توصیه‌ی بازبین کد در گیت‌هاب

نوشین ذاکرزاده[۱]، عماد دیلم صالحی[۲]

[۱] دانشجوی کارشناسی ارشد، دانشکده‌ی مهندسی کامپیوتر، دانشگاه صنعتی شریف، تهران
zakerzadeh@ce.sharif.edu

[۲] دانشجوی کارشناسی ارشد، دانشکده‌ی مهندسی کامپیوتر، دانشگاه صنعتی شریف، تهران
emad44salehi@yahoo.com

چکیده

در توسعه‌ی نرم‌افزار مبتنی بر کشش، هر شخص می‌تواند با ارسال یک درخواست کششی به تیم توسعه، در پروژه مشارکت داشته باشد. این درخواست‌ها، نقش مهمی در گیت‌هاب به‌عنوان یک سکوی کدنویسی اجتماعی، ایفا می‌کنند. پس از دریافت درخواست‌ها، یک یا چند تن از اعضای تیم توسعه‌ی پروژه، تغییرات پیشنهاد شده در درخواست را بررسی کرده و در رابطه با ادغام و یا عدم ادغام درخواست با مخزن پروژه، تصمیم‌گیری می‌کنند. تعداد این درخواست‌های کششی برای پروژه‌های بزرگ و پرطرفدار بسیار زیاد است؛ همین مسأله سبب پاسخ دیرهنگام به درخواست‌ها و یا نادیده گرفتن موارد سودمند می‌شود. بنابراین یافتن بازبین کد برای بررسی درخواست‌های کششی، مسأله‌ای مهم در این زمینه به‌شمار می‌رود. محتوای این پژوهش با استفاده از ترکیب دو روش مدل‌سازی موضوع و تحلیل شبکه‌های اجتماعی و همچنین با بهره‌گیری از معنای متنی درخواست‌های کششی گذشته، سعی براین دارد تا بتواند برای درخواست‌های کششی باز آینده، بهترین بازبین‌ها را پیشنهاد دهد. در این پژوهش دو روش، مبتنی بر دو الگوریتم متفاوت مدل‌سازی موضوع (LDA و NMF) پیاده‌سازی شده‌اند که طبق مقایسه بر اساس معیارهای مختلف ارزیابی، روش NMF هم از لحاظ میانگین رتبه متقابل و هم از جنبه میانگین دقت متوسط از روش LDA بهتر عمل کرده است. همچنین روش‌های پیشنهادی این پژوهش با روش‌های دیگر مقالات نیز مقایسه شده است. نتایج نشان می‌دهند، روش‌های دیگر عملکرد بهتری دارند زیرا این روش‌ها بیش از ۱۰ ویژگی مختلف را برای رتبه‌بندی بازبین‌ها درنظر گرفته‌اند.

کلمات کلیدی

توصیه‌ی بازبین، درخواست کششی، گیت‌هاب، گراف، رتبه‌بندی

۱- مقدمه

یکی از محبوب‌ترین مدل‌های مشارکت در توسعه‌ی نرم‌افزار توزیع‌شده[۱]، مدل توسعه‌ی مبتنی بر کشش[۲] است. مشارکت‌کنندگان خارجی بدون نیاز به دسترسی نوشتن در مخزن[۳] اصلی پروژه‌ها، می‌توانند تغییراتی را در یک پروژه‌ی نرم‌افزاری، پیشنهاد دهند. آن‌ها می‌توانند یک کپی از مخزن اصلی ایجاد و به‌طور مستقل روی آن کار کنند. در نهایت، با ارسال یک درخواست کششی برای تیم اصلی توسعه‌دهنده‌ی نرم‌افزار، درخواست ادغام کد پیشنهادی با پروژه را مطرح کنند. در حال حاضر، مدل مبتنی بر کشش، به شکل گسترده‌ای در پروژه‌های منبع باز[۴] موجود در سکوهای کدنویسی اجتماعی، نظیر گیت‌هاب[۵]، مورد استفاده قرار می‌گیرد.

توسعه‌دهندگان خارجی، تنها دسترسی خواندن مخازن عمومی را دارند. به همین خاطر، برای مشارکت در پروژه، افراد با انشعاب مخزن، به نسخه‌ای محلّی از پروژه دست می‌یابند که توانایی تغییر آن را دارند. پس از ایجاد تغییرات مدّنظر، اشخاص می‌توانند با ارسال درخواستی کششی به تیم اصلی توسعه‌دهنده، تقاضای اعمال تغییرات در کد پروژه را داشته باشند. بعد از دریافت درخواست‌ها توسط تیم توسعه‌دهنده، آن‌ها سبک برنامه‌نویسی، کیفیّت

کد و راهکار پیشنهادی را بررسی کرده و در نهایت درباره‌ی ادغام تغییرات با کد اصلی، تصمیم‌گیری می‌کنند.

فرآیند بررسی کد با انتساب بازبین شروع می‌شود. هنگامی‌که درخواست بازبینی جدیدی برای تغییر کد منبع به یک پروژه نرم‌افزاری ارسال می‌شود، باید بازبینان مناسب برای بررسی تغییرات کد، انتخاب شوند. بازبینان منتخب تغییر کد منبع را می‌خوانند و اگر تغییر قابل قبول باشند، آن را تأیید می‌کنند. بنابر صلاح‌دید، ممکن است بازبین‌ها به‌منظور بهبود کیفیت تغییر، نتایج بررسی را برای درخواست‌دهنده، ارسال کنند. درخواست‌دهنده با توجه به نتایج بازبینی، کد خود را تغییر می‌دهد و مجدداً درخواست را ارسال می‌کند. این فرآیند تا زمانی که کیفیّت کد ارسالی رضایت‌بخش باشد، تکرار می‌شود. در نهایت، فردی از اعضای تیم توسعه‌دهنده‌ی پروژه، درخواست‌های تأیید شده را در شاخه‌ی اصلی[۶]، ادغام می‌کند و درخواست‌ها را می‌بندد. به‌علاوه، درخواست‌های ارسال شده‌ای که از نظر بازبین مردود اعلام شوند نیز، بسته می‌شوند.

در حال حاضر، مدیریت درخواست‌های کششی، به‌عنوان مهم‌ترین فعالیت پروژه‌ها در گیت‌هاب شناخته می‌شود؛ چرا که برای یک پروژه در مقیاس بزرگ، هزاران درخواست کششی ارسال می‌شود و یافتن عضوی از تیم توسعه‌دهنده با دانش زمینه‌ای مناسب برای بررسی درخواست‌های کشش، مسأله‌ای مهم و زمان‌بر است. بنابراین، یک چالش کلیدی در توسعه‌ی نرم‌افزار مبتنی بر کشش، یافتن بازبین کد مناسب است.

اغلب اوقات، ارسال‌کننده‌ی درخواست کشش، تمایل دارد شخص خاصی را به‌عنوان بازبین برچسب‌گذاری کند. گیت‌هاب این قابلیت را در اختیار افراد قرار می‌دهد. با قرار دادن نام شخص مورد نظر کنار علامت@، اعلانی برای وی فرستاده می‌شود. در کارهای پیشین بررسی و اثبات شده است که این قابلیّت نیز نمی‌تواند پاسخگوی چالش پیش‌رو باشد. همین مسأله، انگیزه‌ای برای انجام پژوهش‌هایی با هدف خودکارسازی فرآیند توصیه و تخصیص بازبین، شد.

