

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



BÁO CÁO BÀI TẬP LỚN KHO DỮ LIỆU VÀ KHAI PHÁ DỮ LIỆU

**Đề tài: XÂY DỰNG CÂY QUYẾT ĐỊNH
PHÂN LOẠI TIỀN THẬT VÀ TIỀN GIẢ**

Giảng viên: (Cô) Nguyễn Quỳnh Chi

Sinh viên thực hiện: Lớp 01 - Nhóm 09

| | |
|-------------------|------------|
| Lê Văn Sang | B17DCCN530 |
| Trần Việt Huy | B17DCCN326 |
| Nguyễn Hồng Cường | B17DCCN044 |
| Trần Minh Ngọc | B17DCCN470 |
| Nguyễn Phú Thịnh | B17DCCN578 |

Hà Nội, tháng 6 năm 2021

Mục lục

| | |
|--|-----------|
| Chương 1. TỔNG QUAN | 3 |
| 1.1 Mục tiêu | 3 |
| 1.2 Phạm vi | 3 |
| 1.3 Phương pháp nghiên cứu | 4 |
| Chương 2. XÂY DỰNG CÂY QUYẾT ĐỊNH | 5 |
| 2.1 Bài toán | 5 |
| 2.2 Xây dựng cây quyết định | 5 |
| 2.2.1 Thuật toán Cart | 5 |
| 2.2.2 K-Fold Cross Validation | 7 |
| 2.2.3 Điều kiện dừng | 9 |
| 2.3 Prediction | 10 |
| Chương 3. KẾT QUẢ | 12 |
| 3.1 Kết quả chạy mô hình | 12 |
| 3.2 So sánh với thư viện có sẵn | 12 |
| 3.3 Kết luận | 13 |
| Tài liệu tham khảo | 14 |

