Lab Assignment #2

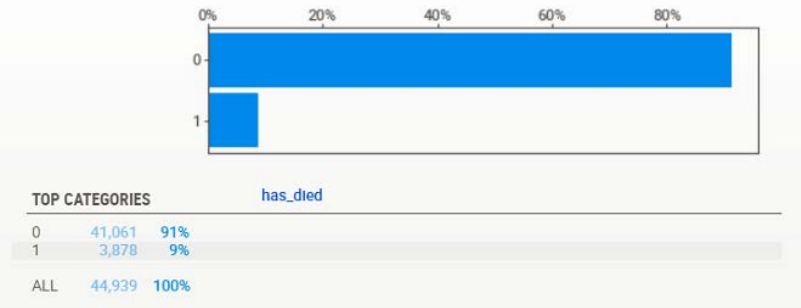
311700014 管科碩二 苗詠哲

I. Data Preprocessing

1. 探索性資料分析 (EDA)

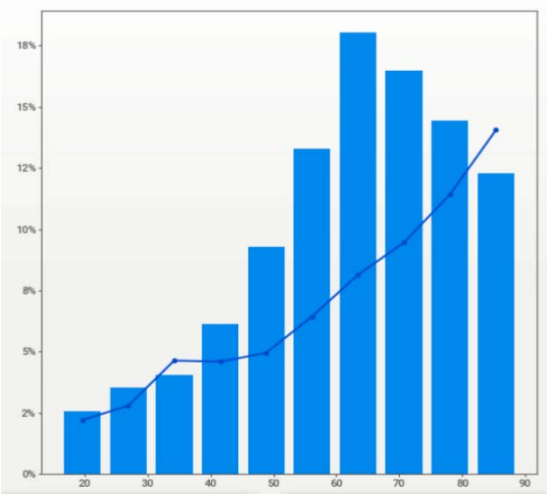
- **目標變數之觀察: 不平衡資料**

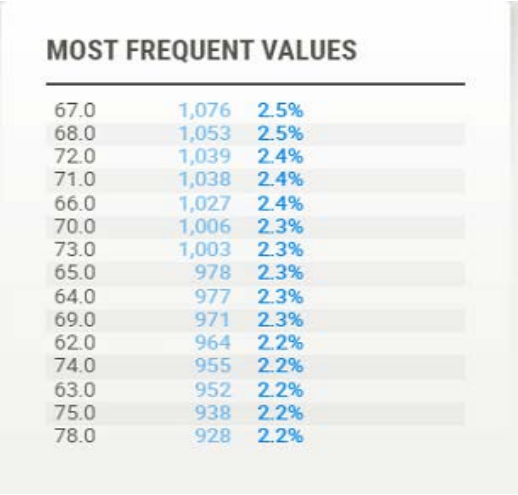
首先通過觀察可以發現，本資料集屬於不平衡資料集，在 'has_died' 欄位中僅 3878 位病患為 1 (代表死亡)，僅佔總資料集的 9%，而非死亡者則 91%，通過此結果可以預期後續會做 Oversampling 來試圖解決資料不平衡問題。



