MIS4311
Machine Learning Applications

Spring 2025

Lecture #5

Support Vector Machine

Support vector machines (SVMs) are a set of supervised learning methods **used for classification regression and outliers detection**.

Support Vector Machine is a discriminative classifier that is formally designed by a **separative hyperplane**.

It is a representation of examples as points in space that are mapped so that the points of **different categories are separated by a gap as wide as possible.**

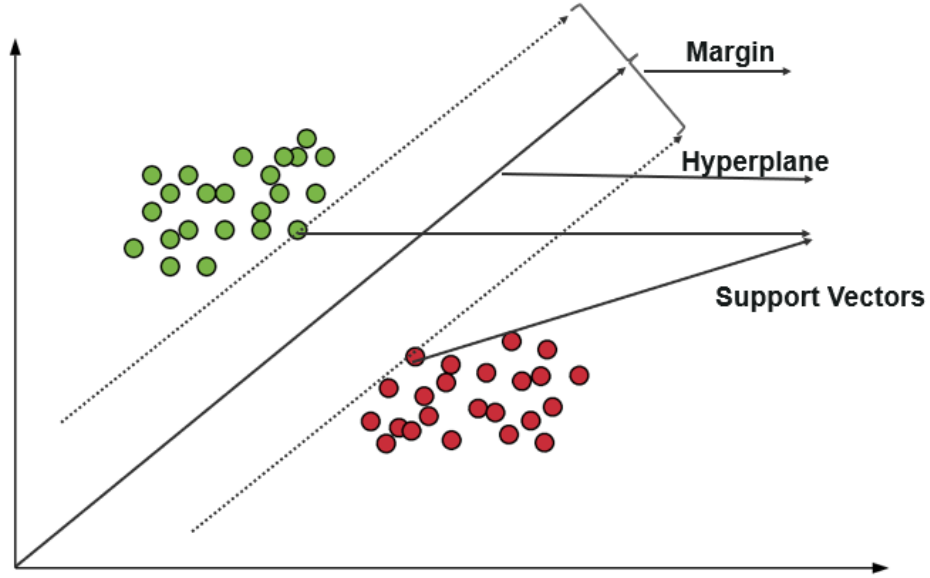
In addition to this, an SVM can also perform **non-linear classification.**

The Margin and Support Vectors

The main objective of a support vector machine is to **segregate the given data in the best possible way.**

When the segregation is done, **the distance between the nearest points is known as the margin (labeled by M).**

Support vectors are the datapoints in each class that lie closest to the classification line (hyperline)



The approach is to **select a hyperplane with the maximum possible margin** between the support vectors in the given data-sets.

Support Vector Machine

Two arguments for the best classifier (one that goes through the middle of two support vectors):

1. The margin should be as large as possible

2. The **support vectors are the most useful datapoints** because they are the ones that we might get wrong.

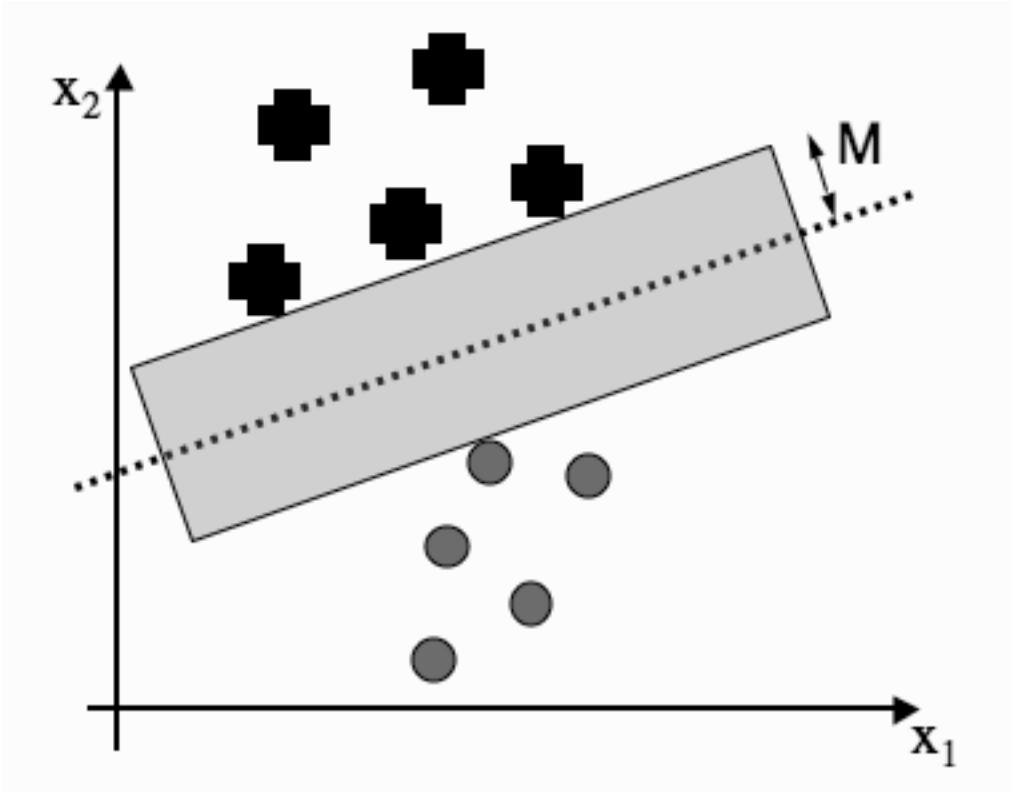
Thus **after training** we **can throw away all of the data except for the support vectors**, and use them for classification, which is a useful **saving in data storage**.