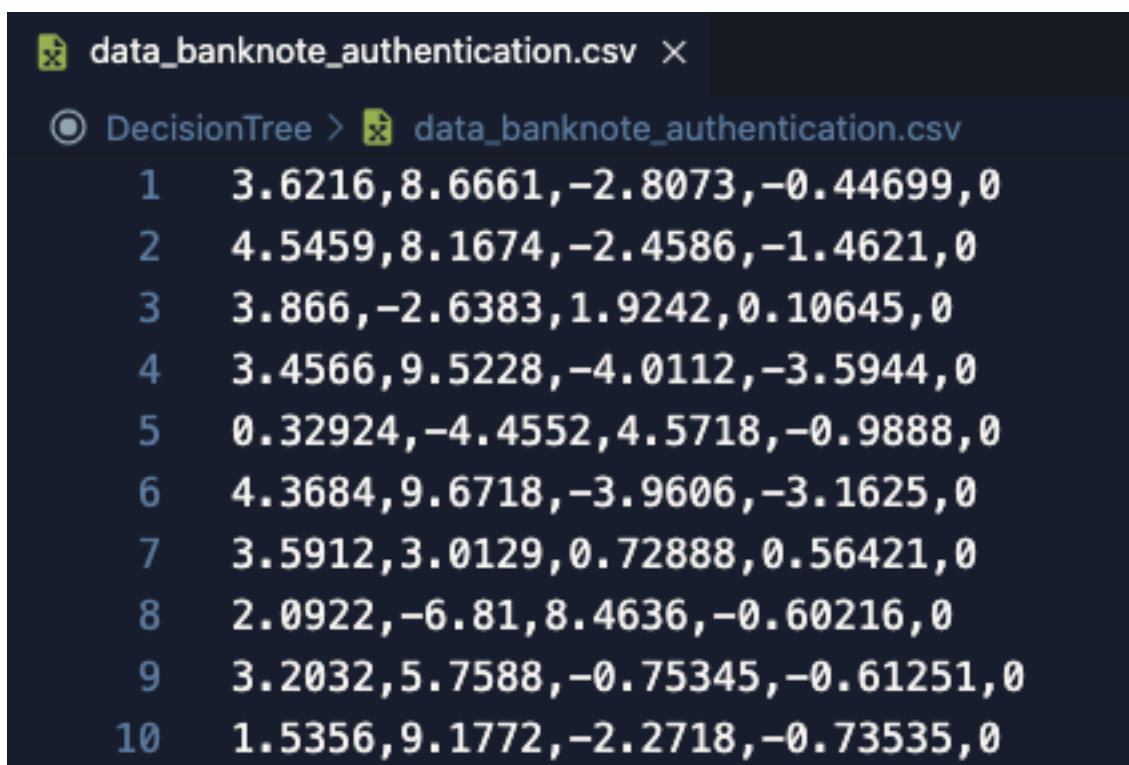
Chương 1. TỔNG QUAN

1.1 Mục tiêu

- Mục tiêu của báo cáo là xây dựng cây quyết định cho bài toán phân loại tiền giả, tiền thật dựa vào 5 attributes (bao gồm cả class)
- Cài đặt và đánh giá thuật toán.
- Sau khi đã viết code xây dựng cây quyết định, sử dụng thư viện sklearn để kiểm tra lại mức độ chính xác.

1.2 Phạm vi

- Tập trung tìm hiểu các kỹ thuật xây dựng cây quyết định
- Sử dụng ngôn ngữ lập trình python để xây dựng cây quyết định phân loại tiền giả, tiền thật dựa vào thuật toán Cart với độ đo không đồng nhất GINI
- Cài đặt và thử nghiệm với dữ liệu đã có là file csv
- Sử dụng thư viện sklearn để kiểm tra lại mức độ chính xác của cây quyết định được xây dựng.
- Mô tả dataset:



```
data_banknote_authentication.csv X
DecisionTree > data_banknote_authentication.csv
1 3.6216,8.6661,-2.8073,-0.44699,0
2 4.5459,8.1674,-2.4586,-1.4621,0
3 3.866,-2.6383,1.9242,0.10645,0
4 3.4566,9.5228,-4.0112,-3.5944,0
5 0.32924,-4.4552,4.5718,-0.9888,0
6 4.3684,9.6718,-3.9606,-3.1625,0
7 3.5912,3.0129,0.72888,0.56421,0
8 2.0922,-6.81,8.4636,-0.60216,0
9 3.2032,5.7588,-0.75345,-0.61251,0
10 1.5356,9.1772,-2.2718,-0.73535,0
```

- Bộ dữ liệu gồm 1372 bản ghi (mỗi row tương ứng 1 bản ghi)
- Dữ liệu được trích xuất từ các hình ảnh được lấy từ các mẫu giống với tiền thật và giả.

- Gồm 5 thuộc tính tương ứng với 5 cột trong mỗi bản ghi: variance, skewness, curtosis, entropy of image, class (integer)

1.3 Phương pháp nghiên cứu

- Phương pháp nghiên cứu tài liệu: Phân tích và tổng hợp các tài liệu về khai phá dữ liệu sử dụng thuật toán về decision tree có thuật toán Cart, phân loại dữ liệu, mô hình dự báo.
- Phương pháp thực nghiệm: Ứng dụng kết hợp kỹ thuật phân loại và mô hình decision tree để phân loại tiền giả, tiền thật dựa vào cây quyết định đã xây dựng. Cuối cùng kiểm thử và đánh giá kết quả dùng thư viện sklearn.

Chương 2. XÂY DỰNG CÂY QUYẾT ĐỊNH

2.1 Bài toán

Xây dựng cây quyết định để phân loại đầu là tiền thật và tiền giả.

| Tên attribute | Ý nghĩa |
|------------------|------------|
| Variance | Phương sai |
| Skewness | Độ lệch |
| Curtosis | Độ nhọn |
| Entropy of image | Entropy |

Đầu ra của cây quyết định là 2 lớp: Tiền thật (1), Tiền giả (0)

2.2 Xây dựng cây quyết định

Các giá trị của các thuộc tính trong dataset đều là các giá trị tuyến tính nên nhóm em quyết định xây dựng cây quyết định nhị phân (Binary Decesion Tree).

