Data Mining Final Competition Project

競賽名稱：Facebook: Human or Robot?

組別：Group 29

組員：

- 練智剛 F74102030
- 常定 利 P76117030
- 林冠宏 N26122107

競賽內容：

- 由於網站上的人類競標者因無法贏得「機器人」的拍賣而變得越來越沮喪。結果導致該網站核心客戶群的使用量直線下降。
- 目標是識別出「機器人」提出的線上拍賣出價，幫助網站所有者輕鬆標記這些用戶，將其從網站中刪除，以防止不公平的拍賣活動。
- 提交的文件會依據 ROC 曲線下的面積進行評分。

比賽資料集：

- Bidder dataset :
    - bidder_id : bidder 的唯一 ID
    - payment_account : bidder 的付款帳戶
    - address : bidder 的郵件地址
    - outcome : bidder 的 label，用來表示他是否為機器人，0 代表人類，1 則代表機器人
- Bid dataset : 包括不同拍賣總共 760 萬次的出價
    - bid_id : bid 的唯一 ID
    - bidder_id – bidder 的唯一 ID (與 bidder dataset 相同)
    - auction – 拍賣的唯一 ID
    - merchandise – 拍賣網站廣告系列的類別，這指的是 bidder 透過搜尋 某個商品而進入此網站
    - device – 手機型號
    - time - 出價時間
    - country - bidder IP 所屬的國家
    - ip – bidder 的 IP
    - url - 紀錄 bidder 從哪個網址跳轉到當前的競拍頁面

特徵工程:

發想

機器人常有的特徵: 反應速度快、有一定的出價模式、對於某項商品很堅持、某些數值異常於(人類)平均

1. 統計每個bidder_id所擁有的各項資料，有幾個獨特的數值

如果是機器人的話可能特定幾項的值會異常的高於平均數值

2. bidder在每次bid之間的時間間隔

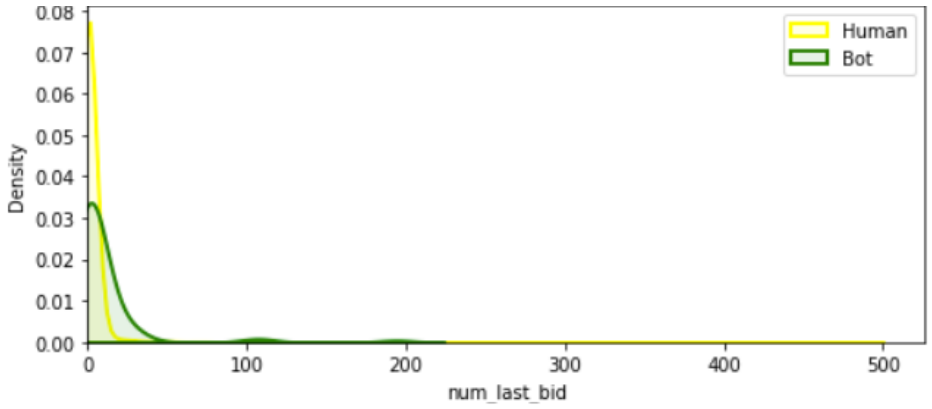
先將資料依照時間順序排列，去計算bidder每兩次出價之間的時間差，如果是人類的話，在兩次出價之間會有一定的時間差異，因為人做出反應、輸入價錢都需要時間。但如果是機器人的話，兩次的判斷時間可以近乎於0

