دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

مبانی هوش محاسباتی

تمرین سری اول

اعضای گروه:

ابوالفضل رنجبر

میعاد کیمیاگری

امیر طاها نجف

استاد درس:

دکتر حسین کارشناس

بهار ۱۴۰۴

۱- بخش اول: مبانی و مفاهیم الگوریتم ژنتیک (GA) و ویژگی‌های آن

۱-۱- الگوریتم‌های تکاملی را توضیح دهید. دلایل اصلی برتری الگوریتم‌های تکاملی نسبت به الگوریتم‌های یادگیری تقویتی در برخی مسائل چیست؟

الگوریتم‌های تکاملی (Evolutionary Algorithms - EAs) روش‌های بهینه‌سازی الهام‌گرفته از فرآیندهای طبیعی مانند انتخاب طبیعی، جهش و ترکیب هستند. آن‌ها برای حل مسائل پیچیده و غیرخطی که روش‌های سنتی در حل آن‌ها ناکارآمدند، استفاده می‌شوند. مراحل اصلی شامل مقداردهی اولیه، ارزیابی، انتخاب، تولید مثل (ترکیب و جهش) و تکرار است.

دلایل برتری الگوریتم‌های تکاملی نسبت به یادگیری تقویتی(RL)

۱. **عدم نیاز به مدل محیط:** الگوریتم‌های تکاملی به اطلاعات دقیق از محیط نیاز ندارند، برخلاف یادگیری تقویتی که معمولا به مدل‌سازی محیط وابسته است.

۲. **کارایی در فضای جستجوی بزرگ:** آن‌ها بهتر می‌توانند در فضاهای جستجوی بزرگ و پیچیده عمل کنند.

۳. **مقاومت در برابر بهینه‌سازی محلی:** مکانیسم‌هایی مانند جهش و ترکیب از گیرکردن در بهینه‌های محلی جلوگیری می‌کنند.

٤. **انعطاف‌پذیری:** برای طیف وسیعی از مسائل بدون نیاز به تنظیمات پیچیده قابل استفاده‌اند.

٥. **کار با توابع هدف گسسته:** برخلاف یادگیری تقویتی، به توابع هدف پیوسته وابسته نیستند.

۱-۲- الگوریتم ژنتیک چیست و چه تفاوتی با سایر الگوریتم‌های تکاملی مانند برنامه‌نویسی تکاملی (EP) یا استراتژی‌های تکامل (ES) دارد؟

الگوریتم ژنتیک یک روش بهینه‌سازی الهام‌گرفته از فرآیند تکامل طبیعی است که با استفاده از عملگرهایی مانند انتخاب، جهش و ترکیب به بهبود راه‌حل‌ها در فضای جستجو می‌پردازد. تفاوت اصلی GA با سایر الگوریتم‌های تکاملی مانند برنامه‌نویسی تکاملی و استراتژی‌های تکامل در نحوه نمایش و تغییر ساختار راه‌حل‌ها است. الگوریتم ژنتیک معمولا از نمایش باینری یا رشته‌ای برای کروموزوم‌ها استفاده می‌کند، در حالی که برنامه‌نویسی تکاملی بیشتر بر بهینه‌سازی پارامترهای عددی با استفاده از جهش متمرکز است و استراتژی‌های تکاملی هم بر بهینه‌سازی بردارهای عددی با استفاده از توزیع‌های احتمالی و تغییرات آماری تاکید دارد. همچنین، الگوریتم ژنتیک بیشتر بر ترکیب ژنتیکی تمرکز دارد، در حالی که برنامه‌نویسی تکاملی و استراتژی‌های تکاملی به تغییرات تدریجی‌تر در پارامترها اهمیت می‌دهند.

۱-۳- عملیات جهش (Mutation) و ترکیب (Crossover) معمولا چگونه روی رشته‌های بیتی (Bitstring) در یک الگوریتم زنتیک اعمال می‌شوند؟ این عملگرها هنگام استفاده برای جایگشت‌ها یا سایر نمایش‌های غیر باینری چگونه باید تغییر یابند؟

در الگوریتم‌های ژنتیک، عملگرهای ترکیب (Crossover) و جهش (Mutation) برای ایجاد تنوع و بهبود جستجو در فضای پاسخ استفاده می‌شوند. نحوه اعمال این عملگرها بستگی به نوع نمایش کروموزوم‌ها دارد. در ادامه، نحوه اجرای این عملگرها بر روی رشته‌های بیتی (Bitstring) و جایگشت‌ها یا نمایش‌های غیر باینری توضیح داده می‌شود.

۱. اعمال روی رشته‌های بیتی(Bitstring Representation)

۱.۱ ترکیب (Crossover)

در نمایش بیتی، کروموزوم‌ها به صورت رشته‌هایی از ۰ و ۱ نمایش داده می‌شوند. روش‌های متداول ترکیب عبارتند از:

- **ترکیب تک‌نقطه‌ای (One-Point Crossover):**
  یک نقطه تصادفی در طول رشته انتخاب شده و قسمت‌های قبل و بعد از آن بین دو والد جابجا می‌شوند.
- **ترکیب دونقطه‌ای (Two-Point Crossover):**
  دو نقطه تصادفی در کروموزوم انتخاب شده و ناحیه بین آن‌ها بین والدین جابجا می‌شود.
- **ترکیب یکنواخت (Uniform Crossover):**
  هر بیت از کروموزوم‌های والد با احتمال مشخصی از یکی از والدین به فرزند منتقل می‌شود.

۲.۱ جهش (Mutation)

عمل جهش معمولا با تغییر تصادفی یک یا چند بیت در کروموزوم انجام می‌شود:

- **جهش بیتی(Bit Flip Mutation):**
  یک بیت به‌صورت تصادفی انتخاب شده و مقدار آن تغییر می‌کند (۰ به ۱ یا ۱ به ۰).
- **جهش چندنقطه‌ای (Multi-Point Mutation):**
  چندین بیت به‌صورت تصادفی انتخاب و مقدار آن‌ها تغییر داده می‌شود.

۲. تغییر عملگرها برای جایگشت‌ها و نمایش‌های غیر باینری

در برخی مسائل (مانند مسئله فروشنده دوره‌گرد)، نمایش جایگشتی یا نمایش عددی استفاده می‌شود. در این موارد، عملگرهای ژنتیک باید تغییر کنند تا ساختار داده حفظ شود.

