به نام خدا

گزارش تحلیلی تمرین اول هوش محاسباتی

موضوع : الگوریتم ژنتیک

استاد: دکتر حسین کارشناس

دستیاران آموزش :

رضا برزگر – علی شاه زمانی – آرمان خلیلی

اعضای گروه :

یونس ایوبی راد

پویا اسفندانی

بخش اول: مبانی و مفاهیم الگوریتم ژنتیک (GA)

(1.1

الگوریتم‌های تکاملی (Evolutionary Algorithms - EAs) دسته‌ای از الگوریتم‌های بهینه‌سازی و جستجو هستند که از مفاهیم زیستی و تکاملی، مانند انتخاب طبیعی، جهش(Mutation) ، ترکیب (Crossover) و بقای شایسته‌ترین‌ها الهام گرفته شده‌اند. این الگوریتم‌ها برای حل مسائل پیچیده‌ای که در آن‌ها فضای جستجو بزرگ، غیرخطی یا چندوجهی است، به کار می‌روند. الگوریتم ژنتیک (Genetic Algorithm - GA) یکی از شناخته‌شده‌ترین زیرمجموعه‌های الگوریتم‌های تکاملی است.

برتری نسبت به یادگیری تقویتی:

- مقیاس‌پذیری بالا ES: به دلیل موازی‌سازی آسان (ارزیابی مستقل نمونه‌ها)، در مقایسه با RL که اغلب به محاسبات متوالی وابسته است، روی سخت‌افزارهای موازی بهتر عمل می‌کند .
- سادگی محاسباتی : برخلاف RL که به الگوریتم‌های پیچیده گرادیانی (مثل Q-Learning یا Policy Gradient) نیاز دارد، ES فقط به نمونه‌گیری تصادفی و میانگین‌گیری وزن‌دار وابسته است .
- پایداری در مسائل پیچیده ES: در مسائل با پاداش‌های پراکنده یا فضای عمل بزرگ) مانند بازی‌های Atari یا شبیه‌سازی‌های رباتیک (عملکرد پایدارتری نشان می‌دهد، زیرا به جای گرادیان محلی، جستجوی سراسری انجام می‌دهد .
- انعطاف‌پذیری: نیازی به تنظیم دقیق نرخ یادگیری یا پارامترهای تخفیف ندارد، که در RL چالش‌برانگیز است.

(1.2

الگوریتم ژنتیک یک روش بهینه‌سازی تکاملی است که با کروموزوم‌ها (مانند باینری یا اعداد) کار می‌کند، از ترکیب (Crossover) و جهش (Mutation) برای یافتن راه‌حل در فضای گسسته یا پیوسته استفاده می‌کند و بیشتر برای مسائل عمومی مثل زمان‌بندی مناسب است. در حالی که برنامه‌نویسی تکاملی (EP) روی تکامل ساختارها (مثل درخت برنامه‌ها) با تمرکز بر جهش متمرکز است و ترکیب کم اهمیت است، و برای تولید خودکار برنامه‌ها کاربرد دارد. از طرف دیگر، استراتژی‌های تکاملی (ES) برای بهینه‌سازی پیوسته عددی طراحی شده‌اند، با بردارهای واقعی و جهش‌های نرمال (با گام تکامل‌پذیر) کار می‌کنند، ترکیب کم است و در مسائل مهندسی دقیق عالی هستند.

(1.3

عملیات جهش :این عملیات زمانی رخ می‌دهد که به یک فرزند با مقداری بیتی (مانند یک رشته باینری) رسیده‌ایم و قصد داریم برخی از بیت‌های آن را به‌صورت تصادفی تغییر دهیم. برای مثال، فرض کنید فرزندی با مقدار 00100101 داریم. در این حالت، برای هر بیت احتمالی برای تغییر وجود دارد. با اعمال عملیات جهش، برخی از بیت‌ها به‌صورت تصادفی تغییر می‌کنند و رشته جدیدی مانند 00110101 به دست می‌آید.

عملیات ترکیب :این عملیات هنگامی انجام می‌شود که دو والد در نظر گرفته شده‌اند و هدف، تولید فرزندی جدید از ترکیب این دو والد باشد. برای این منظور، دو والد با استفاده از عملیات ترکیب ادغام می‌شوند تا فرزند جدیدی ایجاد شود. به‌عنوان مثال، دو رشته 111111 و 000000 را در نظر بگیرید. با انتخاب بخشی از والد اول و ترکیب آن با والد دوم، دو فرزند جدید تولید می‌شود؛ مثلاً از این دو والد می‌توان فرزندان 001111 و 110000 را به وجود آورد. با تغییر نقطه برش، امکان تولید فرزندان دیگری نیز فراهم می‌شود.

استفاده از این عملیات برای مقادیر غیرباینری :برای به‌کارگیری این دو عملیات در مقادیر غیرباینری، ابتدا باید این مقادیر را به شکل باینری تبدیل کنیم. برای نمونه، در مسئله کوله‌پشتی، حضور یا عدم حضور هر آیتم را با 0 و 1 نشان می‌دهیم و طول رشته را برابر با تعداد کل آیتم‌ها در نظر می‌گیریم. یا در مسئله هشت وزیر، هر ستون را به سه بیت تبدیل می‌کنیم تا یک رشته 24 بیتی ایجاد شود. سپس، عملیات جهش و ترکیب را روی این رشته 24 بیتیِ متشکل از 0 و 1 انجام می‌دهیم و در نهایت، این رشته 24 بیتی را به حالت غیرباینری بازمی‌گردانیم.

1.4)

در الگوریتم ژنتیک که بر رشته‌های بیتی عمل می‌کند، لازم است مجموعه‌ای از خواص تغییرناپذیری حفظ شود تا عملکرد الگوریتم بهینه باقی بماند. این خواص شامل موارد زیر است: نخست، تغییرناپذیری نسبت به جابجایی بیت‌ها، به این معنا که برای مثال، رشته‌های 1010 و 0101 در صورتی که ترتیب اهمیت نداشته باشد، باید برازندگی یکسانی داشته باشند. دوم، حفظ محدودیت‌ها، یعنی طول رشته نباید تغییر کند و همواره در فضای مجاز باقی بماند. سوم، تغییرناپذیری نسبت به مقیاس، به این صورت که الگوریتم باید با رشته‌های کوتاه یا بلند به یک شیوه عمل کند و اندازه رشته بر عملکرد آن تأثیر نگذارد. در نهایت، تغییرناپذیری نسبت به نمایش معادل، یعنی الگوریتم باید تشخیص دهد که دو رشته مانند 1111 و 0000، اگر با یک تبدیل معادل باشند، نباید به‌صورت جداگانه بررسی شوند.

این خواص تضمین می‌کنند که الگوریتم ژنتیک تنوع خود را حفظ کند، از راه‌حل‌های نادرست اجتناب ورزد، فضای جستجو را به‌طور مؤثر کاوش کند و با سرعت بیشتری به پاسخ بهینه دست یابد. برای مثال، اگر خاصیت جابجایی رعایت نشود، الگوریتم ممکن است روی یک الگوی خاص متمرکز شود و کارایی آن کاهش یابد.

