Studying the Slave Trade using Ship Logbooks

Emma Cotter

Alicia Clark

Wedward Wei

Project Motivation

- Many slave logs have been transcribed
- Large data set used explicitly to study slave trade
- Want to use this data set to predict if ships were involved in the slave trade in other available data sets

<u>Goal:</u> Identify ships related to the slave trade in the climate data set

How are we going to do this?

1. Join data sets
2. Encode categorical data
3. Train classifier to detect slave ships
4. Visualize trends in classified data

Encoding – Label Encoder

Original Categorical Labels			
Paris	Paris	Amsterdam	Tokyo

Assign a Numerical Value to Each Category		
Paris = 0	Amsterdam =1	Tokyo = 2

Transform the Labels to Numerical Labels			
0	0	1	2

Encoding - One Hot Encoder

Log 1	Log 2	Log 3	Log 4
Paris	Paris	Amsterdam	Tokyo
0	0	1	2

	Paris	Amsterdam	Tokyo
Log 1	1	0	0
Log 2	1	0	0
Log 3	0	1	0
Log 4	0	0	1

scikitlearn Preprocessing

Pros

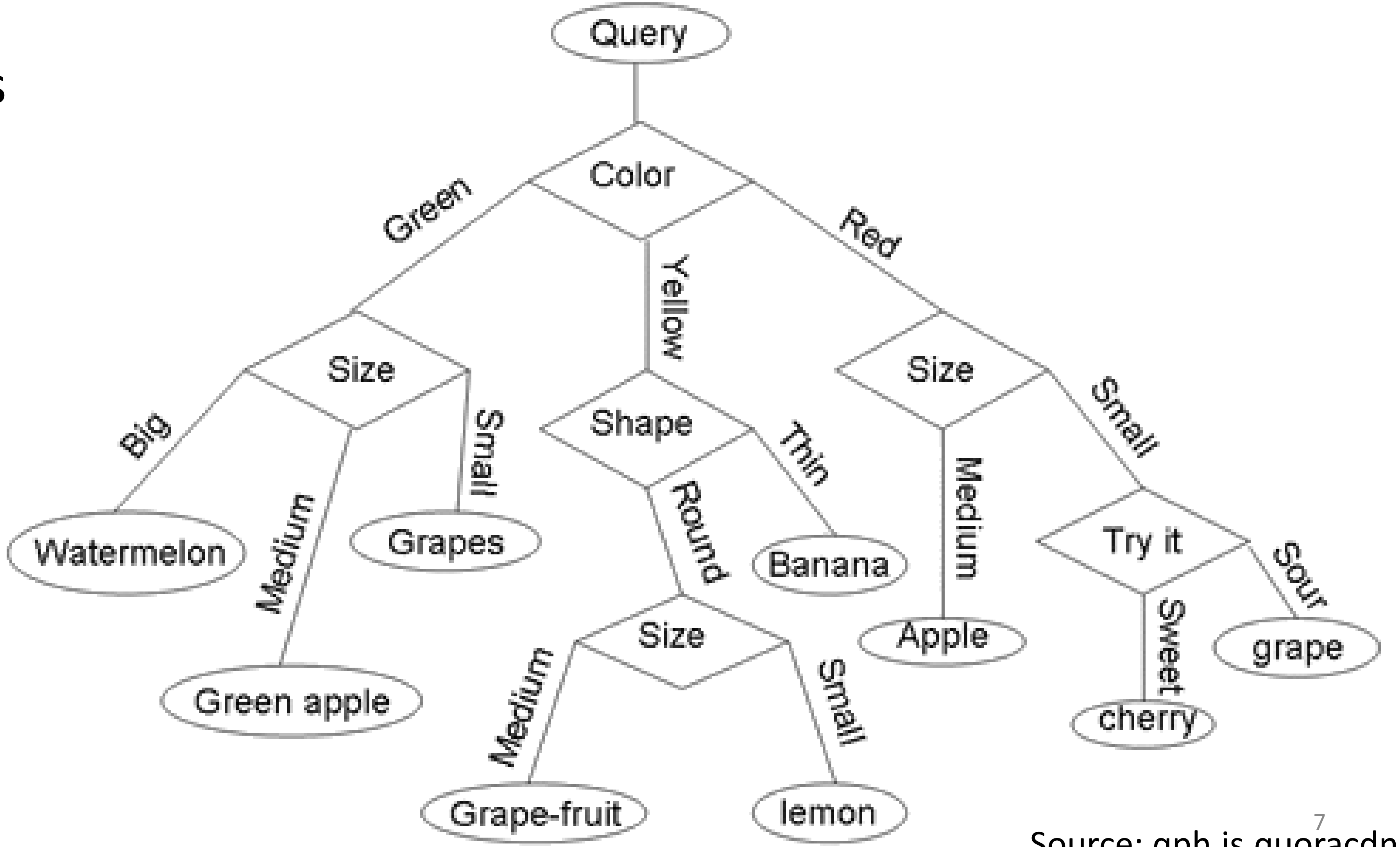
- Fast and works well
- Well maintained on github

