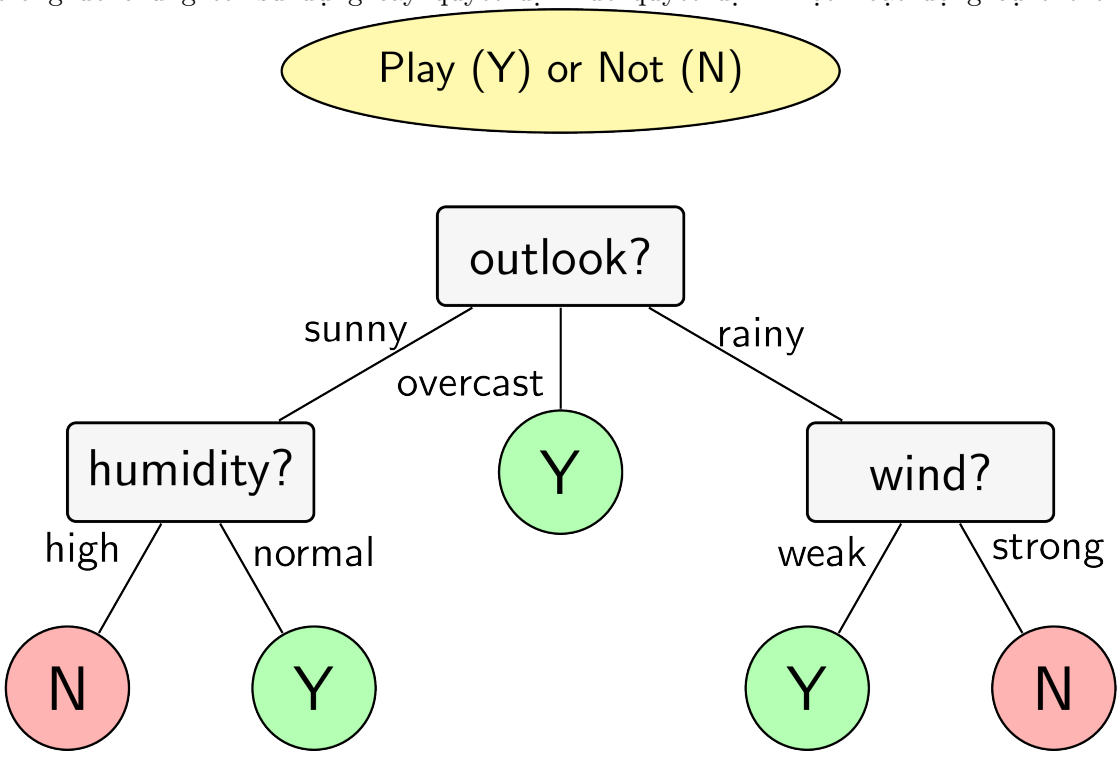
Decision Tree and KNN

August 9, 2020

1 Giới thiệu decision tree

Cây quyết định (Decision Tree): là một model hấp dẫn nếu chúng ta quan tâm đến khả năng diễn giải. Giống như tên của nó "Cây quyết định". Chúng ta có thể khai thác dữ liệu của chúng ta thông qua một loạt câu hỏi và quyết định. Hãy xem xét ví dụ sau, trong đó chúng tôi sử dụng cây quyết định để quyết định một hoạt động cụ thể trong ngày:



Dựa vào các feature của tập dữ liệu huấn luyện, cây quyết định đưa ra một loạt các câu hỏi để trả lời nhãn của dữ liệu như ví dụ trên. Mặc dù ví dụ minh họa dựa trên các biến rời rạc nhưng trong thực tế thì dữ liệu liên tục vẫn có thể được chấp nhận, giống như tập dữ liệu Iris. Ví dụ, chúng ta có thể cắt đoạn dữ liệu về sepal width (chiều rộng đài hoa) để trả lời cho câu hỏi nhị phân như: Chiều rộng đài hoa >= 2.8 cm?

Dùng thuật toán cây quyết định, chúng ta bắt đầu từ gốc (root) và phân tách dữ liệu dựa trên mỗi feature có kết quả lớn nhất của **Information Gain(IG)**. Quá trình này lặp đi lặp lại cho đến khi các node con là các node lá thuần khiết. Trong thực tế, điều này có thể dẫn đến một cây rất sâu với nhiều nút, dễ dẫn đến tình trạng thừa. Vì vậy chúng ta thường muốn cắt tỉa cây bằng cách đặt giới hạn độ sâu cho cây.

2 Information Gain

Để phân tách các node với nhiều thông tin nhất. Chúng ta cần định nghĩa một hàm mục tiêu mà chúng ta muốn tối ưu hóa thông qua thuật toán học cây. Ở đây, hàm mục tiêu của chúng ta dùng để tối đa hóa Information Gain mỗi lần tách. Hàm mục tiêu có thể được định nghĩa như sau:

$$IG(D_p, f) = I(D_p) - \sum_{j=1}^{v} \frac{N_j}{N_p} I(D_j)$$

Trong đó: f: là feature để thực hiện phân chia. v: số thuộc tính rời rạc của f D_p và D_j: là tập dữ liệu của cha và node con thứ j. I: là thước đo độ tạp chất (**impurity**). N_p: là tổng số lượng mẫu training ở node cha. N_j: là tổng số lượng mẫu training ở node con thứ j.

Information Gain đơn giản là độ sai khác giữa độ tạp chất của node cha so với tổng độ tạp chất của các node con. Giảm độ tạp chất của các node con thì Information Gain sẽ càng lớn. Tuy nhiên để đơn giản và để giảm không gian tìm kiếm nên hầu hết các thư viện (bao gồm cả scikit-learn) đều thực hiện bằng cây quyết định nhị phân. Có nghĩa là mỗi node cha được chia thành 2 node con.

$$IG(D_p, f) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$$

3 3 cách để đo độ tạp chất thường dùng

Ba cách để đo độ tạp chất thường dùng trong cây nhị phân là: Gini impurity (I_G), entropy (I_H) và classification error (I_E). Chúng ta bắt đầu với định nghĩa của entropy cho tất cả non-empty class ($p(i|t) \neq 0$).

3.1 Entropy

$$I_H(t) = -\sum_{i=1}^{c} p(i|t) log_2 p(i|t)$$

Trong đó: $\mathbf{p(i|t)}$: là xác xuất mẫu thuộc về class thứ i của node t. Entropy sẽ là 0 nếu tất cả các mẫu thuộc cùng 1 class. c: số class trong dataset

3.2 Gini

Gini impurity có thể được hiểu như là tiêu chuẩn để giảm xác xuất phân loại sai. Công thức gini impurity được định nghĩa như sau:

$$I_G(t) = \sum_{i=1}^{c} p(i|t)(1 - p(i|t)) = 1 - \sum_{i=1}^{c} p(i|t)^2$$

Tuy nhiên, trong thực tế cả gini impurity và entropy đều mang lại kết quả rất giống nhau. Và không đáng để giành nhiều thời gian để đánh giá cây thay vì thử nghiệm các cách cắt tỉa khác nhau.

