



قدم به قدم، همراه دانشجو...

WWW.GhadamYar.Com

جامع ترین و به روزترین پرتال دانشجویی کشور (پرتال دانش)
با ارائه خدمات رایگان، تحصیلی، آموزشی، رفاهی، شغلی و...
برای دانشجویان

- (۱) راهنمای ارتقاء تحصیلی. (کاردانی به کارشناسی، کارشناسی به ارشد و ارشد به دکتری)
- (۲) ارائه سوالات کنکور مقاطع مختلف سالهای گذشته، همراه پاسخ، به صورت رایگان
- (۳) معرفی روش های مقاله و پایان نامه نویسی و ارائه پکیج های آموزشی مربوطه
- (۴) معرفی منابع و کتب مرتبط با کنکورهای تحصیلی (کاردانی تا دکتری)
- (۵) معرفی آموزشگاه ها و مراکز مشاوره تحصیلی معتبر
- (۶) ارائه جزوات و منابع رایگان مرتبط با رشته های تحصیلی
- (۷) راهنمای آزمون های حقوقی به همراه دفترچه سوالات سالهای گذشته (رایگان)
- (۸) راهنمای آزمون های نظام مهندسی به همراه دفترچه سوالات سالهای گذشته (رایگان)
- (۹) آخرین اخبار دانشجویی، در همه مقاطع، از خبرگزاری های پربازدید
- (۱۰) معرفی مراکز ورزشی، تفریحی و فروشگاه های دارای تخفیف دانشجویی
- (۱۱) معرفی همایش ها، کنفرانس ها و نمایشگاه های ویژه دانشجویی
- (۱۲) ارائه اطلاعات مربوط به بورسیه و تحصیل در خارج و معرفی شرکت های معتبر مربوطه
- (۱۳) معرفی مسائل و قوانین مربوط به سربازی، معافیت تحصیلی و امریه
- (۱۴) ارائه خدمات خاص ویژه دانشجویان خارجی
- (۱۵) معرفی انواع بیمه های دانشجویی دارای تخفیف
- (۱۶) صفحه ویژه نقل و انتقالات دانشجویی
- (۱۷) صفحه ویژه ارائه شغل های پاره وقت، اخبار استخدامی
- (۱۸) معرفی خوابگاه های دانشجویی معتبر
- (۱۹) دانلود رایگان نرم افزار و اپلیکیشن های تخصصی و...
- (۲۰) ارائه راهکارهای کارآفرینی، استارت آپ و...
- (۲۱) معرفی مراکز تایپ، ترجمه، پرینت، صحافی و ... به صورت آنلاین
- (۲۲) راهنمای خرید آنلاین ارزی و معرفی شرکت های مطرح
- (۲۳)



WWW.GhadamYar.Ir

WWW.PortaleDanesh.com

WWW.GhadamYar.Org

۰۹۱۲ ۳۰ ۹۰ ۱۰۸

باما همراه باشید...

۰۹۱۲ ۰۹ ۰۳ ۸۰۱



دانشگاه صنعتی امیرکبیر (پلی تکنیک تهران)

دانشکده‌ی مهندسی کامپیوتر و فن‌آوری اطلاعات

گزارش تحقیقی-عملی درس یادگیری ماشین

آموزش گام‌به‌گام تحلیل شبکه اجتماعی در زبان R

نسخه ۱.۰

نگارش

محسن رئیسی

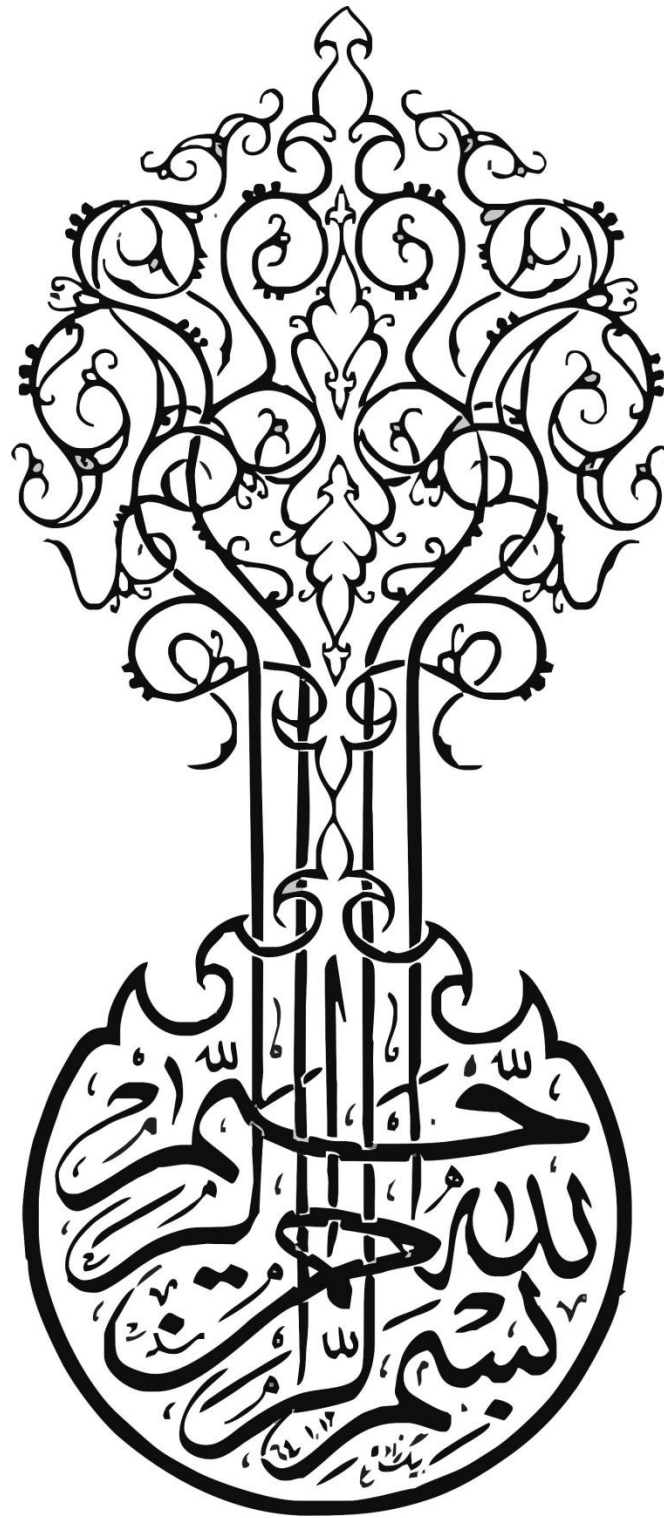
(Raeesi@aut.ac.ir, ceit.aut.ac.ir/~raeesi)

استاد درس

جناب آقای دکتر شیری

تابستان ۱۳۹۰

<http://ceit.aut.ac.ir/~shiry/lecture/machine-learning/ml.html>



پیش‌گفتار

(توصیه می‌شود حتماً قبل از مطالعه متن مستند، این قسمت را مطالعه نمایید.)

مستند پیش‌رو، قصد دارد به دانش‌پژوهان را با تحلیل شبکه‌اجتماعی در R آشنا کند. در این گزارش فرض شده، خواننده محترم تا حدی با زبان R و نحوه کد نویسی در آن آشنا می‌باشد. از این رو به مقدماتی چون انواع متغیرها، ساختار داده‌ای، شرط، حلقه و ... نپرداخته‌است. برای آشنایی بیشتر با این مباحث به *آشنایی با زبان محاسبات آماری R*^۱، نوشته سید سعید موسوی‌ندوشنی مراجعه نمایید^۱.

فصل اول این گزارش در مورد R، تاریخچه و امکانات آن می‌باشد. محیط مورد استفاده در تهیه این گزارش، محیط نرم‌افزار داده‌کاوی Rapidminer می‌باشد. این نرم‌افزار از نسخه ۵ به بعد، افزونه‌ای را برای اجرای دستورات R و استفاده آن در محیط خود ارائه داد. از این‌رو فصل دوم گزارش، به نحوه نصب افزونه R در Rapidminer و چگونگی استفاده از آن اختصاص دارد.

تاکنون چندین کتابخانه برای تحلیل شبکه‌اجتماعی در R، ارائه شده‌است. یکی از قوی‌ترین این کتابخانه‌ها igraph می‌باشد. در این مستند، ما از این کتابخانه برای تعریف گراف و تحلیل شبکه‌اجتماعی استفاده می‌نماییم. از این رو اسکریپت‌ها و کدهای ارائه شده در فصل سوم و چهارم بر اساس این کتابخانه ارائه شده‌اند. منبع اصلی استفاده شده برای این فصول نیز، مستندات و آموزش‌های ارائه شده برای igraph [۱،۲،۳] می‌باشند که سعی شده با توضیحات مناسب در این مستند ارائه شود.

به منظور تسهیل پیمایش متن گزارش، بخش‌های مختلف در فایل PDF، به صورت Bookmarked در آمده‌اند. همچنین برخی منابع کمکی – مثل کدهای فصل سوم و چهارم – به فایل الصاق شده‌است. خوانندگان محترم توجه داشته‌باشند که استفاده از مستند پیش‌رو برای مصارف تجاری مجاز نمی‌باشد.

آرزومند دعای خیر خوانندگان عزیز

محسن رئیسی

^۱ http://cran.r-project.org/doc/contrib/Mousavi-R-lang_in_Farsi.pdf

فهرست مطالب

۱	مقدمه ای بر R	۱
۱-۱	تاریخچه پروژه R	۱
۲-۱	مروری بر امکانات R	۲
۳-۱	تحلیل شبکه اجتماعی در R	۲
۲	نصب R و تعبیه آن در نرم افزار RapidMiner	۳
۱-۲	مراحل نصب هسته اصلی R	۳
۲-۲	نصب افزونه R در Rapidminer	۴
۳-۲	محیط اسکریپت نویسی افزونه R	۷
۴-۲	استفاده از R در Rapidminer	۸
۳	مدلسازی گراف	۱۲
۱-۳	ساخت گراف	۱۴
۲-۳	ایجاد گراف از روی فایل بیرونی	۱۶
۳-۳	دستکاری در ساختار گراف	۱۷
۴-۳	استخراج همسایگی یال ها	۱۷
۵-۳	رسم گراف	۱۸
۶-۳	دسترسی و تغییر صفات گراف، گره ها و یال ها	۲۱
۴	تحلیل شبکه اجتماعی	۲۴
۱-۴	محاسبه انواع تعاریف مرکزیت گره	۲۴
۲-۴	تحلیل یک گراف نسبتا بزرگ	۲۷
۳-۴	ساختار یابی اجتماع	۳۰
۴-۴	بلوک های چسبنده	۳۳
۵	مراجع	۳۵

فهرست اشکال و جداول

۲	شکل ۱. لوگوی پروژه R
۳	شکل ۲. نمایی از یکی از مراحل نصب نرم افزار R
۴	شکل ۳. محیط کنسول نرم افزار R

- شکل ۴. روشن کردن افزونه R در Rapidminer..... ۵
- شکل ۵. اضافه کردن متغیرهای محیطی مرتبط به تنظیمات افزونه R..... ۶
- شکل ۶. اضافه شدن آیکن R در نوارابزار فوقانی..... ۷
- شکل ۷. نمای کلی افزونه R در Rapidminer..... ۸
- شکل ۸. عملگرهای R در نوارابزار..... ۸
- شکل ۹. آیکن فضای کاری‌های مختلف در Rapidminer..... ۹
- شکل ۱۰. مراحل ذخیره داده‌های R در Repository..... ۹
- شکل ۱۱. نحوه تعریف خروجی عملگر (Execute Script (R)..... ۱۰
- شکل ۱۲. مدل فرآیند کلی مراحل تحلیل گراف کلپ کاراته..... ۱۱
- شکل ۱۳. نتیجه ساختاریابی انجام‌شده در فرآیند فوق..... ۱۱
- شکل ۱۴. گراف بدون جهت..... ۱۲
- شکل ۱۵. گراف جهت دار..... ۱۲
- شکل ۱۶. نمونه‌ای از یک شبکه بی‌مقیاس. قطب‌ها به صورت رنگی نشان داده شده‌اند..... ۱۹
- شکل ۱۷. گراف رسم شده در محیط tkplot..... ۲۰
- شکل ۱۸. گراف سه بعدی رسم شده با rgplot..... ۲۱
- شکل ۱۹. میانگین درجه نقاط در گراف قضایی..... ۲۲
- شکل ۲۰. یک گراف تصادفی که رنگ یال‌های آن بر اساس رنگ گره‌ها متمایز گشته‌است..... ۲۳
- شکل ۲۱. گرافی که Degree centrality گره‌های آن در کنار هر گره نوشته شده‌است..... ۲۵
- شکل ۲۲. مرکزیت Betweenness گره‌های گراف..... ۲۶
- شکل ۲۳. مرکزیت Closeness گره‌های گراف..... ۲۶
- شکل ۲۴. مرکزیت Eigenvector گره‌های گراف..... ۲۶
- شکل ۲۵. توزیع درجه‌های ورودی گراف قضایی..... ۲۸
- شکل ۲۶. "ساختار اجتماع" در گراف یک شبکه اجتماعی..... ۳۰
- شکل ۲۷. اجتماعات به دست آمده با الگوریتم fastgreedy روی مجموعه داده کلپ کاراته..... ۳۲
- شکل ۲۸. اجتماعات به دست آمده با الگوریتم Spinglass روی مجموعه داده کلپ کاراته..... ۳۲
- شکل ۲۹. ساختار بلوک‌بندی چسبندگی به دست آمده..... ۳۴

۱ مقدمه ای بر R

زبان R، یک زبان برنامه‌نویسی و محیط نرم‌افزاری برای محاسبات آماری و تحلیل داده است. امروزه این زبان به عنوان یک استاندارد غیر رسمی برای کارهای آماری و داده‌کاوی مطرح می‌باشد [۴]. این زبان در حقیقت نسخه متن‌باز نرم‌افزار S می‌باشد [۵]. زبان R توسط نرم‌افزاری به همین نام که شامل مفسر زبان و محیط اسکریپت‌نویسی است پشتیبانی می‌گردد. این نرم‌افزار، بازمتن بوده و تحت اجازه‌نامه عمومی همگانی گنو عرضه می‌شود. نسخه‌های R بصورت رایگان و برای انواع سیستم‌های عامل (ویندوز، مک و انواع توزیع‌های لینوکس) ارائه شده‌است. هسته اصلی نرم‌افزار R به همراه بیش از ۴۳۰۰ کتابخانه مرتبط با آن، در شبکه comprehensive R archive CRAN (network) در دسترس کاربران می‌باشد. این شبکه که مخزن منابع مرتبط با R می‌باشد، از سرورهای دانشگاه‌های مختلف سراسر جهان برای نگهداری و گرفتن نسخه پشتیبان از این داده‌ها استفاده می‌کند^۱. در میهن عزیز اسلامی ما نیز، دانشگاه فردوسی مشهد یکی از مراکز نگهداری نسخه‌های R می‌باشد^۲.