Cons

- Methods poorly documented by sklearn (need outside documentation to understand)
- One hot encoding is not a good option when there are many possible values (memory issues)

Classifier Options

- Decision Trees
- Naïve bayes
  - Gaussian
  - Multinomial



Source: qph.is.quoracdn.net

Classification Requirements

Data
- Categorical
- Small number of characteristics
- Relatively large training data set

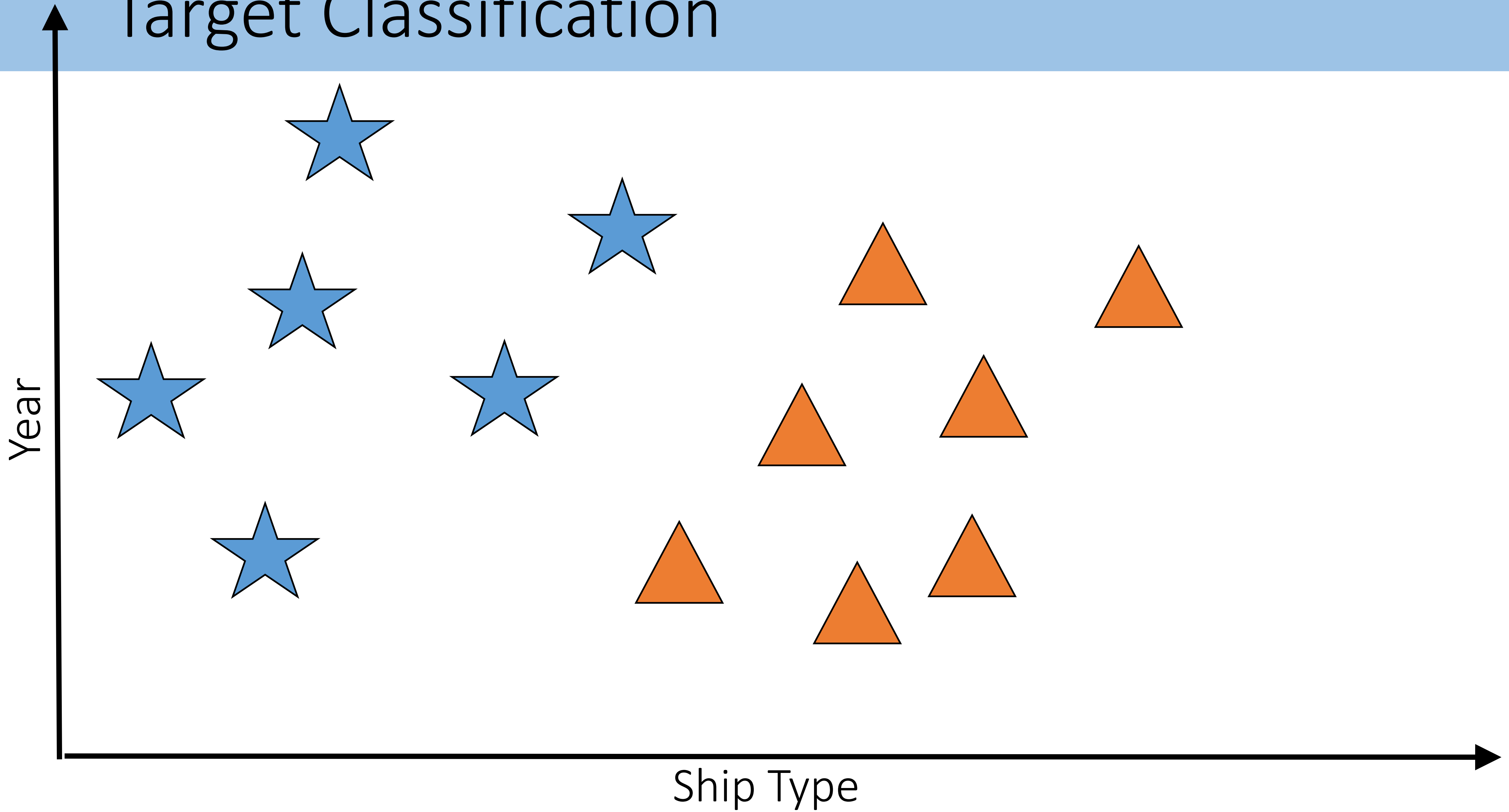
Software
- Python 3.5
- Well-documented

SciKitLearn Multinomial Naïve Bayes

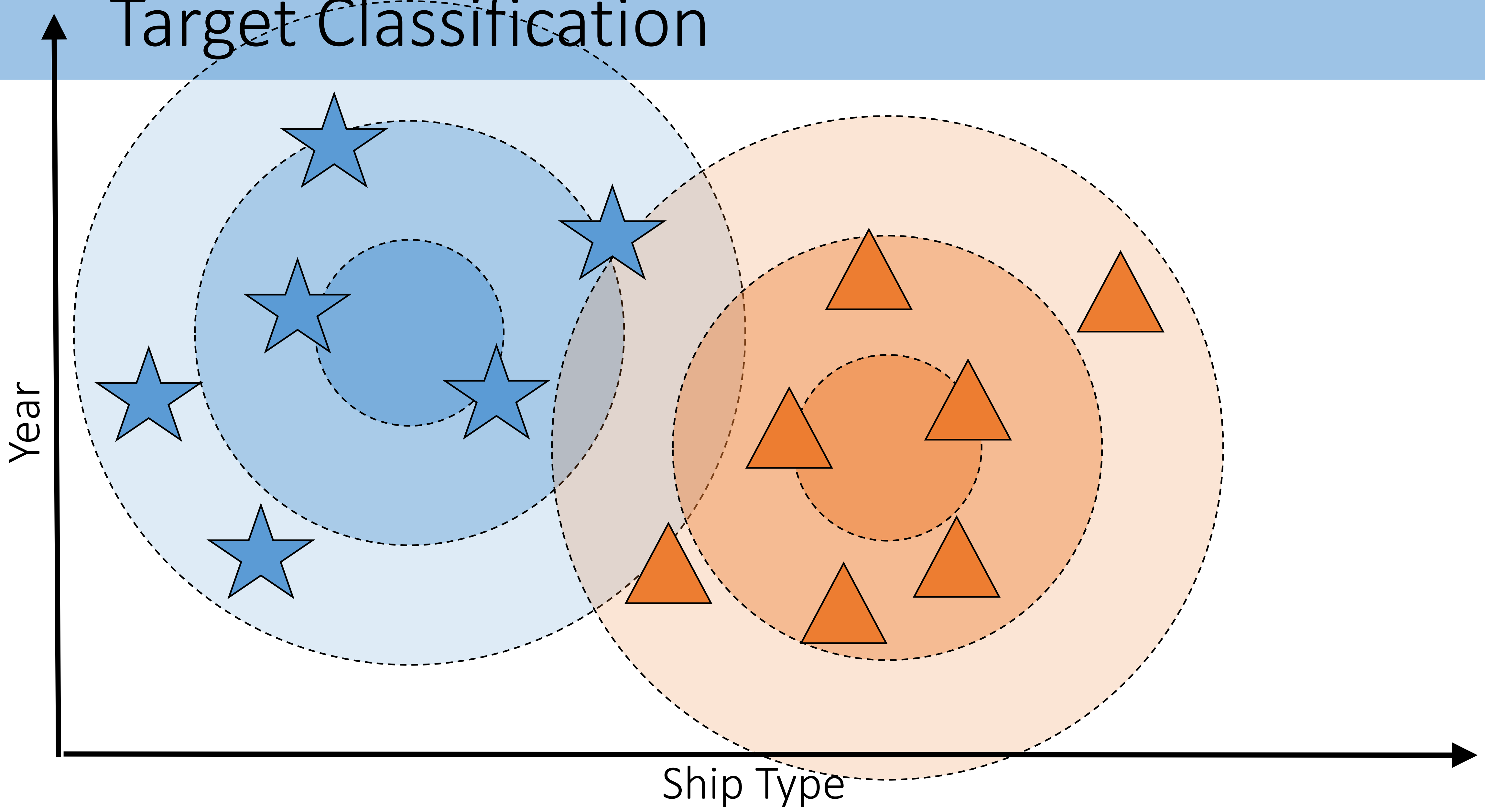
Multinomial Naïve Bayes Classification

- Commonly used in real-world classification:
  - Document classification
  - Spam filtering
- Assumes all data are independent – but still performs well when this is not true