به‌طور کلی پژوهش‌های صورت گرفته در زمینه‌ی توصیه‌ی بازبین در گیت‌هاب، به دو دسته‌ی توصیه‌ی بازبین مبتنی بر تحلیل محتوای درخواست‌ها و توصیه‌ی بازبین مبتنی بر گراف قابل تفکیک هستند.

۱-۱- توصیه‌ی مبتنی بر محتوا

بخش قابل توجّهی از پژوهش‌های صورت گرفته، به بررسی محتوای درخواست‌ها می‌پردازند. از آن‌جایی که درخواست‌های کششی دارای ویژگی‌های غنی زیادی مانند تعداد خطوط تغییر یافته، نام و مسیر فایل و عنوان است؛ این روش‌ها به عملکرد قابل قبولی دست‌یافته‌اند.

تانگ‌تَنُم و همکاران [۴]، یک رویکرد توصیه‌ی بازبین مبتنی بر مکان فایل پیشنهاد داده‌اند. این راهکار، براساس شباهت مسیر فایل‌ها، بازبین‌های مرتبط را پیشنهاد می‌کند. ایده‌ی پیشنهادی پژوهش این است که اگر فایل‌های تغییریافته در یک درخواست کشش، مشابه مسیرفایل‌هایی باشد که توسط یکی از اعضای تیم توسعه، اصلاح و یا بررسی شده؛ این برنامه‌نویس احتمالاً دانش کافی برای بررسی درخواست را دارد. بنابراین ابتدا میزان تشابه مسیر فایل یک درخواست کشش جدید با مسیر فایل درخواست‌های کشش حل شده، محاسبه می‌شود. در گام بعدی، این عدد در قلب امتیاز، به توسعه‌دهندگانی که در درخواست کشش مربوطه نظر داده‌اند، تخصیص داده می‌شود. در نهایت برحسب امتیازهای دریافتی، می‌توان به برترین بازبین‌ها دست‌یافت.

زنجانی و همکاران [۸]، رویکرد دیگری بر اساس مکان فایل، پیشنهاد کرده‌اند. این پژوهش، تعداد دفعات بازبینی فایل‌ها توسط توسعه‌دهندگان را به‌عنوان یک عامل اساسی در انتخاب آن‌ها برای بازبینی درخواست‌ها، درنظر گرفته است.

در پژوهش [۷]، نویسندگان پیشنهاد کرده‌اند، به مسأله‌ی توصیه‌ی بازبین، مانند یک مسأله‌ی رتبه‌بندی نگاه شود. به این ترتیب، بازبین‌های کاندید بر اساس رابطه با یک درخواست کششی معیّن، رتبه‌بندی می‌شوند. در این مقاله، یک مدل رتبه‌بندی بر اساس ۱۴ ویژگی مختلف طراحی و با استفاده از درخواست‌های کششی حل‌شده، آموزش داده شده است. مسیر فایل، تشابه عنوان درخواست‌ها، میزان فعّالیّت توسعه‌دهنده‌ی پروژه و رابطه‌ی اجتماعی او با درخواست‌دهنده، برخی از ویژگی‌های درنظر گرفته شده هستند.

در پژوهش [۲]، کیم و همکاران، راهکار توصیه‌ی بازبین مبتنی بر الگوریتم LDA را پیشنهاد کرده‌اند. درخواست کشش شامل تغییرات کد منبع و توضیحات مفصل آن است. توسعه‌دهندگان معمولاً تمایل دارند تغییراتی را بررسی کنند که با تخصص آن‌ها مرتبط است. این پژوهش پیشنهاد می‌کند تا به کمک درخواست‌های کششی حل شده، تخصص توسعه‌دهندگان را از تغییرات کد منبع، استخراج کنیم. برای این کار می‌بایست با استفاده از الگوریتم LDA، k موضوع را از تغییرات کد منبع استخراج کنیم. در گام بعدی، میزان تخصص هر توسعه‌دهنده در هریک از این k موضوع را به‌دست‌آورده و برحسب موضوع درخواست کششی جدید، آن را به n بازبین برتر پیشنهاد کنیم.

۱-۲- توصیه‌ی مبتنی بر گراف

رویکردهای مبتنی بر گراف در مقایسه با رویکردهای مبتنی بر محتوا، عملکرد مشابهی را ارائه می‌دهند.

یو و همکاران [۶]، پیشنهاد طبقه‌بندی درخواست‌های کششی را مطرح کرده‌اند. در این پژوهش یک شبکه‌ی مبتنی بر نظرات[۷] را نیز پیشنهاد کرده‌اند که رابطه‌ی اجتماعی میان کاربران را در قالب یک گراف جهت‌دار و وزن‌دار، نمایش می‌دهد. اگر کاربر i حداقل یک درخواست کششی حل شده از کاربر j را بازبینی کرده باشد، یالی میان کاربران ایجاد می‌شود. وزن این یال‌ها برحسب تعداد، توزیع و زمان نظرات مشخص می‌شود. برای یک درخواست کششی جدید، با استفاده از این شبکه، درخواست به توسعه‌دهنده‌ای از تیم اصلی، ارجاع داده می‌شود که اغلب به درخواست‌های آن درخواست‌کننده، پاسخ داده است. در پژوهش بعدی [۵]، نویسندگان با بهره‌برداری از محتوای درخواست‌ها و مکان فایل، سعی در بهبود عملکرد پیشنهاد خود داشته‌اند. در این پژوهش، به کمک شباهت کسینوسی[۸] میان یک درخواست کشش جدید و درخواست‌های کششی حل شده، امتیازاتی به بازبین‌های کاندید اعطا می‌شود. کاندیدهایی با امتیاز بالاتر، به‌عنوان بازبین‌های نهایی پیشنهاد می‌شوند.

۲ – معرفی رویکردهای پیشنهادی

در پروژه‌های مختلف گیت‌هاب، عملکرد توصیه‌دهنده‌های مبتنی بر گراف می‌تواند متفاوت باشد و این مسأله به ناپایداری رویکردهای مبتنی بر گراف برای حل مشکلات توصیه‌ی بازبین اشاره می‌کند [۳]. به همین خاطر در این پژوهش از راهکارهای مبتنی بر محتوا استفاده می‌شود.

در ادبیات موضوع، ویژگی‌های شباهت محتوا شامل عناوین درخواست‌های کششی و توضیحات است که با روش‌های مختلفی مانند مدل‌های TF-IDF، بردار می‌شوند.

مطالعات قبلی [۲]، از LDA برای استخراج توزیع موضوع از داده‌های متنی استفاده کردند. به‌همین خاطر در این رویکرد نیز از این الگوریتم بهره گرفته می‌شود. علاوه بر این برای روش دوم پیشنهادی، از الگوریتم NMF برای استخراج توزیع موضوع از داده‌های متنی استفاده می‌کنیم.

پژوهش [۷] ایده‌ی تبدیل مسأله‌ی توصیه‌ی بازبین به یک مسأله‌ی رتبه‌بندی را مطرح می‌کند. با اقتباس از آن، در این پژوهش نیز از رتبه‌بندی بازبینان با استفاده از الگوریتم page rank استفاده می‌شود. بازبین‌هایی با رتبه‌ی بهتر، شانس بیشتری برای قرار گرفتن در تیم بازبینی دارند.

