

Assignment 2

This assignment is based on the sentiment assignment of CS221 at Stanford University

Image credit: pentagram



這份作業將訓練同學運用ML來打造出一個自動 爛番茄影評AI評價系統


本次作業將接續作業一的重點觀念往上搭建，因此歡迎各位同學有任何問題就找助教/講師/同學們討論在 [Python / ML / 微分](#) 尚未弄懂的概念

作業估計時數 - 15 小時

如果作業卡關歡迎各位到社團提問/運用 TA 時間，也非常鼓勵同學們互相討論作業之概念，**但請勿把 code 給任何人看**（也不要將程式碼貼在社團裡）分享您的 code 會剝奪其他學生獨立思考的機會，也會使防抄襲軟體認定有抄襲嫌疑


作業檔案下載

作業開始前，我們先拿 Jerry 最喜歡的電影 The Dark Knight 在爛番茄上的兩個影評來當作開場（courtesy of [Rotten Tomatoes](#)）




Heath Ledger gives one of the best performances of the last twenty years. When you watch his take on the Joker, it's one of complete surprise and mystery. It's frightening with how chaotic and mischievous he makes him.

November 21, 2019 | Rating: 5/5 | [Full Review...](#)




Paul McGuire Grimes
KSTP-TV (St. Paul, MN)



It seems almost cruel to take beloved child archetypes and turn them into projections for adult angst.

January 28, 2020 | [Full Review...](#)



Felicia Feaster
Charleston City Paper

爛番茄影評網將左邊的影評評價為新鮮蕃茄 🍅 (好評!) 右邊的影評則評價為爛番茄 🍄 (差評!) 在疫情爆發前，每週有 20-30 部電影在全美上映；而全美又有不少報章雜誌影評人撰寫評價。這時 AI 就會是一個好幫手，幫爛番茄網站對每個影評自動評價了 💡

本次的作業我們將分成六個 milestones 讓同學們一步一步完成「人工智慧自動評價影評系統」

Milestone 1 - Mini Reviews

假設我們有四則被標注好評 (1) 與壞評 (0) 的影評

- ① (0) pretty bad
- ② (1) good plot
- ③ (0) not good
- ④ (1) pretty scenery

從作業一我們知道：對一段文字 x 我們可以用一個 feature extractor 萃取出 feature vector (以下簡稱 $\Phi(x)$) 再做 machine learning

今天我們想試著使用「每個詞彙 (token) 在文字中出現的次數」來當我們的 $\Phi(x)$

首先，我們將每一段影評 x 丟入 **feature extractor $\Phi(x)$** ，得到 x 裡每個字出現次數的矩陣：["pretty", "good", "bad", "plot", "not", "scenery"]

舉例來說，若我們將 **(0) pretty bad** 這則影評 x 丟入 $\Phi(x)$ 就會得到 **[1, 0, 1, 0, 0, 0]** 而 $y = 0$ 將會被帶入 **loss function 微分式** 運算

回憶一下，在做 **classification problem** 時要使用的 loss function 為 **logistic regression** 公式如下 (σ 為 sigmoid function 的縮寫)：

$$\text{Loss}(y, h) = - (y \log h + (1-y) \log(1-h))$$

$$h = \sigma(w \cdot \Phi(x))$$

而找到最佳 w 的方法為 **Gradient Descent** 公式如下：

$$w = w - \alpha \frac{\partial \text{Loss}(y, h)}{\partial w}$$

$$\frac{\partial \text{Loss}(y, h)}{\partial w} = (h - y) \Phi(x)$$

請將 Milestone 1 開頭的四則影評依序做 **Gradient Descent**。請問：最後得到的 \mathbf{w} 裡 ("pretty", "good", "bad", "plot", "not", "scenery") 對應之權重數值為何？請注意： \mathbf{w} 的起始為 $[0, 0, 0, 0, 0, 0]$ 且我們假設 Learning Rate α 為 **0.5**