۱.۲ ترکیب (Crossover) برای جایگشت‌ها

- **ترکیب چرخه‌ای(Cycle Crossover - CX):**
  بخش‌هایی از والدین به گونه‌ای جابجا می‌شوند که ترتیب عناصر در فرزند حفظ شود.
- **ترکیب سفارش‌شناسی(Order Crossover - OX):**
  یک بخش تصادفی از والد اول حفظ شده و بقیه ژن‌ها از والد دوم به ترتیب اضافه می‌شوند.
- **ترکیب جایگشتی (Partially Mapped Crossover - PMX):**
  بخشی از کروموزوم بین والدین جابجا شده و نگاشت‌ها برای حفظ ترتیب اصلی اعمال می‌شوند.

۲.۲ جهش (Mutation) برای جایگشت‌ها

- **جهش جابجایی (Swap Mutation):**
  دو ژن در جایگشت انتخاب شده و موقعیتشان تغییر می‌کند.
- **جهش معکوس‌سازی (Inversion Mutation):**
  یک بخش از کروموزوم انتخاب شده و ترتیب آن معکوس می‌شود.
- **جهش درج (Insertion Mutation):**
  یک ژن انتخاب شده و در موقعیتی جدید درج می‌شود.

۱-۴- چه خواص یا خصوصیات تغییرناپذیری (Invariance Properties) باید هنگام اجرای یک الگوریتم ژنتیک روی رشته‌های بیتی حفظ شوند؟ نمونه‌هایی ارائه دهید که نشان دهند این ویژ گی‌ها چگونه بر کارایی و رفتار الگوریتم تأثیر می گذارند.

در اجرای الگوریتم ژنتیک روی رشته‌های بیتی، دو خصوصیت تغییرناپذیر عبارت‌اند از حفظ طول کروموزوم‌ها و حفظ تنوع ژنتیکی. هنگام اجرای عملیات ژنتیکی مانند تقاطع و جهش، طول کروموزوم‌ها نباید تغییر کند، چون تغییر طول می‌تواند موجب ناسازگاری در ارزیابی تابع برازندگی شود. علاوه بر این، حفظ تنوع ژنتیکی برای جلوگیری از همگرایی زودرس و گیر افتادن در بهینه‌های محلی ضروری است. بدون تنوع کافی، الگوریتم ممکن است تنها روی چند راه‌حل ضعیف تمرکز کند و به نتایج بهینه نرسد.

برای درک اثر این ویژگی‌ها بر کارایی الگوریتم، فرض کنید در یک مسئله بهینه‌سازی، کروموزوم‌های ۸ بیتی داریم. اگر طول آن‌ها در طی جهش تغییر کند، مقایسه و پردازش آن‌ها دشوار خواهد شد. همچنین، اگر تنوع ژنتیکی کاهش یابد، همه کروموزوم‌ها ممکن است به یک پاسخ مشترک نزدیک شوند و الگوریتم نتواند راه‌حل‌های جدید و بهینه‌تر را کشف کند. در چنین شرایطی، یک راه برای بهبود عملکرد، تنظیم نرخ جهش و تقاطع به گونه‌ای است که هم ساختارهای مهم ژنتیکی حفظ شوند و هم تنوع کافی در جمعیت باقی بماند.

۱-۵- اگر یک الگوریتم ژنتیک برای رشته های بیتی با طول n برای یافتن جواب بهینه، زمان مورد انتظار $O(n^3)$ را صرف کند، این مقدار چگونه با تعداد مراحل مورد انتظار برای جستجوی تصادفی (با توزیع یکنواخت) مقایسه می‌شود؟ علاوه بر این، چه عواملی می توانند بر عملکرد الگوریتم در عمل تاثیر بگذارند؟

در جستجوی تصادفی یکنواخت برای رشته های بیتی با طول، n زمان مورد انتظار برابر 2^n می‌شود زیرا هر بیت دو حالت دارد و چون توزیع یکنواخت است، احتمال مساوی برای هر یک از راه حل‌ها وجود داد بنابراین به طور متوسط، یک جستجوی تصادفی باید نیمی از راه حل‌های ممکن را امتحان کند تا بهترین را پیدا کند.

در نتیجه الگوریتم ژنتیک، به طور قابل توجهی سریع‌تر و کارآمد تر از جستجوی تصادفی است.

عوامل متعددی می‌توانند بر عملکرد واقعی یک الگوریتم ژنتیک در عمل تاثیر بگذارند:

- **اندازه جمعیت:** جمعیت کم ممکن است منجر به از دست دادن تنوع و همگرایی زودرس شود، همچنین در جمعیت بسیار بزرگ می‌تواند هزینه‌های محاسباتی را افزایش دهد و زمان اجرا را طولانی‌تر کند.
- **عملگرهای ژنتیکی:** عملگرهای انتخاب، تقاطع و جهش نقش کلیدی در فرآیند تکامل دارند. انتخاب، افراد با برازش بالاتر را برای تولید نسل بعد انتخاب می‌کند. تقاطع، اطلاعات ژنتیکی والدین را ترکیب می‌کند تا فرزندان جدیدی ایجاد کند. جهش، تغییرات تصادفی کوچکی در ژن‌ها ایجاد می‌کند و به حفظ تنوع ژنتیکی کمک می‌کند. تعادل مناسب بین این عملگرها برای جستجوی موثر ضروری است .
- **همگرایی زودرس:** هنگامی که جمعیت به سرعت به یک یا چند راه‌حل محلی همگرا شود، تنوع ژنتیکی کاهش می‌یابد. استفاده از تکنیک‌های افزایش تنوع (مانند تنظیم نرخ جهش یا معرفی روش‌های انتخاب متنوع) می‌تواند به جلوگیری از این مشکل کمک کند.
- **نحوه نمایش مسئله و کدگذاری:** انتخاب یک نمایش مناسب برای مسئله (مثلا کدگذاری بیتی) به نحوه اجرای عملگرهای ژنتیکی تاثیر می‌گذارد. نمایش بهینه باعث می‌شود عملگرهای انتخاب، تقاطع و جهش بتوانند به‌طور مؤثرتر فضای حل مسئله را کاوش کنند.
- **ویژگی‌های فضای جستجو:** ساختار سطح برازش، تعداد بهینه‌های محلی و شیب‌های موجود در فضای حل مسئله نیز از عوامل مؤثر هستند.

۲- بخش دوم: درک و حل مسائل با الگوریتم ژنتیک

۲-۱- مسئله فروشنده دوره گرد (TSP) را در نظر بگیرید، که هدف آن تعیین کو تاه ترین مسیر ممکن است که از هر مجموعه شهر داده شده دقیقا یک بار بازدید کرده و به شهر مبدأ باز گردد. برای حل این مسئله با به کار گیری یکی الگوریتم ژنتیک، (GA) فرض کنید که هر ژن در یک کروموزوم نشان‌دهنده‌ی یک یال بدون جهت بین دو شهر است. به عنوان مثال، ژن 'TI' نشاندهنده یک اتصال مستقیم بین تهران و اصفهان است، و با توجه به فرض بدون جهت بودن، 'Tl' معادل 'IT' در نظر گرفته می شود.

