Computer vision – HW10

Name: WeiWen Wu

Student ID: 10901007A

Question 1

A、 Source code

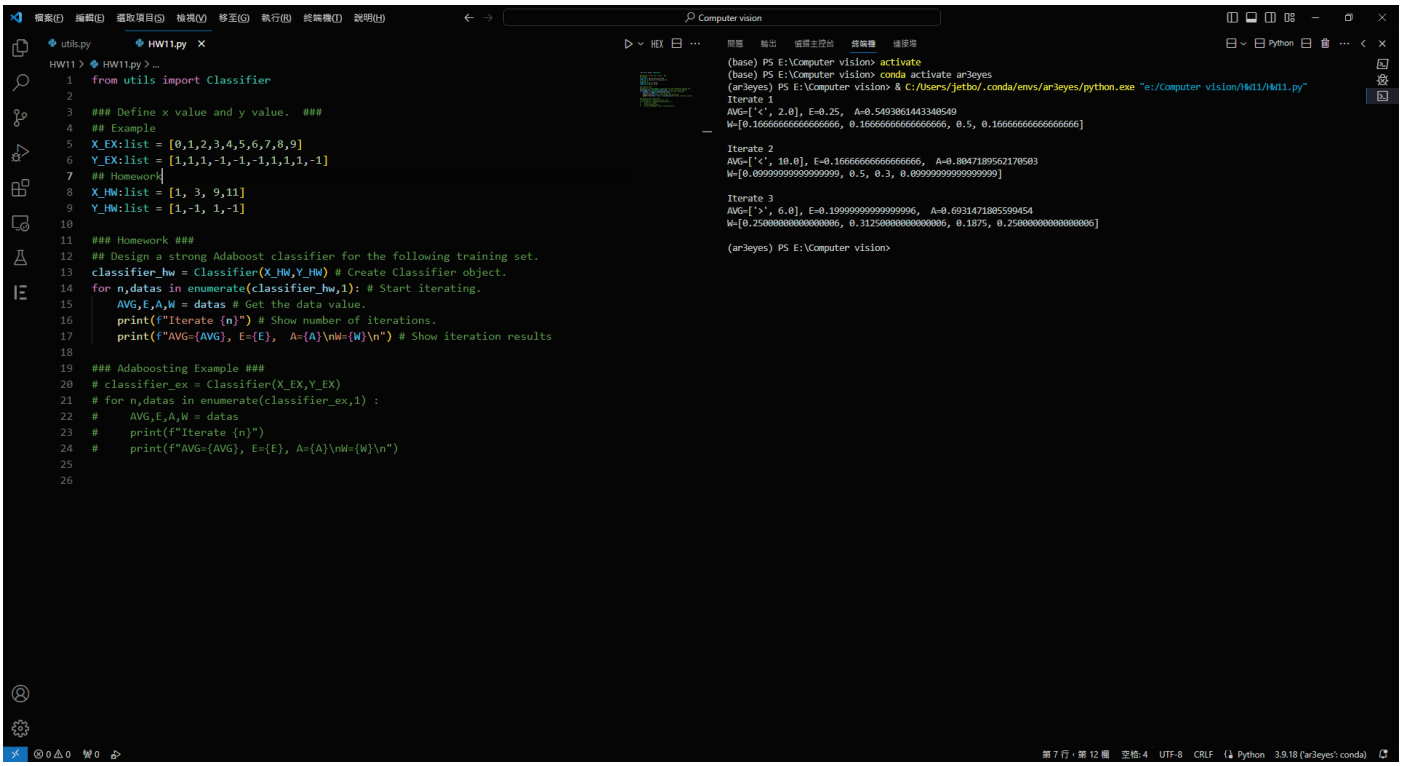
```
from utils import Classifier


### Define x value and y value.  ###
## Example
X_EX:list = [0,1,2,3,4,5,6,7,8,9]
Y_EX:list = [1,1,1,-1,-1,-1,1,1,1,-1]
## Homework
X_HW:list = [1, 3, 9,11]
Y_HW:list = [1,-1, 1,-1]

### Homework ###
## Design a strong Adaboost classifier for the following training set.
classifier_hw = Classifier(X_HW,Y_HW) # Create Classifier object.
for n,datas in enumerate(classifier_hw,1): # Start iterating.
    AVG,E,A,W = datas # Get the data value.
    print(f"Iterate {n}") # Show number of iterations.
    print(f"AVG={AVG}, E={E},  A={A}\nW={W}\n") # Show iteration results

### Adaboosting Example ###
# classifier_ex = Classifier(X_EX,Y_EX)
# for n,datas in enumerate(classifier_ex,1) :
#     AVG,E,A,W = datas
#     print(f"Iterate {n}")
#     print(f"AVG={AVG}, E={E}, A={A}\nW={W}\n")
```

