**التدريبات العملية لمقرر الذكاء الاصطناعي 1-1**

**تدريبات الوحدة الأولى**

**أساسيات الذكاء الاصطناعي**

**عدد تدريبات الوحدة 19**

**تدريبات اليوم الأول 15**

**1-1 : حذف و إضافة عناصر للمكدس :** **ص 30**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **تعريف المكدس و ادخال عناصره** | myStack=[1,21,32,45] |
| **طباعة عناصر المكدس** | print("Initial stack: ", myStack) |
| **طباعة العنصر المحذوف من المكدس** | print(myStack.pop()) |
| **طباعة عناصر المكدس الجديد** | print("The new stack after pop: ", myStack) |
| **اضافة عنصر جديد للمكدس** | myStack.append(78) |
| **طباعة المكدس النهائي** | print("The new stack after push: ", myStack) |
| **الكود البرمجي بالكامل** | |
| myStack=[1,21,32,45]  print("Initial stack: ", myStack)  print(myStack.pop())  print(myStack.pop())  print("The new stack after pop: ", myStack)  myStack.append(78)  print("The new stack after push: ", myStack) | |

**1-2 : حذف جميع عناصر المكدس : ص 30**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **تعريف المكدس و ادخال عناصره** | myStack=[1,21,32,45] |
| **طباعة عناصر المكدس** | print("Initial stack: ", myStack) |
| **عرض طول المكدس** | a=len(myStack) |
| **طباعة طول المكدس** | print("size of stack",a) |
| **حلقة تكرارية لحذف جميع عناصر المكدس و طباعته** | # empty the stack  for i in range(a):      myStack.pop()  print(myStack) |
| **حذف عنصر من مكدس فارغ !** | myStack.pop() |
| **الكود البرمجي بالكامل** | |
| myStack=[1,21,32,45]  print("Initial stack:", myStack)  a=len(myStack)  print("size of stack",a)  # empty the stack  for i in range(a):      myStack.pop()  print(myStack)  myStack.pop() | |

**1-3 : إضافة و حذف عناصر من المكدس وفقا لقائمة إجراءات : ص 31**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **دالة إضافة عنصر للمكدس** | def push(stack,element):     stack.append(element) |
| **دالة حذف عنصر من المكدس**  **دالة اذا كان المكدس فارغ** | def pop(stack):     return stack.pop() def isEmpty(stack):     return len(stack)==0 |
| **دالة انشاء مكدس** | def createStack():     return [] |
| **انشاء مكدس جديد بدون عناصر ( لحين اختيار احد الإجراءات )** | newStack=createStack() |
| **طباعة خيارات الإجراءات الثلاث المطلوبة كتوضيح للمستخدم** | while True:     print("The stack so far is:",newStack)     print("-----------------------------")     print("Choose 1 for push")     print("Choose 2 for pop")     print("Choose 3 for end")     print("-----------------------------")     choice=int(input("Enter your choice: ")) |
| **طباعة رسالة خطأ لأي رقم مخالف للإجراءات الثلاث** | while choice!=1 and choice!=2 and choice!=3:         print ("Error") |
| **السماح للمستخدم بإدخال رقم الاجراء** | choice=int(input("Enter your choice: ")) |
| **دالة شرطية لتنفيذ الأمر الصحيح وفقا لرقم الاجراء الذي ادخله المستخدم** | if choice==1:         x=int(input("Enter element for push: "))         push(newStack,x)     elif choice==2:         if not isEmpty(newStack):             print("The pop element is:",pop(newStack))         else:             print("The stack is empty")     else:         print("End of program")         break; |
| **الكود البرمجي بالكامل** | |
| def push(stack,element):     stack.append(element) def pop(stack):     return stack.pop() def isEmpty(stack):     return len(stack)==0 def createStack():     return [] newStack=createStack() while True:     print("The stack so far is:",newStack)     print("-----------------------------")     print("Choose 1 for push")     print("Choose 2 for pop")     print("Choose 3 for end")     print("-----------------------------")     choice=int(input("Enter your choice: "))     while choice!=1 and choice!=2 and choice!=3:         print ("Error")         choice=int(input("Enter your choice: "))     if choice==1:         x=int(input("Enter element for push: "))         push(newStack,x)     elif choice==2:         if not isEmpty(newStack):             print("The pop element is:",pop(newStack))         else:             print("The stack is empty")     else:         print("End of program")         break; | |

**1-4 : حذف و إضافة عناصر للطابور : ص 36**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **تعريف الطابور و ادخال عناصره** | myQueue=[1,21,32,45] |
| **طباعة عناصر الطابور** | print("Initial queue: ", myQueue) |
| **حذف العنصر من الطابور وفقا للمعامل المكتوب** | myQueue.pop(0) |
| **طباعة عناصر الطابور الجديد** | print("The new queue after pop: ", myQueue) |
| **اضافة عنصر جديد للطابور** | myQueue.append(78) |
| **طباعة الطابور النهائي** | print("The new queue after push: ", myQueue) |
| **الكود البرمجي بالكامل** | |
| myQueue=[1,21,32,45] print("Initial queue: ", myQueue) myQueue.pop(0) myQueue.pop(0) print("The new queue after pop: ", myQueue) myQueue.append(78) print("The new queue after push: ", myQueue) | |

**1-5 : حذف جميع عناصر الطابور: ص 36**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **تعريف الطابور و ادخال عناصره** | myQueue=[1,21,32,45] |
| **طباعة عناصر الطابور** | print("Initial queue: ", myQueue) |
| **عرض طول الطابور** | a=len(myQueue) |
| **طباعة طول الطابور** | print("size of queue ",a) |
| **حلقة تكرارية لحذف جميع عناصر الطابور و طباعته** | # empty the queue for i in range(a):     myQueue.pop(0) print(myQueue) |
| **حذف عنصر من طابور فارغ !** | myQueue.pop(0) |
| **الكود البرمجي بالكامل** | |
| myQueue=[1,21,32,45] print("Initial queue: ", myQueue) a=len(myQueue) print("size of queue ",a) # empty the queue for i in range(a):     myQueue.pop(0) print(myQueue) myQueue.pop(0) | |

**1-6 : انشاء طابور بوظيفة وحدة نمطية: ص 37**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **استيراد مكتبة الطابور** | from queue import\* |
| **انشاء طابور فارغ** | myQueue = Queue() |
| **ادخال عناصر الطابور بوظيفة وحده نمطية و طباعتها** | # add the elements in the queue  myQueue.put("a")  myQueue.put("b")  myQueue.put("c")  myQueue.put("d")  myQueue.put("e")  # print the elements of the queue  for element in list(myQueue.queue):  print(element) |
| **الكود البرمجي بالكامل** | |
| from queue import\*  myQueue = Queue()  # add the elements in the queue  myQueue.put("a")  myQueue.put("b")  myQueue.put("c")  myQueue.put("d")  myQueue.put("e")  # print the elements of the queue  for element in list(myQueue.queue):  print(element) | |

**1-7 : انشاء طابور بوظيفة وحدة نمطية لقيم يدخلها المستخدم: ص 38**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **استيراد مكتبة الطابور** | from queue import\* |
| **انشاء طابور فارغ** | myQueue = Queue() |
| **ادخال عناصر الطابور بوظيفة وحده نمطية و طباعتها** | for i in range(5):  element=input("enter queue element: ")  myQueue.put(element) |
| **طباعة عناصر الطابور داخل قائمة** | for element in list(myQueue.queue):  print(element) |
| **طباعة حجم الطابور** | print ("Queue size is: ",myQueue.qsize()) |
| **الكود البرمجي بالكامل** | |
| from queue import \*  myQueue = Queue()  for i in range(5):  element=input("enter queue element: ")  myQueue.put(element)  for element in list(myQueue.queue):  print(element)  print ("Queue size is: ",myQueue.qsize()) | |

**1-8 : التحقق من أن الطابور ممتلئ أو فارغ: ص 38**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **استيراد مكتبة الطابور** | import queue |
| **انشاء طابور فارغ** | myQueue = Queue() |
| **ادخال عناصر الطابور بوظيفة وحده نمطية** | myQueue.put("a")  myQueue.put("b")  myQueue.put("c")  myQueue.put("d")  myQueue.put("e") |
| **طباعة نتيجة التحقق من امتلاء الطابور بوظيفة وحده نمطية** | checkFull=myQueue.full()  print("Is the queue full? ", checkFull) |
| **طباعة نتيجة التحقق من فراغ الطابور بوظيفة وحده نمطية** | checkEmpty= myQueue.empty()  print("Is the queue empty? ", checkEmpty) |
| **الكود البرمجي بالكامل** | |
| import queue  myQueue = Queue()  myQueue.put("a")  myQueue.put("b")  myQueue.put("c")  myQueue.put("d")  myQueue.put("e")  checkFull=myQueue.full()  print("Is the queue full? ", checkFull)  checkEmpty= myQueue.empty()  print("Is the queue empty? ", checkEmpty) | |

**1-9 : انشاء مكدس و حذف عناصره بوظيفة وحدة نمطية : ص 39**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **استيراد مكتبة الطابور** | from queue import\* |
| **انشاء مكدس جديد بوظيفة وحدة نمطية** | myStack = LifoQueue() |
| **ادخال عناصر المكدس بوظيفة وحده نمطية** | myStack.put("a") myStack.put("b") myStack.put("c") myStack.put("d") myStack.put("e") |
| **طباعة عناصر المكدس المحذوفة بوظيفة وحدة نمطية وفقا لقاعدة LIFO** | for i in range(5):     k=myStack.get()     print(k) |
| **طباعة نتيجة التحقق من فراغ المكدس بوظيفة وحده نمطية** | checkEmpty= myStack.empty() print("Is the stack empty?", checkEmpty) |
| **الكود البرمجي بالكامل** | |
| from queue import \* myStack = LifoQueue() myStack.put("a") myStack.put("b") myStack.put("c") myStack.put("d") myStack.put("e") for i in range(5):     k=myStack.get()     print(k) checkEmpty= myStack.empty() print("Is the stack empty?", checkEmpty) | |

