午餐系統及分析

# 摘要

# 壹、研究動機

之前在學校訂餐的時候，都只能用紙本點餐單來點餐，代訂同學常常把紙本單弄丟，而且紙本單必須手工計算數量、金額，十分不便，紙本單是不完全記名制，只能知道是哪一班點的，常常會有人忘記自己點了什麼餐，直接隨便亂拿一份餐，我們認為應該要有一個比紙本單更優秀的解決辦法，便開始著手製作午餐系統了。

午餐系統在校內推行成功後，我們了解到廠商常常會備料過剩，於是我們打算建立一個模型，可以給廠商做為明天會有多少份餐點的依據。

# 貳、研究目的

本研究旨在於製作一套能夠取代紙本點餐單的系統，並設計一個適當的數學模型供廠商參考明天該準備多少份餐點。

製作一套能夠完全取代紙本單據的系統，這是本研究的第一目標；再根據收集到的資料給予廠商適當的預測，這是本研究的第二目標；我們意外發現對於單層的神經網路模型能夠以三分搜優化，證明演算法可行且具有較高的效率，這是本研究的第三目標。

# 參、研究設備及器材

|  |  |
| --- | --- |
| 器材 | 用途 |
| *Firebase* | *App*分析用的*Third party plugin* |
| *Google Analytics* | 網頁分析用的*Third party plugin* |
| *Ubuntu 18.04 LTS* | *Server OS* |
| *Nginx* | *Web Server Application* |
| *Noip Dynamic Dns* | *Server Domain name* |
| *Mac* | *iOS App*的開發環境 |
| *Iphone* | *iOS App*的測試環境 |
| *Xcode* | *iOS App* *IDE* |
| *Android*手機 | *Android App*的測試環境 |
| *Android Studio* | *Android App IDE* |
| *Chrome* | 網頁前端測試環境 |
| *Visual Studio* | 開發廠商前端的*IDE* |
| *Visaul Studio Code* | 開發後端的*IDE* |
| *Html + Css + Bootstrap* | 網頁前端開發框架 |
| *Javascript + JQuery + Ajax* | 網頁前端開發語言 |
| *Php* | 後端開發語言 |
| *Mysql* | 資料庫 |
| *Kotlin* | *Android App*開發語言 |
| *Swift* | *iOS App*開發語言 |
| *Python* | 數據分析測試語言 |
| *C#* | 廠商前端開發語言 |
| *Linux terminal bash + Crontab* | *Database auto backup* |

# 肆、研究過程及方法

午餐系統後端由*Php*作為後台，*Mysql*作為資料庫，並且輸出結果到交換介面；網頁版前台、*iOS*前台、*Android*前台及其他外掛插件，從交換介面擷取資料，再將資料展現給使用者，下圖為午餐系統的架構圖。

午餐系統每天都會有大量的點餐資料，如果能對這些資料進行分析，就能夠協助廠商預測明天的餐點量，於是我們設計了一個預測模型。

## 系統開發過程

下表簡略的描述了我們的開發過程，我們曾重寫架構，這意味著後前台、後台以及資料庫都必須重新打掉重寫；我們也有不少過渡期功能，例如代訂繳款、合作社繳款，在還沒連結*POS*時，代訂必須跟全班收錢，並且將班上的錢交給合作社，合作社再將錢分配給廠商，這些功能在連結*POS*後便功成身退。現在所見的成品只是冰山一角，很多功能都經過數次重寫或被捨棄掉了。

|  |  |
| --- | --- |
| 日期 | 功能 |
| 2017/11 | 連接學生與代訂，手工抄寫至點餐單，再交給廠商。 |
| 2018/03 | 重寫系統架構，包含前台、後台以及資料庫。 |
| 2018/09 | 連接學生、代訂、合作社、廠商。 |
| 2019/01 | 引入*POS*繳款機制，架空代訂以及合作社。 |

