午餐系統及分析

目錄

[摘要 1](#_Toc2894673)

[壹、研究動機 1](#_Toc2894674)

[貳、研究目的 1](#_Toc2894675)

[參、研究設備及器材 2](#_Toc2894676)

[肆、研究過程及方法 3](#_Toc2894677)

[一、開發過程 3](#_Toc2894678)

[一、系統使用簡介 4](#_Toc2894679)

[三、資料庫 6](#_Toc2894680)

[(一)、Deadlock 6](#_Toc2894681)

[(二)、InnoDB Locks 7](#_Toc2894682)

[四、後端 7](#_Toc2894683)

[(一)、Session阻塞 8](#_Toc2894684)

[(二)、資料庫存取阻塞 8](#_Toc2894685)

[五、前端 9](#_Toc2894686)

[(一)、比較、使用分布 9](#_Toc2894687)

[(二)、Firebase、Google Analytics 10](#_Toc2894688)

[(三)、iOS / Android App 11](#_Toc2894689)

[(四)、網頁前端 12](#_Toc2894690)

[六、預測模型 13](#_Toc2894691)

[(一)、比例模型 14](#_Toc2894692)

[(二)、數量模型 16](#_Toc2894693)

[七、單層類神經網路之優化 18](#_Toc2894694)

[(一)、數學性質 18](#_Toc2894695)

[(二)、演算法 19](#_Toc2894696)

[伍、研究結果 20](#_Toc2894697)

[一、模型的準確度 21](#_Toc2894698)

[二、演算法的效能比較 22](#_Toc2894699)

[陸、討論 23](#_Toc2894700)

[一、安全性與效能如何? 23](#_Toc2894701)

[二、為什麼這樣建立模型? 23](#_Toc2894702)

[三、為什麼不預測餐點細項? 23](#_Toc2894703)

[柒、結論 24](#_Toc2894704)

[一、未來展望 – 推廣演算法 24](#_Toc2894705)

[二、未來展望 – 飛機餐 24](#_Toc2894706)

[三、未來展望 – 建立模型細項 24](#_Toc2894707)

[捌、參考資料及其他 25](#_Toc2894708)

# 摘要

學校原本使用紙本點餐單來訂購午餐，十分不便，為了協助同學訂購午餐，我們在學校建立一套線上訂餐系統，讓同學能夠線上點餐，不再為了訂午餐傷腦筋；每天系統會有大量的資料湧入，可以做數據分析，告訴廠商明天大概要準備多少餐，減少備料的浪費；製作模型時，意外發現可以用三分搜尋法加速演算，於是我們將這個演算法應用於模型中。

# 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點餐數量預測模型，可以給廠商做為明天需要準備多少份餐點的依據。

# 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預測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，預測廠商明天要準備多少餐，這是本研究的第二目標；我們意外發現對於單層的類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 |
| *Google Firebase* | *App*分析用的第三方插件 |
| *Google Analytics* | 網頁分析用的第三方插件 |
| *Ubuntu 18.04 LTS* | 伺服器作業系統 |
| *Nginx* | 網頁伺服器 |
| *No-IP Dynamic DNS* | 伺服器網域 |
| *Mac* | *iOS App*的開發環境 |
| *iPhone* | *iOS App*的測試環境 |
| *Xcode* | *iOS App*的開發工具 |
| *Android*手機 | *Android App*的測試環境 |
| *Android Studio* | *Android App*的開發工具 |
| *Chrome* | 網頁前端測試環境 |
| *Visual Studio* | 開發廠商前端的開發工具 |
| *Visual Studio Code* | 開發後端的開發工具 |
| *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* | 資料庫自動備份 |

# 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了，於是我們設計了一個預測模型。

## 一、開發過程

開發軟體必須要先釐清需求，我們先與合作社、廠商洽談，找出所需的功能，以及訂出完工期限，並請校方提撥經費購買伺服器。

首先設計資料庫，再做出*PHP*後台，此時*JSON*交換介面已經定型，可以開始製作前端。網頁前端與*iOS*前端同時進行開發，網頁前端先行發布，再發布*iOS*前端，最後發布*Android*前端。

開發模型時，我們先以*Python*設計模型，確認模型可行，再將模型改寫成*C#*，並將模型置入廠商管理插件。因為*Python*語言特性，使得修改模型時特別方便，而且多數模型相關的文獻皆以*Python*撰成，先以*Python*作為模型雛形不外乎是個好方法。

