Lecture 14: Empirical Risk Minimization & Clustering

Winter 2018

Kai-Wei Chang

CS @ UCLA

kw+cm146@kwchang.net

The instructor gratefully acknowledges Dan Roth, Vivek Srikuar, Sriram Sankararaman, Fei Sha, Ameet Talwalkar, Eric Eaton, and Jessica Wu whose slides are heavily used, and the many others who made their course material freely available online.

Recap:
Two key ideas to solve multiclass

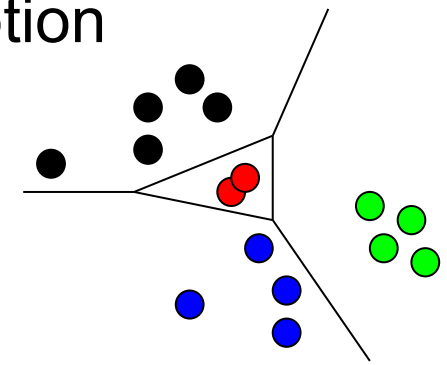
- Reducing multiclass to binary
  - Decompose the multiclass prediction into multiple binary decisions
  - Make final decision based on multiple binary classifiers
- Training a single classifier
  - Minimize the empirical risk
  - Consider all classes simultaneously

One against all strategy

One Versus one

Recap: Comparisons

- One against all
  - O(K) weight vectors to train and store
  - Training set of the binary classifiers may unbalanced
  - Less expressive; make a strong assumption



- One v.s. One (All v.s. All)
  - O(K^2) weight vectors to train and store
  - Size of training set for a pair of labels could be small
    ⇒ overfitting of the binary classifiers
  - Need large space to store model

Problems with Decompositions

- Learning optimizes over *local* metrics
  - Does not guarantee good *global* performance
  - We don't care about the performance of the *local* classifiers
- Poor decomposition $\Rightarrow$ poor performance
  - Difficult local problems
  - Irrelevant local problems
- Efficiency: e.g., All vs. All vs. One vs. All
- Not clear how to generalize multi-class to problems with a very large # of output

Decomposition methods: Summary

- General Ideas:
  - Decompose the multiclass problem into many binary problems
  - Prediction depends on the decomposition
    - Constructs the multiclass label from the output of the binary classifiers
- Learning optimizes local correctness
  - Each binary classifier don't need to be globally correct and isn't aware of the prediction procedure

This Lecture

- Multiclass classification overview
- Reducing multiclass to binary
  - One-against-all & One-vs-one
- Training a single classifier
  - Multiclass Perceptron

Revisit One-again-All learning algorithm

- Learning: Given a dataset $D = \{(x_i, y_i)\}$
  $x_i \in R^n, y_i \in \{1,2,3, \ldots K\}$
- Decompose into K binary classification tasks
  - Learn K models: $w_1, w_2, w_3, \ldots w_K$
  - w_k: separate class k from others
- Prediction

$$\hat{y} = \text{argmax}_{y \in \{1,2,\ldots K\}} \, w_y^T \, x$$

Observation

- At training time, we require $w_i^T x$ to be positive for examples of class i.
- Really, all we need is for $w_i^T x$ to be more than all others ⇒ this is a weaker requirement

For examples with label i, we need

$$w_i^T x > w_j^T x \qquad \text{for all } j$$

Perceptron-style algorithm

For examples with label i, we need

$$w_i^T x > w_j^T x \quad \text{for all } j$$

- For each training example (x, y)
  - If for some y', $w_y^T x \leq w_{y'}^T x$ mistake!
    - $w_y \leftarrow w_y + \eta x$ update to promote y
    - $w_{y'} \leftarrow w_{y'} - \eta x$ update to demote y'

Why add ηx to w_y promote label y:

Before update $s(y) = < w_y^{old}, x >$

After update

$$s(y) = < w_y^{new}, x > = < w_y^{old} + \eta x, x > = < w_y^{old}, x > + \eta < x, x >$$

Note! $< x, x > = x^T x > 0$

A Perceptron-style Algorithm

Given a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}$

Initialize $\boldsymbol{w} \leftarrow \mathbf{0} \in \mathbb{R}^n$

For epoch $1 \ldots T$:

For $(\boldsymbol{x}, y)$ in $\mathcal{D}$:

For $y' \neq y$

if $w_y^T x < w_{y'}^T x$ make a mistake

$w_y \leftarrow w_y + \eta x$ promote y

$w_{y'} \leftarrow w_{y'} - \eta x$ demote y'

Return $\boldsymbol{w}$

How to analyze this algorithm and simplify the update rules?

Prediction: $\text{argmax}_y \; w_y^T x$

Look back – Learning model

- SVM

$$\min_{w,b} \ \frac{1}{2} w^T w + C \sum_i \max(0, 1 - y_i(w^T x_i + b))$$

- Logistic regression (w/ regularization)

$$min_{w,b} \ \frac{1}{2} w^T w + C \sum_i \log(1 + \exp(-y_i(\mathbf{w}^T \mathbf{x}_i + b)))$$

- Linear Regression (w/ regularization)

$$\min_{w,b} \ \frac{1}{2} w^T w + C \sum_i \left(y_i - \ (w^T x_i + b)\right)^2$$

Learning as Loss Minimization

Learning as loss minimization

- The setup
  - Examples $\mathbf{x}$ drawn from a fixed, unknown distribution D
  - Hidden oracle classifier $f(x)$ labels examples
  - We wish to find a hypothesis $\mathrm{h} \in \mathrm{H}$ that mimics $f(x)$
- The ideal situation
  - Define a function L that penalizes bad hypotheses
  - **Learning:** Pick a function h ∈ H to minimize expected loss

$$\min_{h \in H} E_{\mathbf{x} \sim D} \left[L\left(h(\mathbf{x}), f(\mathbf{x})\right) \right]$$

But distribution D is unknown

- Instead, minimize *empirical loss* on the training set

$$\min_{h \in H} \frac{1}{m} \sum_{i} L(h(\mathbf{x}_i), f(\mathbf{x}_i))$$

Empirical loss minimization

Learning = minimize *empirical loss* on the training set

$$\min_{h \in H} \frac{1}{m} \sum_{i} L(h(\mathbf{x}_i), f(\mathbf{x}_i))$$

Is there a problem here?

Empirical loss minimization

Learning = minimize *empirical loss* on the training set

$$\min_{h \in H} \frac{1}{m} \sum_i L(h(\mathbf{x}_i), f(\mathbf{x}_i))$$

Is there a problem here? Overfitting!

We need something that biases the learner towards simpler hypotheses

- Achieved using a regularizer, which penalizes complex hypotheses

$$\min_{h \in H} R(h) + \frac{1}{m} \sum_i [L(h(x_i), f(x_i))]$$

Regularized loss minimization

- Learning: $\min_{h \in H} R(h) + \frac{1}{m} \sum_i [L(h(x_i), f(x_i))]$
- With linear classifiers:

$$H: \{h(x): sgn(w^T x \geq 0)\}, w \in R^d$$

$$\min_{\mathbf{w}} \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_i L(y_i, \mathbf{x}_i, \mathbf{w})$$

- What is a loss function?
  - Loss functions should penalize mistakes
  - We are minimizing average loss over the training data
- What is the ideal loss function for classification?

The 0-1 loss

Penalize classification mistakes between true label y and prediction y'

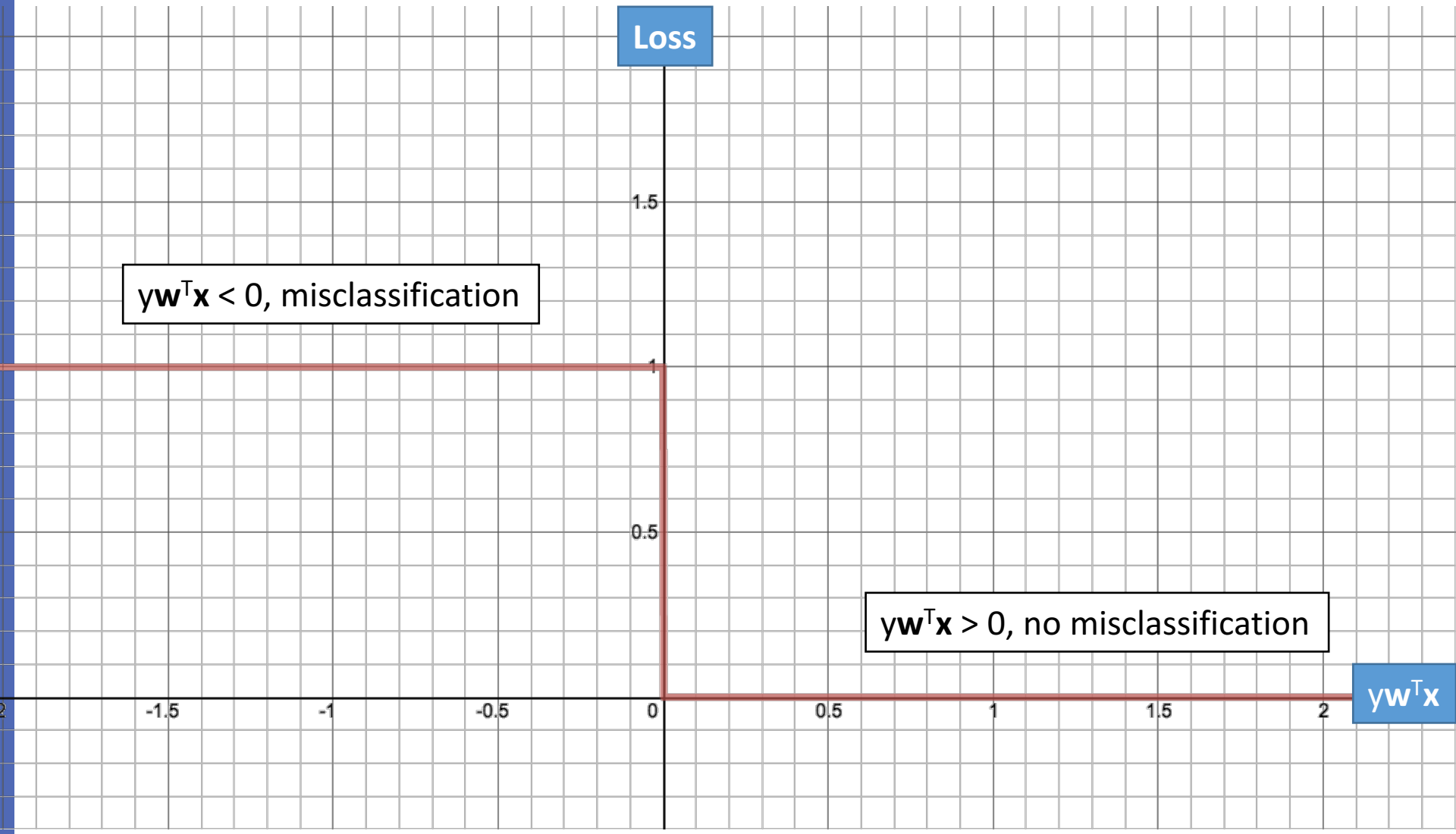
$$L_{0\text{-}1}(y, y') = \begin{cases} 1 & \text{if } y \neq y', \\ 0 & \text{if } y = y'. \end{cases}$$