開發演算法時，我們先以*Python*開發雛形，確定演算法可行；確認可行後，再將模型改寫成*C#*，並為模型製作*GUI*介面，引入廠商管理前端。

## 系統使用簡介

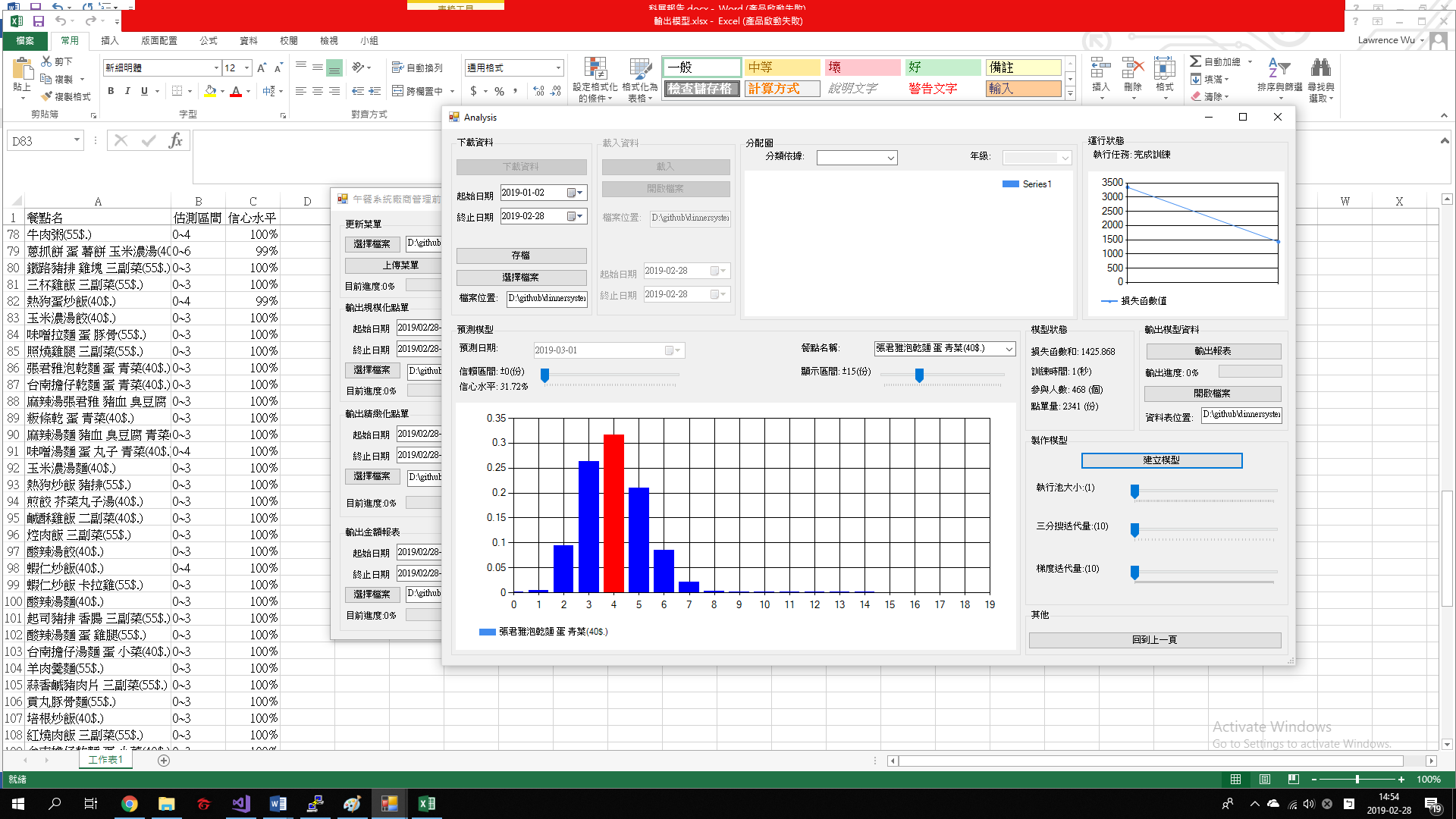
下表為實際使用系統點餐的擷圖，以下簡介以下四種最常被用到的功能。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 登入 | 點餐 | 查看點單 | 繳款 |
| 安卓前端 | C:\Users\lawre\AppData\Local\Microsoft\Windows\INetCache\Content.Word\52995536_341576016452924_4492182019454795776_n.jpg | C:\Users\lawre\AppData\Local\Microsoft\Windows\INetCache\Content.Word\53020164_327182224573781_1994272908897157120_n.jpg | C:\Users\lawre\AppData\Local\Microsoft\Windows\INetCache\Content.Word\52859457_560714564433881_1700737264587374592_n.jpg |  |
| 蘋果前端 | C:\Users\lawre\AppData\Local\Microsoft\Windows\INetCache\Content.Word\52785773_2010996985860581_7006054668753174528_n.jpg |  |  |  |
| 網頁前端 | C:\Users\lawre\AppData\Local\Microsoft\Windows\INetCache\Content.Word\52877884_1913323308776129_3492376395303092224_n.jpg | C:\Users\lawre\AppData\Local\Microsoft\Windows\INetCache\Content.Word\53509579_540146379724500_6811099317139406848_n.jpg |  |  |