## 一、系統使用簡介

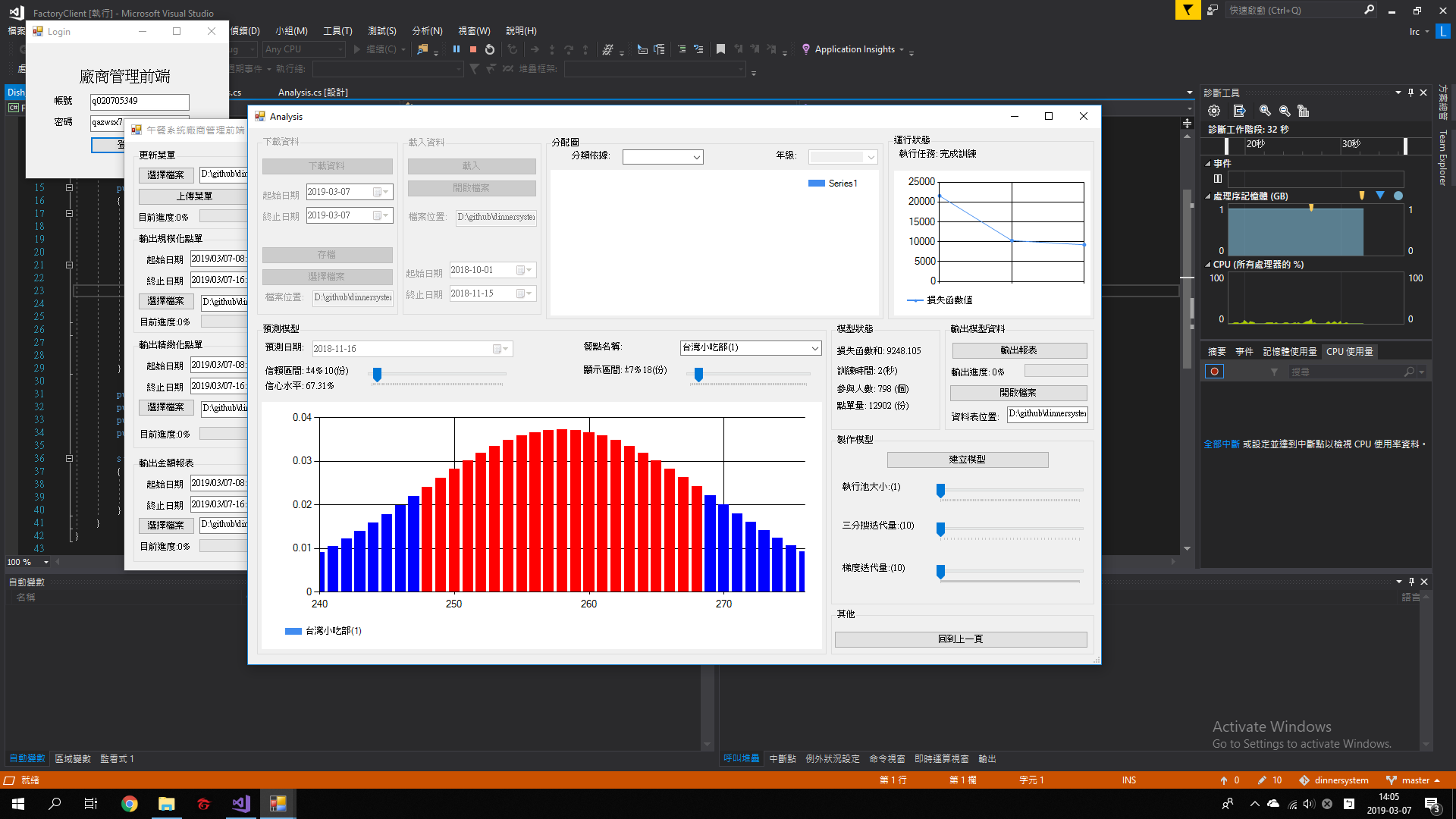
下表為實際使用系統點餐的擷圖，以下簡介以下四種最常被用到的功能。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 登入 | 點餐 | 查看點單 | 繳款 |
| 安卓前端 |  | C:\Users\lawre\AppData\Local\Microsoft\Windows\INetCache\Content.Word\53020164_327182224573781_1994272908897157120_n.jpg | C:\Users\lawre\AppData\Local\Microsoft\Windows\INetCache\Content.Word\52859457_560714564433881_1700737264587374592_n.jpg |  |
| 蘋果前端 |  |  |  |  |
| 網頁前端 |  | 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。

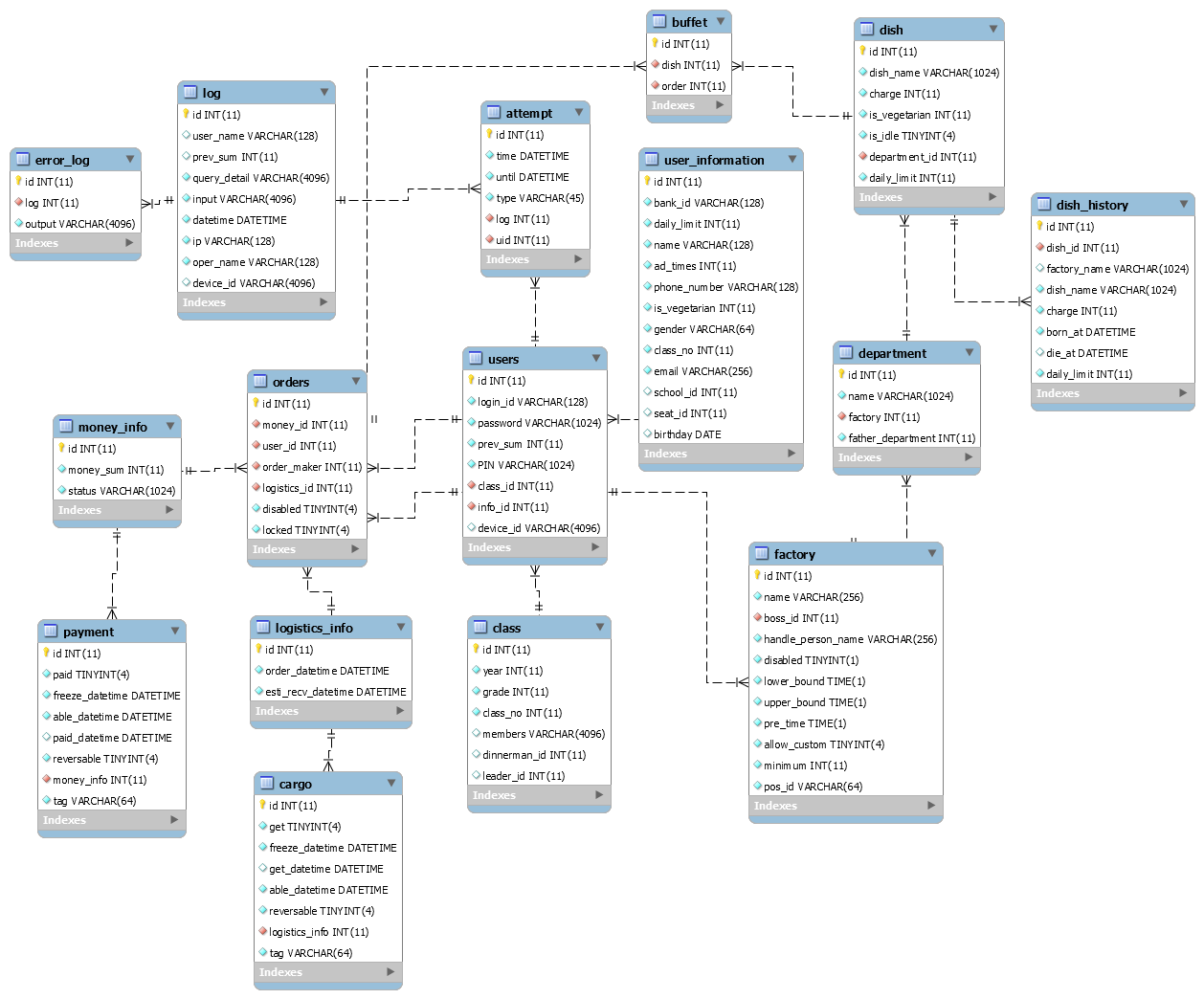
|  |  |  |
| --- | --- | --- |
|  | 廠商管理插件 | 匯出表格 |
| 更新菜單 |  |  |
| 查看點單 |  |  |
| 查看預測模型 |  |  |

