علوم رایانش و فناوری اطلاعات
نشریه علمی انجمن کامپیوتر ایران
مجلد ۱۵، شماره ۱، ۱۳۹۶
صفحات ۵۵ - ۶۲
مقاله عادی

نشریه علوم رایانش و فناوری اطلاعات

شناسایی خودکار نقش‌های الگوهای طراحی از کد برنامه با رویکرد یادگیری ماشین

مهناز باغدار سعید جلیلی

دانشکده مهندسی برق و کامپیوتر، دانشگاه تربیت مدرس، تهران، ایران

چکیده

شناسایی الگوهای طراحی استفاده شده در یک سیستم نرم‌افزاری به نگهداری و مهندسی مجدد نرم‌افزار کمک می‌کند و نیز باعث سهولت فهم کد برنامه‌ها می‌شود. این فهم به انطباق بین کد برنامه‌ها و طراحی آن‌ها، کمک زیادی می‌کند. علاوه براین، پیاده‌سازی‌های مختلف از یک الگوی طراحی، تشخیص نمونه‌ی الگوها از کد برنامه را سخت می‌کند. از آنجایی که هر الگوی طراحی مجموعه‌ای از نقش‌هایی است که توسط کلاس‌ها در برنامه ایفا می‌شوند و در واقع نقش‌ها اجزای اصلی و تعیین کننده در الگوهای طراحی هستند، با تعیین نقشی که هرکلاس در یک نمونه الگو ایفا می‌کند، می‌توان الگوهای طراحی برنامه را شناسایی نمود. بنابراین، در این مقاله، روشی برای شناسایی نقش‌های الگوهای طراحی از کد برنامه پیشنهاد می‌گردد که مسئله تشخیص نقش‌ها را به یک مسئله یادگیری ماشین نگاشت می‌کند. نتایج آزمایشات با استفاده از برنامه‌های واقعی نشان می‌دهد که روش پیشنهادی، روش نسبتا موفقی است.

کلمات کلیدی: شناسایی الگوهای طراحی، شناسایی نقش‌های الگوهای طراحی، یادگیری ماشین.

۱- مقدمه

تولیدکنندگان در فرایند ساخت نرم‌افزار، برای افزایش اطمینان به عملکرد درست برنامه‌ها، و اینکه برنامه‌نویسان پیاده‌سازی را بر مبنای طراحی انجام داده‌اند، سعی می‌کنند میزان انطباق کد برنامه‌ها با طراحی آنها را شناسایی کنند. از آنجایی که معمولا با حجم زیادی از کد برنامه‌نویسی روبرو هستیم، بررسی کل کد برنامه به‌صورت دستی، سخت و خطاخیز است، در نتیجه در عمل تقریبا غیرممکن است. برای تحقق این هدف، سعی شده است قسمت‌هایی از کد برنامه که متناظر با عناصر طراحی تفصیلی هستند مورد بحث و بررسی قرار گیرد.

این عناصر طراحی عبارتند از الگوهای طراحی[1] که در برنامه‌ها به‌صورت گسترده مورد استفاده قرار می‌گیرند. شناسایی الگوهای طراحی استفاده شده در کد برنامه‌ها، باعث سهولت فهم کد برنامه می‌شود و این فهم، به انطباق بین کد برنامه‌ها و طراحی آنها، کمک زیادی می‌کند.

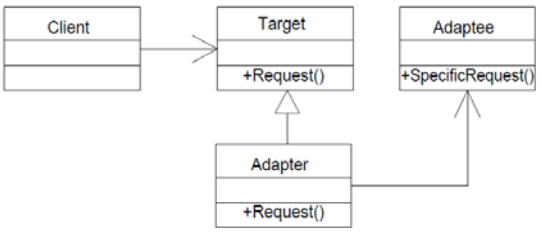
۲- تعریف مسئله

الگوی طراحی[2] یک راه حل بهینه برای یک مسئله‌ی طراحی نرم‌افزار است که بارها اتفاق افتاده است. الگوهای طراحی معمولا مربوط به طراحی شی‌گرا هستند که شامل تعدادی کلاس با روابطی مثل ارث بری[3]، تجمیع[4] و وکالت[5] می‌باشند [۱]. در واقع یک الگوی طراحی معمولا از تعدادی نقش تشکیل می‌شود که هر نقش توسط یک یا چند کلاس ایفا می‌شود. شکل ۱ ساختار الگوی طراحی Adapter را برای نمونه نشان می‌دهد.

شناسایی الگوهای طراحی باعث سهولت فهم کد برنامه می‌شود، مخصوصا در مرحله پیاده‌سازی طراحی‌ها و در مرحله نگهداری، زمانی که مستندات مناسب فراهم نیست. در نتیجه، شناسایی الگوهای طراحی از روی کد برنامه‌ها، برای اطمینان از پیاده‌سازی طراحی‌ها توسط برنامه‌نویسان، نگهداری نرم‌افزار، بازسازی معماری نرم‌افزار، تجدید مستندسازی و ترمیم طراحی نرم‌افزار مفید است.

با این حال شناسایی الگوهای طراحی از کد برنامه آسان نیست؛ تعریف الگوهای طراحی اغلب یک تعریف انتزاعی و غیررسمی است که کمتر به پیاده‌سازی آن می‌پردازد و این تعریف نیز به‌صورت صریح در کد برنامه‌ها مستند نمی‌شود. این

حقیقت باعث می‌شود پیاده‌سازی‌های مختلفی برای یک الگوی طراحی وجود داشته باشد، از این‌رو، برای تشخیص خودکار الگوهای طراحی، نمی‌توان تنها به تعاریف آنها اکتفا نمود. بنابراین در این مقاله از تکنیک‌های یادگیری ماشین برای تشخیص نقش‌های الگوهای طراحی استفاده می‌کنیم که بجای تعاریف الگوها از نمونه‌های الگوهای طراحی که با سلیقه‌های متفاوت پیاده‌سازی شده‌اند، بهره می‌گیرد. در واقع در این پژوهش، نمونه الگوهای طراحی موجود در برنامه از طریق شناسایی نقش‌های کلاس‌ها در الگوها، شناسایی می‌شوند. به عبارت دیگر، هر کلاس در برنامه، به تنهایی و مستقل از سایر کلاس‌ها بررسی می‌شود، با این دید که آیا می‌تواند کاندید یک یا چند نقش باشد یا خیر. در حالیکه در سایر روش‌ها ابتدا ترکیب تمامی کلاس‌هایی که می‌توانند نمونه یک الگوی طراحی باشند شناسایی شده و با در نظر گرفتن ویژگی‌های این کلاس‌ها در کنار هم، الگوهای طراحی موجود در کد برنامه شناسایی می‌شوند.



