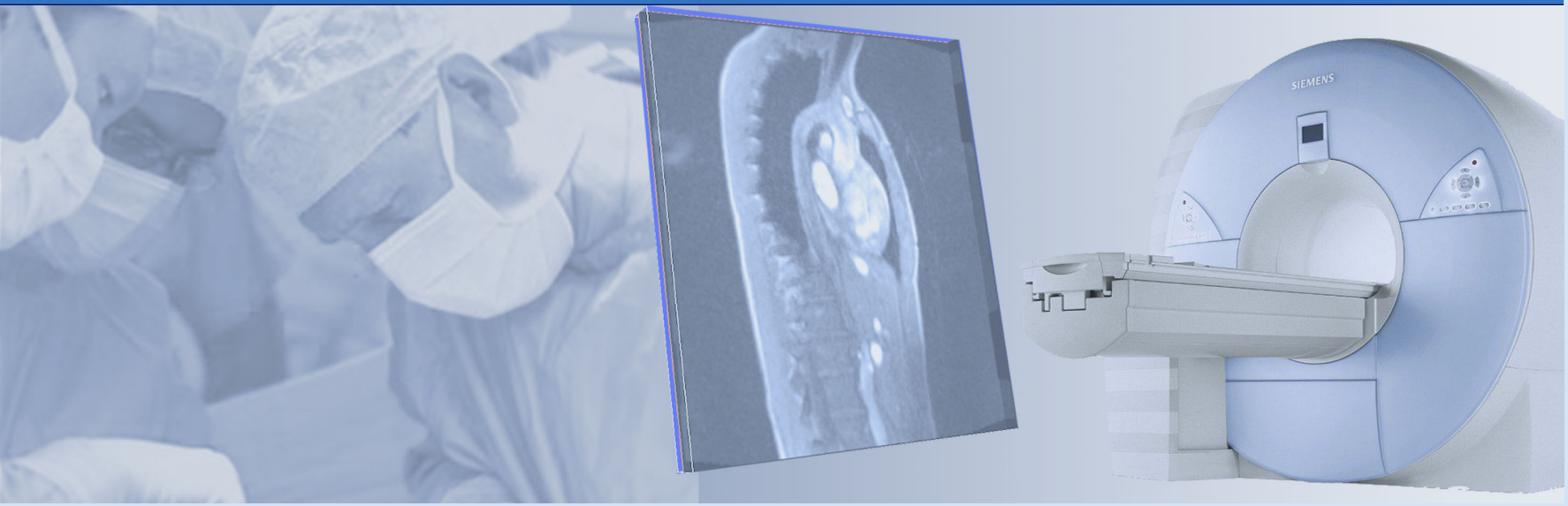
Tutorial Computer- and robot-assisted Surgery



NCT

NATIONALES CENTRUM
FÜR TUMORERKRANKUNGEN
PARTNERSTANDORT DRESDEN
UNIVERSITÄTS KREBSCENTRUM UCC

getragen von:
Deutsches Krebsforschungszentrum
Universitätsklinikum Carl Gustav Carus Dresden
Medizinische Fakultät Carl Gustav Carus, TU Dresden
Helmholtz-Zentrum Dresden-Rossendorf

Sebastian Bodenstedt
Translational Surgical Oncology

Questions lecture

Evaluation Metrics

Evaluation metrics

- Accuracy: how many samples were correctly identified?
  - #correctly identified samples/#samples
  - Works well for balanced data sets, i.e. all classes occurs similarly often
  - Problems with minor classes, e.g. diseases detection:
    if 99% are negative examples, predicting just the negative classes leads to a high accuracy, but misses critical cases
- Confusion matrix

n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

Evaluation metrics (binary problems)

- Terms
  - True positives (TP): number of correctly identified positive examples
  - False positives (FP): number of falsely identified positive examples
  - True negatives (TN): number of correctly identified negative examples
  - False negatives (FN): number of falsely identified negative examples

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

Evaluation metrics (binary problems)

- Precision
  - What portion of the as positive identified samples was correct?

$$Precision = \frac{True\ positives}{True\ positives + False\ positives}$$

		Predicted: Negative	Predicted: Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

- Recall
  - What portion of the positive samples was found?

