K-means Clustering and Extensions

Piyush Rai

Introduction to Machine Learning (CS771A)

September 13, 2018

Announcement: Mid-Sem Exam

- September 20 (Thursday) 13:00-15:00
- Venue: L17, L18, L19 (all OROS)
- Syllabus: Up to what we see today (only 1-2 small questions from today's lecture)
- Open notes, open slides (please only print in 4-up mode), books not allowed
- No electronic items allowed (please keep phones switched off)
- Important: ALL answers have to be written on the question paper itself (dedicated space)
- Important: You need to bring a few other things
  - A notebook for rough work (may use blank pages from your notes)
  - Pen, pencil and eraser (but final answers should be written with pen to avoid smudging)
- A review session: September 15/16/17 (timing/venue TBD)

Recap: Speeding Up Kernel Methods

- Can extract "good" features $\psi(\boldsymbol{x}) \in \mathbb{R}^L$ from a kernel k (with mapping ϕ) such that

$$\psi(\boldsymbol{x}_n)^\top \psi(\boldsymbol{x}_m) \approx \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m)$$

- Unlike the kernel's original mapping ϕ, the mapping ψ is low-dimensional (L is typically small)
- With these features $\psi(\boldsymbol{x}_n)$, we can apply a linear model (both train/test). No need to kernelize.
- Looked at two main approaches to get such an approximate mapping ψ
- Landmark based approach: Using landmark points $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L$ (selected or learned), compute

$$\psi(\boldsymbol{x}_n) = [k(\boldsymbol{z}_1, \boldsymbol{x}_n), k(\boldsymbol{z}_2, \boldsymbol{x}_n), \ldots, k(\boldsymbol{z}_L, \boldsymbol{x}_n)]$$

- Kernel Random Features approach: Can be used for many kernels. For the RBF kernel

$$\begin{aligned} \psi(\boldsymbol{x}_n) &= \frac{1}{\sqrt{L}}[\cos(\boldsymbol{w}_1^\top \boldsymbol{x}_n + b_1), \ldots, \cos(\boldsymbol{w}_L^\top \boldsymbol{x}_n + b_L)] \\ \boldsymbol{w}_\ell &\sim \mathcal{N}(0, \lambda^{-1}\mathbf{I}_D), \quad b_\ell \sim \text{Unif}(0, 2\pi), \quad \ell = 1, \ldots, L \end{aligned}$$

- Some other approaches (that we didn't see): Nyström approx, other low-rank kernel matrix approx

Recap: K-means Algorithm

- Goal: Assign N inputs $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, with each $\boldsymbol{x}_n \in \mathbb{R}^D$, to K clusters (flat partitioning)
- Notation: $z_n \in \{1, \ldots, K\}$ or $\boldsymbol{z}_n$ is a K-dim one-hot vector($z_{nk} = 1$ and $z_n = k$ mean the same)

K-means Algorithm

1. Initialize K cluster means $\mu_1, \ldots, \mu_K$
2. For $n = 1, \ldots, N$, assign each point $\boldsymbol{x}_n$ to the closest cluster

$$z_n = \arg\min_{k \in \{1,\ldots,K\}} ||\boldsymbol{x}_n - \mu_k||^2$$

3. Suppose $\mathcal{C}_k = \{\boldsymbol{x}_n : z_n = k\}$. Re-compute the means

$$\mu_k = \text{mean}(\mathcal{C}_k), \quad k = 1, \ldots, K$$

4. Go to step 2 if not yet converged

- Note: The basic K-means models each cluster only by a mean μ_k. Ignores size/shape of clusters

The K-means Algorithm: Some Comments

- One of the most popular clustering algorithms
- Very widely used, guaranteed to converge (to a local minima; will see a proof)
- Can also be used as a sub-routine in graph clustering (in the Spectral Clustering algorithm)
- Has some shortcomings (as we will see) but can be improved upon
- Some of the many improvements (some of which we will see)
  - Can be kernelized (using kernels or using kernel-based landmarks/random features)
  - More flexible cluster sizes/shapes via probabilistic models (e.g., every cluster is a Gaussian)
  - Soft-clustering (fractional/probabilistic memberships): $\boldsymbol{z}_n$ is a probability vector
  - Overlapping clustering - a point can belong to multiple clusters: $\boldsymbol{z}_n$ is a binary vector
  - .. even deep learning based K-means :-)
- .. so it is worth looking a bit deeply into what K-means is doing

K-means Loss Function: Several Forms, Same Meaning!

