



مبانی داده کاوی

گزارش پروژه

اعضای گروه:

محمد امین نصرتی 9932103، امیرحسین منصوری 9931123

فاز شناخت و فهم مسئله

معرفی مسئله
معرفی سوال و معیار ارزیابی و داده ها
بیان مختصر ایده های تیم
مرور کار های مرتبط

فاز آماده سازی داده ها

تشخیص داده های پرت
تبدیل و استاندارد سازی داده ها

فاز تحلیل اکتشافی داده ها

بررسی و تحلیل روابط تک متغیره بین متغیرهای پیش‌بین و متغیر هدف
بررسی روابط چند متغیره بین متغیرها

فاز پیش مدل (Setup)

چارچوب تقسیم داده ها و به دسته آموزش و تست
روش انجام Cross-Validation
بالانس داده ها

فاز مدل سازی

انتخاب و پیاده سازی الگوریتم‌های لازم
اطمینان از عملکرد بهتر نسبت به موارد قبلی (با بیان دلایل و تفسیر نتایج
بکارگیری توسعه ها و روش های پیشرفته به شرط اثبات در قالب یک مقاله

فاز ارزیابی

معرفی مجموعه معیارهای ارزیابی و محاسبه آن ها و تفسیر نتایج
برآورد خطا
گزارش آزمایشات برگشت به فاز های قبلی برای بهبود ارزیابی
مشخص کردن بهترین مدل از بین مدل های اجرا شده همراه با پارامترهای تعیین شده تحلیل نقاط قوت و ضعف
کار انجام شده و پیشنهادات برای بهبود آینده

فاز شناخت و فهم مسئله

در این فاز به معرفی مسئله، معرفی سوال و معیار ارزیابی داده‌ها، بیان مختصر ایده های تیم و مرور کارهای مرتبط می پردازیم.

معرفی مسئله

استخراج الگوهای اتصالی پرتکرار مغزی در ADHD

اختلال نقص توجه و بیش‌فعالی (ADHD) یکی از شایع‌ترین اختلالات روانپزشکی است که بر کودکان و بزرگسالان تأثیر می‌گذارد. شناسایی و تشخیص دقیق این اختلال از اهمیت بالایی برخوردار است. در این پژوهش، هدف ما بررسی و استخراج الگوهای اتصالی پرتکرار مغزی در افراد مبتلا به ADHD با استفاده از داده‌های fMRI با اندازه‌گیری میزان مصرف اکسیژن نقاط مختلف مغز است.

دلایل استفاده از داده‌های مصرف اکسیژن مغزی

فعالیت مغزی رابطه مستقیمی با میزان انرژی مصرف شده دارد. هنگامی که یک نقطه از مغز فعال می‌شود، نیاز به انرژی بیشتری دارد که این انرژی از طریق افزایش جریان خون و اکسیژن‌رسانی تأمین می‌شود. بنابراین، مصرف اکسیژن در نقاط مختلف مغز می‌تواند به عنوان نشانگری از سطح فعالیت آن نقاط در نظر گرفته شود. با اندازه‌گیری مصرف اکسیژن در نقاط مختلف مغز، می‌توان الگوهای فعالیت مغزی را به دست آورد و از این طریق به شناخت بهتر عملکرد مغز در افراد مبتلا به ADHD رسید.

فرضیه تحقیق

ما بر این باوریم که در افراد مبتلا به ADHD، برخی نقاط مغزی هنگام بروز علائم این اختلال به صورت همزمان فعالیت می‌کنند. این بدین معناست که این نقاط مغزی به طور همزمان انرژی بیشتری مصرف می‌کنند یا فعالیت خود را کاهش می‌دهند. این الگوی هماهنگ در مصرف انرژی می‌تواند نشان‌دهنده اتصالات عملکردی خاصی در مغز این افراد باشد.

هدف

با در دست داشتن داده‌های مربوط به میزان مصرف انرژی نقاط مختلف مغز، قصد داریم بررسی کنیم که چگونه مصرف انرژی در این نقاط با یکدیگر مرتبط است و این الگوها در افراد مبتلا به ADHD به چه صورت می‌باشد. هدف ما این است که با تحلیل این روابط، به درک بهتری از الگوهای اتصالی مغزی در افراد مبتلا به ADHD برسیم و هنگامی که فرد جدید را مورد آزمایش قرار دادیم بتوانیم بفهمیم که داراییه ADHD می باشد یا خیر به نوعی مسئله از دو بخش استخراج ویژگی و تقسیم بندی (classification) تشکیل شده است

روش تحقیق

داده‌ها:

برای این پژوهش، بخشی از داده‌های مربوط به ۳۵۰ نقطه از مغز در ۷۰ زمان مختلف جمع‌آوری شده است. این داده‌ها شامل اندازه‌گیری‌های مصرف اکسیژن در هر نقطه از مغز در طول زمان‌های مختلف است. داده‌های مورد استفاده در این تحقیق از مقالات معتبر و منابع علمی همچون مقاله "pyClusterROI" استخراج شده است.

طریقه جمع‌آوری داده‌ها

شرکت‌کنندگان:

کنترل‌های سالم: ۴۱ نفر (۱۸-۵۵ سال، میانگین: ۳۱.۲، انحراف معیار: ۷.۸، ۱۹ زن) از دانشگاه Emory، که هیچ گونه شرایط نورولوژیک یا روانپزشکی نداشتند.

نمونه ADHD 200: داده‌های ۳۸۳ فرد مبتلا به ADHD و ۵۹۸ فرد کنترل سالم (۷-۲۱ سال) از ۸ سایت مختلف، که همه‌ی داده‌ها به صورت ناشناس در اختیار قرار گرفته‌اند.

تصویربرداری:

کنترل‌های سالم: اسکنر T Siemens Magnetom TIM Trio 3.0 با رزولوشن $1 \times 1 \times 1 \text{ mm}^3$ و توالی MPRAGE برای تصاویر آناتومیک.

نمونه ADHD 200: داده‌های fMRI و MRI ساختاری، پردازش شده توسط Athena Pipeline از Neuro Bureau's ADHD 200 Preprocessing Initiative.

پیچیدگی مسئله

مسئله مورد بررسی با چالش‌های زیادی مواجه است که از جمله این چالش‌ها می‌توان به موارد زیر اشاره کرد

تعداد زیاد نقاط مغز:

تعداد بالای نقاط مغز (معمولاً به عنوان ویژگی‌ها یا فیچرها شناخته می‌شوند) باعث افزایش پیچیدگی مسئله می‌شود. این امر باعث می‌شود که پیدا کردن الگوهای اتصالی مغزی که با اختلال ADHD مرتبط هستند پیچیده‌تر باشد

تعداد زمان‌های بررسی شده برای هر نقطه:

به دلیل کم بودن اندازه سیگنال (70) برای هر داده این امر باعث می‌شود که نتواند مدل ما با توجه به زیاد بودن نقاط به خوبی یاد بگیرد

نیاز به پیش‌پردازش داده‌ها:

بدون اعمال پیش‌پردازش به داده‌ها، امکان استفاده از روش‌های یادگیری ماشینی یا شبکه‌های عصبی برای حل مسئله به صورت موثر واقعی وجود ندارد. به دلیل پیچیدگی بالای داده‌ها و نویزهای ممکن در آن‌ها، پیش‌پردازش داده‌ها ضروری است.

انتخاب ویژگی‌ها مناسب:

از آنجایی که تعداد زیادی از ویژگی‌ها ممکن است موثر نباشند یا حتی به افزایش پیچیدگی مسئله منجر شوند، لازم است که قبل از استفاده از روش‌های یادگیری، ویژگی‌های مناسب انتخاب و یا کاهش بعضی از ویژگی‌ها صورت گیرد.

با توجه به این چالش‌ها، اجرای روش‌هایی مانند استخراج ویژگی‌ها (فیچر اکسترکشن)، استفاده از الگوریتم‌های یادگیری ماشینی و داده کاوی از اهمیت بالایی برخوردار است تا بتوانیم به درک و حل مسئله بهتری دست یابیم.

معرفی سوال و معیار ارزیابی و داده ها

معرفی صورت سوال

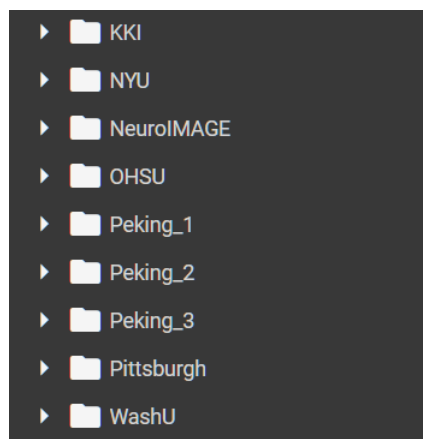
ما در این سوال قصد داریم با گرفتن سیگنال مصرف اکسیژن نقاط مغزی و انجام پیش پردازش روی آن ها و همچنین ساخت مدل برای تشخیص ADHD افراد مبتلا به این بیماری اقدام نماییم

معرفی معیار ارزیابی

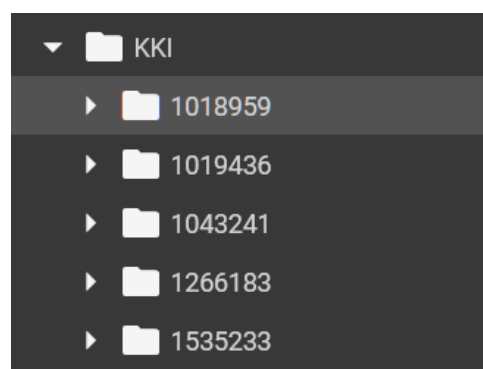
ما برای بررسی اینکه آیا مدل مان نتیجه مورد نظر خود را گزارش میکند از مدل های یادگیری با نظارت استفاده کرده ایم و از معیار ارزیابی accuracy برای صحت سنجی مدل مان استفاده کرده ایم