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，並不是單線程的環境，常常會遇到死結。一個*Procedure*(作業程序)中包裝了多個語句*(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 Transaction*、*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*為了保證執行序安全*(thread safe)*，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執行，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

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。

### (一)、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。

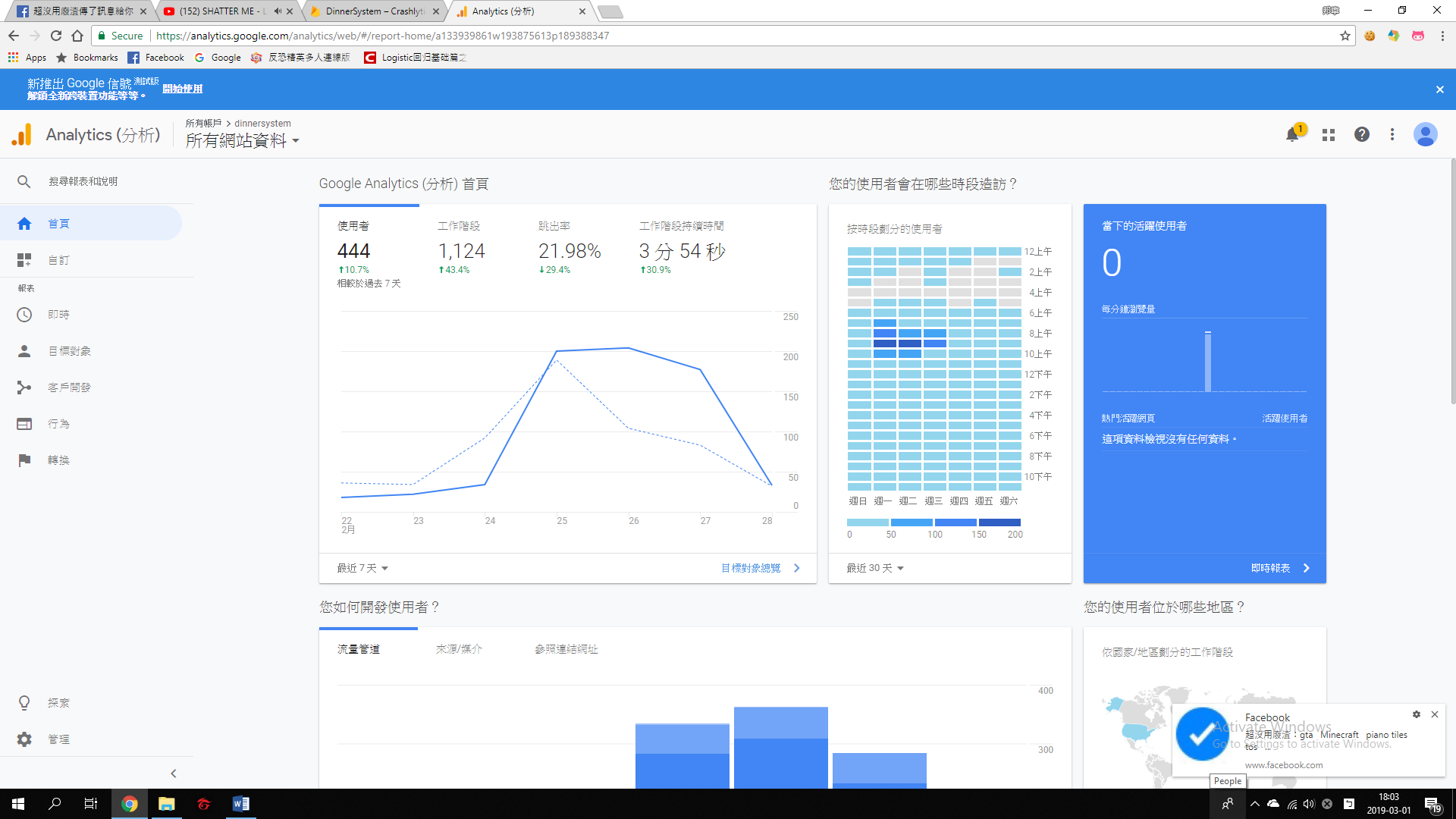
|  |  |  |  |
| --- | --- | --- | --- |
|  | 特色 | 優勢 | 劣勢 |
| 網頁版 | 最早推行的前端 | 不限制於作業系統 | 僅支援主流瀏覽器  不支援自助點餐 |
| *Android*前端 | 具有良好的防呆機制  介面較美觀 | 支援自助點餐 | 開發耗時較久 |
| *iOS*前端 | 具有良好的防呆機制  介面較美觀 | 支援自助點餐 | 開發耗時較久，上架審查機制嚴格 |

### (二)、*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，也方便我們統計使用尖峰。

|  |  |
| --- | --- |
| *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，導致部分使用者受到當機影響。