**1-10 : محاكاة طابور طباعة الملفات : ص 40-41**

**المتغيرات المستخدمة في الأمر البرمجي**

printDocument (متغير لطباعة الوثائق) و printQueueSize (حجم طابور الطباعة الحالي) و printQueueMaxSize (أقصى حجم لطابور الطباعة). كائن( طابور الطباعة) printQueue باستخدام فئة Queue من مكتبة queue، وتحديد الحجم الأقصى, دالة 1 ( فاحصة ) addDocument تضيف المستند لطابور الطباعة دالة 2 ( فاحصة ) printDocument تطبع المستند من طابور الطباعة

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| استيراد مكتبة "queue" المستخدمة لإنشاء هيكل البيانات "طابور".  استيراد مكتبة "time" لاستخدام وظيفة "sleep" لإضافة تأخير في البرنامج. | from queue import \*  import time |
| **تهيئة المتغيرات** | printDocument = " "  printQueueSize = 0  printQueueMaxSize = 7  printQueue = Queue(printQueueMaxSize) |
| يتم تعريف دالة 1 "addDocument" لإضافة مستند إلى طابور الطباعة. تقوم الدالة بفحص إذا ما كان طابور الطباعة ممتلئًا أم لا باستخدام الدالة "qsize" للحصول على حجم الطابور ومقارنته بالحجم الأقصى | def addDocument(document):  printQueueSize = printQueue.qsize() |

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| إذا كان الطابور ممتلئًا، يتم طباعة رسالة تشير إلى أن المستند لن يتم إضافته لأن طابور الطباعة ممتلئ  وإلا يتم إضافة المستند إلى طابور الطباعة باستخدام الدالة "put" ويتم استدعاء الدالة "sleep" لإضافة تأخير لمدة 0.5 ثانية لمحاكاة الوقت المستغرق لإضافة المستند و طباعة رسالة توضح ارسال المستند للطابور | if printQueueSize == printQueueMaxSize:  print("!! ", document, " was not sent to print queue.")  print("The print queue is full.")  print()  return  printQueue.put(document)  time.sleep(0.5) #Wait 5.0 seconds  print(document, " sent to print queue.")  printQueueSizeMessage() |
| تعريف دالة2 "printDocument" لطباعة المستند التالي في طابور الطباعة تقوم الدالة2 بفحص إذا ما كان طابور الطباعة فارغًا أم لا باستخدام الدالة "qsize" للحصول على حجم الطابور ومقارنته بالقيمة صفر. | def printDocument():  printQueueSize = printQueue.qsize() |
| إذا كان الطابور فارغًا، يتم طباعة رسالة تشير إلى أن الطابور فارغ  وإلا يتم استدعاء الدالة "get" لاسترداد الوثيقة التالية في الطابور ويتم استدعاء الدالة "sleep" لإضافة تأخير لمدة ثانية واحدة لمحاكاة الوقت المستغرق لطباعة المستندات | if printQueueSize == 0:  print("!! The print queue is empty.")  print()  return  printDocument = printQueue.get()  time.sleep(1) # wait one second  print ("OK - ", printDocument, " is printed.")  printQueueSizeMessage() |
| طباعة المستندات بالترتيب الذي تم اضافتها في الكود | def printQueueSizeMessage():  printQueueSize = printQueue.qsize()  if printQueueSize == 0:  print ("There are no documents waiting for printing.")  elif printQueueSize == 1:  print ("There is 1 document waiting for printing.")  else:  print ("There are ", printQueueSize, " documents waiting for printing.")  print()  addDocument("Document A")  addDocument("Document B")  addDocument("Document C")  addDocument("Document D")  addDocument("Document E")  addDocument("Document F")  addDocument("Document G")  printDocument()  addDocument("Document H")  printDocument()  addDocument("Document I")  printDocument()  addDocument("Document J")  addDocument("Document K")  printDocument()  printDocument()  printDocument()  printDocument()  printDocument()  printDocument()  printDocument()  printDocument() |
| **الكود البرمجي بالكامل** | |
| from queue import \*  import time  printDocument = " "  printQueueSize = 0  printQueueMaxSize = 7  printQueue = Queue(printQueueMaxSize)  # add a document to print the queue  def addDocument(document):  printQueueSize = printQueue.qsize()  if printQueueSize == printQueueMaxSize:  print("!! ", document, " was not sent to print queue.")  print("The print queue is full.")  print()  return  printQueue.put(document)  time.sleep(0.5) #Wait 5.0 seconds  print(document, " sent to print queue.")  printQueueSizeMessage()      # print a document from the print queue  def printDocument():  printQueueSize = printQueue.qsize()  if printQueueSize == 0:  print("!! The print queue is empty.")  print()  return  printDocument = printQueue.get()  time.sleep(1) # wait one second  print ("OK - ", printDocument, " is printed.")  printQueueSizeMessage()      # print a message with the size of the queue  def printQueueSizeMessage():  printQueueSize = printQueue.qsize()  if printQueueSize == 0:  print ("There are no documents waiting for printing.")  elif printQueueSize == 1:  print ("There is 1 document waiting for printing.")  else:  print ("There are ", printQueueSize, " documents waiting for printing.")    print()  addDocument("Document A")  addDocument("Document B")  addDocument("Document C")  addDocument("Document D")  addDocument("Document E")  addDocument("Document F")  addDocument("Document G")  printDocument()  addDocument("Document H")  printDocument()  addDocument("Document I")  printDocument()  addDocument("Document J")  addDocument("Document K")  printDocument()  printDocument()  printDocument()  printDocument()  printDocument()  printDocument()  printDocument()  printDocument() | |

**1-11 : انشاء عقدة باستخدام الفئة : ص 46**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **تعريف فئة العقدة و خصائصها** | class Node:     def \_\_init\_\_(self, data, next=None): |
| **بيان العقدة** | self.data = data # node data |
| **مؤشر العقدة الأخرى** | self.next = next # Pointer to the next node |
| **تخزين بيان العقدة و طباعتها** | first = Node("Monday") print(first.data) |
| **الكود البرمجي بالكامل** | |
| class Node:     def \_\_init\_\_(self, data, next=None):         self.data = data # node data         self.next = next # Pointer to the next node         # Create a single node first = Node("Monday") print(first.data) | |

**1-12 : انشاء قائمة مترابطة بعقدة واحدة : ص 46**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **تعريف فئة العقدة و خصائصها** | class Node:     def \_\_init\_\_(self, data = None, next=None): |
| **بيان العقدة** | self.data = data |
| **مؤشر العقدة الأخرى** | self.next = next |
| **تعريف فئة القائمة المترابطة و خصائصها** | class LinkedList:     def \_\_init\_\_(self): |
| **رأس عقدة البداية** | self.head = None |
| **تخزين بيان العقدة داخل القائمة المترابطة و طباعتها** | Linkedlist1 = LinkedList() Linkedlist1.head = Node("Monday") print(Linkedlist1.head.data) |
| **الكود البرمجي بالكامل** | |
| # single node class Node:     def \_\_init\_\_(self, data = None, next=None):         self.data = data         self.next = next     # linked list with one head node class LinkedList:     def \_\_init\_\_(self):         self.head = None     # list linked with a single node Linkedlist1 = LinkedList() Linkedlist1.head = Node("Monday") print(Linkedlist1.head.data) | |

**1-13 : انشاء قائمة مترابطة تحتوي عدة عقد:** **ص47**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **تعريف فئة لعقد و خصائصها** | class Node:     def \_\_init\_\_(self, data = None, next=None): |
| **بيان العقد** | self.data = data |
| **المؤشر للعقدة الأخرى** | self.next = next |
| **تعريف فئة القائمة المترابطة و خصائصها** | class LinkedList:     def \_\_init\_\_(self): |
| **رأس العقدة الأولى** | self.head = None |
| **تخزين بيانات العقد تسلسليا داخل القائمة المترابطة** | linked\_list = LinkedList() # the first node linked\_list.head = Node("Monday") # the second node linked\_list.head.next = Node("Tuesday") # the third node linked\_list.head.next.next = Node("Wednesday") # print the linked list node = linked\_list.head |
| **التنقل من عقدة لعقدة أخرى و طباعة العقد** | while node:     print (node.data)     node = node.next |
| **الكود البرمجي بالكامل** | |
| class Node:     def \_\_init\_\_(self, data = None, next=None):         self.data = data         self.next = next      # an empty linked list with a head node. class LinkedList:         def \_\_init\_\_(self):             self.head = None # the main program linked\_list = LinkedList() # the first node linked\_list.head = Node("Monday") # the second node linked\_list.head.next = Node("Tuesday") # the third node linked\_list.head.next.next = Node("Wednesday") # print the linked list node = linked\_list.head while node:     print (node.data)     node = node.next | |

**1-14 : إضافة عقدة إلى القائمة المترابطة : ص 48**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **تعريف فئة لعقد و خصائصها** | class Node:     def \_\_init\_\_(self, data = None, next=None): |
| **بيان العقد** | self.data = data |
| **مؤشر العقدة الأخرى** | self.next = next |
| **تعريف فئة القائمة المترابطة و خصائصها** | class LinkedList:     def \_\_init\_\_(self): |
| **رأس العقدة الأولى** | self.head = None |
| **دالة انشاء العقد** | def insertAfter(new, prev):  new\_node = Node(new)  new\_node.next = prev.next  prev.next = new\_node |
| **ادراج العقدتين للقائمة** | L\_list = LinkedList()  # add the first two nodes  L\_list.head = Node(12)  second = Node(99)  L\_list.head.next = second |
| **إضافة العقدة الجديدة بين العقدتين و طباعة القائمة المترابطة** | insertAfter(37, L\_list.head)  # print the linked list  node = L\_list.head  while node:  print (node.data)  node = node.next |
| **الكود البرمجي بالكامل** | |
| # single node  class Node:  def \_\_init\_\_(self, data = None, next=None):  self.data = data  self.next = next  # linked list with one head node  class LinkedList:  def \_init\_(self):  self.head = None  def insertAfter(new, prev):  # create the new node  new\_node = Node(new)  # make the next of the new node the same as the next of the previous node  new\_node.next = prev.next  # make the next of the previous node the new node  prev.next = new\_node  # create the linked list  L\_list = LinkedList()  # add the first two nodes  L\_list.head = Node(12)  second = Node(99)  L\_list.head.next = second  # insert the new node after node 12 (the head of the list)  insertAfter(37, L\_list.head)  # print the linked list  node = L\_list.head  while node:  print (node.data)  node = node.next | |

