دانشگاه تهران

دانشکده الگوریتم و محاسبات

استاد درس:

دکتر شعبان‌خواه

گزارش دهنده:

مصطفی محمدی قراسویی

۸۱۰۸۹۷۰۳۰

درس:

Fundamental of Neural Networks

تمرین:

SOM Kohonen

شرح

هدف از طراحی این برنامه، اجرا و ارزیابی الگوریتم SOM در سه ساختار زیر می‌باشد:

1. No topological structure,
2. Linear structure,
3. Matrix structure.

طراحی این برنامه بر اساس unsupervised است. این شبکه‌ها به منظور یادگیری ماشین بوده هنگامی که هیچ گونه دسته بندی برای موارد مورد آموزش وجود ندارد می‌باشد. با این کار الگوریتم به صورت خودکار ورودی‌ها را مورد تجزیه و تحلیل قرار داده و هر ورودی که به همدیگر از لحاظ ساختاری شبیه‌تر باشند را در یک دسته قرار می‌دهد. در این روش اطلاعاتی که پس از آموزش به ماشین داده می‌شود قابل تصحیح نبوده و هرگونه شباهتی که به یکی از دسته بندی‌ها داشته باشد، آن ورودی را در آن دسته‌بندی قرار می‌دهد. در یادگیری ماشین به این دسته بندی‌ها کلاستر گفته می‌شود.

ورودی‌های ماشین متشکل از ۲۱ عدد عکس bitmap می‌باشد که نشان دهنده ۷ کاراکتر A, B, C, D, E, F, J و k است که هر کدام با سه قلم مختلف نوشته شده‌اند. عکس‌های ورودی دارای ابعاد ۹*۷ می‌باشد.

تعداد کلاسترهای تعیین شده برای ماشین نیز ۲۵ عدد کلاستر می‌باشد که پس از یادگیری ماشین بر اساس ۲۱ کاراکتر ورودی هرکدام از ورودی‌ها می‌تواند به یکی از کلاسترها انتساب داده شود. براساس اینکه قلم‌های مختلفی در این یادگیری استفاده شده است، با نگاه اولیه تصور می‌شود که ماشین کاراکترهایی که شبیه به یکدیگر هستند را در کلاسترهای یکسانی قرار دهد ولی نتیجه جالبی که به همراه دارد این است که ماشین نمی‌تواند کاملا عقلانی(بر اساس تفکر انسان) کاراکترهای مختلف را بصورت دقیق با یکدیگر در کلاسترهای مجزا قرار دهد و ممکن است که کاراکترهایی که در یک خانواده نیستند در کلاسترهای دیگر قرار بگیرند.

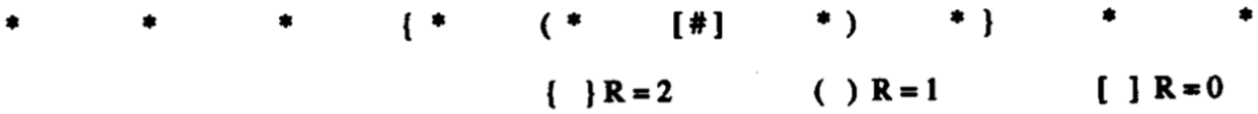
الگوریتم حاضر که توسط برنامه علمی matlab نوشته شده است با سه رویکرد پیاده سازی شده است. اساس یادگیری ماشین الگوریم SOM بر پایه اختلاف فاصله از روش فاصله اقلیدوسی می‌باشد. هر کاراکتر در این ماشین به یک فضای ۶۳ بعدی انتساب داده می‌شود و الگوریتم فاصله اقلیدوسی آن نقطه را با تمامی کلاسترهای موجود که آن‌ها نیز در فضای ۶۳ بعدی هستند مقایسه کرده و مختصات خود را به آن کاراکتر یا کاراکترها نزدیک می‌کند. لازم به ذکر است که جابجایی نقاط در این فضای ۶۳ بعدی برای کلاسترها بوده و مختصات کاراکترها به هیچ عنوان جابجا نمی‌شود. جابجایی کلاسترها در

این فضا که براساس کمترین فاصله اقلیدوسی است آنقدر تکرار می‌شود که دیگر یا جابجایی اتفاق نیافتد و یا اینکه جابجایی کلاسترها آنقدر ناچیز باشد که قابل چشم پوشی باشد.