شکل ۱- ساختار الگوی طراحی Adapter [۱]

چندین روش برای شناسایی خودکار الگوهای طراحی از کد برنامه پیشنهاد شده است که با درنظر گرفتن ویژگی‌های ساختاری یک الگوی طراحی، برنامه را به‌صورت ایستا تجزیه و تحلیل می‌کنند و یا با ترکیب ویژگی‌های ساختاری و رفتاری برنامه، تحلیل را به‌صورت پویا انجام می‌دهند. این بررسی‌ها بوسیله‌ی روش‌هایی از جمله: یادگیری ماشین [۲، ۳]، روش‌های امتیاز ـ تشابه[۶] که مبتنی بر الگوریتم‌های انطباق گراف[۷] است و روش‌های مبتنی بر آنتولوژی[۸] [۴، ۵] انجام می‌شود. به‌طور کلی این رویکردها را به دو دسته تقسیم می‌کنیم: رویکردهای ایستا و پویا.

رویکردهای ایستا [۲، ۳، ۸-۶] بر جنبه‌های ساختاری کد برنامه شامل کلاس‌ها، متدها و روابط بین آن‌ها تمرکز می‌کنند.

رویکردهای پویا [۹، ۱۰] از اطلاعات بدست آمده در حین اجرای برنامه مثل فراخوانی متدها استفاده می‌کنند. یکی از مزایای استفاده از روش‌های پویا قابلیت تمایز بین الگوهای مختلف با ساختار مشابه ولی رفتار متفاوت است. با این حال انتخاب نمونه الگوهای مناسب که تمامی رفتار برنامه را شامل می‌شوند، برای اطمینان از تجزیه و تحلیل پویا، مهم است.

در ادامه نیز به بیان روش پیشنهادی در بخش ۲ می‌پردازیم. نتایج آزمایشات در بخش ۳ ارائه خواهد شد. در پایان نتیجه‌گیری و کارهای آتی بیان می‌شود.

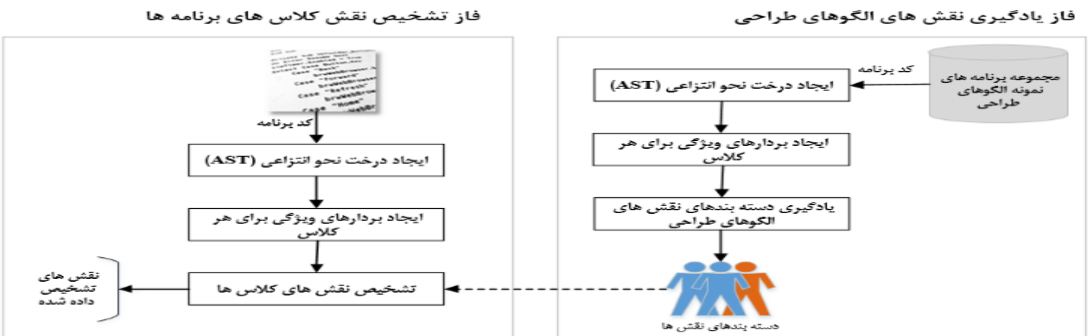
۳- روش پیشنهادی

همانطور که گفته شد، در مهندسی نرم‌افزار هر الگوی طراحی یک راه‌حل کلی برای مسائل طراحی نرم‌افزار است. این راه‌حل‌ها در برنامه‌ها به عنوان ریزمعماری‌ها[۹] پیاده‌سازی می‌شوند [۷]. ریزمعماری‌ها ترکیبی از کلاس‌ها، متدها، فیلدها و روابطی هستند که ساختارهایی مشابه الگوهای طراحی دارند. به‌عبارت دیگر، یک الگوی طراحی مجموعه‌ای از نقش‌هایی است که توسط کلاس‌ها در برنامه ایفا می‌شوند و هر نقش روابطی مثل ارث بری، تجمیع و وکالت با سایر نقش‌ها دارد. بنابراین نقش‌ها اجزای مهم الگوهای طراحی هستند و در واقع با تشخیص نقش هر کلاس در برنامه، نمونه الگوهای طراحی موجود شناسایی می‌شوند.

بنابراین در این مقاله، مسئله‌ی تشخیص نقش‌ها در الگوهای طراحی را به یک مسئله‌ی یادگیری ماشین نگاشت می‌کنیم. شکل ۲ معماری کلی روش پیشنهادی را نشان می‌دهد. بدین‌منظور، هر کلاس در برنامه به‌صورت یک بردار ویژگی نشان داده می‌شود و برای هر نقش در الگوهای طراحی یک دسته‌بند یاد گرفته می‌شود. خصیصه‌های یادگیری استفاده شده در این روش، ریزساختارها[۱۰] شامل EDPs[۱۱] [۱۱] و DPCs[۱۲] [۷] هستند (جدول ۱) که به همراه تعدادی از سنجنده‌های شی‌گرا[۱۳] (جدول ۲) به عنوان ویژگی‌هایی برای هر کلاس در برنامه در نظر گرفته می‌شوند.

