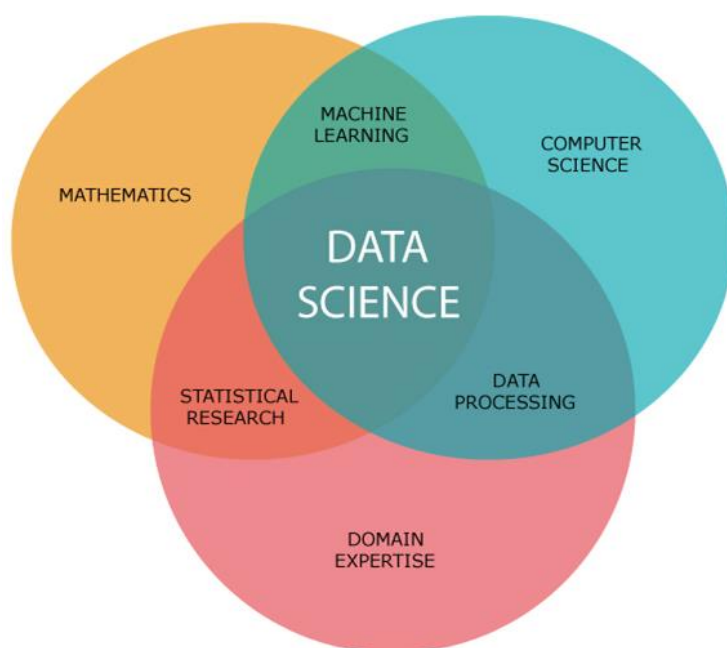


TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN  
KHOA CÔNG NGHỆ THÔNG TIN

BỘ MÔN:  
KHOA HỌC DỮ LIỆU

BÁO CÁO ĐỒ ÁN:

# DỰ ĐOÁN KHUNG GIÁ ĐIỆN THOẠI DI ĐỘNG



GVLT: TRẦN TRUNG KIÊN

## MỤC LỤC

I.	Thông tin nhóm	3
II.	Bài Toán	3
III.	Dữ liệu	4
IV.	Quá trình thực hiện và kết quả	6
V.	Phân công và đánh giá	10
VI.	Tài liệu tham khảo	11

## I. Thông tin nhóm:

MSSV	Họ và tên	Liên lạc
1612888	Phan Minh Sơn	<a href="mailto:1612888@student.hcmus.edu.vn">1612888@student.hcmus.edu.vn</a>
1612892	Trần Mạnh Thắng	<a href="mailto:1612892@student.hcmus.edu.vn">1612892@student.hcmus.edu.vn</a>

## II. Bài toán

Hiện nay, nhu cầu sử dụng smart phone ngày càng tăng cao, số lượng sản phẩm smart phone tung ra thị trường ngày càng nhiều. Để tăng tính cạnh tranh, các nhãn hàng sẽ cải thiện các thông số của điện thoại cũng như đưa ra giá cả hợp lý. Do đó, các công ty cần tham khảo giá thị trường các sản phẩm khác để định giá sản phẩm của mình dựa vào các chi tiết như CPU, RAM, Camera, thương hiệu ... Việc định giá được sản phẩm sẽ giúp nhà cung cấp đưa ra mức giá tốt hơn, doanh số bán hàng sẽ cao hơn. Đồng thời người tiêu dùng cũng sẽ biết được giá trị thực của sản phẩm mà không chạy theo giá trị thương hiệu, giúp chi tiêu sẽ hợp lý hơn.

Ở đồ án này, thực hiện sử dụng các thông tin về chi tiết của điện thoại để dự đoán khoảng giá của điện thoại.

Các thông tin – thuộc tính được sử dụng bao gồm 26 thuộc tính:

- Bluetooth
- width
- Height
- Thick
- Weight
- Memory card
- OS
- SoC
- Wi-Fi
- GPU core
- CPU core
- SIM type
- Number of SIM cards
- USB type
- USB version
- Position tracking
- Display size
- Display resolution
- Display color depth
- Image resolution
- Video resolution
- FPS
- Battery type
- Battery power
- RAM
- Storage

Phân lớp dự đoán:

- 0: price<300\$: mức giá thấp / rẻ.
- 1: 300\$<=price<600\$: mức giá trung bình
- 2: 600\$<=price<900\$: mức giá cao
- 3: 900\$ <=price: mức giá rất cao

### III. Dữ liệu:

Dữ liệu được thu thập từ trang web <https://www.devicespecifications.com>.  
Ta sẽ thu thập bằng cách parse HTML.

