

به نام خدا



استاد: دکتر حسین کارشناس

دستیاران آموزشی:

مهدی مهدیه یونس ایوبی راد

دانیال شفیعی پویا اسفندانی

مبانی هوش مصنوعی و کاربردها

نیم سال اول ۱۴۰۴-۱۴۰۵

ددلاین: ۴ آبان

دانشکده مهندسی کامپیوتر

فاز اول پروژه - الگوریتم‌های جستجو، رگرسیون خطی، تپه‌نوردی و تبرید شبیه‌سازی شده

مرحله اول پروژه درس از سه بخش تشکیل شده است. برای پروژه خود بعد از پیاده‌سازی گزارش تهیه کرده و در آن پیاده‌سازی خود را توضیح داده و نتایج را نمایش دهید. فایل‌های در محیط گیت‌هاب در اختیار شما قرار داده خواهند شد.

## ۱ پرندگان خشمگین: جنگ ستارگان

لوک اسکای واکر شمشیر نوری‌اش را بالا برد تا تخم مرغ‌ها را نجات دهد، اما خوک سیبیلو با سبیل‌های پیچ‌پیچی جلوی او را گرفت و گفت: «اگه می‌خوای بری، باید اول سبیل رو ببوسی!» لوک جا خورد، عقب‌نشینی کرد و گفت: «ای بابا، بهتره تخم مرغ‌ها خودشون خودشون رو نجات بدن!»

ماموریت شما: بعد از اینکه اسکای واکر بیخیال تخم مرغ‌ها شد، تصمیم گرفت ستاره‌های کهکشان را جمع کند. به اسکای واکر کمک کنید تا درمورد مسیر رسیدن به ستاره‌ها تصمیم‌گیری کند. از الگوریتم‌های جستجوی آگاهانه و ناآگاهانه برای هدایت اسکای واکر به سمت ستاره‌ها استفاده می‌شود. الگوریتم‌های در نظر گرفته شده برای پیاده‌سازی:

1. BFS: Breadth-First Search
2. UCS: Uniform Cost Search
3. DLS: Depth Limited Search
4. A\*

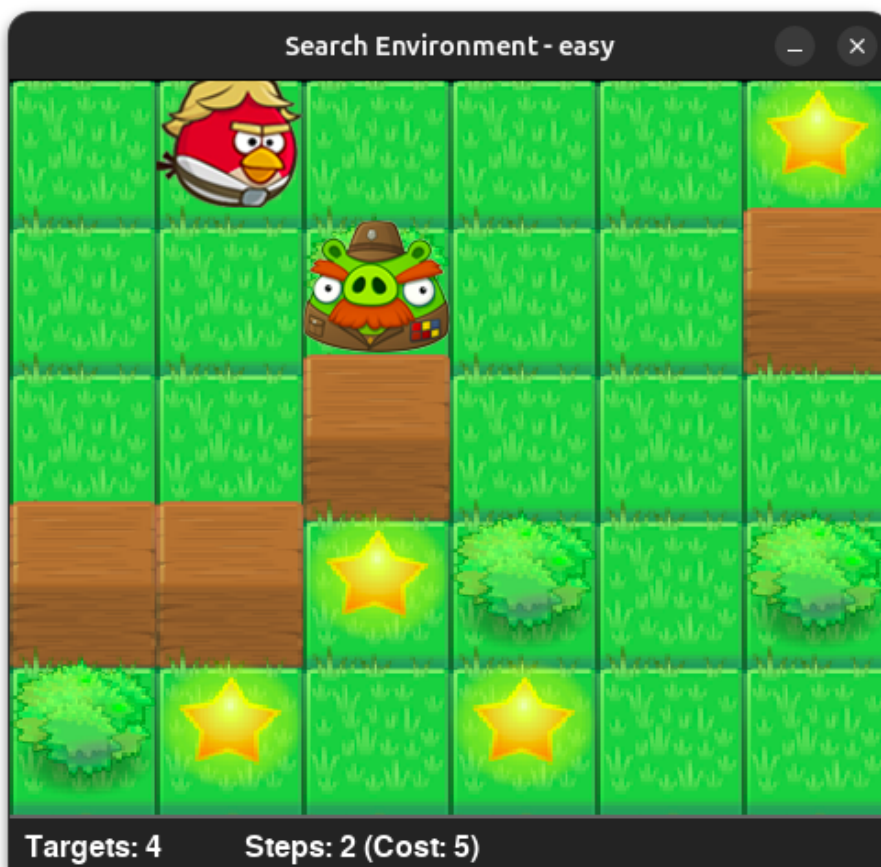
نکته: برای جستجوی عمق محدود می‌توانید یک حداکثر عمق معقول تخمین بزنید یا از الگوریتم IDS استفاده کنید.