در حال حاضر اگر چه هیچ شرکت تجاری پشتیبان رسمی R را بر عهده ندارد، اما شبکه پشتیبانی غیررسمی آن از طریق پیام‌های اینترنتی r-help قابل دسترس است. در حال حاضر R نرم‌افزاری کامل و بی‌نقص نیست و از آنجا که آزاد و رایگان هم می‌باشد، از پیشنهادات کاربران برای اصلاح و توسعه آن استقبال می‌کند.

۱-۱ تاریخچه پروژه R

پروژه R در سال ۱۹۹۱ در گروه آمار دانشگاه Auckland کشور نیوزیلند کلید خورد. بنیان‌گذاران این پروژه آقایان Ross Ihaka و Robert Gentleman بودند. وجه تسمیه این زبان نیز ابتدای نام این در نفر می‌باشد (در دپارتمان آمار این دانشگاه این پروژه به "R&R" نیز معروف می‌باشد). در حال حاضر تیمی متشکل از ۱۹ نفر در حال توسعه این طرح می‌باشند.

R برای اولین بار در سال ۱۹۹۳ به طور عمومی معرفی گردید. در ۱۹۹۵ مارتین ماچلر، راس و رابرت را متقاعد نمود تا تحت مجوز گنو، R را به یک نرم‌افزار رایگان تبدیل نمایند. یک سال بعد لیست رایانامه عمومی این پروژه ایجاد گردید (R-help و R-devel) که به کمک آن توسعه‌دهندگان و کاربران در سراسر جهان با یکدیگر مرتبط می‌شدند. در ۱۹۹۷ هسته اصلی توسعه‌دهنده R که شامل برخی از اعضای پروژه S-Plus نیز بودند ایجاد شد. این

^۱ <http://cran.r-project.org/mirrors.html>

^۲ <http://cran.um.ac.ir/>



گروه، وظیفه کنترل و نظارت بر کد برنامه را به عهده داشت. با تلاش‌های توسعه‌دهندگان، در سال ۲۰۰۰ اولین نسخه R ارائه گردید. آخرین نسخه ارائه شده این نرم‌افزار 2.13.1 می‌باشد که در ۲۰۰۱/۷/۸ منتشر شده‌است.



شکل ۱. لوگوی پروژه R

۲-۱ مروری بر امکانات R

R یک مجموعه کامل از امکانات نرم‌افزاری برای کارکردن با داده‌ها و محاسبه و رسم نمودار می‌باشد. از جمله امکانات این مجموعه می‌توان به موارد زیر اشاره کرد:

- زبان برنامه‌نویسی ساده و پیشرفته شامل عبارت‌های شرطی، حلقه و توابع بازگشتی و ...
- امکانات ذخیره، بازیابی و دستکاری داده‌ها
- مجموعه‌ای قوی از عملگرهای محاسباتی آرایه‌ها و ماتریس‌ها
- بسته‌های نرم‌افزاری قدرتمند برای تجزیه و تحلیل آماری
- کتابخانه‌های انجام عملیات داده‌کاوی و یادگیری ماشین مانند دسته‌بندی، خوشه‌بندی، تحلیل شبکه اجتماعی، یادگیری تقویتی و ...
- امکانات گرافیکی برای تجزیه و تحلیل داده‌ها و رسم نمودار
- کتابخانه‌های خاص منظوره برای انجام عملیات تحلیلی در زمینه‌های مختلف علمی
- دارای مستندات فرمت‌بندی شده و منظم برای استفاده از زبان و کتابخانه‌های مرتبط

۳-۱ تحلیل شبکه اجتماعی در R

یکی از مهمترین ابزارهای تحلیل و مصورسازی شبکه اجتماعی در حوزه آکادمیک و تجاری، R می‌باشد. بسته‌های igraph، statnet، sna، egonet، snort و ... از جمله آن‌ها می‌باشد. در این گزارش ما از igraph به عنوان ابزار استفاده می‌کنیم. ساختار داده‌ای igraph هماهنگ با statnet و sna می‌باشد.



۲ نصب R و تعبیه آن در نرم افزار RapidMiner

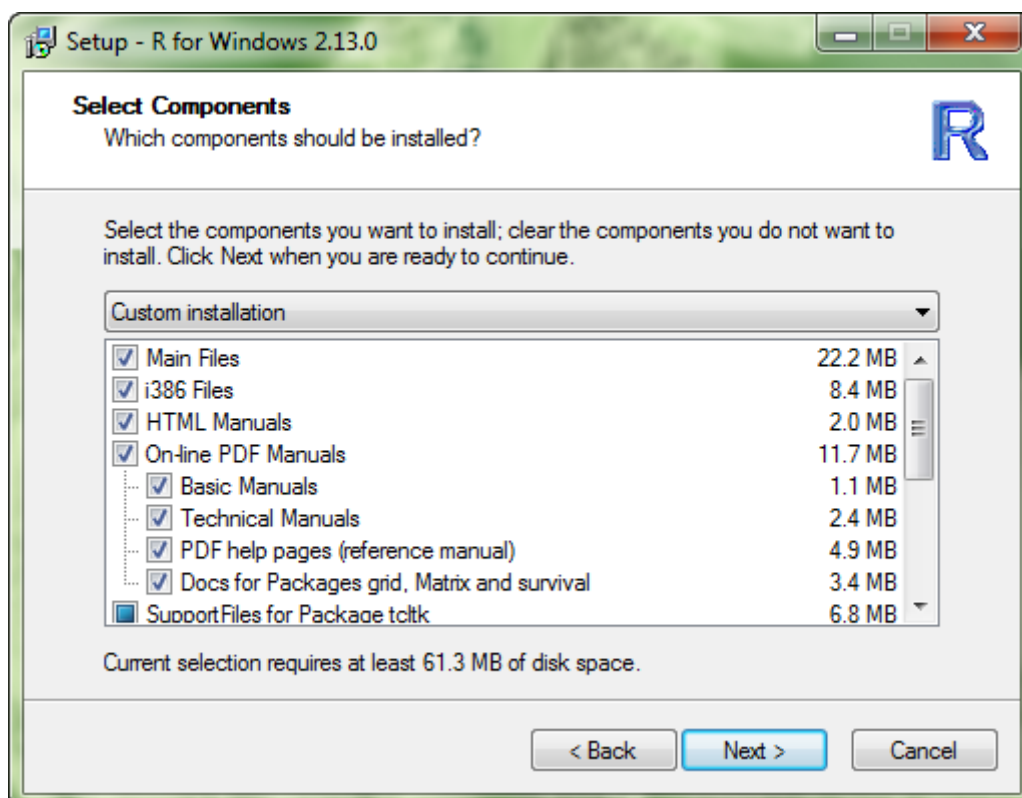
برای استفاده از R در نرم افزار قدرتمند Rapidminer باید ابتدا هسته اصلی R را نصب کنیم و سپس افزونه مربوطه در Rapidminer را فعال و تنظیم نماییم. در ادامه با مراحل این عملیات آشنا خواهیم شد.

۱-۲ مراحل نصب هسته اصلی R

برای نصب هسته اصلی نرم افزار R کافی است که به سایت www.r-project.org مراجعه کرده و به بخش دانلود CRAN بروید. در این صفحه، آدرس های تعدادی از سایت هایی که پروژه R در آن ذخیره می شود قرار دارد. محتوای تمامی این سایت ها یکسان است، بنابراین با انتخاب یکی از آن ها به صفحه دانلود بروید. در این مرحله پس از انتخاب سیستم عامل خود، دو لینک برای شما نمایش داده می شود:

۱. base: هسته اصلی R که برای نصب اولیه به آن نیاز دارید.

۲. contrib: بسته های نرم افزاری که توسط کاربران سراسر جهان ایجاد شده و توسط تیم توسعه دهندگان مدیریت می شود.

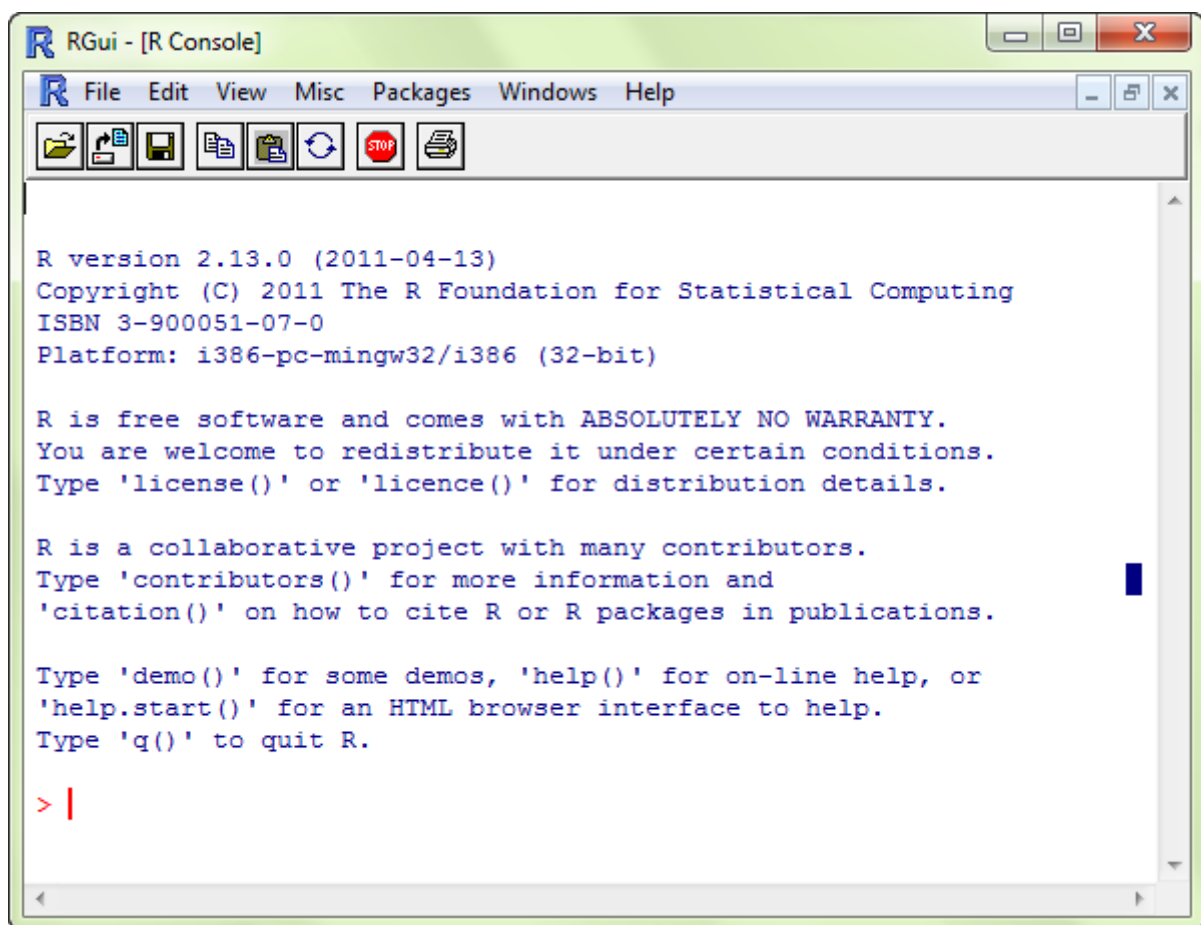


شکل ۲. نمایشی از یکی از مراحل نصب نرم افزار R



از آن‌جا که ما قصد داریم R را برای اولین بار نصب کنیم، روی گزینه base کلیک کرده و جدیدترین نسخه مربوطه را دانلود می‌نماییم. از آن‌جا که نسخه نصب شده توسط نگارنده روی سیستم‌عامل ویندوز می‌باشد، شاید برخی از توضیحات ادامه مورد استفاده کاربران دیگر سیستم‌عامل‌ها نباشد.

نصب R از روی فایل اجرایی بسیار ساده می‌باشد و تنها کافی است با کلیک بر روی فایل اجرایی مراحل نصب را از روی Wizard انجام دهید. در مراحل نصب باید اطلاعاتی از قبیل پوشه نصب، کامپوننت‌هایی که باید نصب شود (شکل ۲) و ... را وارد نمایید. پس از نصب شما می‌توانید با استفاده از یک کنسول (شکل ۳) و نوشتن اسکریپت‌های مربوطه به زبان R از امکانات این نرم‌افزار استفاده نمایید.



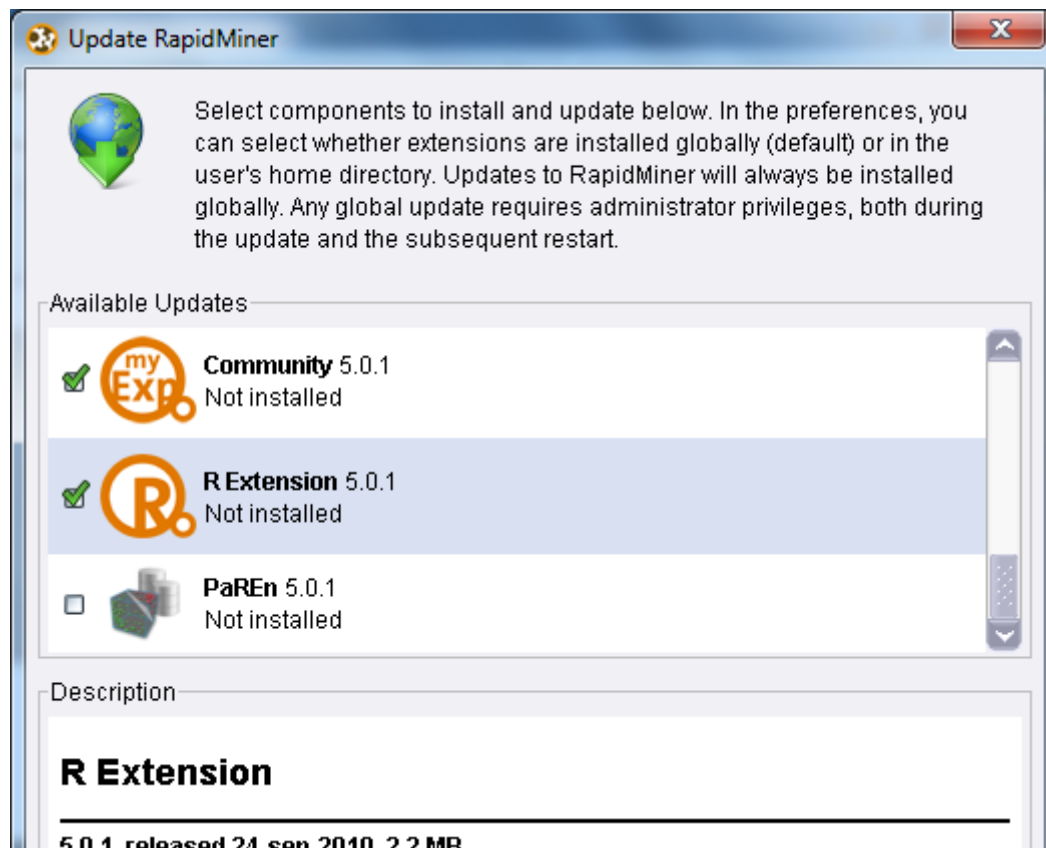
شکل ۳. محیط کنسول نرم‌افزار R

۲-۲ نصب افزونه R در Rapidminer

نرم‌افزار Rapidminer از نسخه ۵ به بعد، افزونه R را به امکانات خود افزود. به کمک این افزونه شما می‌توانید از محیط Rapidminer برای نوشتن اسکریپت‌های R و ایجاد ماژول‌های قابل استفاده در دیگر بخش‌ها، استفاده کنید. برای نصب افزونه R مرحله ابتدایی بدین صورت است که از منوی help، گزینه Update Rapidminer را



انتخاب می‌کنید و سپس بر روی گزینه install کلیک نمایید (شکل ۴). بعد از انجام این کار، افزونه مورد نظر از اینترنت دانلود شده و نصب می‌گردد. برای انجام بقیه مراحل باید پس انجام تنظیماتی که در ادامه بیان می‌گردد، Rapidminer را مجدداً راه‌اندازی کنید.