تفاوت روش ما با سایر روش‌ها در استفاده از ریزساختارها این است که، در روش پیشنهادی، تعدادی از ریزساختارها به‌صورت ویژگی‌های عددی برای هر کلاس در برنامه و مستقل از کلاس‌های دیگر در نظر گرفته شده‌اند در حالیکه در روش‌های دیگر [۳، ۷] ریزساختارها به‌صورت ویژگی‌های دودویی بین جفت کلاس‌ها در یک نمونه الگوی طراحی نشان داده می‌شوند. این تفاوت ناشی از این است که هدف ما، یادگیری نقش‌های الگوهای طراحی به عنوان عناصر اصلی و تعیین کننده در الگوهای طراحی است، در حالیکه در سایر روش‌ها، هدف یادگیری مستقیم الگوها است. علاوه بر این، استفاده از ریزساختارها به‌صورت ویژگی‌های دودویی بین جفت کلاس‌ها، نیاز به در نظر گرفتن همه ترکیبات ممکن کلاس‌ها با ریزساختارها دارد که با توجه به تعداد زیاد کلاس‌ها در برنامه و در نتیجه تعداد بالای ویژگی‌ها، به‌نظر می‌آید از لحاظ زمانی مقرون به‌صرفه نباشد.

برای یادگیری دسته‌بندهای[۱۴] نقش‌ها، ماشین بردار پشتیبان[۱۵] که یک دسته‌بند دودویی است را استفاده می‌کنیم. مقادیر پارامترهای (C, γ) برای ماشین بردار پشتیبان و نیز کرنل مناسب (خطی، چندجمله‌ای و ...) با توجه به طبیعت مسئله انتخاب می‌شود. سپس برای هر نقش در مجموعه آموزشی یک دسته‌بند براساس بهترین پارامترهای بدست آمده، یادگرفته می‌شود و دسته‌بند توسط داده‌های آزمایشی مورد ارزیابی قرار می‌گیرد.



شکل ۲- معماری روش پیشنهادی

با توجه به اینکه در فاز تشخیص نقش کلاس‌های برنامه، هر کلاس در برنامه می‌تواند در نمونه الگوهای مختلفی قرار گیرد و در نتیجه نقش‌های مختلفی در الگوهای متفاوت ایفا کند، بنابراین به هر کلاس ممکن است چندین نقش نسبت داده شود. بنابراین برای تشخیص صحیح نقش‌های هر کلاس، یک آستانه اطمینان برای هر دسته‌بند نقش در نظر می‌گیریم، به‌طوری‌که نقش‌هایی با مقدار احتمالی برابر یا بالاتر از این آستانه اطمینان، به‌عنوان مجموعه نقش‌های ممکن برای کلاس مورد ارزیابی انتخاب می‌شوند.

۳-۱- ریزساختارها

ریزساختارها، ساختارهای موجود در کد برنامه هستند که به‌صورت عناصر برنامه (کلاس‌ها، متدها، خصیصه‌ها و ...) یا روابط بین دو عنصر از برنامه نشان داده می‌شوند. تعریف ریزساختارها اجازه می‌دهد که سیستم نرم‌افزاری را به‌صورت یک گراف نگاه کنیم، به‌طوری‌که کلاس‌ها، متدها و ... را به‌عنوان نودهای گراف و ریزساختارها را یال‌های گراف در نظر بگیریم در این مقاله، دو دسته از ریزساختارها استفاده می‌شوند: EDP و ریزساختارها درEDP، ساختارهای پایه‌ای برنامه‌نویسی (مثل ایجاد شیء یا فراخوانی متدها) را نمایش می‌دهند و مستقل از هر زبان‌نویسی شی‌گرا هستند.

ریزساختارها در DPC [۳] به شناسایی ساختارهای پایه و نیز ویژه الگوهای طراحی کمک می‌کنند و تمرکز اصلی آن‌ها روی ساختارهایی است که در پیاده‌سازی الگوهای طراحی مرسوم هستند. آن‌ها اطلاعات بیشتری درباره نقش‌های الگوهای طراحی در اختیار می‌گذارند. این اطلاعات در حوزه نقش‌ها در ترکیب با ریزساختارهای EDP مورد استفاده قرار می‌گیرد و با وجود اینکه برای شناسایی روابط بین نقش‌ها مفید هستند، منجربه استخراج الگوها نیز می‌شود. به‌طور کلی این ریزساختارها با تجزیه و تحلیل دستورات و عناصر یک کلاس مثل فراخوانی متدها یا تعاریف فیلدها، شناسایی می‌شوند.

جدول ۱ ریزساختارهایی را که به عنوان ویژگی در یادگیری نقش‌های الگوهای طراحی استفاده شده‌اند، نشان می‌دهد. به‌عنوان مثال ویژگی facadmethod برای هر کلاس در برنامه، براساس رابطه ۱ محاسبه می‌شود که x، تعداد متدهای کلاس c که در بدنه خود، فقط متدهایی را فراخوانی می‌کنند که این متدها عضو اجداد یا اینتر فیس‌های ارث بری شده نیستند و y، تعداد کل متدهای کلاس c می‌باشد:

$$f = \frac{x}{y} \qquad (۱)$$

۳-۲- سنجنده‌های شی‌گرا

سنجنده‌ها، اندازه‌گیری‌های کمی هستند که برای ارزیابی و پیش‌بینی کیفیت نرم‌افزار استفاده می‌شوند و به طور کلی به هشت دسته تقسیم می‌شوند؛ سنجنده‌های پایه، چسبندگی، پیچیدگی، همبستگی، حداکثری، ارث‌بری، ارث‌بری مبتنی بر اتصال، سنجنده‌های چندریختی [۱۲]. تعدادی از سنجنده‌هایی که در این مقاله به عنوان خصیصه‌های کلاس‌های نماینده‌ی نقش‌های الگوهای طراحی استفاده شده‌اند، در جدول ۲ نمایش داده شده است.

