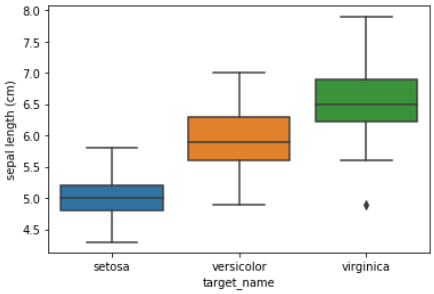
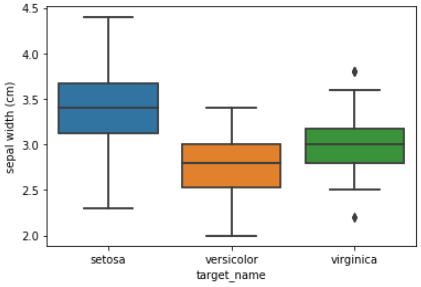
Machine Learning Homework 1 Report

一、Iris Dataset

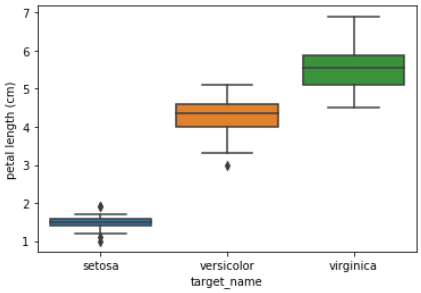
1. Data Visualization



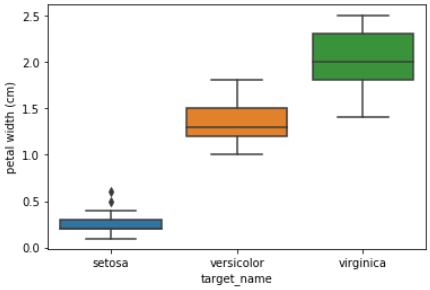
(a)



(b)

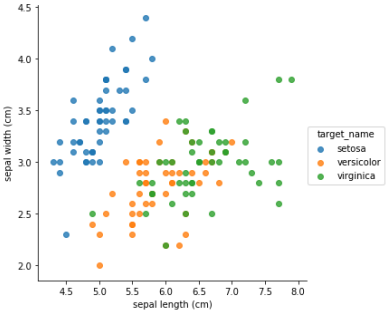


(c)

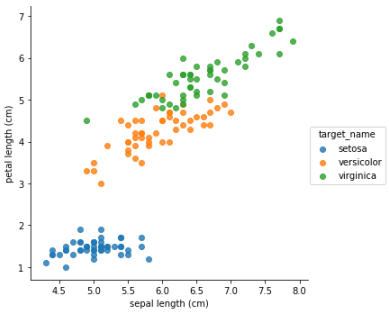


(d)

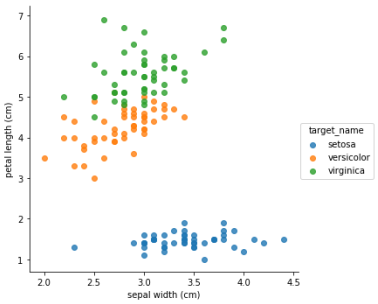
Fig.1 Boxplot of Iris Dataset with 4 different features



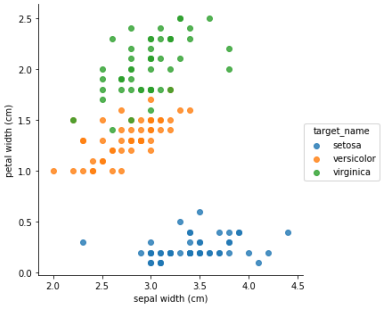
(a)



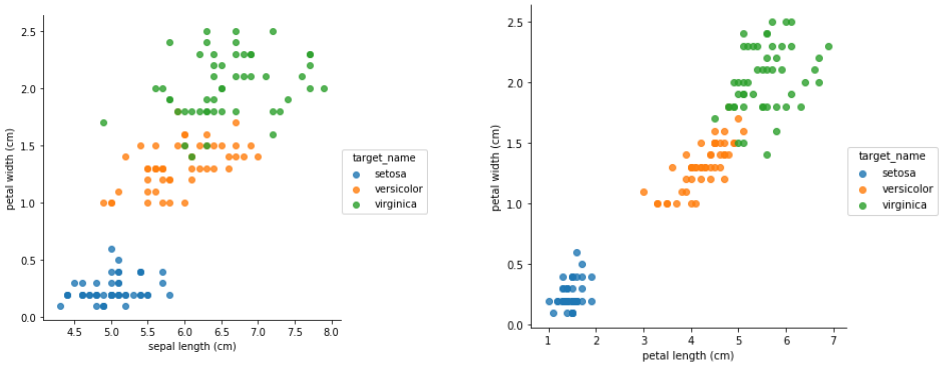
(b)



(c)



(d)



(e) (f)

Fig.2 Scatter plot of features in Iris dataset

從 Fig.1 和 Fig.2 中可以看出，花瓣的長寬是一個較為明顯的特徵，將 setosa 和 versicolor, virginica 兩類分開，且相較於花萼的長寬，versicolor 和 virginica 在花瓣方面有更顯著的不同。

有了這些資料分佈的圖片，可以更加瞭解為什麼 decision tree 會做出這樣的提問，也可以刪減不重要的特徵達到降低維度的效果。

2. Model

我們這組使用 python 作為建置 model 和視覺化的主要環境，在 model 的部分使用 sklearn 這個 package；在視覺化的部分則使用 seaborn 和 matplotlib 作為主要工具。

Preprocessing 的策略是先把.txt 處理成.csv，方便我們後面使用 pandas 做後續的 training data 和 testing data 的準備。而且使用 pandas 的 dataframe 我們可以利用 describe()取得統計的基本數據，如下表。

Table 1. Statistic features of Setosa

	Sepal Length(cm)	Sepal Width(cm)	Petal Length(cm)	Petal Width(cm)
mean	5.006	3.418	1.464	0.244
Std	0.3524897	0.3810244	0.1735112	0.1072095
Min	4.3	2.3	1	0.1
25%	4.8	3.125	1.4	0.2
50%	5	3.4	1.5	0.2
75%	5.2	3.675	1.575	0.3
Max	5.8	4.4	1.9	0.6

