The 9th International Conference on Web Research
ICWR2023
May 3th-4th, 2023, Tehran, Iran

Code Data Augmentation to Improve Language Model's Performance in Requirement to Code Traceability Link Recovery

Ali Majidzadeh, Mehrdad Ashtiani, Morteza Zakeri Nasrabadi

School of computer engineering, Iran University of Science and Technology, Tehran, Iran
masiha.majidzadeh@gmail.com, m_ashtiani@iust.ac.ir, morteza_zakeri@comp.iust.ac.ir

Abstract

Data augmentation is a method to efficiently use the existing data to train deep neural networks. Maintaining requirement traceability links helps to improve software quality and prevent defects by aiding software development management. To ease this maintenance, automatic link recovery techniques can be used. One of the recent techniques to do this is to use a language model. We propose three code data augmentation techniques to improve language models' performance in requirement to code traceability link recovery. These three techniques are rename variable, swap operands, and swap statements. These are general techniques that can be implemented for different programming languages, and have the capacity to generate a variety of outputs randomly, which can improve the generalization of the model. The results of the evaluations show that code data augmentation improves the language model's performance in recovering doc-method links that are similar to requirement-method links. Using code data augmentation, the precision is increased from 0.669 to 0.722, the recall is increased from 0.574 to 0.601, and the Wilcoxon test shows that the improvements are significant.

Keywords: Software Traceability, Requirements Traceability, Data Augmentation, Language Model.

the 9th International Conference on Web Research
ICWR2023
May 3th-4th, 2023, Tehran, Iran

افزایش داده‌های کد برای بهبود عملکرد مدل زبان در ترمیم پیوندهای ردپذیری نیازمندی به کد

علی مجیدزاده[۱]، مهرداد آشتیانی[۲]، مرتضی ذاکری نصرآبادی[۳]

[۱] کارشناسی ارشد، دانشکده کامپیوتر، دانشگاه علم و صنعت ایران، تهران،
masiha.majidzadeh@gmail.com

[۲] استادیار، دانشکده کامپیوتر، دانشگاه علم و صنعت ایران، تهران
m_ashtiani@iust.ac.ir

[۳] کاندید دکتری، دانشکده کامپیوتر، دانشگاه علم و صنعت ایران، تهران
morteza_zakeri@comp.iust.ac.ir

چکیده

افزایش داده روشی برای رفع نیاز داده و استفاده بیشتر از داده‌های موجود برای آموزش شبکه‌های عصبی عمیق است. نگه‌داری پیوندهای ردپذیری نیازمندی به مدیریت توسعه نرم‌افزار کمک کرده و باعث بهبود کیفیت نرم‌افزار می‌شود. برای کمک به نگه‌داری این پیوندها، می‌توان از روش‌های ترمیم خودکار پیوندها استفاده نمود. یکی از روش‌های اخیر ترمیم خودکار، استفاده از مدل زبان است. در این کار سه روش افزایش داده‌های کد برای بهبود مدل زبان در کاربرد ترمیم پیوندهای ردپذیری نیازمندی ارائه شده‌اند. این سه روش، تغییر نام متغیر، جابه‌جایی عملوندها و جابه‌جایی جملات هستند. این روش‌ها کلی بوده که برای بسیاری از زبان‌های برنامه‌نویسی قابل پیاده‌سازی هستند و همچنین قابلیت تولید حالات مختلف به صورت تصادفی دارند که می‌تواند قابلیت تعمیم مدل را بهبود بخشد. نتایج ارزیابی مدل روی داده‌های مستندات به تابع که مشابه داده‌های نیازمندی به تابع هستند نشان‌دهنده بهبود عملکرد مدل زبان با استفاده از افزایش داده‌های کد است. در این ارزیابی، با استفاده از افزایش داده‌های کد، دقت مدل از ۰.٦٦٩ به ۰.۷۲۲ و یادآوری آن از ۰.٥٧٤ به ۰.٦۰۱ رسیده است و طبق آزمایش ویلکوکسون، بهبود قابل توجهی داشته است.

کلمات کلیدی

ردپذیری نرم‌افزار، ردپذیری نیازمندی، افزایش داده، مدل زبان.