### ۱.۱ پیش‌نیازها

برای اجرای محیط نیاز به نصب کتابخانه‌های pygame و numpy دارید.

## ۲.۱ محیط

محیط بازی یک محیط شطرنجی با ابعاد  $N$  در  $M$  است که یک محیط قطعی و مشاهده‌پذیر است.



شکل ۱: نمونه‌ای از محیط بازی که باید در آن ستاره‌ها جمع‌آوری شود و از برخورد با خوک اجتناب شود

اجزای محیط:

- اسکای‌والکر: باید به او کمک کنید همه‌ی ستاره‌ها را جمع کند و به خانه برگرداند.
- خوک سیبیلو: بعضی مواقع یک خوک در زمین بازی وجود دارد. خوک اکثراً متحرک است و حرکت  $n$ ام آن از قبل قابل تعیین است.
- بوته: هزینه‌ی عبور از آن ۲۰ برابر چمن عادی است
- جعبه: عبور از آن ممکن نیست.
- ستاره‌ها: باید همه‌ی ستاره‌های محیط توسط اسکای‌والکر جمع‌آوری شوند.

## ۳.۱ مراحل پیاده‌سازی

برای تسهیل پیاده‌سازی جستجو، توابع مفیدی برای تحلیل و جستجو در محیط پیاده‌شده‌اند. این توابع بر روی اشیاء کلاس GridState عمل می‌کنند. نحوه‌ی دسترسی به آن‌ها به صورت `state.func(args)` است و نوع خروجی آن‌ها در ادامه خواهد آمد. فهرست توابع تعامل با محیط که در اختیار شما قرار می‌گیرند:

۱. `get_grid_size()->tuple`: ابعاد نقشه را بر می‌گرداند

۲. `get_enemy_position()->tuple`: موقعیت فعلی دشمن را برمی‌گرداند.

۳. `get_enemy_next_position()->tuple`: موقعیت بعدی دشمن در مسیر حرکت را به صورت یک tuple برمی‌گرداند.

۴. `get_enemy_path()->list`: مسیر کامل حرکت دشمن را برمی‌گرداند.

۵. `get_agent_position()->tuple`: موقعیت فعلی عامل را برمی‌گرداند.

۶. `get_terrain_cost(pos)->float`: هزینه حرکت بر روی یک سلول خاص را با در نظر گرفتن نوع زمین و برخورد با دشمن محاسبه می‌کند.

۷. `get_targets_positions()->list`: موقعیت‌های تمام اهداف موجود در بازی را برمی‌گرداند.

۸. `get_bushes_positions()->list`: موقعیت‌های تمام بوته‌ها موجود در بازی را برمی‌گرداند.

۹. `is_rock_position(position)->bool`: مشخص می‌کند آیا در این موقعیت بوته است.

۱۰. `get_rocks_positions()->list`: موقعیت‌های تمام دیوارها موجود در بازی را برمی‌گرداند.

۱۱. `is_bush_position(position)->bool`: مشخص می‌کند آیا در این موقعیت بوته است.

۱۲. `is_goal_state()->bool`: بررسی می‌کند آیا تمام اهداف جمع‌آوری شده‌اند یا خیر.

۱۳. `is_collision_state()->bool`: بررسی می‌کند آیا عامل با دشمن برخورد کرده است یا خیر.

۱۴. `get_successors()->list()`: حالت‌های ممکن بعدی را با در نظر گرفتن حرکات مجاز تولید می‌کند. خروجی شامل یک لیست از ترکیب کنش، هزینه، و موقعیتی که به آن منجر می‌شود خواهد بود.

**نکته:** در صورتی که در آرگومان تابع عبور از دیوار `True` شود، `get_successor(toward_walls=True)` حرکت به سمت دیوارها به عنوان حرکت مجاز در نظر گرفته می‌شود و به عنوان کنش‌های مجاز بر می‌گردد.

در این محیط اگر عاملی به سمت دیوار یا خارج از محیط کنش انجام دهد، در وضعیت حاصل از آن کنش موقعیت عامل تغییر نخواهد کرد و فقط *step* تغییر می‌کند.

```
1 from env.env import play
2
3 def bfs(initial_state):
4     # Implement the algorithm here
5     return actions # Return your actions here
6
7 if __name__ == "__main__":
8     """
9     Place your algorithm as the input of function, choose the map
10    and set the delay. Then run the game.
11    """
12    play("map_name", bfs, delay=500)
13
```

Listing 1: مثالی از ساختار main.py که باید توسط شما ایجاد و نوشته شود.

