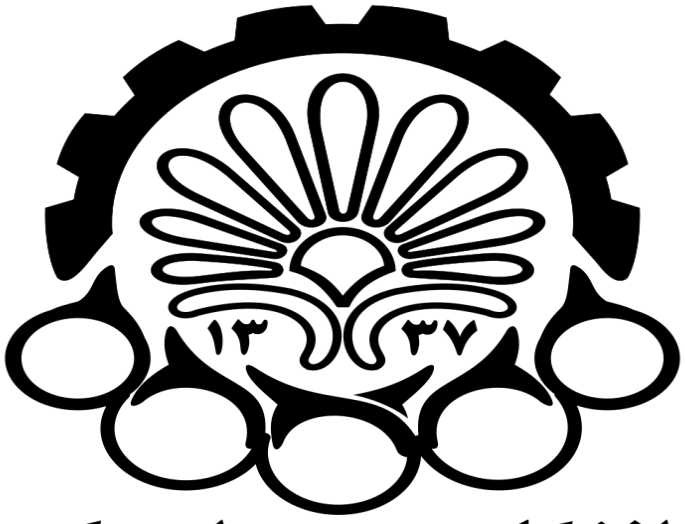


دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

درس رایانش عصبی و یادگیری عمیق

استاد صفابخش

نیما پری فرد

۴۰۲۱۳۱۰۱۷

فهرست تمرین شماره ۱

بخش ۱

SOM، یک الگوریتم یادگیری بدون نظارت، ویژگی‌های متمایزی نسبت به سایر روش‌های خوشه‌بندی دارد که می‌توان آن‌ها را در چندین بعد مختلف مورد بررسی قرار داد، از جمله ساختار، روش آموزشی، و کاربردهای معمول:

۱. مکانیزم آموزشی

SOM: از یادگیری رقابتی استفاده می‌کند که در آن نورون‌ها برای تبدیل شدن به واحد تطبیق بهترین (BMU) با یکدیگر رقابت می‌کنند و یک تابع همسایگی وزن‌ها را به شیوه‌ای تحت تأثیر توپولوژی به‌روزرسانی می‌کند. این روش به حفظ خواص توپولوژیکی فضای ورودی کمک می‌کند.

سایر الگوریتم‌های خوشه‌بندی: مانند k-means سلسله مراتبی یا DBSCAN، معمولاً شامل تابع همسایگی نمی‌شوند و نگران حفظ خواص توپولوژیکی فضای ورودی نیستند. آن‌ها بر اساس معیارهای فاصله مانند فاصله اقلیدسی یا منهتن خوشه‌بندی می‌کنند بدون اینکه ساختاری را در داده‌ها فراتر از گروه‌بندی‌های ساده در نظر بگیرند.

۲. حفظ توپولوژی

SOM: به صراحت به دنبال ایجاد نقشه‌ای است که ساختار توپولوژیکی داده‌های ورودی را حفظ کند، که این ویژگی منحصر به فردی در میان الگوریتم‌های خوشه‌بندی است. این ویژگی SOM را برای تجسم و کاوش داده‌های چندبعدی به ویژه مفید می‌سازد.

سایر الگوریتم‌ها: به طور کلی، توپولوژی را در فضای خروجی حفظ نمی‌کنند؛ هدف آن‌ها عمدتاً گروه‌بندی آیتم‌های مشابه بر اساس معیار انتخابی شباهت یا فاصله است بدون آنکه چیدمان فضایی سازمان‌یافته‌ای داشته باشند. k-Means سریع‌تر و قابل مقیاس‌پذیری بیشتری است اما خصوصیات توپولوژیکی را حفظ نمی‌کند و می‌تواند نسبت به نقاط پرت حساس باشد. در شناسایی خوشه‌های با اشکال و اندازه‌های متفاوت

مؤثر است و نسبت به نقاط پرت مقاوم است، اما روش مستقیمی برای تجسم ساختار داده‌ها ارائه نمی‌دهد و ممکن است نسبت به انتخاب پارامترهای تراکم حساس باشد.

۳. انعطاف‌پذیری و کاربرد

SOM: از نظر سازگاری با انواع داده‌ها بسیار انعطاف‌پذیر است و می‌توان از آن برای کارهایی فراتر از خوشه‌بندی، مانند کاهش بعد و استخراج ویژگی استفاده کرد. SOMها در زمینه‌هایی مانند تشخیص الگو، تشخیص گفتار و حتی روباتیک به دلیل توانایی‌شان در مدل‌سازی روابط پیچیده درون داده‌ها، موفق بوده‌اند.

سایر روش‌ها: مانند الگوریتم‌های سلسله مراتبی و تقسیم‌بندی مانند k-means یا روش‌های مبتنی بر چگالی مانند DBSCAN، اغلب به کارهای خوشه‌بندی خاص محدود می‌شوند. آن‌ها ممکن است در برخی کاربردها عالی عمل کنند (مثلاً DBSCAN برای داده‌ها با خوشه‌های چگالی مشابه خوب است)، اما به اندازه SOM در کارهای خارج از خوشه‌بندی کاربردی نیستند.

۴. حساسیت پارامتری

SOM: در حالی که SOMها در بسیاری از تنظیمات مقاوم هستند، نیاز به تنظیم دقیق پارامترها مانند اندازه شبکه، نرخ یادگیری و تابع همسایگی دارند. این پارامترها به طور قابل توجهی بر کیفیت نقشه و توانایی آن در تعمیم از داده‌های آموزشی تأثیر می‌گذارند.

