Modele liniare generalizate
Cursul 8

Programare şi modelare probabilistă - anul III

Facultatea de Informatică, UAIC

e-mail: adrian.zalinescu@uaic.ro

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

4 Decembrie 2023

Introducere

- În regresia liniară am folosit până acum combinaţii liniare ale variabilelor de input pentru a prezice media unei variabile de output.
- De asemenea, am presupus că cea din urmă urmează o distribuţie Gaussiană, cu excepţia cazurilor de inferenţă robustă, când am înlocuit-o cu o distribuţie de tip Student.
- În cele ce urmează, vom vedea mai multe exemple în care este bine să înlocuim distribuţia Gaussiană cu altele.

Plan

1. Regresia logistică
   - Setul de date Iris
   - Regresia logistică multiplă
2. Regresia softmax
3. Modele discriminative şi generative

Modele liniare generalizate

Aceste modele se bazează pe o idee simplă:

- pentru a prezice media unei variabile de output, putem aplica o funcţie arbitrară unei combinaţii liniare ale variabilelor de input:

$$\mu = f(\alpha + X\beta).$$

- Aici f este o funcţie, numită *funcţia de legătură inversă* (*inverse link function*).
- Există multe astfel de funcţii dintre care putem alege: cea mai simplă este funcţia identitate, care duce la regresia liniară clasică (simplă sau multiplă).
- O situaţie în care dorim să folosim altfel de funcţie de legătură inversă este cea în care lucrăm cu variabile *nominale* (*categoriale*), precum culorile, genurile, speciile sau afinităţile politice.
- Niciuna dintre aceste variabile nu este bine modelată de v.a. Gaussiene.

Regresia logistică

- Problemele de regresie sunt despre prezicerea unei valori continue pentru o variabilă de output, date fiind una sau mai multe variabile de input.
- Pe de altă parte, *clasificarea* este despre asignarea unei valori discrete unei variabile de input, date fiind variabilele de input.
- În ambele cazuri, sarcina noastră este de a propune un model care asociază corect variabilelor de input pe cea de output.
- Din punctul de vedere al Învăţării automate, atât regresia, cât şi clasificare fac parte din algoritmii de *învăţare supervizată*.

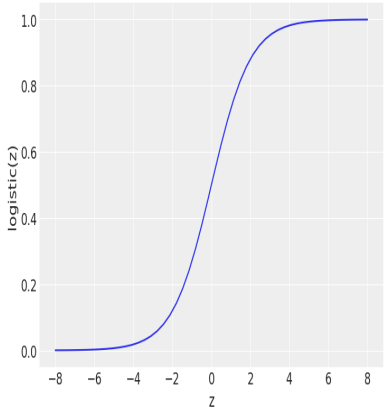
Regresia logistică este un model liniar generalizat prin folosirea funcţiei logistice ca funcţie de legătură inversă:

$$\text{logistic}(z) = \frac{1}{1 + e^{-z}}.$$

- Proprietatea esenţială a acestei funcţii este că valorile ei sunt întotdeauna cuprinse între 0 şi 1.
- Astfel, putem vedea această funcţie ca o modalitate convenabilă de a comprima valorile obţinute dintr-un model liniar în valori cu care putem hrăni (feed) o distribuţie Bernoulli.

- Datorită aspectului ei în formă de *S*, funcţia logistică se mai numeşte funcţia *sigmoid*:

```
import pymc as pm
import numpy as np
import pandas as pd
import pytensor as pt
import seaborn as sns
import scipy.stats as stats
from scipy.special import expit as logistic
import matplotlib.pyplot as plt
import arviz as az
az.style.use('arviz-darkgrid')

z = np.linspace(-8, 8)
plt.plot(z, 1 / (1 + np.exp(-z)))
plt.xlabel('z')
plt.ylabel('logistic(z)')
```



Să începem cu cazul în care avem doar două clase de instanţe, ca bun/rău, sigur/nesigur, noros/însorit, bolnav/sănătos, etc.