تابع اکتشافی مورد استفاده در الگوریتم  $A^*$  در ابتدای تعریف می‌شود که باید آن را پیاده‌سازی کنید. در پیاده‌سازی این تابع می‌توانید از توابع تعامل با محیط که در اختیار شما قرار گرفته است استفاده کنید.

```
1 def astar(initial_state):
2     def heuristic(args*):
3         # Implement heuristic here
4         return h
5
6     # Implement A* here
7     return actions
8
```

Listing 2: مثالی از ساختار تابع  $A^*$

توجه کنید که توابع پیاده‌سازی شده باید بتوانند در نقشه‌های مختلف به کار گرفته شوند. نمونه‌هایی که می‌توانند از نظر ابعاد محیط، تعداد ستاره‌ها و موقعیت اولیه اسکای‌واکر و حرکت خوک متفاوت باشند. به همین دلیل سه محیط اولیه در اختیار شما قرار داده شده است تا برای آزمایش توابع پیاده‌سازی شده مورد استفاده قرار گیرد.

## ۴.۱ نحوه ارزیابی

هر بار پس از اجرای محیط، یک گزارش در ترمینال چاپ می‌شود که نشان‌دهنده امتیاز کسب شده، زمان اجرا و تعداد گره‌های بسط داده شده است. ارزیابی کیفیت تابع اکتشافی پیاده‌سازی شده براساس تعداد گره‌های بسط داده شده توسط الگوریتم جستجوی A\* خواهد بود، که در جدول زیر امتیاز مرتبط برای محیط‌های easy، medium و hard نشان داده شده است:

امتیاز	گره‌های بسط داده شده در نقشه سخت	گره‌های بسط داده شده در نقشه متوسط	گره‌های بسط داده شده در نقشه ساده
۱۰۰٪	کمتر از ۲۵۰۰	کمتر از ۵۰۰	کمتر از ۱۰۰
۸۰٪	کمتر از ۲۰۰۰۰	کمتر از ۱۰۰۰۰	کمتر از ۱۵۰
۵۰٪	کمتر از ۱۵۰۰۰۰	کمتر از ۵۰۰۰۰	کمتر از ۲۰۰

جدول ۱: جدول امتیازات

نکته: اجرای الگوریتم‌های BFS، DLS، UCS روی محیط hard اجباری نیست.

همانطور که قبلاً گفته شد در ارزیابی پیاده‌سازی انجام شده از نمونه محیط‌های دیگری نیز استفاده می‌شود که انتظار می‌رود تعداد گره‌های بسط داده شده در حد قابل قبول باشد. بنابراین علاوه بر صحت پیاده‌سازی الگوریتم‌های BFS، DLS، UCS و A\* در فایل main.py، تابع اکتشافی پیاده‌سازی شده نیز از نظر تعداد گره‌های بسط داده شده مورد بررسی قرار خواهد گرفت.

## ۲ رگرسیون خطی: پیش‌بینی قطر سیارک‌ها

در صنعت هوافضا، شناسایی اجرام نزدیک به زمین (Near-Earth Objects - NEO) و ارزیابی خطر برخورد آن‌ها با سیاره ما از اهمیت بالایی برخوردار است. سیارک‌هایی که مداری نزدیک به زمین دارند و ابعادشان به اندازه‌ای بزرگ است که در صورت برخورد، خسارات قابل توجهی ایجاد کنند، به عنوان سیارک‌های بالقوه خطرناک (Potentially Hazardous Asteroids - PHA) شناخته می‌شوند.

آزمایشگاه پیش‌رانش جت ناسا (JPL) یکی از پیشگامان جهان در رصد و مطالعه این اجرام است. این سازمان با جمع‌آوری یک مجموعه داده عظیم و غنی از ویژگی‌های مداری و فیزیکی اجرام فضایی، به ما این امکان را می‌دهد تا با استفاده از ابزارهای علم داده، به درک عمیق‌تری از آن‌ها برسیم.

### ۱.۲ صورت مسئله

یکی از حیاتی‌ترین ویژگی‌ها برای ارزیابی خطر یک سیارک، قطر (Diameter) آن است. با این حال، اندازه‌گیری دقیق قطر سیارک‌ها از فاصله‌های بسیار دور، یک چالش بزرگ فنی است. به دلیل محدودیت‌های فناوری رصد، شرایط جوی و خطاهای اندازه‌گیری، داده‌های مربوط به قطر برای تعداد بسیار زیادی از سیارک‌های کشف‌شده، گمشده (Missing) یا پرت (Outlier) هستند. این شکاف اطلاعاتی، تحلیل‌های علمی و ارزیابی دقیق خطر را با مشکل مواجه می‌کند.

