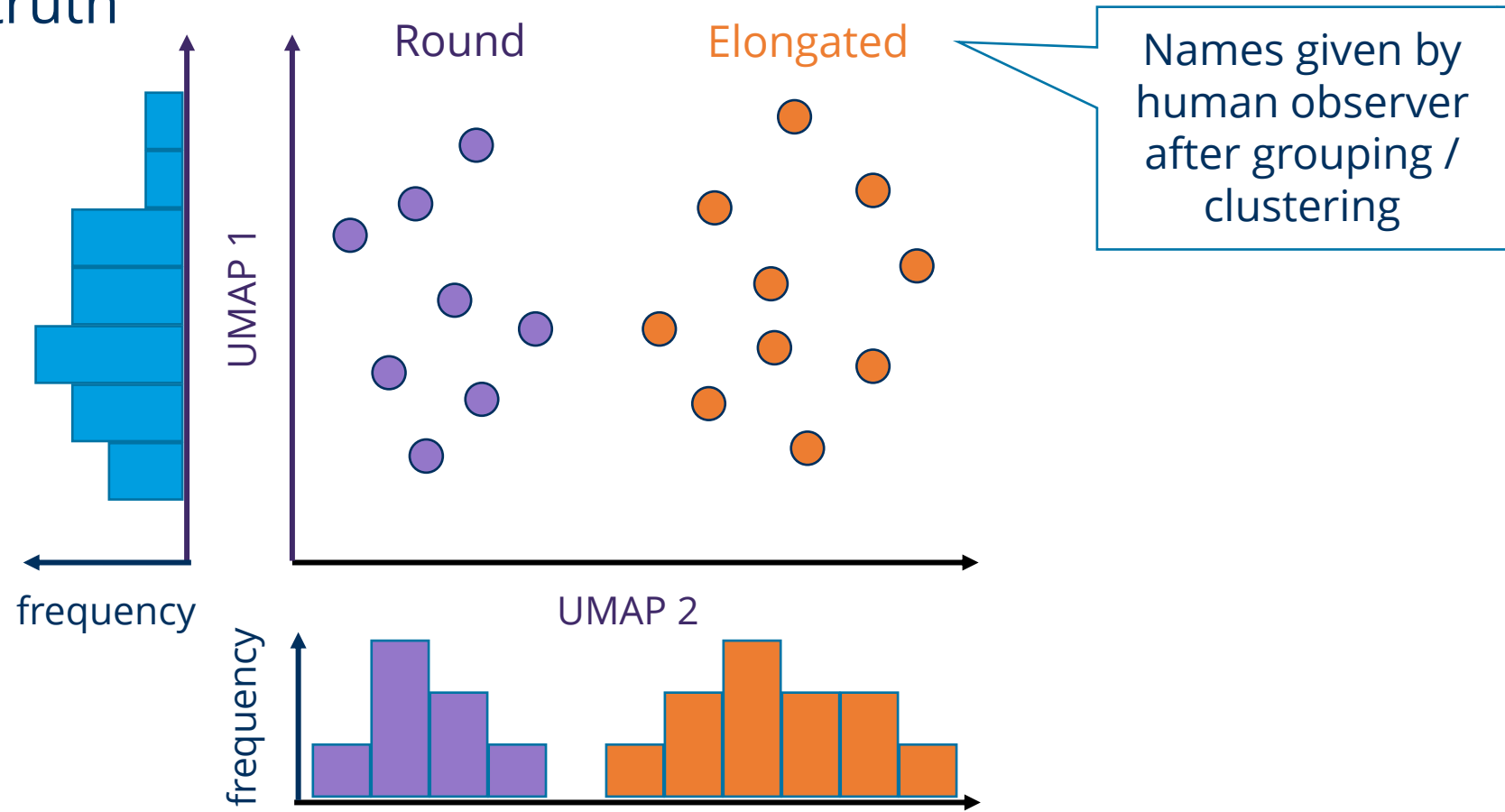
Clustering

Unsupervised machine learning may include grouping objects without given ground truth



Clustering

Unsupervised machine learning may include grouping objects without given ground truth



K-Means Clustering

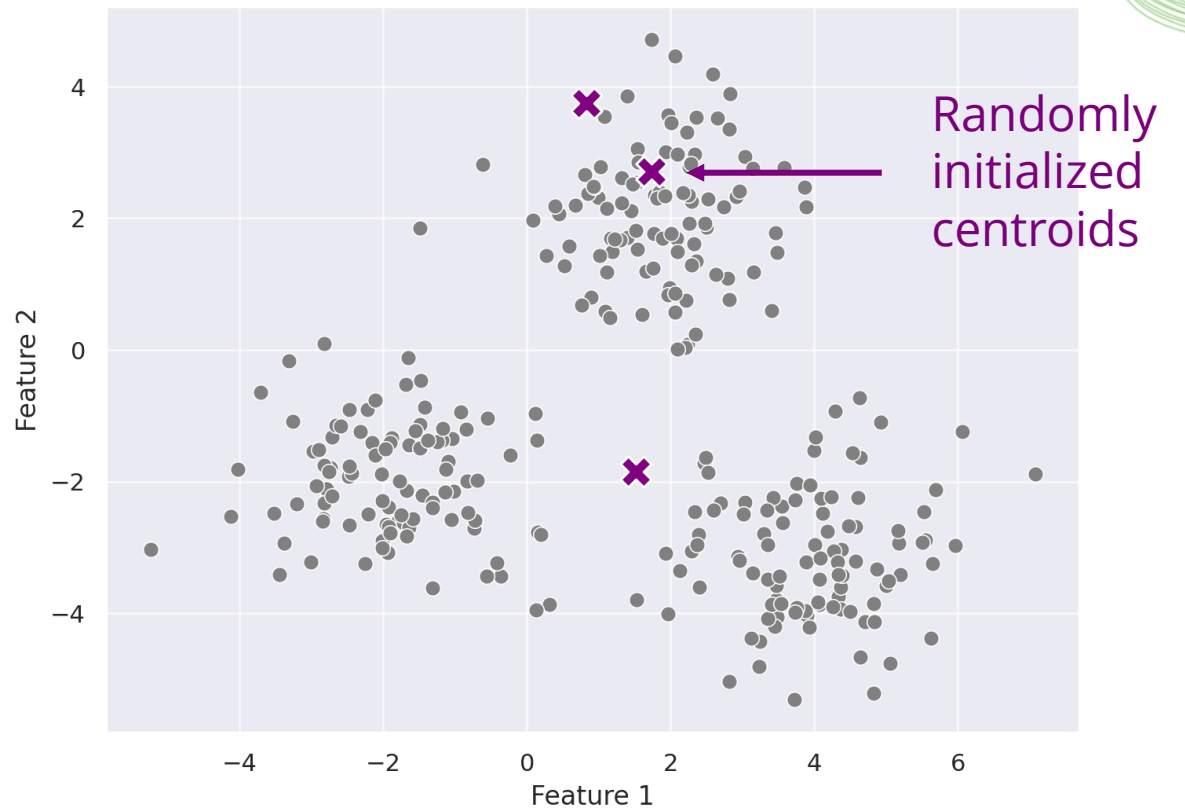
Goal: group data points into k groups so that variance within group is minimal.