سایر الگوریتم‌ها: نیز نیاز به تنظیم پارامتر دارند (مانند تعداد خوشه‌ها در k-means یا آستانه چگالی در DBSCAN)، اما پارامترها به شکل‌های ساده‌تری بر مدل تأثیر می‌گذارند، مانند تأثیر مستقیم بر اندازه خوشه یا تعداد خوشه‌ها.

۵. پیچیدگی محاسباتی

SOM: می‌تواند از نظر محاسباتی پرهزینه باشد، به ویژه هنگامی که اندازه داده‌های ورودی و تعداد نورون‌ها در نقشه افزایش می‌یابد. با این حال، طبیعت تکراری SOM و سازگاری تدریجی نورون‌ها می‌تواند آن را برای سناریوهای یادگیری آنلاین مناسب سازد.

سایر الگوریتم‌ها: به طور قابل توجهی در نیازهای محاسباتی‌شان متفاوت هستند. به عنوان مثال، k-means نسبتاً کارآمد است، اما خوشه‌بندی سلسله مراتبی می‌تواند به دلیل پیچیدگی زمانی معمولاً مربعی خود با افزایش تعداد نمونه‌ها از نظر محاسباتی سنگین شود. روش‌های سلسله‌مراتبی گروه‌بندی داده‌ها را در سطوح چندگانه فراهم می‌کنند که برای برخی کاربردها مفید است، اما معمولاً با اندازه‌های بزرگ دیتاست به خوبی مقیاس‌پذیری ندارند و قابلیت‌های تجسمی مشابهی را ارائه نمی‌دهند.

دلایل اینکه SOMها می‌توانند رویکرد مناسبی برای این پروژه باشند:

۱. سازماندهی کارآمد داده‌ها

خوشه‌بندی و تجسم: SOMها در خوشه‌بندی داده‌های چندبعدی به نمایندگی دوبعدی برجسته هستند ضمن حفظ خصوصیات توپولوژیکی داده‌ها. این امر تجسم و تفسیر روابط بین نقاط مختلف داده را آسان‌تر می‌کند که برای وظایف جستجوی معنایی که درک نزدیکی مفاهیم حیاتی است، بی‌نظیر است.

کاهش بعد: SOMها پیچیدگی داده‌ها را با نقشه‌برداری آن‌ها به فضای کم‌بعدی کاهش می‌دهند. این کاهش می‌تواند فرآیند جستجو را ساده‌سازی کند زیرا محاسبه فاصله‌ها بین نقاط داده سریع‌تر و آسان‌تر می‌شود.

۲. خصوصیات حفظ توپولوژی

شباهت داده‌ها: SOMها تضمین می‌کنند که نقاط داده مشابه در نزدیکی یکدیگر روی نقشه قرار گیرند. این خاصیت می‌تواند برای جستجوهای معنایی استفاده شود، زیرا شناسایی سریع خوشه‌ها یا مناطق نقشه که اطلاعات مرتبط احتمالاً در آنجا یافت می‌شود را امکان‌پذیر می‌سازد.

جهت‌یابی در داده‌ها: با SOMها، جهت‌یابی از طریق داده‌های خوشه‌ای شده بیشتر شهودی می‌شود. پردازش پرس‌وجوها با یافتن نزدیک‌ترین خوشه در شبکه SOM انجام می‌گیرد، که ممکن است فضای جستجو را کاهش دهد و کارایی فرآیند بازیابی را بهبود بخشد.

۳. قابلیت مقیاس‌پذیری و یادگیری تدریجی

رسیدگی به مجموعه داده‌های بزرگ: در حالی که SOMها برای راه‌اندازی منابع محاسباتی اولیه نیاز دارند، آنها نسبتاً کارآمد در رسیدگی به مجموعه داده‌های بزرگ پس از آموزش هستند. آنها می‌توانند اطلاعات جدید را بدون نیاز به آموزش کامل مجدد پردازش کنند، که برای پایگاه‌های داده پویا که داده‌های جدید به طور مداوم اضافه می‌شود، مناسب است.

یادگیری تدریجی: SOMها می‌توانند با تنظیم اندک وزن‌های خود به داده‌های جدید واکنش نشان دهند، به جای اینکه از ابتدا آموزش دیده شوند. این ویژگی برای کاربردهایی مانند جستجوی معنایی که مجموعه داده‌ها به طور مداوم تکامل می‌یابد، مفید است.

۴. قابلیت‌های پردازش موازی

سرعت: SOMها را می‌توان به گونه‌ای پیاده‌سازی کرد که از پردازش موازی بهره ببرند، در نتیجه محاسبه فاصله‌ها و تنظیم وزن‌ها در سراسر شبکه را تسریع می‌کند. این مزیت به ویژه هنگام مواجهه با پایگاه‌های داده وسیع که برای جستجوی معنایی لازم است، مفید است.

۵. کاهش بار محاسباتی

پیش‌خوشه‌بندی: با سازماندهی داده‌ها به صورت خوشه‌ها پیش از این، SOMها می‌توانند بار محاسباتی را در طی مرحله جستجو به طور قابل توجهی کاهش دهند. به جای مقایسه بردار پرس‌وجو با هر بردار داده در پایگاه داده، مقایسه‌ها می‌تواند محدود به بردارهای نماینده هر خوشه یا مناطق خاصی از شبکه SOM شود.

کاربرد در جستجوی معنایی

استفاده از SOMها برای سازماندهی داده‌های خارجی برای یک مدل زبانی شامل نقشه‌برداری داده‌های متنی به یک فرمت ساختاریافته (شبکه SOM) است که در آن شباهت‌های معنایی موقعیت‌های نزدیکی را تعیین می‌کند. در طی یک پرس‌وجو، به جای جستجو در تمام داده‌های خارجی، جستجو می‌تواند به بخش‌های مرتبط از شبکه SOM محدود شود، که زمان و منابع محاسباتی لازم را به طور چشمگیری کاهش می‌دهد.

مراجع:

ResearchGate Contributor. A Review on Clustering Techniques including Deep Learning, Fuzzy Logic, and Self-Organizing Maps." ResearchGate, ۲۰۱۹.

بخش 2

برای اینکه روند آموزش SOM را ببینیم، باید اول با مفهوم کوانتیزه سازی بردار اشنا شویم.

کوانتیزه کردن بردار یک تکنیک استفاده شده در پردازش سیگنال، فشرده‌سازی داده، و تشخیص الگو است که شامل کوانتایز کردن داده‌های پیوسته یا گسسته به مجموعه‌ای محدود از بردارهای نماینده، به نام بردارهای کدبوک یا مراکز کلان است. هدف از کوانتیزه کردن بردار کاهش انحراف بین داده‌های ورودی و بردارهای کدبوک است، تا به این ترتیب نمایش فشرده‌ای از داده حاصل شود که حداکثر اطلاعات را حفظ کند.

روند آموزش شبکه som:

- مقداردهی اولیه: در ابتدا، وزن‌های گره‌های شبکه (نورون‌ها) به صورت تصادفی تعیین می‌شوند. فرض کنید برای هر نورون i بردار وزن متناظر mi داشته‌باشیم که با بعد داده ورودی برابر است.
- رقابت: هنگامی که داده‌های ورودی ارائه می‌شوند، هر نورون برای برنده یا نورونی که بیشترین شباهت با داده‌ی ورودی دارد، رقابت می‌کند. این معمولاً با محاسبه فاصله بین داده و وزن‌هر نورون تعیین می‌شود، اغلب با استفاده از اندازه‌گیری‌هایی مانند فاصله اقلیدسی. بهترین واحد منطبق هستیم نورونی است که بردار وزن مشابه ترین به داده ورودی است.

$$c = \text{argmin}\, x(t) - m_i(t)$$

که c شماره بهترین واحد منطبق در iteration t است.

- همکاری: پس از پیدا کردن بهترین واحد منطبق، یک تابع همسایگی شعاع تأثیر را روی نقشه تعیین می‌کند. نورون‌هایی که در این شعاع قرار دارند تنظیم می‌شوند تا شبیه‌تر به بردار ورودی شوند. تابع همسایگی معمولاً در ابتدا بزرگ است و به مرور زمان کاهش می‌یابد، یادگیری را به تدریج متمرکز می‌کند. یک انتخاب معمول تابع گاوسی است که در مرکز بهترین واحد منطبق قرار دارد.

$$h(t) = e^{-||rc - ri||^2 / 2(\sigma)^2}$$

- تطبیق: نورون برنده (همراه با نورون‌های مجاور در برخی موارد) وزن‌های خود را تنظیم می‌کند تا به داده‌های ورودی شبیه‌تر شود. این فرآیند تنظیم باعث می‌شود که شبکه به طوری که با ساختار زیرین داده‌های ورودی تطابق داشته باشد، خود را سازماندهی کند. در این جا طبق قانون هب نزدیک می شویم.

$$mi(t+1) = m(t) + \alpha\, h(t)(x(t) - mi(t))$$

- تکرارها: مراحل ۲ و ۳ برای تعدادی تکرار یا تا رسیدن به همگرایی تکرار می‌شوند، این امر به شبکه اجازه می‌دهد تا به طور تدریجی سازماندهی خود را بهبود بخشد و الگوهای داده را بهتر ضبط کند.

در نهایت نوورن ها هر کدام به عنوان یک نمایه که نماینده زیرمجوعه‌ای از داده هاست، سازمان‌دهی شبکه را شکل می دهند.

مراجع:

Kohonen, T. (۱۹۹۰). The Self-organizing Map. Proceedings of the IEEE, ۷۸(۹), .۱۴۶۴-۱۴۸۰

بخش 3

در کد زیر مجموعه داده را بارگزاری کردم. تاریخ‌ها را برای نمایش روی som map نگه داشتم.

Pure df : برای اینکه متن اصلی بدون حذف واژگان اضافه و توکن‌سازی نگه‌دارم.

Labels: تاریخ وقوع هر اتفاق را نگه‌داشتم تا روی نقشه som نمایش‌دهم.

Load the dataset

```
# Load the dataset
columns = ['id', 'content']  columns: list (2)
df = pd.read_csv('WikipediaEvents.csv', header=None, names=columns)  df: DataFrame (473, 1)  columns: list (2)
df.drop('id', axis=1, inplace=True)  df: DataFrame (473, 1)
pure_df = df.copy()  pure_df: DataFrame (473, 2)  df: DataFrame (473, 1)
labels = df['content'].apply(lambda x: x.split('-', 1)[0])  labels: (0, 'January 1, 2022 ') (1, 'January 2, 2022 ')
labels  labels: (0, 'January 1, 2022 ') (1, 'January 2, 2022 ') (2, 'January 4, 2022 ') (3, 'January 5, 2022 ') (4,
Executed at 2024.04.25 13:28:34 in 26ms
```

در کد زیر با کمک کتابخانه nltk لغات اضافه را حذف‌کردم و توکن‌سازی کلمات را انجام دادم. همچنین واژگلن لغات را lower کردم.

