Machine Learning 2016, Fall, NTU
Final Report

Topic: Outbrain Click Prediction

Team name: NTU_b03901027_寫 code 寫到海枯石爛滄海桑田

(NTU_b03901027_從一開始就雷的美好世界)

Members&Work division:

林恆陞 b03202019： Cyber Security Attack Defender

徐彥旻 b03901027： Cyber Security Attack Defender, report writing

陳景由 b03901152： Transfer Learning on Stack Exchange Tags

蔡丞昊 b03901028： Outbrain Click Prediction, report writing

1. Model Description

我們實作了三種不同的模型：FTRL、BTB、FFM。

(1)Follow-The-Regularized-Leader Proximal：

是 Google 針對廣告點擊率（Click Through Rate）的預測提出的機器學習演算法，演算法如下：

Algorithm 1 Per-Coordinate FTRL-Proximal with L_1 and L_2 Regularization for Logistic Regression

#With per-coordinate learning rates of Eq. (2).
Input: parameters α, β, λ_1, λ_2
$(\forall i \in \{1, \dots, d\})$, initialize $z_i = 0$ and $n_i = 0$
for $t = 1$ **to** T **do**
  Receive feature vector $\mathbf{x}_t$ and let $I = \{i \mid x_i \neq 0\}$
  For $i \in I$ compute

$$w_{t,i} = \begin{cases} 0 & \text{if } |z_i| \leq \lambda_1 \\ -\left(\frac{\beta+\sqrt{n_i}}{\alpha} + \lambda_2\right)^{-1}(z_i - \text{sgn}(z_i)\lambda_1) & \text{otherwise.} \end{cases}$$

  Predict $p_t = \sigma(\mathbf{x}_t \cdot \mathbf{w})$ using the $w_{t,i}$ computed above
  Observe label $y_t \in \{0, 1\}$
  for all $i \in I$ **do**
    $g_i = (p_t - y_t)x_i$ *#gradient of loss w.r.t. w_i*
    $\sigma_i = \frac{1}{\alpha}\left(\sqrt{n_i + g_i^2} - \sqrt{n_i}\right)$ *#equals $\frac{1}{\eta_{t,i}} - \frac{1}{\eta_{t-1,i}}$*
    $z_i \leftarrow z_i + g_i - \sigma_i w_{t,i}$
    $n_i \leftarrow n_i + g_i^2$
  end for
end for

上圖是 FTRL 之 pseudocode，其中，α, β, λ_1, λ_2 為手動調整之參數，x_t 為一筆資料，共有 T 筆，每次更新參數時，會用儲存的 g, σ, z, n 來算出 w，再回去更新 g, σ, z, n。FTRL 的核心是 Online Logistic Regression，但加入了 λ_1 和 λ_2 Regularization。更新每個 x 的 w 時，會參考λ_1的值，因此能夠達到稀疏性；λ_2的加入則是使求解結果更加平滑，且防止 overfitting。

Vowpal Wabbit 專案是快速的非核心 (out-of-core) 學習系統，跟一些其他的在線（ online ）演算法實作相似，就一筆資料來說，可以將十的十二次方的稀疏特徵有效率地應用在機器學習問題上。我們使用 Vowpal Wabbit 來實作 FTRL。

(2)BTB（Beat-The-Benchmark）：

完全不考慮 display_id 與 ad_id 背後所代表的 feature（即不考慮不同 display_id 或 ad_id 彼此之間的相關性），只根據廣告本身的 popularity（自身的點擊率）來做排序。他的概念是，一個廣告會不會點擊，只與他本身成功不成功有關係，而與使用者喜好、網頁的內容等無關。

