به نام خدا

داده کاوی و یادگیری ماشین

گزارش تمرین اول

استاد درس :

جناب آقای دکتر زارع

دستیاران استاد :

سرکار خانم حسنی

آقای داوردوست

آقای اسماعیلی

حسین رفیعی زاده

830400027

فرض میکنیم X_1، X_2 دو متغیر تصادفی باشند که مقادیر $\{0,1\}$ میگیرند و $P(X_i=0)=P(X_i=1)=\frac{1}{2}$

بازی $i=1,2$ و X_3 را $X_1 \oplus X_2$ تعریف میکنیم

X_1, X_2, X_3 mutually indpen نیستند چون که X_3 تابع قطعی X_1، X_2 هست

X_1, X_2, X_3 pairwise independent هستند چون که X_1, X_2 از لحاظ ساختاری مستقل هستند

- برای X_1, X_3 داریم: $a,b \in \{0,1\}$

$$P(X_1=a, X_3=b) = P(X_1=a, X_1 \oplus X_2 = b)$$
$$= P(X_1=a, X_2 \oplus a = b)$$
$$= P(X_1=a, X_2 = a \oplus b)$$
$$= P(X_1=a)\, P(X_2 = a \oplus b)$$
$$= \frac{1}{2} \cdot \frac{1}{2}$$

- در اینجا X_1, X_2 هر دو مستقل و متغیرهای تصادفی برنولی هستند و همچنین $a \in \{0,1\}$

$$P(X_3 = a)$$
$$= P(X_1 \oplus X_2 = a)$$
$$= P(X_1 \oplus X_2 = a, X_1 = X_2) + P(X_1 \oplus X_2 = a, X_1 \neq X_2)$$
$$= P(X_1 = X_2)\, P(X_1 \oplus X_2 = a \mid X_1 = X_2) + P(X_1 \neq X_2)\, P(X_1 \oplus X_2 = a \mid X_1 \neq X_2)$$
$$= \frac{1}{2} \cdot 1_{\{a=0\}} + \frac{1}{2} \cdot 1_{\{a=1\}}$$

- بنابراین X_3 یک متغیر تصادفی برنولی $(\frac{1}{2})$ میباشد بنابراین میتوانیم بنویسیم:

$$P(X_1=a, X_3=b) = \frac{1}{2} \cdot \frac{1}{2} = P(X_1=a)\, P(X_3=b)$$

بنابراین:

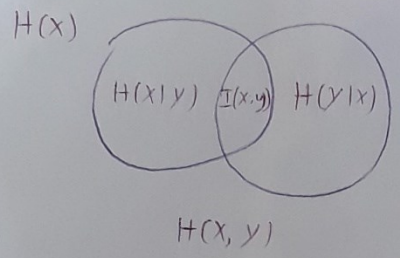
- برای همه‌ی $a,b \in \{0,1\}$: X_1، X_3 مستقل هستند و از نظر تقارن X_2، X_3 هم مستقل هستند

بنابراین X_1, X_2, X_3، به صورت جفتی (pairwise independent) هستند

سوال اول i) 2.7

در نظریه احتمالات mutual information بین دو متغیر تصادفی معیاری برای نشان دادن میزان وابستگی آن دو متغیر می‌باشد، مفهوم mutual information ذاتاً مرتبط با آنتروپی یک متغیر تصادفی که میزان اطلاعات موجود در یک متغیر تصادفی را نشان دهد.

- اطلاعات متقابل میزان شباهت بین توزیع مشترک $P(X,Y)$ و ضرب احتمال‌های حاشیه‌ای یعنی $P(X)P(Y)$ را مشخص می‌سازد.



- این نمودار من رابطه معیارهای اطلاعاتی مختلف متغیرهای تصادفی X, Y را نشان دهد.

- اطلاعات متقابل بین دو متغیر تصادفی X, Y را به صورت روبرو تعریف می‌کنیم:

$$I(X;Y) = \sum_{y\in Y}\sum_{x\in X} p(x,y)\log\left(\frac{p(x,y)}{p(x)\,p(y)}\right)$$

- در رابطه فوق $p(x,y)$ تابع توزیع احتمال مشترک X, Y، $p(x)$، $p(y)$ تابع های توزیع احتمال حاشیه‌ای به ترتیب X, Y می‌باشد.

- رابطه mutual information تحت ~~رابطه~~ زیر قابل بیان می‌باشد:

$$I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

- در این رابطه $H(X)$, $H(Y)$ آنتروپی‌های حاشیه‌ای، $H(X|Y), H(Y|X)$ آنتروپی های شرطی می‌باشند.

- حالا در این جا اثبات رابطه $I(X;Y) = H(Y) - H(Y|X)$ را نشان می‌دهیم:

$$I(X;Y) = \sum_{x,y} p(x,y)\log\frac{p(x,y)}{p(x)p(y)} = \sum_{x,y} p(x,y)\log\frac{p(x,y)}{p(x)} - \sum_{x,y} p(x,y)\log p(y)$$

$$= \sum_{x,y} p(x)p(y|x)\log p(y|x) - \sum_{x,y} p(x,y)\log p(y) = \sum_{x} p(x)\left(\sum_{y} p(y|x)\log p(y|x)\right) - \sum_{y}\log p(y)\left(\sum_{x} p(x,y)\right) = -\sum_{x} p(x)\,H(Y|X=x) - \sum_{y}\log p(y)\,p(y)$$

$$= -H(Y|X) + H(Y) = H(Y) - H(Y|X)$$

2.12 (i

• توزیع بتا، توزیع احتمال پیوسته‌ای است که برد آن $[0,1]$ تعریف می‌گردد. توزیع بتا دارای دو پارامتر آزاد هست که این دو پارامتر α, β هستند