الف) اگر تعداد کل شهرها ۱۰ باشد، هر یک کروموزوم به چند ژن نیاز دارد؟

چون هر ژن نشاند دهنده یک یال بین دو شهر است، تعداد ژن‌های مورد نیاز برای هر کروموزوم برابر با تعداد شهرهای مسیر است. بنابراین، برای ۱۰ شهر، هر کروموزوم شامل ۱۰ ژن خواهد بود.

ب) الفبای الگوریتم (مجموعه ژن‌های منحصر به فرد) شامل چند ژن یکتاست؟

برای محاسبه تعداد ژن‌های منحصر به فرد، باید تعداد یال‌های ممکن بین ۱۰ شهر را محاسبه کنیم. چون جهت نداریم پس این مقدار برابر با تعداد ترکیبات دو شهر از بین ۱۰ شهر بدون توجه به ترتیب است. این تعداد برابر است با ترکیب ۲ از ۱۰ که محاسبه آن به صورت زیر است:

$$\binom{10}{2} = \frac{9 \times 10}{2} = 45$$

الفبای الگوریتم شامل ۴۵ ژن یکتا است.

۲-۲- فرض کنید یک الگوریتم ژنتیک از کروموزوم‌هایی به شکل $x = abcdefgh$ با طول ثابت هشت ژن استفاده می‌کند. هر ژن میتواند هر عددی بین ۰ تا ۹ باشد. مقدار برازش (Fitness) یک فرد/کروموزوم x به‌صورت زیر محاسبه می‌شود.

$$f(x) = (a + b) - (c + d) + (e + f) - (g + h)$$

و فرض کنید که جمعیت اولیه شامل چهار فرد با کروموزوم‌های زیر باشد:

$$x_1 = 65413532$$

$$x_2 = 87126601$$

$$x_3 = 23921285$$

$$x_4 = 41852094$$

الف) برازندگی (Fitness) هر فرد/کروموزوم را با نشان دادن تمام مراحل محاسبه کنید، و آن‌ها را به ترتیب از بیشترین مقدار برازش تا کمترین مرتب کنید.

$$f_2 = (8+7) - (1+2) + (6+6) - (0+1) = 23$$

$$f_1 = (6+5) - (4+1) + (3+5) - (3+2) = 9$$

$$f_3 = (2+3) - (9+2) + (1+2) - (8+5) = -16$$

$$f_4 = (4+1) - (8+5) + (2+0) - (9+4) = -19$$

ب) عملیات ترکیب (Crossover) زیر را انجام دهید:

- دو فرد با بالاترین مقدار برازش را با استفاده از ترکیب تک نقطه‌ای (One-point crossover) در نقطهٔ میانی ترکیب کنید.

$$x_1 = 6541.3532$$

$$x_2 = 8712.6601$$

$$c_1 = 65416601$$

$$c_2 = 87123532$$

- دومین و سومین فرد برتر را با استفاده از ترکیب دو نقطه‌ای (Two-point crossover) در نقاط بین ژن‌های bc و fg ترکیب کنید.

$x_1 = 65.4135.32$ دومین فرد

$x_3 = 23.9212.85$ سومین فرد

$$c_3 = 65921232$$

$$c_4 = 23413585$$

- فرد اول و سوم برتر را با استفاده از ترکیب یکنواخت (Uniform Crossover) ترکیب کنید.

در ترکیب یکنواخت با استفاده از یک ماسک فرضی (احتمالی) اقدام به انتخاب هر ژن از والدین می‌شود.

ماسک فرضی: ۰۱۰۱۰۱۰۱ (۰ به معنی انتخاب ژن از والد اول و ۱ به معنی انتخاب ژن از والد دوم)

$$x_2 = 87126601$$

$$x_3 = 23921285$$

$$c_5 = 83126205$$

برای افزایش تعداد فرزندان به عدد ۶، همان ماسک به‌صورت معکوس استفاده می‌شود.

$$c_6 = 27921681$$

ج) فرض کنید جمعیت جدید شامل شش فرد حاصل از عملیات ترکیب در سوال قبل باشد. مقدار برازش این جمعیت جدید را محاسبه کنیدو تمامی مراجل محاسبات را نشان دهید. آیا مقدار برازش کلی بهبود یافته است؟

$$c_1 = 65416601 \text{ - } c_2 = 87123532 \text{ - } c_3 = 65921232$$

$$c_4 = 23413585 \text{ - } c_5 = 83126205 \text{ - } c_6 = 27921681$$

$$f(c_1) = (6+5) - (4+1) + (6+6) - (0+1) = 17$$

$$f(c_2) = (8+7) - (1+2) + (3+5) - (3+2) = 15$$

$$f(c_3) = (6+5) - (9+2) + (1+2) - (3+2) = -2$$

$$f(c_4) = (2+3) - (4+1) + (3+5) - (8+5) = -5$$

$$f(c_5) = (8+3) - (1+2) + (6+2) - (0+5) = 11$$

$$f(c_6) = (2+7) - (9+2) + (1+6) - (8+1) = -4$$

برای ارزیابی کلی، میانگین و حدود برازندگی فرزندان مورد بررسی قرار داده می‌شود.

$$avg_x = 0.75 \ , \ avg_{children} \approx 5.33$$

با توجه به افزایش میانگین و همچنین تغییر کمترین بارزندگی از (۱۹-) به (۵-)، می‌توان بهبود برازش را نتیجه گرفت.

