دانشگاه خوارزمی

دانشکده علوم ریاضی و کامپیوتر

گروه علوم کامپیوتر

گزارش نهایی پروژه کارشناسی

عنوان پروژه:
تشخیص هوشمند سقوط افراد سالمند

نگارش:

سید محمد فتاحیان

استاد راهنما:

دکتر محمد سلطانیان

نیمسال اخذ پروژه (نیمسال دوم سال تحصیلی 00-1399)

ماه و سال دفاع از پروژه (تیر ماه 1400)


بسم الله الرحمن الرحیم

تقدیم به پدر و مادر عزیزم

این پروژه را به پدر و مادر عزیزم که در تمام این مدت همراه من بودند تقدیم می‌کنم و امیدوارم در مراحل پیشرو نیز از همراهی آنان برخوردار باشم.

تشکر و قدردانی

از تمام کسانی که من را در انجام این پروژه همراهی کردن به ویژه جناب دکتر سلطانیان کمال سپاس گزاری را دارم.

چکیده

با توجه به آمار سازمان‌های جهانی جمعیت جهان به شدت رو به پیر شدن است و جمعیت افراد مسن تقریبا در تمام کشورهای جهان رو به افزایش است. سقوط یا زمین خوردن یکی از دلایل اصلی آسیب، معلولیت و مرگ در افراد مسن است. سقوط و صدمات حاصل از این اتفاق با افزایش استفاده از خدمات مراقبت‌های بهداشتی و هزینه‌های بالای مراقبت‌های بهداشتی برای جامعه همراه است.

هدف اصلی این پروژه طراحی، پیاده سازی و درنهایت مقایسه عملکرد، دقت و میزان یادگیری مدل‌های یادگیری ماشین برای تشخیص سقوط است. در این پروژه برای پیاده سازی مدل‌ها از داده‌های پایگاه داده SisFall که شامل پارامترهای 3سنسور(2سنسور شتاب سنج و 1سنسور ژیروسکوپ) است، استفاده شده.

در این پروژه ما با یک مسئله‌ی کلاس بندی با دو کلاس سقوط و عدم سقوط روبرو هستیم. در این پروژه روش‌های یادگیری ماشین بردار پشتیبان[1]، k نزدیکترین همسایگی، Logistic Regression، و شبکه‌های عصبی عمیق پیاده سازی و با معیارهای accuracy، precision، recall و نمره F1 مورد مقایسه قرار گرفته است.

موضوعی که در این پروژه بسیار پراهمیت است استفاده از تکنیک‌هایی برای بالا بردن تعادل میان داده‌های مجموعه داده SisFall بوده که تاثیر زیادی بر معیارها و شاخص‌های آماری عملکرد روش‌های یادگیری ماشین به کار گرفته شده در این پروژه داشته است.

در انتها نتیجه نهایی به دست آمده از این پروژه با مقایسه مدل‌های پیاده سازی شده این است که استفاده از روش شبکه‌های عصبی عمیق مدل دقیق‌تر و مطمئن‌تری را برای تشخیص سقوط به ما ارائه می‌دهد.

کلیدواژه: تشخیص سقوط، شبکه عصبی عمیق، دقت طبقه بندی، معیار نمره F1، روش بردار پشتیبان، روش k نزدیکترین همسایگی، مجموعه داده SisFall.

[1] SVM

فهرست نوشتار

فهرست شکل‌ها

فهرست جدول‌ها

فصل 1:

مقدمه

1-1- انگیزه پژوهش

مشکل سقوط در افراد سالمند (به ویژه آن‌هایی که به تنهایی زندگی می‌کنند) با افزایش جمعیت این افراد در بیشتر جوامع، روز به روز پراهمیت‌تر و به یک مشکل جهانی بخصوص در کشورهایی مثل ژاپن که جمعیت سالمندان رو به افزایش است تبدیل شده. از عواقب ناشی از سقوط افراد سالمند می‌توان به پیامدهای جسمی، مالی، پزشکی و روانی اشاره کرد که با افزایش این آسیب‌ها به ویژه در بعد پزشکی و مالی می‌تواند بر جامعه، دولت و اقتصاد یک کشور بسیار تاثیرگذار باشد. به طور مثال متوسط نرخ بستری مربوط به سقوط در بیمارستان در میان افراد مسن در انگلستان 169 در هر 10 هزار نفر و در بریتیش کلمبیا ، کانادا ، 155 در هر 10 هزار نفر و در در استرالیای غربی این میزان به 297 در هر 10 هزار نفر افزایش می‌یابد. به طور متوسط مدت بستری در بیمارستان برای یک فرد مسن که سقوط می کند 4-15 روز است. این میانگین در صورت آسیب رسان بودن سقوط(به عنوان مثال ، شکستگی) به 20 روز افزایش می یابد. در میان بزرگسالان 65 سال به بالا در آمریکا و کانادا ، آسیب های ناشی از سقوط 6-7 درصد از کل بستری شدن در بیمارستان و بیش از 50 درصد از کل بستری شدن در بیمارستان به دلیل آسیب های تصادفی است. در سال 2000 ، کل هزینه های مستقیم صدمات ناشی از سقوط در میان سالمندان آمریکایی بیشتر از 19 میلیارد دلار بوده است.

1-2- مروری بر پیشینه و کارهای مشابه

راه حل های تشخیص سقوط را می توان در سه دسته اصلی تقسیم بندی کرد: دستگاه‌های پوشیدنی، دستگاه های محیطی و دستگاه های مبتنی بر دوربین. رویکرد اول مستلزم آن است که افراد سالخورده نوعی از دستگاه ها را که شامل سنسورهایی، مانند شتاب سنج ها و یا ژیروسکوپ ها که برای تشخیص حرکت بدن هستند، بپوشند. با کوچک شدن و کاهش هزینه‌ی شتاب سنج‌ها و در دسترس بودن فن‌آوری‌های ارتباط بی‌سیم، سیستم‌های قابل پوشیدن و مقرون به صرفه‌ای را فراهم کرده که سالمندان می‌توانند در حالی که فعالیت های روزانه خود را انجام می دهند، بپوشند. به همین دلایل، در چند سال گذشته، استفاده از دستگاه های قابل حمل در نظارت به سلامت بیماران به طور قابل توجهی افزایش یافته است. با این حال، این دستگاه ها دارای برخی نقاط ضعف هستند از جمله امکان فراموشی در پوشیدن دستگاه، دردسترس نبودن دستگاه به دلیل اتمام شارژ باتری، پوشیدن دستگاه در موقعیت اشتباه بدن[1] یا آسیب دیدن به طور تصادفی.

[1] بهینه‌ترین موقعیت برای قرار گرفتن سنسورهای تشخیص سقوط در پا و دست است. در واقع اطلاعات منتقل شده توسط تمام بخش‌های بدن برای دستیابی به تشخیص سقوط، مازاد است و عملکرد مناسب ، از نظر دقت ، فقط با مشاهده پا و دست قابل دستیابی است.[1]

در رابطه با تشخیص سقوط با استفاده از دستگاه های محیطی(سنسورهای صوتی[1]) و دستگاه های مبتنی بر دوربین[2]، دستگاه‌های پوشیدنی این مزیت را دارند که نیازی به نصب در تمام نقاط محل سکونت را ندارند در حالی که برای مثال در این دو رویکرد در تمام اتاق‌ها ردیاب های ویدئویی سه بعدی یا آنالیز کننده‌های صوتی مورد نیاز است. از طرف دیگر زمانی که اشیا بزرگ (مانند اساس خانه) محدوده دید دوربین‌ها را مسدود کنند این راه حل دیگر کاربردی نیست زیرا یکی از نیازهای اولیه این رویکرد عدم وجود نقطه کور است. نقص مهم دیگر هر راه حل مبتنی بر دوربین ، نقض حریم خصوصی شخص تحت نظارت است.

به طور کلی، استفاده از گجت‌های پوشیدنی به دلیل گسترش و استفاده همگانی از تلفن‌های همراه و ساعت‌های هوشمند می‌تواند به راحتی از طرف افراد سالمند نیز مورد پذیرش قرار گیرند و این گجت‌ها در صورت دریافت داده‌های غیر معمول یا تشخیص سقوط به سرعت به مراکز درمانی و اورژانس و یا به نزدیکان شخص اطلاع رسانی کند.