مختصات کلاسترها را در یک ماتریس ۶۳*۲۵ نگهداری می‌کنیم. لازم به ذکر است که با این کار کلاسترها در یک آرایه یک بعدی هستند که بعد دوم آن مختصات آنها می‌باشد. انتخاب مختصات کلاسترها به صورت کاملا تصادفی بین صفر و یک انتخاب شده است. انتخاب کلاستر برای به روزرسانی وزن‌های خود(به عبارت بهتر، به روزرسانی نقطه خود در فضای ۶۳ بعدی) به سه روش اتفاق می‌افتد. در روش اول دقیقا کلاستری مختصات خود را به روز رسانی می‌کند که کمترین فاصله اقلیدوسی را در بین تمامی کلاسترهای دیگر داشته باشد. در روش دوم هنگامی که یک کلاستر با توجه به فاصله اقلیدوسی برای به روز رسانی انتخاب می‌شود، دو کلاستر قبل و بعد آن نیز (با توجه به وضعیت قرارگیری آن کلاستر در ماتریس کلاسترها) وزن خود را آپیدیت می‌کنند. در روش سوم نیز این کار توسط کمی تغییر در مکان ذخیره سازی مختصات کلاسترها اتفاق می‌افتد؛ به این معنی که کلاسترها را در یک ماتریس سع بعدی است، دو بعد سطر و ستون برای وضعیت کلاسترها نسبت به یکدیگر و بعد سوم برای نگهداری مختصات کلاسترها. با این تعریف ماتریس سع بعدی کلاسترها به شکل ۶۳*۵*۵ است. در روش سوم هنگامی که یک کلاستر کاندید به روزرسانی وزن‌هایش است چهار کلاستر بالا، پایین، چپ و راست آن نیز وزن خود را به روز رسانی می‌کنند.

به این نکته باید توجه داشت همانگونه که الگوریتم در مراحل مختلفی در حال اجرا است، می‌بایست که آرام آرام وضعیت به‌روز رسانی کلاستری در آن کاهش پیدا کند. به عنوان مثال اگر در ۱۰۰ تکرار اول قرار است که به غیر از کلاستر کاندید وزن کلاسترهای چپ و راست آن کلاستر نیز آپدیت شوند، در تکرار ۱۰۱ به بعد دیگر فقط خود کلاستر کاندید به روز رسانی می‌شود. این رویکرد به radius معروف است و آن را با R نمایش می‌دهند.

R نشان دهنده چگونگی به روز رسانی کلاستر ها است که در تکرارهای مختلف در اجرای یک بار الگوریتم از عدد آن کم می‌شود. شکل زیر نشان دهنده وضعیت کلاستر ها در آرایه دو بعدی و سه بعدی است که برای سهولت بعدهای مختصات آن حذف شده است.



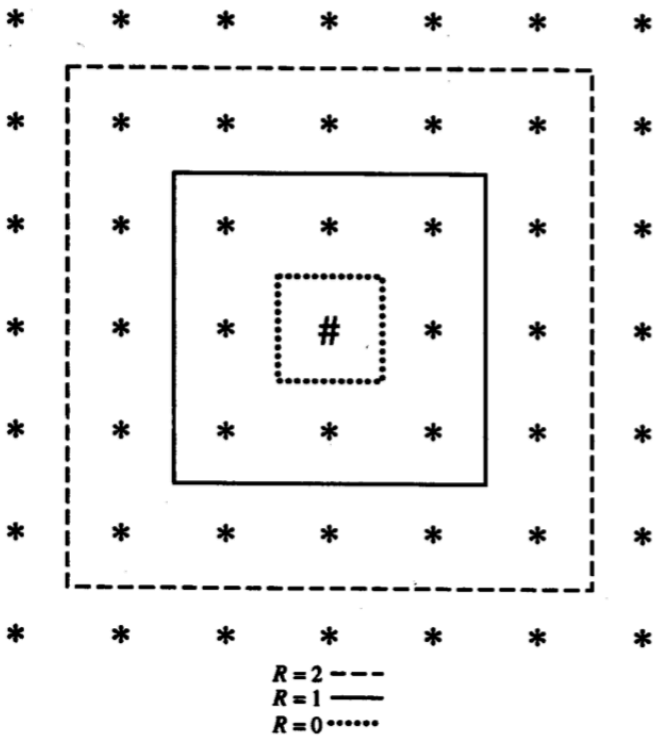


Figure 4.7 Neighborhoods for rectangular grid.

شکل زیر نیز نشان‌دهنده وضعیت کلاسترها هنگامی که در یک ماتریس دو بعدی قرار دارند را نشان می‌دهد.

