Modele mixte

Cursul 10

Programare şi modelare probabilistă - anul III

Facultatea de Informatică, UAIC

e-mail: adrian.zalinescu@uaic.ro

web: https://sites.google.com/view/fiicoursepmp/home

18 Decembrie 2023

În statistică, *modelele mixte* (sau *de mixtură*) reprezintă o modalitate comună de a crea noi modele.

- Acestea sunt construite prin mixarea unor distribuţii mai simple, obţinând astfel modele mai complexe.
- De exemplu, se pot mixa două distribuţii Gausienne pentru a obţine o distribuţie bimodală, sau un număr arbitrar de distribuţii Gaussiene pentru a obţine o distribuţie oarecare.
- Se pot mixa orice distribuţii dorim, nu neapărat distribuţii normale.
- Modelele mixte sunt folosite în diverse scopuri, ca:
  - modelarea directă a sub-populaţiilor sau
  - manipularea unor distribuţii complicate, ce nu pot fi descrise prin distribuţii simple.

Modele mixte

Acestea apar în mod natural atunci când *populaţia* (în sens statistic) este o combinaţie de *sub-populaţii* distincte. De exemplu:

- distribuţia înălţimilor într-o populaţie de adulţi, ce poate fi descrisă ca o mixtură de sub-populaţii de femei şi de bărbaţi.
- clusterizarea cifrelor (din sistemul decimal), caz în care vom avea 10 sub-populaţii (cifrele!)

Dacă ştim din ce sub-populaţie o anumită observaţie face parte, atunci o idee este să modelăm fiecare populaţie ca un grup separat. Dar, în general nu avem acces la această informaţie!

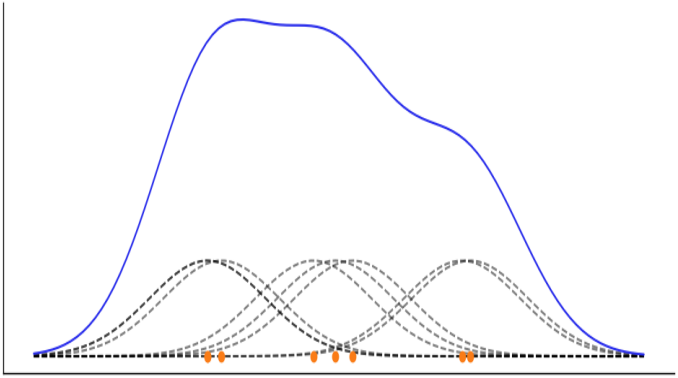
Atunci când construim un model mixt, nu este neapărat necesar să credem că descriem cu adevărat sub-populaţii în datele folosite.

- Modelele mixte pot fi văzute ca un truc statistic pentru a adăuga flexibilitate.

De exemplu:

- distribuţia Gaussiană poate fi văzută ca o aproximare rezonabilă a multor distribuţii uni-modale, mai mult sau mai puţin simetrice.
- Dar ce putem spune despre distribuţiile bimodale sau multimodale?
- Aici se poate folosi o mixtură de distribuţii Gaussiene, cu medii diferite şi (eventual, dar nu necesar) deviaţii standard diferite.
- Astfel, adăugăm flexibilitate modelului pentru a calibra distribuţii complexe de date.

În această figură, observăm combinarea a opt distribuţii Gaussiene cu aceaşi deviaţie standard şi cu medii reprezentate de punctele portocalii.



Modele mixte finite

O modalitate de a construi modele mixte este să considerăm o medie ponderată a două sau mai multe densităţi de distribuţii.
Obţinem astfel un *model mixt finit*:

$$p(y \mid \theta) = \sum_{i=1}^{K} w_i p_i(y \mid \theta_i), \text{ unde}$$

- w_i este *ponderea* componentei i şi poate fi interpretată ca probabilitatea acesteia; astfel, $w_i \in [0, 1]$ şi $\sum_{i=1}^{K} w_i = 1$;
- *componentele* $p_i(y \mid \theta_i)$ pot fi orice distribuţii, de la unele simple, precum cea Gaussiană sau Poisson, la obiecte mai complexe, ca modele ierarhice sau reţele neuronale.
- K este un număr finit (în general, dar nu necesar, $K \leq 20$). Valoarea lui K trebuie dată înainte, fie că ştim valoarea exactă, fie că o intuim.

