


# Data Mining 2020 HW2 - Decision Tree

---

—  P76094321 盧宥霖

Report HackMD link (<https://hackmd.io/@wMsNOR2hQmKOfM82HEL4mw/SJwHltMqw>).

## Description

---

Construct a classification model to observe the difference between real 'right' data and modeled data

## Generating Data

---

### Motivation

The Formosan black bear (臺灣黑熊, *Ursus thibetanus formosanus*) is a subspecies of the Asiatic black bear. It is one of the most representative animals of Taiwan. However, the populations of wild Formosan black bears have been declining, and is now considered "endangered". Here, I would like to construct a classifier to predict whether a certain bear can still survive in a month.

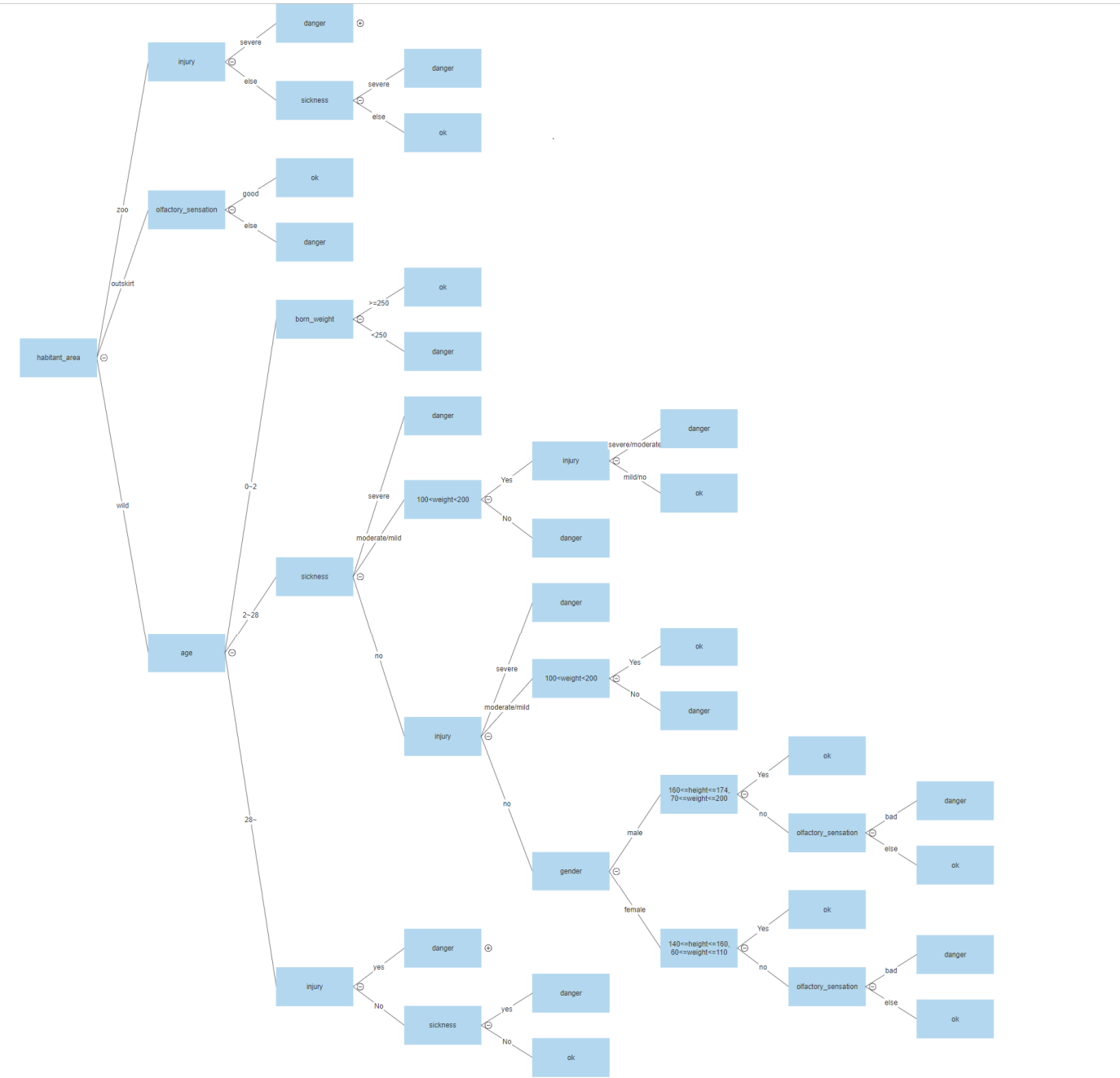
### Features

Feature name	Possible Values	Description
Age	0~35	Age of the bear. Generated randomly
gender	male/female	Gender of the bear.
height	positive number	Height of the bear. Generated with normal distribution.
weight	positive number	Weight of the bear. Generated with normal distribution.
habitant area	zoo/outskirts/wild	Where the bear lives.
habitant height	positive number	Height of the bear's habitant. Generated with normal distribution.
born weight	positive number	Weight when the bear was born. Generated with normal distribution.
claw size	positive number	Length of the bear's claw. Generated with normal distribution.
injury	no/mild/moderate/severe	Whether the bear is injured.
sickness	no/mild/moderate/severe	Whether the bear is is suffering illness.
olfactory sensation	bad/normal/good	Ability of the bear to sense the surroundings.
tail length	positive number	Generated with normal distribution.
ear size	positive number	Generated with normal distribution.
head size	positive number	Generated with normal distribution.
front paw	positive number	Size of the bear's front paw. Generated with normal distribution.
rear paw	positive number	Size of the bear's rear paw. Generated with normal distribution.
fur length	positive number	Generated with normal distribution.
fur color	dark/light	The color of the fur. Generated randomly.
shoulder width	positive number	Generated with normal distribution.

Feature name	Possible Values	Description
food tend	meat/balanced/fruit	Whether the bear prefers eating fruits or meat. Generated randomly.