STEP 1: Seed k initial cluster centroids randomly

STEP 2: Assign all points to nearest centroid

$$d(p, q) = \sqrt{\sum_{i=1}^n (p_i - q_i)^2} = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}$$

n - dimensionality, in this example = 2



K-Means Clustering

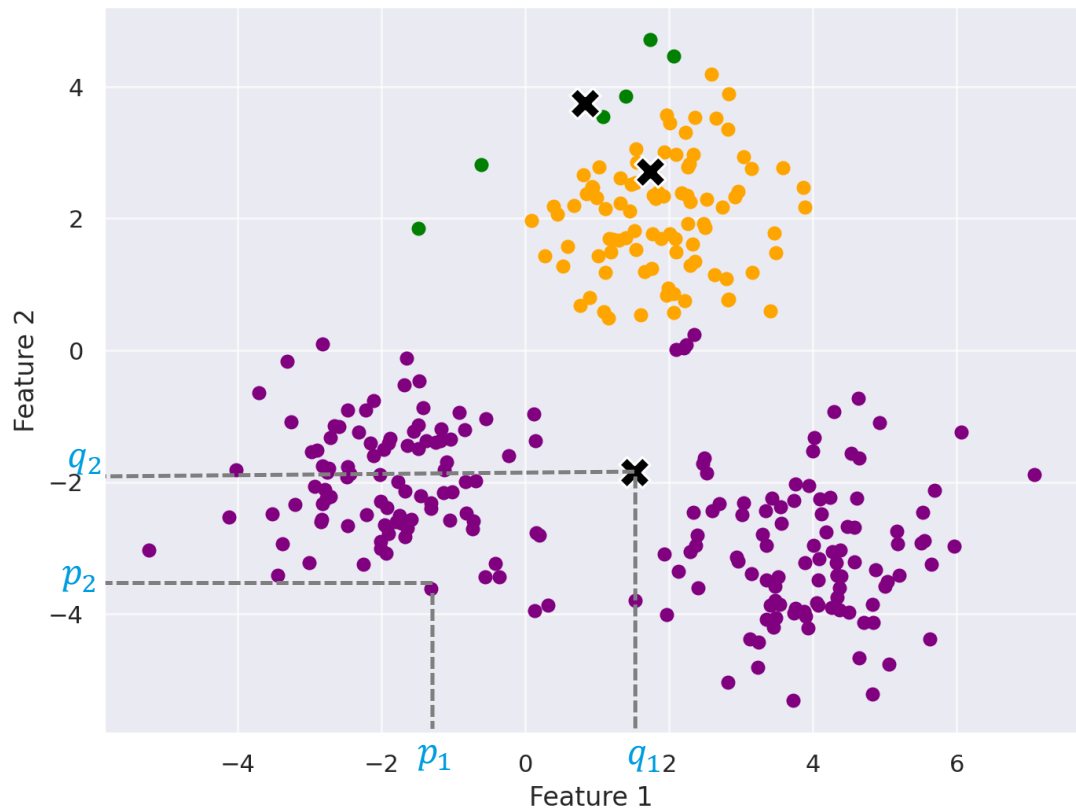
Goal: group data points into k groups so that variance within group is minimal.

STEP 1: Seed k initial cluster centroids randomly

STEP 2: Assign all points to nearest centroid

$$d(p, q) = \sqrt{\sum_{i=1}^n (p_i - q_i)^2} = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}$$

n - dimensionality, in this example = 2



K-Means Clustering

Goal: group data points into k groups so that variance within group is minimal.

STEP 3: Determine new centroid positions as mean position of all assigned points.

$$\text{New centroid}_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

C_i - the number of data points in cluster i

Repeat steps 2-3: the assignment and update steps are repeated iteratively until:

- Centroids not changing anymore,
- Point assignments not changing anymore or
- Maximum number of iterations reached



K-Means Clustering

Goal: group data points into k groups so that variance within group is minimal.