۱- مقدمه

ردپذیری نرم‌افزار [۱]توانایی ارتباط دادن فرآورده‌های[۲] ساخته شده در هنگام توسعه یک سیستم نرم‌افزاری برای توصیف سیستم از دیدها و سطوح انتزاع متفاوت است [1]. فرآورده نرم‌افزار قطعه اطلاعاتی است که در توسعه نرم‌افزار ساخته شده و نگه‌داری می‌شود. برای مثال نیازمندی‌ها، اسناد طراحی، کد منبع[۳]، مشخصات آزمون، راهنماها و گزارش‌های نقص [2]. پژوهش‌ها نشان می‌دهند که درجه کامل بودن ردپذیری نیازمندی‌ها ارتباط عکس با نرخ به وجود آمدن مشکلات نرم‌افزاری دارد. نرخ اشکالات، واحدی کمی برای توصیف کیفیت نرم‌افزار است [3]. همچنین هزینه ساختن و نگه‌داری پیوندهای ردپذیری زیاد بوده و معمولا پیوندهای ردپذیری دستی ناقص بوده و دقیق نیستند [4]. بنابراین برای کمک به مدیریت توسعه نرم‌افزار، و در نتیجه بهبود کیفیت نرم‌افزار و جلوگیری از نقص‌های نرم‌افزاری، به روش‌هایی برای ترمیم خودکار پیوندهای ردپذیری نیاز است.

هدف این پژوهش، بهبود عملکرد مدل زبان در ترمیم پیوندهای ردپذیری نیازمندی به کد با استفاده از افزایش داده‌های کد است. مدل زبان، مدلی است که قابلیت محاسبه احتمال درست بودن دنباله‌ای از کلمات را داشته باشد [5]. افزایش داده روشی برای استفاده بیشتر از داده‌های موجود است، که با تولید داده‌های مصنوعی و اضافه کردن آن‌ها به داده‌های آموزش انجام می‌شود [6]. برای بهبود عملکرد مدل زبان، سه روش کلی افزایش داده‌های کد یا به اختصار افزایش کد ارائه شده است. کلی بودن این سه روش به معنی توانایی استفاده در زبان‌های برنامه‌نویسی مختلف و همچنین تولنایی تولید موارد متفاوت به صورت تصادفی است. در ادامه در بخش ۲ به کارهای مرتبط اخیر پرداخته، در بخش ۳ روش پیشنهادی شرح داده شده، سپس در بخش ۴

شرایط ارزیابی و نتایج آن بررسی و در بخش ۵ جمع‌بندی و نتیجه‌گیری شده و درباره کارهای آینده بحث می‌شود.

۲- کارهای مرتبط

کارهای مرتبط در زمینه ترمیم خودکار پیوندهای ردپذیری عموما در دو نوع پیوندهای نیازمندی به کد [7, 8, 4] و آزمون به کد [9, 10, 11, 12] انجام شده اند. کارهای مرتبط با ترمیم پیوندهای ردپذیری نیازمندی در سال‌های ۲۰۱۲ تا ۲۰۱۶ میلادی روی روش‌های بازیابی اطلاعات تمرکز داشته‌اند. از سال ۲۰۱۷، پژوهشگران در کنار روش‌های بازیابی اطلاعات، به روش‌های ابتکاری، یادگیری ماشین و شبکه‌های عصبی مصنوعی روی آوردند که پتانسیل بیشتری دارند [13]. دو مورد از کارهایی که از شبکه عصبی استفاده کرده‌اند بدون در نظر گرفتن کد بوده و روی ارتباطات بین نیازمندی‌ها تمرکز داشته‌اند [14, 15] که در یکی از آن‌ها روشی ترکیبی از شبکه عصبی و بازیابی اطلاعات با در نظر گرفتن چندمعنایی در نیازمندی‌ها ارائه شده است [15]. در کار دیگری که چند روش شامل شبکه عصبی آزمایش شده‌اند، روش‌های بازیابی اطلاعات بولی گسترش‌یافته و BM25 بهترین نتیجه را داشته‌اند [16]. در سال ۲۰۲۱ در روشی بر اساس یادگیری عمیق که در آن از مدل زبان استفاده شده است، نتایج بسیار بهتری نسبت به روش‌های بازیابی اطلاعات بدست آمده است [4]. یکی از کارهای مرتبط اخیر در ترمیم پیوندهای ردپذیری نیازمندی، روی نیازمندی‌های از نوع موارد کاربرد تمرکز داشته و در آن با استفاده از یادگیری ماشین و استدلال منطقی پیوندهای ردپذیری ترمیم می‌شوند. استدلال منطقی در این کار، از ارتباطات بین موارد کاربرد و همچنین کد بهره گرفته و پس از ترمیم اولیه، پیوندهای مرتبط بیشتری با استفاده از این روابط ترمیم می‌شوند [8]. البته تمرکز این روش روی موارد کاربرد، کاربرد این روش را محدود به این نوع نیازمندی کرده است. یکی دیگر از کارهای اخیر T-BERT [4] است که از مدل زبان BERT استفاده می‌کند. مدل زبان BERT قابلیت پیش‌آموزش[۴] با داده‌های بدون برچسب را داشته که می‌توان در مرحله بعد در کاربردهای مختلف با اضافه کردن یک لایه خروجی تنظیم دقیق[۵] کرد [17]. نتایج این کار از کارهایی که از روش‌های بازیابی اطلاعات مانند نمایه‌سازی معانی پنهان[۶] یا LSI استفاده می‌کنند بهتر است. برای مثال در CODFREL [7] با استفاده از LSI که روشی برای بازیابی اطلاعات با استخراج معانی است [18] و الگوریتم ژنتیک پیوندهای نیازمندی به کد به صورت دانه ریز ترمیم می‌شوند و در آن ادعا شده که این روش بهتر از کاربرد تنها LSI برای یافتن پیوندهای نیازمندی به تابع عمل می‌کند، اما همچنان عملکرد مدل زبان در این کاربرد بسیار بهتر است. در T-BERT از داده‌های موضوع[۷] به تصدیق[۸] برای ارزیابی استفاده شده که در آن تصدیق همراه با پیام تصدیق است. در این پژوهش مانند T-BERT از مدل زبان پیش‌آموزش شده[۹] CodeBERT و داده‌های CodeSearchNet[۱۰] برای آموزش استفاده می‌شود. بر خلاف T-BERT، از داده‌های CodeSearchNet که دارای پیوندهای مستندات به تابع است به صورت مستقیم برای ارزیابی استفاده می‌شود. مقایسه کارهای مرتبط اخیر در زمینه ترمیم ردپذیری نیازمندی، در جدول (۱) آمده است.