2.2.1 Thuật toán Cart

- Cấu tạo của cây quyết định bao gồm các nút trong cây và nút lá:
 - + Nút trong cây là các thuộc tính
 - + Nút lá là các giá trị của lớp
- Việc xây dựng cây quyết định chúng ta sử dụng chiến lược tham lam: phân tách bản ghi thành các nhánh dựa trên phép kiểm tra giá trị thuộc tính để tối ưu một tiêu chí nào đó. Việc tìm điều kiện để kiểm tra còn phụ thuộc vào số lượng nhánh muốn phân tách: có thể phân tách thành hai hoặc nhiều nhánh.
- Các feature dữ liệu của bài toán là cho dưới dạng tuyến tính chứ không phải dưới dạng categorical, do đó sẽ lựa chọn một ngưỡng để phân nhánh bằng cách duyệt qua toàn bộ các value có thể của feature đó trong tập bản ghi ở node đang xét và tính toán gini_split để chọn được ngưỡng threshold tốt nhất cho việc chia ra 2 node con.
- Tại mỗi nút, ta sẽ xét lần lượt từng thuộc tính (giả sử số thứ tự của thuộc tính trong list thuộc tính là index).

Với mỗi thuộc tính, xét lần lượt từng bản ghi (từng row trong dataset). Với mỗi bản ghi, ta lấy ra giá trị của thuộc tính đó (là `row[index]`).

Dựa vào giá trị này, ta sẽ phân số bản ghi tại nút đó thành 2 phần:

- + < row[index] : left
- + >= row[index] : right

```
83 # Split a dataset based on an attribute and an attribute value
84 def test_split_group(index, value, dataset):
85     left, right = list(), list()
86     for row in dataset:
87         if row[index] < value:
88             left.append(row)
89         else:
90             right.append(row)
91     return left, right
92
93 # Select the best split point for a dataset
94 def get_split(dataset):
95     class_values = list(set(row[-1] for row in dataset))
96     b_index, b_value, b_score, b_groups, b_name = 999, 999, 999, None, None
97     for index in range(len(dataset[0])-1):
98         for row in dataset:
99             groups = test_split_group(index, row[index], dataset)
100             gini = gini_index(groups, class_values)
101             if gini < b_score:
102                 b_index, b_value, b_score, b_groups, b_name = index, row[index], gini, groups, schema[index]
103     return {'index':b_index, 'value':b_value, 'groups':b_groups, 'name': b_name}
104
```

Khi có nhiều cách phân nhánh mỗi cách có thể phân ra một số node nhất định. Cho nên, lúc này có thêm công thức để tìm ra các phân chia tối ưu nhất:

$$G_{split} = \sum_{i=1}^k \frac{N_i}{N} G(i)$$

Trong đó:

- N_i là số điểm dữ liệu có trong node của nhánh được phân
- N là số điểm dữ liệu có trong node được dùng để phân nhánh

Hệ số G_{split} càng nhỏ thì cách phân nhánh đó càng tối ưu.

Hàm tính G_{split} với đầu vào: groups là nhóm các bản ghi sẽ được chia vào cùng một nhánh; classes là tập các giá trị đầu ra (giá trị node lá) có trong các bản ghi.

```

70 # Calculate the Gini index for a split dataset
71 def gini_index(groups, classes):
72     # count all samples at split point
73     n_instances = float(sum([len(group) for group in groups]))
74     # sum weighted Gini index for each group
75     gini_split = 0.0
76     for group in groups:
77         size = float(len(group))
78         # avoid divide by zero
79         if size == 0:
80             continue
81         score = 0.0
82         # score the group based on the score for each class
83         for class_val in classes:
84             p = [row[-1] for row in group].count(class_val) / size
85             score += p * p
86         # weight the group score by its relative size
87         gini_split += (1.0 - score) * (size / n_instances)
88     return gini_split

```

2.2.2 K-Fold Cross Validation

Cross validation là một kỹ thuật lấy mẫu để đánh giá mô hình học máy trong trường hợp dataset không nhiều. Tham số quan trọng trong kỹ thuật này là k, đại diện cho số nhóm mà dữ liệu sẽ được chia ra.