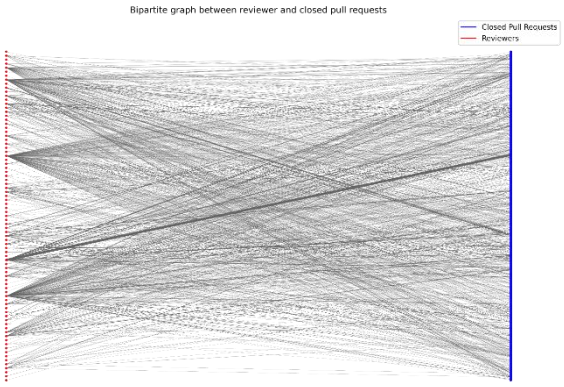
از طرفی، یو و همکاران [۵]، ایده‌ی مرتبط کردن بازبین‌های کاندید با درخواست‌های کششی حل‌شده، برحسب تعداد نظرات بازبین‌ها را مطرح کرده‌اند. در این پژوهش نیز از این مورد نیز برای ساخت گراف دو بخشی بین بازبین‌ها و درخواست‌های کششی بسته استفاده شده است. وزن یال‌ها در این گراف، تعداد نظرات بازبین‌ها در یک درخواست کششی است.

همچنین برای تبدیل گراف دو بخشی وزن دار بین بازبینان و درخواست‌های کششی بسته که پیش‌تر گفته شد به گرافی وزن دار که فقط بازبینان در آن حضور دارند، از پروسه‌ای استفاده می‌کنیم که در بخش پیاده سازی رویکرد پیشنهادی، به تفضیل به آن خواهیم پرداخت. وزن یال‌های این گراف نشان دهنده اولویت تخصصی است. در نهایت با استفاده از الگوریتم رتبه‌بندی page rank، برترین بازبین‌ها را برمی‌گزینیم.

۳ – پیاده‌سازی رویکرد پیشنهادی

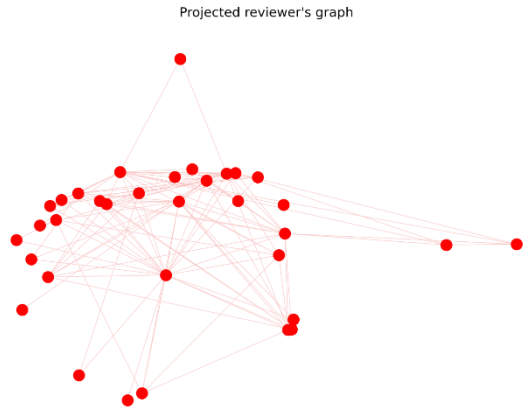
۳-۱- اصول کلی پیاده‌سازی

منطق کلی پیاده‌سازی‌ها به اینصورت است که ابتدا برای یک مخزن کد مشخص که کاربر تعریف می‌کند، با استفاده از گیت‌هاب API، تمام درخواست‌های کششی بسته آن را استخراج می‌کنیم. سپس برای هر کدام از این درخواست‌های کششی بسته، تمامی نظرات کاربران را استخراج کرده و سپس بررسی می‌کنیم (درخواست‌های کششی بسته‌ای که هیچ نظری برای آنها ثبت نشده است، حذف می‌شوند) در هر کدام نظرات چه کاربرهایی به عنوان بازبین ظاهر شده‌اند (کسی که درخواست کششی را ایجاد کرده یا ربات بوده است را حذف می‌کنیم). سپس با استفاده از این اطلاعات که در هر درخواست کششی بسته چه کاربرهایی به عنوان بازبین ظاهر شده‌اند، یک گراف دو بخشی برای شهودی سازی ایجاد می‌کنیم (مانند شکل ۱) در یک سمت درخواست‌های کششی بسته و در سمت دیگر بازبین‌ها هستند و بین این دو بخش یال وزن دار وجود دارد که وزن یال به این معنی است که یک بازبین چند بار در یک درخواست کششی بسته نظر داده است (وزن یال‌ها در گراف برای شهودی سازی بهتر، به صورت شدت رنگ نشان داده شده است). سپس برای همه درخواست‌های کششی بسته یک پیکره[۹] ایجاد می‌شود.



شکل (۱) : گراف دو بخشی بین بازبین‌ها و درخواست‌های کششی بسته برای یک مخزن کد

در ادامه درخواست‌های کششی باز نیز به صورت کامل جمع آوری شدند و برای اینکه مراحل انجام کار به صورت واضح توضیح داده شود و برای خواننده کد گیج کننده نباشد، ابتدا تمام مراحلی که در ادامه توضیح داده می‌شود به صورت گام به گام بر روی یکی از درخواست‌های کششی باز (درخواست‌های کششی بازی که بیش‌ترین تعداد نظرات برایش داده شده باشد) انجام شده است و سپس برای اینکه بفهمیم مقدار پارامترهای ارزیابی روش ما چیست این روش‌های گام به گام را برای همه درخواست‌های کششی باز اجرا می‌کنیم و سپس معیارهای ارزیابی خود را بدست می‌آوریم. با توجه به توضیح بالا، ابتدا یکی از درخواست‌های کششی باز را انتخاب میکنیم (درخواست‌های کششی بازی که بیش‌ترین تعداد نظرات برایش داده شده باشد) و برای آن یک پیکره ایجاد می‌کنیم. حال برای این درخواست کششی باز به تمامی بازبین‌های فعلی آن دسترسی داریم. سپس معیار شباهت کسینوسی را بین تمامی درخواست‌های کششی بسته و درخواست کششی بازی که پیش‌تر داشتیم، بدست می‌آوریم (این معیار با استفاده از مدل‌سازی موضوعی[۱۰] که توسط روش‌های TF-IDF و LDA/NMF بدست آمده است، محاسبه می‌شود). با استفاده از این معیار متوجه می‌شویم کدام درخواست‌های کششی بسته با درخواست کششی باز مورد نظر شباهت بالاتری دارند. در قسمت قبلی که قصد داشتیم معیار شباهت کسینوسی را محاسبه کنیم، جایی است که دو روش ارائه شده متفاوت می‌باشند. در یکی از روش‌ها از الگوریتم LDA و در دیگری از الگوریتم NMF برای مدل سازی موضوعی استفاده می‌کنیم. سپس با توجه به اینکه فهمیدیم کدام درخواست‌های کششی بسته شباهت بیش‌تری با درخواست کششی باز مورد نظر دارند، زیرگراف دو بخشی مربوط به آن درخواست‌های کششی بسته را از گراف دو بخشی کلی که در شکل ۱ مشاهده می‌شود، استخراج می‌کنیم. سپس از این زیرگراف دو بخشی، گراف مربوط به رابطه بازبین‌ها[۱۱] را استخراج[۱۲] می‌کنیم (شکل ۲). این گراف یک گراف وزن دار است که وزن بین هر دو راس، با استفاده از وزن موجود در گراف دو بخشی و عدد شباهتی که با استفاده از معیار کسینوسی بدست آمده است محاسبه می‌شود.



شکل (۲) : گراف وزن دار بین بازبین‌ها

در گام بعدی بر روی گراف بالا، الگوریتم page rank را اجرا می‌کنیم تا متوجه شویم کدام بازبین‌ها، معیار رنک بالاتری دارند. سپس آن بازبین‌ها به عنوان بازبین‌های مورد نظر برای درخواست‌های کششی باز توصیه می‌شوند. تمامی مراحل بالا که گام به گام توضیح داده شد، برای یک درخواست کششی باز انجام شد. در گام بعدی برای تمامی درخواست‌های کششی باز موجود در مخزن کد، گام‌های بالا طی می‌شود تا برای هر درخواست کششی باز، یکسری بازبین توصیه شود. سپس با توجه به این بازبین‌های توصیه شده و بازبین‌های اصلی برای هر درخواست کششی باز، اقدام به محاسبه معیارهای ارزیابی مختلفی از جمله: دقت[۱۳]، صحت[۱۴]، پوشش[۱۵]، معیار اف۱[۱۶]، میانگین رتبه متقابل[۱۷] و میانگین دقت متوسط[۱۸] می‌شود.