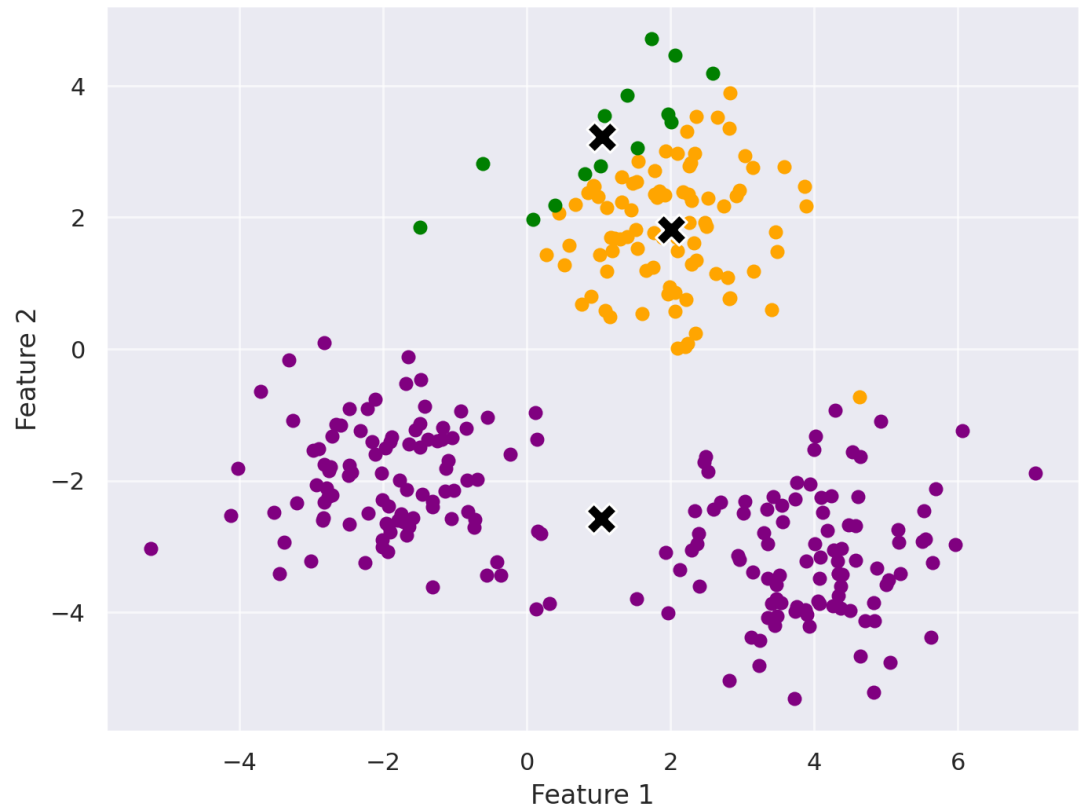
STEP 3: Determine new centroid positions as mean position of all assigned points.

$$\text{New centroid}_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

C_i - the number of data points in cluster i

Repeat steps 2-3: the assignment and update steps are repeated iteratively until:

- Centroids not changing anymore,
- Point assignments not changing anymore or
- Maximum number of iterations reached



K-Means Clustering

Goal: group data points into k groups so that variance within group is minimal.