第一則影評被丟入後，您應該會看到 \mathbf{w} 被更新為
 $[-0.25, 0, -0.25, 0, 0, 0]$

第二則影評被丟入後，您應該會看到 \mathbf{w} 被更新為
 $[-0.25, 0.25, -0.25, 0.25, 0, 0]$

第三則影評被丟入後，您應該會看到 \mathbf{w} 被更新為
 $[-0.25, -0.031088, -0.25, 0.25, -0.281088, 0]$

第四則影評被丟入後，您應該會看到 \mathbf{w} 被更新為
 $[0.031088, -0.031088, -0.25, 0.25, -0.281088, 0.281088]$

請將算式過程寫在紙上，並拍照上傳至名為 Milestone 1 之資料夾中

hint: 先對每則評論做 $\mathbf{w} \cdot \Phi(\mathbf{x})$ 再做 $\mathbf{h} = \sigma(\mathbf{w} \cdot \Phi(\mathbf{x}))$ 最後再做 $\mathbf{w} = \mathbf{w} - \alpha(\mathbf{h} - \mathbf{y})\Phi(\mathbf{x})$

Milestone 2 - Derivatives

請試著推導 $(\mathbf{h} - \mathbf{y})\Phi(\mathbf{x})$ 是怎麼得到的？首先，請推導下列算式之答案

$$\text{Loss}(\mathbf{y}, \mathbf{h}) = - (\mathbf{y} \log \mathbf{h} + (1 - \mathbf{y}) \log(1 - \mathbf{h}))$$

$$\frac{\partial \text{Loss}(\mathbf{y}, \mathbf{h})}{\partial \mathbf{h}} = ?$$

再來，請計算下列算式之答案

$$h = \sigma(k) = \frac{1}{1+e^{-k}}$$

$$\frac{\partial h}{\partial k} = \frac{\partial(\frac{1}{1+e^{-k}})}{\partial k} = \frac{\partial(\frac{e^k}{e^k+1})}{\partial k} = ?$$

緊接著計算下列算式之答案

$$k = w \cdot \Phi(x)$$

$$\frac{\partial k}{\partial w} = ?$$

最後，試著把您剛剛計算的三個結果相乘

$$\frac{\partial \text{Loss}(y, h)}{\partial w} = \left(\frac{\partial \text{Loss}(y, h)}{\partial h} \right) \left(\frac{\partial h}{\partial k} \right) \left(\frac{\partial k}{\partial w} \right)$$

答案是否為 $(h-y)\Phi(x)$ 呢？如果不是，可以翻閱上課筆記，看看當時 Jerry 是如何推導的 🧑🏫🏢 如果還是不會請不用擔心，歡迎跟 助教 / Jerry 約時間討論。

微分的運算在機器學習中真的扮演很重要的角色，再麻煩同學們花時間了解！

Milestone 3 - Sparse Vector

使用「每個詞彙 (token) 在一段文字中出現的次數」來當我們的 feature extractor 看似簡單明瞭，然而，這個概念卻會讓我們電腦的記憶體出現儲存上的困難...

假設我們今天要製作一個「字典單字矩陣」來記錄「一段文字裡的每個字彙在該矩陣出現的次數」。拿 $x = \text{"This is a really good movie"}$ 來舉例，同學會看到 $\Phi(x)$ 如下圖所示：

a	1
aa	0
.	:
.	0
.	:
good	1
.	:
.	0
is	1
.	:
.	0
movie	1
.	:
.	0
really	1
.	:
.	0
this	1
.	0
.	:
.	0
.	:
.	0

假設一部英文字典大約包含470,000個單字，則我們製作的矩陣 $\Phi(x)$ 卻只有 6 個 1、40幾萬個 0（這種矩陣我們又稱之為 **Sparse Vector**）。很明顯，這個方法實在是太浪費記憶體了... 因此，我們想用其他方法來取而代之💡

首先，我們先製作一個 Python dictionary 計算文字 x 裡每個 token 出現的次數。假設 $x1 = \text{"This is a really good movie"}$ 則 $\Phi(x1) = \{\text{'This':1, 'is':1, 'a':1, 'really':1, 'good':1, 'movie':1}\}$