The objective is to find the optimal decision boundary for a given set of datapoints.

Support Vector Machine

In the perceptron algorithm, we have a weight (**w**) vector and an input vector **x**.

The output is y = **w**·**x** + b, with b being the contribution from the bias weight.



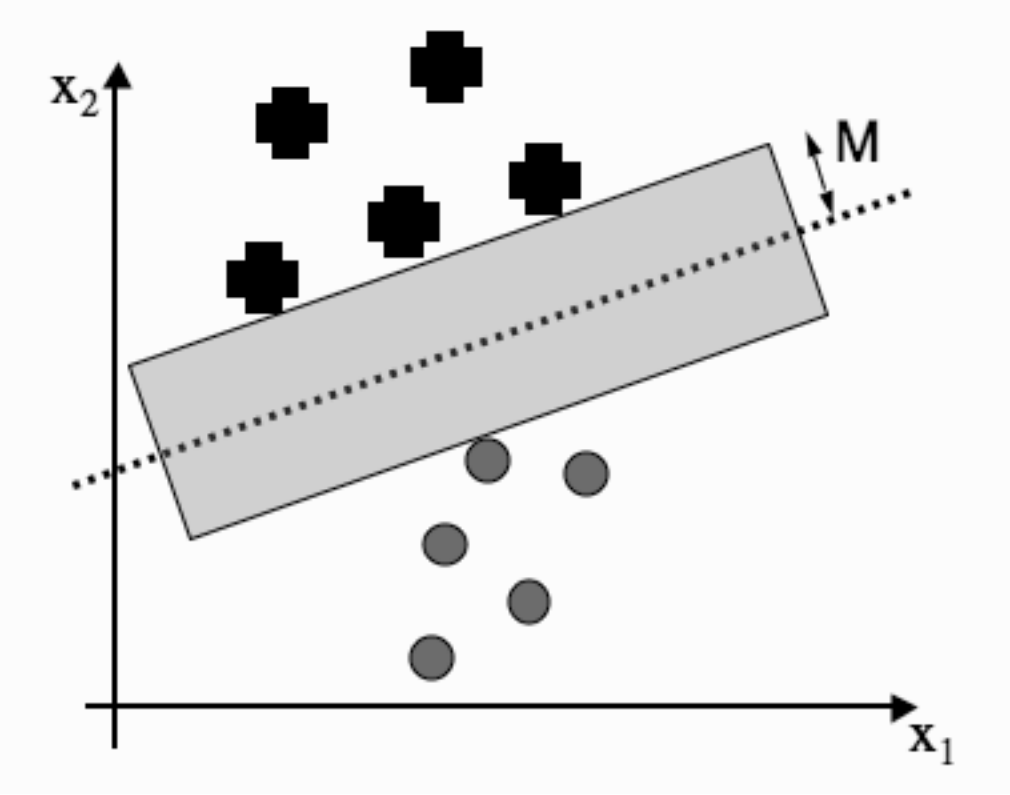
We use the classifier line by saying that any **x** value that gives a positive value for **w**·**x** + b is above the line, and so is an example of the '+' class, while any **x** that gives a negative value is in the 'o' class.