- For linear classifiers, the prediction is sgn($\mathbf{w}^T\mathbf{x}$)
  - Mistake if $y\,\mathbf{w}^T\mathbf{x} \leq 0$

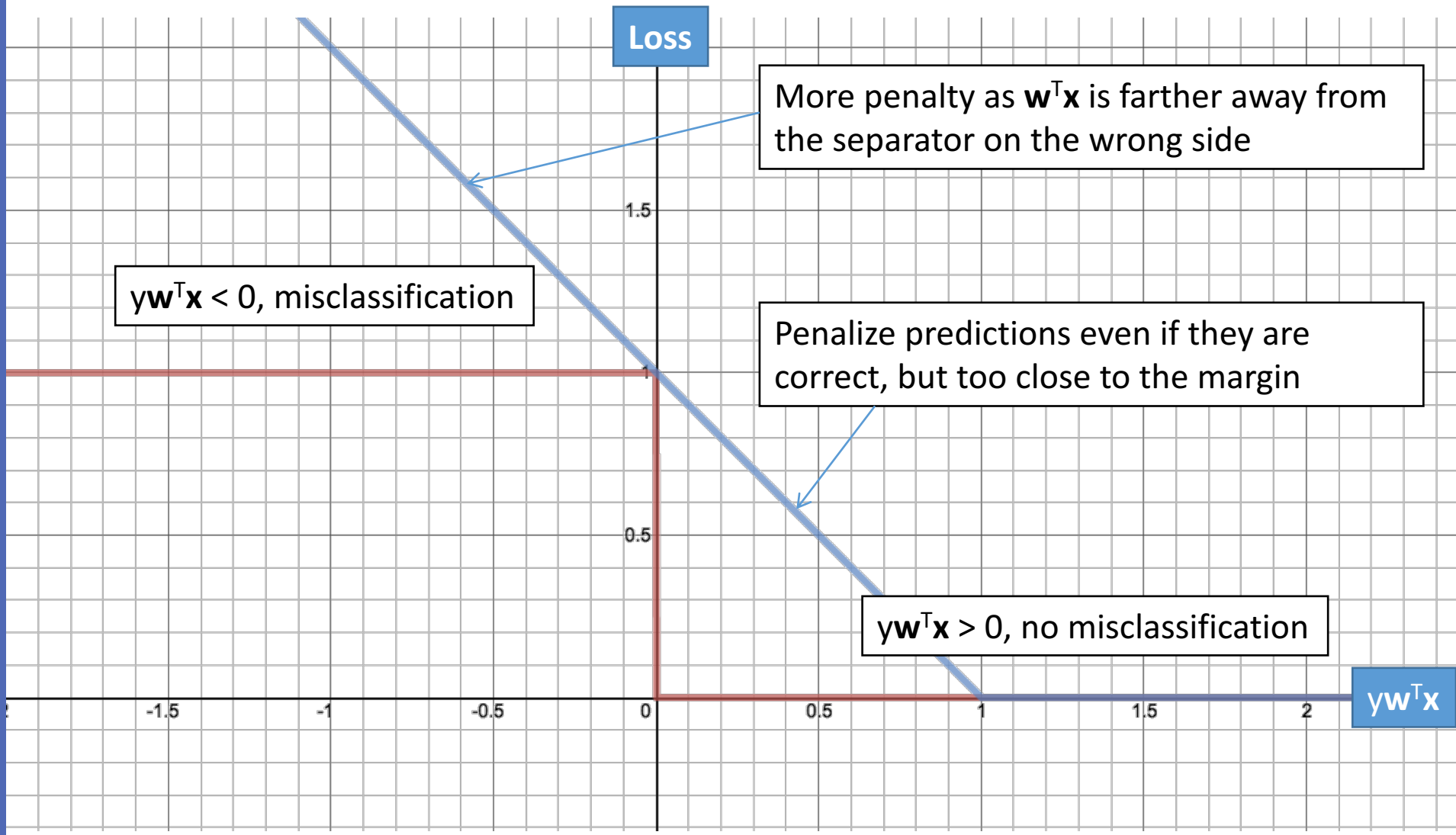
$$L_{0\text{-}1}(y, \mathbf{x}, \mathbf{w}) = \begin{cases} 1 & \text{if } y\ \mathbf{w}^T\mathbf{x} \leq 0, \\ 0 & \text{otherwise.} \end{cases}$$

Minimizing 0-1 loss is intractable. Need surrogates

The 0-1 loss


Loss
1.5
1
0.5
0
-1.5
-1
-0.5
0.5
1
1.5
2
$y\mathbf{w}^T\mathbf{x} < 0$, misclassification
$y\mathbf{w}^T\mathbf{x} > 0$, no misclassification
$y\mathbf{w}^T\mathbf{x}$

Compare to the hinge loss



Support Vector Machines

- SVM = linear classifier combined with regularization

- Ideally, we would like to minimize 0-1 loss,
  - But we can't for computational reasons

- SVM minimizes hinge loss

$$L_{Hinge}(y, \mathbf{x}, \mathbf{w}) = \max(0, 1 - y\mathbf{w}^T\mathbf{x})$$

  - Variants exist

$$\min_{h \in H} \text{regularizer}(h) + C \frac{1}{m} \sum_i L(h(\mathbf{x}_i), f(\mathbf{x}_i))$$

The loss function zoo

Many loss functions exist

- Perceptron loss

$$L_{Perceptron}(y, \mathbf{x}, \mathbf{w}) = \max(0, -y\mathbf{w}^T\mathbf{x})$$

- Hinge loss (SVM)

$$L_{Hinge}(y, \mathbf{x}, \mathbf{w}) = \max(0, 1 - y\mathbf{w}^T\mathbf{x})$$

- Exponential loss (AdaBoost)

$$L_{Exponential}(y, \mathbf{x}, \mathbf{w}) = e^{-y\mathbf{w}^T\mathbf{x}}$$

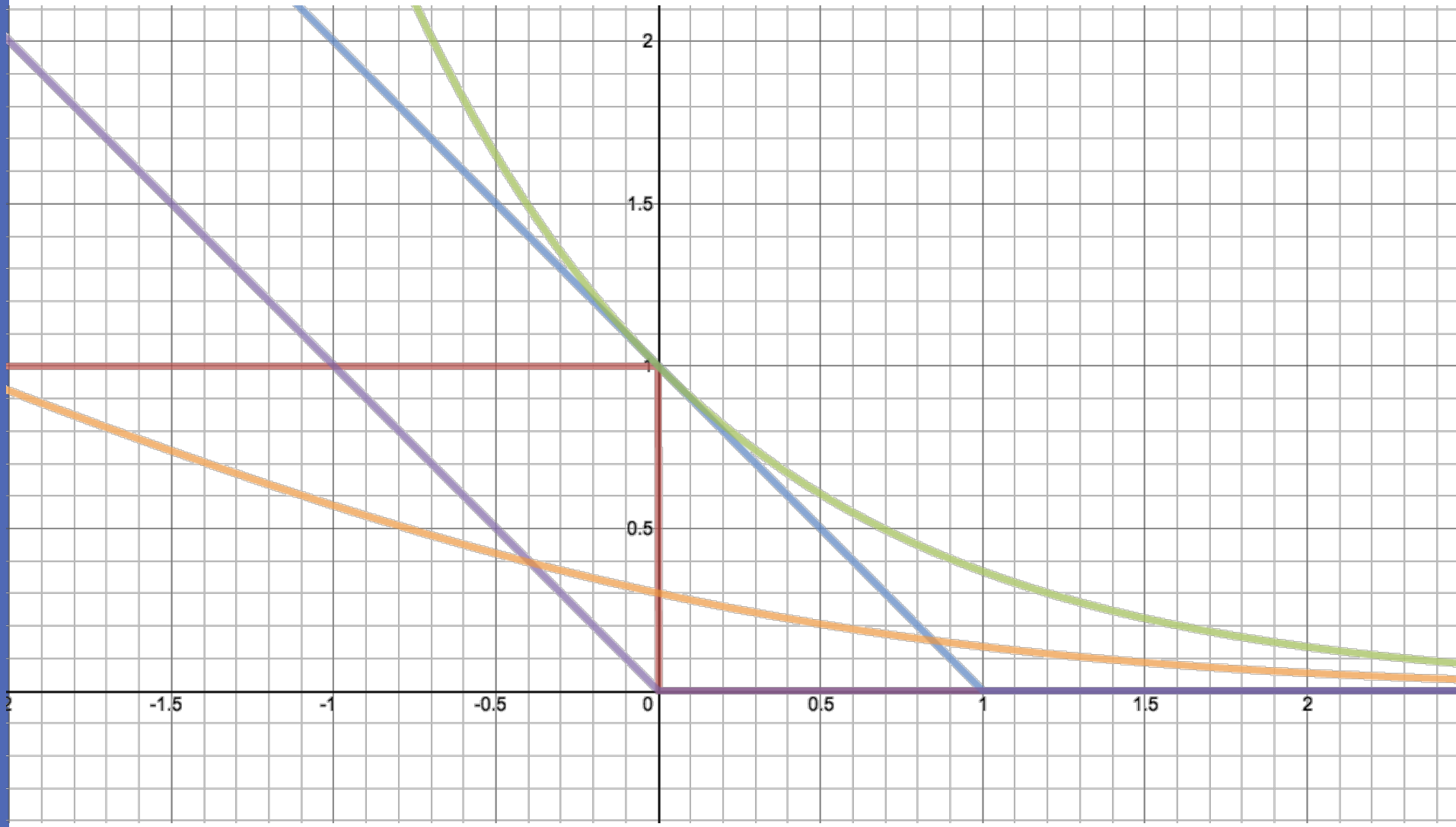
- Logistic loss (logistic regression)

$$L_{Logistic}(y, \mathbf{x}, \mathbf{w}) = \log(1 + e^{-y\mathbf{w}^T\mathbf{x}})$$