شکل ۴. روشن کردن افزونه R در Rapidminer

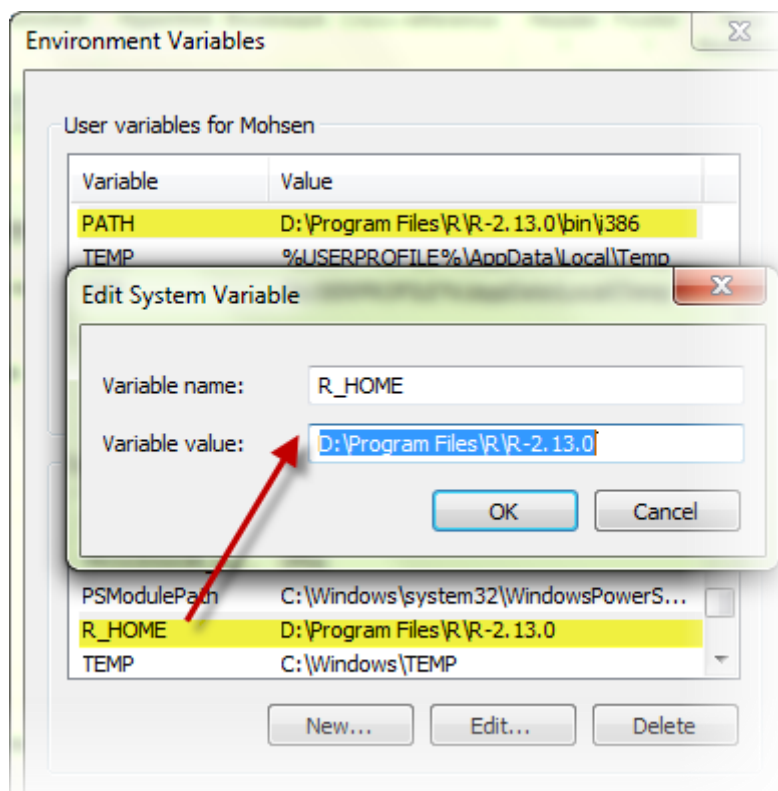
بقیه مراحل نصب R، به تنظیم بخش‌های مختلف مربوط می‌شود که به صورت زیر انجام می‌پذیرد:

۱. بعد از راه‌اندازی نسخه مناسب R، شما باید بسته نرم‌افزاری rJava را از CRAN دانلود و نصب نمایید. بدین منظور به محیط کنسول R بروید و در محل اسکریپت نویسی آن عبارت `install.packages("rJava")` را تایپ نمایید. در صورتی که به اینترنت متصل باشید این بسته به صورت خودکار دانلود و نصب می‌گردد.
۲. عبارت `libPaths()` را در کنسول تایپ کنید. در یکی از پوشه‌های نمایش داده شده، بسته rJava نصب شده‌است. این آدرس را برای هنگام باز کردن Rapidminer به خاطر بسپارید.



۳. برای این که مطمئن شوید Rapidminer می تواند کتابخانه های R را پیدا کند، باید متغیرهای محیطی^۱ سیستم عامل خود را تنظیم نمایید. دقت کنید که اسم این متغیر حساس به حروف کوچک و بزرگ می باشد

- متغیر PATH را با مقدار <R installation directory>\bin وارد نمایید. به عنوان نمونه این مقدار باید مشابه C:\Program Files\Rapid-I\RapidMiner5\jre\bin باشد. این پوشه باید شامل فایل های dll^۲ نرم افزار R و کتابخانه های مرتبط باشد. دقت کنید که این پوشه باید شامل فایل R.dll باشد.
- چنانچه در متغیرهای محیطی سیستم عامل شما JAVA_HOME ی که به پوشه نصب جاوا اشاره کند، وجود ندارد؛ باید این متغیر را نیز اضافه نمایید. در صورتی که جاوا نیز روی رایانه شما نصب نمی باشد، مقدار <RapidMiner installation directory>\jre را برای این متغیر وارد نمایید. دقت کنید که جاوای نصب شده روی رایانه شما باید از لحاظ ۳۲ یا ۶۴ بیت بودن، مطابق R و Rapidminer باشد.



شکل ۵. اضافه کردن متغیرهای محیطی مرتبط به تنظیمات افزونه R

^۱ Environment Variable

^۲ dynamic link library



• متغیر R_HOME نیز باید به متغیرهای محیطی اضافه گردد. مقدار این به پوشه نسخه

نصب شده R اشاره کند. (به عنوان مثال R-2.13.0\Program Files\R) شکل ۵

چگونگی تنظیم این متغیر را نشان می دهد.

۴. در این مرحله شما می توانید نرم افزار Rapidminer را باز کنید. قبل از شروع به کار برنامه، صفحه ای

به شما نمایش داده می شود که مراحل نصب روی آن نوشته شده است. روی دکمه Next کنید. در

صفحه بعدی از شما خواسته می شود آدرس کتابخانه jri با rJava نصب کرده اید را مشخص نمایید.

بدین منظور آدرس libPaths را از مرحله ۲ به خاطر بیاورید و به <libpath directory>/rJava/jri/ بروید. در این پوشه فایل jri.dll را انتخاب نمایید.

۵. اگر فایل مشخص شده را انتخاب کرده باشید، برنامه بسته می شود. دوباره نرم افزار Rapidminer را

اجرا نمایید. چنانچه افزونه با موفقیت نصب شده باشد، با هیچ دستور خطایی در هنگام اجرا برخورد

نخواهید کرد. در غیر این صورت حتما یکی از مراحل فوق را به درستی انجام نداده اید.

در صورت نصب صحیح افزونه، باید پس از بالا آمدن برنامه باید علامت R در نوار ابزار فوقانی (مطابق شکل ۶)

اضافه شده باشد.



شکل ۶. اضافه شدن آیکن R در نوار ابزار فوقانی

۳-۲ محیط اسکریپت نویسی افزونه R

محیط R در نرم افزار Rapidminer به همان صورت اسکریپت نویسی تعبیه شده است [۶]. شکل ۷ نمای کلی

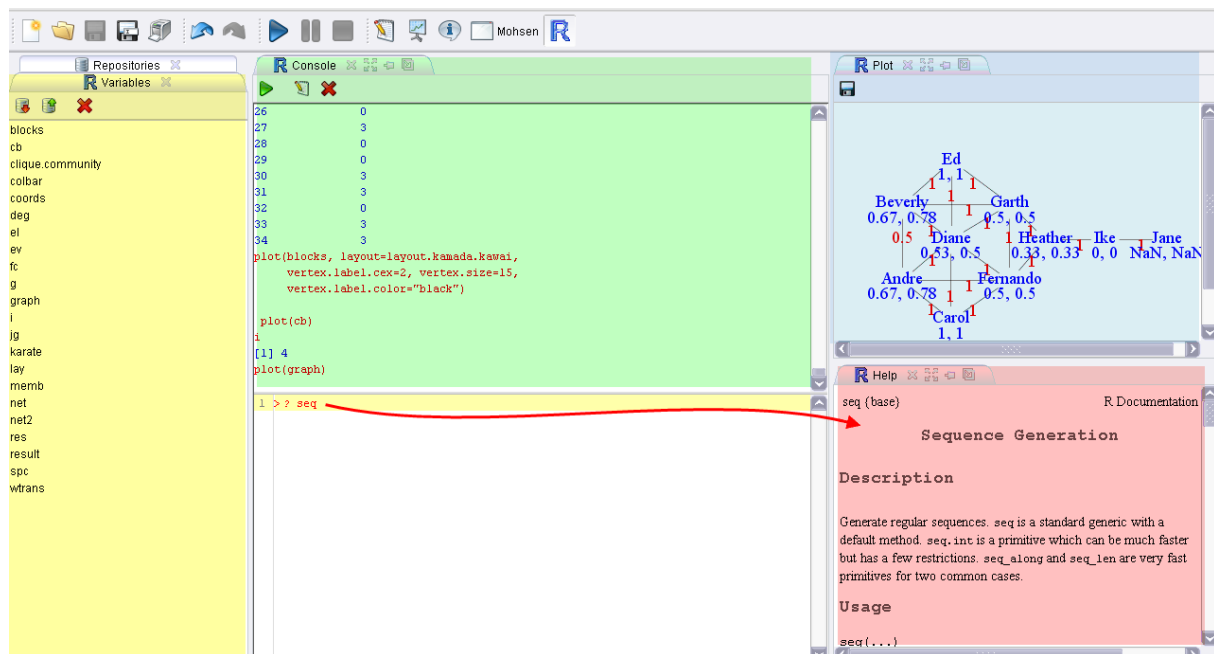
آن نشان داده شده است. در قسمت وسط پایین، شما امکان نوشتن اسکریپت های مورد نظر را دارید. با زدن Enter

این اسکریپت ها اجرا می شوند و نتایج آن در کنسول بالای آن (نمایش داده شده با رنگ سبز) نشان داده می شوند.

متغیرهایی که در حین کار با برنامه ایجاد می کنید در پنل سمت چپ نمایش داده می شوند. چنانچه شما نموداری را



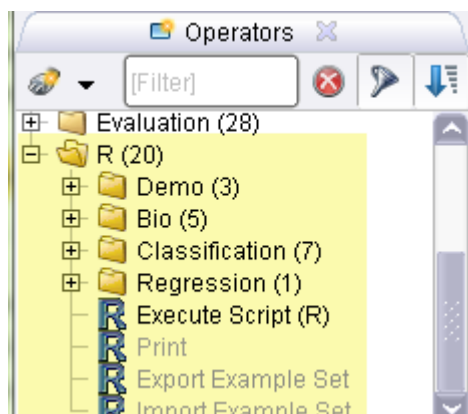
رسم کنید، در پنل نشان‌داده شده با رنگ آبی نمایش داده می‌شود. شکل رسم شده را می‌توانید با دکمه گوشه پنل در انواع فرمت‌های تصویر ذخیره نمایید. با دوبار کلیک روی عنوان هر کدام از این پنل‌ها، آن‌ها به حالت تمام صفحه و برعکس در خواهند آمد. آخرین پنلی که در این صفحه وجود دارد، پنل راهنماست. در صورتی که شما می‌خواهید به راهنمای دستور X دسترسی پیدا کنید، می‌توانید به کمک اسکریپت **X ?** این کار را انجام دهید.



شکل ۷. نمای کلی افزونه R در Rapidminer

۲-۴ استفاده از R در Rapidminer

برای استفاده از R در محیط طراحی (Design)، مجموعه عملگرهای مخصوصی در محیط Rapidminer تعبیه شده‌است. شکل ۸ این عملگرها را نشان می‌دهد. برخی از این عملگرها خاص منظوره و برخی عام منظوره می‌باشند.



شکل ۸. عملگرهای R در نوار ابزار



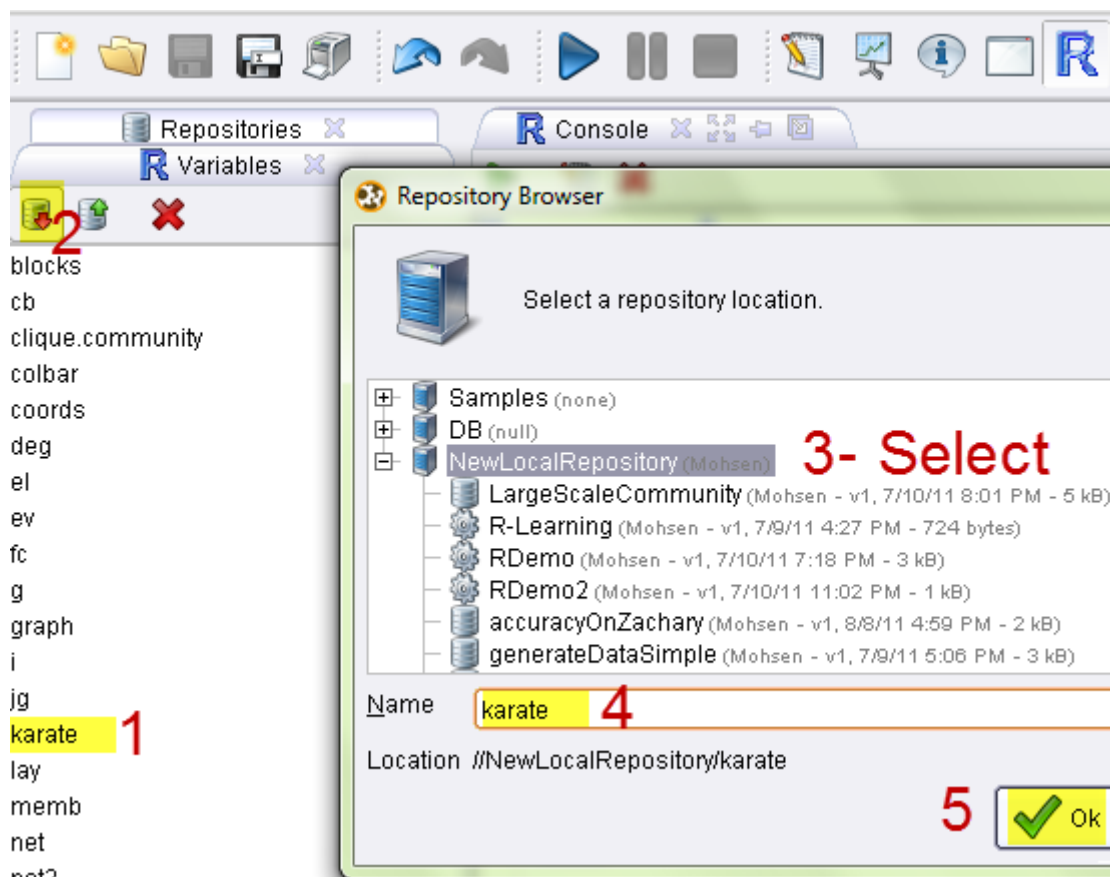
در Rapidminer چند محیط کاری (Perspective) وجود دارد. مهم ترین آن ها عبارتند از:

۱. محیط طراحی: فضای کار ایجاد جریان کار داده کاوی.
۲. محیط نتایج: فضای کار مشاهده نتایج بدست آمده.
۳. محیط R: فضای کار اسکریپت نویسی به زبان R.