In Python:

```
from sklearn import cluster
```

Create

```
clusterer = cluster.KMeans(n_clusters=3)  
clusterer.fit(X)
```

Predict

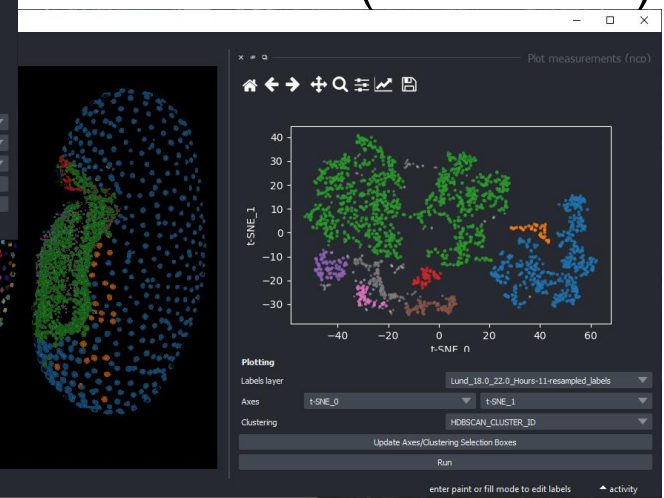
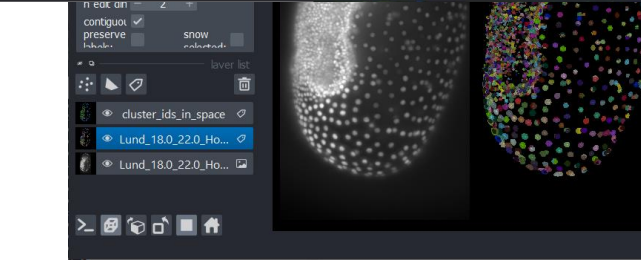
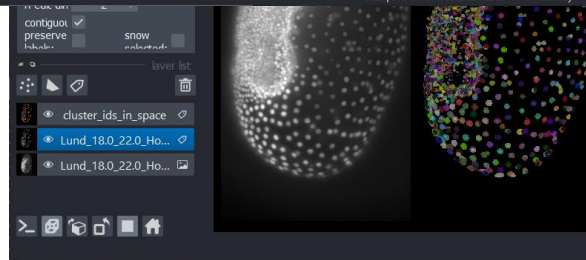
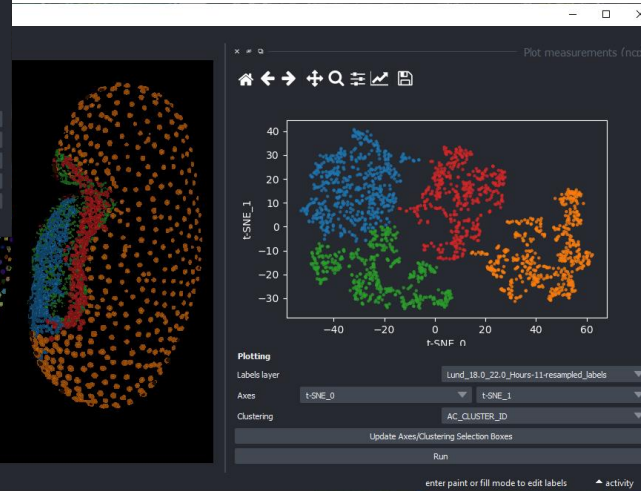
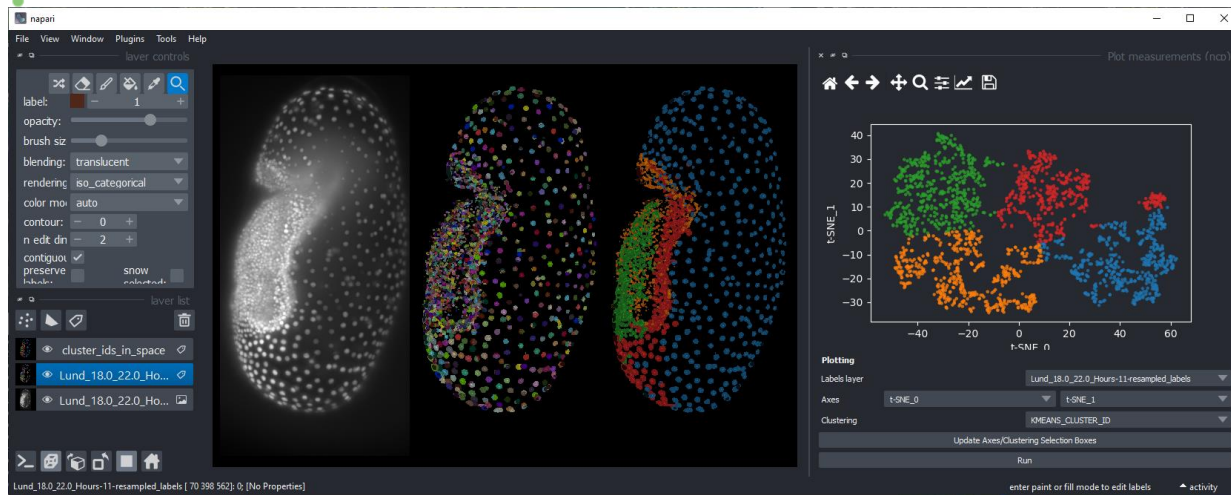
```
predicted_class = clusterer.predict(X)
```

Clustering

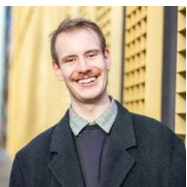
K-means clustering

Agglomerative clustering

Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN)