1.5)

مقایسه زمان الگوریتم ژنتیک با جستجوی تصادفی:

الگوریتم ژنتیک با زمان O(n^3) برای رشته‌های بیتی با طول n عمل می‌کند، در حالی که جستجوی تصادفی با توزیع یکنواخت، به دلیل 2^n رشته ممکن، زمان O(2^n) دارد O(n^3) . (چندجمله‌ای) بسیار سریع‌تر از O(2^n) (نمایی) است، به ویژه برای n های بزرگ، زیرا الگوریتم ژنتیک جستجو را هدایت می‌کند، اما جستجوی تصادفی شانسی است.

عوامل مؤثر بر عملکرد:

- اندازه جمعیت : تنوع بیشتر، اما زمان‌برتر.
- نرخ جهش و ترکیب : تعادل برای جلوگیری از بهینه محلی یا تصادفی شدن.
- تابع برازندگی : هدایت‌کننده اصلی الگوریتم.
- فضای جستجو و شرایط اولیه :تأثیرگذار بر سرعت و دقت.

الگوریتم ژنتیک با تنظیم مناسب، کارایی بهتری نسبت به جستجوی تصادفی دارد.

بخش دوم: درک و حل مسائل با الگوریتم ژنتیک

2.1)

الف) 10 ژن

هر کروموزوم باید یک دور همیلتونی را نشان دهد. در یک دور همیلتونی با n شهر، دقیقاً n یال وجود دارد، زیرا هر شهر به شهر بعدی متصل می‌شود و در نهایت به مبدأ بازمی‌گردد.

با فرض n=10 شهر، هر کروموزوم به 10ژن نیاز دارد، زیرا یک مسیر بسته با 10 شهر شامل 10 یال است.

ب) 45 ژن یکتا

الفبای الگوریتم مجموعه ژن‌های منحصربه‌فرد ممکن را شامل می‌شود. هر ژن یک یال بدون جهت بین دو شهر را نشان می‌دهد. با n=10 شهر، تعداد یال‌های ممکن در یک گراف کامل (بدون جهت) از فرمول ترکیب محاسبه می‌شود:

تعداد یال = $n(n-1)/2$

تعداد کل ژن‌های یکتا (یال‌های بدون جهت ممکن) برابر با 45 است. هر ژن) مثل ('TI' یک اتصال منحصربه‌فرد را نشان می‌دهد و چون یال‌ها بدون جهت هستند، 'TI'و 'IT' یکسان تلقی می‌شوند.

2.2)

الف)

برازندگی x1 : 2-3-5+3+1-4-5+6 = 9

برازندگی x2 : 1-0-6+6+2-1-7+8 = 23

برزندگی x3 :5-8-2+1+2-9-3+2 = -16

برازندگی x4 : 4-9-0+2+5-8-1+4 = -19

مرتب سازی بر اساس برازندگی : X2 - X1 - X3 - X4

ب)

1)

حالت اول به وجود آوردن دو فرزند با one point cross over

6	5	4	1	3	5	3	2

8	7	1	2	6	6	0	1

رشته های به وجود آمده از one point cross over برابر است با 65416601 و 87123532

2)

2	3	9	2	1	2	8	5

8	7	1	2	6	6	0	1

رشته های حاصل از به وجود آمدن two point cross over برابر است با 23126685 و 87921201

3)

اگر بخواهیم با استفاده ازuniform crossover ترکیب را انجام دهیم باید به صورت رندوم هر کدام را از یکی از والد ها انتخاب کنیم

6	5	4	1	3	5	3	2

8	7	1	2	6	6	0	1

دو فرزند به وجود آمده برابر هستند با 87113632 و 65426501

ج)

جمعیت به وجود آمده جدید به همراه برازندگی آنها برابر هست.

15 = 6+5-4-2+6+5-0-1

18 = 8+7-1-1+3+6-3-2

1 = 2+3-1-2+6+6-8-5

6 = 8+7-9-2+1+2-0-1

17 = 6+5-4-1+6+6-0-1

15 = 8+7-1-2+3+5-3-2

میتوان گفت فرزندان جدید به عدد بالاتری میل میکنند و کلیت آنها به عدد بالاتری میل کرده ولی هیچ کدام از ماکسیموم حالت قبلی بهتر نشده (البته چون الگوریتم uniform crossover به صورت رندوم بود و در یکی دیگر از حالت ها میتوانست پاسخ خیلی خیلی بهتری بدهد) ولی برازندگی کلی جمعیت بهبود داشت و هیچ کدام از فرزندان به مقدار منفی ای میل نکردند.

د)

بهترین و بهینه ترین حالت ممکن برابر است با 99009900 که برازندگی ان برابر است با 36

ه)

خیر قطعا بدون عملگر جهش نمیتوان به بهترین پاسخ ممکن رسید میتوان پاسخ های بهتری را پیدا کرد ولی هرگز نمیتوان بدون استفاده از جهش به بهترین پاسخ ممکن رسید

2.3)

الف)

محاسبه برازندگی

X1 = 4

X2 =-1

X3 =-2

X4 = 7

ب)

جمع تمام برازندگی ها با عدد 3

X1 = 7

X2 = 2

X3 = 1

X4 = 10

پ)

حاصل جمع تمام برازندگی ها برابر است با 43 = 2*10+1*3+3*2+2*7

ت)

احتمال انتخاب شدن هر کدام از x1 برابر است با 7/43 واحتمال انتخاب شدن هر x2 برابر است با 2/42 و احتمال انتخاب شدن هر x3 برابر است با 1/43 و احتمال انتخاب شدن هر x4 = 10/43 و احتمال کل برابر است با

X1=14/43

X2=6/43

X3=3/43

X4=20/43

ث)

مزیت این تابع برازندگی این است که هرگز مقادیر منفی نمیگیرید

برازندگی مقادیر جدید برابر هستند با

X1 = 16

X2 = 1

X3 = 4

X4 = 49

حاصل جمع برازندگی ها برابر است با 145

و احتمال انتخاب برابر میشود با

32/145

3/145

12/145

98/145

ج)

فشار انتخاب به شدت افزایش می‌یابد به دلیل اینکه اختلاف اعدادی با برازندگی بیشتر و اعدادی با برازندگی کمتر خیلی خیلی زیاد میشود احتمال انتخاب شدن اعداد پایین تر کمتر شده است و جست و جو در محیط کاهش یافته و به همین دلیل جمعیت به سرعت بیشتری به بهینه ترین مقدار رسیده میل میکند و به دلیل کاوش نکردن در محیط به سرعت در یک بهینه محلی(که میتواند بهینه سراسری باشد) میل میکند.