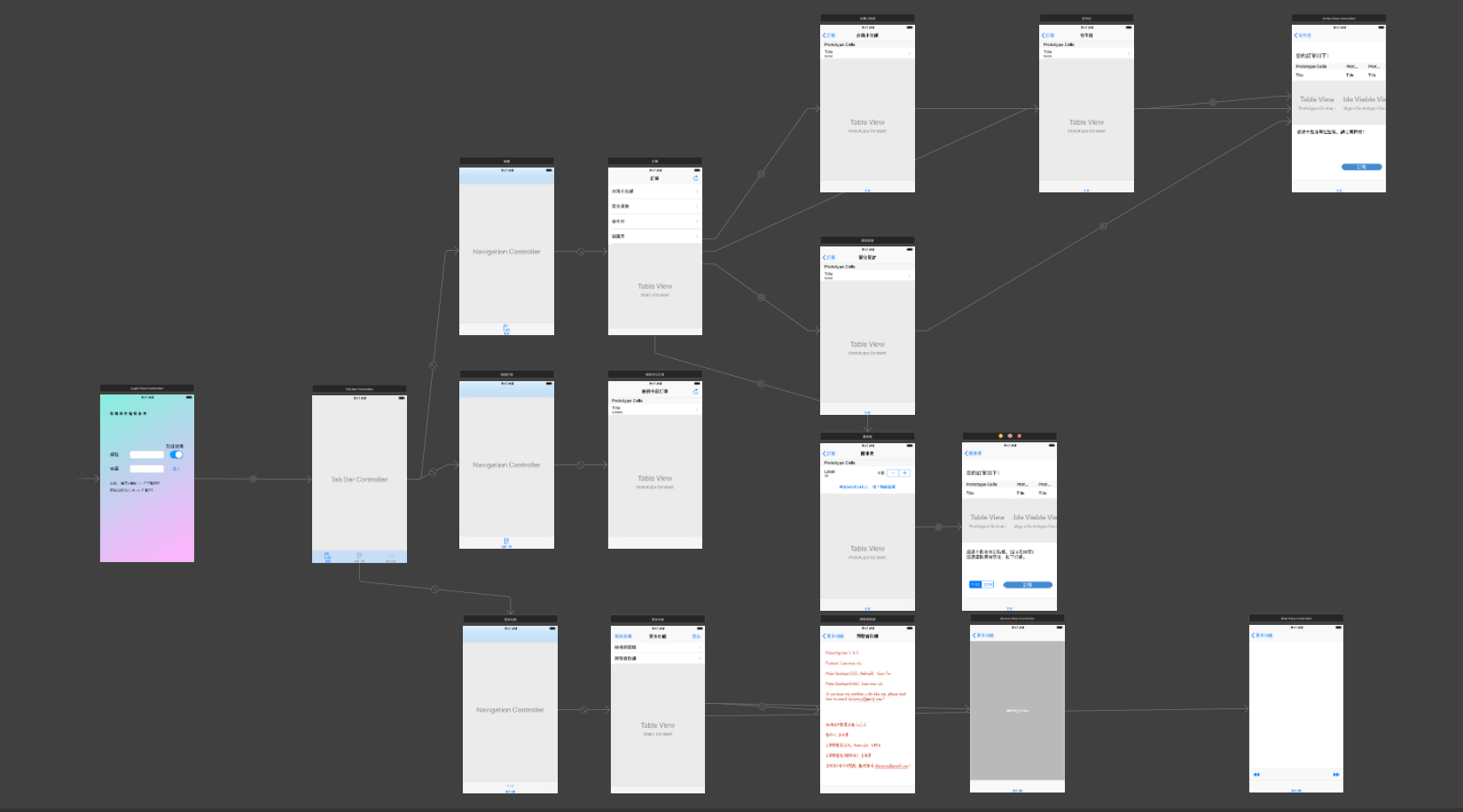


上圖為*Google Analytic*s的輸出報表，2/25人數上漲是因為當天星期一，學校需要訂購午餐；2/28人數下跌是因為當天放假，所以學校不需要訂餐。

### (三)、*iOS / Android App*

*Android App*與*iOS App*的架構大同小異，不過需要將同一個架構撰成兩種不同的程式語言，並放到兩種不同的平台上運作，下圖為架構圖。

下圖為*iOS App*的頁面關係圖，每一條線代表頁面之間的轉移。



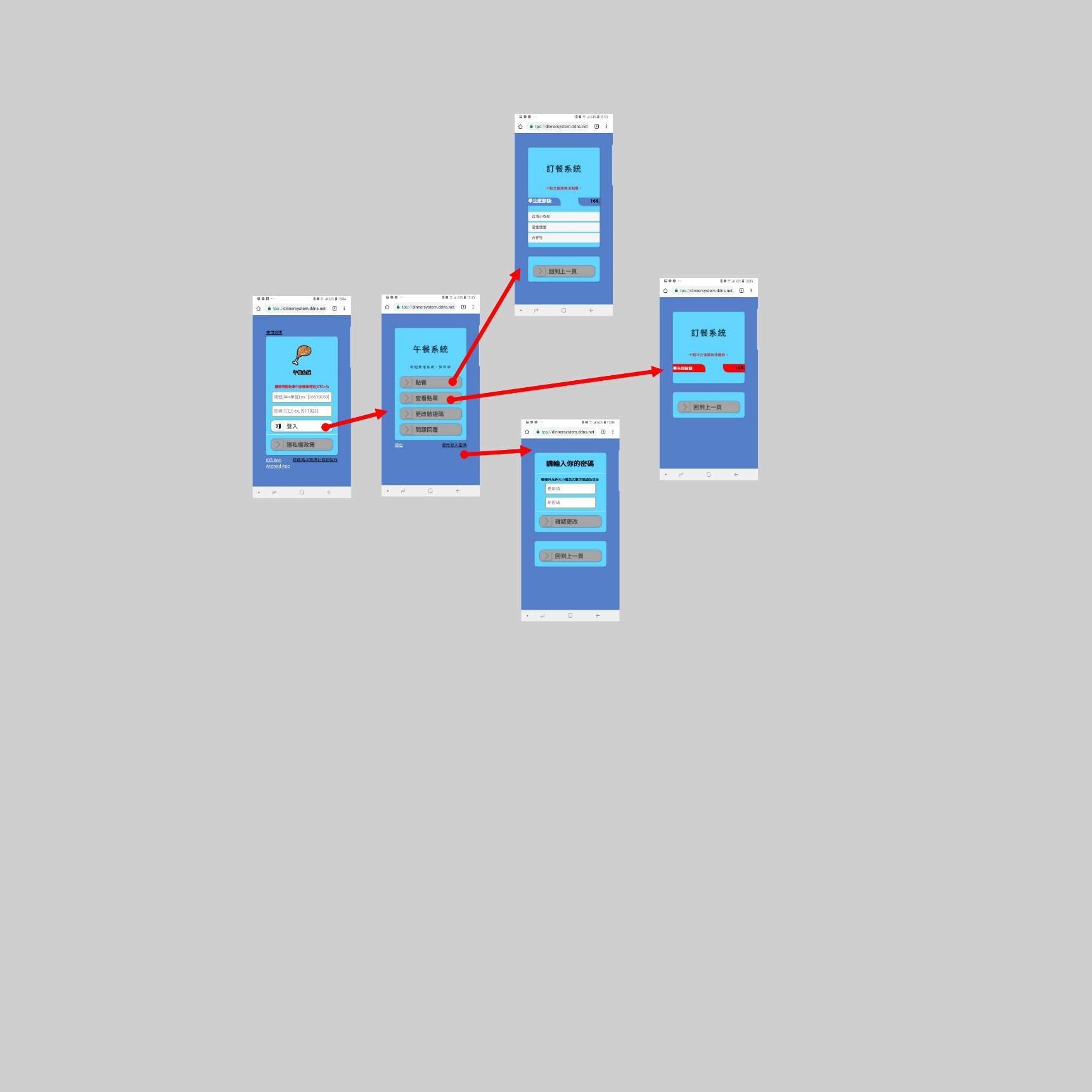
*Android App*與*iOS App*具有良好的防呆功能，而且使用者介面較為美觀；*Android App*與*iOS App*能夠訂購客製點單，也就是像自助餐那樣的點單，而這項功能在網頁版上並未實作。

