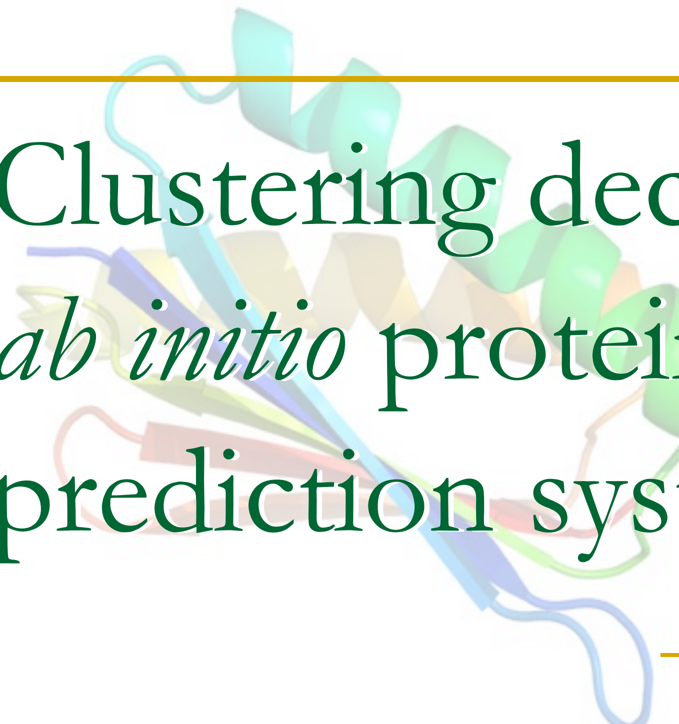
# Clustering decoys produced by *ab initio* protein structure prediction systems



Shuai Cheng Li
*City University of Hong Kong*

RosettaCon 2012
July 29th – August 1st

# Protein structure prediction

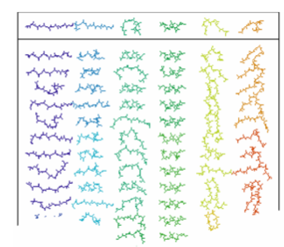
- The prediction of the three-dimensional structure of a protein from its amino acid sequence (primary structure)
  - Secondary structure prediction
  - Tertiary structure prediction
  - Quaternary structure prediction

# *Ab initio* protein structure prediction

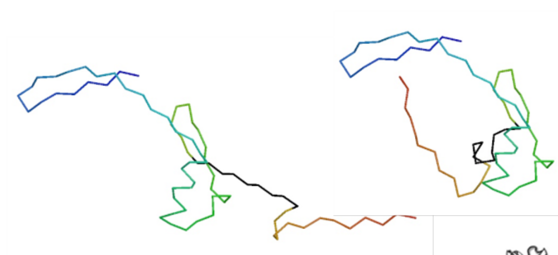
- Reconstruct the tertiary structure “from scratch”
- Typically modeled as a problem of finding the most *stable* (in terms of energy) structure that an amino acid sequence folds into
  - Enormous number of structures to search
  - Adding biases into the search
    - Threading / Assembly / Refinement

# Threading / Assembly / Refinement

- Proposed in ROSETTA (Simons et al. 1999)
  - Used by many other methods
    - I-TASSER (Wu, Skolnick and Zhang, 2007)
    - Fragment-HMM (Li *et al*. 2008)
    - *etc.*
- Threading
  - Scans the amino acid sequence of an unknown structure against a database of solved structures

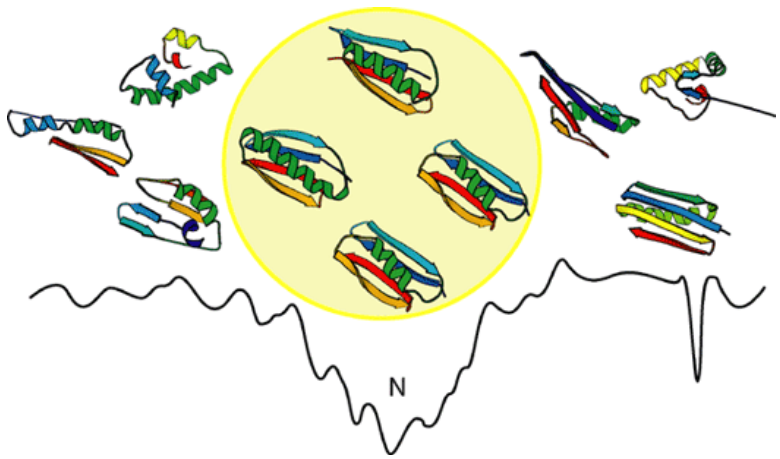


- Assembly / Refinement:
  - Depends on the method



# Finding representative decoys

- Candidate structures called decoys are generated
- Decoys need to be clustered before the representative ones are determined

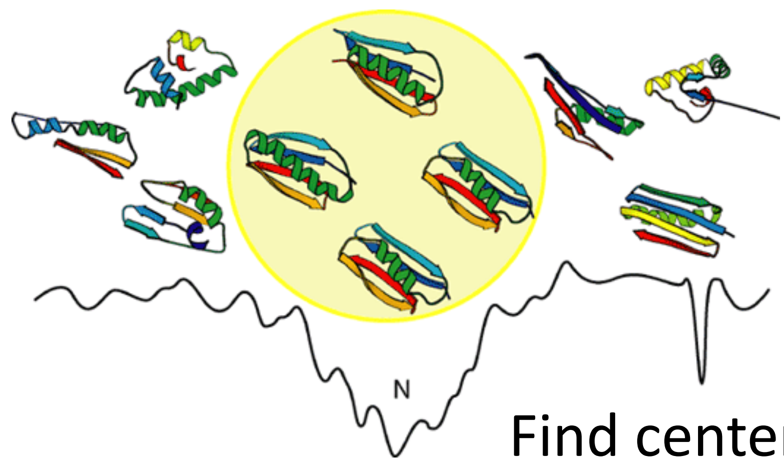



- Typically, thousands to tens of thousands such decoys are generated

Image source: http://depts.washington.edu/yeastrc/pages/rosetta.html

# Finding representative decoys

- Candidate structures called decoys are generated
- Decoys need to be clustered before the representative ones are determined




Find center of largest cluster as the representative decoy

- Typically, thousands to tens of thousands such decoys are generated

Image source: http://depts.washington.edu/yeastrc/pages/rosetta.html

# Clustering in ROSETTA

- In most systems (including ROSETTA, I-TASSER and Fragment-HMM), clustering is done as follows
  - Starting with the set of generated decoys, a threshold *d* is first decided.
  - From the set, the decoy with the most neighboring decoys within **RMSD** *d* from it is found, and is reported as the highest ranking decoy. (Ties are broken arbitrarily.)
  - This decoy and all of its neighbors (the first cluster) are then removed from the set, after which the decoy with the most neighbors within **RMSD** *d* is again found.
  - This decoy is reported as the second highest ranking decoy, and together with all its neighbors (the second cluster) are removed from the set.
  - Similarly the third highest ranking decoy is then found, and so on.