1-3- هدف و مهمترین دستاوردها

مهمترین هدف این پروژه تشخیص هوشمند سقوط با پیاده سازی و استفاده از مدل‌ها و الگوریتم‌های یادگیری ماشین و یادگیری عمیق و در نهایت پیدا کردن بهترین و بهینه‌ترین روش برای تشخیص با مقایسه شاخص‌های آماری و همچنین بهینه کردن Confusion Matrix و بالا بردن معیارهای accuracy، precision، recall و نمره F1، این روش‌ها با استفاده از تکنیک‌های پنجره بندی و متعادل سازی داده‌های مجموعه داده است.

1-4- خلاصه مطالب بیان‌شده در این گزارش

در این گزارش، ابتدا در فصل دوم به معرفی روش‌های پایه‌ای در زمینه تشخیص سقوط می‌پردازیم. سپس، روش‌های SVM، k نزدیکترین همسایگی، Logistic Regression، و شبکه‌های عصبی عمیق به‌عنوان مهمترین روش‌های مطرح در زمینه تشخیص سقوط به‌صورت جزئی‌تر مورد بررسی قرار می‌گیرند. در فصل سوم، ابتدا مجموعه‌داده‌های SisFall و SisFall Enhanced در این پروژه مورد استفاده قرار گرفته‌اند توضیح داده می‌شود. سپس الگوریتم ارائه‌شده در این پایان‌نامه به‌تفصیل بررسی می‌شود و در نهایت نحوه پیاده سازی مدل‌ها و کدهای اجرایی آن‌ها و توضیحاتی در نحوه عملکرد کدها به همراه نتایج مدل‌ها به صورت مرحله

[1] Acoustic Sensors

[2] Computer Vision

به مرحله و سپس نتایج شبیه‌سازی‌ها همراه با تحلیل آن‌ها با استفاده از شاخص‌های precision، recall، F1score و تعداد باری که سقوط در هر مرحله درست تشخیص داده شده، آورده می‌شوند. این گزارش در فصل چهارم با ارائه نتیجه‌گیری و پیشنهاد کارهای آینده به اتمام می‌رسد. مراجع مورد استفاده نیز در انتهای گزارش ارائه می‌شوند.

فصل 2:

تشخیص خودکار سقوط

2-1- روش‌های موجود در زمینه تشخیص سقوط

هنگامی که یک فرد ، به ویژه یک فرد مسن ، زمین می خورد ، اغلب منجر به عوارض زیادی از جمله آسیب های تهدید کننده زندگی می شود. در بسیاری از موارد، افرادی که سقوط می کنند ممکن است بلافاصله کمک پزشکی دریافت نکنند (به ویژه افرادی که به تنهایی زندگی می‌کنند) و عدم این اتفاق برای جلوگیری از عوارض و آسیب‌های عمده بسیار حائز اهمیت است. از این رو دستگاه‌های ردیابی و تشخیص خودکار سقوط برای مقابله با این مسئله هستند بسیار مهم هستند.

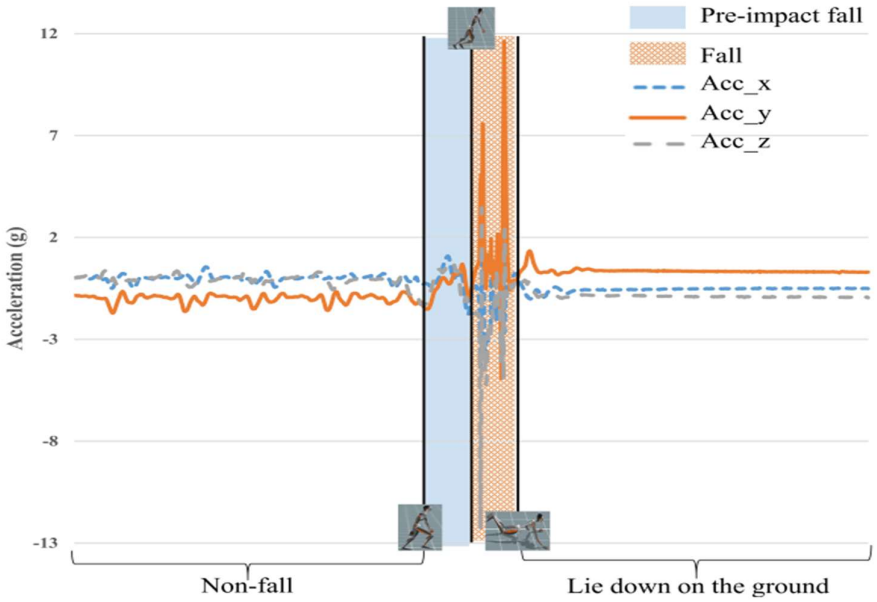
به طور کلی برای تشخیص هوشمند سقوط می‌توان از روش‌ها و مدل‌های مختلف هوش مصنوعی از جمله Computer Vision که مبتنی بر داده‌های دوربین‌ها است[2] که در این مقاله نویسندگان از طریق بررسی و تحلیل تغییر شکل‌های بدن از دنباله‌ای از فریم‌های فیلم سقوط را از فعالیت‌های عادی فرد تشخیص دادند که دارای دقت بسیار بالایی است ولی به دلیل حجم زیاد داده‌های ورودی و محاسبات زیاد پیچیدگی فراوان و سرعت پایینی دارد؛

در این پروژه روش دیگری که مدل‌های یادگیری ماشین SVM، k نزدیکترین همسایگی، Logistic Regression، و شبکه‌های عصبی عمیق که مبتنی به داده‌های سنسورهای شتاب سنج‌های و ژیروسکوپ 3محوری هستند، پیاده سازی و از مجموعه داده SisFall برای پایگاه داده استفاده شده است.

2-2- جمع‌بندی

در این پروژه مدل‌های یادگیری ماشین SVM، k نزدیکترین همسایگی، Logistic Regression و شبکه‌های عصبی عمیق با لایه‌های متفاوت پیاده سازی و بررسی شد که چالش اصلیی که در این پروژه با آن رو به رو شدیم این بود که فقط در لحظاتی که سنسورها یک قله peak نشان می‌دانند که در واقع دقیقا همان لحظه‌ای بود که سقوط اتفاق می‌افتد است و به دلیل محدود بودن و کم بودن این نقاط نسبت به حالتی که سنسورهای شتاب سنج و ژیروسکوپ مقدارهای عادی گزارش می‌دهند (که شامل زمان‌های قبل و بعد از سقوط هم است) کافی بود که مدل‌های یادگیری ماشین فقط عدم سقوط تشخیص دهند و از مقدارهایی که سقوط را نشان می‌دهند صرف نظر کند و با این اتفاق بدون داشتن نرخ یادگیری مناسب می‌توانستیم به دقت بالایی دست پیدا کنیم که این موضوع اصلا نتیجه مناسبی برای ما نیست. به دلیل وجود این مشکل برای رفع این آن و گذشتن از این چالش ما روی تکنیک‌های پنجره‌بندی و متعادل کردن داده تمرکز زیادی کردیم که به طور خلاصه هدف از این کار دادن ارزش بیشتر یا به زبان دیگر وزن‌دار کردن حالاتی بود که در آن‌ها سقوط اتفاق

افتاده و ما برچسب سقوط داریم. این امر در نهایت منجر به رسیدن مقدار مطلوبی از شاخص‌های یادگیری با دقت مناسب که نیاز داریم در تمام مدل‌های پیاده سازی شده در این پروژه شد.



شکل 1-2. نمودار یک سنسور شتاب سنج 3محوری برای سقوط

فصل 3:

روش پیشنهادی

3-1- مقدمه

سقوط یکی از نگرانی های اصلی ایمنی افراد سالمند است و تشخیص سریع آن گام مهمی در جهت تعمین سلامت این افراد به ویژه آن‌هایی که به تنهایی زندگی می‌کنند است و که از آسیب‌های شدید که ممکن است به دلیل تاخیر نیروی امدادی و کمک پزشکی رخ دهد جلوگیری کرده و از حجم قابل توجهی از آسیب‌های جانی، مالی، روانی و اجتماعی که ممکن است در اثر این اتفاق رخ دهد جلوگیری کند. در این پروژه قصد داریم با پیاده سازی الگوریتم‌های هوش مصنوعی سقوط را به صورت هوشمند شناسایی کنیم و تشخیص دهیم.

