- Data

  Train.csv：# 100,000 (Y：# 2,000 / N：# 98,000)

  Test.csv ：# 150,000

類別型(有序)	Ex：{低、中、中高、高}	共 6 個
類別型(是非)	Ex：{Y、N}	共 79 個
類別型(名目)	Ex ：{M(男)、F(女)}	共 4 個
數值型(連續)	Ex：{0.125、0.375、0}	共 21 個
數值型(離散)	Ex ：{0、1、2、3}	共 20 個

以上，是我們將所有的原始資料除去「CUS_ID」及目標變數「Y1」後稍作分類，以便於後續的資料整理。

- Data Cleaning

Categorical columns

1) Ordinary Features

```
def order_features(df):
    order_mapping = {'低':1,'中':2,'中高':3,'高':4}
    col=['AGE','APC_1ST_AGE','INSD_1ST_AGE','RFM_R','REBUY_TIMES_CNT','LIFE_CNT']
    for i in col:
        df[i] = df[i].map(order_mapping)
        df[i] = df[i].fillna(0)
    return df
```

針對「有序特徵」給予由小到大之整數進行相應的替換，空值的部分因其無法帶來可比較的訊息，以「零值」做填補。

2) Binary Features

```
def Y_N(df):
    count = 0
    transform={'Y':1,'N':0}
    for i in df.columns:
        if re.match(r'IF|FINANCETOOLS|X_|IM_IS|LAST|^[A-Z].*IND$', i):
            df[i] = df[i].map(transform)
            df[i] = df[i].fillna(2)
            count += 1
    print("number of Y/N columns ：", count)

    try:
        df['Y1']=df['Y1'].map(transform)
    except:
        pass

    return df
```

針對內容為「Y / N」的二值特徵給予「1 / 0」的數值替換，因「零值」已有代表訊息，此處針對空值改以數值「2」做填補。

3) Nominal Features

```
def OHE(df):
    col = df.select_dtypes(include='object').columns
    col = col.append(pd.Index(["MARRIAGE_CD"])) # Notice !
    print('The remaining categorical columns：', len(col), "\n", col)

    c3 = {}
    for c in col:
        c3[c] = 'ohe_' + c
        df[c] = df[c].fillna("NaN")

    df = pd.get_dummies(df, columns=col, drop_first=True, prefix=c3)

    print('Shape：', df.shape)
    return df
```

針對「名目特徵」我們將空值也歸為一項類別進行 dummy 變數的轉換，特別注意的是「MARRIAGE_CD」在原始資料中屬於數值型態「0,1,2」，但其應為無法進行大小比較之類別訊息，因此在此處特別加入處理。

Numeric columns

1) Outliers

```
def outlier_check_IQR(check_col, col_checked, df):
    for i in check_col:
        col_value = df[i].dropna()
        Percentile = np.percentile(col_value.unique(), [0,25,50,75,100])
        IQR = Percentile[3] - Percentile[1] #四分位距
        UpLimit = Percentile[3] + IQR*1.5 # 約 +2.698 std
        DownLimit = Percentile[1] - IQR*1.5

        if len(col_value[col_value > UpLimit]) > 0 or len(col_value[col_value < DownLimit]) > 0:
            y_1 = sum(df.iloc[col_value[col_value > UpLimit]]["Y1"]) + \
                  sum(df.iloc[col_value[col_value < DownLimit]]["Y1"])
            print('col: %s, Over UpLimit: %d, Under DownLimit: %d, Y=1: %d' % \
                  (i, len(col_value[col_value > UpLimit]),len(col_value[col_value < DownLimit]),y_1))

            col_checked.append(i)

    print('\n')
    print('col_checked :' , len(col_checked))
    print('No outliers: ', set(check_col)-set(col_checked))
```

離群值的檢查以$[Q1 - 1.5 \times IQR\ , Q3 + 1.5 \times IQR]$ 定義為正常範圍界線，查看離群部分是否帶有特別的訊息(Y1 = 1)，若沒有就不做額外的處理(※考量到 test 資料集同樣存在離群值及欲使用之模型並不敏感於離群值)。依結果顯示，此處皆不作額外處理。