**1-15 : حذف عقدة من القائمة المترابطة :** **ص 49**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **تعريف فئة لعقد و خصائصها** | class Node:  def \_\_init\_\_(self, data = None, next=None):  self.data = data  self.next = next  class LinkedList:  def \_\_init\_\_(self):  self.head = None |
| **تعريف دالة حذف العقدة** | def deleteNode(key, follow):  # store the head node  temp = follow.head |
| **حلقة تكرارية لتحديد العقد و مؤشرات العقد** | while(temp is not None):  if temp.data == key:  break  prev = temp  temp = temp.next  prev.next = temp.next  temp = None |
| **ادراج العقد للقائمة وربط المؤشرات** | L\_list = LinkedList()  # add the first three nodes  L\_list.head = Node(12)  second = Node(37)  third = Node(99)  L\_list.head.next = second  second.next = third |
| **حذف العقدة المطلوبة و طباعة بقية العقد** | deleteNode(37,L\_list)  # print the linked list  node = L\_list.head  while node:  print (node.data)  node = node.next |
| **الكود البرمجي بالكامل** | |
| # single node  class Node:  def \_\_init\_\_(self, data = None, next=None):  self.data = data  self.next = next  # linked list with one head node  class LinkedList:  def \_\_init\_\_(self):  self.head = None  def deleteNode(key, follow):  # store the head node  temp = follow.head  # find the key to be deleted,  # the trace of the previous node to be changed  while(temp is not None):  if temp.data == key:  break  prev = temp  temp = temp.next  # unlink the node from the linked list  prev.next = temp.next  temp = None  # create the linked list  L\_list = LinkedList()  # add the first three nodes  L\_list.head = Node(12)  second = Node(37)  third = Node(99)  L\_list.head.next = second  second.next = third  # delete node 37  deleteNode(37,L\_list)  # print the linked list  node = L\_list.head  while node:  print (node.data)  node = node.next | |

**التدريبات العملية لمقرر الذكاء الاصطناعي 1-1**

**تدريبات الوحدة الأولى**

**أساسيات الذكاء الاصطناعي**

**عدد تدريبات الوحدة 19**

**تدريبات اليوم الثاني 4**

**1-16 : إنشاء شجرة باستخدام قاموس البايثون : ص 56**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **إنشاء الشجرة** | myTree = { |
| **انشاء الجذر** | "a": ["b", "c"], # node |
| **انشاء العقد وفقا للمخطط** | "b": ["d", "e"],     "c": [None, "f"],     "d": [None, None],     "e": [None, None],     "f": [None, None], |
| **طباعة عناصر الشجرة** | print(myTree) |
| **الكود البرمجي بالكامل** | |
| myTree = {     "a": ["b", "c"], # node     "b": ["d", "e"],     "c": [None, "f"],     "d": [None, None],     "e": [None, None],     "f": [None, None],     } print(myTree) | |

**1-17 : إنشاء شجرة بطباعة عدد العقد المنبثقة و عناصرها : ص 56**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **إنشاء الشجرة بتحديد الأصل و الفرع لكل مستوى** | myTree = {"Data Structures":["Linear","Non-linear"],      "Linear":["Stack","Queue","Linked List"],     "Non-linear":["Tree", "Graph"]} |
| **حلقة تكرارية لأصول الشجرة مع طباعة الأصل و عدد عقد كل أصل** | for parent in myTree:     print(parent, "has",len(myTree[parent]),"nodes" ) |
| **حلقة تكرارية لفروع الشجرة مع طباعة الفرع** | for children in myTree[parent]:         print(" ",children) |
| **الكود البرمجي بالكامل** | |
| myTree = {"Data Structures":["Linear","Non-linear"],     "Linear":["Stack","Queue","Linked List"],     "Non-linear":["Tree", "Graph"]} for parent in myTree:     print(parent, "has",len(myTree[parent]),"nodes" )     for children in myTree[parent]:         print(" ",children) | |

**1-18 : إنشاء مخطط موجه : ص 61**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **إنشاء عقد المخطط الموجه** | myGraph = { "a" : ["b","c"],  "b" : ["c", "d"],  "c" : ["d", "e"],  "d" : [],  "e" : [],  } |
| **طباعة المخطط الموجه** | print(myGraph) |
| **الكود البرمجي بالكامل** | |
| myGraph = { "a" : ["b","c"],  "b" : ["c", "d"],  "c" : ["d", "e"],  "d" : [],  "e" : [],  }  print(myGraph) | |

**1-19 : إنشاء مخطط غير موجه و إضافة حواف جديدة : ص 61-62**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **انشاء دالة اضافة الحواف للعقد في المخطط الغير موجه لاستدعائها لاحقا** | # function for adding an edge to a graph  def addEdge(graph,u,v):  graph[u].append(v)  def generate\_edges(graph):  edges = [] |
| **حلقات تكرارية لاضافة الحواف للعقد و العقد المجاورة لها** | for node in graph:  for neighbour in graph[node]:  # if edge exists then append to the list  edges.append((node, neighbour))  return edges |
| **انشاء المخطط الغير موجه و تحديد العقد و مساراتها و طباعة المخطط** | myGraph = {"a" : ["b","c"],  "b" : ["c", "d"],  "c" : ["d", "e"],  "d" : [],  "e" : [],  }  print("The graph contents")  print(generate\_edges(myGraph)) |
| **استدعاء دالة إضافة الحواف للمخطط وفق العقد المطلوبة وطباعة المخطط الغير متجه الجديد** | addEdge(myGraph,'a','e')  addEdge(myGraph,'c','f')  print("The new graph after adding new edges")  print(generate\_edges(myGraph)) |
| **الكود البرمجي بالكامل** | |
| # function for adding an edge to a graph  def addEdge(graph,u,v):  graph[u].append(v)  # function for generating the edges of a graph  def generate\_edges(graph):  edges = []  # for each node in graph  for node in graph:  # for each neighbouring node of a single node  for neighbour in graph[node]:  # if edge exists then append to the list  edges.append((node, neighbour))  return edges  # main program  # initialisation of graph as dictionary  myGraph = {"a" : ["b","c"],  "b" : ["c", "d"],  "c" : ["d", "e"],  "d" : [],  "e" : [],  }  # print the graph contents  print("The graph contents")  print(generate\_edges(myGraph))  # add more edges to the graph  addEdge(myGraph,'a','e')  addEdge(myGraph,'c','f')  # print the graph after adding new edges  print("The new graph after adding new edges")  print(generate\_edges(myGraph)) | |

**التدريبات العملية لمقرر الذكاء الاصطناعي 1-1**

**تدريبات الوحدة الثانية**

**خوارزميات الذكاء الاصطناعي**

**عدد تدريبات الوحدة 19**

**تدريبات اليوم الثاني 4**

**2-1 : حساب المتوسط : ص 72**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **أنشاء دالة للجمع لاستدعائها لاحقا**  **تحديد طول القائمة ( عدد الاعداد )**  **و حلقة تكرارية لجمع الرقم الجديد مع الأرقام السابقة في القائمة** | def mySumGrade (gradesList):     sumGrade=0     l=len(gradesList)     for i in range(l):         sumGrade=sumGrade+gradesList[i]     return sumGrade |
| **انشاء دالة للمتوسط لاستدعائها لاحقا** | def avgFunc (gradesList):     s=mySumGrade(gradesList)     l=len(gradesList)     avg=s/l     return avg |
| **ادخال العناصر** | grades=[89,88,98,95] |
| **استدعاء دالة المتوسط ثم طباعة قيمة المتوسط** | averageGrade=avgFunc(grades) print ("The average grade is: ",averageGrade) |
| **الكود البرمجي بالكامل** | |
| def mySumGrade (gradesList):     sumGrade=0     l=len(gradesList)     for i in range(l):         sumGrade=sumGrade+gradesList[i]     return sumGrade def avgFunc (gradesList):     s=mySumGrade(gradesList)     l=len(gradesList)     avg=s/l     return avg # program section grades=[89,88,98,95] averageGrade=avgFunc(grades) print ("The average grade is: ",averageGrade) | |

**2-2 : حساب مضروب رقم باستخدام حلقة for : ص 73**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **أنشاء دالة المضروب لاستدعائها لاحقا** | def factorialLoop(n): |
| **حالتي دالة الاستدعاء التكرارية** | result = 1  for i in range(2,n+1):  result = result \* i  return result |
| **ادخال العدد و استدعاء دالة المضروب** | num = int(input("Type a number: ")) f=factorial(num) |
| **طباعة نتيجة مضروب العدد** | print("The factorial of ", num, " is: ", f) |
| **الكود البرمجي بالكامل** | |
| def factorialLoop(n):  result = 1  for i in range(2,n+1):  result = result \* i  return result  # main program  num = int(input("Type a number: "))  f=factorialLoop(num)  print("The factorial of ", num, " is:",f) | |

**2-3 : حساب مضروب رقم بدالة المضروب : ص 74**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **أنشاء دالة المضروب لاستدعائها لاحقا** | def factorial(x): |
| **حالتي دالة الاستدعاء التكرارية**  **تطبيق معادلة المضروب** | if x == 0:         return 1     else:         return (x \* factorial(x-1)) |
| **ادخال العدد و استدعاء دالة المضروب** | num = int(input("Type a number: ")) f=factorial(num) |
| **طباعة نتيجة مضروب العدد** | print("The factorial of ", num, " is: ", f) |
| **الكود البرمجي بالكامل** | |
| def factorial(x):     if x == 0:         return 1     else:         return (x \* factorial(x-1)) # main program num = int(input("Type a number: ")) f=factorial(num) print("The factorial of ", num, " is: ", f) | |

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **أنشاء دالة للعدد الأكبر ليتم استدعائها لاحقا** | def findMaxRecursion(A,n): |
| **حالتي دالة الاستدعاء التكرارية**  **اذا كانت القائمة مكونة من عنصر واحد فهو الأكبر أو يتم مقارنة عناصر القائمة وفقا لفهرس العناصر** | if n==1:  m = A[n-1]  else:  m = max(A[n-1],findMaxRecursion(A,n-1))  return m |
| **تخزين الاعداد في القائمة و تخزين طول القائمة** | myList = [4,73,-5,42]  l = len(myList) |
| **استدعاء الدالة و طباعة أكبر عنصر فيها** | myMaxRecursion = findMaxRecursion(myList,l)  print("Max with recursion is: ", myMaxRecursion) |
| **الكود البرمجي بالكامل** | |
| def findMaxRecursion(A,n):  if n==1:  m = A[n-1]  else:  m = max(A[n-1],findMaxRecursion(A,n-1))  return m  myList = [4,73,-5,42]  l = len(myList)  myMaxRecursion = findMaxRecursion(myList,l)  print("Max with recursion is: ", myMaxRecursion) | |

**2-4 بالطريقة الثانية : استخراج أكبر عنصر في قائمة : ص 75**

**2-4 بالطريقة الأولى : استخراج أكبر عنصر في قائمة : ص 75**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **أنشاء دالة للعدد الأكبر ليتم استدعائها لاحقا** | def findMaxIteration(A,n): |
| **حالتي دالة الاستدعاء التكرارية**  **حلقة تكرارية لادخال عناصر القائمة و مقارنة كل عنصرين , تحديد الأكبر بينهما** | m = A[0]  for i in range(1,n):  m = max(m,A[i])  return m |
| **تخزين الاعداد في القائمة و تخزين طول القائمة** | myList = [4,73,-5,42]  l = len(myList) |
| **استدعاء الدالة و طباعة أكبر عنصر فيها** | myMaxIteration = findMaxIteration(myList,l)  print("Max with iteration is: ", myMaxIteration) |
| **الكود البرمجي بالكامل** | |
| def findMaxIteration(A,n):  m = A[0]  for i in range(1,n):  m = max(m,A[i])  return m  # main program  myList = [4,73,-5,42]  l = len(myList)  myMaxIteration = findMaxIteration(myList,l)  print("Max with iteration is: ", myMaxIteration) | |

**2-5 بالطريقة الأولى : حساب مضاعف رقم : ص 76**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **أنشاء دالة لحساب مضاعف العدد لاستدعائها لاحقا** | def powerFunRecursive(baseNum,expNum): |
| **حالتي دالة الاستدعاء التكرارية**  **أذا كانت قوة العدد = 1 سيطبع العدد كما هو و إلا سيحسب الدالة بضرب العدد مرات عدد القوة** | if(expNum==1):  return(baseNum)  else:  return(baseNum\*powerFunRecursive(baseNum,expNum-1)) |
| **ادخال العدد و القوة المرفوع لها** | base = int(input("Enter number: "))  exp = int(input("Enter exponent: ")) |
| **استدعاء الدالة و**  **طباعة نتيجة مضاعف العدد** | numPowerRecursion = powerFunRecursive(base,exp)  print( "Recursion: ", base, " raised to ", exp, " = ",numPowerRecursion) |
| **الكود البرمجي بالكامل** | |
| def powerFunRecursive(baseNum,expNum):  if(expNum==1):  return(baseNum)  else:  return(baseNum\*powerFunRecursive(baseNum,expNum-1))  # main program  base = int(input("Enter number: "))  exp = int(input("Enter exponent: "))  numPowerRecursion = powerFunRecursive(base,exp)  print( "Recursion: ", base, " raised to ", exp, " = ",numPowerRecursion) | |

**2-5 بالطريقة الثانية : حساب مضاعف رقم : ص 76**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **أنشاء دالة لحساب مضاعف العدد لاستدعائها لاحقا** | def powerFunIteration(baseNum,expNum): |
| **حالتي دالة الاستدعاء التكرارية**  **وضع القوة = 1 ثم حلقة تكرارية لضرب العدد في القوة** | numPower = 1  for i in range(exp):  numPower = numPower\*base  return numPower |
| **ادخال العدد و القوة المرفوع لها** | base = int(input("Enter number: "))  exp = int(input("Enter exponent: ")) |
| **استدعاء الدالة و**  **طباعة نتيجة مضاعف العدد** | numPowerIteration = powerFunIteration(base,exp)  print( "Iteration: ", base, " raised to ", exp, " = ",numPowerIteration) |
| **الكود البرمجي بالكامل** | |
| def powerFunIteration(baseNum,expNum):  numPower = 1  for i in range(exp):  numPower = numPower\*base  return numPower  base = int(input("Enter number: "))  exp = int(input("Enter exponent: "))  numPowerIteration = powerFunIteration(base,exp)  print( "Iteration: ", base, " raised to ", exp, " = ",numPowerIteration) | |

**2-6 : تطبيق خوارزمية الاتساعBFS على مخطط : ص 81-82**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **ادخال عقد المخطط** | graph = {  "A" : ["B","C"],  "B" : ["D","E"],  "C" : ["F"],  "D" : [],  "E" : [],  "F" : []  } |
| **أنشاء دالة خوارزمية الاتساع لاستدعائها لاحقا** | visitedBFS = []  def bfs(visited, graph, node):  visited.append(node)  queue.append(node) |
| **حلقة تكرارية لحذف العقدة من الطابور و طباعتها**  **فحص العقدة التالية حسب المستوى و اضافتها للطابور** | while queue:  n = queue.pop(0)  print (n, end = " ")  for neighbor in graph[n]:  if neighbor not in visited:  visited.append(neighbor)  queue.append(neighbor) |
| **طباعة جميع عقد المخطط بدأ من الجذر A بخوارزمية BFS** | bfs(visitedBFS, graph, "A") |
| **الكود البرمجي بالكامل** | |
| graph = {  "A" : ["B","C"],  "B" : ["D","E"],  "C" : ["F"],  "D" : [],  "E" : [],  "F" : []  }  visitedBFS = [] # List to keep track of visited nodes  queue = [] # Initialize a queue  # bfs function  def bfs(visited, graph, node):  visited.append(node)  queue.append(node)  while queue:  n = queue.pop(0)  print (n, end = " ")  for neighbor in graph[n]:  if neighbor not in visited:  visited.append(neighbor)  queue.append(neighbor)  # main program  bfs(visitedBFS, graph, "A") | |

**2-7 : تطبيق خوارزمية العمق DFS على مخطط : ص 84**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **ادخال عقد المخطط** | graph = {  "A" : ["B","C"],  "B" : ["D","E"],  "C" : ["F"],  "D" : [],  "E" : [],  "F" : []  } |
| **أنشاء دالة خوارزمية العمق لاستدعائها لاحقا يحتوي تعريف الدالة على ( العقد التي تم زيارتها و المخطط و العقدة التي سنبدأ بها )**  **فحص العقد الى الحواف و اضافتها للمكدس ( يستخدم المكدس بصورة غير مباشرة لتتبع الاستدعاءات التكرارية )** | visitedDFS = []  def dfs(visited, graph, node):     if node not in visited:         print(node, end=" ")         visited.append(node) |
| **اظهار نتيجة بحث عقد المخطط بخوارزمية DFS** | for neighbor in graph[node]:             dfs(visited, graph, neighbor) dfs(visitedDFS, graph, "A") |
| **الكود البرمجي بالكامل** | |
| graph = {     "A": ["B", "C"],     "B": ["D", "E"],     "C": ["F"],     "D": [],     "E": [],     "F": [] } visitedDFS = []  # list to keep track of visited nodes # dfs function def dfs(visited, graph, node):     if node not in visited:         print(node, end=" ")         visited.append(node)         for neighbor in graph[node]:             dfs(visited, graph, neighbor) dfs(visitedDFS, graph, "A") | |

**التدريبات العملية لمقرر الذكاء الاصطناعي 1-1**

**تدريبات الوحدة الثانية**

**خوارزميات الذكاء الاصطناعي**

**6 تدريبات**

**2-8 : التشخيص الطبي الاصدار الأول: ص 90 الى92**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الكود البرمجي بالكامل** |
| **استدعاء مكتبة JSON**  **متغير لتخزين بيانات ملف الإصدار الأول** 'symptom\_mapping\_v1  **فتح الملف السابق في مكتبة JOSN على شكل قاموس و اسناده لمتغير**  **طباعة البيانات بتنسيق و اختيار المسافة البادئة المرغوبة** | import json # a library used to save and load JSON files  # the file with the symptom mapping  symptom\_mapping\_file='symptom\_mapping\_v1.json'  # open the mapping JSON file and load it into a dictionary  with open(symptom\_mapping\_file) as f:  mapping=json.load(f)  # print the JSON file  print(json.dumps(mapping, indent=2)) |
| **تصنيف المرض وفقا للعرض المصاحب :**  **تعريف دالة على هيئة قائمة يكون معاملها هو العرض المرضي و جعلها قائمة فارغة**  **دالة الشرط if لربط الأعراض بالتصنيف الملائم للمرض الطبي**  **استخدام قاعدة المعرفة التالية**  **( اذا كان لدى المريض على الأقل ثلاثا من جميع الأعراض المحتملة للمرض يضاف المرض كتشخيص محتمل)** | def diagnose\_v1(patient\_symptoms:list):  diagnosis=[] # the list of possible diseases  if "vomiting" in patient\_symptoms:  if "abdominal pain" in patient\_symptoms:  if "diarrhea" in patient\_symptoms:  # 1:vomiting, 2:abdominal pain, 3:diarrhea  diagnosis.append('food poisoning')  elif 'fever' in patient\_symptoms:  # 1:vomiting, 2:abdominal pain, 3:fever  diagnosis.append('food poisoning')  diagnosis.append('appendicitis')  elif "lower back pain" in patient\_symptoms and 'fever' in patient\_symptoms:  # 1:vomiting, 2:lower back pain, 3:fever  diagnosis.append('kidney stones')  elif "abdominal pain" in patient\_symptoms and\  "diarrhea" in patient\_symptoms and\  "fever" in patient\_symptoms:\  # 1:abdominal pain, 2:diarrhea, 3:fever  diagnosis.append('food poisoning')  return diagnosis |
| **اختبار النظام على ثلاث مرض**  **ادخال اعراض ثلاث مرضى و تشخيص المرض وفقا للأعراض المدونة** | # Patient 1  my\_symptoms=['abdominal pain', 'fever', 'vomiting']  diagnosis=diagnose\_v1(my\_symptoms)  print('Most likely diagnosis:',diagnosis)  # Patient 2  my\_symptoms=['vomiting', 'lower back pain', 'fever' ]  diagnosis=diagnose\_v1(my\_symptoms)  print('Most likely diagnosis:',diagnosis)  # Patient 3  my\_symptoms=['fever', 'cough', 'vomiting']  diagnosis=diagnose\_v1(my\_symptoms)  print('Most likely diagnosis:',diagnosis) |