در این فصل گزارش، به نحوه پیاده سازی و نتیجه‌ی نهایی حاصل از آن پرداخته شده است. در ادامه این فصل ابتدا مجموعه داده مورد استفاده در شبیه‌سازی‌ها در بخش 3-2 مورد بررسی قرار گرفته است. در قسمت 3-3 توضیحی راجع به پلتفرم کدنویسی گوگل کولب که در پیاده سازی پروژه از آن استفاده شده، آورده شده. الگوریتم‌های تشخیص سقوط پیاده‌سازی شده در این پروژه در بخش 3-4 توضیح داده می‌شود. سپس در قسمت 3-5 کدها و توابع مهم که برای پیاده سازی نیاز بود، نوشته شده به همراه توضیحی راجع به آن‌ها آورده شده. سپس در بخش 3-6 پیاده سازی مدل‌ها و کدهای آن‌ها به همراه توضیح آنها و نتایج شبیه سازی (که شامل جداول و توضیحاتی راجع به نتایج است) ارائه شده. در بخش 3-7 نتایج به دست آمده با از مدل‌های پیاده سازی شده، مقایسه شده است.

3-2- مجموعه‌داده[1]

مجموعه داده مورد استفاده در شبیه‌سازی‌ها در این بخش مورد بررسی قرار می‌گیرد.

3-2-1- توضیحات کلی مجموعه داده SisFall

برای پیاده‌سازی مدل‌های این پروژه از مجموعه داده 'SisFall: A Fall and Movement Dataset' استفاده شده که در سال 2016 و توسط دپارتمان مهندسی دانشگاه Udea تهیه شده است. این پایگاه داده از 4510 فایل تشکیل شده است که هر فایل شامل یک فعالیت مجزا است.

[1] Dataset

جدول 1-3. فعالیت‌های عادی روزانه[1]

کد فایل	فعالیت انجام شده	تعداد تکرار	مدت‌زمان (ثانیه)
D01	آهسته راه‌رفتن	1	**100**
D02	تند راه‌رفتن	1	**100**
D03	نرم دویدن	1	**100**
D04	دویدن با سرعت زیاد	1	**100**
D05	بالا و پایین رفتن از پله‌ها با سرعت کم	5	**25**
D06	بالا و پایین رفتن از پله‌ها با سرعت زیاد	5	**25**
D07	آرام نشستن روی صندلی، صبر کردن و آرام بلند شدن	5	**12**
D08	سریع نشستن روی صندلی، صبر کردن و سریع بلند شدن	5	**12**
D09	آرام نشستن روی صندلی کوتاه، صبر کردن و آرام بلند شدن	5	**12**
D10	سریع نشستن روی صندلی کوتاه، صبر کردن و سریع بلند شدن	5	**12**
D11	نشستن، تلاش برای بلند شدن و دوباره نش	5	**12**
D12	نشستن، آرام درازکشیدن صبر کردن و دوباره نشستن	5	**12**
D13	نشستن، سریع درازکشیدن، صبر کردن و دوباره نشستن	5	**12**
D14	درازکشیدن روی یک شانه و سپس درازکشیدن روی شانه دیگر	5	**12**
D15	ایستادن، آرام نشستن روی زانو و دوباره ایستادن	5	**12**
D16	ایستادن، نشستن و دوباره ایستادن	5	**12**
D17	ایستادن، سوارشدن توی ماشین و دوباره بیرون آمدن	5	**12**
D18	تلوتلوخوردن موقع راه‌رفتن	5	**12**
D19	با تمام قدرت پریدن بدون افتادن	5	**12**

جدول 2-3. انواع شرایط سقوط

کد فایل	فعالیت انجام شده	تعداد تکرار	مدت‌زمان (ثانیه)
F01	افتادن از جلو موقع راه‌رفتن براثر لیز خوردن	5	**15**
F02	افتادن از عقب موقع راه‌رفتن براثر لیز خوردن	5	**15**
F03	افتادن از کنار موقع راه‌رفتن براثر لیز خوردن	5	**15**
F04	افتادن از جلو موقع راه‌رفتن براثر لغزش	5	**15**
F05	افتادن از جلو موقع دویدن براثر لغزش	5	**15**
F06	غش کردن موقع راه‌رفتن	5	**15**
F07	غش کردن موقع راه‌رفتن و کمک‌گرفتن از دست‌ها	5	**15**
F08	افتادن از جلو موقع تلاش کردن برای بلند شدن	5	**15**
F09	افتادن از کنار موقع تلاش کردن برای بلند شدن	5	**15**
F10	افتادن از جلو موقع تلاش کردن برای نشستن	5	**15**
F11	افتادن از عقب موقع تلاش کردن برای نشستن	5	**15**
F12	افتادن از کنار موقع تلاش کردن برای نشستن	5	**15**
F13	افتادن از جلو موقع نشستن براثر خوابیدن یا غش کردن	5	**15**

[1] Activities of Daily Living (ADL)

F14	افتادن از عقب موقع نشستن براثر خوابیدن یا غش کردن	5	**15**
F15	افتادن از کنار موقع نشستن براثر خوابیدن یا غش کردن	5	**15**

جدول 3-3. مشخصات افراد شرکت کننده

کد فرد	سن	قد	وزن	جنسیت
SA01	26	165	53	زن
SA02	23	176	58	مرد
SA03	19	156	48	زن
SA04	23	170	72	مرد
SA05	22	172	70	مرد
SA06	21	169	58	مرد
SA07	21	156	63	زن
SA08	21	149	41	زن
SA09	24	165	64	مرد
SA10	21	177	67	مرد
SA11	19	170	80	مرد
SA12	25	153	47	زن
SA13	22	157	55	زن
SA14	27	160	46	زن
SA15	25	160	52	زن
SA16	20	169	61	زن
SA17	23	182	75	مرد
SA18	23	181	73	مرد
SA19	30	170	76	مرد
SA20	30	150	42	زن
SA21	30	183	68	مرد
SA22	19	158	50	زن
SA23	24	156	48	زن
SE01	71	171	102	مرد
SE02	75	150	57	زن
SE03	62	150	51	زن
SE04	63	160	59	زن
SE05	63	165	72	مرد
SE06	60	163	79	مرد
SE07	65	168	76	مرد
SE08	68	163	62	زن
SE09	66	167	65	مرد
SE10	64	156	66	زن
SE11	66	169	63	زن
SE12	69	164	56	مرد
SE13	65	171	73	مرد
SE14	67	163	58	مرد
SE15	64	150	50	زن

SA: افراد بزرگسال بین 19 تا 30 سال
SE: افراد مسن بین 60 تا 75 سال
افراد مسن فقط ADL را شبیه سازی می کنند به غیر SE06 ، که یک متخصص در جودو است که سقوط را هم شبیه سازی می کند. افراد مسن به دلیل توصیه های پزشک متخصص در ورزش فعالیت های D06، D13، D18 و D19 را انجام ندادند. علاوه بر این ، برخی از افراد مسن به دلیل نقص شخصی (یا توصیه پزشکی) برخی فعالیت ها را انجام نمی دهند.

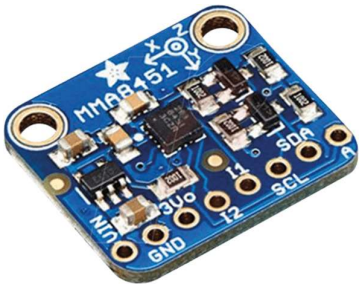
3-2-2- مجموعه داده SisFall Enhanced

علاوه بر مجموعه داده SisFall، از مجموعه داده SisFall Enhanced نیز در این پروژه استفاده شده است که در این مجموعه داده برچسب‌های مربوط فایل در مجموعه داده اولیه آورده شده است. در این مجموعه داده حالات مختلف سقوط به 3 دسته تقسیم شده‌اند به این صورت که حالات عدم سقوط دارای برچسب 0، حالات پیش از سقوط دارای برچسب 2 و حالات سقوط دارای برچسب 1 هستند.

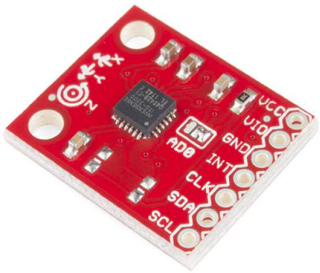
3-2-3- سنسورهای استفاده شده در مجموعه داده

برای تولید این مجموعه داده از سه سنسور شامل دو شتاب‌سنج 3محوری[1] ADXL345 و MMA8451Q و یک ژیروسکوپ 3محوری[2] ITG3200 استفاده شده است.

شکل 3-1. شتاب‌سنج ADXL345 با دامنه +-16g



شکل 3-2. شتاب‌سنج MMA8451 با دامنه +-8g



شکل 3-3. ژیروسکوپ ITG3200 با دامنه +-2000°/s

[1] Triaxial accelerometer
[2] Triaxial gyroscope

که هرکدام از این سنسورها از سه پارامتر در جهت‌های x و y و z تشکیل شده اند. به همین خاطر ستون‌های هر یک از فایل‌های مجموعه از نه ستون تشکیل شده که ستون اول تا سوم مربوط به شتاب‌سنج ADXL345، ستون چهارم تا ششم مربوط به ژیروسکوپ ITG3200 و ستون هفتم تا نهم مربوط به شتاب‌سنج MMA8451Q است.