假設今天我們又得到了一段影評 $x2 = \text{'It really is a bad movie'}$ ，則 $\Phi(x2) = \{\text{'It':1, 'really':1, 'is':1, 'a': 1, 'bad':1, 'movie':1}\}$

您的工作就是要用一個 Python dictionary 儲存每一則影評產出的 feature vector 與對應之權重。舉例來說，若 w 為一個起始值為空的 dict ($w = \{ \}$)。假設經過數學計算， $\Phi(x1)$ 得到 0.25 的權重，則 $w = \{\text{'This': 0.25, 'is': 0.25, 'a': 0.25, 'really': 0.25, 'good': 0.25, 'movie':0.25}\}$

再來假設經過數學計算， $\Phi(x2)$ 得到 -0.24 的權重，則 $w = \{\text{'This': 0.25, 'is': 0.01, 'a': 0.01, 'really': 0.01, 'good': 0.25, 'movie':0.01, 'It': -0.24, 'bad': -0.24}\}$

為了完成上述之運算，請先編輯 `submission.py` 裡名為

`def extractWordFeatures(x: str) -> FeatureVector:` 之函式。將一段文字 x 輸入，return 一個 `Dict[str, int]` 計算每個字彙 (token) 在 x 出現次數

```
#####  
# Milestone 3a: feature extraction  
  
def extractWordFeatures(x: str) -> FeatureVector:  
    """  
    Extract word features for a string x. Words are delimited by  
    whitespace characters only.  
    @param string x:  
    @return dict: feature vector representation of x.  
    Example: "I am what I am" --> {'I': 2, 'am': 2, 'what': 1}  
    """  
    # BEGIN YOUR CODE (our solution is 4 lines of code, but don't worry if you deviate from this)  
    raise Exception("Not implemented yet")  
    # END_YOUR_CODE
```

請注意：`raise Exception("Not implemented yet")` 是 Python 手動產生 error message 的方法（以前我們是使用 `pass` 來讓電腦忽略還沒寫程式的部分）因此，當同學要開始寫作業時，請直接將這行刪除即可

眼睛很尖的同學應該會發現，在 `extractWordFeatures` 上方註解有提到「our solution is 4 lines of code」不知道同學是否好奇該如何達到 🤖

請同學看一下 `submission.py` 上方有一行程式碼叫

```
from collections import defaultdict
```

`collections` 是一個很方便 Python 工具包（`defaultdict` 是其中的一個好用的「小幫手」）。`defaultdict` 的使用方法：當丟入一個 `function` 到它的括弧裡面，您製造出 dict 就會以該 `function` 當作 value 的初始值

舉例來說，假設我想以一個名為 dict 的 Python dict 儲存特斯拉每天的股價 (`Dict[str: list[float]]`)。使用 `d = defaultdict(list)`，我可以在 `d` 是空的時候 就使用 `d['Tesla'].append(421.26)` 因為每個 key 被加入 `d` 時都會直接得到一個 `[]`

請同學上網搜尋一下，若我們想要用 `defaultdict` 製造出一個以 0 當作 value 預設值的 dict，該丟入什麼 function 到 `defaultdict` 的括弧裡呢？

（請注意：上網搜尋資料時請務必用英文，不要用中文）

請執行 `grader.py` 並看看是否可以通過 3a-0 的測試 (通過者可以獲得 10/100)

再來，請上網搜尋 Python dictionary 裡面的一個名為 `get` 的 method. 請在閱讀完 documentation 後，使用 `get` 完成 `util.py` 檔案裡一個名為 `def increment(d1, scale, d2)` 的函式，讓 `d2` 的數值乘上 `scale` 後可以被加到 `d1` 這個 dict 之中 (https://www.w3schools.com/python/ref_dictionary_get.asp)

Hint: `d1.get(key, 0)` 應該會派上用場

執行 `grader.py` 是否可以通過 3b-0 的測試 (請注意: `d2` 的 value 不一定是 1)

