به نام خدا

پروژه دوم هوش مصنوعی

توضیح فایل :main

۱ .توابع تعریف شده:

read_input(file_path)

- **هدف:**

این تابع فایل متنی‌ای که مسیرش از طریق ورودی file_pathمشخص شده است را می‌خواند و داده‌ها را به دو بخش تفکیک می‌کند:

1. **بازده‌های مورد انتظار :(Expected Returns)** لیستی از اعداد که در خط اول فایل متنی قرار دارند.
2. **ماتریس کوواریانس :(Covariance Matrix)** ماتریسی که در خطوط بعدی فایل ذخیره شده و بیانگر روابط همبستگی بین دارایی‌ها است.

- **گام‌های اصلی عملکرد:**

1. **خواندن فایل :**فایل مشخص‌شده باز می‌شود و تمامی خطوط آن در متغیر linesذخیره می‌شود.
2. **تجزیه خط اول :**خط اول فایل متنی که حاوی بازده‌های مورد انتظار است، با استفاده از splitتجزیه شده و به نوع داده‌ای floatتبدیل می‌شود.
3. **ساخت ماتریس کوواریانس:**

بقیه خطوط (خطوط دوم به بعد) به صورت ماتریس دوبعدی با استفاده از تبدیل هر خط به آرایه‌ای از اعداد شناور (float) و سپس استفاده از numpyبرای تبدیل به آرایه‌ی دوبعدی ذخیره می‌شوند.

- **ورودی:**
  - file_path: رشته‌ای که مسیر فایل متنی مورد نظر را نشان می‌دهد.
- **خروجی:**
  - expected_returns: لیستی شامل مقادیر بازده‌های مورد انتظار.
  - covariance_matrix: یک آرایه دوبعدی (ماتریس) که بیانگر ماتریس کوواریانس است.

---

۲ .متغیرهای کلیدی:

- **file_path**: مسیر فایل متنی که اطلاعات از آن خوانده می‌شود. در اینجا مقدار آن "text.txt"است.
- **expected_returns**: لیست اعداد شناور که بیانگر بازده مورد انتظار برای هر دارایی در پرتفوی است.

- **covariance_matrix**:ماتریس کوواریانس که روابط همبستگی بازده‌های دارایی‌ها را نشان می‌دهد.

---

۳ .ساختار فایل ورودی:(text.txt)

- **خط اول :**شامل بازده‌های مورد انتظار دارایی‌ها به صورت اعداد جداشده با فاصله.
  - مثال0.05 0.03 0.02 :
- **خطوط بعدی :**ماتریس کوواریانس به صورت اعداد جداشده با فاصله، به ازای هر خط یک ردیف از ماتریس.

---

۴ .خروجی چاپ‌شده:

- **expected_returns**:بازده مورد انتظار دارایی‌ها که از فایل استخراج شده.
- **covariance_matrix**:ماتریس کوواریانس که بیانگر روابط همبستگی بین دارایی‌ها است.

.کاربرد:

این بخش برای خواندن و آماده‌سازی داده‌های ورودی استفاده می‌شود و داده‌ها را به شکلی که برای الگوریتم‌های مختلف)مثل Beam Search یا (Genetic Algorithm قابل استفاده باشد، آماده می‌کند.

فایل initial

۱ .توابع تعریف شده:

generate_initial_portfolio(num_assets, random_state=42)

- **هدف:**

تولید یک آرایه از وزن‌های تصادفی برای دارایی‌ها، به گونه‌ای که مجموع این وزن‌ها برابر با ۱ باشد (تخصیص کامل سرمایه).

- **گام‌های عملکرد:**

1. **تنظیم بذر تصادفی:**

تابع از random.seed(random_state)استفاده می‌کند تا نتایج تصادفی تولیدشده قابل بازتولید باشند.

2. **تولید اعداد تصادفی:**

یک لیست از num_assetsمقدار تصادفی ایجاد می‌شود که نشان‌دهنده وزن‌های خام برای هر دارایی است.