شکل ۹. آیکن فضای کاری های مختلف در Rapidminer

برای این که بتوانیم داده های R را در محیط طراحی وارد نماییم، ابتدا باید در محیط R آن ها را در عنوان یک Repository ذخیره نماییم. به عنوان نمونه، در شکل ۱۰ ذخیره مجموعه داده کاراته نشان داده شده است.



شکل ۱۰. مراحل ذخیره داده های R در Repository

حال به محیط طراحی می رویم، و با کشیدن (Drag کردن) منبع داده کاراته به نمای Process، آن را در فرآیند داده کاوی وارد می نماییم. به طور پیش فرض عملگر بازیابی (Retrive) برای بارگذاری مجموعه داده، به فرآیند اصلی اضافه می گردد و تنها پارامتر آن - که Repository Entry است،- به صورت خودکار پر می شود.



خروجی عملگر قبلی را به ورودی یک عملگر Execute Script (R) وصل کنید. این عملگر، یک عملگر عام منظوره است و اسکریپت های دلخواه را اجرا می نماید. برای تنظیم این عملگر باید سه پارامتر اسکریپت، نام ورودی ها و خروجی ها را وارد نمایید. ما قصد داریم روی گراف کاراته، ساختاریابی اجتماع انجام دهیم. بنابراین کد عملیات به صورت زیر است. karate، همان مجموعه داده ورودی است.

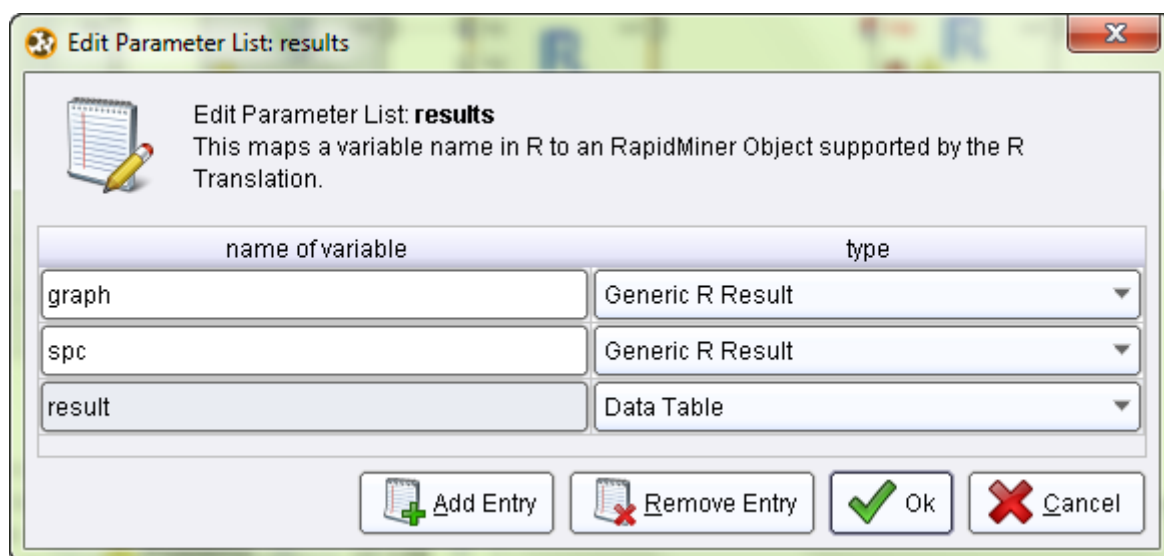
```
library('igraph')
graph <- karate
graph <- simplify(graph)
spc <- spinglass.community(graph, spins=20)
result <- as.data.frame(spc$membership)
```

متغیرهای مشخص شده با رنگ قرمز، خروجی های این اسکریپت هستند. کلا دو نوع خروجی برای عملگر فوق وجود دارد:

۱. Generic R Result: که شامل انواع متغیرها، وکتورها، آرایه ها و ... R می شود. این خروجی ها تنها قابل پردازش برای عملگرهای افزونه R هستند.

۲. Data Table: این خروجی معادل data frame در R است. حتما باید جنس این نوع خروجی در اسکریپت، به data frame تبدیل شده باشد. چنانچه بخواهید نتایج اسکریپت برای عملگرهای Rapidminer قابل فهم باشد و به صورت جدول نمایش داده شود، باید از این نوع استفاده کنید.

توجه داشته باشید که ترتیب تعریف خروجی ها در این عملگر مهم است. به طوری که پرت های خروجی به همان ترتیب تعریف شده در پارامتر Result، پر می شوند.



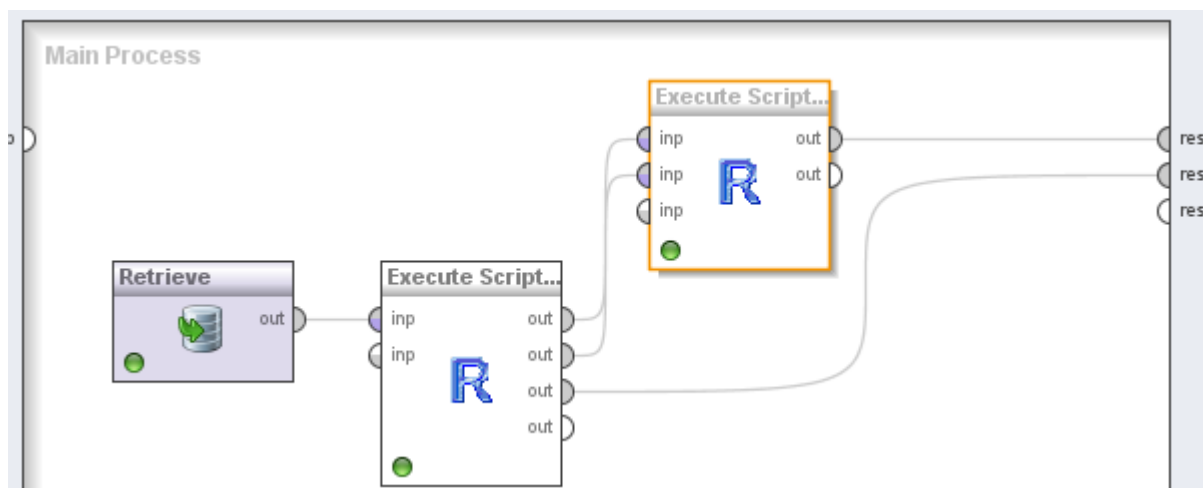
شکل ۱۱. نحوه تعریف خروجی عملگر Execute Script (R)

از آنجا که قصد داریم نتایج ساختاریابی پیشین را رسم و در Clipboard به صورت تصویر ضبط نماییم، یک عملگر دیگر ایجاد می‌کنیم که این عملیات را انجام دهد.

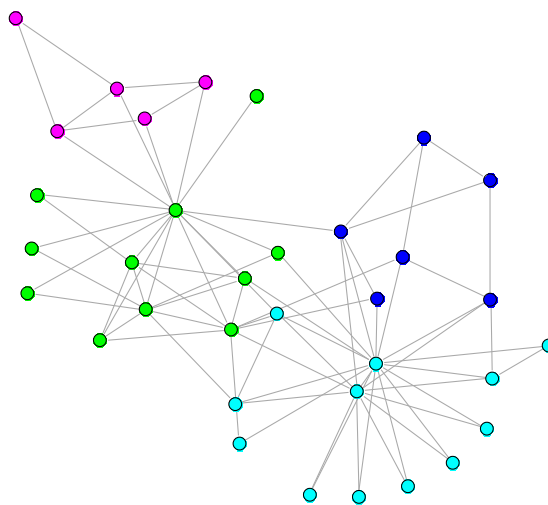
```
library('igraph')
res <- as.data.frame(spc$size)

x11()
plot(graph, layout=layout.kamada.kawai(graph), vertex.size=5,
vertex.label=NA, vertex.color=spc$membership+1, asp=FALSE)
savePlot(filename = "clipboard", type = "png")
```

مدل فرآیند کلی، مراحل بالا در شکل ۱۲ و نتیجه اجرای ساختاریابی در شکل ۱۳ نمایش داده شده است. البته باید توجه داشت، افزونه فعلی R، هنوز دارای مشکلات بسیاری است و اجرای برخی از دستورات با مشکلاتی مواجه است. از جمله آن می‌توان به بسته شدن پنجره نتایج رسم شده با اتمام اجرای فرآیند، اشاره نمود.



شکل ۱۲. مدل فرآیند کلی مراحل تحلیل گراف کلپ کاراته



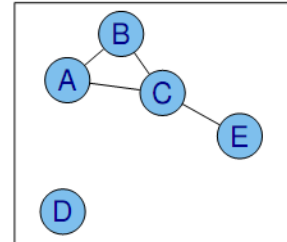
شکل ۱۳. نتیجه ساختاریابی انجام شده در فرآیند فوق

۳ مدلسازی گراف

از آن جا که اساس کار ما بر کتابخانه igraph استوار است، در ادامه نحوه مدلسازی گراف در این کتابخانه را بررسی می‌نماییم. گراف یک ارتباط باینری (یال‌ها) بین اعضای یک مجموعه (گره‌ها) است. اگر دو عضوی که با هم رابطه دارند بدون ترتیب باشند، آن‌گاه گراف بدون جهت است (شکل ۱۴) و گرنه گراف جهت دار می‌باشد (شکل ۱۵).

$\text{vertices} = \{A, B, C, D, E\}$

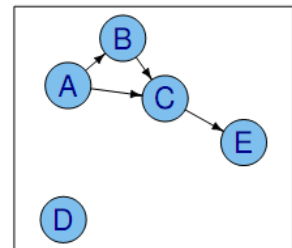
$\text{edges} = (\{A, B\}, \{A, C\}, \{B, C\}, \{C, E\}).$



شکل ۱۴. گراف بدون جهت

$\text{vertices} = \{A, B, C, D, E\}$

$\text{edges} = ((A, B), (A, C), (B, C), (C, E)).$



شکل ۱۵. گراف جهت دار

همواره گره‌ها از صفر شماره‌گذاری می‌شود و به صورت متناوب تا $n-1$ ادامه می‌یابد. به عنوان نمونه در گراف شکل ۱۴، $A=0, B=1, C=2$ و ... می‌باشد. تعریف گراف‌های شکل ۱۴ و شکل ۱۵ در R به صورت زیر می‌باشد: توجه داشته باشید که برای استفاده از کتابخانه‌های igraph ابتدا باید آن را با `library(igraph)` بارگذاری کنید.

```
## Load the igraph package
library(igraph)

## Create a small graph, A->B, A->C, B->C, C->E, D
## A=0, B=1, C=2, D=3, E=4
g <- graph( c(0,1, 0,2, 1,2, 2,4), n=5 )

## Print a graph to the screen
g

## Create an undirected graph as well
## A--B, A--C, B--C, C--E, D
g2 <- graph( c(0,1, 0,2, 1,2, 2,4), n=5, dir=FALSE )
```



g2

برای این که جنس هر یک از متغیرها را مشخص کنید، می‌توان از دستور class استفاده نمود.

```
## How to decide what kind of object a variable refers to
class(g2)
class(1)
class("foobar")
```

نتیجه اجرای دستورات بالا به صورت زیر خواهد بود:

```
[1] "igraph"
[1] "numeric"
[1] "character"
```

به کمک تابع `is.igraph()` می‌توانید از گراف بودن یک شی اطمینان حاصل نمایید:

```
## Is this object an igraph graph?
is.igraph(g)
is.igraph(1:10)
```

چنانچه یک شی گراف باشد، دستورات زیر اطلاعات بیشتری از ساختار گراف و تعداد گره و یال‌ها ارائه می‌کنند:

```
## Summary, number of vertices, edges
summary(g)
vcount(g)
ecount(g)
```

دستور جهت‌دار بودن گراف را مشخص می‌کند. این تابع تنها برای گراف تعریف شده و برای دیگر اشیا خطا می‌دهد. شما می‌توانید هر یک از گراف‌های جهت‌دار و بدون جهت را به یکدیگر تبدیل نمایید. چنانچه یک گراف جهت‌دار، بدون جهت شود تمامی یال‌ها فقط به یال بدون جهت تبدیل می‌گردد. چنانچه ارتباط یک یال دوطرفه باشد نیز دو یال به یک یال تبدیل می‌شود. این درحالی است که در تبدیل بدون جهت به جهت‌دار هر یال به دو یال رفت و برگشت مبدل می‌گردد.

```
## Is the graph directed?
is.directed(g)
is.directed(g2)
is.directed(1:10)

## Convert from directed to undirected
as.undirected(g)

## And back
as.directed(as.undirected(g))
```

در یک گراف می‌توان چند یال بین دو گره تعریف کرد. مثلاً در گراف $E = ((AB); (AB); (AC); (BC))$:

```
## Multiple edges
g <- graph( c(0,1, 0,1, 0,2, 1,2 ), n=5 )
```



تشخیص ساده بودن یا چند یالی بودن گراف با توابع `is.simple(g)` و `is.multiple(g)` امکان پذیر است. همچنین اگر بخواهیم یک گراف چندیاله را به گراف ساده تبدیل نماییم، می توان از دستور زیر استفاده نمود.

```
## Remove multiple edges
g <- simplify(g)
is.simple(g)
```

یکی دیگر از قابلیت هایی که امکان تعریف آن برای گراف وجود دارد، افزودن طوقه می باشد. با `simplify` کردن یک گراف طوقه ها نیز از بین خواهد رفت.

```
## Loop edges
g <- graph( c(0,0,0,1, 0,2, 1,2, 3,4), n=5 )
is.simple(g)
is.loop(g)

## Remove loop edges
g <- simplify(g)
is.simple(g)
```

یکی از قابلیت های `igraph` برای شی گراف، امکان نامگذاری گره ها می باشد. در اسکریپت زیر ابتدا یک گراف ۱۰ گره ای به صورت حلقوی (هر گره تنها با یال قبلی و بعدی رابطه دارد. همچنین یال ابتدا و انتهای نیز به هم متصل می گردند.) ایجاد می شود. اسامی گره ها به ترتیب ۱۰ حرف ابتدایی الفبای انگلیسی تعیین می گردد.

```
## Naming vertices
g <- graph.ring(10)
V(g)$name <- letters[1:10]
V(g)$name
g
print(g, v=T)
```

دو نکته جانبی در کد بالا وجود دارد. اول این که با افزودن `$name` یک صفت به `V(g)` اضافه کردیم که در ادامه قابل دسترسی است. افزودن صفات دیگر هم مشابه همین فرآیند است. نکته دوم تابع `print` است که به ما اجازه می دهد تا اطلاعاتی که از گراف در کنسول چاپ می شود را مدیریت نماییم. مثلاً در کد بالا گره ها را نیز به مواردی که نشان داده می شود اضافه نمودیم.