Kỹ thuật này thường bao gồm các bước như sau:

- Bước 1. Xáo trộn dataset một cách ngẫu nhiên
 - Bước 2. Chia dataset thành k nhóm
 - Bước 3. Với mỗi nhóm:
 - + Sử dụng nhóm hiện tại để đánh giá hiệu quả mô hình
 - + Các nhóm còn lại được sử dụng để huấn luyện mô hình
 - + Huấn luyện mô hình
 - + Đánh giá và xóa các trọng số của mô hình
 - Bước 4. Tổng hợp hiệu quả của mô hình dựa từ các số liệu đánh giá
- Để chia dataset ra thành k folds thì ở đây ta sử dụng hàm *cross_validation_split()* với đầu vào là bộ dataset với n_folds là số fold muốn chia ra.
 Trong hàm này có sử dụng hàm *randrange()* để chọn bản ghi ngẫu nhiên do tập dataset ban đầu được sắp xếp một nửa đầu là bản ghi class 0, nửa sau là bản ghi class 1.
 - Sau khi đã chia dataset ra là k folds, ta thực hiện xây cây quyết định với 1 fold là dùng để test, k-1 folds còn lại để train, lần lượt đối với từng fold.

```

31 # Split a dataset into k folds
32 def cross_validation_split(dataset, n_folds):
33     dataset_split = list()
34     dataset_copy = list(dataset)
35     fold_size = int(len(dataset) / n_folds)
36     for i in range(n_folds):
37         fold = list()
38         while len(fold) < fold_size:
39             index = randrange(len(dataset_copy))
40             fold.append(dataset_copy.pop(index))
41         dataset_split.append(fold)
42     return dataset_split
43
44 # Evaluate an algorithm using a cross validation split
45 def evaluate_algorithm(dataset, algorithm, n_folds, *args):
46     folds = cross_validation_split(dataset, n_folds)
47     scores = list()
48     for fold in folds:
49         train_set = list(folds)
50         train_set.remove(fold)
51         train_set = sum(train_set, [])
52         test_set = list()
53         for row in fold:
54             row_copy = list(row)
55             test_set.append(row_copy)
56             row_copy[-1] = None
57         predicted = algorithm(train_set, test_set, *args)
58         actual = [row[-1] for row in fold]
59         accuracy = accuracy_metric(actual, predicted)
60         scores.append(accuracy)
61     return scores

```

Mỗi mẫu chỉ được gán cho duy nhất một nhóm và phải ở nguyên trong nhóm đó cho đến hết quá trình. Việc hủy mô hình sau mỗi lần đánh giá là bắt buộc, tránh trường hợp mô hình ghi nhớ nhãn của tập test trong lần đánh giá trước.

Kết quả tổng hợp thường là trung bình của các lần đánh giá. Ngoài ra việc bổ sung thông tin về phương sai và độ lệch chuẩn vào kết quả tổng hợp cũng được sử dụng trong thực tế.

Ba chiến thuật phổ biến để lựa chọn k:

- + Đại diện: Giá trị của k được chọn để mỗi tập train/test đủ lớn, có thể đại diện về mặt thống kê cho dataset chứa nó.
- + $k=10$: Giá trị của k được gán cố định bằng 10, một giá trị thường được sử dụng và được chứng minh là cho sai số nhỏ, phương sai thấp (thông qua thực nghiệm).
- + $k=n$: Giá trị của k được gán cố định bằng n, với n là kích thước của dataset, như vậy mỗi mẫu sẽ được sử dụng để đánh giá mô hình một lần.