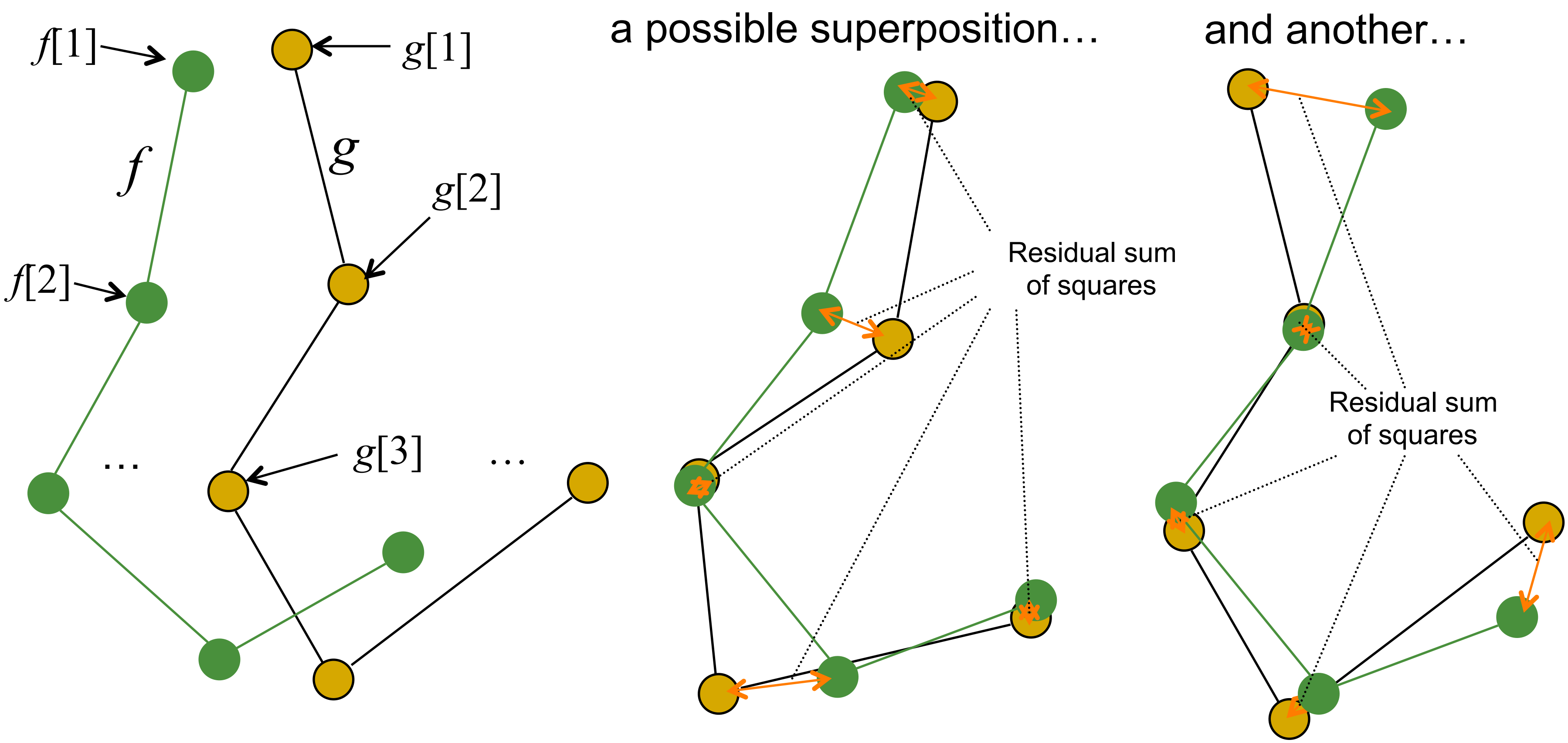
# Root Mean Squared Deviation (RMSD)

- Given two structures of length $n$,
  - $S_1=(S_{1,1}, S_{1,2}, \ldots S_{1,n})$
  - $S_2=(S_{2,1}, S_{2,2}, \ldots S_{2,n})$

  The RMSD between $S_1$ and $S_2$ is computed as

$$\mathrm{RMSD}(S_1, S_2) = \min_{R,T} \sqrt{\frac{\sum_{i=1}^{n} \left\| RS_{1,i} - S_{2,i} - T \right\|^2}{n}}$$

# Root Mean Squared Deviation (RMSD)




Aim is to find the superposition ($R$, $T$) which minimizes $\frac{1}{\sqrt{n}}\sqrt{\sum_{i=1}^{n}\left\|RS_{1,i}-S_{2,i}-T\right\|^2}$

# Implementations of ROSETTA's clustering

- ROSETTA (Simon *et al.*, 1999)
  - Uses a slow but accurate method for determining threshold $d$
- SPICKER (Zhang and Skolnick, 2003)
  - Straight-forward re-implementation of ROSETTA's clustering method in FORTRAN
  - No attempt at accurately determining threshold $d$
  - No dynamic memory allocation – clusters at most 10,000 decoys
- SCUD (Li and Zhou, 2005)
  - Faster computation by using an approximation of RMSD instead of actual RMSD
- Calibur (Li and Ng, 2010)
  - Uses heuristics to speed-up clustering with RMSD

# Find decoys with the most neighbors

- Given a threshold for similarity $t$:

Exhaustive method

For each decoy $d$,

$N[d] \leftarrow 0$, ($N[d]$ = number of neighbors of $d$)

For each decoy $d'$,

If $\mathrm{RMSD}(d, d') \leq t$; then $N[d] \leftarrow N[d] + 1$.

Output the decoys with the largest $N[d]$.

- Runtime is $O(n^2)$, $n$ = number of decoys
- Two problems:
  1. How to determine the threshold $t$?
  2. Expensive RMSD computation slow for large $n$ ($\geq 10000$)

# Calibur: Speeding-up exhaustive method

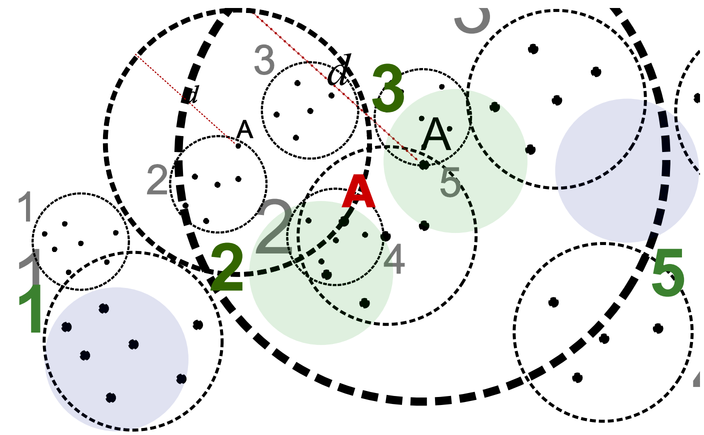
- **Group decoys into proximity groups**

  Example: Groups 1-5

  - When finding decoys similar to decoy **A**:

    1) All decoys in **Groups 2 and 3** are within RMSD $d$

    2) All decoys in **Groups 1 and 5** are above RMSD $d$




- **Use efficiently computable lowerbounds and upperbounds of RMSD to skip RMSD computation whenever possible**, i.e.

$$\text{Lowerbound\_of\_RMSD}(d, d') \geq d \Rightarrow \text{RMSD}(d, d') \geq d$$

$$\text{Upperbound\_of\_RMSD}(d, d') \leq d \Rightarrow \text{RMSD}(d, d') \leq d$$

# Calibur: Threshold determination

- Threshold determination used in ROSETTA and I-TASSER
  - Based on largest number of neighboring decoys
    - Example: Find $t$ such that the largest $N[d]$ is of size about 10~20% of the total number of decoys
  - Problem: difficult to compute

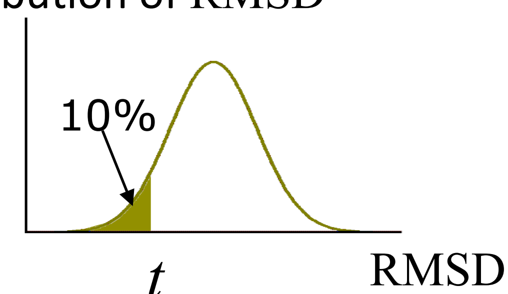
**Calibur's threshold finding principle**

Consider two decoys as significantly similar iff their RMSD is relatively small among all pairwise RMSDs

⟹ Find $t$ such that only ~10% percent of all pairwise RMSDs are below $t$

- Observation: pairwise RMSDs follow normal distribution
- $t$ can be estimated efficiently using sampling distribution

Distribution of RMSD



# Calibur: Filtering outliers

- Method: Discard decoys with low similarity to other decoys
- Difficulty: To retain all high ranking decoys, and the decoys which are within distance $d$ from them ("good" decoys)
- Assume: Every high ranking decoy is within distance $d$ from $10\%$ of all decoys

**Calibur's filtering of outliers**

Randomly sample $x$ decoys.