۳-۱ ساخت گراف

یکی از ابزارهای `igraph` برای ایجاد گراف از روی نام یال ها `graph.formula` می باشد. به کمک این تابع تنها شما کافی است مسیرهای مختلف بین یال ها را در گراف مشخص نمایید. کد زیر ایجاد یک گراف غیرجهت دار را



نشان می‌دهد. عملگر "-" (عملگر یال) نشان‌دهنده وجود ارتباط بین دو گره می‌باشد. به کمک این تابع نیازی به تعریف مجموعه گره‌ها به صورت جداگانه وجود ندارد.

```
## A simple undirected graph
g <- graph.formula(Alice-Bob-Cecil-Alice,
                   Daniel-Cecil-Eugene, Cecil-Gordon )
```

عملگر ":" برای تعریف یک مجموعه گره استفاده می‌شود. اگر دو مجموعه با عملگر یال به هم متصل باشند، تمامی گره‌های دو مجموعه با هم ارتباط دارند. بنابراین این عملگر نسبت به "-" تقدم دارد. به عنوان نمونه در اسکرپت زیر، Cecil:Daniel-Eugene:Gordon مشخص‌کننده یال‌های Cecil-Eugene، Cecil-Gordon و Daniel-Gordon را مشخص می‌دهد.

```
## Another undirected graph, ":" notation
g2 <- graph.formula(Alice-Bob:Cecil:Daniel,
                   Cecil:Daniel-Eugene:Gordon )
```

برای تعریف گراف جهت‌دار در "+" استفاده می‌نماییم. این عملگر نشان‌دهنده جهت می‌باشد. توجه کنید که "+++" یک یال دو طرفه است، نه دو یال یک طرفه. زیاد بودن تعداد "-"ها برای نمایش یک یال بدون معنی است.

```
## A directed graph
g3 <- graph.formula(Alice +++ Bob --+ Cecil
                   +-- Daniel, Eugene --+ Gordon:Helen )
## "Arrows" can be arbitrarily long
g5 <- graph.formula( Alice +-----+ Bob )
```

همچنین برخی از گره‌ها می‌توانند به صورت ایزوله تعریف شوند.

```
## A graph with isolate vertices
g4 <- graph.formula(Alice -- Bob -- Daniel,
                   Cecil:Gordon, Helen )
```

از دیگر راه‌های ایجاد گراف جهت‌دار، استفاده از تابع cbind است. به کمک این تابع می‌توان با الصاق دو از مجموعه گره‌ها که به تعداد مساوی گره دارند، مجموعه‌ای از زوج مرتب‌ها به وجود آورد. زوج مرتب‌هایی که از ترکیب اعضای نظیر به وجود آمده‌اند. بعد از ایجاد این مجموعه مرکب به کمک تابع graph.edgelist می‌توان گرافی مورد نظر را ایجاد نمود.

```
## From edge lists
el <- cbind( c(0, 0, 1, 2),
            c(1, 2, 2, 4) )
g <- graph.edgelist(el)
g
```

فرآیند فوق را به جای شماره گره، می‌توان با نام‌های سمبلیک گره‌ها نیز انجام داد.



```
## Symbolic edge lists
el <- cbind( c("Alice", "Alice", "Bob", "Cecil"),
             c("Bob", "Cecil", "Cecil", "Ed") )
g <- graph.edgelist(el)
g
summary(g)
```

یک روش متداول دیگر برای ایجاد گراف، استفاده ماتریس مجاورت است. در زیر یک گراف ۱۰ در ۱۰ با یال‌های تصادفی ایجاد شده‌است.

```
## Adjacency matrices
A <- matrix(sample(0:1, 100, rep=TRUE), 10, 10)
g <- graph.adjacency(A)
```

۲-۳ ایجاد گراف از روی فایل بیرونی

تمام گراف‌هایی که از روی داده‌های دنیای واقع ایجاد می‌شوند، در فایل‌های متنی ذخیره می‌شوند. یکی از نیازهای اصلی کاربران، import کردن این داده‌ها در R است. در ادامه فرآیند ایجاد گراف از روی data frame نشان داده شده‌است. برای خواندن این اطلاعات ابتدا نیاز داریم از کدی که هنوز در igraph تعبیه نشده استفاده کنیم.

```
## We need some extra code, not in the current version yet
source("http://cneurocv.s.rmki.kfki.hu/igraph/plus.R")
```

سپس به ترتیب گره‌ها و یال‌های مورد نظر را بارگذاری نماییم.

```
## Read the comma-separated value file
vertices <- read.csv("http://cneurocv.s.rmki.kfki.hu/igraph/judicial.csv")

## Read the edges, just a table, space separated
edges <- read.table("http://cneurocv.s.rmki.kfki.hu/igraph/allcites.txt")
```

با ترکیب یال‌ها و گره‌ها به کمک graph.data.frame می‌توان گراف را ایجاد نمود.

```
## And create the graph
jg <- graph.data.frame(edges, vertices=vertices, dir=TRUE)
summary(jg)
```

چنانچه بخواهید می‌توانید این گراف را برای استفاده‌های بعدی در یک فایل ذخیره نمایید. با روشن کردن قابلیت فشرده‌سازی، حجم این فایل نیز کاهش می‌یابد.

```
## Save it for future use
save(jg, file="judicial.Rdata.gz", compress=TRUE)
```



۳-۳ دستکاری در ساختار گراف

از عملیات شایع روی گراف اضافه و کم کردن گره و یال‌های آن می‌باشد. در این قسمت گراف g2 که قبلاً آن را ایجاد نمودیم دستکاری می‌نماییم. در ابتدا گره اول آن را حذف می‌کنیم.

```
## Remove alice
g3 <- delete.vertices(g2, 0)
```

اضافه کردن گره به کمک تابع add.vertices انجام می‌شود. ورودی‌های این تابع گراف و تعداد گره‌ای که باید اضافه شود است. کد جدیدی که در اسکریپت زیر اضافه شده‌است، igrph.par می‌باشد. به کمک این تابع می‌توان پارامترهای igraph را تنظیم نمود. مثلاً ما چاپ صفات گره‌ها را فعال کردیم تا اسامی نمایش داده شود.

```
## Add three new vertices
g4 <- add.vertices(g3, 3)
igraph.par("print.vertex.attributes", TRUE)
g4
```

add.vertices این امکان را نیز به ما می‌دهد که هنگام اضافه کردن گره اسامی گره‌ها را نیز مشخص نماییم. ورودی اسامی حتماً باید به صورت لیست باشد، در مثال زیر ما صفت name را به گره‌ها اضافه می‌نماییم.

```
## Add three new vertices, with names this time
g4 <- add.vertices(g3, 3, attr=list(name=c("Ha", "Ik", "Ja")))
g4
```

مشابه گره، اضافه کردن یال نیز به راحتی با تابع add.edges امکان‌پذیر است.

```
## Add some edges as well
g4 <- add.edges(g4, c(6,7, 5,7))
g4
```

۴-۳ استخراج همسایگی یال‌ها

آموزش این بخش روی گراف g2 انجام می‌شود که با اسکریپت زیر آن را ایجاد می‌نماییم. اگر بخواهیم شناسه یک یال را بیابیم، دستور as.vector این قابلیت را برای ما فراهم می‌کند.

```
## Create directed example graph
g2 <- graph.formula(Alice -+ Bob:Cecil:Daniel,
                   Cecil:Daniel ++ Eugene:Gordon )
print(g2, v=T)

## Get the id of the edge
as.vector(E(g2, P=c(0,1)))
```



در بسیاری از الگوریتم‌ها ما نیاز داریم به یال‌های مجاور یک گره دسترسی پیدا کنیم. تابع `adj()` این قابلیت را در اختیار ما قرار می‌دهد. ورودی این تابع یک یا چند گره می‌باشد.

```
## All adjacent edges of a vertex
E(g2)[ adj(2) ]

## Or multiple vertices
E(g2)[ adj(c(2,0)) ]
```

چنانچه گرافی جهت‌دار باشد، می‌توان بین یال‌های ورودی و خروجی از یک گره تمایز قائل شد و به صورت جداگانه به هر یک دسترسی داشت.

```
## Outgoing edges
E(g2)[ from(2) ]

## Incoming edges
E(g2)[ to(2) ]
```

یکی از قابلیت‌های خوب `igraph` این است که ما می‌توانیم یال‌هایی که بین یک مسیر از گره‌ها قرار دارند را استخراج نماییم. البته این قابلیت تنها هنگامی قابل استفاده است که یک مسیر کامل را در ورودی تابع قرار دهیم. چنانچه یکی از گره‌های مسیر ذکر نشود، با خطا مواجه می‌شویم.

```
## Edges along a path
E(g2, path=c(0,3,4))
```

نتیجه دستور بالا به صورت زیر خواهد بود:

Edge sequence:

```
[2] Alice -> Daniel
[8] Daniel -> Eugene
```

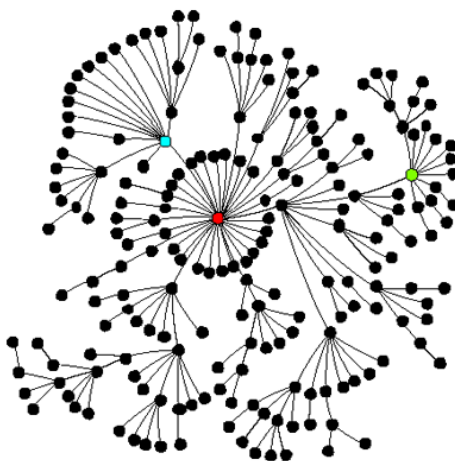
۳-۵ رسم گراف

برای رسم گراف از سه تابع زیر می‌توان استفاده نمود:

۱. `plot`، رسم ساده، دوبعدی و غیر تعاملی گراف است. این تابع پیاده‌سازی تابع عمومی `plot` می‌باشد.
۲. `tkplot`، تابعی برای رسم دوبعدی و تعاملی گراف می‌باشد که از کتابخانه `tcltk` استفاده می‌کند. این تابع برای رسم گراف‌هایی تا اندازه متوسط -حداکثر چند هزار گره- استفاده می‌شود، زیرا مصرف منابع آن زیاد است. به کمک `tkplot` می‌توان برخی از ویژگی‌های گراف را در زمان اجرا تغییر داد.
۳. `rglplot`: یک تابع آزمایشی برای رسم سه‌بعدی توابع است که از `OpenGL` استفاده می‌نماید. در گراف رسم‌شده می‌توان عملیات بزرگنمایی، چرخش و انتقال را انجام داد، اما جای گره‌ها ثابت است.



در ادامه برای رسم، ابتدا یک شبکه بی‌مقیاس (scale-free) می‌سازیم. شبکه‌های بی‌مقیاس که توسط آلبرت و باراباسی گسترش یافتند [۷]، با فرآیند اتصال ترجیحی رشد می‌کنند. شکل ۱۶ یک شبکه بی‌مقیاس اتصال‌های تازه به طور ترجیحی به سمت گره‌های «قطب» (مرکزی-hub) جهت‌گیری می‌کنند.



شکل ۱۶. نمونه‌ای از یک شبکه بی‌مقیاس. قطب‌ها به صورت رنگی نشان داده شده‌اند.

شبکه‌های بی‌مقیاس عدد «جهش» (hops) متوسط کوچکی برای رفتن از یک گره در شبکه به گره دیگر دارند. این مطلب با زمان صرف شده برای ارسال یک پیام از عرض شبکه در ارتباط است. شبکه‌های بی‌مقیاس به دلیل اینکه در طی یک حمله عمدی به گره‌های مرکزی متلاشی می‌شوند، طراحی خوبی برای شبکه‌های مخابراتی به شمار نمی‌روند. به هر حال، آن‌ها کاملاً به نقص‌های تصادفی مقاومت می‌کنند. در سطح TCP/IP (اما نه در سطح فیزیکی) اینترنت نیز یک شبکه بی‌مقیاس است. برای نمونه، google.com به عنوان «گره قطب» اصلی عمل می‌کند. این ساختار منطقی بی‌مقیاس اینترنت، گسترش ویروس‌های کامپیوتری را تقویت می‌کند.

در اسکرپت زیر تابع barabasi.game یک شبکه بی‌مقیاس یک اتصاله با ۱۰۰ گره ایجاد می‌کند. یک اتصاله گراف بودن به این معناست که هر گره جدید به یک گره موجود متصل می‌شود.

```
## Create a scale-free network
g <- barabasi.game(100, m=1)
g <- simplify(g)
```

در این قسمت ابتدا آرایش قرار گیری گره‌ها و یال‌ها را در رسم مشخص می‌کنیم. آرایشی که به عنوان پیش‌فرض در این کد تنظیم شده، آرایش معروف Fruchterman-Reingold می‌باشد [۸]. برای رسم از تابع plot استفاده شده است.

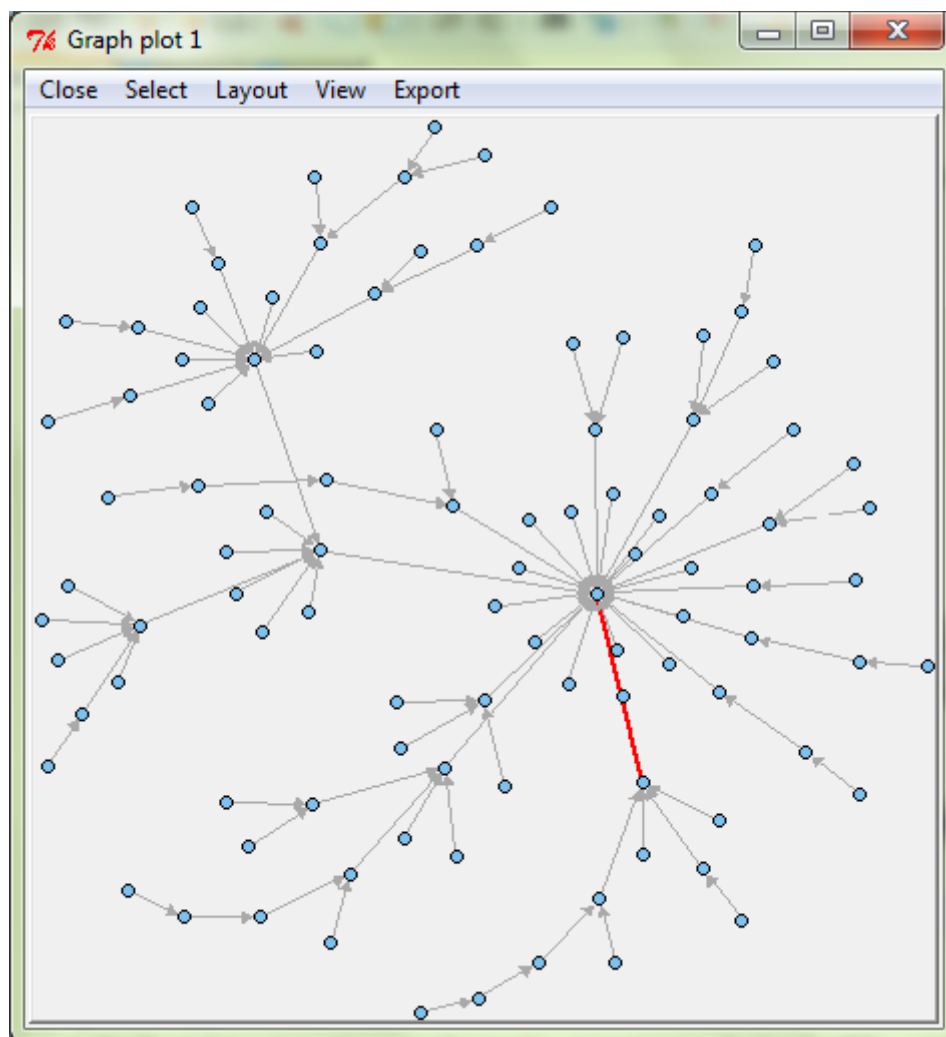
```
## simple plot
igraph.par("plot.layout", layout.fruchterman.reingold)
plot(g, vertex.size=3, vertex.label=NA, edge.arrow.size=0.6)
```



در این بخش از tkplot برای ساخت یک شمای تعاملی از گراف بهره گرفته شده است.

```
## interactive
id <- tkplot(g, vertex.size=3, vertex.label=NA,
edge.arrow.size=0.6)
coords <- tkplot.getcoords(id)
```

شکل زیر، نتیجه اسکرپیت فوق را نمایش می دهد.