برای تبدیل داده های شتاب[1] داده شده در بیت به جاذبه، از این معادله استفاده شده است:

$$\text{Acceleration(g)} = [(2*\text{Range}) / (2\text{^Resolution})] * \text{AD} \quad (3\text{-}1)$$

برای تبدیل داده چرخش[2] داده شده در بیت به سرعت زاویه ای، از این معادله استفاده شده است:

$$\text{Angular velocity (°/s)} = [(2*\text{Range}) / (2\text{^Resolution})] * \text{RD} \quad (3\text{-}2)$$

3-3- پلتفرم کدنویسی و کتابخانه‌های مورداستفاده

برای کدنویسی این پروژه از زبان python3.7 به‌عنوان زبان برنامه‌نویسی مرجع و برای پیاده‌سازی کدها از پلتفرم google colab استفاده شده که 12.69 GB Ram و 107 GB Disk به‌عنوان سخت‌افزار در اختیار کاربر قرار می‌دهد. همین‌طور Tensorflow و Sklearn کتابخانه‌های مهمی هستند که برای پیاده‌سازی کدها استفاده شده است.

3-4- الگوریتم پیشنهادی

الگوریتم پیشنهادی مورد استفاده در این بخش توضیح داده می‌شود. علاوه بر توضیح الگوریتم اصلی مثلاً الگوریتم طبقه بندی سقوط، ویژگی[3] های مورد استفاده، نحوه پنجره بندی، نحوه انتخاب پارامترها، نحوه تقسیم مجموعه داده به زیرمجموعه‌های آموزش، آزمون، و ارزیابی و هر توضیح دیگری در موارد مشابه باید در این قسمت آورده شود.

در این پروژه قصد داریم با استفاده از مدل‌های هوش مصنوعی SVM، k نزدیکترین همسایگی، Logistic Regression و شبکه‌های عصبی چندلایه، سقوط یک شخص را به صورت هوشمند با استفاده از داده‌های مجموعه داده تشخیص دهیم.

[1] acceleration data (AD)

[2] rotation data (RD)

[3] Feature

در الگوریتم پیاده‌سازی شده در این پروژه ابتدا آدرس همه فایل‌های داخل مجموعه داده جمع‌آوری شده و در داخل یک آرایه 2بعدی ذخیره می‌شود. سپس این آدرس‌ها به صورت تصادفی بین زیرمجموعه‌های آموزش، آزمون و ارزیابی تقسیم‌بندی می‌شوند به طوری که 70 درصد آدرس‌های مجموعه داده به آموزش و 30 درصد به آزمون و ارزیابی مدل‌ها اختصاص داده شود. سپس برای همه آدرس‌های موجود در هر یک از این زیرمجموعه‌ها سه عملیات مهم انجام می‌شود. ابتدا داده‌ها و پارامترهای سنسورها برای هر آدرس خوانده شده و در مرحله بعدی ما ویژگی‌هایی که برای آموزش مدل‌ها در نظر داریم را از این پارامترها استخراج کرده و در آرایه‌های مجزا ذخیره می‌کنیم.

ویژگی‌هایی که برای آموزش هرکدام از این مدل‌ها استفاده شده، جذر مجموع توان دوهای مقادیر پارامترهای محوری سه سنسور موجود در این مجموعه داده به همراه برچسب‌های موجود در مجموعه داده Sisfall_Enhanced است.

پس از آن این ویژگی‌ها و برچسب‌های استخراج شده برای هر بخش به طور مجزا پنجره‌بندی می‌شوند و در نهایت سه (دو) مجموعه پنجره‌بندی شده در اختیار ما قرار می‌گیرد تا استفاده از آن‌ها مدل‌ها آموزش ببینند.

درپایان برای ارزیابی و مقایسه هرکدام از مدل‌های پیاده سازی شده از confusion matrix و سه معیار recall ، precision و F1score استفاده شده است.

3-5- پیاده‌سازی روش پیشنهادی

3-5-1- کتابخانه‌ها و frameworkها

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
import tensorflow as tf
import os
import glob
from math import sqrt
from sklearn import preprocessing
from sklearn import svm
window_size = 200
```

شکل 4-3. کتابخانه‌ها و framework

در پیاده‌سازی این پروژه از کتابخانه‌های Pandas برای کارکردن با dataframe ها و فایل‌های CSV، از Numpy برای کار کردن با vectorها و همین‌طور انجام عملیات‌های ریاضی، از Tensorflow برای پیاده‌سازی شبکه عصبی و از preprocessing کتابخانه SKLearn برای normalization استفاده شده است.

3-5-2- دریافت مجموعه داده‌های SisFall و SisFall Enhanced

```
!wget http://sistemic.udea.edu.co/wp-content/uploads/2016/03/SisFall_dataset.zip
!gdown --id 1gvOuxPc8dNgTnxuvPcVuCKifOf98-TV0
!unzip SisFall_dataset.zip
!unzip SisFall_enhanced.zip
```

شکل 3-5. دریافت مجموعه داده‌ها

با استفاده از دستورات !wget و !gdown مجموعه داده‌های Sisfall و Sisfall_enhanced را دانلود و با استفاده از دستور !unzip این مجموعه داده‌ها را از حالت zip خارج کرده و uncompress می‌کنیم.

3-5-3- استخراج آدرس فایل‌های داخل مجموعه داده

```
def get_file_name(path):
  allfiles = []
  allFolders = glob.glob(path + "*")
  for files in allFolders:
      allfiles.append(glob.glob(files+"/*.txt"))
  if 'desktop.ini' in allfiles:
        allfiles.remove('desktop.ini')
  return np.hstack(allfiles)
```

شکل 3-6. استخراج همه آدرس‌ها

اینجا تابع get_all_files با استفاده از path ورودی، به دنبال تمام فایل‌های txt. ای می‌گردد که نوه‌ی (داخل یکی از فولدرهای فرزند) آن path باشند. سپس آرایه‌ای از آدرس تمام فایل‌های txt. مطلوب را

برمی‌گرداند.

3-5-4- استخراج اطلاعات مورد نیاز از هر فایل داخل مجموعه داده

```
def read_data(data_path):
    data = pd.read_csv(data_path, header=None)
    data.columns = ['ADXL345_x', 'ADXL345_y', 'ADXL345_z', 'ITG3200_x', 'ITG3200_y', 'ITG3200_z', 'MMA8451Q_x',
                    'MMA8451Q_y', 'MMA8451Q_z']
    data['MMA8451Q_z'] = data['MMA8451Q_z'].map(lambda x: str(x)[:-1])
    for name in data.columns :
      data[name] = data[name].astype(int)
    return data
```

شکل 3-7. خواندن اطلاعات مجموعه داده از آدرس

تابع read_data مسئول خواندن اطلاعات یک فایل مشخص است. این تابع به این شکل عمل می‌کند که ابتدا هر یک ستون‌ها را نام‌گذاری می‌کند. هر فایل شامل ۹ ستون است که ۳ تای اول مربوط به شتاب‌سنج ADXL345 و ۳ تای دوم مربوط به ژیروسکوپ ITG3200 و ۳ تای آخر مربوط به شتاب‌سنج MMA8451Q هستند. اینجا به آخرین کاراکتر پارامتر z شتاب‌سنج MMA8451Q که یک (;) است، نیازی نیست. در نتیجه روی این ستون map زده و هر عضو آن را به عضوی تبدیل می‌کنیم که کاراکتر آخر در آن حذف شده است. حال می‌توان همه‌ی رشته‌ها را به اعداد float تبدیل کرد.

3-5-5- استخراج برچسب‌های مربوط به هر فایل از مجموعه داده Enhanced

```
def get_label(data_path):
    label = data_path[21]
    if label =='D':
      return int(0)
    elif label =='F':
      label_path = data_path.replace('dataset','enhanced')
      labels = pd.read_csv(label_path,header=None)
      labels[labels == 2] = 1
      return labels
```

شکل 3-8. گرفتن برچسب‌ها از دیتاست مکمل

فایل‌ها با استفاده از کاراکتر اول نام آن‌ها دسته‌بندی می‌شوند. به این صورت که اگر با F شروع بشوند، مربوط به حالت fall و اگر با D شروع شوند، مربوط به not-fall هستند. تابع get_label در حالتی که path ورودی، فایل not-fallای را مشخص کند یک بردار تماما صفر برمی‌گرداند و درصورتی که فایل fall بود، ابتدا مسیر فایل مقصد را به نسخه‌ی enhanced در دیتاست Sisfall_enhanced تبدیل می‌کند تا به معادل همین فایل در دیتاست مکمل ارجاع داده شود و برچسب‌های جدید را به شکل یک بردار به‌عنوان خروجی برگرداند.

نکته‌ی قابل توجه این است که در دیتاست مذکور، برای لحظاتی که شخص در حال افتادن است (نه stable است و نه به زمین افتاده) از عدد ۲ استفاده شده. ما برای اینکه می‌خواهیم مدل ۲ کلاسه آموزش دهیم، این اعداد ۲ را تبدیل به ۱ (معادل افتاده) می‌کنیم.

3-5-6- استخراج ویژگی‌های مدل از هر فایل