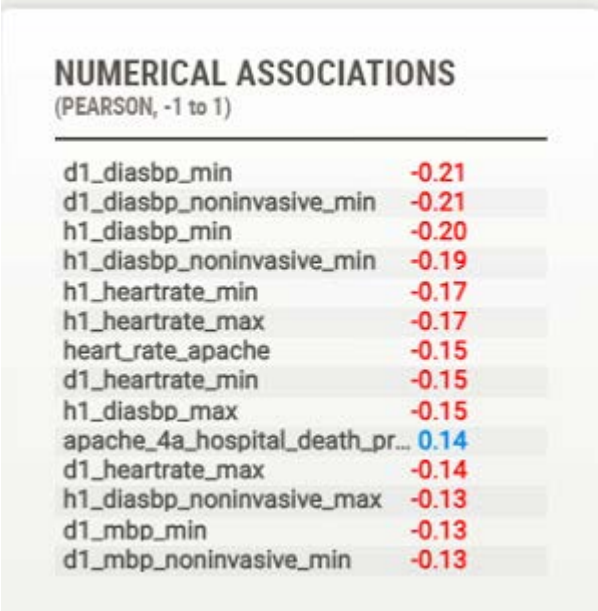
- **基本變數觀察: 死者中多數為高齡病患，同時年齡也可能影響生理表徵**

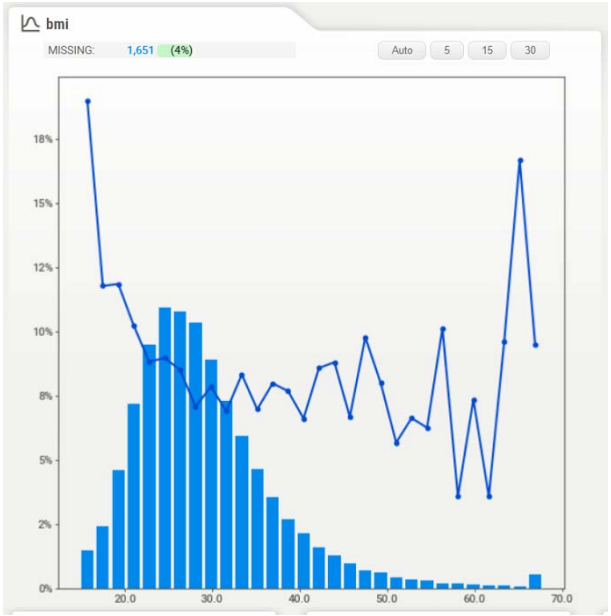
通過以下的報表可以發現，死亡的患者多半為 60~80 歲居多，並且以 67 這個年齡為大宗，在 1076 位 67 歲的高齡患者中就有 2.5% 為死亡





此外，通過下方左邊的年齡與其他數值變數相關係數表，也可以發現年齡與血氧濃度、心率，以及血壓等有著負向的關係，高齡患者通常伴隨著較差的心率、較低的血氧，以及較低的血壓。





除了年齡這項基本的人口變數外，出於個人好奇也針對 BMI 做了類似的調查(上右)，發現過瘦以及過胖者死亡病患所佔的比率較高。

- **基本變數觀察: 通過逐一排查對欄位類型進行分類**

欄位類型	欄位名稱
數值型	['age', 'bmi', 'height', 'weight', '_days', '_diagnosis', '_min', '_max', '_death_prob', 'heart_rate_apache', 'map_apache', 'resprate_apache', 'temp_apache']
類別型	['ethnicity', 'gender', 'icu_admit_source', '_type', '_bodysystem', 'hepatic_failure', 'immunosuppression', 'leukemia', 'lymphoma', 'solid_tumor_with_metastasis', '_diagnosis', 'gcs_eyes_apache', 'gcs_motor_apache', 'gcs_verbal_apache', 'arf_apache', 'gcs_unable_apache', 'intubated_apache', 'ventilated_apache']
待刪除	['encounter_id', 'patient_id']

欄位名稱前綴底線符號者代表所有以該名稱結尾之欄位，而通過 EDA 的探索後大致總結了需要解決的問題，包含：缺失資料處理、類別欄位資料 Encoding 方法，以及資料不平衡問題。