Table 2. Statistic features of Versicolor

	Sepal Length(cm)	Sepal Width(cm)	Petal Length(cm)	Petal Width(cm)
mean	5.936	2.77	4.26	1.326
Std	0.51617	0.3138	0.46991	0.19775
Min	4.9	2	3	1
25%	5.6	2.525	4	1.2
50%	5.9	2.8	4.35	1.3
75%	6.3	3	4.6	1.5
Max	7	3.4	5.1	1.8

Table 3. Statistic features of Virginica

	Sepal Length(cm)	Sepal Width(cm)	Petal Length(cm)	Petal Width(cm)
mean	6.588	2.974	5.552	2.206
Std	0.63588	0.3225	0.55189	0.27465
Min	4.9	2.2	4.5	1.4
25%	6.225	2.8	5.1	1.8
50%	6.5	2	5.55	2
75%	6.9	3.175	5.875	2.3
Max	7.9	3.8	6.9	2.5

至於 training data 和 testing data 的區分，則是使用 sklearn 中的 train_test_split()函式，區分的比例為 0.8, 0.2。以 Iris 這個 dataset 來說，就是 120 筆的訓練資料，30 筆的測試資料。

進行完 preprocessing 後，就可以開始著手做 model 的撰寫。由於 sklearn 已經內建 tree 這個 classifier 並且有許多參數可做調整，所以我們並沒有在自行撰寫一個。在 random forest 自行撰寫版中，也引用了 sklearn 的 tree 作為建構 forest 的基礎。

在自行撰寫的 Random Forest 中，總共撰寫了三個函式，基礎的 randomForest()函式，裏面包含了依照使用者輸入的樹的數量建構森林的方法，以及集合利用投票進行最後預測的 list。subsample()函式，根據使用者所希望每棵樹所採納的樣本比例以及特徵數量，從樣本中取樣。最後則是 bagging_pred()函式集合所有樹對一個樣本的預測，並進行投票。

在 Iris dataset 的 random forest 模型中，我們使用了 10 棵樹，採樣的比例為訓練資料的 0.7，並使用全部的特徵。

關於 decision tree 和 random forest 的模型我們總共測試了兩種計算純度的方法，一個是 Shannon's entropy，另一個是 Gini impurity。Fig.3 和 Fig.4 就是分別用兩種不同的標準計算出來的 decision tree。

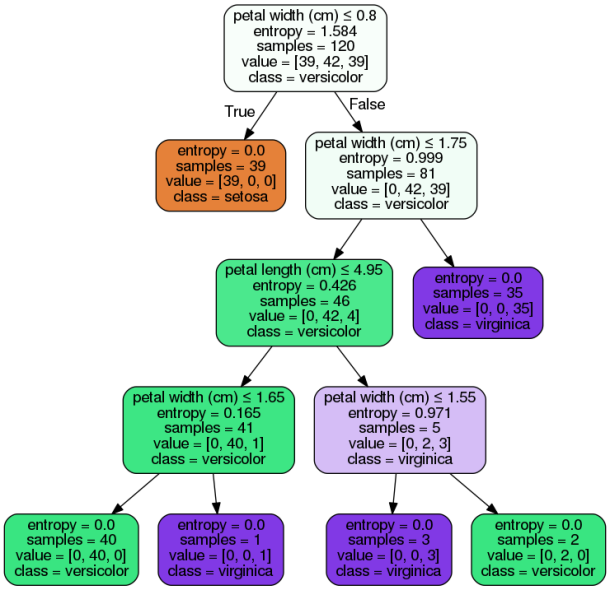


Fig. 3 Decision tree used entropy as criterion

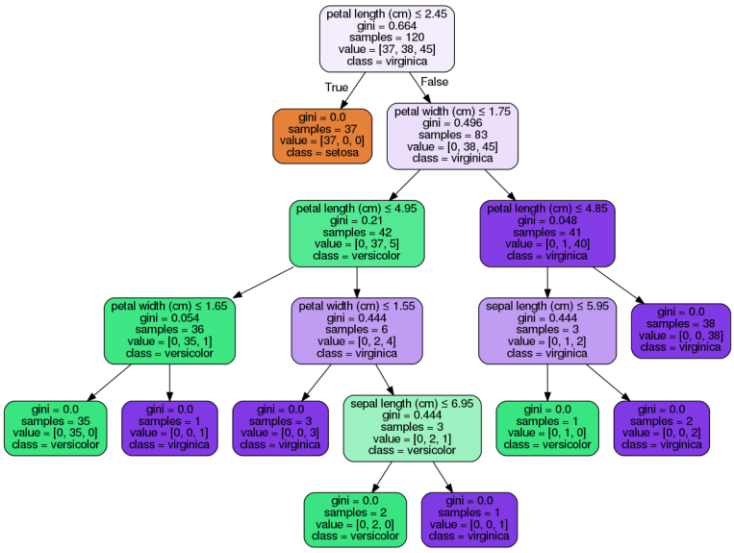
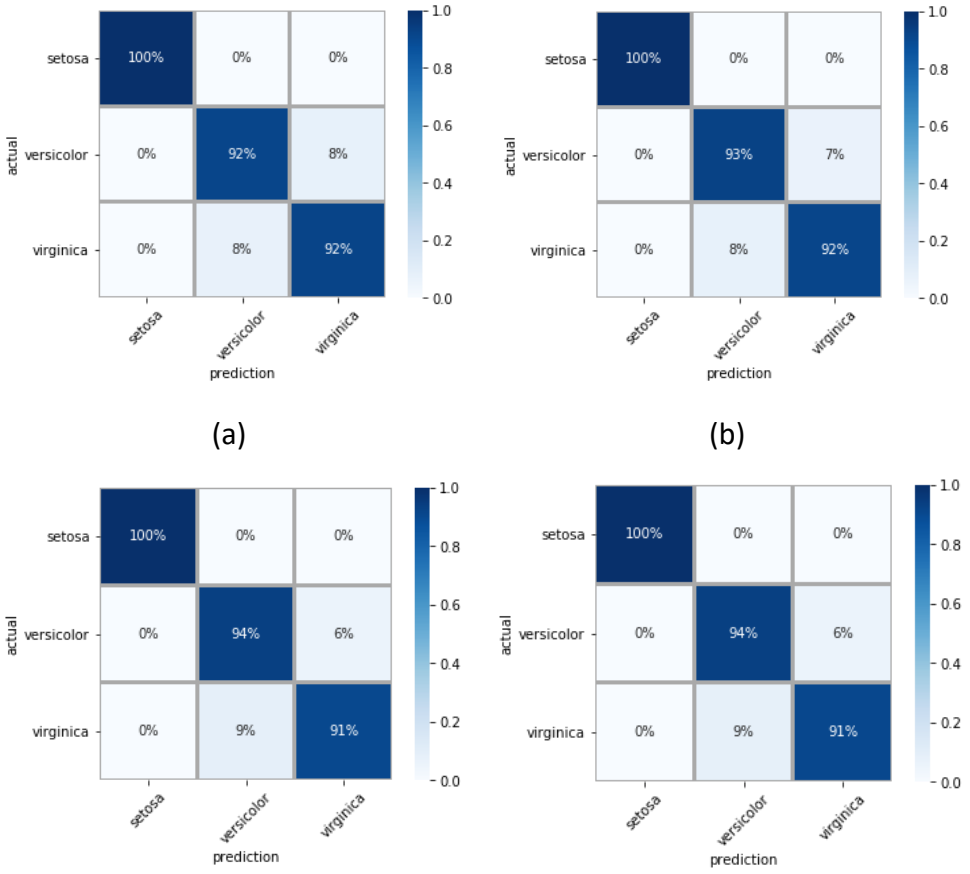