$$Recall = \frac{True\ positives}{True\ positives + False\ negatives}$$

		Predicted: Negative	Predicted: Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

NCT

Evaluation metrics (binary problems)

- F1-score
  - Combination of precision and recall
    - How balanced are the two for your classifier?

$$F1-score = \frac{2 \times Precision * Recall}{Precision + Recall}$$

  - Why use F1 instead of accurcacy?
    - True negatives have no impact

- Binary metrics for multi-class problems
  - Generally binary metrics can be computed for each class
    - Positive samples vs negative samples for that class
  - The metrics for each class can then be aggregated, e.g. using mean or median

NCT

K-Nearest Neighbor

NCT

K-Nearest Neighbors

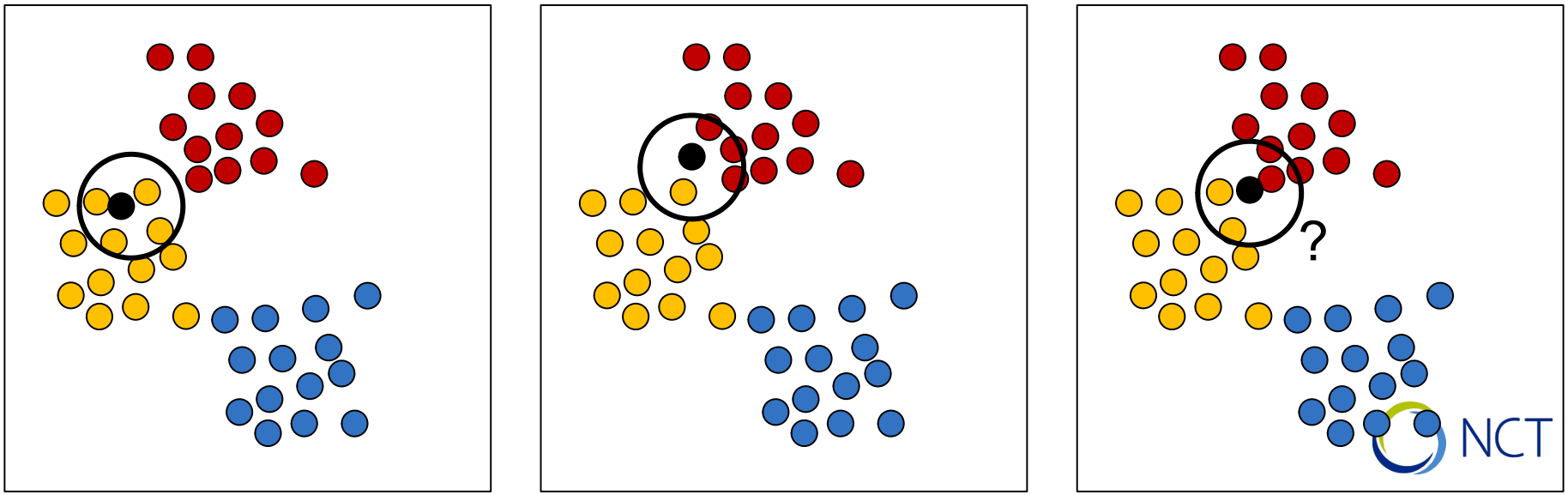
Idea

- Birds of a feather flock together

  => samples with similar values belong to the same label

- Instance-based learning, no generalization

K = 4



Principal Component Analysis (PCA)

Singular Value Decomposition (SVD)

- Factorization of a *m* x *n* matrix into $A = U \Sigma V^T$
  - *U*: *m* x *m* orthonormal matrix. Contains eigenvectors of AA^T
  - *V*: *n* x *n* orthonormal matrix. Contains eigenvectors of A^TA
  - *Σ*: *m* x *n* diagonal matrix. Contains eigenvalues of A^TA

$$\boldsymbol{A} = \begin{pmatrix} \vdots & \dots & \vdots \\ \boldsymbol{u}_1 & \dots & \boldsymbol{u}_n \\ \vdots & \dots & \vdots \end{pmatrix} \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{pmatrix} \begin{pmatrix} \dots & \mathbf{v}_1^T & \dots \\ \vdots & \vdots & \vdots \\ \dots & \mathbf{v}_n^T & \dots \end{pmatrix}$$