For each decoy $y$, discard $y$ if it is not within $2d$ from any of the sampled decoys.

- Analysis: Probability that a "good" decoy is within distance $2d$ from a random decoy = $0.1$

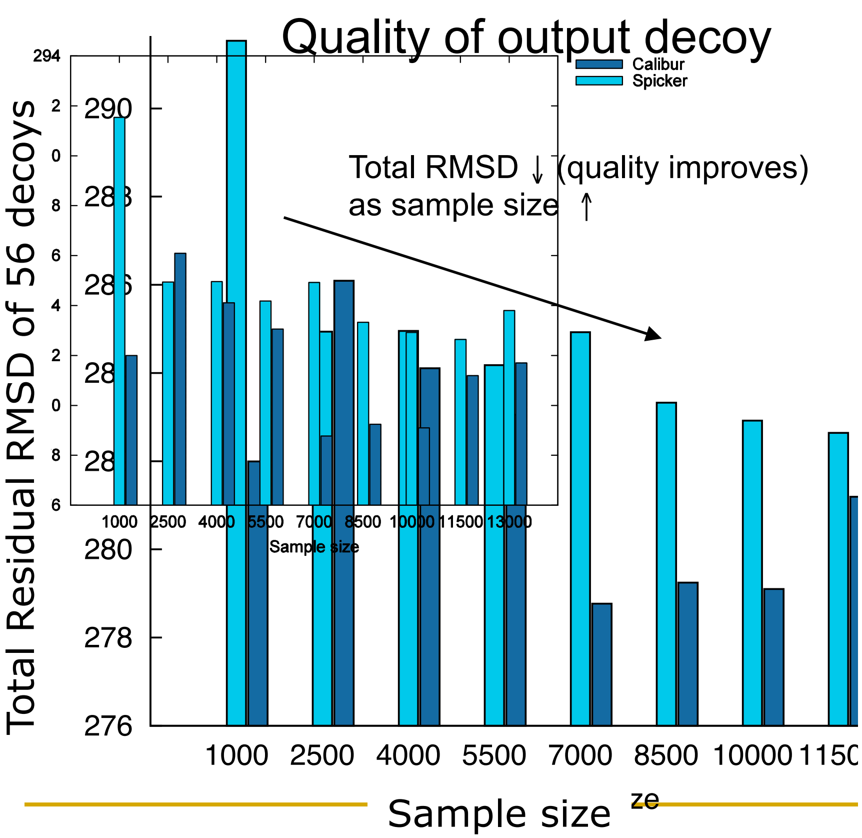
$\Rightarrow$ Probability that a "good" decoy is within distance $2d$ from at least one of $x$ decoys = $1 - 0.9^x$ ($\geq 0.99999$ for $x=100$)

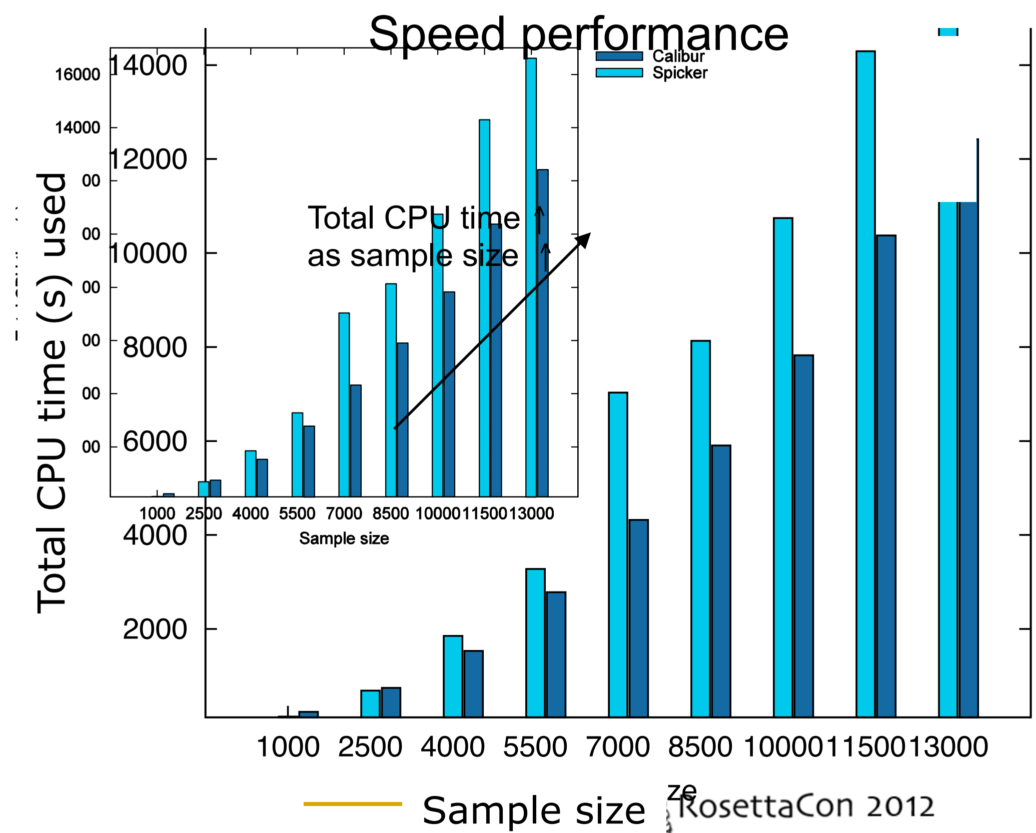
$\Rightarrow$ Highly unlikely to discard "good" decoys

# Calibur: Results

- Compared with SPICKER (clustering tool used in I-TASSER)
- 56 proteins + 56 sets of decoys, each set of size >12000
- Experiment on samples of sizes 1000, 2500, 4000, …, 13000

SPICKER Calibur



# How about other clustering methods?

- For instance, $k$-means clustering

# $k$-means Clustering

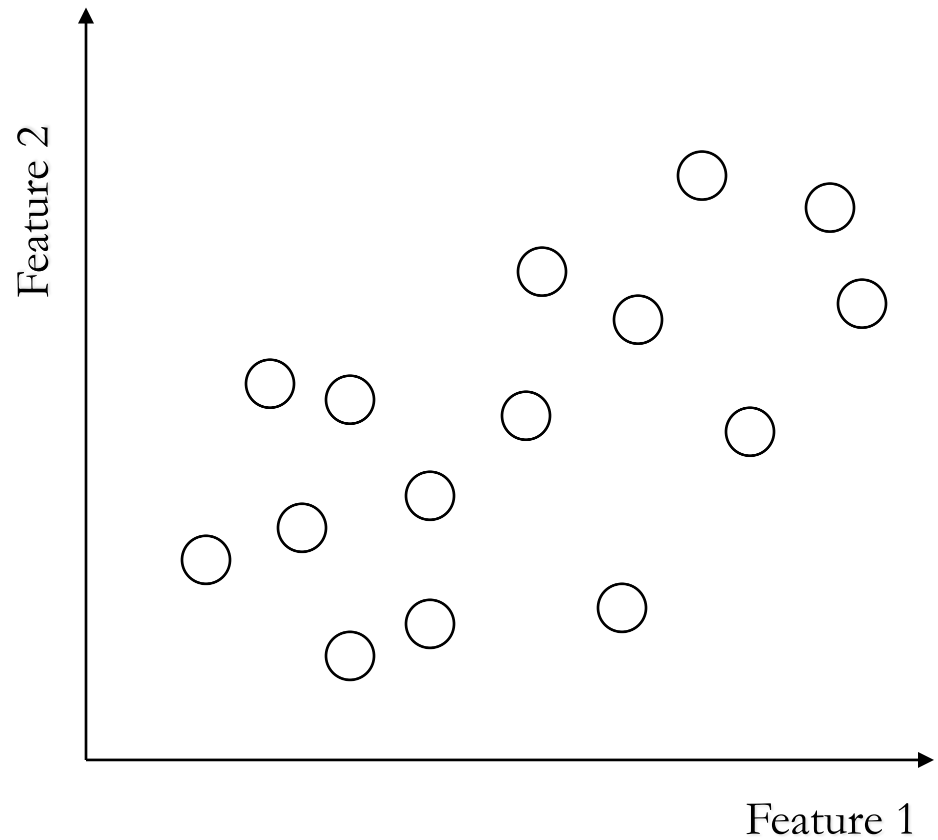
- $k$-means clustering is a heuristic method which aims to solve the following problem:
  - Given $n$ decoys $S_1, S_2, \ldots, S_n$, $k$-means clustering aims to cluster the decoys into $k$ sets, $\mathbf{A} = \{A_1, A_2, \ldots, A_k\}$, to minimize