2) NaN

(1) KNN + mean（※未採用）

```
col_continuous, col_discrete = [], []
for i in check_col:
    #連續型數值(結尾AMT者、BMI、APC_1ST_YEARDIF、TERMINATION_RATE)
    if re.match(r'\w+AMT$|BMI|APC_1ST_YEARDIF|TERMINATION_RATE', i):
        col_continuous.append(i)

    #離散型數值
    else:
        col_discrete.append(i)

print('col_continuous:%d, col_discrete:%d, columns need to check:%d' % \
      (len(col_continuous), len(col_discrete), len(col_continuous)+ len(col_discrete)))
```

```
col_continuous:21, col_discrete:15, columns need to check:36
```

以「連續型數值」和「離散型數值」分別查看含空值之特徵欄位，並依據「Y1 = 1」的比例決定填補的方式。

連續型數值：

```
col: APC_1ST_YEARDIF, NaNs: 43282, Y=1: 387, P(NaNs|Y=1): 0.1935
col: ANNUAL_PREMIUM_AMT, NaNs: 62445, Y=1: 604, P(NaNs|Y=1): 0.3020
col: ANNUAL_INCOME_AMT, NaNs: 39201, Y=1: 407, P(NaNs|Y=1): 0.2035
col: BMI, NaNs: 16645, Y=1: 427, P(NaNs|Y=1): 0.2135
col: TERMINATION_RATE, NaNs: 43282, Y=1: 387, P(NaNs|Y=1): 0.1935
col: DIBBENEFIT_AMT, NaNs: 27540, Y=1: 333, P(NaNs|Y=1): 0.1665
col: DIBACCIDENT_AMT, NaNs: 27540, Y=1: 333, P(NaNs|Y=1): 0.1665
col: POLICY_VALUE_AMT, NaNs: 27540, Y=1: 333, P(NaNs|Y=1): 0.1665
col: ANNUITY_AMT, NaNs: 27540, Y=1: 333, P(NaNs|Y=1): 0.1665
col: EXPIRATION_AMT, NaNs: 27540, Y=1: 333, P(NaNs|Y=1): 0.1665
col: ACCIDENT_HOSPITAL_REC_AMT, NaNs: 27540, Y=1: 333, P(NaNs|Y=1): 0.1665
col: DISEASES_HOSPITAL_REC_AMT, NaNs: 27540, Y=1: 333, P(NaNs|Y=1): 0.1665
col: OUTPATIENT_SURGERY_AMT, NaNs: 27540, Y=1: 333, P(NaNs|Y=1): 0.1665
col: INPATIENT_SURGERY_AMT, NaNs: 27540, Y=1: 333, P(NaNs|Y=1): 0.1665
col: PAY_LIMIT_MED_MISC_AMT, NaNs: 27540, Y=1: 333, P(NaNs|Y=1): 0.1665
col: FIRST_CANCER_AMT, NaNs: 27540, Y=1: 333, P(NaNs|Y=1): 0.1665
col: ILL_ACCELERATION_AMT, NaNs: 27540, Y=1: 333, P(NaNs|Y=1): 0.1665
col: ILL_ADDITIONAL_AMT, NaNs: 27540, Y=1: 333, P(NaNs|Y=1): 0.1665
col: LONG_TERM_CARE_AMT, NaNs: 27540, Y=1: 333, P(NaNs|Y=1): 0.1665
col: MONTHLY_CARE_AMT, NaNs: 27540, Y=1: 333, P(NaNs|Y=1): 0.1665
```