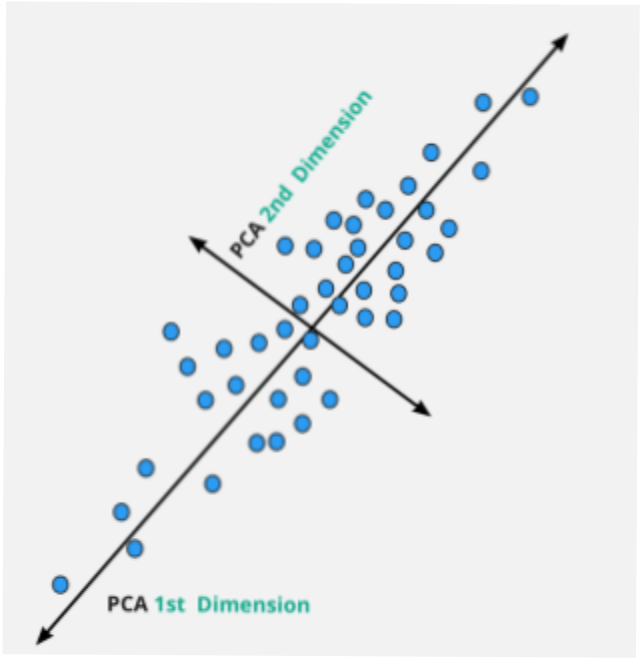
$$\boldsymbol{A} = \begin{pmatrix} \vdots & \dots & \vdots \\ \boldsymbol{u}_1 & \dots & \boldsymbol{u}_n \\ \vdots & \dots & \vdots \end{pmatrix} \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{pmatrix} \begin{pmatrix} \vdots & \dots & \vdots \\ \boldsymbol{v}_1 & \dots & \boldsymbol{v}_n \\ \vdots & \dots & \vdots \end{pmatrix}^T$$

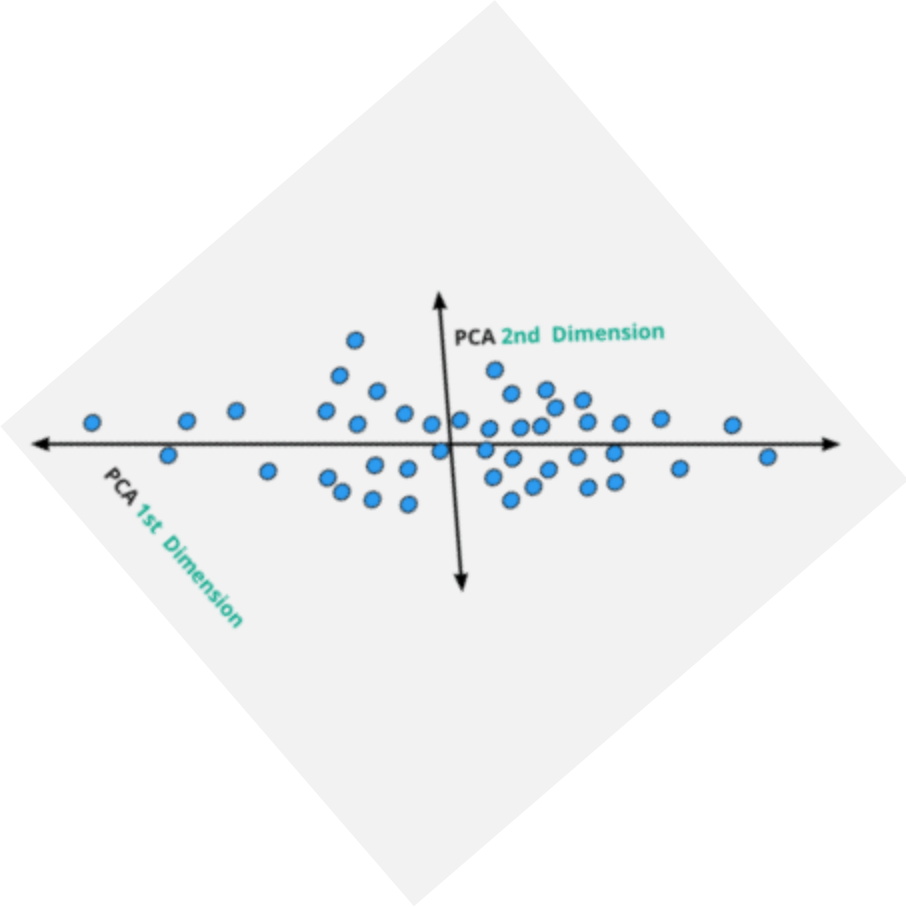
Reminder eigenvector and eigenvalue: $Av = \lambda v$