$$\arg\min_{\mathrm{A}} \sum_{i=1}^{k} \sum_{S_j \in A_i} \left\| S_j - \mu_i \right\|^2$$

where $\mu_i$ is the centroid of the set of decoys $A_i$

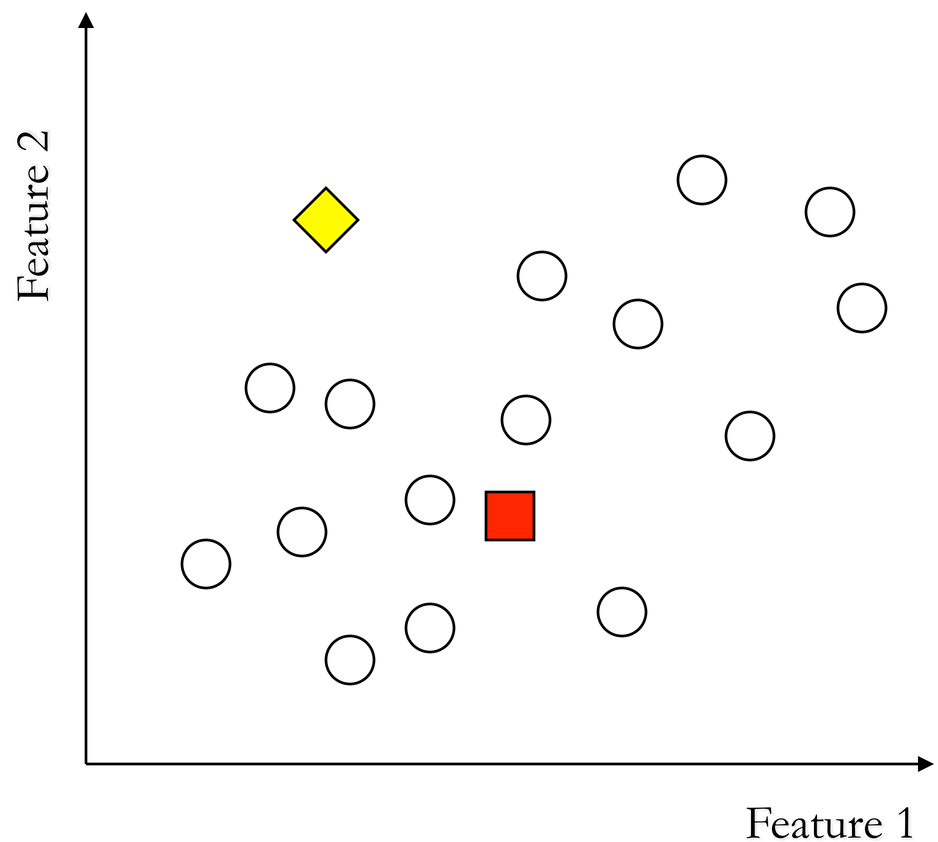
# *k*-means Clustering

- Problem: cluster examples into $k$ groups
- Example: Cluster the given examples into 2 groups