The features were designed referencing data from [台灣黑熊保育協會\(TBBCA\)](https://www.taiwanbear.org.tw/).  
(<https://www.taiwanbear.org.tw/>), and [Wikipedia](https://zh.wikipedia.org/wiki/%E5%8F%B0%E7%81%A3%E9%BB%91%E7%86%8A).  
(<https://zh.wikipedia.org/wiki/%E5%8F%B0%E7%81%A3%E9%BB%91%E7%86%8A>).

## Designed Absolutely Correct Rules



## Automatic data generation

After designing the rules to tell whether a bear can survive, I used the following program to simulate data. Some of the features were generated randomly, while most of them were sampled from a certain normal distribution, centered at the average value in real world data.

```
class Bear():
    def __init__(self):
        self.age = random.randint(0, 35)
        self.gender = features.gender[random.randint(0, 2)]
        self.height = random.normal(loc=160, scale=7)
        if self.gender=='female':
            self.weight = random.normal(loc=85, scale=12.5)
        else:
            self.weight = random.normal(loc=135, scale=32.5)
        self.habitant_area = features.habitant_area[random.randint(0, 3)]
        self.habitant_height = random.normal(loc=2000, scale=800)
        self.born_weight = random.normal(loc=325, scale=38)
        self.claw_size = random.normal(loc=4, scale=1)
        self.injury = random.choice(features.injury, 1, p=[0.1, 0.2, 0.4, 0.3])
        self.sickness = random.choice(features.sickness, 1, p=[0.1, 0.2, 0.4, 0.3])
        tmp = random.normal(loc=5, scale=2)
        if tmp<1.5:
            self.olfactory_sensation = 'bad'
        elif tmp>8.5:
            self.olfactory_sensation = 'good'
        else:
            self.olfactory_sensation = 'average'
        self.tail_length = random.normal(loc=8, scale=1)
        self.ear_size = random.normal(loc=10, scale=1)
        self.head_size = random.normal(loc=50, scale=5)
        self.front_paw = random.normal(loc=12, scale=1.5)
        self.rear_paw = random.normal(loc=20, scale=1.5)
        self.fur_length = random.normal(loc=8, scale=2)
        self.fur_color = features.fur_color[random.randint(0, 2)]
        self.shoulder_width = random.normal(loc=65, scale=2)
        self.food_tend = features.food_tend[random.randint(0, 3)]
        self.generate_label()
```