最後，請完成 `util.py` 檔案裡名為 `def dotProduct(d1, d2):` 的函式，來讓兩個 `Dict[str, int]` 達到像兩個矩陣內積的效果。舉例來說，假設

`d1 = {'Movie': 0.1, 'is': 0.2, 'good': 0.25}`

`d2 = {'Movie': 1, 'is': 1, 'good': 1, ...}`

則 `dotProduct(d1, d2)` 會得到 $0.1*1+0.2*1+0.25*1 = 0.55$

思考一下：程式碼 if 裡面使用 recursion 的意義為何呢 🤔

請執行 `grader.py` 並看看是否可以通過 **3c-0** 的測試 (通過者總共可獲得 **30/100**)。若超過時間 (TimeOut)，思考一下該怎麼加速 `dotProduct`?

```
def dotProduct(d1, d2):
    """
    @param dict d1: a feature vector represented by a mapping from a feature (string) to a weight (float).
    @param dict d2: same as d1
    @return float: the dot product between d1 and d2
    """
    if len(d1) < len(d2):
        return dotProduct(d2, d1)
    else:
        # BEGIN_YOUR_CODE (our solution is 1 line of code, but don't worry if you deviate from this)
        raise Exception("Not implemented yet")
        # END_YOUR_CODE
```

Milestone 4 - Sentiment Classification

我們要把所有 milestones 寫好的東西拼湊成一個「人工智慧自動評價影評系統」了！請同學在撰寫 Milestone 4 時，**一定要用上**前面寫的 `def increment(d1, scale, d2)` 以及 `def dotProduct(d1, d2)`

請完成 `submission.py` 檔案裡名為 `def learnPredictor(trainExamples: List[Tuple[Any, int]], validationExamples: List[Tuple[Any, int]], featureExtractor: Callable[[str], FeatureVector], numEpochs: int, alpha: float) -> WeightVector` 的函式。該函式的 parameters 介紹如下

- **trainExamples** 是一個儲存許多 tuple 的 list. 回憶一下，tuple 這個資料結構特別之處在於它是唯一 **immutable** 的資料結構。因此，我們常常在 ML 的程式中以 **(data, label)** 的形式儲存 training data。

我們已經將名為 **polarity.train** 的文字檔處理好丟進 **trainExamples** 裡了，請同學們不用擔心。然而，若同學們打開 **polarity.train** 會發現，裡面的 label 用 **+1** 來代表好評、**-1** 代表負評。這個是**非常嚴重的問題**！為什麼呢？因為 **logistic regression** 的所有式子都是建構在好評為 **+1**、負評為 **0** 的基礎上推導出來的。那是不是代表我們的 Model 就不能用了呢？其實不然，只要大家記得在將資料輸入 Model 前，把 **-1** 轉成 **0** 就好了 😊 每一位資料科學家在建構模型前一定要研究「資料到底長怎樣？可以用什麼模型？」

- **validationExamples** 也是一個儲存許多 tuple 的 list. 與 **trainExamples** 不同之處在於，它的內容物為 **polarity.dev** 的文字檔 (label 依舊是 **+1** 與 **-1**)。在 ML 裡，我們常常會把資料分成 **training set** 與 **validation set**：前者會重複很多次的 **Gradient Descent** 來得到一個最佳 **weights**；後者則只會在我們得到 **weights** 後拿來測試看看是否可以讓沒見過的資料 (**unseen data**) 也產生很高的準確率。**請注意：您在 Milestone 5 才會用到 validationExamples!** 這邊只是先跟各位同學講解概念而已，不用擔心
- **featureExtractor** 是一個 **function**！也就是同學在上面寫好的 **extractWordFeatures**。在進階程式撰寫時很常將一個 function 當作 **argument** 丟入另一個 function (舉例來說像是滑鼠偵測、鍵盤偵測的程式)。這邊要請同學注意的是：丟一個 function 進入另一個 function 是不加括弧的！
- **numEpochs** 是一個 **int**，控制我們要對所有資料跑「幾輪」的 **gradient descent** (一輪的意思是「將所有在 **trainExamples** 裡的資料都跑過一次 **gradient descent**」)
- **alpha** 是一個 **float**，就是我們講的 **learning rate**, 或 **step size**，控制每次 **gradient descent** 更新的數值大小
- **WeightVector (Dict[str, float])** 記錄著每個字彙的權重，也就是我們上面一直提到的 **weights**