下表為實際使用廠商管理插件的截圖，以下簡介三種最常被用到的功能。

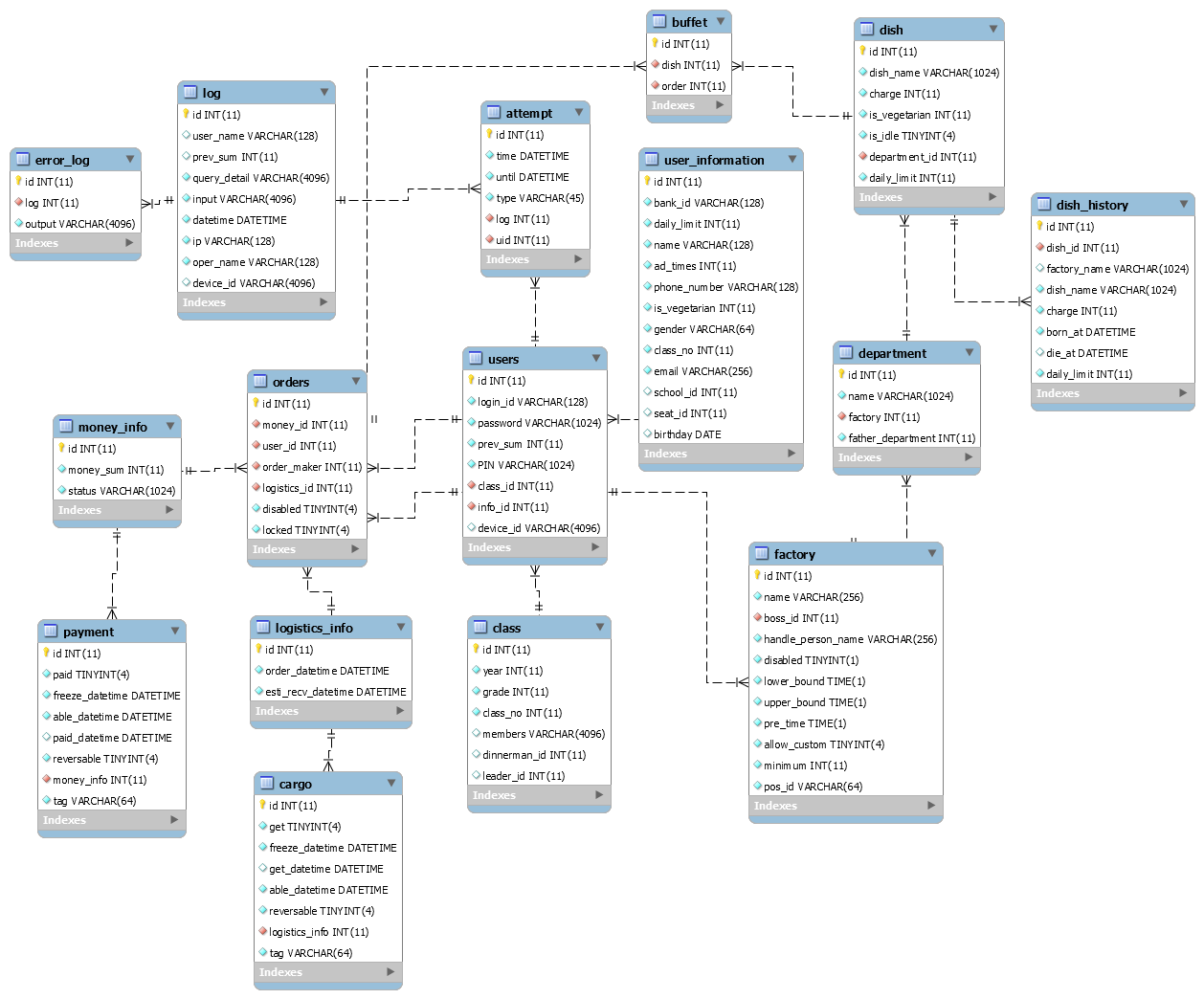
|  |  |  |
| --- | --- | --- |
|  | 廠商管理插件 | 匯出表格 |
| 更新菜單 |  |  |
| 查看點單 |  |  |
| 查看預測模型 |  |  |

下圖為實際使用預測模型的截圖，我們特別製作可以查看模型的GUI介面，方便廠商估計明天需要準備多少餐點。



## 三、資料庫

下列為午餐系統的資料結構模型，每一條線代表一個關聯性連接。



午餐系統的資料庫為關聯性資料庫，並使用*innoDB*做為引擎，*innoDB*支援交易機制，比起*myisam*，使用*innoDB*更方便處理死結回溯的問題。

### (一)、*Deadlock*

在單線程的測試環境下，很少會遇到資料庫死結*(Deadlock)*，而在系統真正運行的時候，常會遇到不可預知的死結。一個*Procedure*包裝了多個語句*(Syntax)*，若是在尚未執行完*Procedure*前，發生了死結，則會發生不可預期的後果。

我們針對容易發生死結的*Procedure*加上*start transcation*、*rollback*、*commit*，若是在*Procedure*尚未結束前發生死結，則回溯*(Rollback)*整個*Procedure*的操作。

### (二)、*InnoDB Locks*

下表為*InnoDB*內建的鎖，例如資料庫同時接到「繳款」以及「刪除」的指令，資料庫會先在該筆訂單上*IX*鎖，確保另外一個指令沒辦法更改資料，再拿到*X*鎖修改資料，最後釋放*X*鎖以及*IX*鎖，使得另外一個指令開始運行。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | X | IX | S | IS |
| X | 衝突 | 衝突 | 衝突 | 衝突 |
| IX |  | 相容 | 衝突 | 相容 |
| S |  |  | 相容 | 相容 |
| IS |  |  |  | 相容 |