After that, the generated "bear" will be labeled by the absolutely correct rules. It will be labeled "danger" if it has a high possibility to die within a month, "ok" otherwise. The following program is the if-else version of the designed rule.

```

if self.habitant_area=='zoo':
    if self.injury=='severe' or self.sickness=='severe':
        self.label = 'danger'
    else:
        self.label = 'ok'
elif self.habitant_area=='outskirts':
    if self.olfactory_sensation=='good':
        self.label = 'ok'
    else:
        self.label = 'danger'
else:
    if self.age<2:
        if self.born_weight<250:
            self.label = 'danger'
        else:
            self.label = 'ok'
    elif self.age>28:
        if self.injury=='no' or self.sickness=='no':
            self.label = 'ok'
        else:
            self.label = 'danger'
    else: # midage
        if self.sickness=='severe':
            self.label = 'danger'
        elif self.sickness=='moderate':
            if self.weight>100 and self.weight<200 and (self.injury=='mild' or
                self.label = 'ok'
            else:
                self.label = 'danger'
        else:
            if self.injury=='severe':
                self.label = 'danger'
            elif self.injury=='moderate':
                if self.weight>100 and self.weight<200:
                    self.label = 'ok'
                else:
                    self.label = 'danger'
            else:
                if self.gender=='male':
                    if 160>self.height and self.height>174 and 70>self.weight
                        self.label = 'danger'
                    else:
                        self.label = 'ok'
                else: #female
                    if 140>self.height and self.height>160 and 60>self.weight
                        self.label = 'danger'
                    else:
                        self.label = 'ok'

```

## Training classifier

---

Here, we would like to train a classifier. Given the features of a bear, it should predict whether the bear can survive within a month.

## Decision tree

First, we use the decision tree in sklearn. For testing, 1000 data were generated. 75% were separated into training set and the rest as test set. The implementation in python are shown below:

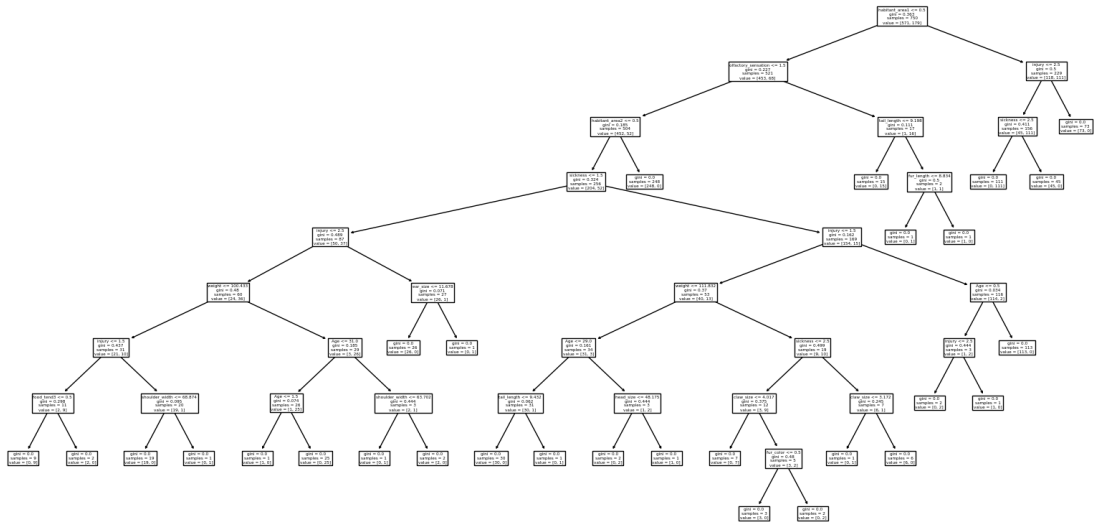
```
from bear_generator import BearGenerator
from bear import Bear
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import OneHotEncoder
from sklearn import tree
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

generator = BearGenerator()
dataset = generator.generate_dataset(1000)
X = dataset[0]
Y = dataset[1]

x_train, x_test, y_train, y_test = train_test_split(X,Y)

model = DecisionTreeClassifier().fit(x_train, y_train)
res = model.predict(x_test)
acc = accuracy_score(y_test, res)
print('Accuracy: ' + str(acc))
plt.figure()
tree.plot_tree(model)
plt.show()
```

This results in an accuracy of 0.968, with a tree of height 10.