مراحل توضیح داده شده در بالا گام‌ها و ایده‌های کلی برای پیاده‌سازی را بیان می‌کند. در ادامه به جزئیات پیاده‌سازی در هر گام می‌پردازیم.

همچنین ذکر این نکته ضروری است که برای این پروژه در ابتدا با استفاده از نرم افزار خزشگر پارس‌هاب[۱۹] مخازن کد موجود در گیت‌هاب استخراج و در یک فایل اسکل ذخیره شدند. سپس این مخازن با توجه به تعداد درخواست‌های کششی بسته و بازشان فیلتر شدند (مخازنی که تعداد درخواست‌های کششی بسته یا باز زیر ۵۰ داشتند حذف شدند به این دلیل که از یکطرف مدل سازی موضوعی مناسبی انجام نمی‌شد و از طرف دیگر در محاسبات مربوط به ارزیابی، بر روی تعداد کافی و مناسبی از درخواست‌های کششی میانگین گیری انجام نمی‌شد). سپس مخازن باقی مانده کاندیدهای خوبی برای اجرای الگوریتم پیشنهادی هستند.

۳-۲- جزئیات پیاده‌سازی

همانطور که در قسمت قبلی اشاره شد، دو روش پیاده‌سازی شده تنها در الگوریتم مدل‌سازی موضوع با هم تفاوت دارند و در نتیجه تمامی موارد دیگر پیاده‌سازی عینا شبیه هم هستند. در این پیاده‌سازی، برای راحتی خواننده کد، فایل جوپیتر نوت بوک[۲۰] را به ۱۵ گام تقسیم بندی کردیم که در ادامه به معرفی این ۱۵ گام و تشریح آنها می‌پردازیم.

در گام اول در ابتدای کد به عنوان ورودی چند پارامتر را می‌گیرد که این ورودی‌ها عبارتند از:

ACCESS_TOKEN: این پارامتر، توکنی[۲۱] را به عنوان ورودی می‌گیرد که با استفاده از آن بتواند از گیت‌هاب API استفاده کند و به داده مخازن کد و داده‌های مورد نیاز، دسترسی پیدا کند. برای ایجاد این توکن، باید ابتدا در سایت گیت‌هاب ثبت نام کرد و سپس در بخش تنظیمات کاربر در قسمت تنظیمات توسعه دهنده[۲۲]، می‌توان این توکن را ایجاد کرد.

REPO: این پارامتر، نام مخزن کد مورد نظر کاربر برای بررسی و ارائه بازبین برای درخواست‌های کششی بازش را نگه می‌دارد.

limit_pr: این پارامتر به این دلیل قرار داده شده است که تعداد درخواست‌های کششی بسته در مخازن کد بسیار زیاد است و هنگامی که بخواهیم از این درخواست‌های کششی استفاده کنیم و برای هر کدام تمام نظرات را استخراج کنیم و بر اساس آنها مدل‌سازی موضوع و تحلیل شبکه‌های اجتماعی[۲۳] انجام دهیم، زمان اجرای برنامه به شدت بالا می‌رود. در نتیجه این پارامتر تعریف شده است تا بتوان پس از استخراج تمام درخواست‌های کششی بسته برای هر مخزن کد، بر روی تعداد درخواست‌های کششی بسته به منظور مدل سازی کنترل داشت. این پارامتر به صورت پیشفرض بر روی مقدار ۵۰۰ تنظیم شده است.

similarity_threshold: این پارامتر مقدار آستانه برای انتخاب درصدی از درخواست‌های کششی بسته است که در نتایج محاسبه می‌شوند. این پارامتر در گام هشتم مقدار دهی شده است.

Num_of_Open_PR: این پارامتر بیان می‌کند که برای چه تعدادی از درخواست‌های کششی باز موجود در مخزن کد، بازبین توصیه کند. در حالت ایده آل این مقدار باید برابر کل درخواست‌های کششی باز باشد اما به دلیل زمان اجرای بسیار بالا، این برابر نصف درخواست‌های کششی باز قرار داده شده است. این پارامتر در گام سیزدهم مقدار دهی شده است. همچنین در این گام، تمامی درخواست‌های کششی بسته مربوط به مخزن مورد نظر جمع آوری شدند. سپس تعدادی از آن‌ها که کاربر توسط پارامتر *limit_pr* مشخص میکند، انتخاب می‌شوند و در پارامتر *closed_prs* ذخیره می‌شوند تا بعدا بر روی این درخواست‌های کششی بسته، مدلسازی و موضوعی و غیره انجام شود.

در گام‌های دوم و سوم با توجه به درخواست‌های کششی بسته‌ی جمع آوری شده در قسمت قبلی، در این قسمت برای هر درخواست کششی بسته، تمامی نظرات مرتبط با آن نیز جمع آوری شده. سپس، تمامی بازبین‌های فعلی مربوط به هر درخواست کششی بسته جمع آوری می‌شوند (کسی که درخواست کششی را ایجاد کرده یا ربات بوده است را حذف می‌کنیم). با توجه به این اطلاعات یک گراف دو بخشی بین درخواست‌های کششی بسته و بازبین‌های مرتبط با آن‌ها تشکیل می‌شود. در گام سوم رسم گراف دو بخشی را توسط کتابخانه networkx داریم. این گراف دو بخشی وزن دار است و وزن یال به این معنی است که یک بازبین چند بار در یک درخواست کششی بسته نظر داده است. همچنین رئوس قرمز در این گراف، بازبین‌ها و رئوس آبی درخواست‌های کششی بسته هستند. به علاوه در پایین گراف اطلاعات کلی گراف از جمله تعداد رئوس کلی گراف و یال‌ها و تعداد بازبین‌ها نمایش داده می‌شوند.

در گام چهارم ابتدا کتابخانه‌های مرتبط با پیش پردازش داده‌ها بارگذاری شدند و سپس تابعی که برای پیش پردازش بر روی درخواست‌های کششی انجام می‌شود، تعریف شده است. ورودی این تابع به صورت یک متن است (که در ادامه می‌بینیم که درخواست‌های کششی را به صورت پیکره در می‌آوریم و سپس به عنوان ورودی به این تابع می‌دهیم) که اعمالی از جمله حذف فاصله بین کلمات، حذف علائم نگارشی و لغات با طول کمتر از سه، حذف کاراکترهای خاص، ارقام، آدرس سایت‌ها و غیره و انجام اعمال ریشه‌یابی[۲۴]، حذف کلمات توقف[۲۵] و غیره به عنوان پیش پردازش بر روی آن انجام می‌شود. در ادامه این

گام، برای درخواست‌های کششی بسته جمع آوری شده ابتدا یک پیکره ساخته شده و سپس بر روی آن پیش پردازش صورت می‌گیرد.