Support Vector Machine

In SVM approach, we also check whether the absolute value is less than our margin M, which would put it inside the grey box in the Figure.

$\mathbf{w} \cdot \mathbf{x} = \sum_i w_i x_i$. This can also be written as $\mathbf{w}^T \mathbf{x}$,

to simplify notation.



For a given margin value M we can say that any point **x** where $\mathbf{w}^T\mathbf{x}+b \geq M$ is a plus (+), and any point where $\mathbf{w}^T\mathbf{x} + b \leq -M$ is a circle (o).

The actual separating hyperplane is specified by $\mathbf{w}^T\mathbf{x} + b = 0$

Advantages of SVM

Effective in **high dimensional spaces**

Effective in cases where the number of dimensions is greater than the number of samples

Uses a subset of training points in the decision function that makes it **memory efficient**

Different **kernel functions can be specified** for the decision function that also makes it versatile (changeable)

Disadvantages of SVM

If the number of features is much larger than the number of samples, **avoid over-fitting in choosing kernel functions and regularization term is crucial.**

SVMs do **not directly provide probability estimates.**

The usage of SVMs

handwriting recognition,
face detection,
email classification
gene classification

Support Vector Machines (SVM) with Python

Support vector machine (SVM) is a **supervised** machine learning algorithm that can be used for both **regression and classification**.

The main concept of SVM is to plot each data item as a point in n-dimensional space with the **value of each feature being the value of a particular coordinate**. Here **n** would be **the features we would have**.

We build an SVM classifier by using **scikit-learn** and **iris dataset**. Scikitlearn library has the **sklearn.svm** module and provides **sklearn.svm.svc** for classification.

The SVM classifier to predict the class of the iris plant based on 4 features are shown below.

SVM with Python

iris dataset contains 3 classes of 50 instances each, where each class refers to a type of iris plant. **Each instance** has the **four features** namely **sepal length, sepal width, petal length and petal width**.

The SVM classifier to predict the class of the iris plant based on 4 features.

```
import pandas as pd
import numpy as np
from sklearn import svm, datasets
import matplotlib.pyplot as plt
iris = datasets.load_iris()       #load input data
```

SVM with Python

```
X = iris.data[:, :2]                    #take the first 2 features
y = iris.target
```

*Plot the support vector machine boundaries with original data. We are creating a mesh to plot.

```
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (x_max / x_min)/100
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
np.arange(y_min, y_max, h))
X_plot = np.c_[xx.ravel(), yy.ravel()]
```

SVM with Python

```
array = np.arange(0, 3, 0.5).reshape(3,2)
print(array)
[[0.  0.5]
 [1.  1.5]
 [2.  2.5]]
print(array.ravel())
[0.  0.5 1.  1.5 2.  2.5]          #flattened by ravel


np.c_[np.array([1,2,3]), np.array([4,5,6])]
Output: array([[1, 4],
               [2, 5],
               [3, 6]])
```

SVM with Python

Create the SVM classifier object with regularization parameter C=1

C = 1.0
svc_classifier = svm.SVC(kernel='linear', C=C,
decision_function_shape='ovr').fit(X, y)

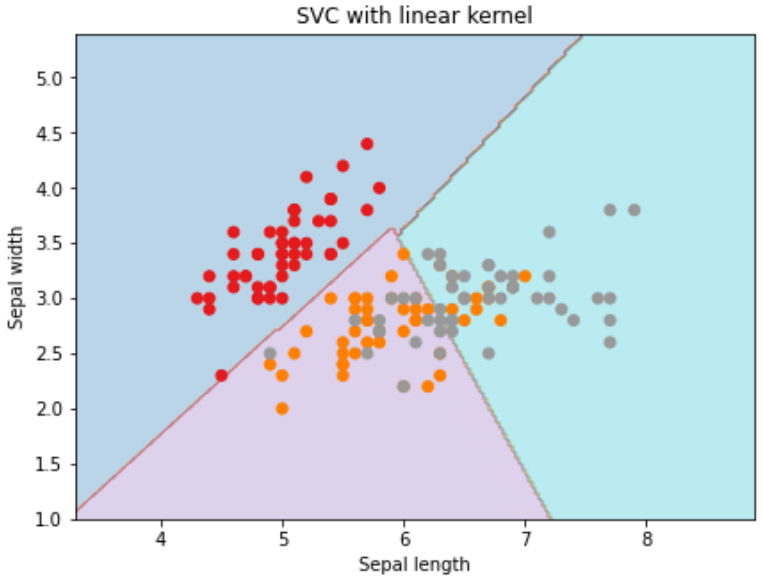
Kernel is a technique used by SVM. Basically these are the functions which take low-dimensional input space and transform it to a higher dimensional space. It converts non-separable problem to separable problem. The kernel function can be any one among linear, polynomial, rbf (Radial Basis Function) and sigmoid.