- Mai întâi, codăm aceste clase ca 0 şi 1 (eşec/succes), deci $y \in \{0, 1\}$ (unde y este variabila de output).
- Ca în modelul aruncării unei monede, vom folosi distribuţia Bernoulli pentru verosimilitate;
- diferenţa este că parametrul θ nu mai este generat dintr-o distribuţie Beta (a priori), ci dintr-un model liniar:

$$\begin{aligned} \theta &= \text{logistic}(\alpha + x\beta) \\ y &= \text{Bernoulli}(\theta) \end{aligned}$$

Setul de date Iris

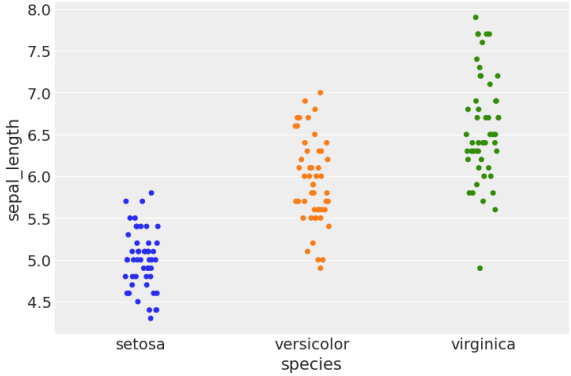
Acesta este un set clasic de date, ce cuprinde caracteristici ale trei tipuri de flori de Iris înrudite: *setosa*, *virginica* şi *versicolor*, anume lungimea şi lăţimea petalelor, lungimea şi lăţimea sepalelor (frunze modificate ale căror rol este să protejeze petalele):

```
iris = pd.read_csv('./data/iris.csv')
iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Pentru vizulizarea datelor în funcţie de `sepal_length` folosim funcţia `stripplot` din `seaborn`:

sns.stripplot(x="species", y="sepal_length", data=iris, jitter=True, hue="species", legend=False)

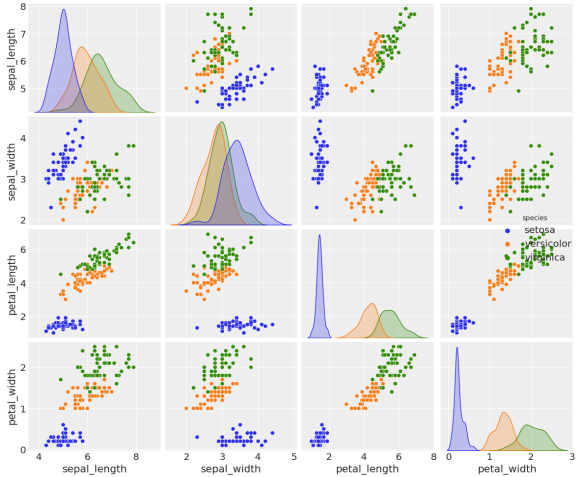


Observăm că axa y este continuă, în timp ce x este categorică – dispersia (`jitter`) de-a lungul axei Ox nu are niciun înţeles special: este folosită doar pentru o mai bună vizualizare.

Un alt mod de a inspecta datele este de a folosi matricea de tip `scatter` cu funcţia `pairplot`:

- datele vor fi aranjate într-un `grid` de tip 4×4, deoarece avem 4 caracteristici în acest set de date:

```
sns.pairplot(iris, hue='species', diag_kind='kde')
```



Modelul logistic aplicat setului de date Iris

Vom începe cu cea mai simplă problemă de clasificare, în care avem două clase, `setosa` şi `versicolor`, şi doar o singură variabilă de input (caracteristică), `sepal_length`.

- Cu `pandas`, codificăm cele două clase folosind 0 şi 1:

```
df = iris.query("species == ('setosa', 'versicolor')")
y_0 = pd.Categorical(df['species']).codes
x_n = 'sepal_length'
x_0 = df[x_n].values
x_c = x_0 - x_0.mean()
```

- Observăm că, la fel ca în cazul regresiei liniare, am centrat datele.

Modelul

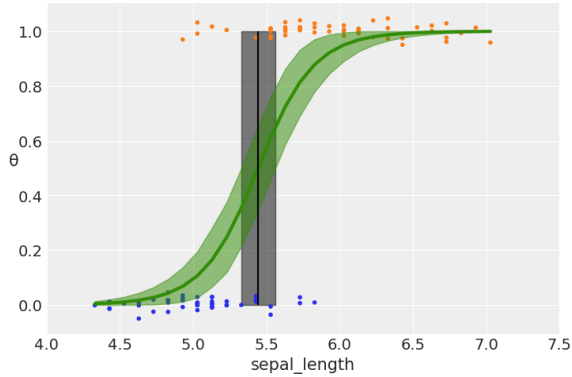
```
with pm.Model() as model_0:
    α = pm.Normal('α', mu=0, sigma=10)
    β = pm.Normal('β', mu=0, sigma=10)

    μ = α + pm.math.dot(x_c, β)
    θ = pm.Deterministic('θ', pm.math.sigmoid(μ))
    bd = pm.Deterministic('bd', -α/β)

    yl = pm.Bernoulli('yl', p=θ, observed=y_0)

    idata_0 = pm.sample(1000, return_inferencedata=True)
```

- θ este rezultatul funcţiei logistic aplicate variabilei μ;
- funcţia logistic este reprezentată aici de `pm.math.sigmoid`;
- bd este frontiera (boundary) domeniului de decizie, a cărei valoare este folosită pentru a separa clasele (vezi slide-urile următoare)

Vizualizarea datelor:

- O primă analiză a inferenţei, pe care o omitem, se face cu `summary` şi `plot_trace`.
- Vom vizualiza datele, împreună cu graficul funcţiei logistice şi frontiera de decizie:

```
posterior_0 = idata_0.posterior.stack(samples=("chain", "draw"))

theta = posterior_0['θ'].mean("samples")
idx = np.argsort(x_c)
plt.plot(x_c[idx], theta[idx], color='C2', lw=3)
plt.vlines(posterior_0['bd'].mean(), 0, 1, color='k')
bd_hpd = az.hdi(posterior_0['bd'].values)
plt.fill_betweenx([0, 1], bd_hpd[0], bd_hpd[1], color='k', alpha=0.5)

plt.scatter(x_c, np.random.normal(y_0, 0.02),
            marker='.', color=[f'C{x}' for x in y_0])
az.plot_hdi(x_c, posterior_0['θ'].T, color='C2', smooth=False)

plt.xlabel(x_n)
plt.ylabel('θ', rotation=0)
# use original scale for xticks
locs, _ = plt.xticks()
plt.xticks(locs, np.round(locs + x_0.mean(), 1))
```



- linia în formă de S reprezintă media lui θ: poate fi interpretată ca probabilitatea ca floarea să fie `versicolor` ştiind valoarea lungimii sepalei
- domeniul semitransparent în jurul acesteia reprezintă intervalul HDI 94%
- linia verticală este media valorii frontierei de decizie, cu intervalul HDI 94% reprezentat de banda semitransparentă din jur.

Frontiera de decizie

Este definită ca valoarea lui x (variabila de input) pentru care $\theta = 0.5$:

$$0.5 = \text{logistic}(\alpha + x\beta) \iff 0 = \alpha + x\beta \iff x = -\alpha/\beta.$$

Observaţii:

- Valoarea lui θ este interpretată ca $p(y = 1 \mid x)$.
- Astfel, θ modelează de asemenea media unei variabile dihotomice (de tip Bernoulli), de aceea $\theta \in [0, 1]$.
- Cu ajutorul acesteia, introducem o regulă pentru a converti această probabilitate într-o clasificare: dacă $\theta > 0.5$, atunci asignăm lui x valoarea 1, altfel îi asignăm valoarea 0.
- Valoarea de 0.5 nu este singura posibilă: în cazul acesta, ea este rezonabilă deoarece este acelaşi lucru pentru noi să clasificăm o `setosa` drept o `versicolor` ca o `versicolor` drept `setosa`.
- Dar în general, costul asociat erorii de clasificare nu este neapărat simetric, caz în care valoarea de 0.5 poate fi schimbată.

Regresia logistică multiplă

În mod similar regresiei liniare multiple, regresia logistică multiplă foloseşte mai multe variabile independente (de input).

- În exemplul anterior, să încercăm să combinăm lungimea sepalei cu lăţimea ei.
- Vom pre-procesa datele mai întâi:

```
df = iris.query("species == ('setosa', 'versicolor')")
y_1 = pd.Categorical(df['species']).codes
x_n = ['sepal_length', 'sepal_width']
x_1 = df[x_n].values
```

Înainte de a implementa modelul, vom deduce frontiera de decizie.
Avem

$$\theta = \text{logistic}(\alpha + \beta_1 x_1 + \beta_2 x_2)$$

Astfel, cu $\theta = 0.5$

$$0.5 = \text{logistic}(\alpha + \beta_1 x_1 + \beta_2 x_2) \Longleftrightarrow 0 = \alpha + \beta_1 x_1 + \beta_2 x_2,$$

de unde

$$x_2 = -\frac{\alpha}{\beta_2} + \left(-\frac{\beta_1}{\beta_2} x_1\right),$$

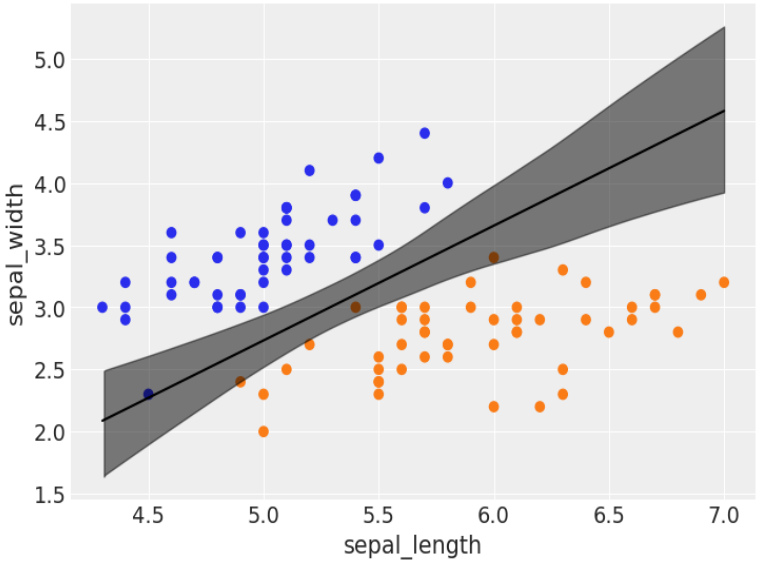
care reprezintă o dreaptă (cu $-\alpha/\beta_2$ intercepţia şi $-\beta_1/\beta_2$ panta ei).
Astfel, domeniile de decizie corespunzătoare celor două clase vor fi două semiplane separate printr-o dreaptă drept frontieră de decizie.

Implementarea modelului

```
with pm.Model() as model_1:
    α = pm.Normal('α', mu=0, sigma=10)
    β = pm.Normal('β', mu=0, sigma=2, shape=len(x_n))

    μ = α + pm.math.dot(x_1, β)
    θ = pm.Deterministic('θ', 1 / (1 + pm.math.exp(-μ)))
    bd = pm.Deterministic('bd', -α/β[1] - β[0]/β[1] * x_1[:,0])

    yl = pm.Bernoulli('yl', p=θ, observed=y_1)