در گام پنجم همانطور که پیش‌تر توضیح داده شد، ابتدا تمامی جزئیات پیاده‌سازی را ابتدا برای یک درخواست کششی باز و سپس کل پیاده‌سازی را بر روی یک دسته بزرگی از درخواست‌های کششی باز انجام می‌دهیم. در این گام ابتدا کلیه درخواست‌های کششی باز استخراج می‌شوند. سپس درخواست کششی بازی که بیش‌ترین تعداد نظرات را دارد از میان همه درخواست‌های کششی باز انتخاب می‌شود. دلیل اینکه درخواست کششی باز با بیش‌ترین تعداد نظرات انتخاب می‌شود این است که کاندید بهتری برای ارائه بازبین است زیرا هر چه تعداد نظرات کمتر باشد افراد کمتری مشارکت داشته‌اند و در نتیجه ارائه بازبین برای آن جالب نخواهد بود. در ادامه برای این درخواست کششی باز نیز یک پیکره ساخته می‌شود و سپس پیش پردازش بر روی آن انجام می‌شود.

در گام ششم، تمامی نظرات موجود در درخواست کششی باز جمع آوری شده و سپس تمامی بازبین‌های فعلی موجود در این درخواست جمع آوری می‌شوند (کسی که درخواست کششی را ایجاد کرده یا ربات بوده است را حذف می‌کنیم). همچنین از میان بازبین‌های جمع آوری شده، کسانی که پیش‌تر در گراف دو بخشی ظاهر نشدند را حذف می‌کنیم زیرا درباره این بازبین‌ها نمی‌توانیم نظری بدهیم. سپس بازبین‌های باقی مانده نمایش داده می‌شوند.

در گام هفتم، تابع محاسبه شباهت کسینوسی تعریف شده است. دو روش پیاده‌سازی شده، در این گام با هم تفاوت دارند. ورودی این تابع سه مورد است: ۱) *closed_prs_meta*: این آرگومان درخواست‌های کششی بسته قبل از پیش پردازش است که به صورت لیستی از دیکشنری‌ها ذخیره شدند. برای هر درخواست کششی بسته یک دیکشنری وجود دارد که شناسه[۲۶]، موضوع[۲۷] ، بدنه[۲۸] و نظرات به عنوان کلیدهای این دیکشنری هستند. این آرگومان تابع در گام دوم بدست آمده است.

۲) *closed_prs_corpus*: این آرگومان درخواست‌های کششی بسته بعد از پیش پردازش است که به صورت یک دیکشنری است که کلید آن شماره آیدی درخواست کششی بسته است و مقادیر مربوط به هر کلید یک لیستی از کلمات پیکره است.

۳) *open_pr_corpus*: این آرگومان درخواست‌های کششی باز بعد از پیش پردازش است که به صورت یک لیستی از کلمات است که بعد از پیش پردازش بدست آمده‌اند.

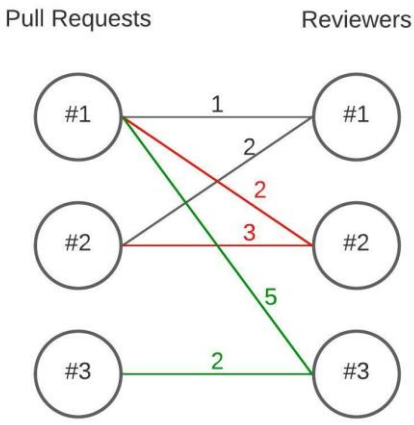
ابتدا یک متغیر با نام *corpus_data* ایجاد می‌کنیم که در آن تمامی درخواست‌های کششی بسته بعد از پیش پردازش را به صورت کلمه کلمه جدا شده اضافه می‌کنیم. سپس درخواست کششی باز بعد از پیش پردازش را نیز به صورت کلمه کلمه جدا شده اضافه می‌کنیم. سپس با استفاده از کتابخانه gensim، یک دیکشنری می‌سازیم که کلمات را به مقدار عددی متناظرشان نگاشت می‌کند. سپس با استفاده از filter_extremes، توکن‌ها را در دیکشنری بر اساس تکرارشان فیلتر می‌کنیم. این مرحله تمامی توکن‌هایی را که کمتر از ۱۵ بار یا بیش‌تر از ۰.۸ اندازه کل پیکره تکرار شده باشند حذف می‌کند.

در قدم بعدی به وسیله doc2bow کل پیکره‌ای را که در مراحل قبل بدست آوردیم، به وسیله دیکشنری که در مرحله قبل فیلتر کردیم به دسته ای از کلمات تبدیل می‌کنیم و نام آن را *bow_corpus* می‌گذاریم. سپس بر روی این متغیر TF-IDF اعمال می‌کنیم و در متغیر دیگری با نام *corpus_tfidf* ذخیره می‌کنیم. این قسمت جایی است که دو روش پیاده‌سازی شده از هم متمایز می‌شوند. در یک روش از الگوریتم LDA و در دیگری از الگوریتم NMF برای مدلسازی موضوعی استفاده شده است. در آخر نیز شباهت بین درخواست کششی باز و درخواست‌های کششی بسته با استفاده از تابع شباهت کسینوسی محاسبه می‌شود و در لیستی ذخیره می‌گردد و به عنوان خروجی تابع بازگردانده می‌شود.

در گام هشتم تابعی که در گام قبلی ایجاد شد را با توجه به آرگومان‌های تشریح شده تابع در گام قبلی فراخوانی می‌کنیم و سپس ۵ تا از شبیه‌ترین درخواست‌های کششی بسته به درخواست کششی باز را به عنوان خروجی نمایش می‌دهیم.

در گام‌های نهم و دهم با توجه به اینکه در گام قبلی متوجه شدیم کدام درخواست‌های کششی بسته با درخواست کششی باز انتخابی شباهت بیش‌تری دارند، در این گام با استفاده از این درخواست‌های کششی بسته مشابه، یک زیرگراف از گراف دو بخشی وزن دار که در گام دوم و سوم بدست آوردیم می‌سازیم. سپس این زیرگراف دو بخشی بدست آمده را به گرافی وزن دار تبدیل می‌کنیم که فقط در آن بازبین‌ها حضور داشته باشند (با استفاده از الگوریتم projection که در کتابخانه networkx وجود دارد). برای بدست آوردن وزن یال‌های موجود در گراف بازبین‌ها، همانطور که در بخش‌های ابتدایی اشاره شد، الگوریتم‌های مختلفی توسط نویسندگان متفاوت پیشنهاد شده است. در این پژوهش این وزن‌ها به وسیله: ۱) وزن یال‌ها در گراف دو بخشی و ۲) معیار شباهتی که در گام هشتم بدست آمده است محاسبه می‌شوند. تابع *custom_weight* که در ابتدای این بخش تعریف شده است برای محاسبه وزن یال بین رئوس موجود در گراف بازبین‌ها ایجاد شده است. این تابع به این ترتیب کار می‌کند که برای هر بازبین i و j تابع، برای تمامی درخواست‌های کششی بسته مشترک این دو بازبین در گراف دو بخشی مانند k، وزن یال بین رئوس i و k با وزن یال بین رئوس j و k جمع شده و حاصل در معیار شباهت بین درخواست کششی بسته k و درخواست کششی باز ضرب می‌شود.