KNN Regression

Mean

離散型數值：

```
col: OCCUPATION_CLASS_CD, NaNs: 3960, Y=1: 286, P(NaNs|Y=1): 0.1430
col: LEVEL, NaNs: 43305, Y=1: 390, P(NaNs|Y=1): 0.1950
col: RFM_M_LEVEL, NaNs: 43282, Y=1: 387, P(NaNs|Y=1): 0.1935
col: L1YR_C_CNT, NaNs: 87936, Y=1: 1436, P(NaNs|Y=1): 0.7180
col: INSD_LAST_YEARDIF_CNT, NaNs: 171, Y=1: 64, P(NaNs|Y=1): 0.0320
```

KNN Classifier

Mean

(2) mean

```
def Fill_NaN(df, method):
    imr = Imputer(missing_values='NaN', strategy=method, axis=0).fit(df.values)
    imputed_data = imr.transform(df.values)

    #turn numpy.ndarray back to dataframe
    col={}
    for j,c in enumerate(df.columns):
        col[c] = imputed_data[:, j]

    df = pd.DataFrame(col)
    return df
```

嘗試了多種填補方式，包括：「平均值」、「中位數」、「眾數」以及「KNN 預測」等，依據而後套用至模型的表現，最終採用單一「平均值」填補法。

- Data Processing（※未採用）

針對部分欄位進行運算得出新的特徵欄位，保留與目標變數「Y1」之相關性大於舊有欄位者：

1) L1YR_C_CNT_over15

```
Y1=1:  [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 29.0]
Y1=0:  [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0, 21.0, 22.0, 23.0, 24.0, 25.0, 27.0, 29.0, 30.0, 31.0, 35.0, 37.0, 41.0]
test:  [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0, 20.0, 21.0, 22.0, 23.0, 24.0, 25.0, 29.0, 47.0]
```

```
# check 'L1YR_C_CNT' -create new column(L1YR_C_CNT <= 15 = 1 else = 0)
new_col = 'L1YR_C_CNT_over15'
old_col = 'L1YR_C_CNT'
data_train_knn[new_col] = 0
index = data_train_knn[old_col][data_train_knn[old_col] <= 15].index.tolist()
data_train_knn[new_col][index] = 1

data_test_knn[new_col]=0
index = data_test_knn[old_col][data_test_knn[old_col] <= 15].index.tolist()
data_test_knn[new_col][index] = 1
```

在離散型數值的空值檢查中，發現 L1YR_C_CNT 的空值欄位下有著高比例的 Y1=1 值，我們再針對「非空值」的部分查看其變數，可以發現 Y1=1 者大部分集中在變數≤ 15的區間，因此新增欄位「L1YR_C_CNT_over15」。

2) ISSUE_IND_SUM

```
col_ind = [i for i in train.columns if re.match(r'IF_ISSUE_._IND', i)]
train['ISSUE_IND_SUM'] = train[col_ind].sum(axis=1)
test['ISSUE_IND_SUM'] = test[col_ind].sum(axis=1)
```

目前「壽險保單」的持有有效「主約」件數總和。

3) ISSUE_ADD_SUM

```
col_ind = [i for i in train.columns if re.match(r'IF_ADD_._IND', i)]
train['ISSUE_ADD_SUM'] = train[col_ind].sum(axis=1)
test['ISSUE_ADD_SUM'] = test[col_ind].sum(axis=1)
```

目前「壽險保單」的持有有效「附約」件數總和。

4) ANNUAL_INCOME_AMT

```
a = train['ANNUAL_INCOME_AMT'].rank() - train['ANNUAL_PREMIUM_AMT'].rank()
train['ANNUAL_INCOME_minus_PREMIUM'] = a.apply(lambda x: 1 if x > 0 else 0)

b = test['ANNUAL_INCOME_AMT'].rank() - test['ANNUAL_PREMIUM_AMT'].rank()
test['ANNUAL_INCOME_minus_PREMIUM'] = b.apply(lambda x: 1 if x > 0 else 0)
```

「年收入」是否大於「年繳化保費」(數值經 rank 排名轉換)。

5) IM_IS_SUM_IND

```
col_ind = [i for i in train.columns if re.match(r'IM_IS_._IND', i)]
train['IM_IS_SUM_IND'] = train[col_ind].sum(axis=1)
test['IM_IS_SUM_IND'] = test[col_ind].sum(axis=1)
```