بخش سوم

پیاده سازی الگوریتم ژنتیک برای انتخاب ویژگی و بهبود مدل طبقه بندی

مقدمه پروژه

هدف این پروژه، پیاده‌سازی و ارزیابی یک الگوریتم ژنتیک (Genetic Algorithm - GA) برای انتخاب بهینه ویژگی‌ها از مجموعه داده‌های مشتریان یک فروشگاه است تا دقت مدل‌های طبقه‌بندی بهبود یابد. الگوریتم ژنتیک با جستجوی هوشمند در فضای ویژگی‌ها، زیرمجموعه‌ای از ویژگی‌ها را انتخاب می‌کند که عملکرد مدل را به حداکثر برساند. در این پروژه، از ابزارهای پیش‌پردازش داده، تشخیص ناهنجاری‌ها و مدل طبقه‌بندی (درخت تصمیم) استفاده شده است. نتایج با معیارهایی مانند دقت (Accuracy)ارزیابی می‌شوند.

شرح برنامه پیاده سازی شده:

- بخش اول: خواندن و پیش پردازش داده

کتابخانه های استفاده شده :

Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, MinMaxScaler, StandardScaler
from sklearn.impute import KNNImputer, SimpleImputer
from sklearn.neighbors import LocalOutlierFactor
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import random
```

در این بخش، کتابخانه‌ها و ماژول‌های مورد نیاز برای اجرای پروژه وارد شده‌اند. کتابخانه pandas برای مدیریت و تحلیل داده‌ها به‌صورت جدولی و numpy برای عملیات عددی استفاده می‌شود. از matplotlib.pyplot برای ترسیم نمودارها و مصورسازی نتایج بهره گرفته شده است. ابزارهای پیش‌پردازش از sklearn.preprocessing شامل LabelEncoder برای تبدیل داده‌های متنی به عددی، MinMaxScaler و StandardScaler برای نرمال‌سازی داده‌ها هستند. برای مدیریت داده‌های گمشده، KNNImputer (پر کردن با نزدیک‌ترین همسایگان) و SimpleImputer(پر کردن با میانگین یا میانه) به کار رفته‌اند. الگوریتم LocalOutlierFactor برای شناسایی و حذف ناهنجاری‌ها، DecisionTreeClassifier به‌عنوان مدل پایه طبقه‌بندی، و accuracy_score برای ارزیابی دقت مدل استفاده شده‌اند. در نهایت، ماژول random برای تولید اعداد تصادفی در الگوریتم ژنتیک مورد نیاز است.

این بخش پایه و اساس پروژه را فراهم می‌کند و ابزارهای لازم برای پیش‌پردازش داده‌ها، پیاده‌سازی الگوریتم ژنتیک و ارزیابی مدل را در اختیار قرار می‌دهد.

Data preprocessing

```
# File paths for datasets
train_file_path = "Train.csv"
test_file_path = "Test.csv"

# Load datasets
train_df = pd.read_csv(train_file_path)
test_df = pd.read_csv(test_file_path)
train_df.head()
```

```
train_df.info()
```

Remove unused columns

```
# Drop 'ID' column
train_df.drop(columns=['ID'], inplace=True)
test_df.drop(columns=['ID'], inplace=True)
```

در این بخش، فایل‌های آموزشی (Train.csv) و آزمایشی (Test.csv) با pandas به صورت جداول داده (train_df و test_df) بارگذاری می‌شوند. سپس، ستون 'ID' به دلیل عدم ارتباط با ویژگی‌های مشتریان، از هر دو مجموعه حذف می‌شود. این کار با متد drop و inplace=True انجام می‌گیرد تا داده‌ها برای مراحل بعدی آماده شوند.

Encodeng

```
# Identify categorical and numerical columns
categorical_cols = train_df.select_dtypes(include=['object']).columns.tolist()
numerical_cols = train_df.select_dtypes(exclude=['object']).columns.tolist()
categorical_cols.remove('Segmentation')
```

```
categorical_cols
```

```
numerical_cols
```

```
# Encode categorical columns
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    train_df[col] = le.fit_transform(train_df[col])
    test_df[col] = le.transform(test_df[col])
    label_encoders[col] = le
```

در این بخش، ستون‌های داده به دو دسته تقسیم می‌شوند: ستون‌های دسته‌ای (متنی) با select_dtypes('object') در categorical_cols و ستون‌های عددی در numerical_cols شناسایی می‌شوند. ستون 'Segmentation' (متغیر هدف) از لیست دسته‌ای حذف می‌شود. سپس، ستون‌های دسته‌ای با LabelEncoderبه مقادیر عددی تبدیل می‌شوند؛ این کار با fit_transformبرای train_df و transform برای test_df انجام می‌شود و اشیاء کدگذار در label_encoders ذخیره می‌گردند.

Replace missing values

```
# Impute missing values in numerical columns with KNN
knn_imputer = KNNImputer(n_neighbors=5)
train_df[numerical_cols] = knn_imputer.fit_transform(train_df[numerical_cols])
test_df[numerical_cols] = knn_imputer.transform(test_df[numerical_cols])

# Impute missing values in categorical columns with most frequent
categorical_imputer = SimpleImputer(strategy='most_frequent')
train_df[categorical_cols] = categorical_imputer.fit_transform(train_df[categorical_cols])
test_df[categorical_cols] = categorical_imputer.transform(test_df[categorical_cols])
```

در این بخش، مقادیر گمشده در داده‌ها پر می‌شوند. برای ستون‌های عددی (numerical_cols) ، از KNNImputer با 5 همسایه نزدیک استفاده شده تا مقادیر گمشده در train_df با fit_transform و در test_df با transform پر شوند. برای ستون‌های دسته‌ای

(categorical_cols)، از SimpleImputer با استراتژی «بیش‌ترین تکرار (most frequent) »استفاده شده تا مقادیر گمشده در هر دو مجموعه داده جایگزین شوند. این کار داده‌ها را برای مراحل بعدی آماده می‌کند.

Scale numerical features

```
scaler = MinMaxScaler()
train_df[numerical_cols] = scaler.fit_transform(train_df[numerical_cols])
test_df[numerical_cols] = scaler.transform(test_df[numerical_cols])
```

در این بخش، ستون‌های عددی (numerical_cols) با MinMaxScaler مقیاس‌بندی می‌شوند تا مقادیر به بازه [0, 1] تبدیل شوند. این کار با fit_transform روی train_df و transform روی test_df انجام می‌شود تا داده‌ها برای مدل‌سازی یکنواخت شوند.

Remove Outlier

```
# Remove outliers with LOF
lof = LocalOutlierFactor(n_neighbors=30, contamination=0.05)
outlier_flags = lof.fit_predict(train_df[numerical_cols])
train_df = train_df[outlier_flags == 1]
```

در این بخش، ناهنجاری‌ها از داده‌های آموزشی (train_df) با استفاده از LocalOutlierFactor (LOF) حذف می‌شوند .با LOF 30 همسایه و نرخ آلودگی 0.05، نمونه‌های عددی ناهنجار را شناسایی می‌کند. پرچم‌های خروجی (outlier_flags) با fit_predict تولید شده و نمونه‌هایی که علامت 1 دارند (غیرناهنجار) حفظ می‌شوند تا داده‌ها پاک‌سازی شوند.

Split features and target

