Inteligencia Artificial Generativa - Obligatorio 1

Matias Molinolo, Martina Diaz

Facultad de Ingeniería, Universidad ORT Uruguay

3 de octubre de 2023

https://github.com/matiasmolinolo/iag-2023

Índice

1. Estimación de distribuciones

1.1. Dataset 1: Tennis

El objetivo fue estimar distribuciones de probabilidad a partir de un dataset 'Tennis' con datos discretos. Inicialmente se calculó la probabilidad conjunta $p(x, y)$ y las probabilidades condicionales $p(y|x)$ y $p(x|y)$, siendo x la variable discreta 'Outlook' e y la variable booleana 'Tennis'.

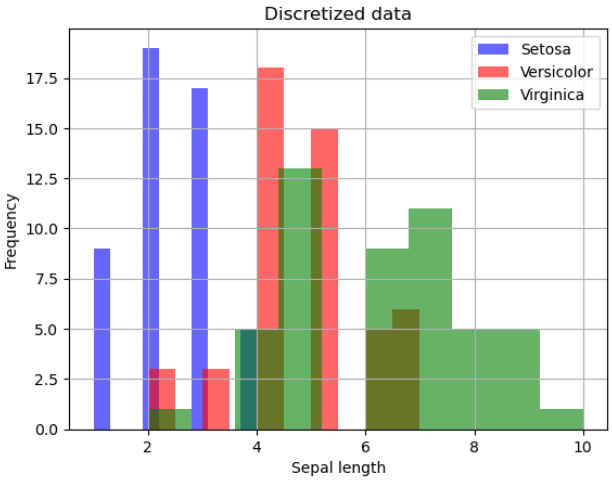
En segundo lugar buscamos aproximar $p(y, o, h, w, t)$ a través de la ecuación $p(y, o, h, w, t) = p(y) \times p(o|y) \times p(h|y, o) \times p(w|y, o) \times p(t|y, o, h, w)$. Para ello agrupamos las columnas necesarias del dataset y contabilizamos las ocurrencias de cada valor utilizando las funciones `groupby` y `size` de pandas DataFrame. Como resultado, observamos una probabilidad uniforme para las combinaciones de (y, o, h, w, t) que aparecen en el dataset, lo cual se debe a que los datos provistos contienen como máximo una única ocurrencia cada combinación, por lo que la probabilidad de todos los casos posibles es 0 o 0,071429.

Outlook	**Tennis**	**Humidity**	**Wind**	**Temp**	**Probability**
Overcast	Yes	High	Strong	Mild	0.071429
			Weak	Hot	0.071429
		Normal	Strong	Cool	0.071429
			Weak	Hot	0.071429
Rain	No	High	Strong	Mild	0.071429
		Normal	Strong	Cool	0.071429
	Yes	High	Weak	Mild	0.071429
		Normal	Weak	Cool	0.071429
				Mild	0.071429
Sunny	No	High	Strong	Hot	0.071429
			Weak	Hot	0.071429
				Mild	0.071429
	Yes	Normal	Strong	Mild	0.071429
			Weak	Cool	0.071429

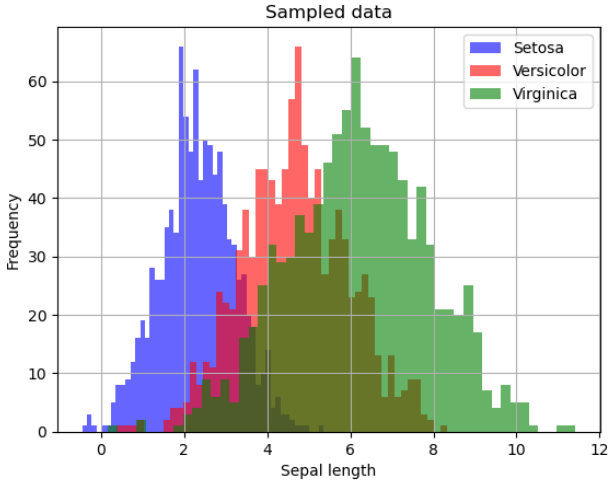
Cuadro 1: Probabilidad conjunta $p(y, o, h, w, t)$

1.2. Dataset 2: Iris

En este dataset de sklearn buscamos estimar distribuciones de probabilidad continuas. Para calcular las probabilidades $p(y|x)$ y $p(x|y)$, siendo y una variable categórica y x la variable continua 'sepal length', lo que hicimos fue discretizar los datos de X en 10 bins entre 4,3 y 7,9. A partir de los mismos calculamos la frecuencia de cada bin para cada clase de y, lo que nos permitió calcular las probabilidades. Esto lo hicimos a partir de histogramas, además de ser una ayuda gráfica para observar la distribución de la frecuencia de los datos, también nos permite obtener los valores de esa distribución. Además, utilizamos numpy para obtener la media y la desviación estándar, esto nos permitió muestrear datos y comprobar que la distribución de probabilidades era una distribución normal.



(a) Histograma de datos discretos



(b) Histograma de sampleo de datos

Por otra parte, utilizamos también un modelo de tipo mezcla de gaussianas para obtener la probabilidad de x, donde la mezcla de gaussianas se calcula de la siguiente manera:

$$p(x) = \sum_{y} p(y) \times \mathcal{N}(\mu_y, \sigma_y)$$

Siendo μ la media y σ la desviación estándar ya calculadas, muestreamos una clase y equiprobable y dentro de la misma muestreamos X con distribución normal.

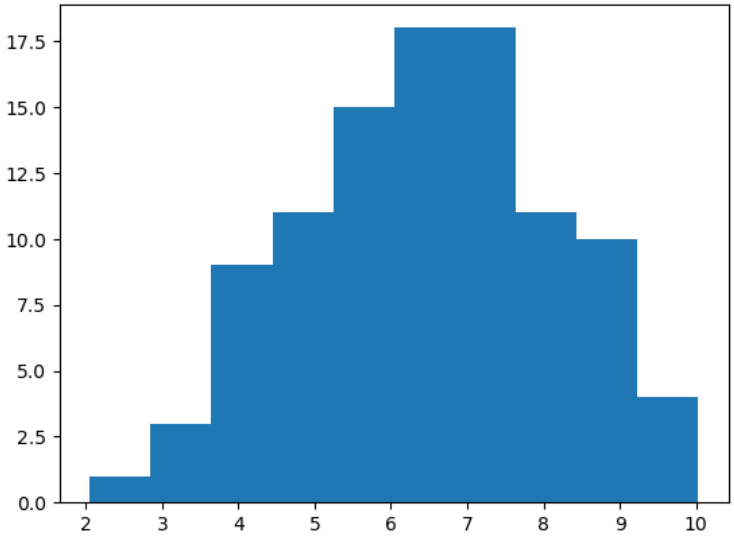


Figura 2: Mezcla de gaussianas

En conclusión, como resultado obtuvimos tres normales distintas, cada una con una media y desviación estándar distinta, que componen tres distribuciones de probabilidades distintas.

Setosa	Versicolor	Virginica
1.000000	0.000000	0.000000
0.826087	0.130435	0.043478
0.850000	0.150000	0.000000
0.178571	0.642857	0.178571
0.000000	0.535714	0.464286
0.000000	0.357143	0.642857
0.000000	0.352941	0.647059
0.000000	0.000000	1.000000
0.000000	0.000000	1.000000
0.000000	0.000000	1.000000

Cuadro 2: Probabilidad condicional $p(y|x)$

Setosa	Versicolor	Virginica
0.18	0.00	0.00
0.38	0.06	0.02
0.34	0.06	0.00
0.10	0.36	0.10
0.00	0.30	0.26
0.00	0.10	0.18
0.00	0.12	0.22
0.00	0.00	0.10
0.00	0.00	0.10
0.00	0.00	0.02

Cuadro 3: Probabilidad condicional $p(x|y)$

2. Modelos generativos

2.1. NADE: Neural Autoregressive Distribution Estimator

Intentamos reproducir en Pytorch la arquitectura del NADE propuesta por [1], con ciertas modificaciones que resultaron de la prueba y error de la arquitectura.

Uria et al. propone una arquitectura autorregresiva que parte de la base que cualquier distribución D-dimensional p puede expresarse como un producto de distribuciones unidimensionales [1]:

$$p(\boldsymbol{x}) = \prod_{d=1}^{D} p(x_{o_d}|\boldsymbol{x}_{o_{<d}})$$

Para ello, nos basamos en el siguiente repositorio de Github: https://github.com/simonjisu/NADE-pytorch/blob/master/model.py, pero con ciertas modificaciones para la optimización de los cálculos.

Nuestra arquitectura usa 192 unidades ocultas, en vez de las 500 propuestas en el paper original. Esto se debe a que tras varios ensayos, encontramos que este número daba una buena performance y un tiempo de ejecución significativamente menor al de la arquitectura original del paper.

El NADE fue entrenado con datos binarizados del dataset MNIST, donde se filtraron todos aquellos elementos que no pertenecían a la clase que quería generarse.

Podemos ver el código de la función `forward` del NADE a continuación:

```
a = self.c.expand(x.shape[0], -1)

p_x_hat = []

for d in range(self.input_dim):
    h_d = torch.sigmoid(a)
    p = torch.sigmoid(h_d @ self.V[d:d+1, :].t() + self.b[d:d+1])
    p_x_hat.append(p)
    a = x[:, d:d+1] @ self.W[:, d:d+1].t() + a

probs = torch.cat(p_x_hat, 1)
return probs
```

Luego de entrenarlo por 50 epochs, donde alcanzamos una loss de 0,29, hicimos un muestreo de la distribución estimada por el NADE con el siguiente código:

```
with torch.no_grad():
preds = torch.zeros(1, input_dim).to(device)
for i in tqdm(range(input_dim)):
    p = model.forward(preds)
    preds[0][i] = torch.bernoulli(p[0][i])

return torch.reshape(preds.cpu(), (28, 28))
```

Donde obtuvimos el siguiente resultado:

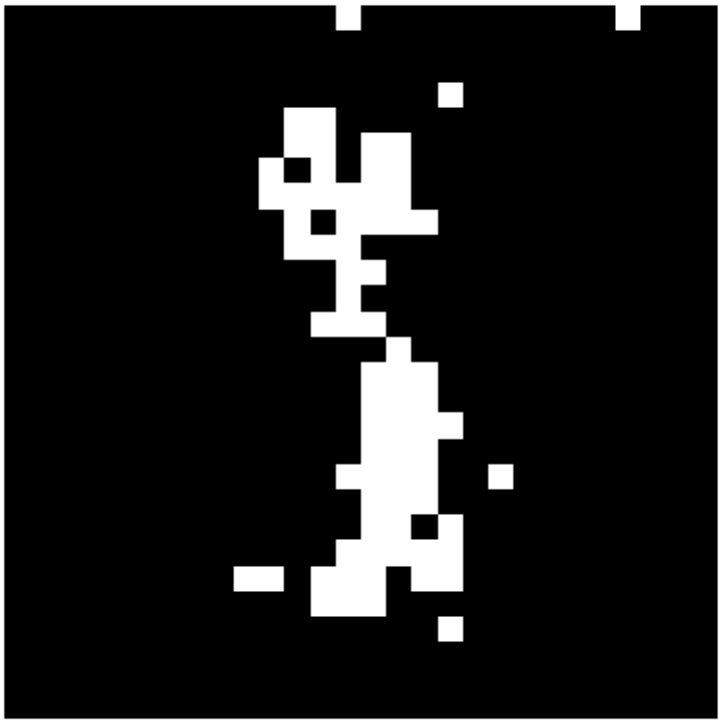


Figura 3: Distribución estimada por el NADE

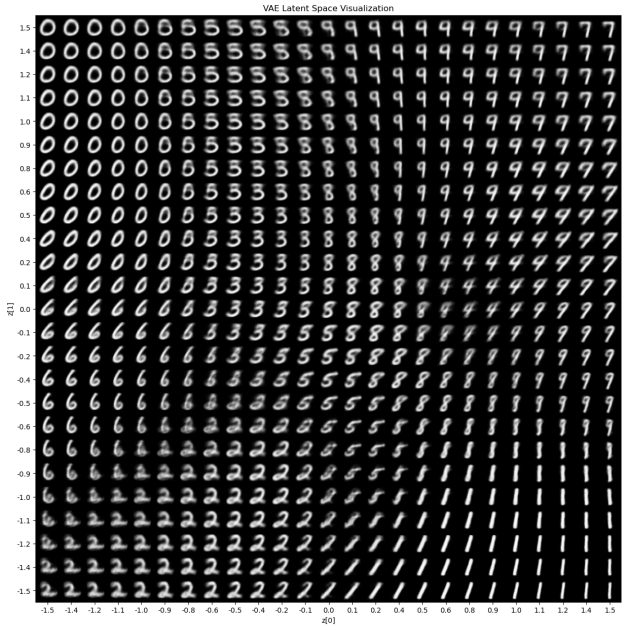
Podemos ver que no es una representación totalmente acertada, pero se puede apreciar claramente que se asemeja a la clase original de los 1. Estimamos que esto puede deberse a falta de entrenamiento u otros factores.

2.2. VAE: Variational Autoencoder

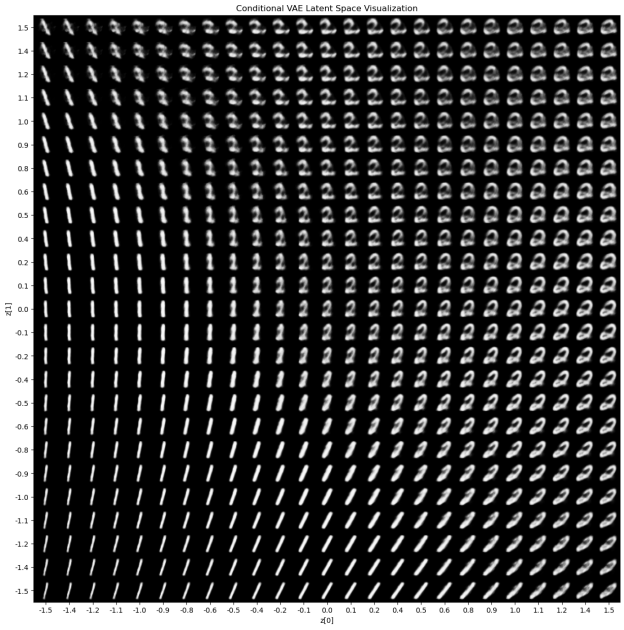
A partir del tutorial de Doersch [2], entrenamos un VAE y un VAE condicional, es decir, que genera una nueva imagen dada una clase específica.

Nuevamente, entrenamos esta red con el dataset del MNIST, pero esta vez sin filtrar ni binarizar.

Es interesante visualizar el espacio latente del VAE y el VAE condicional, que podemos observar en las siguientes figuras:



(a) Espacio latente del VAE



(b) Espacio latente del VAE condicional

Podemos apreciar como en el espacio latente del VAE se generan nuevas imágenes que se asemejan a todas las clases del dataset, mientras que en el espacio latente del VAE condicional se generan nuevas imágenes acotadas a la clase que indicamos como condicional.

A diferencia del NADE, tanto el entrenamiento como el sampling del VAE es mucho más rápido y genera mejores resultados en menos epochs - esto se debe al carácter autorregresivo del NADE, que hace imposible su paralelización, a diferencia del VAE que si es paralelizable.

Referencias

[1] B. Uria, M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle, "Neural autoregressive distribution estimation," 2016.

[2] C. Doersch, "Tutorial on variational autoencoders," 2021.