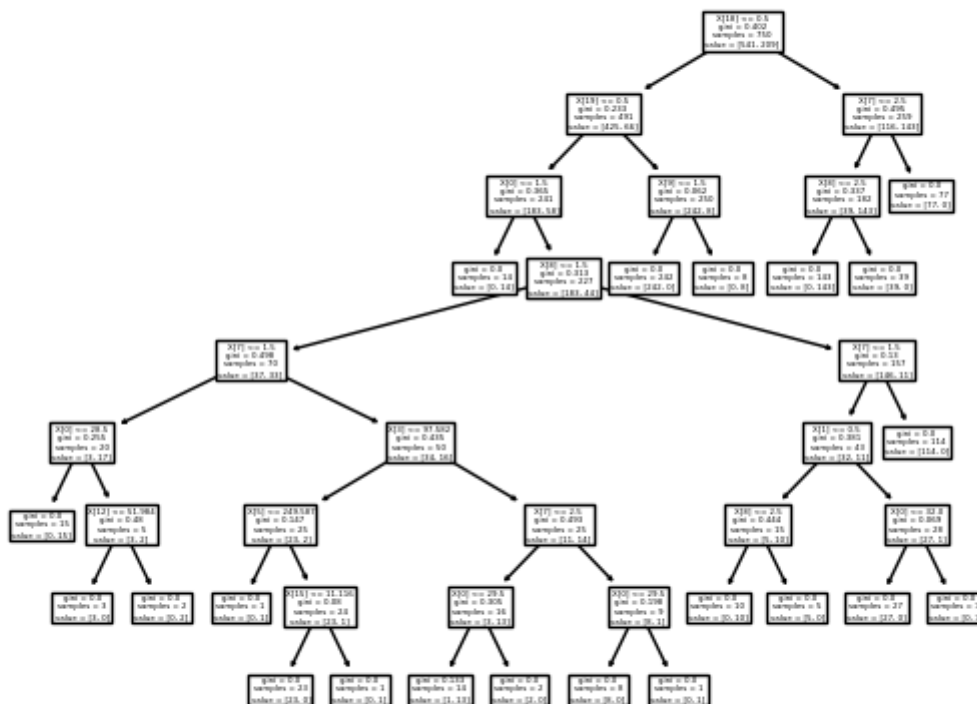


We can see that the results are slightly different than the designed rules, ending up with a deeper decision tree. The main rules, such as habitant area, injury, sickness, age, etc. were found, while the model automatically found some more rules such as shoulder width, food tendency, fur color, head size, which were not used in the designed rules, somehow helps fitting the data more accurately. This could be due to some of the features being slightly unbalanced during the randomized data generation process. Thus, the model could be overfitting the training set a bit and misclassify some data in the test set. However, 0.968 is a pretty good accuracy score and is often considered acceptable.

## Decision tree with max height limitation

Here, I tried limiting the tree's maximum height to avoid the overfitting problem. The tree looked more similar to the original rules, but somehow often lead to a bit of accuracy decrease. In my opinion, the model might not be able to perfectly learn the rules with only 1000 data, with each being a 20+ dimension vector. This causes the algorithm trying to fit

the dataset with deeper depth than its actually needed. Thus, limiting the maximum height could possibly cause underfitting and affect the accuracy on the test set.



## Decision tree with different data size

Here, I would like to increase the dataset size and see if the decision tree can really learn the rules with a huge amount of data. Besides, in real life scenario, getting such amount of data is often hard, so I would also want to test the model performance on a more reasonable data size.