د) با بررسی تابع برازش و در نظر گرفتن این که ژن‌ها فقط می‌توانند اعداد ۰ تا ۹ باشند، کروموزومی را بیابید که بیشترین مقدار برازش ممکن را داشته باشد (جواب بهینه). همچنین مقدار بیشینهٔ برازش را محاسبه کنید.

$$f(x) = (a+b) - (c+d) + (e+f) - (g+h)$$

با توجه به تابع برازش، باید مقادیر $abef$ بیشینه (۹) باشند و مقادیر $cdgh$ کمینه (۰) باشند:

کروموزوم: 99009900

$$f(x)_{max} = (9+9) - (0+0) + (9+9) - (0+0) = 36$$

ه) با بررسی جمعیت اولیهٔ الگوریتم، آیا می‌توان گفت که این الگوریتم بدون استفاده از عملگر جهش (Mutation) می‌تواند به بهترین راه حل ممکن دست یابد؟

به وضوح مشاهده می‌شود که هیچ یک از این کروموزوم‌ها به ترکیب ایده‌آل نرسیده‌اند. اگر تنها از عملگرهای ترکیب (Crossover) استفاده شود، تنها ترکیب‌های موجود از ژن‌های موجود در جمعیت اولیه به دست می‌آیند. در این صورت، اگر المان‌های لازم برای ساخت کروموزوم بهینه (مثل داشتن تعداد کافی از ژن‌های ۹ در موقعیت‌های افزوده‌شونده یا ژن‌های ۰ در موقعیت‌های تفریقی) در جمعیت اولیه موجود نباشد، حتی با ترکیب‌های مختلف نیز نمی‌توان به بهترین جواب ممکن رسید.

۲-۳- یک الگوریتم ژنتیکی را در نظر بگیرید که روی یک جمعیت ۱۰ نفری با ترکیب زیر اعمال شده است.

- X = ۱: دو نمونه
- X = ۲: سه نمونه
- X = ۳: دو نمونه
- X = ۴: سه نمونه

تابع برازندگی به‌صورت زیر تعریف شده است:

$$f(x) = x^3 - 4x^2 + 7$$

الف) مقادیر خام برازندگی را برای هر مقدار متمایز از X محاسبه کنید.

کافی است مقادیر را در تابع برازندگی، جایگذاری کنیم.

$$f(1) = 1^3 - 4(1)^2 + 7 \ = \ 4$$
$$f(2) = 2^3 - 4(2)^2 + 7 \ = \ -1$$
$$f(3) = 3^3 - 4(3)^2 + 7 \ = \ -2$$
$$f(4) = 4^3 - 4(4)^2 + 7 \ = \ 7$$

ب) بررسی کنید که **آیا مقدار برازندگی خام برای هر X منفی است یا نه.** در صورت وجود مقادیر منفی، **یک مقدار ثابت را به تمام مقادیر برازندگی اضافه کنید** تا احتمال‌های انتخاب غیرمنفی شوند.

همان طور که در قسمت قبل هم مشاهده شد، مقدار برازندگی برای مقادیر $x = 2$ و $x = 3$ منفی است. کافی است مقدار $c = 2$ که برابر با حداقل مقادیر برازندگی است را به آن‌ها اضافه کنیم تا احتمال‌های انتخاب غیر منفی شوند. پس مقادیر برازندگی جدید به صورت زیر در می‌آیند.

$$f'(1) = f(1) + 2 \ = \ 6$$
$$f'(2) = f(2) + 2 \ = \ 1$$

$$f'(3) = f(3) + 2 \ = \ 0$$
$$f'(4) = f(4) + 2 \ = \ 9$$

پ) مجموع **کل برازندگی جمعیت (پس از هرگونه تغییر لازم)** را محاسبه کنید.

طبق برازندگی‌های قسمت قبل و تعداد هر x داریم:

$$\text{برازندگی جمعیت} = f'(1) \times 2 \ + f'(2) \times 3 \ + \ f'(3) \times 2 + \ f'(4) \times 3$$
$$= \ 6 \times 2 \ + \ 1 \times 3 \ + \ 0 \times 2 + 9 \times 3 \ = \ 42$$

ت) با استفاده از **روش انتخاب چرخ رولت، احتمال انتخاب یک فرد** با x = ۴ و x = ۳ و x = ۲ و x = ۱ را بر اساس مقادیر برازندگی (اصلاح‌شده) تعیین کنید.

در روش انتخاب چرخ رولت، احتمال انتخاب هر مقدار x برابر است با نسبت برازندگی اصلاح‌شده (با توجه به تعداد) آن به مجموع برازندگی‌های کل. بنابراین داریم:

$$P(x = 1) = \frac{6 \times 2}{42} \approx 0.285$$
$$P(x = 2) = \frac{1 \times 3}{42} \approx 0.071$$
$$P(x = 3) = \frac{0 \times 2}{42} = 0.000$$
$$P(x = 4) = \frac{9 \times 3}{42} \approx 0.642$$

ث) فرض کنید که اکنون احتمالات انتخاب با استفاده از تابع برازندگی تغییریافته زیر محاسبه می‌شود:

$$g(x) = [f(x)]^2$$

مزیت این برازندگی جدید چیست؟ **احتمال انتخاب هر فرد را با استفاده از g(x) مجددا محاسبه کنید.**

به نظرم مزیت این تابع برازندگی حسابی بودن (مجموعه اعداد صحیح مثبت) خروجی آن است. احتمالات جدید این‌گونه به دست می‌آیند:

$$\boldsymbol{g(1)} = f(1)^2 \ = \ 4^2 = 16$$
$$\boldsymbol{g(2)} = f(2)^2 \ = \ -1^2 = 1$$

$$\boldsymbol{g(3)} = f(3)^2 \ = \ -2^2 = 4$$

$$\boldsymbol{g(4)} = f(4)^2 \ = \ 7^2 = 49$$

چون همه مقادیر مثبت هستند، نیازی به شیفت آن‌ها نیست. در مرحله بعد برازندگی کل جمعیت را حساب می‌کنیم:

$$\text{برازندگی جمعیت} = g'(1) \times 2 \ + g'(2) \times 3 \ + \ g'(3) \times 2 + \ g'(4) \times 3$$
$$= \ 16 \times 2 \ + \ 1 \times 3 \ + \ 4 \times 2 + 49 \times 3 \ = \ 190$$

احتمال انتخاب هر x بر اساس $g(x)$ (روش چرخ رولت) به‌صورت زیر است:

$$P_g(x = 1) = \frac{16 \times 2}{190} \approx 0.168$$

$$P_g(x = 2) = \frac{1 \times 3}{190} \approx 0.015$$

$$P_g(x = 3) = \frac{4 \times 2}{190} \approx 0.042$$

$$P_g(x = 4) = \frac{49 \times 3}{190} \approx 0.773$$

ج) توضیح دهید که استفاده از مقادیر برازندگی به توان دو، یعنی g(x) به جای f(x)، چگونه فشار انتخاب (Selection Pressure) را تحت تأثیر قرار می‌دهد. این موضوع چه تأثیری بر همگرایی و تنوع جمعیت در الگوریتم ژنتیکی خواهد داشت؟

فشار انتخاب یعنی شدت اینکه چقدر افراد برازنده‌تر را در فرآیند انتخاب ترجیح دهیم. اگر فشار انتخاب بالا باشد، فقط برازنده‌ترین راه‌حل‌ها انتخاب می‌شوند و افرادی که برازندگی کمتری دارند، شانس زیادی برای بقا نخواهند داشت. این موضوع می‌تواند باعث همگرایی زودرس شود، به این معنی که الگوریتم در یک بهینه‌ی محلی (Local Optimum) گیر کند و از یافتن پاسخ‌های بهتر باز بماند.