### (四)、網頁前端

進入網頁版前端後，就有連結引導使用者安裝*Android App*與*iOS App*，希望使用者能夠盡量使用*App*，製作網頁前端的目的在於應急，後台上線後必需要有一個前台，不然系統便不是一個完整的系統，而*App*發生異常時，網頁版前端能確保大家都訂得到便當。

網頁前端的架構圖如下，對於資料量較大的輸入，會先在伺服器雲端處理，再將資料傳給前端展示給使用者。

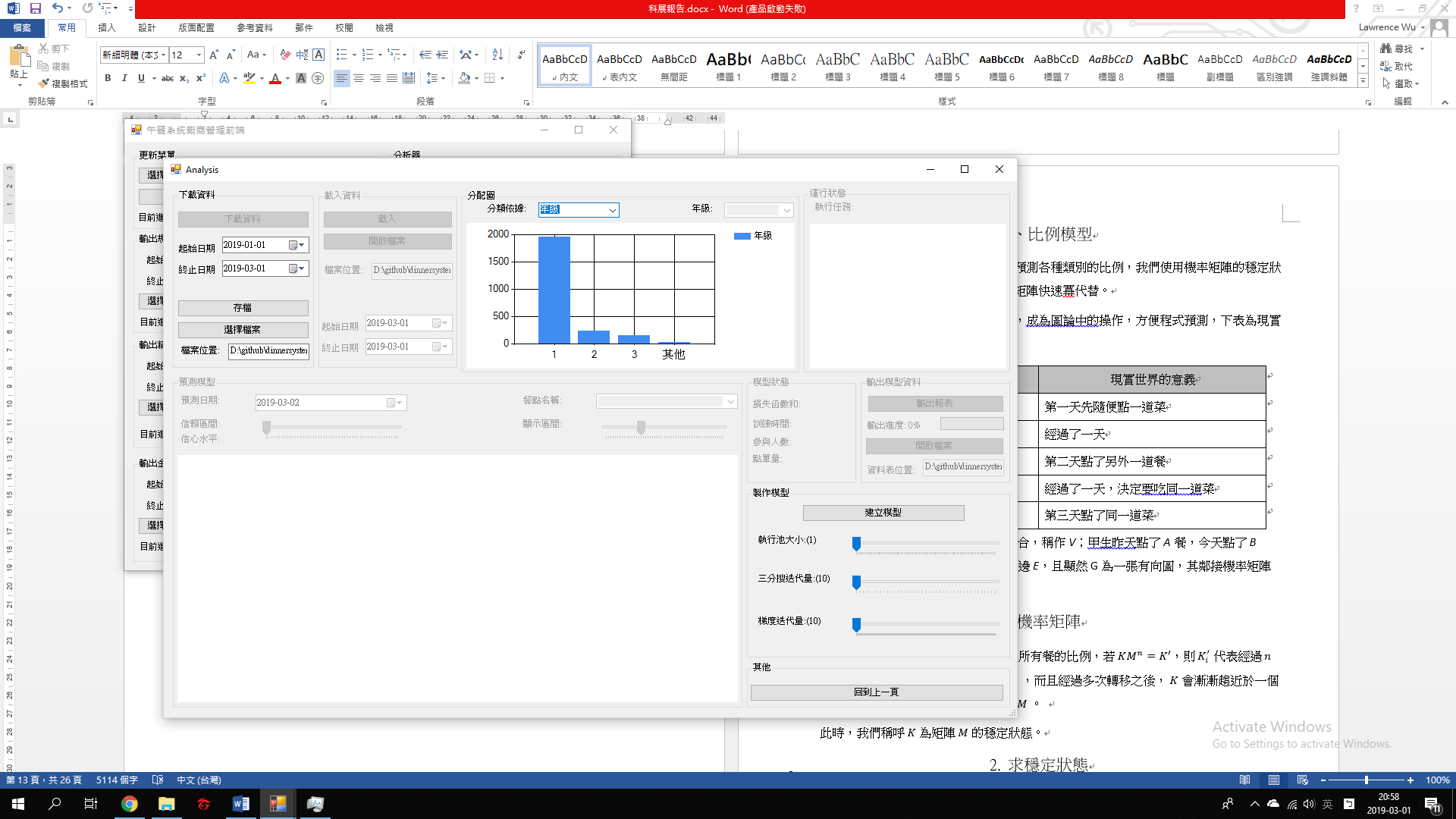
下圖為網頁版的頁面關係圖，每一條線代表頁面之間的轉移。



網頁前端使用了*jQuery*、*Ajax*以及*Bootstrap*作為開發框架，使用*jQuery*的好處在於能夠加快開發，而且不必撰寫冗長的程式碼；使用*Ajax*技術的好處在於不必每次跟伺服器要求資料都必須重新整理一次，能夠直接跟伺服器要求資料，並即時展現給使用者；使用*Bootstrap*的好處在於不必親自撰寫常用元件，能夠直接使用現成的元件。