當您完成 `learnPredictor` 後，請執行 `grader.py` 的檔案檢查您的程式是不是一個成功的 Model? (**training error** 要小於 4% / **validation error** 要小於 30%)

最後再次提醒：數據資料的原檔 **label** 為 +1, -1，一定要記得轉成 +1, 0

請執行 `grader.py` 並看看是否可以通過 4a-0, 4a-1, 4a-2 的測試
(若目前每個測試都通過者，總共可以獲得 70/100)

這份作業除了可以成功判斷一段文字是好評還是差評，更酷的是，程式已經在您跑完 `grader.py` 的那一刻將您程式所有的預測都輸出到 SC201Assignment2 資料夾內的一個名為 **error-analysis** 檔案內 (感謝偉大的 Stanford CS221)! 每個 token 所代表的權重也都被放到另外一個名為 **weights** 的檔案中。

Check them out!

同學們有沒有辦法看出錯誤判斷的例子是因為什麼原因呢 🤔

最後想請問同學一個問題：若我們將 `grader.py` 裡面的 `test4a2()` 第五行的 **numEpochs** 改成 70/80/90/... 並重跑程式，請問您發現什麼有趣的現象？請特別留意 **train error** 與 **validation error** 在不同 numEpochs 下的變化。請將您的答案以文字敘述寫在 `grader.py` 上方的註解處

```
grader.py x
1  """
2  File: grader.py
3
4  Milestone 4 answer: TODO:
5
6  """
7
8
9  #!/usr/bin/python3
10
11 import graderUtil
12 import util
13 import time
14 from util import *
15
16 grader = graderUtil.Grader()
17 submission = grader.load('submission')
18 util = grader.load('util')
```

Milestone 5 - Finishing up

完成「人工智慧自動評價影評系統」後，再來進入測試階段

首先請完成 `submission.py` 檔案裡名為 `def generateDataset(numExamples: int, weights: WeightVector) -> List[Example]:` 的函式，從我們得到的字彙庫 `weights` 重組虛構的例子來測試

函式內的 parameters 介紹如下：

- `numExamples` 是一個 `int`，告訴我們到底要產出多少則虛構的影評 `x` 與其評價 `y`
- `weights` 是我們在 Milestone 4 跑完後得到的 `dict` (key 是每一個 token，value 是該 token 的權重)

我們提供同學們一個非常有趣的新寫法💡

```
56 #####
57 # Milestone 5a: generate test case
58
59 def generateDataset(numExamples: int, weights: WeightVector) -> List[Example]:
60     """
61     Return a set of examples (phi(x), y) randomly which are classified correctly by
62     |weights|.
63     """
64     random.seed(42)
65
66     # Return a single example (phi(x), y).
67     # phi(x) should be a dict whose keys are a subset of the keys in weights
68     # and values can be anything (randomize!) with a nonzero score for the given weight vector.
69     # y should be 1 or -1 as classified by the weight vector.
70
71     # Note that the weight vector can be arbitrary during testing.
72     def generateExample() -> Tuple[Dict[str, int], int]:
73         # BEGIN_YOUR_CODE (our solution is 4 lines of code, but don't worry if you deviate from this)
74         raise Exception("Not implemented yet")
75         # END_YOUR_CODE
76         return phi, y
77
78     return [generateExample() for _ in range(numExamples)]
79
```