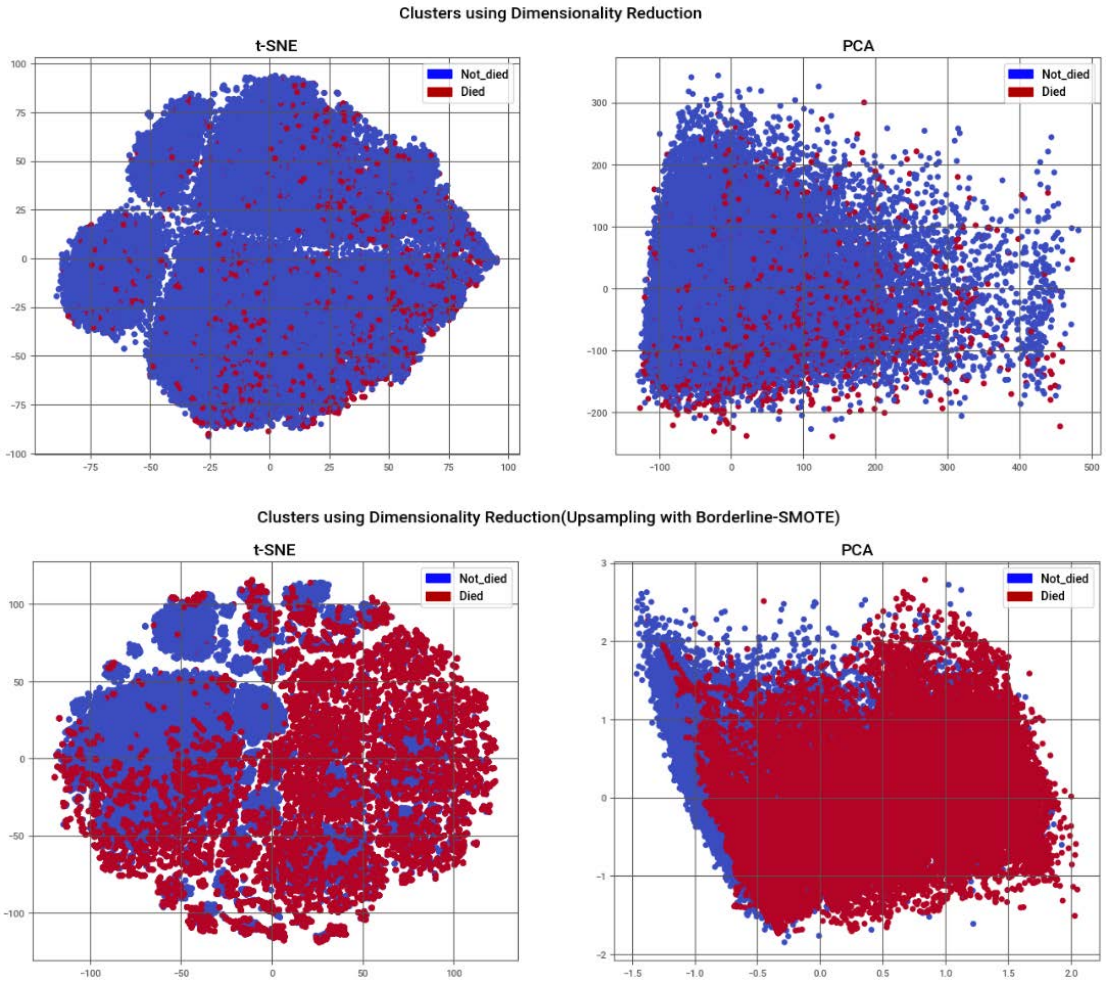
2. 資料清理及不平衡處理

第一，針對**缺失資料處理**，由於有許多病患在多個不同欄位有不同的缺失資料組合，為求統一，**數值資料一律利用平均數進行填值**；**類別資料則將缺失值視為未知**，單獨作為一類處理。

第二，針對**類別變數 Encoding**，考慮到本資料集擁有較多的類別變數，並且類別變數多具有不少的變數內容，為了避免維度爆炸造成訓練資料過於稀疏的問題，不考慮機器學習中常見的 One Hot Encoding 的技術，而**採用 Target Encoding 中 Leave One Out 的方法**，將目標變數作為 Target，將類別變數如：性別，同為男性者是否死亡做平均，以該資料集來說即死亡率，再回填至該筆資料的性別 Feature 中。特別要注意的是，在上數的計算中，會將自身對應到的 Target 做 Leave out 的動作不列入計算，除了可以增加變數的多樣性外，也預期可以讓模型不會對編碼後的結果過於敏感。