## 六、預測模型

下表為2019/01/01到2019/03/01的點餐人數分配圖，其中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、高三習慣自己至合作社買便當，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廠商的供應數量。

我們先用*Python*嘗試建立模型，確認模型與演算法可行後，再將模型與演算法改寫成*C#*，加入廠商管理插件，方便廠商使用模型。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建立模型的速度，數學性質以及演算法將於後續提及。

### (一)、比例模型

設計比例模型旨在於「預測各廠商訂購量的比例」，我們使用機率矩陣的穩定狀態來預測甲生點某廠商便當的期望值，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 | 第一天先隨便點某廠商的便當 |
| 經由一條邊 | 經過了一天 |
| 到達圖上的另外一個點 | 第二天點了另外一家的便當 |
| 經過一個自環 | 經過了一天，決定要吃同一家 |
| 到了同一個點 | 第三天了同一家的便當 |

將上述的對照表，用數學語言描述的話，就是：「將每一家廠商視為圖中的節點集合，稱作；甲生昨天點了*A*廠商的餐，今天點了*B*廠商的餐，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

使用下列二式，只需要的時間內就能得知的值，這種演算法叫做快速冪。

下式顯然成立。

使用快速冪搭配上式計算下式，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」，模型的處理流程如下。

先將每個人的點餐資料整理出來，根據下表的模式來轉換資料，交給*logistic*模型預測今天那個人點餐的機率有多少，再將每個人今天的點餐機率交給統計演算法，最終得出總餐數的機率分布。

|  |  |
| --- | --- |
| 解釋 | 模型輸入值 |
| 為模型的訓練輸入參數，。 |  |
| 為模型的訓練輸出函數。 |  |
| 為模型預測今天點餐的機率。 |  |

#### *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。

而損失函數越大，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

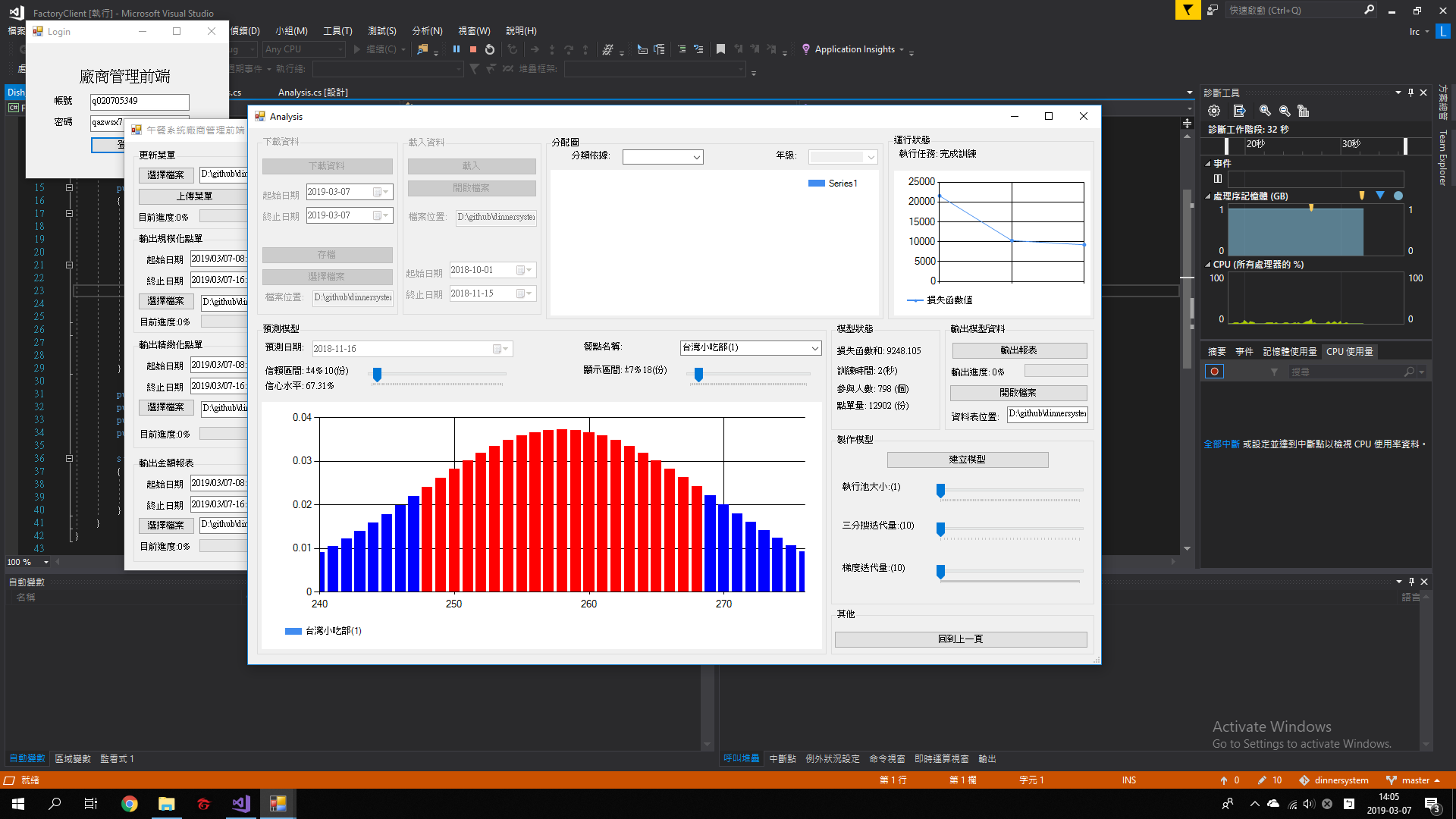
我們想要知道有幾個人點餐的機率最高，我們將甲生點餐的事件寫成，甲生不點餐的事件寫成，總共有個人點餐的事件寫成。