Pentru a rezolva un model de mixtură, tot ce trebuie să facem este să asignăm fiecare observaţie (din setul de date) uneia dintre componente.

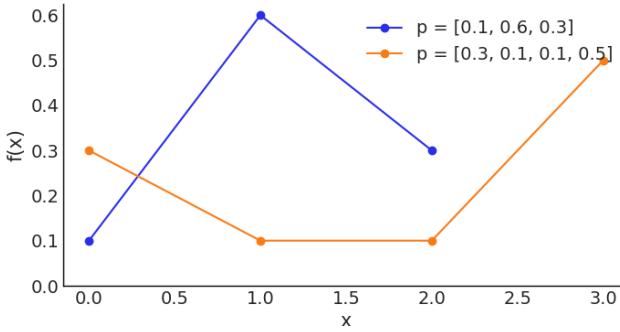
- Într-un model probabilist, putem face acest lucru prin introducerea unei variabile aleatoare z, a cărei utilitate este de a specifica cărei componente îi este asignată o observaţie particulară.
- Această variabilă este cunoscută drept *variabilă latentă*, numită astfel pentru că nu este direct observabilă.
- Variabila z este modelată conform unei *distribuţii categoriale*: o variabilă aleatoare ce ia K valori posibile, fiecare cu o anumită probabilitate w_i.
- Un model mixt este generalizarea modelului aruncării unei monede (modelul Beta-binomial), unde aveam de ales între două rezultate posibile: stemă sau ban, modelat cu o distribuţie binomială, iar parametrul era modelat de o distribuţie a priori Beta.
- În cazul general, în loc de distribuţia binomială vom considera una categorială, iar în loc de distribuţia Beta vom alege o distribuţie Dirichlet.

Distribuţii categoriale

O *distribuţie categorială* este cea mai generală distribuţie discretă, parametrizată folosind probabilităţile fiecărui rezultat posibil:

$$\begin{pmatrix} x_1 & x_2 & \dots & x_n \\ p_1 & p_2 & \dots & p_n \end{pmatrix}.$$

Exemplu de două distribuţii categoriale:



Distribuţia Dirichlet

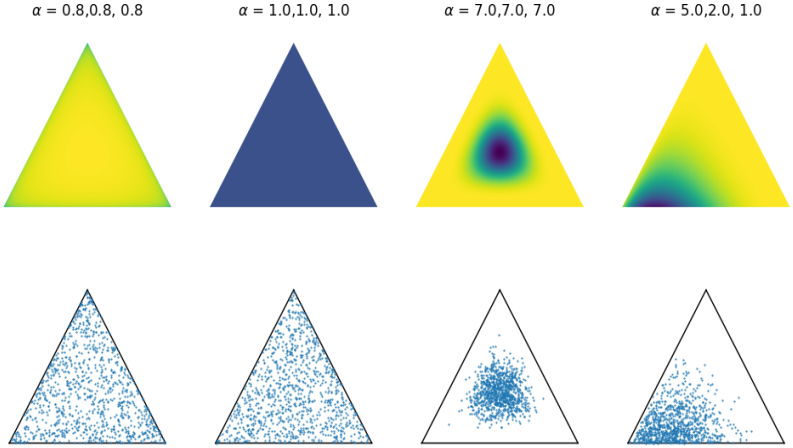
Această distribuţie ia valori într-un *simplex*, ce este analogul unui triunghi, dar într-un număr arbitrar de dimensiuni: un 1-simplex este un segment, un 2-simplex este un triunghi, un 3-simplex este un tetraedru, etc.

- De ce folosim un simplex? Pentru că parametrii distribuţiei latente reprezintă un vector de lungime K:

$$\{(w_1, \ldots, w_K) \in \mathbb{R}_+^K \mid w_1 + \cdots + w_k = 1\},$$

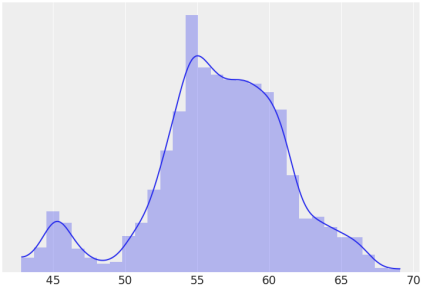
  iar aceşti parametri îi putem privi drept coordonate (nu carteziene) a unui punct interior unui K-simplex.
- O variabilă aleatoare distribuită Dirichlet ia valori într-un K simplex, conform unui parametru α ce este un vector K-dimensional:

Vizualizarea unor distribuţii Dirichlet într-un 3-simplex (triunghi):



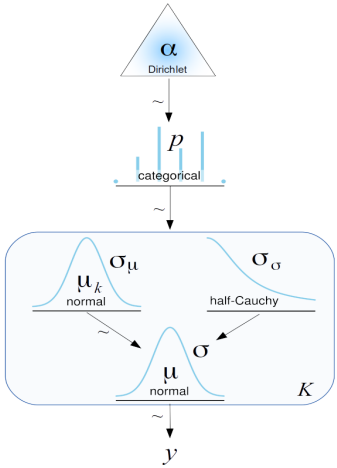
Exemplu:
Considerăm din nou date privind deplasarea chimică a unor nuclee (legate de exemplul din Cursul 5):

```
cs = pd.read_csv('./data/chemical_shifts_theo_exp.csv')
cs_exp = cs['exp']
az.plot_kde(cs_exp)
plt.hist(cs_exp, density=True, bins=30, alpha=0.3)
plt.yticks([])
```



- Putem observa că datele nu pot fi bine descrise folosind o singură distribuţie Gaussiană;
- poate trei sau patru ar fi mai potrivite.

Modelul:
Vizualizare cu diagrame Kruschke:



- Dreptunghiul cu colţurile rotunjite indică faptul că avem K componente şi că variabila latentă (categorială) decide care anume va fi folosită pentru a descrie o anumită observaţie
- În această diagramă, doar μ_k depinde de diverse componente, în timp ce σ_μ şi σ_σ sunt partajate pentru toate componentele. Acest lucru se poate schimba.

Inferenţa (cu `clusters=2`)**:**

```
clusters = 2
with pm.Model() as model_kg:
    p = pm.Dirichlet('p', a=np.ones(clusters))
    z = pm.Categorical('z', p=p, shape=len(cs_exp))
    means = pm.Normal('means', mu=cs_exp.mean(), sigma=10, shape=clusters)
    sd = pm.HalfNormal('sd', sigma=10)
    y = pm.Normal('y', mu=means[z], sigma=sd, observed=cs_exp)
    idata_kg = pm.sample(return_inferencedata=True)
```

Acest cod rulează greu, iar eşantionul a posteriori nu arată foarte bine. Motivul pentru acest lucru este că s-a inclus în mod *explicit* variabila z în model, iar eşantionarea acesteia duce de obicei la o mixare lentă şi o explorare inefectivă a capetelor distribuţiei.

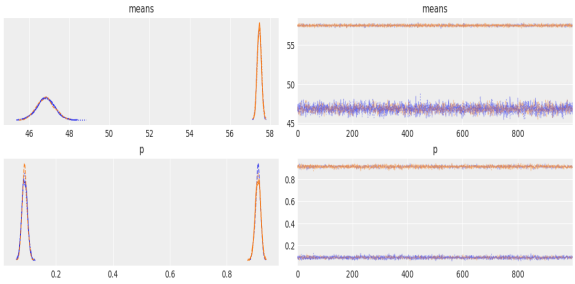
O posibilitate de a rezolva acest lucru este de a reparametriza modelul:

- Observăm că variabila observată depinde condiţional şi de variabila latentă z: $p(y \mid \theta, z)$.
- Am putea obţine de aici (ştiind că z este categorială) direct distribuţia marginală $p(y \mid \theta)$.
- Din fericire, acest lucru se realizează automat în PyMC, cu ajutorul distribuţiei `NormalMixture`:

```
clusters = 2
with pm.Model() as model_mg:
    p = pm.Dirichlet('p', a=np.ones(clusters))
    means = pm.Normal('means', mu=cs_exp.mean(), sigma=10, shape=clusters)
    sd = pm.HalfNormal('sigma', sigma=10)
    y = pm.NormalMixture('y', w=p, mu=means, sigma=sd, observed=cs_exp)
    idata_mg = pm.sample(random_seed=123, return_inferencedata=True)
```

Vizualizarea distribuţiei a posteriori:

```
varnames = ['means', 'p']
az.plot_trace(idata_mg, varnames)
```



sau `az.summary(idata_mg, varnames)` :

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
means[0]	49.494	4.618	46.176	57.665	2.292	1.755	7.0	31.0	1.53
means[1]	54.807	4.609	46.478	57.700	2.293	1.755	7.0	28.0	1.53
p[0]	0.295	0.355	0.074	0.919	0.177	0.135	7.0	30.0	1.53
p[1]	0.705	0.355	0.081	0.926	0.177	0.135	7.0	30.0	1.53

Ne-identifiabilitatea parametrilor

Observăm că pentru medii, ambele urmează distribuţii bimodale cu valori în jurul valorilor 47 şi 57.5. La fel pentru valorile lui p.