• محاسبه توزیع بتا:

$$f'(x) = \frac{df(x)}{dx} = \frac{1}{B(a,b)} \frac{d}{dx} x^{\alpha-1}(1-x)^{b-1}$$

$$= \frac{1}{B(a,b)} \left[(\alpha-1) x^{\alpha-2}(1-x)^{b-1} - x^{\alpha-1}(b-1)(1-x)^{b-2} \right]$$

$$= \frac{1}{B(a,b)} x^{\alpha-2}(1-x)^{b-2} \left[(\alpha-1)(1-x) - (b-1)x \right]$$

$$f'(x) = 0 \Rightarrow (\alpha-1)(1-x) - (b-1)x = 0 \Rightarrow \alpha - 1 - x(\alpha - 1 + b - 1) = 0$$

$$\Rightarrow \boxed{x = \frac{\alpha-1}{\alpha+b-2}} \Rightarrow$$ مود توزیع بتا برابر با $\frac{\alpha-1}{\alpha+b-2}$ می‌باشد زمانی که $b>1, \alpha>1$ باشد

توزیع بتا $B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$

• محاسبه میانگین:

$$\mu = E[X] = \frac{\int_0^1 x^{\alpha}(1-x)^{\beta-1} dx}{B(\alpha,\beta)} = \frac{B(\alpha+1,\beta)}{B(\alpha,\beta)} = \frac{\Gamma(\alpha+1)\Gamma(\beta)\Gamma(\alpha+\beta)}{\Gamma(\alpha+\beta+1)\Gamma(\alpha)\Gamma(\beta)}$$

$$\boxed{= \frac{\alpha}{\alpha+\beta}} = \text{mean} = E[x]$$

• محاسبه واریانس:

$$\sigma^2 + \mu^2 = E[X^2] = \frac{B(\alpha+2,\beta)}{B(\alpha,\beta)} = \frac{\alpha(\alpha+1)}{(\alpha+\beta)(\alpha+\beta+1)}$$

$$\sigma^2 = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

2.16 (i

$$f(x) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}$$

- k را برابر یک قرار دهیم $(k=1)$:

$$f(x) = \frac{1}{\lambda}\underbrace{\left(\frac{x}{\lambda}\right)^{0}}_{1} e^{-\frac{x}{\lambda}} = \frac{1}{\lambda} * e^{-\frac{x}{\lambda}}$$

- زمانی که ما k را برابر یک قرار دادیم در توزیع Wicbull توزیع ما تبدیل به توزیع exponential میشود

$$f(x|\lambda) = \begin{cases} \lambda e^{-\lambda x} & \text{for } x > 0 \\ 0 & \text{for } x \leq 0 \end{cases}$$

- ((در این توزیع جایی که $\lambda > 0$ میباشد ما این رو rate of distrbution مینامیم))

توزیع exponential برای مدل سازی فرایندی که برخ داده است مناسب میباشد ، مواقعی که فرایند ما پیوسته و با نرخ ثابتی استفاده میشود . که میانگین توزیع به صورت زیر به دست محاسبه میشود :

$$E[X] = \int_0^{\infty} x\,\lambda e^{-\lambda x}\,dx = \lambda\left[\frac{-x e^{-\lambda x}}{\lambda}\Big|_0^{\infty} + \frac{1}{\lambda}\int_0^{\infty} e^{-\lambda x}\,dx\right]$$

$$= \lambda\left[0 + \frac{1}{\lambda}\,\frac{-e^{-\lambda x}}{\lambda}\Big|_0^{\infty}\right]$$

$$= \lambda\,\frac{1}{\lambda^2} = \frac{1}{\lambda}$$

- ~~k برابر در قرار دهیم $f(x;k)$~~ :

~~$f(x;\lambda) = \frac{\lambda}{\lambda}\left(\frac{x}{\lambda}\right)^{1} e^{-\left(\frac{x}{\lambda}\right)^{\lambda}}$~~

$$f_{X_1, X_2, \ldots, X_n}(x_1, x_2, \ldots, x_n, \frac{1}{\lambda}) = \prod_{i=1}^{n} f(\lambda, x_i)$$

- $X_1, X_2, \ldots, X_n$ ها مستقل و هم توزیع میباشند اگر تابع احتمال توام این نمونه های تصادفی با حاصل ضرب تابع احتمال آن ها برابر باشد. (در واقع تابع احتمال آن ها برابر است با حاصل ضرب چگالی آن ها!)

$$L\left(x_1, x_2, \dots, x_n, \frac{1}{\lambda}\right) = \prod_{i=1}^{n} \frac{1}{\lambda} e^{\frac{-x_i}{\lambda}}$$

$$= \prod_{i=1}^{n} \frac{1}{\lambda} \exp\left(-\frac{1}{\lambda} x_i\right)$$

$$= \left(\frac{1}{\lambda}\right)^n \exp\left(-\frac{1}{\lambda} \sum_{i=1}^{n} x_i\right)$$

- از آنجایی که هدف پیدا کردن بیشینه Likelihood است، می‌توان از Ln تابع Likelihood بیشینه‌سازی استفاده کرد چون که Ln یک تابع یکنوا هست.