حالا، از آنجایی که تابع برازندگی مقدار برازندگی را به توان دو می‌رساند، تفاوت بین افرادی که برازندگی بیشتری دارند و آن‌هایی که برازندگی کمتری دارند، بیشتر می‌شود. این باعث ایجاد نوعی اختلاف طبقاتی در جمعیت می‌شود؛ یعنی افراد بسیار برازنده خیلی بیشتر از بقیه شانس انتخاب دارند. در نتیجه، فشار انتخاب افزایش پیدا می‌کند و فقط آن‌هایی که خیلی برازنده هستند انتخاب می‌شوند. این موضوع تنوع جمعیت را کاهش داده و باعث می‌شود الگوریتم در همان پاسخ‌های اولیه گیر کند و امکان کشف راه‌حل‌های متنوع‌تر و شاید بهینه‌تر را از دست بدهد، که این همان همگرایی زودرس است.

۳- بخش سوم: پیاده‌سازی، ارزیابی و تجزیه تحلیل الگوریتم ژنتیک جهت انتخاب بهترین ویژگی‌ها برای بهبود مدل‌های طبقه‌بندی مشتریان فروشگاه

در این تمرین الگوریتم ژنتیک از ابتدا پیاده سازی شده تا مهم‌ترین ویژگی‌ها برای مسئلهٔ طبقه‌بندی مشتریان یک فروشگاه انتخاب شود. اهداف این تمرین شامل موارد زیر است:

- طراحی و پیاده سازی الگوریتم ژنتیک برای انتخاب ویژگی با انتخاب پارامترهای کلیدی مانند اندازهٔ جمعیت، تابع برازش و نرخ جهش.
- شناسایی بهترین ۳، ۵ و ۸ ویژگی از میان ویژگی‌های مجموعه داده.
- مقایسه عملکرد یک مدل طبقه بندی (در این تمرین Descision Tree Classifier) با استفاده از هر یک ۳ ویژگی منتخب برتر، ۵ ویژگی منتخب برتر، ۸ ویژگی منتخب برتر و همهٔ ویژگی‌ها.

۳-۱- پیش پردازش داده‌ها

در این بخش از کد، سه تابع اصلی به منظور آماده‌سازی داده‌ها برای تحلیل و مدل‌سازی طراحی شده‌اند. ابتدا تابعی با نام handle_missing_values_inplace مقادیر گمشده در داده‌ها را در قالب درجا برطرف می‌کند؛ در ستون‌های عددی از میانه هر ستون و در ستون‌های متنی از مقدار پرتکرار (mode) استفاده می‌شود تا داده‌های ناقص تکمیل شوند. سپس تابع remove_outlier با استفاده از روش IQR، داده‌های پرت را از میان ویژگی‌های عددی حذف می‌کند؛ این تابع ابتدا چارک‌های اول و سوم را محاسبه کرده و با تعیین حدود مجاز به کمک ضریب k (معمولا ۱.۵)، داده‌هایی که خارج از این محدوده قرار دارند را فیلتر می‌کند. در نهایت، تابع encode به رمزگذاری ویژگی‌های دسته‌ای می‌پردازد تا داده‌های متنی به صورت عددی تبدیل شوند؛ این تبدیل از طریق تابع pd.factorize انجام می‌شود و در صورت وجود DataFrame مرجع، سعی بر حفظ یکسان بودن دسته‌بندی‌ها صورت می‌گیرد. این توابع به طور کلی به بهبود کیفیت داده‌ها و آماده‌سازی آن‌ها برای مراحل بعدی تحلیل و مدلسازی کمک می‌کنند.

```
def handle_missing_values_inplace(data_frame: pd.DataFrame) -> None:
  numeric_cols = data_frame.select_dtypes(include=[np.number]).columns.tolist()
  numeric_cols = data_frame.select_dtypes(include=[np.number]).columns.tolist()
  print("Numeric Columns:", numeric_cols)
  for col in numeric_cols:
    data_frame[col] = data_frame[col].fillna(data_frame[col].median())
  categorical_cols = data_frame.select_dtypes(include=['object']).columns.tolist()
  print("Categorical Columns:", categorical_cols)
  for col in categorical_cols:
    data_frame[col] = data_frame[col].fillna(data_frame[col].mode()[0])
def remove_outlier(df: pd.DataFrame, exclude_cols=[], k=1.5) -> pd.DataFrame:
  init_len = len(df)
```

```
  cols = df.select_dtypes(include=[np.number]).columns.tolist()
  for col in cols:
    if col in exclude_cols:
      continue
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - k * IQR
    upper_bound = Q3 + k * IQR
    df = df.loc[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
  print(f"Removed {init_len - len(df)} outliers.")
  return df
def encode(df: pd.DataFrame, ref_df: pd.DataFrame = None) -> pd.DataFrame:
    categorical_cols = df.select_dtypes(include=['object']).columns
    for col in categorical_cols:
        if ref_df is not None:
            df[col] = pd.Categorical(df[col], categories=ref_df[col].unique()).codes
        else:
            df[col] = pd.factorize(df[col])[0]
    return df
```

۳-۲- پیاده‌سازی الگوریتم ژنتیک (GA) برای انتخاب ویژگی‌ها

۳-۲-۱- تعریف اجزای الگوریتم ژنتیک

کروموزوم‌ها را رشته‌هایی باینری در نظر گرفتیم و هر بیت از آن‌ها نشان دهنده انتخاب شدن یا نشدن یک ویژگی می‌باشد. همان طور که در تکه کد زیر هم مشخص است، تابع gen_init_population یک جمعیت اولیه برای الگوریتم ژنتیکی ایجاد می‌کند. این تابع با دریافت تعداد کروموزوم‌ها و تعداد ویژگی‌ها، برای هر کروموزوم یک آرایه دودویی تصادفی تولید می‌کند که هر مقدار آن ۰ یا ۱ است. اگر کروموزومی فقط شامل صفر باشد (هیچ ویژگی‌ای انتخاب نشده باشد)، یک مقدار تصادفی آن را به ۱ تغییر می‌دهد تا حداقل یک ویژگی انتخاب شود.

```
def gen_init_population(pop_size, n_features):
  population = []
  for _ in range(pop_size):
    chromosome = np.random.randint(2, size=n_features)
    if not sum(chromosome):
      chromosome[np.random.randint(n_features)] = 1
    population.append(chromosome)
  return np.array(population)
gen_init_population(10, 5)
```