Fig.4 Decision tree used Gini impurity as criterion

而針對每一個不同的 model 設定，進行了三個不同的檢測：平均、Resubstitution 和 K-fold(K=3)。平均即是進行 n 次測試後，計算其混淆矩陣的平均以及準確度的平均，至於 Resubstitution 和 K-fold 都只有進行準確度的運算而已。

在進行 K-fold 的檢測時，Decision Tree 是使用 sklearn 中的 cross_val_score() 進行檢測。然而 random forest 是自行撰寫的無法套用 cross_val_score()這個函式，所以 K-fold 也是自行在 code 中對 dataset 進行切割後進行檢測，並沒有引用 sklearn 的函式。

3. Testing Results



(a)

(b)

(c)

(d)

Fig. 5 (a)Confusion matrices of Iris dataset with entropy decision tree model
(b)Confusion matrices of Iris dataset with Gini decision tree model
(c)Confusion matrices of Iris dataset with entropy random forest model
(d)Confusion matrices of Iris dataset with Gini random forest model

根據 Fig. 1 以及 Fig. 2 所給的視覺化資料對照 Fig. 3, Fig. 4 的範例可以發現 Setosa 是三類 Iris 中最好分辨的，簡單地利用花瓣的數值就可以進行區分，所以在混淆矩陣中拿到 100%的準確率也不是一件令人意外的事情。相對地，要區分 Versicolor 和 Virginica 這兩類是有些許困難的。從上面的 Fig. 2 的分佈可以看得出兩者是有些混雜在一起，所以 model 也較難辨認出二者的差別。

Table 4. The results of different models testing on Iris dataset

Experiments		Evaluation Method	Accuracy
Dataset	Iris	Average(100)	95.07%
Criterion	Gini	Resubstitution	100%
Model	Decision Tree	K-fold(K=3)	96.04%
Dataset	Iris	Average(100)	95.10%
Criterion	Gini	Resubstitution	98.78%
Model	Random Forest	K-fold(K=3)	94.68%
Dataset	Iris	Average(100)	94.83%
Criterion	Entropy	Resubstitution	100%
Model	Decision Tree	K-fold(K=3)	96.04%
Dataset	Iris	Average(100)	94.90%
Criterion	Entropy	Resubstitution	99.17%
Model	Random Forest	K-fold(K=3)	94.00%

從 Table 4.中，可以發現不論我們使用何種 Model 或檢驗方式，都可以有很高的正確率。不過，個人認為，這並不能歸功於 model 強，而是 Iris 這個 dataset 本身就算是容易的。

至於 decision tree 能在 Resubstitution 的檢測方法中拿到 100%的正確率，也不意外，畢竟 decision tree 停止生長的條件之一，就是區分完所有的測試資料。

不過出乎意料之外的是，random forest 的表現是略遜於 decision tree 的。個人的解釋為，因為這筆資料量過小，容易在 subsample 中取得未處於 versicolor 和 virginica 交界的資料，進而使邊界的劃分有所差異，導致錯誤的判斷。

雖然在這個 dataset 中 random forest 小輸於 decision tree，但在下一個 google play 的資料測試集中，卻是比 decision tree 好上 3~5%的 accuracy。

二、Google Play Dataset

1. Preprocessing

我們這組希望可以建造一個 decision tree 或 random forest 模型，能夠預測 APP 放到 google play 商店後的下載數。但是在將資料送進 model 前，我們必須先對資料進行一些處理，方才能夠開始訓練。

首先我們必須要把原本 googleplaystore.csv 中有些移位的資料用手動的方式把資料移回來，另外我們也要將檔案儲存成用 utf-8 加密的形式，不然在 pandas 把.csv 匯入 dataframe 時會發生錯誤。

當處理完上述的部分，接著要決定能夠使用的特徵，在這邊我們並沒有使用 PCA(Principle Component Analysis)、t-SNE 等降維方式，而只是考量一個 APP 大概會有哪些資訊。

- Category
- Size
- Type
- Price
- Content Rating
- Ratings

其中我們決定保留 Ratings 的原因是，一款 APP 在上市前可能會拿給朋友進行測試或者是封測，那時候可能就可以獲得類似 Ratings 的評價。

以下則是我們捨去的特徵：

- Genres
- Last Updated
- Current Version
- Android Version
- APP
- Reviews

捨去黑色部分的原因，主要是認為分類過細或者是認為沒有足夠影響力等。然而 Reviews 卻是一個和下載量明顯正相關的特徵，我們在此將其捨去的原因為下載量導致了更多的回覆，並非更多的回覆導致更多的下載量，而且一個並未上市的 APP 更不應該知道自己會有多少回覆數。

實際上，我們也進行過把 Reviews 加進 features 中的實驗，可以提升約 30%的準確度。然而，若我們將此特徵應用於 model 上，某種程度來說，已經算是作弊了。

決定完要使用的資料後，我們將資料先進行代碼化(tokenize)和去除雜訊，像是 Size 中，我們統一使用 MB 作為單位，去掉了原本資料內涵的單位 K、M、G 等字元；在 Price 上我們統一把美金符號去除；而在下載數方面，我們使用了兩種分類法。