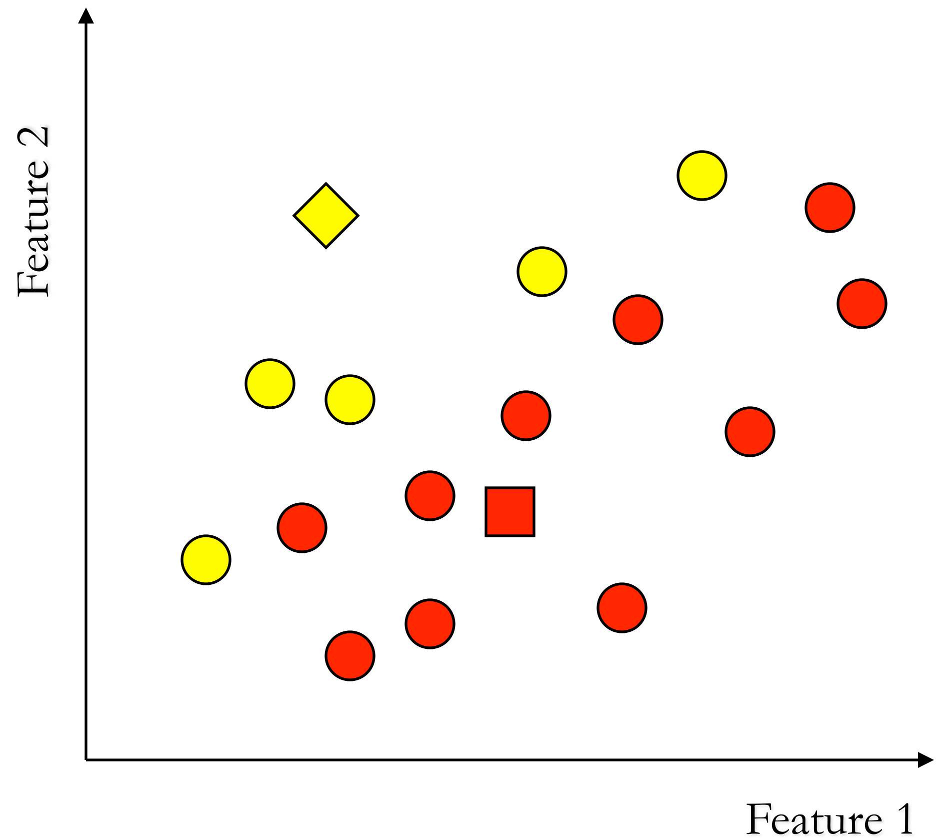
# *k*-means Clustering

- Randomly initialize cluster centers



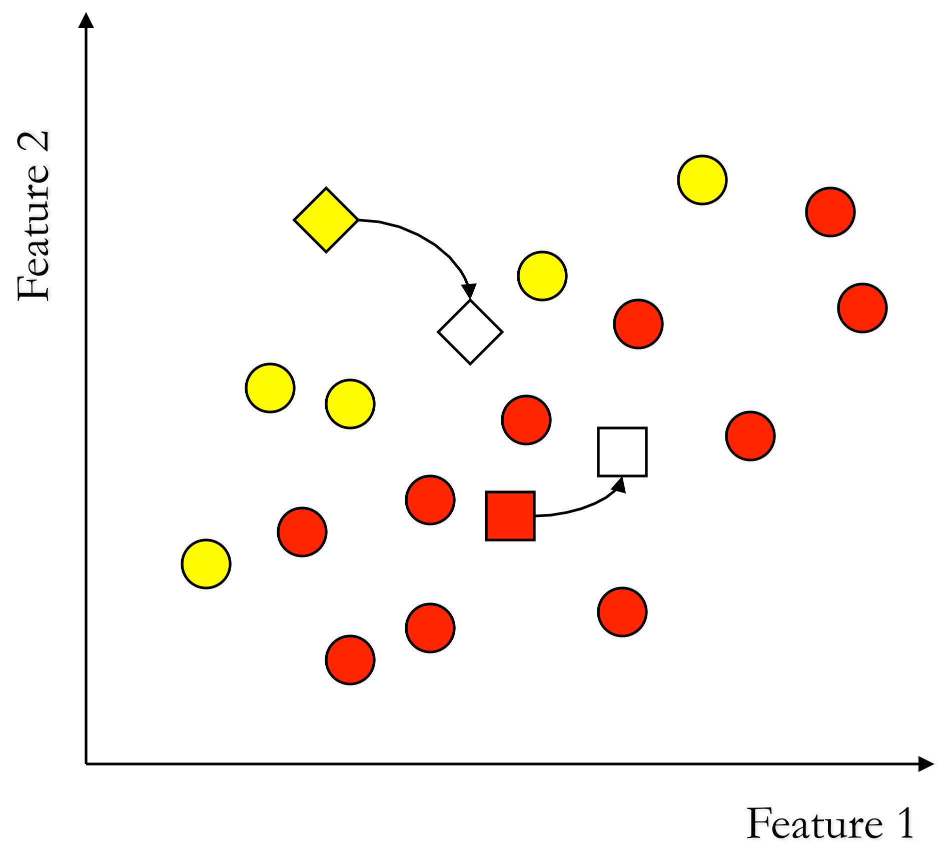
# *k*-means Clustering

- Classify samples according to the nearest cluster center
- Different distance measures can be used, e.g.
  - Euclidean distance
  - Manhattan distance