## 四、後端

下圖為後端的架構，我們將後端分成多個模組，方便維護，也方便開發新功能。

#### (一)、*Session* 阻塞

*php*為了保證執行緒安全，同一個*session*同時只能給一個請求使用。在每個請求都不會提前釋放*session*的狀況下，同時送出大量請求，會使得效能非常低落，如下圖所示。

上圖中，請求(3)等候前面兩個請求處理完才開始執行，成為緩慢的串行命令。如果程式會先複製好*session*再執行，則每個請求只需要等候其他請求複製完資料，就能夠先開始執行了，如下圖所示。

上圖中，請求(3)僅等候前面兩個請求複製資料，因為不必等候其他請求，因此能受益於*CPU*的平行處理，使得效能提升。

#### (二)、資料庫存取阻塞

資料庫的存取速度遠遠低於記憶體的存取速度，如果每次使用常駐資料時，都向資料庫要求一次資料，則這些常駐資料請求會拖累系統效能。後端會將常駐資料先快取於*session*，需要使用資料時直接從*session*調用資料，就不必再向資料庫請求資料了。

## 五、前端

前端的任務在於將資料轉換成使用者能看懂的格式，並且協助使用者對伺服器下達命令，我們共有三種給學生的前端，分別是*iOS*、*Android*以及網頁前端。

#### (二)、比較、使用分布

下圖為各種前端的使用率分配圖，網頁佔了接近一半的使用率；安卓少了一些，不過使用率仍比蘋果高；蘋果的使用率最低，占了五分之一再少一點。

我們首先發布了網頁前端，再來是*iOS App*，最後發布*Android App*，而下表比較了三種前端之間的優劣利弊。

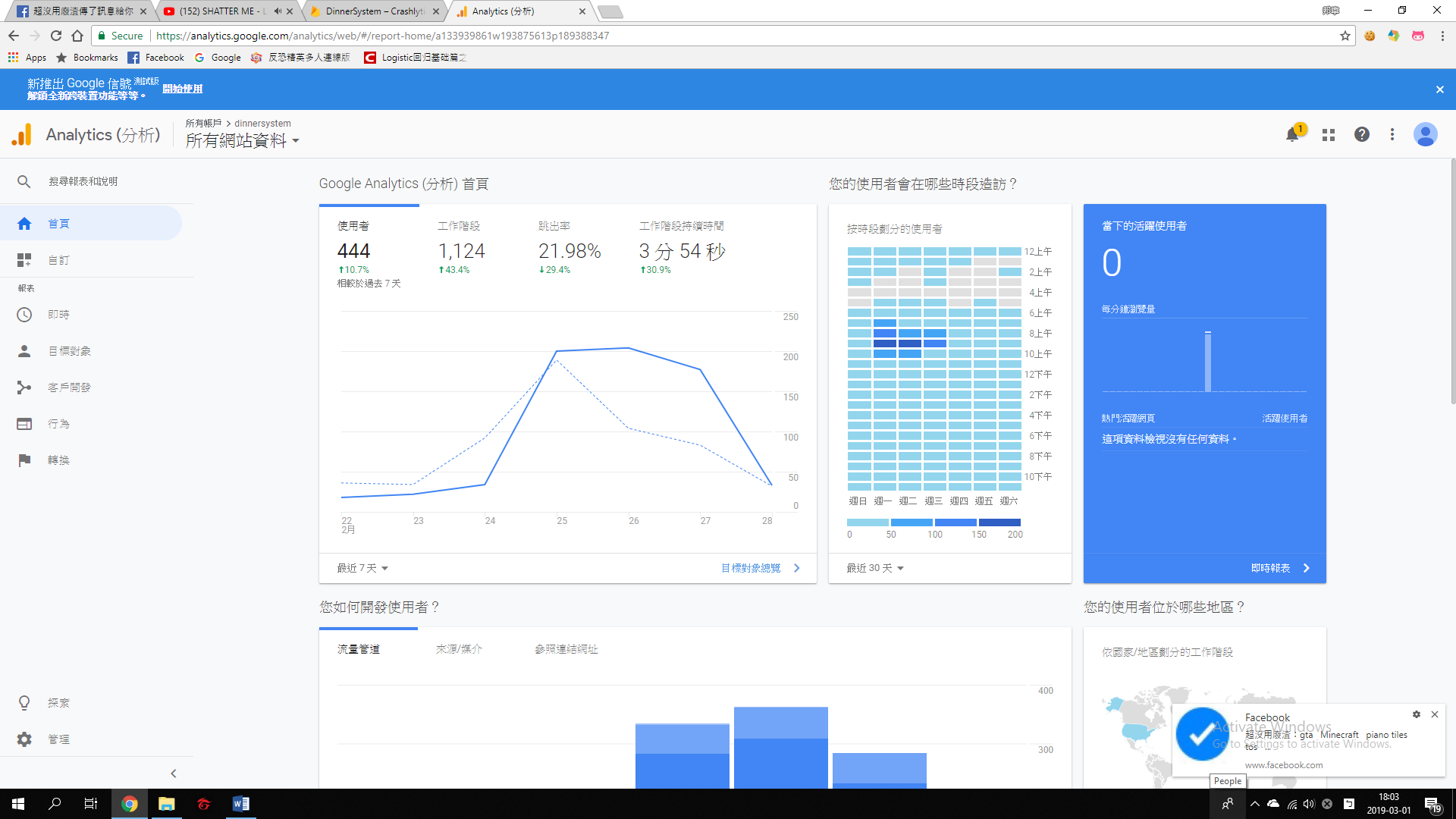
|  |  |  |  |
| --- | --- | --- | --- |
|  | 特色 | 優勢 | 劣勢 |
| 網頁版 | 最早推行的前端 | 不論系統，皆可使用 | 僅支援主流瀏覽器  不支援自助點餐 |
| *Android*前端 | 具有良好的防呆機制  介面較美觀 | 支援自助點餐 | 開發耗時較久 |
| *iOS*前端 | 具有良好的防呆機制  介面較美觀 | 支援自助點餐 | 開發耗時較久，上架審查機制嚴格 |