- از هر دو طرف عبارت Ln میگیریم:

$$Ln\left(Likelihood\left(\frac{1}{\lambda}, x_1, \dots, x_n\right)\right) = n\, Ln\left(\frac{1}{\lambda}\right) - \frac{1}{\lambda} \sum_{i=1}^{n} x_i$$

- Ln تابع Likelihood دست در همان نقطه‌ای بیشینه می‌شود که تابع Likelihood بیشینه خود را به دست می‌آورد!

- حالا مشتق میگیریم بر حسب پارامتر λ:

$$\frac{d}{d\lambda} Likelihood\left(\frac{1}{\lambda}, x_1, x_2, \dots, x_n\right) = \frac{d}{d\lambda}\left(n\, Ln\left(\frac{1}{\lambda}\right) - \frac{1}{\lambda} \sum_{i=1}^{n} x_i\right)$$

$$= n\lambda + \frac{1}{\lambda^2} \sum_{i=1}^{n} x_i = 0$$

- حالی مشخص کردن نقطه بیشینه مشتق تابع Likelihood را برابر صفر گذاشتم.

$$= \sum_{i=1}^{n} x_i = -n\lambda^2$$

$$\lambda = \frac{-\sqrt{\sum_{i=1}^{n} x_i}}{\sqrt{n}}$$ به شرط آنکه $n \neq 0$ باشد

$$f(t) = \frac{B^{\alpha}}{\Gamma(\alpha)} \left(\frac{1}{t}\right)^{\alpha+1} e^{-B/t}$$

- وقتی که میخواهیم توزیع پسین رو به دست آوریم باید تابع Likelihood ضرب در prior distribution کنیم

- برای راحتی کار k را برابر یک قرار میدهیم

$$\left[\frac{B^{\alpha}}{\Gamma(\alpha)} \left(\frac{1}{\lambda}\right)^{\alpha+1} e^{\frac{-B}{\lambda}}\right] * \left[n \ln\left(\frac{1}{\lambda}\right) - \frac{1}{\lambda} \sum_{i=1}^{n} x_i\right]$$

- حل این رابطه منجر به توزیع گاما میشود

حالا k را برابر ۲ قرار میدهیم $(k=2)$:

$$f(x) = \frac{k}{\lambda} \left(\frac{x}{\lambda^{k}}\right)^{k-1} e^{-(x/\lambda)^{k}} \xrightarrow{k=2} f(x) = \frac{2}{\lambda} \left(\frac{x}{\lambda}\right)^{2-1} e^{-(x/\lambda)^{2}}$$

$$f(x) = \frac{2x}{\lambda^{2}} e^{-(x/\lambda)^{2}}$$

$$L(\lambda, x_1, x_2, \ldots, x_n) = 2^{n} \prod_{i=1}^{n} \frac{x_i}{\lambda^{2}} e^{-(x_i/\lambda)^{2}}$$

$$= \prod_{j=1}^{N} f(x_i, \lambda) = 2^{n} \times \frac{1}{\lambda^{2n}} \prod_{i=1}^{n} x_i \times$$

$$\rightarrow \times e^{-\sum \frac{x_i^{2}}{\lambda^{2}}}$$

- از هردو طرف عبارت Ln میگیریم :

$$Ln(Likelihood(\lambda)) = n + \sum_{i=1}^{n} Ln\, x_i - 2n\, Ln(\lambda) - \frac{1}{\lambda^{2}} \sum_{i=1}^{n} x_i^{2}$$

• حال نسبت به λ مشتق میگیریم:

$$\frac{d}{d\lambda}(\text{likelihood}(\lambda)) = \frac{-2n}{\lambda} + 2\lambda^{-3}\sum_{i=1}^{n} x_i^2$$

• برای مشخص کردن نقطه بیشینه مشتق تابع likelihood را برابر صفر میگذاریم

$$\frac{-2n}{\lambda} + 2\lambda^{-3}\sum_{i=1}^{n} x_i^2 = 0$$

$$\frac{2n}{\lambda} - 2\lambda^{-3}\sum_{i=1}^{n} x_i^2 = 0 \Rightarrow n - \frac{1}{\lambda^2}\sum_{i=1}^{n} x_i^2 = 0$$

$$\Rightarrow n = \frac{1}{\lambda^2}\sum_{i=1}^{n} x_i^2 \Rightarrow \lambda^2 n = \sum_{i=1}^{n} x_i^2 \Rightarrow$$

$$\lambda^2 = \frac{\sum_{i=1}^{n} x_i^2}{n} \Rightarrow \lambda = +\sqrt{\frac{\sum x_i^2}{n}}$$

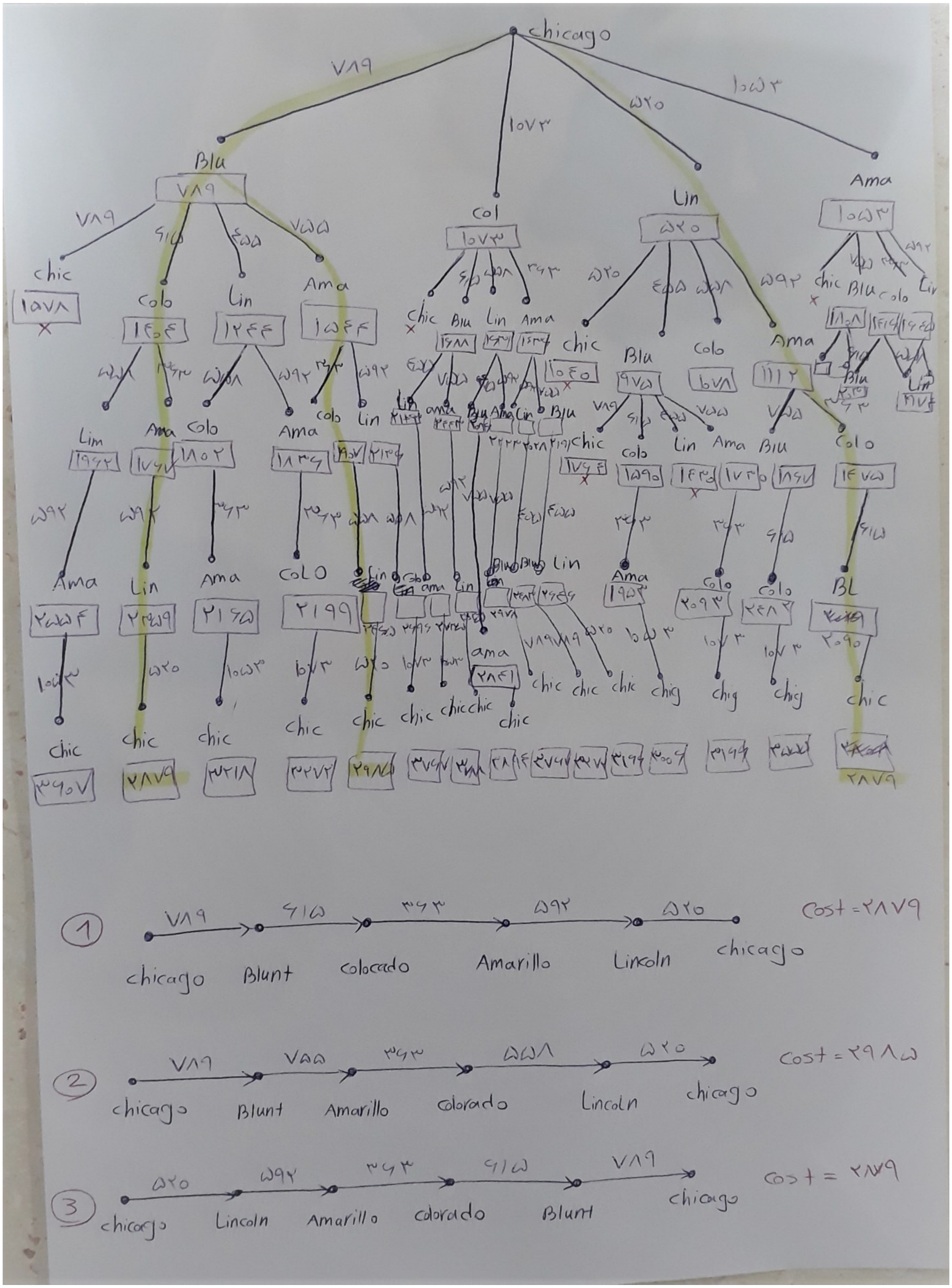
~~$f(t) = \frac{k}{\lambda}\left(\frac{\pi}{\lambda}\right)$~~

$$f(t) = \frac{B^\alpha}{\Gamma(\alpha)}\left(\frac{1}{t}\right)^{\alpha+1} e^{-B/t}$$

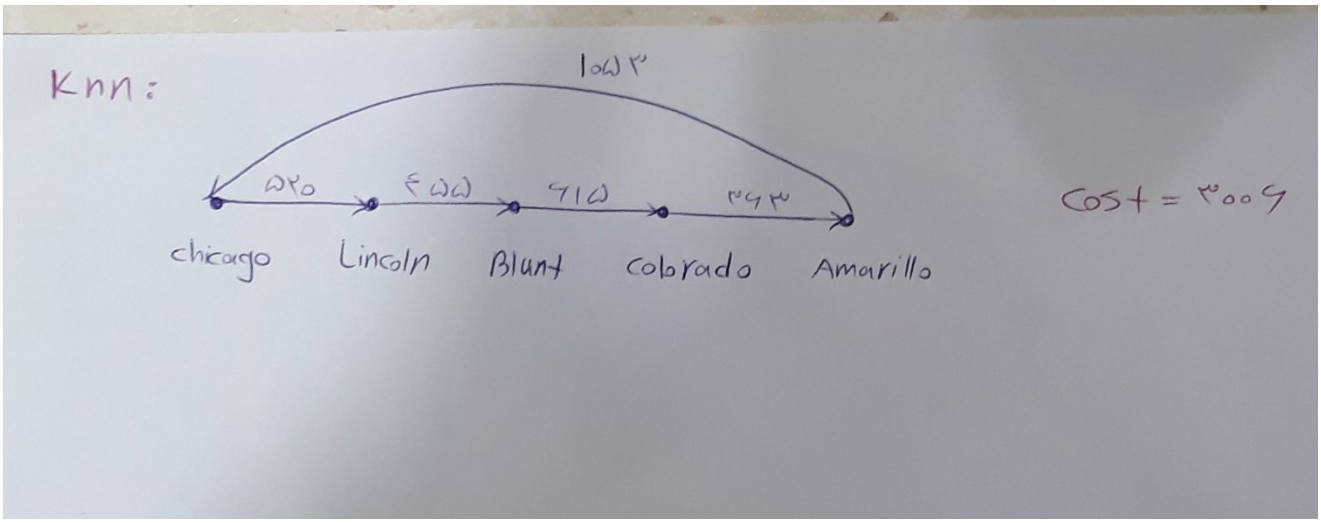
• $k=1$

• تابع likelihood را باید ضرب در prior dist کنیم برای به دست آوردن توزیع پسین!

$$\frac{B^\alpha}{\Gamma(\alpha)}\left(\frac{1}{t}\right)^{\alpha+1} e^{-B/\alpha} * \frac{\mu^n}{\lambda^{2n}}\prod_{i=1}^{N} x_i\, e^{-\frac{\sum x_i^2}{\lambda^2}}$$


chicago
1
chicago ۷۸۹ Blunt ۶۱۵ Colocado ۳۶۳ Amarillo ۵۹۲ Lincoln ۵۲۰ chicago
Cost = ۲۸۷۹
2
chicago ۷۸۹ Blunt ۷۵۵ Amarillo ۳۶۳ Colorado ۵۵۸ Lincoln ۵۲۰ chicago
Cost = ۲۹۸۵
3
chicago ۵۲۰ Lincoln ۵۹۲ Amarillo ۳۶۳ colorado ۶۱۵ Blunt ۷۸۹ chicago
Cost = ۲۸۷۹

سوال 2



الگوریتم ucs سه راه با هزینه های کمتر نسبت به knn به ما میدهد پس لزوما الگوریتم knn همیشه خوب عمل نمیکند.

سوال 3) k-NN implementation

i) نحوه خواندن فایل Dataset و label ها از ورودی:

```
10    df = pd.read_csv('uspsdata.csv' ,sep="\t")
11    lbl = pd.read_csv('uspscl.csv' ,sep="\t")
```

در اینجا با استفاده از کتابخانه pandas که نام مستعار Pd برای آن گذاشته ایم فایل ها csv دیتاست و لیبل را میخوانیم و متغیرهایی به اسم lbl وdf تعریف کردیم که دیتا فریم را در قالب متغیری با این اسم ذخیره میکنیم.

```
df = df.assign(lbls=lbl.values)
```

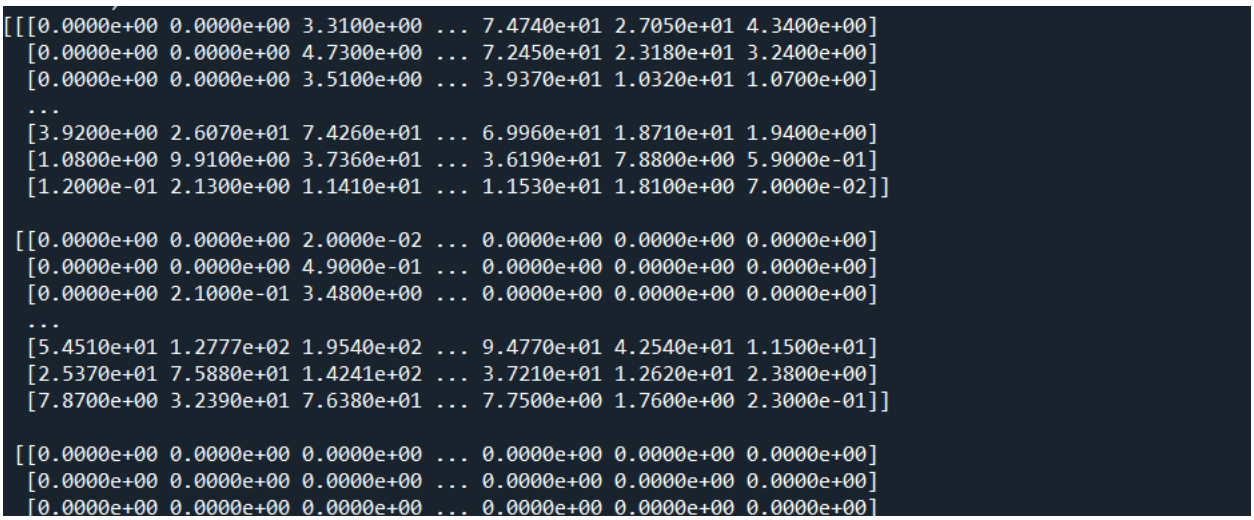
یک ستون به دیتاست اضافه میکنیم و لیبل ها را در آن قرار میدهیم.

```
25
26    a_3d_array = np.zeros((200,16,16))
27
```

یک آرایه سه بعدی با استفاده از تابع zeros از کتابخانه numpy تعریف کردم چون که هر سطر از دیتاست 256 دیتا دارد که میشود یک ماتریس 16*16 و دیتاست ما هم 200 سطر داریم پس میشود یک ماتریس یا آرایه 200*16*16 تایی.

```
27
28  for i in range(0,199):
29
30      a_3d_array[i]=(np.array(df.iloc[i])).reshape(16, 16)
31
```

دیتا ها را داخل این ماتریس سه بعدی که تعریف کردیم میریزیم از تابع iloc استفاده کردم برای خواندن سطر های دیتاست و هر سطر پس خواندن با استفاده از تابع reshape تبدیل میشود میشود به یک ماتریس 16*16.



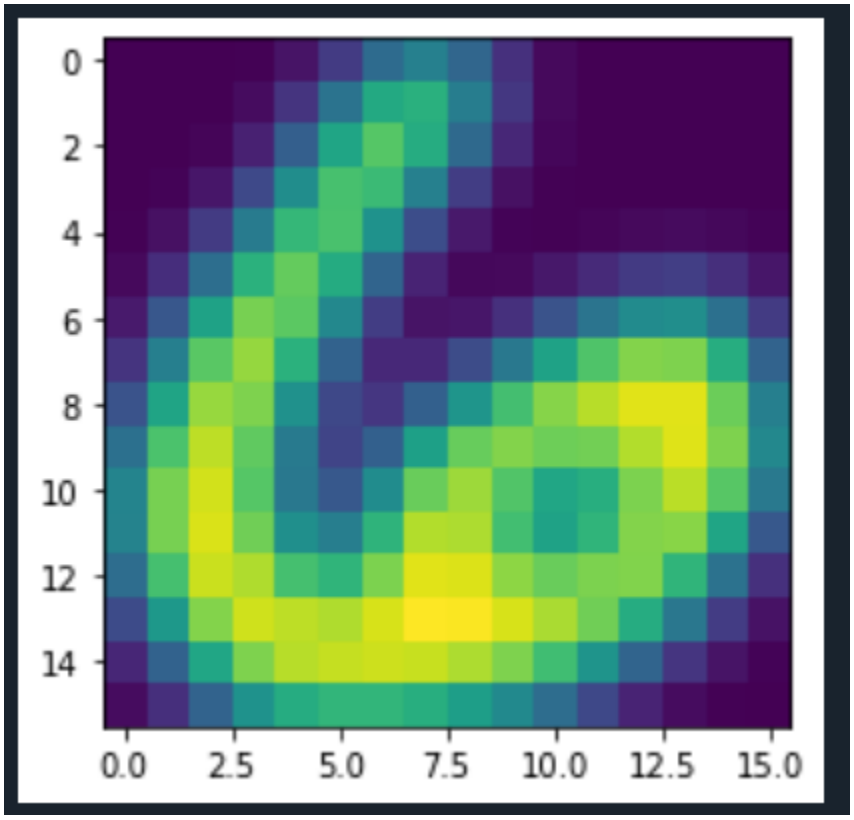
```
[[[0.0000e+00 0.0000e+00 3.3100e+00 ... 7.4740e+01 2.7050e+01 4.3400e+00]
  [0.0000e+00 0.0000e+00 4.7300e+00 ... 7.2450e+01 2.3180e+01 3.2400e+00]
  [0.0000e+00 0.0000e+00 3.5100e+00 ... 3.9370e+01 1.0320e+01 1.0700e+00]
  ...
  [3.9200e+00 2.6070e+01 7.4260e+01 ... 6.9960e+01 1.8710e+01 1.9400e+00]
  [1.0800e+00 9.9100e+00 3.7360e+01 ... 3.6190e+01 7.8800e+00 5.9000e-01]
  [1.2000e-01 2.1300e+00 1.1410e+01 ... 1.1530e+01 1.8100e+00 7.0000e-02]]

 [[0.0000e+00 0.0000e+00 2.0000e-02 ... 0.0000e+00 0.0000e+00 0.0000e+00]
  [0.0000e+00 0.0000e+00 4.9000e-01 ... 0.0000e+00 0.0000e+00 0.0000e+00]
  [0.0000e+00 2.1000e-01 3.4800e+00 ... 0.0000e+00 0.0000e+00 0.0000e+00]
  ...
  [5.4510e+01 1.2777e+02 1.9540e+02 ... 9.4770e+01 4.2540e+01 1.1500e+01]
  [2.5370e+01 7.5880e+01 1.4241e+02 ... 3.7210e+01 1.2620e+01 2.3800e+00]
  [7.8700e+00 3.2390e+01 7.6380e+01 ... 7.7500e+00 1.7600e+00 2.3000e-01]]

 [[0.0000e+00 0.0000e+00 0.0000e+00 ... 0.0000e+00 0.0000e+00 0.0000e+00]
  [0.0000e+00 0.0000e+00 0.0000e+00 ... 0.0000e+00 0.0000e+00 0.0000e+00]
  [0.0000e+00 0.0000e+00 0.0000e+00 ... 0.0000e+00 0.0000e+00 0.0000e+00]
```

print(a_3d_array)

(ii

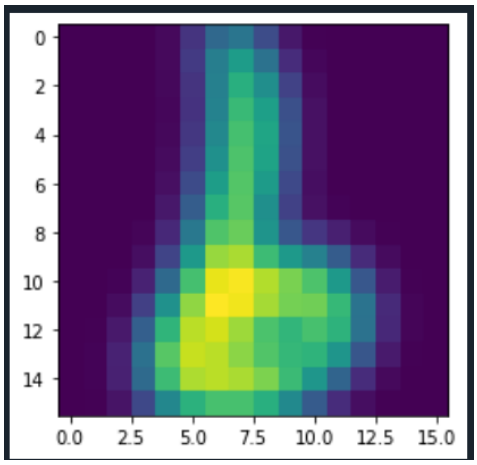
```
arr1=np.array(df.iloc[1])
newarr = arr1.reshape(16, 16)
plt.imshow(newarr, interpolation='nearest')
```

سطر اول خوانده شد از دیتاست و تبدیل به آرایه شد سپس اون رو به یک آرایه 16*16 ریشیپ میکنیم و سپس با استفاده از کتابخانه Pyplot و تابع imsow عکس رو نمایش میدهیم.



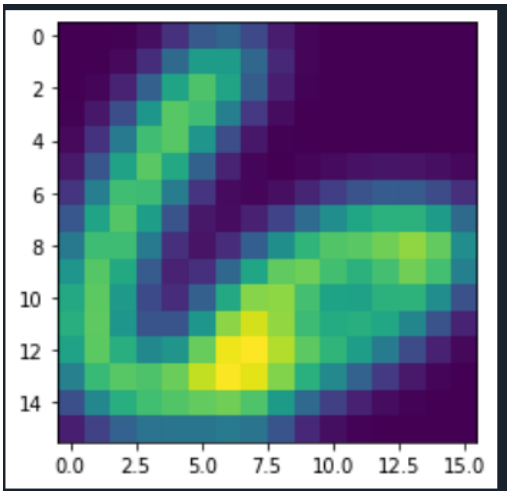
Output

```
arr1=np.array(df.iloc[2])
newarr = arr1.reshape(16, 16)
plt.imshow(newarr, interpolation='nearest')
```



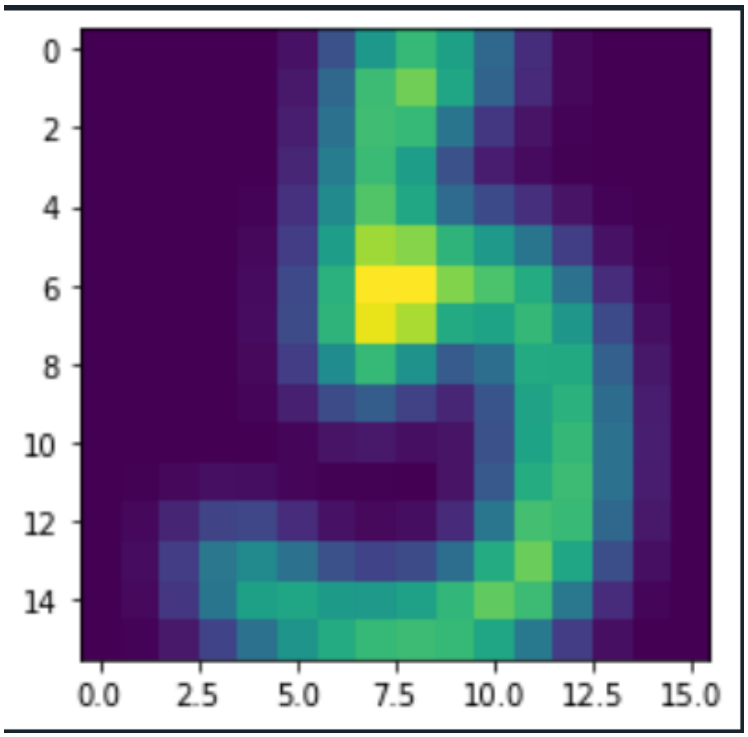
Output

```
arr1=np.array(df.iloc[3])
newarr = arr1.reshape(16, 16)
plt.imshow(newarr, interpolation='nearest')
```



Output

```
arr1=np.array(df.iloc[4])
newarr = arr1.reshape(16, 16)
plt.imshow(newarr, interpolation='nearest')
```



Output

(iii

```
59
60
61    train, validate, test = np.split(df.sample(frac=1), [int(.6*len(a_3d_array)), int(.8*len(a_3d_array))])
62
```

با استفاده تابع split و کتابخانه numpy داده ها را به سه بخش train(60%) test(20%), validation(20%), تقسیم کردیم.

84	0.00	0.00	0.00	0.00	0.00	...	22.82	3.74	0.04	0.00	-1
54	0.00	0.00	0.01	1.10	8.99	...	2.60	0.28	0.00	0.00	1
96	0.00	0.00	0.00	0.60	8.23	...	0.10	0.00	0.00	0.00	1
114	0.00	0.00	0.00	0.00	2.05	...	59.54	29.86	9.39	1.32	-1
165	0.00	0.18	3.23	13.74	28.09	...	6.64	0.98	0.00	0.00	-1
..	...	...	...	...	...	...	...	...	...	...	...
72	0.00	0.00	0.00	0.00	2.00	...	3.67	0.33	0.00	0.00	1
121	0.00	0.00	0.00	0.00	0.50	...	79.08	44.54	18.71	5.03	1
105	12.77	32.63	57.71	81.93	105.45	...	28.92	7.96	1.08	0.03	-1
1	0.00	0.00	0.02	1.86	12.63	...	23.76	7.75	1.76	0.23	1
120	0.00	0.00	0.00	0.00	0.21	...	5.24	0.54	0.00	0.00	1

[120 rows x 257 columns]

Train Data

34	0.00	0.00	0.25	2.21	12.03	...	40.08	19.99	7.41	1.67	1
22	0.00	0.00	0.00	0.16	1.98	...	62.78	33.44	12.42	2.74	1
43	0.00	0.00	0.00	0.00	0.00	...	28.97	6.16	0.45	0.00	-1
8	0.00	0.00	0.11	2.19	13.29	...	39.52	13.01	2.12	0.01	-1
41	0.00	0.00	0.03	0.53	2.56	...	2.12	0.15	0.00	0.00	-1
191	0.00	0.00	0.00	0.00	0.00	...	84.44	37.45	9.13	0.80	1
9	0.00	0.00	0.00	0.00	0.00	...	17.00	3.17	0.13	0.00	1
185	0.00	0.00	0.00	0.00	0.01	...	12.07	2.66	0.22	0.00	1
70	0.00	0.00	0.00	3.47	23.53	...	0.52	0.00	0.00	0.00	1
73	0.00	0.00	0.72	6.17	22.54	...	14.74	3.55	0.35	0.00	-1
63	0.00	0.00	0.00	0.00	0.53	...	3.63	0.06	0.00	0.00	1
61	0.00	0.00	0.00	0.00	1.08	...	12.91	2.03	0.03	0.00	1
184	0.00	0.00	0.00	0.00	0.26	...	8.04	2.54	0.42	0.00	1
62	0.00	0.00	0.00	0.08	3.92	...	138.24	82.51	29.77	4.75	-1
81	0.00	0.00	7.94	35.36	62.06	...	2.90	0.33	0.00	0.00	1
143	0.00	0.00	0.23	5.27	26.03	...	61.64	36.69	17.03	5.26	-1
146	0.25	2.26	8.01	17.07	27.14	...	157.63	118.64	63.26	20.27	-1
21	0.00	0.00	0.00	0.02	0.84	...	8.07	2.22	0.30	0.00	1

[40 rows x 257 columns]

validation Data

19	0.88	4.65	14.96	35.60	66.67	...	20.97	5.39	0.64	0.00	-1
135	0.00	0.00	0.00	3.93	25.14	...	0.17	0.00	0.00	0.00	-1
26	0.00	0.15	1.38	7.77	29.49	...	123.94	98.61	60.84	23.65	-1
193	0.00	0.00	0.00	1.88	13.30	...	79.62	45.24	18.72	4.37	-1
28	0.00	0.00	0.00	0.00	0.00	...	119.72	73.05	29.16	5.70	-1
30	0.00	0.00	0.00	0.00	0.00	...	77.96	31.77	6.80	0.42	1
196	0.00	0.00	0.00	0.00	0.00	...	4.11	0.82	0.00	0.00	1
183	0.00	0.00	0.00	0.00	0.58	...	0.16	0.01	0.00	0.00	-1
20	0.00	0.00	0.00	0.00	0.06	...	16.24	3.09	0.18	0.00	1
80	0.00	0.00	0.00	0.00	0.23	...	22.66	6.15	0.80	0.01	1
170	0.00	0.00	0.00	0.00	0.00	...	1.68	0.32	0.00	0.00	-1
25	0.00	0.00	0.00	0.34	2.18	...	88.88	60.51	31.46	10.77	-1
5	0.00	0.00	0.00	0.00	0.00	...	0.51	0.00	0.00	0.00	1
172	5.98	16.16	25.61	24.10	13.06	...	0.18	0.00	0.00	0.00	-1
127	0.00	0.00	0.00	0.00	0.32	...	21.51	5.19	0.50	0.00	1
10	0.00	0.00	0.00	0.00	0.40	...	3.35	0.13	0.00	0.00	1
47	0.00	0.00	0.00	0.00	0.00	...	9.82	0.70	0.00	0.00	-1

[39 rows x 257 columns]

Test Data

(iv

```
87
88  def euclidean_distance(row1, row2):
89      distance = 0.0
90      for i in range(len(row1)-1):
91          distance += (row1[i] - row2[i])**2
92      return sqrt(distance)
93
```

در الگوریتم KNN برای اینکه ما نزدیک ترین همسایه رو پیدا کنیم نیاز داریم فاصله بین سطر ها را پیدا کنیم، ورودی این تابع دو تا سطر میباشد که فاصله اقلیدسی بین اون ها رو پیدا میکند و هر چه این مقدار کمتر باشد دو سطر به هم شبیه تر هستند.

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

Euclidean distance

```
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors
```

پس از محاسبه فواصل باید تمام سطر های دیتاست را بر اساس فاصله آن ها تا داده جدید را مرتب کنیم سپس میتوانیم k تا از شبیه ترین همسایه ها را پیدا کنیم.

```python
def predict_classification(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
    output_values = [row[-1] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction
```

از مشابه ترین همسایه گان جمع آوری شده از دیتاست به داده جدید میتوانیم برای پیش بینی استفاده کنیم، با اجرای تابع MAX بر روی لیست، دسته ای که بیشترین تکرار را دارد انتخاب میکنیم.

```python
# kNN Algorithm
def k_nearest_neighbors(train, test, num_neighbors):
    predictions = list()
    for row in test:
        output = predict_classification(train, row, num_neighbors)
        predictions.append(output)
    return(predictions)
```

از این تابع برای مدیریت کار با الگوریتم KNN استفاده میکنیم که داده های train و test و تعداد همسایه ها را به ورودی تابع میدهیم.

```python
def get_error_rate(training_dataset, test_dataset,k):

    errors_count = 0
    for test_data in test_dataset:
        predicted_class = predict_classification(training_dataset, test_data,k)

        if predicted_class != test_data[256]:
            errors_count += 1
        return errors_count / len(test_dataset)
```

تابع محاسبه error rate

Error rate = Correct Prediction / Total Test Data

```python
k=1
error=get_error_rate(TrainArray,TestArray,k)
print('error: %s' % error)
```

Input

```
error: 0.02564102564102564
```

Output

```
k=3
error=get_error_rate(TrainArray,TestArray,k)
print('error: %s' % error)
```

Input

```
error: 0.05128205128205128
```

Output

```
k=5
error=get_error_rate(TrainArray,TestArray,k)
print('error: %s' % error)
```

Input

```
error: 0.02564102564102564
```

Output

```
k=7
error=get_error_rate(TrainArray,TestArray,k)
print('error: %s' % error)
```

Input

```
error: 0.05128205128205128
```

Output