**Computer vision – HW10**

[Question 1 2](#_Toc152357977)

[A、 Source code 2](#_Toc152357978)

[B、 Result map 3](#_Toc152357979)

[Appendix(utils.py) 4](#_Toc152357980)

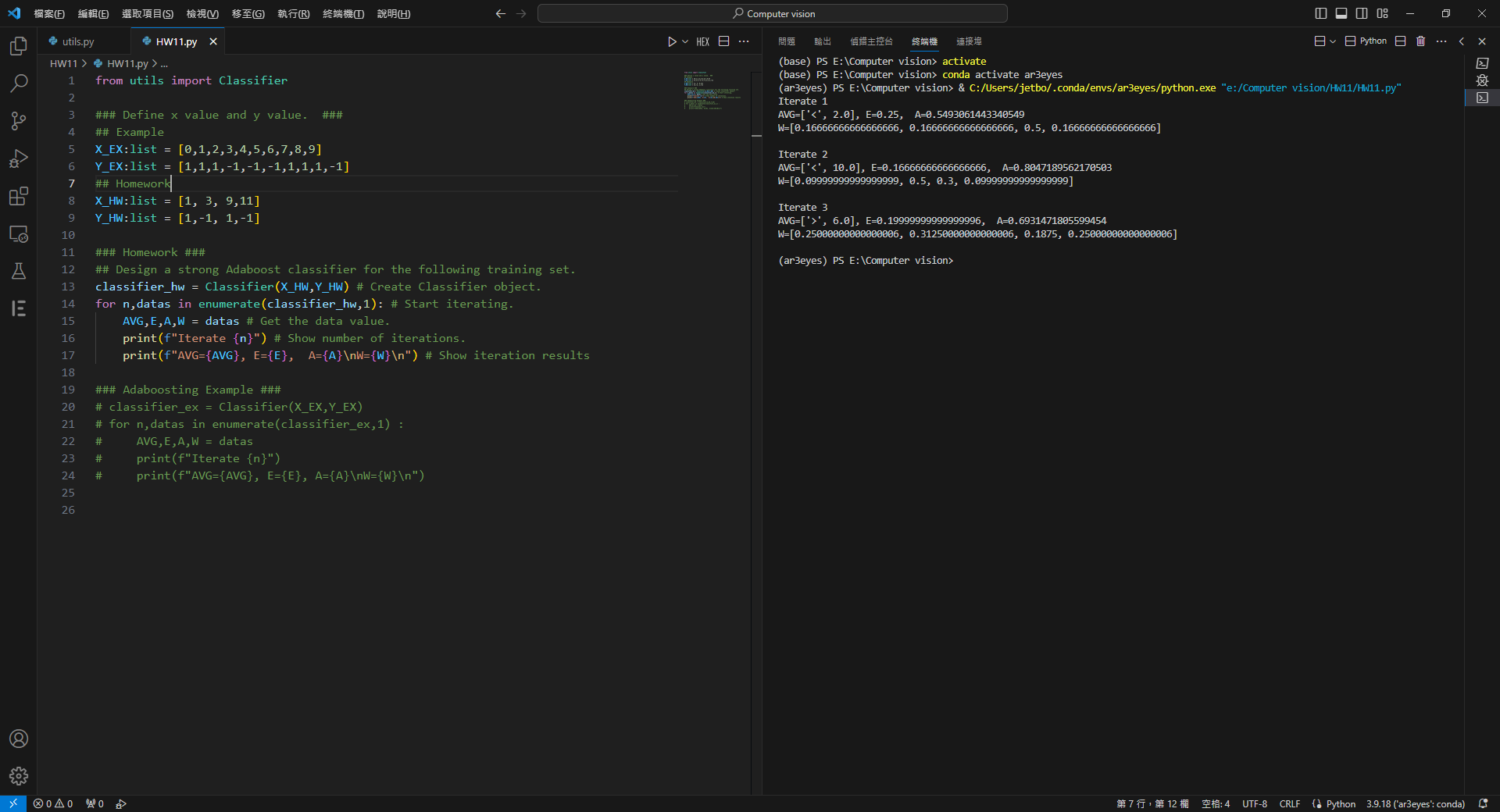
|  |  |
| --- | --- |
| Name: | WeiWen Wu |
| Student ID: | 10901007A |

# Question 1

## 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") |

## Result map



# 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**[float]=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 |