That's the relevant part of my summary regarding clustering and data analysis. In the end, there are a couple of pseudocodes that I wrote for exam preparation.
I also worked with a book called "A first course in systems biology by Voit".
I also know a couple of methods from a machine learning online course that I did as a preparation for the actual lecture this semester. But they are not in this summary (only in my head).
If you wish I can also send you the relevant slides/pdfs. Anyway, here you go!

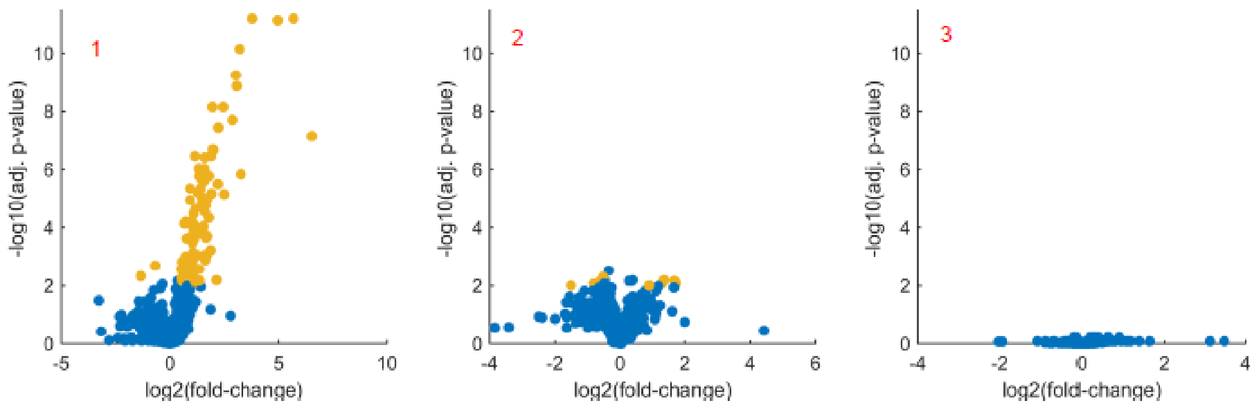
3 common problems in OMICS data: Overfitting: data is **noisy** and one might identify wrong patterns with the noisy data, leading to very **weak reproducibility (#of detected features >> #of samples)**.
In experiments, keep in mind to carry out positive and/or negative controls too.

Univariate analysis: Consider only one 1 feature at a time and compare it in the two groups (such as healthy vs sick or wildtype vs mutant etc.).
In order to derive a quantization of the difference of a feature in two groups, calculate the fold change:

FC = mean(group_1)/mean(group_2) (you can also use log_2(FC) to have a more symmetric visualization of the fold change).

Naturally, proceed with t-tests and the like (use alpha = 5% to get statistically acceptable answers. Alpha = 1% is nice to have.).

Fold change situations:

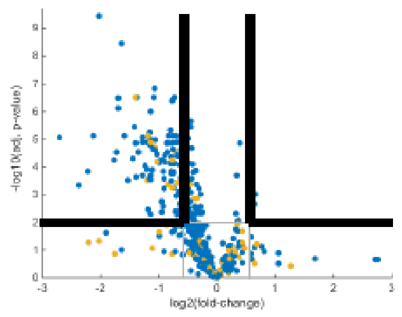


Zero significant changes 3	Check data quality Low p-values, high FC > increase number of replicates
Very few significant changes or very few “strong” changes 2	Simplest case > What is the identity of the markers?
A lot of significant changes No clear ranking 1	Quite common, but hard to generate hypotheses > Too many hits > Combination of primary and/or secondary effects > **Enrichment analysis!**

Note: One single feature can belong into multiple groups or processes.

In order to analyze the first scenario, where there are a lot of significant changes one can proceed with an enrichment analysis (gene set enrichment analysis in the context of genes). One can find out whether some features were overrepresented or not.

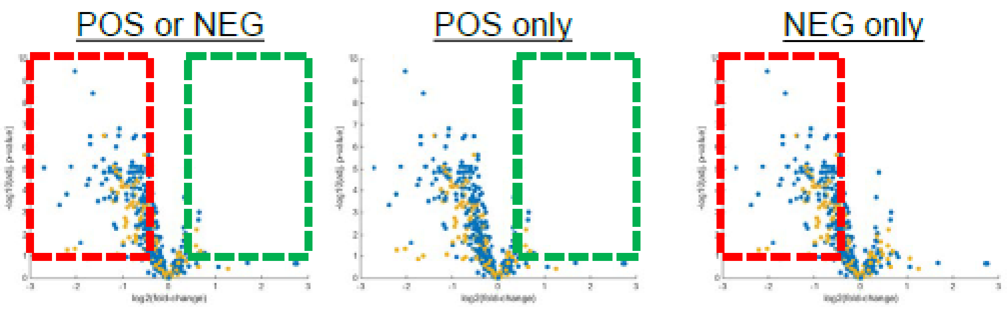
With a Fisher’s exact test, one can calculate the probability of obtaining the same results again assuming the null hypothesis is correct.
In MATLAB: [~,p] = fishertest([A B; C D], ‘tail’, ‘right ’)
To find the right numbers A, B ,C ,D I have to define them properly according to a contingency table:



	In group (G+)🟡	Not in group (G-) 🔵
Significant (S+)	A	B
Not significant (S-)	C	D

GSEA procedure:

1. **Decide on what signs do we want to include:**



2. **Procedure for 1 pathway/group/process:**
    1. Choose very permissive thresholds (e.g. log2FC > 0.2 and p-value < 0.1)
    2. Rank all hits on either p-value (preferred) or log2FC
    3. Build contingency tables with either 2, 3, 4, …., ALL top hits
    4. For each table, calculate p-value using Fisher's exact test
    5. Keep lowest p-value = "best" enrichment

3. **Repeat for all pathways**
4. **FDR-correction (Benjamini-Hochberg or Storey)**

Feature selection

In systems biology (and biology in general), we want to find the major component that influences the behaviour or outcome of a biological system (can be phenotype for example that is strongly influenced by one single gene/transcription factor, a group of genes or maybe miRNA/siRNA or small molecules).

Feature selection is used for: finding correlations or causalities of a particular phenotype regarding its underlying molecular mechanisms; finding lower dimensional models that are easier and cheaper to study; removing noisy features.

Mathematically, evaluate:

$$\arg\max_{j}(r(\mathbf{x}(:,j),\mathbf{y}))$$

With r: R^n -> R^n, x(:,j) is a a vector with the expression levels of the object in question of object j across all n elements (e.g. object = gene and elements = patients, so x is the vector of the expression levels of gene j across all patients n) and y is the vector of phenotypes of the patients.

General framework: 1) compute score r(j) for all j (for-loop or while-loop for example).
2) sort features acccording to score r(j) (highest to lowest for example) and return a sorted list.

There are two approaches to determine the number of most relevant genes:

1) Generate a random feature selection z and then chose: r(j) > r(z). (**Probe method**)

2) Carry out statistical tests on all scores r(j) and choose those that are statistically significant (value must be lower than significance value alpha). (**Significance method**)

Due to lots of statistical tests carried out, false positives are prone to occur. Therefore, use corrections such as **bonferroni correction** (divide alpha by number of total tests to be carried out; is rather conservative though which is why fewer significant genes) or **false discovery rate** (less conservative).

Keep in mind that such lists can be rather unstable that is that when you only analyze the subset of the data, it can give a completely different ranking.
To counter the problem, one can either calculate the average rank of all genes in all experiments or calculate the probability to be in the top-k genes.

Limits of univariate selection:

It can only capture the effect of one single gene, it ignores the additivity of multiple genes, it ignores the correlation of genes, it ignores the interaction of genes.

Multivariate selection: solve

$$\arg\min_{\beta} ||\mathbf{y} - \mathbf{X}\beta||_2^2$$

Lasso model:

$$\arg\min_{\beta} ||\mathbf{y} - \mathbf{X}\beta||_2^2 + \lambda_1 ||\beta||_1$$

Idea: Reward solutions ($\boldsymbol{\beta}$) in which few entries of $\boldsymbol{\beta}$ are non-zero.

Concept: This is achieved by minimizing the L1-norm of $\boldsymbol{\beta}$:

$$||\beta||_1 = \sum_{i=1}^{d} |\beta_i|$$

Disadvantage: If there are groups of correlated features, the Lasso often picks just one feature from a group.

Ridge regression:

$$\arg\min_{\beta} ||\mathbf{y} - \mathbf{X}\beta||_2^2 + \lambda_2||\beta||_2^2$$

Idea: Reward solutions (**β**) in which correlated features get similar weights.

Concept: This is achieved by minimizing the L2-norm of **β**:

$$||\beta||_2 = \sqrt{\sum_{i=1}^{d} \beta_i^2}$$

Disadvantage: The solution is often not sparse.

The L2-norm rewards reducing larger values more than smaller values, whereas the L1-norm rewards both cases identically.