3. **نرمال‌سازی:**

مجموع مقادیر تصادفی تولیدشده محاسبه می‌شود و هر مقدار تقسیم بر این مجموع می‌شود تا مجموع وزن‌ها برابر با ۱ شود.

- **ورودی:**
  - num_assets: تعداد دارایی‌ها.
  - random_state: مقدار بذر تصادفی برای تولید مقادیر ثابت. پیش‌فرض ۴۲ است.
- **خروجی:**
  - portfolio: یک لیست شامل وزن‌های نرمال‌شده برای دارایی‌ها.

---

۲ .متغیرها:

- **num_assets**:

تعداد دارایی‌ها که با استفاده از طول متغیر expected_returns تعیین می‌شود.

  - مقدار expected_returns از فایل دیگر (main.py) وارد شده است و شامل بازده مورد انتظار هر دارایی است.

- **initial_portfolio**:

پرتفوی اولیه تولیدشده شامل وزن‌های تصادفی برای دارایی‌ها است که مجموع این وزن‌ها برابر با ۱ می‌باشد.

---

۴ .کاربرد:

این تابع برای ایجاد یک مقدار اولیه تصادفی از وزن‌های دارایی‌ها استفاده می‌شود. این مقادیر به الگوریتم‌های مختلف بهینه‌سازی داده می‌شود تا به‌عنوان نقطه شروع برای جستجوی بهترین پرتفوی عمل کنند.

---

۵ .نکات مهم:

- با تغییر مقدار random_state، وزن‌های تصادفی متفاوتی تولید خواهد شد.
- این تابع تضمین می‌کند که وزن هر دارایی مثبت است و مجموع وزن‌ها دقیقاً برابر با ۱ می‌باشد.

فایل fitness

۱ .توابع تعریف شده

calculate_fitness(weights, expected_returns, covariance_matrix)

- **هدف:**

محاسبه مقدار فیتنس پرتفوی، که معمولاً به‌عنوان نسبت **بازده به ریسک** تعریف می‌شود.

- **ورودی‌ها:**
  - weights: وزن‌های دارایی‌ها (لیستی از مقادیر تخصیص به هر دارایی، مجموع آن‌ها باید ۱ باشد).
  - expected_returns: لیستی از بازده‌های مورد انتظار برای هر دارایی.
  - covariance_matrix: ماتریس کوواریانس بازده‌ها، که وابستگی بین دارایی‌ها را نشان می‌دهد.
- **عملکرد:**

1. از تابع calculate_portfolio_return برای محاسبه بازده پرتفوی استفاده می‌کند.
2. از تابع calculate_portfolio_risk برای محاسبه ریسک پرتفوی استفاده می‌کند.
3. نسبت بازده به ریسک محاسبه می‌شود:
   - اگر ریسک صفر باشد، فیتنس برابر با ۰ برگردانده می‌شود (برای جلوگیری از تقسیم بر صفر).

- **خروجی:**
  - مقدار فیتنس، که نشان‌دهنده کیفیت پرتفوی است.

---

calculate_portfolio_risk(weights, covariance_matrix)

- **هدف:**

محاسبه **ریسک پرتفوی** با استفاده از وزن‌های دارایی و ماتریس کوواریانس.

- **فرمول:**
  - w: بردار وزن‌ها.
  - C: ماتریس کوواریانس.
- **عملکرد:**

1. وزن‌ها را به آرایه NumPy تبدیل می‌کند.
2. از عملیات ماتریسی برای محاسبه ریسک استفاده می‌کند:

- ابتدا حاصل‌ضرب داخلی وزن‌ها محاسبه می‌شود.
- سپس وزن‌ها دوباره در این مقدار ضرب می‌شوند.

3. مقدار نهایی ریشه‌گیری می‌شود تا انحراف معیار به‌دست آید.

- **خروجی:**
  - یک عدد که بیانگر ریسک پرتفوی است.

---

calculate_portfolio_return(weights, expected_returns)

- **هدف:**

محاسبه **بازده پرتفوی** با توجه به وزن‌ها و بازده‌های مورد انتظار.