**2-8 : التشخيص الطبي الاصدار الثاني: ص 93 الى95**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الكود البرمجي بالكامل** |
| **متغير لتخزين بيانات ملف الإصدار الثاني** 'symptom\_mapping\_v2  **فتح الملف السابق في مكتبة JOSN على شكل قاموس و اسناده لمتغير**  **طباعة البيانات بتنسيق و اختيار المسافة البادئة المرغوبة** | symptom\_mapping\_file='symptom\_mapping\_v2.json'  with open(symptom\_mapping\_file) as f:      mapping=json.load(f)  print(json.dumps(mapping, indent=2)) |
| **تصنيف المرض وفقا للعرض المصاحب :**  **تعريف دالة على هيئة قائمة بثلاث معاملات ( قائمة الأعراض – ملف تعيين الأمراض – الحد الأدنى من الاعراض المتطابقة ) و جعلها قائمة فارغة**  **فتح ملف تعيين الأمراض كقاموس**  **استخدام حلقةFOR بفحص كل مرض بشكل فردي**  **اذا كانت قيمة العداد تفوق الحد الأدنى من الأعراض يتم إضافة المرض**  **استخدام قاعدة المعرفة التالية**  **( حساب عدد الأعراض المطابقة لكل مرض و السماح للمستخدم بتحديد عدد الأعراض المطابقة التي يجب توافرها في المرض لتضمينه في التشخيص )** | def diagnose\_v2(patient\_symptoms:list,  symptom\_mapping\_file:str,  matching\_symptoms\_lower\_bound:int):  diagnosis=[]  with open(symptom\_mapping\_file) as f:  mapping=json.load(f)  disease\_info=mapping['diseases']  for disease in disease\_info:  counter=0  disease\_symptoms=disease\_info[disease]  for symptom in patient\_symptoms:  if symptom in disease\_symptoms:  counter+=1  if counter>=matching\_symptoms\_lower\_bound:  diagnosis.append(disease)  return diagnosis  my\_symptoms=["stuffy nose", "runny nose", "sneezing", "sore throat"]  diagnosis=diagnose\_v2(my\_symptoms,'symptom\_mapping\_v2.json' , 3)  print('Most likely diagnosis:',diagnosis)  my\_symptoms=["stuffy nose", "runny nose", "sneezing", "sore throat"]  diagnosis=diagnose\_v2(my\_symptoms, 'symptom\_mapping\_v2.json' , 4)  print('Most likely diagnosis:',diagnosis)  my\_symptoms=['fever', 'cough', 'vomiting']  diagnosis=diagnose\_v2(my\_symptoms, 'symptom\_mapping\_v2.json' , 3)  print('Most likely diagnosis:',diagnosis) |
| **ادخال اعراض ثلاث مرضى و تشخيص المرض وفقا بمقارنة كل عرض على حدة مع الاعراض المعروفة للمرض و زيادة العداد في كل مرة يجد تطابق**  **( وفقا لعدد الاعراض )** | # Patient 1  my\_symptoms=["stuffy nose", "runny nose", "sneezing", "sore throat"]  diagnosis=diagnose\_v2(my\_symptoms,'symptom\_mapping\_v2.json' , 3)  print('Most likely diagnosis:',diagnosis)  # Patient 2  my\_symptoms=["stuffy nose", "runny nose", "sneezing", "sore throat"]  diagnosis=diagnose\_v2(my\_symptoms, 'symptom\_mapping\_v2.json' , 4)  print('Most likely diagnosis:',diagnosis)  # Patient 3  my\_symptoms=['fever', 'cough', 'vomiting']  diagnosis=diagnose\_v2(my\_symptoms, 'symptom\_mapping\_v2.json' , 3)  print('Most likely diagnosis:',diagnosis) |

**2-8 : التشخيص الطبي الاصدار الثالث: ص 96 الى98**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الكود البرمجي بالكامل** |
| **متغير لتخزين بيانات ملف الإصدار الثالث** 'symptom\_mapping\_ v3  **فتح الملف السابق في مكتبة JOSN على شكل قاموس و اسناده لمتغير**  **طباعة البيانات بتنسيق و اختيار المسافة البادئة المرغوبة** | symptom\_mapping\_file='symptom\_mapping\_v3.json'  # open the mapping JSON file and load it into a dictionary  with open(symptom\_mapping\_file) as f:      mapping=json.load(f)  # print the JSON file  print(json.dumps(mapping, indent=2)) |
| **تعريف دالة تحتوي متغيرات ( أعراض المريض – جميع الأعراض للأمراض المختلفة – وزن للأعراض الأكثر شيوعا – ووزن للأعراض الأقل شيوعا )**  **فتح ملف المرض كقاموس**  **متغير يحتوي معلومات الأمراض**  **مقارنة أعراض المريض بأعراض كل مرض ( هل هي اعراض شائعة أم أقل شيوعا )**  **استخدام قاعدة المعرفة التالية**  **( بعض الأعراض أكثر شيوعا من أخرى للمرض نفسه , فيتم إعطاء أوزان مخصصة للأعراض الأكثر و الأقل شيوعا و المستخدم يحدد الأوزان التي يراها مناسبة فيشخص المرض أو الأمراض ذات المجموع الموزون الأعلى في التشخيص )**  **مقارنة اعلى درجة اشتباه**  **اذا كانت مساويه للصفر تعود الدالة بقيمة فارغة أو يطبع جميع الأمراض التي لها الدرجة القصوى** | from collections import defaultdict  def diagnose\_v3(patient\_symptoms:list,  symptom\_mapping\_file:str,  very\_common\_weight:float=1,  less\_common\_weight:float=0.5  ):  with open(symptom\_mapping\_file) as f:  mapping=json.load(f)  disease\_info=mapping['diseases']  # holds a symptom-based score for each potential disease  disease\_scores=defaultdict(int)  for disease in disease\_info:  # get the very common symptoms of the disease  very\_common\_symptoms=disease\_info[disease]['very common']  # get the less common symptoms for this disease  less\_common\_symptoms=disease\_info[disease]['less common']  for symptom in patient\_symptoms:  if symptom in very\_common\_symptoms:  disease\_scores[disease]+=very\_common\_weight  elif symptom in less\_common\_symptoms:  disease\_scores[disease]+=less\_common\_weight  # find the max score all candidate diseases  max\_score=max(disease\_scores.values())  if max\_score==0:  return []  else:  # get all diseases that have the max score  diagnosis=[disease for disease in disease\_scores if disease\_scores  [disease]==max\_score]  return diagnosis, max\_score |
| **ادخال اعراض ثلاث مرضى و تشخيص المرض وفقا لوزن الأعراض الأكثر شيوعا للمرض** | # Patient 1  my\_symptoms=["headache", "tiredness", "cough"]  diagnosis=diagnose\_v3(my\_symptoms, 'symptom\_mapping\_v3.json')  print('Most likely diagnosis:',diagnosis)  # Patient 2  my\_symptoms=["stuffy nose", "runny nose", "sneezing", "sore throat"]  diagnosis=diagnose\_v3(my\_symptoms, 'symptom\_mapping\_v3.json')  print('Most likely diagnosis:',diagnosis)  # Patient 3  my\_symptoms=["stuffy nose", "runny nose", "sneezing", "sore throat"]  diagnosis=diagnose\_v3(my\_symptoms, 'symptom\_mapping\_v3.json', 1, 1)  print('Most likely diagnosis:',diagnosis) |

**2-8 : التشخيص الطبي الاصدار الرابع: ص 99 الى104**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الكود البرمجي بالكامل** |
| **استدعاء مكتبة pandas لتحليل البيانات و استيراد ملفات من نوع css و تخزينها في متغير**    **( البيانات تشمل ملف 2000 مريض و العمود الأخير يمثل تشخيص الخبير البشري )** | import pandas as pd # import pandas to load and process spreadsheet-type data  medical\_dataset=pd.read\_csv('medical\_data.csv') # load a medical dataset.  medical\_dataset |
| **امر برمجي اخياري اذا رغبت في التأكد من الأمراض التي شخصها الخبير البشري** | set(medical\_dataset['diagnosis']) |
| **استخدام خوارزميات معينة عبر مكتبة skleearn الخاصة بتعليم الآلة و نمذجتها**  **تهيئة الألة لتدريب على النموذج**  **استخدام نموذج شجرة القرار ( مصنف شجرة القرار ) من مكتبة البايثون سكليرن skearn**  **مرحلة التدريب**  **اسقاط عمود التشخيص للحصول على الاعراض فقط**  **الحصول على عمود التشخيص لاستخدامه كهدف التصنيف**  **بناء شجرة القرار** | from sklearn.tree import DecisionTreeClassifier  def diagnose\_v4(train\_dataset:pd.DataFrame):  # create a DecisionTreeClassifier  model=DecisionTreeClassifier(random\_state=1)  # drop the diagnosis column to get only the symptoms  train\_patient\_symptoms=train\_dataset.drop(columns=['diagnosis'])  # get the diagnosis column, to be used as the classification target  train\_diagnoses=train\_dataset['diagnosis']  # build a decision tree  model.fit(train\_patient\_symptoms, train\_diagnoses)  # return the trained model  return model |
| **عملية التدريب لا تكون على جميع البيانات سيتم تقسيمها لبيانات تدريب 70 % و اختبار 30 %**  **بطريقة عشوائية ( 1 )**  **طباعة النتائج حيث ان العمود الأول عدد المرضى و الثاني يمثل عدد أعمدة البيانات )** | from sklearn.model\_selection import train\_test\_split  # use the function to split the data, get 30% for testing and 70% for training.  train\_data, test\_data = train\_test\_split(medical\_dataset, test\_size=0.3,  random\_state=1)  #print the shapes (rows x columns) of the two datasets  print(train\_data.shape)  print(test\_data.shape) |
| **التمثيل البياني لشجرة القرار** | from sklearn.tree import plot\_tree  import matplotlib.pyplot as plt  my\_tree=diagnose\_v4(train\_data) # train a model  print(my\_tree.classes\_) # print the possible target labels (diagnoses)  plt.figure(figsize=(12,6)) # size of the visualization, in inches  # plot the tree  plot\_tree(my\_tree,  max\_depth=2,  fontsize=10,  feature\_names=medical\_dataset.columns[:-1]  ) |
| **نسبة دقة النموذج بعد عمليات الاختبار**  **تحققت دقة النموذج بنسبة 81.6 و هذا يعني انه من بين 600 حالة اختبار شخصت شجرة القرار 4الة بشكل صحيح** | # functions used to evaluate a classifier  from sklearn.metrics import accuracy\_score,confusion\_matrix  # drop the diagnosis column to get only the symptoms  test\_patient\_symptoms=test\_data.drop(columns=['diagnosis'])  # get the diagnosis column, to be used as the classification target  test\_diagnoses=test\_data['diagnosis']  # guess the most likely diagnoses  pred=my\_tree.predict(test\_patient\_symptoms)  # print the achieved accuracy score  accuracy\_score(test\_diagnoses,pred) |
| **طباعة مصفوفة الدقة للنموذج لاستعراض الأمثلة التي صنفت**  **البيانات على الخط القطري للمصفوفة تمثل التشخيص الصحيح المتوقع**  **و التي على خارج الخط القطري للمصفوفة تمثل أخطاء النموذج** | confusion\_matrix(test\_diagnoses,pred) |