در این برنامه نویسی ما از هر سه مدل پیاده‌سازی توپولوژیکال استفاده خواهیم کرد با این تفاول که در استراکچر ماتریسی کلاسترها به جای استفاده از شکل به روز رسانی بالا، کلاسترها را بصورت لوزی به‌روز رسانی می‌کنیم به این صورت که اگر مثلا کلاستر $X_{i,j}$ کاندید به روز رسانی است، کلاسترهای $X_{i-1,j}$, $X_{i+1,j}$, $X_{i,j-1}$ و $X_{i,j+1}$ نیز به روز رسانی خواهند شد و پس از گذشت چند دور تکرار در الگوریتم این به روزرسانی به صفر رسیده و فقط کلاستر کاندید به روز رسانی می‌شود.

Pseudocode الگوریتم SOM Kohonen را در زیر مشاهده می‌کنید.

Step 0. **Initialize weights w_{ij}. (Possible choices are discussed below.)**
Set topological neighborhood parameters.
Set learning rate parameters.

Step 1. **While stopping condition is false, do Steps 2–8.**

Step 2. **For each input vector x, do Steps 3–5.**

Step 3. **For each j, compute:**

$$D(j) = \sum_i (w_{ij} - x_i)^2.$$

Step 4. **Find index J such that $D(J)$ is a minimum.**

Step 5. **For all units j within a specified neighborhood of J, and for all i:**

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + \alpha[x_i - w_{ij}(\text{old})].$$

Step 6. **Update learning rate.**

Step 7. **Reduce radius of topological neighborhood at specified times.**

Step 8. **Test stopping condition.**

در همه کدهای نوشته شده در این برنامه به منظور پیاده سازی هر سه مدل SOM از یک فایل تابع به نام minEuclidient.m به منظور پیدا کردن نزدیکترین نقطه مختصاتی کلاسترها و کاراکترها استفاده شده است.

هر سه مدل پیاده سازی در این گزارش به صورت جداگانه بوده و هر کدام با اینکه به یکدیگر شباهت‌های بسیار زیادی دارند، اما تفاوت در این کدها فقط در به روز رسانی کلاسترها می‌باشد که در بخش‌های قبلی این گزارش صحبت شده است.

کد اصلی این برنامه به نام minEuclidient.m است؛ در این کد پیاده سازی به صورت ساختار بدون توپولوژیک می‌باشد. کد اجرا شده دارای دو section به نام‌های initialize values و learning می‌باشد.

متغیرها و قسمت‌های مختلف این برنامه در خود برنامه به صورت comment مورد توجه قرار گرفته است که برای خواننده مفهموم مورد نیاز برنامه را برساند.

لازم به ذکر است که در این برنامه به جای اینکه تکرار (iteration) برنامه SOM به صورت پویا باشد، تکرار بر اساس عدد صحیح بوده و مورد توجه قرار گرفته است. در پیاده سازی بعدی که تمرین Kmeans است این بررسی اندازه تغییرات به صورت پویا صورت می‌گیرد. دلیل این کار تمییز دادن تفاوت بین دو نوع پیاده سازی بوده است.

اجراها

پیاده سازی برنامه به روش no topological structure نتیجه‌های زیر حاصل شد:

	First Run	Second Run	Third Run	Fourth Run	Fifth run	Sixth	Seventh run	Eighth run	Ninth run	Tenth run
{'A1'}	17	12	14	8	9	18	25	1	18	1
{'A2'}	17	12	14	8	9	18	25	1	18	1
{'A3'}	11	12	8	12	24	18	25	6	14	3
{'B1'}	6	21	22	6	13	8	3	11	15	2
{'B2'}	15	25	4	22	4	6	5	5	20	20
{'B3'}	6	21	22	6	13	8	3	11	15	2
{'C1'}	15	25	4	22	4	6	5	5	20	20
{'C2'}	15	25	4	22	4	6	5	5	20	20
{'C3'}	15	25	4	22	4	6	5	5	20	20
{'D1'}	6	21	22	6	13	8	3	11	15	2
{'D2'}	15	25	4	22	4	6	5	5	20	20
{'D3'}	6	21	22	6	13	8	3	11	15	2
{'E1'}	24	21	22	15	21	8	3	11	13	2