http://www.cs.toronto.edu/~mbrubake/teaching/C11/Handouts/AdaBoost.pdf

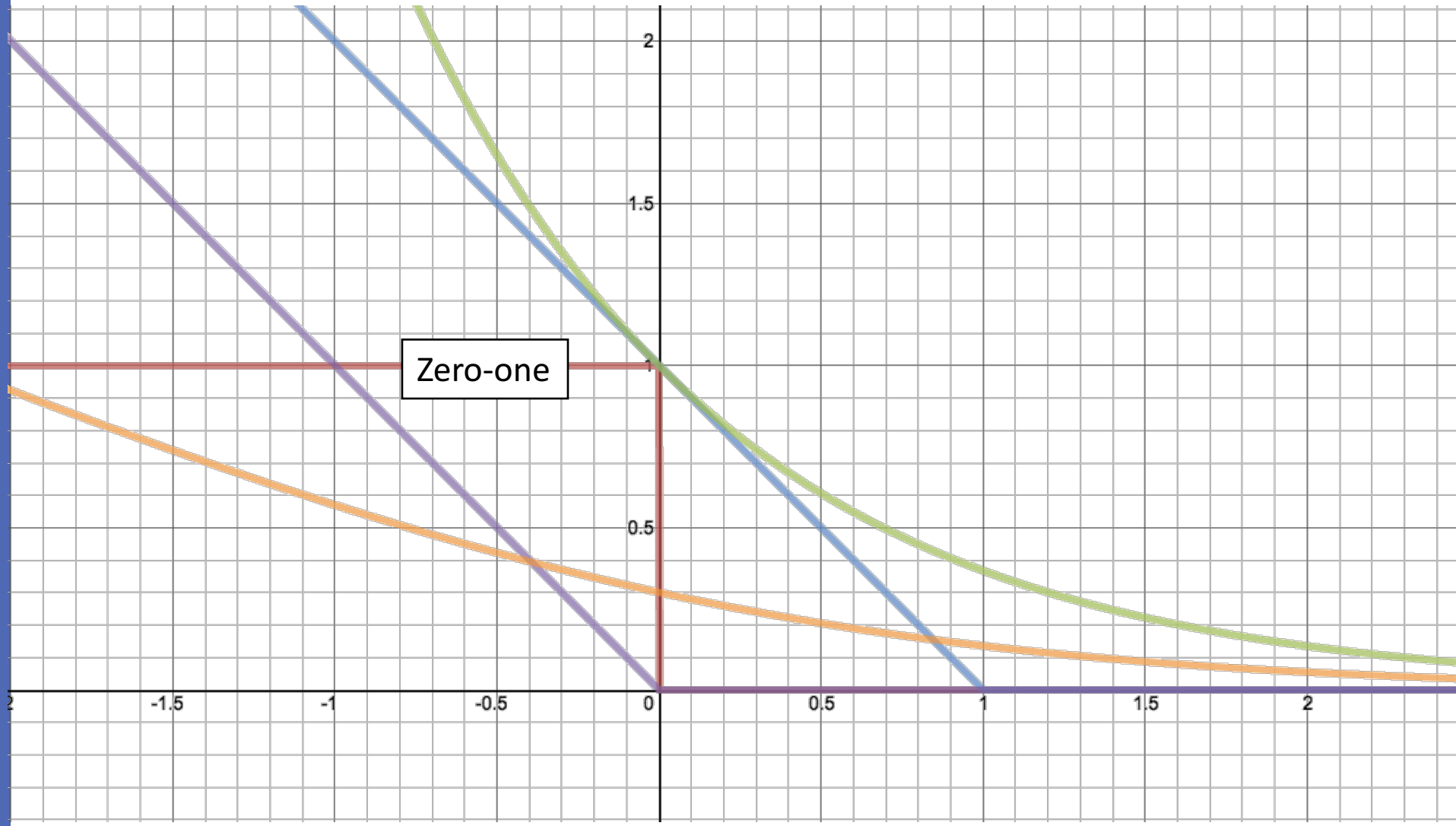
$$\min_{h \in H} \text{regularizer}(h) + C \frac{1}{m} \sum_i L(h(\mathbf{x}_i), f(\mathbf{x}_i))$$

The loss function zoo



$$\min_{h \in H} \text{regularizer}(h) + C \frac{1}{m} \sum_i L(h(\mathbf{x}_i), f(\mathbf{x}_i))$$

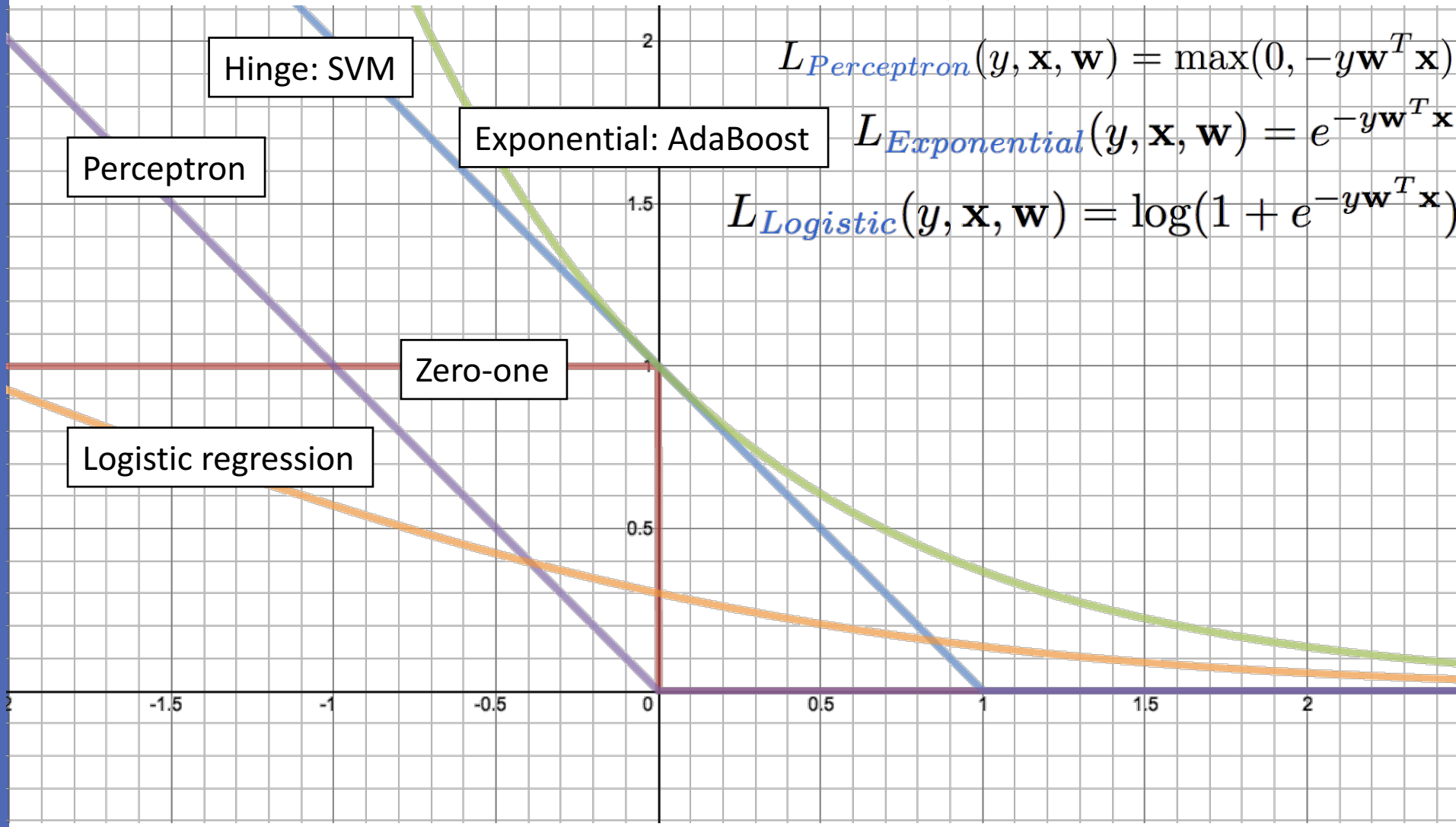
The loss function zoo



$$\min_{h \in H} \text{regularizer}(h) + C \frac{1}{m} \sum_i L(h(\mathbf{x}_i), f(\mathbf{x}_i))$$

The loss function zoo

$$L_{Hinge}(y, \mathbf{x}, \mathbf{w}) = \max(0, 1 - y\mathbf{w}^T\mathbf{x})$$



$$L_{Perceptron}(y, \mathbf{x}, \mathbf{w}) = \max(0, -y\mathbf{w}^T\mathbf{x})$$

$$L_{Exponential}(y, \mathbf{x}, \mathbf{w}) = e^{-y\mathbf{w}^T\mathbf{x}}$$

$$L_{Logistic}(y, \mathbf{x}, \mathbf{w}) = \log(1 + e^{-y\mathbf{w}^T\mathbf{x}})$$

Approximate 0-1 loss

- Usually, we cannot minimize 0-1 loss
  - It is a combinatorial optimization problem: NP-hard
- Idea: minimizing its upper-bound

Many choices of R(w)

- Minimizing the empirical loss with linear function

$$\min_{w \in R^d} \; R(w) \; + \; \frac{1}{\left|\widehat{D}\right|} \sum_{(x,y)\in \widehat{D}} [L(x, w, y)]$$

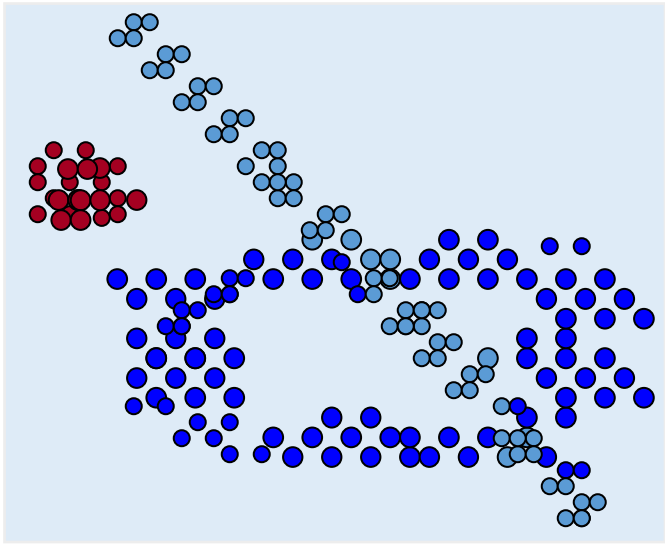
- Prefer simpler model: (how?)
  - Sparse:
    $R(w)$ = #non-zero elements in w (L0 regularizer)
    $R(w) = \sum_i |w_i|$ (L1 regularizer)
  - Gaussian prior (large margin w/ hinge loss):
    $R(w) = \sum_i w_i^2 = w^T w$ (L2 regularizer)

Learning via Loss Minimization: Summary

- Learning via Loss Minimization
  - Write down a loss function
  - Minimize empirical loss
- Regularize to avoid overfitting
  - Neural networks use other strategies such as dropout
- Widely applicable, different loss functions and regularizers

Clustering

How many clusters are there?



Goal of Clustering

- Given a collection of data points, the goal is to find structure in the data: organize that data into sensible groups.

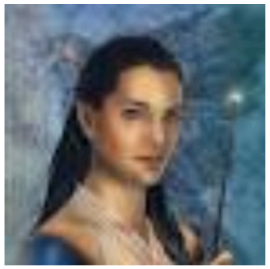
- Applications
  - Topics in news articles
  - Identify communities within social networks

Application in Hogwarts (Harry potter)

- Sorting Hat – cluster students into four groups based on four underlying prototypes

Godric Gryffindor

Helga Hufflepuff



Rowena Ravenclaw

Salazar Slytherin

How many "kinds of monsters" are there? How to define clusters?