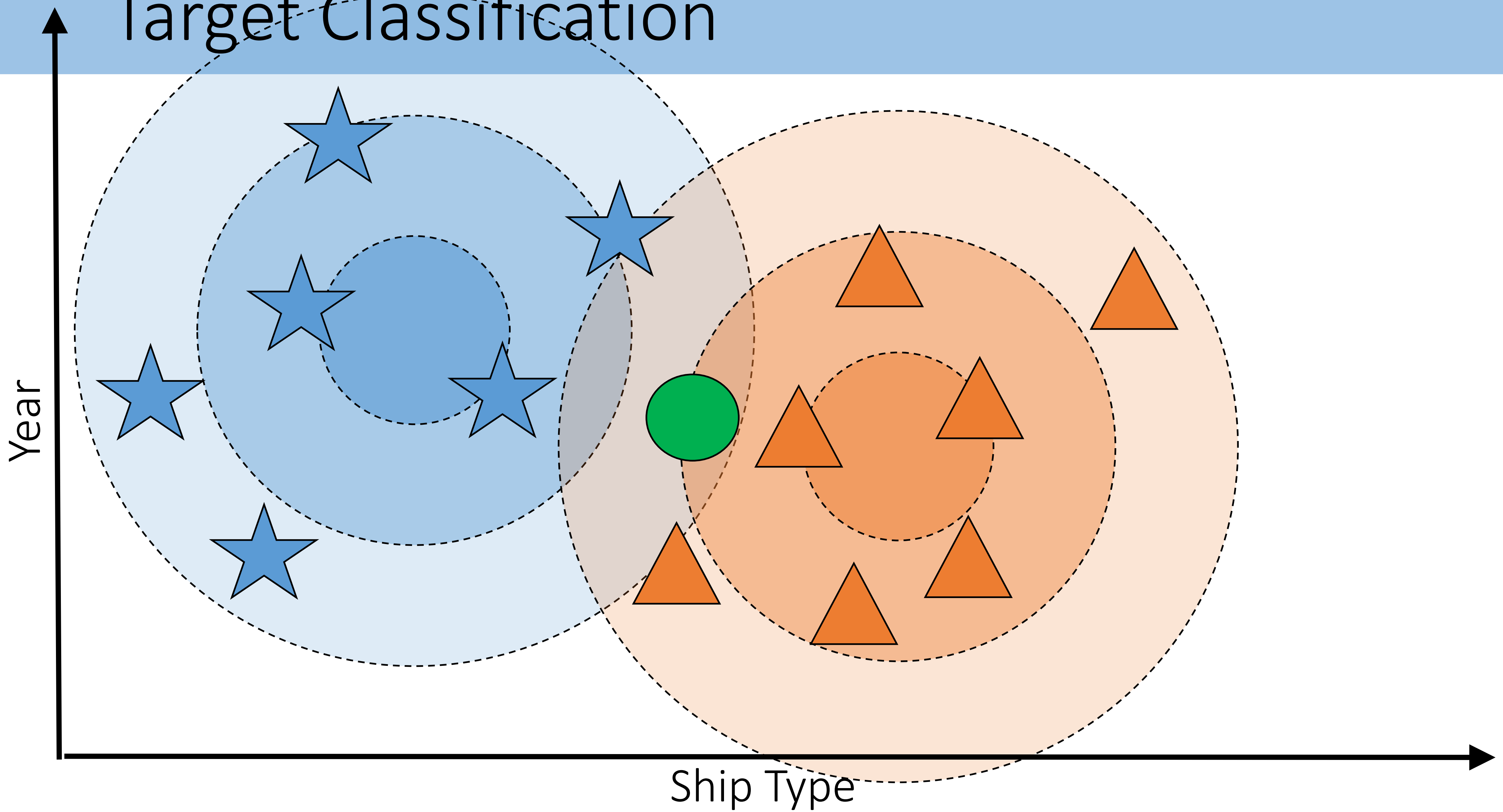
Target Classification



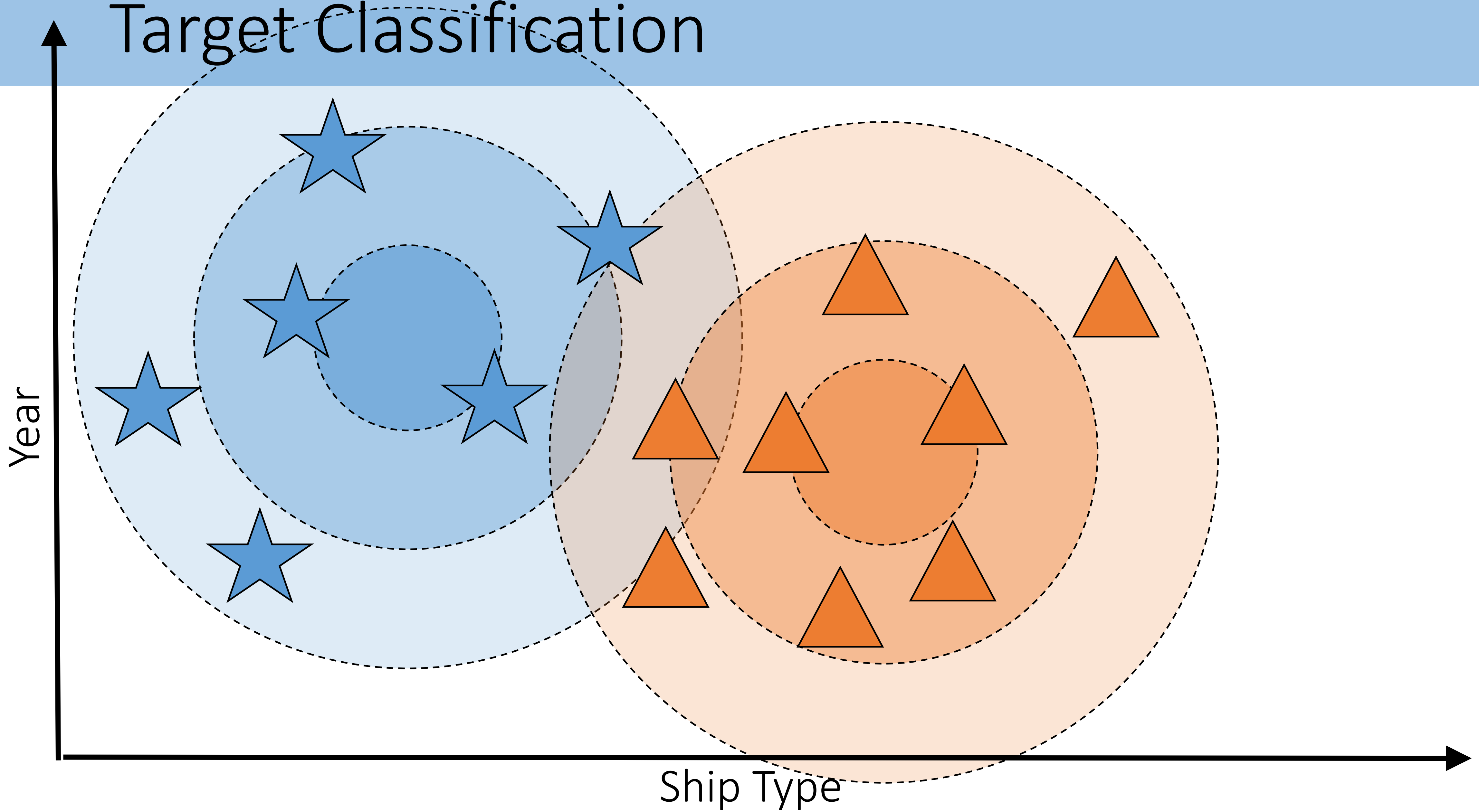
Target Classification



Target Classification



Target Classification



scikitlearn Multinomial Naïve Bayes

Pros

- Use is well documented
- scikitlearn is widely used
- Issues are well addressed on github