جدول ۱- ریزساختارهای استفاده شده در تشخیص نقش‌ها [۷، ۱۱]

دسته	ویژگی‌ها (ریزساختارها)	نوع ویژگی
Clue	Final class	دودویی
	Interface and class inherited	دودویی
	Multiple interfaces inherited	دودویی
	Template implementer	دودویی
	Proxy class	دودویی
	Protected instantiation	دودویی
	Cross relationship	دودویی
	Leaf class	دودویی
	Node class	دودویی
	# facade method	عددی
	# static flag	عددی
	# controlled self-instantiation	عددی
	# private self-instance	عددی
	# static self-instance	عددی
	# single self-instance	عددی
	# instance in abstract class	عددی
	# same interface container	عددی
	# same interface instance	عددی
	# controlled parameter	عددی
	# adapter method	عددی
	# interface method	عددی
	# component method	عددی
	# template method	عددی
	# factory method	عددی
	# instance in abstract referred	عددی
	# abstract cyclic call	عددی
	# methods whit concrete product getter	عددی
	# method with concrete product returned	عددی
	# methods with empty product getter	عددی
	# empty method	عددی
	# multiple returns	عددی
EDP	Create object	عددی
	Abstract interface	عددی
	Retrieve	عددی
	Delegate	عددی
	Inheritance	عددی
	Redirect	عددی
	Conglomeration	عددی
	Recursion	عددی
	Revert method	عددی
	Extend method	عددی
	Delegated conglomeration	عددی
	Redirected recursion	عددی
	Delegate in family	عددی
	Redirect in family	عددی

جدول ۲- سنجنده‌های شی‌گرا [۱۲، ۱۳]

نوع	سنجنده	شرح
پایه	NOA	Number of Attributes
	NOO	Number of Operations
ارث‌بری	NOCC	Number of Child Classes
	DOIH	Depth of Inheritance Hierarchy
	NAC	Number of Ancestor Classes
چندریختی	NOAM	Number of Overridden Methods
اتصال	CTA	Coupling through abstract data type

۴- ارزیابی روش پیشنهادی

روش پیشنهادی روی نمونه‌های الگوهای طراحی که شامل یک تا پنج نقش هستند، ارزیابی می‌شود. به این منظور از مجموعه داده P-MARt[۱۶] [۱۴] استفاده می‌کنیم که از نه سیستم نرم‌افزاری متن باز تشکیل شده است. این مجموعه داده توسط افراد خبره گردآوری شده و بطور گسترده برای ارزیابی ابزارهای شناسایی الگوهای طراحی مورد استفاده قرار می‌گیرد. در این مجموعه داده، نمونه الگوهای طراحی در هر برنامه و نقش هر کلاس در یک نمونه الگو مشخص شده است. جدول ۳ سیستم‌های نرم‌افزاری موجود در مجموعه داده و برخی از مشخصات آنها را نشان می‌دهد. الگوهای طراحی مورد بررسی در این پژوهش به همراه نقش‌های آن‌ها و تعداد نمونه‌های موجود از هر کدام در مجموعه داده نیز در جدول ۴ نشان داده شده است.

جدول ۳- مشخصات سیستم‌های نرم‌افزاری استفاده شده در آزمایشات [۳]

پروژه	تعداد بسته‌ها	تعداد کلاس‌ها
JHotDraw 5.1	۱۱	۱۵۵
JRefactory 2.6.24	۴۹	۵۶۹
QuickUML 2001	۱۱	۱۵۶
Lexi 0.1.1	۶	۲۴
Nutch 0.4	۱۹	۱۶۵
PMD 1.8	۳۵	۴۴۶
JUnit 3.7	۱۰	۷۸
MapperXML 1.9.7	۲۵	۲۱۷
Netbeans 1.0.x	۱۸۴	۲۴۴۴

برای سنجش روش پیشنهادی از سه معیار ارزیابی: دقت، بازخوانی، F_1 استفاده شده است. نحوه محاسبه معیارهای ارزیابی در روابط ۲-۴ نشان داده شده است.

$$P = \frac{TP}{TP + FP} \quad (۲)$$

$$R = \frac{TP}{TP + FN} \quad (۳)$$

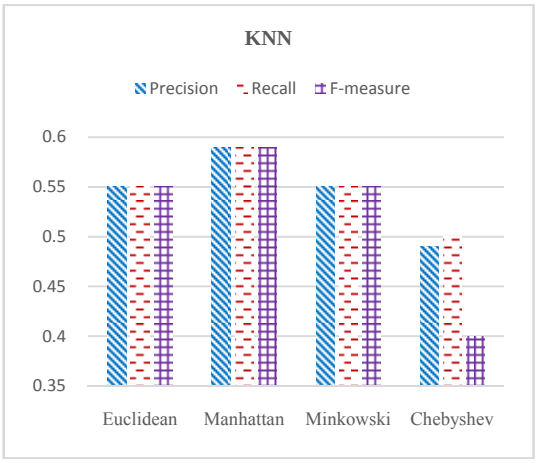
$$F_1 = \frac{2 \times P \times R}{P + R} \quad (۴)$$

روش ارائه شده روی ۴۸ نقش مربوط به ۱۶ الگوی طراحی (جدول ۵) از مجموع ۲۳ الگوهای طراحی GoF[۱۷] ارزیابی می‌گردد. الگوها براساس بیشترین کاربرد و تکرارشان در مجموعه داده انتخاب شده‌اند. بنابراین به منظور ایجاد مجموعه داده، ابتدا کد برنامه‌های ذکر شده در جدول ۳، با استفاده از کتابخانه JDT به درخت نحو انتزاعی[۱۸] معادل تبدیل می‌شود، سپس برای همه کلاس‌های موجود در این برنامه‌ها، ۵۲ ویژگی (۱۴ EDP ، ۳۱ DPC و ۷ سنجنده‌ی شئ‌گرا) گفته شده در بخش ۳، توسط بازدید کننده‌هایی[۱۹] که درخت نحو انتزاعی تولید شده را پویش می‌کنند، استخراج می‌شوند و فراوانی هر ویژگی در کلاس‌های برنامه محاسبه می‌شود و نمونه‌ها به‌صورت دستی، با استفاده از نقش‌های تعیین شده برای آن‌ها در مجموعه داده PMARt، برچسب‌گذاری می‌شوند. به‌منظور انتخاب بهترین روش دسته‌بندی، یادگیری با روش‌های مختلف و با استراتژی 10-foldc.v روی مجموعه داده حاصل اعمال می‌گردد. این آزمایشات با استفاده ابزار WEKA و روی سیستمی با پردازنده Intel Corei7 و دارای ۱۶G حافظه داخلی انجام شده است.