Laura Žigutytė
@zigutyte



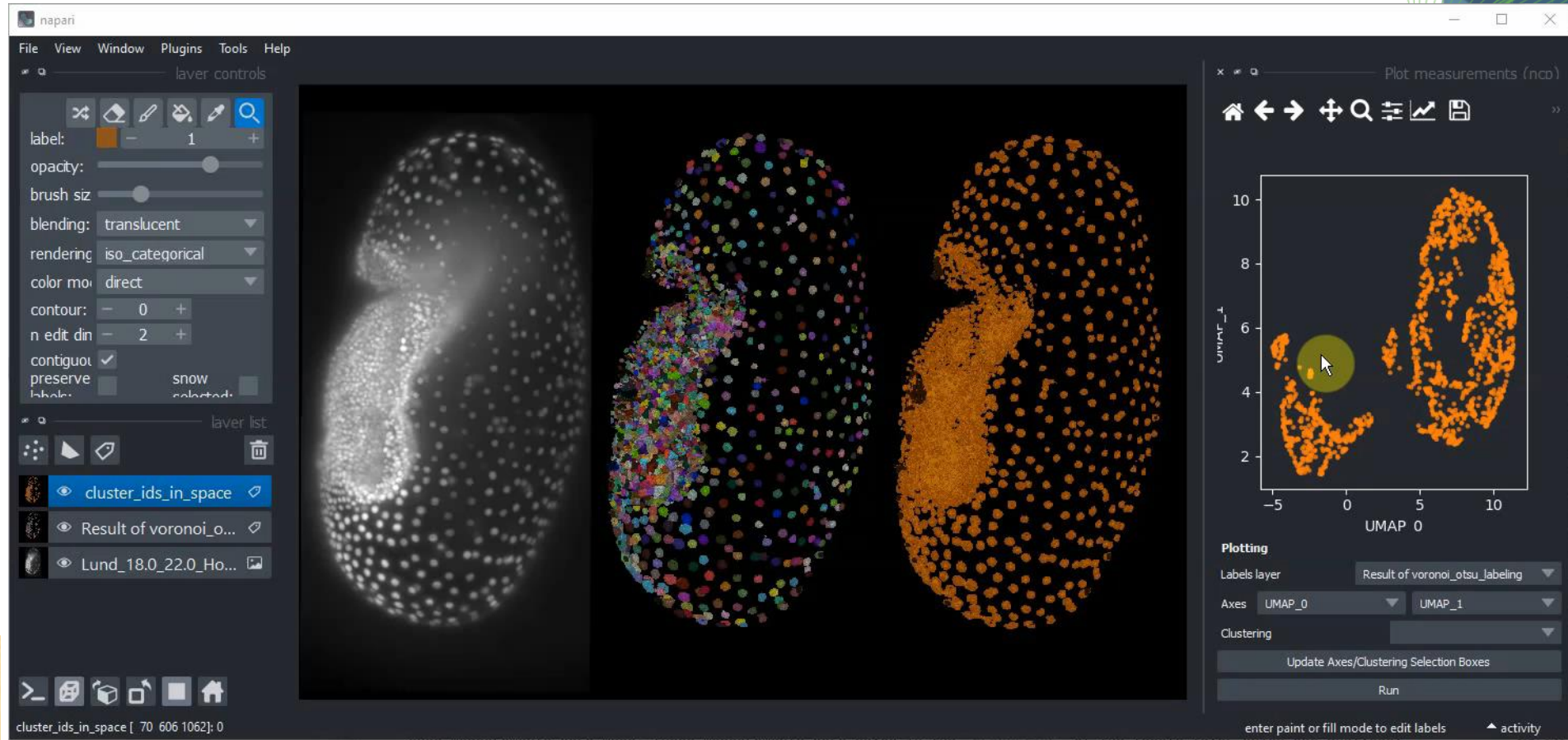
Ryan Savill
@RyanSavill4



Marcelo Zoccoler
@zoccoler-marcelo

Manual clustering

To better understand relationships between data



Laura Žigutyte
@zigutyte

Ryan Savill
@RyanSavill4

Marcelo Zoccoler
@zoccolermarcelo

Exercises

Robert Haase

Funded by



Bundesministerium
für Bildung
und Forschung

SACHSEN



Diese Maßnahme wird gefördert durch die Bundesregierung
aufgrund eines Beschlusses des Deutschen Bundestages.
Diese Maßnahme wird mitfinanziert durch Steuermittel auf
der Grundlage des von den Abgeordneten des Sächsischen
Landtags beschlossenen Haushaltes.

Exercise: Feature exploration

Use dimensionality reduction to elaborate features that might allow round and elongated objects

The image displays four overlapping JupyterLab notebook windows. The top-left window shows code for loading an image and segmenting it into objects. The top-right window shows a notebook with a 'TERACTION' section. The bottom-left window shows a 'Correlation statistics' section with a table of feature correlations. The bottom-right window shows a notebook with an 'Exercise' section.

```
[2]: image = imread("data/blob.tif")
labels = label(image > 120)
labels = cle.exclude_small_labels(labels, max_size=10)
labels = cle.exclude_labels_on_edges(labels)
stackview.insight(labels)
```

```
[16]: def colorize(styler):
styler.background_gradient(axis=None, cmap="PiYG")
return styler

df = measurements.corr().T
df.style.pipe(colorize)
```

	label	area	bbox_area	equivalent_diameter	convex_area	max_intensity	mean_intensity
label	1.000000	0.261682	0.223070	0.249249	0.250594	0.110791	
area	0.261682	1.000000	0.973718	0.978723	0.997560	0.511730	
bbox_area	0.223070	0.973718	1.000000	0.948328	0.985584	0.481524	
equivalent_diameter	0.249249	0.978723	0.948328	1.000000	0.974614	0.633984	
convex_area	0.250594	0.997560	0.985584	0.974614	1.000000	0.506730	
max_intensity	0.110791	0.511730	0.481524	0.633984	0.506730	1.000000	
mean_intensity	0.235692	0.530250	0.476951	0.618553	0.517356	0.825115	
min_intensity	nan	nan	nan	nan	nan	nan	

Exercise

The UMAP-generation above is done without parameters such as centroid and orientation. Why?

Exercise

Repeat the procedure above with the dataset human_mitosis. Identify parameters for differentiating the small bright cells from the others. (hint)

```
[ ]: image = human_mitosis()
stackview.insight(image)
```

Pixel classification / object segmentation

Use Napari to segment objects

Interactive pixel classification and object segmentation in Napari

In this exercise we will train a [Random Forest Classifier](#) for pixel classification and convert the result in an instance segmentation. We will use the napari plugin [napari-accelerated-pixel-and-object-classification](#).

Getting started

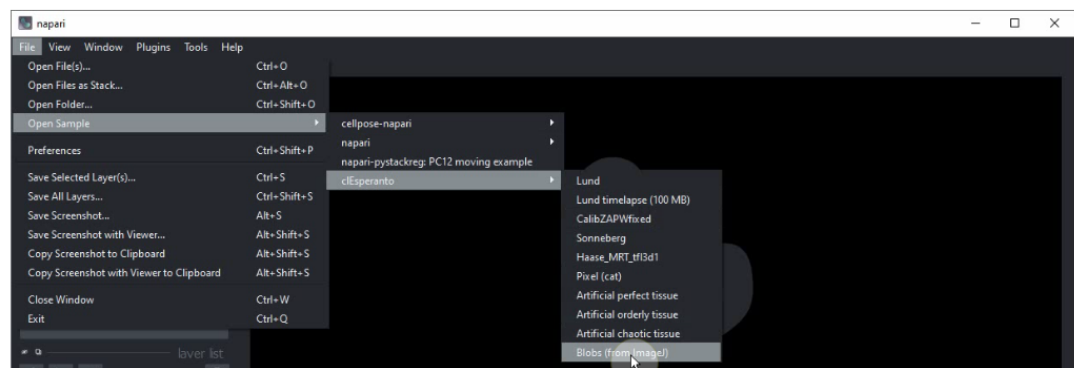
Open a terminal window and activate your conda environment:

```
conda activate devbio-napari-env
```