Elastic net:

$$\arg\min_{\beta} ||\mathbf{y} - \mathbf{X}\beta||_2^2 + \lambda_1||\beta||_1 + \lambda_2||\beta||_2^2$$

Idea: Reward solutions (**β**) in which groups of correlated features get similar weights and few weights are non-zero.

Concept: This is achieved by simultaneous minimization of the L1-norm and the L2-norm of **β**.

Disadvantage: Two parameters have be to set.

Avoid selection bias (overfitting) by only using feature selection on the **training datasets** and making predictions from the previously selected features only on the **separate test dataset**.

Clustering

k-means: most common and popular clustering method: Lloyd’s algorithm:

1. Randomly pick k points as initial cluster means.

2. Assign each points to its nearest cluster mean.

3. Recompute the mean of each cluster.

4. Repeat steps 2 and 3 until cluster assignment does not change any more.

Limitations of k-mean:

Number of clusters k has to be specified by the user (how many clusters should I choose?).
Since the first centroids are initialized randomly, there can be different clusters depending on the initial point of initilization (idea for a solution: repeat clustering with the same number of clusters

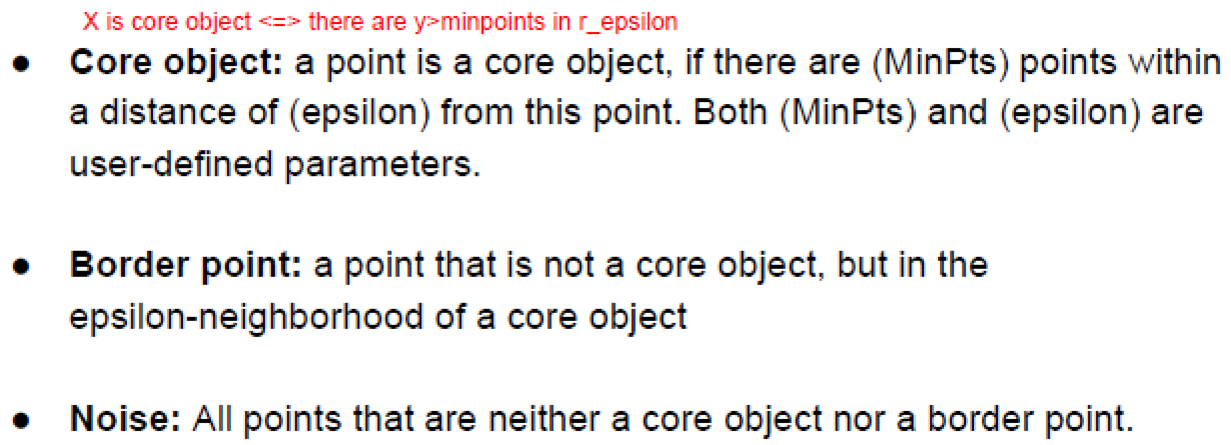
several times until you can identify the most common cluster).
k-means works very well with spherical data distributions but fails with non-spherical ones such as concentric circles – it misses clusters/it cannot identify them.
Solutions of Lloyd's algorithm are local optimum – better solutions might exist.

Graph based clustering:

Principally, one starts with a network consisting of nodes and weighted paths:
1) Remove all weights, such that weigths > user_defined_weight_theta
=> all paths removed that have a smaller weight than theta.
2) Find all remaining connected components in the resulting graph.
3) Each component (a subgraph basically) is a cluster.

Since this also captures noise data (graph clusters are NOT robust to noisy data), we end up with incorrect graphs, which is why use DBSCAN (:= density based spatial clustering of applications of noise).

Definitions:


X is core object <=> there are y>minpoints in r_epsilon

- **Core object:** a point is a core object, if there are (MinPts) points within a distance of (epsilon) from this point. Both (MinPts) and (epsilon) are user-defined parameters.
- **Border point:** a point that is not a core object, but in the epsilon-neighborhood of a core object
- **Noise:** All points that are neither a core object nor a border point.

DBSCAN algorithm:

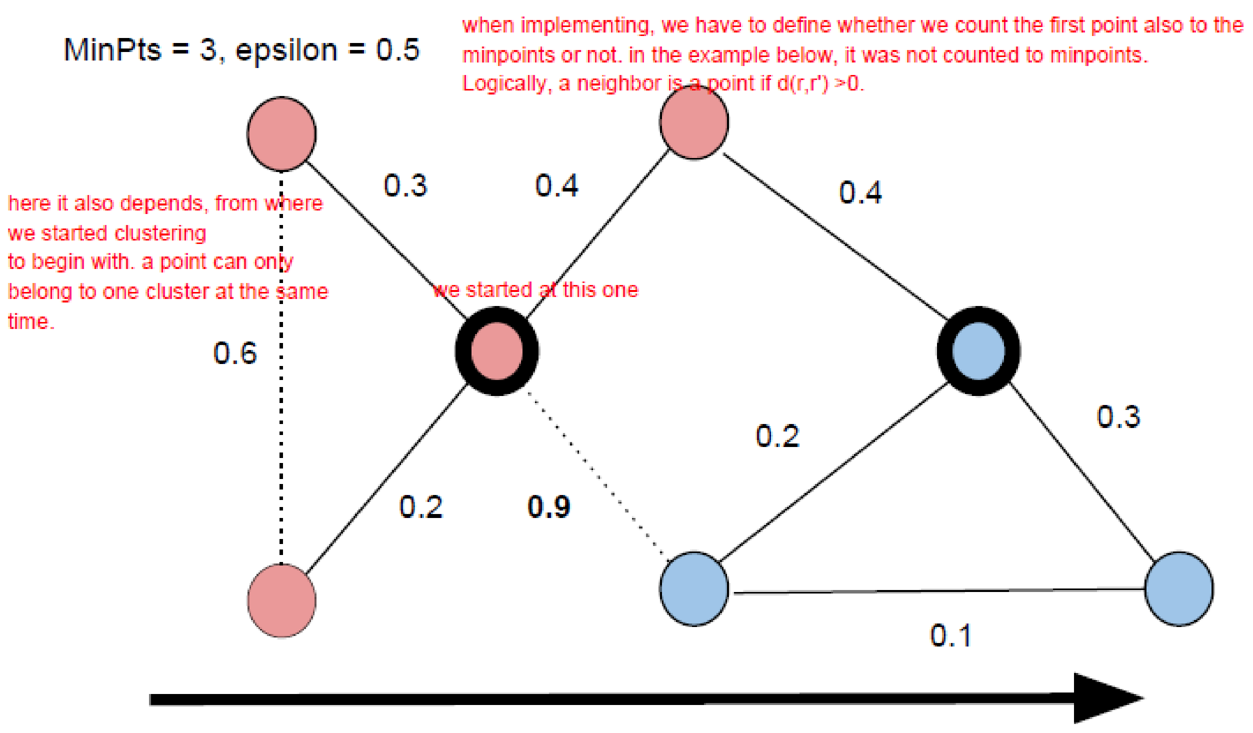
1. Pick a point that has not been assigned to a cluster yet.

2. If it is a core object, add it to a new cluster C (if not, label it as "noise").

3. Add its neighbors to the same cluster C.

4. Check for each of the neighbors whether it is a core object.
 a. If yes, assign its neighbors to C and perform Step 4 for these neighbors.
 b. If no, do not further extend the cluster from this point.

5. Return to Step 1 until all points have been clustered.


MinPts = 3, epsilon = 0.5
when implementing, we have to define whether we count the first point also to the minpoints or not. in the example below, it was not counted to minpoints. Logically, a neighbor is a point if d(r,r') >0.
here it also depends, from where we started clustering to begin with. a point can only belong to one cluster at the same time.
we started at this one
0.3
0.4
0.4
0.6
0.2
0.3
0.2
0.9
0.1

Advantages of DBSCAN:
No need to set the number of clusters as in k-means.
It is able to find clusters that k-means misses.
It is more robust to noise data than is the naïve graph-based clustering approach.
(Successful in many clustering applications.)

Disadvantages of DBSCAN:

Two parameters have to be set. If densitiy varies as is the case in high-dimensional data it will often miss clusters. The results are initialization dependent.

Hierarchical clustering: This type of clustering can identify clusters in clusters. It can give a sort of 3D insight into the clustering of data (hierarchy).

Principally, every data point is defined as its own cluster, than the two most similar will always form a cluster together. Then all these resulting clusters will be clustered together with their most similar one and so on until there is only one cluster left (trivial case basically).

Distance types in hierarchical clustering: single link, average link, complete link:

$$d_{single} = \min\{d(\mathbf{x}, \mathbf{x}') | \mathbf{x} \in C, \mathbf{x}' \in C'\}$$

$$d_{average} = \mathrm{mean}\{d(\mathbf{x}, \mathbf{x}') | \mathbf{x} \in C, \mathbf{x}' \in C'\}$$

$$d_{complete} = \max\{d(\mathbf{x}, \mathbf{x}') | \mathbf{x} \in C, \mathbf{x}' \in C'\}$$