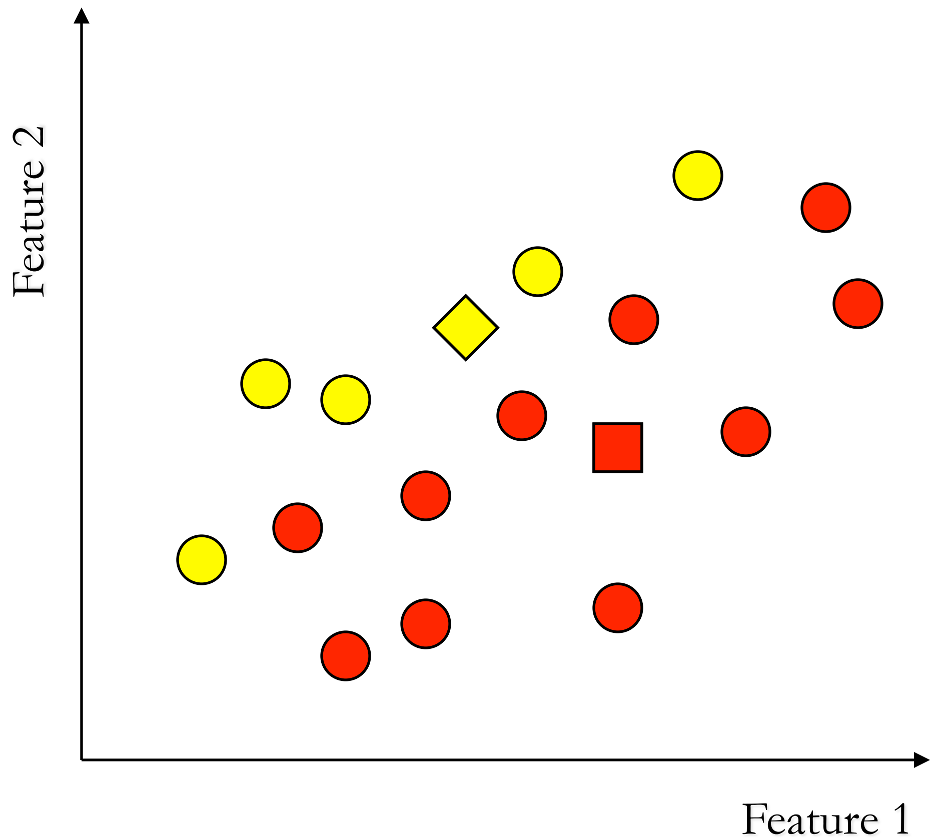
# *k*-means Clustering

- Re-compute cluster centers



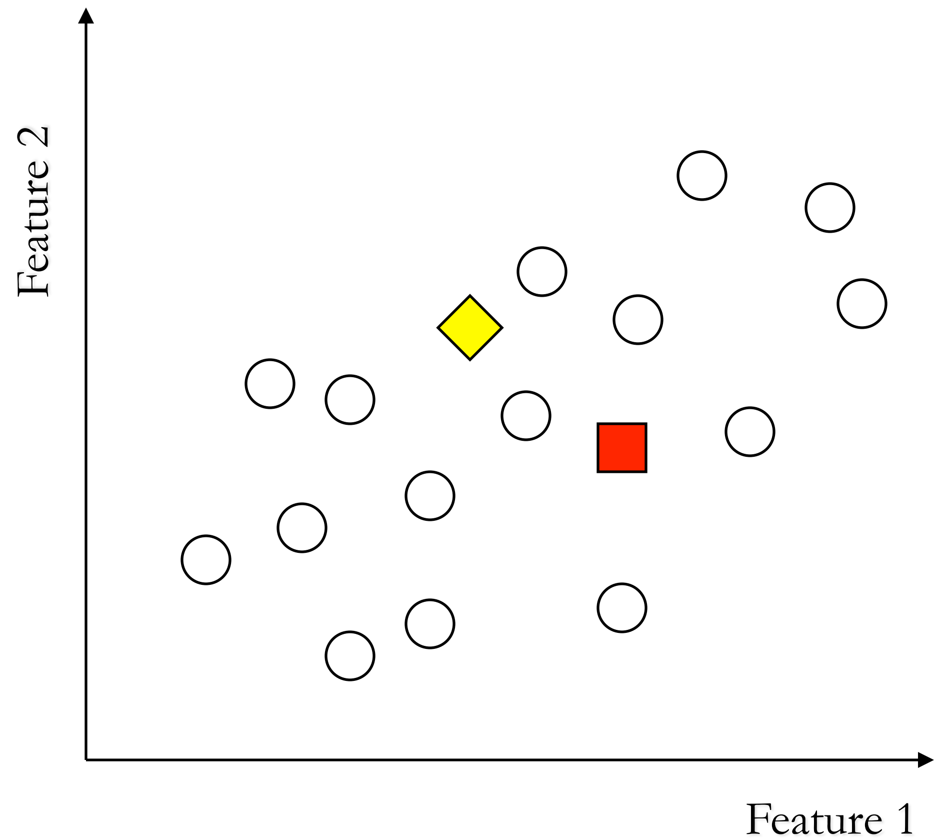
# *k*-means Clustering

- Cluster centers re-computed



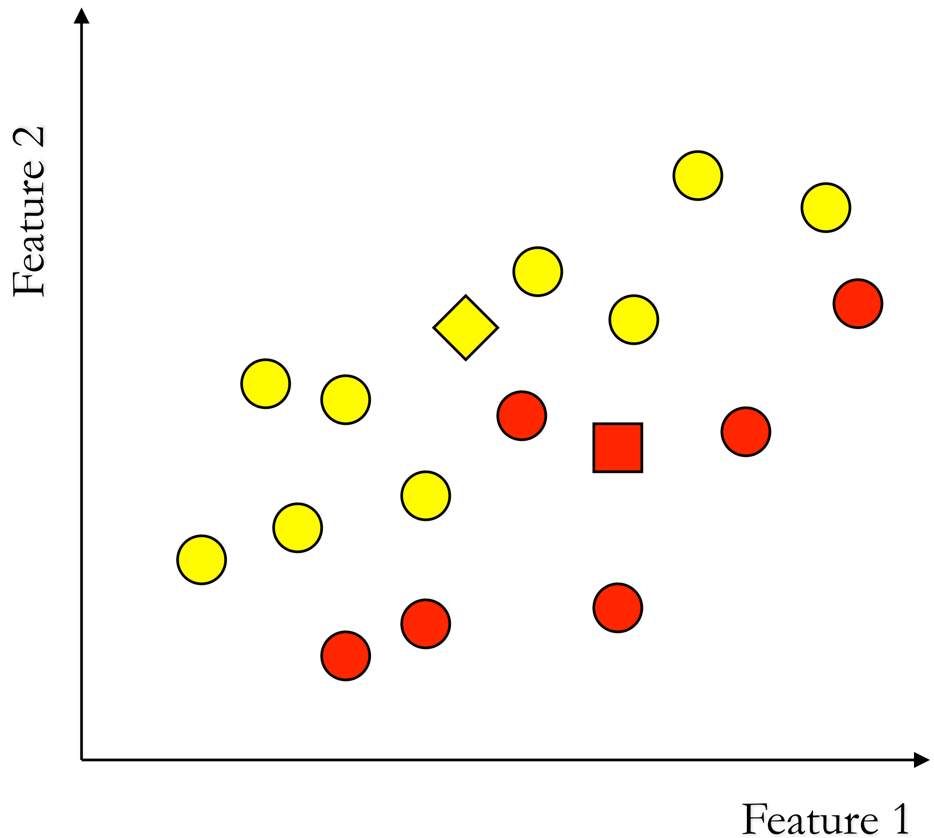
# *k*-means Clustering

- Reset clusters



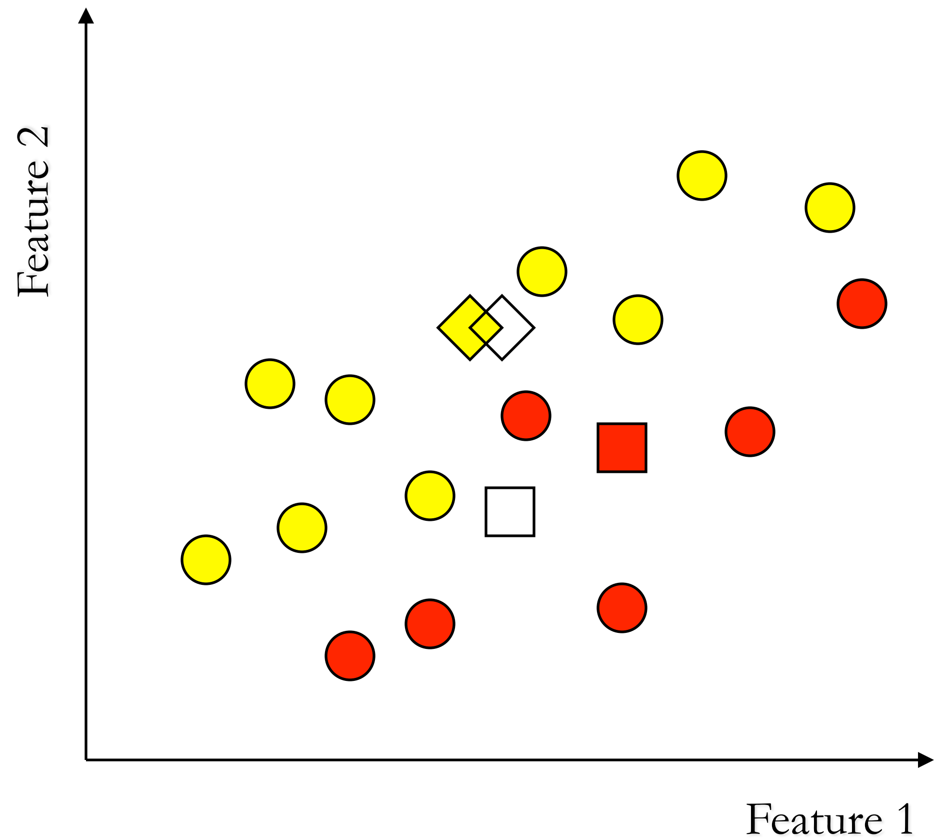
# *k*-means Clustering

- Re-classify samples according to the nearest cluster center

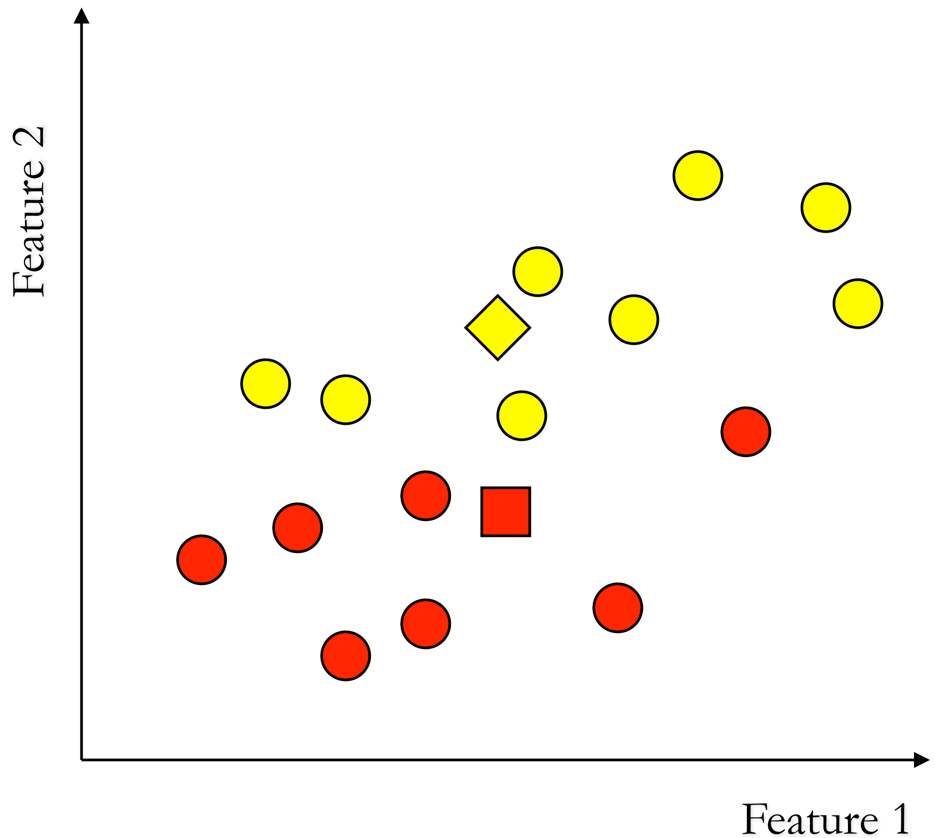


# *k*-means Clustering

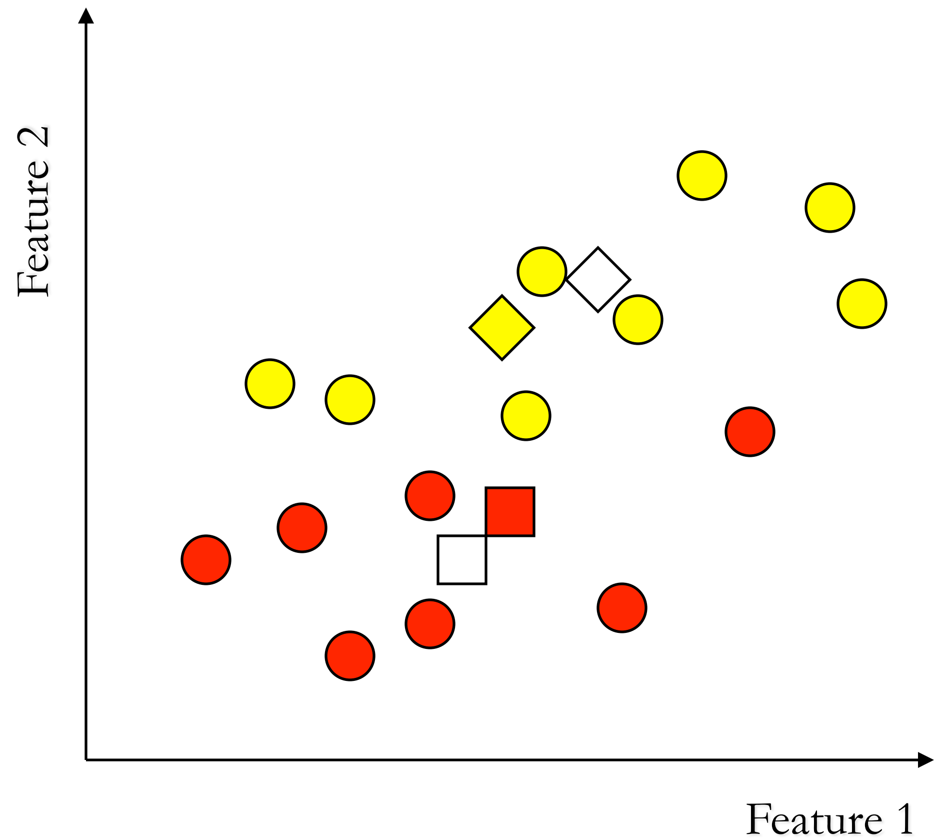
- Re-compute cluster centers



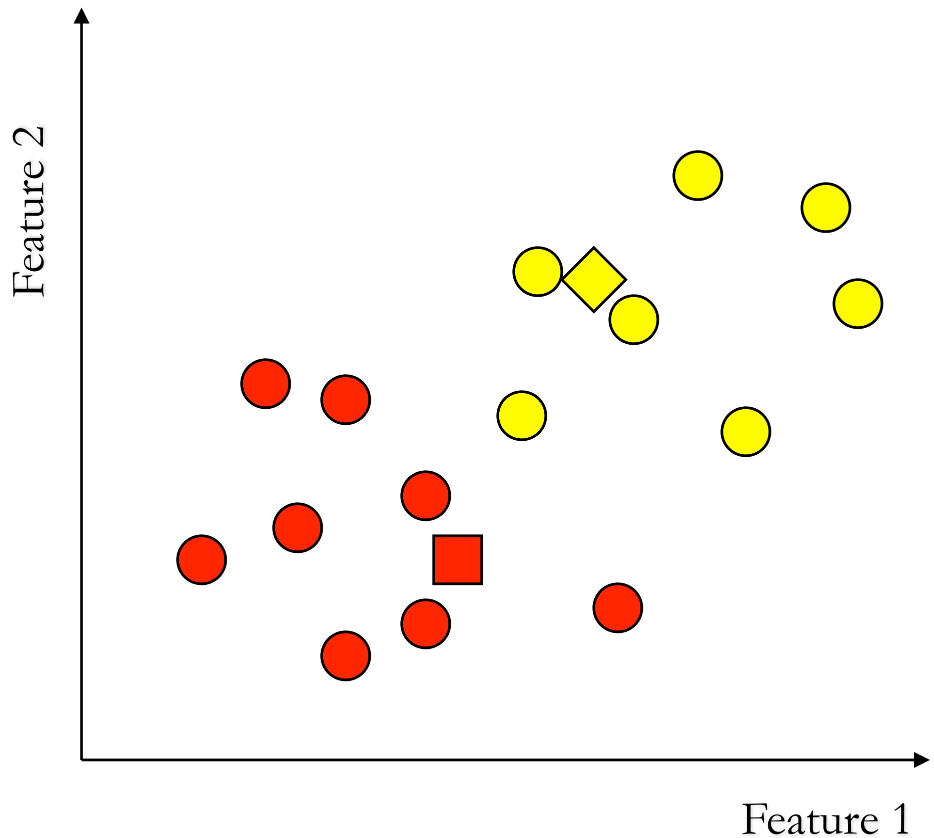
# *k*-means Clustering



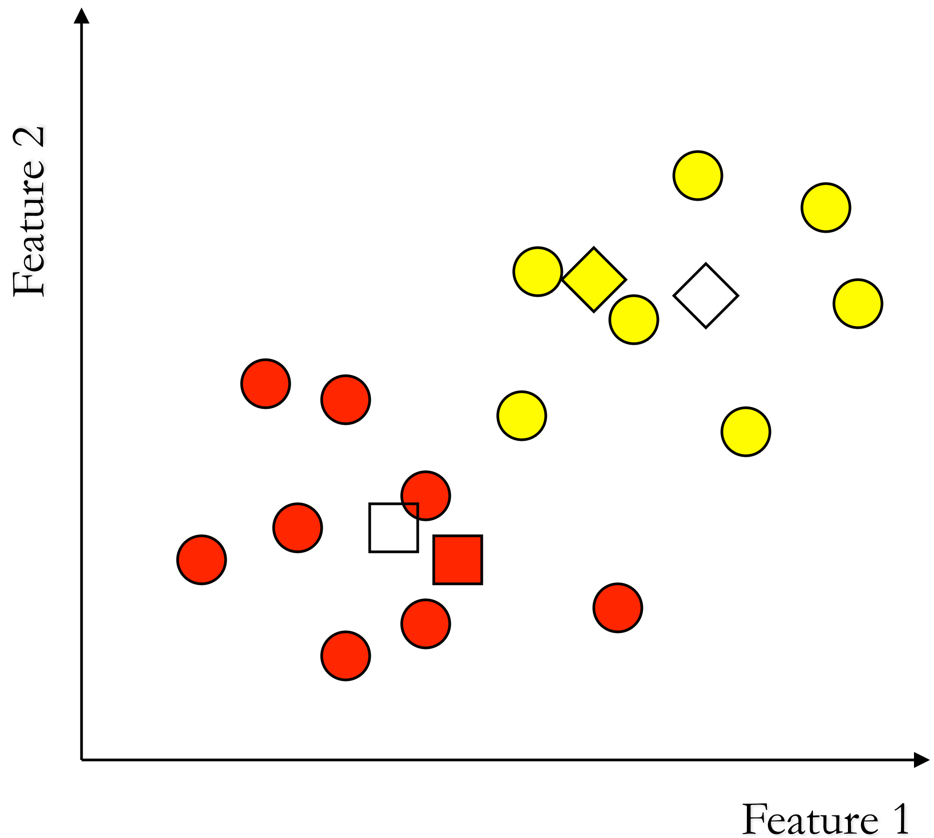
# *k*-means Clustering



# *k*-means Clustering

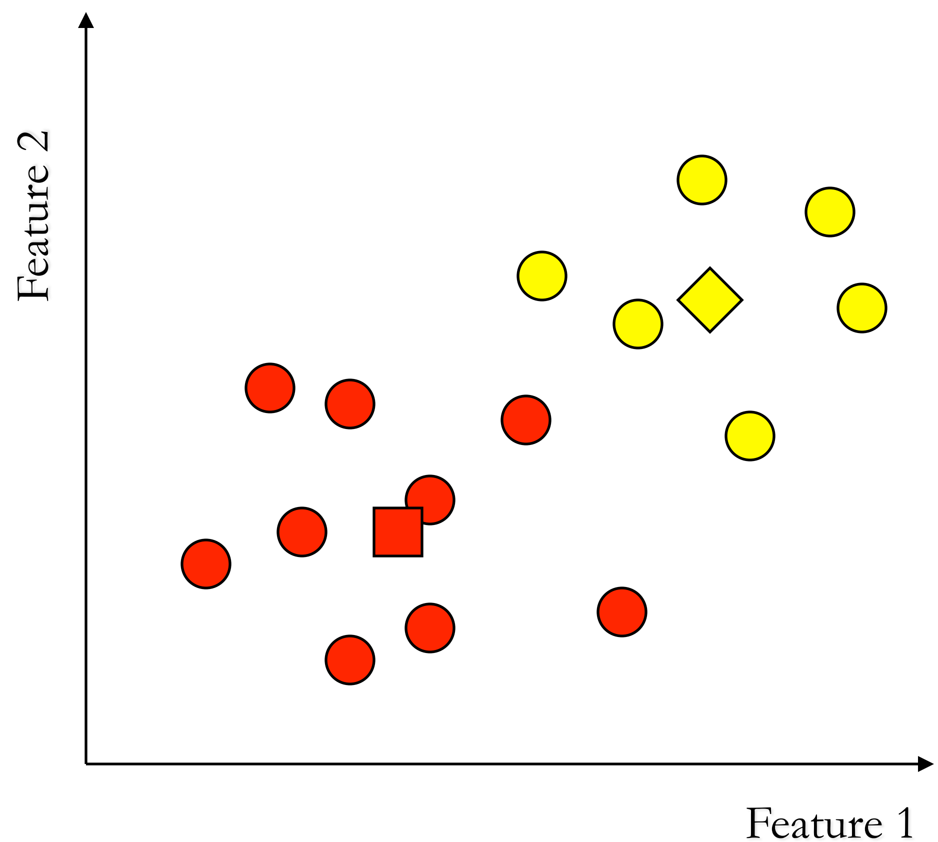


# *k*-means Clustering



# *k*-means Clustering

- Loop until no changes in cluster centers



# Results using *k*-means clustering

- Pleiades (Harder *et al.*, 2011)
  - *k*-means Clustering
  - Uses an approximation (Gaussian integral) of RMSD instead of computing the actual RMSD

- Results obtained by Pleiades
  - *k*-means performed better than ROSETTA's clustering
  - Using RMSD results in slower computation, but resulted in better final decoys than when using Gaussian integral

# The Onion method

- Onion (Li *et al.*, 2011)