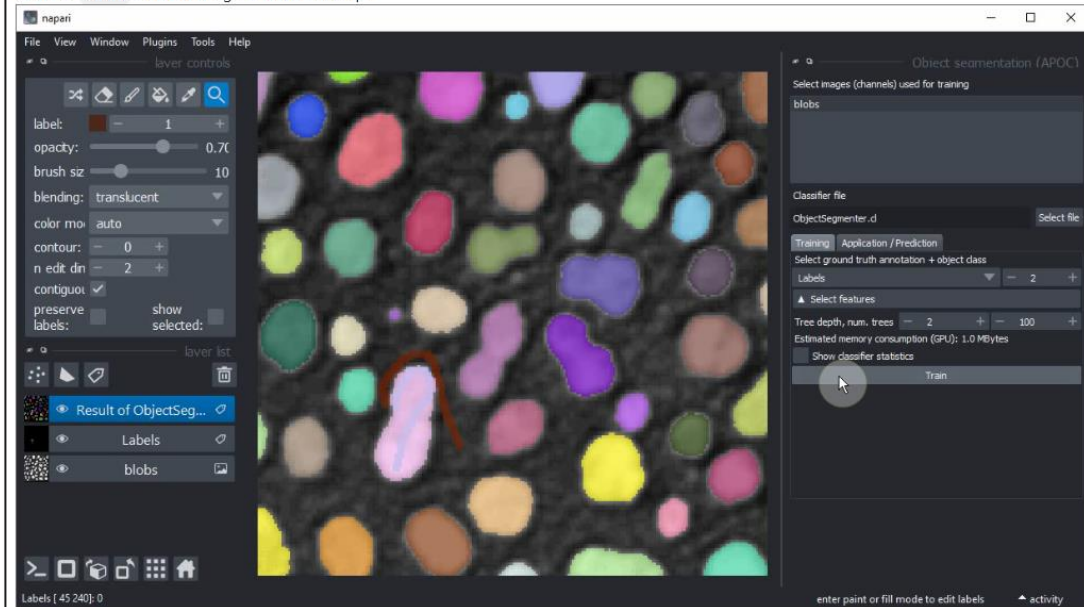
Afterwards, start up Napari:

```
napari
```

Load the "Blobs" example dataset from the menu `File > Open Sample > c1Esperanto > Blobs (from ImageJ)`



Click on `Train`. A label image should show up.



If the segmentation works well, consider backing up the `objectSegmenter.c1` file that has been saved. If you didn't change the file location before training, it will be located in the folder from where you started napari on the command line.

Object classification

Use Napari to classify round and elongated objects

Interactive object classification in Napari

In this exercise we will train a [Random Forest Classifier](#) for classifying segmented objects. We will use the napari plugin [napari-accelerated-pixel-and-object-classification](#).

Getting started

Open a terminal window and activate your conda environment:

```
conda activate devbio-napari-env
```

Afterwards, start up Napari:

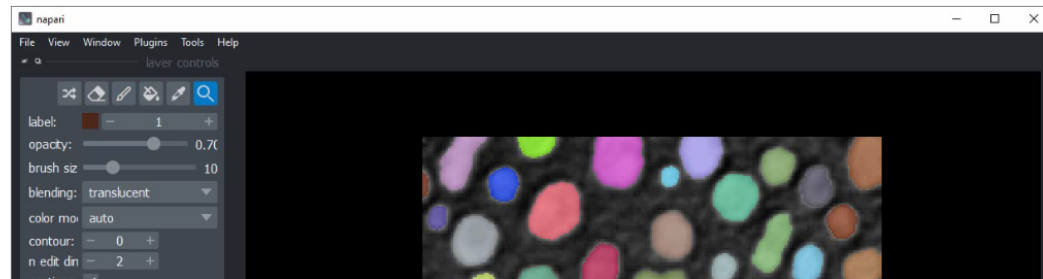
```
napari
```

Load the "Blobs" example dataset from the menu `File > Open Sample > cIesperanto > Blobs (from ImageJ)`

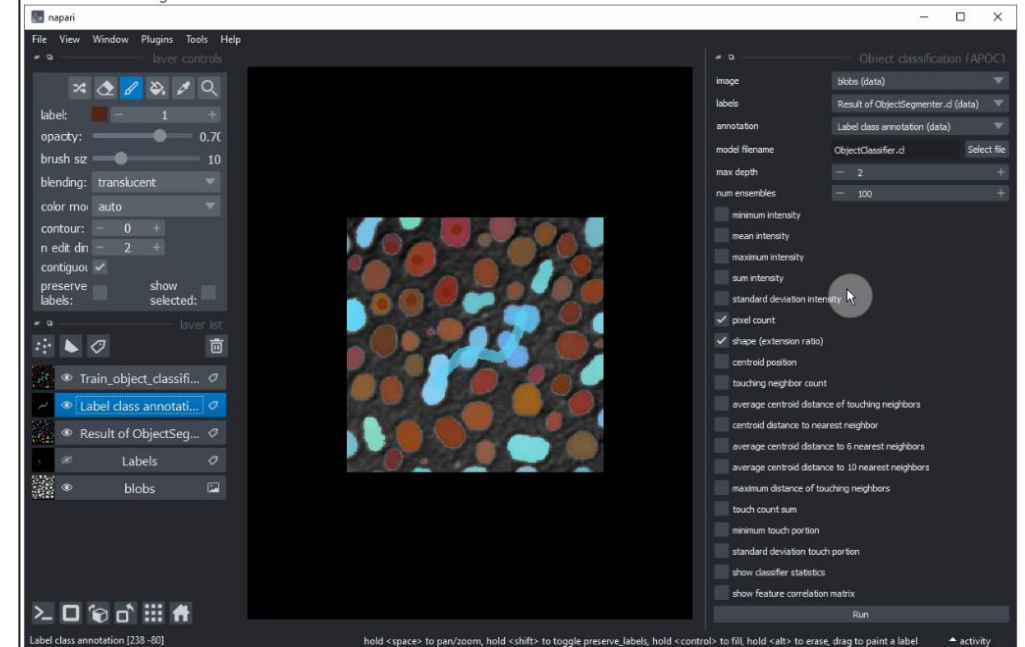
We furthermore need a label image. You can create it using the pixel classifier trained earlier or using the menu `Tools > Segmentation / Labeling > Gauss-Otsu Labeling (clesperanto)`.