#### (一)、*Firebase、Google Analytics*

*Firebase*為*Google*提供的一項服務，其中分析的功能在開發者社群中大獲好評，我們使用*Firebase*作為*iOS*、*Android*的數據分析器，*Firebase*的其中一項功能，叫做*Crashlytics*，專門分析使用者的當機資料，我們使用*Crashlytics*來盡速搶修當機問題；*Google Analytics*也是由*Google*提供的一項服務，我們使用*Google Analytics*作為網頁的數據分析器，方便我們統計使用人數，也方便我們統計用戶尖峰。

|  |  |
| --- | --- |
| *iOS App* | *Android App* |
|  |  |

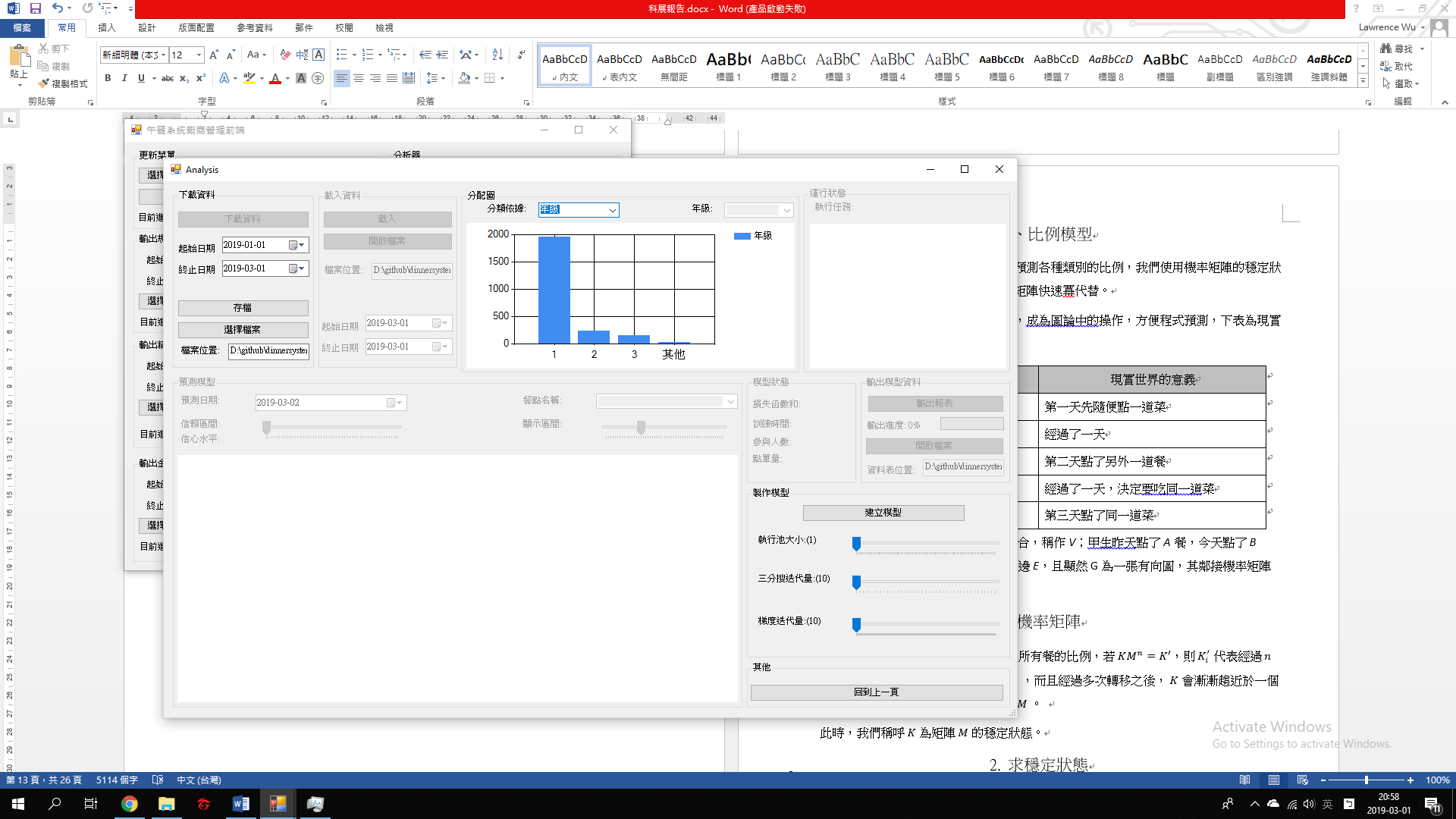
上表為*Crashlytics*的輸出報表，我們在2/26發布了緊急更新，而前後端之間有些許部分尚未協調完成，導致使用者受到當機影響。