Notation: $\mathbf{X}$ is $N \times D$, $\mathbf{Z}$ is $N \times K$ (each row is a one-hot $\boldsymbol{z}_n$), $\boldsymbol{\mu}$ is $K \times D$ (each row is a μ_k)

$$\mathcal{L}(\mathbf{X}, \mathbf{Z}, \boldsymbol{\mu}) = \sum_{n=1}^{N} ||\boldsymbol{x}_n - \mu_{z_n}||^2$$

“distortion” on assignment to cluster z_n

$$\mathcal{L}(\mathbf{X}, \mathbf{Z}, \boldsymbol{\mu}) = \sum_{k=1}^{K} \underbrace{\sum_{n:z_n=k} ||\boldsymbol{x}_n - \mu_k||^2}_{\text{within cluster variance}}$$

$$\mathcal{L}(\mathbf{X}, \mathbf{Z}, \boldsymbol{\mu}) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \mu_k||^2$$

$$\mathcal{L}(\mathbf{X}, \mathbf{Z}, \boldsymbol{\mu}) = \underbrace{||\mathbf{X} - \mathbf{Z}\boldsymbol{\mu}||_F^2}_{\text{as matrix factorization}}$$

Total “distortion” or reconstruction error

$$\boxed{\{\hat{\mathbf{Z}}, \hat{\boldsymbol{\mu}}\} = \arg\min_{\mathbf{Z}, \boldsymbol{\mu}} \mathcal{L}(\mathbf{X}, \mathbf{Z}, \boldsymbol{\mu})}$$

Note: Replacing ℓ_2 squared (Euclidean) distance by absolute (ℓ_1) distance gives the K-medians algorithm (more robust to outliers)

Note: Most unsup. learning algos try to minimize the distortion or reconstruction error of $\mathbf{X}$ from $\mathbf{Z}$

Optimizing the K-means Loss Function

- So the K-means problem is

$$\{\hat{\mathbf{Z}}, \hat{\boldsymbol{\mu}}\} = \arg\min_{\mathbf{Z},\boldsymbol{\mu}} \mathcal{L}(\mathbf{X}, \mathbf{Z}, \boldsymbol{\mu}) = \arg\min_{\mathbf{Z},\boldsymbol{\mu}} \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \mu_k||^2$$

- Can't optimize it jointly for $\mathbf{Z}$ and $\boldsymbol{\mu}$. Let's try alternating optimization for $\mathbf{Z}$ and $\boldsymbol{\mu}$

Alternating Optimization for K-means Problem

1. Fix $\boldsymbol{\mu}$ as $\hat{\boldsymbol{\mu}}$ and find the optimal $\mathbf{Z}$ as

$$\hat{\mathbf{Z}} = \arg\min_{\mathbf{Z}} \mathcal{L}(\mathbf{X}, \mathbf{Z}, \hat{\boldsymbol{\mu}}) \quad \text{(still not easy - next slide)}$$

2. Fix $\mathbf{Z}$ as $\hat{\mathbf{Z}}$ and find the optimal $\boldsymbol{\mu}$ as

$$\hat{\boldsymbol{\mu}} = \arg\min_{\boldsymbol{\mu}} \mathcal{L}(\mathbf{X}, \hat{\mathbf{Z}}, \boldsymbol{\mu})$$

3. Go to step 1 if not yet converged

Solving for Z

- Solving for $\mathbf{Z}$ with $\boldsymbol{\mu}$ fixed at $\hat{\boldsymbol{\mu}}$

$$\hat{\mathbf{Z}} = \arg\min_{\mathbf{Z}} \mathcal{L}(\mathbf{X}, \mathbf{Z}, \hat{\boldsymbol{\mu}}) = \arg\min_{\mathbf{Z}} \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \hat{\mu}_k||^2$$

- Still not easy. Since $\mathbf{Z}$ is discrete, it is an NP-hard problem
    - Combinatorial optimization: K^N possibilities for $\mathbf{Z}$ ($N \times K$ matrix with one-hot rows)
- A greedy approach: Optimize $\mathbf{Z}$ one row ($\boldsymbol{z}_n$) at a time keeping all others $\boldsymbol{z}_n$'s (and $\boldsymbol{\mu}$) fixed

$$\hat{\boldsymbol{z}}_n = \arg\min_{\boldsymbol{z}_n} \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \hat{\mu}_k||^2 = \arg\min_{z_n} ||\boldsymbol{x}_n - \hat{\mu}_{z_n}||^2$$

- Easy to see that this is minimized by assigning $\boldsymbol{x}_n$ to the closest mean
    - This is exactly what the K-means algo does!