### ۲.۲ هدف پروژه

هدف شما در این پروژه، توسعه یک مدل رگرسیون برای پر کردن این شکاف اطلاعاتی است. شما با استفاده از ویژگی‌های موجود مانند پارامترهای مداری و قدر مطلق (درخشندگی)، قطر سیارک‌ها را پیش‌بینی خواهید کرد. مدلی که شما می‌سازید، می‌تواند به دانشمندان کمک کند تا تخمین دقیق‌تری از ابعاد اجرام آسمانی داشته باشند و کیفیت مطالعات خود را بهبود بخشند.

### ۳.۲ رویکرد حل مسئله

تحلیل‌های اولیه روی داده‌ها نشان می‌دهد که بین برخی ویژگی‌ها (مانند قدر مطلق) و قطر سیارک، یک رابطه نسبتاً خطی وجود دارد. به همین دلیل، استفاده از یک مدل رگرسیون خطی، نقطه شروعی منطقی و قدرتمند برای حل این مسئله است. برای آموزش مدل و یافتن بهترین پارامترها، شما الگوریتم بهینه‌سازی کاهش گرادیان تصادفی (Stochastic Gradient Descent) (SGD)) را از ابتدا (from scratch) پیاده‌سازی کرده و سپس با معیارهای ارزیابی مورد نظر کیفیت مدل را در پیش‌بینی مقادیر جدید بررسی خواهید کرد.

## ۴.۲ آشنایی با ستون‌های دیتاست

### ۱.۴.۲ ویژگی‌های فیزیکی و طبقه‌بندی

این ستون‌ها مشخصات فیزیکی و نوع خطر هر سیارک را توصیف می‌کنند.

- full\_name: نام کامل یا شناسه رسمی جرم آسمانی.
- NEO (Near-Earth Object): یک پرچم (flag) است که مشخص می‌کند آیا این جرم یک جرم نزدیک به زمین است یا خیر.
- PHA (Potentially Hazardous Asteroid): یک پرچم است که نشان می‌دهد آیا این سیارک به عنوان یک «سیارک بالقوه خطرناک» برای زمین طبقه‌بندی می‌شود یا خیر.
- H (Absolute Magnitude): قدر مطلق سیارک. این پارامتر نشان‌دهنده درخشندگی ذاتی جرم است و یکی از مهم‌ترین ویژگی‌ها برای تخمین اندازه آن محسوب می‌شود.
- diameter: قطر سیارک بر حسب کیلومتر (km). این همان ستون هدف (Target) است که شما قصد پیش‌بینی مقادیر گمشده آن را دارید.
- albedo: آلبدو یا ضریب بازتاب سطح سیارک. این مقدار نشان می‌دهد چه کسری از نور خورشید از سطح آن بازتاب می‌شود و به جنس و ترکیبات سطحی آن بستگی دارد.
- diameter\_sigma: میزان عدم قطعیت (خطا) در اندازه‌گیری قطر، بر حسب کیلومتر.

### ۲.۴.۲ ویژگی‌های مداری (Orbital Features)

این ستون‌ها هندسه، اندازه و شکل مدار حرکت سیارک به دور خورشید را توصیف می‌کنند.

- e (Eccentricity): خروج از مرکز مدار. این عدد نشان می‌دهد مدار چقدر «کشیده» است. مقدار صفر برای یک دایره کامل و مقادیر نزدیک به یک برای مدارهای بسیار بیضوی است.
- a (Semi-major axis): نیم‌قطر بزرگ مدار بر حسب واحد نجومی (AU). این پارامتر، اندازه متوسط مدار سیارک را نشان می‌دهد.
- q (Perihelion distance): فاصله حضیض یا نزدیک‌ترین فاصله سیارک تا خورشید در مدارش (بر حسب AU).

- $i$  (Inclination): شیب مداری. زاویه مدار سیارک نسبت به صفحه مداری زمین (دایرة البروج) بر حسب درجه.
- $moid\_ld$  (Earth Minimum Orbit Intersection Distance): حداقل فاصله تقاطع مداری با زمین. این پارامتر نشان‌دهنده کمترین فاصله‌ای است که مدار سیارک به مدار زمین می‌رسد (بر حسب AU). این یک فاکتور کلیدی در تعیین خطرناک بودن یک سیارک است.
- $ad$  (Aphelion Distance): فاصله اوج یا دورترین فاصله سیارک از خورشید در مدارش (بر حسب AU).
- $n$  (Mean Motion): حرکت میانگین. سرعت زاویه‌ای متوسط سیارک در مدارش، که معمولاً بر حسب درجه در روز ( $deg/d$ ) بیان می‌شود.
- $per$  (Orbital Period): دوره تناوب مداری بر حسب روز. مدت زمانی که طول می‌کشد تا سیارک یک دور کامل به دور خورشید بگردد.
- $per\_y$  (Orbital Period in Years): دوره تناوب مداری بر حسب سال.