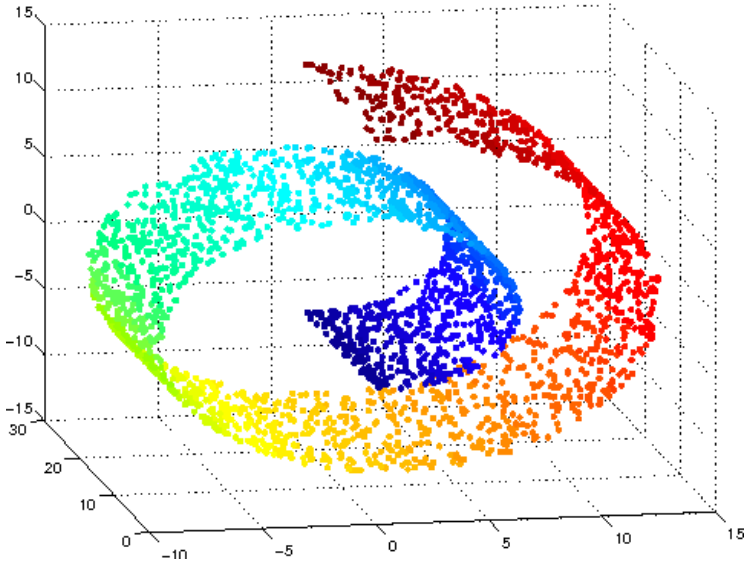


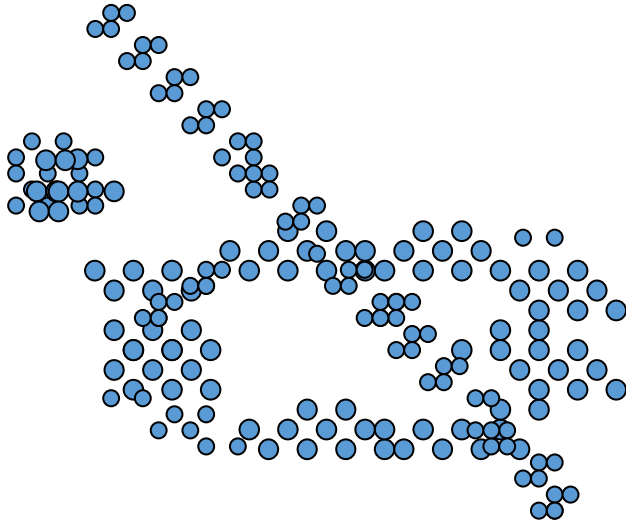





How to define clusters

- A set of entities which are “alike”
- May be described as connected regions of a multi-dimensional space
- ”We recognize a cluster when we see it”





Clustering

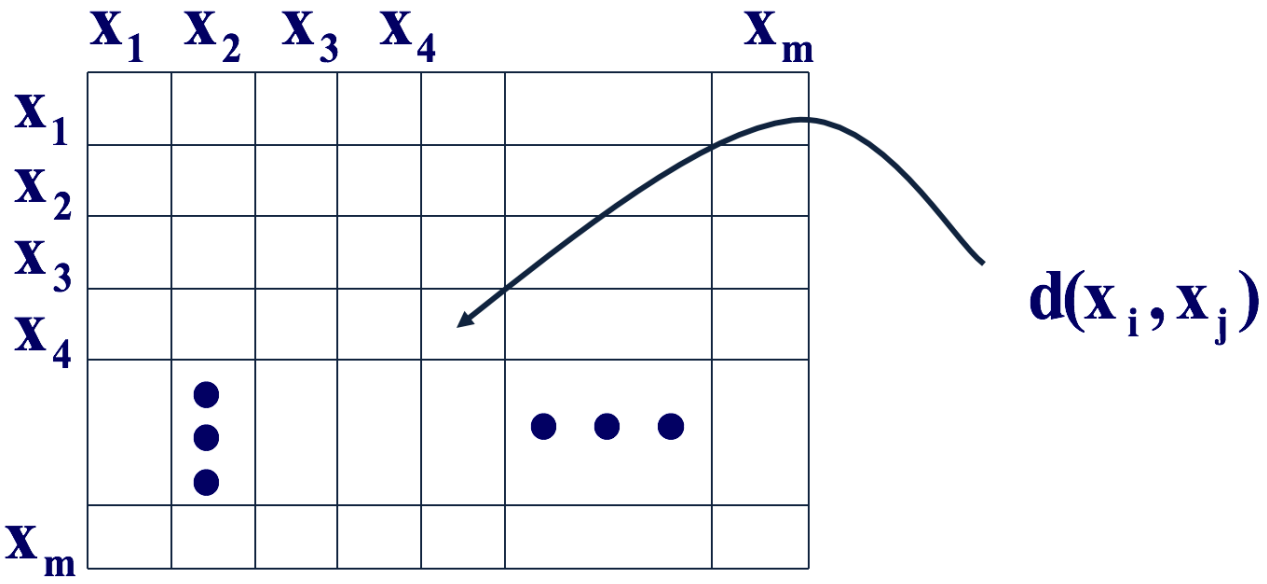
- An optimization problem:
  - Given a set of points and a pairwise distance, devise an algorithm f that splits the data so that it optimizes some conditions.

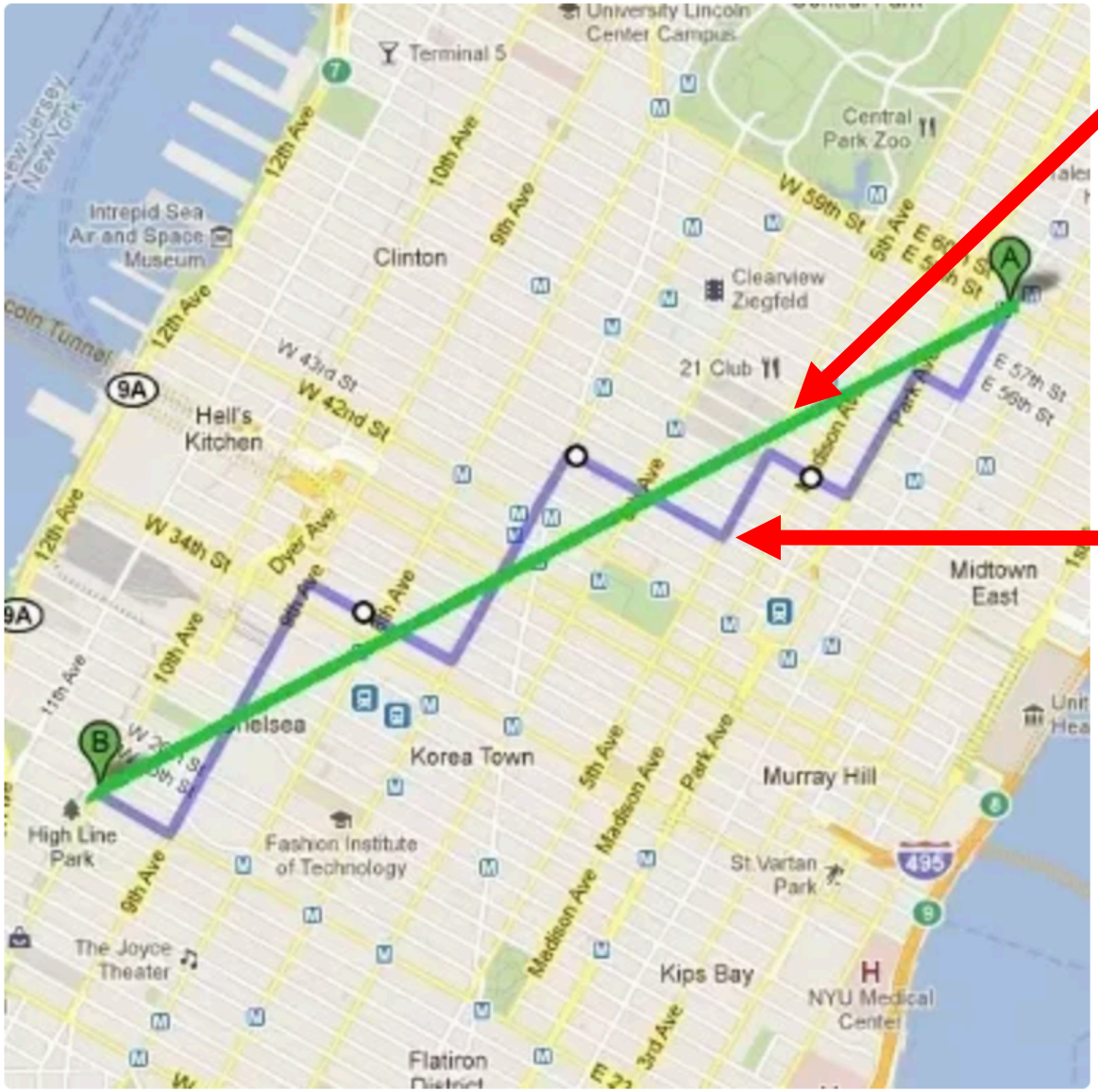
Setup Given $\mathcal{D} = \{\boldsymbol{x}_n\}_{n=1}^N$ and K, we want to output

- $\{\boldsymbol{\mu}_k\}_{k=1}^K$: prototypes (or centroids) of clusters
- $A(\boldsymbol{x}_n) \in \{1, 2, \ldots, K\}$: the cluster membership, i.e., the cluster ID assigned to $\boldsymbol{x}_n$

Pairwise distance

- We assume the pairwise distances are given
  - We assume assume that the input to the problem is a matrix of distances between all pairs




Euclidean distance
Manhattan distance
Terminal 5
Central Park Zoo
Intrepid Sea Air and Space Museum
Clinton
Clearview Ziegfeld
21 Club
Hell's Kitchen
Midtown East
Chelsea
Korea Town
Murray Hill
High Line Park
Fashion Institute of Technology
St Vartan Park
The Joyce Theater
Kips Bay
NYU Medical Center
Flatiron
12th Ave
10th Ave
9th Ave
8th Ave
6th Ave
5th Ave
Madison Ave
Park Ave
W 59th St
W 43rd St
W 42nd St
W 34th St
E 57th St
E 59th St
Dyer Ave
9A
495

Recap: Distance between instances

Numeric features, represented as n dimensional vectors

- Euclidean distance

$$||\mathbf{x}_1 - \mathbf{x}_2||_2 = \sqrt{\sum_{i=1}^{n} (\mathbf{x}_{1,i} - \mathbf{x}_{2,i})^2}$$

- Manhattan distance

$$||\mathbf{x}_1 - \mathbf{x}_2||_1 = \sum_{i=1}^{n} |\mathbf{x}_{1,i} - \mathbf{x}_{2,i}|$$