برای بهتر نشان دادن نحوه محاسبه تابع وزن برای یال‌های گراف بازبین‌ها به مثالی که در ادامه آمده است توجه کنید. فرض کنید بعد از پیدا کردن زیرگراف دو بخشی با توجه به معیار شباهت کسینوسی، گراف موجود در شکل ۳ بدست آمده است:



شکل (۳) : زیر گراف وزن دار دو بخشی بین درخواست‌های کششی بسته و بازبین‌ها

همچنین فرض کنید که معیار شباهت بین درخواست‌های کششی بسته ذکر شده در شکل بالا و درخواست کششی باز مورد نظر به صورت جدول ۱ باشد:

جدول (۱) : معیار شباهت میان درخواست‌های کششی بسته و درخواست کششی باز

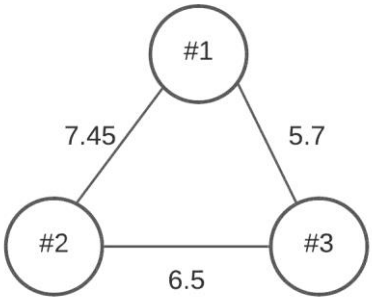
Closed PRs	Open PR	Cosine Similarity
# PR 1	# OPR 1	0.95
# PR 2	# OPR 1	0.92
# PR 3	# OPR 1	0.76

به این ترتیب اگر بخواهیم با استفاده از اطلاعات بالا یال وزن دار میان بازبین‌های ۱ و ۲ را در گراف بازبین‌ها بازسازی کنیم، داریم:

وزن یال میان بازبین‌های ۱و۲= (۱ + ۲) × ۰.۹۵ + (۲ + ۳) × ۰.۹۲ = ۷.۴۵

دقیقا عین محاسبات بالا را برای محاسبه وزن یال میان بازبین‌های ۱-۳ و ۲-۳ انجام می‌دهیم و به گراف شکل ۴ می‌رسیم.

همچنین در گام دهم، گراف بین بازبین‌ها و اطلاعات کلی از گراف مانند تعداد رئوس، یال‌ها و غیره نشان داده شده است.



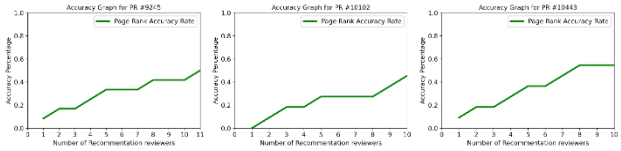
شکل (٤) : گراف وزن دار ساخته شده بین بازبین‌ها

در گام یازدهم با توجه به اینکه گراف بازبین‌ها را از گام قبلی داریم، الگوریتم pagerank را با استفاده از کتابخانه networkx بر روی این گراف پیاده‌سازی می‌کنیم تا بازبین‌ها را رنک بندی کند. نکته بسیار مهم این است که تعداد بازبین‌های پیشنهاد شده نهایی برابر تعداد بازبین‌های فعلی درخواست کششی باز درنظر گرفته شده‌اند. علت این امر این است که اگر تعداد بیش‌تری بازبین پیشنهاد بدهیم عایده‌ای و فقط دقت را پایین می‌آوریم. به همین دلیل در ابتدای این بخش متغیر *limit_recomm* را تعریف کردیم و مقدار آن را برابر تعداد بازبین‌های فعلی درخواست کششی باز قرار داده ایم. در انتها هم بازبین‌های فعلی مربوط به درخواست کششی باز و بازبین‌های پیشنهادی برای آن چاپ شده‌اند.

در گام دوازدهم، برخی از معیارهای ارزیابی از جمله دقت، صحت، پوشش، میانگین رتبه متقابل و معیار اف۱ نشان داده می‌شوند.

تا اینجا برای یک درخواست کششی باز تمامی الگوریتم‌ها و مراحل را جزء به جزء اجرا کردیم تا جزئیات برای خواننده واضح باشد. در گام سیزدهم تمامی الگوریتم‌ها و مراحل بالا را برای تعداد دلخواهی (مقدار پیشفرض = نصف درخواست‌های کششی باز) از درخواست‌های کششی باز انجام می‌دهیم. سپس تمامی معیارهای ارزیابی را برای تمامی این درخواست‌های کششی باز بدست می‌آوریم. در این گام به اینصورت عمل می‌کنیم که ابتدا مشخص می‌کنیم چه تعداد از درخواست‌های کششی باز را انتخاب می‌کنیم و مراحل بالا را بر روی آنها اجرا می‌کنیم. این تعداد در متغییر *Num_of_Open_PR* مشخص می‌شود که به صورت پیشفرض برابر نصف درخواست‌های کششی باز مقدار دهی شده است. دلیل اینکه برابر تمام درخواست‌های کششی باز قرار داده نشده است جلوگیری از طولانی شدن اجرای برنامه است. سپس این تعداد مشخص شده از درخواست‌های کششی باز را بر اساس تعداد نظرات موجود در آنها مرتب می‌کنیم، سپس به ترتیب بر روی این درخواست‌ها گام‌های ۵ تا ۱۲ اجرا می‌شود و تمامی معیارهای ارزیابی و نتایج برای تمامی درخواست‌های کششی باز انتخاب شده محاسبه و ذخیره می‌شوند. نکته مهمی که باید اشاره شود (پیش‌تر هم اشاره شد) این است که برای هر درخواست کششی باز، تعداد بازبین‌های پیشنهادی به تعداد بازبین‌های فعلی است.

در گام چهاردهم، برای سه تا از درخواست‌های کششی بازی (بیش‌تر از این تعداد فضا شلوغ می‌شد) که دقت محاسبه شده برای آنها صفر نیست (چون برای برخی از درخواست‌های کششی باز، ممکن است بازبین‌های پیشنهادی هیچکدام از بازبین‌های فعلی نباشند و دقت صفر شود)، به عنوان نمونه انتخاب می‌شوند و برای آنها نمودار دقت (محور عمودی) بر اساس تعداد بازبین پیشنهادی (محور افقی) رسم می‌شود. نکته مهم این است که حداکثر مقدار محور افقی برای هر درخواست کششی باز می‌تواند متفاوت باشد چون همانطور که گفته شد، برای هر درخواست کششی باز، تعداد بازبین‌های پیشنهادی حداکثر برابر تعداد بازبین‌های فعلی آن درخواست است. این نمودارها بیان می‌کنند که اگر تعداد بازبین‌های پیشنهادی را افزایش دهیم آیا دقت هم افزایش پیدا می‌کند یا ثابت می‌ماند. یکی از این نمودارها که برای پروژه بیت کوین است به عنوان نمونه در شکل ۵ آورده شده است.



شکل (٥) : نمودار دقت بر حسب تعداد بازبین‌های پیشنهادی برای ۳ درخواست کششی باز

در گام پانزدهم، معیارهای ارزیابی برای کلیه درخواست‌های کششی باز محاسبه می‌شوند.

٤ – ارزیابی و تحلیل رویکرد پیشنهادی

در این قسمت به بررسی نتایج روش‌های پیشنهادی خود و مقایسه آن‌ها با روش‌های نوین دیگری که در مقالات مهم مورد بررسی قرار گرفته است می‌پردازیم. در ادامه، روش‌های پیاده‌سازی شده در این پژوهش را LDA Method و NMF Method می‌نامیم.

همانطور که در جداولی که در ادامه آمده است مشاهده می‌شود، در جدول ۲، روش‌های ارائه شده در این پژوهش را با روش ارائه شده در [۱] مقایسه می‌کنیم. این روش از ۱۱ ویژگی برای اندازه‌گیری ارتباطات اجتماعی، موقعیت فایل‌ها و فعالیت توسعه دهندگان استفاده می‌کند. سپس توسط این ۱۱ ویژگی، درخواست‌های کششی را به توسعه دهندگان مختلف معرفی می‌کند. در جدول

۳، روش‌های ارائه شده در این پژوهش را با روش ارائه شده در [۷] مقایسه می‌کنیم. این روش، یک سیستم رتبه‌بندی بین بازبینان یک درخواست کششی ارائه می‌کند. سیستم رتبه‌بندی ارائه شده در این مقاله، از ۱۴ ویژگی استفاده می‌کند. دلیل عدم ادغام این دو جدول، شلوغ شدن فضا و ناخوانا شدن اعداد موجود در جداول بوده است.