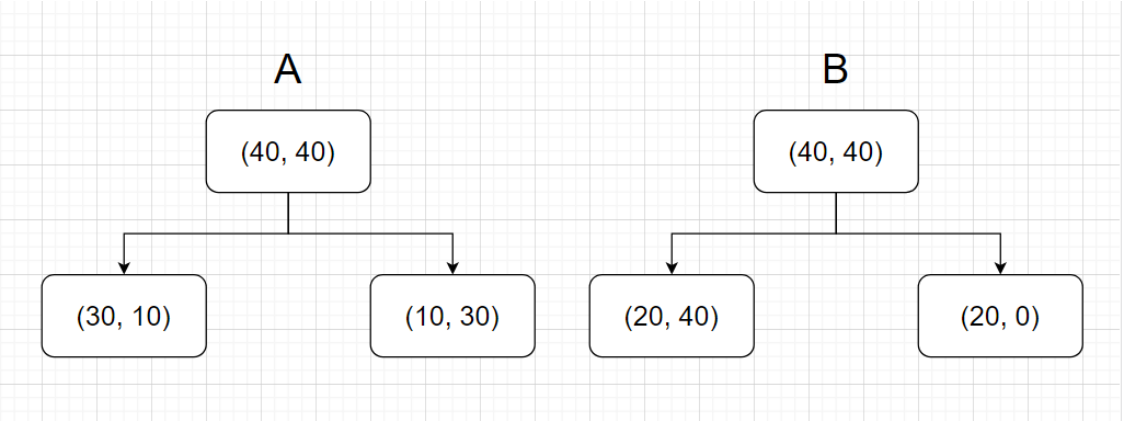
3.3 Classification error:

$$I_E(t) = 1 - max\{p(i|t)\}$$

Đây là một tiêu chí để cắt tỉa tuy nhiên không khuyến khích để phát triển cây quyết định. Vì nó ít nhạy cảm hơn với sự thay đổi xác suất class của các node.

3.4 Ví dụ 1

Cho 2 kịch bản xây decision tree sau và chúng ta sẽ xem xét việc áp dụng các độ đo impurity



Chúng ta bắt đầu với tập dữ liệu D_p ở node cha, bao gồm 40 mẫu ở class 1 và 40 mẫu ở class 2. Từ tập dữ liệu này chúng ta phân tách thành 2 tập dữ liệu con là D_{left} và D_{right}. Ở đây chúng ta sẽ dùng độ đo classification error:

$$I_E(D_p) = 1 - 0.5 = 0.5$$

$$A: \quad I_E(D_{left}) = 1 - \frac{3}{4} = 0.25$$

$$A: \quad I_E(D_{Right}) = 1 - \frac{3}{4} = 0.25$$

$$A: \quad IG_E = 0.5 - \frac{4}{8}0.25 - \frac{4}{8}0.25 = 0.25$$

$$B: \quad I_E(D_{left}) = 1 - \frac{4}{6} = \frac{1}{3}$$

$$B: \quad I_E(D_{Right}) = 1 - 1 = 0$$

$$B: \quad IG_E = 0.5 - \frac{6}{8} \times \frac{1}{3} - 0 = 0.25$$

Khi dùng classification error thì information gain của cả 2 A và B đều bằng nhau là 0.25. Dùng độ đo gini impurity:

$$I_G(D_p) = 1 - (0.5^2 + 0.5^2) = 0.5$$

$$A: \quad I_G(D_{left}) = 1 - \left(\left(\frac{3}{4}\right)^2 + \left(\frac{1}{4}\right)^2\right) = \frac{3}{8} = 0.375$$

$$A: \quad I_G(D_{right}) = 1 - \left(\left(\frac{1}{4}\right)^2 + \left(\frac{3}{4}\right)^2\right) = \frac{3}{8} = 0.375$$

$$A: \quad IG_G = 0.5 - \frac{4}{8}0.375 - \frac{4}{8}0.375 = 0.125$$

$$B: \quad I_G(D_{left}) = 1 - \left(\left(\frac{2}{6}\right)^2 + \left(\frac{4}{6}\right)^2\right) = \frac{4}{9} = 0.\overline{4}$$

$$B: \quad I_G(D_{right}) = 1 - (1^2 + 0^2) = 0$$

$$B: \quad IG_G = 0.5 - \frac{6}{8}0.\overline{4} - \ 0 = 0.\overline{16}$$

Tuy nhiên khi dùng độ đo gini impurity sẽ ủng hộ B ($IG_G = 0.1\bar{6}$) hơn A ($IG_G = 0.125$). Tương tự entropy cũng ủng hộ B hơn A:

$$I_H(D_p) = -(0.5\ log_2(0.5) + 0.5\ log_2(0.5)) = 1$$

$$A: I_H(D_{left}) = -\left(\frac{3}{4}\ log_2\left(\frac{3}{4}\right) + \frac{1}{4}\ log_2\left(\frac{1}{4}\right)\right) = 0.81$$

$$A: I_H(D_{right}) = -\left(\frac{1}{4}\ log_2\left(\frac{1}{4}\right) + \frac{3}{4}\ log_2\left(\frac{3}{4}\right)\right) = 0.81$$

$$A: IG_H = 1 - \frac{4}{8}0.81 - \frac{4}{8}0.81 = 0.19$$

$$B: I_H(D_{left}) = -\left(\frac{2}{6}\ log_2\left(\frac{2}{6}\right) + \frac{4}{6}\ log_2\left(\frac{4}{6}\right)\right) = 0.92$$