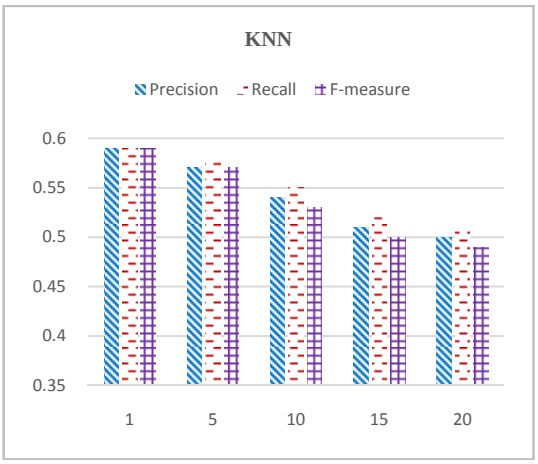
شکل ۳ مقایسه بین معیارهای ارزیابی با استفاده از دسته‌بند KNN و به ازای K=1 و فاصله‌های مختلف را نشان می‌دهد که مشاهده می‌کنیم بهترین نتیجه با استفاده از فاصله منهتن حاصل می‌شود. مقایسه نتایج با استفاده از فاصله منهتن و به‌ازای Kهای مختلف نیز در شکل ۴ نشان داده شده است و همان‌طور که می‌بینیم به ازای k=1 نتایج بهتری حاصل شده است.

جدول ۴- الگوهای طراحی، نقش‌های آن‌ها [۱] و تعداد نمونه‌های هر نقش در مجموعه داده

الگوی طراحی	نقش	تعداد
Abstract Factory	Abstract Factory	۷
	Concrete Factory	۲۰
	Concrete Product	۱۱۳
	Abstract product	۱۷
Factory Method	Creator	۳
	Concrete Creator	۲۸
	Concrete Product	۱۱۳
	Abstract product	۱۷
Template Method	Abstract class	۶
	Concrete class	۷۳
Adapter	Adapter	۶۰
	Adaptee	۵۱
	Target	۱۴
Observer	Subject	۶
	Concrete subject	۶۰
	Observer	۱۴
	Concrete observer	۲۴
Singleton	Singleton	۱۲
Composite	Composite	۱۲
	Component	۷
	leaf	۹۲
State	Context	۲۳
	State	۳
	Concrete state	۲۵
Strategy	Context	۲۳
	Strategy	۷
	Concrete strategy	۱۹
Command	Invoker	۱۸
	Receiver	۹
	Command	۴
	Concrete command	۳۱
Visitor	Visitor	۳
	Concrete visitor	۳۳
	Element	۳
	Concrete element	۹۸
Iterator	Aggregate	۳
	Concrete Aggregate	۱۰
	Iterator	۸
	Concrete Iterator	۱۰
Builder	Builder	۴
	Concrete Builder	۲۸
	Director	۶
	Product	۸
Prototype	Prototype	۲
	Concrete Prototype	۲۷
Decorator	Component	۲
	Concrete Component	۵۵
		۱
	Decorator	۶
Facade	Facade	۱
	Subsystem Class	۱۰

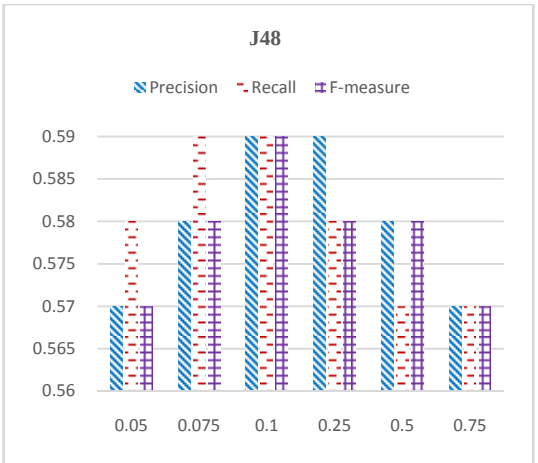


شکل ۳- مقایسه نتایج ارزیابی شناسایی نقش‌ها با استفاده از دسته‌بند KNN با k=1 و به ازای فاصله‌های متفاوت



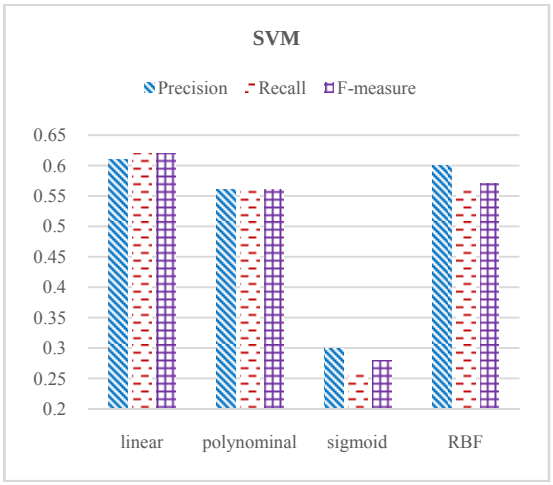
شکل ۴- مقایسه نتایج ارزیابی شناسایی نقش‌ها با استفاده از دسته‌بند KNN با فاصله منهتن و به ازای مقادیر مختلف K

شکل ۵ نتایج مربوط به دسته‌بند درخت تصمیم به ازای مقادیر مختلف θ (پارامتر حرص کردن) را نشان می‌دهد که مشاهده می‌کنیم بهترین نتایج به‌ازای θ=0.1 به‌دست آمده است.