Có hơn 11000 link sản phẩm, tuy nhiên chỉ có gần 3000 sản phẩm là có ghi bảng giá. Ta sẽ train với gần 3000 sản phẩm đó (nếu cần thiết sẽ tự thu thập giá trị của các sản phẩm không có bảng giá để tăng kích thước bộ train).

Cụ thể:

Thực hiện thu thập 2 loại data:

#### **Loại 1: Data về những dữ liệu cơ bản của điện thoại**

Tập *dataset.csv*.

Số lượng, thuộc tính:

```
RangeIndex: 2811 entries, 0 to 2810
Data columns (total 21 columns):
Brand          2811 non-null object
Name           2811 non-null object
Dim            2811 non-null object
Weight         2811 non-null object
Bat            2811 non-null object
Blue           2811 non-null object
CPU            2811 non-null object
Cam            2811 non-null object
Cores          2811 non-null object
Dis            2811 non-null object
GPU            2811 non-null object
Memory card    2811 non-null object
OS             2811 non-null object
Pos            2811 non-null object
RAM            2811 non-null object
SIM            2811 non-null object
SoC            2811 non-null object
Storage        2811 non-null object
USB            2811 non-null object
Wi-Fi          2811 non-null object
Average price  2811 non-null object
```

#### **Loại 2: Data chi tiết về điện thoại**

Các tập *o1.csv* đến *o4.csv*.

Số lượng mẫu: 2857

Số lượng thuộc tính: 100

Do chi tiết điện thoại quá nhiều thuộc tính nên ta chỉ chọn lựa lại các thuộc tính để thực hiện bài toán. Hầu hết các tính nằm trong loại 1.

Cụ thể, dữ liệu sau khi tiền xử lý ở file `mobile_dataset.csv` bao gồm 26 thuộc tính dự đoán và 1 thuộc tính phân lớp (Đã trình bày ở mục I).

Dữ liệu được thu thập và chuẩn hóa một cách hợp lý phục vụ cho quá trình học và kiểm thử (Phần chuẩn hóa dữ liệu được trình bày cụ thể trong `Preprocessing.ipynb`):

*1- 2 thuộc tính Brand, Name sẽ bị loại bỏ.*

Giải thích: Brand và Name có ảnh hưởng đến giá của sản phẩm, song, nó không thể làm giá quá chênh lệch giữa các nhãn hiệu khi cùng cấu hình. Do đó, nó không gây ảnh hưởng đến loại giá cần dự đoán.

*2- Loại bỏ các thuộc tính có tỉ lệ giá trị thiếu > 20%.*

*3- PriceRange được chuẩn hóa như đã nêu ở phần 1.*

*4- Đối với dữ liệu thiếu:*

- Dữ liệu dạng numeric sẽ sử dụng mean điền giá trị thiếu.
- Dữ liệu dạng categorical, sử dụng mode để điền giá trị thiếu.

(được xử lý ở phần đầu file `train-and-test-with-some-models.ipynb`).

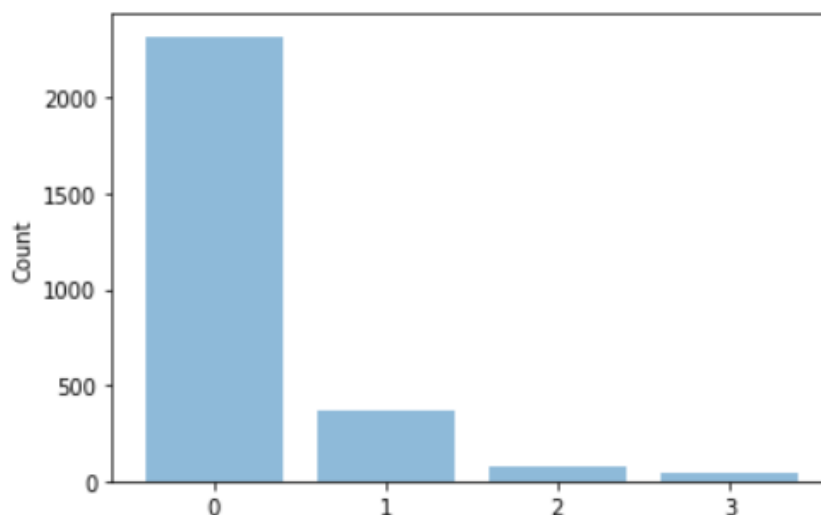
Như vậy, kết quả cuối cùng thu được, dùng để học và kiểm thử như sau:

*Tổng số mẫu: 2811 mẫu.*

*Tổng số thuộc tính để dự đoán: 26 thuộc tính (đã trình bày ở phần I).*

*Thuộc tính phân lớp: price\_range.*

Phân bố của các lớp:



#### IV. Quá trình thực hiện và kết quả:

Đối với bộ dữ liệu đã xử lý, thực hiện chia 2 tập train và test với tỷ lệ 4:1. Sử dụng các thuật toán khác nhau do thư viện sklearn hỗ trợ.