- **فرمول:**
  - w: بردار وزن‌ها.
  - r: بردار بازده‌های مورد انتظار.
- **عملکرد:**

1. از ضرب داخلی (np.dot) وزن‌ها و بازده‌های مورد انتظار استفاده می‌کند.
2. خروجی مقدار بازده کلی پرتفوی است.

- **خروجی:**
  - یک عدد که بازده کلی پرتفوی را نشان می‌دهد.

۲

ریسک پرتفوی:

portfolio_risk = calculate_portfolio_risk(weights, covariance_matrix)

بازده پرتفوی:

portfolio_return = calculate_portfolio_return(weights, expected_returns)

فیتنس:

fitness = calculate_fitness(weights, expected_returns, covariance_matrix)

---

۳ .کاربرد:

- **ریسک پرتفوی :**محاسبه میزان نوسانات کل پرتفوی.
- **بازده پرتفوی :**محاسبه میانگین بازده مورد انتظار.
- **فیتنس :**معیاری برای ارزیابی کیفیت پرتفوی، که معمولاً در الگوریتم‌های بهینه‌سازی مانند **Genetic Algorithm**استفاده می‌شود.

---

۴ .نکات مهم:

- وزن‌های دارایی‌ها باید نرمال‌شده باشند (مجموع آن‌ها برابر با ۱ باشد).
- اگر ریسک برابر با صفر باشد، فیتنس به‌صورت پیش‌فرض صفر برگردانده می‌شود.

فایل SA

این کد یک پیاده‌سازی از الگوریتم **Simulated Annealing (SA)**است که برای حل مسائل بهینه‌سازی استفاده می‌شود. هدف این کد، پیدا کردن بهترین ترکیب وزنی دارایی‌ها در یک پرتفوی با بیشترین بازده نسبت به ریسک است.

---

۱ .توضیح توابع و منطق کلی

تابع simulated_annealing

- **هدف:**

بهینه‌سازی پرتفوی با استفاده از الگوریتم .Simulated Annealing این الگوریتم به تقلید از فرآیند سرد شدن مواد در متالورژی، سعی می‌کند به مرور زمان به جواب بهینه برسد.

- **ورودی‌ها:**
  - expected_returns:بازده‌های مورد انتظار برای هر دارایی.
  - covariance_matrix:ماتریس کوواریانس بازده‌ها.
  - initial_portfolio:پرتفوی اولیه شامل وزن‌های تخصیص به هر دارایی.
  - initial_temperature:دمای اولیه برای فرآیند (پیش‌فرض: ۱۰۰۰).
  - cooling_rate:نرخ کاهش دما (پیش‌فرض: ۰.۹۹).
  - max_iterations:حداکثر تعداد تکرار (پیش‌فرض: ۱۰,۰۰۰).

- random_state: مقدار بذر تصادفی برای تکرارپذیری.

- **مراحل اجرا:**

1. **مقداردهی اولیه:**
   - پرتفوی اولیه و فیتنس آن محاسبه می‌شوند.
   - دمای اولیه تنظیم می‌شود.
2. **تکرارها:(Loop)**
   - یک پرتفوی جدید با تغییر تصادفی مقادیر وزنی تولید می‌شود:

```
new_portfolio = np.array(current_portfolio) + np.random.uniform(-0.1, 0.1,
len(current_portfolio))
```

این مقادیر نرمال‌سازی و مثبت می‌شوند:

```
new_portfolio = np.abs(new_portfolio)
new_portfolio /= new_portfolio.sum()
```

   - فیتنس پرتفوی جدید محاسبه می‌شود.
   - اگر پرتفوی جدید بهتر باشد (delta_fitness > 0) یا شرایط پذیرش برای پرتفوی ضعیف‌تر فراهم باشد (بر اساس دما:(

```
np.random.rand() < np.exp(delta_fitness / temperature)
```

پرتفوی جدید پذیرفته می‌شود.