上圖為*Google Analytic*s的輸出報表，2/25人數暴漲是因為當天星期一，學校需要訂購午餐；2/28人數暴跌是因為當天放假，所以學校不需要訂餐。

## 六、預測模型

下表為2019/01/01到2019/03/01的點餐人數分配圖，顯然高一為使用系統的主要族群。跟合作社、廠商打聽消息後，我們得知高二、高三習慣自己下去買便當，也有些許高二、高三選擇不購買校內便當。



我們將預測模型拆成兩個子模型，一為比例模型，二為數量模型。顧名思義，比例模型能夠給出明天的點餐比例，數量模型能夠給出明天的點餐總數，兩個模型一起使用就能得到明天各種餐點的數量。

不單以數量模型預估

我們在設計數量模型時，意外發現該模型滿足特別的數學性質，可以使用三分搜尋演算法加快

### (一)、比例模型

設計比例模型旨在於「預測未來各類別訂購量的比例」，我們使用機率矩陣的穩定狀態來預測比例，無法求得穩定狀態時，以平均狀態代替。

我們可以將現實世界中的操作抽象化，成為圖論中的操作，方便程式預測，下表為現實世界中的操作與圖論中的操作對照表。

|  |  |
| --- | --- |
| 圖論上的操作 | 現實世界的意義 |
| 從圖上的任意一個點出發 | 第一天先隨便點一種菜 |
| 經由任意一條邊 | 經過了一天 |
| 到達圖上的另外一個點 | 第二天點了另外一種餐 |
| 經過一個自環 | 經過了一天，決定要吃同一種菜 |
| 到了同一個點 | 第三天點了同一種菜 |

將上述的對照表，用數學語言描述的話，就是：「將每一種餐點視為圖中的集合，稱作；甲生昨天點了*A*種餐，今天點了*B*種餐，視為在圖中由到的有向邊。」為一張有向圖，的鄰接機率矩陣稱為，定義的穩定狀態為。

根據穩定狀態定義，顯然只有一組解或是有無限多組解，如果僅存在一組解，則解穩定狀態；如果有無限多組解，則用快速冪來計算平均狀態。

|  |  |  |
| --- | --- | --- |
|  | 解穩定狀態 | 求平均狀態 |
| 優點 | 能夠算出實際穩定狀態 | 保證有輸出值 |
| 缺點 | 不保證有輸出值 | 只能得到近似狀態 |

#### 解穩定狀態

根據穩定狀態定義，為階方陣，下式顯然。

先耗費的時間對進行分解，即可在的時間內得到的值，進而判斷解是否唯一；如果唯一，則使用矩陣在時間內求出的值；如果不唯一，則丟出例外狀況。

#### 2. 求平均狀態

根據下列二式，不難發現只需要的時間內就能得知的值。

下式顯然成立。

根據下式，我們只需要的時間就能求出的值。

雖然不滿足，但是代表了長期來看的平均狀態，實際上應用的效果會比任意選取一個穩定狀態還要好，所以我們採用取長期狀態的平均值。

### (二)、數量模型

設計數量模型旨在於「預測總餐數會有多少份」，數量模型的核心為*logistic*模型，我們對每個人建立一個*logistic*模型，再使用統計演算法求出總餐數約有多少份。