```
# Split features and target
x_train = train_df.drop(columns=['Segmentation'])
y_train = train_df['Segmentation']
x_test = test_df.drop(columns=['Segmentation'])
y_test = test_df['Segmentation']

# Number of features
num_features = x_train.shape[1]
```

در این بخش، داده‌ها به ویژگی‌ها و متغیر هدف تقسیم می‌شوند. در train_df و test_df ، ستون 'Segmentation' (هدف) جدا شده و در y_train و y_test ذخیره می‌شود. بقیه ستون‌ها به‌عنوان ویژگی‌ها در x_train و x_test قرار می‌گیرند. سپس، تعداد ویژگی‌ها با shape[1] محاسبه و در num_features ذخیره می‌شود تا برای الگوریتم ژنتیک استفاده شود.

- بخش 2 : پیاده سازی الگوریتم ژنتیک

Genetic Algorithm

Components of Genetic Algorithm

- binary chromosomes
- Create population

```
# Create initial population of binary chromosomes
def create_population(size, num_features):
    np.random.seed(10)
    return [np.random.randint(0, 2, num_features).tolist() for _ in range(size)]
```

در این بخش، جمعیت اولیه الگوریتم ژنتیک ایجاد می‌شود که نقطه شروع فرآیند بهینه‌سازی انتخاب ویژگی‌هاست.
تابعcreate_population دو پارامتر دریافت می‌کند: size (اندازه جمعیت) و num_features(تعداد ویژگی‌ها). هر کروموزوم یک رشته باینری به طول num_features است که هر بیت (0 یا 1) نشان‌دهنده عدم انتخاب یا انتخاب یک ویژگی است. با استفاده از np.random.randint(0, 2, num_features)، مقادیر تصادفی 0 و 1 برای هر ویژگی تولید می‌شود و این کار برای size کروموزوم تکرار می‌گردد. تابع tolist() آرایه‌ها را به لیست تبدیل می‌کند تا پردازش بعدی آسان‌تر شود. تنظیم np.random.seed(10) تضمین می‌کند که نتایج تصادفی قابل تکرار باشند. این جمعیت اولیه، تنوع لازم برای شروع جستجوی تکاملی را فراهم می‌کند و پایه‌ای برای ارزیابی و بهبود در نسل‌های بعدی است.

Fitness function

```
# Fitness function with optional penalty for feature count deviation
def fitness_function(chromosome, target_num_features=None):
    selected_features = [i for i in range(num_features) if chromosome[i] == 1]
    if len(selected_features) == 0:
        return 0
    X_train_sel = x_train.iloc[:, selected_features]
    X_test_sel = x_test.iloc[:, selected_features]
    model = DecisionTreeClassifier()
    model.fit(X_train_sel, y_train)
    predictions = model.predict(X_test_sel)
    accuracy = accuracy_score(y_test, predictions)
    if target_num_features:
        penalty = abs(len(selected_features) - target_num_features) * 0.05
        return accuracy - penalty
    return accuracy
```

در این بخش، تابع برازندگی (Fitness Function) پیاده‌سازی شده است که معیار اصلی ارزیابی کروموزوم‌ها در الگوریتم ژنتیک را تشکیل می‌دهد. تابع fitness_function یک کروموزوم (رشته باینری) و پارامتر اختیاری target_num_features (تعداد ویژگی‌های هدف) را دریافت می‌کند. ابتدا، اندیس ویژگی‌های انتخاب‌شده (بیت‌های 1 در کروموزوم) در لیست selected_features جمع‌آوری می‌شود. اگر هیچ ویژگی انتخاب نشود، امتیاز 0 بازگردانده می‌شود تا از کروموزوم‌های نامعتبر جلوگیری شود.

سپس، زیرمجموعه‌ای از داده‌های آموزشی (X_train_sel) و آزمایشی (X_test_sel) با استفاده از ویژگی‌های انتخاب‌شده استخراج می‌شود. یک مدل درخت تصمیم (DecisionTreeClassifier) روی داده‌های آموزشی آموزش داده شده و پیش‌بینی‌ها روی داده‌های آزمایشی انجام می‌شود. دقت مدل با accuracy_score محاسبه می‌گردد. اگر target_num_features مشخص شده باشد، جریمه‌ای (Penalty) بر اساس اختلاف تعداد ویژگی‌های انتخاب‌شده با مقدار هدف اعمال می‌شود (به ازای هر اختلاف، 0.05 کسر می‌شود) تا تعادل بین دقت و تعداد ویژگی‌ها حفظ شود. در غیر این صورت، تنها دقت مدل به‌عنوان امتیاز برازندگی بازگردانده می‌شود. این تابع، عملکرد هر کروموزوم را بر اساس توانایی آن در بهبود طبقه‌بندی مشتریان ارزیابی می‌کند.

Selection Method

```python
# Tournament selection method
def tournament_selection(population, scores, k=3):
    selected = random.choices(list(zip(population, scores)), k=k)
    return max(selected, key=lambda x: x[1])[0]

# Roulette wheel selection method
def roulette_wheel_selection(population, scores):
    total_fitness = sum(scores)
    if total_fitness == 0:
        return random.choice(population)
    pick = random.uniform(0, total_fitness)
    current = 0
    for chromosome, score in zip(population, scores):
        current += score
        if current > pick:
            return chromosome
    return population[-1]
```

در این بخش، دو روش انتخاب برای الگوریتم ژنتیک پیاده‌سازی شده‌اند که کروموزوم‌های برتر را برای تولید نسل بعدی تعیین می‌کنند.

1. انتخاب تورنمنتی(tournament_selection) : این تابع سه پارامتر دریافت می‌کند:(population (جمعیت کروموزوم‌ها)، scores(امتیازات برازندگی) و k (تعداد شرکت‌کنندگان در تورنمنت، پیش‌فرض 3) . با استفاده از random.choices، به‌صورت تصادفی k کروموزوم به همراه امتیازاتشان انتخاب می‌شوند. سپس، کروموزومی که بالاترین امتیاز برازندگی را دارد (با استفاده از max و کلید x[1] که امتیاز است)، به‌عنوان برنده انتخاب و بازگردانده می‌شود. این روش تعادلی بین تصادفی بودن و انتخاب بهترین‌ها ایجاد می‌کند.

2. انتخاب چرخ رولت(roulette_wheel_selection) : این تابع population و scores را دریافت می‌کند. ابتدا مجموع امتیازات برازندگی (total_fitness) محاسبه می‌شود. اگر این مقدار صفر باشد، یک کروموزوم به‌صورت تصادفی انتخاب می‌شود. در غیر این صورت، یک مقدار تصادفی (pick) بین 0 و مجموع امتیازات تولید می‌شود. سپس، امتیازات کروموزوم‌ها به‌تدریج جمع شده و اولین کروموزومی که مجموع تجمعی‌اش از pick بیشتر شود، انتخاب می‌گردد. اگر هیچ‌کدام انتخاب نشوند، آخرین کروموزوم بازگردانده می‌شود. این روش شانس انتخاب را متناسب با برازندگی هر کروموزوم تنظیم می‌کند.

این دو روش، مکانیزم‌هایی برای انتخاب والدین در الگوریتم ژنتیک فراهم می‌کنند که تنوع و همگرایی به سمت راه‌حل بهینه را متعادل می‌سازند.

Crossover Mechanism