##### MLP Classifier - Multi-layer Perceptron classifier

MLP là một thuật toán học có giám sát. Nó học một hàm xấp xỉ hàm phi tuyến tính để phân loại hoặc hồi quy bằng cách training trên bộ dữ liệu có một tập hợp các features và một giá trị mục tiêu y. Nó khác với hồi quy logistic, ở chỗ giữa lớp đầu vào và lớp đầu ra, có thể có một hoặc nhiều lớp phi tuyến tính, được gọi là các lớp ẩn.

```
# MLP Classifier
full_pipeline = Pipeline(steps = [('preprocess_pipeline', preprocess_pipeline),
                                  ('NeuralNet', MLPClassifier(hidden_layer_sizes=(20), activation='tanh',
                                                              solver='lbfgs', random_state=0, max_iter=100
                                                              0))])

train_accuracy = []
validation_accuracy = []
alphas = [0.1, 0.5, 1, 10, 100, 1000]

for alpha in alphas:
    full_pipeline.set_params(NeuralNet__alpha = alpha)
    full_pipeline.fit(train_X_df, train_Y_df)
    train_accuracy.append(full_pipeline.score(train_X_df, train_Y_df) * 100)
    validation_accuracy.append(full_pipeline.score(validation_X, validation_Y) * 100)

print(train_accuracy)
print(validation_accuracy)
```

[93.3570581257414, 97.80545670225386, 98.75444839857651, 93.95017793594306, 86.1803084223013, 82.62158956109134]  
[85.05338078291815, 84.51957295373666, 83.98576512455516, 85.40925266903915, 85.76512455516014, 82.56227758007117]

Thực hiện chạy với tham số alpha thay đổi. Kết quả có thấy kết quả train, test khá cao. Kết quả đáng ghi nhận là: train\_accuracy = 98.75 và validation\_accuracy = 83.98.

##### Logistic Regression

Phương pháp hồi quy logistic là một mô hình hồi quy nhằm dự đoán giá trị đầu ra rời rạc (discrete target variable) y ứng với một véc-tơ đầu vào X. Việc này tương đương với chuyện phân loại các đầu vào X vào các lớp y tương ứng. Đầu ra có thể được thể hiện dưới dạng xác suất (probability).

```
# Logistic Regression
full_pipeline = Pipeline(steps = [('preprocess_pipeline', preprocess_pipeline),
                                  ('LogReg', LogisticRegression(multi_class = 'multinomial', solver = 'sa
                                  g', max_iter = 10000))])

full_pipeline.fit(train_X_df, train_Y_df)
print('Train accuracy: ', full_pipeline.score(train_X_df, train_Y_df) * 100)
print('Validation accuracy: ', full_pipeline.score(validation_X, validation_Y) * 100)
```

Train accuracy: 88.67141162514828  
Validation accuracy: 85.76512455516014

## Decision Tree

Cây quyết định (DTs) là một phương pháp học tập có giám sát không tham số được sử dụng để phân loại và hồi quy. Mục tiêu là tạo ra một mô hình dự đoán giá trị của biến mục tiêu bằng cách học các luật quyết định đơn giản được suy ra từ các features trong data.

Các lá của cây đề cập đến các lớp trong đó tập dữ liệu được chia.

Cây càng sâu, luật quyết định càng phức tạp và mô hình càng phức tạp.

```
# Decision Tree
full_pipeline = Pipeline(steps = [('preprocess_pipeline', preprocess_pipeline),
                                  ('Decision_Tree', DecisionTreeClassifier(random_state=101))])

full_pipeline.fit(train_X_df, train_Y_df)
print('Train accuracy: ', full_pipeline.score(train_X_df, train_Y_df) * 100)
print('Validation accuracy: ', full_pipeline.score(validation_X, validation_Y) * 100)

Train accuracy: 100.0
Validation accuracy: 79.1814946619217
```

