

Data Reporting

Dcard潛力熱門文章預測

林子涵

16th April, 2020



Outline

1. Context
2. Data Exploring and Cleaning
3. Model Training and Evaluation
4. Model Testing
5. Conclusion

Context

在 Dcard 上，使用者常常透過「熱門文章」列表版面來得知目前站上最火熱的討論話題。熱門文章意味著此類文章能吸引多數人的目光、多數人會對此文章有興趣，因此若能即早（例如：在其登上熱門列表前）推薦此文章給使用者，將能有效提升此文章的傳播（研究顯示(Henrique Pinto et al., 2013; Sunstein, 2019)，以Youtube為例，發燒影片在登上發燒前，閱聽人的分享或按讚的意願更高。因此能在達到發燒門檻前，使該影片觸及愈多閱聽人，將能大大提升其分享與按讚率）。更甚者，若能成功推薦使用者喜愛的影片，亦能提升使用者對平台的好感與信任度，而推薦潛在熱門文章將是最為保險的作法之一（如上所述，熱門文章意味多數人會感興趣的文章）。

綜合以上兩點，本報告將以預測潛力熱門文章為目標，並以文章生成後10小時內的被愛心、分享、收藏與留言的數據為基礎，預測其是否有機會成為熱門文章（在此報告中，熱門文章定義為文章發出的 36 小時內愛心數大於等於1000者）。

Data Exploring and Cleaning

首先，連接database取得訓練模型所需要的資料，共有五個tables。

```
# Connector function
def postgres_connector(host, port, database, user, password=None):
    user_info = user if password is None else user + ':' + password
    # example: postgres://federer:grandestlam@localhost:5432/tennis
    url = 'postgres://%s@%s:%s/%s' % (user_info, host, port, database)
    return sqlalchemy.create_engine(url, client_encoding='utf-8')

# Get connect engine
engine = postgres_connector(
    "35.187.144.113",
    5432,
    "intern_task",
    "candidate",
    "dcard-data-intern-2020"
)

sql_cmd = "SELECT * FROM posts_train"
sql_cmd2 = "SELECT * FROM post_shared_train"
sql_cmd3 = "SELECT * FROM post_comment_created_train"
sql_cmd4 = "SELECT * FROM post_liked_train"
sql_cmd5 = "SELECT * FROM post_collected_train"

df_posts = pd.read_sql(sql_cmd, engine)
df_shared_hr = pd.read_sql(sql_cmd2, engine)
df_commented_hr = pd.read_sql(sql_cmd3, engine)
df_liked_hr = pd.read_sql(sql_cmd4, engine)
df_collected_hr = pd.read_sql(sql_cmd5, engine)
```

由於其中四個表格為文章**每小時**（發文後的10小時內）的被分享數、留言數、愛心數與收藏數，同一文章會重複於同一table中出現。因此在同一table中，我們將同一文章歸類在一起，即顯示其發文後**10小時內累積**的被分享數、留言數、愛心數與收藏數。

```
#Grouping the same posts to get each post's data in 10 hours after created

#number of "share" in 10 hours
post_shared_in_10 = df_shared_hr[["post_key", "count"]].groupby(["post_key"], as_index=False, sort=False).sum()
post_shared_in_10.rename(columns={"count": "share_count_in_10"})

#number of "comment" in 10 hours
post_commented_in_10 = df_commented_hr[["post_key", "count"]].groupby(["post_key"], as_index=False, sort=False).sum()
post_commented_in_10.rename(columns={"count": "comment_count_in_10"})

#number of "Like" in 10 hour
post_liked_in_10 = df_liked_hr[["post_key", "count"]].groupby(["post_key"], as_index=False, sort=False).sum()
post_liked_in_10.rename(columns={"count": "like_count_in_10"})

#number of "collect" in 10 hour
post_collected_in_10 = df_collected_hr[["post_key", "count"]].groupby(["post_key"], as_index=False, sort=False).sum()
post_collected_in_10.rename(columns={"count": "collect_count_in_10"})
```

然而，記錄36小時內累積愛心數量的table之文章數有793751筆，但其他四個tables（歸類後）的文章數量卻不一致，且皆小於793751筆。故透過查詢各table最小值後進一步得知，在發文後10小時內未有任何被分享數、留言數、愛心數或收藏數之文章，將不會被記錄於這四個tables中（即不會以0記錄之）。此同時排除這些NaN為沒收集到的數據（i.e., missing data）之可能。

```
#check the number of posts in each category(i.e., share, comment, like, and collect)
a = df_posts["post_key"]
b = df_shared_hr["post_key"]
c = df_commented_hr["post_key"]
d = df_liked_hr["post_key"]
e = df_collected_hr["post_key"]

number_of_posts_a = len(np.unique(a))
number_of_posts_b = len(np.unique(b))
number_of_posts_c = len(np.unique(c))
number_of_posts_d = len(np.unique(d))
number_of_posts_e = len(np.unique(e))
```

```
min(df_posts["like_count_36_hour"])
```

0

```
min(df_liked_hr["count"])
```

1

基於此，我們在tables合併後，將這些10小時內未被分享、留言、愛心或收藏的NaN資料補上0值。

```
#Merge different tables (i.e., categories)
df_for_train = df_posts.merge(post_shared_in_10, how='outer', on='post_key')
df_for_train = df_for_train.merge(post_commented_in_10, how='outer', on='post_key')
df_for_train = df_for_train.merge(post_liked_in_10, how='outer', on='post_key')
df_for_train = df_for_train.merge(post_collected_in_10, how='outer', on='post_key')
df_for_train.columns = ['post_key', 'created_at_hour', 'like_count_36_hour', 'share_count_in_10',
                        'comment_count_in_10', 'like_count_in_10', 'collect_count_in_10']

#NaN means zero since we knew posts without any share, comment, like, or collect will not be shown in each category
#Therefore, it is reasonable to replace NaN with zero
df_for_train.fillna(0, inplace=True)
```