شکل ۱۷. گراف رسم شده در محیط tkplot

در این بخش شمای ۳ بعدی گراف فوق را به کمک rgplot رسم شده است.

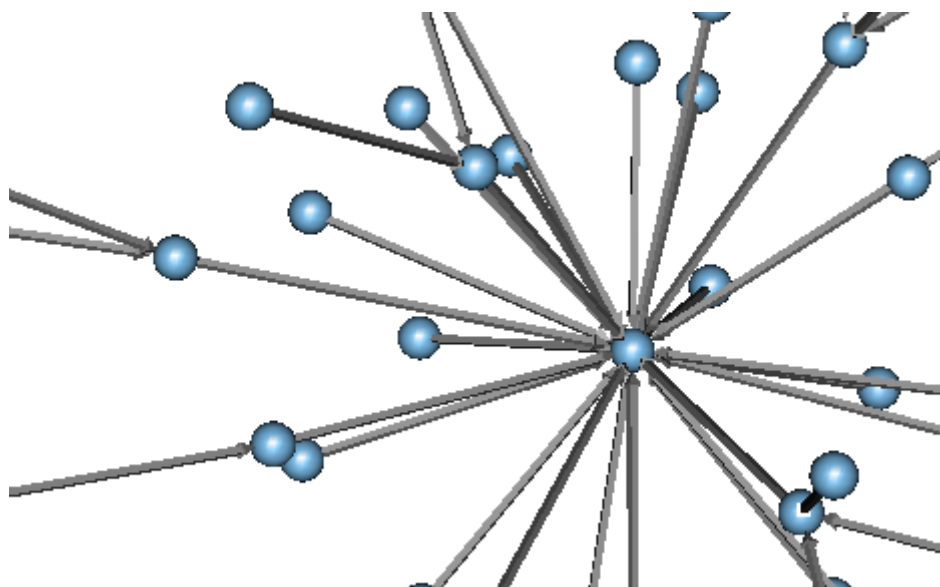
```
## 3D
open3d()
rglplot(g, vertex.size=3, vertex.label=NA,
edge.arrow.size=0.6)
```

در کد زیر رسم گراف بالا با آرایش کامادا-کاوی انجام شده است [۹]. توجه کنید که $\text{dim}=3$ سه بعدی بودن

آرایش گراف را نشان می دهد.




```
## A bit better 3D
coords <- layout.kamada.kawai(g, dim=3)
open3d()
rglplot(g, vertex.size=3, vertex.label=NA,
edge.arrow.size=0.6, layout=coords)
```



شکل ۱۸. گراف سه بعدی رسم شده با rglplot

۳-۶ دسترسی و تغییر صفات گراف، گره‌ها و یال‌ها

گراف، گره‌ها و یال‌ها هر کدام می‌توانند صفات جداگانه‌ای داشته‌باشند که ویژگی‌های آن‌ها را مشخص نمایند. در این قسمت از مجموعه‌داده قضایی که آن را در بخش ۳-۲ بازیابی نمودیم استفاده می‌کنیم.

```
## Load the jurisdiction network
load("judicial.Rdata.gz")

## If we don't have it then create it again
if (!exists("jg")) {
  source("http://cneurocv.s.rmki.kfki.hu/igraph/plus.R")
  vertices <- read.csv("
    http://cneurocv.s.rmki.kfki.hu/igraph/judicial.csv")
  edges <- read.table("
    http://cneurocv.s.rmki.kfki.hu/igraph/allcites.txt")
  jg <- graph.data.frame(edges, vertices=vertices, dir=TRUE)
}
```

معمولا توابع زیر برای مقداردهی و پرس‌وجو از این صفات استفاده می‌گردد:

```
set.graph.attribute, get.graph.attribute, list.graph.attributes,
set.vertex.attribute, get.vertex.attribute, list.vertex.attributes,
set.edge.attribute, get.edge.attribute, list.edge.attributes
```



توابع فوق، عملیات مقداردهی، بازیابی و ارائه لیست صفات را انجام می‌دهند. البته راه ساده‌تر دسترسی استفاده از \$ در جلوی متغیرهاست. در کد زیر چگونگی دسترسی بخشی از صفات پشت سر هم نشان داده شده است.

```
## What do we have?
summary(jg)
V(jg)$year[1:100]
V(jg)$parties[1:10]
```

پرس‌وجو روی صفات نیز مشابه دستور فوق و بدون نیاز به حلقه انجام‌پذیر است. با توجه به Index شدن این صفات نتایج به سرعت محاسبه می‌گردد.

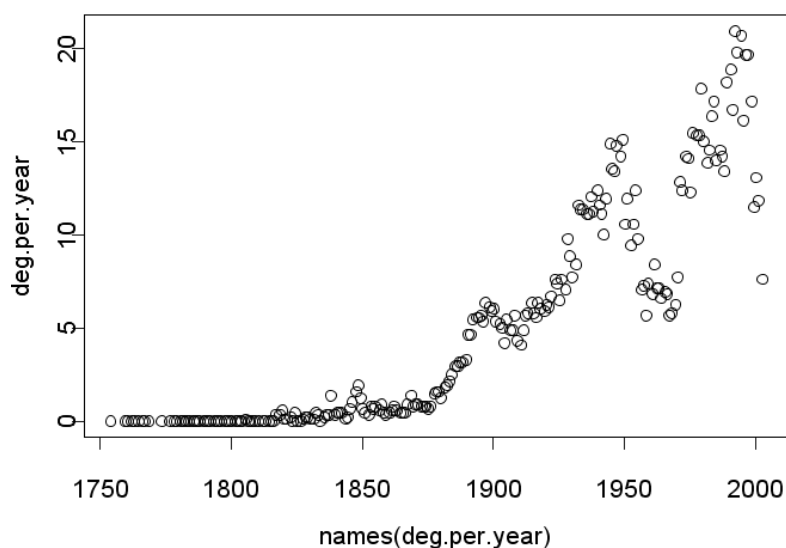
```
## Select vertices based on attributes
V(jg) [ year >= 1990 ]
V(jg) [ overruled!=0 ]
```

اسکرپت زیر یک عملیات پیچیده را روی صفت سال در گره‌ها انجام می‌دهد. این کد مشابه Group by در SQL عمل می‌کند و میانگین درجه گره‌ها را در گره‌های هر سال محاسبه می‌نماید.

```
## Group network measures based on attributes
deg.per.year <- tapply(degree(jg, mode="out"), V(jg)$year,
mean)

## Plot it
plot( names(deg.per.year), deg.per.year )
```

همان‌گونه که نمودار زیر نشان می‌دهد روند میانگین درجه نقاط به مرور زمان در حال زیاد شدن است. این بدان معناست که استناد به پرونده‌های قضایی در حال زیاد شدن می‌باشد.



شکل ۱۹. میانگین درجه نقاط در گراف قضایی



یک مثال پیشرفته تر از نحوه دسترسی به صفات در زیر آمده است. برای این مثال از یک گراف تصادفی Erdos-Renyi استفاده می کنیم [۱۰]. رنگ گره ها به صورت تصادفی قرمز یا سیاه در نظر گرفته می شود.

```
## A more advanced example
g <- erdos.renyi.game(100, 1/100)
V(g)$color <- sample( c("red", "black"),
                      vcount(g), rep=TRUE)
E(g)$color <- "grey"
```

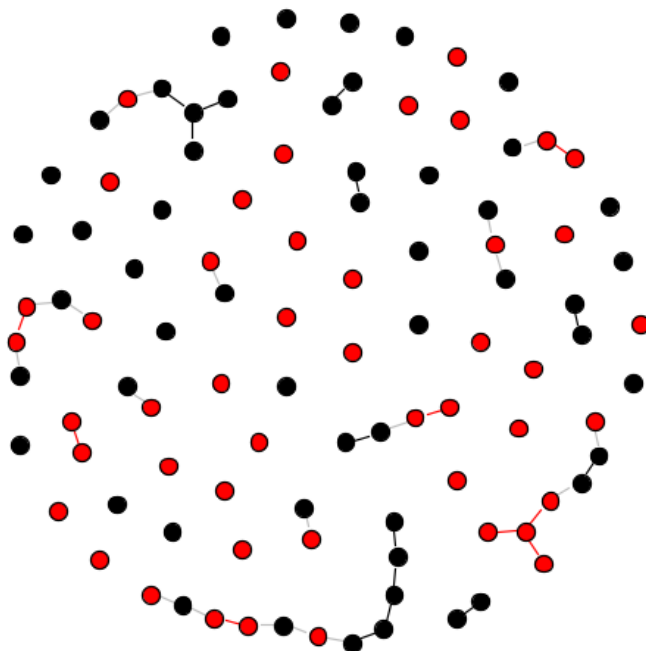
در این مرحله گره های قرمز و سیاه را از یکدیگر جدا می نمایم.

```
red <- V(g)[ color == "red" ]
bl <- V(g)[ color == "black" ]
```

حال به کمک یک پرس و جوی پیچیده بین رنگ یال هایی که دو طرف قرمز و دو طرف آبی هستند، تمایز قائل می شویم. کد زیر نشان می دهد که در هنگام ریختن یک مقدار در متغیر، پرس و جو می تواند در عبارت سمت چپ نیز اعمال گردد.

```
E(g)[ red %--% red ]$color <- "red"
E(g)[ bl %--% bl ]$color <- "black"
plot(g, vertex.size=5, layout=
     layout.fruchterman.reingold)
```

شکل رسم شده گراف بالا به صورت زیر خواهد بود.



شکل ۲۰. یک گراف تصادفی که رنگ یال های آن بر اساس رنگ گره ها متمایز گشته است.



۴ تحلیل شبکه اجتماعی

برای بیان مجموعه‌ی روابط پیچیده‌ی میان افراد در سیستم‌های اجتماعی و غیر آن از شبکه استفاده می‌شود. مطالعاتی نظیر یافتن افراد تاثیرگذار، ساختاریابی اجتماعات درون شبکه، پیش‌بینی ارتباط بین افراد و ... به کمک تحلیل همین شبکه‌ها صورت می‌پذیرد. در ادامه به برخی تکنیک‌های تحلیل شبکه اجتماعی در igraph اشاره می‌نماییم.

۴-۱ محاسبه انواع تعاریف مرکزیت گره

تاکنون معیارهای متفاوتی برای محاسبه مرکزیت یک گره ارائه شده‌است. مرکزیت، اهمیت نسبی یک گره در گراف را نشان می‌دهد [۱۱]. به عنوان نمونه میزان اهمیت یک فرد در شبکه اجتماعی یا میزان اهمیت یک اتاق در بنا و ... از جمله کاربردهای این معیار می‌باشد. ۴ تا از مشهورترین تعاریف مرکزیت به شرح زیر است:

۱. **Degree centrality**: تعداد گره‌های مجاور هر گره (تعداد همسایه‌ها).
 ۲. **Betweenness centrality**: تعداد کوتاه‌ترین مسیرهایی که از یک گره عبور می‌نماید.
 ۳. **Closeness centrality**: میانگین کوتاه‌ترین مسیرها بین هر فرد و دیگر افراد در شبکه.
 ۴. **Eigenvector centrality**: این معیار یک امتیاز نسبی به تمام گره‌ها نسبت می‌دهد، محاسبه این معیار بر اساس این اصل انجام می‌شود که ارتباط با گره‌های با امتیاز بالا، نقش بیشتری از ارتباط با گره‌های با امتیاز پایین دارد. الگوریتم Google PageRank از جمله این روش‌ها به حساب می‌آید.
- البته معیارهای مرکزیتی که در igraph پیاده‌سازی شده‌است به این ۴ روش ختم نشده و روش‌های دیگری چون Burt's constraint و Transitivity را هم شامل می‌شود.