第三，**Normalization**，為了確保每個變數 Scale 不要差異過大，造成 Oversampling 以及模型的計算上有偏誤，因此對資料做 **min max normalization**。

第四，**不平衡資料處理**，採用常聽聞的 **Borderline SMOTE**，與傳統 SMOTE 不同的是 Borderline SMOTE 更注重於邊界的少數類樣本，期望通過謹慎的取樣位分類模型帶來更加的效能。下方為經過 t-SNE 以及 PCA 降至二維後的生成樣本實驗圖，雖然在二維上看似重疊不少，但期待在原始資料的高維空間中會有更佳的分類邊界出現。



II. Classification Method

1. 分類模型：LightGBM (Reference、Package Site)

LightGBM 是由微軟所開發的模型，屬於梯度提升樹類型的模型，同類型有名的模型包含 Xgboost 等，LightGBM 有以下幾點特色：

第一，採用 leaf-wise 的生長策略，與傳統以深度為優先的 level-wise 生長不同，此策略會優先擴展具有更多信息增益的節點，而不會逐層掃描後同時運算每 level 的節點，造成無效率的問題。

第二，離散化學習：在 LightGBM 中，會將連續的特徵資料做離散的動作，直觀上可以理解為繪製質方圖時，會將連續資料做分桶 (bins)，這樣的分桶計算有助於減少模型計算成本，使得其在處理大型資料集時能更具有優運算上的優勢。

第三，Gradient-based One-Side Sampling, GOSS: 單邊梯度取樣使得 LightGBM 在取樣時會優先取樣梯度值前幾名的樣本，因為梯度較大的樣本在分裂節點時往往可以帶來更大的信息貢獻。

2. Reproduce

在提交的檔案中，可以在 code 資料夾內找到 for_reproduce.ipynb 的檔案，僅需依照該筆記本內的三個區塊逐一執行，即可在相同的目錄下得到 reproduce.csv 的預測檔案。需要注意的是，必須確保已 fine tune 完的模型檔案 model.pkl、老師提供之 train_X.csv、train_y.csv，以及 test_X.csv 皆與該本筆記本放置於同一目錄內 (此外也在 code 資料夾內提供完整筆記本 main.ipynb，包含所有從資料探索、分析、清理、模型訓練、調整超參數，以及最終重新訓練模型的過程，可供查閱)。

III. Results & Analysis

1. Cross-validation method 以及每層分數

為了在提交預測檔案至 Kaggle 前有能力可以準確地評估模型，我在所有流程開始之前有先切出一份 private test dataset 作為私人的測試資料集，並且採用了 5-fold 的 CV，大體上來說，整個資料集會被切分成 train、validation、test(private)，以及最終要預測上傳的 test(public)。

為了確保本報告提到的第一部分內，諸多的資料處理流程僅操作在每層 fold 中的訓練資料集上，我在 main.ipynb 中的 score_model() function 內，手動實作了 Cross-validation 如下圖：

```python
# ================================================================================
# 手做 K-fold Cross Validation

AUROC_scores = []
f1_macro_scores = []

for train_fold_index, val_fold_index in cv.split(X_train, y_train):
    X_train_fold, y_train_fold = X_train.iloc[train_fold_index], y_train.iloc[train_fold_index]
    X_val_fold,   y_val_fold   = X_train.iloc[val_fold_index],   y_train.iloc[val_fold_index]

    #資料前處理: Imputation
    X_train_fold[num_cols] = X_train_fold[num_cols].fillna(X_train_fold[num_cols].mean())
    X_train_fold[cat_cols] = X_train_fold[cat_cols].fillna('unknow')

    X_val_fold[num_cols] = X_val_fold[num_cols].fillna(X_train_fold[num_cols].mean()) # 注意裡面是用 training data 的統計資料
    X_val_fold[cat_cols] = X_val_fold[cat_cols].fillna('unknow')


    # 資料前處理: Target encoding
    target_encoder.fit(X_train_fold[cat_cols], y_train_fold)
    X_train_fold[cat_cols] = target_encoder.transform(X_train_fold[cat_cols])
    X_val_fold[cat_cols]   = target_encoder.transform(X_val_fold[cat_cols])

    # 資料前處理: Normalization
    X_train_fold = scaler.fit_transform(X_train_fold)
    X_val_fold = scaler.transform(X_val_fold)

    # 資料前處理: Oversampling
    X_train_fold_oversample, y_train_fold_oversample = bordsmoter.fit_resample(X_train_fold, y_train_fold)

    # 建立 model 實例
    if params is None:
        model_obj = model
    else:
        model_obj = model.set_params(**params)

    # Fit model
    model_obj.fit(X_train_fold_oversample, y_train_fold_oversample)

    # Predict and Evaluation
    y_val_pred = model_obj.predict(X_val_fold)
    y_val_proba = model_obj.predict_proba(X_val_fold)[:, 1]
    AUROC_scores.append(roc_auc_score(y_val_fold, y_val_proba))
    f1_macro_scores.append(f1_score(y_val_fold, y_val_pred, average='macro'))
# ================================================================================
```