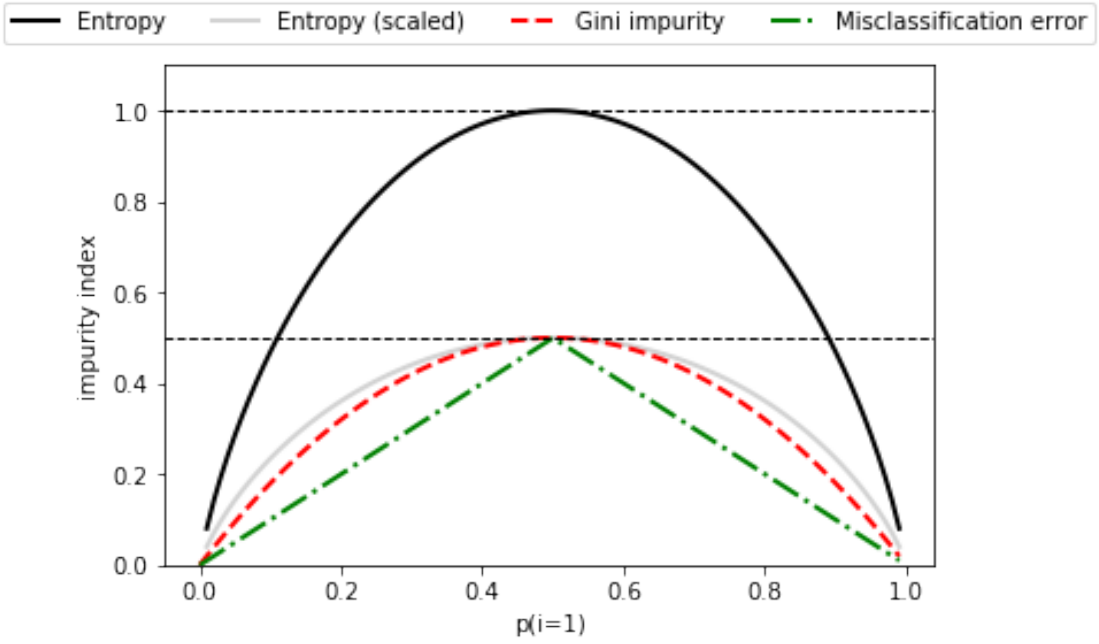
$$B: I_H(D_{right}) = 0$$

$$B: IG_H = 1 - \frac{6}{8}0.92 - 0 = 0.31$$

Để so sánh 3 tiêu chí độ đo ta dùng 1 biểu đồ biểu diễn phạm vị xác suất [0,1] cho class 1. Lưu ý rằng ở đây sẽ thêm một phiên bản thu nhỏ của entropy (entropy / 2) để quan sát tạp chất gini là thước đo trung gian giữa entropy và lỗi phân loại. Mã này như sau:

[66]:
```
import matplotlib.pyplot as plt
import numpy as np
def gini(p):
    return (p)*(1 - (p)) + (1 - p)*(1 - (1-p))
def entropy(p):
    return - p*np.log2(p) - (1 - p)*np.log2((1 - p))
def error(p):
    return 1 - np.max([p, 1 - p])
x = np.arange(0.0, 1.0, 0.01)
ent = [entropy(p) if p != 0 else None for p in x]
sc_ent = [e*0.5 if e else None for e in ent]
err = [error(i) for i in x]
```

```
fig = plt.figure()
ax = plt.subplot(111)
for i, lab, ls, c, in zip([ent, sc_ent, gini(x), err], ['Entropy', 'Entropy␣
 ↪(scaled)', 'Gini impurity', 'Misclassification error'],
                          ['-', '-', '--', '-.'], ['black', 'lightgray', 'red',␣
 ↪'green', 'cyan']):
    line = ax.plot(x, i, label=lab, linestyle=ls, lw=2, color=c)
ax.legend(loc='upper center', bbox_to_anchor=(0.5, 1.15), ncol=5,␣
 ↪fancybox=True, shadow=False)
ax.axhline(y=0.5, linewidth=1, color='k', linestyle='--')
ax.axhline(y=1.0, linewidth=1, color='k', linestyle='--')
plt.ylim([0, 1.1])
plt.xlabel('p(i=1)')
plt.ylabel('impurity index')
plt.show()
```



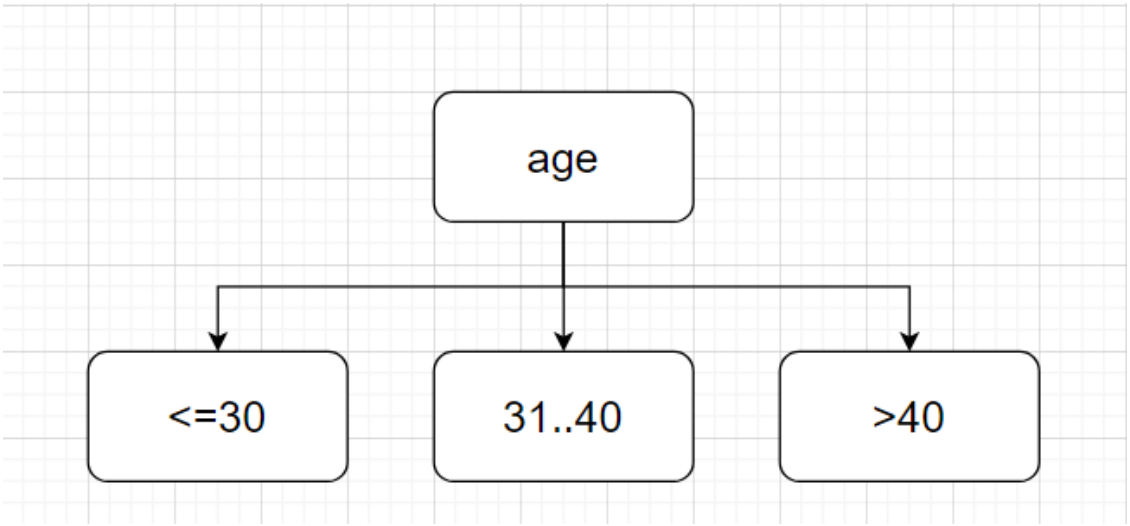
3.5 Ví dụ 2

Cho bảng dữ liệu ban đầu:

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31..40	high	no	fair	yes

age	income	student	credit_rating	buys_computer
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31..40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31..40	medium	no	excellent	yes
31..40	high	yes	fair	yes
>40	medium	no	excellent	no

Ta bắt đầu với feature age: gồm 3 nhánh con là <=30, 31..40, >40



Nhánh <=30:

income	student	credit_rating	buys_computer
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

Độ đo gini của nhánh <=30:

income	yes	no	gini(yes,no)
high	0	2	0
medium	1	1	0.5

income	yes	no	gini(yes,no)
low	1	0	0

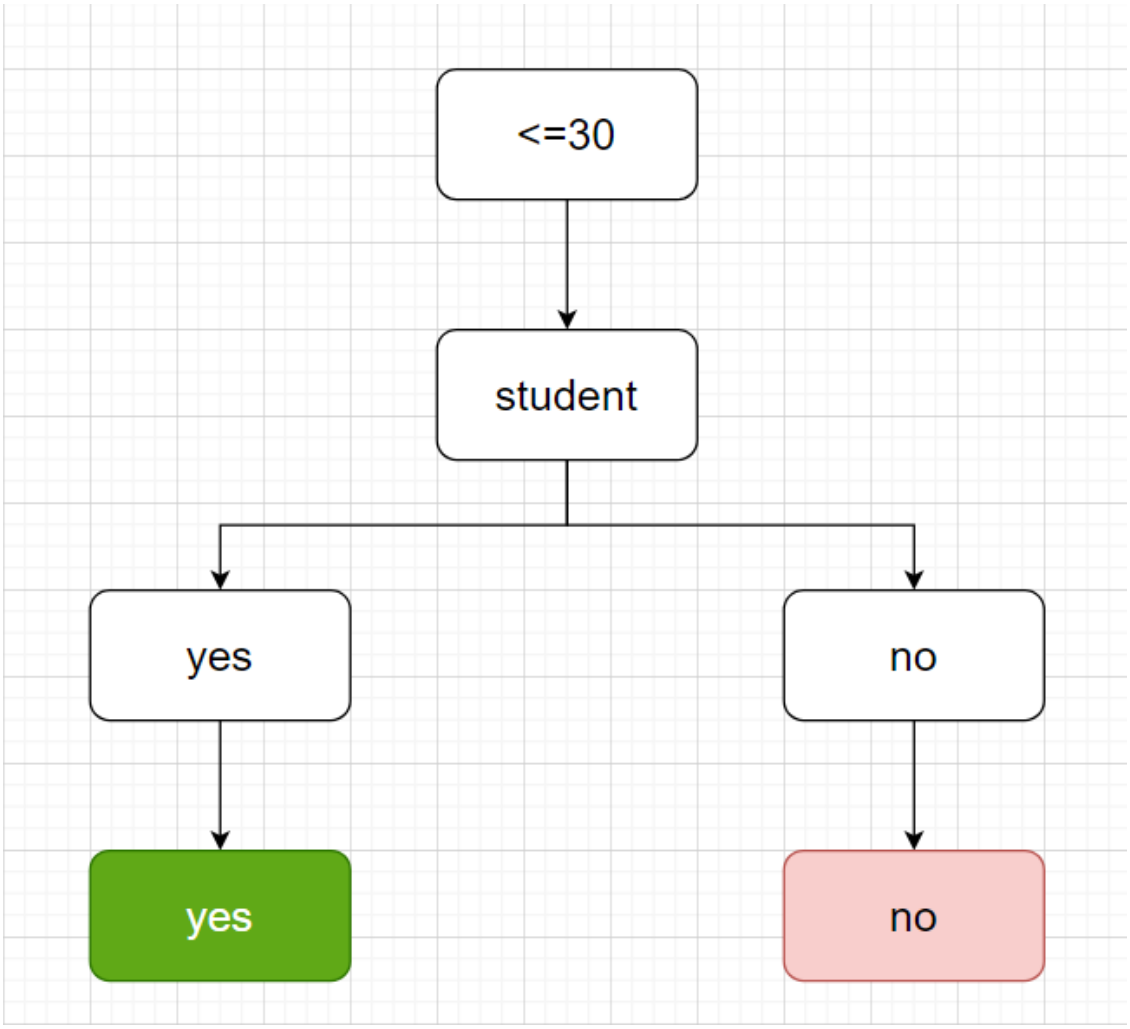
$=> IG_{income} = \frac{2}{5}gini(0,2) + \frac{2}{5}gini(1,1) + \frac{1}{5}gini(1,0) = \frac{1}{5} = 0.2$

Tương tự như trên ta có:

$IG_{student} = \frac{2}{5}gini(2,0) + \frac{3}{5}gini(0,3) = 0.72$ $IG_{credit\_rating} = \frac{3}{5}gini(1,2) + \frac{2}{5}gini(1,1) = 0.47$

Lấy max $IG_{income}, IG_{student}, IG_{credit\_rating}$ ta được $IG_{income} = 0.72$

Từ trên ta có được nhánh của <= 30 là:

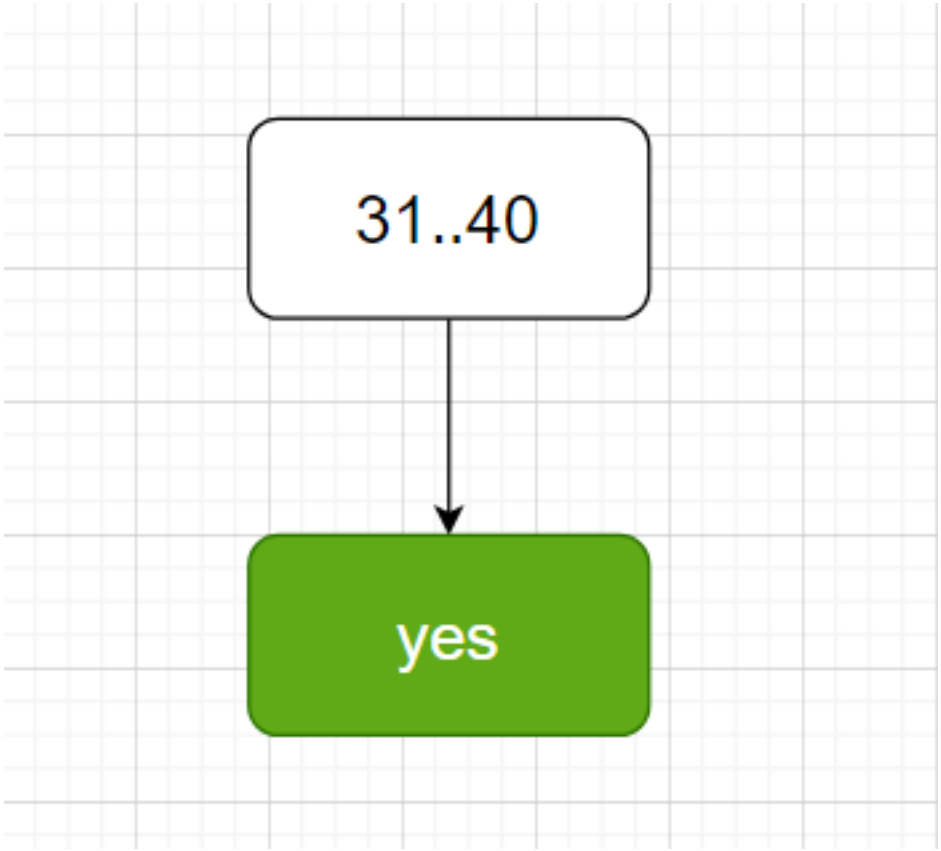


Nhánh 31..40:

income	student	credit_rating	buys_computer
high	no	fair	yes
low	yes	excellent	yes

income	student	credit_rating	buys_computer
medium	no	excellent	yes
high	yes	fair	yes

Do dữ liệu của buys_computer luôn luôn là yes nên ta có nhánh của 31..40 là:



Nhánh >40:

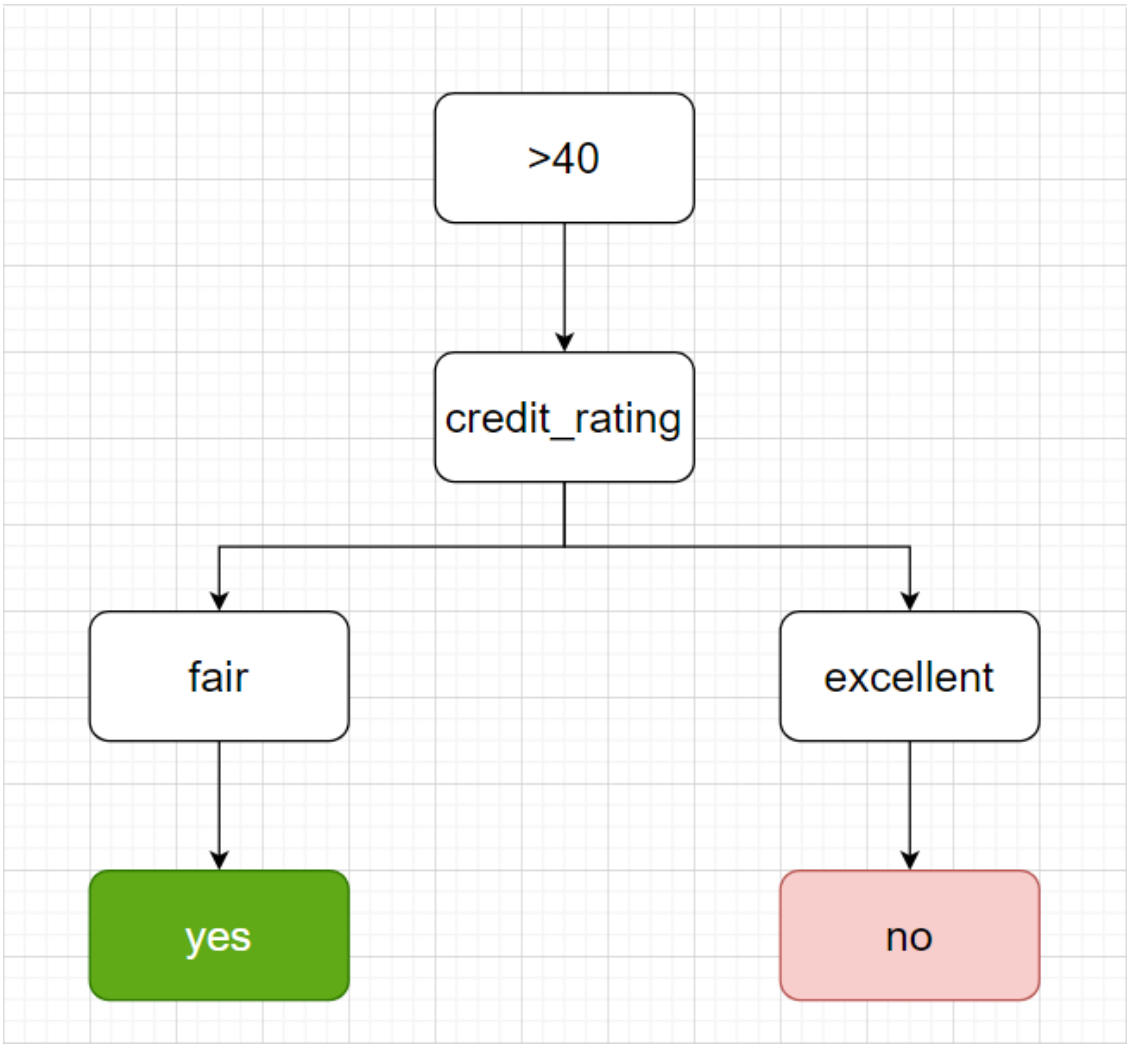
income	student	credit_rating	buys_computer
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

Tương tự như nhánh <=30 ta có:

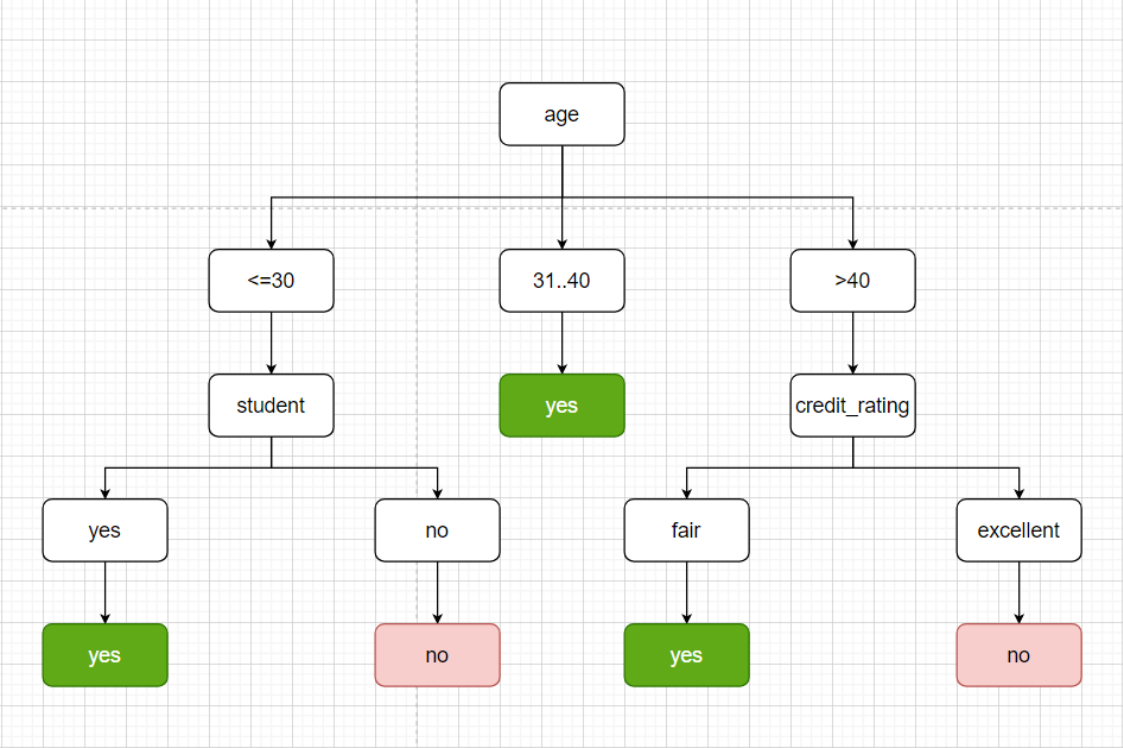
$IG_{income} = \frac{3}{5}gini(2,1) + \frac{2}{5}gini(1,1) = 0.47$ $IG_{student} = \frac{3}{5}gini(2,1) + \frac{2}{5}gini(1,1) = 0.47$
$IG_{credit\_rating} = \frac{3}{5}gini(3,0) + \frac{2}{5}gini(0,2) = 0.72$

Lấy max $IG_{income}, IG_{student}, IG_{credit\_rating}$ ta được $IG_{credit\_rating} = 0.72$

Từ trên ta có được nhánh của >40 là:



Gộp 3 nhánh <=30, 31..40, >40 lại ta được cây quyết định cuối cùng là:

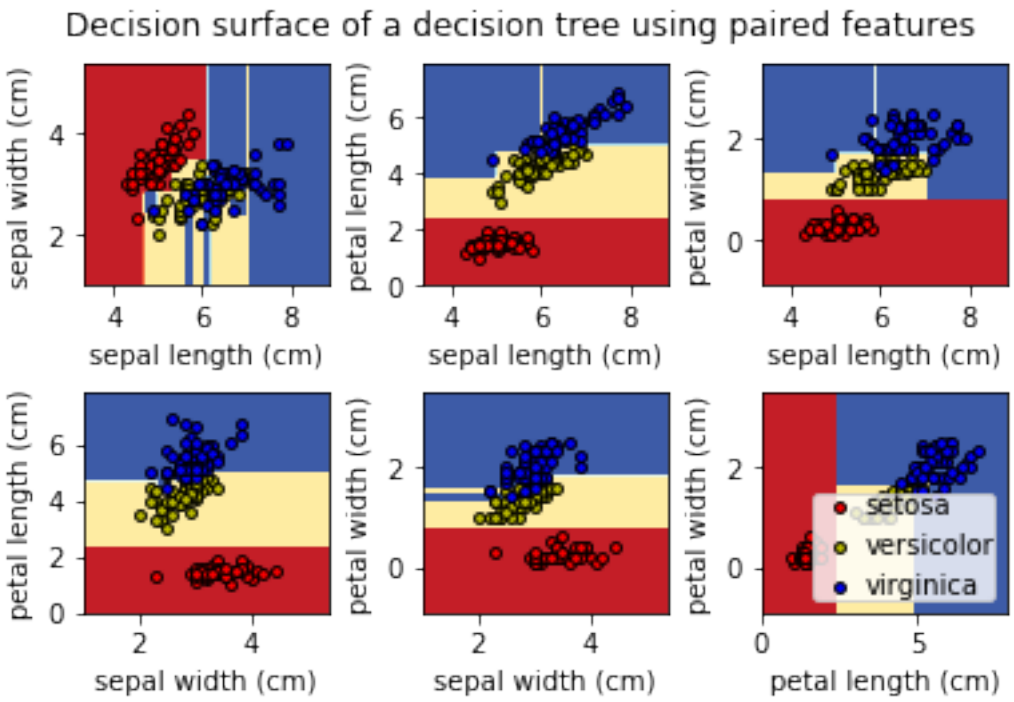


3.6 Ví dụ 3

Dùng dataset Iris, cho mỗi cặp thuộc tính (6 cặp), học decision tree và show trên mặt phẳng 2 chiều các boundaries (vì dùng hết 4 chiều thì khó show)

[67]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree

# Parameters
n_classes = 3
plot_colors = "ryb"
plot_step = 0.02

# Load data
iris = load_iris()

for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3],
                                [1, 2], [1, 3], [2, 3]]):
    # We only take the two corresponding features
    X = iris.data[:, pair]
    y = iris.target
```

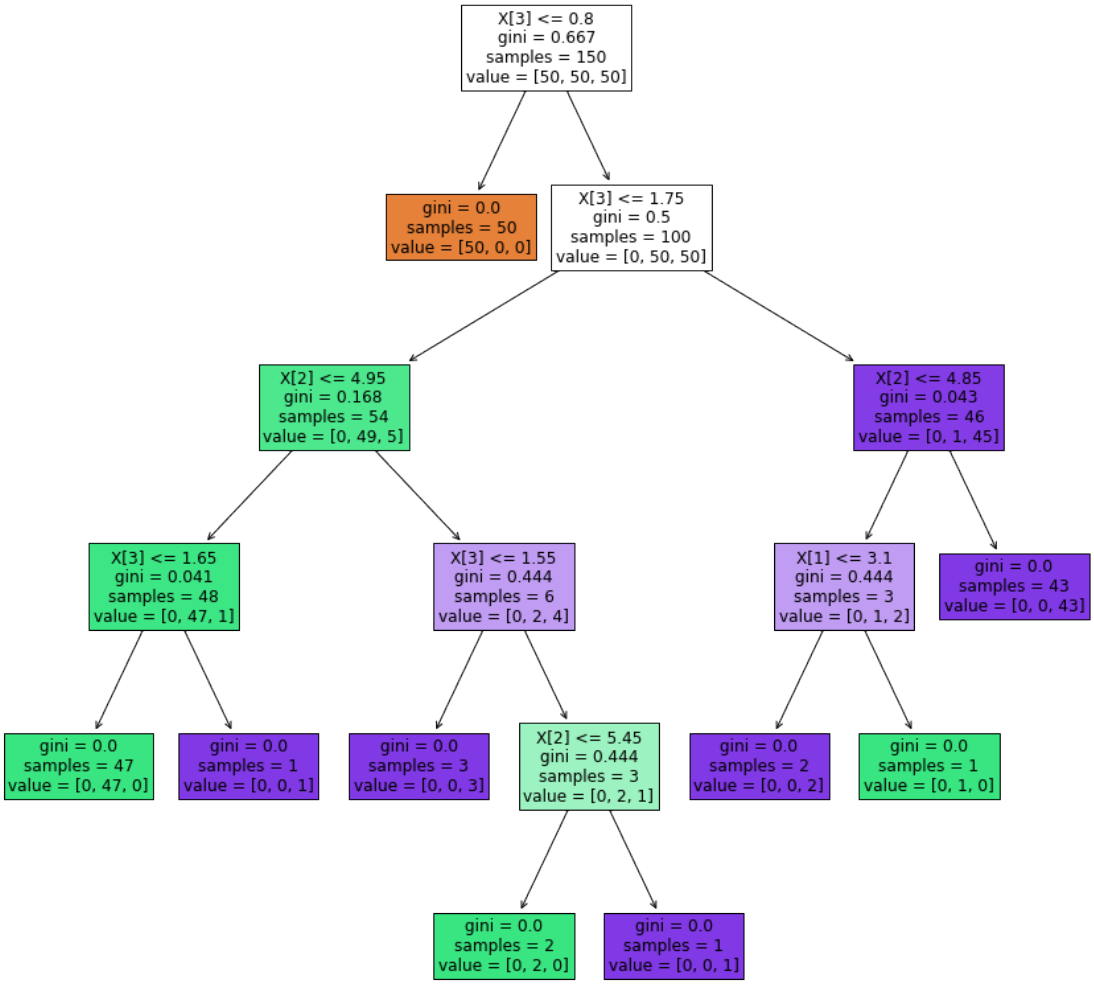
```
    # Train
    clf = DecisionTreeClassifier().fit(X, y)

    # Plot the decision boundary
    plt.subplot(2, 3, pairidx + 1)

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                         np.arange(y_min, y_max, plot_step))
    plt.tight_layout(h_pad=0.5, w_pad=0.5, pad=2.5)

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    cs = plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)

    plt.xlabel(iris.feature_names[pair[0]])
    plt.ylabel(iris.feature_names[pair[1]])

    # Plot the training points
    for i, color in zip(range(n_classes), plot_colors):
        idx = np.where(y == i)
        plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i],
                    cmap=plt.cm.RdYlBu, edgecolor='black', s=15)

plt.suptitle("Decision surface of a decision tree using paired features")
plt.legend(loc='lower right', borderpad=0, handletextpad=0)
plt.axis("tight")
plt.show()
```



Dùng lib để tạo decision tree từ toàn bộ 4 chiều của Iris và show

[68]:
```
plt.figure()
plt.figure(figsize=(15,15))
clf = DecisionTreeClassifier().fit(iris.data, iris.target)
plot_tree(clf, filled=True)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



4 Phối hợp đa cây quyết định với random forests

4.1 Overview

Random Forest có thể coi như một tập hợp của cây quyết định. Ý tưởng đằng sau là lấy trung bình nhiều cây quyết định mà mỗi cây đều mang tính high variance. Nó có thể xây dựng một model tổng quát hơn, ít overfitting hơn. Random Forest dc ví như là Black box, nó dự báo tốt nhưng rất khó giải thích cơ chế thực sự đằng sau

Chọn mẫu có thay thế và ko thay thế: Giả sử bạn có một hộp chứa các viên bi đánh số 1 tới 5 1. Chọn mẫu có thay thế: chọn xong bi lại bỏ nó vào, chọn tiếp: 1,3,3,4,1 2. Chọn mẫu ko thay thế: chọn xong bi ko dc bỏ nó vào để tránh chọn lại nó: 1,3,2,5,4

4.2 Các bước thực hiện

1. Rút ra 1 tập con của training set (bootstrap sample) có n điểm dữ liệu dc chọn ngẫu nhiên (chọn mẫu có thay thế)
2. Từ tập boostrap sample tạo decision tree: mỗi node ngẫu nhiên chọn d features (chọn mẫu ko thay thế). Chia node này ra các nhánh dựa trên feature nào tối ưu hàm mục tiêu nhất (VD: có info gain cao nhất,...)
3. Mỗi lần bạn dùng bước 1,2 là tạo ra 1 cây, hãy tạo đủ k cây
4. Kết hợp kết quả dự đoán từ các cây này, dùng majority vote cho ra kết quả cuối cùng.