To return a one-vs-rest (‘**ovr**’) decision function of shape (n_samples, n_classes) as all other classifiers.

SVM with Python

Use SVM classifier object to predict

Z = svc_classifier.predict(X_plot)



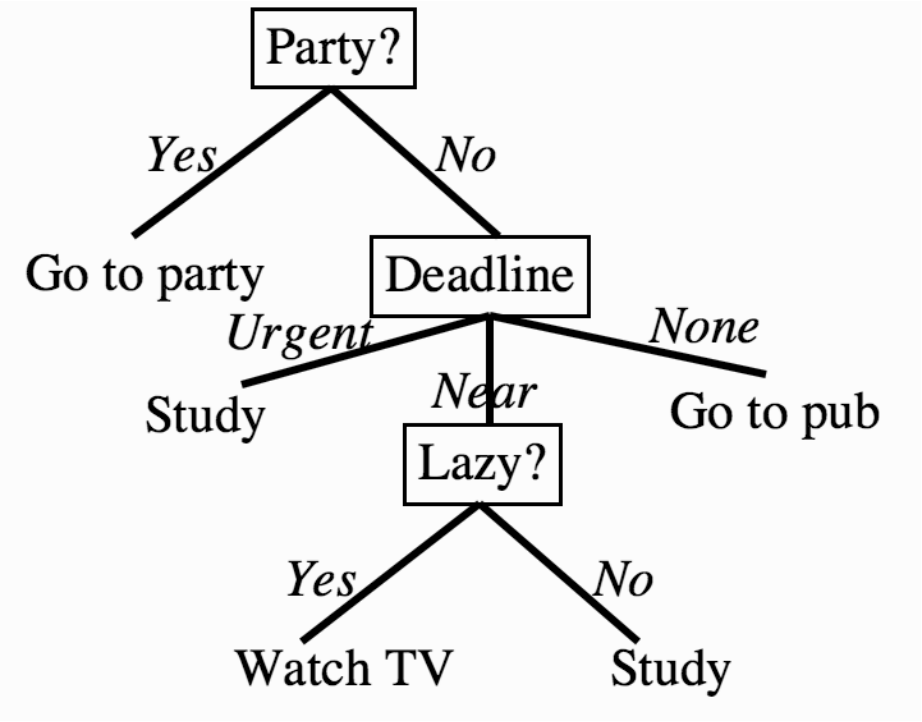
Plot:

Z = Z.reshape(xx.shape) #reshape according to xx
plt.figure(figsize=(15, 5))
plt.subplot(121)
plt.contourf(xx, yy, Z, cmap=plt.cm.tab10, alpha=0.3) #cmap=colormap
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1)
c=y giving color to the plot with respect to y
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('SVC with linear kernel')

The Decision Tree Algorithm

A Decision Tree is a **Supervised** Machine Learning algorithm that can be easily visualized using a connected acyclic (non-cyclic) graph.

A decision tree is a flowchart-like tree structure where an **internal node** represents a feature (or attribute), **the branch** represents a decision rule, and each **leaf node** represents the outcome.



The **topmost node** in a decision tree is known as the **root node**.

It learns to **partition on the basis of the attribute value**.

The Decision Tree Algorithm

It partitions the tree in a recursive manner called **recursive partitioning**.

It's visualization like a flowchart diagram which easily **mimics the human level thinking.** That is why decision trees are **easy to understand and interpret.**

Its training time is faster compared to the neural network algorithm.

The time complexity of decision trees is a function of **the number of records and attributes in the given data.**

The decision tree is a **distribution-free** or **non-parametric** method which does not depend upon probability distribution assumptions.