```
def add_features(dataset,data_path):
    new_dataset = pd.DataFrame()
    new_dataset['acc_1'] = dataset.apply(
        lambda row: sqrt((row.ADXL345_x ** 2 + row.ADXL345_y ** 2 + row.ADXL345_z ** 2)), axis=1)
    new_dataset['acc_2'] = dataset.apply(
        lambda row: sqrt((row.MMA8451Q_x ** 2 + row.MMA8451Q_y ** 2 + row.MMA8451Q_z ** 2)), axis=1)
    new_dataset['geo'] = dataset.apply(
        lambda row: sqrt((row.ITG3200_x ** 2 + row.ITG3200_y ** 2 + row.ITG3200_z ** 2)), axis=1)
    new_dataset['label'] = get_label(data_path)
    return np.ceil(new_dataset.to_numpy())
```

شکل 9-3. استخراج ویژگی‌ها

در تابع add_feature یک جدول جدید ساخته می‌شود که سه ستون اول آن به ترتیب به ریشه‌ی دوم مجموع مجذورات پارامترهای سنسورهای اول تا سوم اشاره می‌کند. در کنار این سه ستون هم با استفاده از آدرس ورودی تابع و تابع get_label برچسب نهایی آن فایل خاص هم استخراج می‌شود و در ستون چهارم dataframe جدید قرار می‌گیرد.

نهایتا همه مقادیر اعداد در dataframe تا دو رقم اعشار گرد می‌شوند و به شکل یک numpy_array برگردانده می‌شوند.

3-5-7- تقسیم‌بندی آدرس‌ها به سه دسته آموزش، ارزیابی و آزمون

```
def split_address(dataset):
  np.random.shuffle(dataset)
  train, validate, test = np.split(dataset, [int(len(dataset)*0.7), int(len(dataset)*0.8)])
  return train, validate, test
```

شکل 3-10. تقسیم بندی آدرس‌ها به زیر مجموعه‌های آموزش، ارزیابی و آزمون

تابع split_address اول تمام مسیرهای دیتاست (اینجا Sisfall) را دریافت کرده، آن‌ها را بُر زده (اصطلاحا shuffle کرده) و نهایتا ۷۰ درصد آن‌ها را تحت عنوان داده‌ی train و ۲۰ درصد را تحت عنوان داده‌ی test و ۱۰ درصد مابقی را نیز به‌عنوان داده‌ی validation مدل برمی‌گرداند.

نکته‌ی حائز اهمیت این است که در حالتی که شبکه عصبی ما همه‌ی epochهای خود را طی کند، نیازی به آدرس‌های validation نداریم و می‌توانیم آن‌ها را به آدرس‌های test اضافه کنیم.

3-5-8- استخراج ویژگی‌های مورد نیاز از آدرس‌ها

```
def datasets_to_nparray(datasets_address_array):
  result = np.empty((0, 4), int)
  for address in datasets_address_array:
    result = np.concatenate(
        (result,add_features(read_data(address),address)),axis=0)
  return result
```

شکل 3-11. استخراج ویژگی‌ها از آدرس‌ها

تابع datasets_to_nparray یک numpy array با چهار ستون (سه ستون ویژگی و یک ستون برچسب) ایجاد می‌کند. سپس این تابع روی آرایه‌ای از تمام آدرس‌های ورودی حرکت کرده و داده‌های هر کدام از این فایل‌ها را به این numpy array اضافه می‌کند. در واقع داده‌های تمام فایل‌ها را در این آرایه، concatenate می‌کند تا همه را در یک آرایه داشته باشیم. سپس این آرایه به عنوان result برگردانده می‌شود.

9-5-3- پنجره بندی

```
def windowing(dataset,window_size):
  window = window_size * (dataset.shape[1]-1)
  cut = dataset.shape[0] % window_size
  feature = dataset[:-cut,0:-1]
  label = dataset[:-cut,-1]
  feature = feature.ravel().reshape(feature.size//window,window)
  label = label.reshape(label.size//window_size,window_size)
  label = label.sum(axis=1)
  label[label > 0] = 1
  return feature,label
```

شکل 12-3. پنجره بندی دیتاست

در این تابع قصد داریم ویژگی‌ها را پنجره بندی کنیم که به این منظور نیاز است اندازه هر پنجره را بگیریم. ابعاد هر پنجره ۲۰ در نظر گرفته شده است. پس با توجه به اینکه ۳ ویژگی کلی داریم، در هر پنجره ۶۰۰ ویژگی متفاوت به همراه یک مقدار مربوط به برچسب آن پنجره قرار می‌گیرد و نهایتا ۶۰۱ داده داخل هر پنجره قرار می‌گیرد.

حال چون، ممکن است برای ایجاد آخرین پنجره داده کافی وجود نداشته باشد(با توجه به سایز هر پنجره و تعداد ویژگی‌ها) باید داده‌های اضافی که در هیچ پنجره‌ی پُری جا نمی‌شوند و باقی می‌مانند، حذف شوند.

سپس برای پنجره‌بندی باید ویژگی‌ها و ستون‌ها را دوباره تقسیم‌بندی و به پنجره تبدیل کنیم. این کار را با تبدیل کردن ویژگی‌ها به یک آرایه‌ی ۱ بعدی (با استفاده از دستور ravel) و تبدیل آن آرایه به پنجره (با استفاده از دستور reshape) انجام می‌دهیم.

این امر روی برچسب‌ها نیز اتفاق می‌افتد. با این تفاوت که اگر حتی در یک بخش کوچک از هر پنجره سقوط رخ داده باشد، همه آن پنجره به‌عنوان پنجره افتادن در نظر گرفته می‌شود.

3-5-10- ذخیره سازی به شکل tensor

```
def dataset_to_tensor(validate,test,train,window_size):
  validate_feature , validate_label  = windowing(datasets_to_nparray(validate),window_size)
  np.savez('Sisfall_data_validation', inputs=validate_feature, targets=validate_label)
  test_feature , test_label = windowing(datasets_to_nparray(test),window_size)
  np.savez('Sisfall_data_test', inputs=test_feature, targets=test_label)
  train_feature , train_label = windowing(datasets_to_nparray(train),window_size)
  np.savez('Sisfall_data_train', inputs=train_feature, targets=train_label)
```

شکل 3-13. تبدیل numpy array به tensor

برای پیاده سازی شبکه عصبی داده‌های ورودی به مدل باید به شکل tensor باشند بنابراین در این قسمت، داده‌ها باید برای ورودی داده شدن به Tensorflow آماده شوند. اولین قدم این است که آن‌ها را تبدیل به tensor کنیم. سپس tensor های نهایی مربوط به ۳ دسته‌ی train، test و validation را در ۳ فایل uncompressed با دستور np.savez به فرمت npz ذخیره می‌کنیم. از این تابع ساخته‌شده در بخش بعدی استفاده می‌شود.

3-5-11- آماده سازی داده‌ها برای مدل سازی

```
train, validate, test = split_address(get_file_name("SisFall_dataset/"))

dataset_to_tensor(validate,test,train,window_size)
```

شکل 3-14. آماده سازی داده‌ها

حال با استفاده از تابع split_address که در بخش 3-5-7 ساخته بودیم، داده‌ها را به ۳ دسته می‌شکنیم و با استفاده از تابع dataset_to_tensor آن‌ها را به‌عنوان tensor ذخیره می‌کنیم.

```
npz = np.load("Sisfall_data_train.npz")
train_inputs = preprocessing.scale(npz["inputs"].astype(np.float))
train_targets = npz["targets"].astype(np.int)

npz = np.load("Sisfall_data_validation.npz")
validation_inputs = preprocessing.scale(npz["inputs"].astype(np.float))
validation_targets = npz["targets"].astype(np.int)

npz = np.load("Sisfall_data_test.npz")
test_inputs = preprocessing.scale(npz["inputs"].astype(np.float))
test_targets = npz["targets"].astype(np.int)
```

شکل 15-3. ذخیره سازی و نرمال سازی فایل‌های tensor

حال با استفاده از کتابخانه‌ی SKLearn مقادیرtensor ها را از فایل‌های npz مذکور خوانده، آن‌ها را نرمال سازی[1] کرده (برای افزایش سرعت و دقت مدل‌ها) و در ۳ متغیر نگه می‌داریم.