Solving for $\boldsymbol{\mu}$

- Solving for $\boldsymbol{\mu}$ with $\mathbf{Z}$ fixed at $\hat{\mathbf{Z}}$

$$\hat{\boldsymbol{\mu}} = \arg\min_{\boldsymbol{\mu}} \mathcal{L}(\mathbf{X}, \hat{\mathbf{Z}}, \boldsymbol{\mu}) = \arg\min_{\boldsymbol{\mu}} \sum_{k=1}^{K} \sum_{n:\hat{z}_n=k} ||\boldsymbol{x}_n - \mu_k||^2$$

- This is not that hard to solve (μ_k's are real-valued vectors, can optimize easily)
- Note that each μ_k can be optimized independently

$$\hat{\mu}_k = \arg\min_{\mu_k} \sum_{n:\hat{z}_n=k} ||\boldsymbol{x}_n - \mu_k||^2$$

- (Verify) This is minimized by setting $\hat{\mu}_k$ to be mean of points currently in cluster k
  - This is exactly what the K-means algo does!

Convergence of K-means Algorithm

- Each step (updating $\mathbf{Z}$ or $\boldsymbol{\mu}$) can **never increase** the K-means loss
- When we update $\mathbf{Z}$ from $\mathbf{Z}^{(t-1)}$ to $\mathbf{Z}^{(t)}$

$$\mathcal{L}(\mathbf{X}, \mathbf{Z}^{(t)}, \boldsymbol{\mu}^{(t-1)}) \leq \mathcal{L}(\mathbf{X}, \mathbf{Z}^{(t-1)}, \boldsymbol{\mu}^{(t-1)})$$

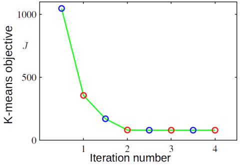
  because the new $\mathbf{Z}^{(t)} = \arg\min_{\mathbf{Z}} \mathcal{L}(\mathbf{X}, \mathbf{Z}, \boldsymbol{\mu}^{(t-1)})$

- When we update $\boldsymbol{\mu}$ from $\boldsymbol{\mu}^{(t-1)}$ to $\boldsymbol{\mu}^{(t)}$

$$\mathcal{L}(\mathbf{X}, \mathbf{Z}^{(t)}, \boldsymbol{\mu}^{(t)}) \leq \mathcal{L}(\mathbf{X}, \mathbf{Z}^{(t)}, \boldsymbol{\mu}^{(t-1)})$$

  because the new $\boldsymbol{\mu}^{(t)} = \arg\min_{\boldsymbol{\mu}} \mathcal{L}(\mathbf{X}, \mathbf{Z}^{(t)}, \boldsymbol{\mu})$

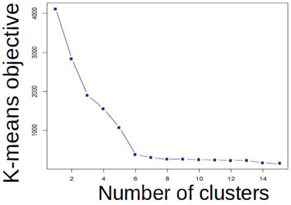
- Thus the K-means algorithm monotonically decreases the objective



(blue: after Z updated, red: after μ updated)

K-means: Choosing K

- One way to select K for the K-means algorithm is to try different values of K, plot the K-means objective versus K, and look at the "elbow-point"



- For the above plot, $K = 6$ is the elbow point
- Can also information criterion such as AIC (Akaike Information Criterion)

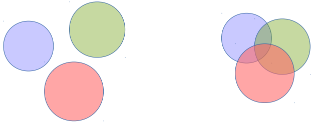
$$AIC = 2\mathcal{L}(\hat{\boldsymbol{\mu}}, \mathbf{X}, \hat{\mathbf{Z}}) + KD$$

.. and choose the K that has the smallest AIC (discourages large K)

- Several other approaches when using probabilistic models for clustering, e.g., comparing marginal likelihood $p(\mathbf{X}|K)$, using nonparametric Bayesian models, etc.

