INFOs für DENNIS😊

values the user should be able to change in chronological order they appear in the code:

**blue= which function**

**orange= user input, depicted is default**

TSNE:

tsne = TSNE(n\_components=2, perplexity=20, n\_iter=500 )

KNN:

Here, as discussed before, the python code so far takes the given input of n\_neighbors and calculates in a second step the optimal number of neigbours. So it would be nice, if we could show the user the output of the code eg. here:

The optimal number of neighbors is 1

And then ask if he wants to accept the optimal number or put in another value, before we actually show him the knn plots.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(arry\_norm, condition2, test\_size=0.2, random\_state=12345)

model = KNeighborsClassifier(n\_neighbors=3)

Random forest classifier:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(arry\_all, condition2, test\_size=0.2, random\_state=12345, stratify=condition2)

clf=RandomForestClassifier(n\_estimators=1000, random\_state=12345, min\_samples\_split= 2, min\_samples\_leaf= 2, max\_features="sqrt", max\_depth=110, bootstrap=True)

plots\_all(df\_all, order) # plots all the boxplots

here user should be able to check the features he wants displayed as boxplots, and it would be very nice if we could add the colour ticker.

sns.boxplot(x="condition",y="density", data=df\_unique,order=order, palette="Paired")

sns.swarmplot(x="condition", y="density", data=df\_unique, order=order, color="0.2", alpha=0.7, size=4)

plt.title("ND density")

plt.xlabel('condition')

plt.ylabel('density')

INFOS für Shelli/text GUI:

Segmentation and feature extraction:

This part of the NANONET expects one folder with TIFF images, displaying your regions of interest (ROIs). For certain features (such as Nanodomain density) it is important that your ROIs all have the same size. The app will prompt you to enter the size in pixels. This can be checked in common image analysis tool such as Fiji. As output you will get a results excel sheet with all the features extracted. For feature annotation see….

Comparison and Classification between conditions

This part of the NANONET expects at minimum 2 input folders with TIFF images, displaying your regions of interest (ROIs) you want to compare. For certain features (such as Nanodomain density) it is important that your ROIs all have the same size. The app will prompt you to enter the size in pixels. This can be checked in common image analysis tool such as Fiji. The extracted features of each condition (image folder) will be passed on to train a Random Forest and a k-NN classifier. Additionally you will get Boxplots comparing chosen features between your conditions, a TSNE map and an inter-feature correlation map.

Feature annotation:

"name": name of the image

"area": area in microns per segmented nanodomain

"mean\_area": mean area in microns of segmented nanodomains per image

"var\_area": variance of nanodomain area within one picture

"density": density of nanodomains within one picture

"intensity": mean intensity per segmented nanodomain

"relative\_intensity": mean intensity per segmented nanodomain/ mean intensity of the image

"mean\_intensity": mean intensity of segmented nanodomains per image

"var\_intensity": variance of nanodomain intensity per image

"max\_intensity": maximum intensity per nanodomain

"mean\_max\_intensity”: mean maximum nanodomain intensity per image

"min\_intensity": minimum intensity per nanodomain

"mean\_min\_intensity": mean minimum intensity of nanodomains per image

"area\_filled": area in microns of the nanodomain with all the holes filled in

"mean\_area\_filled": mean area in microns of the nanodomain with all the holes filled in per image

"major\_axis\_length": length in microns of the major axis of the ellipse that has the same normalized second central moments as the nanodomain

"mean\_major\_axis\_length": mean length in microns of the major axis of the ellipse that has the same normalized second central moments as the nanodomains per image

"minor\_axis\_length": length in microns of the minor axis of the ellipse that has the same normalized second central moments as the nanodomain

"mean\_minor\_axis\_length": mean length in microns of the minor axis of the ellipse that has the same normalized second central moments as the nanodomains per image

"eccentricity": Eccentricity of the ellipse that has the same second-moments as the nanodoamin. The eccentricity is the ratio of the focal distance (distance between focal points) over the major axis length. The value is in the interval [0, 1). When it is 0, the ellipse becomes a circle

"mean\_eccentricity": mean of the nanodomain eccentricity per image

"equivalent\_diameter\_area": The diameter in microns of a circle with the same area as the nanodomain

"mean\_equivalent\_diameter\_area": mean of the equivalent\_diameter\_area per image

"perimeter": Perimeter in microns of object which approximates the contour as a line through the centers of border pixels using a 4-connectivity

"mean\_perimeter": mean perimeter per image

"label": label number of segmented nanodomain

"sum\_label": sum of all labels (number of segmented nanodomain) per image

“var\_intensity\_image”: intensity variance per image

“mean\_intensity\_image”: mean intensity per image

“mean\_intensity\_blurred”: mean intensity of gaussian blurred image

“var\_intensity\_blurred”: variance intensity of gaussian blurred image

“SCI”: spatial clustering index

Train test split: <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html>

sets how much of your data should be used to train the classifier v.s test the model.

**Stratify: default=condition** If not None, data is split in a stratified fashion, assuring equal representation of conditions classes in the split dataset.

**test\_size: *float, default=0.25*** The float should be between 0.0 and 1.0 and represents the proportion of the dataset to include in the test split. For example the default 0.25 uses 75% of the data to train the model and 25% to test it.

RF: <https://scikit-learn.org/stable/modules/ensemble.html#forest>

In random forests (see [**RandomForestClassifier**](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier) and [**RandomForestRegressor**](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html#sklearn.ensemble.RandomForestRegressor) classes), each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set.