**2-9 : أنشاء متاهة 3\*3: ص 109**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **استدعاء مكتبة البيانات الرقمية** | import numpy as np |
| **انشاء متاهة 3\*3** | small\_maze=np.zeros((3,3)) |
| **تحديد احداثيات العوائق** | blocks=[(1, 1), (2, 1), (2, 2)] |
| **تقدير رقم العائق ب 1 ( و يمكن تقدير أي رقم اخر )** | for block in blocks:  small\_maze[block]=1 |
| **عرض احداثيات المتاهة حيث ان القيمة صفر تعني فارغ و القيمة 1 تعني ان هناك عائق** | small\_maze |
| **الكود البرمجي بالكامل** | |
| import numpy as np  # create a numeric 3 x 3 matrix full of zeros.  small\_maze=np.zeros((3,3))  # coordinates of the cells occupied by blocks  blocks=[(1, 1), (2, 1), (2, 2)]  for block in blocks:  # set the value of block-occupied cells to be equal to 1  small\_maze[block]=1  small\_maze | |

**2-10 : أنشاء متاهة كبيرة و معقدة: ص 110**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **انشاء متاهة بحجم و عوائق**  **انشاء متاهة بأي عدد من الصفوف و الأعمدة و تخزينها في متغير**  **توليد أي عدد من الخلايا عشوائية التي تشكل العوائق** | import random  random\_maze=np.zeros((10,10))  # coordinates of 30 random cells occupied by blocks  blocks=[(random.randint(0,9),random.randint(0,9)) for i in range(30)]  for block in blocks:  random\_maze[block]=1 |
| **تمثيل المتاهة بالوان و تنسيقات اختيارية** | import matplotlib.pyplot as plt # library used for visualization  def plot\_maze(maze):  ax = plt.gca() # create a new figure  ax.invert\_yaxis() # invert the y-axis to match the matrix  ax.axis('off') # hide the axis labels  ax.set\_aspect('equal') # make sure the cells are rectangular  plt.pcolormesh(maze, edgecolors='black', linewidth=2,cmap='Accent')  plt.show()  plot\_maze(random\_maze) |
| **الكود البرمجي بالكامل** | |
| import random  random\_maze=np.zeros((10,10))  # coordinates of 30 random cells occupied by blocks  blocks=[(random.randint(0,9),random.randint(0,9)) for i in range(30)]  for block in blocks:  random\_maze[block]=1  random\_maze[block]=1  import matplotlib.pyplot as plt # library used for visualization  def plot\_maze(maze):  ax = plt.gca() # create a new figure  ax.invert\_yaxis() # invert the y-axis to match the matrix  ax.axis('off') # hide the axis labels  ax.set\_aspect('equal') # make sure the cells are rectangular  plt.pcolormesh(maze, edgecolors='black', linewidth=2,cmap='Accent')  plt.show()  plot\_maze(random\_maze) | |

**2-11 : تحديد الخلاياالفارغة المجاورة لخلية محددة: ص 111**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الأمر البرمجي** |
| **انشاء دالة لاستدعاء قائمة تحتوي على كل الخلايا الفارغة و المجاورة لخلية محددة في أي متاهة** | def get\_accessible\_neighbors(maze:np.ndarray, cell:tuple):  # list of accessible neighbors, initialized to empty  neighbors=[]  x,y=cell  # for each adjacent cell position  for i,j in [(x-1,y-1),(x-1,y),(x-1,y+1),(x,y-1),(x,y+1),(x+1,y-1),(x+1,y),  (x+1,y+1)]:  # if the adjacent cell is within the bounds of the grid and is not occupied by a block  if i>=0 and j>=0 and i<len(maze) and j<len(maze[0]) and maze[(i,j)]==0:  neighbors.append(((i,j),1))  return neighbors |
| **اختبار اظهار الخلايا المجاورة الفارغة لخلية ما** | get\_accessible\_neighbors(small\_maze, (0,0)) |
| **الكود البرمجي بالكامل** | |
| def get\_accessible\_neighbors(maze:np.ndarray, cell:tuple):  # list of accessible neighbors, initialized to empty  neighbors=[]  x,y=cell  # for each adjacent cell position  for i,j in [(x-1,y-1),(x-1,y),(x-1,y+1),(x,y-1),(x,y+1),(x+1,y-1),(x+1,y), (x+1,y+1)]:    # if the adjacent cell is within the bounds of the grid and is not occupied by a block  if i>=0 and j>=0 and i<len(maze) and j<len(maze[0]) and maze[(i,j)]==0:    neighbors.append(((i,j),1))  return neighbors  get\_accessible\_neighbors(small\_maze, (0,0)) | |

**2-12 :استخدام خوارزمية البحث بالاتساع في حل المتاهة: ص 112 الى 114ص**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الكود البرمجي بالكامل** |
| **استخدام دالة reconstruct\_shortest\_path لاعادة و بناء المسار الأقصر و استدعائه لتتبع المسار بصورة عكسية من خلية الهدف إلى خلية البداية** | |
| def reconstruct\_shortest\_path(parent:dict, start\_cell:tuple, target\_cell:tuple):  shortest\_path = []  my\_parent=target\_cell # start with the target\_cell  # keep going from parent to parent until the search cell has been reached  while my\_parent!=start\_cell:  shortest\_path.append(my\_parent) # append the parent  my\_parent=parent[my\_parent] # get the parent of the current parent  shortest\_path.append(start\_cell) # append the start cell to complete the path  shortest\_path.reverse() # reverse the shortest path  return shortest\_path | |
| **get\_accessible\_neighbors( ) و reconstruct\_shortest\_path بمساعدة الدالتين bfs\_maze\_solver تنفيذ دالة** | |
| from typing import Callable # used to call a function as an argument of another function  def bfs\_maze\_solver(start\_cell:tuple,  target\_cell:tuple,  maze:np.ndarray,  get\_neighbors: Callable,  verbose:bool=False): # by default, suppresses descriptive output text  cell\_visits=0 # keeps track of the number of cells that were visited during the search  visited = set() # keeps track of the cells that have already been visited  to\_expand = [] # keeps track of the cells that have to be expanded  # add the start cell to the two lists  visited.add(start\_cell)  to\_expand.append(start\_cell)  # remembers the shortest distance from the start cell to each other cell  shortest\_distance = {}  # the shortest distance from the start cell to itself, zero  shortest\_distance[start\_cell] = 0  # remembers the direct parent of each cell on the shortest path from the start\_cell to the cell  parent = {}  #the parent of the start cell is itself  parent[start\_cell] = start\_cell  while len(to\_expand)>0:  next\_cell = to\_expand.pop(0) # get the next cell and remove it from the expansion list  if verbose:  print('\nExpanding cell', next\_cell)  # for each neighbor of this cell  for neighbor,cost in get\_neighbors(maze, next\_cell):  if verbose:  print('\tVisiting neighbor cell',neighbor)  cell\_visits+=1  if neighbor not in visited: # if this is the first time this neighbor is visited  visited.add(neighbor)  to\_expand.append(neighbor)  parent[neighbor]= next\_cell  shortest\_distance[neighbor]=shortest\_distance[next\_cell]+cost  # target reached  if neighbor==target\_cell:  # get the shortest path to the target cell, reconstructed in reverse.  shortest\_path = reconstruct\_shortest\_path(parent, start\_cell, target\_cell)  return shortest\_path, shortest\_distance[target\_cell],cell\_visits  else: # this neighbor has been visited before  # if the current shortest distance to the neighbor is longer than the shortest  # distance to next\_cell plus the cost of transitioning from next\_cell to this neighbor  if shortest\_distance[neighbor]>shortest\_distance[next\_cell]+cost:  parent[neighbor]=next\_cell  shortest\_distance[neighbor]=shortest\_distance[next\_cell]+cost  # search complete but the target was never reached, no path exists  return None,None,None | |
| **لحل متاهة 3\*3 bfs\_maze\_solver استخدام دالة** | |
| start\_cell=(2,0) # start cell, marked by a star in the 3x3 maze  target\_cell=(1,2) # target cell, marked by an "X" in the 3x3 maze  solution, distance, cell\_visits=bfs\_maze\_solver(start\_cell,  target\_cell,  small\_maze,  get\_accessible\_neighbors,  verbose=True)  print('\nShortest Path:', solution)  print('Cells on the Shortest Path:', len(solution))  print('Shortest Path Distance:', distance)  print('Number of cell visits:', cell\_visits) | |