此方程式接收一個由 StratifiedKFold(n_splits=5, random_state=5201314, shuffle=True) 建立的 cv 物件，並通過呼叫 cv.split() 方法實現訓練資料集和驗證資料集的切分，通過迭代對每次不同的訓練集資料做包含：缺失值處理、類別資料 Encoding、標準化、過採樣，以及最後的模型訓練。並將每一折的 AUROC、F1_macro 保存後返回。

值得注意的是在缺失值處理時，針對 validation data 的各數值欄位是運用 train data 個數值欄位所計算出的平均值進行填值的；以及在 target encoding、normalization 時也都是通過訓練資料的值去進行計算，在讓 validation data 去做變換，防止資訊洩漏的問題。

訓練過程分數如下：

```
   AUROC in each fold: [0.86371493 0.87165734 0.86825698 0.88402637 0.86726338], average:  0.87098
F1_macro in each fold: [0.69468358 0.70459722 0.7224211  0.7098041  0.70169701], average:  0.70664
```

私有測試集分數如下：

```
F1_macro: 0.7303
   AUROC: 0.8885
Accuracy: 0.9243
Precision: 0.5816
  Recall: 0.4407
```

可以看到在五折平均的 F1 macro 分數為 0.707；而私有測試集的 F1 macro 分數為 0.73。

2. 超參數優化

為了確提高模型表現，以及穩定度，運用 Optuna 這個超參數優化的套件進行超參數優化，具體優化的參數明確地定義於 object() function 中，包含最大枝葉數、學習率、最大深度、L1 正則項強度、L2 正則項強度等等。