接著，進行feature transformation，將預測標的（outcome variable）：36小時內累積愛心數量，轉換為dummy variable，門檻設定為1000，即數量大於等於1000者為True，反之為False。

同時，將發文日期轉換為年、月、星期、日與時間五個predictive variables，因為根據研究(Sabate et al., 2014)，（以facebook為例）發文時間和按讚與分享數有顯著性關係。

```
#Feature engineering

#Make posts with like_count_36_hour >= 1000 "trending posts" which means to make the numeric variable "dummy"
df_for_train['trending'] = df_for_train['like_count_36_hour'] >= 1000

#Make time of being created "variables"
from datetime import datetime as dt

df_for_train['year'] = df_for_train['created_at_hour'].dt.year
df_for_train['month'] = df_for_train['created_at_hour'].dt.month
df_for_train['weekday'] = df_for_train['created_at_hour'].dt.weekday
df_for_train['day'] = df_for_train['created_at_hour'].dt.day
df_for_train['hour'] = df_for_train['created_at_hour'].dt.hour
```

並將原發文日期（已轉換為五個variables）與36小時愛心累計（已轉換為布林值）兩個columns刪除：

```
df_for_train.set_index('post_key', inplace=True)
df_for_train.drop(["created_at_hour", "like_count_36_hour"], axis=1, inplace=True)
df_for_train
```

	share_count_in_10	comment_count_in_10	like_count_in_10	collect_count_in_10	trending	year	month	weekday	day	hour
post_key										
0002f1f8-c96b-4332-8d19-9cdfa9900f75	1.0	3.0	37.0	4.0	False	2019	6	5	1	5
000c74b1-533d-4445-94ab-038ed4b9a28d	0.0	0.0	12.0	0.0	False	2019	9	4	13	15
000d9763-e88c-408e-907c-02db7656bb1f	1.0	38.0	30.0	13.0	False	2019	8	0	26	19
000ffc2c-cc94-410a-9125-e98d1a21d2e2	0.0	9.0	1.0	1.0	False	2019	5	1	21	15
001472bc-cd2e-4366-8db7-6a11bc3baf10	0.0	0.0	5.0	0.0	False	2019	10	1	8	14
...
998b59dc-a05b-4200-9507-f83e81cd46a6	0.0	5.0	0.0	0.0	False	2019	9	2	25	17
8df6e4e7-a460-40ff-9912-6095059af490	0.0	5.0	0.0	2.0	False	2019	10	2	16	2
4d74806c-323a-4922-a268-2f8fbaf83140	0.0	1.0	0.0	0.0	False	2019	5	6	19	12
bd61a284-b510-4f29-ba11-39e060a244ac	0.0	0.0	0.0	0.0	False	2019	6	0	17	6

793751 rows × 10 columns

共有十個variables（其中一個為outcome variable，其餘皆為predictive variables）。

```
df_for_train.info()
df_for_train.describe()
```

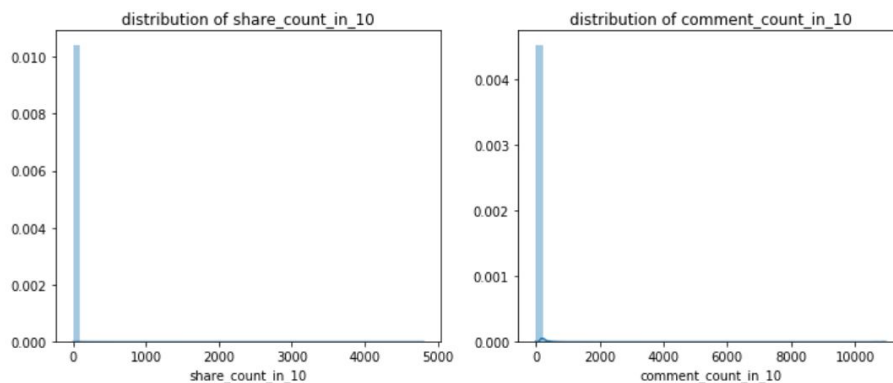
```
<class 'pandas.core.frame.DataFrame'>
Index: 793751 entries, 0002f1f8-c96b-4332-8d19-9cdfa9900f75 to 4d33e5ad-cf73-41d4-b013-3a8064d239d5
Data columns (total 10 columns):
share_count_in_10    793751 non-null float64
comment_count_in_10  793751 non-null float64
like_count_in_10     793751 non-null float64
collect_count_in_10  793751 non-null float64
trending             793751 non-null bool
year                 793751 non-null int64
month                793751 non-null int64
weekday              793751 non-null int64
day                  793751 non-null int64
hour                 793751 non-null int64
dtypes: bool(1), float64(4), int64(5)
```

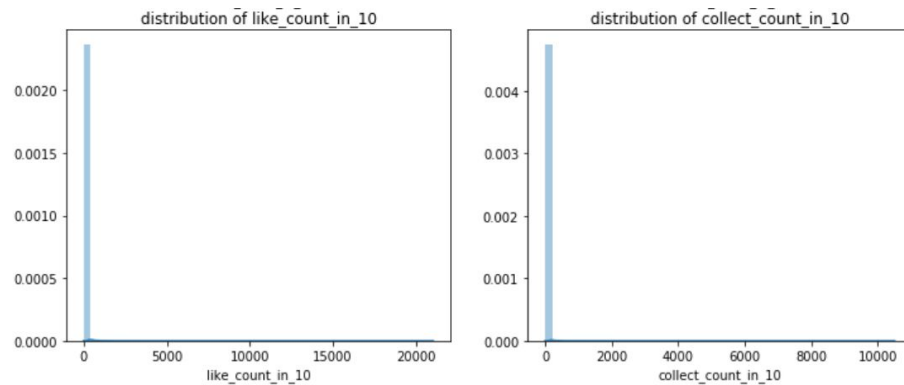
接著我們透過使用seaborn的displot來觀察10小時內的被分享、留言、愛心或收藏之數量分布（包含密度函數），可以發現原本的資料非常歪斜（skewed to the right）。因此我們進一步透過取Log的方式，來讓資料分布較為接近常態分佈，藉此提升預測的準確度。

```
import matplotlib.pyplot as plt
import seaborn as sns

fig, axes = plt.subplots(nrows=2,ncols=2)
fig.set_size_inches(12, 10)
sns.distplot(df_for_train["share_count_in_10"],ax=axes[0][0])
sns.distplot(df_for_train["comment_count_in_10"],ax=axes[0][1])
sns.distplot(df_for_train["like_count_in_10"],ax=axes[1][0])
sns.distplot(df_for_train["collect_count_in_10"],ax=axes[1][1])

axes[0][0].set(xlabel='share_count_in_10',title="distribution of share_count_in_10")
axes[0][1].set(xlabel='comment_count_in_10',title="distribution of comment_count_in_10")
axes[1][0].set(xlabel='like_count_in_10',title="distribution of like_count_in_10")
axes[1][1].set(xlabel='collect_count_in_10',title="distribution of collect_count_in_10")
```