```
print(validation_inputs.shape)
print(test_inputs.shape)
print(train_inputs.shape)
```

```
(7863, 600)
(15492, 600)
(55938, 600)
```

شکل 16-3. ابعاد 3 دسته آموزش، ارزیابی و آزمون

3-6- پیاده سازی مدل‌ها و نتایج شبیه سازی

همان طور که در قسمت‌های مختلف این گزارش گفته شد برای پیاده سازی این پروژه از مدل‌های هوش مصنوعی SVM، k نزدیکترین همسایگی، Logistic Regression و شبکه‌های عصبی عمیق استفاده می‌کنیم. همچنین در این بخش نتایج شبیه سازی مدل‌ها بعد از متعادل سازی داده‌ها مجموعه آموزش نیز بیان شده است.

[1] normalize

1-6-3- متعادل سازی داده‌های مجموعه آموزش

```
new_train_dataset = np.concatenate((train_inputs,train_targets.reshape((train_targets.shape[0],1))),axis=1)
df = pd.DataFrame(new_train_dataset, columns = None)
df.columns = [*df.columns[:-1], 'label']
reps = [128 if val == 1 else 1 for val in df.label]
df = df.loc[np.repeat(df.index.values, reps)].reset_index(drop=True)
new_train = df.to_numpy()
print(new_train_dataset.shape)
new_train.shape
```

شکل 17-3. نمونه شکل

هدف از متعادل سازی مجموعه آموزش بالا بردن اهمیت زمان‌هایی است که fall اتفاق افتاده و داده‌ها برچسب 1دارند (در واقع وزن‌دار کردن حالت fall در مجموعه آموزش) زیرا همان طور که در بخش 2-2 گفته شد حالت‌های fall بسیار کم هستند؛ پس ما با افزایش اهمیت حالات fall، قصد داریم این حالت‌ها را بهتر تشخیص دهیم.

برای انجام این کار باید تعداد تکرار[1] هرکدام از برچسب‌های fall را افزایش دهیم. به همین منظور ابتدا باید سطرهایی از دیتاست که fall هستند را تشخیص دهیم سپس با تکرار این سطرها به تعداد آن‌ها بیفزاییم.

[1] repetition

3-6-2- شبکه عصبی 4لایه به همراه مجموعه validation

```
input_size = 3
output_size = 1
hidden_layer_size = 50

model = tf.keras.Sequential([
                            tf.keras.layers.Dense(hidden_layer_size,activation="relu"),
                            tf.keras.layers.Dense(hidden_layer_size, activation="relu"),
                            tf.keras.layers.Dense(hidden_layer_size,activation="relu"),
                            tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

batch_size = 100
max_epochs = 50

early_stopping = tf.keras.callbacks.EarlyStopping(patience=4)

model.fit(train_inputs,
          train_targets,
          batch_size=batch_size,
          epochs=max_epochs,
          callbacks=[early_stopping],
          shuffle = True,
          validation_data=(validation_inputs,validation_targets),
          verbose = 1
          )
```

شکل 18-3. مدل شبکه عصبی 4لایه با validation

شبکه‌ی عصبی پیاده‌سازی شده، شامل ۴ لایه می‌باشد که هر لایه داخلی آن‌ها از پنجاه node ساخته شده است. تابع فعال‌ساز لایه‌های داخلی Relu است و برای فعال‌سازی لایه‌ی آخر، با توجه به binary بودن خروجی نهایی از تابع Sigmoid استفاده شده است. Optimizer این شبکه‌ی عصبی تابع Adam بوده و loss به شکل binary_corssentropy محاسبه می‌شود. شبکه‌ی عصبی ما در نهایت یک پنجره‌ی ۶۰۰ عضوی را دریافت کرده و یکی از دو عدد صفر به معنای not-fall یا یک به معنای fall و را برمی‌گرداند.

```
[[16050     0]
 [  660    53]]
```

	Precision	Recall	F1score
0	1.0	0.074334	0.148668

شکل 19-3. نتایج شبکه عصبی 4لایه با validation

در تصویر بالا پارامترهای آماری و confusion matrix این شبکه‌ی عصبی نشان داده شده است.

با توجه به نتایج، دیده می‌شود که این شبکه‌ی عصبی با به دلیل اینکه اکثر داده‌های ورودی‌اش برچسب صفر (not-fall) داشته‌اند، درصد زیادی از پیش‌بینی‌ها را not-fall در نظر گرفته است. به همین دلیل این مدل نرخ یادگیری پایینی داشته و معیارهای recall و F1score آن پایین است.

3-6-3- شبکه عصبی 4لایه بدون مجموعه validation

```
input_size = 600
output_size = 1
hidden_layer_size = 50

model = tf.keras.Sequential([
                            tf.keras.layers.Dense(input_size,activation="relu"),
                            tf.keras.layers.Dense(hidden_layer_size, activation="relu"),
                            tf.keras.layers.Dense(hidden_layer_size,activation="relu"),
                            tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

batch_size = 100
max_epochs = 50

model.fit(train_inputs,
          train_targets,
          batch_size=batch_size,
          epochs=max_epochs,
          shuffle = True,
          verbose = 1
          )
```

شکل 3-20. شبکه عصبی 4لایه بدون validation

برای حل این مشکل، یکی از راه‌های معمول، اجرای مدل بدون دیتاست validation است.

این بار دیتاست validation را به دیتاست test اضافه می‌کنیم و مدل را به‌صورت کامل train می‌کنیم.

نتایج تست مدل جدید را در تصویر زیر مشاهده می‌کنید.

```
------ Confusion Matrix Created ------
array([[22542,     3],
       [  751,   395]], dtype=int32)
```

	Precision	Recall	F1score
0	0.992462	0.344677	0.684158

شکل 21-3. نتایج شبکه عصبی بدون validation

مشکلی که در مرحله‌ی قبل به‌وجود آمده بود تا حد زیادی رفع شده است.

همین‌طور، پارامترهای آماری precision، recall و F1score رشد چشم‌گیری داشتند.

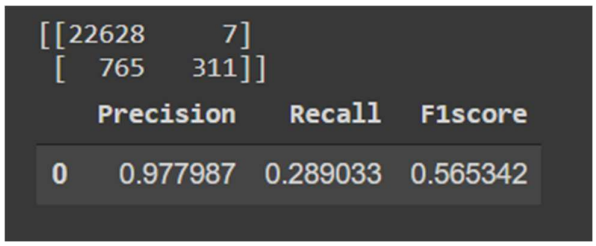
3-6-4- شبکه عصبی 5لایه[1]

برای پیاده سازی این مدل مانند شبکه عصبی 4لایه عمل می‌کنیم و یک لایه به مدل قبلی اضافه می‌کنیم.

```
input_size = 600
output_size = 1
hidden_layer_size = 50

model = tf.keras.Sequential([
                            tf.keras.layers.Dense(input_size,activation="relu"),
                            tf.keras.layers.Dense(hidden_layer_size,activation="relu"),
                            tf.keras.layers.Dense(hidden_layer_size,activation="relu"),
                            tf.keras.layers.Dense(hidden_layer_size,activation="relu"),
                            tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

batch_size = 100
max_epochs = 50

model.fit(train_inputs,
          train_targets,
          batch_size=batch_size,
          epochs=max_epochs,
          shuffle = True,
          verbose = 1
          )
```

شکل 22-3. مدل شبکه عصبی 5لایه

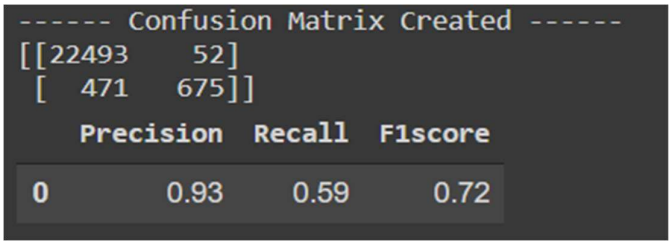
[1] پس از رسیدن به نتیجه این که شبکه عصبی بدون validation نتایج بهتری دارد، برای پیاده سازی شبکه عصبی 5لایه و شبکه عصبی‌های متعادل سازی شده هم بدون validation در نظر گرفته شده‌اند.



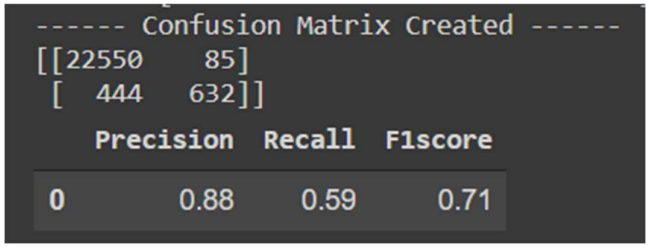
شکل 23-3. نتایج شبکه عصبی 5لایه

5-6-3- شبکه عصبی 4لایه و 5لایه پس از متعادل سازی

برای پیاده سازی شبکه عصبی 4لایه و 5لایه که داده‌های آموزش متعادل سازی شده است، میزان تکرار حالت fall در هر دو شبکه عصبی 305 در نظر گرفته شده است.



شکل 24-3. نتایج شبکه عصبی 4لایه پس از متعادل سازی



شکل 25-3. نتایج شبکه عصبی 5لایه پس از متعادل سازی

3-6-6- مدل Logistic Regression