在一開始，僅僅只是將下載數的加號去除，雖然資料庫中有 0 以及 0+兩

種下載量，但是 0 到 0+只需要自己下載即可完成，所以可以把他視為一類。

{0: 0, 1: 1, 5: 2, 10: 3, 50: 4, 100: 5, 500: 6, 1000: 7, 5000: 8, 10000: 9, 50000: 10, 100000: 11, 500000: 12, 1000000: 13, 5000000: 14, 10000000: 15, 50000000: 16, 100000000: 17, 500000000: 18, 1000000000: 19}

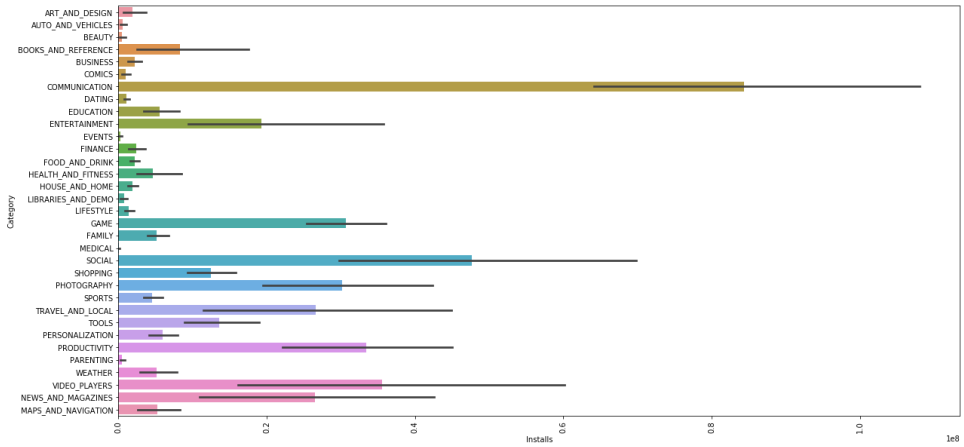
上面則是我們在 gpdata_preprocess.py 中所使用的 dictionary，來對下載量進行分類和代碼化。

然而，全部的資料都代碼化後，我們依然還有個問題，那就是 missing data。在這裡我們使用的策略是，利用該下載量特徵的眾數來填充原本遺失的資料。至於為什麼不用平均的原因為，在此我們的數值都是代碼化後的結果，用平均如果有小數點則顯得奇怪。

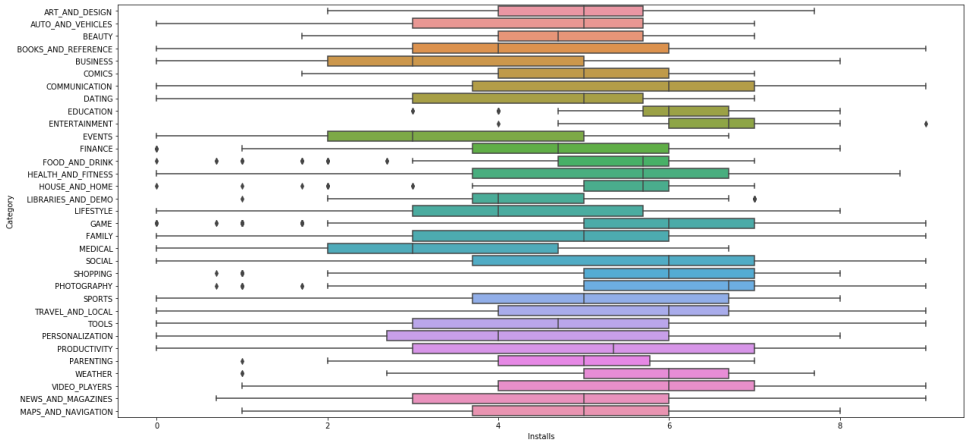
有處理良好的資料後，就可以進行視覺化以及丟進 model 中進行訓練以及預測。

2. Data Visualization

和前面的 Iris dataset 一樣，我們使用 matplotlib 和 seaborn 進行我們的視覺化。



(a)



(b)