是否持有「特定商品 A-D」加總。

- Model

Regression

1) 相關係數分析

計算每個變數之間的相關係數，刪除相關性大於 0.95 的特徵。

```
data_final.corr()
```

2) 特徵重要性分析

(1) 用 Random Forest 去分析，將資料分成 train, validation, 以及 testing，分析後，16, 32, 64, 128, 256, 512, 1024, 2048 顆分類樹中，以 1024 顆分類樹的表現最佳。

```
X, y = data_train.drop(['CUS_ID', 'Y1'], axis = 1), data_train['Y1']
feat_labels = X.columns
print("Shape of feat_labels: ", feat_labels.shape)

forest = RandomForestClassifier(n_estimators=1024, random_state=1, class_weight='balanced')

forest.fit(X, y)

importances = forest.feature_importances_
indices = np.argsort(importances)[::-1]
```

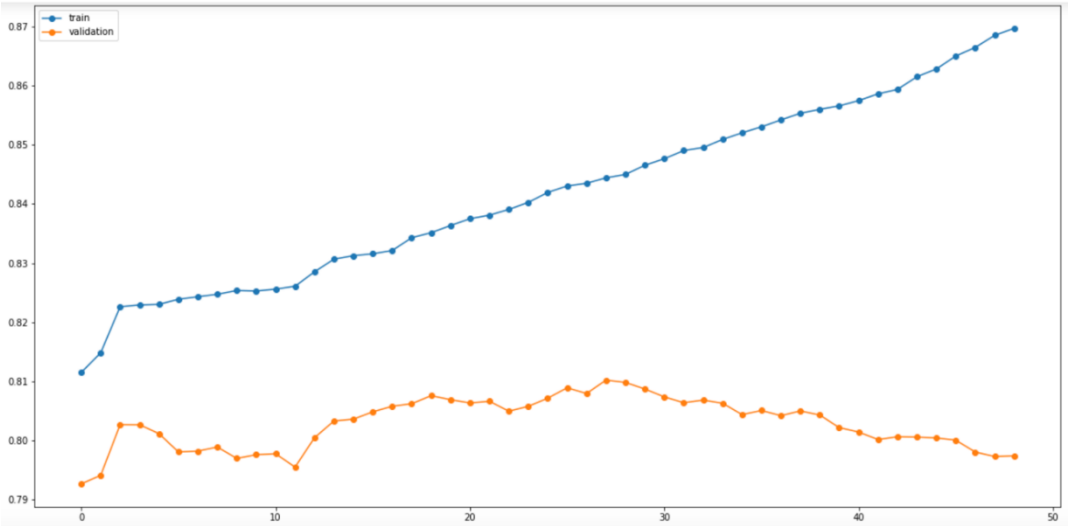
(2)分析後將特徵排序：

```
1)  INSD_LAST_YEARDIF_CNT          0.044806
2)  BMI                            0.036628
3)  AGE                            0.035691
4)  DIEACCIDENT_AMT                0.035334
5)  L1YR_GROSS_PRE_AMT             0.032938
6)  OCCUPATION_CLASS_CD            0.032780
7)  ANNUAL_INCOME_AMT              0.028881
8)  TOOL_VISIT_1YEAR_CNT           0.027695
9)  CHANNEL_A_POL_CNT              0.027062
```