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。



觀察圖表，我們可以得知模型顯示在區間內的機率和有67%。

## 七、單層類神經網路之優化

我們在數量模型中使用*Logistic*模型，等同於一層以*Sigmoid*作為激勵函數的類神經網路，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：函數圖形有最大值且唯一 |

*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，則函數圖形有最大值且唯一。

|  |
| --- |
| 定理一：任意點上的梯度作為橫軸，函數值作為縱軸，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逼近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) break; //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;  } |

# 伍、研究結果

這是午餐系統的實際使用擷圖，下面四項功能是最常被使用的功能。

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

## 一、模型的準確度

下表為各種名詞的解釋，為模型的人數和，為當天點餐人數和，為模型預測廠商會有多少份點單，為廠商實際有多少份點單。

|  |  |
| --- | --- |
| 解釋 | 公式 |
| 差異百分比 |  |
| 平均比例差異 |  |
| 誤差和 |  |

我們任選了五天來展示模型預測值與真實資料的差異性，見下表。

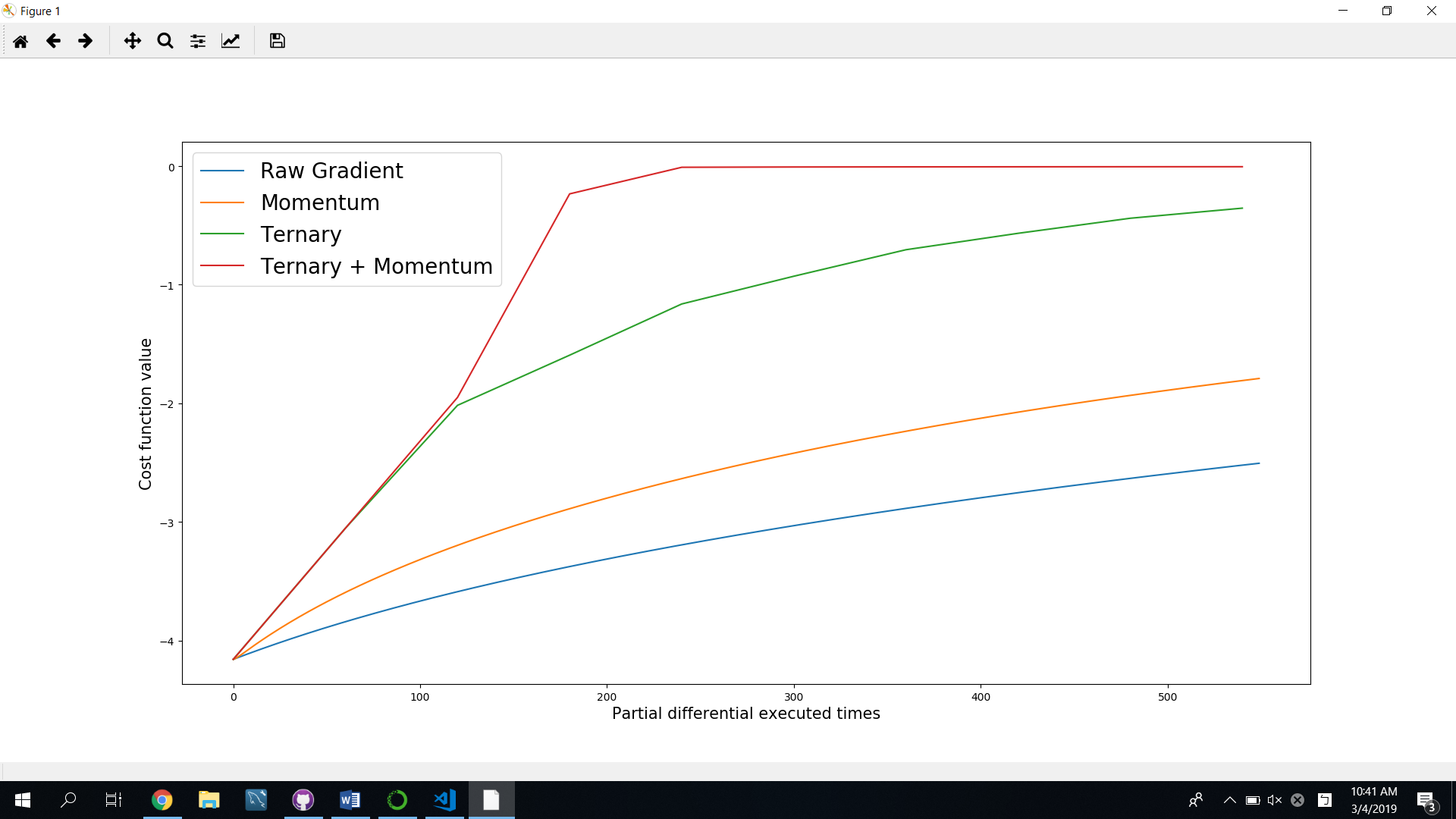
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 數量模型 | | | 比例模型 | 整體模型 |
| 日期 | 預測值 | 實際值 | 差異百分比 | 平均比例差異 | 誤差和 |
| 11/02 | 397(份) | 381(份) | 4.2% | 0.4% | 15(份) |
| 11/05 | 348(份) | 347(份) | 0.3% | 0.2% | 1(份) |
| 11/16 | 312(份) | 288(份) | 8.3% | 3.4% | 24(份) |
| 12/18 | 324(份) | 271(份) | 19.6% | 3.6% | 51(份) |
| 01/08 | 235(份) | 241(份) | 2.5% | 1.9% | 13(份) |

