Decard Intern Homework

政治大學資訊管理學系 施瑋昱

相關 code 放至 github: https://github.com/Tunahaha/like_count-prediction

其中只有代碼沒有資料，有 py 版跟 notebook 版

內容

相關演算法:

回歸:

Gradient Boosting:

一種集成學習方法，它將多個弱學習器（通常是 decision tree）組合在一起，以建立一個更強大的模型。梯度提升的主要思想是逐步學習一系列弱學習器，每個學習器都專注於修正前一個學習器的錯誤。簡單來說就是修正前人的錯誤。

梯度提升的基本過程如下：

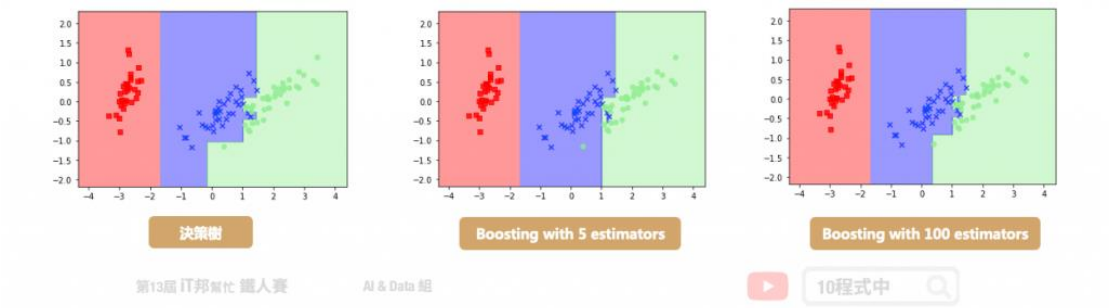
- 初始化：訓練一個基本的弱學習器（例如，決策樹），並計算其預測錯誤。
- 迭代學習：
  - a. 根據前一個學習器的預測錯誤計算梯度（損失函數的負梯度）。
  - b. 訓練一個新的弱學習器，使其對計算出的梯度進行擬合。
  - c. 更新模型，將新學習器的預測結果與先前學習器的結果相結合。
- 重複上一步，直到達到所需的學習器數量或達到性能閾值。

優點:

- 高性能：通常，梯度提升模型在許多機器學習問題上的性能都非常好，有時甚至超過其他先進的方法，如神經網絡。
- 靈活性：梯度提升方法可以與各種損失函數和評估指標一起使用，並且可以輕鬆地擴展到多類問題。
- 解釋性：由於梯度提升方法主要使用決策樹作為基本學習器，因此模型具有一定程度的解釋性。

缺點:

- 訓練時間：梯度提升方法通常需要較長的訓練時間，因為學習器必須按順序學習。



圖片來源: https://ithelp.ithome.com.tw/articles/10273094

XGBboost:

XGboost 全名為 eXtreme Gradient Boosting，由華盛頓大學博士生陳天奇所提出來的，它是以 Gradient Boosting 為基礎。

- 速度和性能： XGBoost 的目標是在保持模型精度的同時提高計算速度。它使用高效的並行和分佈式計算，從而使得大型數據集的訓練過程更快。
- 正則化： XGBoost 在梯度提升算法的基礎上引入了 L1（Lasso）和 L2（Ridge）正則化。正則化有助於防止過擬合，使模型更加穩健。

- 稀疏數據支持： XGBoost 可以自動處理稀疏數據，這對於有大量缺失值或零值的數據集非常有用。
- 內置交叉驗證： XGBoost 允許在每一輪 boosting 迭代中使用內置的交叉驗證來評估模型性能。這有助於選擇最佳迭代次數並防止過擬合。
- 可自定義目標函數和評估指標： XGBoost 支持自定義目標函數和評估指標，這使得它可以用於解決各種機器學習問題，如分類、回歸、排序等。

儘管 Gradient Boosting 和 XGBboost 都是基於相同的原理，但它們之間還是存在一些重要的區別：

- 速度和性能：XGBoost 是為了提高 Gradient Boosting 演算法的效率和性能而開發的。XGBoost 使用了高效的並行處理和樹剪枝技術，這使得它在計算速度和訓練時間上比傳統的 Gradient Boosting 方法更快。此外，XGBoost 在大型數據集上表現出更好的擴展性。
- 正則化：XGBoost 在梯度提升框架中引入了 L1（Lasso）和 L2（Ridge）正則化項，以減少模型的過擬合。這使得 XGBoost 的模型在測試數據上的泛化性能通常優於傳統的 Gradient Boosting 模型。
- 稀疏數據支持：XGBoost 能夠自動處理稀疏數據，這對於有大量缺失值或零值的數據集非常有用。傳統的 Gradient Boosting 方法可能需要額外的預處理步驟來處理這些數據。
- 內置交叉驗證：XGBoost 允許在每一輪 boosting 迭代中使用內置的交叉驗證來評估模型性能。這有助於選擇最佳迭代次數並防止過擬合。傳統的 Gradient Boosting 方法可能需要單獨實現交叉驗證。
- 可自定義目標函數和評估指標：XGBoost 支持自定義目標函數和評估指標，這使得它可以用於解決各種機器學習問題，如分類、回歸、排序等。傳統的 Gradient Boosting 方法可能需要手動修改損失函數和評估指標。

文本處理

TF-IDF:

TF-IDF（Term Frequency-Inverse Document Frequency）是一種用於衡量文本中單詞重要性的統計方法。它在信息檢索、文本挖掘和自然語言處理領域中廣泛應用。

TF-IDF 是基於兩個主要概念：詞頻（Term Frequency，TF）和逆文檔頻率（Inverse Document Frequency，IDF）。這兩個概念的組合可以幫助我們了解單詞在特定文檔中以及整個文檔集合中的重要性。

- 詞頻（TF）：詞頻表示單詞在文檔中出現的次數。單詞在文檔中出現的次數越多，它的詞頻值就越高。詞頻可以用以下公式計算：

  $$TF(t,d) = (\textit{單詞} t \textit{ 在文檔 } d \textit{ 中的出現次數}) / (\textit{文檔 } d \textit{ 中的單詞總數})$$

  其中 t 表示單詞，d 表示文檔。這個公式將詞頻值歸一化，以避免文檔長度對結果的影響。

- 逆文檔頻率（IDF）：逆文檔頻率表示單詞在文檔集合中的罕見程度。單詞在越少的文檔中出現，其逆文檔頻率就越高。IDF 可以用以下公式計算：

$$IDF(t,D) = log(\text{文檔集合 } D \text{ 的總文檔數} / \text{包含單詞 } t \text{ 的文檔數})$$

其中 t 表示單詞，D 表示文檔集合。通過取對數，我們可以減少詞頻值的範圍，避免對結果的影響過大。

將 TF 和 IDF 組合在一起，我們可以得到 TF-IDF 值，它用於衡量單詞在特定文檔中相對於整個文檔集合的重要性。TF-IDF 可以用以下公式計算

$$TF - IDF(t,d,D) = TF(t,d) * IDF(t,D)$$

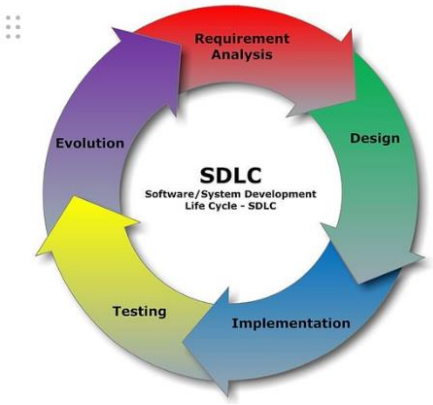
其中 t 表示單詞，d 表示文檔，D 表示文檔集合。
TF-IDF 值的主要優點是它能夠突顯在特定文檔中重要但在整個文檔集合中不常見的單詞。這使得 TF-IDF 值在文本分析和機器學習領域非常有用。
TF-IDF 矩陣，如下圖所示。

詞彙	文件1	文件2	…	文件 D
文字 1	0.48	0.03	…	0.00
文字 2	0.00	0.37	…	0.38
文字 3	0.05	0.08	…	0.22
…	…	…	…	…
文字 T	0.12	0.19	…	0.00

圖片來源: https://taweihuang.hpd.io/2017/03/01/tfidf/

- 文本相似性：通過計算文檔之間的 TF-IDF 向量的餘弦相似度，可以獲得文檔之間的相似性度量。這對於檢索與查詢相關的文檔、文檔分類和聚類非常有用。
- 關鍵詞提取：TF-IDF 值可以幫助我們確定特定文檔中的關鍵詞，這對於文檔摘要、搜索引擎優化（SEO）和主題建模等任務非常有價值。
- 特徵表示：TF-IDF 值可用於將原始文本數據轉換為數字特徵向量，以便用於機器學習算法。這使得 TF-IDF 成為自然語言處理（NLP）領域中常用的特徵提取方法之一。

實驗過程：



如果把資料分析看作是 SDLC 的一種的話
我將其分為 5 個步驟:
1. 分析資料
2. 選擇適合模型
3. 實現
4. 評估準確度
5. 改進

1. 分析資料表:

	title	created_at	like_count_1h	like_count_2h	like_count_3h	like_count_4h	like_count_5h	like_count_6h	comment_count_1h	comment_count_2h	comment_count_3h	comment_count_4h	comment_count_5h	comment_count_6h	forum_id	author_id	forum_stats	like_count_24h
0	我的拼音是	2022-10-05 14:20:21 UTC	12	15	15	15	16	18	10	10	10	10	10	10	598518	428921	0.7	26
1	#請益 播種穿搭	2022-10-05 14:28:13 UTC	0	0	3	4	4	4	2	5	8	9	9	9	399302	650840	63.9	11
2	無關的花花草	2022-10-06 07:18:22 UTC	3	7	8	11	12	14	1	1	2	3	3	3	650776	717288	19.2	19
3	文學增強 課本	2022-09-20 11:39:14 UTC	2	7	11	24	26	26	2	2	8	32	38	63	471023	173889	7.9	29
4	一般課程	2022-09-05 10:18:24 UTC	3	7	7	10	10	11	15	26	35	38	48	49	230184	594332	36.2	16

Feature
- title: 文章標題
- created_at: 文章發佈時間
- like_count_1~6h: 文章發佈後 1~6 小時的累積愛心數
- comment_count_1~6h: 文章發佈後 1~6 小時的累積留言數
- forum_id: 文章發佈看板 ID
- author_id: 文章作者 ID
- forum_stats: 看板資訊

Label
- like_count_24h: 文章發布後 24 小時的累積愛心數

我會把 feature 分成以下幾類:
- 文本資料: title
- 時間資料: created_at
- 數值資料: like_count_1~6h、comment_count_1~6h、forum_stats
- 類別資料: forum_id、author_id

2. 選擇模型:

考慮過許多模型，但最終選擇 XGBooxt 是因為以下幾點
- 多樣性特徵：這個數據集包含了多種類型的特徵，例如連續特徵（如 like_count_1h、comment_count_1h 等）和類別特徵（如 forum_id、author_id 等）。XGBoost 能夠有效地處理多種類型的特徵並自動學習特徵間的交互關係。
- 數據量：雖然 50000 條記錄並不算大數據，但足以讓 XGBoost 表現出優越

性。XGBoost 具有良好的擴展性，可以在大型數據集上保持高效性能。

- 模型複雜度：XGBoost 使用梯度提升方法建立多棵決策樹，使其能夠學習到數據中的高度非線性和複雜的模式。這對於解釋影響結果（如 like_count_24h）的多種因素非常重要。
- 正則化：XGBoost 具有內置的正則化，可以防止模型過擬合，提高泛化能力。這對於處理具有噪聲或異常值的實際數據非常有用。
- 靈活性：XGBoost 提供了多種超參數，可以根據數據集特點進行調整，以達到最佳性能。例如，可以調整樹的深度、學習率、樣本比例等，以尋求更好的結果。

3. 實現:

1. 特徵工程:

首先，為避免因特徵數量不合，導致模型無法運行，將三個資料及合併，並以’set’做區分

```
df, features = join_df(train,test,Private_Test_set)
train['set'] = 'train'
test['set']='test'
Private_Test_set['set'] = 'Private_Test_set'
df = pd.concat([train, test,Private_Test_set])
```

接著依據前面所述，分為四個部分

時間處理:

created_at :2023-02-23 00:18:49 UTC

簡單對 UTC 轉換為相應時區的時間並拆解

```
df['created_at'] = pd.to_datetime(df['created_at']).dt.tz_localize(None)
df['created_at'] = pd.to_datetime(df['created_at'], format='%Y-%m-%d %H:%M:%S')
```

利用時間生成更多特徵

```
df['created_at_since_start'] = (df['created_at'] -
df['created_at'].min()).dt.days
df['created_at_of_week'] = df['created_at'].dt.dayofweek
df['created_at_year'] = df['created_at'].dt.year
df['created_at_year_month'] = df['created_at'].dt.month
df['created_at_day'] = pd.to_datetime(df['created_at']).dt.day
df['created_at_hour'] = pd.to_datetime(df['created_at']).dt.hour
```

created_at_since_start:從這 data_set 最早建立開始，這篇文章存在多久，為了查看特定的時間，例如星期六大家比較有空，或是晚上大家比較有空可以看文章，所以剩下分別是文章對應的星期、年、月、小時，

文本資料:

會用到以下函數:

```
def utils_preprocess_text(text, flg_stemm=False, flg_lemm=True):
    seg_list =jieba.cut(text)
    text = " ".join(seg_list)
```

```
    return text
```

這個 function 幫助我們做文本切割
我對 title 做了一些特徵工程

計算 title 的長度
```
df['title_len'] = df['title'].apply(lambda x: len(x))
```

做文本切割
```
df["clean_title"] = df["title"].apply(lambda x: utils_preprocess_text(x,
flg_stemm=False, flg_lemm=True, ))
```

計算有幾個單字
```
df['clean_title_word_count'] = df["clean_title"].apply(lambda x:
len(str(x).split(" ")))
```

計算有幾個字
```
df['clean_title_char_count'] = df["clean_title"].apply(lambda x: sum(len(word)
for word in str(x).split(" ")))
```

計算 title 的平均單字長度
```
df['clean_title_avg_word_length'] = df['clean_title_char_count'] /
df['clean_title_word_count']
```

計算 title 的情感分數
```
df["clean_title_sentiment"] = df['clean_title'].apply(lambda x:
TextBlob(x).sentiment.polarity)
```

對 title 進行文本特徵抽取，將其轉換為 TF-IDF 特徵矩陣。具體來說，這個函數使用 utils_preprocess_text 函數對文本進行預處理，將其切分成單詞，並去掉出現次數過低或過高的單詞
```
vectorizer = TfidfVectorizer(tokenizer=utils_preprocess_text, min_df=5,
max_df=0.8, ngram_range=(1, 2), max_features=500)
```

```
title_tfidf_matrix = vectorizer.fit_transform(df['title'])
```

數值資料:

進行一些簡單的特徵工程，例如計算 like_count 跟 comment_count 的相關性或是 like_count 的比率等等

```
for i in range(1, 6):
    df[f'like_count_diff_{i}_{i+1}h'] = df[f'like_count_{i+1}h'] -
df[f'like_count_{i}h']
    df[f'comment_count_diff_{i}_{i+1}h'] = df[f'comment_count_{i+1}h'] -
df[f'comment_count_{i}h']
```

```
for i in range(1, 7):
    df[f'like_count_ratio_{i}h'] = df[f'like_count_{i}h'] /
(df[f'like_count_{i}h'].sum() + 1e-8)
    df[f'comment_count_ratio_{i}h'] = df[f'comment_count_{i}h'] /
(df[f'comment_count_{i}h'].sum() + 1e-8)
```

```
for i in range(1, 7):
    df[f'like_comment_ratio_{i}h'] = df[f'like_count_{i}h'] /
(df[f'comment_count_{i}h'] + 1e-8)
    df[f'like_comment_diff_{i}h'] = df[f'like_count_{i}h'] -
df[f'comment_count_{i}h']
```

```
df['like_comment_corr'] = df.apply(calculate_correlation, axis=1)
```

類別資料:

對類別資料進行 One-Hot Encoding，使其不會因類別大小而影響分析

```
df = pd.get_dummies(df, columns=cat_cols)
```

最後:

對 label 進行對數轉換，對數轉換可以幫助我們把數據的尺度縮小，並且可以降低異常值的影響。此外，對數轉換還可以使數據符合正態分佈

```
df[num_cols + ['like_count_24h']] = df[num_cols +
['like_count_24h']].apply(lambda x: np.log1p(x))
```

還有計算兩個標準差以外的值，稍微降低其權重

```
y_train_mean = np.mean(y_train)
y_train_std = np.std(y_train)
sample_weights = np.ones_like(y_train, dtype=np.float32)
std2_outliers = (np.abs(y_train - y_train_mean) >= 2 * y_train_std)
sample_weights[std2_outliers] = 0.7
```

2. 訓練模型

1. 把訓練資料拆成訓練和驗證資料

```
X = train[features]
y = train[TARGET_COL]
Private_Test_set=Private_Test_set.drop(TARGET_COL, 1)
X_train, X_val, y_train, y_val = train_test_split(train[cat_num_cols],y,
test_size=0.2, random_state = 23)
```

2.透過設置超參數 param、num_round 來訓練 XGBoost

```
param = {
    'max_depth': args.max_depth,
    'eta': args.eta,
    'objective': 'reg:squarederror',
    'eval_metric': args.eval_metric,
```

```
}
num_round = args.num_round
```

- max_depth 代表數最大的深度
- eta 代表 learning rate
- objective 為損失函數
- eval_metric 為評估方式，因要求維 MAPE，因此設定為 MAPE
- num_round 為迭代次數

```
bst = xgb.train(param, dtrain, num_round, evals=[(dtrain, 'train'), (dtest, 'test')], early_stopping_rounds=5)
```

3.評估準確度

利用 test 對模型做評估

```
tes_matrix=xgb.DMatrix(test[cat_num_cols])
test_pre=bst.predict(tes_matrix)
test_pre = np.round(np.expm1(test_pre)).astype(int).ravel()
test_true=np.round(np.expm1(test[TARGET_COL])).astype(int)
print(cal_mape(test_true,test_pre))
```

最後輸出 private_test 預測值

```
pri_tes_matrix=xgb.DMatrix(Private_Test_set[cat_num_cols])
pri_xgboost=bst.predict(pri_tes_matrix)
pri_xgboost = np.round(np.expm1(pri_xgboost)).astype(int).ravel()
Private_Test_pred = pd.DataFrame({'like_count_24h': pri_xgboost})
Private_Test_pred.to_csv('result.csv', index=False)
```

並重複調整參數
細節都在 code 裡
用 py 檔可以利用 CMD 調整超參數

4. 結果:

我把超參數調成如下圖，得到最好的 mape，但 py 檔中，eta 我是設定為 0.05，這樣跑起來會快很多。

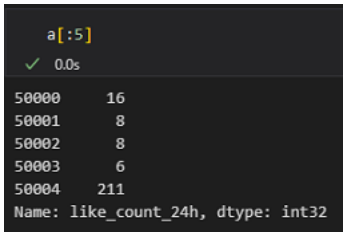
```
param2 = {
    'max_depth': 9,
    'eta': 0.01,
    'objective': 'reg:squarederror',
    'eval_metric': 'mape',
}
num_round2 = 400
```

```
print(cal_mape(a,xgboost))
✓ 0.0s
33.02151680026614
```

下圖為 public_test 預測出來的結果

```
xgboost[:5]
✓ 0.0s
array([ 19,  11,   8,   7, 226])
```

下圖為 public_test 實際的結果



看得出雖有所差異，但差異不大

想法:

當處理複合型資料時，需要先對各個特徵之間的關係有一定了解，並有一個清晰的處理方案。在我處理這種資料時，我遇到了一些小問題，例如使用 One-Hot Encoding 導致矩陣變得非常龐大。為了解決這個問題，我評估了幾種不同的方法，包括稀疏矩陣和降維，但最終我仍然決定保持原本的方法，因為稀疏矩陣的操作比密集矩陣複雜，而降維可能會失去一些重要的資訊，對模型效果造成負面影響。

在選擇模型時，我考慮了多種組合，包括 LightGBM、CatBoost、神經網路和 XGBoost，而最終我選擇了 XGBoost，因為它是 Kaggle 競賽的常勝軍，而且其在訓練時間和準確性方面的表現都非常出色。我也考慮使用 Stacking 的方式進一步提高模型的準確性，但由於訓練時間過長，最終我放棄了這個選項。

當然，除了模型選擇之外，我也考慮了特徵選擇等更高效的方法，但在追求準確性的前提下，我決定遵循傳統的特徵工程方法，以保持最好的準確性。

最後在評估方式方面，原本是想使用 rmse 當作評估指標，但因為要求要是 MAPE 的關係就沒使用了。

經過這次作業的挑戰，我對於特徵工程和模型選擇的理解更加深入了，並且學習到如何處理複合型資料。這也讓我更有信心在未來的數據科學專案中探索更複雜的問題。最後，感謝您閱讀我的報告。