Fig.6(a) Barplot of installs with different categories

(b)Boxplot of installs after logarithm with different categories

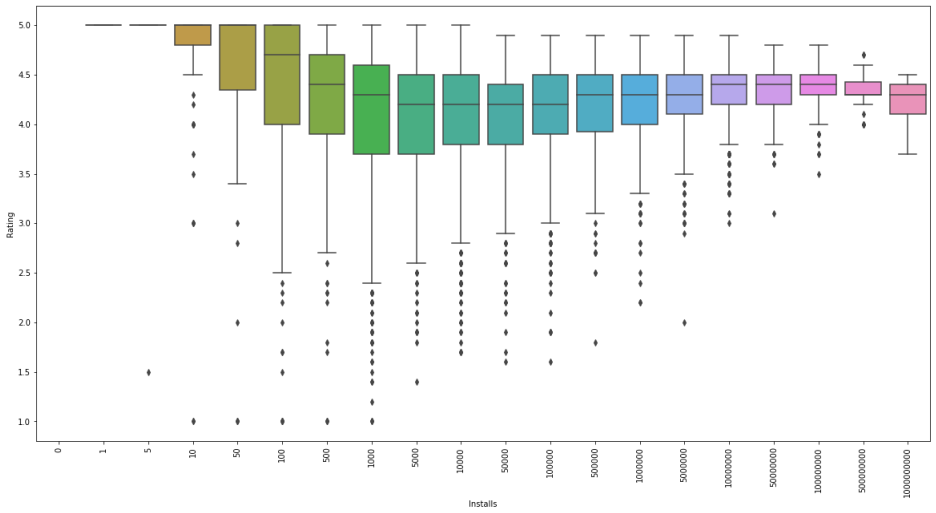


Fig.7 Boxplot of ratings with different installs

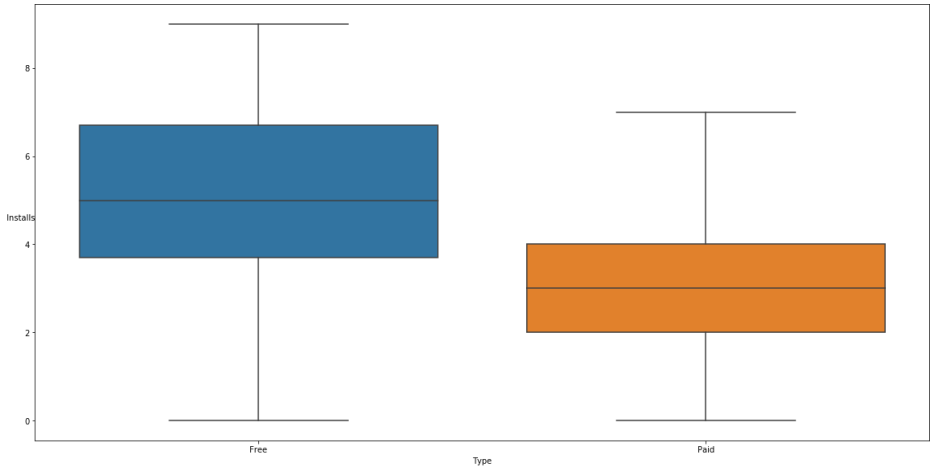


Fig.8 Boxplot of installs with binary features paid or free

在 Fig.6 中利用分類(Category)和下載量(Installs)做出長條圖和盒狀圖，然而在長條圖上的下載量平均容易受到極值影響，但若直接畫成盒狀圖，又有許多的值相對過小，無法看出分布，於是我們對全部的下載量取對數($\log_{10}$)，大多數的資料為十的冪次則是以 10 為底的原因。

Fig.7 則是一張很有趣的圖，評分呈現由高到低再由低至高的走向，而全距也有從大到小，再由小到大的走向。推測原因是，小的下載量時，下載的可能都是親朋好友，並沒有用客觀的方式審視這個 APP，到下載量中段時，則有較為客觀的資料，開始有負評的出現，然後到高下載量的部分我則因為 APP 確實滿足了大量使用者的需求，所以評分也大多為高分的資料。

Fig,8 顯示了是否付費的下載量分布，可以得到無須付費的軟體通常有較高的下載量，而付費的軟體則擁有較少的下載量。

有了這些基本資訊，我們知道，這些資訊或多或少，都會影響一個 APP 的下載量多寡。因此，我們就可以開始對 model 進行測試。

3. Model

我們和 Iris dataset 一樣，使用了 decision tree 和 random forest 兩種 model 來建構我們的學習模型。

不過和 Iris dataset 的 random forest 不一樣的是，我們在這個挑戰中，使用了 100 棵樹，並且有使用全部特徵，以及隨機取 3 種特徵兩種模型進行測試。

然而一開始我們測試時，準確度非常的低。以 Gini impurity 作為 criterion 的 Decision Tree 和 Random Forest 分別只有 25.50%和 26.63%的正確率，Resubstitution 分別為 84.58%和 84.42%，K-fold(k=3)檢測時，Decision Tree 更只有 17.46%的正確率，相對地 Random Forest 則有 24.75%的正確率。

這時，我們覺得其實沒有必要知道詳細下載量，我們只需要知道下載量的高低即可，所以我們將原本的 20 個不同的目標(target)，縮減為下面四個。

0~100 為低下載量，101~10000 為中下載量，10001~1000000 為高下載量，1000001~以上則為極高下載量。

Table 5. Statistic features of low installs

	Category	Rating	Size	Type	Price	Content Rating
Count	755	137	702	754	755	755
Mean	15.973510	4.548175	12.941027	0.194960	2.666967	1.316556
Std	8.723689	0.935344	17.352967	0.396433	23.201706	0.881089
Min	0	1	0	0	0	1
25%	11	4.5	0.004	0	0	1
50%	15	5	10	0	0	1
75%	23	5	22	0	0	1
Max	32	5	99	1	399.99	4

Table 6. Statistic features of medium installs

	Category	Rating	Size	Type	Price	Content Rating
Count	2434	1656	2231	2434	2434	2434
Mean	16.595316	4.123128	12.050026	0.144618	2.084182	1.291701
Std	8.461037	0.774271	17.975517	0.351787	23.051846	0.839062
Min	0	1	0.001	0	0	1
25%	11	3.8	0.0038	0	0	1
50%	15	4.3	0.0084	0	0	1
75%	24	4.7	19	0	0	1
Max	32	5	100	1	399.99	5

Table 7. Statistic features of high installs

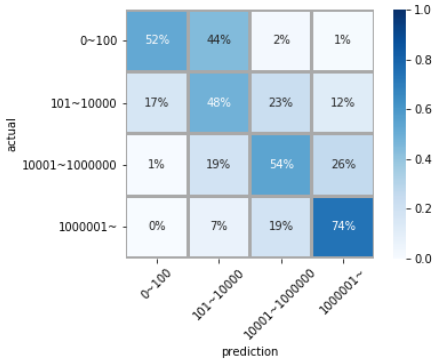
	Category	Rating	Size	Type	Price	Content Rating
Count	3241	3165	2878	3241	3241	3241
Mean	16.139155	4.088878	17.498471	0.084542	1.223844	1.429189
Std	8.311085	0.526608	22.035840	0.278242	17.992298	0.963532
Min	0	1.6	0.001	0	0	0
25%	11	3.9	0.005	0	0	1
50%	14	4.2	12	0	0	1
75%	23	4.5	27	0	0	1
Max	32	5	100	1	400	5

Table 8. Statistic features of extremely high installs

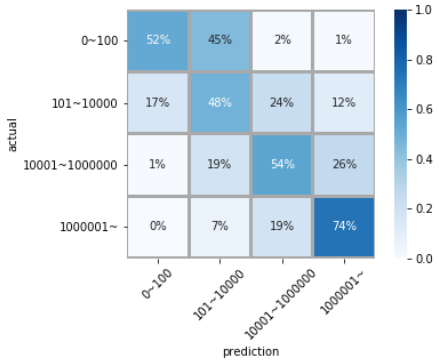
	Category	Rating	Size	Type	Price	Content Rating
Count	4411	4409	3019	4411	4411	4411
Mean	17.359329	4.279791	30.666753	0.006121	0.018982	1.612333
Std	8.217004	0.304887	27.608940	0.078006	0.296707	1.114437
Min	0	2	0.0012	0	0	0
25%	11	4.1	11	0	0	1
50%	14	4.3	24	0	0	1
75%	25	4.5	49	0	0	2
Max	32	4.9	100	1	6.99	4