4.3 Một vài lưu ý

Lợi thế lớn của Random Forest là bạn ko phải lo nhiều về việc chọn hyperparam thật tốt. Bạn chỉ cần quan tâm tới số cây trong rừng (k), thường là càng nhiều cây càng tốt cho việc predict, nhưng đổi lại là bạn phải tính nhiều cây (chi phí tính tăng)

Một bootstrap sample xây 1 decision tree: - Nếu quá ít điểm dữ liệu (n nhỏ) thì các tree sẽ rất khác nhau. Nó tăng tính ngẫu nhiên (randomness), giúp chống overfitting. Tuy vậy, ít quá thì thường dẫn tới chuyện model ko dc tổng quát lắm - Điều ngược lại xảy ra cho việc n lớn. - Đa số các trường hợp, n được cho là bằng với số records trong training set, thường thì nó cho ta một sự đánh đổi hợp lí giữa bias-variance. - Về lượng features d, mỗi lần phân chia, ta muốn chọn một số nhỏ hơn số lượng feature sẵn có trong training set, scikit-learn mặc định d = $\sqrt{m}$

4.4 Code example

[69]:
```
from sklearn import datasets

#Load dataset
iris = datasets.load_iris()
```

[70]:
```
# Creating a DataFrame of given iris dataset.
import pandas as pd
data=pd.DataFrame({
    'sepal length':iris.data[:,0],
    'sepal width':iris.data[:,1],
    'petal length':iris.data[:,2],
    'petal width':iris.data[:,3],
    'species':iris.target
})
data.head()
```

[70]:

	sepal length	sepal width	petal length	petal width	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0

```
4           5.0          3.6           1.4          0.2         0
```

[71]:
```
# Import train_test_split function
from sklearn.model_selection import train_test_split

X=data[['sepal length', 'sepal width', 'petal length', 'petal width']]  #␣
 ↪Features
y=data['species']  # Labels

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70%␣
 ↪training and 30% test
```

[72]:
```
#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
```

[73]:
```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 1.0
```

[74]:
```
clf.predict([[3, 5, 4, 2]])
```

[74]:
```
array([1])
```

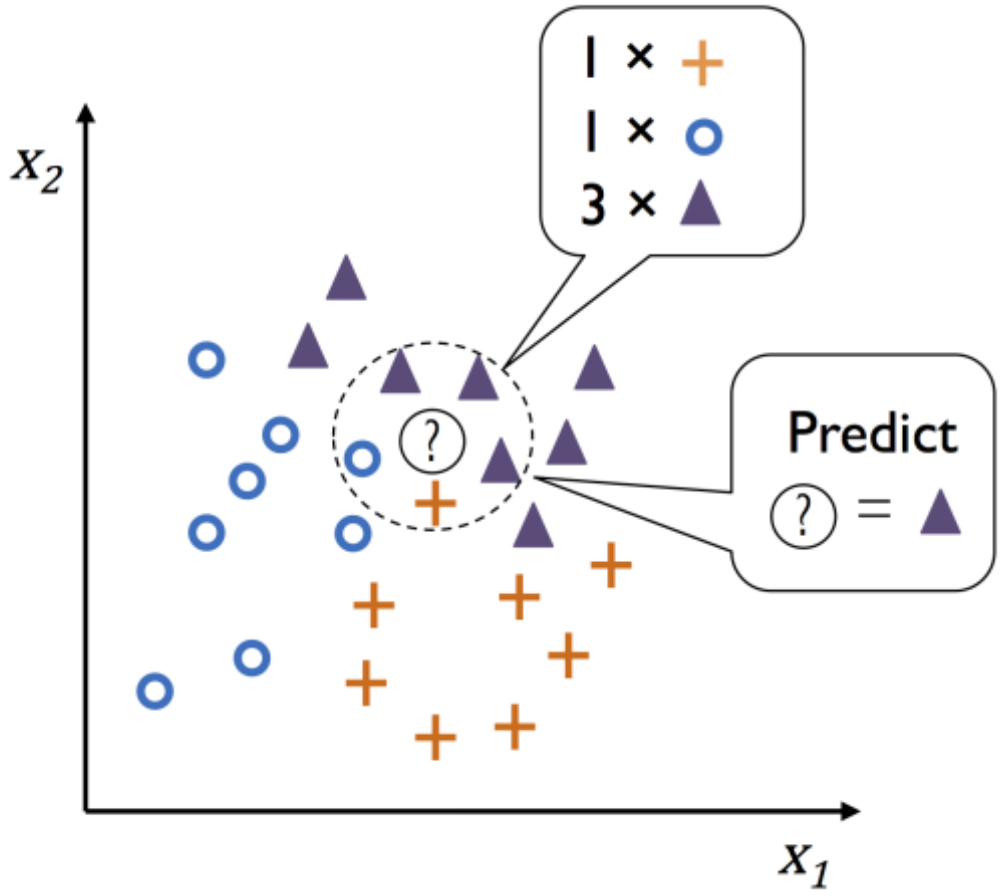
(Virginia based on 100 trees)

5 K-nearest neighbors (KNN)

5.1 Overview

KNN là 1 điển hình của **lazy learner**. Vì khi training, thuật toán này không học một điều gì từ dữ liệu training.