```python
# Multi-point crossover between two parents
def multi_point_crossover(parent1, parent2, num_points=2):
    points = sorted(random.sample(range(1, num_features - 1), min(num_points, num_features - 2)))
    child1, child2 = parent1[:], parent2[:]
    for i in range(len(points)):
        if i % 2 == 0:
            child1[points[i]:] = parent2[points[i]:]
            child2[points[i]:] = parent1[points[i]:]
    return child1, child2


# Uniform crossover between two parents
def uniform_crossover(parent1, parent2):
    child1, child2 = parent1[:], parent2[:]
    for i in range(len(parent1)):
        if random.random() < 0.5:
            child1[i], child2[i] = child2[i], child1[i]
    return child1, child2
```

در این بخش، دو روش ترکیب (Crossover) برای تولید فرزندان جدید از والدین در الگوریتم ژنتیک پیاده‌سازی شده‌اند.

1. ترکیب چندنقطه‌ای(multi_point_crossover) : این تابع دو والد (parent1 و parent2) و تعداد نقاط برش (num_points، پیش‌فرض 2) را دریافت می‌کند. با استفاده از random.sample، نقاط برش تصادفی بین 1 تا numfeatures−1 انتخاب و مرتب می‌شوند. نسخه‌های کپی از والدین (child1 و child2) ایجاد شده و سپس، در نقاط برش زوج (شاخص‌های 0، 2، و غیره)، بخش‌های بعد از هر نقطه بین والدین جابه‌جا می‌شود. این روش تنوع را با حفظ بخش‌هایی از هر والد ایجاد می‌کند و دو فرزند جدید تولید می‌کند.

2. ترکیب یکنواخت(uniform_crossover) : این تابع نیز دو والد را دریافت می‌کند و دو فرزند (child1 و child2) را از کپی والدین می‌سازد. برای هر موقعیت در کروموزوم، با احتمال 50٪ (با random.random() < 0.5)، مقادیر بین دو فرزند جابه‌جا می‌شوند. این روش ترکیب تصادفی‌تر است و به هر ژن اجازه می‌دهد به‌صورت مستقل از یکی از والدین انتخاب شود، که تنوع بیشتری در فرزندان ایجاد می‌کند.

هر دو روش، اطلاعات ژنتیکی والدین را ترکیب می‌کنند تا کروموزوم‌های جدیدی برای نسل بعدی تولید شود، که در انتخاب ویژگی‌ها به یافتن زیرمجموعه‌های بهینه‌تر کمک می‌کند.

Mutation

```python
# Mutation of a chromosome based on mutation rate
def mutate(chromosome, mutation_rate=0.1):
    for i in range(len(chromosome)):
        if random.random() < mutation_rate:
            chromosome[i] = 1 - chromosome[i]
    return chromosome
```

در این بخش، عملیات جهش (Mutation) برای ایجاد تنوع در کروموزوم‌ها پیاده‌سازی شده است. تابع mutate یک کروموزوم و نرخ جهش(mutation_rate، پیش‌فرض 0.1) را دریافت می‌کند. برای هر بیت در کروموزوم، با احتمال mutation_rate (مثلاً 10٪)، یک مقدار تصادفی با random.random() تولید می‌شود؛ اگر این مقدار کمتر از نرخ جهش باشد، بیت تغییر می‌کند (از 0 به 1 یا از 1 به 0) که با عملیات chromosome[i] - 1انجام می‌شود. این فرآیند به‌صورت درون‌جا (in-place) روی کروموزوم اعمال شده و سپس کروموزوم تغییر یافته بازگردانده می‌شود. جهش به حفظ تنوع در جمعیت کمک می‌کند و از گیر افتادن الگوریتم در بهینه‌های محلی جلوگیری می‌کند، که در انتخاب ویژگی‌ها برای بهبود مدل طبقه‌بندی حیاتی است.

تابع genetic_algorithmالگوریتم ژنتیک را اجرا می‌کند. ابتدا با create_populationجمعیت اولیه ساخته می‌شود. در هر نسل (تا حداکثر generations)، امتیازات برازندگی با fitness_functionمحاسبه می‌شود. اگر تغییر بهترین امتیاز نسبت به نسل قبل کمتر از convergence_thresholdباشد، الگوریتم متوقف می‌شود. در غیر این صورت، والدین با روش انتخاب tournament) یا roulette)تعیین شده، با روش ترکیب multi_point) یا uniform)فرزندان تولید می‌شوند و با mutateجهش می‌یابند. جمعیت جدید جایگزین قبلی می‌شود. در پایان، بهترین کروموزوم انتخاب شده و اندیس ویژگی‌های 1 (انتخاب‌شده) بازگردانده می‌شود.

Genetic algorithm implementation

```
# Main genetic algorithm function
def genetic_algorithm(pop_size=20, generations=30, mutation_rate=0.1, selection_method='tournament',
                      crossover_method='multi_point', target_num_features=None, convergence_threshold=0.001):
    population = create_population(pop_size, num_features)
    best_score_prev = -1
    for generation in range(generations):
        scores = [fitness_function(ch, target_num_features) for ch in population]
        best_score = max(scores)
        if abs(best_score - best_score_prev) < convergence_threshold:
            print(f"Converged at generation {generation}")
            break
        best_score_prev = best_score
        new_population = []
        for _ in range(pop_size // 2):
            if selection_method == 'tournament':
                parent1 = tournament_selection(population, scores)
                parent2 = tournament_selection(population, scores)
            elif selection_method == 'roulette':
                parent1 = roulette_wheel_selection(population, scores)
                parent2 = roulette_wheel_selection(population, scores)
            if crossover_method == 'multi_point':
                child1, child2 = multi_point_crossover(parent1, parent2)
            elif crossover_method == 'uniform':
                child1, child2 = uniform_crossover(parent1, parent2)
            new_population.extend([mutate(child1, mutation_rate), mutate(child2, mutation_rate)])
        population = new_population
        if generation == generations:
            print(f"Reach maximum generation size")
    best_chromosome = max(population, key=lambda x: fitness_function(x, target_num_features))
    return [i for i in range(num_features) if best_chromosome[i] == 1]
```

تابع genetic_algorithm الگوریتم ژنتیک را اجرا می‌کند. ابتدا با create_population جمعیت اولیه ساخته می‌شود. در هر نسل (تا حداکثر generations)، امتیازات برازندگی با fitness_function محاسبه می‌شود. اگر تغییر بهترین امتیاز نسبت به نسل قبل کمتر از convergence_threshold باشد، الگوریتم متوقف می‌شود. در غیر این صورت، والدین با روش انتخاب (tournament یا roulette)تعیین شده، با روش ترکیب(multi_point یا uniform) فرزندان تولید می‌شوند و با mutate جهش می‌یابند. جمعیت جدید جایگزین قبلی می‌شود. در پایان، بهترین کروموزوم انتخاب شده و اندیس ویژگی‌های 1 (انتخاب‌شده) بازگردانده می‌شود.

- بخش سوم: اجرای الگوریتم ژنتیک و مدل طبقه بندی

Plot Feature Distribution