3) Polynomial

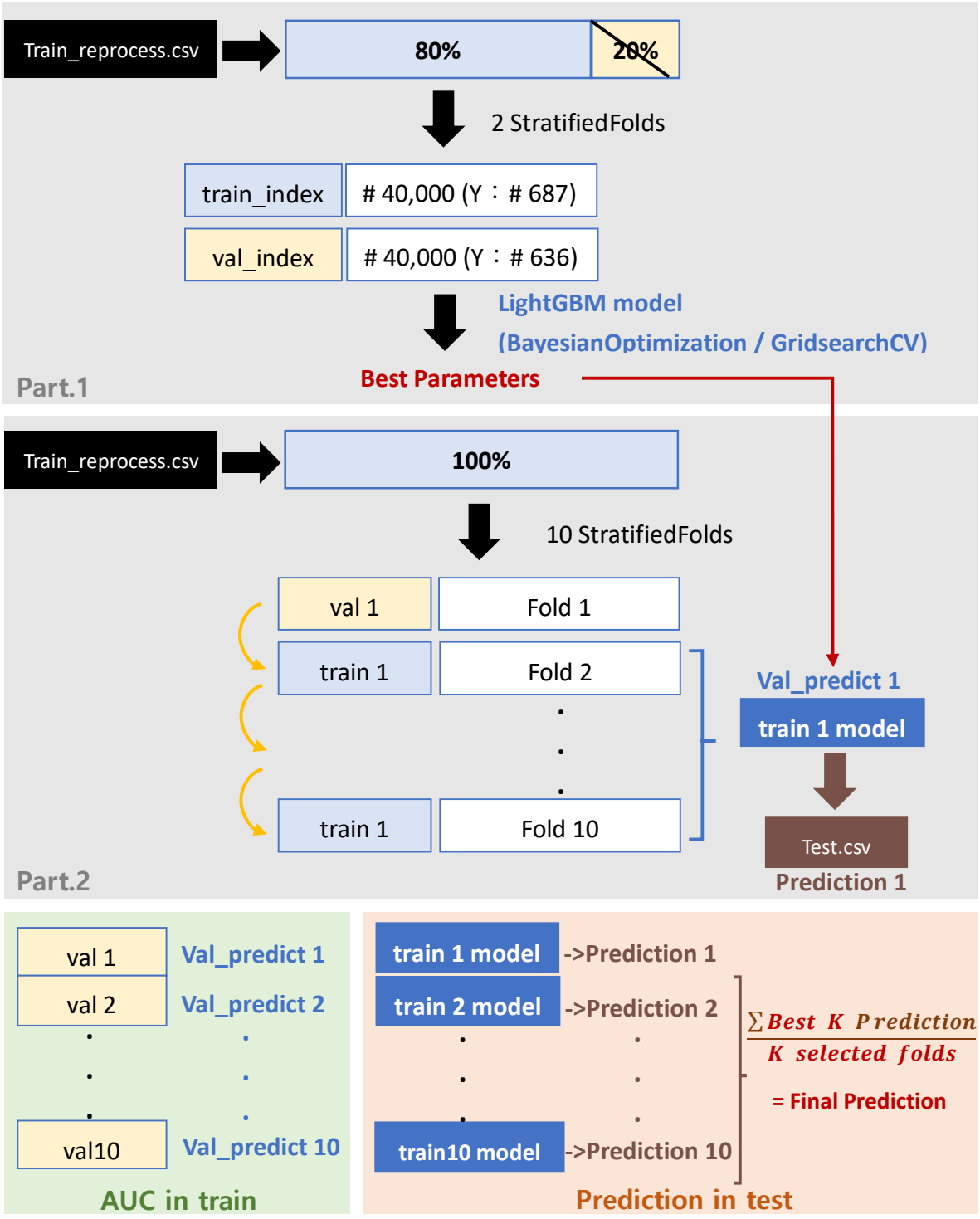
將重要的特徵群做 2 次 Polynomial，並在實驗下，發現將最重要的 28 個特徵做 2 次 Polynomial 加上剩下的特徵去訓練出來的分數最高。

```
def poly(num, dt):return PolynomialFeatures(num, interaction_only=False).fit_transform(dt)
```