## Random Forest

Random Forest là phương pháp xây dựng một tập hợp rất nhiều cây quyết định. Mỗi cây quyết định được tạo nên ngẫu nhiên từ việc tái chọn mẫu (chọn random 1 phần của data để xây dựng) và random các biến từ toàn bộ các biến trong data. Random Forest sử dụng phương pháp voting để đưa ra quyết định về biến target cần được dự báo. Câu trả lời nhận được nhiều bầu chọn nhất từ các cây quyết định sẽ được lựa chọn.

```
# Random Forest
full_pipeline = Pipeline(steps = [('preprocess_pipeline', preprocess_pipeline),
                                  ('Random_Forest', RandomForestClassifier(n_estimators = 100, random_state=101, criterion = 'entropy', oob_score = True))])

full_pipeline.fit(train_X_df, train_Y_df)
print('Train accuracy: ', full_pipeline.score(train_X_df, train_Y_df) * 100)
print('Validation accuracy: ', full_pipeline.score(validation_X, validation_Y) * 100)

Train accuracy: 100.0
Validation accuracy: 85.23131672597864
```

## KNN

KNN là thuật toán phân loại đơn giản nhất. Thuật toán phân loại này không phụ thuộc vào cấu trúc của dữ liệu. Khi đưa một dữ liệu mới vào, k hàng xóm gần nhất của nó (k nearest neighbors) từ dữ liệu đào tạo được kiểm tra. Khoảng cách giữa hai ví dụ có thể là khoảng cách euclidean giữa các vector đặc trưng của chúng. Lớp chiếm đa số trong số k hàng xóm gần nhất được coi là lớp cho dữ liệu mới đó

```
# KNN
full_pipeline = Pipeline(steps = [('preprocess_pipeline', preprocess_pipeline),
                                  ('KNN', KNeighborsClassifier(n_neighbors=3))])

full_pipeline.fit(train_X_df, train_Y_df)
print('Train accuracy: ', full_pipeline.score(train_X_df, train_Y_df) * 100)
print('Validation accuracy: ', full_pipeline.score(validation_X, validation_Y) * 100)

Train accuracy: 90.0355871886121
Validation accuracy: 83.98576512455516
```

*Nhận xét: Thuật toán Random Forest đạt các kết quả ở tập train và validation ổn nhất.*

Ta thực hiện tùy chỉnh tham số trên Random Forest để đạt kết quả tốt nhất.

```
n_es=[x for x in range(100,1050,50)]
criteria=['entropy','gini']
max_f=['auto','log2',None]
#print(n_es)
#n_estimators = 100,random_state=101, criterion = 'entropy'
for criterion in criteria:
    print(criterion)
    for f in max_f:
        print(f)
        for n in n_es:
            print(n)
            full_pipeline = Pipeline(steps = [('preprocess_pipeline', preprocess_pipeline),
                                              ('Random_Forest', RandomForestClassifier(oob_score = True,random_state=40))])
            full_pipeline.set_params(Random_Forest__n_estimators=n,Random_Forest__criterion=criterion,
                                     Random_Forest__max_features=f,Random_Forest__random_state=54)
            full_pipeline.fit(train_X_df, train_Y_df)
            print('Train accuracy: ', full_pipeline.score(train_X_df, train_Y_df) * 100)
            print('Test accuracy: ', full_pipeline.score(validation_X, validation_Y) * 100)
```

Kết quả tốt nhất thu được với bộ tham số thay đổi là:

*criterion='gini', n\_estimators=600, max\_features=auto là:*

*Train=100*

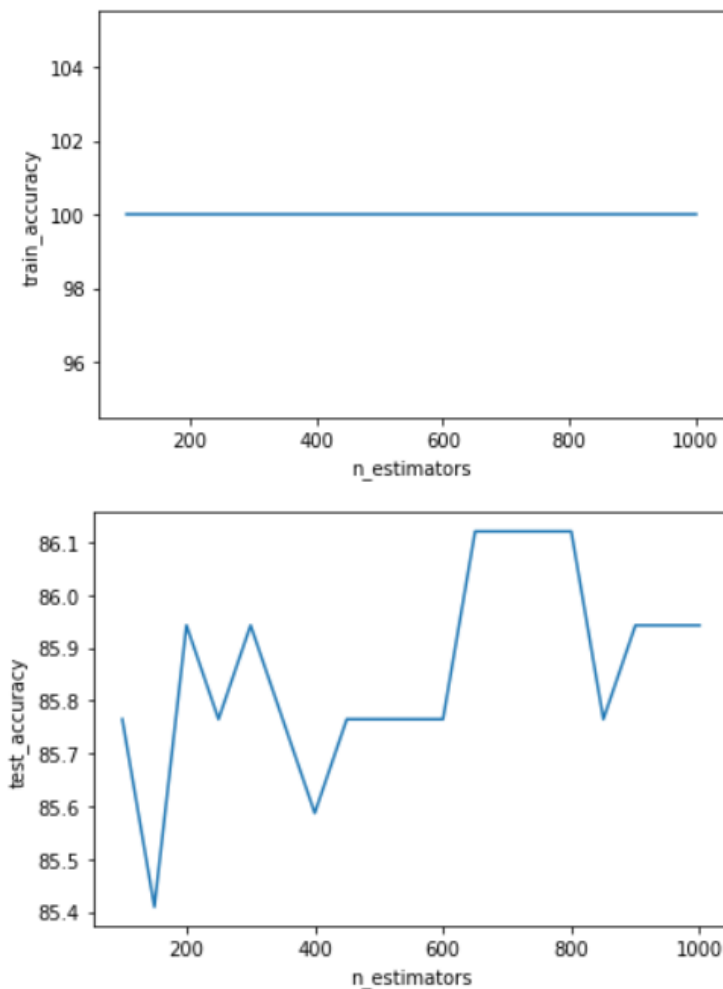
*Test=86.833*

Nhận xét, qua thực nghiệm cho thấy:

- *n\_estimators* và *random\_state* có ảnh hưởng đến độ chính xác.
- Độ chính xác trên tập train ít thay đổi, độ chính xác trên tập test thay đổi nhiều hơn.



Ảnh hưởng của  $n\_estimators$ :



$n\_estimators$  tăng:  $train\_accuracy$  gần như không đổi,  $test\_accuracy$  có thay đổi nhiều.

### Lí do chọn Random Forest khi so sánh:

- Với *Logistic Regression*

Đối với bộ dữ liệu mất cân bằng như chúng ta đang xét, Random Forest có thể là lựa chọn tốt hơn [1]

- Với *Decision Tree*

Do Random Forest xây dựng nhiều cây quyết định khác nhau và thực hiện voting (độ bao quát cho nhiều trường hợp sẽ cao hơn) nên Random Forest ít bị overfitting hơn Decision Tree (cây quyết định) cho một tập dữ liệu cụ thể.

- Với KNN:

Độ tổng quát trên toàn bộ dữ liệu của Random Forest sẽ tốt hơn nhiều so với KNN do KNN chỉ xét trên n điểm gần nhất

## V. Phân công và đánh giá:

Nhận xét về quá trình thực hiện được:

Về thu thập dữ liệu:

- Bài toán chưa thực sự thực tế.
- Nguồn dữ liệu khá khó tìm do các trang thương mại thường không đồng bộ nhiều về thuộc tính sản phẩm.
- Dữ liệu thô được lấy dễ bởi html request nhưng do có vấn đề với bộ ba giá trị (RAM, storage, price) (ví dụ: cùng giá trị RAM và Storage nhưng lại có 2 mức giá khác nhau cho 2 phiên bản khác)-> giải pháp lấy giá trị trung bình : giải pháp còn chưa thỏa đáng.
- Xử lý dữ liệu thiếu: sử dụng mean cho numeric và mode cho categorical: giải pháp chưa thỏa đáng.

Về thực hiện học và kiểm thử:

- Sử dụng được nhiều thuật toán classification lên data, cho kết quả khá tốt.
- Tuy nhiên, do ảnh hưởng của dữ liệu nên kết quả trên tập test không đạt được đến 90%.

Đã học được:

- Cách thu thập dữ liệu.
- Cách xử lý dữ liệu.
- Sử dụng thư viện với các mô hình máy học khác nhau.

Câu hỏi cần giải đáp:

- Đối với bài toán như ban đầu mong muốn là regression để tính đúng giá, nhưng việc này lại quá khó do có quá nhiều mức giá khác nhau nên sẽ có những giá chỉ xuất hiện một lần trong dataset. Vậy trong thực tế có những mô hình và bài toán nào đã giải quyết được vấn đề này.
- Chiến lược chọn mô hình tối ưu là gì thay vì phải thử nghiệm hết tất cả.

Phân công:

Công việc	Người thực hiện	Đánh giá
Thu thập dữ liệu – thông tin cơ bản	Trần Mạnh Thắng	HT
Thu thập dữ liệu – Thông tin chi tiết	Phan Minh Sơn	HT
Chuẩn hóa dữ liệu	Trần Mạnh Thắng	HT
Học và kiểm thử	Trần Mạnh Thắng	HT
Thay đổi tham số để tối ưu kết quả	Phan Minh Sơn	HT
Báo cáo, Slide	Phan Minh Sơn	HT

**VI. Tài liệu tham khảo:**

<https://scikit-learn.org/stable/index.html>

<http://davidsiroky.faculty.asu.edu/PA2016.pdf>