經過這樣的縮減，模型的準確率不再像剛剛一樣慘不忍睹，正確率可以達到 50%以上，雖然如此，我們的 model 依然有許多的問題，個人覺得有待修繕。

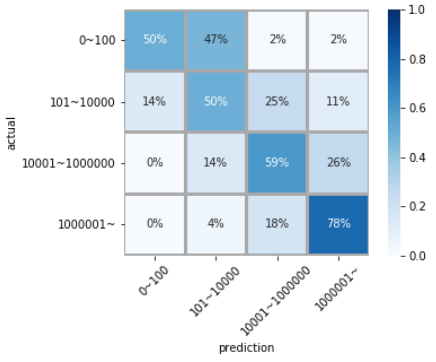
4. Testing Results



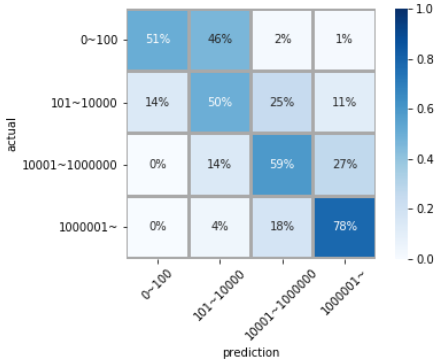
(a)



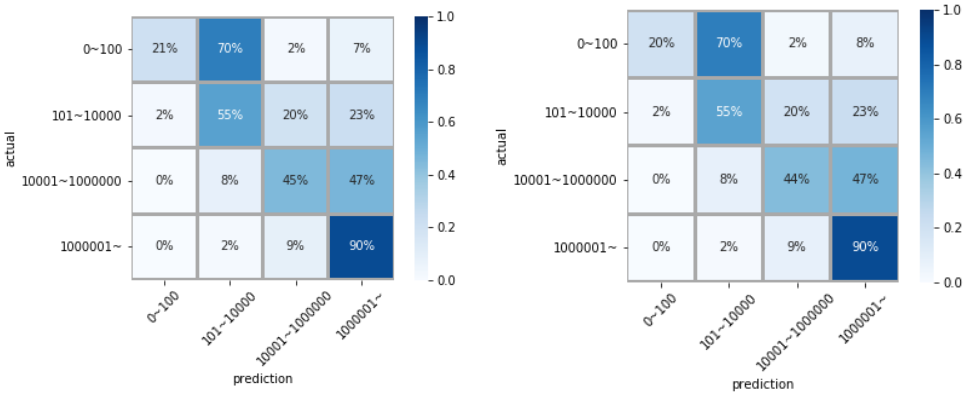
(b)



(c)



(d)



(e) (f)

Fig.9 (a)Decision tree used Gini as criterion

(b)Decision tree used entropy as criterion

(c)Random forest with all the features used Gini as criterion

(d)Random forest with all the features used entropy as criterion

(e)Random forest with 3 of random chosen features used Gini as criterion

(f)Random forest with 3 of random chosen features used entropy as criterion

Table. 9 The results of different models testing on Google Play dataset

Experiments		Evaluation Method	Accuracy
Dataset	Google Play	Average (100)	60.89%
Criterion	Gini	Resubstitution	95.78%
Model	Decision Tree	K-fold(K=3)	54.76%
Dataset	Google Play	Average (100)	60.74%
Criterion	Entropy	Resubstitution	95.94%
Model	Decision Tree	K-fold(K=3)	54.86%
Dataset	Google Play	Average (100)	64.02%
Criterion	Gini	Resubstitution	95.23%
Model	Random Forest (6)	K-fold(K=3)	63.90%
Dataset	Google Play	Average (100)	64.21%
Criterion	Entropy	Resubstitution	95.27%
Model	Random Forest (6)	K-fold(K=3)	63.66%
Dataset	Google Play	Average (100)	63.86%
Criterion	Gini	Resubstitution	73.97%

Model	Random Forest (3)	K-fold(K=3)	63.49%
Dataset	Google Play	Average (100)	63.43%
Criterion	Entropy	Resubstitution	73.66%
Model	Random Forest (3)	K-fold(K=3)	62.82%

從 confusion matrices 中我們可以看到這些 model 的一個重大問題。雖然準確率提升了，但對於下載量偏少的資料預測還是實屬薄弱，而且都有把低下載量預測成中下載量的趨勢。

另一個較為有趣的點是，如果我們採用 random feature selection 的話，相較於使用全部 features 的 random forest 在極高下載量有較好的表現，反之，在低下載量的表現，就顯得差強人意。

我們可以從 k-fold 中看到資料量對於 model 的影響，decision tree 對於資料量有較敏感的反應，相對地 random forest 就可以看出他即使有較少的訓練資料，依然可以較快地掌握資料特性做出預測。

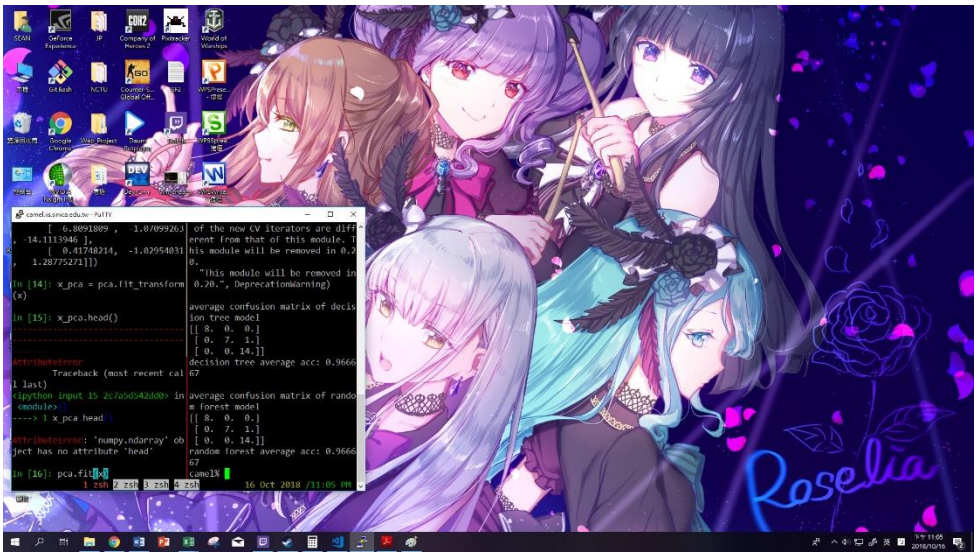
除了以上的 model 之外，曾經也嘗試過利用 PCA 降低維數後利用 sklearn.decomposition.PCA 中的 fit_transform()把資料分布降低到低維空間，然而經過測試後，不論是利用 decision tree 或者是 random forest 都沒有進步，因此我並沒有把結果記錄下來。