在12/18當天有一個班級出外班遊，導致點單數特別少，一個班級約有四十人，加上四十人後，模型的預測值與實際資料相差不遠。如果沒有遇到特殊狀況，如段考、班遊或隔宿旅行等活動，模型將會有良好的準確度，可供廠商作為參考。

## 二、演算法的效能比較

下表為本實驗中各演算法採用的公式，以及各種參數的意義。

|  |  |  |
| --- | --- | --- |
|  | 公式 | 解釋 |
| 梯度上升 |  | 為梯度上升的參數 |
| 三分搜 |  | 為三分搜循法 |
| 動量法 |  | 為動量衰退率 |
| 動量法 + 三分搜 |  | 為前一次的移動量 |



在過大的狀態下，梯度上升有可能會發散；梯度上升搭配動量法，在過大的狀態下，損失函數可能不減反增；梯度上升搭配三分搜尋法，就算過大，三分搜尋法的算法特性使得損失函數依舊嚴格遞減，不過有可能卡在局部最小值出不來；動量法搭配三分搜尋法，不僅能夠處理過大的，也能夠避免卡在局部最小值。

因為「動量法搭配三分搜尋法」不只能夠處理過大的，還能夠避免卡在局部最小值，所以我們採用「動量法搭配三分搜尋法」作為廠商管理插件中迭代的演算法。

# 陸、討論

## 一、安全性與效能如何?

後台防堵了*SQL Injection*、*XSS*等常見的漏洞，而且輸入有嚴格的限制，確保安全性；我們在*OSI*第七層使用*HTTPS*協定，加強傳輸安全性；後台的效能直接影響了整套系統的效能，我們對於最常見的效能瓶頸「*Session*阻塞」、「資料庫阻塞」進行了優化，使得後台的回應效能增加。

## 二、為什麼這樣建立模型?

小明因為今天早上的課很無聊，所以開始滑手機，突然想到可以訂午餐，而決定要訂購午餐；小王媽媽昨天不用加班，晚上幫小王準備了便當，於是小王今天就不用訂午餐了。課表為一個禮拜循環，多數家長的班表也是一個禮拜循環，於是我們以星期幾作為模型的輸入。

目前有三家廠商，其中有一家專門提供素食便當，他們的客戶群非常穩定；而另外兩家廠商間的差異性並不大，選擇買哪家廠商的便當幾乎是隨機行為。這種行為模式我們以馬可夫模型來預測，而穩定狀態就是會買哪一家便當的機率。

## 三、為什麼不預測餐點細項?

廠商的供應餐點常常變化，到了夏天廠商通常會供應酸辣清爽的餐點，到了冬天廠商通常會供應小火鍋；如果豬瘟來襲，豬肉價格上漲，廠商會改用其他肉類；如果颱風來襲，高麗菜上漲，廠商就會以其他菜色代替。而可能有人特別討厭吃高麗菜，或是有人不喜歡吃雞肉，或是有人喜歡在夏天吃小火鍋，菜色的變動將會影響模型的預測能力，所以我們不預測餐點細項。

# 柒、結論

本作品不只是紙上談兵，而是經過實際上線運作的一套系統；經過實驗證明，本模型具有良好的預測能力；而建立模型時，意外發現的迭代演算法是可行的，經過實測得知有更好的演算效能。使用系統能讓點餐更方便，使用模型能讓廠商減少食材浪費，本作品能提高社會經濟效益，降低社會成本，而且更加環保，如果加以推廣，就能對社會有更大的幫助。

## 一、未來展望 – 推廣演算法

類神經網路常常使用*Sigmoid*、*Tanh*以及*Relu*作為激勵函數，也常常使用*Momentum*、*Adagrad*以及*SGD*演算法，在我們所研究的演算法中，僅探討了*Sigmoid*搭配*Momentum*演算法。未來可以將演算法推廣至各種函數，或是超過一層的神經網路，或是討論與哪一種演算法的結合最好。

## 二、未來展望 – 飛機餐

如果能將系統推廣到長途客機上，使用座位上的螢幕點餐，就能簡化空服員的作業流程，也方便收集乘客的點餐數據。飛機餐是昂貴的，因為要將餐點保存在飛機上的冰箱中，再將餐點加熱才能送出去；飛機上追求品質服務，必須要盡量追求讓乘客吃到自己喜歡的餐點，如果飛機上備料不夠，乘客就容易吃不到喜歡的餐點，如果能將模型應用在準備飛機餐上，一方面能夠減少航空公司的開銷，一方面能夠增加服務品質，一舉兩得。

## 三、未來展望 – 建立模型細項

我們認為天氣、月份、性別以及身高體重等，都有可能與選擇點什麼餐相關，像是天氣熱時，可能就比較喜歡吃涼麵；台灣在夏秋之際容易有颱風，可能導致菜價波動，使得廠商改買其他菜；身高偏高、體重偏重的人可能會買比較大份的便當。如果以這些資料作為輸入，也許就能製作一個細項模型，預測各種餐點會有多少人買。

# 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>

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

Frontend + Backend Source Code: [https://github.com/lawrence910426/dinnersystem /Applications](https://github.com/lawrence910426/dinnersystem/tree/master/Applications)

Factory Client Source Code: [https://github.com/lawrence910426/dinnersystem /Attach/FactoryClient](https://github.com/lawrence910426/dinnersystem/tree/master/Attach/FactoryClient)

Android App Source Code: <https://github.com/seanpai96/dinnersys-android>

iOS App Source Code: <https://github.com/seanpai96/dinnersys-ios>

Momentum Method: <https://blog.csdn.net/tsyccnh/article/details/76270707>

機器學習: <https://activity.ntsec.gov.tw/activity/race-2/2018/pdf/TISF2018-190003.pdf>

系統網址: <https://dinnersystem.com>

iOS App: <https://itunes.apple.com/app/apple-store/id1352943874?mt=8>

Android App: <https://play.google.com/store/apps/details?id=seanpai.dinnersystem>

測試用帳號密碼: dinnersys ,2rjurrru