B、 Result map


utils.py
HW11.py
HW11 > HW11.py > ...
from utils import Classifier
Define x value and y value.
Example
X_EX:list = [0,1,2,3,4,5,6,7,8,9]
Y_EX:list = [1,1,1,-1,-1,-1,1,1,1,-1]
Homework
X_HW:list = [1, 3, 9,11]
Y_HW:list = [1,-1, 1,-1]
Homework
Design a strong Adaboost classifier for the following training set.
classifier_hw = Classifier(X_HW,Y_HW) # Create Classifier object.
for n,datas in enumerate(classifier_hw,1): # Start iterating.
AVG,E,A,W = datas # Get the data value.
print(f"Iterate {n}") # Show number of iterations.
print(f"AVG={AVG}, E={E}, A={A}\nW={W}\n") # Show iteration results
Adaboosting Example
classifier_ex = Classifier(X_EX,Y_EX)
for n,datas in enumerate(classifier_ex,1) :
AVG,E,A,W = datas
print(f"Iterate {n}")
print(f"AVG={AVG}, E={E}, A={A}\nW={W}\n")
(base) PS E:\Computer vision> activate
(base) PS E:\Computer vision> conda activate ar3eyes
(ar3eyes) PS E:\Computer vision> & C:/Users/jetbo/.conda/envs/ar3eyes/python.exe "e:/Computer vision/HW11/HW11.py"
Iterate 1
AVG=['<', 2.0], E=0.25, A=0.5493061443340549
W=[0.16666666666666666, 0.16666666666666666, 0.5, 0.16666666666666666]
Iterate 2
AVG=['<', 10.0], E=0.16666666666666666, A=0.8047189562170503
W=[0.09999999999999999, 0.5, 0.3, 0.09999999999999999]
Iterate 3
AVG=['>', 6.0], E=0.19999999999999996, A=0.6931471805599454
W=[0.2500000000000006, 0.31250000000000006, 0.1875, 0.2500000000000006]
(ar3eyes) PS E:\Computer vision>