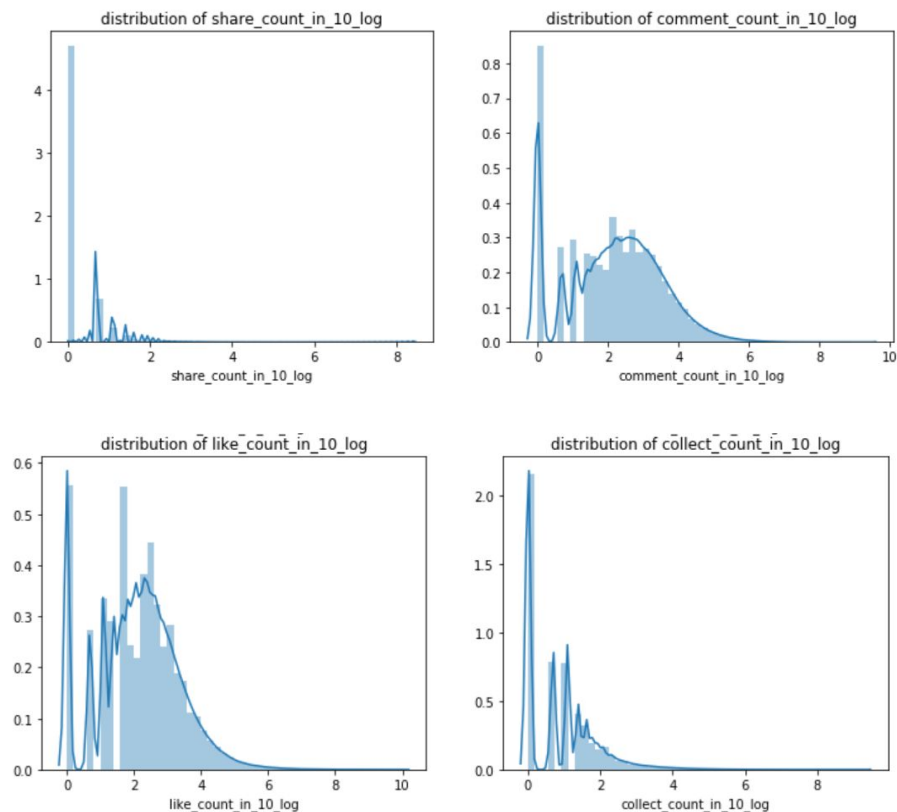




*#As seen, the distributions are very skewed to the right
#Therefore, it is reasonable to Logarithmize the variables*

```
fig, axes = plt.subplots(nrows=2,ncols=2)
fig.set_size_inches(12, 10)
sns.distplot(np.log1p(df_for_train["share_count_in_10"]),ax=axes[0][0])
sns.distplot(np.log1p(df_for_train["comment_count_in_10"]),ax=axes[0][1])
sns.distplot(np.log1p(df_for_train["like_count_in_10"]),ax=axes[1][0])
sns.distplot(np.log1p(df_for_train["collect_count_in_10"]),ax=axes[1][1])

axes[0][0].set(xlabel='share_count_in_10_log',title="distribution of share_count_in_10_log")
axes[0][1].set(xlabel='comment_count_in_10_log',title="distribution of comment_count_in_10_log")
axes[1][0].set(xlabel='like_count_in_10_log',title="distribution of like_count_in_10_log")
axes[1][1].set(xlabel='collect_count_in_10_log',title="distribution of collect_count_in_10_log")
```



將原本被分享、留言、愛心或收藏改為取log後的數值：

```
df_for_train['share_count_in_10_log'] = np.log1p(df_for_train["share_count_in_10"])
df_for_train['comment_count_in_10_log'] = np.log1p(df_for_train["comment_count_in_10"])
df_for_train['like_count_in_10_log'] = np.log1p(df_for_train["like_count_in_10"])
df_for_train['collect_count_in_10_log'] = np.log1p(df_for_train["collect_count_in_10"])

df_for_train.drop(["share_count_in_10", "comment_count_in_10", "like_count_in_10", "collect_count_in_10"], axis=1, inplace=True)
df_for_train
```

	trending	year	month	weekday	day	hour	share_count_in_10_log	comment_count_in_10_log	like_count_in_10_log	collect_count_in_10_log
post_key										
0002f1f8-c96b-4332-8d19-9cdfa9900f75	False	2019	6	5	1	5	0.693147	1.386294	3.637586	1.609438
000c74b1-533d-4445-94ab-038ed4b9a28d	False	2019	9	4	13	15	0.000000	0.000000	2.564949	0.000000
000d9763-e88c-408e-907c-02db7656bb1f	False	2019	8	0	26	19	0.693147	3.663562	3.433987	2.639057
000ffc2c-cc94-410a-9125-e98d1a21d2e2	False	2019	5	1	21	15	0.000000	2.302585	0.693147	0.693147
001472bc-cd2e-4366-8db7-6a11bc3baf10	False	2019	10	1	8	14	0.000000	0.000000	1.791759	0.000000
...
998b59dc-a05b-4200-9507-f83e81cd46a6	False	2019	9	2	25	17	0.000000	1.791759	0.000000	0.000000