## ۳.۴.۲ طبقه‌بندی و کیفیت داده (Classification & Data Quality)

- $class$  (Orbital Class): کلاس مداری سیارک که آن را بر اساس نوع مدارش طبقه‌بندی می‌کند. کلاس‌های معروف عبارتند از:
  - $MBA$ : سیارک‌های کمربند اصلی (بین مریخ و مشتری).
  - $OMB$ ،  $IMB$ : کلاس‌های دیگر مرتبط با نزدیکی به مدار زمین.
- $rms$  (Root Mean Square): ریشه میانگین مربعات خطا. این مقدار کیفیت تطابق مدل مداری محاسبه‌شده با داده‌های رصدی واقعی را نشان می‌دهد. مقدار  $RMS$  کمتر به معنای یک مدار دقیق‌تر و قابل اطمینان‌تر است.
- پارامترهای عدم قطعیت (Uncertainty Parameters): ستون‌هایی که با پیشوند  $\sigma\_$  شروع می‌شوند (مانند  $\sigma\_e$  یا  $\sigma\_a$ )، نشان‌دهنده میزان عدم قطعیت یا خطای آماری در اندازه‌گیری آن پارامتر هستند. هر چه این مقدار کمتر باشد، اندازه‌گیری دقیق‌تر است.

## ۵.۲ ابزارها و محیط توسعه

- برای انجام این پروژه، انعطاف‌پذیری در انتخاب ابزارها را دارید، اما موارد زیر به عنوان یک راهنمای شروع پیشنهاد می‌شوند.
- محیط پیاده‌سازی:



– توصیه می‌شود پروژه خود را در یک **جوئیتر نوتبوک** (ipynb) پیاده‌سازی کنید. این فرمت به شما اجازه می‌دهد کد، خروجی‌ها و توضیحات را در یک فایل واحد داشته باشید و هر بخش از کد را به صورت مستقل و تکرارپذیر اجرا نمایید.

– برای اجرای نوتبوک از محیط Jupyter Notebook یا Google Colab استفاده کنید.

## • کتابخانه‌های پیشنهادی:

- **Pandas**: ابزار اصلی شما برای بارگذاری، مدیریت و دستکاری داده‌ها خواهد بود.
- **NumPy**: برای تمام عملیات عددی سریع، محاسبات ریاضی و جبر خطی ضروری است.
- **Matplotlib & Seaborn**: برای مصورسازی داده‌ها، رسم نمودارهای تحلیل اکتشافی (EDA) و نمایش نتایج مدل.
- **Scikit-learn (sklearn)**: هرچند شما مدل و بهینه‌ساز اصلی را خودتان پیاده‌سازی می‌کنید، استفاده از توابع کمکی این کتابخانه بلامانع است:
  - \* `sklearn.preprocessing`: برای پیش‌پردازش سریع داده مانند مقیاس‌بندی ویژگی‌ها.
  - \* `sklearn.metrics`: برای محاسبه آسان و دقیق معیارهای ارزیابی مانند  $R^2$  و MSE.

## ۶.۲ مراحل پیاده‌سازی پروژه

### ۱.۶.۲ بخش اول: آماده‌سازی و شناخت داده

قبل از نوشتن هرگونه مدل، باید داده‌ها را به خوبی بشناسیم و آن‌ها را برای فرآیند یادگیری آماده کنیم. این مرحله یکی از مهم‌ترین بخش‌های هر پروژه یادگیری ماشین و علم داده است.

- **بارگذاری و شناخت مجموعه داده**: مجموعه داده را بارگذاری و به ۲ بخش آموزش و آزمایش تقسیم بندی کنید. این عملیات را با استفاده از ماژول `train_test_split` در `sklearn.model_selection` می‌توانید انجام دهید. برای یکسان بودن نتایج در اجراهای مختلف از `random_state = 42` در آرگومان ماژول استفاده کنید. از بخش آموزش (train) برای آموزش مدل و از بخش آزمایش (test) برای ارزیابی مدل آموزش دیده استفاده کنید. مثال:

```
1 import pandas as pd
2 from sklearn.model_selection import train_test_split
3
4 df = pd.read_csv('dataset.csv')
```

```

5
6 X = df.drop(columns=['diameter']) # Features
7 y = df['diameter'] # Target variable
8
9 # Split the data
10 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
11 random_state=42)