Để đánh giá hiệu quả của một mô hình, dựa trên bộ test đã được xây dựng, ta tính được số kết quả predict đúng với kết quả thực.

Công thức:

Score = tổng số kết quả predict đúng / tổng số bộ test.

```

33  # Calculate accuracy percentage
34  def accuracy_metric(actual, predicted):
35      correct = 0
36      for i in range(len(actual)):
37          if actual[i] == predicted[i]:
38              correct += 1
39      return correct / float(len(actual)) * 100.0

```

2.2.3 Điều kiện dừng

Điều kiện dừng là:

- + Dừng phân nhánh một nút khi các bản ghi thuộc cùng một lớp
- + Dừng phân nhánh một nút khi tất cả các bản ghi có giá trị thuộc tính giống nhau
- + Kết thúc sớm trong một số trường hợp đặc biệt

Trong trường hợp overfitting tức là cây quyết định của chúng ta tạo ra là quá sum suê, nhiều hơn những gì chúng ta cần thì để xử lý overfitting sử dụng max_depth và min_size.

- + Khi độ sâu của cây vượt quá max_depth thì sẽ không tiếp tục phát triển cây nữa mà sẽ chọn số giá trị nhãn chiếm nhiều hơn trong tập bản ghi ở node để làm giá trị lá.
- + Khi số bản ghi ở một node nhỏ hơn min_size thì cũng không tiếp tục phát triển cây nữa mà sẽ chọn số giá trị nhãn chiếm nhiều hơn trong tập bản ghi ở node để làm giá trị lá.

```

102 # Create a terminal node value
103 def to_terminal(group):
104     outcomes = [row[-1] for row in group]
105     return max(set(outcomes), key=outcomes.count)
106
107 # Create child splits for a node or make terminal
108 def split(node, max_depth, min_size, depth):
109     left, right = node['groups']
110     del(node['groups'])
111     # check for a no split
112     if not left or not right:
113         node['left'] = node['right'] = to_terminal(left + right)
114         return
115     # check for max depth
116     if depth >= max_depth:
117         node['left'], node['right'] = to_terminal(left), to_terminal(right)
118         return
119     # process left child
120     if len(left) <= min_size:
121         node['left'] = to_terminal(left)
122     else:
123         node['left'] = get_split(left)
124         split(node['left'], max_depth, min_size, depth+1)
125     # process right child
126     if len(right) <= min_size:
127         node['right'] = to_terminal(right)
128     else:
129         node['right'] = get_split(right)
130         split(node['right'], max_depth, min_size, depth+1)

```

2.3 Prediction

Sau khi đã xây dựng được cây quyết định, ta đến bước test.

Trong mỗi vòng lặp, ta sử dụng k-1 folds làm bộ train và 1 fold làm bộ test. Như vậy, với mỗi bản ghi ta sẽ thực hiện predict.

Hàm predict được thể hiện như sau:

```

143 # Make a prediction with a decision tree
144 def predict(node, row):
145     if row[node['index']] < node['value']:
146         if isinstance(node['left'], dict):
147             return predict(node['left'], row)
148         else:
149             return node['left']
150     else:
151         if isinstance(node['right'], dict):
152             return predict(node['right'], row)
153         else:
154             return node['right']

```

Tại mỗi row,

- Nếu giá trị tại *node['index']* nhỏ hơn giá trị tại *node['value']*
 - + nếu *node['left']* là một dictionary, tức là còn chia nhánh tiếp thì ta sẽ thực hiện hàm predict với nhánh trái.
 - + nếu không thì trả về class của *node['left']* (nhánh trái).
- Nếu giá trị tại *node['index']* lớn hơn hoặc bằng giá trị tại *node['value']*
 - + nếu *node['right']* là một dictionary, tức là còn chia nhánh tiếp thì ta sẽ thực hiện hàm predict với nhánh phải.
 - + nếu không thì trả về class của *node['right']* (nhánh phải).