3. **به‌روزرسانی بهترین جواب:**
   - اگر فیتنس پرتفوی جاری از بهترین فیتنس قبلی بهتر باشد، بهترین پرتفوی به‌روزرسانی می‌شود.
4. **کاهش دما:**

```
temperature *= cooling_rate
```

دما با نرخ کاهش دما (Cooling Rate) کاهش می‌یابد.

5. **خاتمه:**
   - الگوریتم زمانی خاتمه می‌یابد که دما بسیار کم شود یا تعداد تکرارها تمام شود.

- **خروجی‌ها:**
  - best_portfolio: بهترین پرتفوی پیدا شده.
  - best_fitness: بهترین مقدار فیتنس.

- fitness_history: لیست فیتنس‌های بهترین پرتفوی در طول تکرارها.

---

۲ .جزئیات مربوط به متغیرها

new_portfolio

پرتفوی جدید با افزودن نویز تصادفی به پرتفوی جاری تولید می‌شود. سپس:

- منفی‌ها به مقادیر مثبت تبدیل می‌شوند.
- وزن‌ها نرمال‌سازی می‌شوند تا مجموعشان برابر ۱ باشد.

delta_fitness

تفاوت بین فیتنس پرتفوی جدید و پرتفوی جاری است. بر اساس این مقدار و دما، تصمیم به پذیرش پرتفوی جدید گرفته می‌شود.

temperature

دمای الگوریتم است که با کاهش تدریجی، احتمال پذیرش حالت‌های کمتر بهینه را کاهش می‌دهد.

---

۳ .توضیح خط‌ها

خطوط کلیدی

- **محاسبه فیتنس پرتفوی:**

```
current_fitness = calculate_fitness(current_portfolio, expected_returns, covariance_matrix)
```

این خط فیتنس پرتفوی جاری را محاسبه می‌کند.

- **احتمال پذیرش پرتفوی ضعیف‌تر:**

```
np.random.rand() < np.exp(delta_fitness / temperature)
```

این شرط احتمال پذیرش پرتفوی جدید با فیتنس کمتر را تعیین می‌کند، که وابسته به مقدار دما است. در دماهای بالا، پذیرش حالت‌های ضعیف‌تر محتمل‌تر است.

- **نرمال‌سازی وزن‌ها:**

```
new_portfolio = np.abs(new_portfolio)
new_portfolio /= new_portfolio.sum()
```

این اطمینان می‌دهد که مقادیر وزنی مثبت و مجموع آن‌ها برابر با ۱ است.

---

۵ .کاربرد

- این الگوریتم برای مسائل **بهینه‌سازی غیرخطی** مانند **مدیریت سرمایه‌گذاری** استفاده می‌شود.
- به دلیل پذیرش حالت‌های کمتر بهینه در ابتدا، از گیر افتادن در کمینه‌های محلی جلوگیری می‌کند.

---

۶ .نمودار تاریخچه فیتنس

تابع plot_fitness_historyبرای نمایش تغییرات فیتنس در طول تکرارها استفاده می‌شود. این نمودار نشان می‌دهد که چگونه الگوریتم به تدریج به سمت جواب بهینه همگرا می‌شود.

فایل Beam

این کد پیاده‌سازی الگوریتم **Beam Search**برای یافتن بهترین ترکیب وزنی در یک پرتفوی سرمایه‌گذاری است. هدف آن، پیدا کردن ترکیب بهینه‌ای است که حداکثر بازده نسبت به ریسک را داشته باشد.

---

۱ .توضیح کلی الگوریتمBeam Search

Beam Searchالگوریتمی است که در آن مجموعه‌ای از کاندیداها (beam) به طور همزمان بررسی می‌شوند و در هر مرحله، تنها تعداد محدودی از بهترین گزینه‌ها برای ادامه جستجو انتخاب می‌شوند. این الگوریتم در مقایسه با روش‌های دیگر مانندSimulated Annealing ، بهینه‌سازی چندین گزینه به طور همزمان را امکان‌پذیر می‌کند.