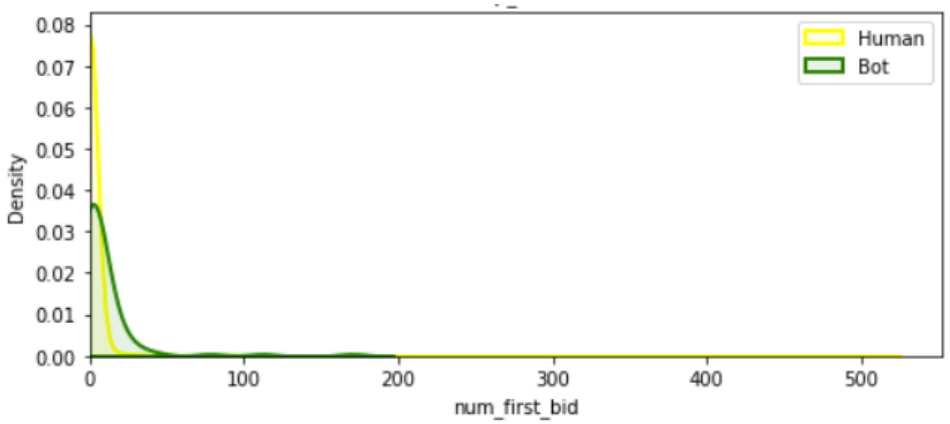
3. 平均值、 中位數、最小值、 最大值

計算一些在統計上常用到的值，有助於用在後續的判斷。

4. 一個bidder_id有幾次的最先出價和最後出價

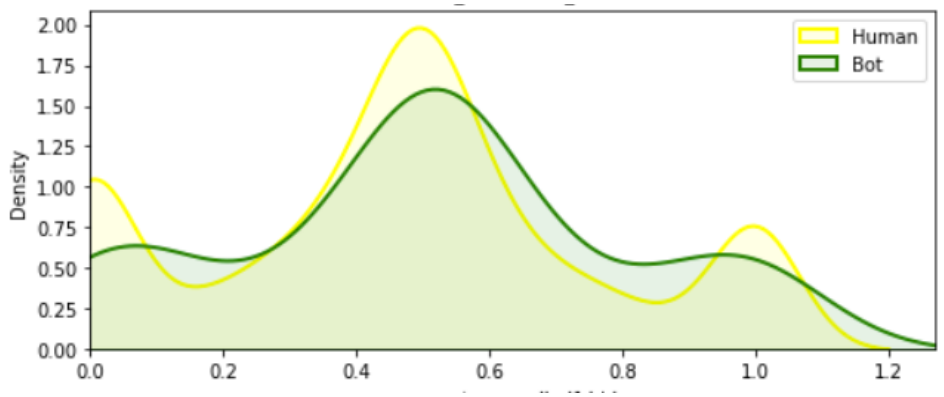
如果是機器人的話，可能商品一上架就會馬上出價，速度快於平常人的反應。如過在機器人的設定上是一定要贏得競標，則最後出價的次數也會高於一般數值。





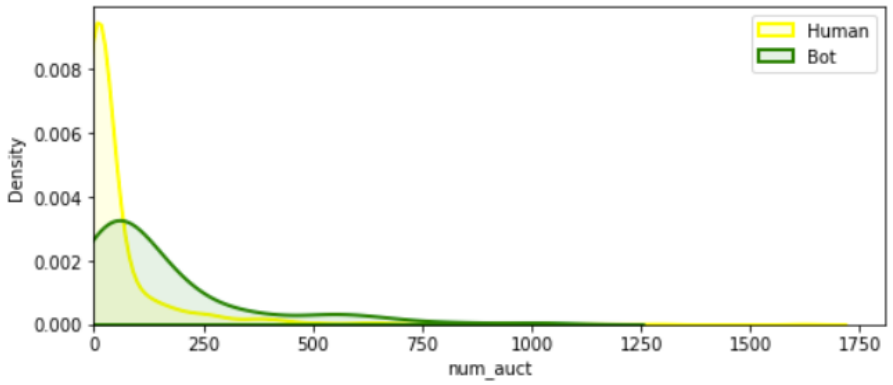
5. 在最後一段時間出價的出價次數

不同於第四點的是，如果機器人的設定是要以最低價格得標，那在競標的前期，機器人可能不會出價，因為出價的越頻繁價格可能被提升得越多。所以若是希望以最低價格得標，會在接近截止的時候才開始出價，有比較大的機會以最低的價格得到商品。



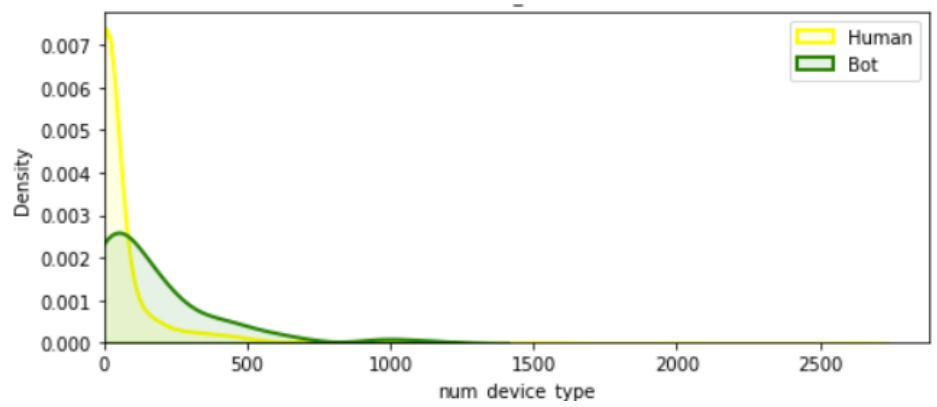
6. 一個bidder_id的出價次數最大值

因為機器人的設定可能是只要有人一出比較高的價格，就馬上出一個更高一點的價格，那這樣出價次數就會很高



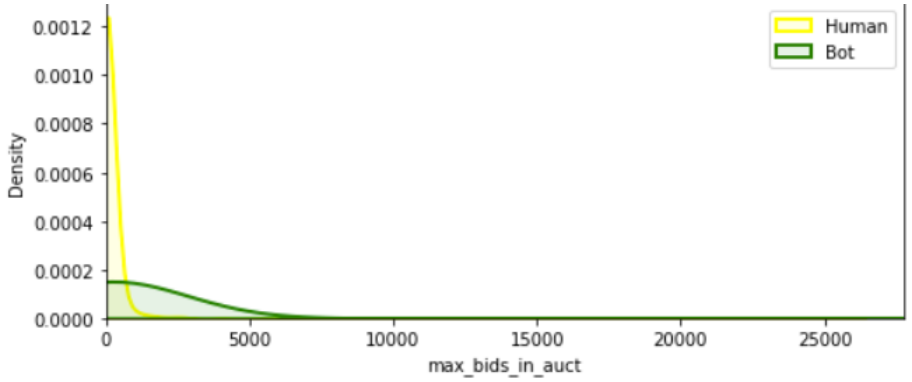
7. 在不同的裝置上的出價次數

如果是一個長時間的競標項目，人可能會傾向於用不同的裝置出價，例如在家可能用電腦、出門在外用手機、在公司用公司電腦，然而機器人可能都在同一台電腦上執行程式，所以切換裝置的數量少。



8. 同一個bidder出價不同的商品

同一個機器人可能可以同時追蹤不一樣的商品並且出價



模型：Random forest

- Random forest 其實就是由很多棵 decision tree 所組成。顧名思義就是由許多不同的決策樹所組成的一個分類器可想成是結合多個弱分類器來建構一個更強分類器，這種方法又稱為Ensemble Method。是使用 Bagging 加上隨機特徵採樣的方法以大幅增進最終的運算結果所產生出來的演算法。
- 因為此競賽的預測結果是採二分法，所以我們選擇用 Random forest 模型來做訓練，而在 decision tree 演算法中，當模型的樹太深的話容易讓模型 Overfitting。因此藉由 Random forest 以多棵不同樹組成的概念，讓結果比較不容易 Overfitting，並使得預測能力更提升。

模型架構：

- 模型初始化：我們建立了五個 Random foreset 分類器 `rf1` 到 `rf5` 和一個過取樣器 `RandomOverSampler`以處理不平衡的資料。
- 模型組合：接著將過取樣器與每個 Random foreset 模型結合在一起，構建了五個帶有過取樣步驟的 Random foreset 模型 `pp1` 到 `pp5`。

[212]:
```
rf1 = RandomForestClassifier(random_state = 0)
rf2 = RandomForestClassifier(random_state = 123)
rf3 = RandomForestClassifier(random_state = 456)
rf4 = RandomForestClassifier(random_state = 789)
rf5 = RandomForestClassifier(random_state = 999)
ros = RandomOverSampler(sampling_strategy = 0.1, random_state = 456)

pp1 = make_pipeline(ros, rf1)
pp2 = make_pipeline(ros, rf2)
pp3 = make_pipeline(ros, rf3)
pp4 = make_pipeline(ros, rf4)
pp5 = make_pipeline(ros, rf5)
base_models = [pp1, pp2, pp3, pp4, pp5]
```