- L_p-norm
  - Euclidean = L_2
  - Manhattan = L_1

$$||\mathbf{x}_1 - \mathbf{x}_2||_p = \left(\sum_{i=1}^{n} |\mathbf{x}_{1,i} - \mathbf{x}_{2,i}|^p \right)^{\frac{1}{p}}$$

P > 0

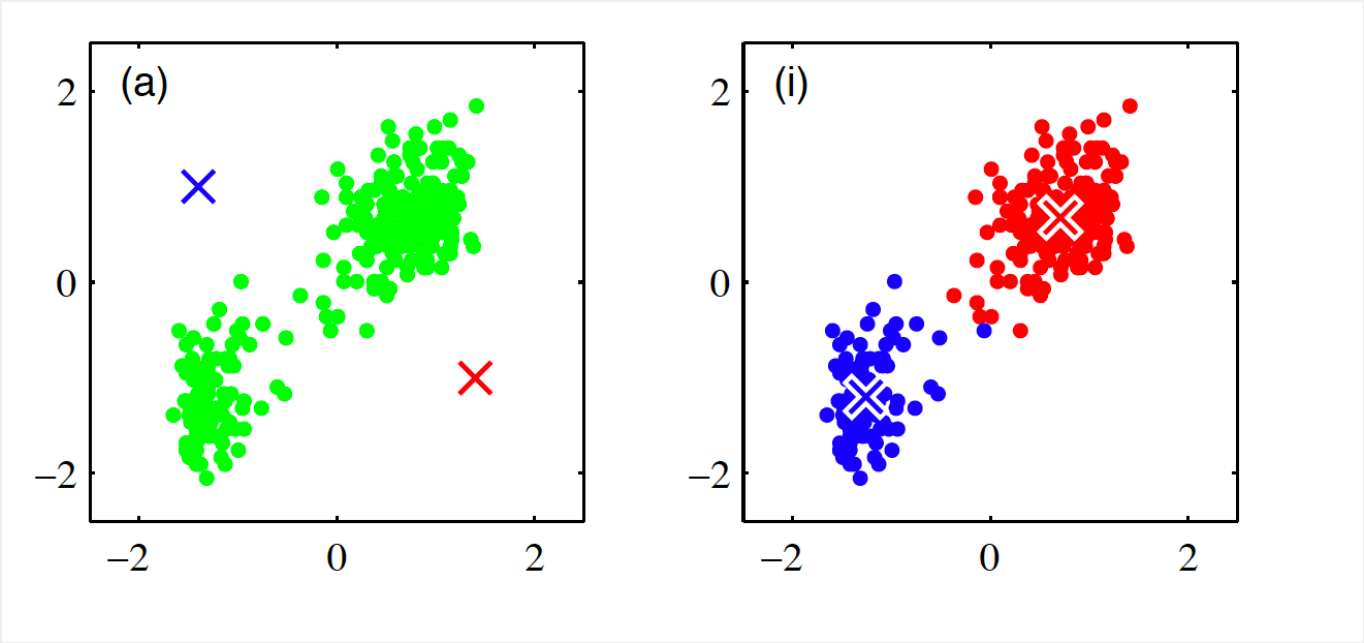
Toy Example – cluster them to two group



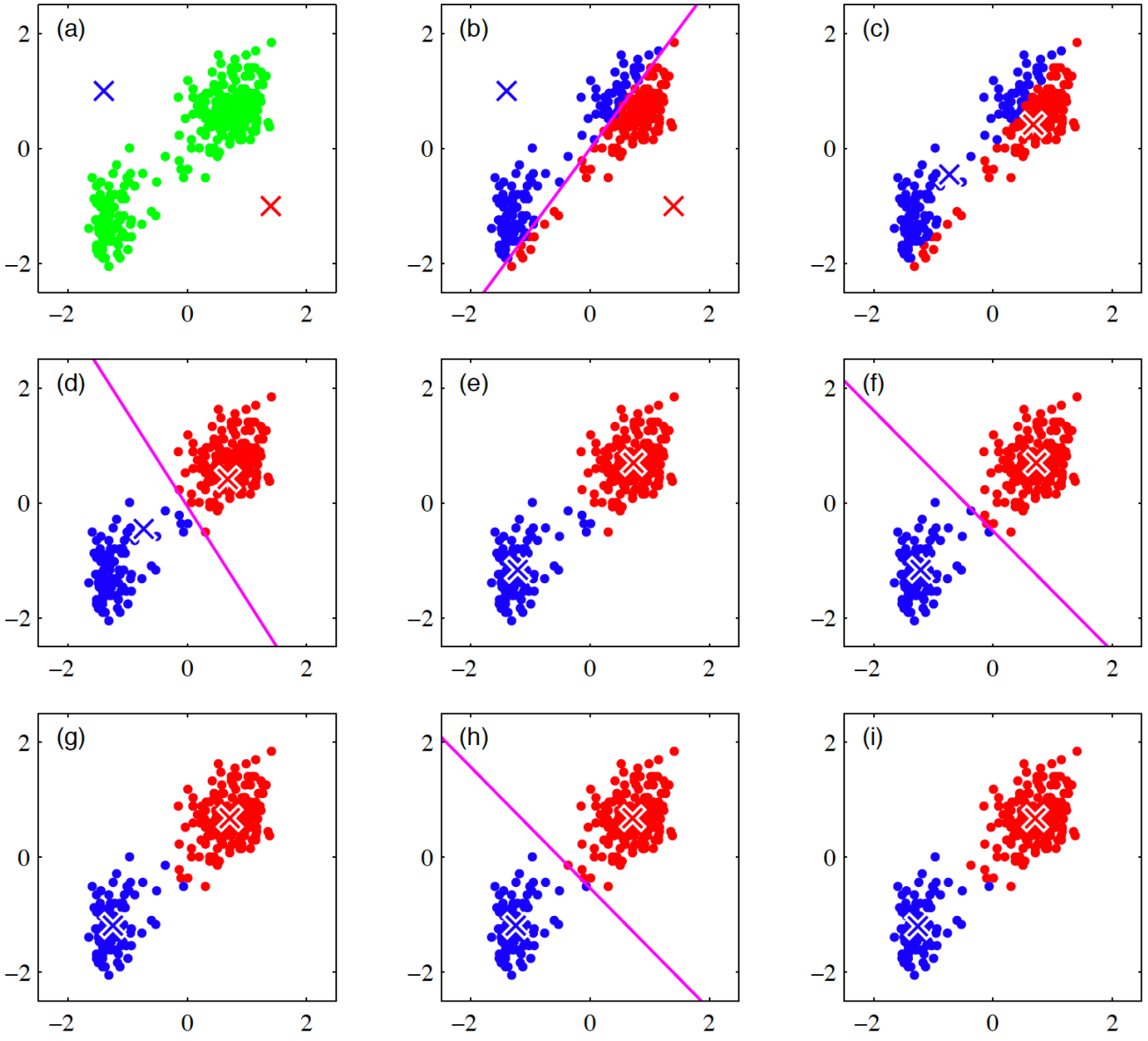




Toy Example – cluster them to two group



Intuition of K-Means



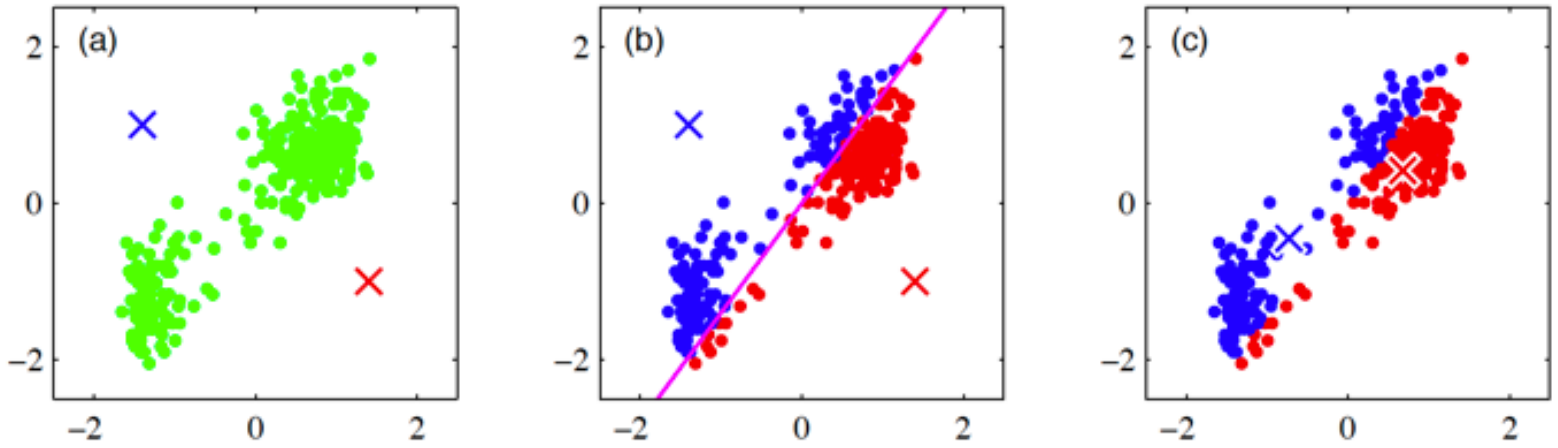
Intuition@ Hogwarts

- Sorting Hat – cluster students into four groups based on four underlying prototypes

- The prototype of each house is the average of all students of the house
  - Alternatively, updating the prototype & the student assignment

K-means clustering

- Intuition: Data points assigned to cluster k should be close to μ_k the prototype



K-means clustering

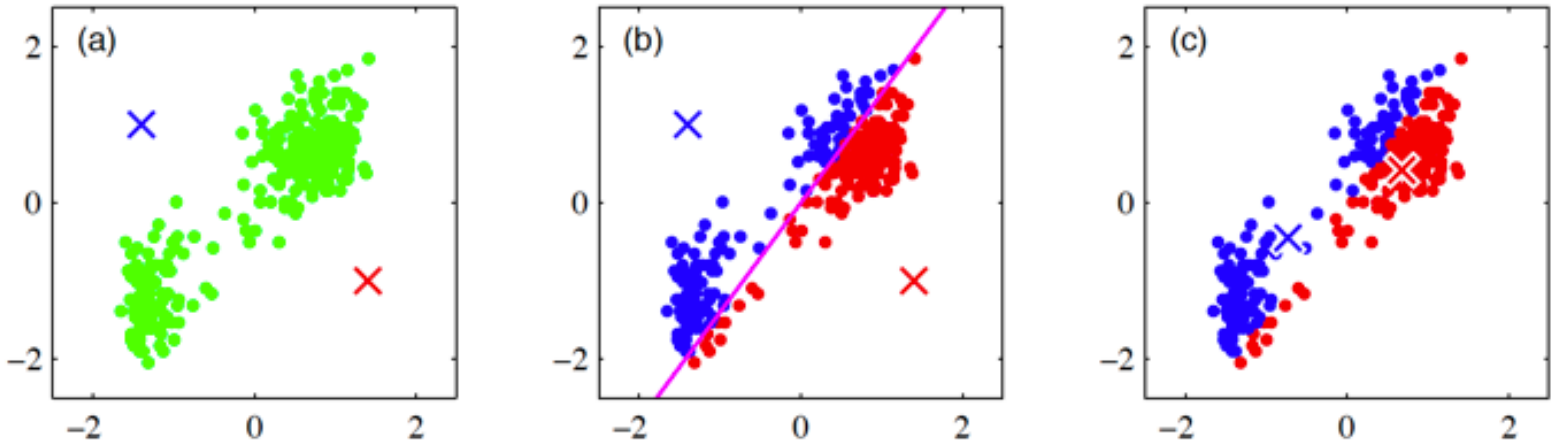
- Distortion measure
  (cost function for clustering)