۳-۲-۲- تعریف تابع برازش

در این بخش از کد تابع برازش به گونه ای تعریف شده است که کیفیت یک کروموزوم را بر اساس عملکرد آن در درخت تصمیم ارزیابی می‌کند. ابتدا، ویژگی‌های انتخاب‌شده از طریق کروموزوم استخراج می‌شوند و در صورتی که هیچ ویژگی‌ای انتخاب نشده باشد، مقدار ۰ بازگردانده می‌شود. سپس، مجموعه داده‌های انتخاب‌شده به دو بخش آموزشی و تست با نسبت ۸۰٪ به ۲۰٪ تقسیم می‌شوند تا مدل بر اساس آن‌ها آموزش ببیند. در نهایت، دقت مدل به عنوان معیار ارزیابی کیفیت کروموزوم محاسبه و بازگردانده می‌شود.

```
def fitness_evaluation(chromosome, feature_matrix, target_labels):
    selected_feature_indices = np.where(chromosome == 1)[0]
    if len(selected_feature_indices) == 0:
        return 0
    selected_features = feature_matrix[:, selected_feature_indices]
    X_train, X_test, y_train, y_test = train_test_split(selected_features,
target_labels, test_size=0.2, random_state=50)
    classifier = DecisionTreeClassifier(random_state=50)
    classifier.fit(X_train, y_train)
    predictions = classifier.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    return accuracy
```

۳-۲-۳- استراتژی انتخاب

این بخش شامل دو روش برای انتخاب والدین در الگوریتم ژنتیک است: **تورنومنت** و **چرخ رولت**. در روش تورنومنت، با انتخاب تصادفی، tournament_size عضو از جمعیت (پیش‌فرض ۳)، عضو با بالاترین برازندگی به عنوان برنده انتخاب می‌شود. این روش تضمین می‌کند که اعضای قوی‌تر شانس بیشتری برای بقا داشته باشند و تنوع جمعیت را تا حدی حفظ می‌کند. در روش چرخ رولت، احتمال انتخاب هر کروموزوم متناسب با مقدار برازندگی آن است. اگر مجموع برازندگی‌ها صفر باشد، احتمال انتخاب یکسان برای همه اعضا در نظر گرفته می‌شود. این روش ممکن است به دلیل تمرکز بر اعضای برتر، منجر به همگرایی زودرس شود، اما در محیط‌هایی با تنوع بالای برازندگی عملکرد مؤثری دارد. هر دو روش به گونه‌ای طراحی شده‌اند که از جمعیت فعلی و مقادیر برازندگی برای انتخاب والدین استفاده کنند و کروموزوم انتخاب‌شده را به عنوان خروجی بازگردانند.

```
def tournament_selection(population, fitness_scores, tournament_size=3):
  selected_indices = np.random.choice(len(population), tournament_size, replace=False)
  selected_fitness = fitness_scores[selected_indices]
  winner_index = selected_indices[np.argmax(selected_fitness)]
  return population[winner_index]
def roulette_wheel_selection(population, fitness_scores):
  total_fitness = np.sum(fitness_scores)
  if total_fitness == 0:
    probabilities = np.ones(len(fitness_scores)) / len(fitness_scores)
```

```
  else:
    probabilities = fitness_scores / total_fitness
  selected_index = np.random.choice(len(population), p=probabilities)
  return population[selected_index]
```

۳-۲-۴- عملگر ترکیب (Crossover)

همان‌طور که در کد زیر هم مشخص است، تابع single_point_crossover یک نقطه تصادفی در کروموزوم انتخاب کرده و از آن نقطه به بعد، بخش‌های دو والد را جابه‌جا می‌کند. اگر مقدار تصادفی کمتر از crossover_rate باشد، ترکیب انجام می‌شود؛ در غیر این صورت، والدین بدون تغییر باقی می‌مانند.

تابع uniform_crossover هر ژن را به‌صورت مستقل بررسی کرده و با احتمال ۵۰ درصد آن را از یکی از والدین انتخاب می‌کند. در نتیجه، فرزندان ترکیبی تصادفی از ژن‌های دو والد خواهند بود.

```
def single_point_crossover(parent1: np.ndarray, parent2: np.ndarray,
crossover_rate: float = 0.8) -> tuple[np.ndarray, np.ndarray]:
  if len(parent1) != len(parent2):
    raise ValueError("Parents must have the same length")
  if np.random.rand() < crossover_rate:
    crossover_point = np.random.randint(1, len(parent1))
    child1 = np.concatenate((parent1[:crossover_point], parent2[crossover_point:]))
    child2 = np.concatenate((parent2[:crossover_point], parent1[crossover_point:]))
  else:
    child1 = parent1
    child2 = parent2
  return child1, child2
def uniform_crossover(parent1: np.ndarray, parent2: np.ndarray) -> tuple[np.ndarray,
np.ndarray]:
  if len(parent1) != len(parent2):
    raise ValueError("Parents must have the same length")
  child1 = np.where(np.random.rand(len(parent1)) < 0.5, parent1, parent2)
  child2 = np.where(np.random.rand(len(parent1)) < 0.5, parent2, parent1)
  return child1, child2
```

۳-۲-۵- جهش (Mutation)

در این تابع با یک نرخ جهش (پیش‌فرض ۰.۰۵) تغییراتی در کروموزوم ایجاد می‌کند. هر ژن با احتمال مشخص برعکس (flip) می‌شود (۰ به ۱ و برعکس). اگر کروموزوم بعد از جهش کاملاً صفر شود، یک ژن به‌صورت رندوم (۱) می‌شود تا از حذف کامل ویژگی‌ها جلوگیری شود.