- 我們編寫了一個用於交叉驗證的函式。此函式在資料集上進行了多次交叉驗證，用於評估模型的性能。

```
def rf_cv(models, X, y):
    start = time()

    rskfold = RepeatedStratifiedKFold(n_splits = 10, n_repeats = 3, random_state = 456)
    k_fold_AUC = []

    for train_index, test_index in rskfold.split(X, y):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        y_proba = []
        for model in models:
            model.fit(X_train, y_train)
            sub_y_proba = model.predict_proba(X_test)[:, 1]
            y_proba.append(sub_y_proba)
        y_proba = np.mean(y_proba, axis = 0)

        AUC = roc_auc_score(y_test, y_proba)
        k_fold_AUC.append(AUC)

    mean_AUC = np.mean(k_fold_AUC)

    end = time()

    print(f"Time elapsed: {(end - start):.4f} seconds")
    print(f"AUC:{mean_AUC:.4f}")

    return k_fold_AUC
```

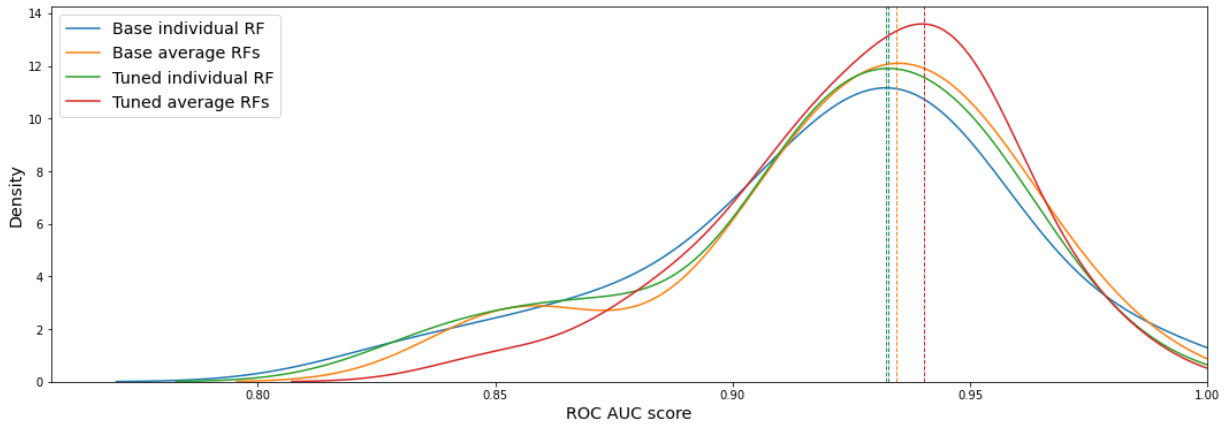
- 再來我們建立了一個用來微調參數的 `grid`，其中包含 Random foreset 模型的不同參數組合。

```
grid = {'randomforestclassifier__n_estimators': [50, 50, 60],
        'randomforestclassifier__max_depth': [None, 3, 3, 5],
        'randomforestclassifier__min_samples_split': [2, 4, 5],
        'randomforestclassifier__min_samples_leaf': [1, 2, 4],
       }
```

- 透過 `GridSearchCV` 在不同的參數組合上進行訓練和調優，並找到了每個模型的最佳參數組合。

```
start = time()

best_models = []

for model in base_models:

    search = GridSearchCV(estimator = model,
                          param_grid = grid,
                          scoring = 'roc_auc',
                          cv = 3,
                          verbose = 2,
                          n_jobs = -1)
    search.fit(X,y)
    best_models.append(search.best_estimator_)

end = time()

print(f"Time Elapsed: {(end - start):.4f} seconds")
```

最後我們繪製了 ROC AUC 分數的分佈，以比較整合平均和參數調整前後的差異



Private Score & Public Score：