- Astfel, nu putem distinge între parametri, fenomen cunoscut în statistică drept *non-identifiabilitate a parametrilor*.
- Acest lucru se întâmplă din cauză că modelul rămâne acelaşi, indiferent că prima componentă are media 47, iar a doua are media 57.5 sau vice-versa.
- Dacă este posibil, modelul ar trebui să înlăture această problemă a non-identifiabilităţii, prin măcar una din metodele următoare:
  - forţarea componentelor să fie ordonate: de exemplu, prin ordonarea mediilor componentelor în ordine crecătoare;
  - prin folosirea distribuţiilor a priori informative.

În PyMC, o modalitate simplă de a forţa parametrii să fie ordonaţi este folosirea funcţiei `pm.potential()`.

- Un *potenţial* este un factor arbitrar ce se adaugă verosimilităţii, fără a adăuga o variabilă modelului.
- Diferenţa faţă de verosimilitate este că potenţialul nu depinde neapărat de date, spre deosebire de verosimilitate.
- Putem folosi un potenţial pentru a forţa o constrângere:
  - astfel, dacă acea constrângere este satisfăcută, adăugăm 0 verosimilităţii;
  - dacă nu, adăugăm un factor de $-\infty$.
- Astfel, modelul consideră imposibil ca parametrii să violeze constrângerea respectivă, în timp ce modelul rămâne neperturbat:

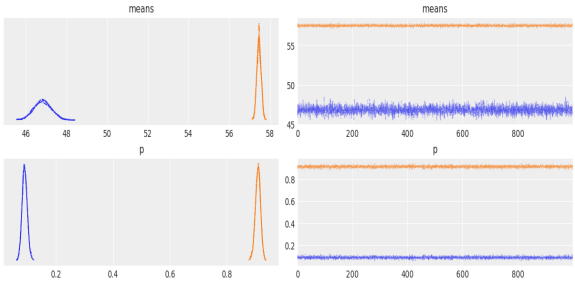
```
import pytensor.tensor as pt

clusters = 2
with pm.Model() as model_mgp:
    p = pm.Dirichlet('p', a=np.ones(clusters))
    means = pm.Normal('means', mu=np.array([.9, 1]) * cs_exp.mean(),
                      sigma=10, shape=clusters)
    sd = pm.HalfNormal('sd', sigma=10)
    order_means = pm.Potential('order_means',
                               pt.switch(means[1]-means[0] < 0,
                                         -np.inf, 0))
    y = pm.NormalMixture('y', w=p, mu=means, sigma=sd, observed=cs_exp)
    idata_mgp = pm.sample(1000, random_seed=123, return_inferencedata=True)
```

În loc de un potenţial se poate utiliza o transformare ordonată:

transform=pm.distributions.transforms.ordered

Comparaţie cu modelul anterior:

```
varnames = ['means', 'p']
az.plot_trace(idata_mgp, varnames)
```



sau `az.summary(idata_mgp, varnames)` :

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
means[0]	46.821	0.431	46.017	47.615	0.007	0.005	4029.0	3466.0	1.0
means[1]	57.463	0.103	57.276	57.654	0.002	0.001	4622.0	2892.0	1.0
p[0]	0.090	0.009	0.073	0.107	0.000	0.000	3813.0	2899.0	1.0
p[1]	0.910	0.009	0.893	0.927	0.000	0.000	3813.0	2899.0	1.0
sd	3.653	0.076	3.508	3.795	0.001	0.001	4517.0	3138.0	1.0

O altă constrângere care poate fi adăugată modelului este să ne asigurăm că toate componentele au o probabilitate nenulă, adică fiecare componentă a mixturii primeşte măcar o observaţie:

```
p_min = pm.Potential('p_min', pt.switch(tt.min(p) < min_p, -np.inf, 0))
```

Aici putem seta `p_min` la o valoare arbitrară, dar rezonabilă, ca 0.1 sau 0.01.