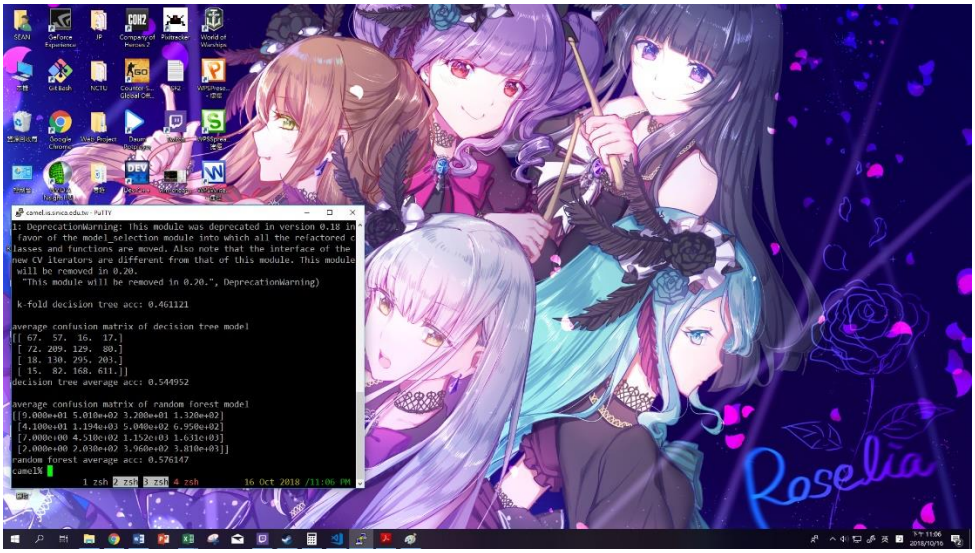
個人認為，可能這些特徵就已經足夠代表資料的特性，進行 PCA 降維之後可能破壞了原本資料的一些結構或者是特性，使得 model 沒有顯著的提升。