**2-13 :استخدام خوارزمية البحث بأولوية الأفضل : ص 117 الى ص 120**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الكود البرمجي بالكامل** |
|  | |
| تحديد أفضل خلية مرشحة من بين مجموعة من الخلايا المرشحة استنادًا إلى مسافة معينة وتقدير (heuristic) ثابت  **تعريف دالتين**  constant\_heuristi تأخذ مدخلين على هيئة أزواج وترجع قيمة ثابته تساوي 1( كل قيمة تقدير لاي خلية مرشحة هي نفسها بعض النظر عن الخلية المرشحة أو الخلية الهدف  get\_best\_candidateتأخذ ثلاث معاملات   * + - expansion\_candidates: مجموعة تحتوي على الخلايا المرشحة.     - shortest\_distance: قاموس يربط الخلايا بأقصر المسافات إليها.   + heuristic: دالة تُستخدم لحساب التقدير (heuristic) بين خليتين، ولكنها لا تستخدم في هذا الكود. بدلاً من ذلك، يُستخدم دائمًا قيمة تقدير ثابتة تساوي 1   و نستخدم حلقة for لفحص الخلايا المرشحة في المجموعة و تحديد الأفضل باستخدام طابور الأولوية | |
| def constant\_heuristic(candidate\_cell:tuple, target\_cell:tuple):  return 1  #  def get\_best\_candidate(expansion\_candidates:set,  shortest\_distance:dict,  heuristic:Callable): # قيمة ناتجة عن تقدير المسافة بين الخلية الحالية والمرشحة    winner = None # متغير اخزن بداخله القيم الفائزة    best\_estimate= sys.maxsize  for candidate in expansion\_candidates:  # تطبيق الدالة الاستدلالية واحتسابها في الخلايا المرشحة  candidate\_estimate=shortest\_distance[candidate]+heuristic(candidate,target\_cell)  if candidate\_estimate < best\_estimate:  winner = candidate  best\_estimate=candidate\_estimate  return winner | |
| الكود السابق يقوم بحل مشكلة المتاهة باستخدام خوارزمية A\* للبحث بين خلية البداية وخلية الهدف في لوحة المتاه) | |
| import sys  def astar\_maze\_solver(start\_cell:tuple,  target\_cell:tuple,  maze:np.ndarray,  get\_neighbors: Callable,  heuristic:Callable,  verbose:bool=False):    cell\_visits=0    shortest\_distance = {}  shortest\_distance[start\_cell] = 0  parent = {}  parent[start\_cell] = start\_cell  expansion\_candidates = set([start\_cell])  fully\_expanded = set()  # while there are still cells to be expanded  while len(expansion\_candidates) > 0:    best\_cell = get\_best\_candidate(expansion\_candidates,shortest\_distance,heuristic)  if best\_cell == None: break    if verbose: print('\nExpanding cell', best\_cell)    # if the target cell has been reached, reconstruct the shortest path and exit  if best\_cell == target\_cell:    shortest\_path=reconstruct\_shortest\_path(parent,start\_cell,target\_cell)    return shortest\_path, shortest\_distance[target\_cell],cell\_visits    for neighbor,cost in get\_neighbors(maze, best\_cell):    if verbose: print('\nVisiting neighbor cell', neighbor)    cell\_visits+=1    # first time this neighbor is reached  if neighbor not in expansion\_candidates and neighbor not in fully\_expanded:    expansion\_candidates.add(neighbor)  parent[neighbor] = best\_cell # mark the best\_cell as this neighbor's parent  # update the shortest distance for this neighbor  shortest\_distance[neighbor] = shortest\_distance[best\_cell] + cost  # this neighbor has been visited before, but a better (shorter) path to it has just been found  elif shortest\_distance[neighbor] > shortest\_distance[best\_cell] + cost:    shortest\_distance[neighbor] = shortest\_distance[best\_cell] + cost  parent[neighbor] = best\_cell  if neighbor in fully\_expanded:  fully\_expanded.remove(neighbor)  expansion\_candidates.add(neighbor)  # all neighbors of best\_cell have been inspected at this point  expansion\_candidates.remove(best\_cell)  fully\_expanded.add(best\_cell)  return None, None, None # no solution was found | |
| استخدام خوارزمية A\* لحل مشكلة المتاهة بين الخلية البداية (2,0) والخلية المستهدفة (1,2) في لوحة المتاهة small\_maze. بعد تنفيذ الكود  سيتم طباعة المخرجات التالية:   * Shortest Path: المسار الأقصر بين الخلية البداية والخلية المستهدفة. إذا تم العثور على حلاً، ستظهر هنا قائمة الخلايا التي تشكل المسار الأقصر. * Cells on the Shortest Path: عدد الخلايا على المسار الأقصر بين الخلية البداية والخلية المستهدفة. * Shortest Path Distance: المسافة على المسار الأقصر بين الخلية البداية والخلية المستهدفة. * Number of cell visits: عدد الزيارات التي تمت للخلايا خلال عملية البحث. | |
| start\_cell=(2,0) # start cell, marked by a star in the 3x3 maze  target\_cell=(1,2) # target cell, marked by an "X" in the 3x3 maze  solution, distance, cell\_visits=bfs\_maze\_solver(start\_cell,  target\_cell,  small\_maze,  get\_accessible\_neighbors,  verbose=True)  print('\nShortest Path:', solution)  print('Cells on the Shortest Path:', len(solution))  print('Shortest Path Distance:', distance)  print('Number of cell visits:', cell\_visits) | |

**التدريبات العملية لمقرر الذكاء الاصطناعي 1-1**

**تدريبات الوحدة الثالثة**

**معالجة اللغات الطبيعية**

**الدرس الأول**

**التعلم الموجه**

**4 تدريبات ( متصلة )**

**3-1 :التنبؤ بالانطباع العام عن الفيلم : ص 135 الى ص 142**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الكود البرمجي بالكامل** |
| **استيراد مكتبة بانداس الخاصة بالتعامل مع جداول البيانات** | |
| # install the pandas library, if it is missing.  !pip install pandas  import pandas as pd | |
| **تحميل مجموعة بيانات التدريب و الاختبار** | |
| imdb\_train\_reviews=pd.read\_csv('imdb\_train.csv')  imdb\_test\_reviews=pd.read\_csv('imdb\_test.csv')  imdb\_train\_reviews | |
| **اسناد أعمدة النص و القيم إلى متغيرات مستقلة في أمثلة التدريب و الاختبار ( استخراج النص من عمود النص للتدريب و الاختبار و استخراج التسميات من عمود التسمية لكل من عمود التدريب و الاختبار هل هو تعليق سلبي أم ايجابي)**  **y و القيم المستهدفة x حيث أن البيانات المدخلة للتنبؤ** | |
| # extract the text from the 'text' column for both training and testing.  X\_train\_text=imdb\_train\_reviews['text']  X\_test\_text=imdb\_test\_reviews['text']  # extract the labels from the 'label' column for both training and testing.  Y\_train=imdb\_train\_reviews['label']  Y\_test=imdb\_test\_reviews['label']  X\_train\_text # training data in text forma | |
| **تجهيز البيانات ( بيانات التدريب )**  **المتوفرة في مكتبة سكليرن لتحويل البيانات إلى متجهات** CountVectorizer **استخدام أداة**  **تحويل البيانات النصية إلى أرقام حقيقة لفهمها من قبل خوارزمية التعلم** | |
| from sklearn.feature\_extraction.text import CountVectorizer  # the min\_df parameter is used to ignore terms that appear in less than 10 reviews.  vectorizer\_v1 = CountVectorizer(min\_df=10)  vectorizer\_v1.fit(X\_train\_text) # fit the vectorizer on the training data.  # use the fitted vectorizer to vectorize the data.  X\_train\_v1 = vectorizer\_v1.transform(X\_train\_text)  X\_train\_v1 | |
| **تحويل القيم السابقة الى مصفوفة**  **لوضع كل كلمة في عمود و يوضح عدد مرات ظهورها Dense استخدام التنسيق الكثيف** | |
| X\_train\_v1\_dense=pd.DataFrame(X\_train\_v1.toarray(),  columns=vectorizer\_v1.get\_feature\_names\_out())  X\_train\_v1\_dense | |
| **التي تحدد حجم الكائنات في البايثون بالبايت getsizeof( ) استخدام الدالة**  **لتوفير مساحة الذاكرة و ذلك بالاحتفاظ بالمدخلات الغير صفرية في كل عمود ( المصفوفة المتباعدة )** | |
| from sys import getsizeof  print('\nMegaBytes of RAM memory used by the raw text format:',  getsizeof(X\_train\_text)/1000000)  print('\nMegaBytes of RAM memory used by the dense matrix format:',  getsizeof(X\_train\_v1\_dense)/1000000)  print('\nMegaBytes of RAM memory used by the sparse format:',  getsizeof(X\_train\_v1)/1000000) | |
| **حذف المصفوفة الكثيفة لتوفير حجم الذاكرة** | |
| # delete the dense matrix.  del X\_train\_v1\_dense | |

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الكود البرمجي بالكامل** |
| **مرحلة التنبؤ**  **بناء خط أنابيب التنبؤ الأول و استخدام مصنف بايز الساذج المدرب مسبقا على التعلم بالمتجهات**  **احتمالات الكلمات أو العبارات المحددة الواردة في النص للتنبؤ باحتمال انتمائه إلى تصنيف محدد** | |
| from sklearn.naive\_bayes import MultinomialNB  model\_v1=MultinomialNB() # a Naive Bayes Classifier  model\_v1.fit(X\_train\_v1, Y\_train) # fit the classifier on the vectorized training data.  from sklearn.pipeline import make\_pipeline  # create a prediction pipeline: first vectorize using vectorizer\_v1, then use model\_v1 to predict.  prediction\_pipeline\_v1 = make\_pipeline(vectorizer\_v1, model\_v1) | |
| **اختبار مقطع برمجي** | |
| prediction\_pipeline\_v1.predict(['One of the best movies of the year. Excellent cast and very interesting plot.',  'I was very disappointed with his film. I lost all interest after 30 minutes' ]) | |
| **تنبؤ خط الأنابيب بالقيمة الإيجابية و السلبية للتقييمين بشكل صحيح** | |
| prediction\_pipeline\_v1.predict\_proba(['One of the best movies of the year. Excellent cast and very interesting plot.',  'I was very disappointed with his film. I lost all interest after 30 minutes' ]) | |
| **IMDb اختبار دقة خط الأنابيب الجديد في تصنيف التقييمات في مجموعة بيانات** **اختبار** | |
| predictions\_v1 = prediction\_pipeline\_v1.predict(X\_test\_text) # vectorize the text data, then predict.  predictions\_v1 | |
| **تحليل و تصوير نتائج خطوط أنابيب التصنيف و تمثيل مصفوفة الدقة ( تقريب دقيق لدرجة التنبؤ )** | |
| from sklearn.metrics import accuracy\_score  accuracy\_score(Y\_test, predictions\_v1) # get the achieved accuracy | |
| **scikit-Piot( ) استدعاء مكتبة** | |
| !pip install scikit-plot; | |
| **تمثيل مصفوفة الدقة** | |
| import scikitplot; # import the library  class\_names=['neg','pos'] # pick intuitive names for the 0 and 1 labels.  # plot the confusion matrix.  scikitplot.metrics.plot\_confusion\_matrix(  [class\_names[i] for i in Y\_test],  [class\_names[i] for i in predictions\_v1],  title="Confusion Matrix", # title to use  cmap="Purples", # color palette to use  figsize=(5,5) # figure size  ); | |