---

۲ .ساختار کلی تابع

ورودی‌ها

- :expected_returnsلیستی از بازده‌های مورد انتظار برای هر دارایی.
- :covariance_matrixماتریس کوواریانس بازده‌ها.
- :beam_widthتعداد کاندیداهای موازی که در هر مرحله بررسی می‌شوند (پیش‌فرض: ۵).
- :max_iterationsحداکثر تعداد تکرار الگوریتم.
- :random_stateمقدار بذر تصادفی برای اطمینان از تکرارپذیری.

خروجی‌ها

- :best_portfolioبهترین پرتفوی پیدا شده.
- :best_fitnessبهترین مقدار فیتنس.

- fitness_history: لیستی از مقادیر بهترین فیتنس در هر مرحله.

---

۳ .مراحل اجرا

۱ .مقداردهی اولیه

- ابتدا، پرتفوی‌های اولیه به تعداد beam_width تولید می‌شوند:

```
beam = [generate_initial_portfolio(num_assets, random_state + i) for i in
range(beam_width)]
```

- مقدار فیتنس این پرتفوی‌ها محاسبه می‌شود:

```
fitness_scores = [calculate_fitness(p, expected_returns, covariance_matrix) for p in beam]
```

- بهترین پرتفوی و مقدار فیتنس اولیه تعیین می‌شوند:

```
best_portfolio = beam[np.argmax(fitness_scores)]
best_fitness = max(fitness_scores)
```

---

۲ .تکرار برای بهینه‌سازی

- در هر تکرار:

1. یک **beam جدید** ایجاد می‌شود:
   - برای هر پرتفوی در beam فعلی، پرتفوی‌های جدیدی با نویز تصادفی تولید می‌شود:

```
new_portfolio = np.array(portfolio) + np.random.uniform(-0.1, 0.1, num_assets)
```

مقادیر منفی حذف و وزن‌ها نرمال‌سازی می‌شوند:

```
new_portfolio = np.abs(new_portfolio)
new_portfolio /= new_portfolio.sum()
```

2. **انتخاب بهترین beam_width پرتفوی جدید:**
   - فیتنس تمامی پرتفوی‌های جدید محاسبه شده و beam_width پرتفوی با بیشترین مقدار فیتنس انتخاب می‌شوند:

```
top_indices = np.argsort(new_fitness_scores)[-beam_width:]
beam = [new_beam[i] for i in top_indices]
```

3. **به‌روزرسانی بهترین جواب:**

- اگر بهترین فیتنس در بین پرتفوی‌های جدید بهتر از بهترین فیتنس کلی باشد، بهترین جواب به‌روزرسانی می‌شود:

```
if max(fitness_scores) > best_fitness:
    best_fitness = max(fitness_scores)
    best_portfolio = beam[np.argmax(fitness_scores)]
```

---

۳ .خاتمه الگوریتم

- الگوریتم پس از تعداد مشخصی از تکرارها (max_iterations) خاتمه می‌یابد.
- بهترین پرتفوی و تاریخچه فیتنس بازگردانده می‌شوند:

```
return best_portfolio, best_fitness, fitness_history
```

---

۴ .توضیح خط‌ها

خطوط مهم:

- **تولید پرتفوی جدید:**

```
new_portfolio = np.array(portfolio) + np.random.uniform(-0.1, 0.1, num_assets)
```

به هر وزن نویز تصادفی اضافه می‌شود تا تغییرات کوچک ایجاد شود.

- **نرمال‌سازی وزن‌ها:**

```
new_portfolio = np.abs(new_portfolio)
new_portfolio /= new_portfolio.sum()
```

مقادیر نرمال‌سازی می‌شوند تا مجموع وزن‌ها برابر ۱ باشد.

- **انتخاب بهترین پرتفوی‌ها:**

```
top_indices = np.argsort(new_fitness_scores)[-beam_width:]
beam = [new_beam[i] for i in top_indices]
```

این خط پرتفوی‌های جدید را بر اساس فیتنس مرتب کرده و beam_widthپرتفوی برتر را انتخاب می‌کند.