Thuật toán **KNN** tự nó khá đơn giản và có thể được tóm tắt bởi các bước sau: 1. Chọn số **K** và **độ đo khoảng cách**. 2. Tìm **K điểm hàng xóm gần nhất** với điểm dữ liệu muốn phân loại. 3. Gán nhãn cho điểm dữ liệu theo **số phiếu nhiều nhất**



Dựa trên **độ đo khoảng cách** đã chọn, thuật toán KNN tìm **K** điểm trong tập huấn luyện có gần nhất (giống nhất) với điểm muốn phân loại. Nhãn của điểm dữ liệu này sẽ dựa trên số phiếu nhiều nhất của K điểm gần nhất

5.2 Some of distance metric used in KNN

identifier	class name	args	distance function
“euclidean”	EuclideanDistance	*	sqrt(sum((x - y)^2))
“manhattan”	ManhattanDistance	*	sum(|x - y|)
“chebyshev”	ChebyshevDistance	*	max(|x - y|)
“minkowski”	MinkowskiDistance	p	sum(|x - y|^p)(1/p)
“wminkowski”	WMinkowskiDistance	p, w	sum(|w * (x - y)|^p)(1/p))
“seuclidean”	SEuclideanDistance	V	sqrt(sum((x - y)^2 / V))
“mahalanobis”	MahalanobisDistance	V or VI	sqrt((x - y)’ V^-1 (x - y))

5.3 Ưu điểm

- Ưu điểm chính của **cách tiếp cận từ bộ nhớ** là khi thêm dữ liệu training mới thì việc phân loại sẽ thích ứng ngay (Độ phức tạp của thuật toán khi traning bằng **0**)
- Không cần giả sử gì về phân phối của các class.

5.4 Nhược điểm

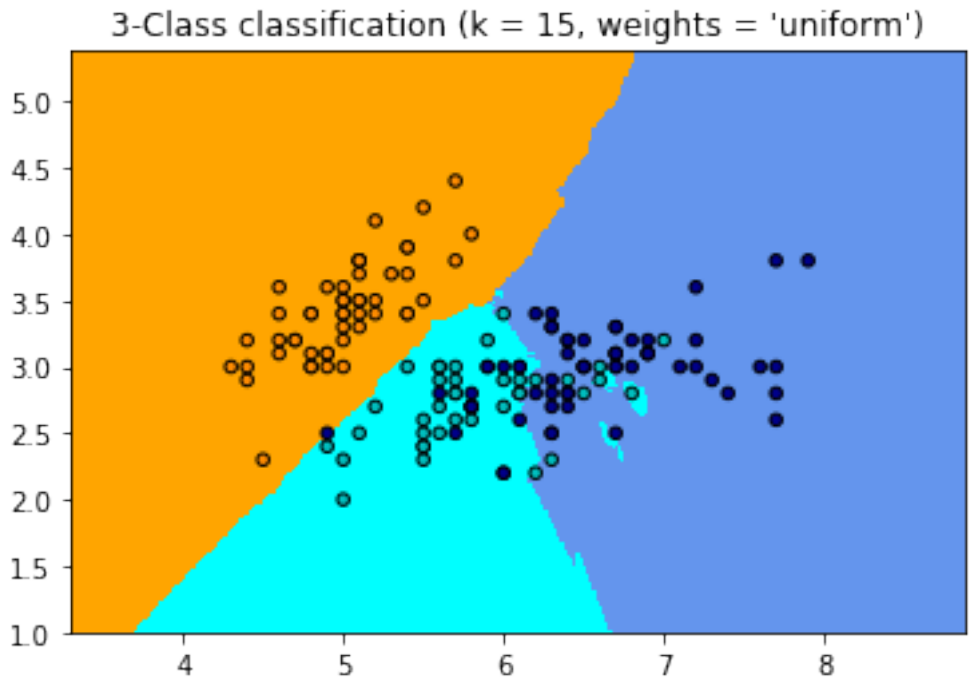
- KNN rất nhạy cảm với nhiễu khi K nhỏ.
- KNN là một thuật toán mà mọi tính toán đều nằm ở khâu test. Trong đó việc tính khoảng cách tới từng điểm dữ liệu trong training set sẽ tốn rất nhiều thời gian, đặc biệt là với các cơ sở dữ liệu có số chiều lớn và có nhiều điểm dữ liệu. Với **K càng lớn** thì độ phức tạp cũng sẽ **tăng lên**. Ngoài ra, việc lưu toàn bộ dữ liệu trong bộ nhớ cũng ảnh hưởng tới hiệu năng của KNN.

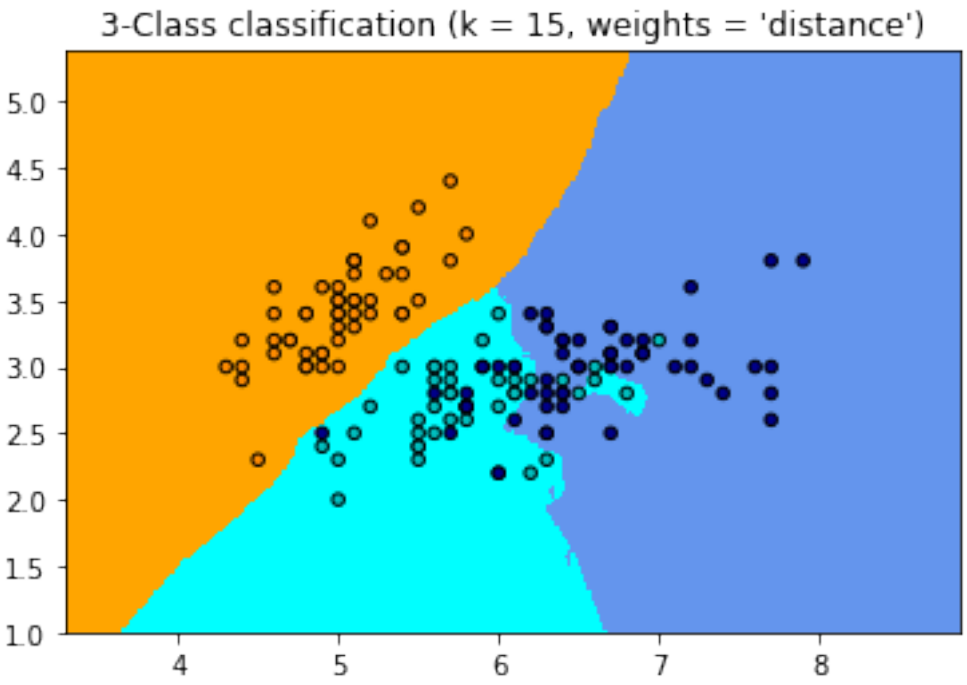
5.5 Code example

Và bây giờ, ta lại gọi thư viện để hiện thực KNN, ở đây thì weight dc chọn là uniform và distance: uniform: tất cả các neighbor khi vote thì quan trọng như nhau distance: các neighbor khi vote thì weight sẽ là nghịch đảo khoảng cách, neighbor gần hơn thì sẽ vote nặng hơn

[75]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets

n_neighbors = 15

# import some data to play with
iris = datasets.load_iris()

# we only take the first two features. We could avoid this ugly
# slicing by using a two-dim dataset
X = iris.data[:, :2]
y = iris.target

h = .02  # step size in the mesh

# Create color maps
cmap_light = ListedColormap(['orange', 'cyan', 'cornflowerblue'])
cmap_bold = ListedColormap(['darkorange', 'c', 'darkblue'])

for weights in ['uniform', 'distance']:
    # we create an instance of Neighbours Classifier and fit the data.
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(X, y)
```

```
    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold,
                edgecolor='k', s=20)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title("3-Class classification (k = %i, weights = '%s')"
              % (n_neighbors, weights))

plt.show()
```




3-Class classification (k = 15, weights = 'distance')
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8