به دلیل محدودیت‌های گیت‌هاب API، روش‌های پیاده‌سازی شده تنها بر روی ۵ مخزن بزرگ و پر بازدید گیت‌هاب اجرا گردید که نتایج آن‌ها در جداول ۲ و ۳ آورده شده است. دلیل اصلی که این ۵ مخزن مورد بررسی قرار گرفتند این بود که بتوانیم نتایج روش‌های پیاده‌سلزی شده را با روش‌های مهم ارائه شده در مقالات دیگر مقایسه کنیم. نتایج حاصل از روش‌های CoreDevRec و learning-to-rank که در جداول ۲ و ۳ مورد استفاده قرار گرفته‌اند، از [۷] آورده شده است.

با مقایسه روش‌های LDA Method و NMF Method (روش‌های پیاده‌سازی شده در این پژوهش) متوجه می‌شویم که از لحاظ دقت بازبین‌های پیشنهادی با هم تفاوتی ندارند. اما نکته بسیار مهم و تفاوت عمده این دو روش، تفاوت در معیارهای میانگین دقت متوسط و میانگین رتبه متقابل است. تفاوت بین این دو معیار نشان می‌دهد که روش NMF Method در اکثر مخازن (بجز برخی مانند pandas) اندکی بهتر از روش LDA Method در هر دو معیار میانگین دقت متوسط و میانگین رتبه متقابل عمل می‌کند.

بهتر عمل کردن روش NMF Method به صورت کلی در معیار میانگین رتبه متقابل به این معناست که این روش بازبین‌ها را در مکان‌ها (رتبه‌های) بهتر و واقعی‌تری (نسبت به موقعیت بازبین‌های فعلی هر درخواست کششی) نسبت به روش LDA Method پیشنهاد می‌دهد. این معیار بسیار مهم است و نباید نادیده گرفته شود چون به عنوان مثال اگر در یک درخواست کششی فقط نیاز داشته باشیم تا ۳ بازبین پیشنهاد دهیم، روشی که میانگین رتبه متقابل بالاتری دارد اولویت دارد. بهتر عمل کردن روش NMF Method به صورت کلی در معیار میانگین دقت متوسط به این معناست که این روش در کل مدل بهتری برای ارائه بازبین برای درخواست‌های کششی باز نسبت به روش LDA Method است.

تا اینجا به این نتیجه رسیدیم که روش NMF Method بهتر از LDA Method عمل می‌کند. حال طبق جدول ۲، واضح است که روش CoreDevRec نسبت به روش NMF Method با اختلاف در همه معیارهای ارزیابی برتر است. دلیل عمده این اتفاق این است که همانطور که پیشتر هم اشاره شد، این روش از ۱۱ ویژگی برای اندازه‌گیری ارتباطات اجتماعی، موقعیت فایل‌ها و فعالیت توسعه دهندگان استفاده می‌کند. درنتیجه، این روش به مراتب نسبت به روش‌های ارائه شده در این پژوهش که تنها از نظرات موجود در درخواست‌های کششی بسته برای رتبه‌بندی بازبین‌ها استفاده می‌کند، بهتر است. با این حال با توجه به جدول ۲، روش ارائه شده در این پژوهش هم خیلی بدتر روش CoreDevRec عمل نکرده است.

تا اینجا به این نتیجه رسیدیم که روش CoreDevRec بهتر از روش NMF Method و این روش هم بهتر از LDA Method عمل می‌کند. با مقایسه روش‌های Learning-to-Rank و CoreDevRec طبق جدول ۲ و ۳ به این نتیجه می‌رسیم که روش اولی به صورت کلی (بجز در مخزن node) میانگین دقت متوسط بالاتری نسبت به دیگر روش دارد. در نتیجه، به صورت کلی، دقت روش اولی بهتر از دومی است. از نظر معیار میانگین رتبه متقابل هم به وضوح روش اول، روش دوم را مغلوب می‌کند. در نتیجه روش اول به صورت کلی، رتبه‌بندی بهتری را برای بازبینان ارائه می‌دهد. همه این مقایسه‌ها نشان می‌دهد که روش Learning-to-Rank به مراتب بهتر از روش CoreDevRec عمل می‌کند.

برای خلاصه کردن مطالب بالا، جدول ۴ تهیه شده است که رتبه‌بندی ۴ روش گفته شده در این قسمت را بر اساس معیارهای ارزیابی نشان می‌دهد.

جدول (٤) : رتبه‌بندی چهار روش ارائه شده در این بخش

Rank	Approach
1	Learning-to-Rank
2	CoreDevRec
3	NMF Method
4	LDA Method

۵ – تهدیدات علیه اعتبار

موارد متعددی در این پیاده‌سازی وجود دارد که می‌توانند تهدیدی علیه اعتبارسنجی و معیارهای ارزیابی باشند. چند نمونه از این موارد عبارتند از:

۱) عدم استفاده از تمامی درخواست‌های کششی: در این پیاده‌سازی از ۵۰۰ درخواست کششی بسته برای مدلسازی و از نیمی از درخواست‌های کششی باز برای ارزیابی معیارها استفاده شده است. دلیل این امر آن است که استفاده از تمامی درخواست‌های کششی، نیاز به صرف زمان زیادی برای اجرا شدن برنامه دارد. در حالت ایده‌آل اگر از تمام درخواست‌های کششی استفاده شود، نتایج به واقعیت نزدیک‌تر می‌شوند. البته شایان ذکر است که در این پیاده‌سازی امکان این وجود دارد که تعداد درخواست‌های کششی بسته برای مدلسازی و تعداد درخواست‌های کششی باز برای ارزیابی معیارها به عنوان ورودی به برنامه داده شود. این مقادیر به ترتیب در پارامترهای *limit_pr* و *Num_of_Open_PR* ذخیره می‌شوند.

۲) بهبود پارامترهای استفاده شده در تابع تعیین وزن یال‌ها در گراف بازبین‌ها: در این پیاده‌سازی، همانطور که در قسمت نهم و دهم توضیح داده شد، برای محاسبه وزن یال‌ها در گراف بازبین‌ها، از تابع تعریف شده *custom_weight* استفاده شده است که یکی از پارامترهای استفاده شده در این تابع، مقدار شباهت کوسینوسی است که فرض ساده شده‌ای برای استفاده است. برای نتیجه‌گیری بهتر، می‌توان از پارامترهای بهبود داده شده بهتری استفاده کرد.

۳) تنها نظرات موجود در درخواست‌های کششی در نظر گرفته شده است: نظرات در درخواست‌های کششی به احتمال زیاد تمامی تعاملات بین کاربران را نشان نمی‌دهد. بحث از طریق کانال‌های دیگر مانند ایمیل یا گفتگوهای حضوری نیز می‌تواند تفاوت قابل توجهی ایجاد کند. این می‌تواند یکی از دلایلی باشد که روش‌های عمومی‌تر که معمولا به عنوان روش پایه در نظر گرفته می‌شوند، می‌توانند از برخی روش‌های گرافی بهتر عمل کنند.