۳- روش پیشنهادی

روش پیشنهادی شامل سه روش افزایش کد است. این سه روش تغییر نام متغیر، جابه‌جایی عملوندها و جابه‌جایی جملات هستند. در ادامه هر کدام از این روش‌ها توضیح داده شده‌اند. این روش‌ها به علت محدود بودن محدوده‌ی کد به تعریف تابع و نداشتن اطلاعات معنایی از اطراف، طوری طراحی شده‌اند که با ابزار نحوی، بدون ابزار معنایی قابل انجام باشند.

۳-۱- تغییر نام متغیر

برای تغییر نام متغیر، تعاریف متغیرهای محلی پیدا شده و استفاده‌های آن‌ها در تابع با توجه مکان استفاده‌شان شناسایی می‌شوند. تعاریف متغیرهای محلی به صورت تصادفی انتخاب شده و در صورت انتخاب، نام آن‌ها به همراه استفاده‌هایشان تغییر پیدا می‌کنند. برای تغییر نام از مخفف سازی بخش‌هایی یا کل نام، تغییر نحوه بیان متغیر شمارشی، تغییر قرارداد اسم‌گذاری[۱۱] و جابه‌جایی کلمات موجود در نام و ترکیب‌های تصادفی از این روش‌ها استفاده می‌شود. در تولید نام‌ها توجه شده که نام‌های تکراری، نام‌هایی که با عدد شروع شوند و نام‌های رزرو شده زبان تولید نشوند.

۳-۲- جابه‌جایی عملوندها

بعضی عملگرها قابلیت جابه‌جایی طرفین خود را دارند. این عملگرها شامل «و» و «یا» منطقی، تساوی و عدم تساوی، عملگرهای نامساوی مانند عملگر کوچکتر و عملگرهای جمع و ضرب در صورت عددی بودن عملوندها است. در صورت نامساوی بودن عملگر و جابه‌جایی طرفین آن، جهت عملگر تغییر می‌کند. برای مثال عملگر کوچکتر با عملگر بزرگتر جایگزین می‌شود. برای عملگرهای جمع و ضرب، بررسی می‌شود که عبارات یا مقدارهای دو طرف آن عددی باشند و در صورت نداشتن اطلاعات از نوع داده آن‌ها، اجازه جابه‌جایی داده نمی‌شود.