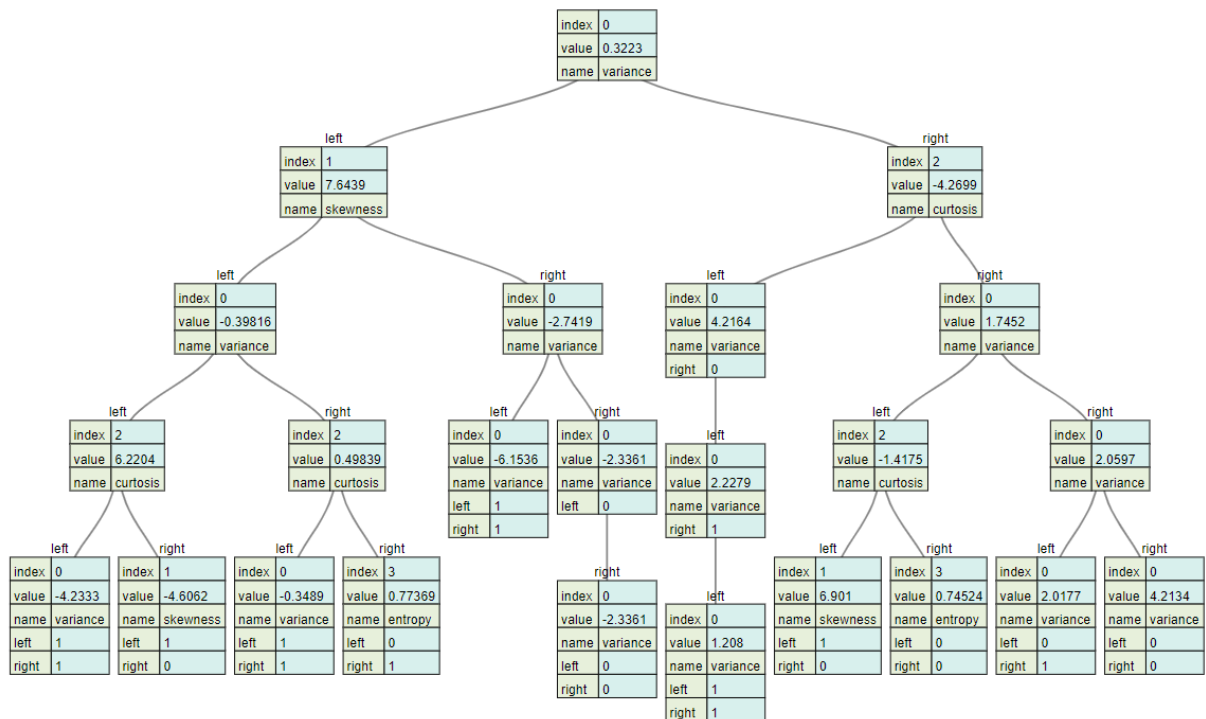
Chương 3. KẾT QUẢ

3.1 - Kết quả chạy mô hình

Kết quả sau khi chạy, với mỗi lần chọn fold ta sẽ ra được một cây quyết định có dạng danh sách các key:value như hình dưới.

```
(env) C:\Users\Admin\Desktop\hoc_tron_tiep\khoa_va_khai_phan_lieu\DecisionTree
> Evn A python test.py
Shape of data: (1372, 5)
{'index': 0, 'value': 0.76163, 'name': 'variance', 'left': {'index': 1, 'value': 6.5989, 'name': 'skewness', 'left': {'index': 0, 'value': -0.38398, 'name': 'variance', 'left': {'index': 2, 'value': 6.7
left': {'index': 0, 'value': -1.2369, 'name': 'variance', 'left': 1.0, 'right': 1.0}, 'right': {'index': 1, 'value': -4.6062, 'name': 'skewness', 'left': 1.0, 'right': 0.0}},
right': {'index': 2, 'value': 2.1992, 'name': 'curtosis', 'left': {'index': 1, 'value': 5.527, 'name': 'skewness', 'left': 1.0, 'right': 0.0}, 'right': {'index': 1, 'value': -5.4981, 'name': 'skewness',
left': 1.0, 'right': 0.0}}, 'right': {'index': 0, 'value': -2.7143, 'name': 'variance', 'left': {'index': 0, 'value': -6.2003, 'name': 'variance', 'left': 1.0, 'right': 1.0}, 'right': {'index': 0, 'v
value': -1.5222, 'name': 'variance', 'left': {'index': 0, 'value': -1.8411, 'name': 'variance', 'left': 0.0, 'right': 0.0}, 'right': {'index': 0, 'value': -1.5222, 'name': 'variance', 'left': 0.0, 'right
0.0}}}, 'right': {'index': 0, 'value': 1.594, 'name': 'variance', 'left': {'index': 2, 'value': 2.2718, 'name': 'curtosis', 'left': {'index': 1, 'value': 7.6377, 'name': 'skewness', 'left': {'index
0, 'value': 1.0277, 'name': 'entropy', 'left': {'index': 3, 'value': 1.0271, 'name': 'entropy', 'left': {'index': 3, 'value': 0.65472, 'name': 'entropy',
left': 0.0, 'right': 1.0}}, 'right': {'index': 0, 'value': 2.0466, 'name': 'variance', 'left': {'index': 0, 'value': 2.831, 'name': 'variance', 'left': {'index': 0, 'value': 2.0165, 'nam
e': 'variance', 'left': 0.0, 'right': 1.0}, 'right': {'index': 0, 'value': 3.5458, 'name': 'variance', 'left': {'index': 0, 'value': 2.1265, 'name': 'variance', 'left': 0.0, 'right': 0.0}