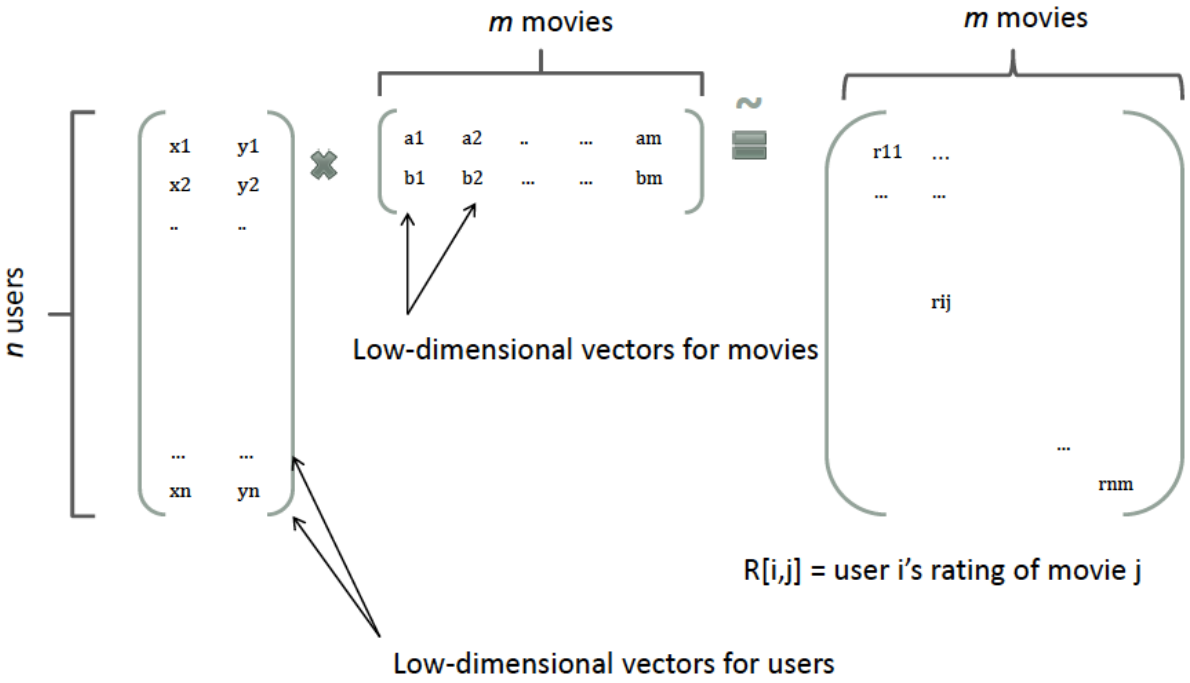
(3)FFM (Field-aware Factorization Machines) ：

在討論 FFM 以前，先簡短了解 FM。FM 的基礎模型是 Linear Regression 與交叉項，表達式如下：

$$y = \omega_0 + \sum \omega_i x_i + \sum\sum \omega_{ij} x_i x_j$$

交叉項的參數共有$\frac{n(n-1)}{2}$個，在此任兩個ω_{ij}都是獨立的。然而在處理稀疏問題時，交叉項的訓練是非常困難的，因為ω_{ij}的訓練需要大量x_i與x_j都非零的樣本；由於數據很稀疏，滿足「x_i與x_j都非零」的樣本將會非常少。訓練樣本的不足，將導致ω_{ij}不準確，最終嚴重影響模型的性能。

為了解決交叉項的學習問題，FM 將ω_{ij}組成的矩陣 W 分解成兩個矩陣相乘：



$y = \omega_0 + \sum \omega_i x_i + \sum\sum \langle v_i, v_j \rangle x_i x_j$，其中$\langle v_i, v_j \rangle = \sum v_{i,f} \cdot v_{j,f}$。交叉項的參數將由 n x n 減少到 n x k，降低了因數據稀疏，導致交叉項參數學習不充份的影響。

並且 FM 的複雜度還可以進一步由 O(nk^2)優化到 O(nk)，提升模型訓練的速度：

$$\sum \langle v_i, v_j \rangle x_i = \frac{1}{2} \sum \left(\left(\sum v_{i,f} x_i \right)^2 - \sum v_{i,f}^2 x_i^2 \right)$$

而在 FFM 中，每一維x_i針對其他特徵x_j都會學習一個v_{i,f_j}：

$$y = \omega_0 + \sum \omega_i x_i + \sum\sum \left\langle v_{i,f_j}, v_{j,f_i} \right\rangle x_i x_j$$