同學看到這邊應該會覺得很驚訝：竟然有一個名為 `def generateExample()` 的函式在 `def generateDataset(numExamples: int, weights: WeightVector)` 區間內。這是 Python 語言獨有的寫法。好處是我們無須把在 `generateDataset()` 區間內的變數傳入 `generateExample()` 的括弧內就可以直接使用所有屬於 `generateDataset()` 的 local variables 了。


舉例來說，假設您在 `generateExample()` 區間內要使用 `generateDataset()` 其中的一個 param: `weights`，您可以直接使用！無須在 `generateExample()` 括弧內定義一個名為 `weights` 的 param 了

請同學完成 `def generateExample()` 回傳一個長度隨機 ($1 \sim \text{len}(\text{weights})$) 的 feature vector (以下簡稱 `phi`)：

key 為存在 `weights` 裡面的詞彙、value 為該詞彙出現的次數。然而，這個虛構的 `phi` 代表的是好評(+1)還是壞評(-1)呢？(這邊的壞評要用 `-1` 因為我們要跟輸入的資料一至)

要判斷好評、壞評就要拿我們的 `phi` 與 `weights` 內積看出來的數值 ≥ 0 還是 < 0 來決定

請執行 `grader.py` 並看看是否可以通過 `5a-0, 5a-1` 的測試



學會怎麼在 function 裡面寫 function 這個技能後，請同學們回到 `learnPredictor()` 處理一個困難的部分：函式註解提到 “**You should call `evaluatePredictor()` on both `trainExamples` and `validationExamples` ... after each epoch**” 這邊要請同學自己製造一個名為 `def predictor(x)` 的函式、丟入位於 `util.py` 檔案裡的 `evaluatePredictor()`、看看每一個 epoch 結束後的 **Training Error** 與 **Validation Error** 各為多少。(提示：`def predictor(x)` 應該要寫在 `learnPredictor()` 裡面) 若您成功完成，Console 上應該會看到與下圖一樣之片段輸出字樣：

```
----- START PART 4a-2: test classifier on real polarity dev dataset
Read 3554 examples from polarity.train
Read 3554 examples from polarity.dev
Training Error: (0 epoch): 0.3044456949915588
Validation Error: (0 epoch): 0.3531232414181204
Training Error: (1 epoch): 0.24366910523353968
Validation Error: (1 epoch): 0.33230163196398427
Training Error: (2 epoch): 0.21102982554867755
Validation Error: (2 epoch): 0.31907709622960045
Training Error: (3 epoch): 0.18880135059088352
Validation Error: (3 epoch): 0.30894766460326395
Training Error: (4 epoch): 0.1710748452447946
Validation Error: (4 epoch): 0.30388294879009564
Training Error: (5 epoch): 0.15616207090602138
Validation Error: (5 epoch): 0.3019133370849747
Training Error: (6 epoch): 0.14518851997749016
Validation Error: (6 epoch): 0.29938097917839057
Training Error: (7 epoch): 0.13590320765334835
Validation Error: (7 epoch): 0.29825548677546426
Training Error: (8 epoch): 0.12858750703432753
Validation Error: (8 epoch): 0.29487900956668545
Training Error: (9 epoch): 0.12042768711311198
Validation Error: (9 epoch): 0.2940348902644907
Training Error: (10 epoch): 0.11451885199774901
Validation Error: (10 epoch): 0.2920652785593697
Training Error: (11 epoch): 0.10776589758019134
Validation Error: (11 epoch): 0.28700056274620145
```

再來我們想嘗試一個更大膽的 feature extractor- **extractCharacterFeatures**

試想：若我們今天不要以「每個 token 出現的次數」當作 feature vector，而改以「長度 n 的英文字出現次數」來當作 feature vector，這樣是否可以讓演算法得到更好的準確率？

舉例來說，若我們使用 **extractWordFeatures('Such a good film')** 會得到

{'Such':1, 'a':1, 'good':1, 'film':1}

然而，若使用 **extractor = extractCharacterFeatures(4)** 再呼叫 **extractor('Such a good film')** 我們會得到

{'Such':1, 'ucha': 1, 'chag': 1, 'hago': 1, 'agoo': 1, 'good': 1, 'oodf': 1, 'odfi': 1, 'dfil': 1, 'film': 1}