Sum of distances of all the points to their cluster center

$$J(\{r_{nk}\}, \{\boldsymbol{\mu}_k\}) = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|_2^2$$

where $r_{nk} \in \{0, 1\}$ is an indicator variable

$$r_{nk} = 1 \quad \text{if and only if} \quad A(\boldsymbol{x}_n) = k$$



K-means objective

$$argmin_{\{r_{nk}\},\{\boldsymbol{\mu}_k\}} J(\{r_{nk}\},\{\boldsymbol{\mu}_k\}) = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}\|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|_2^2$$

where $r_{nk} \in \{0,1\}$ is an indicator variable

$$r_{nk} = 1 \quad \text{if and only if} \quad A(\boldsymbol{x}_n) = k$$

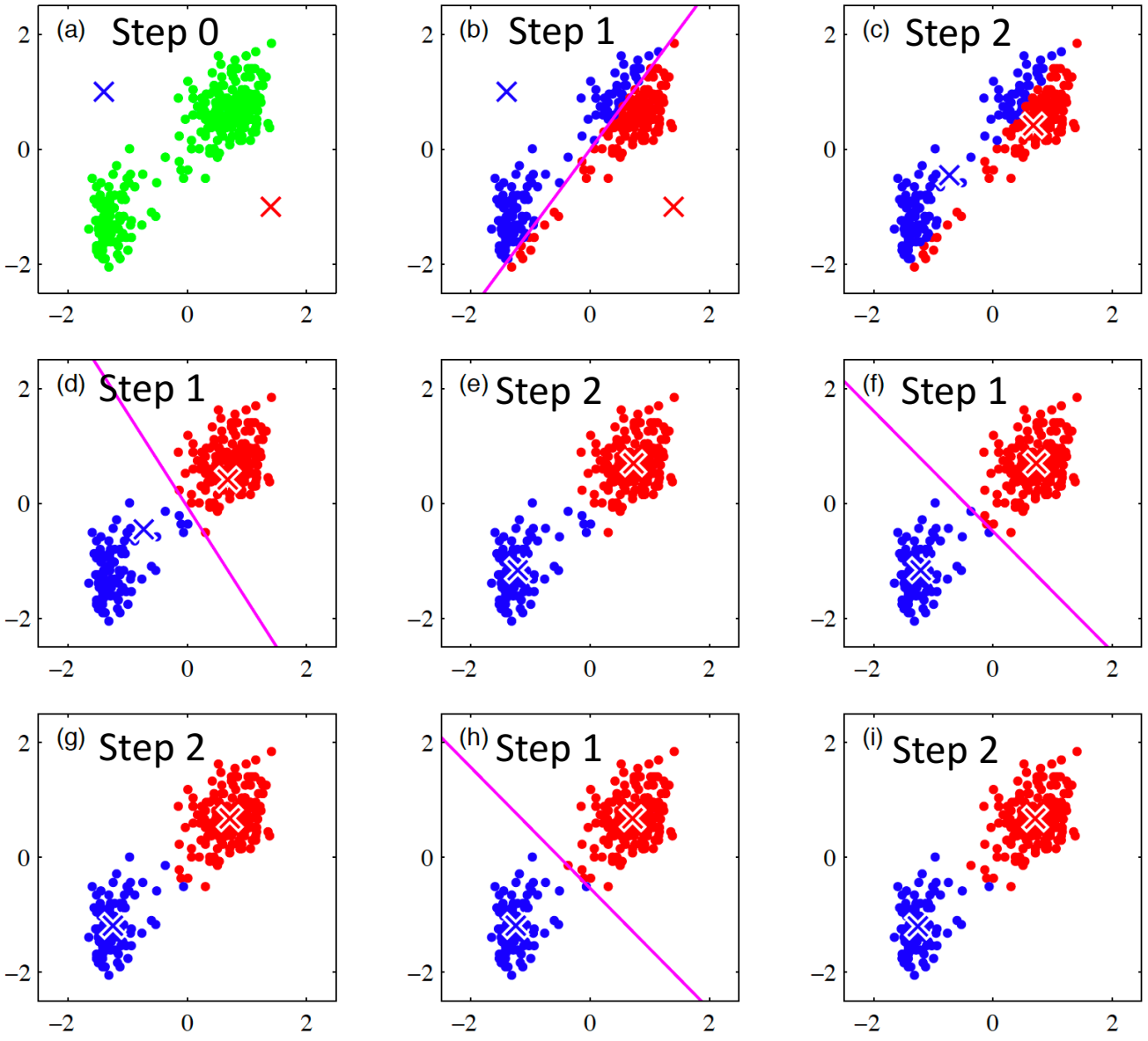
- It is a non-convex objective function
- Minimizing the above objective is NP-hard.

K-means algorithm
a.k.a Llyod's algorithm

- A greedy algorithm for minimizing K-means objective – alternative update $\{r_{nk}\}, \{\boldsymbol{\mu}_k\}$

- Step 0: randomly assign the cluster centers $\{\mu_k\}$
- Step 1: Minimize J over $\{r_{nk}\}$ -- Assign every point to the closest cluster center
- Step 2: Minimize J over $\{\mu_k\}$ -- update the cluster centers
- Loop until it converges

Example


(a) Step 0
(b) Step 1
(c) Step 2
(d) Step 1
(e) Step 2
(f) Step 1
(g) Step 2
(h) Step 1
(i) Step 2

K-means algorithm a.k.a Llyod's algorithm

- Step 0: randomly assign the cluster centers $\{\mu_k\}$
- Step 1: Minimize J over $\{r_{nk}\}$ -- Assign every point to the closest cluster center

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\boldsymbol{x}_n - \boldsymbol{\mu}_j\|_2^2 \\ 0 & \text{otherwise} \end{cases}$$

- Step 2: Minimize J over $\{\mu_k\}$ -- update the cluster centers

$$\boldsymbol{\mu}_k = \frac{\sum_n r_{nk}\boldsymbol{x}_n}{\sum_n r_{nk}}$$

- Loop until it converges

Remarks

- Prototype μ_k is the mean of data points assigned to the cluster k, hence `K-means'
- μ_k may not in the training set
- k need to be pre-defined
  - There are some other approaches for the case k is unknown – not cover in class
- The procedure reduces J in both Step 1 and Step 2 and thus makes improvements on each iteration

Application: Vector Quantization

- Replace data point with associated prototype μ_k
- i.e., compress the data points into i) a codebook of all the prototypes; ii) a list of indices to the codebook for the data points

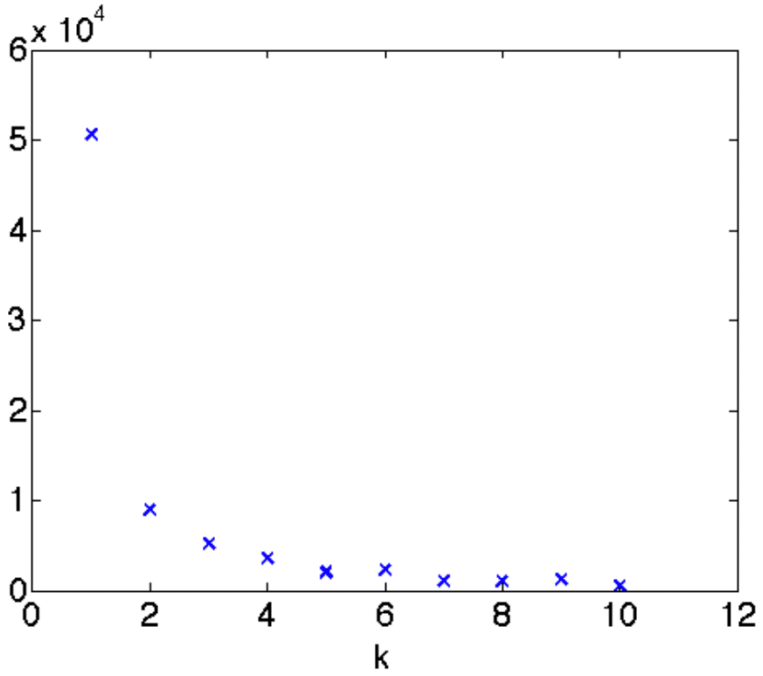
- E.g., Clustering pixels and vector quantizing them. From left to right: Original image, quantized with large K, medium K, and a small K.

Properties of the K-means algorithm

- Does the K-means algorithm converge
  - Yes
- How long does it take to converge?
  - In the worst case, exponential in the number of data points
  - In practice, very quick
- How good is its solution?
  - Local minimum (depends on the initialzation)
  - It can be sensitive to the outlier

Choosing K

- Increasing K will always decrease the optimal value of the K-means objective.
  - Analogous to overfitting in supervised learning.



K-means can be sensitive to the outlier

- One data point can make the center shift

K-medoids

- K-means is sensitive to outliers.
- In some applications we want the prototypes to be one of the points.
- Leads to K-medoids.