Decision trees **can handle high-dimensional data** with good accuracy.

The Decision Tree Algorithm

The basic idea behind any decision tree algorithm is as follows:

1) **Select the best attribute** using Attribute Selection Measures (ASM) to split the records.

2) Make that attribute a decision node and **breaks the dataset into smaller subsets.**

3) Start tree building by **repeating this process recursively** for each child until **one of the conditions will match**:

    - All the tuples belong to the same attribute value.
    - There are no more remaining attributes.
    - There are no more instances

The Decision Tree Algorithm

There are a few different decision tree algorithms, but they are almost all variants of the same principle: the algorithms build the tree starting at the root, **choosing the most informative feature at each step.**

Attribute selection measure is a heuristic for selecting the **splitting** criterion that **partitions data in the best possible manner.**

It is also known as splitting rules because it **helps us to determine breakpoints for tuples on a given node**.

ASM **provides a rank to each feature** (or attribute) by explaining the given dataset.

The most popular **selection measures** are Information Gain, Gain Ratio, and Gini Index.

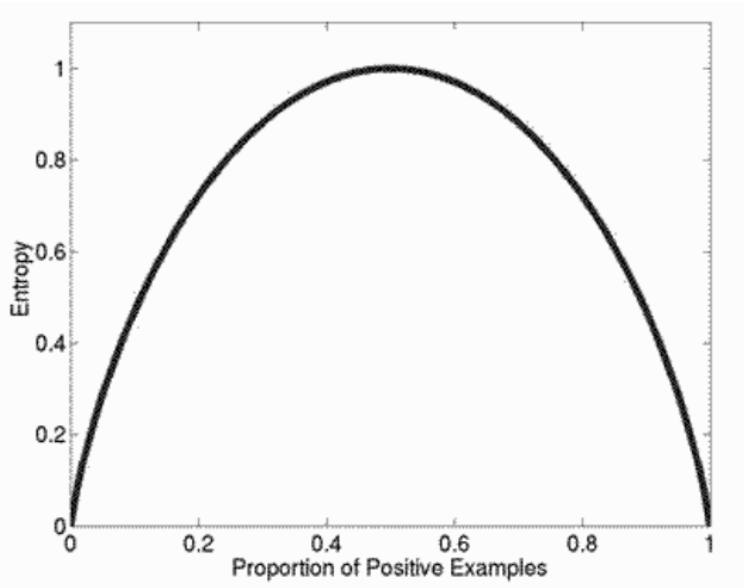
The Decision Tree Algorithm

The measure of **information entropy** describes the amount of impurity in a set of features.

The entropy H of a set of probabilities is:

$$\text{Entropy}(p) = -\sum_i p_i \log_2 p_i,$$

where the logarithm is base 2 because we are imagining that we encode everything using binary digits (bits), and we define 0 log 0 = 0.



If all of the examples are positive, then we don't get any extra information from knowing the value of the feature for any particular example, since whatever the value of the feature, the example will be positive. Thus, the entropy of that feature is 0.

However, if the feature **separates the examples into 50% positive and 50% negative**, then the amount of entropy is at a maximum, and knowing about **that feature is very useful to us.**

The Decision Tree Algorithm

For our decision tree, **the best feature to pick** as the one to classify on now is the one that **gives you the most information**, i.e., the one with **the highest entropy**. After using that feature, we re-evaluate the entropy of each feature and again pick the one with the highest entropy.

The important idea is to work out how much the entropy of the whole training set would decrease if we choose each particular feature for the next classification step. This is known as the **information gain**, and it is defined as the entropy of the whole set minus the entropy when a particular feature is chosen.

$$\text{Gain}(S, F) = \text{Entropy}(S) - \sum_{f \in values(F)} \frac{|S_f|}{|S|} \text{Entropy}(S_f). \qquad (12.2)$$

where S is the set of examples, F is a possible feature out of the set of all possible ones, and $|S_f|$ is a count of the number of members of S that have value f for feature F .

The Decision Tree Algorithm

As an example, suppose that we have data (with outcomes)
S = {s1 = true, s2 = false, s3 = false, s4 = false}

and one feature F that can have values {f1, f2, f3}.