شکل ۵- مقایسه نتایج ارزیابی شناسایی نقش‌ها با استفاده از دسته‌بند درخت تصمیم و به ازای مقادیر مختلف θ

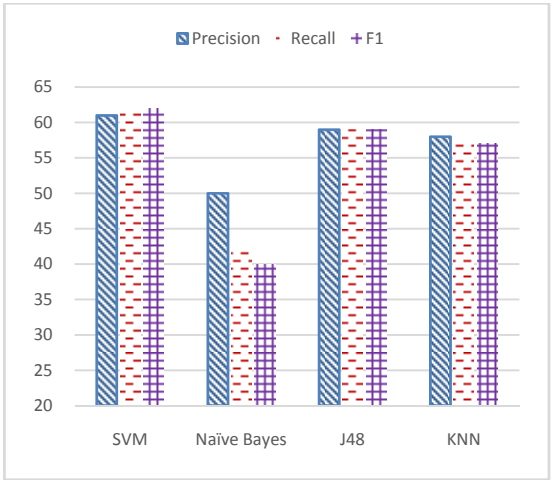
شکل ۶ نتایج دسته‌بند SVM به ازای هسته‌ها و پارامترهای متفاوت ارائه شده است. این نتایج با استفاده از بهترین مقدار پارامتر C=50 گزارش شده است. مقدار پارامتر گاما نیز برای هسته‌های دوجمله‌ای، سیگموئید و RBF برابر با (تعداد ویژگی‌ها/1) مقداردهی شده است. همان‌طور که مشاهده می‌کنیم بهترین نتایج با استفاده از هسته خطی به‌دست آمده است.



شکل ۶- مقایسه نتایج ارزیابی شناسایی نقشه ابا استفاده از دسته‌بند SVM و به ازای هسته‌های مختلف و پارامتر c=50

شکل ۷ مقایسه بین معیارهای ارزیابی برای شناسایی نقش‌های کلاس‌ها با استفاده از چهار دسته‌بند KNN، J48، Naïve Bayes و SVM را نشان می‌دهد. این مقایسه بر اساس بهترین پارامترهای به‌دست آمده برای هر دسته‌بند انجام شده است؛ دسته‌بند KNN بهترین نتیجه را به ازای k=1 و با فاصله منهتن می‌دهد، نتایج مربوط به درخت تصمیم به‌ازای بهترین θ که برابر با ۰.۱ است، گزارش شده است. SVM نیز با هسته‌ی خطی بهترین نتیجه را داده است.

باتوجه به نتایج دسته‌بندی، بهترین نتایج با استفاده از دسته‌بند SVM به‌دست آمده است. بنابراین، ما دسته‌بند SVM را برای یادگیری نقش‌های الگوهای طراحی استفاده می‌کنیم. به دلیل نامتوان بودن تعداد نمونه‌ها در بعضی دسته‌ها و نیز حساس بودن SVM به داده‌های نامتوازن، روش SMOTE در WEKA را برای تولید داده‌های مصنوعی استفاده کردیم.



شکل ۷- مقایسه الگوریتم‌های یادگیری مختلف برای شناسایی نقش‌ها

همانطور که گفته شد در مجموعه داده P-MARt یک کلاس می‌تواند نقش‌های مختلفی در چندین الگو ایفا کند، بنابراین یک نمونه کلاس در ارزیابی نیز ممکن است در چندین دسته قرار گیرد. بدین منظور، با توجه به احتمالی که هر دسته‌بند برای نمونه‌ای که به‌منظور شناسایی به آن داده شده است، می‌دهد، حدآستانه‌ای در نظر می‌گیریم؛ به‌طوریکه نقش‌هایی که دسته‌بندهای آن‌ها، برای یک نمونه آزمون، احتمالی برابر یا بیشتر از حدآستانه تعیین شده می‌دهند، به‌عنوان نقش‌های آن کلاس در نظر گرفته می‌شوند. نتایج ارزیابی شناسایی نقش‌های کلاس‌ها با فرض اینکه یک نمونه بتواند چندین برچسب داشته باشد، با استفاده از دسته‌بند SVM و با سه حدآستانه مختلف در نظر گرفته شده در جدول ۵ نشان داده شده است. با توجه به اهمیت از دست ندادن اطلاعات، معیار بازخوانی از ارزش بالاتری نسبت به معیار دقت برخوردار است. بدین منظور حدآستانه ۰.۷۵ را که مقدار دقت و بازخوانی قابل قبولی دارد، برای شناسایی نقش‌های الگوهای طراحی در نظر می‌گیریم. با حد آستانه ۰.۷۵، حداقل و حداکثر تعداد نقش‌های نسبت داده شده به کلاس‌ها در نمونه برنامه‌های الگوهای طراحی به ترتیب برابر با یک و شش می‌باشد.

۵- نتیجه‌گیری

در این پژوهش، یک روش مبتنی بر یادگیری ماشین برای شناسایی نقش‌ها در الگوهای طراحی از کد برنامه‌ها ارائه گردید.