模型的輸入長怎樣，模型的輸出長怎樣。

#### 1. *logistic*模型

*logistic*模型最主要的函數為函數，如下。

輸入一個向量，模型即輸出一個值，其中為代表模型的向量。

我們使用*Cross Entropy*方法來建立損失函數，如下。

顯然損失函數越大，該模型精確度就愈高，與皆已給定，我們需要找出一個使得函數值最大，找出的演算法將於後續提及。

若輸入，不論如何模型都將輸出0.5，這樣的模型不是我們想要的，所以我們需要一個常數來修正模型，其中為一個常數，將模型修正如下。

若則，為維向量，使用這個方法即可找出模型的常數項。

#### 2. 統計演算法

我們想要知道有幾個人點餐的機率最高，我們將甲生點餐的事件寫成，甲生不點餐的事件寫成，總共有個人點餐的事件寫成。

枚舉每一個人點或是不點餐的狀態，再將所有機率加總，其演算法的時間複雜度為，不甚理想，我們需要對演算法優化。

我們使用*DP*來進行優化，優化之後只需要的時間複雜度，效率大幅提升，以下是*DP*的虛擬碼。

|  |
| --- |
| *Algorithm: Dynamic Programming Sum* |
| float dp[N] ,tmp[N] ,odd[N];  dp[0] = 1 - odd[0] ,dp[1] = odd[0];  for(int i = 1;i != N;i++) {  for(int j = 0;j != N;j++)  if(j == 0) tmp[j] = dp[0] \* (1 - odd[i]);  else tmp[j] = dp[j] \* (1 - odd[i]) + dp[j - 1] \* odd[i];  for(int j = 0;j != N;j++) dp[j] = tmp[j];  } |