معرفی مجموعه داده

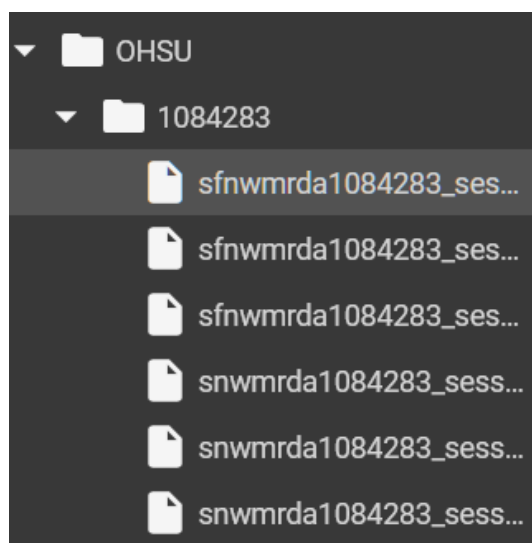
مجموعه داده متشکل از داده های ضبط شده در چندین مرکز است، داده های ضبط در هر مرکز، پوشه مربوطه تجمیع شده است، شکل زیر شمایی از این پوشه ها را ارائه می دهد.



در ذیل پوشه هر مرکز، پوشه های مربوط به هر شرکت کننده قرار دارد که با شناسه ای یکتا نامگذاری شده است.



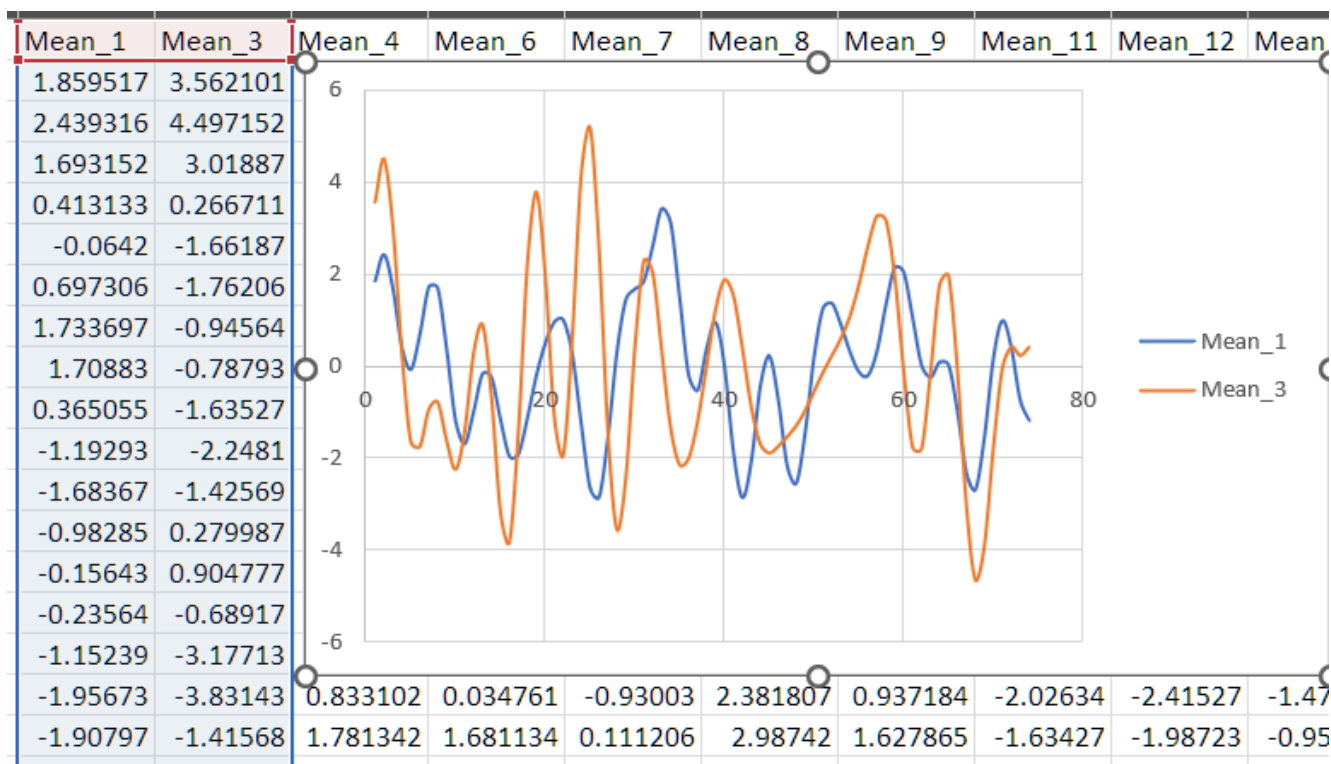
از هر فرد چند رکورد گرفته شده که در دو ورژن فیلتر شده و فیلتر نشده در دیتاست موجود می باشد. ورژن های فیلتر شده با نام sfm شروع می شوند. لازم به ذکر است هر رکورد یک فایل D1 می باشد.



در شکل زیر نمایی از هر رکورد دیتاست را مشاهده می کنید. در مجموع 1168 رکورد فیلتر شده در دیتاست موجود می باشد.

	File	Sub-brick	Mean_1	Mean_3	Mean_4	Mean_6	Mean_7	Mean_8	Mean_9	Mean_11	...
0	sfnmrda1735881_session_1_rest_1.nii.gz	0[?]	-0.517987	3.427561	-2.148021	-6.815048	0.620432	0.355868	-0.557647	-2.004494	...
1	sfnmrda1735881_session_1_rest_1.nii.gz	1[?]	-4.026902	5.466705	-3.553522	-9.352972	0.267772	1.228278	-0.585813	-2.479551	...
2	sfnmrda1735881_session_1_rest_1.nii.gz	2[?]	-5.876693	5.171324	-2.713049	-6.606369	-0.033277	1.131013	-0.753898	-0.767932	...
3	sfnmrda1735881_session_1_rest_1.nii.gz	3[?]	-5.064620	1.947243	0.339070	0.319957	0.006022	-0.583068	-1.135782	1.955628	...
4	sfnmrda1735881_session_1_rest_1.nii.gz	4[?]	-2.372915	-2.258673	3.644671	6.102208	0.187876	-2.756585	-1.435164	3.870939	...

هر رکورد، متشکل از 353 ستون است که ستون اول نام فایل و ستون دوم شماره نمونه و 351 ستون بعدی، میانگین مصرف اکسیژن در نقطه A ام از مغز را مشخص می کنند. هر سطر یک نمونه است که شماره نمونه



در ستون دوم مشخص شده و سطر های متوالی نشانگر نمونه های متوالی گرفته شده در طول آزمایش

هستند. برای درک بهتر، روند میانگین مصرف اکسیژن در نقطه 1 و 3 در شکل زیر قابل مشاهده است.

اگر این دو روند به هم شبیه باشند، گوییم این دو نقطه از مغز به هم متصل هستند.

ادعا می شود در افراد دارای ADHD این روند مربوط به اکسیژن در برخی از این نقاط مشابه یکدیگر است که در افراد فاقد اختلال ADHD مشاهده نمی شود. در پوشه مربوط به هر مرکز یک فایل CSV با پسوند '_phenotypic.csv' موجود است که در ستون "ScanDir ID" آن شناسه بیمار قرار داشته و در ستون "DX" تشخیص اختلال ADHD قرار دارد. این ستون می تواند چهار مقدار داشته باشد:

- 0: فرد نرمال است.
- 1: فرد دارای ADHD-Combined است.
- 2: فرد دارای ADHD-Hyperactive/Impulsive است.
- 3: فرد دارای ADHD-Inattentive است.

در پروژه‌ی حاضر با هدف استخراج الگوهای مکرر در ارتباطات مغزی برای تشخیص ADHD، ایده‌های مختلفی مطرح و پیاده‌سازی شدند. تیم تحقیقاتی ما به منظور رسیدن به این هدف از مجموعه‌ای از روش‌ها و الگوریتم‌های متنوع استفاده کرده است. در ادامه به بیان مختصر این ایده‌ها و دلایل انتخاب هر یک از آنها می‌پردازیم.

1. استفاده از معیارهای شباهت مختلف

برای اندازه‌گیری شباهت بین نقاط مختلف مغز، از چهار معیار شباهت استفاده شد:

- **فاصله اقلیدسی:** برای محاسبه فاصله مستقیم بین نقاط.
- **معیار کسینوسی:** برای اندازه‌گیری شباهت جهت‌گیری بین بردارها.
- **معیار پیرسون:** برای بررسی همبستگی خطی بین نقاط.
- **معیار ویولت:** برای اندازه‌گیری شباهت نمایش ویولت داده‌ها.

2. مدل‌سازی ماتریس شباهت‌ها

ماتریس شباهت‌های حاصل، که یک ماتریس 351×351 بود، به روش‌های مختلف مدل‌سازی و تجزیه و تحلیل شد:

- **مدل ترانسفورمر:** ماتریس شباهت‌ها به صورت تصویر در نظر گرفته شد و به یک مدل ترانسفورمر وارد شد تا از قابلیت‌های توجه خودکار برای کشف الگوهای پیچیده استفاده شود.
- **شبکه عصبی گرافی (GNN):** ماتریس شباهت‌ها به عنوان ماتریس مجاورت برای GNN استفاده شد تا از قابلیت‌های یادگیری از داده‌های ساختاریافته بهره ببریم.
- **مدل‌های سنتی:** بخش بالایی ماتریس شباهت‌ها به مدل‌های سنتی مانند SVM وارد شد تا از قدرت این مدل‌ها در تمایز داده‌ها استفاده کنیم.

3. استفاده از الگوریتم‌های سنتی گراف

برای تحلیل دقیق‌تر الگوهای مغزی، از الگوریتم‌های سنتی گراف مانند مجموع وزن نودها و ماتریس لاپلاسین استفاده شد. این الگوریتم‌ها به ما کمک کردند تا ویژگی‌های مهم ساختاری گراف را استخراج و تحلیل کنیم.

4. استفاده از مدل‌های یادگیری ماشین

در کنار روش‌های فوق، از مدل‌های مختلف یادگیری ماشین برای تحلیل و طبقه‌بندی داده‌ها استفاده شد:

- **ماشین بردار پشتیبان (SVM):** به دلیل توانایی در مدیریت داده‌های با بعد بالا و انعطاف‌پذیری در استفاده از توابع هسته‌ای مختلف.
- **جنگل تصادفی (Random Forest):** به دلیل توانایی در ترکیب چندین درخت تصمیم‌گیری و بهبود

دقت و کنترل بیش‌برازش.

- پرسپترون چندلایه (Multilayer Perceptron): به دلیل قدرت بالا در مدل‌سازی و پیش‌بینی الگوهای پیچیده.
- آدا‌بوست (AdaBoost): به عنوان یک واریانت از جنگل تصادفی که توانایی تقویت مدل‌های ساده‌تر و افزایش دقت پیش‌بینی را دارد.

5. استفاده از مدل YOLO v8

مدل YOLO v8 به دلیل استفاده از مکانیزم توجه خودکار و توانایی در تشخیص الگوهای پیچیده و اطلاعات متنی در داده‌ها، برای بهبود عملکرد طبقه‌بندی انتخاب شد.

این ایده‌ها و روش‌های متنوع به ما امکان دادند تا به طور دقیق‌تر و کامل‌تری الگوهای همبستگی مغزی در افراد دارای ADHD را تحلیل کنیم و تفاوت‌های معنادار بین این افراد و سایرین را شناسایی نماییم. هر یک از این روش‌ها به نحوی به بهبود دقت و تفسیر نتایج کمک کرده و ابزارهای قدرتمندی برای پیش‌بینی و تحلیل داده‌های پیچیده ارائه دادند.

مرور کارهای مرتبط

در سال‌های اخیر، تحقیقات بسیاری در زمینه تشخیص اختلال نقص توجه و بیش‌فعالی (ADHD) با استفاده از داده‌های fMRI و روش‌های یادگیری ماشین انجام شده است. در یک مطالعه، مدل‌های یادگیری عمیق مانند شبکه‌های عصبی کانولوشن (CNN) برای تحلیل الگوهای حرکتی در کودکان مبتلا به ADHD استفاده شده است. دیگری از تکنیک‌های کاهش بعد و شبکه‌های عصبی عمیق برای نقشه‌برداری شبکه‌های مغزی بهره برده است. تحقیقات بر استفاده از شبکه‌های عصبی گرافی (GNN) برای مدل‌سازی روابط پیچیده در داده‌های مغزی تاکید داشته‌اند، که این روش‌ها بهبود دقت در طبقه‌بندی افراد مبتلا به ADHD را نشان داده‌اند. این نشان‌دهنده پتانسیل بالای روش‌های یادگیری ماشین و داده‌کاوی در تشخیص دقیق‌تر ADHD و درک بهتر از ساختارهای مغزی مرتبط با این اختلال هستند.

فاز آماده سازی داده ها

در این فاز به پیش پردازش داده‌ها، تشخیص داده های پرت حداقل و تبدیل و استاندارد سازی داده‌ها می پردازیم. کدهای این قسمت در فایل EDA.py قابل مشاهده است.

تشخیص داده های پرت

در این مرحله سعی روش isolation forest را برای بدست آوردن داده های پرت امتحان شد. بهبودی در دقت cross validation حاصل نشد. لذا در سایر تست ها داده ای به عنوان داده پرت حذف نشد.

تبدیل و استاندارد سازی داده‌ها

در فاز شناخت و فهم مسئله، بخش معرفی سوال و معیار ارزیابی و داده ها بیان شد، مجموعه داده‌ی خام متشکل از 1198 رکورد فیلتر شده از مصرف اکسیژن 351 قسمت از مغز افراد مختلف است و هدف ما استخراج ویژگی از شباهت روند مصرف اکسیژن در این قسمت‌هایی از مغز فرد است. بنابراین، نیاز است در

هر رکورد، تعداد $\binom{351}{2} = 61425$ شباهت محاسبه شود.

با توجه به حجم زیاد داده‌های خام، گسسته بودن داده‌ها در پوشه ها و فایل ها و فرآیند پیچیده تجمیع داده ها و هزینه پردازشی و زمان زیاد محاسبه همبستگی‌ها در فازهای آتی، تصمیم گرفته شد از داده‌های خام چند مجموعه دیتاست آماده شود. هر یک از این دیتاست ها برای یک معیار شباهت هستند و در آنها معیار شباهت برای تمامی نقاط مغزی تمامی رکورد ها محاسبه شده است. این کار در فایل

SimilarityMeasureComputation.ipynb انجام شده است.

چند سلول اول کتابخانه های مورد نیاز نصب و import شده اند و نوتبوک به گوگل درایو متصل شده و دیتاست استخراج شده است.

Preparation

```
[ ] #download packages
!pip install -q unrar jaxwt
```

```
from google.colab import drive
drive.mount('/content/drive')
```

در بلوک زیر آدرس فایل های رکوردها استخراج شده و همراه در شناسه بیمار در لیستی ذخیره می شوند.

```
# Define the main directory
main_dir = 'folders'

# Lists to store file paths and folder paths
file_list = []
folder_list = []

# Traverse through the main directory and all its subdirectories
for root, dirs, files in os.walk(main_dir):
    # Find files matching the pattern sfn*.1D in the current directory
    for file_name in glob.glob(os.path.join(root, 'sfn*.1D')):
        # Append the file path to the file_list
        file_list.append(file_name)
    # Append the folder path to the folder_list
    folder_list.append(root.split("/")[2])

# Print the lists to verify
print("Files:", file_list)
print("Folders:", folder_list)

# Verify that both lists have the same size
assert len(file_list) == len(folder_list), "The lists are not the same size"
```

```
Files: ['folders/Pittsburgh/0016001/sfnwmrda0016001_session_1_rest_1_cc400_TCs.1D', 'folders/Pittsburgh/0016001/sfnwmrda0016001_session_1_rest_1_cc400_TCs.1D', 'folders/Pittsburgh/0016001/sfnwmrda0016001_session_1_rest_1_cc400_TCs.1D', 'folders/Pittsburgh/0016001/sfnwmrda0016001_session_1_rest_1_cc400_TCs.1D']
Folders: ['0016001', '0016001', '0016010', '0016079', '0016032', '0016076', '0016039', '0016075', '0016058', '0016060', '0016006', '0016018', '0016051', '0016053', '0016085', '0016081']
```

در بلوک زیر فایل های حاوی تشخیص مربوط به هر مرکز ثبت داده خوانده و همه فایل ها در `diagnosis_df` تجميع می شوند تا در ادامه به عنوان ستون لیبل استفاده شوند.

```
dataframes = []
root = "/content/folders"
for dir in os.listdir(root):
    dirpath = os.path.join(root, dir)
    for filename in glob.glob1(dirpath, '*_phenotypic.csv'):
        filepath = os.path.join(dirpath, filename)
        df = pd.read_csv(filepath)

        try:
            df = df.loc[:, ["ScanDir ID", "DX"]]
        except:
            df.rename(columns={"ScanDirID": "ScanDir ID"}, inplace=True)
            try:
                df = df.loc[:, ["ScanDir ID", "DX"]]
            except:
                print("The file \'%s\' does not have ScanDir ID or DX columns"%filepath)

# Check for NaN values
if df.isnull().values.any():
    print("The file \'%s\' has nan values"%filepath)

dataframes.append(df)

diagnosis_df = pd.concat(dataframes, ignore_index=True)
```

در این بلوک چک شده، تشخیص برای تمامی رکورد ها موجود باشد.

```
ScanDirIDs = diagnosis_df["ScanDir ID"].tolist()
assert len(
    set(map(int, folder_list))-set(ScanDirIDs)
) == 0
```

در این بلوک چک شده، تمامی رکورد ها حاوی تمامی 351 نقطه مغزی باشند.

```
columns_Set = pd.read_csv(file_list[0], sep="\t").drop(["File", "Sub-brick"], axis=1).columns
columns_Set = set(columns_Set.tolist())
for file in file_list:
    df = pd.read_csv(file, sep="\t").drop(["File", "Sub-brick"], axis=1)
    assert columns_Set == set(df.columns.tolist())
```

با توجه به حجم زیاد محاسبات و وکتوری بودن محاسبات شباهت میان دو سری زمانی تصمیم گرفته شد از کتابخانه jax ماژول numpy استفاده شود. این ماژول به ما کمک کرد با نوشتن کد طبق api کتابخانه numpy محاسباتمان را روی gpu انجام دهیم. سرعت محاسبات را بهبود دهیم.

در تابع زیر تمامی رکورد ها خوانده شده و طبق تابع distance فاصله دو به نقاط مغزی محاسبه شده و در یک آرایه ذخیره می شوند.

```
def optimized_jax_code(file_list, name2save, distance, diagnosis_df):
    results_list = []

    for file_path in file_list:

        df = pd.read_csv(file_path, sep="\t").drop(["File", "Sub-brick"], axis=1)
        columns = df.columns

        # Convert the dataframe to a JAX array
        data = jnp.array(df.values)

        # Generate all combinations of i and j for upper triangle
        i_indices, j_indices = jnp.triu_indices(351, k=1)

        # Vectorize the calculation over all combinations
        calc_adjacency_batched = vmap(lambda i, j: distance(data[:, i], data[:, j]))

        values = calc_adjacency_batched(i_indices, j_indices)

        # Convert JAX array to numpy array for DataFrame insertion
        upper_triangle_values = jnp.array(values)

        # Convert to a numpy array and append the filename at the beginning
        results_list.append([file_path] + upper_triangle_values.tolist())
```

در ادامه این تابع آرایه محاسبه شده به همراه نام رکورد به عنوان یک سطر در دیتاست ثانویه قرار گرفته و تشخیص ADHD به عنوان یک ستون به دیتاست ثانویه اضافه می شود و در انتهای در یک فایل ذخیره می شود.

```
# Convert the list of arrays to a DataFrame
results_df = pd.DataFrame(results_list, columns=["File"] + [f"Value_{i}" for i in range(upper_triangle_values.shape[0])])

##### Add corresponding diagnosis to the dataframe

# Extract the ID from the 'File' column in diagnosis df
results_df["Extracted ID"] = results_df["File"].apply(lambda x: int(x.split('/')[2]))

# Merge results_df with labels on the extracted ID and 'ScanDir ID'
results_df = results_df.merge(diagnosis_df, how='left', left_on='Extracted ID', right_on='ScanDir ID')

# Drop Extracted ID column
results_df.drop("Extracted ID", axis=1, inplace=True)

# Save the DataFrame to a file if needed
results_df.to_csv("/content/drive/MyDrive/DM_project2/full_"+name2save+".csv", index=False)
```

در ادامه چهار معیار با استفاده از ماژول numpy در jax محاسبه شده اند. به تابع optimized_jax_code پاس داده می شوند و تا عملیات محاسبه و ذخیره سازی دیتاست های ثانویه انجام گیرد.

```
Pearsonr calculation

@jit
def pearsonr(x, y):
    x_diff = x - jnp.mean(x)
    y_diff = y - jnp.mean(y)
    numerator = jnp.sum(x_diff * y_diff)
    denominator = jnp.sqrt(jnp.sum(x_diff ** 2) * jnp.sum(y_diff ** 2))
    return numerator / denominator

[ ] optimized_jax_code(file_list, "pearsonr", pearsonr, diagnosis_df)

educian distance

@jit
def educian(x, y):
    return jnp.sqrt(
        jnp.sum((x - y)**2)
    )

[ ] optimized_jax_code(file_list, "educian", educian, diagnosis_df)
```

```
Cosine Similiarity

@jit
def cosine_similarity(x, y):
    x_normalized = x / jnp.linalg.norm(x)
    y_normalized = y / jnp.linalg.norm(y)
    return jnp.dot(x_normalized, y_normalized)

[ ] optimized_jax_code(file_list, "cosine_similarity", cosine_similarity, diagnosis_df)

Wavelet

[ ] def wavelet(x,y):
    wavelet = 'db1'
    coeffs_x = jwv.wavedec(x, wavelet)
    coeffs_y = jwv.wavedec(y, wavelet)

    # Flatten the coefficients for comparison
    flattened_coeffs_x = jnp.hstack(coeffs_x)
    flattened_coeffs_y = jnp.hstack(coeffs_y)

    # Compute the Euclidean distance between the wavelet coefficients
    return jnp.linalg.norm(flattened_coeffs_x - flattened_coeffs_y)

[ ] optimized_jax_code(file_list, "wavelet", wavelet, diagnosis_df)
```

لازم به ذکر است محاسبه تبدیل Wavelet سیگنال ها با استفاده از کتابخانه jaxwt صورت گرفت. در ادامه

پروژه از این دیتاست ها به جای دیتاست خام استفاده خواهد شد.

در ادامه، ما به غیر از این ماتریس های شباهت/همبستگی/اتصال مغزی داده ها را به فرم ماتریس مجاورت

گراف و یک تصویر از ماتریس همبستگی نیز تبدیل کردیم.

برای تبدیل دیتاست full_pearson_r.csv به یک دیتاست تصویر با قطعه کد زیر توسعه داده شد.

```
import numpy as np
import os
from concurrent.futures import ThreadPoolExecutor
from PIL import Image

def generate_adjacency_matrix(filename, values, label):
    """Generate an adjacency matrix from the upper triangle values."""
    # Retrieve the upper triangle values from the row and drop NaNs
    upper_triangle_values = values[~np.isnan(values)]

    # Calculate the size of the adjacency matrix
    n = int(np.sqrt(2 * len(upper_triangle_values) + 0.25) - 0.5)

    # Create a zero-filled matrix
    adjacency_matrix = np.zeros((n, n))

    # Fill the upper triangle of the adjacency matrix
    triu_indices = np.triu_indices(n)
    adjacency_matrix[triu_indices] = upper_triangle_values

    # Mirror the upper triangle to fill the lower triangle
    adjacency_matrix += adjacency_matrix.T

    # Determine the binary label
    binary_label = 1 if label == 1 else 0

    return filename, adjacency_matrix, binary_label

def save_as_grayscale_image(filename, matrix, label, output_dir):
    """Save the adjacency matrix as a grayscale image in the appropriate label directory."""
    label_dir = os.path.join(output_dir, str(label))
    os.makedirs(label_dir, exist_ok=True)

    # Normalize the matrix to the range [0, 255]
    matrix_normalized = ((matrix - np.min(matrix)) / (np.max(matrix) - np.min(matrix)) * 255).astype(np.uint8)

    # Convert the matrix to a grayscale image
    image = Image.fromarray(matrix_normalized, mode='L')

    # Construct the output file path
    base_filename = os.path.splitext(os.path.basename(filename))[0]
    output_path = os.path.join(label_dir, f"{base_filename}.png")

    # Save the image
    image.save(output_path)

def load_csv_data(csv_file):
    """Load filenames, values, and labels from the CSV file."""
    df = pd.read_csv(csv_file, engine='c')
    filenames = df['File'].tolist()
    labels = df['DX'].tolist()
    values = df.drop(['File', 'ScanDir ID', 'DX'], axis=1).values
    return filenames, values, labels

def process_and_save_image(filename, values, label, output_dir):
    """Process a single row to generate an adjacency matrix and save it as an image."""
    filename, adjacency_matrix, binary_label = generate_adjacency_matrix(filename, values, label)
    save_as_grayscale_image(filename, adjacency_matrix, binary_label, output_dir)

def save_grayscale_images(csv_file, output_dir):
    """Main function to process the CSV file and save all adjacency matrices as grayscale images."""
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    filenames, values, labels = load_csv_data(csv_file)

    with ThreadPoolExecutor() as executor:
        futures = [executor.submit(process_and_save_image, filenames[i], values[i], labels[i], output_dir) for i in range(len(filenames))]
        for future in futures:
            future.result()

# Example usage
csv_file = "/content/drive/MyDrive/DW_project2/full_pearsonr.csv"
output_dir = "output"
save_grayscale_images(csv_file, output_dir)
```

فاز تحلیل اکتشافی داده ها

با توجه به این مهم که دیتاست های ثانویه تولیدی خود در بردارنده اطلاعات دو یا چند ویژگی از دیتاست خام می باشند. لذا تمامی تحلیل های این بخش در حیطه بررسی روابط چند متغیر بین متغیرها قرار می گیرد.

بررسی و تحلیل روابط تک متغیره بین متغیرهای پیش بین و متغیر هدف

با توجه اینکه هدف این پروژه استخراج اتصالات مغزی تبیین کننده ADHD می باشد. لذا استفاده از تک ویژگی از دیتاست خام به عنوان ویژگی توجیهی ندارد. لذا تحلیل اکتشافی در این حوزه انجام نگرفت.

بررسی روابط چند متغیره بین متغیرها

در اولین سوال سعی می کنیم، به این پرسش پاسخ دهیم که آیا ارتباطات متمایز کننده‌ای میان قسمت‌های مغز وجود دارند که از وجود یا عدم وجود این ارتباط‌ها بتوان ADHD داشتن فرد را نتیجه گرفت؟

از دیتاست ثانویه pearson_r.csv استفاده کردیم. ستون‌های این دیتا فریم یک همبستگی میان دو نقطه مغزی هستند و سطرهای این دیتا فریم رکورد مغزی گرفته شده از یک فرد می باشد. پیش از گفتیم که ذهنیت اولیه ما این است که نقاط مغزی در افراد دارای ADHD وجود دارند که همبستگی بالایی با هم در رکورد افراد دارای ADHD داشته و همین نقاط در افرادی که ADHD ندارد. همبستگی پایینی دارند.

ابتدا میانگین هر ستون از داده‌های مرتبط با افراد دارای ADHD و افراد بدون ADHD محاسبه شد. این میانگین‌ها به عنوان نماینده‌ای از همبستگی بین دو نقطه مغزی خاص در افراد مختلف در نظر گرفته شدند. سپس، با استفاده از آزمون t مستقل، تفاوت‌های معنادار بین میانگین‌های همبستگی این دو گروه بررسی شد.

نتایج آزمون t نشان داد که برخی از ستون‌ها (همبستگی‌ها) در افراد دارای ADHD به طور معناداری بالاتر از افراد بدون ADHD هستند. این ستون‌ها به عنوان ستون‌های مهم شناسایی شدند و تفاوت میانگین آنها نیز ثبت گردید.

برای اجرای این تحلیل، مراحل زیر انجام شد:

1. محاسبه میانگین همبستگی برای هر ستون در دو گروه افراد دارای ADHD و افراد بدون ADHD.
2. مقایسه میانگین همبستگی بین دو گروه با استفاده از آزمون t مستقل.
3. شناسایی ستون‌هایی که تفاوت معنادار در میانگین همبستگی بین دو گروه دارند (با p-value کمتر از 0.05).

کد مورد استفاده برای تحلیل

```
# Identify highly correlated nodes
high_correlation_threshold = 0.7
high_correlation_columns = [col for col in value_columns if df_pearsonr[col].abs().mean() > high_correlation_threshold]

#divide records into their classes
df_1 = df_pearsonr[df_pearsonr['DX Binary'] == 1]
df_0 = df_pearsonr[df_pearsonr['DX Binary'] == 0]

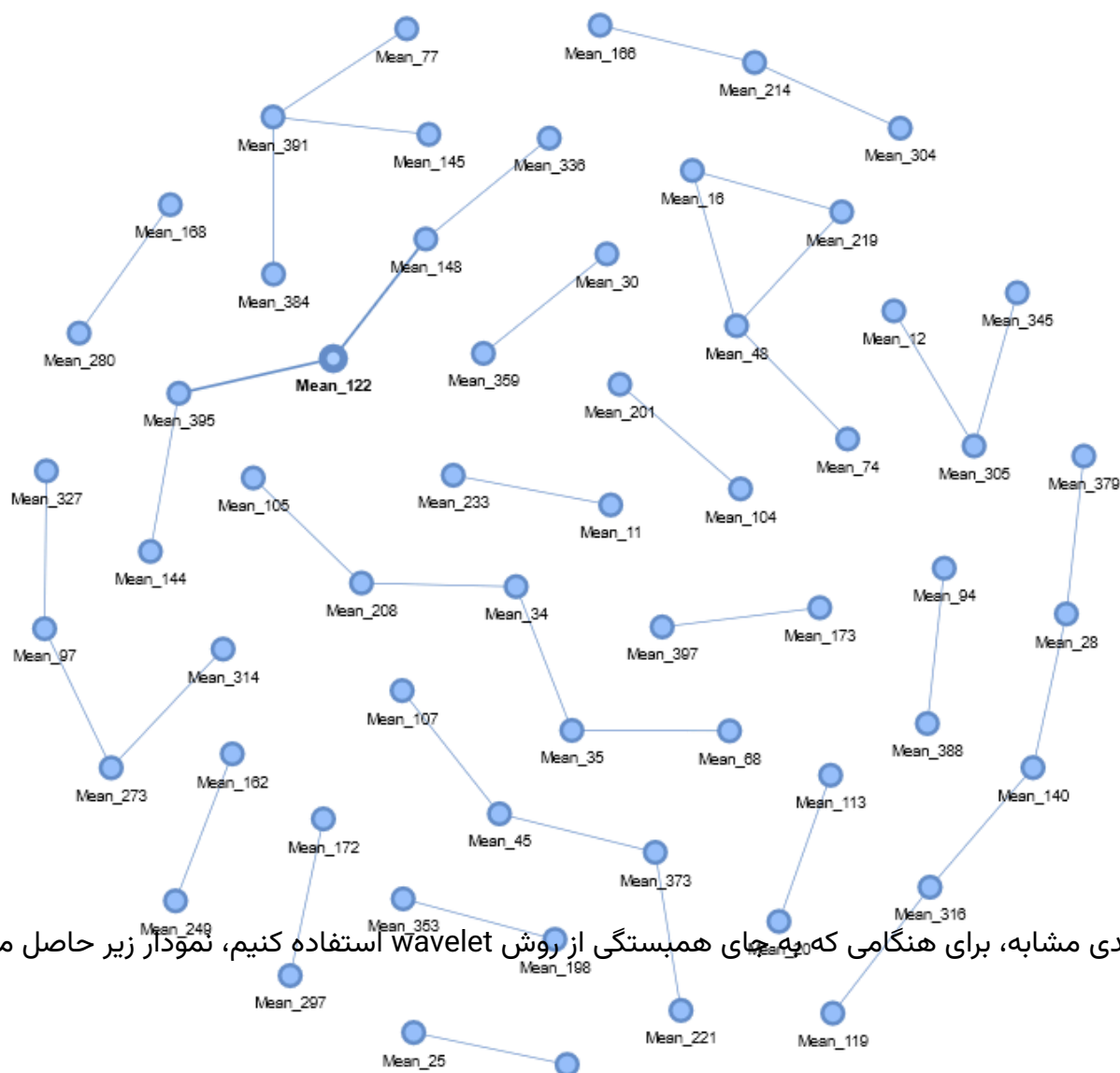
#Compute the mean of each Value_i column for each group.
means_1 = df_1[high_correlation_columns].mean()
means_0 = df_0[high_correlation_columns].mean()
significant_columns=[]

for col in high_correlation_columns:
    if means_1[col] > means_0[col]:
        t_stat, p_value = ttest_ind(df_1[col].dropna(), df_0[col].dropna())
        if p_value < 0.05:
            significant_columns.append(col)

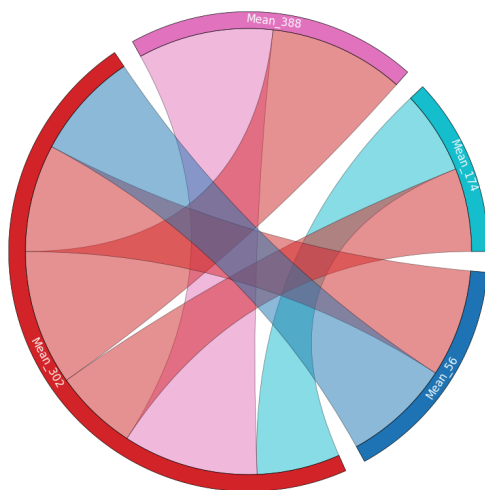
print("Significant columns with higher mean for DX Binary = 1:")
print(significant_columns)
```

Significant columns with higher mean for DX Binary = 1:
['Value_2634', 'Value_3038', 'Value_4163', 'Value_4321', 'Value_5565', 'Value_6813', 'Value_7900', 'Value_8104', 'Value_8741', 'Value_9744', 'Value_9904', 'Value_...

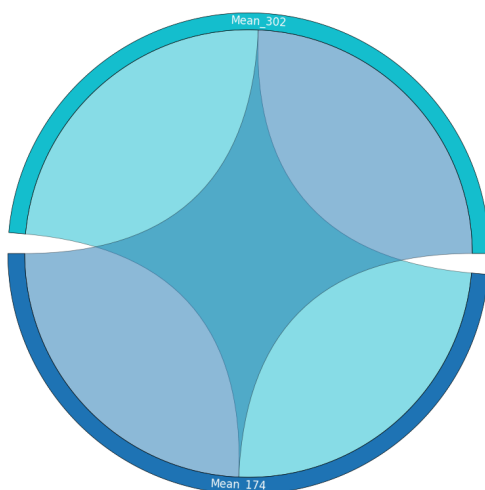
از اتصالات مغزی تبدیل کردیم:



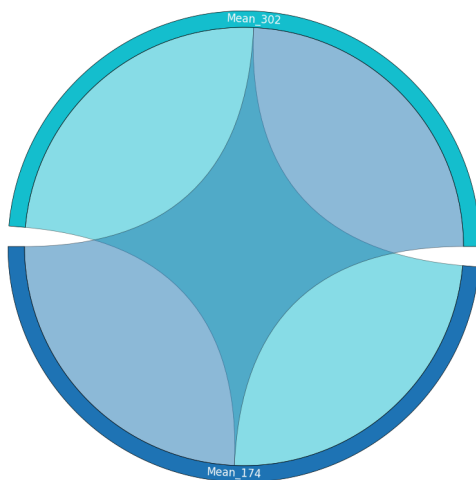
در روندی مشابه، برای هنگامی که به جای همبستگی از روش Wavelet استفاده کنیم، نمودار زیر حاصل می شود:



اجرای این روند، هنگامی که از فاصله اقلدیدی استفاده کنیم:



در پایان خروجی الگوریتم برای فاصله کسینوسی به شکل زیر است:



ادامه استفاده می شوند.

در ادامه چند نمودار ارائه می ده

ماتریس میانگین همبستگی در افراد دارای ADHD

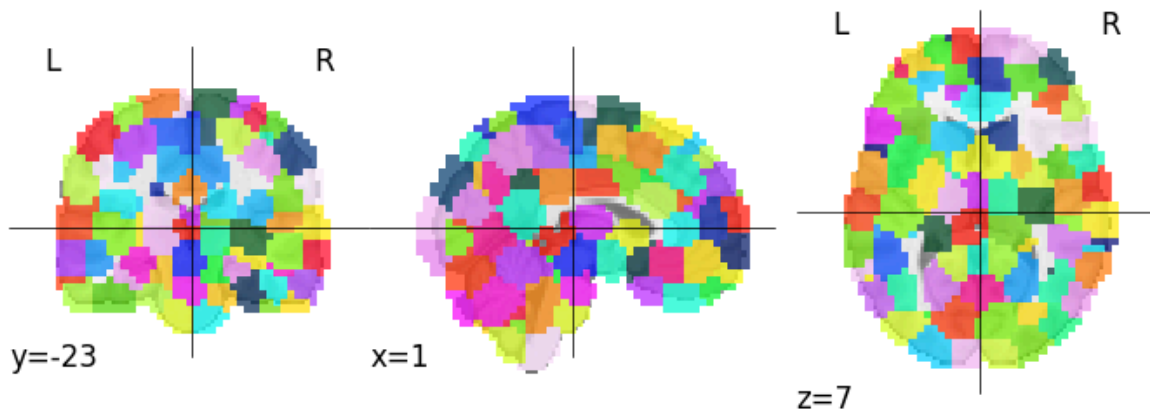


ماتریس میانگین همبستگی در افراد فاقد ADHD

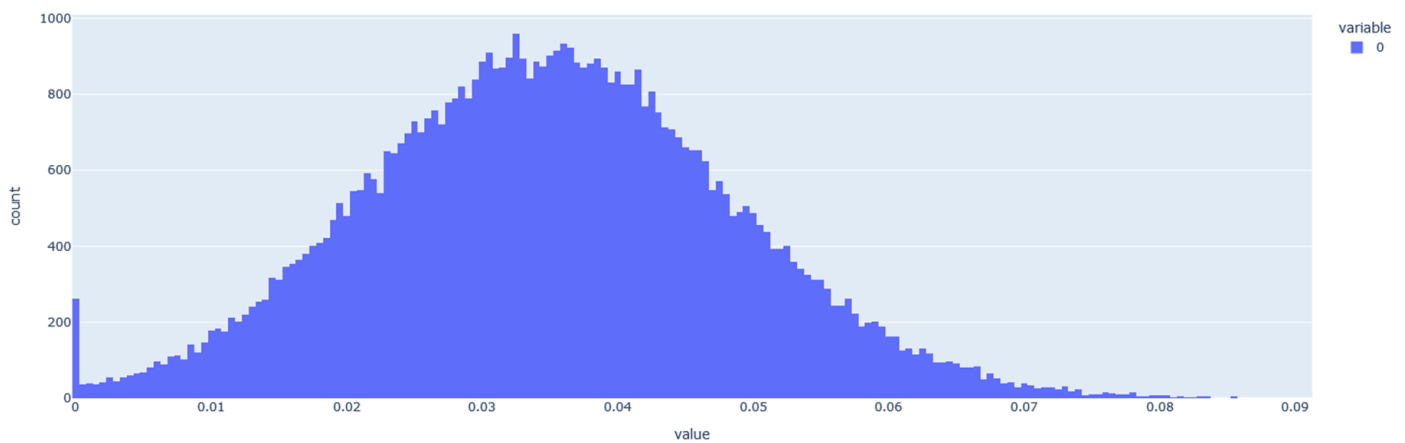


شمایی از نقاط مغزی مورد بحث در این پروژه که با استفاده از کتاب خانه nilearn ترسیم شده است.

Parcellation Visualization



همچنین شکل زیر شمایی از Mutual Information میان هر همبستگی و متغیر پیش‌بین را ارائه می‌دهد.

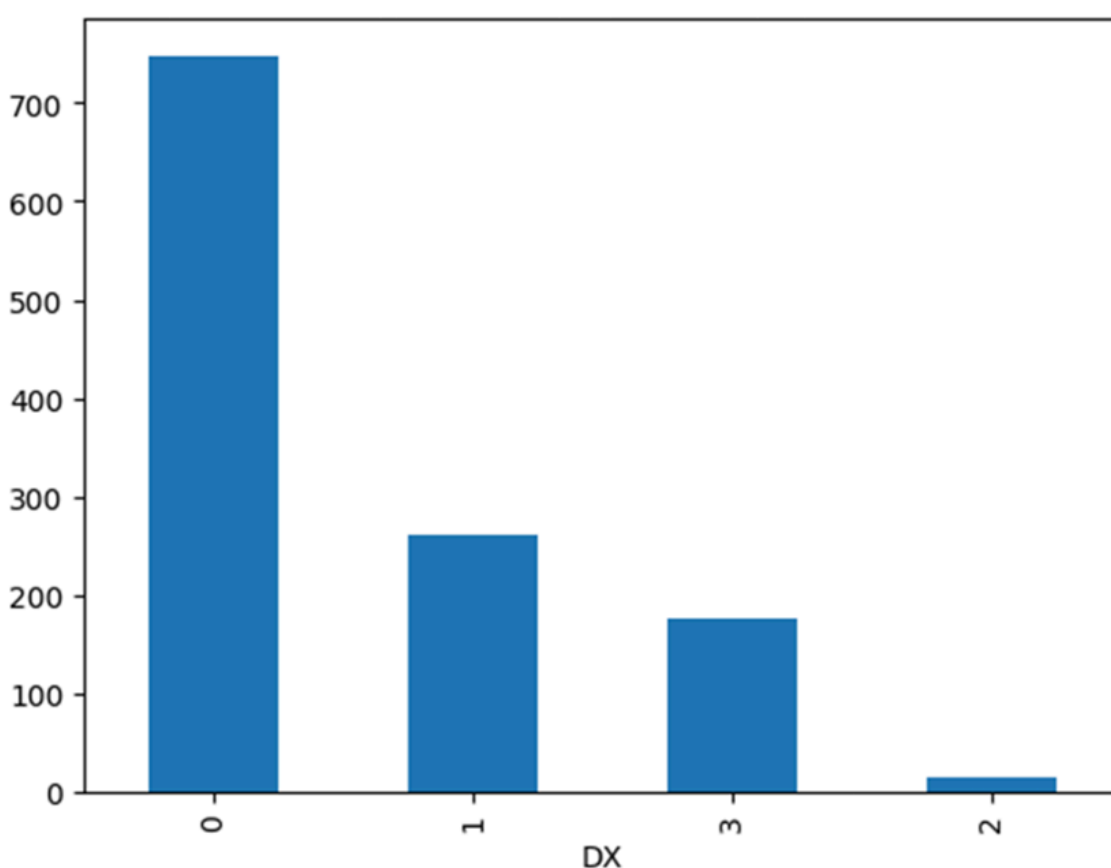


فاز پیش مدل (Setup)

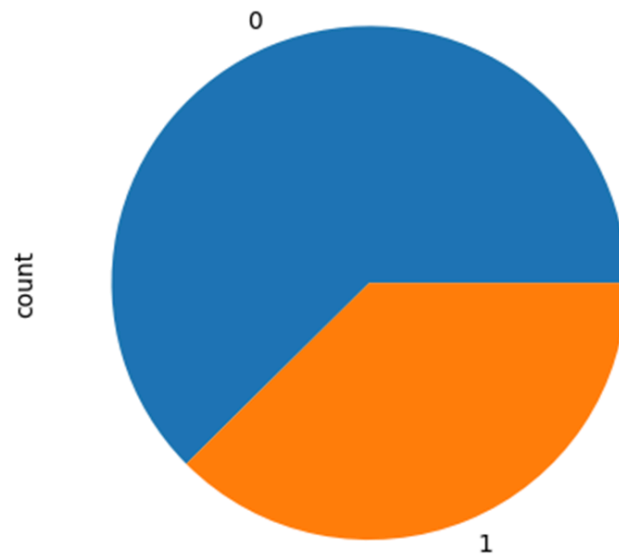
توضیحات این بخش در فایل Setup.py آماده شده اما پیاده سازی های مربوط به این بخش در ابتدا هر فایل مدل سازی صورت گرفته است.

بالانس داده ها

مجموعه داده مورد استفاده در این پروژه در حالت چهار کلاسه، دارای توزیع زیر است:



همانگونه که از این توزیع داده ها مشخص است، در این حالت داده ها بالانس نمی باشند. حال توزیع دو کلاسه داده ها را رسم می کنیم که در آن از دسته بندی انواع ADHD صرف نظر کردیم و داده ها را در دو دسته دارا و فاقد ADHD تعریف می کنیم.



با توجه به عدم امکان استفاده از روش های معمول برای افزایش دیتا برای متعادل سازی دسته ها در حالت چهار کلاسه، کارمان را روی حالت دو کلاسه تعریف می کنیم که داده های نسبتا متوازی دارد.

چارچوب تقسیم داده ها و به دسته آموزش و تست

0.1 از داده ها را برای تست جدا کرده و مابقی برای آموزش استفاده می شوند.

روش انجام Cross-Validation

با توجه به تعریف دو کلاسه داده های ما و اینکه توزیع داده ها تقریبا بالانس است پس نیازی به استفاده از Stratified k-fold cross-validation نیست و از k-fold cross-validation استفاده می کنیم.

فاز مدل سازی

انتخاب و پیاده سازی الگوریتم های لازم

در این بخش از پروژه، برای تحلیل و استخراج الگوهای همبستگی مغزی در افراد دارای ADHD، سه الگوریتم مهم و پرکاربرد در حوزه یادگیری ماشین را انتخاب و پیاده سازی کردیم. این الگوریتم ها شامل ماشین بردار پشتیبان (SVM)، جنگل تصادفی (Random Forest) و پرسپترون چندلایه (Multilayer Perceptron) می باشند. همچنین، در برخی از تست ها از الگوریتم AdaBoost که یک واریانت از جنگل تصادفی است نیز استفاده کردیم. در ادامه به توضیح هر یک از این الگوریتم ها و دلایل انتخاب آنها می پردازیم.

به دلیل تعداد بالای فیچر های ورودی که همان شباهت ها / همبستگیها و... می باشند. ورودی دادن مستقیم آن ها به مدل های یادگیری ماشین ممکن نیست. پس چالش اصلی ما کاهش تعداد این فیچر های ورودی می باشد. در هر دسته مدل، یک دسته روش تست شده است.

در روش های مبتنی بر گراف، از threshold تعریف شده و فیچر هایی که شباهت بیشتر آن مقدار داشته اند، به عنوان وجود یال بین دو نقطه ای که شباهت برایش محاسبه شده، در نظر گرفته شده.

در روش yolo اصولا مکانیسم خودتوجهی عملیات شناسایی فیچرهای مهم را انجام می دهد.

در این پروژه، از چندین روش برای انتخاب ویژگی ها در مدل های یادگیری ماشین استفاده کردیم:

1. آستانه واریانس (Variance Threshold): این روش ویژگی هایی را که واریانس آنها کمتر از یک مقدار معین است حذف می کند.
2. انتخاب ویژگی ترتیبی (Sequential Feature Selector): این روش به طور ترتیبی ویژگی ها را انتخاب یا حذف می کند تا مدل بهینه ایجاد شود.
3. KBest با اطلاعات متقابل (KBest-Mutual Information): این روش ویژگی هایی را که بیشترین اطلاعات متقابل با هدف دارند، انتخاب می کند.
4. ماکسیم شباهت (Max Similarity): در این روش، رکوردهایی که شباهت آنها با سایر رکوردها بیشتر از 0.9 است، نگه داشته می شوند.
5. مینیم شباهت (Min Similarity): در این روش، رکوردهایی که شباهت آنها با سایر رکوردها کمتر از 0.3 است، نگه داشته می شوند.

ماشین بردار پشتیبان (Support Vector Machine)

مدل های SVM در مدیریت داده های با بعد بالا عملکرد بسیار خوبی دارند و با استفاده از توابع هسته ای

مختلف، امکان مدیریت روابط خطی و غیرخطی را فراهم می‌کنند. این ویژگی‌ها، SVM را به یک ابزار قدرتمند و انعطاف‌پذیر تبدیل کرده و آن را در برابر بیش‌برازش مقاوم می‌سازد. دلیل انتخاب این الگوریتم در این پروژه، توانایی آن در تجزیه و تحلیل داده‌های پیچیده و چندبعدی همبستگی مغزی و تمایز مؤثر بین گروه‌های مختلف می‌باشد.

نمونه کدی از پیاده سازی الگوریتم SVM

```
clf = make_pipeline(VarianceThreshold(threshold=(0.103)), StandardScaler(), SVC(gamma='auto'))
k_folds = KFold(n_splits = 10)

for kernel in ['linear', 'poly', 'rbf', 'sigmoid']:
    originalclass = []
    predictedclass = []
    clf.set_params(svc__kernel=kernel)
    scores = cross_val_score(clf, X_train, y_train, cv = k_folds, scoring='accuracy')
    print(f"kernel {kernel}")
    print(classification_report(originalclass, predictedclass))
```

در این کد، ابتدا با Variance Threshold تعداد ویژگی‌های ورودی کاهش داده شده و سپس استاندارد سازی شده است. در ادامه، الگوریتم SVM با چهار کرنل خطی، چند جمله ای و rbf و sigmoid آموزش داده شده است و سپس cross validation با 10 فولد و معیار accuracy انجام گرفته و در انتها از classification report برای محاسبه f1-score و ... استفاده شده است.

جنگل تصادفی (Random Forest)

جنگل تصادفی یک روش یادگیری تجمعی است که با ساخت چندین درخت تصمیم‌گیری بر روی زیرمجموعه‌های مختلف داده‌ها و ترکیب پیش‌بینی‌های هر درخت، دقت و کنترل بیش‌برازش را بهبود می‌بخشد. این الگوریتم به دلیل توانایی بالای آن در مدیریت داده‌های پیچیده و جلوگیری از بیش‌برازش، برای این پروژه انتخاب شده است. همچنین، قابلیت تفسیر ساده‌تر نتایج و شناسایی ویژگی‌های مهم از دیگر مزایای استفاده از جنگل تصادفی می‌باشد.

random forest

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import pandas as pd

# Step 1: Retrieve adjacency matrices and labels
csv_file = '/content/pearsonr.csv'
df = pd.read_csv(csv_file)
values = df.drop(['File', 'ScanDir ID', 'DX'], axis=1).values
print(values)

# Step 2: Convert labels to binary: 0 as class 0, and 1, 2, 3 as class 1
labels = df['DX'].tolist()
binary_labels = [0 if label == 0 else 1 for label in labels]

# Step 3: Train/Test Split
X_train, X_test, y_train, y_test = train_test_split(values, binary_labels, test_size=0.2, random_state=42)

# Step 4: Random Forest Training
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)

# Step 5: Evaluation
y_pred = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

پرسپترون چندلایه (Multilayer Perceptron)

شبکه‌های عصبی مجموعه‌ای از الگوریتم‌ها هستند که بر اساس ساختار مغز انسان مدل‌سازی شده‌اند و برای شناسایی الگوها به کار می‌روند. این الگوریتم‌ها توانایی بالایی در شناسایی روابط پیچیده غیرخطی در داده‌ها دارند. دلیل انتخاب پرسپترون چندلایه در این پروژه، قدرت بالای آن در مدل‌سازی و پیش‌بینی الگوهای همبستگی مغزی و تحلیل دقیق‌تر داده‌ها است. این الگوریتم به ما کمک می‌کند تا روابط پنهان و پیچیده بین نقاط مغزی را به خوبی شناسایی کنیم. این مدل در متن، مدل‌های پیشرفته تست شده (لایه‌های gcن را صفر کردیم).

آدا بوست (AdaBoost)

در برخی از تست‌ها، از الگوریتم AdaBoost که یک واریانت از جنگل تصادفی است نیز استفاده کردیم. AdaBoost با ترکیب مدل‌های ساده‌تر و تقویت آنها، بهبود دقت و کنترل بیشتری بر روی داده‌ها فراهم می‌کند. این الگوریتم با توجه به توانایی‌اش در افزایش دقت پیش‌بینی و مدیریت بهتر خطاها، انتخاب مناسبی برای تست‌های تکمیلی در این پروژه بوده است.

نمونه ای از قطعه کد مربوط به مدل Adaboost

```
[ ] clf = make_pipeline(StandardScaler(), PCA(n_components=30), AdaBoostClassifier(n_estimators=1000))
    k_folds = KFold(n_splits = 10)
    originalclass = []
    predictedclass = []
    scores = cross_val_score(clf, X_train, y_train, cv = k_folds, scoring=make_scorer(acc_score))
    print(classification_report(originalclass, predictedclass))
```

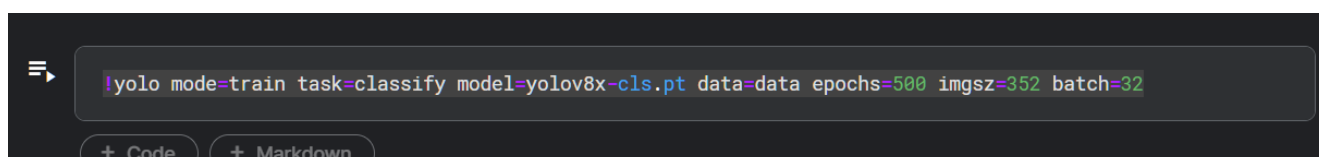
مشابه مدل های پیشین ابتدا داده ها استاندارد سازی شده و سپس تعداد فیچر ها با یک الگوریتم مثل pca کاهش یافته و سپس مدل adaboost با تعداد زیر مدل معین روی 10 فولد آموزش داده شده و classification report برایش محاسبه شده است.

بکارگیری توسعه ها و روش های پیشرفته به شرط اثبات در قالب یک مقاله

مدل YOLOv8

ما مدل YOLOv8 را برای مدل سازی داده ها انتخاب کردیم زیرا به عنوان یک مدل مبتنی بر ترانسفورمر از مکانیزم های توجه خودکار بهره می برد. این مکانیزم ها به مدل امکان می دهند تا به همه قسمت های تصویر به شکل یکسان توجه نشد و همانطور که می دانیم از ماتریس همبستگی برخی همبستگی ها مهم تر از سایرین بوده که این مکانیسم می تواند، به صورت خودکار این بخش ها را شناسایی کرده و عملا برای ما feature selection انجام دهد. ویژگی دیگر این مدل این است که این مدل بسیار پیچیده قبلا روی یک دیتاست تصویر جنرال آموزش دیده و وزن های تقریبا بهینه ای برای دیتاست جنرال اولیه بدست آمده و ما آموزشمان با دیتاست خودمان را از این نقطه تقریبا بهینه شروع می کنیم و به نقطه بهینه جهانی می رسیم. این ویژگی ترانسفور به ما کمک می کند که از مدل پیچیده با تعداد وزن بالا بدون بیش برازش روی دیتاست خودمان کمک بگیریم. این پیچیدگی زیاد مدل امکان استخراج الگوهای بسیار پیچیده مستتر در دیتاست را فراهم می کند.