برای بررسی ۴ معیار مرکزیت بالا ابتدا یک گراف می‌سازیم و سپس آن به گراف ساده تبدیل می‌نماییم.

```
g <- graph.formula(Andre----Beverly:Diane:Fernando:Carol,
  Beverly--Andre:Diane:Garth:Ed,
  Carol----Andre:Diane:Fernando,
  Diane----Andre:Carol:Fernando:Garth:Ed,
  Ed-----Beverly:Diane:Garth,
  Fernando-Carol:Andre:Diane:Garth:Heather,
  Garth----Ed:Beverly:Diane:Fernando:Heather,
  Heather--Fernando:Garth:Ike,
  Ike-----Heather:Jane)

g <- simplify(g)
```



در ادامه موقعیت گره‌ها را در هنگام رسم مشخص می‌کنیم و نام گره‌ها را برچسب آن‌ها در هنگام رسم مبدل می‌سازیم. حال شکل اولیه گراف آماده است و می‌توانیم آن را رسم کنیم.

```
## Hand-tuned coordinates
coords <- c(5, 5, 119, 256, 119, 256, 120, 340, 478,
           622, 116, 330, 231, 116, 5, 330, 451, 231, 231)
coords <- matrix(coords, nc=2)

## Labels the same as names
V(g)$label <- V(g)$name
g$layout <- coords # $

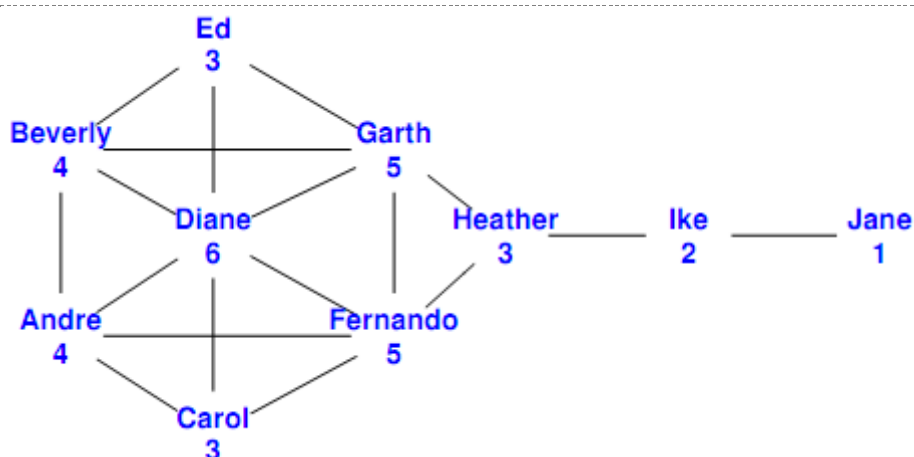
## Take a look at it
plot(g, asp=FALSE, vertex.label.color="blue",
     vertex.label.cex=1.5, vertex.label.font=2,
     vertex.size=20, vertex.color="white",
     vertex.frame.color="white", edge.color="black")
```

در ادامه هر کدام از معیارهای مرکزیت را محاسبه می‌کنیم و در کنار هر گره نمایش می‌دهیم. در زیر ابتدا این عملیات با معیار درجه انجام شده‌است.

```
## Add degree centrality to labels
V(g)$label <- paste(sep="\n", V(g)$name, degree(g))
```

با رسم این گراف با کد زیر، شکل ۲۱ نمایش داده می‌شود.

```
## And plot again
plot(g, asp=FALSE, vertex.label.color="blue",
     vertex.label.cex=1.5, vertex.label.font=2,
     vertex.size=20, vertex.color="white",
     vertex.frame.color="white", edge.color="black")
```



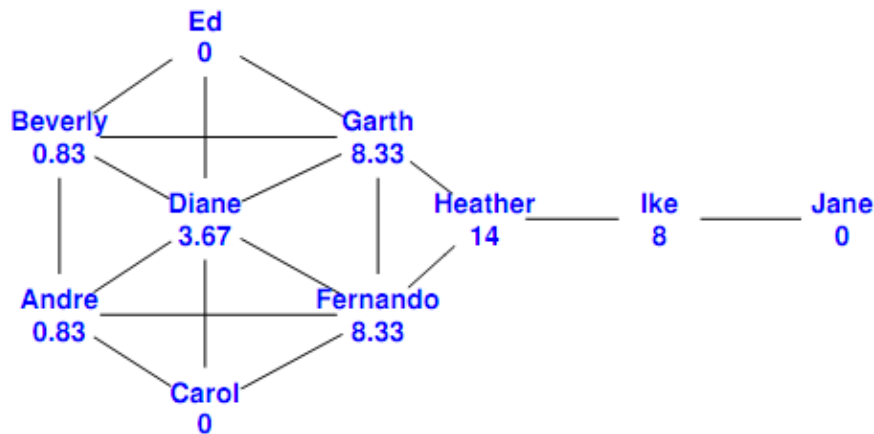
شکل ۲۱. گرافی که Degree centrality گره‌های آن در کنار هر گره نوشته شده‌است.



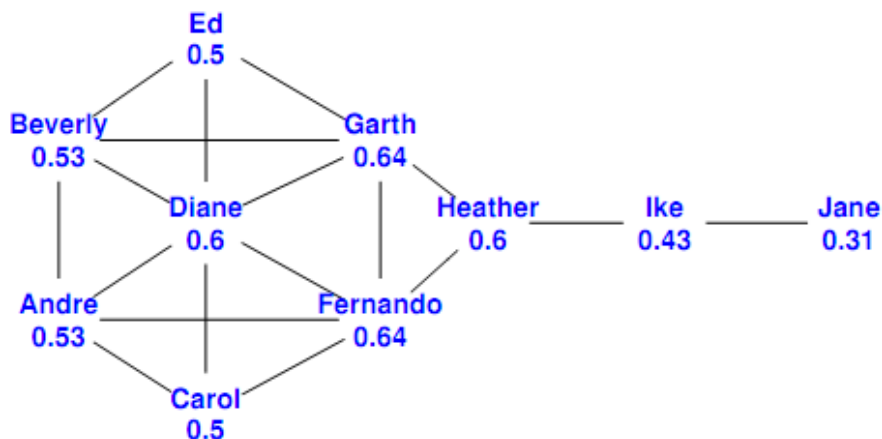
سه معیار دیگر مرکزیت به کمک توابع `betweenness`، `closeness` و `evcent` قابل محاسبه است. فقط کافی

است در کد بالا به جای `degree(g)`، عبارتی چون `format(evcent(g)$vector, digits=1)`

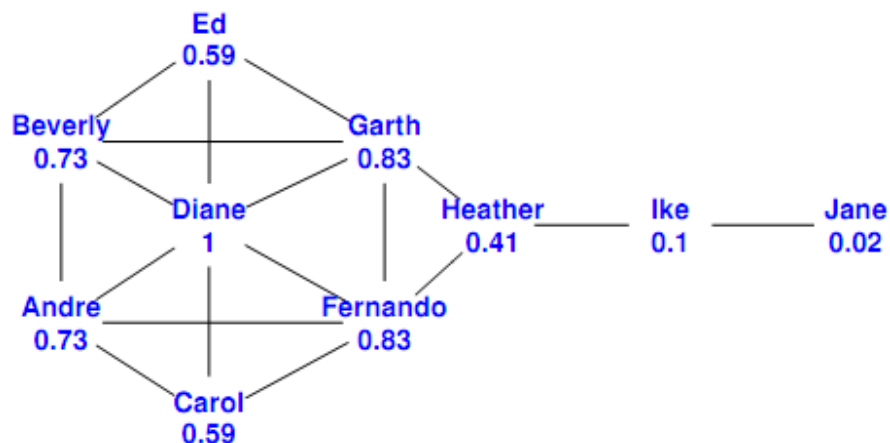
وارد کنید. نتایج به دست آمده برای این سه معیار در شکل‌های زیر نمایش داده شده‌است.



شکل ۲۲. مرکزیت Betweenness گره‌های گراف



شکل ۲۳. مرکزیت Closeness گره‌های گراف



شکل ۲۴. مرکزیت Eigenvector گره‌های گراف



۲-۴ تحلیل یک گراف نسبتاً بزرگ

در این بخش با نحوه دریافت اطلاعات یک گراف نسبتاً بزرگ آشنایی می‌شویم. به علت بزرگ بودن گراف استفاده از برخی از دستورات (مثل رسم گراف) بسیار زمان‌بر است و بعضاً به بسته شدن نرم‌افزار می‌انجامد. از این گذشته با رسم این گراف نیز در زمان زیاد، به علت بزرگی - نمی‌توان به خوبی اطلاعات آن را استخراج نمود. از این رو باید از به کار بردن این دستورات - به جز در موارد ضروری - پرهیز نمود. جدول ۱ سرعت اجرای هر یک از توابع را نشان می‌دهد و بیان می‌کند هر کدام از توابع - در حالت معمول - برای گرافی با چند گره قابلیت محاسبه در زمان منطقی را دارا می‌باشد. اغلب توابعی که در جدول با عنوان "بسیار کند" رده‌بندی شده‌اند، به خاطر پیچیدگی محاسباتی الگوریتم آن‌ها در این رده حضور دارند.

جدول ۱. سرعت توابع igraph

creating graphs (most of the time)- structural modification(add/delete edges/vertices)- subgraph-simplify- graph.decompose- degree- clusters- graph.density- is.simple, is.loop, is.multiple- articulation points and biconnected components- ARPACK stuff: page.rank, hub.score, authority.score, evcent- transitivity- Burts constraint- dyad & triad census, graph motifs- k-cores- MST- reciprocity- modularity- closeness and(edge) betweenness estimation- shortest paths from one source- generating $G_{n,p}$ and $G_{n,m}$ graphs- generating PA graphs with various PA exponents- topological sort	توابع سریع (میلیون‌ها گره)
closeness- diameter- betweenness- all pairs shortest paths, average path length- most layout generators	توابع کند (ده‌هزار گره)
cliques- cohesive blocks- edge/vertex connectivity- maximum flows and minimum cuts- bonpow- alpha centrality- (sub)graphisomorphism	توابع بسیار کند (صد گره)

گرافی که قصد داریم به تحلیل آن بپردازیم، همان گراف اطلاعات قضایی است. ابتدا این گراف را از روی فایل ذخیره‌شده، load می‌کنیم. این گراف که ۳۰۲۸۸ گره و بیش از ۲۱۶ هزار یال دارد، یک گراف جهت‌دار، ساده (غیرچندگانه) و ناهمبند می‌باشد.



```
## Load the jurisdiction network
load("judicial.Rdata.gz")

## Basic data
summary(jg)

## Is it a simple graph?
is.simple(jg)

## Is it connected?
is.connected(jg)
```

با دستور `no.clusters` تعداد مولفه‌های همبند گراف به دست می‌آید. این گراف ۴۸۸۱ مولفه همبند جداگانه دارد. از این میان ۴۸۷۱ گره منفرد (بدون یال)، ۸ مولفه با اندازه ۳ و یک مولفه با اندازه ۴ داریم. مولفه مهم گراف که بزرگترین مولفه همبند آن نیز است، دارای ۲۵۳۸۹ گره می‌باشد.

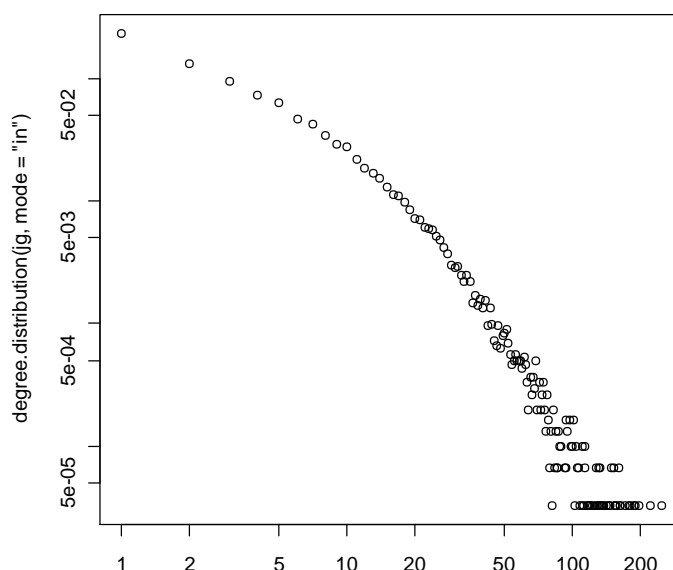
```
## How many components?
no.clusters(jg)

## How big are these?
table(clusters(jg)$size)
```

یکی از مهمترین تحلیل‌ها، تحلیل درجه گره‌ها در گراف می‌باشد. تحلیل درجه ورودی و خروجی گره‌ها با محور y لگاریتمی به صورت زیر می‌باشد. شکل ۲۵ توزیع درجات ورودی را نشان می‌دهد.

```
## In-degree distribution
plot(degree.distribution(jg, mode="in"), log="xy")

## Out-degree distribution
plot(degree.distribution(jg, mode="out"), log="xy")
```



شکل ۲۵. توزیع درجه‌های ورودی گراف قضایی



بیشترین درجه یک گره (مجموع ورودی و خروجی) در این گراف ۳۱۳ است.

```
## Largest in- and out-degree, total degree
max(degree(jg, mode="in"))
max(degree(jg, mode="out"))
max(degree(jg, mode="all"))
```

چگالی گراف، نسبت تعداد یال‌های موجود به تعداد یال‌های ممکن می‌باشد. این معیار برای گراف فوق برابر با ۰.۰۰۰۲۳۶ است. این بدان معناست که گراف فوق بسیار پراکنده می‌باشد.

```
## Density
graph.density(jg)
```

تراگذاری یک گراف، احتمال این که همسایه‌های یک گره با یکدیگر مرتبط باشند را مشخص می‌کند. مقدار تراگذاری را یک بار برای گراف امور قضایی و یک بار برای یک گراف تصادفی محاسبه می‌کنیم. نتایج به دست آمده به ترتیب ۰.۱۲۶ و ۰.۰۰۰۵۳۴ می‌شود. تفاوت فاحش این مقادیر، معناداری ارتباطات طبیعی را نشان می‌دهد.

```
## Transitivity
transitivity(jg)

## Transitivity of a random graph of the same size
g <- erdos.renyi.game(vcount(jg), ecount(jg), type="gnm")
transitivity(g)
```

دستور `dyad.census`، انواع ارتباطات بین جفت گره‌ها را سرشماری و در سه گروه دسته‌بندی می‌کند:

۱. `mut`: تعداد گره‌های با ارتباط دوطرفه.

۲. `asym`: تعداد گره‌های با ارتباط یکطرفه.

۳. `null`: تعداد گره‌هایی که هیچ ارتباطی با هم ندارند.

تابع دیگر سرشماری کننده، `triad.census` می‌باشد. که تعداد زیرگراف‌های با سه یال را می‌شمارد. همانگونه که انتظار می‌رود، با توجه به وضعیت بودن یا نبودن یال‌های جهت‌دار، شانزده ترکیب مختلف خواهیم داشت. (به این زیرگراف‌ها `motifs` هم اطلاق می‌شود). شمارش در این ۱۶ دسته انجام می‌شود.

```
## Dyad census & Triad census
dyad.census(jg)
## triad.census(jg)
```

محاسبه امتیاز `Hubs` و `Authorities` که `Kleinberg` در الگوریتم `HITS` ارائه داد، به صورت زیر است. برای محاسبه این دو امتیاز از ضرب برداری و بردارهای ویژه ماتریس مجاورت استفاده می‌شود

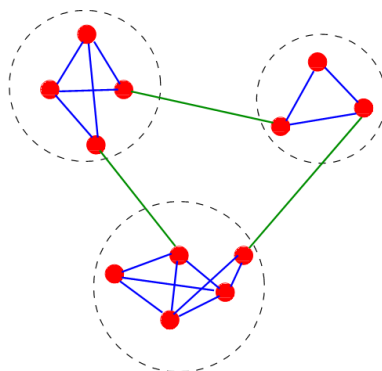
```
## Authority and Hub scores
authority.score(jg)$vector
cor(authority.score(jg)$vector, V(jg)$auth)

hub.score(jg)$vector
cor(hub.score(jg)$vector, V(jg)$hub)
```



۳-۴ ساختاریابی اجتماع

یکی از موضوعات مطرح در تحلیل شبکه‌های اجتماعی، یافتن ساختار اجتماعات موجود در شبکه‌ی اجتماعی، بر اساس شباهت‌های افراد درون شبکه می‌باشد. یک شبکه اجتماعی هنگامی "ساختار اجتماع" (Community Structure) دارد که به صورت طبیعی به گروه‌هایی از گره‌ها تقسیم شده باشد که درون گروه‌ها ارتباط متراکم و مابین گروه‌ها ارتباط تُنک داشته باشد. شکل زیر یک شبکه اجتماعی و ساختار اجتماع درون آن را نمایش داده‌است.