Preprocess_text:

- حذف لغات اضافه
- توکن‌سازی
- کوچک‌کردن همه واژگان

Preprocessing Removing Stopwords, Tokenization

```
nltk.download('punkt')
nltk.download('stopwords')

stop_words = set(stopwords.words('english'))  stop_words: {'a', 'about', 'above', 'after', 'again', 'against', 'ain', 'all', 'am',

def preprocess_text(text):
    if isinstance(text, str):
        # Tokenization
        tokens = word_tokenize(text)
    elif isinstance(text, list):
        tokens = text
    else:
        raise TypeError("Input should be a string or a list of words")

    # Stop words removal
    new_tokens = [word.lower() for word in tokens if word not in stop_words]

    return new_tokens

df['content'] = df['content'].apply(preprocess_text)  df: DataFrame (473, 1)  df: DataFrame (473, 1)
Executed at 2024.04.25 13:28:35 in 188ms
```

در کد زیر یکی از وزن‌های مدل glove که متن را به بردارهایی ۱۰۰ تایی تبدیل می‌کند، دانلود کردم و با استفاده از کتابخانه genism آن را لود کردم.

```
# Load the GloVe model
glove_input_file = '../../glove.6B.100d.txt'   glove_input_file: '../../glove.6B.100d.txt'
word2vec_output_file = 'glove.6B.100d.txt.word2vec'   word2vec_output_file: 'glove.6B.100d.txt.wo
glove2word2vec(glove_input_file, word2vec_output_file)   glove_input_file: '../../glove.6B.100d.t
glove_model = KeyedVectors.load_word2vec_format(word2vec_output_file, binary=False)   glove_model
Executed at 2024.04.25 13:31:39 in 3m 1s 863ms
```

در کد زیر هر متن ستون content را وارد حلقه کردم، برای هر سطر هر لغتش را تبدیل به بردارکردم و بعد بردار ها را با هم میانگین گرفتم.

```
def check_existence(word, model, vector_size):
    try:
        return model[word]
    except:
        return np.zeros(vector_size)
data = pd.DataFrame(df['content'].apply(lambda words: np.mean([check_existence(word, glove_model, 100) for word in words], axis=0)))
data = pd.DataFrame(data['content'].to_list())   data: DataFrame (473, 101)   data: DataFrame (473, 101)
Executed at 2024.04.25 14:31:52 in 113ms
```

بخش 4

توضیح در مورد هر پارامتر Minisom:

x (int): تعداد ستون‌ها در شبکه SOM. این پارامتر بعد افقی شبکه را مشخص می‌کند.

y (int): تعداد ردیف‌ها در شبکه SOM. این پارامتر بعد عمودی شبکه را تعیین می‌کند.

input_len (int): تعداد ویژگی داده ورودی

sigma(float): شعاع تابع همسایگی در ابتدای آموزش. این پارامتر تأثیرگذاری همسایگی نورون برنده را در طول فرآیند آموزش تحت تأثیر قرار می‌دهد. این مقدار بر اساس تابع تحلیل پذیری با گذشت زمان کاهش می‌یابد تا تنظیمات دقیق‌تری هنگام پیشرفت آموزش امکان‌پذیر شود. این پارامتر دارای پیش‌فرض ۰.۱ است.

learning_rate(float): نرخ یادگیری اولیه، که بر اندازه به‌روزرسانی‌های انجام شده بر وزن‌های SOM در طول آموزش تأثیر می‌گذارد. این مقدار نیز بر اساس تابع تحلیل پذیری با گذشت زمان کاهش می‌یابد تا یادگیری هنگام همگرایی مدل پایدار شود. این پارامتر دارای مقدار پیش‌فرض ۰.۵ است.

decay_function: تابعی که sigma و learning_rate را به مرور زمان کاهش می‌دهد. این تابع سه پارامتر را می‌پذیرد: مقدار فعلی (نرخ یادگیری یا سیگما)، شماره تکرار فعلی، و حداکثر تعداد تکرارها. این به تنظیم دقیق مدل کمک می‌کند. می توان یک تابع عمومی برای اینکه نرخ که sigma و learning_rate را به روش دلخواه خودمان کاهش دهیم.

تابع پیش فرض برای این پارامتر asymptotic_decay هست. که فرمول ان طبق متون کتابخانه به صورت زیر است.

```
    where T is #num_iteration/2)

decay_function : function (default=asymptotic_decay)
    Function that reduces learning_rate and sigma at each iteration
    the default function is:
                learning_rate / (1+t/(max_iterarations/2))
```

برای اینکه تابع عمومی تعریف کنیم باید شروط زیر را داشته باشد.

```
    A custom decay function will need to to take in input
    three parameters in the following order:

    1. learning rate
    2. current iteration
    3. maximum number of iterations allowed


    Note that if a lambda function is used to define the decay
    MiniSom will not be pickable anymore.
```

neighborhood_function: تابعی که برای محاسبه تأثیر به‌روزرسانی نورون بر همسایه‌های آن استفاده می‌شود. گزینه‌ها شامل 'gaussian'، 'mexican_hat'، 'bubble'، و 'triangle' می‌شوند. هر تابع توزیع مکانی متفاوتی از یادگیری در اطراف نورون برنده را تعیین می‌کند.

```
neighborhood_function : string, optional (default='gaussian')
    Function that weights the neighborhood of a position in the map.
    Possible values: 'gaussian', 'mexican_hat', 'bubble', 'triangle'
```