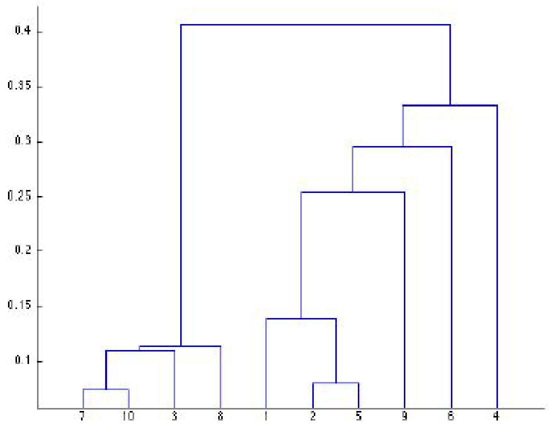
MATLAB

```
z =rand(10,2)

d = pdist(z)

l =linkage(d)

dendrogram(l)
```



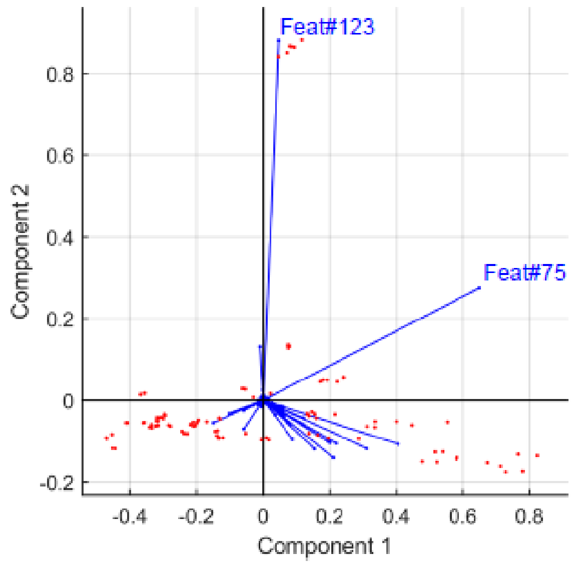
Dimension reduction

We want to analyze multidimensional, multifeature data. The idea is to transform such huge matrices into another form, which is easier to computationally analyze and it still preserves the underlying information (e.g. by removing redundancies and noise data etc.).

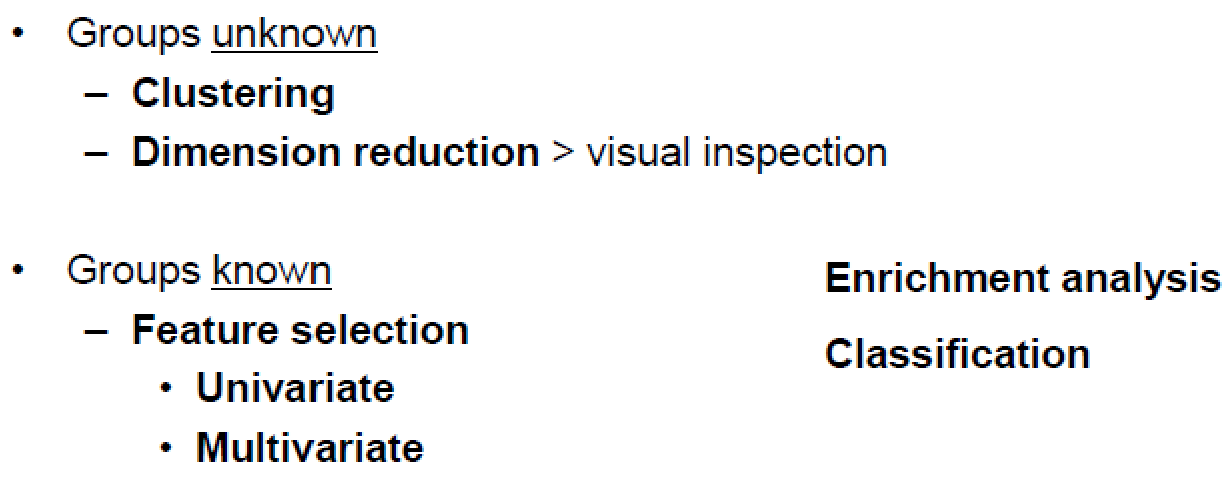
Principal component analysis: Method used to reduce dimensions (variables) of a multidimensional dataset for a better visualization, for quality control (it can spot systematic errors) and for preprocessing before other data mining techniques are used. (I have to be able to draw the PC's in a given dataset).

Bi-plot:

Sample scores and Feature loadings are scaled and overlaid to emphasize Associations.



- Samples that are close have similar profile
- Features that are close are correlating
- Samples that are close to a feature have high levels of the same feature.
- Samples that are opposite to a feature, have low levels of the same feature.