ما از ورژن CLI مدل استفاده کردیم و مدل را پارامتر های زیر اجرای کردیم.



```
!yolo mode=train task=classify model=yolov8x-cls.pt data=data epochs=500 imgsz=352 batch=32
```

The screenshot shows a dark-themed terminal window. At the top left, there is a hamburger menu icon. Below it, a command is entered in a light blue box: `!yolo mode=train task=classify model=yolov8x-cls.pt data=data epochs=500 imgsz=352 batch=32`. At the bottom of the terminal, there are two buttons: `+ Code` and `+ Markdown`.

جزئیات بیشتر در فایل ImageClassification.py قابل مشاهده است.

فاز ارزیابی

معرفی مجموعه معیارهای ارزیابی و محاسبه آن ها و تفسیر نتایج

در این پروژه، از دو معیار مهم برای ارزیابی عملکرد مدل های مورد استفاده بهره برده ایم:

- دقت (Accuracy): این معیار نشان می دهد که مدل ما چه تعداد از پیش بینی های صحیح را از مجموع پیش بینی ها انجام داده است.
- F1 Score: این معیار ترکیبی از دقت (Precision) و بازخوانی (Recall) است که توازن بین این دو را برقرار می کند و ما از امتیاز F1 به این دلیل استفاده کردیم که در برخی موارد ممکن است دقت بالا به تنهایی کافی نباشد. در تشخیص اختلالات مانند ADHD، مهم است که نه تنها پیش بینی های صحیح زیادی داشته باشیم، بلکه مدل بتواند موارد مثبت واقعی (True Positives) را به خوبی شناسایی کند و از شناسایی نادرست موارد منفی (False Positives) جلوگیری کند. F1 score این توازن را فراهم می کند و تضمین می کند که مدل ما نه تنها دقیق بلکه جامع نیز باشد.

		ACTUAL	
		Positive	Negative
PREDICTED	Positive	True Positive (TP) 5	False Positive (FP) 10
	Negative	False Negative (FN) 15	True Negative (TN) 70

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = 0.75$$
$$\text{Recall} = \frac{TP}{TP + FN} = 0.25$$
$$\text{Precision} = \frac{TP}{TP + FP} = 0.33$$
$$\text{F1 Score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} = 0.28$$

در عمل اعداد گزارش شده در ادامه، از classification_report از کتابخانه sklearn استفاده شده است.

نتایج بدست آمده داده روی داده تست

برآورد خطا

نتایج 10-fold cross validation روی مدل های ML به شرح زیر است:

Model	#Input Features	Accuracy	Weighted Avg. F1-Score
VarianceThreshold-0.103+StandardScaler+SVC-Linear	29	0.62	0.53
VarianceThreshold-0.103+StandardScaler+SVC-Poly	29	0.62	0.53
VarianceThreshold-0.103+StandardScaler+SVC-RBF	29	0.63	0.58
VarianceThreshold-0.103+StandardScaler+SVC-Sigmoid	29	0.56	0.56
VarianceThreshold-0.103+StandardScaler+AdaBoost-1000	29	0.59	0.58
VarianceThreshold-0.103+StandardScaler+GaussianNB	29	0.63	0.62
SequentialFeatureSelector	-	-	-
KBest-MutualInformation+StandardScaler+SVC-Linear	30	0.61	0.56
KBest-MutualInformation+StandardScaler+SVC-Poly	30	0.62	0.63
KBest-MutualInformation+StandardScaler+SVC-RBF	30	0.62	0.58
KBest-MutualInformation+StandardScaler+SVC-Sigmoid	30	0.63	0.59
Sim>0.9+StandardScaler+AdaBoost-50	30	0.64	0.63
Sim>0.9+StandardScaler+AdaBoost-1000	30	0.61	0.61
Sim<0.3+StandardScaler+AdaBoost-50	30	0.59	0.58
Sim<0.3+StandardScaler+AdaBoost-1000	30	0.56	0.55

همانطور که از نتایج مشاهده می شود، بهترین نتیجه زمانی حاصل شد که ویژگی هایی با شباهت ماکسیمم بیشتر از 0.9 انتخاب شدند. این ویژگی ها پس از استانداردسازی، با استفاده از مدل AdaBoost و 50 درخت آموزش داده شدند. در این حالت، دقت مدل 64٪ و امتیاز F1 برابر 0.63 به دست آمد.

چون دقت این روش ها در مقایسه با روش های پیشرفته کم است، از گزارش نتایج این مدل ها روی داده تست صرف نظر می کنیم. اما اجرای این مدل ها روی داده تست دقتی کمتر 70٪ را نشان می داد.

نتایج مدل های پیشرفته

با توجه به اینکه آموزش مدل های پیشرفته چندین ساعت به طول می انجامید، اجرای اعتبارسنجی متقابل (Cross Validation) روی این مدل ها امکان پذیر نبود. به همین دلیل، تنها نتایج به دست آمده از داده های تست گزارش شده اند. این رویکرد به ما اجازه می دهد تا عملکرد مدل ها را با داده های واقعی ارزیابی کنیم، هرچند که نتایج نهایی ممکن است دقیق ترین ارزیابی ممکن نباشد. با این حال، نتایج گزارش شده، نمای کلی از قابلیت های مدل در تشخیص اختلال ADHD را ارائه می دهند.

Data	Model Name	Accuracy
Pearson r-test	GNN	77.08
Pearson r-test	GCNN	57.03
Pearsonr-test	Graph Traditional SVM	69.75
Pearsonr-test	Mlp	65.55
Pearsonr-test	Random Forest	58.33
Pearsonr-test	Yolov	80.7

همان طور که از جدول فوق قابل مشاهده است، GCNN , YOLOV بهترین نتایج را در این بخش می دهند.

گزارش آزمایشات برگشت به فاز های قبلی برای بهبود ارزیابی

همان طور که در نوت بوک ها قابل مشاهده است، سعی شد مدل ها روی دیتاست های مختلف اجرای شود اما چون نتایج بهبود چشمگیری نداشتند کمتر 3%. در گزارش به عنوان نتیجه مستقل آن ها را گزارش نمی کنیم. سعی شد چند مدل روی حالت 4 کلاسه آموزش داده شود اما classification report نشان می داد مدل همه داده ها را با به عنوان کلاس بیشتر (نداشتن ADHD) پیشبینی می کرد.

سعی شد، یک الگوریتم clustering روی داده های آموزش داده شود. که یادگیری روی هر کلاستر انجام شود اما sihouette score نشان می داد که خود الگوریتم کلاستریگ نتوانسته به طور مطلوب کلاستریگ را انجام دهد.

مشخص کردن بهترین مدل از بین مدل های اجرا شده همراه با پارامترهای تعیین شده

بهترین مدل، از میان مدل های اجرا شده مدل YOLO با 500 اپیاک آموزش و batch=32 و imgz=352 است. سایر هایپرپارامتر ها، به صورت دیفالت تعیین شده اند و در [لینک](#) قابل مشاهده هستند.

تحلیل نقاط قوت و ضعف کار انجام شده و پیشنهادات برای بهبود آینده

نقطه قوت کار انجام شده استفاده از الگوریتم های متنوع و دیدگاه های مختلف نسبت به اتصال نقاط مغزی بود. همچنین تست روش های مختلف استخراج ویژگی از سیگنال های مغزی نیز از دیگر مزایای این پروژه بود. با توجه به محدودیت های عنوان درس، امکان استفاده از مدل های پیشرفته تر به صورت گسترده فراهم نبود. با این حال، ممکن است استفاده از مدل های ترانسفورمر پیشرفته تر نتایج بهتری به همراه داشته باشد. پیشنهاد می شود در تحقیقات آینده، بهره گیری از این مدل ها برای بهبود دقت و عملکرد مدل های تشخیصی مورد بررسی قرار گیرد.

برای کار های آینده می توان از دیدگاه های دیگری به دیتاست خام نگاه کرد:

در مسابقه ای که دیتاست آن استخراج شده بود، بهترین دقت توسط تیمی به دست آمده بود که از داده های بیولوژیکی مانند سن و فشار خون استفاده کرده و کاری با داده های fMRI نداشتند. این نشان می دهد که تمرکز بر ویژگی های بیولوژیکی می تواند در بهبود دقت تشخیص مؤثر باشد. پیشنهاد می شود در آینده، علاوه بر داده های fMRI، از ویژگی های بیولوژیکی نیز استفاده شود تا مدل های تشخیصی بهتری توسعه یابند.

محاسبه شباهت‌ها و همبستگی‌های سیگنال‌های نقاط مغزی یک فرآیند پیچیده است که منجر به افزایش تعداد ویژگی‌های ورودی می‌شد. یکی از راه‌های بهبود این مسئله، استخراج ویژگی‌های مستقل از خود سیگنال هر نقطه مغزی، بدون در نظر گرفتن ارتباطات بین نقاط، است. این روش ممکن است در تشخیص ADHD مؤثرتر باشد. از سوی دیگر، می‌توان محاسبه همبستگی را به مدل‌های هوش مصنوعی سپرد. به عنوان مثال، با استفاده از مدل‌های LSTM و RNN، داده‌های خام به صورت یک دنباله به مدل ورودی داده شوند تا الگوهای زمانی و ارتباطات پیچیده بهتر شناسایی شوند.