Best Score in Regression Model：0.8340055635

LightGBM



整體訓練過程如上圖所示，一共分為兩部分：

1) 尋找最佳超參數

此部分我們隨機抽取 80%的處理過後之訓練用資料，再將其進行 2 折的交叉切分(使用分層採樣，確保訓練集與驗證集中各類樣本的比例與原始數據集中相同)，而後分別使用「Bayesian Optimization」及「GridSearchCV」兩種方式來對 LightGBM 模型進行最優超參數組合的搜索。LightGBM 模組擁有眾多可調動參數，我們只根據欲分析的資料型態，選取重要參數(如下圖所示)進行調整與試驗，其餘則保持官方預設值。

```
parameters = {'boosting_type': 'gbdt',              #訓練方式(梯度提升決策樹)
              'objective': 'binary',                #目標函數(二分類任務)
              'importance_type': 'split',
              'is_unbalance': True,                 #非平衡數據
              'metric': 'auc',                      #評價指標(損失函數)
              'verbose': 0,
              'n_estimators': 200,                  #樹的數量
              'n_jobs': -1,
              'random_state': 1,
              'learning_rate':learning_rate,        #學習率
              'max_depth':  max_depth,              #樹的深度(if -1 means no limit)
              'num_leaves': num_leaves,             #葉子節點數，調節樹的複雜度 (< 2^(max_depth))
              'lambda_l1': lambda_l1,               #L1正則化項
              'lambda_l2': lambda_l2,               #L2正則化項
              'feature_fraction': feature_fraction, #特徵採樣比例
              'bagging_fraction': bagging_fraction, #數據採樣比例
              'bagging_freq': bagging_freq,         #每K輪迭代執行一次bagging
              'cat_smooth': 1}
```

首先使用「Bayesian Optimization」進行大範圍的超參數挑選(因其透過構建目標函數的機率模型，每一次的超參數選擇會基於上一次的評估，相比 Grid Search 的遍歷式搜索更加快速、有效)，而後使用「GridSearchCV」於小範圍的嘗試，得到優化器計算出之最佳參數後，再經由人為的小幅度調整得到最終參數組合：

```
best_parameters ={'boosting_type': 'gbdt', 'importance_type': 'split', 'learning_rate': 0.1,
                  'max_depth': 20, 'n_estimators': 200, 'n_jobs': -1,'num_leaves': 37,
                  'objective': 'binary', 'random_state': 1, 'silent': True, 'metric': 'auc',
                  'verbose': 0, 'feature_fraction': 0.6, 'bagging_fraction': 0.7, 'bagging_freq': 8,
                  'lambda_l1': 0.1, 'lambda_l2': 35, 'cat_smooth': 1}
```

2) 模型訓練與預測

最後針對全部的訓練集資料進行 10 折的交叉切分，每次取 1 折做為驗證集，9 折作為訓練集進行訓練，訓練完畢之模型將：

(1) 預測該輪的驗證集資料(Val_predict)以評估模型效能

(2) 預測測試資料集(Prediction)

一共輪替 10 次，獲得：

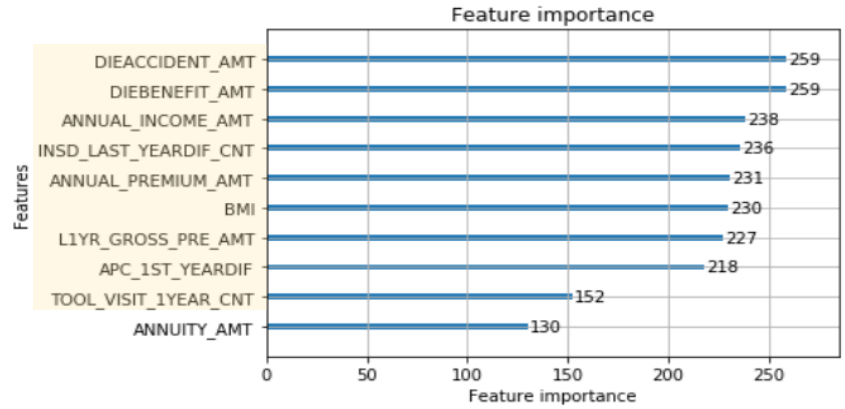
(1) 10 組的 Val_predict 將合成針對完整訓練資料集的預測，經由與真實訓練集資料的比對計算模型成效。

```
AUC in train data: 0.8422 | It costs 109.814151 sec
```