NCT

PCA

- Curse of dimensionality => number of required training samples increase (possibly exponentially) with the number of dimensions
- Idea: Re-orient data to maximize variance along axis and remove low-variance dimensions





NCT

PCA

- Steps

  1)Normalize data

$$z = \frac{value - mean}{standard\ deviation}$$

  2)Calculate covariance matrix (describes spread of the data)

$$\Sigma = \frac{1}{N} D \cdot D^T$$

  3)Compute eigenvectors and eigenvalues of covariance matrix (via SVD)
    - Eigenvector are principal axis
    - Eigenvalues indicate variance or “data spread” along axis

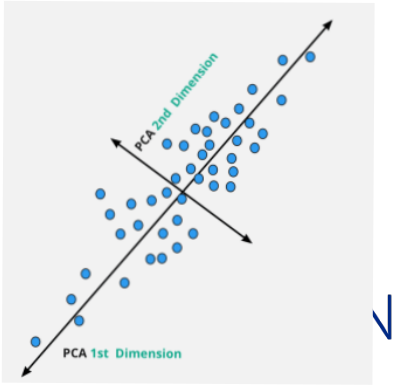
  4)Build rotation matrix from principal axis (basis change)

  5)Rotate data and remove dimensions with low variance

  Retain the first t components that describe e.g. 98%of variation

$$V_T = \sum \lambda_i$$

$$\sum_{i=1}^{t} \lambda_i \geq f_v V_T$$



Introduction: Machine Learning with Scikit-learn

NCT

Scikit-learn

- Machine-learning library for Python
  - Includes many machine learning methods
    - Clustering
    - Random Forest
    - Support Vector Machine
    - …
  - Uses numpy for most computations

Scikit-learn

- Linear classifier
  - Initialize model

```
clf = SVC(kernel="linear")
```

  - Fit model to data (x is data and y the corresponding labels)

```
clf.fit(x, y)
```

  - Predict label for data

```
pred_train = clf.predict(x)
pred_test = clf.predict(x_test)
```

Scikit-learn

- K-nearest neighbor
  - Initialize model (n_neighbors: number of neighbors to consider)

```
clf = KNeighborsClassifier(n_neighbors=3)
```

  - Fit model to data (x is data and y the corresponding labels)

```
clf.fit(x, y)
```

  - Predict label for data

```
pred_train = clf.predict(x)
pred_test = clf.predict(x_test)
```

Scikit-learn

- Decision tree
  - Initialize model (max_depth: how deep is the tree allowed to become)

```
clf = DecisionTreeClassifier(random_state=0,
max_depth=5)
```

  - Fit model to data (x is data and y the corresponding labels)

```
clf.fit(x, y)
```

  - Predict label for data

```
pred_train = clf.predict(x)
pred_test = clf.predict(x_test)
```

Scikit-learn

- Random forst
  - Initialize model (max_depth: how deep is the tree allowed to become, n_estimators: how many trees should the ensemble contain)

```
clf = RandomForestClassifier(max_depth=5,
random_state=0, criterion="entropy", n_estimators=100)
```

  - Fit model to data (x is data and y the corresponding labels)

```
clf.fit(x, y)
```

  - Predict label for data

```
pred_train = clf.predict(x)
pred_test = clf.predict(x_test)
```

Scikit-learn

- PCA
  - Initialize model (n_components: how many components should the PCA retain)

```
pca = PCA(n_components=4)
```

  - Fit model to data (x is data and y the corresponding labels)

```
pca.fit(x)
```

  - Predict label for data

```
x = pca.transform(x)
x_test = pca.transform(x_test)
```

Are there any questions ?