```
x = train_inputs
y = train_targets
x_test = test_inputs
y_test = test_targets

reg = LogisticRegression()
reg.fit(x,y)
```

شکل 3-26. مدل Logistic Regression

برای پیاده سازی این مدل در گام اول ویژگی‌ها و برچسب‌های train و test را به صورت جداگانه تعیین میکنیم و سپس مدل را با پارمترهای از پیش مشخص شده x و y، fit میکنیم.

```
score on test: 0.953188974716137
score on train: 0.9612071291117386
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     22545
           1       0.56      0.15      0.24      1146

    accuracy                           0.95     23691
   macro avg       0.76      0.57      0.61     23691
weighted avg       0.94      0.95      0.94     23691

[[22409   136]
 [  973   173]]
```

شکل 3-27. نتایج پیاده سازی مدل Logistic Regression

از نتایج به دست آمده می‌بینیم که مدل نهایی با اینکه روی داده‌های fall دقت خوبی دارد، اما روی داده‌های not-fall عملکرد قابل قبولی ندارد. پارامترهای recall و F1score هم نمایان‌گر این موضوع هستند. البته در confusion matrix مشخص می‌شود که تعداد fallهای از دست رفته بیش از حد انتظار است.

3-6-7- مدل Logistic Regression پس از متعادل سازی

برای پیاده سازی این مدل تعداد تکرار حالت fall برای برای هر سطر با برچسب 1 در دیتاست آموزش، 128

در نظر گرفته شده است.

```
score on test: 0.5080832383605589
score on train: 0.8988153186072915
              precision    recall  f1-score   support

           0       0.99      0.49      0.65     22545
           1       0.08      0.93      0.16      1146

    accuracy                           0.51     23691
   macro avg       0.54      0.71      0.40     23691
weighted avg       0.95      0.51      0.63     23691

[[10968 11577]
 [   77  1069]]
```

شکل 28-3. نتایج مدل Logistic Regression پس از متعادل سازی

3-6-8- مدل SVM

```
svm=LinearSVC(C=0.0001)
svm.fit(x, y)
print("score on test: " + str(svm.score(x_test, y_test)))
print("score on train: "+ str(svm.score(x, y)))
y_pred = svm.predict(x_test)
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

شکل 29-3. مدل SVM

برای پیاده سازی این مدل با پارامتر منظم‌سازی[1] c = 0.0001، آن را روی پارمترهای ویژگی‌ها و برچسب‌ها fit کردیم.

[1] Regularization parameter

```
score on test: 0.9580623993318619
score on train: 0.9538953913679591
              precision    recall  f1-score   support

           0       0.96      1.00      0.98     16050
           1       0.63      0.03      0.06       713

    accuracy                           0.96     16763
   macro avg       0.80      0.52      0.52     16763
weighted avg       0.94      0.96      0.94     16763

[[16036    14]
 [  689    24]]
```

شکل 30-3. نتیجه مدل SVM

در نتایج به‌دست آمده مشخص است که این مدل تقریبا همه‌ی برچسب‌ها را not-fall پیش‌بینی می‌کند و در حد انتظار عمل نمی‌کند.

3-6-9- مدل SVM پس از متعادل سازی

با متعادل شدن مجموعه داده‌های آموزش عملکرد و دقت مدل روی حالت fall بسیار افزایش میابد. چون هدف ما در این پروژه تشخیص سقوط است، تعداد تکرار بیشتر حالت fall برای ما نتیجه بهتری دارد (زیرا تشخیص اشتباه در حالت not-fall تنها با یک دکمه بر روی دستگاه حل می‌شود).

```
score on test: 0.9484852201643567
score on train: 0.8842727157841527
              precision    recall  f1-score   support

           0       0.97      0.98      0.97     15615
           1       0.37      0.31      0.34       691

    accuracy                           0.95     16306
   macro avg       0.67      0.64      0.66     16306
weighted avg       0.94      0.95      0.95     16306

[[15251   364]
 [  476   215]]
```

شکل 31-3. متعادل سازی با تکرار 4

```
score on test: 0.9256715319514289
score on train: 0.8388991624061786
              precision    recall  f1-score   support

           0       0.98      0.94      0.96     15615
           1       0.29      0.52      0.37       691

    accuracy                           0.93     16306
   macro avg       0.63      0.73      0.67     16306
weighted avg       0.95      0.93      0.94     16306

[[14736   879]
 [  333   358]]
```

شکل 32-3. متعادل سازی با تکرار 8

```
score on test: 0.8708450876977799
score on train: 0.8044435463285969
              precision    recall  f1-score   support

           0       0.98      0.88      0.93     15615
           1       0.19      0.65      0.30       691

    accuracy                           0.87     16306
   macro avg       0.59      0.77      0.61     16306
weighted avg       0.95      0.87      0.90     16306

[[13751  1864]
 [  242   449]]
```

شکل 33-3. متعادل سازی با تکرار 16

```
score on test: 0.5930332392984178
score on train: 0.8352220337107661
              precision    recall  f1-score   support

           0       0.99      0.58      0.73     15615
           1       0.08      0.88      0.15       691

    accuracy                           0.59     16306
   macro avg       0.54      0.73      0.44     16306
weighted avg       0.95      0.59      0.71     16306

[[9062 6553]
 [  83  608]]
```

شکل 35-3. متعادل سازی با تکرار 64

```
score on test: 0.7711885195633509
score on train: 0.7965462573342026
              precision    recall  f1-score   support

           0       0.99      0.77      0.87     15615
           1       0.13      0.75      0.22       691

    accuracy                           0.77     16306
   macro avg       0.56      0.76      0.54     16306
weighted avg       0.95      0.77      0.84     16306

[[12054  3561]
 [  170   521]]
```

شکل 34-3. متعادل سازی با تکرار 32

```
score on test: 0.2778142337290461
score on train: 0.8778162681761033
              precision    recall  f1-score   support

           0       1.00      0.25      0.40     16050
           1       0.05      0.97      0.10       713

    accuracy                           0.28     16763
   macro avg       0.52      0.61      0.25     16763
weighted avg       0.96      0.28      0.38     16763

[[ 3963 12087]
 [   19   694]]
```

شکل 36-3. متعادل سازی با تکرار 128

3-6-10- مدل KNN

در مدل KNN، با استفاده از تابع کلاس‌بندی این مدل با تعداد 4 همسایگی پیاده سازی کردیم و سپس مدل را روی پارامترهای ویژگی و برچسب fit کردیم.

```
x = train_inputs
y = train_targets
x_test = test_inputs
y_test = test_targets


knn = KNeighborsClassifier(n_neighbors=4)
knn.fit(x, y)
```

شکل 37-3. پیاده سازی مدل KNN

	precision	recall	f1-score	support
0	0.98	1.00	0.99	22635
1	0.93	0.53	0.68	1076
accuracy			0.98	23711
macro avg	0.95	0.77	0.83	23711
weighted avg	0.98	0.98	0.97	23711

```
[[22590    45]
 [  503   573]]
```

شکل 38-3. نتایج مدل KNN

در تصویر بالا نتایج حاصل از پیاده سازی مدل KNN را می‌بینیم که نسبت به مدل‌های دیگر دارای شاخص‌های آماری به نسبت بهتری است و تقریبا نصف حالات سقوط را بدون متعادل سازی مجموعه آموزش تشخیص داده است.