- **به‌روزرسانی بهترین جواب:**

```
if max(fitness_scores) > best_fitness:
    best_fitness = max(fitness_scores)
```

```
best_portfolio = beam[np.argmax(fitness_scores)]
```

---

۶ .مزایا و معایب Beam Search

مزایا:

- بررسی چندین حالت به طور همزمان (موازی).
- انتخاب بهترین کاندیداها در هر مرحله.

معایب:

- احتمال از دست دادن برخی مسیرهای بالقوه خوب به دلیل محدودیت در beam_width.

---

۷ .نمودار تاریخچه فیتنس

تابع plot_fitness_historyنمودار تغییرات فیتنس در طول تکرارها را رسم می‌کند. این نمودار کمک می‌کند مشاهده کنیم چگونه الگوریتم به جواب بهینه همگرا می‌شود.

فایل RLBS

این کد، پیاده‌سازی الگوریتم **Random Local Beam Search (RLBS)** برای پیدا کردن پرتفوی بهینه سرمایه‌گذاری است. این الگوریتم بهبود یافته‌ای از الگوریتم Beam Search است که در آن، برای هر پرتفوی، تعداد مشخصی از همسایگان محلی به صورت تصادفی بررسی می‌شود.

---

۱ .الگوریتم Random Local Beam Search

RLBS شبیه به Beam Search عمل می‌کند، اما با افزودن مفهوم **همسایگی تصادفی** برای بهبود پرتفوی‌ها. در هر مرحله:

1. از پرتفوی‌های موجود در beam شروع می‌شود.
2. تعداد مشخصی همسایه برای هر پرتفوی ایجاد می‌شود.
3. بهترین پرتفوی‌ها از مجموعه کل beam) اولیه و همسایگان جدید (انتخاب می‌شوند.

---

۲ .ساختار کلی تابع

ورودی‌ها

- expected_returns: بازده مورد انتظار هر دارایی.
- covariance_matrix: ماتریس کوواریانس بین بازده‌ها.
- beam_width: تعداد پرتفوی‌هایی که به صورت موازی بررسی می‌شوند (پیش‌فرض: ۵).
- neighbors_per_portfolio: تعداد همسایگان تصادفی که برای هر پرتفوی ایجاد می‌شود.
- max_iterations: حداکثر تعداد تکرار الگوریتم.
- random_state: بذر تصادفی برای تکرارپذیری.

خروجی‌ها

- best_portfolio: بهترین پرتفوی پیدا شده.
- best_fitness: مقدار فیتنس متناظر با بهترین پرتفوی.
- fitness_history: تاریخچه مقادیر فیتنس بهترین پرتفوی در هر مرحله.

---

۳ .مراحل اجرا

۱ .مقداردهی اولیه

- beam: شامل beam_width پرتفوی اولیه است که به صورت تصادفی تولید می‌شوند:

```
beam = [generate_initial_portfolio(num_assets, random_state + i) for i in
range(beam_width)]
```

- محاسبه فیتنس برای پرتفوی‌های اولیه:

```
fitness_scores = [calculate_fitness(p, expected_returns, covariance_matrix) for p in beam]
```

- تعیین بهترین پرتفوی و مقدار فیتنس:

```
best_portfolio = beam[np.argmax(fitness_scores)]
best_fitness = max(fitness_scores)
```

---

۲ .تکرار برای بهینه‌سازی

- الگوریتم در هر مرحله، موارد زیر را انجام می‌دهد:

1. **ایجاد همسایگان برای هر پرتفوی**:
   - برای هر پرتفوی در beam ، تعدادی همسایه ایجاد می‌شود:

```
new_portfolio = np.array(portfolio) + np.random.uniform(-0.1, 0.1, num_assets)
```

```
new_portfolio = np.abs(new_portfolio)
new_portfolio /= new_portfolio.sum()
```

2. **محاسبه فیتنس همسایگان:**
   - مقدار فیتنس برای تمامی پرتفوی‌های جدید محاسبه می‌شود:

```
new_fitness_scores.append(calculate_fitness(new_portfolio, expected_returns, covariance_matrix))
```