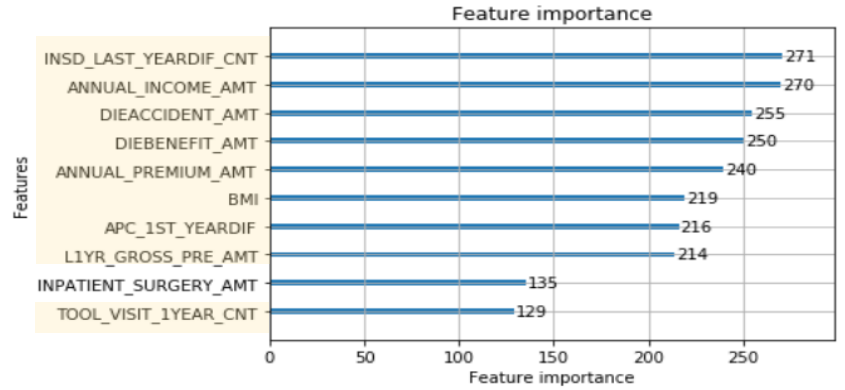
(2) 10 組的 Prediction 則代表了 10 個模型針對測試資料集的預測，我們挑選其中在訓練集上表現最佳的 4 個模型，將其預測值取平均作為最終提交預測值。

```
predictions_transform = sum(predictions[:,[1,5,6,7]].T) /4
```

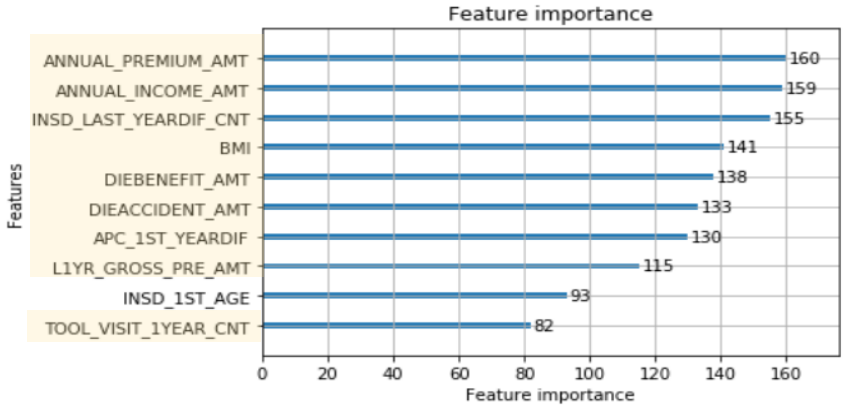
3) 重要特徵



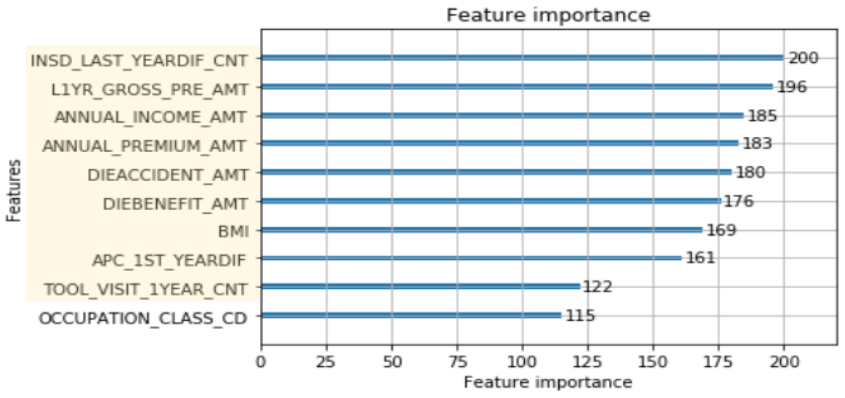
Fold2



Fold6



Fold7



Fold8

Best Score in LightGBM Model : 0.8506553367