其複雜度為 O(kn^2)。

實作上我使用台大團隊開發的 libffm 這個 open source 工具。這個版本省略了常數項和一次項：

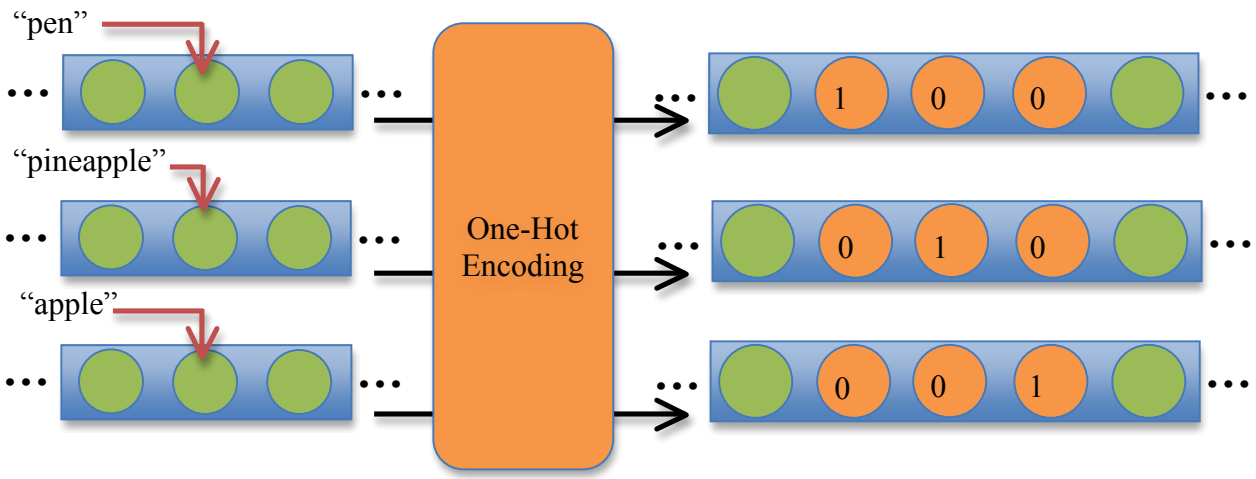
$$\phi(w,x) = \sum \langle w_{j_1,f_2}, w_{j_2,f_1} \rangle x_{j_1} x_{j_2}$$

2. Preprocessing/Feature Engineering

(1) One-Hot encoding

如果有某項特徵是離散的，也就是不在連續的空間上，那麼如果將這個特徵轉為單維的實數的話，那麼原本離散的值就會在這樣的過程中產生彼此距離不一的關係，舉例而言，如果其中一個特徵的宇集為{“apple”，“pineapple”，“pen”}，此時若將 “apple“ 轉換對應成數值 1，將 “pineapple” 轉換對應為數值 2，將 ” pen” 轉換對應為數值 3，就會包含了「 “apple” 與 “pineapple” 的距離」和「 “pineapple” 與 “pen” 的距離」是一樣遠的，這樣的隱含關係在特徵轉換當中。

因此，為了解決這樣的問題， one-hot 會將離散的特徵轉換對應為長度與特徵之宇集元素數 (cardinality of universal set) 的向量， 在對應的維度中為數值 1 ，其他的維度為數值 0。承上一段的例子，特徵宇集為{“apple”，“pineapple”，“pen”}，在 One-hot 的操作中，我們可以將 “apple” 對應到 [1,0,0] ；將 “pineapple” 對應到 [0,1,0] ；將 “pen” 對應到 [0,0,1] 。藉此避免有誰跟誰比較近的隱含關係。這樣的前處理會增加模型的參數，使得特徵變的稀疏(sparse，很多維都是 0 的意思)，但是也讓模型能夠更容易的分出不同特徵的影響，使得模型的表現可能變得更好。



(2)Feature Choosing

BTB 的部份，只取 train 的資料。 FTRL 與 FFM 則選擇性加入 document、ad、display 之背後資料（如 topic、advertiser、uuid 等），詳細內容會在 Discussion 裡說明。

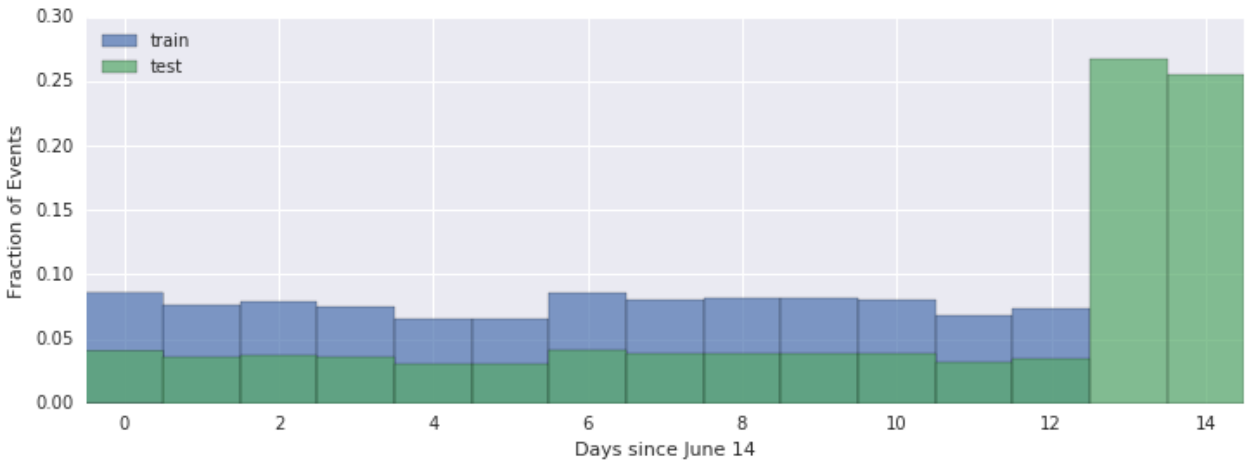
(3)Feature Hash Trick

預先設定好向量的長度，利用 Hash function 將字串轉換為整數，取絕對值後對預先設定好的向量長度取模運算，讓對應的維度設為一。在實作上，我們紀錄有一的維度的 indices ，形成 sparse matrix

3. Experiments and Discussion

(1)Validation :

Validation data 的切法非常重要，由於 Kaggle 每天上傳的次數非常受限，validation data 一定要能夠準確反映出我們模型的準確度。於是我們參考 joconnor 在 Kaggle 的 Kernel 上的文章 *Date Exploration and Train/Test Split*，並依照 test 資料的分佈：50%在最後兩天的模式，將 train 同樣切出 50%在 9~10 天的 validation data。



(2)Feature Choosing

BTB

若僅根據每個廣告的 popularity 來計算點擊率的話，成績大概在 0.63 左右；然而根據 anokas 的文章 *Outbrain EDA*，有 47937415 個 user 會重複點擊同一個頁面多次，因此我們加入「使用者是否看過此廣告」的 feature，若看過則判斷點擊率為 1，也就是一定會點擊，則成績將會大幅提升至 0.65。

FTRL & FFM：

我選擇了 promote_content（包含 document_id、campaign_id、advertiser_id）與 events（包含 uuid、platform、geo_location）。我也嘗試過使用全部的資料（document_topic、document_category 等等），但出來的分數就會變得非常低（大約在 0.5 左右），推測是由於多出來的 feature 會成為干擾項，使模型的性能變差。其實由 Kaggle 在 document_topic 等資料給出的 confidence_level 都非常低就知道，這些資料可能不太準確。

(3)Feature Engineering

在 One-Hot Encoding 中，理想上我們應該要讓每一個 vector 的長度都越大越好，如此在 hash 的時候才能夠盡量減少不同 feature 但出現在 vector 同一個位置的情況。然後受限於電腦的配置，我們在這一塊沒有辦法處理的很好。

(4)Model Evaluation：

BTB

作為 data 量非常少、想法簡單的 model，一開始看到我非常嗤之以鼻，然而結果卻異常地驚人，比之 FTRL 與 FFM 也不遑多讓，而且訓練模型所需要的時間非常少（5~30 分鐘），Kaggle 的最高分數為 0.65251。

FTRL

使用 vowpal wabbit 加速後，耗時大約在 2~3 小時，Kaggle 的最高分數為 0.65551。

FFM

若不計算資料前處理（耗時約 2~3 小時），使用 libffm 工具耗時約 1 小時左右，Kaggle 的最高分數為 0.66639。