- Groups unknown
  - **Clustering**
  - **Dimension reduction** > visual inspection
- Groups known
  - **Feature selection**
    - **Univariate**
    - **Multivariate**

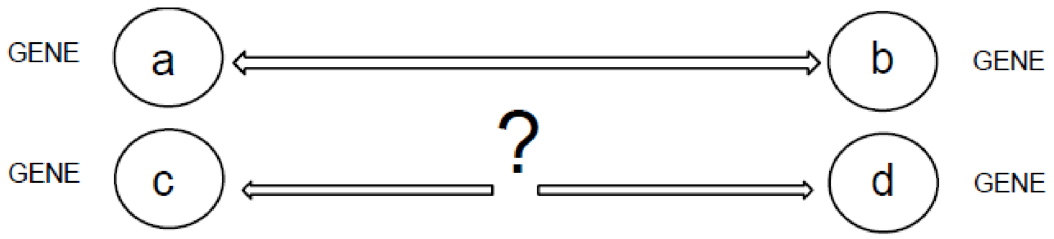
Enrichment analysis

Classification

Link prediction

In link prediction, we try to predict missing connections in networks that cannot or are hard to be found by experimental means in the lab (it is also used outside of biology, e.g. in predicting the preference of movies in netflix).

General problem:



Def. supervised link prediction: We are given an example of pre-existing links and learn a prediction model based on these examples.

Def. unsupervised link prediction: The prediction model is based on a rule and predicts according to a rule instead of learning from examples.

Approaches to link prediction

An example for unsupervised link prediction:

Similiarity based approach: If s(a,b) > theta, theta is user defined, then infer a link between a and b.
Problem: How to set theta? Is similarity necessary for an interaction?
Similarity measures that can be used: Pearson's correlation coefficient, mutual information, number of shared neighbours, string kernels that count common subsequences in two proteins sequences (k-mers).

2nd example for unsupervised link prediction:

Similarity based approach (kernel approach): 1) **Tensor pairwise approach**

Given two pairs of nodes *(a, b)* and (c, d).

$$k_{tensor}((a, b), (c, d)) = k_{nodes}(a, c)\ k_{nodes}(b, d) + k_{nodes}(a, d)\ k_{nodes}(b, c);$$

kernel-function is big, when the k_nodes are similiar:
k_tensor = N, N big => similarity big

This kernel quantifies the similarity of the source and target nodes in both edges, for both directions.

k_{nodes} is a kernel that measures the similarity of two nodes.

2) **Metric learning pairwise approach**:

Given two pairs of nodes (a, b) and (c, d).

maybe genes react, when gene A has feature A and gene B has feature B and then a reaction between them occurs

$$k_{ml}((a, b), (c, d)) = [(\varphi(a) - \varphi(b))^T(\varphi(c) - \varphi(d))]$$

$\varphi(g)$ is a vector that describes features of gene/protein *g*.

A pair *(a, b)* is similar to a pair *(c, d)* if $a - b$ is similar to $c - d$, or if $a - b$ is similar to $d - c$.
this function is about differences, instead of the similarity of the starting node and end node as in the previous slide

An example for supervised link prediction:

Cluster based learning: A predictor can learn from already known interactions and non-interactions: Predict a link between a and b if a and b belong to the same cluster C.
Advantage: More general than pairwise similarity.
Problem: Biologically unrealistic maybe, since interaction only depends of membership of a cluster.

2nd example of supervised link prediction:

Inferring interaction probability between genes: latent group models.
Division into two phases: training phase and prediction phase.

Training phase:

1. Pick a subset S of genes.

2. Cluster genes from S into k different groups based on their gene expression profiles.

3. For each pair of clusters i and j, determine the empirical interaction probability p_ij.

Prediction phase:

1. Given a new pair of genes a and b.

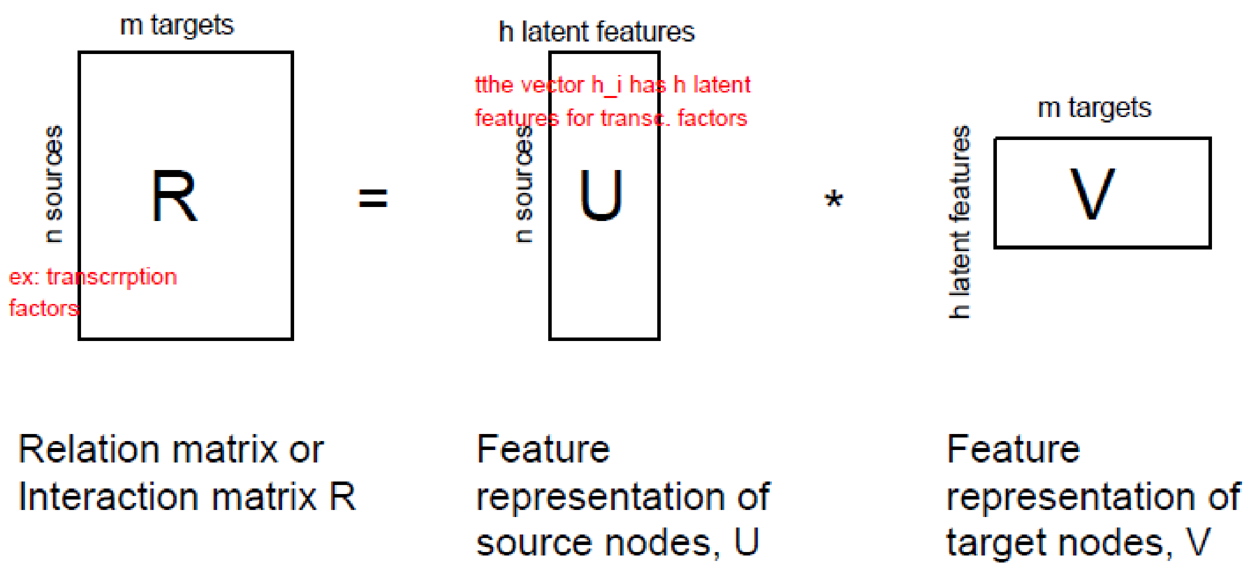
2. Assign a and b to the most similar Clusters C_a and C_b.

3. Report interaction probability p_a,b learned in the training phase.

In cluster based link prediction, the interaction of two genes only depends on **one latent variable**, which is cluster membership. Naturally, one can predict links between two genes with several latent or hidden variables. A latent variable can be thought of an imaginary, virtual or theoretical characterstic which has no biological origin, but is computationally derived.

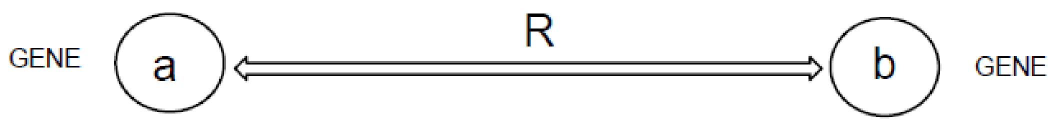
Latent feature based link prediction:

Given a matrix R, which has numbers (data, measurements of expression, activity etc.) as its entries. Then, R = U*V, where U has the colomns as its latent features and V has its rows as latent features:


m targets
n sources
R
ex: transcrrption factors
=
h latent features
tthe vector h_i has h latent features for transc. factors
n sources
U
*
m targets
h latent features
V
Relation matrix or Interaction matrix R
Feature representation of source nodes, U
Feature representation of target nodes, V

R has missing entries (ideally). Matrix completion is equivalent to link prediction in the systems biology interpretation.

Evaluation of link predictions



True Label (R) vs. Predicted Label (R*)	R = 1	R = -1	
R* = 1	TP	FP	
R* = -1	FN	TN	
	P	N	

P = positive
TP = true pos.
FN = false neg.

N = negative
FP = false pos.
TN = true neg.

TP+TN is true predictions

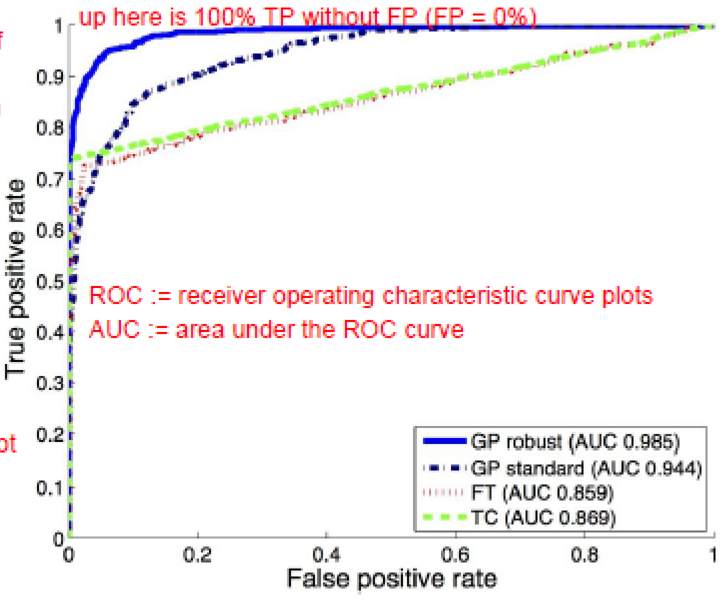
Formulas:

- Accuracy = (TP + TN)/(TP + TN + FP + FN)
- Sensitivity = TP/P, also called recall or true positive rate
- Specificity = TN/N
- False positive rate = FP/N = 1 – specificity
- Precision = TP/(TP + FP)

ROC & AUC

we dont know where to cut off the rangliste of all possible and most probable interaction thats why we do iteratively:
first: only top interaction is TP all other all negative
2nd: only top 2 intera. is TP, all other neg.
...
i-th: only top-i intera is TP, all other neg.

thats why we get a plot and not only a single dot



The AUC is the probability that the classifier will correctly discover which one is the positive example.

Problem: Even when AUC is high, a classifier can be rather useless when one class is a lot larger than the other class.

PSEUDOCODES:

Score: Calculates the goodness of fit (page 2 in my summary – it is the phi(K_m) function with unknown parameter K_m)

```
1    score = 0                 %initialize a variable score
2    for i in length(metabolites)
3       for j in 1:n    %1 to n are the discrete time points
4          temp = ((simulation_data(j,i) – experimental_data(j,i))/stand_deviation(i))**2
5              score = score + temp
6       end
7    end
```

k-means (Lloyd's algorithm): naïve clustering algorithm

```
1    init_ind = randperm(size(data), number_of_clusters)
2    centroids = (data(init_ind), :)
3    previous_ind = zeros(size(data), 1)      %vector that will contain previous indices to check
                                                   convergence
4    while true
5       distances = euclidean_dist(data, centroids)
6       new_ind = min(distances, [], 2)       %returns min value of all columns as row vector
7       if all(previous_ind == new_ind), then break    %end here
8       else: prevous_ind = new_ind
9       for j in 1 to number_of_clusters: centroids(j,:) = mean(data(new_ind==j,:),1); end
```

Add "iter = 1" before the while-loop and "iter += 1" after the for-loop but still inside the while-loop to get the number of iterations till convergence.

Sorting algorithm: Sort a vector (I think something is missing, therefore search in internet or in python)

```
1    Let vec be the vector with values (data)
2?   ind_vec = indices(vector)             %ind_vec contains the indices of vec
3    for k in length(vec)
4    for j in length(vec-1)
5       if vec(k) > vec(j+1), then temp = vec(j+1)
6          vec(k) = vec(j+1)            %value in vec(k) is saved into vec(j+1)
7          vec(k) = temp                %the smaller value is restored into the spot vec(k)
8       end; end; end
```

simplex algorithm: ??? (what's a simplex algorithm – Sysbio-5.5 and internet, because it's not really explained in the pdf)

Flux Variability Analysis (=: FVA):

Inputs: S is a m x n stoichiometric matrix (has the reaction rates as its entries)
alpha and beta are two n x 1 vectors with lower and upper boundary conditions for reactions (optimization function)

```
1   F = zeros(n,2)
2   for d in 1:2
3      for z in 1:n
4         w = zeros(n,1)
5         w(z) = (-1)**d
6         Run FBA, that is, solve objective_func(v0,w,S,alpha,beta) = v_z
          %maximize w^T * v, s.t. Sv = 0 under constraints alpha_i and beta_i (⇔ FBA)
7               F(z,d) = v_z
8      end
9   end
```

OptKnock: ??? (where to find and what is that)

Link prediction: … (Sysbio-12)

DSCAN algorithm (Sysbio-10): Noise-robust graph based clustering

Inputs: S is a m x m matrix with the weighted distances between two nodes. Note that it has 0 on its diagonal and is symmetric.
minpoints and r_eps are user inputs, with minpoints being the minimal points of a point in its neighbourhood with radius r_eps. Cluster is a list that saves the core objects and its neighbours and temp is a list.

```
1    Pick a random point and check if it is a core object ⇔ point x has >= minpoints in r_eps
1a   count = 0; m = length(S(x,:))
1b   for j in 1:m
1c      if S(x,m) > r_eps, then count += 1; add neighbour_index to temp;
1d   if count >= minpoints, add all neighbours and x to Cluster        %then x is a core object
1e   else: define x as noise and pick another random point
2    for all neighbours n_i, repeat 1a-e: while temp_next or temp ~= empty
2a   for n in length(temp): set count to 0 again
2b      if S(temp(n),m) > r_eps, then count += 1; add neighbour_index to temp_next;
2c   if count >= minpoints, add all neighbours and x to Cluster        %then x is a core object
```