	First Run	Second Run	Third Run	Fourth Run	Fifth run	Sixth	Seventh run	Eighth run	Ninth run	Tenth run
{'E2'}	15	25	4	22	4	6	5	5	20	20
{'E3'}	24	21	22	15	21	8	3	11	13	2
{'J1'}	20	18	9	11	5	25	20	22	12	18
{'J2'}	20	18	9	11	5	25	20	22	12	18
{'J3'}	20	18	9	11	5	25	20	22	12	18
{'K1'}	24	21	22	15	21	8	3	11	13	2
{'K2'}	14	25	13	22	20	23	5	5	20	20
{'K3'}	24	21	22	15	21	8	3	11	13	2

پیاده سازی برنامه به روش Linear topological structure نتیجه‌های زیر حاصل شد:

	First Run	Second Run	Third Run	Fourth Run	Fifth run	Sixth	Seventh run	Eighth run	Ninth run	Tenth run
{'A1'}	1	5	16	25	14	3	24	9	1	5
{'A2'}	1	7	16	25	14	1	24	9	1	5
{'A3'}	1	3	18	23	12	5	24	11	4	3
{'B1'}	16	23	11	18	24	21	19	2	16	9
{'B2'}	18	21	10	17	20	22	18	3	17	11
{'B3'}	15	23	12	19	24	20	20	1	14	9
{'C1'}	5	19	7	14	18	25	15	5	25	17
{'C2'}	5	19	7	14	18	25	15	5	23	15
{'C3'}	5	19	7	14	18	25	15	5	23	15
{'D1'}	13	23	12	19	22	19	20	1	15	9
{'D2'}	20	21	9	16	20	23	17	4	21	11
{'D3'}	13	23	12	19	22	19	20	1	15	9
{'E1'}	9	25	14	21	7	15	22	1	13	7
{'E2'}	18	21	9	16	9	23	17	3	18	11
{'E3'}	9	25	14	21	7	15	22	1	13	7
{'J1'}	3	17	3	12	16	13	11	7	6	1
{'J2'}	3	15	5	10	16	11	13	7	9	1
{'J3'}	3	17	5	12	16	13	13	7	6	1
{'K1'}	11	25	14	21	3	17	22	1	11	7
{'K2'}	7	1	9	16	9	23	17	22	19	13
{'K3'}	11	25	14	21	5	17	22	1	11	7

پیاده سازی برنامه به روش diamond topological structure نتیجه‌های زیر حاصل شد:

	First Run i\j	Second i\j	Third Run	Fourth Run	Fifth run	Sixth	Seventh run	Eighth run	Ninth run	Tenth run
{'A1'}	1,1	3,1								
{'A2'}	1,1	3,1								
{'A3'}	5,4	2,5								
{'B1'}	4,1	3,3								
{'B2'}	5,1	3,5								
{'B3'}	3,1	3,3								
{'C1'}	4,5	1,5								
{'C2'}	4,5	1,5								
{'C3'}	4,5	1,5								
{'D1'}	3,2	4,4								
{'D2'}	4,2	3,5								
{'D3'}	3,2	4,4								
{'E1'}	4,3	2,2								
{'E2'}	5,2	2,4								
{'E3'}	3,4	2,2								
{'J1'}	2,5	1,1								
{'J2'}	2,5	1,1								
{'J3'}	2,5	1,1								
{'K1'}	2,3	1,3								
{'K2'}	1,4	5,5								
{'K3'}	2,3	1,3								

لازم به ذکر است چون خروجی برنامه به صورت diamond structure است، مجبور به ذخیره آن به صورت یک ماتریس سه بعدی ۲۱*۵*۵ هستیم. در این متغیر هر سطح از ماتریس بیان کننده قرار گیری یک کاراکتر در بین ۲۵ کلاستر موجود می‌باشد. مثالا مکان انتساب کاراکتر A1 در این ماتریس در سطح ۱ ماتریس یعنی output(:,:,1) مشخص شده است و کاراکتر A2 در سطح ۲ ماتریس یعنی output(:,:,2) و در نهایت کاراکتر K3 در سطح output(:,:,21) واقع شده است.

نتیجه گیری

همان گونه که از اجراهای مختلف در هر سه توپولوژی مشخص است، برنامه در کل قادر به تشخیص صد در صد کاراکترهای هم‌خانواده نیست و شماره گذاری کاراکترهای می‌تواند در توپولوژی‌های مختلف رویکرد متفاوتی داشته باشند. مثلا سه

کاراکتر $j1, j2$ و $j3$ در توپولوژی بدون استراکچر و توپولوزی استراکچر لوزی در یک گروه قرار گرفته‌اند ولی در توپولوژی استراکچر خطی در برخی موارد یک گروه‌های مختلفی قرار می‌گیرند.

پایان گزارش