۴) یافته‌های این مطالعه بر اساس پروژه‌های منبع باز در گیت‌هاب بوده است و مشخص نیست که آیا نتایج را می‌توان به پروژه‌های تجاری یا پروژه‌های منبع باز در سایر پلتفرم‌های کدگذاری اجتماعی مانند بیت‌باکت[۲۹] تعمیم داد یا خیر.

جدول (۲) : مقایسه روش‌های ارائه شده توسط خودمان با روش CoreDevRec

	LDA Method					NMF Method					CoreDevRec				
	Accuracy					Accuracy					Accuracy				
Repo	Top 1	Top 3	Top 5	MAP	MRR	Top 1	Top 3	Top 5	MAP	MRR	Top 1	Top 3	Top 5	MAP	MRR
bitcoin/bitcoin	25.5	32.3	50	0.325	0.271	25.5	32.3	50	0.329	0.272	37.2	63.6	73.8	0.401	0.540
pandas-dev/pandas	35.6	39	60	0.421	0.433	35.6	39	60	0.427	0.431	67.3	88.1	94.9	0.703	0.794
scikit-learn/scikit-learn	19.6	29	46.4	0.232	0.382	19.6	29	46.4	0.219	0.389	65.9	83.6	88.1	0.688	0.771
electron/electron	20.8	35	37.5	0.311	0.314	20.8	35	37.5	0.311	0.314	46.1	70.4	83	0.581	0.601
nodejs/node	21.9	43.5	58.3	0.337	0.513	21.9	43.5	58.3	0.345	0.542	41.1	67.6	72.4	0.529	0.572

جدول (۳) : مقایسه روش‌های ارائه شده توسط خودمان با روش Learning-to-Rank

	LDA Method					NMF Method					Learning-to-Rank				
	Accuracy					Accuracy					Accuracy				
Repo	Top 1	Top 3	Top 5	MAP	MRR	Top 1	Top 3	Top 5	MAP	MRR	Top 1	Top 3	Top 5	MAP	MRR
bitcoin/bitcoin	25.5	32.3	50	0.325	0.271	25.5	32.3	50	0.329	0.272	45.3	70.8	81.1	0.525	0.608
pandas-dev/pandas	35.6	39	60	0.421	0.433	35.6	39	60	0.427	0.431	73.8	91.7	96.7	0.788	0.834
scikit-learn/scikit-learn	19.6	29	46.4	0.232	0.382	19.6	29	46.4	0.219	0.389	73.5	93.3	97.1	0.771	0.837
electron/electron	20.8	35	37.5	0.311	0.314	20.8	35	37.5	0.311	0.314	55.3	76.6	86.4	0.611	0.687
nodejs/node	21.9	43.5	58.3	0.337	0.513	21.9	43.5	58.3	0.345	0.542	48.7	71.9	76.2	0.429	0.622

این پیاده‌سازی، دانش و تخصص بازبین‌ها را در درخواست‌های کششی بسته شده گذشته با ترکیب روش‌های مدلسازی موضوعی و تحلیل شبکه‌های اجتماعی ارزیابی می‌کند. در کارهای آینده درنظر داریم تا بتوانیم با استفاده از مدل‌های دیگر مانند ماشین‌های بردار پشتیبانی[۳۰]، یادگیری عمیق[۳۱]، شبکه عصبی گرافی[۳۲] و غیره، سیستم توصیه‌گر بازبین ارائه دهیم. همچنین در کارهای آینده به امید آن هستیم تا با رفع نواقص بالا، نتایج پارامترهای ارزیابی را به واقعیت نزدیک‌تر کنیم.

٦ – نتیجه‌گیری

مدل توسعه‌ی مبتنی بر کشش یک الگوی توسعه‌ی نرم افزار برای پشتیبانی از توسعه‌ی توزیع شده است. توسعه‌دهنده‌ای که می‌خواهد در یک پروژه مشارکت کند، می‌تواند با ارسال یک درخواست کششی، یکپارچه‌سازی تغییرات کد را درخواست کند. پس از دریافت درخواست کششی، تیم توسعه تغییرات را بررسی کرده و درباره‌ی پذیرش یا رد درخواست، تصمیم‌گیری می‌کنند. در این پژوهش، ما یک رویکرد یادگیری رتبه‌بندی را برای تقلید از فرآیند توصیه‌ی بازبین به‌کار گرفته شده توسط تیم توسعه معرفی کردیم. رویکرد ما از یک مدل رتبه‌بندی برای رتبه‌بندی همه‌ی بازبین‌های کاندید برای بازبینی یک درخواست کششی جدید استفاده می‌کند. در این پژوهش دو روش، مبتنی بر دو الگوریتم متفاوت مدل‌سازی موضوع (LDA و NMF) پیاده‌سازی شده‌اند. این دو روش پیاده‌سازی شده سپس توسط معیارهای ارزیابی متنوعی مانند دقت، صحت، میانگین رتبه متقابل و میانگین دقت متوسط با یکدیگر مقایسه شدند. همچنین در بخش ارزیابی، دو رویکرد دیگر که به نوبه خود پیشرو هستند، با روش‌های ارائه شده در این پژوهش مقایسه شدند. نتایج نشان داد، اگرچه این روش‌ها ممکن است با توجه به معیارهای ارزیابی بر روش‌های ارائه شده در این پژوهش غلبه کنند اما این روش‌ها بعضا بیش از ۱۰ پارامتر مختلف را برای رتبه‌بندی بازبین‌ها درنظر گرفته‌اند.

مراجع

[1] Jing Jiang, Jia-Huan He, and Xue-Yuan Chen, "*CoreDevRec: Automatic core member recommendation for contribution evaluation*,'' Journal of Computer Science Technology., vol. 30, no. 5, pp. 998–1016, Sep. 2015.

[2] Kim, Jungil, and Eunjoo Lee. *"Understanding review expertise of developers: A reviewer recommendation approach based on latent Dirichlet allocation."* Symmetry 10.4 (2018): 114.

[3] Luong, Toan, Ryan Silva, and Apollo Kaneko. *"Github Reviewer Recommendation using Graph."*

[4] Thongtanunam, Patanamon, et al. *"Who should review my code? a file location-based code-reviewer recommendation approach for modern code review."* 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER). IEEE, 2015.

[5] Yu, Yue, et al. *"Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment?."* Information and Software Technology 74 (2016): 204-218.

[6] Yu, Yue, et al. *"Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration."* 2014 21st Asia-Pacific Software Engineering Conference. Vol. 1. IEEE, 2014.

[7] Ye, Xin. *"Learning to rank reviewers for pull requests."* IEEE Access 7 (2019): 85382-85391.

[8] Zanjani, Motahareh Bahrami, Huzefa Kagdi, and Christian Bird. *"Automatically recommending peer reviewers in modern code review."* IEEE Transactions on Software Engineering 42.6 (2015): 530-543.

زیرنویس‌ها

Distributed [1]
Pull [2]
Repository [3]
Open source [4]
GitHub [5]
Branch [6]
Comments [7]
Cosine similarity [8]
Corpus [9]
Topic Modeling [10]
Reviewer only graph [11]
Projected graph [12]
Accuracy [13]
Precision [14]
Recall [15]
F1-score [16]
MRR [17]
MAP [18]
Parsehub [19]
Jupyter Notebook [20]
Token [21]
Developers Setting [22]
Social Network Analysis [23]
Lemmatization [24]
Stop words [25]
ID [26]
Title [27]
Body [28]
BitBucket [29]
Support Vector Machine [30]
Deep Learning [31]
Graph Neural Network [32]