In the example, the feature value for s1 could be f2, for s2 it could be f2, for s3, f3 and for s4, f1

Then we can calculate the entropy of S as (where $\oplus$ means true, of which we have one example, and $\ominus$ means false, of which we have three examples):

$$\begin{aligned} \text{Entropy}(S) &= -p_{\oplus}\log_2 p_{\oplus} - p_{\ominus}\log_2 p_{\ominus} \\ &= -\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} \\ &= 0.5 + 0.311 = 0.811. \end{aligned}$$

The Decision Tree Algorithm

$$\begin{aligned}
\frac{|S_{f_1}|}{|S|}\text{Entropy}(S_{f_1}) &= \frac{1}{4} \times \left(-\frac{0}{1}\log_2\frac{0}{1} - \frac{1}{1}\log_2\frac{1}{1}\right) \\
&= 0 \qquad (12.4) \\
\frac{|S_{f_2}|}{|S|}\text{Entropy}(S_{f_2}) &= \frac{2}{4} \times \left(-\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2}\right) \\
&= \frac{1}{2} \qquad (12.5) \\
\frac{|S_{f_3}|}{|S|}\text{Entropy}(S_{f_3}) &= \frac{1}{4} \times \left(-\frac{0}{1}\log_2\frac{0}{1} - \frac{1}{1}\log_2\frac{1}{1}\right) \\
&= 0 \qquad (12.6)
\end{aligned}$$

The information gain from adding this feature is the entropy of S minus the sum of the three values above:

$$\text{Gain}(S, F) = 0.811 - (0 + 0.5 + 0) = 0.311.$$

The algorithm computes this information gain for each feature and chooses the one that produces the highest value.

Decision Tree Classifier with Python

A decision tree is basically **a binary tree flowchart** where each node splits a group of observations according to some feature variable.
In the example, we are building a Decision Tree classifier for predicting male or female. We will take a very small data set having 19 samples. These samples would consist of two features – 'height' and 'length of hair'.

Prerequisite:
For building the following classifier, we need to install **pydotplus** and **graphviz.**

conda install -c conda-forge pydotplus
conda install -c anaconda graphviz

Decision Tree Classifier with Python

Using modules:

```
import pydotplus
from sklearn import tree
from sklearn.model_selection import train_test_split
import collections
```

Dataset:

```
X=[[165,19],[175,32],[136,35],[174,65],[141,28],[176,15],[131,32
],[166,6],[128,32],[179,10],[136,34],[186,2],[126,25],[176,28],[11
2,38],[169,9],[171,36],[116,25],[196,25]]

Y=['Man','Woman','Woman','Man','Woman','Man','Woman','Man','Wo
man','Man','Woman','Man','Woman','Woman','Woman','Man','Woman'
,'Woman','Man']
```

Decision Tree Classifier with Python

Split data:

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.40, random_state=5)

Train: fit the model

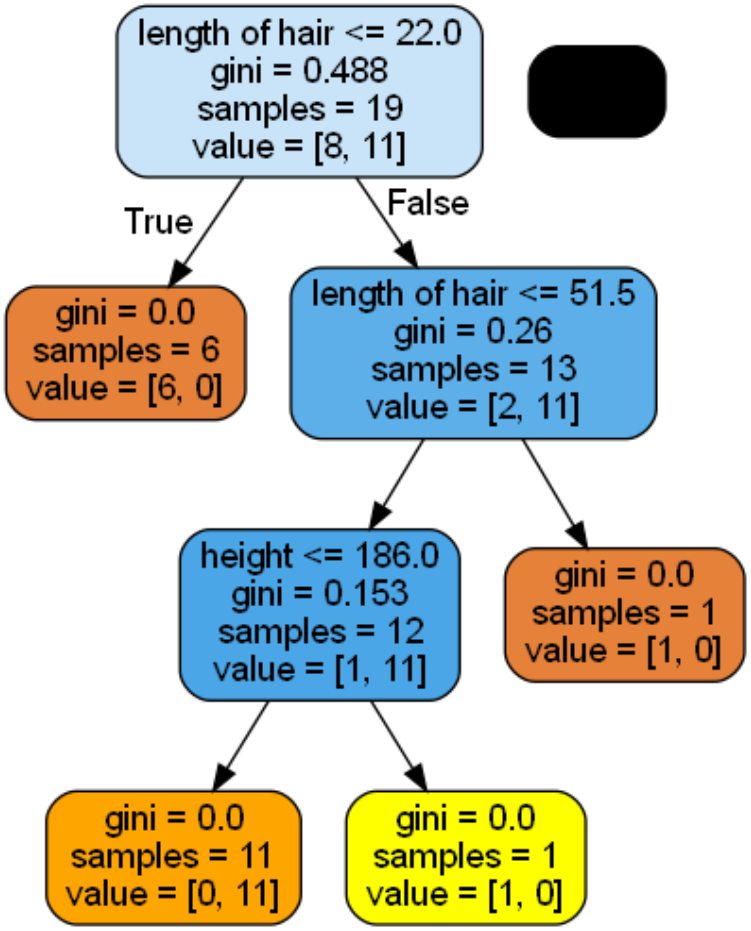
clf = tree.DecisionTreeClassifier()