right': {'index': 0, 'value': 3.5458, 'name': 'variance', 'left': 0.0, 'right': 0.0}}}}
{'index': 0, 'value': 0.32444, 'name': 'variance', 'left': {'index': 1, 'value': 7.6274, 'name': 'skewness', 'left': {'index': 0, 'value': -0.45062, 'name': 'variance', 'left': {'index': 2, 'value': 6.2
204, 'name': 'curtosis', 'left': {'index': 0, 'value': -5.119, 'name': 'variance', 'left': 1.0, 'right': 1.0}, 'right': {'index': 1, 'value': 0.070346, 'name': 'skewness', 'left': 1.0, 'right': 0.0}},
right': {'index': 0, 'value': 1.0747, 'name': 'curtosis', 'left': {'index': 1, 'value': 6.2761, 'name': 'skewness', 'left': 1.0, 'right': 0.0}, 'right': {'index': 3, 'value': 0.97322, 'name': 'entropy',
left': 1.0, 'right': 1.0}}, 'right': {'index': 0, 'value': -4.2859, 'name': 'variance', 'left': 1.0, 'right': {'index': 0, 'value': -1.2537, 'name': 'variance', 'left': {'index': 0, 'value': -1.7539,
name': 'variance', 'left': 0.0, 'right': 0.0}, 'right': {'index': 0, 'value': -1.2537, 'name': 'variance', 'left': 0.0, 'right': 0.0}}}, 'right': {'index': 2, 'value': -4.3839, 'name': 'curtosis', 'le
ft': {'index': 0, 'value': 4.2164, 'name': 'variance', 'left': {'index': 0, 'value': 0.89512, 'name': 'variance', 'left': 1.0, 'right': {'index': 0, 'value': 0.89512, 'name': 'variance', 'left': 1.0, 'r
right': 1.0}}, 'right': {'index': 0, 'value': 0.89566, 'name': 'variance', 'left': {'index': 2, 'value': 0.3493, 'name': 'curtosis', 'left': {'index': 1, 'value': 6.6424, 'name': 'skewness
left': 1.0, 'right': 0.0}, 'right': {'index': 0, 'value': 0.5693, 'name': 'variance', 'left': 0.0, 'right': 0.0}}, 'right': {'index': 0, 'value': 1.7452, 'name': 'variance', 'left': {'index': 2, 'v
value': 2.7473, 'name': 'curtosis', 'left': {'index': 0, 'value': 0.0, 'right': 0.0}, 'right': {'index': 0, 'value': 2.0193, 'name': 'variance', 'left': 0.0, 'right': 0.0}}}}
Scores Min: [06.5014577294753, 97.376093294440685]
Mean Accuracy Gini: 96.939%
Scores Gini Sklearn: [08.54227405247813, 94.89795918367348]
Mean Accuracy Gini Sklearn: 96.720%
```