**3-2 :النموذج المحايد المحلي القابل للتفسير و الشرح: ص 142 الى ص 145**

|  |  |
| --- | --- |
| **شرح الأمر البرمجي** | **الكود البرمجي بالكامل** |
| **استيراد مكتبة النموذج المحايد lime الخاصة بتفسير التنبؤات التي قامت بها نماذج الصندوق الأسود** | |
| !pip install lime | |
| **تطبيق النموذج على عبارة سلبية و التأكد من صحة و دقة التنبؤ** | |
| from lime.lime\_text import LimeTextExplainer  # create a local explainer for explaining individual predictions  explainer\_v1 = LimeTextExplainer(class\_names=class\_names)  # an example of an obviously negative review  easy\_example='This movie was horrible. The actors were terrible and the plot was very boring.'  # use the prediction pipeline to get the prediction probabilities for this example  print(prediction\_pipeline\_v1.predict\_proba([easy\_example])) | |
| **اظهار درجة لكل كلمة تمثل معامل في نموذج الانحدار الخطي البسيط لتقديم التفسير ( يمكن زيادة او انقاص الخصائص المعروضة )** | |
| # explain the prediction for this example.  exp = explainer\_v1.explain\_instance(easy\_example.lower(),prediction\_pipeline\_v1.predict\_proba, num\_features=10)  # print the words with the strongest influence on the prediction.  exp.as\_list() | |
| **التصور المرئي للدرجات السابقة الخاصة بالكلمات** | |
| # visualize the impact of the most influential words.  fig = exp.as\_pyplot\_figure() | |
| **استخدام النموذج المحايد لتحليل عبارة سلبية** | |
| exp.show\_in\_notebook() | |
| **IMDB استخدام النموذج المحايد لعبارة معقدة مأخوذة من مجموعة اختبار** | |
| # an example of a positive review that is mis-classified as negative by prediction\_pipeline\_v1  mistake\_example= X\_test\_text[4600]  mistake\_example | |
| **تقييم النموذج المحايد للعبارة السابقة** | |
| # get the correct labels of this example.  print('Correct Label:', class\_names[Y\_test[4600]])  # get the prediction probabilities for this example  print('Prediction Probabilities for neg, pos:',prediction\_pipeline\_v1.predict\_proba([mistake\_example])) | |
| **استخدام المفسر لتوضيح السبب وراء اتخاذ نموذج التنبؤ مثل هذا القرار الخاطئ** | |
| # explain the prediction for this example.  exp = explainer\_v1.explain\_instance(mistake\_example, prediction\_pipeline\_v1.predict\_proba, num\_features=10)  # visualize the explanation.  fig = exp.as\_pyplot\_figure() | |

**3-3 :تحسين البرمجة الاتجاهية للنصوص: ص 146 الى ص 149**

|  |  |  |
| --- | --- | --- |
| **شرح الأمر البرمجي** | **الكود البرمجي بالكامل** | |
| **استيراد مكتبة nltk الخاصة مهام معالجة اللغات الطبيعية المتنوعة** | | |
| !pip install nltk | | |
| **استيراد مكتبة Gensim الخاصة مهام معالجة اللغات الطبيعية المتنوعة** | | |
| !pip install gensim | | |
| **الخاصة بالبحث في النصوص و معالجتها باستخدام التعبيرات النمطية re استدعاء مكتبة** | | |
| import nltk # import nltk  nltk.download('punkt') # install nltk's tokenization tool, used to split a text into sentences.  import re # import re  from gensim.models.phrases import Phrases, ENGLISH\_CONNECTOR\_WORDS # import tools from the gensim library | | |
| **Nitk التابعة لمكتبة sent\_tokenize () استخدام دالة**  **لتقسيم المستند على قائمة من الجمل المقسمة** | | |
| # convert a given doc to a list of tokenized sentences.  def tokenize\_doc(doc:str):  return [re.findall(r'\b\w+\b', sent.lower()) for sent in nltk.sent\_tokenize(doc)] | | |
| **مثال لتقسيم جملة لكلمات** | | |
| raw\_text='The movie was too long. I fell asleep after the first 2 hours.'  tokenized\_sentences=tokenize\_doc(raw\_text)  tokenized\_sentences | | |
| **أربع متغيرات phrases () استقبال الدالة** | | |
| sentences=[] # list of all the tokenized sentences across all the docs in this dataset  for doc in X\_train\_text: # for each doc in this dataset  sentences+=tokenize\_doc(doc) # get the list of tokenized sentences in this doc  # build a phrase model on the given data  imdb\_phrase\_model = Phrases(sentences,  connector\_words=ENGLISH\_CONNECTOR\_WORDS,  scoring='npmi',  threshold=0.25).freeze() | | |
| **على العبارة phrases () تطبيق الدالة** | | |
| imdb\_phrase\_model[tokenized\_sentences[0]] | | |
| **على العبارة phrases () تطبيق الدالة** | | |
| imdb\_phrase\_model[tokenized\_sentences[1]] | | |
| **تفسير العبارات في وثيقة معطاه** | | |
| def annotate\_phrases(doc:str, phrase\_model):  sentences=tokenize\_doc(doc)# split the document into tokenized sentences.  tokens=[] # list of all the words and phrases found in the doc  for sentence in sentences: # for each sentence  # use the phrase model to get tokens and append them to the list.  tokens+=phrase\_model[sentence]  return ' '.join(tokens) # join all the tokens together to create a new annotated document | | |
| **لتفسير كل من تقنيات التدريب و الاختبار من مجموعة بيانات annotate\_phrases () استخدام الدالة** | | |
| # annotate all the test and train reviews.  X\_train\_text\_annotated=[annotate\_phrases(doc,imdb\_phrase\_model) for doc in X\_train\_text]  X\_test\_text\_annotated=[annotate\_phrases(text,imdb\_phrase\_model) for text in X\_test\_text]  # visualize the explanation. | | |
| **IMDBمثال لمستند من بيانات التدريب من موقع** | | |
| # an example of an annotated document from the imdb training data  X\_train\_text\_annotated[0] | | |
| **3-4 :تحسين البرمجة الاتجاهية للنصوص: ص 149 الى ص 151** | | |
| **شرح الأمر البرمجي** | | **الكود البرمجي بالكامل** |
| **تدريب نموذج TF-IDF (Term Frequency-Inverse Document Frequency) باستخدام مجموعة البيانات التدريبية IMDb واستخدامه لتحويل النصوص إلى تمثيل TF-IDF.** | | |
| from sklearn.feature\_extraction.text import TfidfVectorizer  # Train a TF-IDF model with the IMDb training dataset  vectorizer\_tf = TfidfVectorizer(min\_df=10)  vectorizer\_tf.fit(X\_train\_text\_annotated)  X\_train\_tf = vectorizer\_tf.transform(X\_train\_text\_annotated) | | |
| **تدريب نموذج جديد باستخدام التمثيل TF-IDF للبيانات النصية وتقنية Naive Bayes من نوع MultinomialNB. وبعد ذلك، يتم تنفيذ التنبؤات على مجموعة البيانات الاختبارية وحساب الدقة** | | |
| # train a new Naive Bayes Classifier on the newly vectorized data.  model\_tf =MultinomialNB()  model\_tf.fit(X\_train\_tf, Y\_train)  # create a new prediction pipeline.  prediction\_pipeline\_tf = make\_pipeline(vectorizer\_tf, model\_tf)  # get predictions using the new pipeline.  predictions\_tf = prediction\_pipeline\_tf.predict(X\_test\_text\_annotated)  # print the achieved accuracy.  accuracy\_score(Y\_test, predictions\_tf) | | |
| **استخدام مثال من مجموعة البيانات الاختبارية الذي قد يكون قد أحدث ارتباكًا للنموذج السابق. الهدف هو فحص التصنيف الصحيح لهذا المثال والتنبؤ بالتصنيف الذي قام به النموذج الجديد** | | |
| # get the review example that confused the previous algorithm  mistake\_example\_annotated=X\_test\_text\_annotated[4600]  print('\nReview:',mistake\_example\_annotated)  # get the correct labels of this example.  print('\nCorrect Label:', class\_names[Y\_test[4600]])  # get the prediction probabilities for this example.  print('\nPrediction Probabilities for neg, pos:',prediction\_pipeline\_tf.predict\_proba([mistake\_example\_annotated])) | | |
| إنشاء مفسر (explainer) باستخدام مكتبة LimeTextExplainer ومن ثم تم استخدامه لشرح التنبؤ الذي تم بواسطة النموذج الجديد للمثال الذي تم استخدامه في الخطوات السابقة. الهدف هو توضيح سبب تصنيف النموذج لهذا المثال مع التركيز على السمات الرئيسية المسهمة في القرار | | |
| # create an explainer.  explainer\_tf = LimeTextExplainer(class\_names=class\_names)  # explain the prediction of the second pipeline for this example.  exp = explainer\_tf.explain\_instance(mistake\_example\_annotated, prediction\_pipeline\_tf.predict\_proba, num\_features=10)  # visualize the results.  fig = exp.as\_pyplot\_figure() | | |
|  | | |