Topology: شکل نقشه. گزینه‌های ممکن 'rectangular' و 'hexagonal' هستند. این تأثیری بر نحوه چیدمان نورون‌ها و تعامل آن‌ها با یکدیگر دارد.

```
topology : string, optional (default='rectangular')
    Topology of the map.
    Possible values: 'rectangular', 'hexagonal'
```

activation_distance: معیار فاصله استفاده شده برای شناسایی بهترین واحد مطابقت (نورون برنده) برای هر بردار ورودی. گزینه‌ها شامل 'euclidean'، 'cosine'، 'manhattan'، و 'chebyshev' می‌شوند. توابع سفارشی نیز می‌توانند برای محاسبه انواع دیگر فاصله‌ها ارائه شوند.

```
activation_distance : string, callable optional (default='euclidean')
    Distance used to activate the map.
    Possible values: 'euclidean', 'cosine', 'manhattan', 'chebyshev'

    Example of callable that can be passed:

    def euclidean(x, w):
        return linalg.norm(subtract(x, w), axis=-1)
```

random_seed : اگر مقداری دلخواه تعیین شود، باعث می‌شود که هر دفعه ک ما آموزش می بینیم با مقدار رندم شروع نکند و با مقادیر قبلی کار را شروع می‌کند.

حال در کد برای اینکه بهترین هایپرپارامتر را برای استفاده از مدل به دست آوردم. و در نهایت خروجی را در کد همانطور در شکل زیر مشاهده می کنید گزارش دادم.

Train and Tune Hyperparmeter the MiniSom Model, Get Winning Units

```
grid_sigma = [0.1, 0.5, .01]
grid_learning_rate = [0.1, 0.5, 0.01]
grid_x_y = [(2,2), (6,6), (10,10)]
num_features = data.shape[1]
best_params = None
best_score = np.inf
for sigma in grid_sigma:
    for learning_rate in grid_learning_rate:
        for x_y in grid_x_y:
            som = MiniSom(x=x_y[0], y=x_y[1], input_len=num_features, sigma=sigma, learning_rate=learning_rate, neighborhood_function='gaussian')
            som.random_weights_init(data.values)
            som.train_random(data.values, num_iteration=10000, verbose=True)

            score = som.quantization_error(data.values)
            if score < best_score:
                best_params = {'sigma': sigma, 'learning_rate': learning_rate, 'x_y': x_y}
                best_score = score
                best_som = som

# Print the best parameters and the corresponding score
print(f"Best HyperParameter for SOM model in our Data: {best_params} with score: {best_score}")
som = best_som
winning_units = np.array([som.winner(x) for x in data.values])
```

Executed at 2024.04.25 16:25:18 in 35s 591ms

در نهایت بعد از آموزش مدل پارامتر های زیر به دست آمد.

- x: ۱۰
- y: ۱۰
- sigma: ۰.۱

- Learning_rate: ۰.۵
- Neighborhood_function: gaussian
- بقیه هم مقدار ست شده خود کتابخانه را دارند.

```
quantization error: 0.9682613896974946
[ 10000 / 10000 ] 100% - 0:00:00 left
quantization error: 0.7756139870418185
[ 10000 / 10000 ] 100% - 0:00:00 left
quantization error: 1.1996410527398578
[ 10000 / 10000 ] 100% - 0:00:00 left
quantization error: 0.9825785556436287
[ 10000 / 10000 ] 100% - 0:00:00 left
quantization error: 0.8348580797346502
Best HyperParameter for SOM model in our Data: {'sigma': 0.1, 'learning_rate': 0.5, 'x_y': (10, 10)} with score: 0.7671698298316052
```

Add the Winning Units Columns to the Data

```
1  data['winning_unit'] = list(map(str, winning_units))   data: DataFrame (473, 101)   winning_units: ndarray (473, 2)
2  pure_df['winning_unit'] = list(map(str, winning_units))   pure_df: DataFrame (473, 2)   winning_units: ndarray (473, 2)
Executed at 2024.04.25 14:32:09 in 80ms
```

```
1  pure_df   pure_df: DataFrame (473, 2)
Executed at 2024.04.25 15:40:50 in 57ms
```

1-10 | 473 rows × 2 columns pd.Dataframe | CSV

	content	winning_unit
0	January 1, 2022 - The Regional Comprehensive Ec…	[2 3]
1	January 2, 2022 - Abdalla Hamdok resigns as Pri…	[8 5]
2	January 4, 2022 - The five permanent members of…	[2 2]
3	January 5, 2022 - A nationwide state of emergen…	[5 7]
4	January 6, 2022 - The CSTO deploys a "peacekeep…	[7 3]
5	January 7, 2022 - COVID-19 pandemic: The number…	[2 1]
6	January 9, 2022 - February 6 - The 2021 Africa …	[3 5]
7	January 10, 2022 - The first successful heart t…	[1 3]
8	January 15, 2022 - A large eruption of Hunga To…	[1 5]
9	January 16, 2022 - World No. 1 tennis champion …	[0 2]