Khi visualize một cây quyết định ta được hình dạng cây như sau:



3.2 So sánh với thư viện có sẵn

Để so sánh kết quả của cây quyết định tự xây dựng, ở đây ta sử dụng thư viện Sklearn.

Hàm xây dựng cây quyết định dựa vào thư viện Sklearn:

```
165 # Classification and Regression Tree Algorithm using sklearn
166 def decision_tree_lib(train, test, max_depth, min_size):
167     x_train = []
168     y_train = []
169     for ele in train:
170         x_train.append(ele[:-1])
171         y_train.append(ele[-1])
172     x_train = np.array(x_train)
173     y_train = np.array(y_train)
174     clf = DecisionTreeClassifier(criterion="gini", splitter="best", random_state=0, max_depth=max_depth, min_samples_split=min_size+1)
175     tree = clf.fit(x_train, y_train)
176     predictions = list()
177     for row in test:
178         prediction = tree.predict(np.array(row[:-1]).reshape(1, -1))
179         predictions.append(prediction)
180     return(predictions)
```

Với mỗi vòng lặp, hàm xây dựng cây quyết định với thư viện Sklearn được sử dụng và cũng tính điểm như cây quyết định được xây dựng bằng code của nhóm.

Trong đó, Mean Accuracy Gini là trung bình các lần tính điểm.

```
Scores Gini: [96.50145772594753, 97.37609329446065]
Mean Accuracy Gini: 96.939%
Scores Gini Sklearn: [98.54227405247813, 94.89795918367348]
Mean Accuracy Gini Sklearn: 96.720%
```

Dựa theo kết quả mô hình ở trên, ta thấy sự khác biệt giữa sử dụng thư viện Sklearn và mô hình do chúng em xây dựng không quá lớn, do vậy có thể đánh giá thuật toán tự xây dựng cũng cho kết quả rất khả quan. Tuy nhiên, vẫn có sự sai khác nhất định, nguyên nhân có thể đến từ các bước xây dựng thuật toán, các phương pháp tiền xử lý,...

3.3 Kết luận

Trong quá trình nghiên cứu, tìm hiểu và hoàn thành bài tập lớn với Data Mining nhận thấy đây là một lĩnh vực nghiên cứu rộng lớn, còn nhiều điều cần phải khám phá. Do thời gian thực hiện hạn chế nên chúng em mới chỉ tìm hiểu để xây dựng và giải quyết bài toán được đến như vậy. Trong thời gian tới em sẽ cố gắng tiếp tục nghiên cứu và hoàn thiện nhằm làm cho bài toán có kết quả tốt hơn.

Một lần nữa chúng em xin được chân thành cảm ơn sự hướng dẫn tận tình của cô!

Tài liệu tham khảo

[1] Bài giảng kho dữ liệu và khai phá dữ liệu, Tác giả Nguyễn Quỳnh Chi, năm 2014

[2] Dataset

https://archive.ics.uci.edu/ml/datasets/banknote+authentication?fbclid=IwAR0OkHtqjnFfj1THpaZ7aEAisdZRtjcaY5mEmMCfSuPxj0sPYIc6_amc7yI