Data size	accuracy	accuracy(max_height=8)
20000	1.0	0.9924
10000	0.9968	0.9888
5000	0.992	0.9784
1000	0.972	0.972
500	0.928	0.92
100	0.88	0.92
Realistic	0.5641025641025641	0.5897435897435898



We can see that decision tree model learns better with more data. When the data size is around 20000, the model can achieve 100% accuracy. Another interesting fact found is, with the dataset size small, limiting the maximum tree height can sometimes help, preventing the tree to overfit on the small amount of data which cannot reflect the whole dataset. Finally, I tried using this model on a more realistic scenario. According to the Taiwan Black Bear Conservation Association, the population of the Formosan black bear is only around 400, while they can hardly find more than 10 of them each time they investigate the mountains. This proportion is pretty unbalanced, and I would like to test if the decision tree model can still be accurate. Thus, I generated the data with a unbalanced training/test split of 10:390. This results in a pretty bad accuracy, showing that a small and skewed data is very hard for decision trees (and probably other ML models) to learn the rules.

## Other models

Now let's try doing the same task with different ML models.

```
datasize = 100
Model: Nearest Neighbors Accuracy: 0.44
Model: Linear SVM Accuracy: 0.72
Model: RBF SVM Accuracy: 0.72
Model: Gaussian Process Accuracy: 0.68
Model: Decision Tree Accuracy: 0.68
Model: Random Forest Accuracy: 0.8
Model: AdaBoost Accuracy: 0.8
Model: Naive Bayes Accuracy: 0.76
```

```
datasize = 500
Model: Nearest Neighbors Accuracy: 0.704
Model: Linear SVM Accuracy: 0.84
Model: RBF SVM Accuracy: 0.728
Model: Gaussian Process Accuracy: 0.704
Model: Decision Tree Accuracy: 0.984
Model: Random Forest Accuracy: 0.936
Model: AdaBoost Accuracy: 0.904
Model: Naive Bayes Accuracy: 0.872
```

```
datasize = 1000
Model: Nearest Neighbors Accuracy: 0.656
Model: Linear SVM Accuracy: 0.856
Model: RBF SVM Accuracy: 0.752
Model: Gaussian Process Accuracy: 0.624
Model: Decision Tree Accuracy: 0.952
Model: Random Forest Accuracy: 0.92
Model: AdaBoost Accuracy: 0.872
Model: Naive Bayes Accuracy: 0.848
```

realistic dataset

Model: Nearest Neighbors Accuracy: 0.7025641025641025

Model: Linear SVM Accuracy: 0.5897435897435898

Model: RBF SVM Accuracy: 0.7025641025641025

Model: Gaussian Process Accuracy: 0.6948717948717948

Model: Decision Tree Accuracy: 0.6717948717948717

Model: Random Forest Accuracy: 0.7102564102564103

Model: AdaBoost Accuracy: 0.6717948717948717

Model: Naive Bayes Accuracy: 0.7025641025641025

We can see that decision tree needs more data to perform well. With the small dataset where datasize = 100, models such as Naive Bayes and adaboost outperform other models. With the data size increasing, decision tree takes the lead and has the most accurate prediction. In the dataset simulating realistic scenario, the Random Forest algorithm tops others with an accuracy of 0.71.

## Decision tree with hidden data

In the project announcement slides, the TA mentioned about "Hidden rules", which are rules the model found but not in the designed rules. This inspired me with the following thought: what if there exists some rules used but not present to the model? In real life experience, there are often features being predictive but not recorded in the dataset for training. I would like to find out whether the existence of such rules affect our decision tree model. To do so, I slightly adjusted the designed rules. Previously, whether a cub(bears less than 2 years old) can survive depends only on the weight when they were born. Since cubs usually stay with its mother for 2 years, a new rule considering if their parents is still alive will be added. Here, we assume that if a cub's parents were dead, it cannot survive the next month.

```
if self.age<2:
    if self.parent_alive=='dead':
        self.label = 'danger'
    else:
        if self.born_weight<250:
            self.label = 'danger'
        else:
            self.label = 'ok'
```