793751 rows × 10 columns

接著，將此dataset以7:3的比例切分為訓練用data以及驗證用data。

```
predictors_train = df_for_train.drop(["trending"], axis=1)
```

```
outcome_train = df_for_train["trending"]
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.utils import resample

predictors_train_train, predictors_train_test, outcome_train_train, outcome_train_test = train_test_split(predictors_train,
                                                                                                         outcome_train,
                                                                                                         test_size=0.3,
                                                                                                         random_state=12,
                                                                                                         stratify=outcome_train)
```

同時，我們亦觀察到此dataset嚴重不平衡，即熱門文章數量相對而言非常少。

```
outcome_train_train.value_counts()
```

```
False    542761
True      12864
Name: trending, dtype: int64
```

因此，將針對此狀況，將Upsample熱門文章，讓其數量與一般文章相同（即542761筆）。

```
outcome_train_train.value_counts()
#There are 542761 non-trending posts and 12864 trending ones

pointseven_df_for_train = pd.merge(predictors_train_train, outcome_train_train, left_index=True, right_index=True)
df_majority = pointseven_df_for_train.loc[pointseven_df_for_train.trending == False].copy()
df_minority = pointseven_df_for_train.loc[pointseven_df_for_train.trending == True].copy()

# Upsample minority class
df_minority_upsampled = resample(df_minority, replace=True, n_samples=542761, random_state=42)

# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_majority, df_minority_upsampled])

# Display new class counts
print(df_upsampled.trending.value_counts())
```

```
True      542761
False     542761
Name: trending, dtype: int64
```

最後將此upsampled dataset切分為predictors與outcome，即完成資料清洗：

```
predictors_train_upsampled = df_upsampled.drop(["trending"], axis=1)
```

```
outcome_train_upsampled = df_upsampled["trending"]
```

Model Training and Evaluation

有鑑於dataset中的資料分布並非常態分佈（即便進行logrithmization處理），本報告擬選擇不受資料分配限制（即distribution-free，對資料長相的要求小）的模型進行訓練。同時，理論上，由於預測標的（即文章將會熱門與否）的形成因素眾多，可以合理判斷除了dataset中的9個predictors外，還有其他因素（例如文章內容、點選愛心的組成族群等），因此不同sampled dataset間的差異可能具顯著性差異，因此本報告亦擬選擇使用集成法(ensemble methods)之模型，以提高模型的泛化能力（降低overfitting與variance）。

基於以上兩項條件，本研究選擇以下四分類（因預測標的為二元變數）模型進行訓練：

1. Random Forest Classifier
2. Gradient Boosting Classifier
3. XGBoost Classifier
4. Adaboost Classifier

（註：本報告亦使用交叉驗證(Cross Validation)與網格搜尋(Grid Search)調整各模型的參數，但礙於使用硬體運算設備之限制，只針對部分參數進行5-folders網格搜尋調參。）

```
#Model training and selection among bagging and boosting algorithms
#There are randomforest, gradient boosting, XGBoosting, and Adaboost in order

from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
import xgboost as xgb
from sklearn.ensemble import AdaBoostClassifier
```

Random Forest :

```
#RandomForest (bagging)
rfc = RandomForestClassifier(max_features = "sqrt")

param_grid = {"n_estimators": [500, 1000]}
CV_rfc = GridSearchCV(estimator = rfc, param_grid = param_grid, cv = 5)
CV_rfc.fit(predictors_train_upsampled, outcome_train_upsampled)
```

Gradient Boosting :

```
#GradientBoosting (boosting1)
gbc = GradientBoostingClassifier(max_features = "sqrt")

param_grid_2 = {"learning_rate": [0.25, 0.5, 0.75], "n_estimators": [500, 1000]}
CV_gbc = GridSearchCV(estimator = gbc, param_grid = param_grid_2, cv = 5)
CV_gbc.fit(predictors_train_upsampled, outcome_train_upsampled)
```

XGBoost :

```
#XGBoosting (boosting2)
xgb_model = xgb.XGBClassifier()

param_grid_3 = {"learning_rate": [0.25, 0.5, 0.75], "n_estimators": [500, 1000]}

CV_xgb_model = GridSearchCV(estimator = xgb_model, param_grid = param_grid_3, cv = 5)
CV_xgb_model.fit(predictors_train_upsampled, outcome_train_upsampled)
```

Adaboost :

```
#Adaboost (boosting3)
ada = AdaBoostClassifier()

param_grid_4 = {"learning_rate": [0.25, 0.5, 0.75], "n_estimators": [500, 1000]}

CV_ada = GridSearchCV(estimator = ada, param_grid = param_grid_4, cv = 5)
CV_ada.fit(predictors_train_upsampled, outcome_train_upsampled)
```

使用原本保留的30%資料（為避免leakage，此30%是在Upsampling前所切割出來的）進行驗證與評估，藉此選出最適合模型。

```
#For Validation
outcome_rfc_val = CV_rfc.best_estimator_.predict(predictors_train_test)
outcome_gbc_val = CV_gbc.best_estimator_.predict(predictors_train_test)
outcome_xgb_val = CV_xgb_model.best_estimator_.predict(predictors_train_test)
outcome_ada_val = CV_ada.best_estimator_.predict(predictors_train_test)

print(classification_report(outcome_train_test, outcome_rfc_val))
print(classification_report(outcome_train_test, outcome_gbc_val))
print(classification_report(outcome_train_test, outcome_xgb_val))
print(classification_report(outcome_train_test, outcome_ada_val))
```

Random Forest :

	precision	recall	f1-score	support
False	0.99	1.00	0.99	232613
True	0.71	0.50	0.59	5513
accuracy			0.98	238126
macro avg	0.85	0.75	0.79	238126
weighted avg	0.98	0.98	0.98	238126