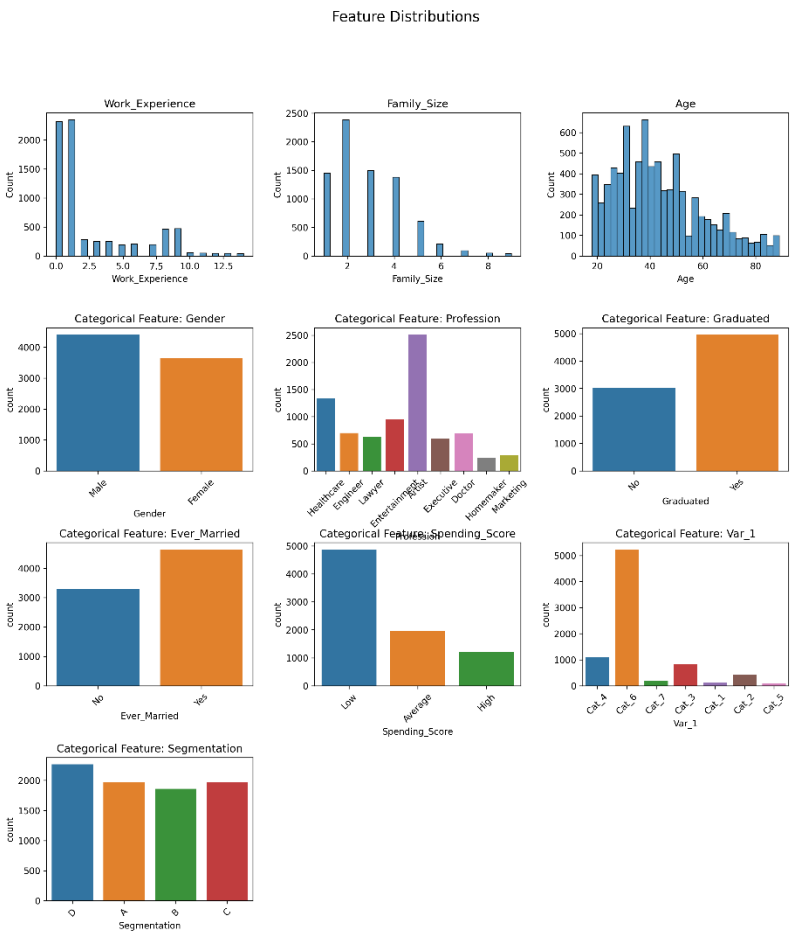
```
def mutate_chromosome(chromosome, mutation_rate=0.05):
```

```
    mutated = chromosome.copy()
    for i in range(len(mutated)):
        if np.random.rand() < mutation_rate:
            mutated[i] = 1 - mutated[i]
    if mutated.sum() == 0:
        mutated[np.random.randint(len(mutated))] = 1
    return mutated
```

۳-۳- انتخاب بهترین ویژگی‌ها

بهترین ویژگی‌ها را، بر اساس میزان انتخاب آن‌ها در طول نسل‌های مختلف تعیین کردیم. در هر نسل، جمعیتی از کروموزوم‌ها که نشان‌دهنده‌ی ترکیب‌های مختلفی از ویژگی‌ها هستند، مورد ارزیابی قرار می‌گیرند و بر اساس مقدار تناسب (fitness) رتبه‌بندی می‌شوند. ویژگی‌هایی که در کروموزوم‌های برتر بیشتر حضور دارند، به‌عنوان ویژگی‌های مهم‌تر در نظر گرفته می‌شوند. طی فرآیند انتخاب، ترکیب و جهش، جمعیت جدیدی تولید شده و این روند تا رسیدن به همگرایی یا حداکثر تعداد نسل ادامه می‌یابد. در نهایت، ویژگی‌ها بر اساس فراوانی انتخاب‌شدنشان مرتب شده و بهترین ترکیب ویژگی‌ها استخراج می‌شود. به علت طولانی بودن کد این بخش، از آوردن آن در این گزارش صرف نظر کردیم. همچنین قابل ذکر است که رسیدن به بیشترین دقت در دسته بندی با استفاده از یک مدل طبقه‌بندی با انتخاب سه ویژگی Age، Gender و Ever_Married به دست می‌آیند. برای به دست آوزدن این سه ویژگی تکه کدی نوشتیم که همه حالات ممکن برای انتخاب ویژگی‌ها را امتحان کند. الگوریتم ژنتیکی که پساده سازی کردیم در اکثر مواقع دو ویژگی Ever_Married و Age را در کروموزوم نهایی انتخاب می‌کرد.

در مورد توزیع ویژگی‌ها هم طبق نمودارهای قابل مشاهده در تصویر ۱ می‌توان گفت که Work_Experience و Family_Size دارای توزیع چوله به راست هستند، به این معنی که بیشتر داده‌ها در مقادیر کم متمرکز شده‌اند. ویژگی Age توزیعی تقریبا نرمال دارد، اما با کمی چولگی به راست. در ویژگی‌های دسته‌ای، Gender و Graduated تقریبا متوازن هستند، اما Spending_Score نشان می‌دهد که بیشتر افراد امتیاز هزینه‌کرد پایینی دارند. Profession نشان می‌دهد که بیشتر افراد در شغل‌های اجرایی مشغول به کارند، در حالی که برخی مشاغل مانند Marketing سهم کمتری دارند. توزیع ویژگی هدف یعنی Segmentation نیز تقریبا متوازن است، هرچند که برخی دسته‌ها کمی غالب‌تر هستند.



تصویر ۱ - توزیع ویژگی‌ها

۳-۴- آموزش و ارزیابی مدل طبقه‌بندی

در این قسمت مدل Decision Classifier Tree را برای دسته‌بندی داده‌ها با استفاده از ویژگی‌های انتخاب‌شده توسط الگوریتم ژنتیک (GA) آموزش داده می‌شود. مدل با مجموعه‌های ۳، ۵، ۸ از ویژگی ها و همه ویژگی‌های انتخاب‌شده آموزش داده می‌شود.