    idata_1 = pm.sample(2000, return_inferencedata=True)
```

Ca mai înainte, vom vizualiza datele şi frontiera de decizie:

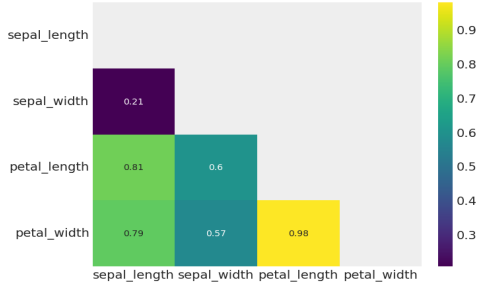
```
idx = np.argsort(x_1[:,0])
bd = idata_1.posterior['bd'].mean(("chain", "draw"))[idx]
plt.scatter(x_1[:,0], x_1[:,1], c=[f'C{x}' for x in y_0])
plt.plot(x_1[:,0][idx], bd, color='k');
az.plot_hdi(x_1[:,0], idata_1.posterior['bd'], color='k')
plt.xlabel(x_n[0])
plt.ylabel(x_n[1])
```


sepal_width
sepal_length
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
4.5
5.0
5.5
6.0
6.5
7.0

Variabile corelate

Dacă avem de a face cu variabile de input (puternic) corelate, acestea se traduc într-un model *nedeterminat*: datele sunt incapabile să restricţioneze parametrii modelului.
Să folosim setul de date Iris pentru a genera o "heatmap" a corelaţiilor:

```
corr = iris[iris['species'] != 'virginica'].corr(numeric_only=False)
mask = np.tri(*corr.shape).T
sns.heatmap(corr.abs(), mask=mask, annot=True, cmap='viridis')
plt.show()
```



- O soluţie atunci când avem de a face cu variabile corelate este să ştergem una (sau mai multe) dintre acestea.
- O alta este să adăugăm mai multă informaţie în distribuţia a priori:
  - astfel, A. Gelman şi Stan Team recomandă (`https://github.com/stan-dev/stan/wiki/Prior-Choice-Recommendations`) centrarea tuturor variabilelor non-binare şi apoi utilizarea

$$\beta \sim \text{Student}(0, \nu, \text{sd});$$

  - aici sd trebuie să ne informeze (slab) asupra valorilor variabilelor centrate, iar
  - parametrul de normalitate ν ar trebui să fie între 3 şi 7.
  - Aceasta spune că ne aşteptăm ca acest coeficient să fie mic, dar utilizăm cozi groase deoarece câteodată vom găsi coeficienţi largi, datorită faptului că distribuţia Student are probabilităţi extreme mai mari (heavier tails) decât cea normală.

Clase nebalansate

Setul de date Iris este complet balansat, în sensul că fiecare categorie are exact 50 de observaţii.

- Multe alte seturi de date sunt constituite din date nebalansate, adică sunt mai multe date dintr-o clasă decât din cealaltă.
- Din această cauză, regresia logistică poate întâmpina dificultăţi, în sensul că graniţa de decizie nu mai poate fi determinată atât de precis.

Exemplu: îndepărtăm mai multe date din clasa `setosa`:

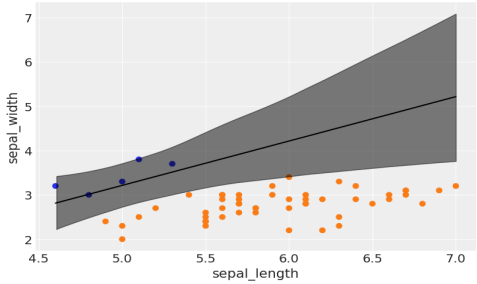
```
df = iris.query("species == ('setosa', 'versicolor')")
df = df[45:]
y_3 = pd.Categorical(df['species']).codes
x_n = ['sepal_length', 'sepal_width']
x_3 = df[x_n].values
```

Apoi, aplicăm regresia logistică multiplă:

```
df = iris.query("species == ('setosa', 'versicolor')")
df = df[45:]
y_3 = pd.Categorical(df['species']).codes
x_n = ['sepal_length', 'sepal_width']
x_3 = df[x_n].values
```

şi vizualizăm graniţa de decizie:

```
idx = np.argsort(x_3[:,0])
bd = idata_3.posterior['bd'].mean(("chain", "draw"))[idx]
plt.scatter(x_3[:,0], x_3[:,1], c= [f'C{x}' for x in y_3])
plt.plot(x_3[:,0][idx], bd, color='k')

az.plot_hdi(x_3[:,0], idata_3.posterior['bd'], color='k')

plt.xlabel(x_n[0])
plt.ylabel(x_n[1])
```



- Observăm că graniţa de decizie s-a mutat spre clasa cea mai mică, iar incertitudinea a devenit mai mare.
- Astfel, dacă nu putem să introducem date balansate, trebuie să fim mult mai atenţi când interpretăm datele (verificarea incertitudinii, verificarea distribuţiei a posteriori predictive).
- O altă soluţie este să impunem mai mult control asupra distribuţiei a priori, dacă este disponibilă informaţie suplimentară.

Regresia softmax

O posibilitate de a generaliza regresia logistică la mai mult de două clase este cu *regresia softmax*. Sunt operate două schimbări faţă de regresia logistică:

1. se înlocuieşte funcţia logistică cu funcţia softmax:

$$\text{softmax}_i(\mu_1, \ldots, \mu_n) = \frac{\exp(\mu_i)}{\sum_{k=1}^{n} \exp(\mu_k)}.$$

   - pentru $n = 2$, softmax înlocuieşte funcţia logistic:

$$\text{softmax}_1(\mu_1, \mu_2) = \frac{e^{\mu_1}}{e^{\mu_1} + e^{\mu_2}} = \text{logistic}(\mu_1 - \mu_2)$$

2. se înlocuieşte distribuţia Bernoulli cu *distribuţia categorială* (exemplu: aruncarea unui zar în loc de cea a unei monede).

Pentru exemplificarea regresiei softmax, vom continua să lucrăm cu setul de date Iris, dar vom utiliza toate cele 3 clase ale sale (`setosa`, `versicolor` şi `virginica`) şi cele 4 caracteristici.
De asemenea, vom standardiza datele:

```
iris = sns.load_dataset('iris')
y_s = pd.Categorical(iris['species']).codes
x_n = iris.columns[:-1]
x_s = iris[x_n].values
x_s = (x_s - x_s.mean(axis=0)) / x_s.std(axis=0)
```

Modelul

```
with pm.Model() as model_s:
    α = pm.Normal('α', mu=0, sigma=5, shape=3)
    β = pm.Normal('β', mu=0, sigma=5, shape=(4,3))
    μ = pm.Deterministic('μ', α + pm.math.dot(x_s, β))
    θ = pm.math.softmax(μ, axis=1)
    yl = pm.Categorical('yl', p=θ, observed=y_s)
    idata_s = pm.sample(2000, target_accept=0.9, return_inferencedata=True)
```

Cât de bine s-a comportat modelul? În codul de mai jos vom folosi mediile parametrilor estimaţi:

- pentru a calcula probabilitatea ca fiecare intrare să aparţină uneia dintre cele trei clase,
- iar apoi să asignăm o clasă folosind funcţia argmax,
- şi în cele din urmă să comparăm rezultatul cu valorile observate.

```
data_pred = idata_s.posterior['μ'].mean(("chain", "draw"))
y_pred = [np.exp(point)/np.sum(np.exp(point), axis=0)
          for point in data_pred]
f'{np.sum(y_s == np.argmax(y_pred, axis=1)) / len(y_s):.2f}'
```

▷ `'0.98'`

Observăm că 98% dintre valori sunt asignate corect (doar 3 erori), ceea ce este foarte bine. Totuşi, un test adevărat ar trebui efectuat pe date care nu au fost folosite pentru calibrarea (fit) modelului.

- O problemă a modelului de mai sus este că distribuţiile marginale ale fiecărui parametru sunt prea largi.
- Acest lucru este datorat faptului că suma tuturor probabilităţilor μ_i este 1, deci avem o problemă de puternică corelare a coeficienţilor.
- Acest lucru se poate corija prin îndepărtarea unui parametru, de exemplu prin fixarea acestuia la o anumită valoare, de exemplu 0:

```
with pm.Model() as model_sf:
    α = pm.Normal('α', mu=0, sigma=2, shape=2)
    β = pm.Normal('β', mu=0, sigma=2, shape=(4,2))
    α_f = pt.concatenate([[0] ,α])
    β_f = pt.concatenate([np.zeros((4,1)) , β], axis=1)
    μ = α_f + pm.math.dot(x_s, β_f)
    θ = pm.math.softmax(μ, axis=1)
    yl = pm.Categorical('yl', p=θ, observed=y_s)
    idata_sf = pm.sample()
```

Modele discriminative şi generative

După cum am văzut, în regresia logistică şi generalizările ei, am încercat să calculăm direct $p(y \mid x)$, adică probabilitatea de a aparţine unei clase, ştiind caracteristica (-ile) x. Cu alte cuvinte:

- am construit modelul prin corelarea directă a lui y în funcţie de x,
- iar apoi am folosit un *prag* pentru a transforma probabilitatea calculată într-o frontieră discretă.

O altă abordare:

- modelăm mai întâi $p(x \mid y)$, adică distribuţia lui x pentru fiecare clasă,
- iar apoi facem asignarea la clase.

Acest tip de model se numeşte *generativ*, deoarece creăm un model din care *generăm* eşantioane pentru fiecare clasă.
În schimb, regresia logistică este *discriminativă*, deoarece nu putem genera exemple din fiecare clasă a modelului.

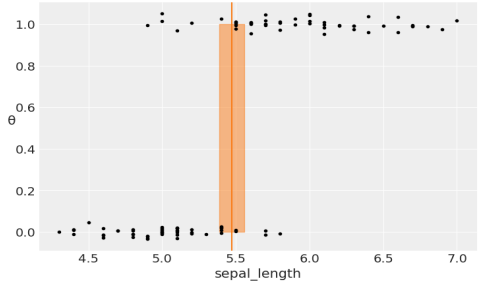
Presupunerea făcută în modelul de mai jos se numeşte *LDA* (*linear discriminant analysis*), care este un model generativ.

```
with pm.Model() as lda:
    μ = pm.Normal('μ', mu=0, sd=10, shape=2)
    σ = pm.HalfNormal('σ', 10)
    setosa = pm.Normal('setosa', mu=μ[0], sd=σ, observed=x_0[:50])
    versicolor = pm.Normal('versicolor', mu=μ[1], sd=σ,
                           observed=x_0[50:])
    bd = pm.Deterministic('bd', (μ[0] + μ[1]) / 2)
    idata_lda = pm.sample(1000, return_inferencedata=True)
```

Vedem că frontiera de decizie este definită ca media dintre mediile Gaussiene ale celor două clase.

Vizualizarea datelor şi a inferenţei:

```
posterior_lda = idata_lda.posterior.stack(samples=("chain", "draw"))
plt.axvline(posterior_lda['bd'].mean(), ymax=1, color='C1')
bd_hpd = az.hdi(posterior_lda['bd'].values)
plt.fill_betweenx([0, 1], bd_hpd[0], bd_hpd[1], color='C1', alpha=0.5)
plt.plot(x_0, np.random.normal(y_0, 0.02), '.', color='k')
plt.ylabel('θ', rotation=0)
plt.xlabel('sepal_length')
```



Comparând rezultatele obţinute cu regresia logistică şi cu LDA, observăm că rezultatele sunt similare.

- Modelul LDA poate fi extins la mai mult de o caracteristică modelând clasele ca fiind distribuţii Gaussiene multivariate.
- De asemenea, este posibil să se relaxeze condiţia ca aceste clase să partajeze o varianţă comună, dând naştere unui model cunoscut drept *QDA* (*quadratic linear discriminant*).
- În general, modelele LDA şi QDA merg mai bine decât regresia logistică în cazul în care caracteristicile folosite sunt Gaussiene (aproximativ);
- în caz contrar, regresia logistică dă rezultate mai bune.