請完成 `submission.py` 檔案裡名為 `def extractCharacterFeatures(n: int) -> Callable[[str], FeatureVector]` 的函式。這個函式非常有趣，它會 return 一個 function 讓其他人使用！如果對它的使用方法感興趣，歡迎參考 `grader.py` 檔案的 `def test5b0()`

請執行 `grader.py` 並看看是否可以通過 5b-0 的測試

```
81 #####
82 # Milestone 5b: character features
83
84 def extractCharacterFeatures(n: int) -> Callable[[str], FeatureVector]:
85     """
86     Return a function that takes a string |x| and returns a sparse feature
87     vector consisting of all n-grams of |x| without spaces mapped to their n-gram counts.
88     EXAMPLE: (n = 3) "I like tacos" -> {'Ili': 1, 'lik': 1, 'ike': 1, ...
89     You may assume that n >= 1.
90     """
91
92     def extract(x: str) -> Dict[str, int]:
93         # BEGIN_YOUR_CODE (our solution is 6 lines of code, but don't worry if you deviate from this)
94         raise Exception("Not implemented yet")
95         # END_YOUR_CODE
96
97     return extract
98
```

完成後請打開 `grader.py` 將 `test4a2()` 的 `featureExtractor = submission.extractWordFeatures` 改成 `featureExtractor = submission.extractCharacterFeatures(4)` (其他地方都不用更動就可以順利執行)

請問：重新執行 `grader.py` 後，兩個 `featureExtractor` 在表現上有沒有什麼顯著的差異？為什麼？請將答案寫在 `grader.py` `test4a2()` 下方的區間（如下圖）

```
def test4a2():
    trainExamples = readExamples('polarity.train')
    validationExamples = readExamples('polarity.dev')
    featureExtractor = submission.extractCharacterFeatures(4)
    weights = submission.LearnPredictor(trainExamples, validationExamples, featureExtractor, numEpochs=40, alpha=0.01)
    outputWeights(weights, 'weights')
    outputErrorAnalysis(validationExamples, featureExtractor, weights, 'error-analysis') # Use this to debug
    trainError = evaluatePredictor(trainExamples, lambda x: (1 if dotProduct(featureExtractor(x), weights) > 0 else -1))
    validationError = evaluatePredictor(validationExamples, lambda x: (1 if dotProduct(featureExtractor(x), weights) > 0 else -1))
    print(("Official: train error = %, validation error = %" % (trainError, validationError)))
    grader.require_is_less_than(0.04, trainError)
    grader.require_is_less_than(0.30, validationError)
    grader.add_basic_part('4a-2', test4a2, max_points=30, max_seconds=40, description="test classifier on real polarity dev dataset")

#####
# Milestone 5: Finishing up
# TODO: Is there any difference between using extractCharacterFeatures and extractWordFeatures? Why?
# TODO: Your answer here
# TODO:
#####
```


最後，請編輯 `interactive.py` 的檔案，使用 `util.py` 檔案裡面一個名為 `def interactivePrompt(featureExtractor, weights)`：來讓使用者透由 Console 輸入的任意影評並得到預測結果：是新鮮蕃茄 🍅 (好評)？還是爛番茄 🍌 (壞評)？

若您撰寫的程式碼無誤，應該會得到與下圖一模一樣之畫面
(請記得將 `grader.py` 裡面的 `numEpochs` 改回 40 個，`alpha=0.01`)：

```
<<< Your review >>>
although this movie suffers from some cliché , this film is still worth watching
Prediction: 1
still          1 * 1.069845092175029 = 1.069845092175029
film          1 * 0.6539496739181195 = 0.6539496739181195
although      1 * 0.5956374899882617 = 0.5956374899882617
worth         1 * 0.5664761564533873 = 0.5664761564533873
from          1 * 0.3951227010940324 = 0.3951227010940324
is            1 * 0.0864217551275587 = 0.0864217551275587
,             1 * 0.060854820899167696 = 0.060854820899167696
some          1 * -0.13929641755107536 = -0.13929641755107536
this          2 * -0.08245531036208927 = -0.16491062072417853
cliché        1 * -0.25610067498981853 = -0.25610067498981853
watching      1 * -0.26216353682064314 = -0.26216353682064314
movie         1 * -0.2713207370137551 = -0.2713207370137551
suffers       1 * -1.1271836282094954 = -1.1271836282094954
```