بخش 5

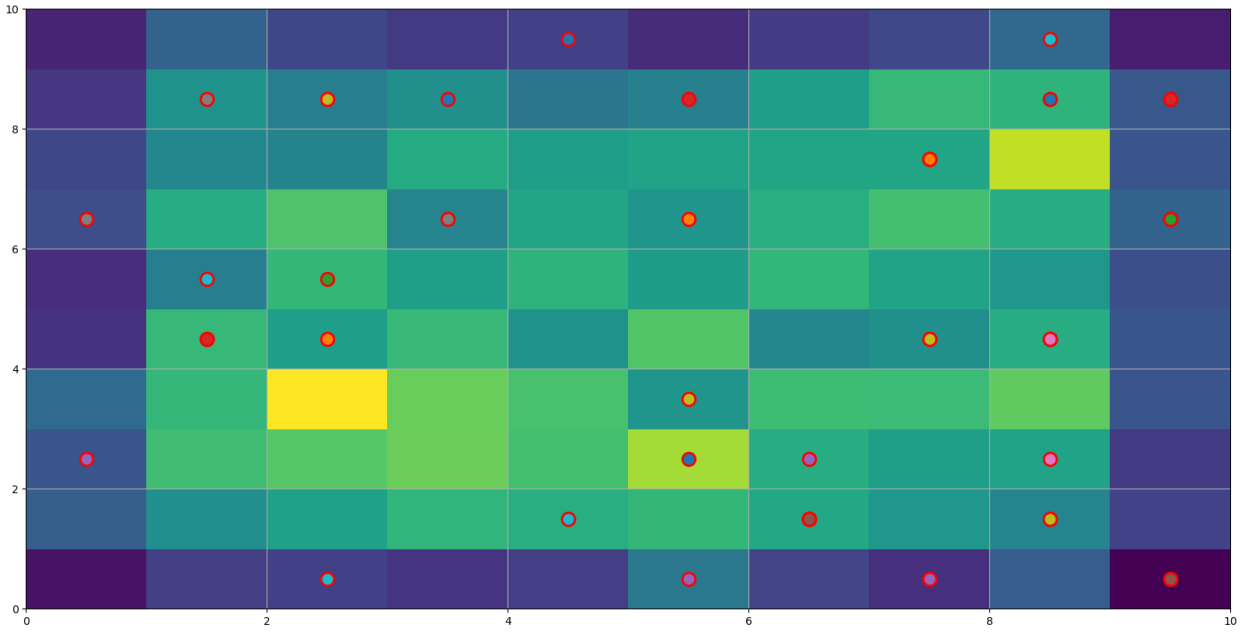
کد زیر نقشه خروجی نورون های som را برای ۵۰ نمونه تصادفی می کشد.

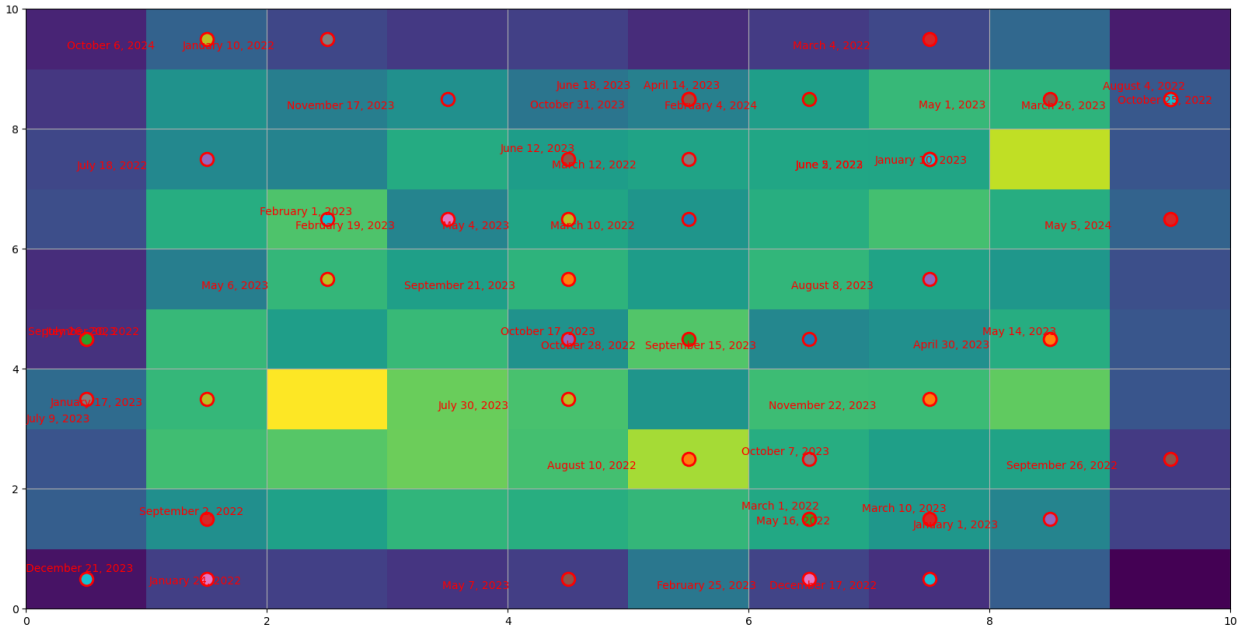
Grid MAP SOM neurons (Randomly Select 50 Events and Plot the Winning Units)

```
# Randomly select 50 events
selected_events = data.sample(n=50)  selected_events: DataFrame (50, 100)  data: DataFrame (473, 100)
selected_winning_unit = winning_units[selected_events.index]  selected_winning_unit: ndarray (50, 2)  winning_units: ndarray (473, 2)  selected_events: DataFrame (50, 100)
selected_labels = labels[selected_events.index]  selected_labels: (247, 'April 14, 2023 ') (334, 'September 21, 2023 ') (266, 'May 14, 2023 ') (436, 'May 5, 2024 ') (349, 'October 17, 20
plt.figure(figsize=(20, 10))
plt.pcolor(som.distance_map().T,)  som: <minisom.MiniSom object at 0x00000133B2067090>
texts = []  texts: list (50)
# Compute the U-matrix
u_matrix = som.distance_map()  # This computes the U-matrix of the SOM  u_matrix: ndarray (10, 10)  som: <minisom.MiniSom object at 0x00000133B2067090>

for w, label in zip(selected_winning_unit, selected_labels):  selected_winning_unit: ndarray (50, 2)  selected_labels: (247, 'April 14, 2023 ') (334, 'September 21, 2023 ') (266, 'May
    plt.plot(w[0] + 0.5, w[1] + 0.5, 'o', markeredgecolor='r',  markersize=12, markeredgewidth=2)  w: [7 0]  w: [7 0]
    texts.append(plt.text(w[0] , w[1] + 0.5, label, ha='center', va='center', color='red', fontsize=10))  texts: list (50)  text: ['September 21, 2023 - Rupert Murdoch announces his re
adjust_text(texts)  texts: list (50)
# Plot the U-matrix
plt.pcolor(u_matrix.T)  u_matrix: ndarray (10, 10)
# adjust the positions of the text labels
plt.axis([0, som.get_weights().shape[0], 0, som.get_weights().shape[1]])  som: <minisom.MiniSom object at 0x00000133B2067090>  som: <minisom.MiniSom object at 0x00000133B2067090>
plt.grid()
plt.show()
Executed at 2024.04.25 17:35:27 in 3s 844ms
```