```
# Function to plot the distribution of selected features
def plot_feature_distribution(df, selected_features, title):
    plt.figure(figsize=(10, 6))
    for i in selected_features:
        plt.hist(df.iloc[:, i], bins=20, alpha=0.5, label=f'Feature {i}')
    plt.title(title)
    plt.xlabel('Value')
    plt.ylabel('Frequency')
    plt.legend()
    plt.show()
```

تابع plot_feature_distribution توزیع ویژگی‌های انتخاب‌شده را ترسیم می‌کند. با ورودی‌های df (داده‌ها)، selected_features (اندیس ویژگی‌ها) و title، برای هر ویژگی یک هیستوگرام با 20 بازه و شفافیت 0.5 رسم می‌شود. نمودار با عنوان، برچسب محورها و راهنما نمایش داده می‌شود تا توزیع مقادیر ویژگی‌ها مصورسازی شود.

Model Evaluation

```python
# Function to evaluate model accuracy using selected features
def evaluate_model(selected_features):
    X_train_sel = x_train.iloc[:, selected_features]
    X_test_sel = x_test.iloc[:, selected_features]
    model = DecisionTreeClassifier()
    model.fit(X_train_sel, y_train)
    predictions = model.predict(X_test_sel)
    return accuracy_score(y_test, predictions)
```

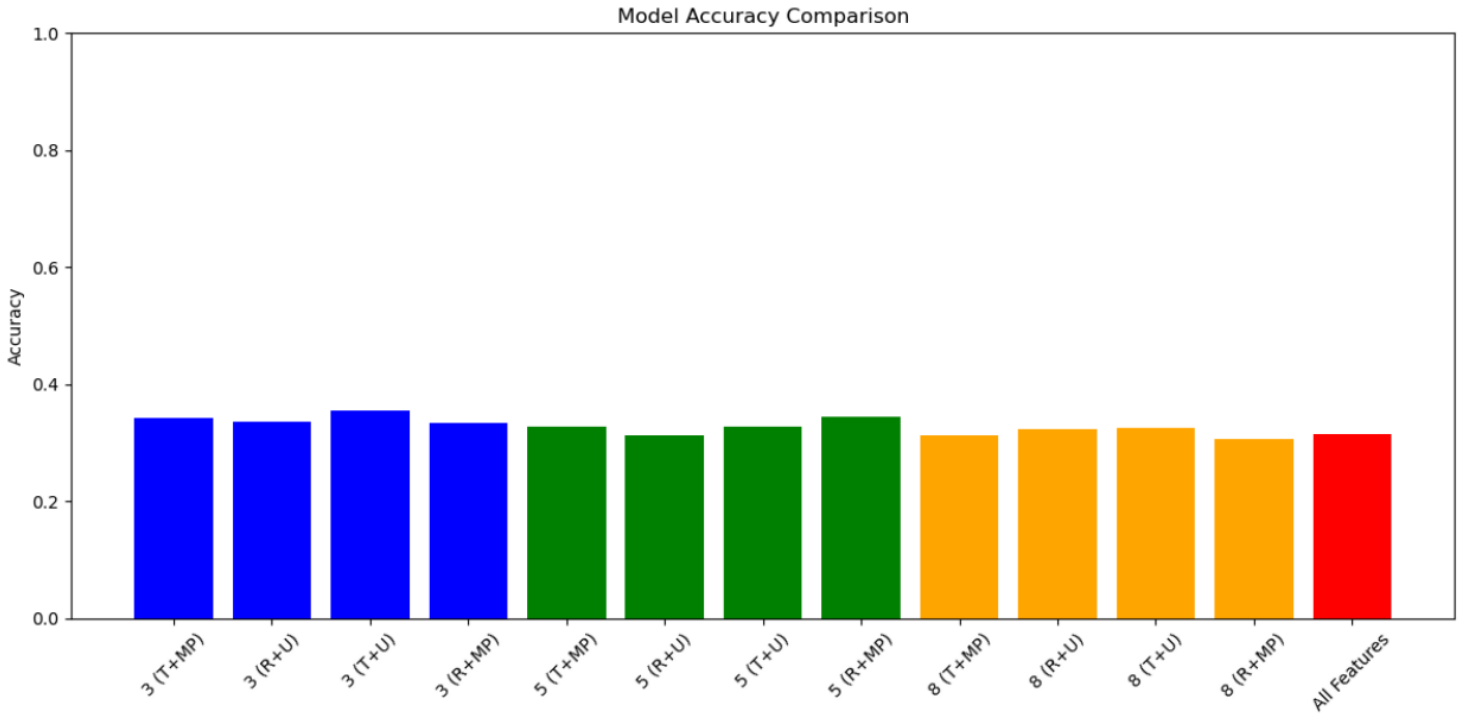
تابع evaluate_model دقت مدل را با ویژگی‌های انتخاب‌شده ارزیابی می‌کند. داده‌های آموزشی (X_train_sel) و آزمایشی (X_test_sel) با اندیس selected_features استخراج می‌شوند. مدل درخت تصمیم روی داده‌های آموزشی آموزش دیده و پیش‌بینی‌ها روی داده آزمایشی انجام می‌شود. در نهایت، دقت با accuracy_score محاسبه و بازگردانده می‌شود.

نتایج

- عملکرد الگوریتم ژنتیک برای انتخاب ویژگی‌ها

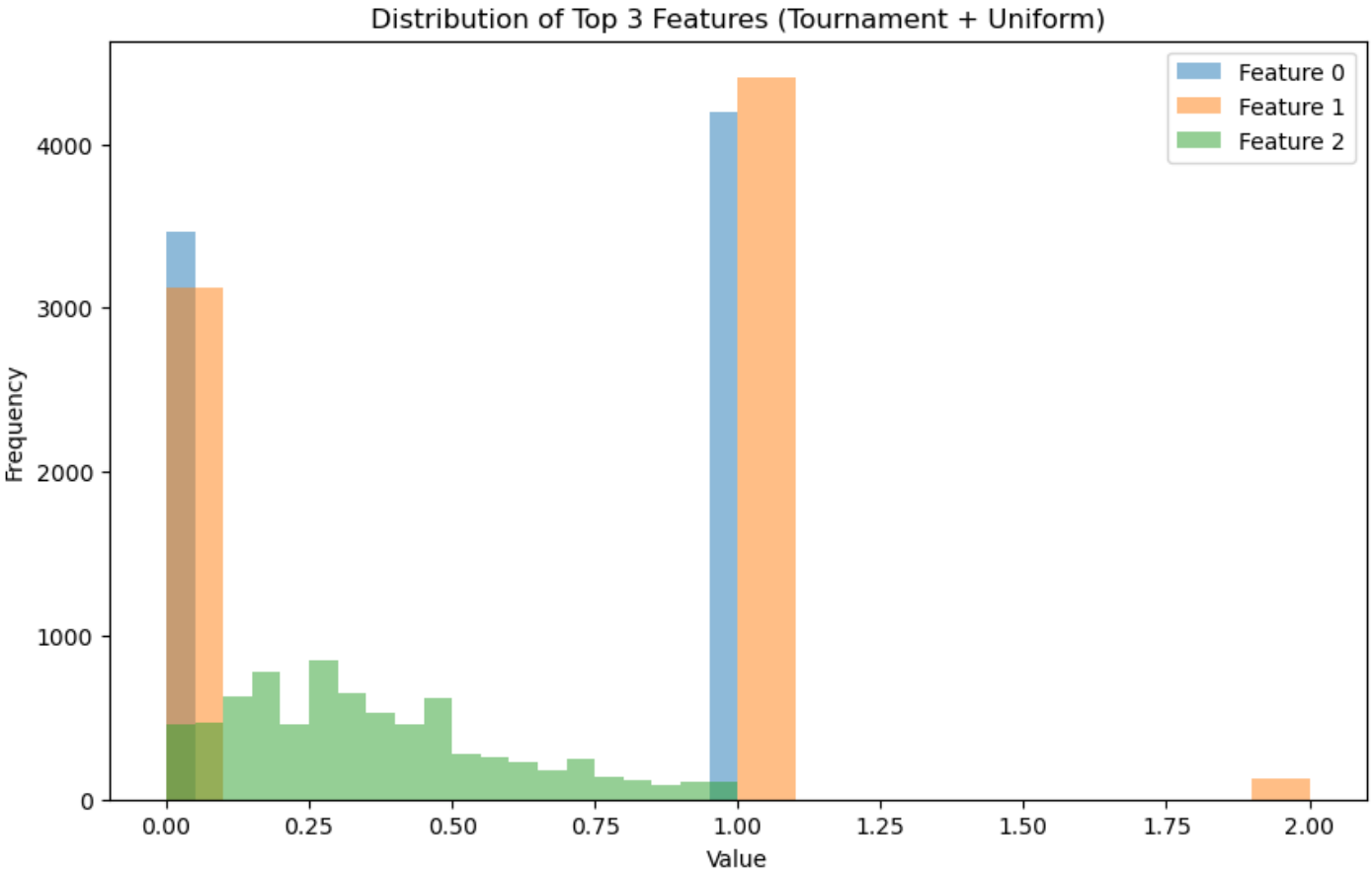
```
Results for 3 features:
Accuracy with Tournament + Multi-point: 0.3418
Accuracy with Roulette + Uniform: 0.3357
Accuracy with Tournament + Uniform: 0.3559
Accuracy with Roulette + Multi-point: 0.3342