演算法所輸出的*dp*陣列將類似常態分佈，如下圖。

根據圖表，我們可以看到模型顯示區間[87 ,87]的機率有87%。

## 七、單層類神經網路之優化

我們在數量模型中使用*Logistic*模型，等同於一個以*Sigmoid*作為激勵函數的神經元，我們將先證明數學性質，再撰寫優化演算法的虛擬瑪。

### (一)、數學性質

損失函數如下，其中與皆已給定。

|  |
| --- |
| 引理一：若不全為零，則的海森矩陣*(Hessian matrix)*於主對角線上恆負。 |

*Proof.*

顯然。

|  |
| --- |
| 引理二：作為橫軸，作為縱軸，該圖最大值存在且唯一 |

*Proof.*

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

若不全為零，根據上表以及勘根定理得知一階導函數必至少有一根，並根據引理一，不難發現一階導函數僅存在一根，且最大值必存在且唯一。

|  |
| --- |
| 定理一：任意點上對的梯度直線作為橫軸，作為縱軸，該圖最大值存在且唯一 |

*Proof.*

將所有滿足梯度為的點視為一個集合，任意一點的梯度必朝向中最接近的點，模型常數項使得不全為零，根據引理二定理成立。

|  |
| --- |
| 定理二：梯度向量長度越短越接近頂點。 |

*Proof.*

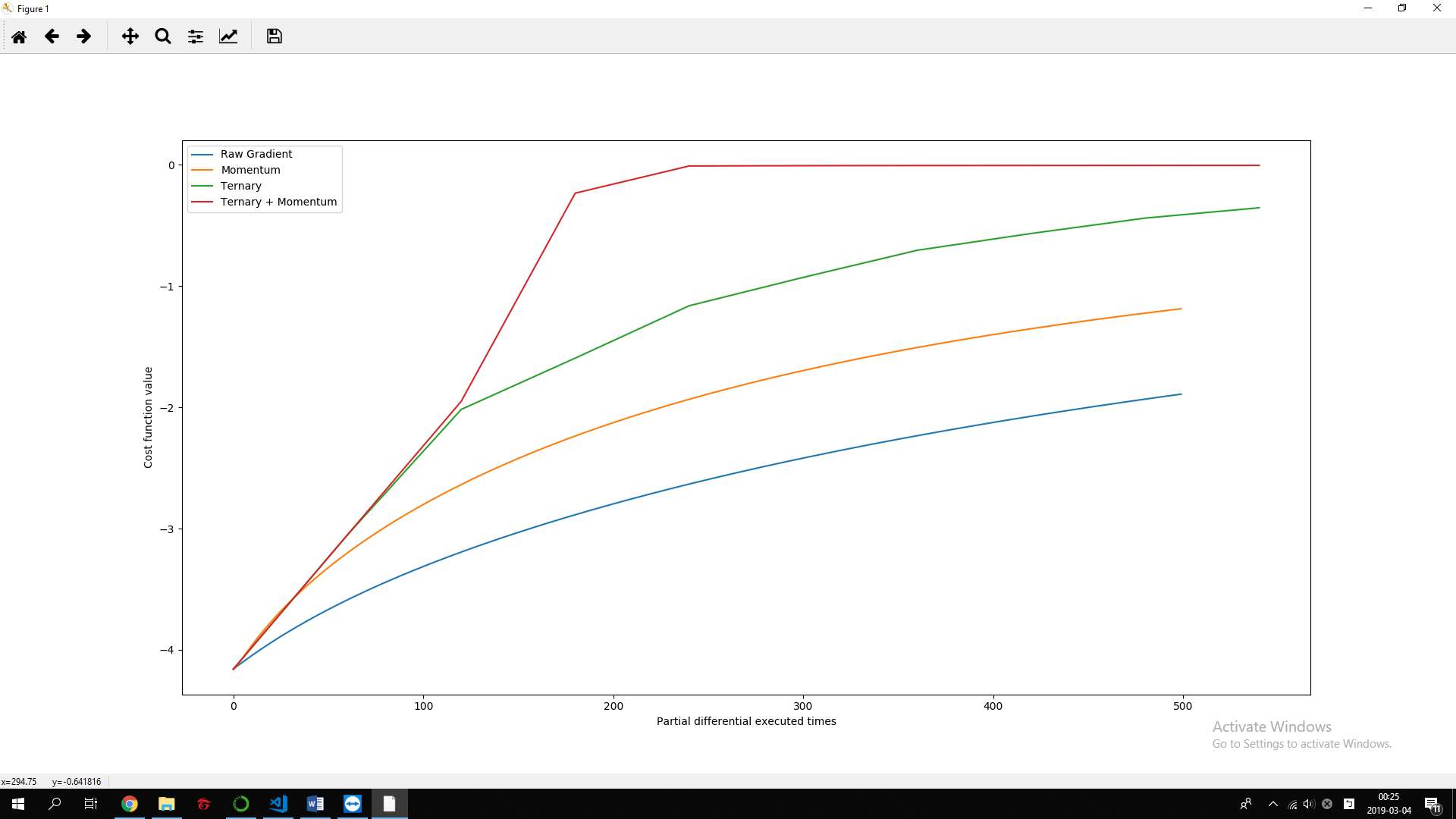
根據引理一，顯然。

### (二)、演算法

三分搜尋法能夠以的時間複雜度的找出一個凸函數的最大值，根據定理一不難發現「對梯度直線及損失函數值三分搜」的演算法是可行的；求向量長度所需的演算量比計算損失函數值還要少，根據定理二得知「對梯度直線及梯度向量長度三分搜」的演算法是可行的。

可能包含局部最小值，動量法的算法特性被視為能夠有效解決局部最小值問題，我們以三分搜尋法搭配動量法進行梯度上升，下述為演算法的虛擬碼。

|  |
| --- |
| *Algorithm: Ternary + Momentum + Gradient* |
| int count = 10 ,ternary = 20;  vector<float> w = 0, grad = gradient(w), prev = 0;  float l ,r ,lmid ,rmid ,alpha = 8, beta = 0.1;  while(count--) {  for(int i = 0 ,l = 0 ,r = alpha;i != ternary;i++) {  lmid = length(gradient((l + l + r) / 3));  rmid = length(gradient((l + r + r) / 3));  if(lmid == rmid) continue; //reaches the maximum precision of float  else if(lmid < rmid) l = (l + l + r) / 3;  else if(lmid > rmid) r = (l + r + r) / 3;  }  tmp = grad \* (l + r) / 2 + prev \* beta;  w = w + tmp;  prev = tmp;  } |

觀察上圖得知三分搜尋法搭配動量法進行梯度上升可以得到最好的效果，

容錯能力

三分搜保證損失函數遞減

# 伍、研究結果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 登入 | 點餐 | 查看點單 | 繳款 |
| 安卓前端 | C:\Users\lawre\AppData\Local\Microsoft\Windows\INetCache\Content.Word\52995536_341576016452924_4492182019454795776_n.jpg | C:\Users\lawre\AppData\Local\Microsoft\Windows\INetCache\Content.Word\53020164_327182224573781_1994272908897157120_n.jpg | C:\Users\lawre\AppData\Local\Microsoft\Windows\INetCache\Content.Word\52859457_560714564433881_1700737264587374592_n.jpg |  |
| 蘋果前端 | C:\Users\lawre\AppData\Local\Microsoft\Windows\INetCache\Content.Word\52785773_2010996985860581_7006054668753174528_n.jpg |  |  |  |
| 網頁前端 | C:\Users\lawre\AppData\Local\Microsoft\Windows\INetCache\Content.Word\52877884_1913323308776129_3492376395303092224_n.jpg | C:\Users\lawre\AppData\Local\Microsoft\Windows\INetCache\Content.Word\53509579_540146379724500_6811099317139406848_n.jpg |  |  |

講一下關於分析器的東西

講一下關於演算法的東西

# 陸、討論

安全性、效能

會不會點餐較可預測，會點什麼餐全憑機運，以及為何要分類

未來展望

類神經網路常常使用*Sigmoid*、*Tanh*以及*Relu*作為激勵函數，

# 柒、結論

讓我過，讓我上大學，我要瘋了

# 捌、參考資料及其他

Logistic regression: <https://blog.csdn.net/SzM21C11U68n04vdcLmJ/article/details/78221784>

Ternary Search: <https://en.wikipedia.org/wiki/Ternary_search>

Php

Bootstrap

Mysql

Kotlin

Android

Elementary Linear Algebra: <https://www.books.com.tw/products/0010682939>

Php + Mysql Source Code:

Factory Client Source Code:

Android App Source Code:

iOS App Source Code: