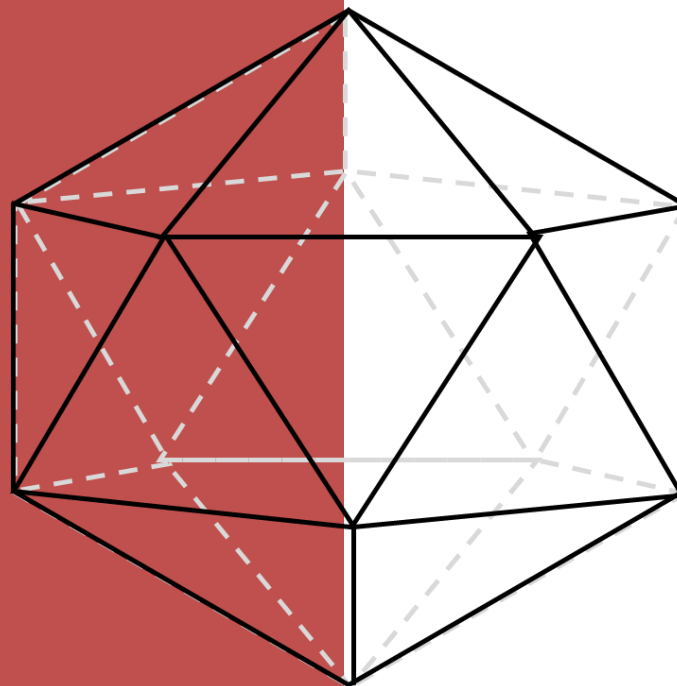


第一組

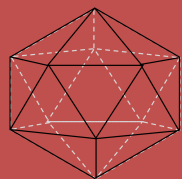


# 巨量資料與金融實務期末報告

會延A 03152138 張永霖  
經延C 03151316 譚聿彰  
企四B 04115102 劉學梅  
企四C 04115235 邱千芳  
企四C 04153330 俞宣卉

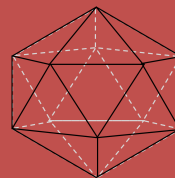
# 目 錄

## CONTENTS



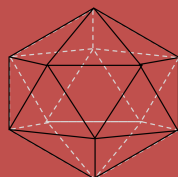
### 1.製作動機

- 一、前言
- 二、資料樣本
- 三、因子使用

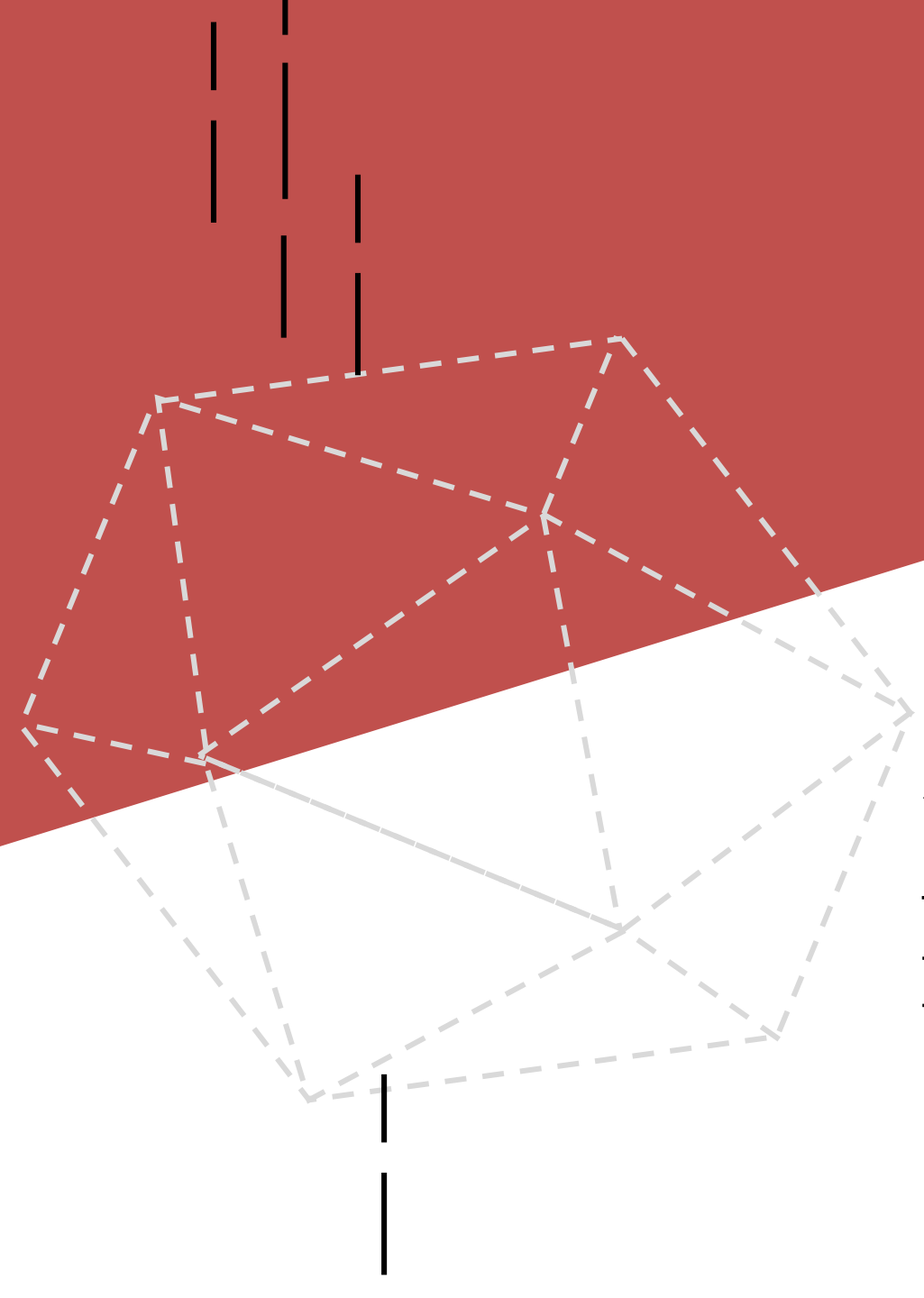


### 2.研究方法

- 一、Random Forest
- 二、XGBoost
- 三、LightGBM



### 3.成果展現

An abstract geometric diagram featuring a complex network of dashed lines forming a wireframe structure. The diagram is set against a background that is split diagonally from the bottom-left to the top-right, with a dark red upper half and a white lower half. Several vertical dashed lines extend from the top edge of the red section into the white section. The wireframe structure consists of multiple interconnected polygons, with some lines crossing each other.

# 製作動機

- 一、前言
- 二、資料樣本
- 三、因子使用

+

## 前言



### 預測下期IC

使用資料處理讀取、因子歷史IC值來  
預測因數的下一期 IC



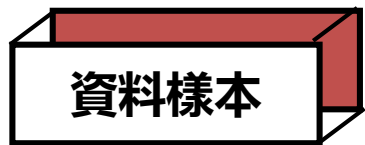
### 預測下期收益率

得到數據並作出排名之後，從而了解  
因子選股帶來的超額收益能力

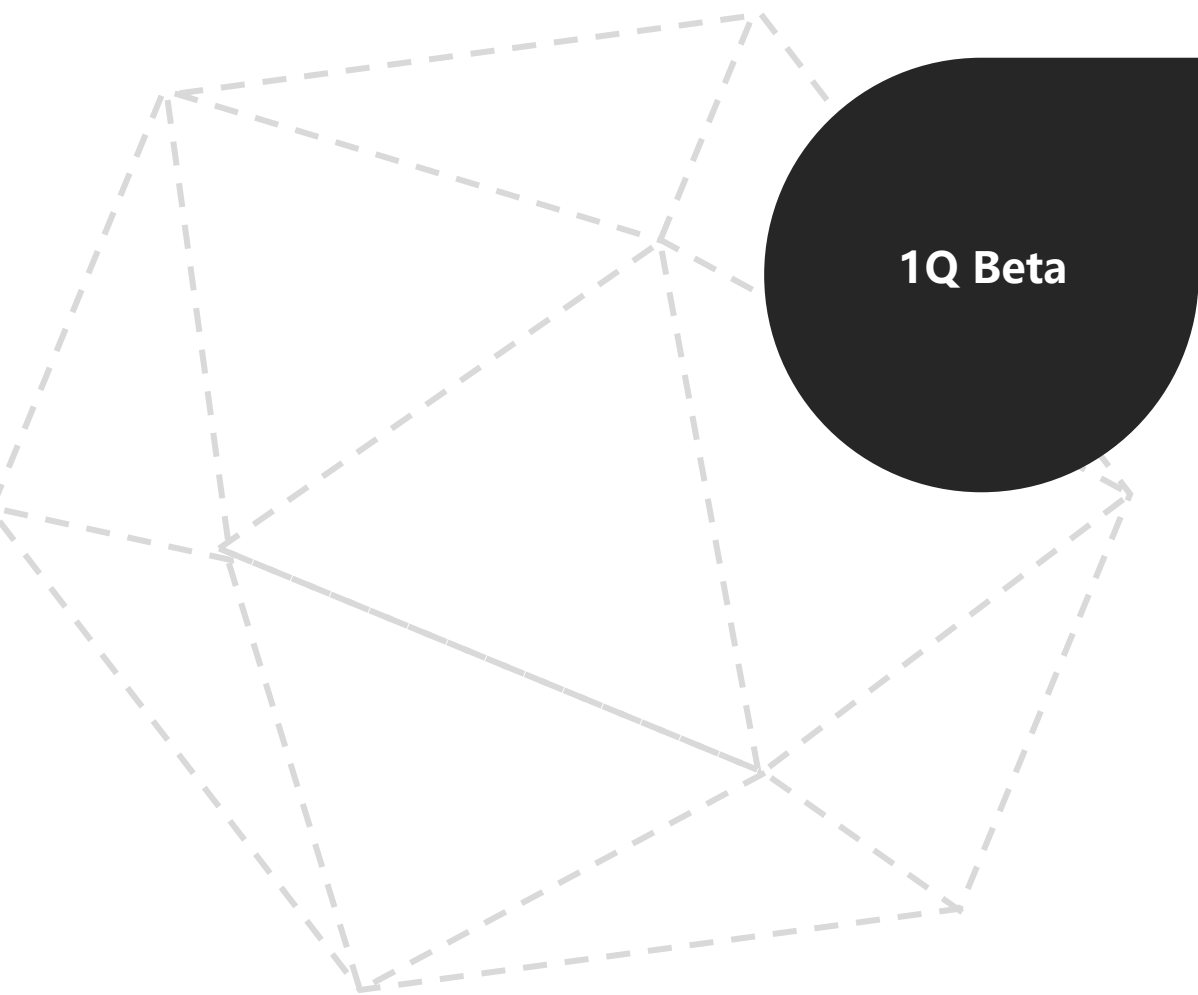


### 透過模型預測未來

透過模型預測，了解下一年度投資哪些公司較好，  
並提高未來能夠獲利的機率及穩定性。



元大台灣 50 ( 0050) + 元大高股息(0056)ETF





# 研究方法

- 一、Random Forest
- 二、XGBoost
- 三、LightGBM

+

## 一、Random Forest

- 一、使用scikit-learn package的Random Forest演算法對IC值進行預測。  
(這邊我們對於原始的參數值進行一些修改。)

```
from sklearn.ensemble import RandomForestRegressor  
regr = RandomForestRegressor(max_depth=2, random_state=0, n_estimators=100)
```

- 二、將整理好的Alpha X、Alpha Y值餵入regr.fit(X,Y)中，進行模型的訓練。  
如果輸出正確則證明使用隨機森林演算法對股票預測訓練成功。

```
In [53]: X=X_features_AA  
         Y=X_target_AA  
         regr.fit(X,Y)
```

```
Out[53]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=2,  
                                max_features='auto', 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=100, n_jobs=None,  
                                oob_score=False, random_state=0, verbose=0, warm_start=False)
```



## 一、隨機森林演算法

三、跑出Alpha的IC預測值。

```
print(regr.predict(y_features_AA))  
[0.37054566]
```

四、一樣步驟處理Beta的IC預測值。

```
In [56]: x=X_features_BB  
         y=X_target_BB  
         regr.fit(x,y)  
  
Out[56]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=2,  
                                max_features='auto', 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=100, n_jobs=None,  
                                oob_score=False, random_state=0, verbose=0, warm_start=False)  
  
In [57]: #print(regr.feature_importances_)  
  
In [58]: print(regr.predict(y_features_BB))  
[-0.05393364]
```

**結論：**  
使用Alpha預測的IC值  
優於Beta所預測的IC值。

## 二、XGBoost

### 一、使用XGBoost的回歸模型，進行Alpha與Beta的IC預測。

```
In [42]: import xgboost  
xgb=xgboost.XGBRegressor(max_depth=50)  
xgb.fit(X,Y)
```

```
Out[42]: XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,  
colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,  
max_depth=50, min_child_weight=1, missing=None, n_estimators=100,  
n_jobs=1, nthread=None, objective='reg:linear', random_state=0,  
reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,  
silent=True, subsample=1)
```

```
In [43]: print(xgb.predict(y_features_AA))  
[0.3414655]
```

```
In [44]: import xgboost  
xgb=xgboost.XGBRegressor(max_depth=50)  
xgb.fit(x,y)
```

```
Out[44]: XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,  
colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,  
max_depth=50, min_child_weight=1, missing=None, n_estimators=100,  
n_jobs=1, nthread=None, objective='reg:linear', random_state=0,  
reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,  
silent=True, subsample=1)
```

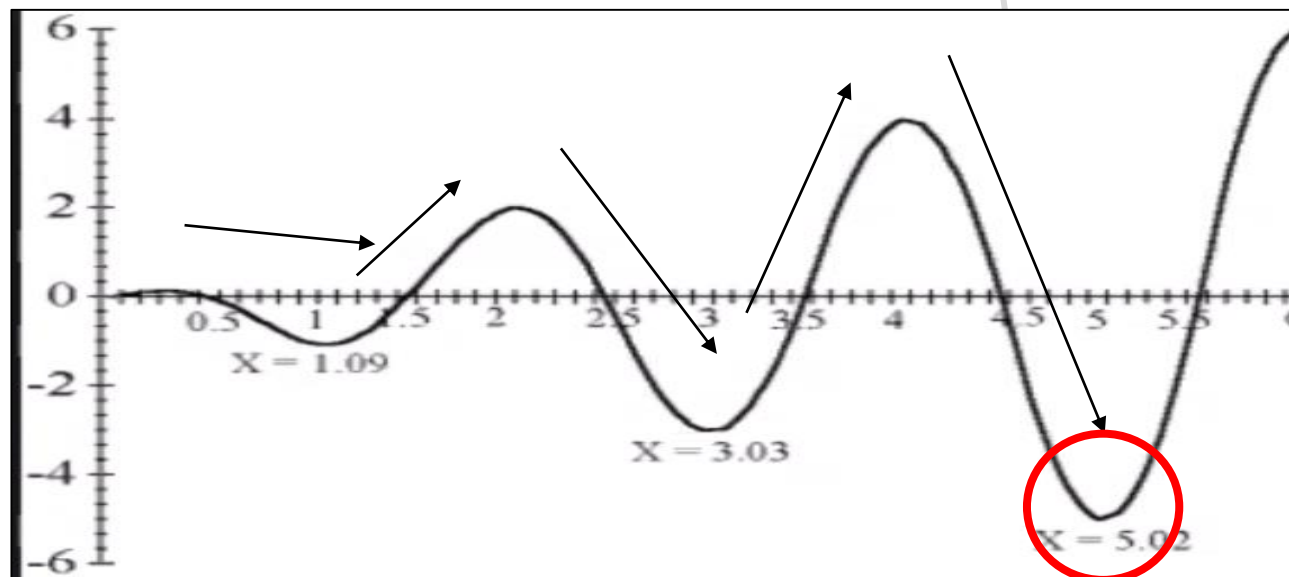
```
In [45]: print(xgb.predict(y_features_BB))  
[-0.1027208]
```

**結論：使用Alpha預測的  
IC值較Beta所預測的佳。**

### 三、LightGBM

使用MSE 作為lost function，尋找global minimum，並找出誤差較小比較符合的模型。

- 好處：好操作，特點會近似值會非常準確，減少計算成本
- 原理：LightGBM = Histogram-based algorithm + GOSS + EFB
  - LGB模型的原理是從繞圈的方式，跑過所有的資料，一步步找到資料的最適化的點。一般gdbt演算法隨著跑的資料越多，要花的時間複雜度越大，但LightGBM可以透過GOSS大大降低計算成本。



### 三、LightGBM

#### 1. Histogram-based algorithm直方圖優化演算法:

將連續的數字，先分入不同的資料群，用梯度來分不同父節點，再沿著父節點找到左節點和右節點，長成一顆樹。



GOSS：尋找梯度進行排序，對之前分錯的重新分類，用前一iteration當一個參考，進行訓練。

### 三、LightGBM

#### 2. GOSS 優點:

##### 計算成本



GOSS 可以大大減少計算成本

##### 近似誤差 approximation error



在  $n \rightarrow \infty$  時,  $O(\sqrt{n})$ , 代表近似值會非常準確。

##### 泛化性能 generalization ability:



如果GOSS近似值是準確的, 則GOSS 的泛化誤差將接近使用完整資料實例計算的誤差。抽樣會增加training model 的多樣性, 這可能有助於提高泛化性能。

GOSS 的計算步驟如下:

1. 排列樣本資料: 根據樣本的梯度將樣本降冪 (decending) 排序。
2. 建立梯度值較大的資料樣本: 保留前 topN (a%) 個資料樣本, 作為資料子集 topSet。  
→  $\text{topSet} = \text{sorted}[1:\text{topN}]$   
 $\text{topN} = (a\%) \times \#\text{samples}$
3. 建立梯度值較小的資料樣本: 對於剩下的資料的樣本 (1-a%), 隨機抽樣獲得大小為 randN (b%) 的資料子集 randSet。  
→  $\text{randSet} = \text{RandomPick}(\text{sorted}[\text{topN}:], \text{randN})$   
 $\text{randN} = (b\%) \times \#\text{samples}$
4. 產生 New Data Set (計算 information gain): 為了不改變原資料分佈, 在計算 information gain 時, 對 randSet 樣本的梯度資料乘以權重:  $w = \frac{1-a}{b}$   
→  $\text{UsedSet} = \text{topSet} + \text{randN}$   
 $\text{UsedN} = (a\%) \times \#\text{samples} + (b\%) \times \#\text{samples}$

### 三、LightGBM

#### 3. EFB

LightGBM 從直方圖優化演算法的想法出發，EFB 演算法也是用 bundle 的方法進行互斥特徵的合併，從而減少特徵的數目，是 LightGBM 用來降維的演算法。

**one-hot encoding 所產生大量的類別特徵**，每多一個特徵就增加一個維度。而 GBDT 演算法在劃分節點時，需要考慮所有特徵，因此 EFB 演算法對不同維度的資料合併在一起使得一個稀疏陣列變成一個稠密矩陣，減少不必要的特徵(降維)。減少時間複雜度。加速訓練。

$$\text{Sparse (大)} : \begin{bmatrix} 0 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \rightarrow \text{Dense (小)} : \begin{bmatrix} 9 & \cdots & -1 \\ \vdots & \ddots & \vdots \\ 5 & \cdots & 12 \end{bmatrix}$$

### 三、LightGBM

操作方法：

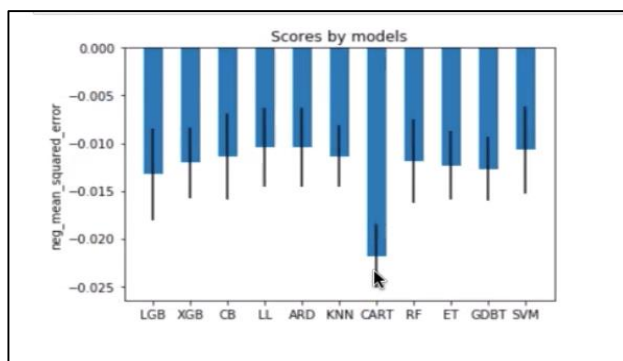
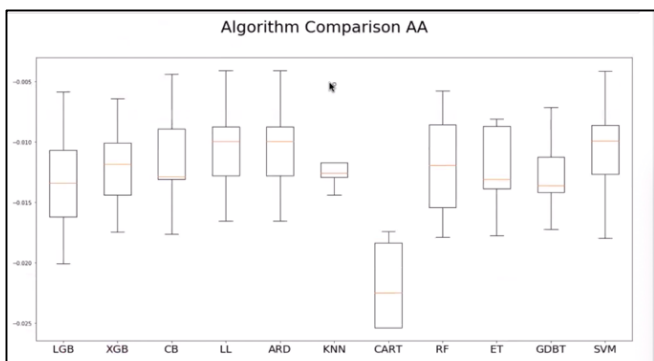
1. 設定好loss function：因為這是一種regression的預測，使用neg\_mean\_squared\_error作為loss function標竿。

```
: %%time
# evaluate each model in turn
results = []
names = []
means = []
stds = []
scoring = 'neg_mean_squared_error'
for name, model in models:
    kfold = model_selection.KFold(n_splits=5, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X_features_AA, X_target_AA, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    means.append(cv_results.mean()) #
    stds.append(cv_results.std()) #
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

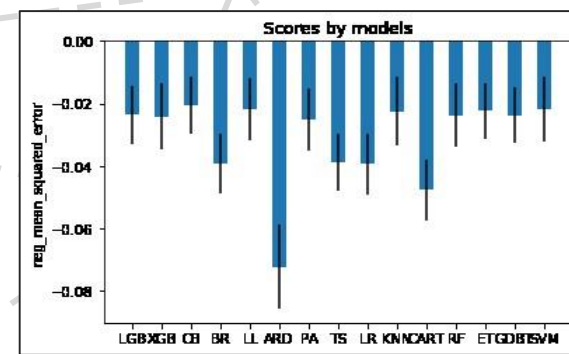
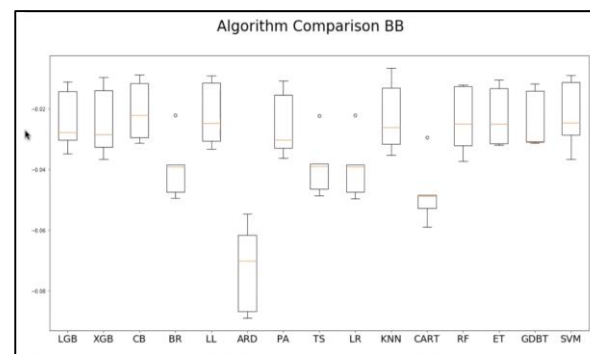


### 三、LightGBM

2. 選擇使用的模型：利用cv先進行選用model，將結果因子進行5fold交叉驗證，找一個不會MSE誤差很大的模型（圖一、二、三），Alpha值比較，如果超過值2就不使用，例如CART誤差值大，最後選擇出LightGBM模型來使用。



<圖一、二：誤差比較AA、平均的rmse AA >



<圖三：誤差比較BB、平均的rmse BB >

小結：最後選擇用LGB模型。



### 三、LightGBM

#### 3. 調整參數：

LightGBM模型原本的參數有很多（圖四），利用skopt 套件的BayesSearchCV 來做，優點是可以依據前一次調整決定下一個要測的參數值。

```
from sklearn.cross_validation import KFold
folds = KFold(n_splits=5, shuffle=True, random_state=1)

#lgb tuning parameter
lgb_search_space = {
    'colsample_bytree': (0.01, 1.0, 'uniform'),
    'learning_rate': (0.01, 1.0, 'log-uniform'),
    'max_depth': (3, 10),
    'min_child_samples': (0, 50),
    'min_child_weight': (0, 10),
    'n_estimators': (100, 400),
    'num_leaves': (10, 100),
    'max_bin': (100, 1000),
    'subsample': (0.5, 1.0, 'uniform'),
    'subsample_freq': (0, 10),
    'reg_lambda': (1e-9, 1000, 'log-uniform'),
    'reg_alpha': (1e-9, 1.0, 'log-uniform'),
    'scale_pos_weight': (1e-6, 500, 'log-uniform'),
}
```

<圖四：LGB模型最原本的參數>

GridSearch	RandomSearch	BayesSearch
X	X	O

lgbmodel

```
LGBMRegressor(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
importance_type='split', learning_rate=0.1, max_depth=-1,
min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
n_estimators=100, n_jobs=-1, num_leaves=31, objective='regression',
random_state=50, reg_alpha=0.0, reg_lambda=0.0, silent=1,
subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
```

<圖五：LGB模型調整後的參數>

### 三、LightGBM

#### 3. 調整參數:

用Kfold將資料折成5份（圖六），準備好訓練參數，放入BayesSearchCV進行調參，命名為lgb\_bayes\_tuner 調整參數，跑出最好參數結果（圖七）。

```
# LGBMRegressor
lgbmodel = lgb.LGBMRegressor(boosting_type='gbdt', random_state=50, objective='regression', silent=1)

folds = KFold(n_splits=5, shuffle=True, random_state=1)

#lgb tuning parameter
lgb_search_space = {
    'colsample_bytree': (0.01, 1.0, 'uniform'),
    'learning_rate': (0.01, 1.0, 'log-uniform'),
    'max_depth': (3, 10),
    'min_child_samples': (0, 50),
    'min_child_weight': (0, 10),
    'n_estimators': (100, 400),
    'num_leaves': (10, 100),
    'max_bin': (100, 1000),
    'subsample': (0.5, 1.0, 'uniform'),
    'subsample_freq': (0, 10),
    'reg_lambda': (1e-9, 1000, 'log-uniform'),
    'reg_alpha': (1e-9, 1.0, 'log-uniform'),
    'scale_pos_weight': (1e-6, 500, 'log-uniform'),
}
```

<圖六： KFold資料折五份 >

```
BayesSearchCV(cv=KFold(n_splits=5, random_state=1, shuffle=True),
              error_score='raise',
              estimator=LGBMRegressor(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
              importance_type='split', learning_rate=0.1, max_depth=-1,
              min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
              n_estimators=100, n_jobs=-1, num_leaves=31, objective='regression',
              random_state=50, reg_alpha=0.0, reg_lambda=0.0, silent=1,
              subsample=1.0, subsample_for_bin=200000, subsample_freq=0),
              fit_params=None, iid=True, n_iter=100, n_jobs=3, n_points=1,
              optimizer_kwargs=None, pre_dispatch='2*n_jobs', random_state=1,
              refit=True, return_train_score=True,
              scoring='neg_mean_squared_error',
              search_spaces={'colsample_bytree': (0.01, 1.0, 'uniform'), 'learning_rate': (0.01, 1.0, 'log-uniform'), 'max_depth': (3, 10), 'min_child_samples': (0, 50), 'min_child_weight': (0, 10), 'n_estimators': (100, 400), 'num_leaves': (10, 100), 'max_bin': (100, 1000), 'subsample': (0.5, 1.0, 'uniform'), 'subsample_freq': (0, 10), 'reg_lambda': (1e-09, 1000, 'log-uniform'), 'reg_alpha': (1e-09, 1.0, 'log-uniform'), 'scale_pos_weight': (1e-06, 500, 'log-uniform')},
              verbose=3)
```

<圖七： 最好的參數結果 >

### 三、LightGBM

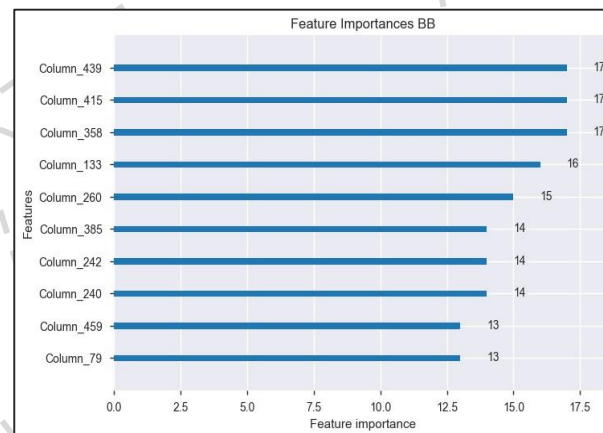
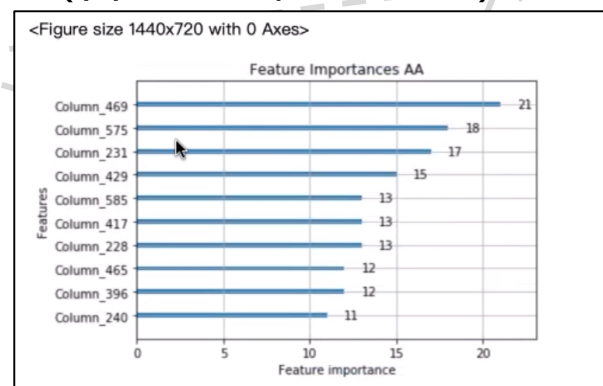
#### 3. 調整參數:

最後訓練成模型gbm\_AA\_tuned (圖八) 結果為0.37851837。

用LightGBM模型可以找出最重要的特徵 (圖九), 例如在這邊是469欄 (台灣指標性的公司)。

```
%%time
gbm_AA_tuned = lgb_bayes_tuner.best_estimator_
gbm_AA_tuned.fit(X_features_AA, X_target_AA)
y_pred_lgb_AA_tuned = gbm_AA_tuned.predict(y_features_AA)
print(y_pred_lgb_AA_tuned)
```

<圖八：組成gbm\_AA >



<圖九：LGB模型特點會跑最重要的欄位 >

### 三、LightGBM

#### 3. 調整參數:

實際值為0.42205，從0.3655調整到0.3785183 有好一些，會有差異是因為資料不足夠。  
結果得到Alpha值0.422 > Beta 0.05951， 所以使用Alpha（圖十）

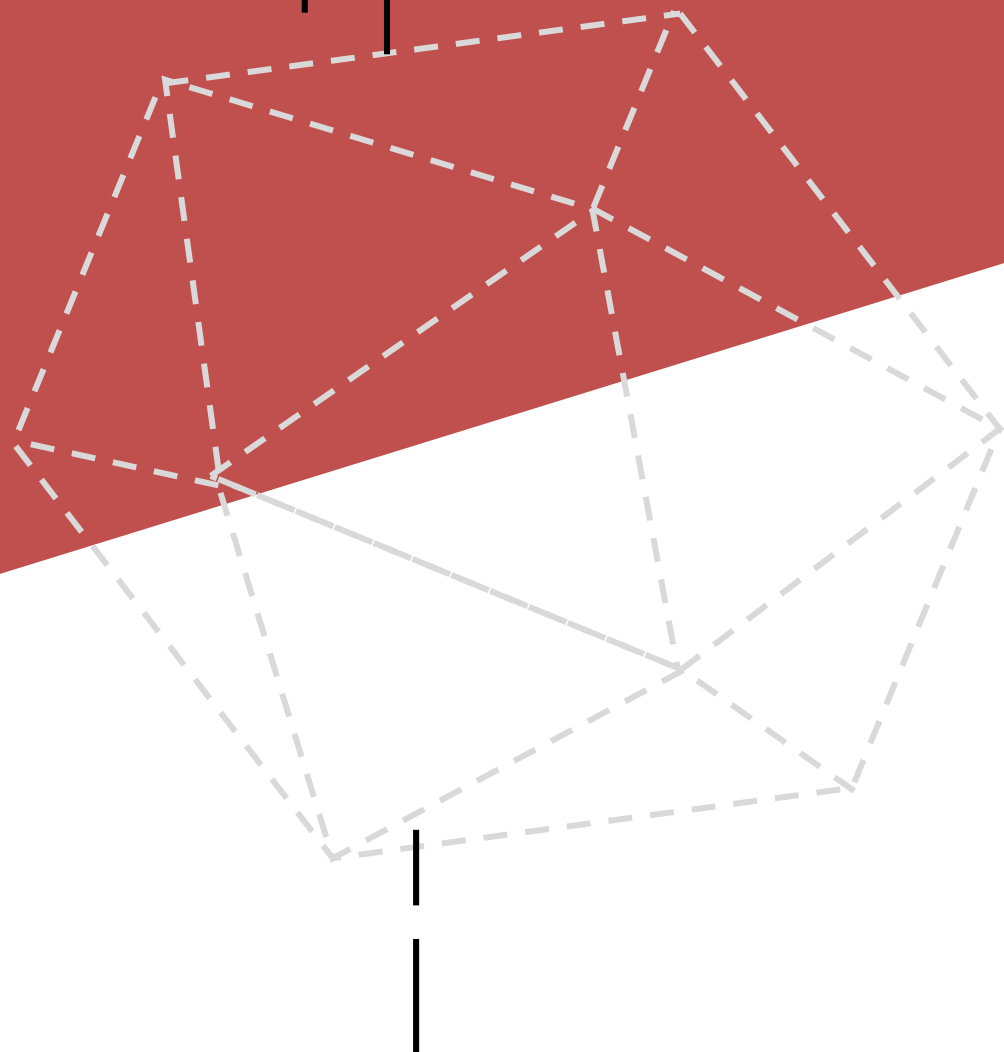
```
print(AA.tail(1).iloc[:, -1].values, y_pred_lgbAA, y_pred_lgb_AA_tuned)
```

```
[0.42205644] [0.36553518] [0.37851837]
```

```
print(BB.tail(1).iloc[:, -1].values, y_pred_lgbBB, y_pred_lgb_BB_tuned)
```

```
[0.05951859] [-0.07004206] [-0.04531292]
```

<圖十：調整值和實際上的值 >



**成果展現**

+

## 成果展現

以下為使用因子、IC值以及模型，預測出來的結果：

1. 做完模型預測之後，可以得到以下表格：

預測IC值	Random Forest	XGBoost	LGB(已調參數)
Alpha	0.37054566	0.3414655	0.37181573
Beta	-0.05393364	-0.1027108	-0.04531292

可以得出第一個小結論：  
使用Alpha預測的IC值都會較Beta所預測的更好，因此使用Alpha。

## 成果展現

進行Blending:

### Blending

```
blendAA = y_pred_lgb_AA_tuned/3+y_pred_rf_AA/3+y_pred_xgb_AA/3  
print(blendAA)
```

```
[0.3621929]
```

```
blendBB = y_pred_lgb_BB_tuned/3+y_pred_rf_BB/3+y_pred_xgb_BB/3  
print(blendBB)
```

```
[-0.07441405]
```

```
blendAA > blendBB
```

```
array([ True])
```

三個模型合併後，可以減少bias



## 成果展現

2.利用比較準確的因子Alpha預測IC值，進行排序找出前後10%的公司（後來改成前1%）。

前10%

AA\_eval\_top

```
Index(['2415 錫新', '1410 南染', '5388 中磊', '1529 樂士', '4306 炎洲', '2337 旺宏',  
      '6128 上福', '2436 偉詮電', '2375 智寶', '3005 神基', '1110 東泥', '1532 勤美',  
      '6213 聯茂', '2480 敦陽科', '2327 國巨', '2429 銘旺科', '1474 弘裕', '1718 中纖',  
      '1470 大統新創', '2440 太空梭', '2029 盛餘', '1541 錫泰', '6120 達運', '2613 中櫃',  
      '2358 廷鑫', '9934 成霖', '2377 微星', '6230 超眾', '1603 華電', '2450 神腦',  
      '3056 總太', '2472 立隆電', '3057 喬鼎', '1457 宜進', '9935 慶豐富', '1512 瑞利',  
      '1516 川飛', '6168 宏齊', '1438 裕豐', '2851 中再保', '2540 愛山林', '1537 廣隆',  
      '2387 精元', '1325 恒大', '1609 大亞', '6165 捷泰', '1215 卜蜂', '9908 大台北',  
      '9942 茂順', '9918 欣天然', '1732 毛寶', '2601 益航', '1232 大統益', '2548 華固',  
      '1218 泰山', '6142 友勁', '4906 正文', '1312 國喬', '2912 統一超', '1447 力鵬'],  
      dtype='object', name='公司')
```

後10%

AA\_eval\_bot

```
Index(['2024 志聯', '1904 正隆', '1414 東和', '2836 高雄銀', '3013 晟銘電', '6176 瑞儀',  
      '1477 聚陽', '2707 晶華', '2524 京城', '5607 遠雄港', '6243 迅杰', '9907 統一實',  
      '9919 康那香', '2020 美亞', '6192 巨路', '2477 美隆電', '1810 和成', '2342 茂矽',  
      '2467 志聖', '2888 新光金', '6205 詮欣', '1720 生達', '2504 國產', '1451 年興',  
      '9937 全國', '3027 盛達', '2615 萬海', '1708 東鹼', '6183 關貿', '1316 上曜',  
      '2832 台產', '2007 燁興', '2606 裕民', '1614 三洋電', '2855 統一證', '2412 中華電',  
      '2906 高林', '2462 良得電', '2823 中壽', '9914 美利達', '2002 中鋼', '2458 義隆',  
      '2427 三商電', '3044 健鼎', '2434 統懋', '2547 日勝生', '1521 大億', '2006 東和鋼鐵',  
      '2882 國泰金', '2884 玉山金', '2890 永豐金', '2887 台新金', '1464 得力', '2881 富邦金',  
      '6108 競國', '2475 華映', '2371 大同', '3051 力特', '2105 正新', '1326 台化'],  
      dtype='object', name='公司')
```



## 成果展現

3. 若想要得到最高報酬，應買進最高等份的投資組合再賣掉最低等份的投資組合，並得到平均報酬率。

```
In [187]: AA_eval = AA.T.sort_values(by = AA.T.columns[-2], axis=0, ascending=False).iloc[:, :-1]
AA_eval_top = AA_eval.head(int(len(AA_eval)*(1/100))).index
AA_eval_bot = AA_eval.tail(int(len(AA_eval)*(1/100))).index
```

```
In [188]: AA_eval_top
```

```
Out[188]: Index(['2415 鋁新', '1410 南染', '5388 中磊', '1529 樂士', '4306 炎洲', '2337 旺宏'], dtype='object', name='公司')
```

```
In [189]: AA_eval_bot
```

```
Out[189]: Index(['6108 競國', '2475 華映', '2371 大同', '3051 力特', '2105 正新', '1326 台化'], dtype='object', name='公司')
```

```
In [190]: AA_topten = sum(P_return.loc[AA_eval_top, P_return.columns[-2]])
print(AA_topten)
```

```
0.8601356241342913
```

```
In [191]: AA_botten = sum(P_return.loc[AA_eval_bot, P_return.columns[-2]])
print(AA_botten)
```

```
-0.5754834215809049
```

```
In [192]: len(AA_eval_top)
```

```
Out[192]: 6
```

```
In [193]: len(AA_eval_bot)
```

```
Out[193]: 6
```

```
In [195]: (AA_topten/len(AA_eval_top) - AA_botten/len(AA_eval_bot))
```

```
Out[195]: 0.23926984095253273
```

由此可知，報酬率為  
**23.92698409%**

## 成果展現

4. 最後找出當月的市場指數報酬，並做比較。（2019年1月）

台指平均報酬率

$= \log(1\text{月底價格}) - \log(\text{前一個月底價格})$

$= 0.905085838\%$

$\log_{10}(9\ 932.26) - \log_{10}(9\ 727.41) =$

**0.00905085838**

<

預測的當月報酬率 = 23.92698409%

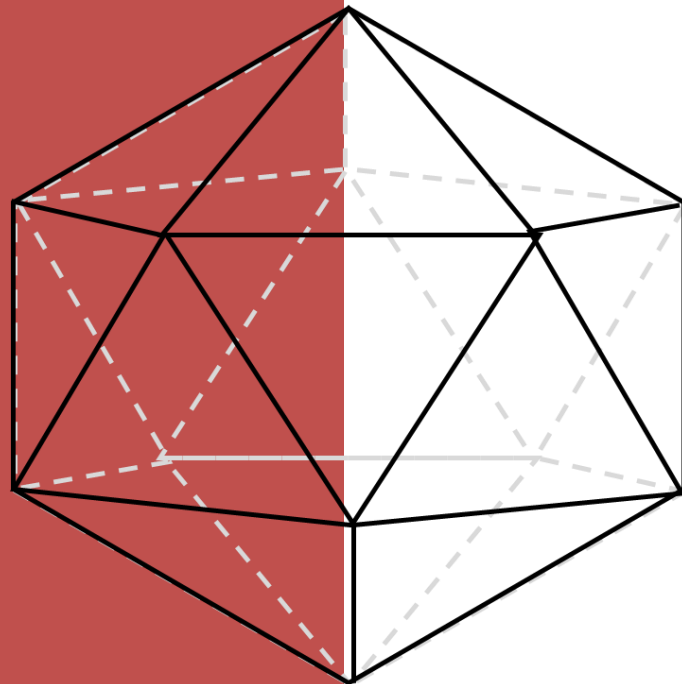
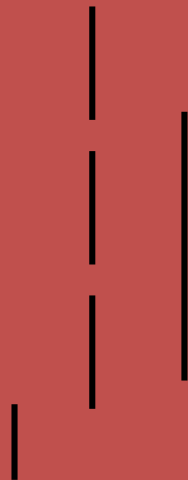
$(\text{AA\_topten} / \text{len}(\text{AA\_eval\_top}) - \text{AA\_botten} / \text{len}(\text{AA\_eval\_bot}))$

0.23926984095253273

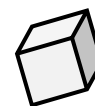
結論：

運用模型預測出來的報酬率 > 實際報酬率（超額報酬）。

+



+



謝 謝 大 家