Furthermore, when splitting each node during the construction of a tree, the best split is found either from all input features or a random subset of size max\_features. (See the [parameter tuning guidelines](https://scikit-learn.org/stable/modules/ensemble.html#random-forest-parameters) for more details).

The purpose of these two sources of randomness is to decrease the variance of the forest estimator. Indeed, individual decision trees typically exhibit high variance and tend to overfit. The injected randomness in forests yield decision trees with somewhat decoupled prediction errors. By taking an average of those predictions, some errors can cancel out. Random forests achieve a reduced variance by combining diverse trees, sometimes at the cost of a slight increase in bias. In practice the variance reduction is often significant hence yielding an overall better model.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

**n\_estimators: int, default=100** The number of trees in the forest.

**max\_depth: *int, default=None*** The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.

* **min\_samples\_split *int or float, default=2*** The minimum number of samples required to split an internal node: If int, then consider min\_samples\_split as the minimum number.
* If float, then min\_samples\_split is a fraction and ceil(min\_samples\_split \* n\_samples) are the minimum number of samples for each split.

**min\_samples\_leaf *int or float, default=1***

The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min\_samples\_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

* If int, then consider min\_samples\_leaf as the minimum number.
* If float, then min\_samples\_leaf is a fraction and ceil(min\_samples\_leaf \* n\_samples) are the minimum number of samples for each node.

Knn classifier:

Neighbors-based classification is a type of instance-based learning or non-generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point.

<https://scikit-learn.org/stable/modules/neighbors.html#classification>

**n\_neighbors *int, default=5:*** Number of neighbors to use by default for [**kneighbors**](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier.kneighbors) queries. Find the K-neighbors of a point. Returns indices of and distances to the neighbors of each point.

TSE: <https://scikit-learn.org/stable/modules/manifold.html#t-sne>

t-SNE [1] is a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler (KL) divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. t-SNE has a cost function that is not convex, i.e. with different initializations we can get different results. <https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html>

The goal is to take a set of points in a high-dimensional space and find a faithful representation of those points in a lower-dimensional space, typically the 2D plane. The algorithm is non-linear and adapts to the underlying data, performing different transformations on different regions. Those differences can be a major source of confusion.

Optimizing the KL divergence can be a little bit tricky sometimes. There are five parameters that control the optimization of t-SNE and therefore possibly the quality of the resulting embedding:

* perplexity
* early exaggeration factor
* learning rate
* maximum number of iterations
* angle (not used in the exact method)

**perplexity**: defines (loosely spoken) how to balance the focus on local and global features of your data. Larger perplexities lead to more nearest neighbors and less sensitive to small structure. Conversely a lower perplexity considers a smaller number of neighbors, and thus ignores more global information in favour of the local neighborhood. As dataset sizes get larger more points will be required to get a reasonable sample of the local neighborhood, and hence larger perplexities may be required. Similarly noisier datasets will require larger perplexity values to encompass enough local neighbors to see beyond the background noise.

the perplexity is defined as k=2(S) where S is the Shannon entropy of the conditional probability distribution. The perplexity of a k-sided die is k, so that k is effectively the number of nearest neighbors t-SNE considers when generating the conditional probabilities.. says (loosely) how to balance attention between local and global aspects of your data. The parameter is, in a sense, a guess about the number of close neighbors each point has. The perplexity value has a complex effect on the resulting pictures.

The maximum number of iterations is usually high enough and does not need any tuning. The optimization consists of two phases: the early exaggeration phase and the final optimization. During early exaggeration the joint probabilities in the original space will be artificially increased by multiplication with a given factor. Larger factors result in larger gaps between natural clusters in the data. If the factor is too high, the KL divergence could increase during this phase. Usually it does not have to be tuned.

**learning rate:** defines the learning rate of

If it is too low gradient descent will get stuck in a bad local minimum. If it is too high the KL divergence will increase during optimization. A heuristic suggested in Belkina et al. (2019) is to set the learning rate to the sample size divided by the early exaggeration factor. We implement this heuristic as learning\_rate='auto' argument.

**learning\_rate*float or ‘auto’, default=200.0***

The learning rate for t-SNE is usually in the range [10.0, 1000.0]. If the learning rate is too high, the data may look like a ‘ball’ with any point approximately equidistant from its nearest neighbours. If the learning rate is too low, most points may look compressed in a dense cloud with few outliers. If the cost function gets stuck in a bad local minimum increasing the learning rate may help. The ‘auto’ option sets the learning\_rate to max(N / early\_exaggeration / 4, 50) where N is the sample size, following [4] and [5]. This will become default in 1.2.

**Angle *float, default=0.5***

Only used if method=’barnes\_hut’ This is the trade-off between speed and accuracy for Barnes-Hut T-SNE. ‘angle’ is the angular size (referred to as theta in [3]) of a distant node as measured from a point. If this size is below ‘angle’ then it is used as a summary node of all points contained within it. This method is not very sensitive to changes in this parameter in the range of 0.2 - 0.8. Angle less than 0.2 has quickly increasing computation time and angle greater 0.8 has quickly increasing error.

The last parameter, angle, is a tradeoff between performance and accuracy. Larger angles imply that we can approximate larger regions by a single point, leading to better speed but less accurate results.

Really good resource: <https://distill.pub/2016/misread-tsne/>

Segmentation pipeline:

Images are gaussian blurred,