Gradient Boosting :

	precision	recall	f1-score	support
False	1.00	0.93	0.96	232613
True	0.23	0.93	0.37	5513
accuracy			0.93	238126
macro avg	0.62	0.93	0.67	238126
weighted avg	0.98	0.93	0.95	238126

XGBoost :

	precision	recall	f1-score	support
False	0.99	0.97	0.98	232613
True	0.37	0.75	0.50	5513
accuracy			0.96	238126
macro avg	0.68	0.86	0.74	238126
weighted avg	0.98	0.96	0.97	238126

Adaboost :

	precision	recall	f1-score	support
False	1.00	0.92	0.96	232613
True	0.21	0.96	0.35	5513
accuracy			0.92	238126
macro avg	0.61	0.94	0.65	238126
weighted avg	0.98	0.92	0.94	238126

本報告以macro-average f1 score作為評估標準，基於此以及上方結果（細節將於 Conclusion Section討論），選擇Random Forest（macro-average f1 score=0.79）作為下階段對測試dataset進行預測之模型。更甚者，為提升模型準確度，將對此Random Forest的分類門檻（原本為0.5）進行優化調整：


```
#Based on the result of validation, the random forest model was chosen
#Now, tuning the threshold of probability for classification

from sklearn.metrics import precision_score, \
    recall_score, confusion_matrix, classification_report, \
    accuracy_score, f1_score
from sklearn import metrics

rfc_val_pr = CV_rfc.best_estimator_.predict_proba(predictors_train_test)

thresholds = []
for thresh in np.arange(0.3, 0.801, 0.01):
    thresh = np.round(thresh, 2)
    res = metrics.f1_score(outcome_train_test, (rfc_val_pr[:,1] > thresh).astype(int))
    thresholds.append([thresh, res])
    print("F1 score at threshold {0} is {1}".format(thresh, res))

thresholds.sort(key=lambda x: x[1], reverse=True)
best_thresh = thresholds[0][0]
print("Best threshold: ", best_thresh)
```



```
F1 score at threshold 0.3 is 0.5902315255312401
F1 score at threshold 0.31 is 0.5929997574973729
F1 score at threshold 0.32 is 0.5961997203257382
F1 score at threshold 0.33 is 0.5982448809026327
F1 score at threshold 0.34 is 0.6007817811012917
F1 score at threshold 0.35 is 0.6018310589048194
F1 score at threshold 0.36 is 0.6040021063717745
F1 score at threshold 0.37 is 0.6055684454756382
F1 score at threshold 0.38 is 0.6052869817128372
F1 score at threshold 0.39 is 0.6067622197721426
F1 score at threshold 0.4 is 0.6060889054912215
F1 score at threshold 0.41 is 0.605070743519134
F1 score at threshold 0.42 is 0.6027686983272448
F1 score at threshold 0.43 is 0.6011110028262353
F1 score at threshold 0.44 is 0.6000197180321404
F1 score at threshold 0.45 is 0.5988430081787354
F1 score at threshold 0.46 is 0.5987261146496814
F1 score at threshold 0.47 is 0.5962936418552267
F1 score at threshold 0.48 is 0.5948632974316488
F1 score at threshold 0.49 is 0.5911092472216397
F1 score at threshold 0.5 is 0.5878349638451722
```

```
F1 score at threshold 0.51 is 0.585654371437789
F1 score at threshold 0.52 is 0.581917211328976
F1 score at threshold 0.53 is 0.5784486557955046
F1 score at threshold 0.54 is 0.5756934387880138
F1 score at threshold 0.55 is 0.5713319048156085
F1 score at threshold 0.56 is 0.5673645880746473
F1 score at threshold 0.57 is 0.5647817574571001
F1 score at threshold 0.58 is 0.5615671641791046
F1 score at threshold 0.59 is 0.5568756616868603
F1 score at threshold 0.6 is 0.5521006408734869
F1 score at threshold 0.61 is 0.5475905058738911
F1 score at threshold 0.62 is 0.5421963918149898
F1 score at threshold 0.63 is 0.5370619123214068
F1 score at threshold 0.64 is 0.5315770035701096
F1 score at threshold 0.65 is 0.5257142857142858
F1 score at threshold 0.66 is 0.5205307961942914
F1 score at threshold 0.67 is 0.5136501516683519
F1 score at threshold 0.68 is 0.50790413054564
F1 score at threshold 0.69 is 0.5019295086184719
F1 score at threshold 0.7 is 0.49565330219281173
```

```
F1 score at threshold 0.71 is 0.4896623920439676
F1 score at threshold 0.72 is 0.48025359926033556
F1 score at threshold 0.73 is 0.470603919477403
F1 score at threshold 0.74 is 0.4663087248322148
F1 score at threshold 0.75 is 0.457707402896197
F1 score at threshold 0.76 is 0.45047748976807644
F1 score at threshold 0.77 is 0.44092972080869214
F1 score at threshold 0.78 is 0.4295637677132536
F1 score at threshold 0.79 is 0.4206838565022422
F1 score at threshold 0.8 is 0.4097989238176154
Best threshold: 0.39
```

得到最優化的分類機率門檻為0.39。

最後，基於此模型，探究9個predictors的重要性排序。（細節將於Conclusion Section討論）

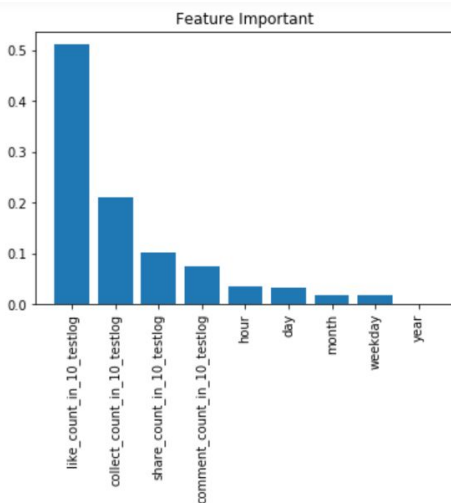
```
#Best threshold: 0.39 (F1 score is 0.6060549430013081)
#Generate feature importance chart

feature_names = np.array(["year", "month", "weekday", "day",
                           "hour", "share_count_in_10_testlog",
                           "comment_count_in_10_testlog", "like_count_in_10_testlog", "collect_count_in_10_testlog"])
importances = CV_rfc.best_estimator_.feature_importances_
indices = np.argsort(importances)[::-1]
type(indices)

plt.figure
plt.title("Feature Important")

plt.bar(range(predictors_train_test.shape[1]), importances[indices])
plt.xticks(range(predictors_train_test.shape[1]), [feature_names[i] for i in indices], rotation=90)

plt.show()
```



最後，使用pickle保存此Random Forest模型，以供後續使用。

```
#Pickle the model
import pickle

TH_train_model_pk1 = open("TH_train_model.pkl", "wb")
pickle.dump(CV_rfc.best_estimator_, TH_train_model_pk1)
TH_train_model_pk1.close()
```


Model Testing

呼叫從上個階段選出的Random Forest模型，使用新的test dataset進行模型測試。

```
#Get the model trained in TH_train.py file
from sklearn.metrics import classification_report
import pickle
import pandas as pd

trained_model_pkl = open("TH_train_model.pkl", "rb")
trained_model = pickle.load(trained_model_pkl)
trained_model

RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                        max_depth=None, max_features='sqrt', max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=500,
                        n_jobs=None, oob_score=False, random_state=None,
                        verbose=0, warm_start=False)
```

接著連接database取得測試用的dataset，並如同上個階段的data cleaning and feature transformation步驟，進行文章歸類、合併5個tables、將日期轉化為獨立5個predictors（即year, month, day, weekday與hour）、使用log轉化被分享、留言、愛心或收藏數等。

取得測試用的dataset：

```
#Embark on testing the model

def postgres_connector(host, port, database, user, password=None):
    user_info = user if password is None else user + ':' + password
    # example: postgresql://federer:grandestslam@localhost:5432/tennis
    url = 'postgres://%s@%s:%s/%s' % (user_info, host, port, database)
    return sqlalchemy.create_engine(url, client_encoding='utf-8')

engine = postgres_connector(
    "35.187.144.113",
    5432,
    "intern_task",
    "candidate",
    "dcard-data-intern-2020"
)

sql_cmd6 = "SELECT * FROM posts_test"
sql_cmd7 = "SELECT * FROM post_shared_test"
sql_cmd8 = "SELECT * FROM post_comment_created_test"
sql_cmd9 = "SELECT * FROM post_liked_test"
sql_cmd10 = "SELECT * FROM post_collected_test"

df_posts_test = pd.read_sql(sql_cmd6, engine)
df_shared_hr_test = pd.read_sql(sql_cmd7, engine)
df_commented_hr_test = pd.read_sql(sql_cmd8, engine)
df_liked_hr_test = pd.read_sql(sql_cmd9, engine)
df_collected_hr_test = pd.read_sql(sql_cmd10, engine)
```


文章歸類：

```
#Group the same posts
post_shared_in_10_test = df_shared_hr_test[["post_key", "count"]].groupby(["post_key"], as_index=False, sort=False).sum()
post_shared_in_10_test.rename(columns={"count": "share_count_in_10_test"})
post_commented_in_10_test = df_commented_hr_test[["post_key", "count"]].groupby(["post_key"], as_index=False, sort=False).sum()
post_commented_in_10_test.rename(columns={"count": "comment_count_in_10_test"})
post_liked_in_10_test = df_liked_hr_test[["post_key", "count"]].groupby(["post_key"], as_index=False, sort=False).sum()
post_liked_in_10_test.rename(columns={"count": "like_count_in_10_test"})
post_collected_in_10_test = df_collected_hr_test[["post_key", "count"]].groupby(["post_key"], as_index=False, sort=False).sum()
post_collected_in_10_test.rename(columns={"count": "collect_count_in_10_test"})
```

合併5個tables：

```
#Merge tables
df_for_test = df_posts_test.merge(post_shared_in_10_test, how='outer', on='post_key')
df_for_test = df_for_test.merge(post_commented_in_10_test, how='outer', on='post_key', inplace=True)
df_for_test = df_for_test.merge(post_liked_in_10_test, how='outer', on='post_key', inplace=True)
df_for_test = df_for_test.merge(post_collected_in_10_test, how='outer', on='post_key', inplace=True)
df_for_test.columns
df_for_test.columns = ['post_key', 'created_at_hour', 'like_count_36_hour', 'share_count_in_10_test',
                      'comment_count_in_10_test', 'like_count_in_10_test', 'collect_count_in_10_test']
df_for_test_2 = df_for_test.fillna(0)
```

將36小時累積愛心數轉化為dummy variable：

```
#Feature Engineering: make like counts in 26 hrs a dummy variable
df_for_test_2['trending'] = df_for_test_2['like_count_36_hour'] >= 1000
```

將日期轉化為獨立5個predictors：

```
#Feature Engineering: make time be variables of year, month, weekday, day, hour
df_for_test_2['year'] = df_for_test_2['created_at_hour'].dt.year
df_for_test_2['month'] = df_for_test_2['created_at_hour'].dt.month
df_for_test_2['weekday'] = df_for_test_2['created_at_hour'].dt.weekday
df_for_test_2['day'] = df_for_test_2['created_at_hour'].dt.day
df_for_test_2['hour'] = df_for_test_2['created_at_hour'].dt.hour

df_for_test_2.set_index(['post_key'], inplace=True)
df_for_test_2.drop(['created_at_hour', "like_count_36_hour"], axis=1, inplace=True)
```

使用log轉化被分享、留言、愛心或收藏數：

```
#Logrithmization
df_for_test_2['share_count_in_10_testlog'] = np.log1p(df_for_test_2["share_count_in_10_test"])
df_for_test_2['comment_count_in_10_testlog'] = np.log1p(df_for_test_2["comment_count_in_10_test"])
df_for_test_2['like_count_in_10_testlog'] = np.log1p(df_for_test_2["like_count_in_10_test"])
df_for_test_2['collect_count_in_10_testlog'] = np.log1p(df_for_test_2["collect_count_in_10_test"])

df_for_test_2.drop(["share_count_in_10_test", "comment_count_in_10_test", "like_count_in_10_test",
                  "collect_count_in_10_test"], axis=1, inplace=True)
df_for_test_2
```

trending year month weekday day hour share_count_in_10_testlog comment_count_in_10_testlog like_count_in_10_testlog collect_count_in_10_testlog

False	2019	11	1	19	9	0.693147	2.197225	3.526361	1.945910
False	2019	11	4	15	14	0.000000	3.332205	2.564949	0.693147
False	2019	12	0	16	6	0.000000	2.197225	0.693147	0.693147
False	2019	12	4	20	10	0.000000	4.553877	2.833213	1.098612
False	2019	11	0	18	2	0.000000	1.386294	3.135494	2.484907
...