```

Listing 3: نمونه کد جدا سازی داده‌ی آموزش و ارزیابی

- (اختیاری) تحلیل اکتشافی داده (EDA) و مصورسازی: با استفاده از نمودارها و ابزارهای آماری، روابط پنهان در داده‌ها را کشف کنید. به خصوص، رابطه بین ویژگی‌های مختلف و ستون هدف (diameter) را بررسی کنید. شما می‌توانید از تحلیل اطلاعات بدست آمده در مهندسی ویژگی، کشف داده پرت، توسعه مدل مناسب و غیره استفاده کنید.
- مدیریت مقادیر گمشده: وجود مقادیر گمشده را بررسی و در صورت وجود با روش انتخابی آن را حذف یا جایگزین کنید.
- کدگذاری ویژگی‌های دسته‌ای (Encoding): مدل‌های ریاضی فقط با اعداد کار می‌کنند. ویژگی‌های غیر عددی (مانند ستون class) را به فرمت عددی تبدیل کنید.
- مقیاس‌بندی ویژگی‌ها (Feature Scaling): مقیاس‌های متفاوت ویژگی‌ها می‌تواند باعث ناپایداری و نوسانات در فرایند یادگیری شود. از تکنیک مقیاس‌بندی مناسب استفاده کنید.
- (اختیاری) مهندسی ویژگی: برخی ویژگی‌ها در مجموعه داده تاثیر بسیار کم یا منفی بر روند آموزش دارند و برخی ویژگی‌ها کلیدی هستند. شما می‌توانید با حذف برخی ویژگی‌ها کیفیت مدل را از نظر محاسباتی و دقت افزایش دهید. همچنین می‌توانید با ترکیب ویژگی‌های موجود، ویژگی‌های جدید موثر بسازید و کیفیت را افزایش دهید.
- (اختیاری) مدیریت داده‌های پرت: داده‌های پرت می‌توانند باعث انحراف مدل، افزایش واریانس و تاثیر منفی بر معیارهای ارزیابی شوند. در صورت حذف داده پرت، از مجموعه داده آزمون نباید حذف صورت بگیرد و از داده آموزش باید محدود باشد.

## ۲.۶.۲ بخش دوم: پیاده‌سازی و آموزش مدل

در این مرحله، قلب پروژه یعنی مدل رگرسیون خطی و الگوریتم بهینه‌سازی آن را از ابتدا (from scratch) پیاده‌سازی می‌کنید.

- تعیین هایپرپارامترها: پارامترهای اصلی آموزش مانند نرخ یادگیری (Learning Rate) و تعداد تکرارها (Epochs) را مشخص کنید.
- پیاده‌سازی الگوریتم SGD: تابع هزینه (Cost Function) برای رگرسیون خطی (مانند MSE) و فرآیند به‌روزرسانی وزن‌ها بر اساس گرادیان را خودتان کدنویسی کنید.
- نکته: استفاده از کتابخانه‌ی scikit-learn برای این کار مجاز نیست.
- آموزش مدل: مدل خود را با مجموعه داده آموزش، تعلیم دهید.
- رسم نمودار تابع زیان (Loss Curve): در حین آموزش، مقدار تابع هزینه را در هر تکرار ذخیره کرده و در نهایت نمودار آن را بر حسب Epoch رسم کنید. این نمودار به شما نشان می‌دهد که آیا مدل در حال یادگیری است یا خیر.

### ۳.۶.۲ بخش سوم: ارزیابی مدل

پس از آموزش، باید عملکرد مدل را روی داده‌هایی که تا به حال ندیده است (مجموعه داده آزمون) بسنجیم.

- محاسبه معیار  $R^2$  score بر روی داده آزمون.
- محاسبه معیار Mean Absolute Error بر روی داده آزمون.
- محاسبه معیار Mean Squared Error بر روی داده آزمون.

### ۴.۶.۲ فاز چهارم: پیاده‌سازی پیشرفته (امتیازی)

برای بهبود عملکرد مدل و به چالش کشیدن خودتان، تکنیک‌های پیشرفته زیر را به بهینه‌ساز SGD خود اضافه کنید. انجام هر یک از موارد زیر نمره اضافه در این مرحله از پروژه خواهد داشت:

- استفاده از تکانه (momentum) در بهینه‌ساز.
- تغییر نرخ یادگیری در حین آموزش با سیاست خاص.
- استفاده از توقف زود هنگام در بهینه‌ساز.
- استفاده از یک روش منظم‌سازی (Regularization) در بهینه‌سازی مدل.

نحوه ارزیابی:

برای کسب نمره کامل این بخش، مدل نهایی شما باید به امتیاز  $R^2 \geq 0.8$  بر روی داده آزمون دست یابد.

## ۳ یافتن نقطه مرکزی DNA با الگوریتم‌های جستجوی محلی

این پروژه به بررسی یکی از مسائل کلاسیک در بیوانفورماتیک، یعنی «مسئله نزدیک‌ترین رشته» (Closest String Problem) یا یافتن نقطه مرکزی برای مجموعه‌ای از رشته‌های DNA می‌پردازد. برای حل این مسئله، دو الگوریتم بهینه‌سازی مشهور، تپه‌نوردی (Hill Climbing) و تبرید شبیه‌سازی‌شده (Simulated Annealing)، پیاده‌سازی و مقایسه می‌شوند.

### ۱.۳ مقدمه

رشته‌های DNA، که از چهار باز نوکلئوتیدی آدنین (A)، سیتوزین (C)، گوانین (G) و تیمین (T) تشکیل شده‌اند، حامل اطلاعات ژنتیکی هستند. در تحقیقات بیوانفورماتیک، دانشمندان اغلب با مجموعه‌های بزرگی از رشته‌های DNA مرتبط (مانند ژن‌های مشابه در گونه‌های مختلف یا نمونه‌های مختلف یک ویروس) سروکار دارند. تحلیل این مجموعه‌ها برای یافتن الگوهای مشترک، مناطق حفاظت‌شده تکاملی، و یا شناسایی یک رشته نماینده از اهمیت بالایی برخوردار است. این «رشته نماینده» که ما آن را مرکز (Center) می‌نامیم، رشته‌ای فرضی است که بیشترین شباهت را به تمام رشته‌های موجود در مجموعه دارد. پیدا کردن چنین مرکزی به ما کمک می‌کند تا ویژگی‌های کلیدی و مشترک یک خانواده از توالی‌های ژنتیکی را درک کنیم.

### ۲.۳ تعاریف

- فاصله (Distance): فاصله بین دو رشته DNA با طول یکسان، تعداد موقعیت‌هایی است که کاراکترهای آن‌ها با هم متفاوت است که به فاصله همینگ هم معروف است.

$$\text{dist}(a, b) = \sum_{a[i] \neq b[i]} 1$$

مثلاً فاصله‌ی دو رشته‌ی زیر، ۲ است:

AGGCT

ACGCA

- مرکز (Center): در یک مجموعه رشته، مرکز رشته‌ای است که بیشترین فاصله آن (بر اساس فاصله همینگ) از هر

یک از رشته‌های مجموعه، کمترین مقدار ممکن باشد.

$$\text{center}(S) = \arg \min_a \{ \max_{i \in S} \{ \text{dist}(a, i) \} \}$$

به عنوان نمونه در مجموعه‌ی زیر مرکز رشته مشخص شده است:

$$\text{center}(\{ACCT, AGCG, ACTG\}) = \{ACCG\}$$

• شعاع (Radius): حداقل بیشترین فاصله ممکن است که یک مرکز می‌تواند از رشته‌های مجموعه داشته باشد.

$$\text{radius}(S) = \min_a \{ \max_{i \in S} \{ \text{dist}(a, i) \} \}$$

به عنوان مثال:

$$\text{radius}(\{AA, AC, AG\}) = 1$$

• حالت همسایگی (Neighboring State): هسته اصلی هر دو الگوریتم، تعریف مفهوم همسایه برای یک رشته است. دو رشته همسایه در نظر گرفته می‌شوند اگر و تنها اگر در دقیقاً یک جایگاه با یکدیگر تفاوت داشته باشند. به زبان ریاضی:

$$(a, b) \in N \iff \exists i : a[i] \neq b[i] \wedge \forall j \neq i : a[j] = b[j]$$

برای مثال، رشته‌های زیر به دلیل تفاوت در جایگاه سوم، همسایه یکدیگر هستند:

*ACGCT*

*ACTCT*

### ۳.۳ هدف پروژه

هدف اصلی این پروژه، پیاده‌سازی و ارزیابی دو الگوریتم جستجوی محلی برای یافتن مرکز و شعاع یک مجموعه از رشته‌های DNA است. در این پروژه، شما توابع لازم برای تعریف مسئله را پیاده‌سازی کرده و سپس دو الگوریتم زیر را برای حل آن به کار می‌گیرید:

۱. **Hill Climbing**: یک روش حریصانه که همیشه به سمت بهترین جواب در همسایگی حرکت می‌کند اما ممکن است در بهینه‌های محلی گرفتار شود. این الگوریتم تنها در صورتی به یک همسایه حرکت می‌کند که مقدار تابع ارزیابی  $f(a)$  در آن همسایه کمتر از حالت فعلی باشد.