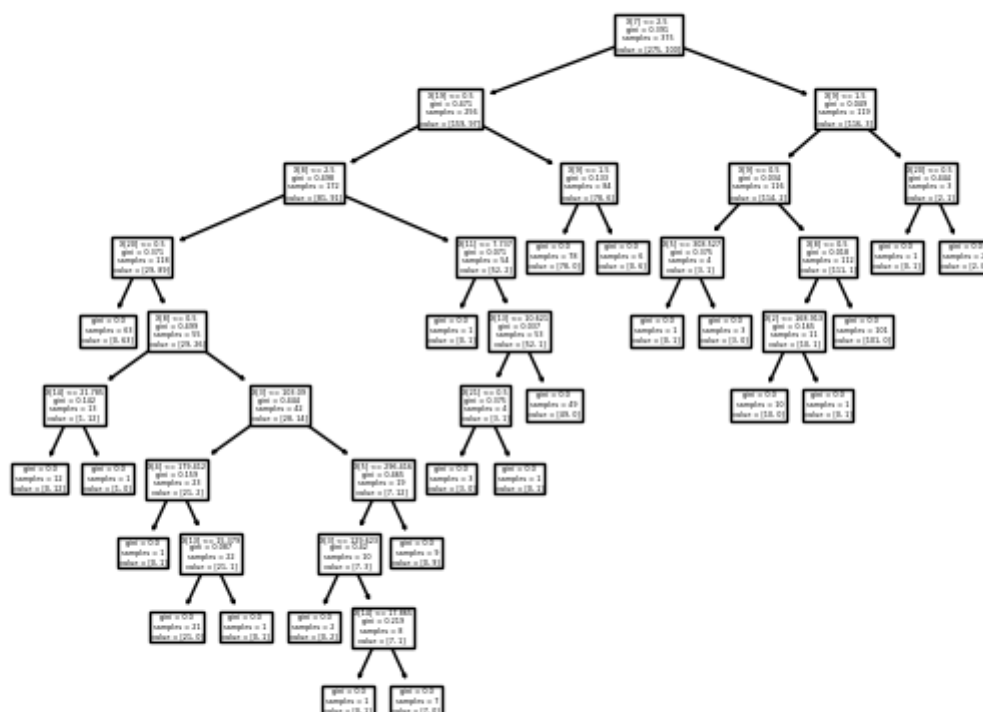
Then, we generated the labels using this rule. However, the `parent_alive` feature will not be shown to the ML model. The results are shown below.

```

datasize = 500
Model: Nearest Neighbors Accuracy: 0.616
Model: Linear SVM Accuracy: 0.776
Model: RBF SVM Accuracy: 0.736
Model: Gaussian Process Accuracy: 0.672
Model: Decision Tree Accuracy: 0.872
Model: Random Forest Accuracy: 0.944
Model: AdaBoost Accuracy: 0.872
Model: Naive Bayes Accuracy: 0.792

```

We can see that such rule decreases model performance. In order to fit to the data, the decision tree model also tries to look for alternative rules, making the tree look more different than the designed rules, mostly being deeper.



## Conclusions

Although looking simple, decision tree is sure a powerful algorithm to learn the real rules. The more data given, the easier it is for decision trees to classify the samples correctly. Besides, in comparison to other complex algorithms, decision trees are more convenient for human to understand and interpret, since it can be visualized and inspected. However, the main drawback of decision trees is that it might suffer from overfitting problem when the data size is small, not being able to represent the dataset well enough. A possible solution worth trying is to limit the maximum height of the tree, this somewhat reduces the model complexity.

## Reference for data generation

---

<https://zh.wikipedia.org/wiki/台灣黑熊>

[.\(https://zh.wikipedia.org/wiki/%E5%8F%B0%E7%81%A3%E9%BB%91%E7%86%8A\).](https://zh.wikipedia.org/wiki/%E5%8F%B0%E7%81%A3%E9%BB%91%E7%86%8A)

<https://www.taiwanbear.org.tw/> [\(https://www.taiwanbear.org.tw/\)](https://www.taiwanbear.org.tw/).