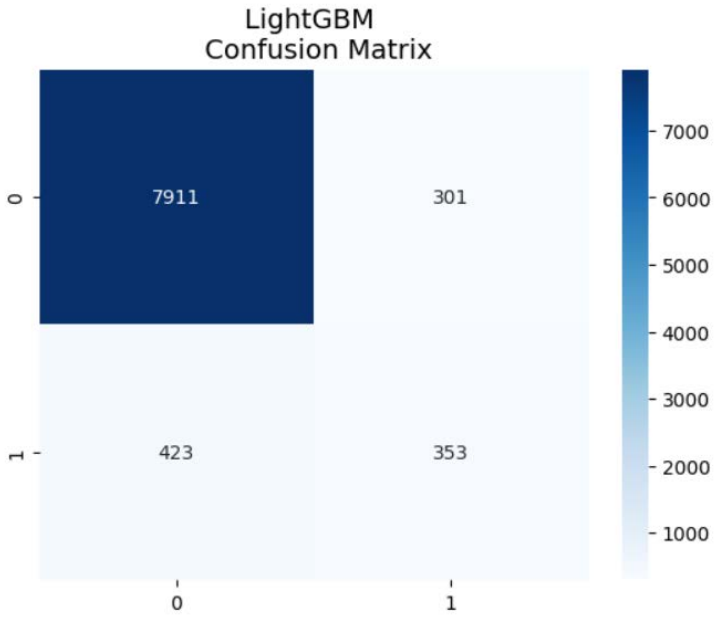
```python
def objective(trial):
    params = {
        'objective': 'binary',
        'metric': 'binary_logloss',
        'boosting_type': 'gbdt',
        'num_leaves': trial.suggest_int('num_leaves', 2, 256),
        'learning_rate': trial.suggest_loguniform('learning_rate', 0.005, 0.1),
        'feature_fraction': trial.suggest_uniform('feature_fraction', 0.1, 1.0),
        'bagging_fraction': trial.suggest_uniform('bagging_fraction', 0.1, 1.0),
        'lambda_l1': trial.suggest_loguniform('lambda_l1', 1e-8, 10.0),
        'lambda_l2': trial.suggest_loguniform('lambda_l2', 1e-8, 10.0),
        'min_child_samples': trial.suggest_int('min_child_samples', 5, 100),
        'min_child_samples': trial.suggest_int('min_child_samples', 5, 100),
        'max_depth': trial.suggest_int('max_depth', 3, 15),
        'min_child_weight': trial.suggest_loguniform('min_child_weight', 1e-8, 1.0),
        'is_unbalance':False,
        'n_jobs':-1,
        'use_missing':False,
        'verbosity':-1
    }

    # 使用 5-fold 平均 f1_macro 調整超參數
    _, f1_macro_score = score_model(LGB, params=params, X=X_train, y=y_train, cv=sss)
    metric = f1_macro_score.mean()

    return metric
```

經過定義後，需要呼叫 Optuna 內的 study 物件的 optimize 方法，並將 object() function 傳入，經過 600 回合的超參數搜尋後在 5 折平均的 F1 macro 得到 0.713 的分數，以及私有測試集 0.725 的成績：

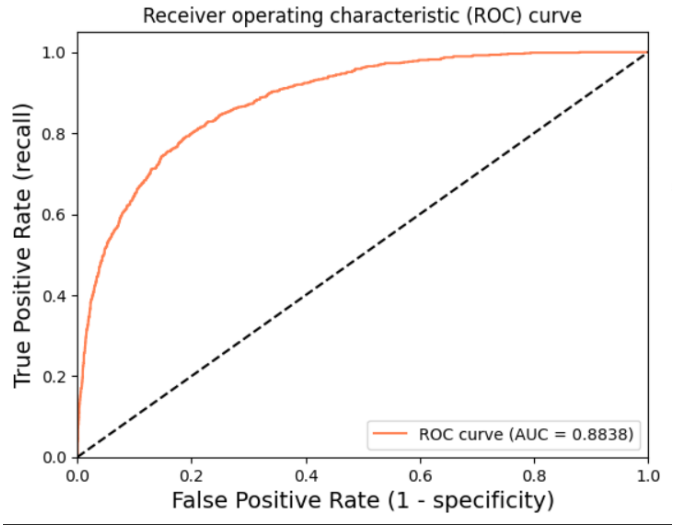
```
F1_macro: 0.725
   AUROC: 0.8838
Accuracy: 0.9194
Precision: 0.5398
  Recall: 0.4549
```

雖然在私有測試集上的分數下降了，但縮小了與 5-fold 平均分數的差異，可以期待模型在此超參數下應該有更穩健的表現 (Kaggle 上的 Public 分數為 0.712)。

除了上圖的四種分數外，從以下的混淆矩陣中可以看出模型在預測能力上的總體表現，在預測錯誤的案例中，相較於偽陽性，模型更傾向於將死亡病例預測為存活。(算是比較樂觀的模型？)

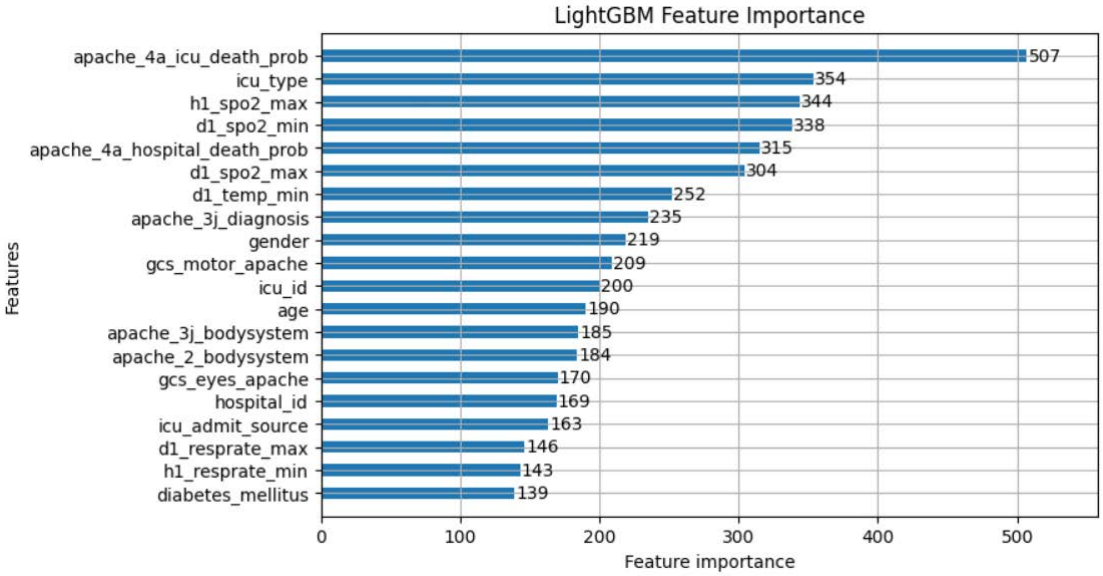


也提供完整 ROC 曲線如下：

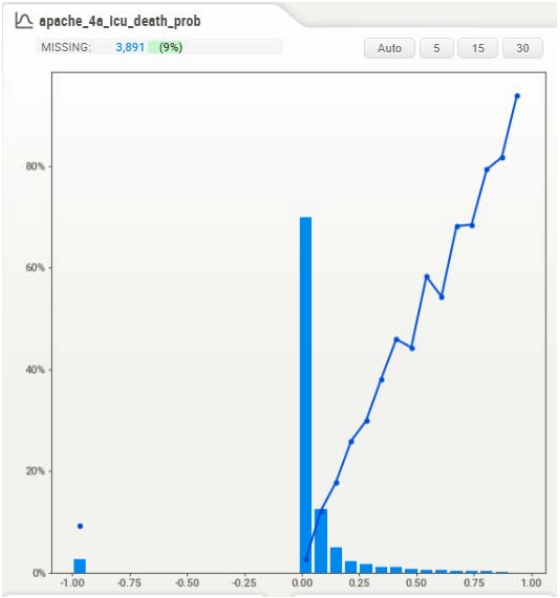


3. 模型解釋

為了理解模型如何做出分類及預測的決定，以下繪製出模型認為在分類上重要的 20 大變數：



可以看出最重要的變數前三名分別為加護病房預測死亡機率、加護病房類型，以及加護病房的照類型，可謂相當符合常理。通常病患在緊急且病重時才會住進加護病房，因此該地可以理解為多數死者生前的最終居所以及照護單位，因此該單位預測隻死亡機率，以及該單位對病患的照護類型、病房類型等，應該都會隱含對於預測是否死亡的重要資訊。



右表為加護病房預測死亡機率以及病患是否為死亡病例的關係圖，直方圖為機率，折線圖為該機率下真實死亡的比率，可以看出非常明顯的關係。

此外 h1_spo2_max、d1_spo2_min、d1_spo2_max、d1_temp_min、d1_resprate_max、h1_resprate_min 等這些特徵皆涉及患者在加護病房入院的第一小時和第一天內的生理數據，包括血氧狀態、體溫和呼吸率等。反映了患者的初期生理狀態，對預測患者是否死亡也起到相當重要的作用。