جدول (۱): مقایسه کارهای مرتبط

نام / روش	نوع نیازمندی	نوع کد	مزایا	معایب
RNN [14]	نیازمندی	–	نتایج بهتر از LSI	نبود ارتباط با کد و عملکرد متوسط
Polysemy [15]	نیازمندی	–	تشخیص چندمعنایی بهتر از LSI	نبود ارتباط با کد
SPLTrac [16]	ویژگی	سند کد	یادآوری بالا	نتایج کلی متوسط
CODFREL [7]	نیازمندی	خطوط کد	دانه ریز بودن محدوده کد	عملکرد متوسط و محاسبات سنگین
ML & Reasoning [8]	مورد کاربرد	کلاس	عملکرد خوب و قابل رقابت	روش محدود به موارد کاربرد
T-BERT [4]	موضوع	تصدیق	نتایج بهتر از روش‌های بازیابی اطلاعات	کاربرد نداشتن پیوند موضوع به تصدیق

۳-۳- جابه‌جایی جملات

پیچیده‌ترین روش ارائه شـــده در این پژوهش برای افزایش کد جابه‌جایی جملات اســت. در این روش، وابســتگی‌ها و ناســازگاری‌های بین جملات با استفاده از مدلی ساده شده از استفاده نام‌ها در زبان‌های برنامه‌نویسی به دست آمده و از آن‌ها برای بررســی امکان جابه‌جایی‌ها اســتفاده می‌شــود. در اولین مرحله، به ازای هر جمله s_i، مجموعه‌های $GET(s_i)$ و $SET(s_i)$ تعریف می‌شوند. مجموعه $GET(s_i)$ نشان‌دهنده نام‌های استفاده شده بدون تغییر روی آن‌ها و $SET(s_i)$ نشان‌دهنده نام‌هایی است که در آن جمله تعریف یا تنظیم شـــده‌اند. همه موارد موجود در مجموعه $SET(s_i)$ در $GET(s_i)$ نیز اضافه می‌شوند، بنابراین $SET(s_i)$ زیرمجموعه $GET(s_i)$ است. دلیل این کار، حفظ ترتیب اعمالی اســـت که نامی را تعریف کرده یا روی آن تاثیر می‌گذارند. شکل (۱) مثالی ساده از ساخت این مجموعه‌ها است.

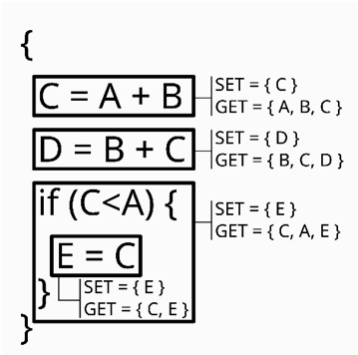
در مرحله دوم، گراف‌های ارتباطات بین جملات در هر قطعه بلوک ساخته می‌شوند. هر قطعه بلوک به صورت جدا بررسی می‌شود. دلیل تقسیم بلوک‌ها به قطعه‌هایی از بلوک‌ها، جلوگیری از جابه‌جایی قبل و بعد عبارات کنترلی مثل return و break است، بنابراین در صورت پیدا کردن جمله‌ای که شامل اینگونه عبارات باشد، بلوک یا قطعه بلوک شامل آن به دو قسمت قبل و بعد آن جمله تبدیل می‌شود. در این کار، با فراخوانی‌های تابع نیز تقسیم بلوک انجام می‌شود، زیرا اطلاعاتی درباره تعریف توابع وجود ندارد و برای احتیاط و حفظ درستی روش از جابه‌جایی جملات قبل و بعد آن‌ها جلوگیری می‌شود. طبق فرمول (۱) به ازای هر بلوک، دو جمله فرضی قبل و بعد آن در نظر گرفته می‌شوند که جمله اول بلوک تنظیم کننده همه جملات و جمله آخر استفاده کننده همه جملات فرض می‌شود. این کار برای حفظ حالت قبل و بعد قطعه بلوک انجام می‌شود. طبق فرمول (۲) در هر قطعه بلوک، به ازای هر نام n، گراف‌های $G_{setter}(n)$ و $G_{incompatiblesetter}(n)$ تعریف شده‌اند. در شکل (۲) مثالی ساده از این ارتباطات آمده است.

$$\begin{cases} SET(s_{-\infty}) = \bigcup_{i \notin \{-\infty,\infty\}} GET(s_i) \\ GET(s_{-\infty}) = \{\} \\ SET(s_{\infty}) = \{\} \\ GET(s_{\infty}) = \bigcup_{i \notin \{-\infty,\infty\}} SET(s_i) \end{cases} \quad (۱)$$

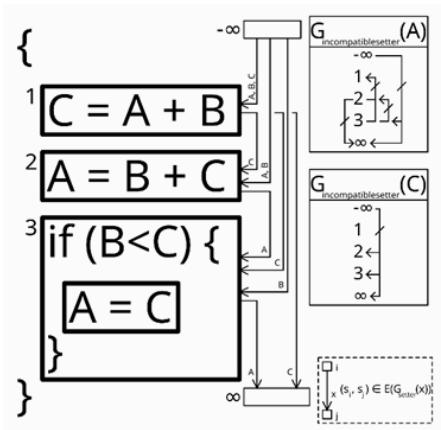
$$\begin{cases} E\left(G_{setter}(n)\right) = \{(s_i, s_j) | \\ \quad n \in SET(s_i) \\ \quad \wedge n \in GET(s_j) \\ \quad \wedge i < j \\ \quad \wedge [\nexists s_k : n \in SET(s_k) \wedge i < k < j]\} \\ \\ E\left(G_{incompatiblesetter}(n)\right) = \{(s_i, s_j) | \\ \quad n \in SET(s_i) \wedge n \in GET(s_j) \\ \quad \wedge (s_i, s_j) \notin E(G_{setter}(n)) \wedge i \neq j\} \end{cases} \quad (۲)$$