جدول ۵- نتایج شناسایی نقش‌های الگوهای طراحی با استفاده از دسته‌بند SVM و با حدآستانه‌های ۰.۱، ۰.۷۵ و ۰.۵

نقش	حد آستانه = ۰.۱		حد آستانه = ۰.۷۵		حد آستانه = ۰.۵	
	دقت	بازخوانی	دقت	بازخوانی	دقت	بازخوانی
Abstract product	۸۱٪	۶۹٪	۳۸٪	۱۰۰٪	۲۷٪	۱۰۰٪
Concrete Product	۵۷٪	۹۷٪	۴۳٪	۱۰۰٪	۳۰٪	۱۰۰٪
Abstract Factory	۷۱٪	۷۱٪	۴۳٪	۸۵٪	۳۲٪	۱۰۰٪
Concrete Factory	۶۲٪	۹۴٪	۴۸٪	۱۰۰٪	۳۰٪	۱۰۰٪
Creator	۳۴٪	۵۰٪	۳۴٪	۵۰٪	۳۳٪	۱۰۰٪
Concrete Creator	۶۳٪	۵۰٪	۵۸٪	۵۸٪	۴۲٪	۷۸٪
Abstract class	۷۱٪	۸۳٪	۴۳٪	۱۰۰٪	۲۵٪	۱۰۰٪
Concrete class	۱۰۰٪	۴۳٪	۸۷٪	۵۹٪	۴۴٪	۱۰۰٪
Adapter	۸۳٪	۵۲٪	۴۹٪	۹۸٪	۲۰٪	۱۰۰٪
Adaptee	۸۱٪	۴۳٪	۵۲٪	۶۲٪	۲۵٪	۸۶٪
Target	۴۶٪	۱۰۰٪	۴۴٪	۱۰۰٪	۳۵٪	۱۰۰٪
Subject	۶۰٪	۷۵٪	۶۷٪	۱۰۰٪	۵۰٪	۱۰۰٪
Concrete subject	۸۰٪	۷۰٪	۶۱٪	۸۹٪	۳۲٪	۹۳٪
Observer	۵۰٪	۷۵٪	۳۴٪	۷۵٪	۲۰٪	۱۰۰٪
Concrete observer	۶۵٪	۱۰۰٪	۵۷٪	۱۰۰٪	۳۰٪	۱۰۰٪
Singleton	۵۰٪	۱۰۰٪	۴۸٪	۱۰۰٪	۳۱٪	۱۰۰٪
Composite	۸۸٪	۸۰٪	۷۷٪	۱۰۰٪	۵۳٪	۱۰۰٪
Component	۸۰٪	۸۰٪	۵۸٪	۸۰٪	۳۸٪	۱۰۰٪
leaf	۸۵٪	۸۴٪	۷۶٪	۹۷٪	۴۵٪	۱۰۰٪
Context	۵۸٪	۹۵٪	۴۲٪	۹۵٪	۲۴٪	۱۰۰٪
State	۶۶٪	۱۰۰٪	۶۷٪	۱۰۰٪	۳۳٪	۱۰۰٪
Concrete state	۵۲٪	۱۰۰٪	۳۴٪	۱۰۰٪	۲۰٪	۱۰۰٪
Strategy	۴۱٪	۱۰۰٪	۳۰٪	۱۰۰٪	۱۷٪	۱۰۰٪
Concrete strategy	۶۳٪	۱۰۰٪	۴۰٪	۱۰۰٪	۲۲٪	۱۰٪
Invoker	۵۴٪	۱۰۰٪	۴۵٪	۱۰۰٪	۲۰٪	۱۰۰٪
Command	۴۲٪	۵۰٪	۴۱٪	۵۰٪	۳۰٪	۵۰٪
Concrete command	۹۶٪	۹۳٪	۶۶٪	۱۰۰٪	۳۷٪	۱۰۰٪
Receiver	۵۴٪	۸۵٪	۴۲٪	۱۰۰٪	۲۰٪	۱۰۰٪
Visitor	۵۰٪	۵۰٪	۵۰٪	۵۰٪	۲۵٪	۵۰٪
Concrete visitor	۶۴٪	۱۰۰٪	۵۳٪	۱۰۰٪	۳۲٪	۱۰۰٪
Element	۲۰٪	۵۰٪	۱۸٪	۵۰٪	۲۲٪	۱۰۰٪
Concrete element	۹۳٪	۹۸٪	۹۳٪	۹۸٪	۹۰٪	۹۸٪
Aggregate	-	۰	۱۴٪	۵۰٪	۱۸٪	۱۰۰٪
Concrete Aggregate	۱۰۰٪	۸۳٪	۱۰۰٪	۸۳٪	۵۰٪	۱۰۰٪
Iterator	-	۰	-	-	-	-
Concrete Iterator	-	-	-	-	۳۰٪	۱۰۰٪
Builder	-	۰	-	-	-	-
Concrete Builder	۹۰٪	۱۰۰٪	۸۱٪	۱۰۰٪	۵۳٪	۱۰۰٪
Director	۲۸٪	۱۰۰٪	۱۸٪	۱۰۰٪	۱۲٪	۱۰۰٪
Product	۵۷٪	۱۰٪	۲۸٪	۱۰٪	۱۳٪	۱۰٪
Prototype	۱۰۰٪	۵۰٪	۱۰۰٪	۱۰۰٪	۵۰٪	۱۰٪
Concrete Prototype	۹۴٪	۹۴٪	۸۶٪	۱۰۰٪	۶۱٪	۱۰۰٪
Concrete Component	-	۰	۷۷٪	۴۰٪	۶۶٪	۸۳٪
Decorator	۵۸٪	۵۰٪	۵۳٪	۵۹٪	۴۱٪	۶۵٪
Concrete Decorator	۱۰۰٪	۱۰۰٪	۱۰۰٪	۱۰۰٪	۵۴٪	۱۰۰٪
Facade	-	-	-	-	-	-
Subsystem Class	۵۴٪	۱۰۰٪	۳۸٪	۱۰۰٪	۲۰٪	۱۰۰٪
Client	۷۱٪	۳۹٪	۶۲٪	۶۴٪	۴۲٪	۹۲٪
میانگین	**۶۶٪**	**۶۹٪**	**۵۴٪**	**۸۴٪**	**۳۴٪**	**۸۹٪**

چون نقش‌ها عناصر اصلی الگوهای طراحی هستند و با تعیین نقش هر کلاس در یک نمونه‌ی الگو، الگوهای طراحی موجود در برنامه شناسایی می‌شوند، تمرکز اصلی این پژوهش روی شناسایی نقش‌های الگوها است. برای یادگیری نقش‌ها، از دسته‌بند SVM استفاده کردیم. برای ارزیابی روش پیشنهادی، مجموعه داده P-MARt را بکار بردیم که از نه نرم‌افزار متن باز جمع‌آوری شده است. نتایج حاصل از آزمایشات نشان از کارایی روش معرفی شده برای تشخیص نقش‌های الگوهای طراحی دارد. در پژوهش‌های آتی، ما قصد داریم از ویژگی‌های رفتاری برنامه‌ها و ارتباط بین نقش‌ها در الگوهای طراحی به منظور شناسایی دقیق‌تر الگوها و نیز کاهش مقدار مثبت کاذب[20] (به دلیل در نظر گرفتن چندین برچسب برای هر نقش در این پژوهش) استفاده نماییم.