Results for 5 features:
Accuracy with Tournament + Multi-point: 0.3285
Accuracy with Roulette + Uniform: 0.3129
Accuracy with Tournament + Uniform: 0.3278
Accuracy with Roulette + Multi-point: 0.3456

Results for 8 features:
Accuracy with Tournament + Multi-point: 0.3137
Accuracy with Roulette + Uniform: 0.3236
Accuracy with Tournament + Uniform: 0.3247
Accuracy with Roulette + Multi-point: 0.3076

Accuracy with all features: 0.3140
```

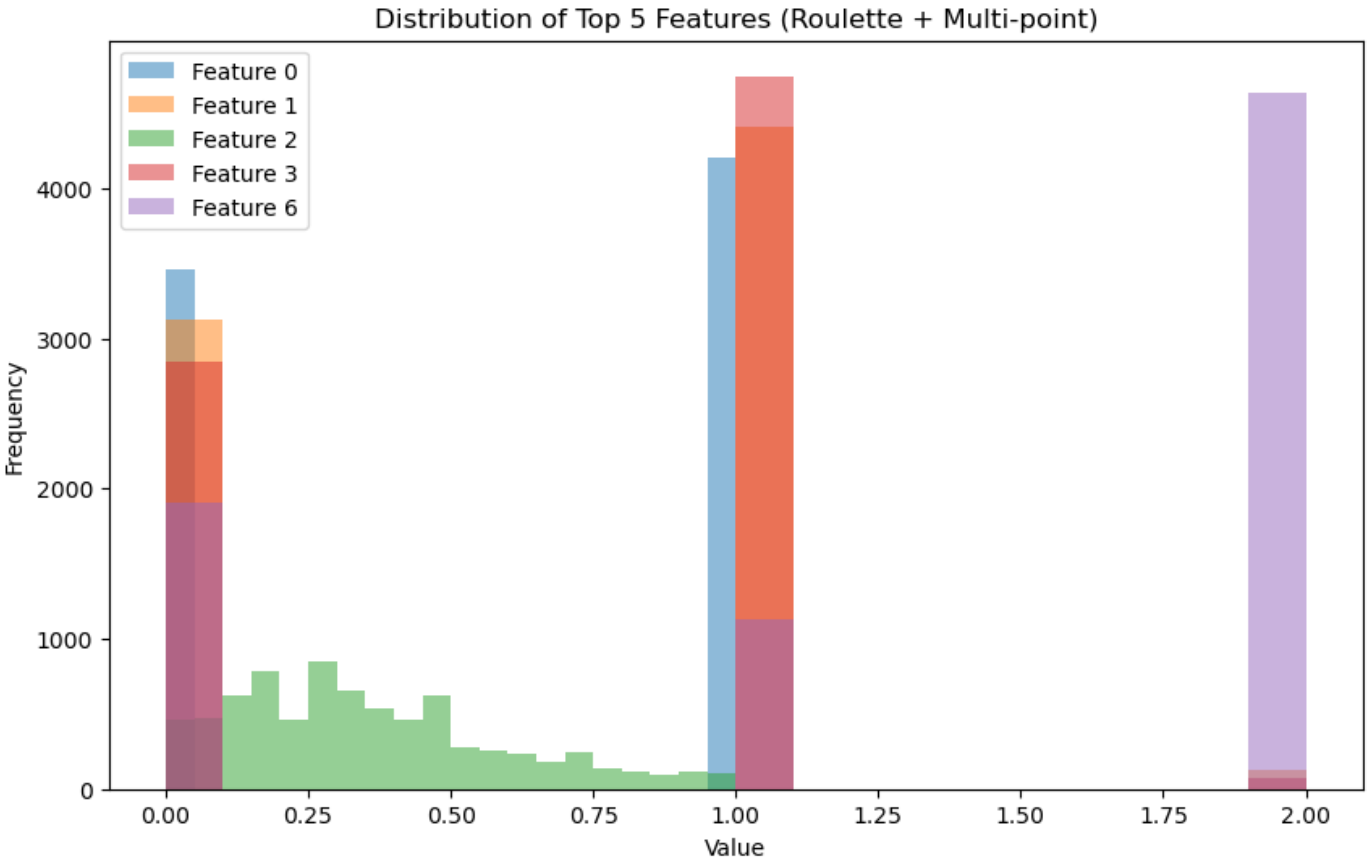
نتایج عملکرد الگوریتم ژنتیک برای انتخاب ویژگی‌ها با ترکیب روش‌های انتخاب (تورنمنتی و چرخ رولت) و ترکیب (چندنقطه‌ای و یکنواخت) ارائه شده است. دقت مدل درخت تصمیم برای 3، 5 و 8 ویژگی انتخاب‌شده و همچنین همه ویژگی‌ها محاسبه شده است. بالاترین دقت برای 3 ویژگی (0.3559) با انتخاب تورنمنتی و ترکیب یکنواخت، برای 5 ویژگی (0.3456) با چرخ رولت و ترکیب چندنقطه‌ای، و برای 8 ویژگی (0.3247) با تورنمنتی و ترکیب یکنواخت به دست آمده است. دقت با همه ویژگی‌ها (0.3140) نشان می‌دهد که انتخاب زیرمجموعه ویژگی‌ها در برخی موارد عملکرد بهتری نسبت به استفاده از همه ویژگی‌ها دارد

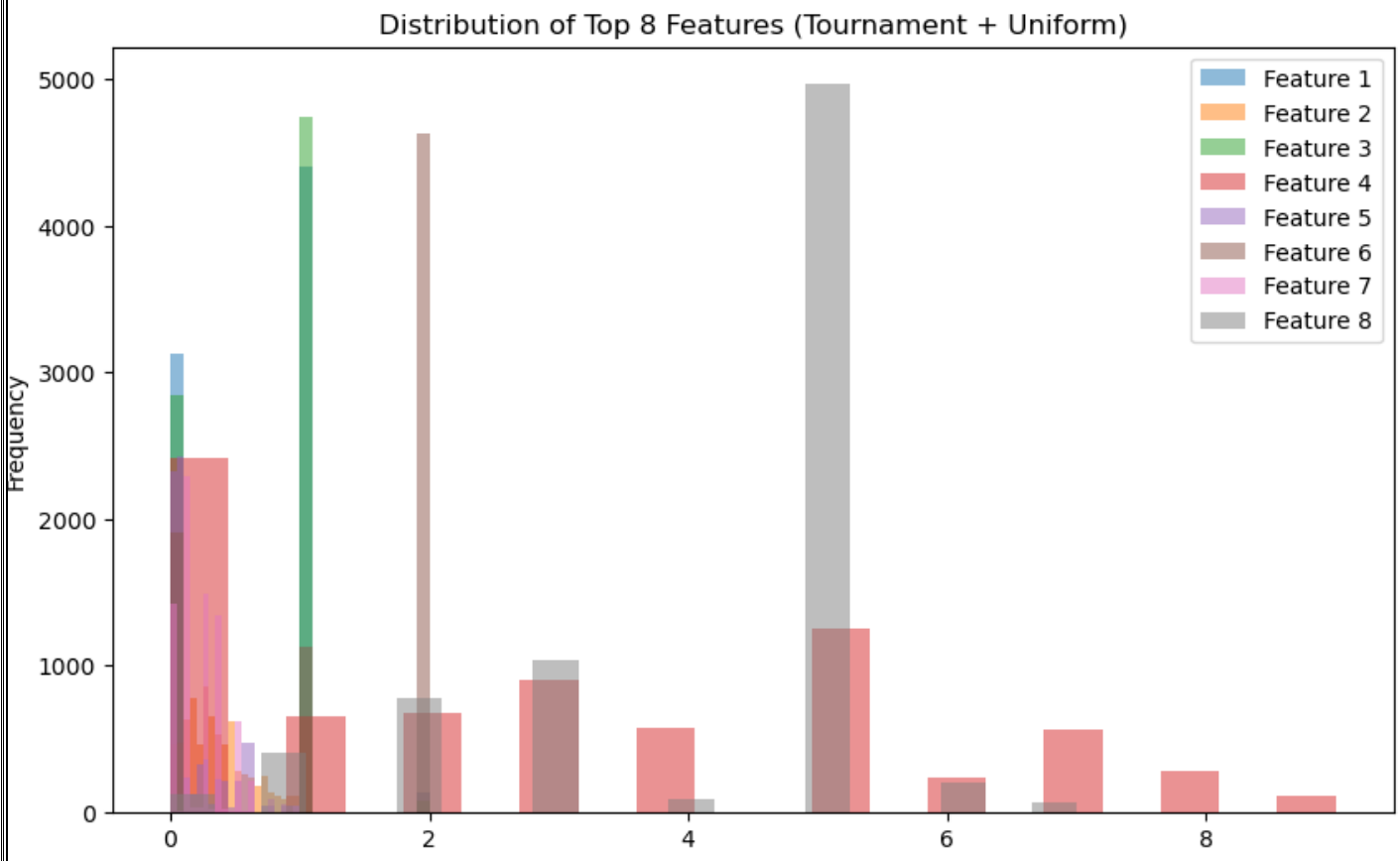


- بهترین ترکیب :
  - 3 ویژگی برتر
  - استراتژی انتخاب Tournoment
  - عملگر ترکیب uniform

- نتایج استخراج ویژگی های منتخب و رسم نمودار توزیع آنها