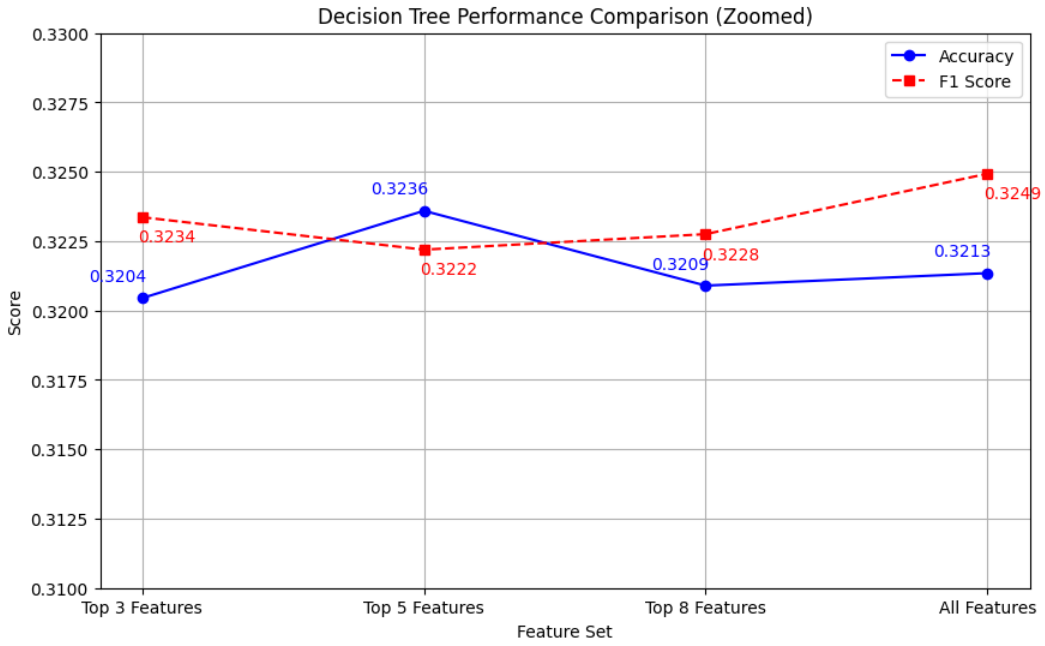
```
from sklearn.metrics import accuracy_score, f1_score,
classification_report, confusion_matrix

def train_and_evaluate_decision_tree(x_train, y_train, x_test, y_test,
selected_features, top_n=None):
    selected_feature_names = [x_train.columns[i] for i in
range(len(selected_features)) if selected_features[i] == 1]
    if top_n:
```

```
        selected_feature_names = [feature for feature, _ in
ranked_features[:top_n]]

    x_train_selected = x_train[selected_feature_names]
    x_test_selected = x_test[selected_feature_names]

    classifier = DecisionTreeClassifier(random_state=50)
    classifier.fit(x_train_selected, y_train)
    y_pred = classifier.predict(x_test_selected)

    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')
    report = classification_report(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Print report
    print(f"Selected Features ({'Top ' + str(top_n) if top_n else
'All'}):", selected_feature_names)
    print("\nModel Performance:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print("\nClassification Report:\n", report)
    print("\nConfusion Matrix:\n", conf_matrix)
    print("------------------------------------------------------------")

    return classifier, selected_feature_names, accuracy

# Evaluate models with different feature sets
classifier_top3, selected_top3, acc_top3 =
train_and_evaluate_decision_tree(
    x_train, y_train, x_test, y_test, best_chromosome, top_n=3)

classifier_top5, selected_top5, acc_top5 =
train_and_evaluate_decision_tree(
    x_train, y_train, x_test, y_test, best_chromosome, top_n=5)

classifier_top8, selected_top8, acc_top8 =
train_and_evaluate_decision_tree(
    x_train, y_train, x_test, y_test, best_chromosome, top_n=8)

classifier_all, selected_all, acc_all  = train_and_evaluate_decision_tree(
    x_train, y_train, x_test, y_test, best_chromosome,
top_n=len(ranked_features))
```

منظور از Accuracy یا دقت، نسبت پیش‌بینی‌های صحیح به کل نمونه‌ها است و F1_ Score، میانگین هارمونیک دقت و بازخوانی است که برای کلاس‌های نامتوازن مناسب‌تر است.

- ۳ ویژگی برتر : 'Profession', 'Spending_Score', 'Graduated'
- ۵ ویژگی برتر: 'Profession', 'Spending_Score', 'Graduated', 'Ever_Married' 'Gender'
- ۸ ویژگی برتر: 'Profession', 'Spending_Score', 'Graduated', 'Ever_Married' 'Gender', 'Age', 'Family_Size', 'Work_Experience'
- و همه ویژگی ها: 'Profession', 'Spending_Score', 'Graduated', 'Ever_Married' 'Gender', 'Age', 'Family_Size', 'Work_Experience', 'Var_1'

در نمودار زیر مقایسه Accuracy و F1_Score برای هر چهار تا مدل آورده شده است.



تصویر ۲- مقایسه دقت به اعزای ویژگی‌های مختلف

مدل با ۵ ویژگی برتر بالاترین میزان دقت را ارائه داد (۰.۳۲۳۶)، اما F1_Score آن نسبت به سایر مدل‌ها پایین‌تر بود.

F1_Score با افزایش ویژگی‌ها یه کم بهتر شد و در حالتی که از همه‌ی ویژگی‌ها استفاده شد، به ۰.۳۲۴۹ رسید.

Accuracy و F1_Score در همه حالتها مقدار نزدیکی به هم داشتند که نشان میدهد تعداد ویژگی ها بر روی عملکرد مدل تاثیر کمی دارد.

با بررسی و مقایسه نتایج مشخص شد که استفاده از ترکیب محدودتری از ویژگی‌های منتخب (۵ ویژگی برتر) عملکرد مدل را کمی بهتر یا حداقل در همان سطحی حفظ کرد که با استفاده از همه‌ی ویژگی‌ها به دست می‌آمد. به‌عبارت دیگر، الگوریتم ژنتیک (GA) ویژگی‌های مهم‌تر را شناسایی کرد و با کاهش تعداد ویژگی‌ها، عملکردی معادل یا گاهی بهتر از حالت استفاده از تمام ویژگی‌ها فراهم می‌کند.

مزایای استفاده از تعداد ویژگی‌های کمتر نسبت به استفاده از همه ویژگی‌ها:

- باعث کاهش پیچیدگی مدل شده و در نتیجه، مدل ساده‌تر و سریع‌تر آموزش می‌بیند.
- باعث کاهش ریسک overfitting: حذف ویژگی‌های غیرمؤثر یا کم‌اهمیت می‌تواند از یادگیری الگوهای تصادفی جلوگیری کند.
- کاهش هزینه و زمان محاسباتی: با داده‌های کمتر، سرعت پیش‌پردازش و آموزش بالاتر می‌رود.

اندازه‌ی جمعیت و نرخ جهش از مهم‌ترین پارامترها هستند؛ زیرا اگر اندازه جمعیت کوچک باشد، تنوع ژنتیکی کافی ایجاد نمی‌شود و الگوریتم در بهینه‌ی محلی گیر می‌کند و اگر بیش از حد بزرگ باشد، محاسبات زمان‌بر می‌شود. همچنین، نرخ جهش نقش کلیدی در ایجاد تنوع دارد؛ نرخ پایین موجب کاهش اکتشاف راه‌حل‌های جدید شده و نرخ بالا می‌تواند ساختار کروموزوم‌ها را مختل کند. روش انتخاب نیز با استفاده از تکنیک‌هایی مانند Roulette Wheel یا Tournament تأثیر قابل توجهی در حفظ یا از بین بردن تنوع ژنتیکی دارد.