- Parametrul α al distribuţiei Dirichlet îi controlează concentraţia.
- O distribuţie plată pe simplex este obţinută cu $\alpha = (1, 1, \ldots, 1)$.
- Observaţii empirice au dus la concluzia că valori de 4 sau 10 sunt valori bune pentru componentele lui α, deoarece duc la distribuţii a posteriori ce au pe fiecare componentă măcar o observaţie, în timp ce reduc şansa de a supraestima numărul de componente.

Alegerea lui K

Una din principalele preocupări legate de modelele mixte finite este să decidem numărul de componente.

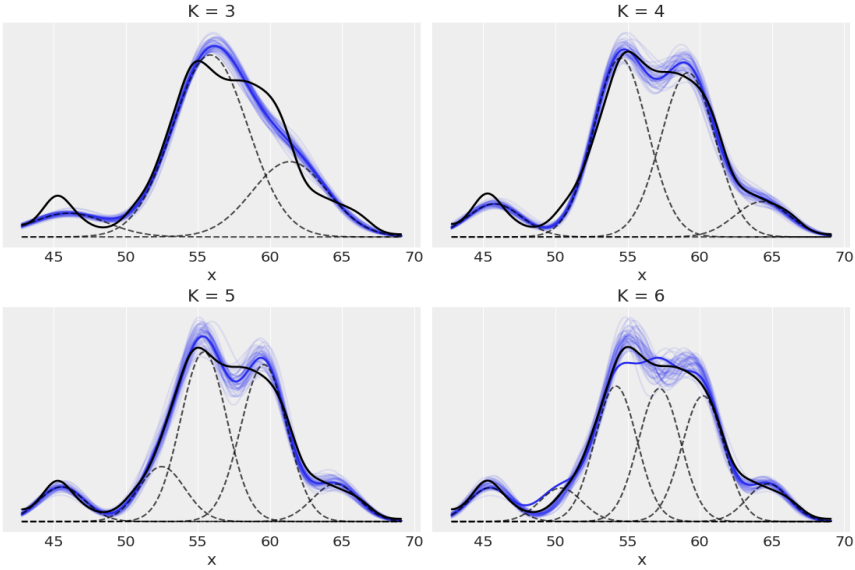
- O regulă de bază ar fi să începem cu un nr. relativ mic de componente, urmând să-l creştem pentru a îmbunătăţi calibrarea modelului.
- Aceasta se face folosind evaluări predictive a posteriori şi/sau măsuri precum WAIC sau LOO.

Să comparăm modelul pentru $K \in \{3, 4, 5, 6\}$.
Pentru aceasta, vom salva `trace` şi `model` pentru utilizări ulterioare:

```
clusters = [3, 4, 5, 6]

models = []
idatas = []
for cluster in clusters:
    with pm.Model() as model:
        p = pm.Dirichlet('p', a=np.ones(cluster))
        means = pm.Normal('means',
                          mu=np.linspace(cs_exp.min(), cs_exp.max(), cluster),
                          sigma=10, shape=cluster,
                          transform=pm.distributions.transforms.ordered)
        sd = pm.HalfNormal('sd', sigma=10)
        y = pm.NormalMixture('y', w=p, mu=means, sigma=sd, observed=cs_exp)
        idata = pm.sample(1000, tune=2000, target_accept=0.9, random_seed=123, return_inferencedata=True)
        idatas.append(idata)
        models.append(model)
```

Pentru a vedea mai bine cum afectează K inferenţa, vom compara calibrările acestor modele cu cea obţinută cu `az.plot_kde`:

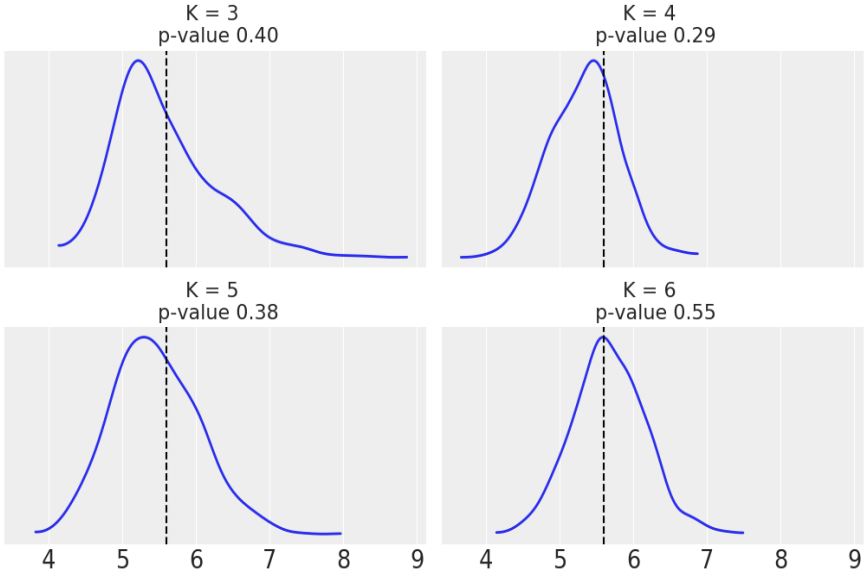
```
_, ax = plt.subplots(2, 2, figsize=(11, 8), constrained_layout=True)

ax = np.ravel(ax)
x = np.linspace(cs_exp.min(), cs_exp.max(), 200)
for idx, idata_x in enumerate(idatas):
    posterior_x = idata_x.posterior.stack(samples=("chain", "draw"))
    x_ = np.array([x] * clusters[idx]).T
    for i in range(50):
        i_ = np.random.randint(0, posterior_x.samples.size)
        means_y = posterior_x['means'][:,i_]
        p_y = posterior_x['p'][:,i_]
        sd = posterior_x['sd'][i_]
        dist = stats.norm(means_y, sd)
        ax[idx].plot(x, np.sum(dist.pdf(x_) * p_y.values, 1), 'C0', alpha=0.1)
    means_y = posterior_x['means'].mean("samples")
    p_y = posterior_x['p'].mean("samples")
    sd = posterior_x['sd'].mean()
    dist = stats.norm(means_y, sd)
    ax[idx].plot(x, np.sum(dist.pdf(x_) * p_y.values, 1), 'C0', lw=2)
    ax[idx].plot(x, dist.pdf(x_) * p_y.values, 'k--', alpha=0.7)
    az.plot_kde(cs_exp, plot_kwargs={'linewidth':2, 'color':'k'}, ax=ax[idx])
    ax[idx].set_title('K = {}'.format(clusters[idx]))
    ax[idx].set_yticks([])
    ax[idx].set_xlabel('x')
```



Se pare că alegerea $K = 3$ nu este prea potrivită, $K = 4, 5$ sau 6 fiind mai bune

Pentru comparare, am putea folosi verificările predictive a posteriori şi valoarea p din "Compararea modelelor":

```
ppc_mm = [pm.sample_posterior_predictive(idatas[i], 1000, models[i])
          for i in range(4)]

fig, ax = plt.subplots(2, 2, figsize=(10, 6), sharex=True, constrained_layout=True)
ax = np.ravel(ax)
def iqr(x, a=0):
    return np.subtract(*np.percentile(x, [75, 25], axis=a))

T_obs = iqr(cs_exp)
for idx, d_sim in enumerate(ppc_mm):
    T_sim = iqr(d_sim.posterior_predictive['y'][:100].T, 1)
    p_value = np.mean(T_sim >= T_obs)
    az.plot_kde(T_sim, ax=ax[idx])
    ax[idx].axvline(T_obs, 0, 1, color='k', ls='--')
    ax[idx].set_title(f'K = {clusters[idx]} \n p-value {p_value:.2f}')
    ax[idx].set_yticks([])
```



$K = 6$ este o alegere bună, deoarece valoarea p este cea mai apropiată de 0.5

După cum am văzut în cursul anterior, o alternativă este să folosim WAIC:

```
[pm.compute_log_likelihood(idatas[i], model=models[i]) for i in range(4)]
comp = az.compare(dict(zip([str(c) for c in clusters], idatas)),
                  method='BB-pseudo-BMA', ic="waic", scale="deviance")
comp
```

	rank	elpd_waic	p_waic	elpd_diff	weight	se	dse	warning	scale
6	0	10251.655876	12.320675	0.000000	9.125400e-01	63.173331	0.000000	False	deviance
5	1	10258.891454	10.126313	7.235578	8.503904e-02	61.382129	3.800239	False	deviance
4	2	10279.031416	7.561068	27.375540	2.420994e-03	60.642637	9.252588	False	deviance
3	3	10356.765531	5.856399	105.109656	6.481359e-13	60.631617	18.309559	False	deviance

Grafic: `az.plot_compare(comp)`