Distribution of Top 3 Features (Tournament + Uniform)
Feature 0
Feature 1
Feature 2
Frequency
Value
0
1000
2000
3000
4000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Distribution of Top 5 Features (Roulette + Multi-point)
Feature 0
Feature 1
Feature 2
Feature 3
Feature 6
Frequency
Value
0
1000
2000
3000
4000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00



سوالات :

1- کدام مجموعه ی ویژگی بهترین عملکرد طبقه بندی را داشت ؟

Gender , EverMarrage , AGE

2- آیا استفاده از GA برای انتخاب ویژگی باعث بهبود مدل شد، یا عملکرد مشابه ی با همه ی ویژگیها داشت؟

بله باعث بهبود مدل شد و در تمام تست ها درصد دقت مدل پس از اعمال الگوریتم بالاتر از مقدار اعمال همه ی ویژگی ها بود

3- مزایا و معایب استفاده از تعداد ویژگیهای کمتر در مقایسه با همه ی ویژگی ها چیست؟

استفاده از تعداد ویژگی‌های کمتر در مقایسه با همه ویژگی‌ها مزایایی مثل کاهش پیچیدگی، افزایش سرعت، بهبود تفسیرپذیری، و حذف نویز دارد، اما معایبی مثل از دست دادن اطلاعات مفید، وابستگی به الگوریتم انتخاب، و کاهش انعطاف‌پذیری نیز به همراه دارد.

4- کدام پارامترهای GA اندازه ی جمعیت، روش انتخاب، نرخ جهش بیشترین تأثیر را بر عملکرد داشتند؟

اندازه جمعیت (به دلیل تنوع کم با 20) و نرخ جهش (به دلیل همگرایی سریع) بیشترین تأثیر را داشتند .روش انتخاب تأثیر کمتری نشان داد، چون دقت‌ها با tournament و roulette نزدیک بودند.