Appendix(utils.py)

```
# utils.py  WeiWen Wu
from typing import Union,Optional
from numpy  import log,exp,sign,float64

class Classifier:
    """
    Adaboost (Adaptive Boosting algorithm)

    # Example (for loop)
    ```
    classifier = Classifier(X,Y)
    for n,datas in enumerate(classifier,1):
        AVG,E,A,W = datas
        print(f"Iterate {n}")
        print(f"AVG={AVG}, E={E}, A={A}, W={W}")
    ```

    # Example (while loop)
    ```
    ### Parameter default value. ###
    classifier = Classifier(X_EX,Y_EX)
    verify = False

    ### Start iterating. ###
    while verify == False:
        AVG,E = classifier.error()
        A = classifier.alpha(E)
        W = classifier.update_weight(alpha=A,avg=AVG)
        verify = classifier.verify()
        print(f"AVG={AVG}, E={E}, A={A}, W={W}")
    ```
    """
    _verify:bool = False
    ### Record ###
    weight_record:list = []
    alpha_record:list = []
    avg_record:list = []
    error_record:list = []
    def __init__(self,x:list,y:list) -> None:
        _x_len = len(x)
```

```
        assert _x_len == len(y)
        self.X = x
        self.Y = y
        self.data_count = _x_len
        self.update_weight()

    def error(self,p:Optional[list]  = None) -> tuple[list[Union[str,int]],float]:
        """Train a weak classifier h(x) weighted training data minimizing the
error."""
        x,y = self.X,self.Y
        value:list = []
        ave:list[float] = []   # average
        data_count:int = self.data_count
        p = self.weight_record[-1] if p == None else p
        for i in range(1,data_count):
            ave.append(['<',(x[i]+x[i-1])/2])
            ave.append(['>',(x[i]+x[i-1])/2])
            _error:int = 0
            for j in range(data_count):
                if (lambda a : -1 if a == 0 else 1)(x[j] < ave[-1][1]) !=
y[j]:_error+=p[j]
            _error_scale = _error
            value.append(_error_scale)
            value.append(1-_error_scale)
        index = value.index(min(value))
        # print(value)
        # print(ave)
        self.avg_record.append(ave[index])
        self.error_record.append(min(value))

        return ave[index], min(value)

    def alpha(self,e:Optional[float]=None) -> float:
        """Compute voting weight of h(x)."""
        e = self.error_record[-1] if e == None else e
        assert e!=0, "<e> must not be 0"
        _result = float(0.5*log((1-e)/e))
        self.alpha_record.append(_result)
        return _result  # a=0.5*log((1-e)/e)
```

```
    def
update_weight(self,_w:Optional[list]=None,alpha:Optional[float]=None,avg:Optional[flo
at]=None) -> list:
        """Recompute weights (Weighting update)."""
        x,y = self.X,self.Y
        data_count = self.data_count
        weight_record = self.weight_record
        if len(weight_record)==0: # Initialize weights.
            _result = [1/data_count for _ in range(data_count)]
        else:
            _w = weight_record[-1] if _w == None else _w
            alpha = self.alpha_record[-1] if alpha == None else alpha
            avg = self.avg_record[-1] if avg == None else avg

            I = self.I
            _temp:list = []
            for n in range(data_count):
                # print(f"{sign(X,avg)[n]}, {Y[n]}")
                _exp:float64 = exp(-alpha*y[n]*I(x,avg[1],avg[0])[n])
                _temp.append(_w[n]*_exp)
            zt:int = sum(_temp)
            _result = [_/zt for _ in _temp] # Normalization.
        self.weight_record.append(_result)
        return _result  # w(i)=wt(i)*exp{-αt*yi*ht(xi)}/Zt

    def I(self,_value:list,threshold:float,symbol:str='<'):
        _result:list = []
        for value in _value:
            _ = 1 if value < threshold else -1
            _result.append(_)
        if symbol == '<':return _result
        elif symbol == '>':return [-_ for _ in _result]
        else: raise ValueError("<symbol> must be '<' or '>'")

    def
sign_of_h(self,alpha_lists:Optional[list]=None,avg_lists:Optional[list]=None):
        """sign(H(x))"""
        x = self.X
        alpha_lists = self.alpha_record if alpha_lists == None else alpha_lists
        avg_lists = self.avg_record if avg_lists == None else avg_lists
        _result:list = []
```

```
        assert len(alpha_lists)==len(avg_lists)
        for n in range(self.data_count):
            _total = 0
            for count in range(len(alpha_lists)):
                a=self.I(x,avg_lists[count][1],avg_lists[count][0])
                _total += alpha_lists[count]*a[n]
            _result.append(sign(_total))
        return _result

    def verify(self,alpha_lists:Optional[list]=None,avg_lists:Optional[list]=None) ->
bool:
        """Verify the answer is correct."""
        y = self.Y
        for n,value in enumerate(self.sign_of_h(alpha_lists,avg_lists)):
            if float(value) != y[n]:return False # If verification error, exit the for
loop.
        return True

    def __next__(self) -> tuple[list[str,int], float, float, list]:
        AVG,E = self.error() # Train a weak classifier weighted training data
minimizing the error.
        A = self.alpha() # Compute voting weight.
        W = self.update_weight() # Recompute weights.
        if self._verify==True: # If the verification is correct, the iteration will
end.
            raise StopIteration("The answer has been verified correct") # Verified
correct.
        self._verify = self.verify() # Verify the answer is correct.
        return AVG,E,A,W

    def __iter__(self): return self
```