clf = clf.fit(X,Y)

Predict:

prediction = clf.predict([[163,15]])

print(prediction)



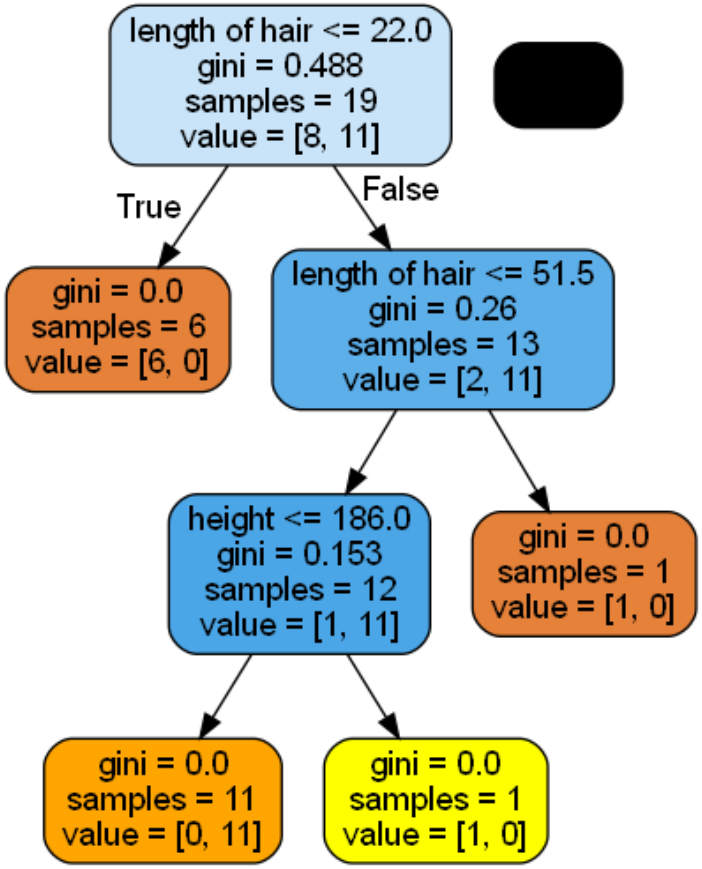
Decision Tree Classifier with Python

length_of_hair <= 22 means that whose hair is shorter than 22cm will follow the True arrow (to the left), and the rest will follow the False arrow (to the right).

gini = 0.488 refers to the quality of the split, and is always a number between 0.0 and 0.5, where 0.0 would mean all of the samples got the same result, and 0.5 would mean that the split is done exactly in the middle.

samples = there are 19 people in total.

value = [8, 11] means that of these 8 are man and 11 are woman.



Decision Tree Classifier with Python

Plot and save figure: visualize the decision tree

```
dot_data
=tree.export_graphviz(clf,feature_names=data_feature_names,out_f
ile=None,filled=True,rounded=True)
graph = pydotplus.graph_from_dot_data(dot_data)
colors = ('orange', 'yellow')
edges = collections.defaultdict(list)
for edge in graph.get_edge_list():
    edges[edge.get_source()].append(int(edge.get_destination()))
for edge in edges:
    edges[edge].sort()
for i in range(2):
    dest = graph.get_node(str(edges[edge][i]))[0]
    dest.set_fillcolor(colors[i])
graph.write_png('Decisiontree.png')
```

Next Week

- K-Nearest Neighbors (KNNs)
- Unsupervised Learning : Clustering (K-means)

Thank you for your participation ☺