3. **انتخاب بهترین پرتفوی‌ها:**
   - بهترین beam_width پرتفوی از مجموعه کل (beam فعلی + همسایگان) انتخاب می‌شوند:

```
top_indices = np.argsort(new_fitness_scores)[-beam_width:]
beam = [new_beam[i] for i in top_indices]
```

4. **به‌روزرسانی بهترین پرتفوی کلی:**
   - اگر پرتفوی جدیدی با فیتنس بهتر پیدا شود، مقدار بهترین پرتفوی و فیتنس به‌روزرسانی می‌شود:

```
if max(fitness_scores) > best_fitness:
    best_fitness = max(fitness_scores)
    best_portfolio = beam[np.argmax(fitness_scores)]
```

---

۳ .خاتمه الگوریتم

- الگوریتم پس از max_iterations خاتمه می‌یابد و نتایج شامل بهترین پرتفوی، مقدار فیتنس و تاریخچه فیتنس بازگردانده می‌شوند:

```
return best_portfolio, best_fitness, fitness_history
```

---

۴ .توضیح خط‌ها

تولید همسایگان تصادفی:

این بخش وظیفه ایجاد تغییرات کوچک در وزن‌های پرتفوی را دارد:

```
new_portfolio = np.array(portfolio) + np.random.uniform(-0.1, 0.1, num_assets)
new_portfolio = np.abs(new_portfolio)
new_portfolio /= new_portfolio.sum()
```

این تغییرات به پرتفوی امکان جستجوی فضای حالت بیشتری را می‌دهد.

انتخاب بهترین پرتفوی‌ها:

در این قسمت، بهترین پرتفوی‌ها بر اساس مقدار فیتنس انتخاب می‌شوند:

```
top_indices = np.argsort(new_fitness_scores)[-beam_width:]
beam = [new_beam[i] for i in top_indices]
```

این انتخاب بر اساس مقادیر بالاترین فیتنس صورت می‌گیرد.

---

۶ .تفاوت RLBS باBeam Search

1. **همسایگان محلی** RLBS :به جای ایجاد مستقیم پرتفوی‌های جدید، برای هر پرتفوی تعدادی همسایه تولید می‌کند.
2. **جستجوی تصادفی بهتر** :این ویژگی امکان کشف بهتر فضای حالت و جلوگیری از گیر افتادن در نقاط بهینه محلی را فراهم می‌کند.

---

۷ .نمودار تاریخچه فیتنس

تابع plot_fitness_historyتغییرات مقدار فیتنس بهترین پرتفوی در هر تکرار را رسم می‌کند. این نمودار روند همگرایی الگوریتم به سمت بهترین جواب را نشان می‌دهد.

فایل GA

این کد الگوریتم ژنتیک (Genetic Algorithm - GA) را برای بهینه‌سازی پرتفوی سرمایه‌گذاری پیاده‌سازی می‌کند. الگوریتم ژنتیک از اصول تکاملی الهام گرفته و با استفاده از **انتخاب، ترکیب(Crossover) ، جهش (Mutation)** و **بقا**، بهترین پرتفوی را پیدا می‌کند.

---

۱ .ساختار کلی الگوریتم ژنتیک

ورودی‌ها

- expected_returns:بازده مورد انتظار هر دارایی.
- covariance_matrix:ماتریس کوواریانس بین بازده‌ها.
- population_size:اندازه جمعیت اولیه (پیش‌فرض: ۱۰).

- generations: تعداد نسل‌ها (پیش‌فرض: ۱۰۰۰).
- mutation_rate: نرخ جهش (پیش‌فرض: ۰.۱).
- random_state: بذر تصادفی برای تولید اعداد تصادفی.

خروجی‌ها

- best_portfolio: بهترین پرتفوی پیدا شده.
- best_fitness: مقدار فیتنس بهترین پرتفوی.
- fitness_history: تاریخچه مقادیر فیتنس در طول نسل‌ها.