K-means: Hard vs Soft Assignments

- Makes hard assignments of points to clusters
  - A point either completely belongs to a cluster or doesn't belong at all
  - No notion of a soft assignment (i.e., probability of being assigned to each cluster: say $K = 3$ and for some point $\boldsymbol{x}_n$, $p_1 = 0.7, p_2 = 0.2, p_3 = 0.1$)



Hard-assignment okay

Hard-assignment tricky

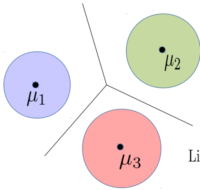
- A heuristic to get soft assignments: Transform distances from clusters into probabilities

$$\gamma_{nk} = \frac{\exp(-||\boldsymbol{x}_n - \mu_k||^2)}{\sum_{\ell=1}^{K} \exp(-||\boldsymbol{x}_n - \mu_\ell||^2)} \quad \text{(prob. that } \boldsymbol{x}_n \text{ belongs to cluster } k\text{)}$$

- These heuristics are used in "fuzzy" or "soft" K-means algorithms
- Soft K-means μ_k updates are slightly different: $\mu_k = \frac{\sum_{n=1}^{N} \gamma_{nk} \boldsymbol{x}_n}{\sum_{n=1}^{N} \gamma_{nk}}$ (all points used, but fractionally)

K-means: Decision Boundaries and Cluster Sizes/Shapes

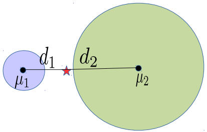
- *K*-mean assumes that the decision boundary between any two clusters is linear
- Reason: The *K*-means loss function implies assumes equal-sized, spherical clusters

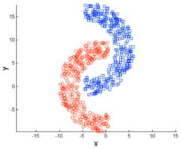


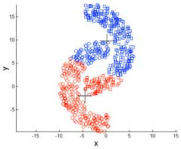
Distance from a cluster $= ||\boldsymbol{x}_n - \mu_k||^2$

Like using Gaussians with equal covariances

- Assumes clusters to be roughly equi-populated, and convex-shaped. Otherwise, may do badly







- Kernel *K*-means can help address some of these issues. Probabilistic models is another option

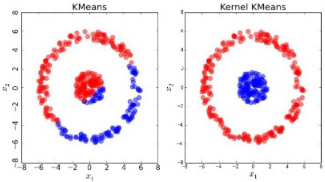
Kernel K-means

- Basic idea: Replace the Euclidean distances in K-means by the kernelized versions

$$
\begin{aligned}
||\phi(\boldsymbol{x}_n) - \phi(\boldsymbol{\mu}_k)||^2 &= ||\phi(\boldsymbol{x}_n)||^2 + ||\phi(\boldsymbol{\mu}_k)||^2 - 2\phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{\mu}_k) \\
&= k(\boldsymbol{x}_n, \boldsymbol{x}_n) + k(\boldsymbol{\mu}_k, \boldsymbol{\mu}_k) - 2k(\boldsymbol{x}_n, \boldsymbol{\mu}_k)
\end{aligned}
$$

- Here $k(.,.)$ denotes the kernel function and ϕ is its (implicit) feature map
- Note: $\phi(\boldsymbol{\mu}_k)$ is the average of ϕ's the data points assigned to cluster k

Kernel K-means vs. K-means



Pyclust: Open Source Data Clustering Pckage

- Can also use landmark or random features approach to make it faster
  - Can then simply run the basic K-means on those features!

Going the Probabilistic Way..

- Assume a generative model for the inputs. Suppose Θ denotes all the unknown parameters
- Clustering then boils down to computing $p(\boldsymbol{z}_n|\boldsymbol{x}_n, \Theta)$ for each $\boldsymbol{x}_n$, where $\boldsymbol{z}_n$ is a latent variable
- Using the Bayes rule, we can write $p(\boldsymbol{z}_n|\boldsymbol{x}_n, \Theta)$ as

$$p(\boldsymbol{z}_n = k|\boldsymbol{x}_n, \Theta) = \frac{p(\boldsymbol{z}_n = k|\Theta)p(\boldsymbol{x}_n|\boldsymbol{z}_n = k, \Theta)}{p(\boldsymbol{x}_n|\Theta)}$$

