به نام خدا

نام و نام خانوادگی:

فرناز خوش دوست آزاد

شماره دانشجویی:

99521253

نام استاد:

دکتر عبدی

نام درس:

هوش مصنوعی و سیستم های خبره

نام پروژه:

پروژه ژنتیک

در قسمت اول ما دیکشنری‌ای از operators و لیستی از terminals داریم و پس از آن کلاسی از operand و operator داریم تا برای درست کردن tree از آن استفاده کنیم و پس از آن تابعی داریم که از آن برای شمارش operandهای یک operator استفاده می‌کنیم که برای sin(x) و cos(x) یک و دیگر عملگرها 2 را برمی‌گرداند.

پس از آن برای درست کردن یک tree ما به کلاس node برای نمایش درخت خود استفاده می‌کنیم.

در کلاس node ما پنج attribute داریم که از parent و left و right و type و data برخوردار هستند و به غیر از آن دو تابع داریم که برای تولید node رندوم و جابجایی nodeها از آنها استفاده می‌کنیم که نام آنها به ترتیب create_random_node و replace_subtree می باشد.

```python
class Node:
    def __init__(self, type: Type, data):
        self.parent = None
        self.left = None
        self.right = None
        self.type = type
        self.data = data

    def create_random_node(type: Type, is_terminal: bool=False, non_zero: bool=False
                           , non_negative: bool=False, dimensions: int=2):
        if type == Type.Operand:
            if is_terminal:
                return Node(type, terminals[rnd.randint(0, dimensions - 2)])
            else:
                data = rnd.randint(min_operand, max_operand)
                if (data <= 0 and non_negative):
                    data = rnd.randint(1, max_operand)
                elif (data == 0 and non_zero):
                    data = rnd.randint(min_operand, -1) if rnd.randint(1, 2) == 1 else rnd.randint(1, max_operand)
        else:
            data = rnd.choice(list(operators.keys()))
        return Node(type, data)

    def replace_subtree(self, new_subtree_root):
        if self.parent is not None:
            if self.parent.left == self:
                self.parent.left = new_subtree_root
            elif self.parent.right == self:
                self.parent.right = new_subtree_root
```

پس از آن از class دیگری به اسم ExpressionTree استفاده می کنیم که تنها attribute آن root می باشد و از آن برای درست کردن tree رندوم و حساب کردن عملیات ریاضی در درخت استفاده می‌کنیم که نام این تابع evaluate می‌باشد و سپس تابع traversal را داریم که از آن برای traverse در یک tree استفاده می‌کنیم و همه ی نودهای left و right را traverse می‌کنیم و تابع get_all_nodes() برای گرفتن همه‌ی نودهای یک tree استفاده می‌شود.

```
def inorder_traversal(self):
    return self.__inorder_traversal(self.root)

def __inorder_traversal(self, node: Node=None):
    if node is None:
        return ''

    if node is not None:
        left = self.__inorder_traversal(node.left)
        right = self.__inorder_traversal(node.right)
        return '(' + left + ' ' + str(node.data) + ' ' + right + ')'

def get_all_nodes(self):
    return self.__get_all_nodes(self.root)

def __get_all_nodes(self, node):
    if node is None:
        return []
    return [node] + self.__get_all_nodes(node.left) + self.__get_all_nodes(node.right)
```

تا اینجای کار تنها کارهای زیر ساختی برای fit کردن درخت را انجام دادیم حال می‌خواهیم از کلاس و توابعی برای predict کردن استفاده کنیم که نام این کلاس gplearn می‌باشد که دارای 6 attribute می باشد که به ترتیب عبارتند از population_size وgenerations و mutation_rate و population و max_depth و dimensions.

از population_size برای تعداد tree های ساخته شده و استفاده می کنیم و آن را به درون population می‌ریزیم و dimension در حقیقت همان تعداد terminals قابل استفاده در این کد می باشد و max_depth حداکثر عمق را در tree به ما نشان می‌دهد و mutation_rate همانطور که از نامش پیداست برای مشخص کردن نرخ جهش از آن استفاده می‌کنیم. در این کلاس در ابتدا تابعی به نام __initialize_generations() استفاده می‌کنیم که در آن به تعداد population_size به درون population list درخت رندوم با استفاده از توابع قبلی که بالا توضیح داده شد اضافه می‌کنیم. پس از آن تابعی مربوط به evaluate کردن داریم و از تابع mean_squared_error در sklearn برای محاسبه ی خطا و کم کردن خطا در مراحل بعدی استفاده می‌کنیم. از cross_over نیز برای جابجایی

در node ها استفاده می‌کنیم که به الهام از مباحث زیستی ژنتیک می‌باشد. پس از آن تابع mutation را داریم که با توجه به نرخ mutation_rate() از آن در populatin خود استفاده خواهیم کرد.

```
def __crossover(self, parent1, parent2):
    crossover_point1 = random.choice(parent1.get_all_nodes())
    crossover_point2 = random.choice(parent2.get_all_nodes())

    child1 = copy.deepcopy(parent1)
    child2 = copy.deepcopy(parent2)

    crossover_point1.replace_subtree(child2)
    crossover_point2.replace_subtree(child1)

    return child1, child2

def __mutation(self, tree):
    mutation_point = random.choice(tree.get_all_nodes())
    new_subtree = ExpressionTree.create_random_tree(random.randint(1, self.max_depth))
    mutation_point.replace_subtree(new_subtree.root)
```

پس از آن تابع fit را داریم که از برای fit و predict کردن توابع با استفاده از mean_squared_error گفته شده در بالاتر از آن استفاده می‌کنیم و پس از آن از توابع مختلف و مشهوری استفاده می‌کنیم و به تابع می‌دهیم تا آن را محاسبه کند و سپس با استفاده از توابع مختلف آن‌ها را نمایش می‌دهیم. که همانطور که مشاهده می شود تا حد خوبی توابع درست پیش بینی شده‌اند.

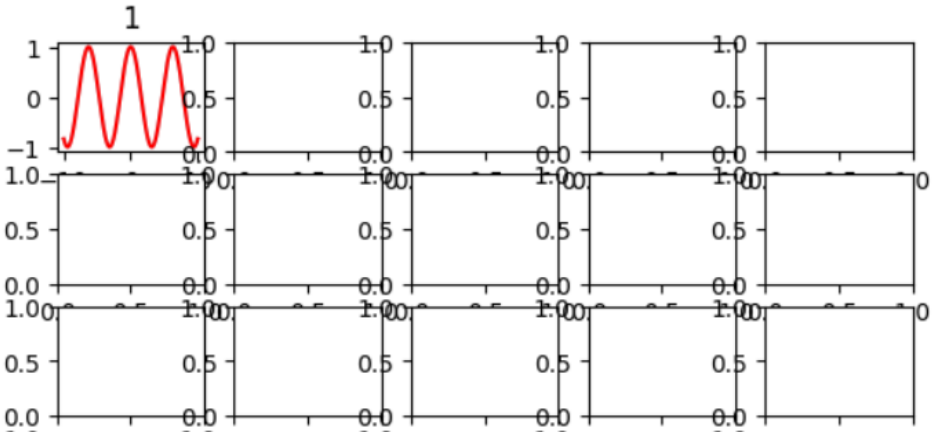
در قسمت زیر نمونه‌هایی از این توابع نشان داده شده است:

first funcion:

y = np.cos(x)

```
[52] x = np.linspace(-10, 10, 1000)
     y = np.cos(x)
     g = gplearn()
     print(g.fit(x, y).inorder_traversal())
```

```
Generation 1, Best Fitness: -0.0
( cos (( x ) / ( -1 )))
```

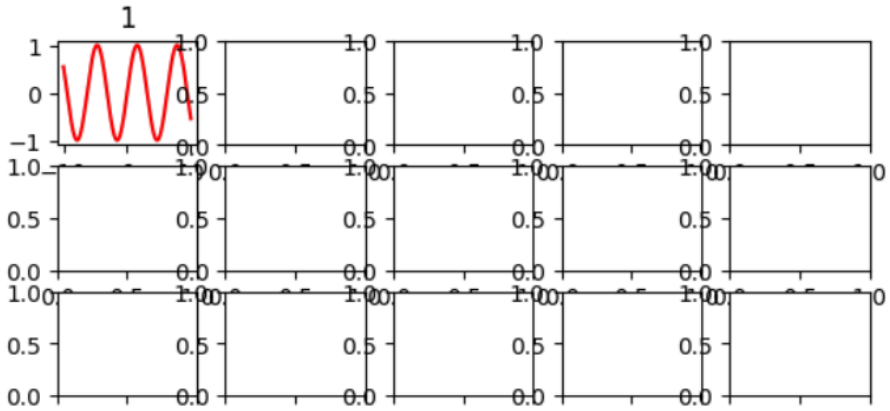


second function:

```
y = np.sin(x)
```

```
x = np.linspace(-10, 10, 1000)
y = np.sin(x)
g = gplearn()
print(g.fit(x, y).inorder_traversal())
```

```
Generation 1, Best Fitness: -0.0
( sin ( x ))
```



✓ 11s

```
x = np.linspace(-10, 10, 1000)
y = 3*x - 4
g = gplearn()
print(g.fit(x, y).inorder_traversal())
```

```
Generation 1, Best Fitness: -16.0
Generation 2, Best Fitness: -16.0
Generation 3, Best Fitness: -16.0
Generation 4, Best Fitness: -16.0
Generation 5, Best Fitness: -16.0
Generation 6, Best Fitness: -16.0
Generation 7, Best Fitness: -16.0
Generation 8, Best Fitness: -16.0
Generation 9, Best Fitness: -16.0
Generation 10, Best Fitness: -16.0
Generation 11, Best Fitness: -16.0
Generation 12, Best Fitness: -16.0
Generation 13, Best Fitness: -16.0
Generation 14, Best Fitness: -16.0
Generation 15, Best Fitness: -16.0
Generation 16, Best Fitness: -16.0
Generation 17, Best Fitness: -16.0
Generation 18, Best Fitness: -16.0
Generation 19, Best Fitness: -16.0
Generation 20, Best Fitness: -16.0
Generation 21, Best Fitness: -16.0
Generation 22, Best Fitness: -16.0
Generation 23, Best Fitness: -16.0
Generation 24, Best Fitness: -16.0
Generation 25, Best Fitness: -16.0
(( x ) * ( 3 ))
```

eighth function:

y = x - 4

```
x = np.linspace(-10, 10, 10000)
# y = np.linspace(0, 10, 10000)
y = x - 4
g = gplearn()
print(g.fit(x, y).inorder_traversal())
```

```
Generation 1, Best Fitness: -0.0
((( x ) + ( 0 )) - ( 5 ))
```