Object classification

Our starting point is a loaded image and a label image with segmented objects. The following procedure is also shown in [this video](#).



Train the classifier again.



If you are happy with the trained classifier, copy the file to a safe place. When training the next classifier this one might be overwritten.

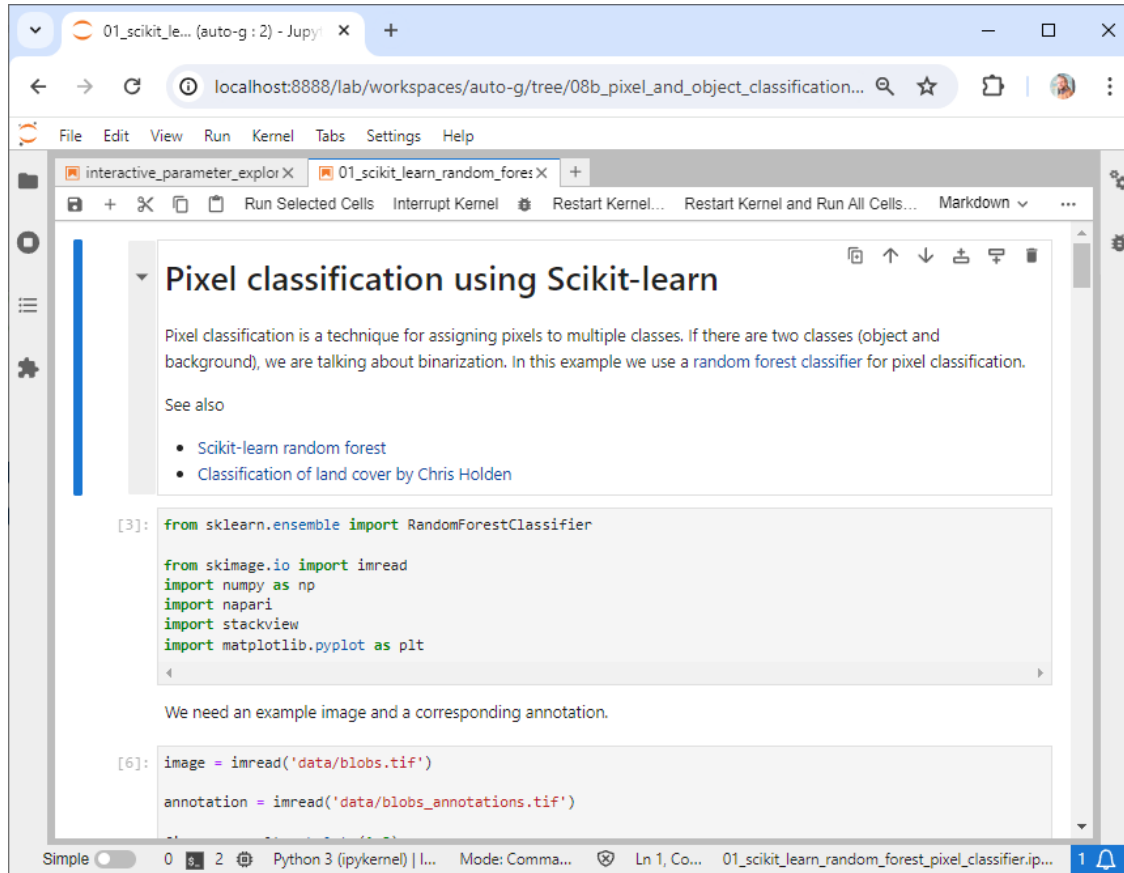
Extra exercise

Retrain the classifier so that it can differentiate three different classes:

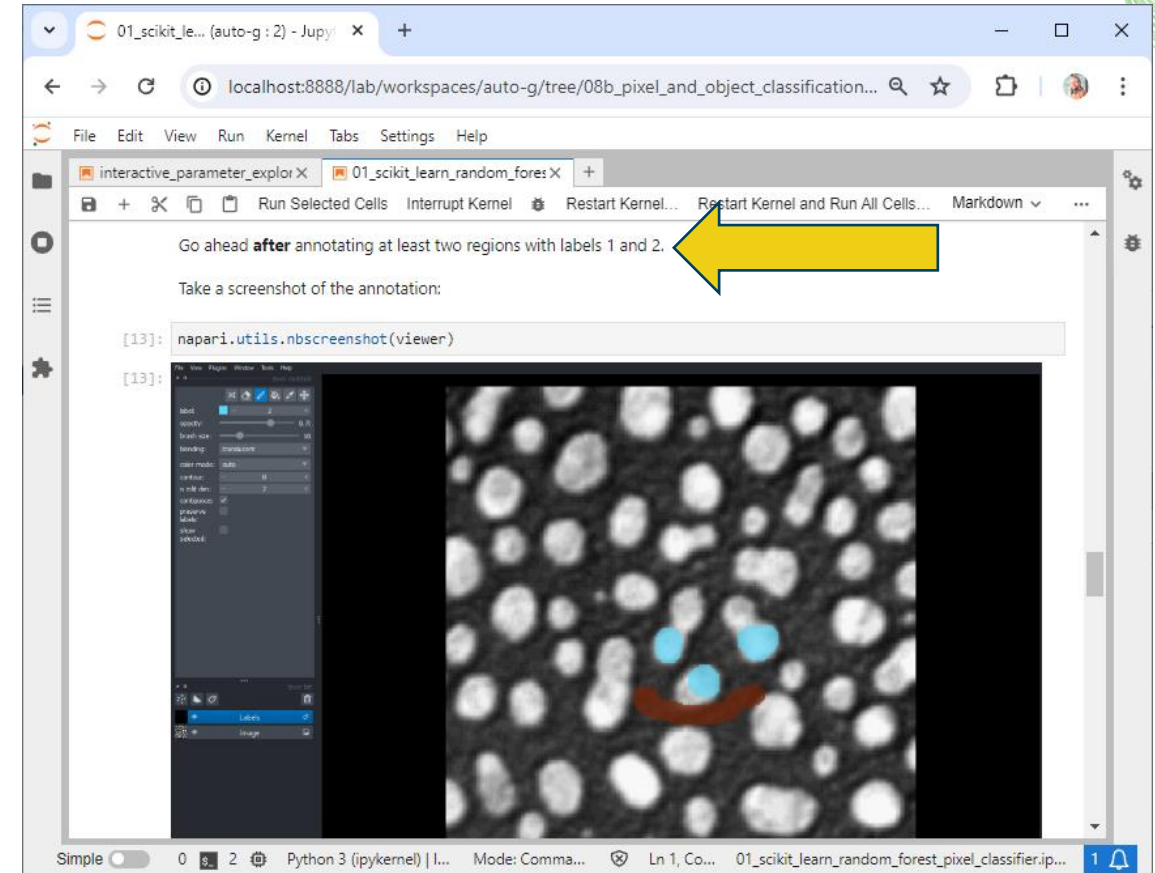
- Small round objects
- Large round objects
- Large elongated objects

Supervised machine learning using Python

Use scikit-learn and apoc in Jupyter Notebooks to train and apply Random Forest Classifiers



The screenshot shows a Jupyter Notebook titled "Pixel classification using Scikit-learn". The text explains that pixel classification is a technique for assigning pixels to multiple classes, specifically binarization for two classes (object and background). It mentions using a random forest classifier. Below the text, there are two code cells. The first cell imports the necessary libraries: `from sklearn.ensemble import RandomForestClassifier`, `from skimage.io import imread`, `import numpy as np`, `import napari`, `import stackview`, and `import matplotlib.pyplot as plt`. The second cell loads an example image and its corresponding annotation: `image = imread('data/blobs.tif')` and `annotation = imread('data/blobs_annotations.tif')`.



The screenshot shows the same Jupyter Notebook with the execution of a screenshot function. The text says: "Go ahead **after** annotating at least two regions with labels 1 and 2." A yellow arrow points to the "Restart Kernel and Run All Cells..." button. Below this, the text says: "Take a screenshot of the annotation:". The code cell `[13]: napari.utils.nbscreenshot(viewer)` is executed. The output shows a screenshot of the napari viewer displaying the image with two regions annotated in blue and red. The napari viewer interface is visible in the background of the screenshot.

Configuring Random Forest Classifiers

Classifier statistics

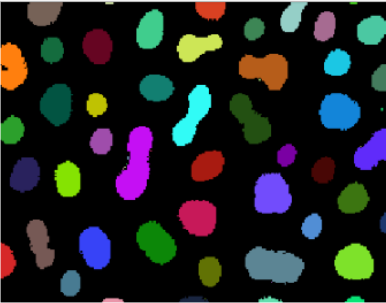
After training, we can print out some statistics from the classifier. It gives us a table of used features and how important the features were for making the pixel classification decision.

```
[6]: shares, counts = classifier.statistics()

def colorize(styler):
    styler.background_gradient(axis=None, cmap="PiYG")
    return styler

df = pd.DataFrame(shares).T
df.style.pipe(colorize)
```

	0	1	2	3	4
original	0.138000	0.046423	0.042312	0.037281	0.062112
gaussian_blur=1	0.228000	0.092846	0.074303	0.105263	0.055901
difference_of_gaussian=1	0.000000	0.108828	0.095975	0.074561	0.086957
laplace_box_of_gaussian_blur=1	0.000000	0.105784	0.089783	0.081140	0.099379
gaussian_blur=5	0.096000	0.064688	0.118679	0.096491	0.130435
difference_of_gaussian=5	0.254000	0.182648	0.112487	0.120614	0.118012
laplace_box_of_gaussian_blur=5	0.209000	0.194064	0.121775	0.118421	0.124224
gaussian_blur=25	0.004000	0.061644	0.113519	0.127193	0.080745
difference_of_gaussian=25	0.031000	0.072298	0.122807	0.127193	0.130435
laplace_box_of_gaussian_blur=25	0.040000	0.070776	0.108359	0.111842	0.111801



The new classifier still produces a very similar result. It takes less features into account, which makes it faster, but potentially also less robust again differences between images and imaging conditions. We just take another look at the classifier statistics:

```
[8]: shares, counts = classifier.statistics()
df = pd.DataFrame(shares).T
df.style.pipe(colorize)
```

	0	1	2
gaussian_blur=1	0.331000	0.349194	0.344620
difference_of_gaussian=5	0.356000	0.329839	0.337096
laplace_box_of_gaussian_blur=5	0.313000	0.320968	0.318284

```
c1_filename = 'data/blobs_object_segmenter_3.c1'

apoc.erase_classifier(c1_filename)
classifier = apoc.ObjectSegmenter(openc1_filename=c1_filename,
    positive_class_identifier=2,
    max_depth=3,
    num_ensembles=1000)

classifier.train(features=features, manual_annotation, image)

segmentation_result = classifier.predict(features=features, image=image)
stackview.imshow(segmentation_result, labels=True)
```