۲. **Simulated Annealing**: یک روش احتمالی که با پذیرش کنترل‌شده حرکت‌های بدتر، قابلیت فرار از بهینه‌های محلی را دارد و می‌تواند به جواب‌های بهتری در فضای جستجوی پیچیده دست یابد. قاعده پذیرش یک همسایه بدتر به صورت زیر است:

$$P(\text{پذیرش}) = \exp\left(-\frac{\Delta f}{T}\right)$$

که در آن  $\Delta f = f(a_{\text{همسایه}}) - f(a_{\text{فعلی}})$  و  $T$  پارامتر دما است. این الگوریتم با یک  $T$  اولیه بالا شروع می‌شود و در هر مرحله با ضریب  $\alpha$  ( $0 < \alpha < 1$ ) سرد می‌شود.

در نهایت، عملکرد این دو الگوریتم از نظر کیفیت جواب نهایی، هزینه (شعاع) و سرعت اجرا با یکدیگر و با یک الگوریتم جستجوی کامل (Brute Force) مقایسه خواهد شد.

### ۴.۳ پیاده‌سازی پروژه

برای پیاده‌سازی این قسمت نیاز است فایل نوت‌بوک DNA\_Center.ipynb تکمیل شود. با استفاده از markdownها و کدهای کامنت‌شده مرحله به مرحله پیاده‌سازی انجام دهید. وظیفه اصلی شما تکمیل توابع خالی پایتون برای پیاده‌سازی دو الگوریتم اصلی و توابع کمکی آنها است:

۱. **initialize\_state**: ایجاد یک رشته DNA تصادفی به عنوان نقطه شروع جستجو.

۲. **calculate\_evaluation**: محاسبه «هزینه» یا «شعاع» برای یک رشته DNA پیشنهادی. این هزینه برابر است با بیشترین فاصله آن رشته با تمام رشته‌های موجود در مجموعه ورودی.

$$f(a) = \max_{i \in S} \text{dist}(a, i)$$

۳. **get\_neighbor\_state**: تولید یک «همسایه» برای یک رشته معین. همسایه رشته‌ای است که فقط در یک کاراکتر با رشته اصلی تفاوت دارد.

۴. **get\_all\_neighbors**: ایجاد لیستی از تمام همسایه‌های ممکن برای یک رشته.

۵. `hill_climbing_descent`: پیاده‌سازی الگوریتم تپه‌نوردی که همیشه به سمت همسایه‌ای با هزینه کمتر حرکت می‌کند.

۶. `simulated_annealing`: پیاده‌سازی الگوریتم تبرید شبیه‌سازی شده که علاوه بر حرکت به سمت جواب‌های بهتر، با احتمالی مشخص جواب‌های بدتر را نیز می‌پذیرد تا از گرفتار شدن در بهینه‌های محلی جلوگیری کند.

### ۵.۳ ارزیابی

در انتهای نوت‌بوک، کدهایی برای اهداف زیر فراهم شده است:

- مقایسه الگوریتم‌ها: یک الگوریتم جستجوی کامل به صورت آماده وجود دارد تا بتوانید صحت و کارایی الگوریتم‌های خود را با آن مقایسه کنید. نتایج سه الگوریتم از نظر جواب نهایی، هزینه (شعاع) و زمان اجرا با هم مقایسه می‌شوند.

- تحلیل همگرایی: نمودارهایی برای مقایسه روند بهبود جواب در دو الگوریتم `Hill Climbing` و `Simulated Annealing` رسم می‌شود.


- تحلیل پارامتر  $\alpha$ : تأثیر نرخ سرد شدن (پارامتر  $\alpha$ ) بر عملکرد الگوریتم تبرید شبیه‌سازی شده بررسی و به صورت بصری نمایش داده می‌شود.

## نکات ارائه

- لطفاً نتایج خود را در قالب یک فایل pdf به همراه کدها تحویل دهید. برای قالب گزارش می‌توانید از قالب دانشکده استفاده کنید.

- در صورت پیاده‌سازی هر یک از این موارد اضافی، تمام اعضای تیم باید دلیل استفاده از آن و منطق استفاده از آن را بدانند. این موضوع باید هم در گزارش قید شود و هم در زمان تحویل پروژه کاملاً بر آن مسلط باشند.

موفق باشید.

 هشدار: لطفاً انجام پروژه را به روز پایانی ددلاین موکول نکنید!