- Assuming $p(\boldsymbol{z}|\Theta)$ as multinoulli(π) and each cluster as Gaussian $p(\boldsymbol{x}|\boldsymbol{z} = k, \Theta) = \mathcal{N}(\boldsymbol{x}|\mu_k, \Sigma_k)$

$$p(\boldsymbol{z}_n = k|\boldsymbol{x}_n, \Theta) \propto \pi_k \times \mathcal{N}(\boldsymbol{x}_n|\mu_k, \Sigma_k)$$

Cluster assignment prob now depends on the number of points in cluster k

Different clusters can have different shapes (covariances)

$$\text{(here } \Theta = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}\text{)}$$

- We know how to estimate Θ for such problems.. if $\boldsymbol{z}_n$ is known (recall generative classification)
- The tricky part here is that we don't know $\boldsymbol{z}_n$. How do we estimate Θ then?
- A solution: Take an alternating approach (like K-means)

Going the Probabilistic Way..

- At a high-level, a probabilistic clustering algorithm would look somewhat like this

Sketch of a Probabilistic Clustering Algorithm

1. Initialize the model parameters Θ somehow
2. Given the current Θ, estimate $\mathbf{Z}$ (cluster assignments) in a soft/hard way

$$\begin{aligned} p(\boldsymbol{z}_n = k|\boldsymbol{x}_n, \Theta) = \gamma_{nk} &= \frac{p(\boldsymbol{z}_n = k|\Theta)p(\boldsymbol{x}_n|\boldsymbol{z}_n = k, \Theta)}{p(\boldsymbol{x}_n|\Theta)}, \quad k = 1, \ldots, K \\ \text{OR} \quad \hat{\boldsymbol{z}}_n &= \arg\max_{k \in \{1,\ldots,K\}} \gamma_{nk} \end{aligned}$$

3. Use $\{\hat{\boldsymbol{z}}_n\}_{n=1}^{N}$ (hard cluster labels) or $\{\gamma_{nk}\}_{n,k=1}^{N,K}$ (soft labels) to update Θ via MLE/MAP (similar to how we do for gen. classification where the labels are known)
4. Note: The soft-label based Θ updates slightly more involved (wait until we see EM)
5. Go to step 2 if not converged yet.

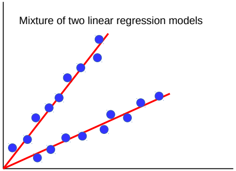
- The above algorithm is an instance of a more general Expectation Maximization (EM) algorithm for latent variable models (we will see this post mid-sem)

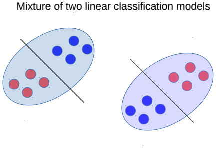
Clustering vs Classification

- Any clustering model typically learns two type of quantities
  - Parameters Θ of the clustering model (e.g., cluster means $\boldsymbol{\mu} = \{\mu_1, \ldots, \mu_K\}$ in K-means)
  - Cluster assignments $\mathbf{Z} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$ for the points
- If the cluster assignments $\mathbf{Z}$ are known, learning the parameters Θ is just like learning the parameters of a classification model (typically generative classification) using labeled data
- Therefore it helps to think of clustering as (generative) classification with unknown labels
- This equivalence is very important and makes it possible to solve clustering problems
- Therefore many clustering problems are typically solved in the following fashion
  1. Initialize Θ somehow
  2. Predict $\mathbf{Z}$ given current estimate of Θ
  3. Use the predicted $\mathbf{Z}$ to improve the estimate of Θ (like learning a generative classification model)
  4. Go to step 2 if not converged yet

Clustering can help supervised learning, too

- Often “difficult” supervised learning problems can be seen as mixture of simpler models
- Example: Nonlinear regression or nonlinear classification as mixture of linear models





- An alternative to kernel methods and deep learning :-)
- Don’t know which point belongs to which linear model ⇒ Clustering problem
- Can therefore solve such problems as follows
  1. Initialize each linear model somehow (maybe randomly)
  2. Cluster the data by assigning each point to its “closest” linear model
  3. (Re-)Learn a linear model for each cluster’s data. Go to step 2 if not converged.
- Often called Mixture of Experts models. Will look at these more formally after mid-sem