Outbrain Click Prediction

Machine Learning Final Project

2016 Fall

Team 1 丁讚讚讚

江東峻 R05922027　陳翰浩 R05922021

傅彥禎 R05922006　李承軒 R05922032

1 Problem Description

Outbrain Click Prediction[1] 是一個在 Kaggle 上，由 Outbrain 發起、預測廣告點擊率 (Click Rate) 的比賽。Training data 及 testing data 由 display_id 和 ad_id 組成，有著相同 display_id 的 ad_ids 為一個 group，我們的目標是將一個 group 內的 ads 依照點擊率「排序」。在 training data 中，一個 group 裡只會有一個 ad 被點擊。除了 training data 及 testing data 以外，還有許多額外的資訊，例如：promoted_content 是有關 ad 的資訊，而 event 是有關 display_id 的資訊，page_view 是有關使用者瀏覽的資訊，documents_* 是各種關於 document 的資訊，這裡的 document 是指網頁本身，也就是一個網頁 (html) 代表一個 document。

2 Models

2.1 Factorization Machine

Factorization Machine(FM)[2] 是一個被廣泛用在 CTR Prediction 的 model，Eq. 1 為 FM 的 objective function，L 是 $(display_id, ad_id)$ 的數量，λ 為 regularization parameter，這個 model 最佳化的目標是 0/1 的 class (y)，使用的 loss function 是 logloss (當然也可以是其他的 loss function)，且用 l2 loss 做 regularization。其中特別的地方是它的 kernel(Eq. 3，FM 的 kernel 強調的是特徵之間的互動，例如：同時出現特徵 1 以及特徵 2 時所代表的意義，因此，kernel 的算法是將各個特徵所對應到的 latent vector 兩兩內積加總。FM 的好處是它收斂很快，很適合處理 CTR 這種非常 sparse 的資料。Fig. 1 為 FM 的圖示，T 為特徵的種類 (num_field)，x_1, x_2, ..., x_T，為各特徵的 index，表示特徵 $x_i = 1$，接著用這些 indices 去 embedding table 中找到對應的 latent vectors，接著再將這些 vectors 兩兩內積加總 (interaction)，最後使用 logloss 作為 loss function。在最佳化時，使用 mini-batch 以及 Adagrad 演算法。

FM 需要存一個 embedding table，大小為 max_features * dim，這是這個 model 中消耗記憶體最多的地方。

$$\min_{\mathbf{w}} -\sum_{i=1}^{L} \left(y_i \log p + (1 - y_i) \log(1 - p) + \frac{\lambda}{2} ||\mathbf{w}||^2\right) \tag{1}$$

$$p = sigmoid(\phi(\mathbf{w}, \mathbf{x})) \tag{2}$$

$$\phi(\mathbf{w}, \mathbf{x})) = \sum_{j_1, j_2 \in C_2} < \mathbf{w}x_{j_1}, \mathbf{w}x_{j_2} > \tag{3}$$

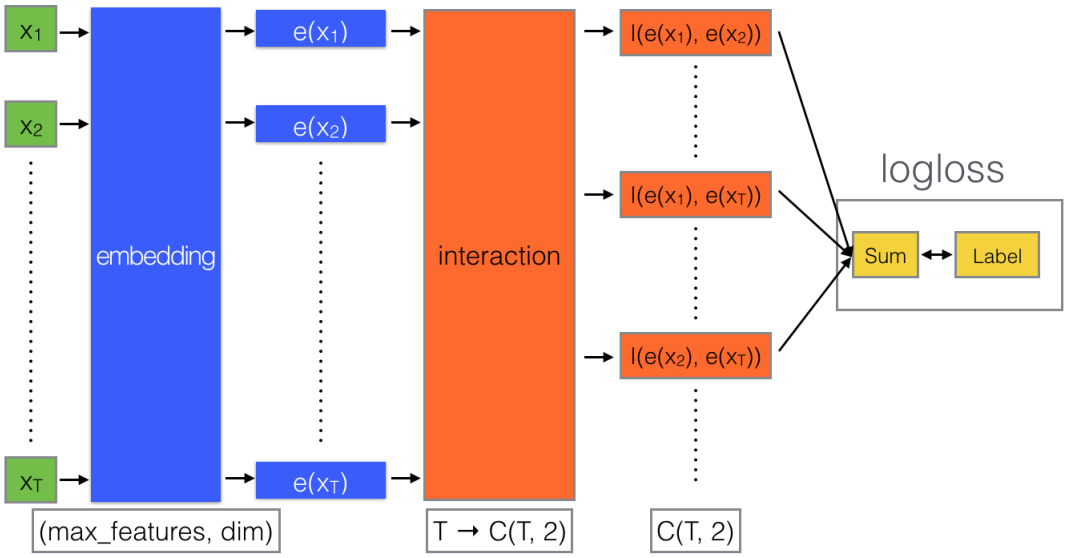

Figure 1: Illustration of Factorization Machine

2.2 Weighted Factorization Machine

2.2.1 Weighted Factorization Machine

在 FM 中，interaction 後得到的那些值是用 uniform 加總。但是，每個 interaction 應該要有不同的重要性，說明哪些種類的特徵之間的互動比較重要，而哪些是比較不重要的。為了模擬這種 interaction 的「重要性」，我們加了一組 weights，將原本的 uniform sum 改成 weighted sum，Fig. 2是 Weighted FM 的圖解，與 FM 的差異只有最後的 weighted sum，而這組 weights 一樣是透過 training 來得到的。

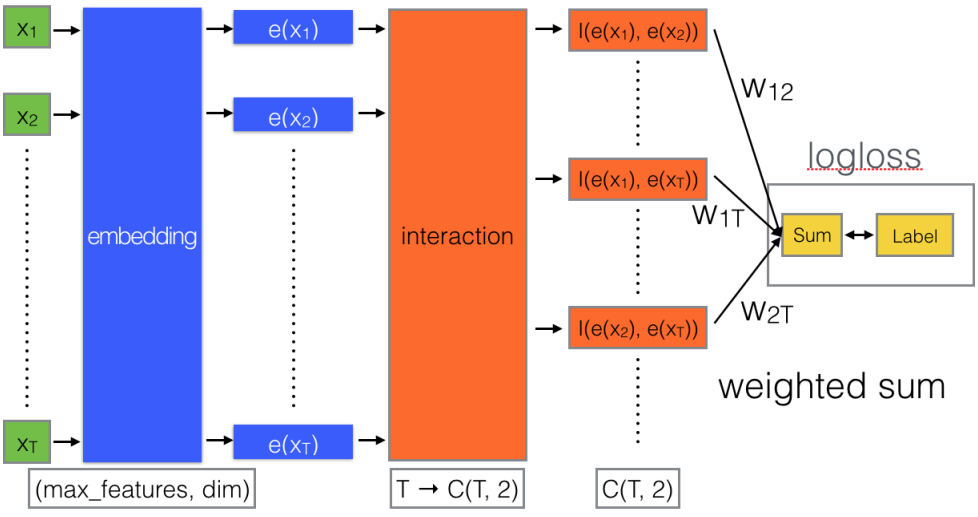

Figure 2: Illustration of Weighted-FM

2.2.2 Factorization Machine Neuron Network

在 Weighted FM 中，我們加了一組 weighted 來表示 interactions 的重要性，但是這些重要性也許不只有一種可能，對於不同的面向來說，重要性可能是不一樣的，例如：有些人看廣告可能比較喜歡看廣告的文字，有些人則是看縮圖。為了模擬這種重要性的多樣性，我們多加了幾組 weights，而這種概念就像是 Neuron Network，我們不只可以增加 weights，還可以增加 layer 數，甚至一些 Neuron Network 的技巧，例如:activation function、dropout 等，也可以引入到這個 model 中，我們稱這個 model 為 Factorization Machine Neuron Network(FM-NN)，Fig. 3為這個 model 的圖示，其中 hidden layer 為 1 層，neuron 數為 2。

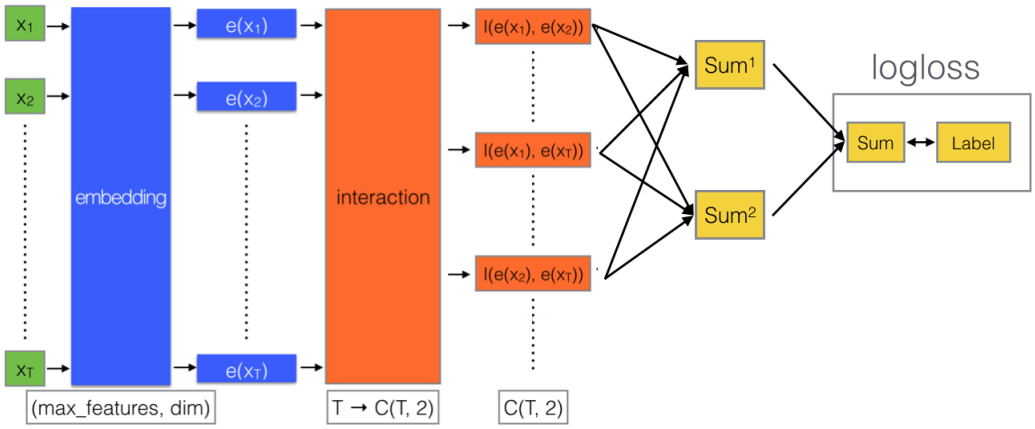

Figure 3: Illustration of Factorization Machine Neuron Network

2.3 Field-aware Factorization Machine

Field-aware Factorization Machine(FFM) 是 CTR 中 state-of-the-art 的 model，在 2016 年由 YuChin Juan 等人提出 [3]，並使用 FFM 贏得了許多 CTR 的比賽。
FFM 與 FM 的差異在於 kernel，FM 認為一個特徵有一個 latent vector，但是 FFM 認為一個特徵對每個 field 都各自有一個 latent vector，field 指的是特徵的種類，例如：ad_id、ad_doc_id、display_doc_id 等，在 FFM 的架構下，每個特徵都有 num_field 個 latent vectors，在做 interaction 時，每個特徵要使用對應 field 的 vectors，例如：ad_id 19 與 display_doc_id 30 互動，「19」必須找到屬於「display_doc_id」的 vector，而「30」必須找到屬於「ad_id」的 vector，然後再做內積後加總。Fig. 4為 FFM 的圖示，特徵 x_i 在通過 embedding table 拿到的是一個 num_field * dim 的 matrix，而 $e(x_i)[field]$ 代表 x_i 的 matrix 中屬於 $field$ 的 latent vector，最後一樣是內積後加總。由於構造與 FM 相似，FFM 一樣可以做成 Weighted FFM 或 FFM-NN 來 model 到各個 interaction 的重要性。
FFM 的 embedding table 與 FM 不同，FFM 必須要存每個特徵對應到其他 field 的 vector，所以 embedding table 的大小會是 max_features * num_field * dim，cost 相當高，因此 dim 相對不能太高。

2.4 List-wise Framework

在 outbrain click prediction 的 dataset 中，每個頁面 (display_id) 只會有一個廣告被點擊，而我們要輸出的是廣告被點擊的機率的排序。而上述介紹的都是 element-wise 的架構，它 model 的是給定一組頁面及廣告，這個廣告被點擊的可能性。但是事實上，一個頁面有許多廣告，而有可能這個頁面中不只一個廣告有著很高的「被點擊的可能性」，但

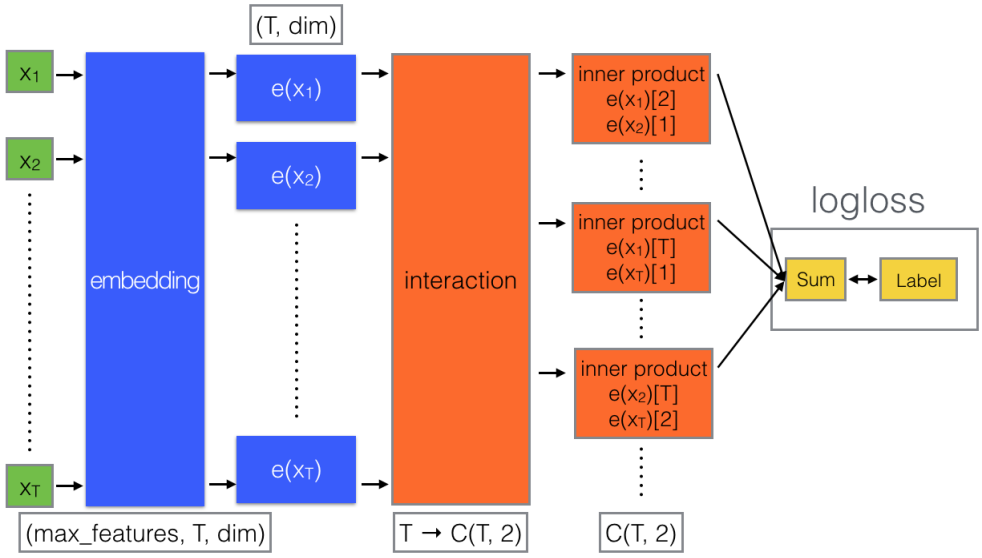

Figure 4: Illustration of Field-aware Factorization Machine

是最後使用者還是只能點擊一個廣告。因此，使用者必須做出選擇，最後點擊那個「比較喜歡」的廣告。在 list-wise 的架構中，我們想要 model 的是給定一個頁面以及這個頁面中的許多廣告，使用者考慮了所有廣告後，最後會點擊哪個。Fig. 5是 list-wise 架構的圖示，我們把一個頁面中的所有廣告都各自放到一個 element-wise 的 model 中，最後再把所有 kernel 的值取 Softmax，最後使用 cross entropy 作為 loss function。透過這種方式把 binary classification 問題轉換成一個 multi-class classification 問題，而每個 class 就是指哪個廣告會被點擊。當然也不一定要各自放到 element-wise 的 model，這其中可以包含一些互動，但在這次的 project 中，我們只提供一個概念，並用簡單的方法來實作，也就是限定廣告之間互動只有在最後的 softmax layer 中，並且是當成一個分類 (classification) 問題而不是排序 (ranking) 問題。

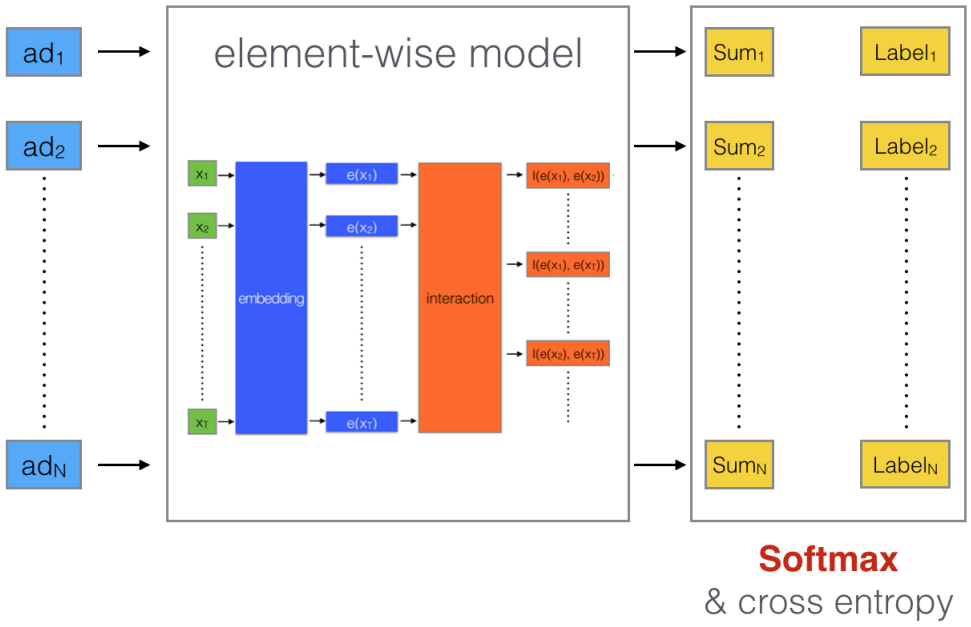

Figure 5: Illustration of lise-wise framework

3 Data Preprocessing and Feature Extraction

3.1 Held-out Data

觀察 dataset 可以發現，training set 比起 testing set 少了最後兩天，而且在 testing set 中，這兩天跟其他天數的比例大概是 1:1。因此，我們在切 held-out data 時，也按照這個比例：先切 training set 的最後兩天，再從前面的天數隨機抽出相同數量作為 held-out data(Fig. 6)。

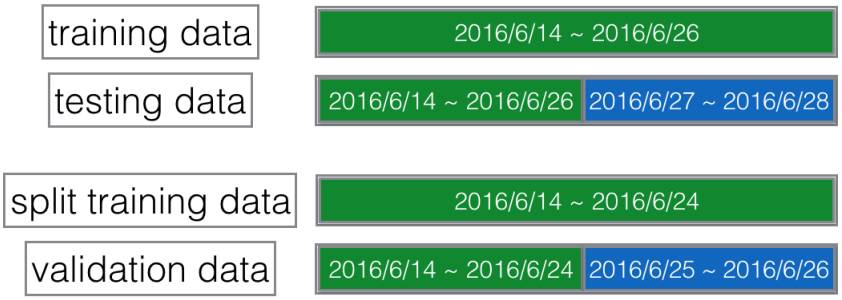

Figure 6: Time Distribution of Dataset

3.2 Categorical Features

經過一些試驗後，我們決定只使用 categorical 的特徵（這會在 Issues 裡討論）。Categorical 的特徵使用 one-hot encoding，例如：topic_id 是 19，所以第 19 個特徵的值是 1。Table. 1 是我們用的所有特徵。

3.3 Hash Trick

因為每個 field 的 index 都是獨立的，因此很可能出現相同的 index，例如：display_id 是 19，且 ad_id 也是 19 的狀況，如果直接使用這些 ids 的話，就會 map 到同樣的 embedding（但事實上不應該是同一個）。因此需要做特徵的轉換，轉換方式如下：

$$new\_index(field, index) = hash(\text{"\$}field\_\$index\text{"}) \tag{4}$$

例如 $display\_id19$ 就會轉換成 $hash(\text{"}display\_id_1 9\text{"})$，$ad\_id19$ 則轉換成 $hash(\text{"}ad\_id_1 9\text{"})$。這個 $hash()$ 可以自己定義，我們為了避免 conflict 的問題，是使用 dictionary 將所有使用過的字串存起來，因此不會有 conflict 的問題。最後總共的特徵數 ($max\_features$) 大約在 130 萬至 140 萬之間，$max\_features$ 會直接影響到 embedding table 的 size，太多的特徵會迫使 dim 只能是較小的數（否則機器的記憶體會裝不下），影響到 performance。

4 Issues

4.1 Missing Values

在做 feature extraction 時常會遇到 missing value 的問題，這個問題來自兩個原因：1. dataset 本來就缺少一部份的值，例如：有些 event 會缺少 location 的資訊。2. 在 training set 中有，但在 testing 中沒有，例如有些 ad_id 只出現在最後兩天，那 training set 使用的特徵 dictionary 便沒有這些資訊，就導致 missing value。

Feature	File	Description
ad_id	clicks_*	這個廣告的 id
ad_doc_id	promoted_content	這個廣告的 document id
ad_campaign_id	promoted_content	這個廣告的 campaign id
ad_advertiser_id	promoted_content	這個廣告的 advertiser id
display_doc_id	events	這個頁面的 document id
platform	events	user 使用的介面
location	events	因為 US data 佔的比例很高，所以除了 US 以外只取 country，US 則取到 state
day	events	將 timestamp 換算成天和小時，因為有兩天不在 training data 裡，所以這裡使用的是 week(天 mod7)
hour	events	因為整個 dataset 是用 UTC-4 為分界，因此這裡的 hour 以及 day 都是以 UTC-4 時間為準
leak	page_views	user 有沒有看過該廣告的 document，有是 1，沒有則是 0
leak_cate	doc_categories	user 有沒有看過該廣告的 document 的 categories，其中只取 confidence_level > 0.3 的 categories 作為一個 document 的 categories，只要看過其中一個 category 便為 1，否則為 0
leak_topic	doc_topics	user 有沒有看過該廣告的 document 的 topics，其中只取 confidence_level > 0.3 的 topics 作為一個 document 的 topics，只要看過其中一個 topic 便為 1，否則為 0
same_cate	doc_categories	如果廣告的 document 的 categories 跟頁面的完全相同，則為 1，否則為 0
same_topic	doc_topics	如果廣告的 document 的 topics 跟頁面的完全相同，則為 1，否則為 0
later_ad	doc_meta	因為廣告一定要比頁面早出現，所以如果廣告的 publish_time 比頁面的 publish_time 晚的話為 1，否則為 0
early_ad	doc_meta	因為廣告一定要先 publish 才能被點擊，所以如果廣告的 publish_time 比 event 的時間 (timestamp) 還早為 1，否則為 0

Table 1: Categorical Features Table

解決方法：只要是 missing value，他的特徵的 index 便為 0，這個 0 對應到一個預留的 embedding，是全部為 0 的 vector，且在 training 時，不會 update 到這個變數。這個做法的好處是當 0 vector 與其他的 vector 互動時，結果都為 0，與沒有這個變數時的 kernel 值相同。

4.2 Numeric Features

我們有嘗試過把 numeric feature 加入 model 中，做法是：把 numeric value 跟 interaction value concatenate，再用 weighted sum。我們也嘗試過各種 normalization 的方法，但是結果都不是很好。而且在這個 dataset 中，大部分都是用 categorical 的特徵表示，只有少

部分能夠用 numeric 的表示法。雖然我們最後沒有使用到 numeric 特徵，但也許其他的 dataset 能適用，所以我們也把這個 model 的 code 放在 github 上。

4.3 Number of Ads

在 list-wise framework 中，每次都要將一個頁面中所有廣告放入，但是 dataset 中每個頁面的廣告數目不同，而 tensorflow 的 input 格式只能是 numpy array（shape 為 (batch_size, num_ads, num_fields)），所以廣告數 (num_ads) 不能是變動的，我們統計了全部頁面的廣告數，發現最多廣告數為 12，因此我們把所有頁面都補足至 12 個廣告，我們用 dummy_ad，也就是特徵的 index 皆為 0，且 label 為 0 的 data，來補齊廣告數不足的頁面。

5 Experiments and Analysis

5.1 Different Models

5.1.1 Settings

這個實驗是在比較不同 model 對於 performance 的影響，我們使用的特徵為 Table. 1中的前 10 個特徵，也就是從 ad_id 到 leak，使用的參數為：embedding dimension: 32, learning_rate: 0.05, regularization term: 0.0, mini-batch_size: 500, iteration: 1。這樣的 Setting 中，得到的 max_features 為 1437122。其中 list-WFM 代表 list-wise 架構下的 Weighted FM，list-WFFM 代表 list-wise 架構下的 Weighted FFM，而 list-FM-NN(layers, neurons) 代表 list-wise 架構下的 FM-NN，hidden layer 為 [layers] 層，[neurons] 個 neuron，linear activation function。

5.1.2 Analysis

Model	$MAP@12_{kaggle}$(public)
FM	0.60624
Weighted FM	0.60709
list-FM	0.67095
list-WFM	0.67339
list-FM-NN(1, 100)	0.67307
list-FM-NN(2, 100)	0.67260
list-FM-NN(1, 500)	0.67265
list-WFFM	0.67402

Table 2: Different Models

首先，「Weighted Sum」這個概念用在 element-wise 架構以及 list-wise 架構上 performace 都有進步，而我們也嘗試了各種 layer 數及 neuron 數的 FM-NN，結果跟 list-WFM 是差不多的，猜測可能原因有兩個：第一是這組參數是根據 list-WFM 調整的，因此 list-WFM 的結果特別好，又或者是 list-FM-NN 跟 list-WFM 是差不多了，也就是只用一層 weights 便足夠了，不需要用到第二層。而 list-WFFM 確實比 list-WFM 還好，而差距應該更大一點的，但是因為只 train 1 個 iteration，所以 initialization 的影響很大。只 train 一個 iteration 是因為 FM 和 FFM 很容易 overfitting，在有切 validation set 的情況下，第 2 個 iteration 通常就已經 overfitting 了，這個狀況在 FFM 的 paper[3] 中也有提到。

5.2 Different Features

5.2.1 Settings

這個實驗是在比較使用不同特徵對於 performance 的影響，使用的參數為：embedding dimension: 16, learning_rate: 0.05, regularization term: 0.0, mini-batch_size: 500, iteration: 1。其中 feature_[feature_num] 代表使用了 Table. 1中的前 [feature_num] 個特徵。

5.2.2 Analysis

Features	$MAP@12_{kaggle}$(public)
features_9	0.65340
features_10	0.67215
features_14	0.67378
features_16	0.67352

Table 3: Different Features

由 Table. 3可以看出，「leak」這個特徵的影響非常大，但是這個差距可能來自於參數（這組參數是根據 features_10 調整的），實際上的差異可能沒有那麼大，而其他的特徵 (feature_14 到 feature_16) 對 performance 的影響可能還在 variance 的範圍內，所以看不出顯著的差異。

5.3 Different Embedding Dimensions

5.3.1 Settings

這個實驗是在比較不同 embedding 維度對於 performance 的影響，我們使用的特徵為 Table. 1中的前 10 個特徵，也就是從 ad_id 到 leak，使用的 model 為 list-wise、weighted sum 的 FM，使用的參數為：learning_rate: 0.05, regularization term: 0.0, mini-batch_size: 500, iteration: 1。這樣的 Setting 中，得到的 max_features 為 1437122。這個實驗的 dimension 從 4 到 128。用 FM 的好處是比起 FFM，FM 比較省空間，所以維度的範圍可以比較大。

5.3.2 Analysis

Dimension	$MAP@12_{kaggle}$(public)
4	0.67419
8	0.67652
64	0.67816
128	0.67921

Table 4: Different Embedding Dimensions

由 Table. 4可以看出，維度越高，performance 越好。但是所需要的 training 時間以及 embedding table 的大小也會變高，維度為 4 的 model 的 training 時間大約為 20 分鐘，但是維度為 128 的 model 的 training 時間大約為 3 小時。可見他們的差異性，當維度足夠大後，增加維度所提高的 performance 應該會越來越小，但在這次實驗中，這件事並不明顯。

6 Contribution

Member	Contribution
江東峻 R05922027	models, feature extraction and experiments
陳翰浩 R05922021	feature extraction and Task 3(Transfer Learning on Stack Exchange Tags)
傅彥禎 R05922006	experiments and Task 3(Transfer Learning on Stack Exchange Tags)
李承軒 R05922032	feature extraction and Task 1(Cyber Security Attack Defender)

Table 5: Member Contribution

References

[1] Outbrain, *Outbrain Click Prediction*

[2] Steffen Rendle, *Factorization Machines*

[3] YuChin Juan, Yong Zhuang, and Wei-Sheng Chin, *Field-aware Factorization Machines for CTR Prediction*