شکل ۲۶. "ساختار اجتماع" در گراف یک شبکه اجتماعی

یک سوال مهم در ساختاریابی این است که ساختار یافته‌شده تا چه حد بهینه است؟ نیومن-گیروان در سال ۲۰۰۴ معیار ماژولاریتی Q را به عنوان شرط انتهایی الگوریتم ارائه داد [۱۲]. اما این معیار به سرعت به یکی از اجزای اصلی دسته‌ی جدیدی از روش‌های ساختاریابی مبدل گشت. تاکنون معیارهای مختلفی برای ماژولاریتی ارائه شده‌است. ما در این آموزش از معیار پایه ماژولاریتی که توسط نیومن ارائه شد استفاده می‌نماییم (عبارت ریاضی ۱).

$$Q = \frac{1}{2m} \sum_{i,j} A_{ij} - \frac{k_i k_j}{2m} \delta(c_i, c_j) \quad (1)$$

در آزمایش اول قصد داریم اندازه ماژولاریتی را در چند گراف بررسی نماییم. برای این منظور ابتدا گرافی با دو قسمت جداگانه ایجاد می‌نماییم.

```
## The concept of modularity
g <- graph.full(5) %du% graph.full(5)
g$layout <- layout.fruchterman.reingold
plot(g)
V(g)$color <- 2
V(g)[0:4]$color <- 1
plot(g)
```

گراف ایجاد شده یک گراف ماژولار است. مقدار ماژولاریتی آن برابر با ۰.۵ می‌باشد.

```
## This graph is modular
modularity(g, membership=V(g)$color-1)
```



این درحالی است که اگر کل گراف را یک اجتماع در نظر بگیریم، مقدار ماژولاریتی برابر با صفر می‌شود.

```
## If we have everyone in the same group,
## that is not modular
V(g)$color <- 1
modularity(g, membership=V(g)$color-1)
```

چنانچه اجتماع‌ها را به صورت تصادفی اختصاص دهیم، این مقدار منفی می‌شود.

```
## If we assign the vertices randomly to two
## groups, that is not very modular, either
V(g)$color <- sample(1:2, vcount(g), rep=TRUE)
plot(g)
modularity(g, membership=V(g)$color-1)
```

برای آزمایش بعدی، گراف کلپ کاراته را داندود می‌کنیم. از آن‌جا که این گراف در فرمت pajek قرار دارد، باید آن را به فرمت igraph تبدیل نماییم. انجام این کار به راحتی با اضافه کردن یک آرگومان به تابع ورود گراف امکان‌پذیر است.

```
## Fast & greedy community detection
karate <- read.graph(
  "cneurocvs.rmki.kfki.hu/igraph/karate.net", format="pajek")
karate <- simplify(karate)
```

الگوریتم fastgreedy که یک الگوریتم بر مبنای ماژولاریتی است [۱۳]، روی این گراف اعمال می‌کنیم. شیوه کار این الگوریتم بدین صورت است که هر گره را یک اجتماع در نظر می‌گیرد و طبق یک روش حریصانه شروع به ادغام این اجتماعات می‌کند. در هر مرحله ماژولاریتی گراف اندازه‌گیری و ذخیره می‌شود، تا این که تمام گره‌ها به یک اجتماع اختصاص یابند. پس از پایان این مرحله جایی که الگوریتم به بیشترین ماژولاریتی رسیده‌بود، به عنوان ساختار اجتماع در نظر می‌گیریم. با اجرای این الگوریتم، بیشترین ماژولاریتی مراحل به ۰.۳۸ می‌رسد.

```
system.time( fc <- fastgreedy.community(karate) )
fc$modularity
max(fc$modularity)
which.max(fc$modularity)
```

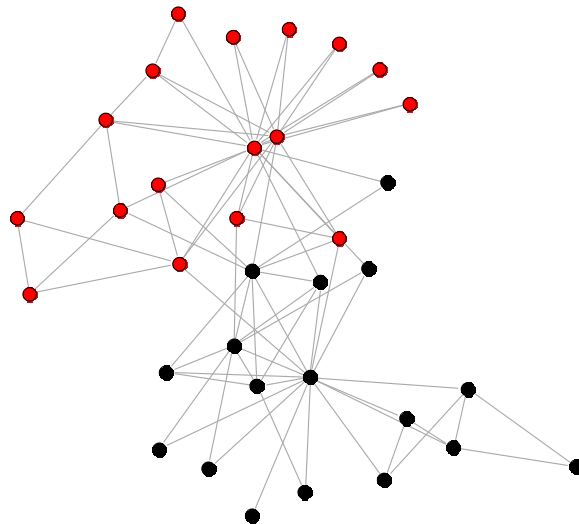
آرایه عضویت بیشترین ماژولاریتی به صورت زیر استخراج می‌گردد.

```
memb <- community.to.membership(karate, fc$merges,
  which.max(fc$modularity))
memb
```

این گراف را رسم می‌کنیم. شکل ۲۷ نتایج به دست‌آمده را نشان می‌دهد.

```
lay <- layout.kamada.kawai(karate)
plot(karate, layout=lay, vertex.size=5, vertex.label=NA,
  vertex.color=memb$membership+1, asp=FALSE)
```





شکل ۲۷. اجتماعات به دست آمده با الگوریتم fastgreedy روی مجموعه داده کلپ کاراته

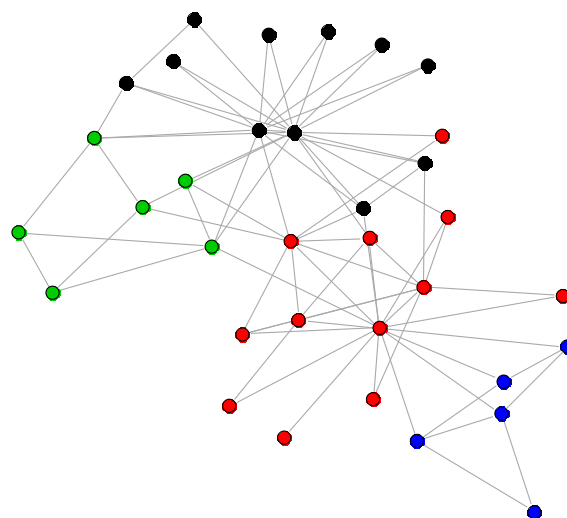
مراحل بالا را روی الگوریتم Spinglass انجام می‌دهیم [۱۴] و نتایج را با هم مقایسه می‌کنیم. ماژولاریتی نتیجه نهایی ۰.۴۲ می‌باشد که به طور قابل ملاحظه‌ای افزایش یافته‌است. البته زمان اجرای الگوریتم پیشین کمتر بود.

```
## Spinglass community detection
system.time(spc <- spinglass.community(karate, spins=20))
spc

x11()
plot(karate, layout=lay, vertex.size=5, vertex.label=NA,
      vertex.color=spc$membership+1, asp=FALSE)
```

ساختار به دست آمده این الگوریتم، دارای چهار اجتماع است. اسکریپ فوق نتایج را رسم نیز می‌نماید (شکل

۲۸). دستور x11 () باعث می‌شود رسم این گراف در پنجره‌ای جداگانه صورت بگیرد.



شکل ۲۸. اجتماعات به دست آمده با الگوریتم Spinglass روی مجموعه داده کلپ کاراته



۴-۴ بلوک‌های چسبنده

چسبندگی ساختاری (Structural cohesion)، کمترین تعداد گره‌هایی از شبکه اجتماعی است که باید آن‌ها را حذف کنیم تا حداقل دو گره که تاکنون به هم متصل بودند، از یکدیگر جدا شوند [۱۵]. در این آزمایش سعی می‌کنیم، بلوک‌هایی از گره‌ها که چسبندگی زیادی دارند و حائل بین بلوک‌ها هستند را بیابیم. ابتدا گرافی که باید روی آن آزمایش انجام دهیم را می‌سازیم و برچسب هر گره را به ترتیب به صورت عددی شماره‌گذاری می‌نماییم.

```
## Load the graph
cb <- graph( c(1,2,1,3,1,4,1,5,1,6,
               2,3,2,4,2,5,2,7,
               3,4,3,6,3,7,
               4,5,4,6,4,7,
               5,6,5,7,5,21,
               6,7,
               7,8,7,11,7,14,7,19,
               8,9,8,11,8,14,
               9,10,
               10,12,10,13,
               11,12,11,14,
               12,16, 13,16, 14,15, 15,16,
               17,18,17,19,17,20,
               18,20,18,21,
               19,20,19,22,19,23,
               20,21, 21,22,21,23,
               22,23)-1, dir=FALSE)

V(cb)$label <- seq(vcount(cb))
```

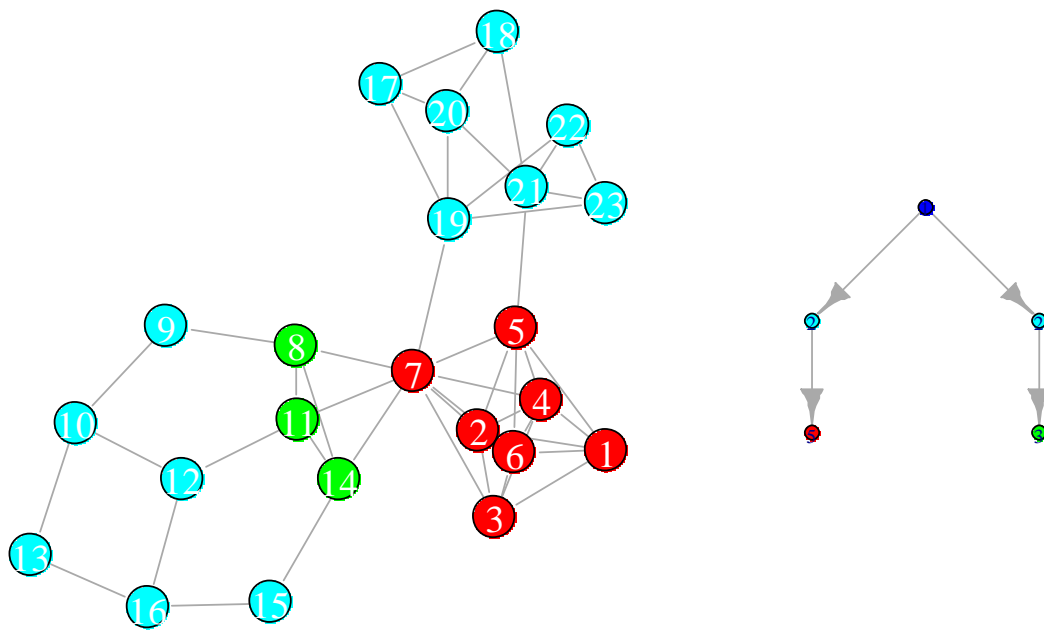
بلوک‌بندی چسبندگی (Cohesive blocking)، یک روش برای مشخص کردن زیرمجموعه‌های سلسله‌مراتبی گره‌های گراف، بر اساس چسبندگی ساختاری (یا اتصال گرهی) است. قبل از تعریف خود بلوک‌بندی چسبندگی لازم است، مفهوم دیگری را تعریف نمایم. برای گراف G ، زیرمجموعه S از گره‌های گراف، maximally k -cohesive است، اگر مجموعه‌ای بزرگتر از S وجود نداشته باشد که S زیرمجموعه آن بوده و اتصال گرهی (vertex connectivity) آن بیشتر یا مساوی با k باشد (به عبارت دیگر S بزرگترین و چسبنده‌ترین جز باشد). با توجه به تعریف فوق، بلوک‌بندی چسبندگی، فرآیندی بازگشتی است که طی آن، زیرمجموعه‌های maximally 1-cohesive یک مجموعه k -cohesive بزرگتر را می‌یابیم ($l > k$). ساختار سلسله‌مراتبی هم از روی همین فرآیند (زیرمجموعه‌های چسبنده تر) به دست می‌آید.

```
blocks <- cohesive.blocks(cb)
blocks
```



نتایج بلوک‌بندی با اسکریپ زیر نمایش داده می‌شود. شکل ۲۹ ساختار سلسله مراتبی و بلوک‌های موجود را نشان می‌دهد.

```
summary(blocks)
blocks$blocks
lapply(blocks$blocks, "+", 1)
blocks$block.cohesion
plot(blocks, layout=layout.kamada.kawai, vertex.label.cex=2,
      vertex.size=15, vertex.label.color="black")
```



شکل ۲۹. ساختار بلوک‌بندی چسبندگی به دست آمده



۵ مراجع

- [1] G. Csárdi and T. Nepusz, “igraph Reference Manual,” URL: <http://igraph.sourceforge.net/documentation.html> (accessed April 20, 2010).
- [2] G. Csardi, “Large-scale network analysis,” 2008.
- [3] G. Csardi, “Package ‘igraph’,” Aug. 2010.
- [4] Wikipedia contributors, “R_(programming_language),” *Wikipedia, The Free Encyclopedia*. Wikimedia Foundation, 23-Jul-2011.
- [5] L. Lam, *An introduction to R*. 2010.
- [6] RapidIVideos, *An Introduction to the Rapidminer R Extension*. YouTube.com , 2010.
- [7] A. L. Barabási and R. Albert, “Emergence of scaling in random networks,” *Science*, vol. 286, no. 5439, p. 509, 1999.
- [8] T. M. J. Fruchterman and E. M. Reingold, “Graph drawing by force-directed placement,” *Software- Practice and Experience*, vol. 21, no. 11, p. 1129–1164, 1991.
- [9] T. Kamada and S. Kawai, “An algorithm for drawing general undirected graphs,” *Information processing letters*, vol. 31, no. 1, p. 7–15, 1989.
- [10] P. Erdos and A. Renyi, “On random graphs,” *Publicationes mathematicae*, vol. 6, no. 290-297, p. 53–54, 1959.
- [11] Wikipedia contributors, “Centrality,” *Wikipedia, The Free Encyclopedia*. Wikimedia Foundation, 27-Jul-2011.
- [12] M. E. J. Newman and M. Girvan, “Finding and evaluating community structure in networks,” *Physical review E*, vol. 69, no. 2, p. 026113, 2004.
- [13] A. Clauset, M. E. J. Newman, and C. Moore, “Finding community structure in very large networks,” *Physical review E*, vol. 70, no. 6, p. 066111, 2004.
- [14] J. Reichardt and S. Bornholdt, “Statistical mechanics of community detection,” *Physical Review E*, vol. 74, no. 1, p. 016110, 2006.
- [15] J. Moody and D. R. White, “Structural Cohesion and Embeddedness: A Hierarchical Concept of Social Groups,” *American Sociological Review*, vol. 68, no. 1, pp. 103-127, Feb. 2003.