در مرحله سوم، جفت‌هایی از جملات به صورت تصادفی با احتمال بیشتر انتخاب جملات نزدیک انتخاب می‌شوند. برای هرکدام از این جفت‌ها، با فرض این که s_a جمله بالایی و s_b جمله پایینی باشد، ۶ شرط که در فرمول (۳) آمده است بررسی می‌شوند. در صورت برقرار بودن این شروط، جابه‌جایی انجام می‌شود. این شروط با بررسی همه حالات جملات قبل، بعد و بین دو جمله مورد نظر برای برقرار نگه داشتن ارتباطات بین جملات به دست آمده‌اند.

$$\begin{cases} \nexists s_i, n: a < i \leq b \wedge (s_a, s_i) \in E\left(G_{setter}(n)\right) \\ \nexists s_i, n: a \leq i < b \wedge (s_i, s_b) \in E\left(G_{setter}(n)\right) \\ \nexists s_i, n: a < i \leq b \\ \quad \wedge (s_i, s_a) \in E\left(G_{incompatiblesetter}(n)\right) \\ \nexists s_i, n: a \leq i < b \\ \quad \wedge (s_b, s_i) \in E\left(G_{incompatiblesetter}(n)\right) \\ \quad \wedge [\exists s_j : (s_j, s_i) \in E\left(G_{setter}(n)\right) \wedge j < a] \\ \nexists s_i, n: b < i \wedge (s_b, s_i) \in E\left(G_{setter}(n)\right) \\ \wedge \left[\exists s_j : (s_j, s_i) \in E\left(G_{incompatiblesetter}(n)\right) \wedge a \leq j < b\right] \\ \nexists s_i, n: b < i \\ \quad \wedge (s_a, s_i) \in E\left(G_{incompatiblesetter}(n)\right) \\ \quad \wedge \left[\exists s_j : (s_j, s_i) \in E\left(G_{setter}(n)\right) \wedge a < j \leq b\right] \end{cases} \quad (۳)$$



شکل (۱): مثالی ساده از ساخت مجموعه‌های نام‌ها



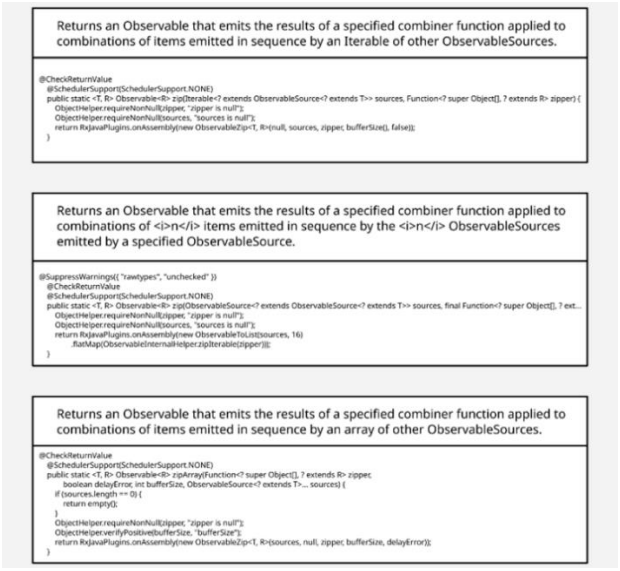
شکل (۲): مثالی از ارتباطات بین جملات

۴- ارزیابی

در این بخش، ابتدا شرایط آزمایش شرح شده، سپس نتایج ارزیابی روش پیشنهادی ارائه و بررسی می‌شود.