K-medoids algorithm

- Step 0: randomly selecting K points as the cluster centers $\{\mu_k\}$
- Step 1: Minimize J over $\{r_{nk}\}$ -- Assign every point to the closest cluster center

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\boldsymbol{x}_n - \boldsymbol{\mu}_j\|_2^2 \\ 0 & \text{otherwise} \end{cases}$$

- Step 2: Update the cluster centers– the porotype for a cluster is the data that is closest to all other data points in the cluster

$$k* = \arg\min_{m:r_{mk}=1} \sum_n r_{nk} \|\boldsymbol{x}_n - \boldsymbol{x}_m\|_2^2$$

$$\boldsymbol{\mu}_k = \boldsymbol{x}_{k*}$$

- Loop until it converges

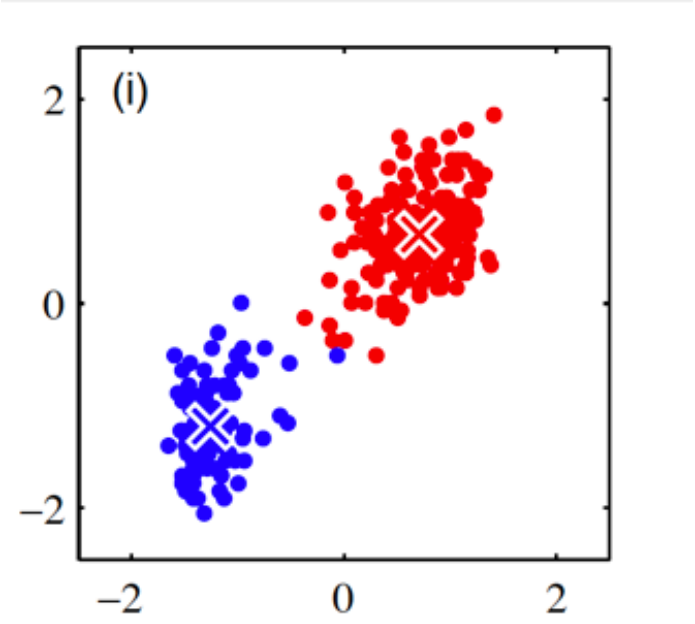
Intuition@ Hogwarts

- Sorting Hat – cluster students into four groups based on four underlying prototypes

- The prototype of each house is the most represented student of the house
  - Alternatively, updating the prototype & the student assignment

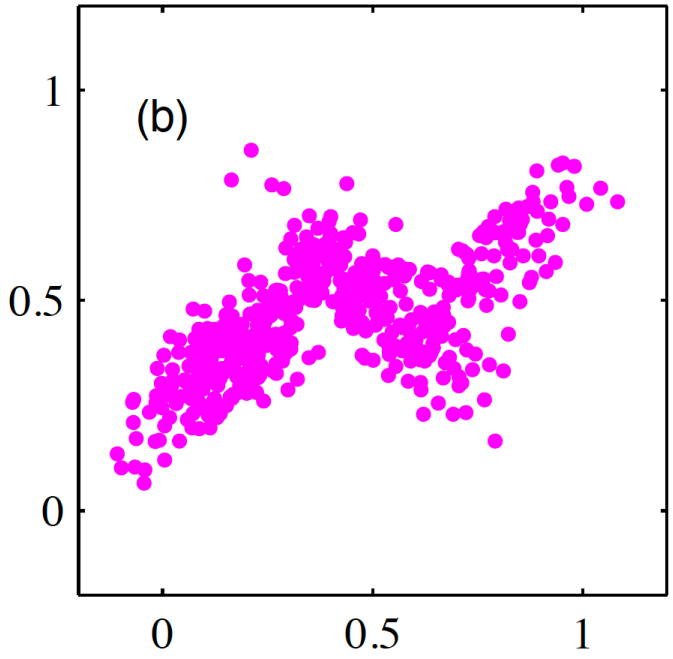
Probabilistic interpretation of clustering?

- Until now, we make a hard assignment for clustering
  - Each point assigns to one cluster
  - Can we allow probability in the assignment?



Gaussian mixture models

- Intuition: we can model each region with a distinct distribution



- Data points seem to form 3 clusters
- We cannot model $p(\boldsymbol{x})$ with simple and known distributions
- E.g., the data is not a Guassian b/c we have 3 distinct concentrated regions

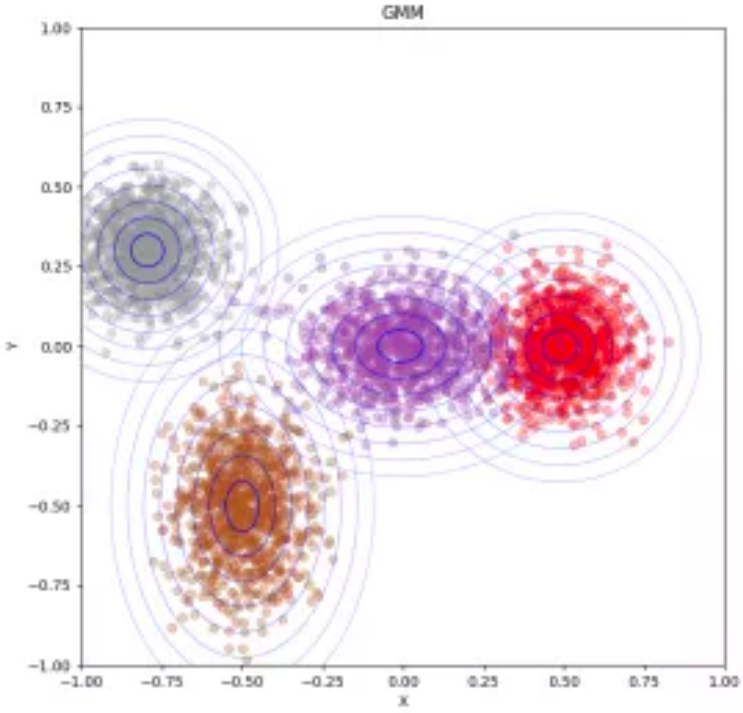
Gaussian mixture models

- We can use a Gaussian distribution or a mixture of Gaussians to model each region

- We need to figure out the cluster assignment and the parameters of Gaussians (mean, variances, etc.) from data

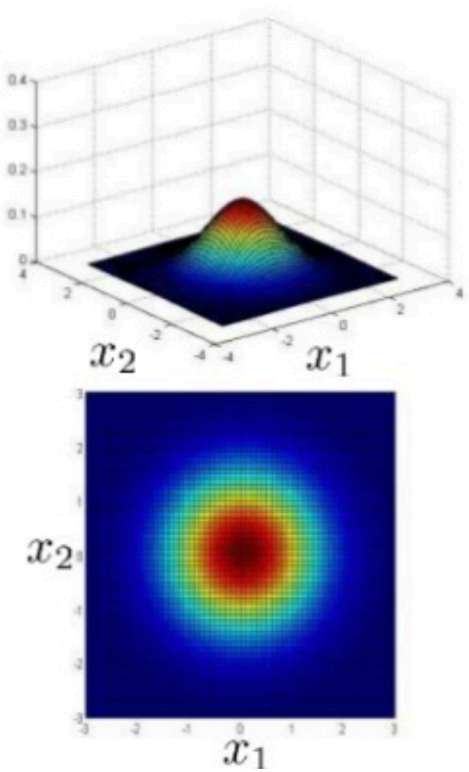
Intuition@ Hogwarts

- Sorting Hat – cluster students into four groups based on four underlying prototypes
- For each house, we learn a distribution

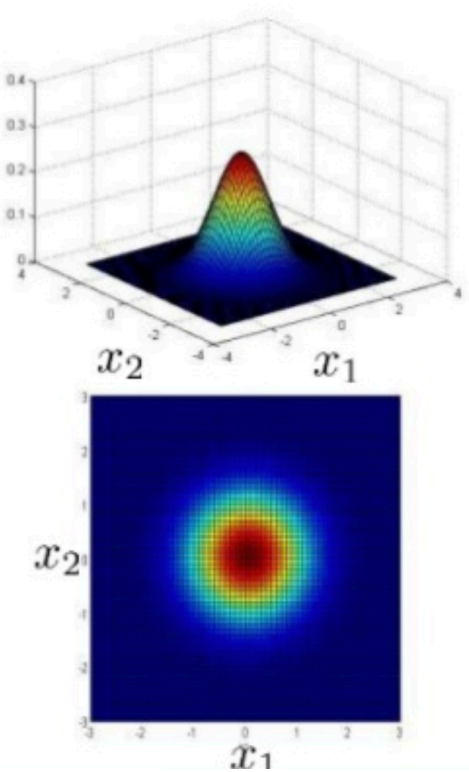


Multivariate Gaussian (Normal) distribution

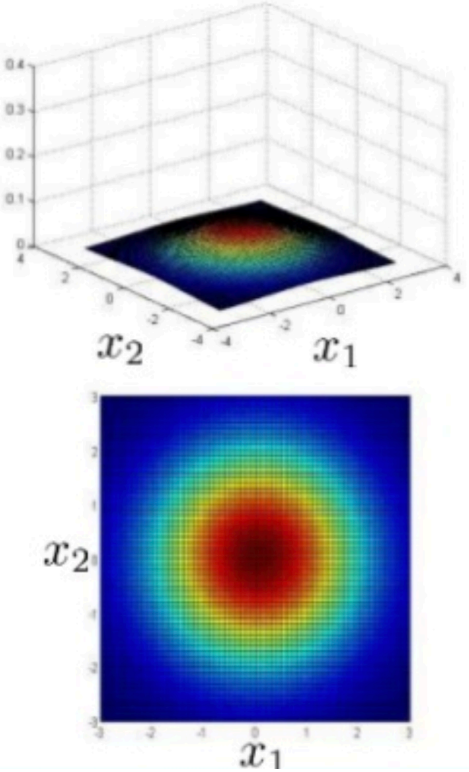
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$$