برای رسم شکل زیر از U-matrix استفاده می کنیم. که ابزاری برای تصویرسازی است که برای تفسیر نقشه‌های خودسازمان‌ده (SOM) استفاده می‌شود. اساساً، U-Matrix فاصله‌ها بین نورون‌ها (یا گره‌ها) در شبکه SOM را نشان می‌دهد. این ماتریس بینشی در مورد ساختار خوشه‌بندی داده‌هایی که روی SOM نقشه‌برداری شده‌اند، با نمایش شباهت نسبی بین نورون‌های مختلف ارائه می‌دهد.





تفسیر هایی که می توان برای شکل بالا داشت.

- در شکل بالا با توجه به رنگ‌ها می توان فاصله نورون‌ها را از هم مشاهده‌کرد.
- با استفاده از Umtrix می توان ناهنجاری‌ها را تشخیص‌داد، اما هدف ما در این سوال این نیست.
- می‌توان از آن برای تشخیص خوشه‌ها استفاده کرد. نورون‌های که فاصله نزدیکی با هم دارند؛ را می‌توان یک خوشه در نظر گرفت.
- بر خلاف دیگر خوشه بندی داده‌های ما اگر در فضای بالا نزدیک هم باشند، som هم می تواند این کار را انجام دهد یعنی som توپولوژی داده را حفظ می کند.

بخش 6

طبق مراحلی که سوال خواسته جلو می رویم:

در مرحله اول داده سوالات را لود کردم پیش پردازش را اعمال کردم. آن را به بردار تبدیل کردم و طبق قسمت های قبل بردار ها را میانگین گرفتم برای هر سوال.

Load Questions and Preprocess and Convert to Vectors

```
qa = pd.read_table("Sample_questions.txt")
selected_questions = qa.copy()
selected_questions['questions'] = selected_questions['questions'].apply(preprocess_text)
qa_data = pd.DataFrame(selected_questions['questions'].apply(lambda words: np.mean([check_existence(word, glove_model, 100) for word in words], axis=0)))
qa_data = pd.DataFrame(qa_data['questions'].to_list())
qa_data
```

Executed at 2024.04.25 14:51:15 in 65ms

14 rows × 100 columns pd.Dataframe

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0.236035	0.741275	0.700114	-0.640573	-0.314830	0.189826	0.103110	0.263565	-0.385023	-0.176869	0.246493	-0.353437	0.099526	-0.117618	-0.325
1	-0.256074	0.341562	0.532542	-0.220475	-0.149556	0.089856	0.061925	-0.013630	0.136688	-0.156098	0.155997	-0.249021	0.061524	0.347821	0.027
2	-0.048132	0.394433	0.241857	-0.293370	-0.164610	0.176291	0.047400	-0.063067	0.111410	-0.051019	0.143254	-0.288612	0.345714	0.121570	0.024
3	-0.036308	0.353315	0.202847	-0.325084	-0.095202	0.101594	-0.056726	-0.043409	0.106152	0.028932	0.208922	-0.256862	0.295652	0.101262	0.015
4	-0.111098	0.087070	0.645264	-0.335650	0.146662	0.115618	0.254202	-0.000368	0.013698	0.040466	0.188262	-0.528798	0.334603	0.223130	-0.081
5	-0.120970	0.107584	0.682126	-0.312577	0.166778	0.146736	0.268182	0.051282	0.006518	0.018522	0.111398	-0.516228	0.340372	0.199134	-0.102
6	0.032992	0.645782	0.623384	-0.322136	-0.527332	0.141340	0.065054	0.224342	-0.494770	-0.216030	0.235785	-0.300272	0.223491	-0.003215	-0.187
7	-0.147866	0.426529	0.680129	-0.128875	-0.343741	0.065327	0.019532	0.352182	-0.531832	-0.121987	0.285270	-0.301825	0.054603	0.100487	-0.204
8	-0.412165	-0.057214	0.605179	-0.077150	0.168243	-0.193899	-0.118774	-0.137969	0.073833	0.070841	-0.041126	0.022936	0.189911	0.373065	0.077
9	-0.165413	-0.065799	0.673076	0.056786	-0.257567	-0.091913	0.108809	0.247559	-0.021642	-0.084594	0.137231	0.194396	0.359868	0.072772	0.275

دراین مرحله هر سوال به مدل som دادم و بهترین واحد منطبق آن را در آوردم و با متن های نورون که قبلا وجود داشته مقایسه کردم تا cosine_similarity به دست آورم. البته این کار را برای تمام پرسش ها انجام دادم. در نهایت برای هر سوال موارد زیر گزارش دادم:

- نورون برنده برای آن پرسش(نمایه مناسب)
- تعداد داده های حاضر در ان نورون
- خود متن پرسش
- بیش‌ترین cosine_similarity برای آن پرسش مورد نظر با مقایسه با داده های ان نورون(اگر بیش ترین نزدیکی بیش از ۰.۸۵ بود متن جواب را پرینت می کردم که در اکثر موارد درست در می امد)

```
for question_text, question in zip(qa.values, qa_data.values):
    question = question.reshape(1, -1)
    # Find the appropriate index corresponding to the question
    question_index = som.winner(question)
    question_index = f"[{question_index[0]} {question_index[1]}]"
    external_added_clusters_dict[question_index] += 1
    relevant_data = data[data['winning_unit'] == question_index]
    pure_relevant_data = pure_df[pure_df['winning_unit'] == question_index]


    # Calculate the most relevant data with cosine similarity measure
    sim = {}
    i = 0
    for relevant, text in zip(relevant_data.values, pure_relevant_data.values):
        relevant = relevant[:-1].reshape(1, -1)
        similarities = cosine_similarity(question, relevant)
        sim[i] = {"similarity": similarities[0][0], "text": text}
        i = i + 1
    max_sim = max(sim.values(), key=lambda x: x['similarity'])['similarity']
    if max_sim > 0.85:
        print(f"The cluster for question: '{question_text[0]}' is {question_index} \n number of relevant data in the index: {len(relevant_data)} \n maximum
         relevant data cosine similarity: {max_sim} \n most relevant data: {max(sim.values(), key=lambda x: x['similarity'])['text'][0]}")
    else:
        print(f"The cluster for question: '{question_text[0]}' is {question_index} \n number of relevant(external) data: {len(relevant_data)} \n maximum relevant
         data cosine similarity: {max_sim}")
    print("----------------------------------------------------------------------------------------------------")
```

Executed at 2024.04.25 14:53:19 in 88ms

```
The cluster for question: 'Who won the 2022 soccer world cup?' is [4 4]
 number of relevant data in the index: 2
 maximum relevant data cosine similarity: 0.8748012436848903
 most relevant data: September, 2024 - 2024 ICC Women's T20 World Cup.
-----------------------------------------------------------------------------------------------
The cluster for question: 'When did Sweden join NATO?' is [0 3]
 number of relevant data in the index: 5
 maximum relevant data cosine similarity: 0.8924374596932739
 most relevant data: March 7, 2024 - As the final Nordic country to join the alliance, Sweden officially joins NATO, becoming its 32nd member after Finland a
  year earlier.
-----------------------------------------------------------------------------------------------
The cluster for question: 'Who joined NATO in 2023?' is [0 5]
 number of relevant(external) data: 18
 maximum relevant data cosine similarity: 0.8333107559335835
-----------------------------------------------------------------------------------------------
The cluster for question: 'Who joined NATO in 2024?' is [0 5]
 number of relevant(external) data: 18
 maximum relevant data cosine similarity: 0.8293136871331017
-----------------------------------------------------------------------------------------------
The cluster for question: 'Which is the 31st member of NATO?' is [0 3]
 number of relevant data in the index: 5
 maximum relevant data cosine similarity: 0.904177406694077
 most relevant data: April 4, 2023 - Finland becomes the 31st member of NATO, doubling the alliance's border with Russia.
-----------------------------------------------------------------------------------------------
```

در این جا تعداد داده های خارجی که اضافه شده به دست آوردم.

Print the External Data Added to the Clusters

```
for i in range(som.get_weights().shape[0]):   som: <minisom.MiniSom object at 0x00000133C273AB90>
    for j in range(som.get_weights().shape[1]):   som: <minisom.MiniSom object at 0x00000133C273AB90>
        if external_added_clusters_dict[f'[{i} {j}]']:   external_added_clusters_dict: {'[0 0]': 1, '[0 1]': 0, '[0 2]': 0, '[0 3]': 3, '[0 4]': 0, '[0 5]': 2, '[0 6]'
            print(f"External data added to this {f'[{i} {j}]'} index is {external_added_clusters_dict[f'[{i} {j}]']}")   i: 9    j: 9    external_added_clusters_dict:
print("Every Other Cluster is 0")
```

Executed at 2024.04.25 14:54:17 in 33ms

```
External data added to this [0 0] index is 1
External data added to this [0 3] index is 3
External data added to this [0 5] index is 2
External data added to this [4 4] index is 3
External data added to this [8 1] index is 3
External data added to this [9 2] index is 2
Every Other Cluster is 0
```

دلایلی که معیار کسینوسی در اینجا مخصوصا مقایسه داده های متنی بهتر است:

۱- اندازه گیری زاویه: برای داده های متنی که بحث تکرار مطرح است اندازه‌گیری زاویه بسیار می تواند مناسب باشد.

۲- نرمال شده و بازه مشخص: به دلیل اینکه دارای بازه بین -۱ و ۱ است، می تواند دید خوبی از عملکرد مدل و نزدیکی داده‌ها به هم به ما بدهد.

۳- مقاوم بودن: به دلیل اینکه متون اندازه متفاوتی دارد شاید سخت باشد از معیار دیگر استفاده کرد ولی این معیار می تواند به راحتی این کار را انجام دهد.

٤- داده های sparse: این معیار به خوبی می تواند با این نوع داده‌ها که متن هم یکی از آن ها محسوب می شود به خوبی کار کند.

٥- مدل های پیش آموزش دیده: طوری که مدل های زبانی آموزش می بینند تا مفهوم را دل خود داشته باشند با معیار کسینوسی بهتر ارتباط می گیرند.