3-6-11- مدل KNN پس از متعادل سازی

برای پیاده سازی مدل **KNN** متعادل سازی شده نرخ تکرار حالت fall در داده‌های آموزش 128 در نظر گرفته شده است.

	precision	recall	f1-score	support
0	0.99	0.99	0.99	22545
1	0.75	0.72	0.73	1146
accuracy			0.97	23691
macro avg	0.87	0.85	0.86	23691
weighted avg	0.97	0.97	0.97	23691

```
[[22275   270]
 [  326   820]]
```

شکل 39-3. نتایج مدل KNN پس از متعادل سازی

12-6-3- نتایج شبیه سازی

در این گزارش چون فقط حالت‌های fall با برچسب 1 برای ما اهمیت دارد، جدول زیر فقط نتایج این حالات را بیان می‌کند.

جدول 4-3. نتایج و معیارهای یادگیری مدل‌های پیاده سازی شده

مدل	Precision	Recall	F1score	Confusion Matrix for Fall
شبکه عصبی 4لایه بدون validation	0.99	0.34	0.68	[751 395]
شبکه عصبی 4لایه پس از متعادل سازی	0.93	0.59	0.72	[471 675]
شبکه عصبی 5لایه بدون validation	0.97	0.29	0.56	[765 311]
شبکه عصبی 5لایه پس از متعادل سازی	0.88	0.59	0.71	[444 632]
Logistic Regression	0.56	0.15	0.24	[973 173]
Logistic Regression پس از متعادل سازی	0.08	0.93	0.16	[77 1069]
SVM	0.63	0.03	0.06	[689 24]
SVM پس از متعادل سازی	0.05	0.97	0.10	[19 694]
KNN	0.93	0.53	0.68	[503 573]
KNN پس از متعادل سازی	0.75	0.72	0.73	[326 820]

7-3- مقایسه و نتیجه‌گیری

در این فصل با مجموعه داده‌های استفاده شده و الگوریتم و مدل‌های پیاده سازی شده آشنا شدیم و نحوه پیاده سازی و عملکرد هر مدل را پیش از متعادل سازی مجموعه داده آموزش و پس از آن مشاهده کردیم. در انتها نیز در بخش 3-6-12 در جدول 3-4 تمام مدل‌های پیاده سازی شده به همراه شاخص‌های آماری و معیارهای یادگیری و تعداد حالات تشخیص درست حالت سقوط آورده شده که با استفاده از آن می‌توان مقایسه دقیقی بر مدل‌ها انجام داد و به نتیجه نهایی رسید.

همان طور که در جدول 3-4 نشان داده شده تمام مدل‌های پیاده سازی شده در این پروژه پس از متعادل سازی داده‌های آموزش رشد محسوسی در شاخص‌های یادگیری و همچنین تشخیص درست حالات سقوط دارند ولی بیشترین نرخ تشخیص درست حالات سقوط پس از متعادل سازی مجموعه آموزش را دو مدل Logistic Regression و SVM دارند و این در حالی است که دقیقا این دو مدل پیش از متعادل سازی بدترین نرخ تشخیص درست حالات سقوط و معیارهای یادگیری precision، recall و F1score را دارند.

نکته حائز اهمیت در اینجا، مدل‌های شبکه عصبی پیاده سازی شده در این پروژه است. اولین نتیجه مهم، رشد چشمگیر شاخص‌های مدل پس از حذف مجموعه validation بود که در بخش 3-6-2 و 3-6-3 کامل به این موضوع اشاره شد. نکته بعدی تاثیر بسیار کم تعداد لایه‌های شبکه عصبی در پیشرفت نتایج مدل‌ها بوده است که این نکته در جدول 3-4 کاملا قابل مشاهده است.

و در انتها مدل KNN که پیش از متعادل سازی داده‌های آموزش می‌توان آن را با توجه به معیارهای نوشته شده در جدول 3-4 به عنوان یک مدل خوب معرفی کرد اما پس از متعادل سازی می‌توان پیشرفتی که در تمام مدل‌ها اتفاق افتاده را در این مدل هم دید.

فصل 4:

جمع‌بندی و پیشنهادها

4-1- نتیجه گیری

در دو فصل اول این گزارش با محتوا کلی و هدف اصلی این پروژه و همچنین راه‌های مختلف انجام این پروژه و پروژه‌های مشابه آشنا شدیم. در فصل سوم به شرح کامل دیتاست‌های مورد نیاز و الگوریتم پیاده سازی شده و نحوه‌ی اجرای آن و همچنین مدل‌های کلاس بندی اجرا شده و نحوه بهبود این مدل‌ها در طول انجام این پروژه کدهای اجرایی آن‌ها و نتایج کامل مدل‌ها و در انتها مقایسه کامل مدل‌های پیاده سازی شده در این پروژه پرداختیم.

دلایل برتری و مزیت‌های اصلی این پروژه نسبت به پروژه‌ها و مدل‌های پیاده شده مشابه در این زمینه می‌توان به کار کردن و استفاده از آدرس فایل‌ها به جای خود فایل‌ها و همین طور numpy array به جای dataframe که باعث افزایش سرعت اجرای الگوریتیم و استفاده از تکنیک‌های پنجره بندی و متعادل سازی مجموعه داده‌های آموزش که عملکرد مدل‌ها تا حد زیادی بهبود بخشید اشاره کرد.

در پایان با مقایسه نتایج به دست آمده از پیاده سازی مدل‌ها با در نظر گرفتن معیارهای یادگیری و شاخص‌های آماری می‌توان شبکه عصبی را به عنوان بهترین مدل معرفی کرد ولی با در نظر گرفتن تعداد تشخیص‌های درست حالت سقوط و زمان پیاده سازی مدل‌ها می‌توان مدل SVM را به عنوان بهترین مدل پیاده سازی این پروژه در نظر گرفت که البته با بهینه‌تر کردن الگوریتم پیاده شده و استفاده از کارهایی که در بخش بعد به آن پرداخته شده می‌توان به پیشرفت چشمگیری در این پروژه رسید.

4-2- پیشنهادهایی برای کارهای آتی

ایده‌ای که در بهبود عملکرد این پروژه می‌توان از آن‌ها استفاده کرد ترکیب رویکرد پیاده شده در این پروژه و باقی رویکردها مانند computer vision و استفاده از نتایج هر دو رویکرد برای نتیجه‌گیری بهتر و دقیق‌تر است.

همچنین استفاده از انحراف معیار داده‌های سنسورها به جای نرم دوم داده‌های سنسورها برای استخراج ویژگی‌ها و یا تغییر شیوه پنجره بندی به نحوی که پنجره‌ها باهم هم‌پوشانی ایجاد کنند نیز می‌تواند در عملکرد مدل‌ها بهبود حاصل کند.

فصل 5:

مراجع

مراجع

[1] Quoc T.Huynh, Uyen D. Nguyen, Lucia B. Irazabal, Nazanin Ghassemian, and Binh Q. Tran. Optimization of an Accelerometer and Gyroscope-Based Fall Detection Algorithm. Hindawi Publishing Corporation. 2015.

[2] Caroline Rougier, Jean Meunier, Alain St-Arnaud, Jacqueline Rousseau. Procrustes Shape Analysis for Fall Detection. The Eighth International Workshop on Visual Surveillance - VS2008, Oct 2008, Marseille, France. 2008.

[3] Michael Marschollek, Anja Rehwald, Klaus-Hendrik Wolf, Matthias Gietzelt, Gerhard Nemitz, Hubertus Meyer zu Schwabedissen and Mareike Schulze. Sensors vs. experts - A performance comparison of sensor-based fall risk assessment vs. conventional assessment in a sample of geriatric patients. BMC Medical Informatics and Decision Making 2011.

[4] Gabriele Rescio, Alessandro Leone, and Pietro Siciliano. Supervised Expert System for Wearable MEMS Accelerometer-Based Fall Detector. Hindawi Publishing Corporation 2013.

[5] Xiaoqun Yu, Hai Qiu and Shuping Xiong. A Novel Hybrid Deep Neural Network to Predict Pre-impact Fall for Older People Based on Wearable Inertial Sensors. Department of Industrial and Systems Engineering, Korea Advanced Institute of Science and Technology, Daejeon, South Korea, CETHIK Group Corporation Research Institute, Hangzhou, China.2020.

[6] https://bitbucket.org/unipv_cvmlab/sisfalltemporallyannotated/src/master/

[7] http://sistemic.udea.edu.co/en/research/projects/english-falls/

Kharazmi University
Faculty of Mathematical Sciences and Computer
Department of Computer Sciences

Final Report of BSc Project

Thesis Title:

Intelligent Fall Detection of Elderly People

By:

Seyed Mohammad Fattahian

Supervisor:

Dr. Soltanian

3992
June 2021