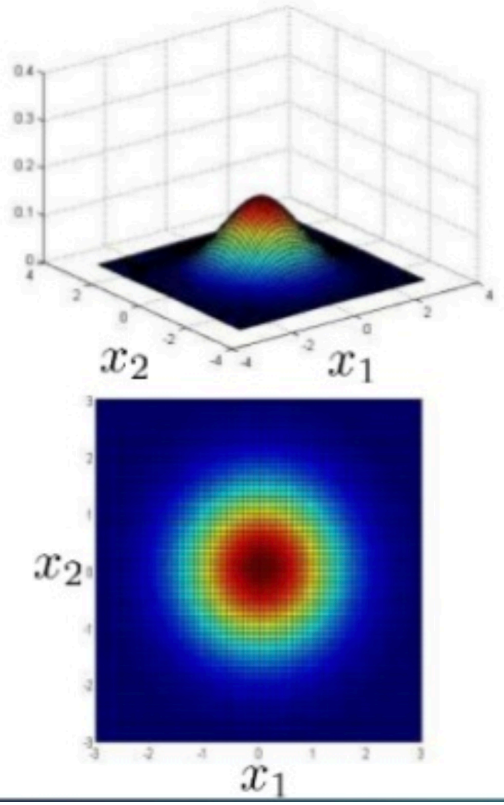


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

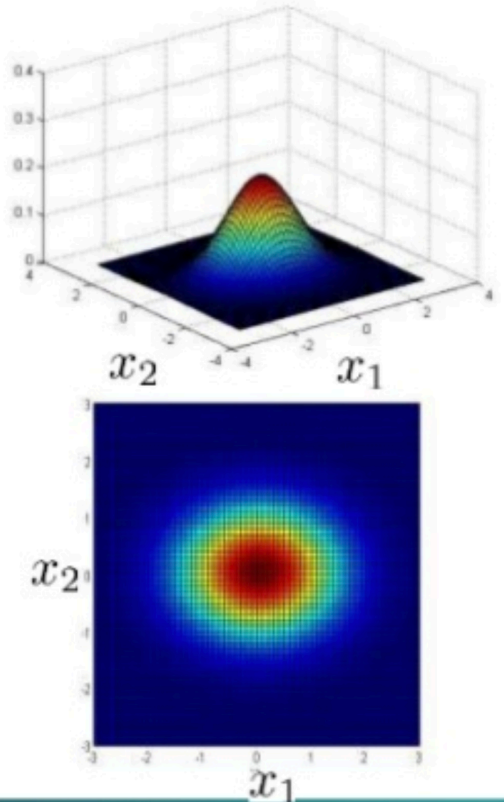


Multivariate Gaussian (Normal) distribution

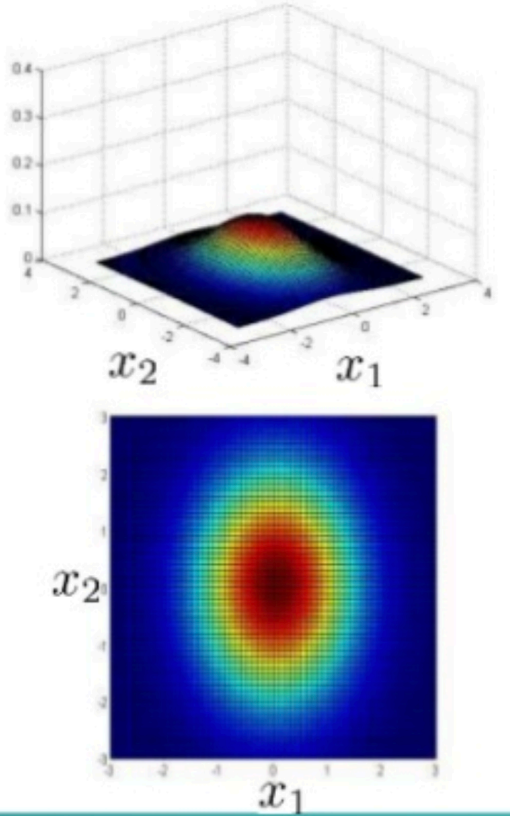
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 0.6 \end{bmatrix}$$



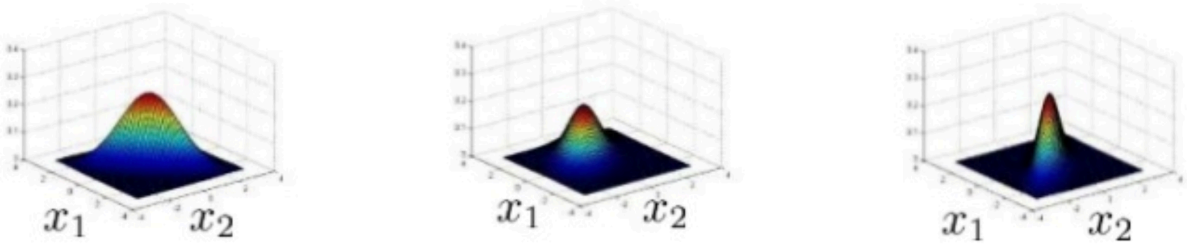
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$



Properties of Gaussian distribution

- Parameters μ, Σ

$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$



Parameter fitting:

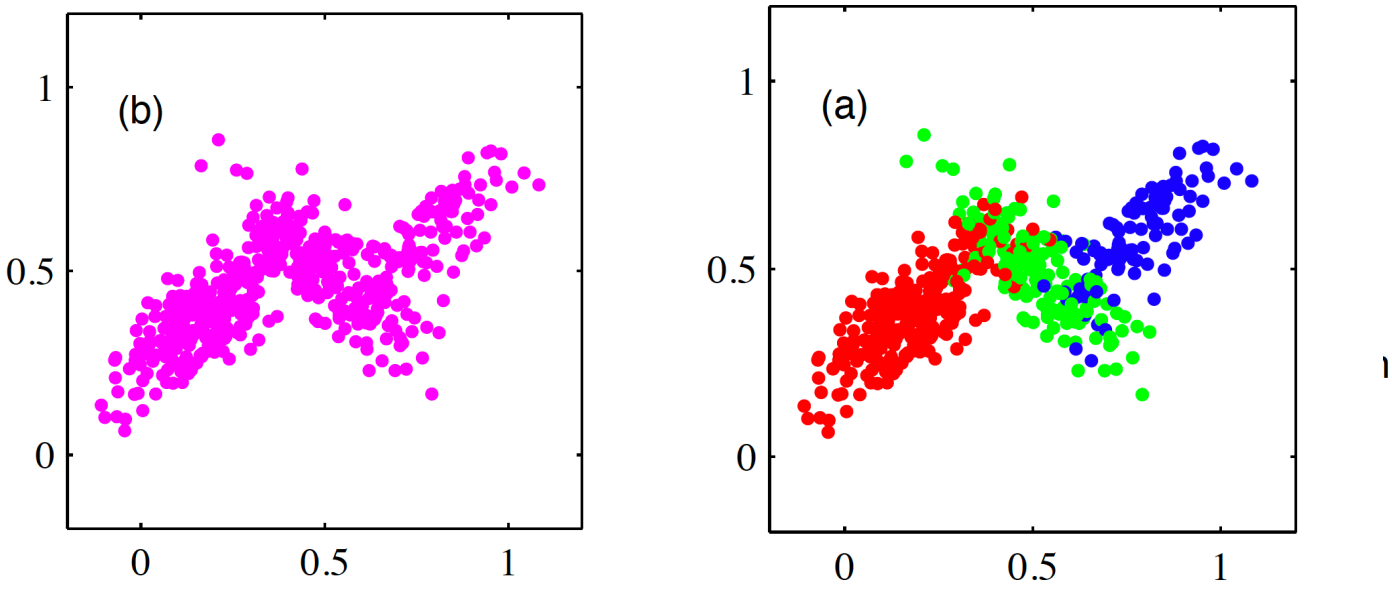
Given training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \qquad \Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)}-\mu)(x^{(i)}-\mu)^T$$

Gaussian mixture models

- Assume the probability density function for x as

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \omega_k N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$



Gaussian mixture models: formal definition

A Gaussian mixture model has the following density function for $\boldsymbol{x}$

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \omega_k N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

- K: the number of Gaussians — they are called (mixture) components
- $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$: mean and covariance matrix of the k-th component
- ω_k: mixture weights – they represent how much each component contributes to the final distribution. It satisfies two properties:

$$\forall\ k,\ \omega_k > 0, \quad \text{and} \quad \sum_k \omega_k = 1$$

The properties ensure $p(\boldsymbol{x})$ is a properly normalized probability density function.

GMM as the marginal distribution P(x) of a joint distribution P(x, z)

Consider the following joint distribution

$$p(\boldsymbol{x}, z) = p(z)p(\boldsymbol{x}|z)$$

where z is a discrete random variable taking values between 1 and K. Denote

$$\omega_k = p(z = k)$$

Now, assume the conditional distributions are Gaussian distributions

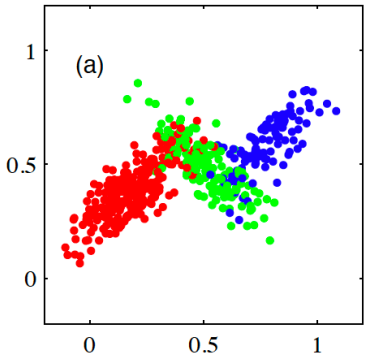
$$p(\boldsymbol{x}|z = k) = N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Then, the marginal distribution of $\boldsymbol{x}$ is

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \omega_k N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

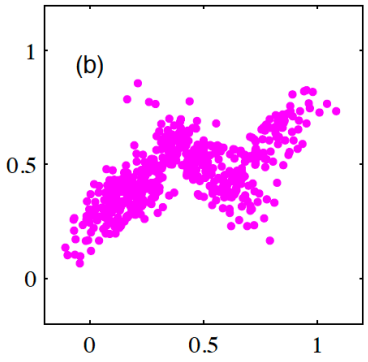
Namely, the Gaussian mixture model

Example



The conditional distribution between $\boldsymbol{x}$ and z (representing color) are

$$p(\boldsymbol{x}|z = red) = N(\boldsymbol{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$$
$$p(\boldsymbol{x}|z = blue) = N(\boldsymbol{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$$
$$p(\boldsymbol{x}|z = green) = N(\boldsymbol{x}|\boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3)$$



The marginal distribution is thus

$$p(\boldsymbol{x}) = p(red)N(\boldsymbol{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) + p(blue)N(\boldsymbol{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) + p(green)N(\boldsymbol{x}|\boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3)$$

ω_k

Parameter estimation for GMMs

- If cluster assignments are observed $\{z_n\}$ are given
  - We know the cluster of each point
  - Let $\gamma_{nk} = 1$ if instance n belongs to cluster k, otherwise $\gamma_{nk} = 0$
- Then the maximum likelihood estimation is

$$\omega_k = \frac{\sum_n \gamma_{nk}}{\sum_k \sum_n \gamma_{nk}}, \quad \boldsymbol{\mu}_k = \frac{1}{\sum_n \gamma_{nk}} \sum_n \gamma_{nk} \boldsymbol{x}_n$$

$$\boldsymbol{\Sigma}_k = \frac{1}{\sum_n \gamma_{nk}} \sum_n \gamma_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}}$$

$$\omega_k = \frac{\sum_n \gamma_{nk}}{\sum_k \sum_n \gamma_{nk}}, \quad \boldsymbol{\mu}_k = \frac{1}{\sum_n \gamma_{nk}} \sum_n \gamma_{nk} \boldsymbol{x}_n$$

$$\boldsymbol{\Sigma}_k = \frac{1}{\sum_n \gamma_{nk}} \sum_n \gamma_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}}$$

Since γ_{nk} is binary, the previous solution is nothing but

- For ω_k: count the number of data points whose z_n is k and divide by the total number of data points (note that $\sum_k \sum_n \gamma_{nk} = N$)
- For $\boldsymbol{\mu}_k$: get all the data points whose z_n is k, compute their mean
- For $\boldsymbol{\Sigma}_k$: get all the data points whose z_n is k, compute their covariance matrix

Parameter estimation for GMMs

- When the cluster assignments are not given – the real case
- We use an approach similar to k-means
- Alternative update the cluster assignment γ_{nk} and parameter estimation $\{\omega_k, \mu_k, \Sigma_k\}$

Iterative procedure

- Let θ represent all parameters $\{\omega_k, \mu_k, \Sigma_k\}$

Step 0: initialize $\boldsymbol{\theta}$ with some values (random or otherwise)

Step 1: compute γ_{nk} using the current $\boldsymbol{\theta}$

Step 2: update $\boldsymbol{\theta}$ using the just computed γ_{nk}

Step 3: go back to Step 1

Estimate γ_{nk}

- γ_{nk} the assignment of instance n to cluster k, can be defined as $\gamma_{nk} = P(z_n = k|x_n)$
- Can be computed vis the posterior probability

$N(x|\mu_k, \Sigma_k)$ ω_k

$$p(z_n = k|\boldsymbol{x}_n) = \frac{p(\boldsymbol{x}_n|z_n = k)p(z_n = k)}{p(\boldsymbol{x}_n)} = \frac{p(\boldsymbol{x}_n|z_n = k)p(z_n = k)}{\sum_{k'=1}^{K} p(\boldsymbol{x}_n|z_n = k')p(z_n = k')}$$