  - Similar to the aim of $k$-means, the objective is: Given $n$ decoys $S_1, S_2, \ldots, S_n$, to cluster the decoys into $k$ sets, $\mathbf{A} = \{A_1, A_2, \ldots, A_k\}$, to minimize

$$\arg\min_{\mathrm{A}} \sum_{i=1}^{k} \sum_{S_j \in A_i} \mathrm{RMSD}(S_j, \mu_i)$$

where $\mu_i$ is the centroid of the set of decoys $A_i$

Recall that the aim of $k$-means was to minimize $\arg\min_{\mathrm{A}} \sum_{i=1}^{k} \sum_{S_j \in A_i} \left\| S_j - \mu_i \right\|^2$

# Onion: The Algorithm

Input: Protein structures $P_1, P_2, \ldots, P_n$, and approximation factors $\eta$, $\varepsilon$.

Output: Representative structure $O$ of approximation by $\eta$, $\varepsilon$

**For** $i \leftarrow 1 \ldots k$ **do**

Randomly pick $\eta$ structures $P_{i_1}, P_{i_2}, \ldots, P_{i_\eta}$

Superimpose $P_{i_1}, P_{i_2}, \ldots, P_{i_\eta}$ to $P_{i_1}$

Create the rotation space for each structure $P_{i_2}, \ldots, P_{i_\eta}$

**For** every $\eta$-1 rotations $R_2, \ldots, R_\eta$ from the respective rotation space **do**

Let $O = (P_1 + R_2 P_2 + \ldots + R_\eta P_\eta) / \eta$ *(That is, the average structure)*

For each input structure $P_1, P_2, \ldots, P_n$, find the optimal rigid transformation $R_i'$ that minimizes $\| O - R_i' P_i \|^2$

Compute $c(O) = \sum_{1 \le i \le n} \| O - R_i' P_i \|$

Output $O$ (and the corresponding $R_i'$) which minimizes $c(O)$

# Onion: Results vs SPICKER/Calibur

- Clustering quality