另外我們這組尚未嘗試對各項特徵進行標準化(normalization)，或許這樣可以有助於正確率提升。

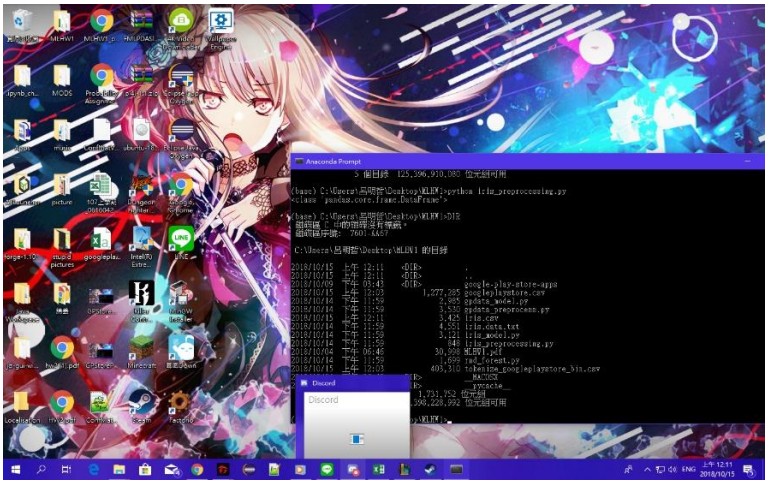
附錄——組員執行程式狀況紀錄

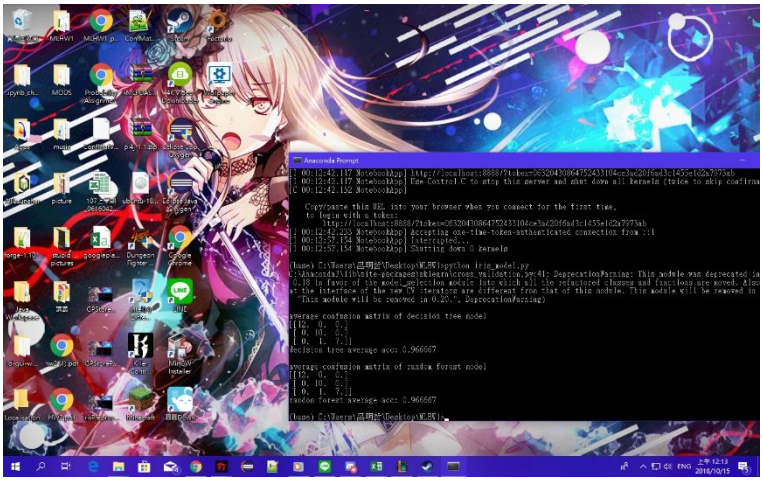
- 夏軒安

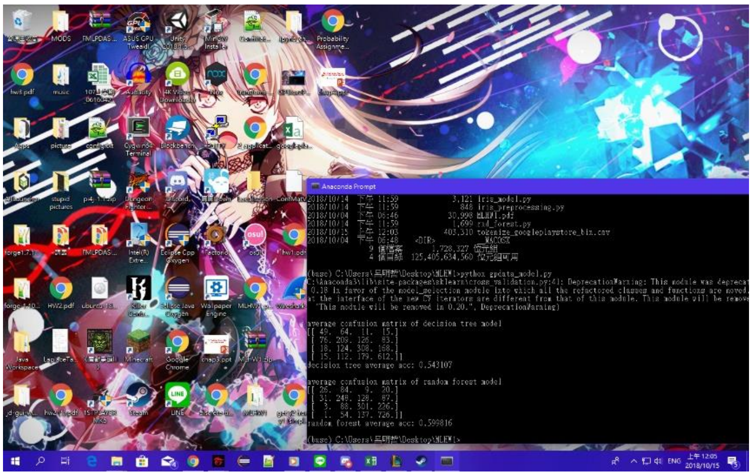




- 呂明哲

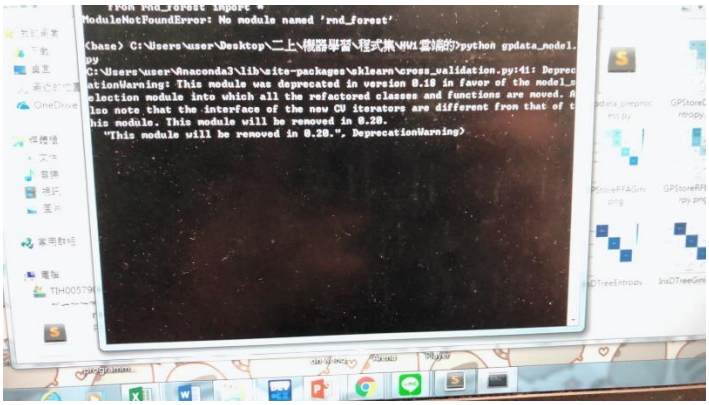


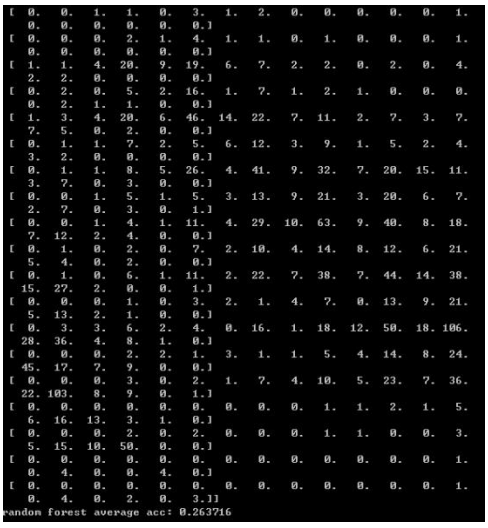



Anaconda Prompt
2018/10/14 下午 11:59 3,121 iris_model.py
2018/10/14 下午 11:59 848 iris_preprocessing.py
2018/10/04 下午 06:46 30,998 MLHW1.pdf
2018/10/14 下午 11:59 1,699 rnd_forest.py
2018/10/15 上午 12:03 403,310 tokenize_googleplaystore_bin.csv
2018/10/04 下午 06:48 <DIR> __MACOSX
9 個檔案 1,728,327 位元組
4 個目錄 125,405,634,560 位元組可用
(base) C:\Users\呂明哲\Desktop\MLHW1>python gpdata_model.py
"This module will be removed in 0.20.", DeprecationWarning)
average confusion matrix of decision tree model
[[49. 64. 11. 15.]
[76. 209. 126. 83.]
[18. 124. 308. 168.]
[15. 112. 179. 612.]]
decision tree average acc: 0.543107
average confusion matrix of random forest model
[[26. 84. 9. 20.]
[31. 248. 128. 87.]
[3. 88. 301. 226.]
[1. 54. 137. 726.]]
random forest average acc: 0.599816
(base) C:\Users\呂明哲\Desktop\MLHW1>
上午 12:05
2018/10/15

- 張皓鈞


Anaconda Prompt - cmd
(base) C:\Users\user\Desktop\二上\機器學習\程式集\HW1雲端的\iris>python iris_model.py
C:\Users\user\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
"This module will be removed in 0.20.", DeprecationWarning)
average confusion matrix of decision tree model
[[7. 0. 0.]
[0. 9. 1.]
[0. 0. 13.]]
decision tree average acc: 0.966667
average confusion matrix of random forest model
[[7. 0. 0.]
[0. 9. 1.]
[0. 0. 13.]]
random forest average acc: 0.966667
(base) C:\Users\user\Desktop\二上\機器學習\程式集\HW1雲端的\iris>python iris_model.py
C:\Users\user\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
"This module will be removed in 0.20.", DeprecationWarning)
average confusion matrix of decision tree model
[[13. 0. 0.]
[0. 9. 0.]
[0. 1. 7.]]
decision tree average acc: 0.966667
average confusion matrix of random forest model
[[13. 0. 0.]
[0. 9. 0.]
[0. 1. 7.]]
random forest average acc: 0.966667


ModuleNotFoundError: No module named 'rnd_forest'
(base) C:\Users\user\Desktop\二上\機器學習\程式集\HW1雲端的>python gpdata_model.py
C:\Users\user\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
"This module will be removed in 0.20.", DeprecationWarning)


[0. 0. 1. 1. 0. 3. 1. 2. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 2. 1. 4. 1. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
[1. 1. 4. 20. 9. 19. 6. 7. 2. 2. 0. 2. 0. 4. 2. 2. 0. 0. 0. 0.]
[0. 2. 0. 5. 2. 16. 1. 7. 1. 2. 1. 0. 0. 0. 0. 2. 1. 1. 0. 0.]
[1. 3. 4. 20. 6. 46. 14. 22. 7. 11. 2. 7. 3. 7. 7. 5. 0. 2. 0. 0.]
[0. 1. 1. 7. 2. 5. 6. 12. 3. 9. 1. 5. 2. 4. 3. 2. 0. 0. 0. 0.]
[0. 1. 1. 8. 5. 26. 4. 41. 9. 32. 7. 20. 15. 11. 3. 7. 0. 3. 0. 0.]
[0. 0. 1. 5. 1. 5. 3. 13. 9. 21. 3. 20. 6. 7. 2. 7. 0. 3. 0. 1.]
[0. 0. 1. 4. 1. 11. 4. 29. 10. 63. 9. 40. 8. 18. 7. 12. 2. 4. 0. 0.]
[0. 1. 0. 2. 0. 7. 2. 10. 4. 14. 8. 12. 6. 21. 5. 4. 0. 2. 0. 0.]
[0. 1. 0. 6. 1. 11. 2. 22. 7. 38. 7. 44. 14. 38. 15. 27. 2. 0. 0. 1.]
[0. 0. 0. 1. 0. 3. 2. 1. 4. 7. 0. 13. 9. 21. 5. 13. 2. 1. 0. 0.]
[0. 3. 3. 6. 2. 4. 0. 16. 1. 18. 12. 50. 18. 106. 28. 36. 4. 8. 1. 0.]
[0. 0. 0. 2. 2. 1. 3. 1. 1. 5. 4. 14. 8. 24. 45. 17. 7. 9. 0. 0.]
[0. 0. 0. 3. 0. 2. 1. 7. 4. 10. 5. 23. 7. 36. 22. 103. 8. 9. 0. 1.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 2. 1. 5. 6. 16. 13. 3. 1. 0.]
[0. 0. 0. 2. 0. 2. 0. 0. 0. 1. 1. 0. 0. 3. 5. 15. 10. 50. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 4. 0. 0. 4. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 4. 0. 2. 0. 3.]]
random forest average acc: 0.263716