Congrats on finishing SC201 Assignment 2



大家應該要對自己感到非常驕傲！
因為這份作業跟史丹佛大學研究所課程 CS221: Artificial Intelligence 的作業二
非常相似，代表各位同學也成為世界上最頂尖的一群菁英了

請同學在繳交作業前，務必確保執行 `grader.py` 可以通過每一項測試、
得到 100/100 (如下圖所示)

```
Here is your grading! Make sure you achieve 100/100
before you upload your work to google sheet

===== END GRADING [100/100 points] =====
```

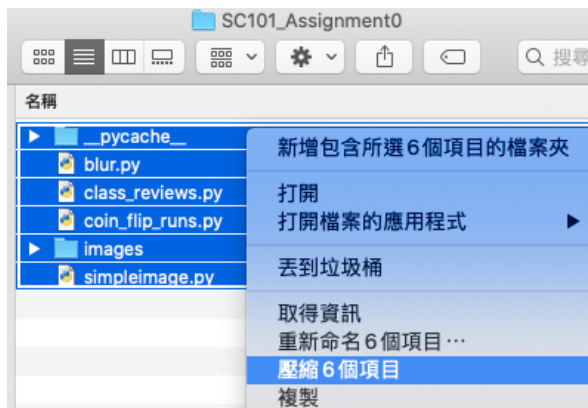

作業繳交

恭喜各位同學完成 SC201 Assignment2! 這份作業將帶領同學前往更艱深的觀念& AI 世界。這份作業的難度與史丹佛大學學生的作業非常相似，代表各位同學也成為世界上最強的一群人了！

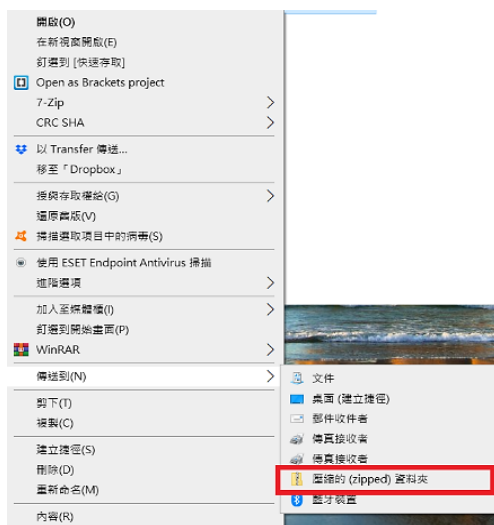
請同學依循下圖，完成作業上傳。

1. 以滑鼠「全選」作業資料夾內的所有檔案，並壓縮檔案。請見下圖說明。

macOS：按右鍵選擇「壓縮n個項目」



Windows：按右鍵選擇「傳送到」→「壓縮的(zipped)資料夾」



2. 將壓縮檔(.zip)重新命名為「a(n)_中文姓名」。如：

Assignment 1命名為Assignment1_中文姓名;

Assignment 2命名為Assignment2_中文姓名; …



3. 將命名好的壓縮檔(.zip)上傳至Google Drive（或任何雲端空間）

1) 搜尋「google drive」

2) 登入後，點選左上角「新增」→「檔案上傳」→選擇作業壓縮檔(.zip)

4. 開啟連結共用設定，並複製下載連結

1) 對檔案按右鍵，點選「共用」

2) 點擊「變更任何知道這個連結的使用者權限」後，權限會變為「可檢視」

3) 點選「複製連結」



5. 待加入課程臉書社團後，將連結上傳至作業貼文提供的「作業提交表單」