---

۲ .مراحل اجرای الگوریتم ژنتیک

۱ .مقداردهی اولیه

1. تولید جمعیت اولیه:
   - جمعیت اولیه شامل population_size پرتفوی تصادفی است:

```
population = [generate_initial_portfolio(num_assets, random_state + i) for i in range(population_size)]
```

   - نرمال‌سازی وزن‌های پرتفوی‌ها:

```
population = [np.array(portfolio) / sum(portfolio) for portfolio in population]
```

2. محاسبه فیتنس:
   - فیتنس برای هر پرتفوی با استفاده از نسبت بازده به ریسک محاسبه می‌شود:

```
fitness_scores = [calculate_fitness(p, expected_returns, covariance_matrix) for p in population]
```

۲ .حلقه اصلی الگوریتم

الگوریتم به تعداد نسل‌های مشخص تکرار می‌شود .(generations) در هر نسل، مراحل زیر اجرا می‌شوند:

1. **انتخاب والدین:**
   - انتخاب والدین با استفاده از روش **چرخ رولت (Roulette Wheel Selection)** انجام می‌شود.
     احتمال انتخاب هر پرتفوی متناسب با فیتنس آن است:

```
probabilities = np.array(fitness_scores) / sum(fitness_scores)

parents_indices = np.random.choice(population_size, size=2, p=probabilities, replace=False)
```

2. **ترکیب:(Crossover)**

- والدین با یکدیگر ترکیب شده و دو فرزند جدید تولید می‌کنند. نقطه برش برای ترکیب به صورت تصادفی
  crossover_point = np.random.randint(1, num_assets - 1)

```
child1 = np.concatenate((parents[0][:crossover_point], parents[1][crossover_point:]))
child2 = np.concatenate((parents[1][:crossover_point], parents[0][crossover_point:]))
```

3. **جهش:(Mutation)**

- با احتمال mutation_rate، به هر فرزند یک مقدار تصادفی اضافه می‌شود تا تنوع بیشتری ایجاد شود:

```
if np.random.rand() < mutation_rate:
    child1 += np.random.uniform(0, 0.1, num_assets)
if np.random.rand() < mutation_rate:
    child2 += np.random.uniform(0, 0.1, num_assets)
```

- فرزندان نرمال‌سازی می‌شوند تا مجموع وزن‌ها برابر با ۱ باشد:

```
child1 = np.abs(child1)
child1 /= child1.sum()
```

4. **به‌روزرسانی جمعیت:**

- جمعیت جدید شامل فرزندان تولید شده است:

```
population = new_population
```

5. **به‌روزرسانی بهترین پرتفوی:**

- اگر پرتفوی جدیدی با فیتنس بهتر پیدا شود، بهترین پرتفوی و مقدار فیتنس به‌روزرسانی می‌شوند:

```
if fitness_scores[max_fitness_index] > best_fitness:
    best_fitness = fitness_scores[max_fitness_index]
    best_portfolio = population[max_fitness_index]
```

۳ .خاتمه الگوریتم

- پس از اتمام تمام نسل‌ها، بهترین پرتفوی و تاریخچه فیتنس بازگردانده می‌شوند:

```
return best_portfolio, best_fitness, fitness_history
```

---

۳ .توضیح اجزا

انتخاب والدین(Selection)

روش چرخ رولت احتمال انتخاب هر پرتفوی را متناسب با مقدار فیتنس آن تنظیم می‌کند. پرتفوی‌های با فیتنس بالاتر شانس بیشتری برای انتخاب دارند.

ترکیب(Crossover)

والدین با ترکیب بخشی از وزن‌های خود، فرزندان جدیدی تولید می‌کنند. این مرحله برای کشف نقاط جدید در فضای حالت مفید است.

جهش(Mutation)

جهش با افزودن تغییرات تصادفی به فرزندان، تنوع را افزایش می‌دهد و از گیر افتادن در بهینه‌های محلی جلوگیری می‌کند.

---

۴ .نمودار تاریخچه فیتنس

تابع plot_fitness_historyتغییرات مقدار فیتنس بهترین پرتفوی در هر نسل را رسم می‌کند. این نمودار روند بهبود پرتفوی‌ها را در طول زمان نشان می‌دهد.