۴-۱- شرایط آزمایش

برای یادگیری و ارزیابی مدل به علت شباهت بالا از کد و تنظیمات T-BERT با تغییراتی جهت کاربرد برای کد جاوا و تغییرات در روند یادگیری استفاده شده است. برای محاسبات از ماشین مجازی با ۶ پردازنده مرکزی مجازی، ۲۱.۵ گیگابایت حافظه اصلی با پردازنده گرافیکی GTX 1080 استفاده شده است. در این کار از داده‌های اعتبارسنجی برای توقف یادگیری استفاده شده است. نرخ یادگیری ابتدا 4×10^{-5} بوده که با زمانبندی خطی کاهش می‌یابد. حداکثر تعداد دوره ۸۰[۱۲] در نظر گرفته شده و در صورت بهبود نیافتن نتایج اعتبارسنجی برای ۳ دوره، آموزش متوقف می‌شود. نرخ محاسبه وجود ارتباط پیوند با مدل زبان با استفاده از پردازنده گرافیکی استفاده شده ۱۹.۲۸ پیوند در ثانیه و با پردازنده مرکزی استفاده شده ۶.۷۲ پیوند در ثانیه است. زمان کل آموزش بدون افزایش کد، ۶۳ ساعت و ۲۶ دقیقه و با افزایش کد ۷۷ ساعت و ۱۰ دقیقه بوده، و زمان رسیدن به بهترین مدل بدون افزایش کد ۳۱:۴۸ و با افزایش کد ۳۱:۱۷ بوده است. برای داده‌های آزمون جاوا در CodeSearchNet، برخلاف داده‌های آموزش و اعتبارسنجی آن، نیاز به حذف موارد بوده است. در این داده‌های موارد مجاور مشابه زیادی وجود دارند که باعث اشتباه بودن نتیجه ارزیابی می‌شوند. برای مثال مدل پیش‌بینی می‌کند که پیوندی وجود دارد اما به علت جدا بودن دو سطر اما تشابه بالا، این پیش‌بینی نادرست حساب شده و روی نتیجه تاثیر می‌گذارد. برای غلبه به این مشکل، از موارد مجاور با شباهت بالا در داده‌های آزمون، یک مورد نگه داشته شده و بقیه حذف شده‌اند. در نتیجه این کار، نتایج اعتبارسنجی با نتایج آزمون اختلاف نداشته که نشان‌دهنده اصلاح داده‌ها است. مثالی از این موارد مجاور در شکل (۳) آمده است.



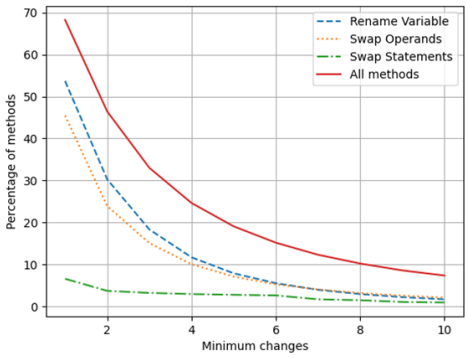
شکل (۳): مثالی از مشکل داده‌های آزمون

در افزایش کد، به ازای هر تابع، اگر مجموع تعداد تغییرات هر سه روش در تلاش اول بین ۴ تا ۷ باشد، یک تابع و اگر ۸ یا بیشتر باشد، دو تابع تولید می‌شود. در تولید اولین مورد، برای تغییر نام و جابه‌جایی عملوندها پارامتر احتمال انتخاب ۰.۷۵ و تعداد تلاش‌های جابه‌جایی جملات دو برابر تعداد جملات هر قطعه بلوک تنظیم شده‌اند. در تولید دومین مورد، احتمال انتخاب برای تغییر نام و جابه‌جایی عملوندها به صورت تصادفی بین ۰.۵ تا ۱ انتخاب شده و تعداد تلاش‌های جابه‌جایی عملوندها به صورت تصادفی ۱ تا ۱۰ برابر تعداد جملات هر قطعه بلوک تعیین می‌شود تا موارد تولید شده نسبت به یکدیگر و تابع مبدا متنوع باشند. با این پارامترها، تعداد توابع تولید شده ۳۴.۸۰ درصد تعداد توابع مبدا بوده که معمولا توابع با خطوط بیشتر هستند.

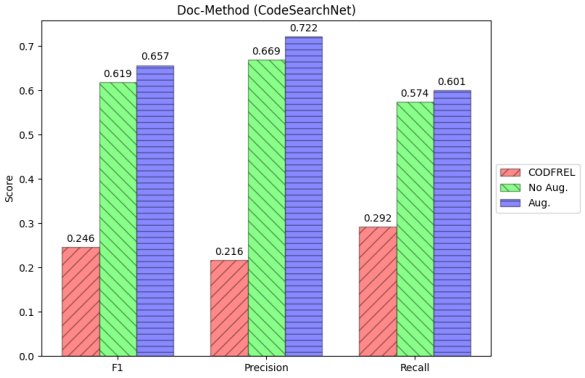
۴-۲- نتایج ارزیابی

در شکل (۴) درصد مشارکت ۳ روش افزایش کد در تولید داده نشان داده شده است. در این نمودار، محور افقی حداقل تعداد تغییرات انجام شده توسط روش یا روش‌ها، و محور عمودی نشان‌دهنده درصد توابعی که با اعمال روش‌ها روی آن‌ها این شرط برقرار می‌شود است. در این نمودار ۳ روش افزایش داده و حالتی که هر ۳ روش اعمال شوند نمایش داده شده‌اند. طبق این آمار، روش تغییر نام متغیر بیشترین شرکت و جابه‌جایی جملات به علت محدودیت‌ها کمترین شرکت در تولید داده‌ها برای افزایش داده را دارد.

در شکل (۵) نتایج ارزیابی مدل آموزش داده شده با افزایش داده و بدون افزایش داده، و همچنین پیاده‌سازی ای از CODFREL روی داده‌های مستندات به تابع نشان داده شده‌اند. در پیاده سازی CODFREL به علت استفاده آن از کلمات کلیدی نیازمندی‌ها و نبود اطلاعات کلمات کلیدی در داده‌های موجود، از از ابزار multi-rake[۱۳] که براساس الگوریتم RAKE [19] است برای به دست آوردن کلمات کلیدی از مستندات استفاده شده است و همچنین پس از آزمایش عملکرد شرط‌های توقف مختلف، شرط توقف پویای ثابت ماندن بهترین مورد از جمعیت برای ۱۰۰ نسل به عنوان شرطی با عملکرد بهتر انتخاب شده است. بقیه شرایط و تنظیمات آزمایش CODFREL طبق کار اصلی است. برچسب‌های محور افقی، امتیازها و انواع هاشور، نشان‌دهنده روش‌ها بوده، و محور عمودی نشان‌دهنده امتیازهای معیارهای ارزیابی بسته به برچسب است. این داده‌ها به علت ارتباط زیاد متن زبان طبیعی‌شان با تابع، مشابه پیوندهای نیازمندی به کد هستند. ارزیابی‌های مدل‌های زبان با محدودیت ۱۰۰۰ سطر و ارزیابی‌های CODFREL به علت طولانی بودن اجرا با محدودیت ۴۰ سطر داده اجرا شده‌اند. در نتیجه آزمون ویلکوکسون، مقدار p-value برای بررسی معنادار بودن تفاوت نتایج بین آزمایش با افزایش کد و بدون آن برای معیار دقت[۱۴] 2.44×10^{-9}، برای یادآوری[۱۵] 8.29×10^{-16} و برای امتیاز F1 مقدار 2.49×10^{-11} است. جمعیت مورد استفاده در آزمایش ویلکوکسون، جمعیت امتیازهای ارزیابی به ازای هر پرس‌وجو[۱۶] یا به عبارتی دیگر نتایج هر مورد مستندات است. مقدارهای به دست آمده نشان‌دهنده احتمال خیلی کم تصادفی بودن این اختلاف و در نتیجه معنادار بودن بهبود عملکرد مدل توسط افزایش کد است.



شکل (٤): درصد شرکت روش‌های افزایش کد



شکل (٥): مقایسه نتایج بدون افزایش کد و با افزایش کد

٥- نتیجه

در این کار دیدیم که افزایش داده‌های کد با استفاده از سه روش کلی تغییر نام متغیر، جابه‌جایی عملوندها و جابه‌جایی جملات، روشی موثر برای بهبود عملکرد مدل زبان در داده‌های مستندات به تابع است. در نتایج ارزیابی نشان داده شد که افزایش کد در همه معیارها بهبود معنادار داشته است و همچنین عملکرد مدل زبان بسیار بهتر از روش CODFREL که استفاده از روش بازیابی اطلاعات LSI و الگوریتم ژنتیک است، می‌باشد. در T-BERT از کارهای مرتبط از مدل زبان استفاده می‌کند، از پیوندهای استخراج شده موضوع به تصدیق برای ارزیابی استفاده شده است که در آن تصدیق‌ها شامل پیام تصدیق نیز هستند. به نظر نمی‌رسد این پیوندها کاربردی برای ردپذیری نرم‌افزار داشته باشند، اما استخراج پیوندها بین موضوع و توابع از این داده‌ها می‌تواند برای کاربرد ردپذیری نرم‌افزار مفید باشد. در کارهای آینده به بررسی انواع دیگر داده‌ها از جمله پیوندهای موضوع به تابع و تلاش برای بهبود مدل در آن کاربرد خواهیم پرداخت. برای این کار باید استخراج داده و ردیابی تصدیق‌ها برای پیدا کردن توابع مربوط انجام شود. پیوندهای موضوع به تابع روابطی غیرمستقیم هستند و انتظار می‌رود ترمیم پیوندهای این نوع دارای دشواری بیشتری نسبت به دیگر انواع پیوندها باشد.

مراجع

[1] G. Spanoudakis and A. Zisman, "Software Traceability: A Roadmap," *Handbook Of Software Engineering And Knowledge Engineering*, vol. 3, pp. 395-428, 2005.

[2] M. Borg, P. Runeson and A. Ardö, "Recovering from a Decade: A Systematic Mapping of Information Retrieval Approaches to Software Traceability," *Empirical Software Engineering*, vol. 19, no. 6, pp. 1565-1616, 2014.

[3] P. Rempel and P. Mäder, "Preventing Defects: The Impact of Requirements Traceability Completeness on Software Quality," *IEEE Transactions on Software Engineering*, vol. 43, no. 8, pp. 777-797, 2017.

[4] J. Lin, Y. Liu, Q. Zeng, M. Jiang and J. Cleland-Huang, "Traceability Transformed: Generating more Accurate Links with Pre-Trained BERT Models," in *Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, May 2021, Madrid, ES, pp. 324-335.

[5] D. Jurafsky and J. H. Martin, "Speech and Language Processing," Pearson, Uttar Pradesh (India), 2020.

[6] I. Goodfellow, Y. Bengio and A. Courville, "Deep Learning," MIT Press, 2016.

[7] D. Blasco, C. Cetina and Ó. Pastor, "A fine-grained requirement traceability evolutionary algorithm: Kromaia, a commercial video game case study," *Information and Software Technology*, vol. 119, p. 106235, 2020.

[8] T. Li, S. Wang, D. Lillis and Z. Yang, "Combining Machine Learning and Logical Reasoning to Improve Requirements Traceability Recovery," *Applied Sciences*, vol. 10, no. 20, p. 7253, 2020.

[9] A. Qusef, G. Bavota, R. Oliveto, A. De Lucia and D. Binkley, "SCOTCH: Test-to-code traceability using slicing and conceptual coupling," in *Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM)*, Sept. 2011, Williamsburg, VA, USA, pp. 63-72.

[10] A. Kicsi, V. Csuvik and L. Vidács, "Large Scale Evaluation of Natural Language Processing Based Test-to-Code Traceability Approaches," *IEEE Access*, vol. 9, pp. 79089-79104, 2021.

[11] B. V. Rompaey and S. Demeyer, "Establishing traceability links between unit test cases and units under test," in *Proceedings of the 13th European Conference on Software Maintenance and Reengineering*, March 2009, Kaiserslautern, Germany, pp. 209-218.

[12] R. White, J. Krinke and R. Tan, "Establishing multilevel test-to-code traceability links," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE'20)*, June 2020, Seoul, South Korea, pp. 861-872.

[13] T. W. W. Aung, H. Huo and Y. Sui, "A literature review of automatic traceability links recovery for software change impact analysis," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 324-335.

[14] J. Guo, J. Cheng and J. Cleland-Huang, "Semantically enhanced software traceability using deep learning techniques," in *Proceedings of the 2017 IEEE/ACM 39th*

International Conference on Software Engineering (ICSE), 2017, pp. 3-14.

[15] W. Wang, N. Niu, H. Liu and Z. Niu, "Enhancing automated requirements traceability by resolving polysemy," in *Proceedings of the 2018 IEEE 26th International Requirements Engineering Conference (RE)*, 2018, pp. 40-51.

[16] T. Vale and E. S. de Almeida, "Experimenting with information retrieval methods in the recovery of feature-code SPL traces," *Empirical Software Engineering*, vol. 24, pp. 1328-1368, 2019.

[17] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," arXiv preprint arXiv:1810.04805. 2018 Oct 11.

[18] B. Rosario, "Latent Semantic Indexing: An overview," *Techn. rep. INFOSYS*, vol. 240, pp. 1-16, 2000.

[19] S. Rose, D. Engel, N. Cramer and W. Cowley, "Automatic keyword extraction from individual documents," *Text mining: applications and theory*, pp. 1-20, 2010.

زیرنویس‌ها

[1]Software Traceability
[2]Artifacts
[3]Source Code
[4]Pre-train
[5]Fine-tune
[6]Latent Semantic Indexing
[7]Issue
[8]Commit
[9]https://github.com/microsoft/CodeBERT
[10]https://github.com/github/CodeSearchNet
[11]ming Convention
[12]Epoch
[13]https://pypi.org/project/multi-rake
[14]Precision
[15]Recall
[16]Query