225986 rows × 10 columns

對呼叫出的模型進行測試：

```
#Testing the model trained in TH_train.py file
predictors_test = df_for_test_2.drop(["trending"], axis=1)
outcome_test = df_for_test_2["trending"]

outcome_test_prdic_pr = trained_model.predict_proba(X = predictors_test)
outcome_test_prdic = (outcome_test_prdic_pr[:,1] >= 0.39).astype('int')
print(classification_report(outcome_test, outcome_test_prdic))
```

	precision	recall	f1-score	support
False	0.99	0.99	0.99	221479
True	0.62	0.53	0.57	4507
accuracy			0.98	225986
macro avg	0.81	0.76	0.78	225986
weighted avg	0.98	0.98	0.98	225986

預測結果的table形式：

```
#Output as a excel file
predictors_test['is_trending'] = outcome_test_prdic
result_xlsx = predictors_test.drop(predictors_test.columns[:9], axis=1)
result_xlsx
```

	is_trending
post_key	
00017cc1-df93-4ce1-be7b-0b3c76cb3dc6	0
00021bc3-7699-4c97-9ec5-20edaac60cc1	0
000295a1-63ed-4081-b55a-a9b7648eaa7c	0
0005d1a6-d21e-4f4b-b753-d0cf5992e136	0
00062d15-4ee0-4a6c-a5f5-44113dc3fb41	0
...	...
f2211e58-2c33-4843-bf4a-12dbc44c87b6	0
bc82ddef-eeeb-42a4-8e40-7ae017295874	0
5f76f740-4f89-4060-8218-4f12d923af0c	0
c7dc5d73-87e7-4abf-bdc2-5e97e3448e9d	0
1f65e657-5b51-4dc9-a836-13f8db0759f9	0

225986 rows × 1 columns

```
import os
os.getcwd()
```

```
'C:\\Users\\lawre\\TH_Portfolio'
```

最後，將預測結果的table輸出成csv檔：

```
result_xlsx.to_csv('Result.csv', encoding = 'utf-8', index = True)
```

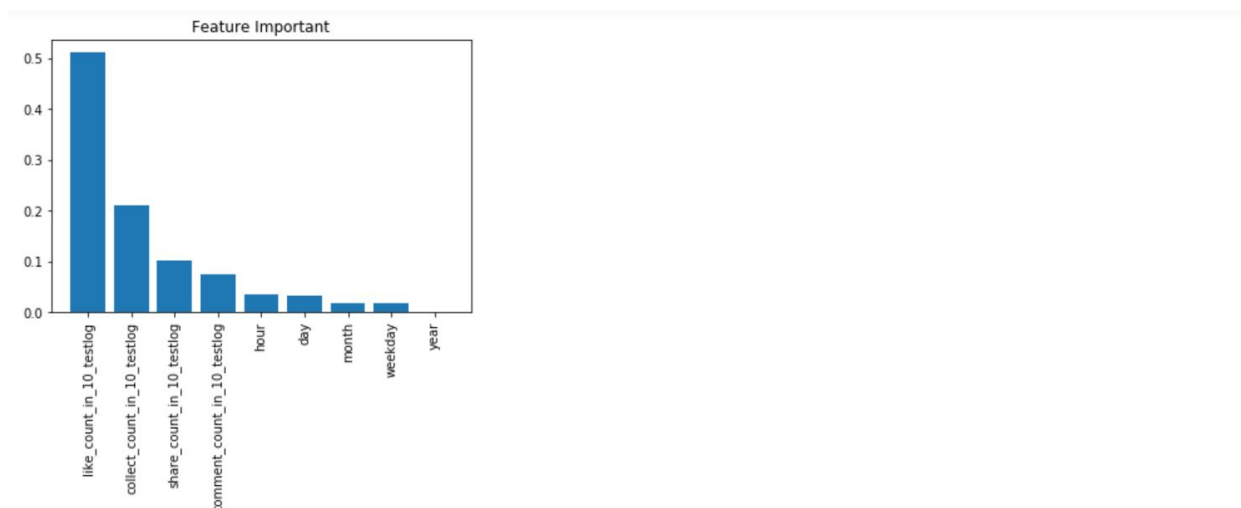
```
1 post_key,is_trending
2 00017cc1-df93-4ce1-be7b-0b3c76cb3dc6,0
3 00021bc3-7699-4c97-9ec5-20edaac60cc1,0
4 000295a1-63ed-4081-b55a-a9b7648eaa7c,0
5 0005d1a6-d21e-4f4b-b753-d0cf5992e136,0
6 00062d15-4ee0-4a6c-a5f5-44113dc3fb41,0
7 000735bf-45db-45ba-bcba-44774896104e,0
8 000897fc-7d83-4a0d-88c1-0e0b815c9edc,0
9 00091347-f0fe-486d-aa9f-cb92f5101d63,0
10 000b727c-0d20-4d68-ae8c-4a7eba2d336b,0
11 000c4253-8399-450b-800d-330628b6e4b8,0
12 000f9540-0306-4210-a25a-f2a77f4d774b,0
13 00141d00-7ba5-4e3a-80d2-31f1e246d514,0
14 001af345-8640-4ed6-a934-03ba56b2ed12,0
15 001b8a46-564a-4c6f-aab3-ee798f9eeda8,0
16 001bdb8c-3f63-4b94-b45e-5f5bab415e40,0
17 001c3095-1f39-434e-819d-0b847118ee20,0
18 00212e82-de75-48f7-809a-449e1f202ede,0
19 00214cfe-c06a-4f34-8a52-ed20c797edea,0
20 0023d107-4169-4f5f-b44c-24cb9a068a9e,0
21 00249878-a5fd-4366-801b-70defe8a1e6a,0
22 0024e224-4302-48b1-8966-2a52c7b1006c,0
23 002fdad4-8978-4ad6-a158-fc71a1a203eb,0
24 00328f47-9e8f-4107-be34-f98414b4caca,0
25 003b74b0-b421-4a77-a218-851cee0b23a1,0
26 003dd0ff-631e-4855-80c2-2cddb481ee069,0
27 0047d525-0088-4d1b-8f77-9e957005370d,1
```