مراجع

[1] E. Gamma, *Design patterns: elements of reusable object-oriented software*: Pearson Education India, 1995.

[2] A. Chihada, S. Jalili, S. M. H. Hasheminejad, and M. H. Zangooei, "Source code and design conformance, design pattern detection from source code by classification approach," *Applied Soft Computing*, vol. 26, pp. 357-367, 2015.

[3] M. Zanoni, F. Arcelli Fontana, and F. Stella, "On applying machine learning techniques for design pattern detection," *Journal of Systems and Software*, vol. 103, pp. 102-117, 2015.

[4] A. Alnusair, T. Zhao, and G. Yan, "Rule-based detection of design patterns in program code," *International Journal on Software Tools for Technology Transfer*, vol. 16, pp. 315-334, 2013.

[5] D. Kirasić, and D. Basch, "Ontology-based design pattern recognition," in *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, 2008, pp. 384-393.

[6] D. Yu, Y. Zhang, and Z. Chen, "A comprehensive approach to the recovery of design pattern instances based on sub-patterns and method signatures," *Journal of Systems and Software*, vol. 103, pp. 1-16, 2015.

[7] F. Arcelli Fontana, M. Zanoni, and S. Maggioni, "Using Design Pattern Clues to Improve the Precision of Design Pattern Detection Tools," *The Journal of Object Technology*, vol. 10, p. 4:1, 2011.

[8] F. A. Fontana, and M. Zanoni, "A tool for design pattern detection and software architecture reconstruction," *Information sciences*, vol. 181, pp. 1306-1324, 2011.

[9] A. D. Lucia, V. Deufemia, C. Gravino, and M. Risi, "Behavioral Pattern Identification through Visual Language Parsing and Code Instrumentation," in Software Maintenance and Reengineering, pp. 99-108, 2009.

[10] A. De Lucia, V. Deufemia, C. Gravino, and M. Risi, "Improving Behavioral Design Pattern Detection through Model Checking," in Software Maintenance and Reengineering (CSMR), pp. 176-185, 2010.

[11] J. M. Smith, and D. Stotts, "Elemental Design Patterns: A formal semantics for composition of OO software architecture," in *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE*, 2002, pp. 183-190.

[12] M. Lanza, and R. Marinescu, *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer Science & Business Media, 2007.

[13] S. K. Dubey, and A. Sharma, "Comparison study and review on object-oriented metrics," *Global Journal of Computer Science and Technology*, vol. 12, 2012.

[14] Y.-G. Guéhéneuc, "P-mart: Pattern-like micro architecture repository," in *EuroPLoP Focus Group on Pattern Repositories*, 2007.

مهناز باغدار درجه کارشناسی‌ارشد را در سال ۱۳۹۵ از دانشگاه تربیت مدرس اخذ کرده است. زمینه پژوهشی مورد علاقه: یادگیری ماشین و ارزیابی طراحی نرم‌افزار (بررسی الگوهای طراحی) می‌باشد.

آدرس پست‌الکترونیکی ایشان عبارت است از:

mahnaz.baghdar@modares.ac.ir

سعید جلیلی درجه کارشناسی‌ارشد را در سال ۱۳۶۴ از دانشگاه صنعتی شریف و درجه دکتری را در سال ۱۳۷۰ از دانشگاه بردفورد انگلستان اخذ کرده است. از سال ۱۳۷۲ عضو هیات علمی گروه مهندسی کامپیوتر دانشگاه تربیت مدرس است. زمینه‌های پژوهشی مورد علاقه: طراحی (معماری، تفصیلی) جستجو بنیان سیستم‌های نرم‌افزاری، ارزیابی معماری نرم‌افزار، وارسی زمان اجرای سیستم‌های نرم‌افزاری حساس به حیات انسان‌ها، وارسی پروتکل‌های رمزنگاری، تشخیص ناهنجاری در شبکه‌های کامپیوتری می‌باشد.

آدرس پست‌الکترونیکی ایشان عبارت است از:

sjalili@modares.ac.ir

اطلاعات بررسی مقاله:

تاریخ ارسال: ۱۳۹۶/۰۴/۰۴

تاریخ اصلاح: ۱۳۹۶/۰۵/۲۰

تاریخ قبول شدن: ۱۳۹۶/۰۶/۰۵

نویسنده مرتبط: مهناز باغدار، دانشکده مهندسی برق و کامپیوتر، دانشگاه تربیت مدرس، تهران، ایران.

[1] Design Pattern
[2] Design Pattern
[3] Inheritance
[4] Aggregation
[5] Delegation
[6] Similarity Scoring

[7]Graph Matching
[8]Ontology Based Methods
[9]Micro Architectures
[10]Micro Structures
[11]Elemental Design Patterns (EDPs)
[12]Design Pattern Clues (DPCs)
[13]Object-Oriented Metrics
[14]Classifier
[15]Support Vector Machine (SVM)
[16]Pattern-like Micro Architecture Repository (PMARt)
[17]Gang of Four
[18]Abstract Syntax Tree (AST)
[19]Visitors
[20]False Positive