  - Decoys obtained are comparable, if not better than SPICKER
- Speed
  - Faster than Calibur

| | | CPU Time | |
|---|---|---|---|
| **Target** | **Size** | **Calibur** | **Onion** |
| 1ah9_ | 27498 | 1125.38 | 166.82 |
| 1aoy_ | 32000 | 3144.66 | 194.16 |
| 1cy5A | 32000 | 3585.62 | 189.07 |
| 1gpt_ | 32000 | 1384.36 | 171.76 |
| 1tfi_ | 32000 | 2111.49 | 303.12 |
| 1thx_ | 32000 | 3939.86 | 268.47 |
| 2a0b_ | 32000 | 3804.93 | 53.19 |

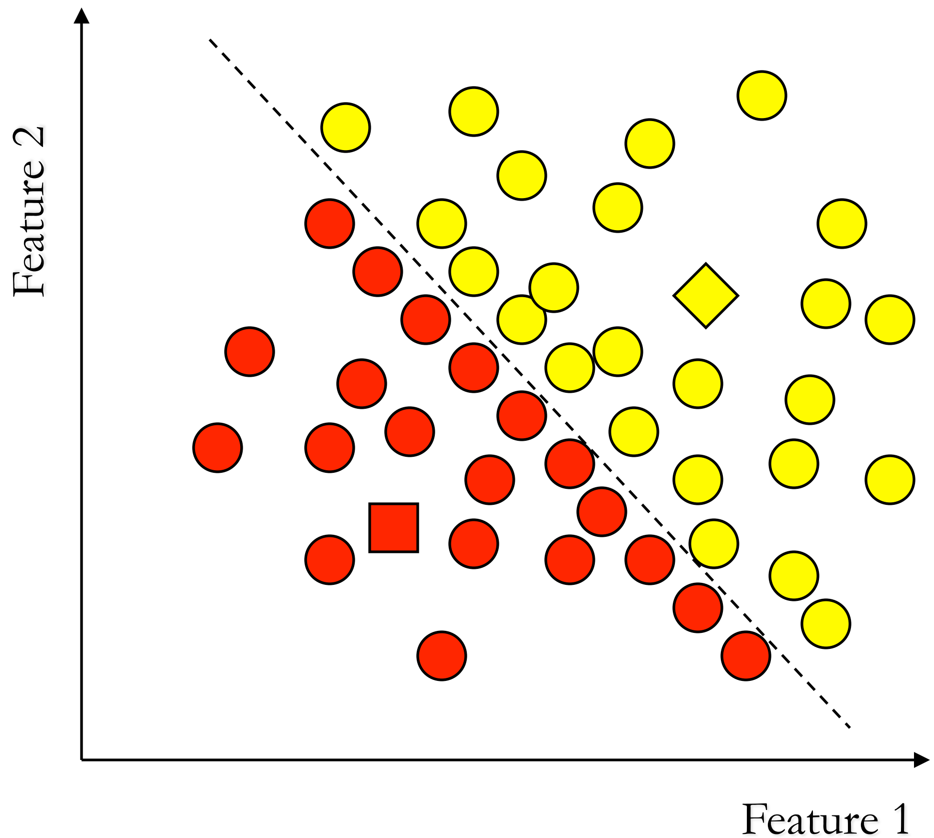
# Comparing Onion to Pleiades

- Both Onion and Pleiades are based on minimizing the “sum-of-square error”
  - No experimental results comparing both methods yet (research separately performed around the same time)
- Theoretically, Onion is better than Pleiades in the sense that
  - Pleiades uses $k$-means, which is a heuristic method in minimizing the sum-of-square error
  - Onion uses a polynomial time approximation scheme
  - That is, Onion offers guarantee in its
    - Runtime
    - Deviation from the optimal solution

# Where do we go from here?

# *k*-means Clustering

- An equidistant line from both centers can be drawn
- For more than 2 clusters, imagine a Voronoi diagram
- Such a clustering is based on the proximity to the centroids
- It may be possible to consider information beyond just "proximity"



# Thanks


*That's all, folks!*