Conclusion

根據最終選用的Random Forest模型在測試dataset的結果（如下圖），其macro-average f1 score為0.78。以其作為評估指標（而非weighted-average f1 score）乃因（一）：dataset高度不平衡，若採用weighted-average f1 score，會使此值虛高；（二）：假設**推薦使用者一偽熱門文章（即預測為熱門文章，但最後卻非熱門文章）**的成本，與**錯失推薦一潛在熱門文章給使用者（即最後發現是熱門文章，當初卻因為預測錯誤而未推薦）**的成本，兩者相等，則「將分類權重視為相等的macro-average f1 score」會是比較好的選擇。

	precision	recall	f1-score	support
False	0.99	0.99	0.99	221479
True	0.63	0.52	0.57	4507
accuracy			0.98	225986
macro avg	0.81	0.76	0.78	225986
weighted avg	0.98	0.98	0.98	225986

同理，若將上述兩成本視之相等，則precision rate與recall rate這兩個被視為trade-off的指標同等重要，沒有一個可被偏廢，因此在兩者間取得良好的平衡亦為選擇此Random Forest預測模型的主要原因，這從其擁有相近的macro-average precision以及recall rate（分別為0.81與0.76），便可體現。而在訓練階段所使用的boosting模型，如Gradient Boosting與AdaBoosting皆有高recall rate但低precision rate，比較適合在**錯失推薦一潛在熱門文章給使用者的成本「遠大於」推薦使用者一偽熱門文章**的條件下使用。而XGBoosting模型則擁有只略低於此Random Forest模型的macro-average f1 score，但其有稍高的recall rate以及較低的precision rate，因此較適合**錯失推薦一潛在熱門文章給使用者的成本「略大於」推薦使用者一偽熱門文章**。但整體而言，基於兩者成本一樣的假設下，此Random Forest模型乃最佳模型。



最後，根據此模型的feature importance結果（如上圖），可發現發文十小時內的被愛心數對於熱門文章最具預測力，其次為收藏、分享與留言數，而發文時間（不論是幾點、甚至星期幾）則相對不具預測力。從行為科學的角度，前四項指標屬於indicators of social norms/preferences，閱讀者能從這些指標得知他人對此文章的態度，許多研究皆顯示 (Allcott, 2011)，得知他人的態度（例如有人也點愛心），會鼓勵該使用者也點愛心。從此角度來解讀此feature importance，前四項指標高於後四項時間點指標便具合理性。

此外，在Dcard的使用介面中（如下圖），愛心數（包含其他emojis）和留言數是兩個出現在標題下的量化指標。因此，此兩項指標具備「在不須點入內文觀看的情況下」向使用者暗示其他閱聽人對此文章的態度之功能。就上述行為科學的角度，在標題下顯示 indicators of social norms/preferences亦可鼓勵使用者點擊該文章閱讀，而當能觸發愈多人閱讀該文章，相對應的愛心數、分享數、留言數與收藏數亦會因此提升（因為閱讀基數提升），此文章變為熱門文章的機率也就提高。因此，身為兩個與標題一同顯示的量化指標之一，愛心數成為最具預測力的變數亦得到合理解釋。

而留言數之所以無法具備相同效果，可能原因有二。第一，在Random Forest演算法中，若兩自變數高度相關，彼此的重要性會相互嚴重影響，最終會導致只有其中之一具高重要性，而另外之一將變得微不足道（類似線性回歸中的covariance）；若是此原因，則前者為愛心數，後者就為留言數。原因二，相較於點擊愛心只需一個動作（one click），留言則需要花費相對較多的時間，此將影響留言數對熱門文章的預測力。故基於以上討論結果，本報告建議，若與標題一同顯示並具備"one click"特性的indicators of social norms/preferences（如愛心數、分享數與收藏數）能提升觸擊率，則可以考慮將上述三指標一起或擇二放置在標題下，並透過A/B Test找出最佳組合（與原本的愛心+留言數比較）。



YouTuber

追蹤

熱門 最新 板規



(˘▽˘)ノ♪ · 4月15日 12:45

米鹿準備開始轉換跑道了(?)

感覺要發一波新單曲或rap了，連最後的押韻都不放過呢👉



2174

回應

171



收藏



國立中興大學 · 4月15日 13:58

應該有更好的處理方式吧

阿圓說話了



557

回應

34



收藏

Reference

1. Pinto, H., Almeida, J. M., & Gonçalves, M. A. (2013, February). Using early view patterns to predict the popularity of youtube videos. *In Proceedings of the sixth ACM international conference on Web search and data mining* (pp. 365-374).
2. Sunstein, C. R. (2019). *How change happens*. Mit Press.
3. Sabate, F., Berbegal-Mirabent, J., Cañabate, A., & Lebherz, P. R. (2014). Factors influencing popularity of branded content in Facebook fan pages. *European Management Journal*, 32(6), 1001-1011.
4. Allcott, H. (2011). Social norms and energy conservation. *Journal of public Economics*, 95(9-10), 1082-1095.