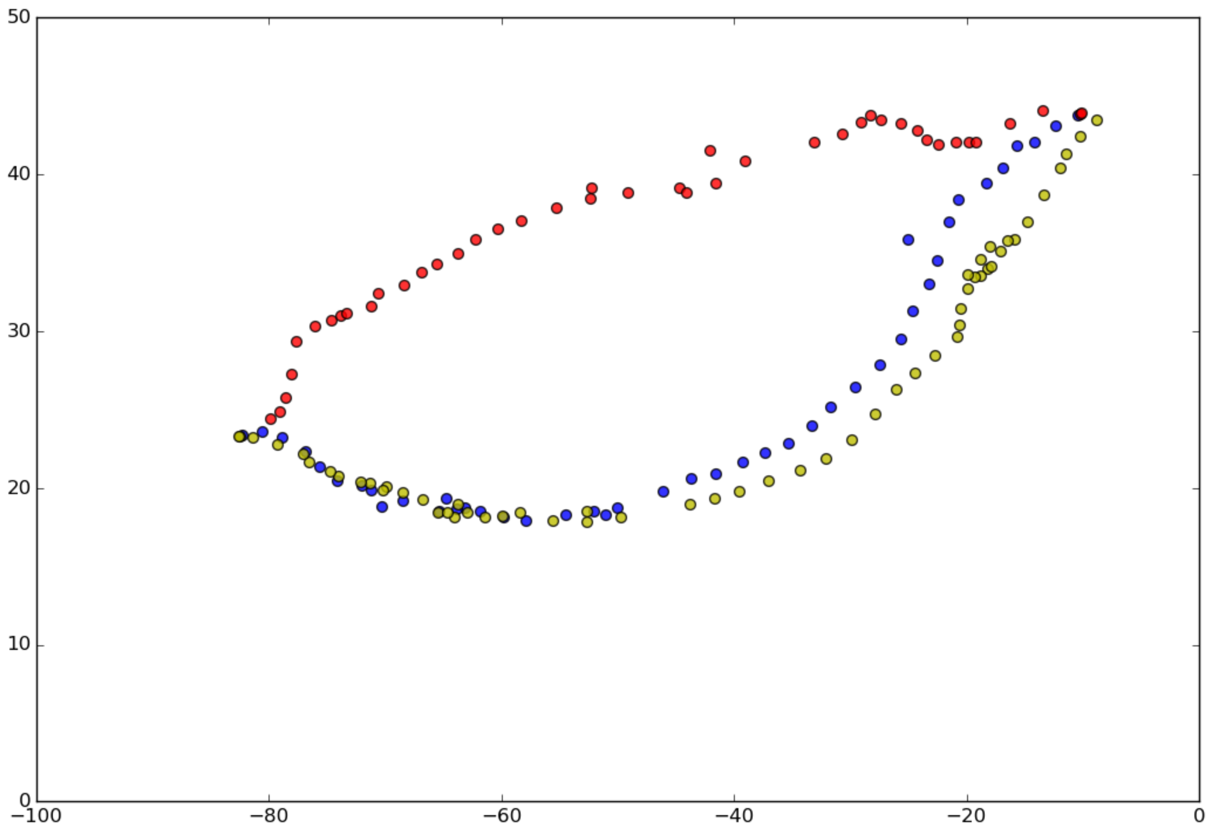
Cons

- Poor documentation on how the algorithm works
- Is it the best classification algorithm for our data?

Visualization Packages - matplotlib

- Matplotlib Scatter

```
import matplotlib.pyplot as plt
position_list = list(zip(position["Lat3"],position["Lon3"]))
plt.xlim(-100,0)
plt.ylim(0,50)
plt.figure(figsize=(40, 20))
for i in position_list:
        plt.scatter(i[1], i[0],s=100,c=colors[color],alpha=0.8)
plt.show()
```

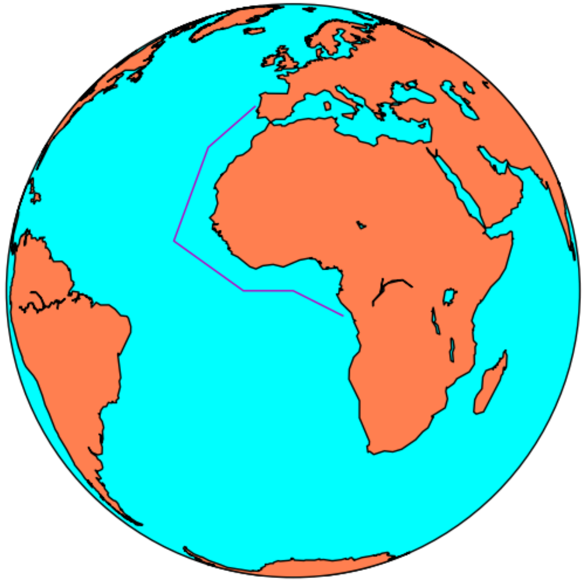


- Basemap
- Fusion Table’s Map

Visualization Packages - basemap

- Matplotlib Scatter
- Basemap

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
map = Basemap(projection='ortho',
          lat_0=0, lon_0=0)
map.drawmapboundary(fill_color='aqua')
map.fillcontinents(color='coral',lake_color='aqua')
map.drawcoastlines()
Lons = [-10, -20, -25, -10, 0, 10]
lats = [40, 30, 10, 0, 0, -5]
x, y = map(lons, lats)
map.plot(x, y, marker=None,color='m')
plt.show()
```



- Fusion Table's Map

https://basemaptutorial.readthedocs.org/en/latest/plotting_data.html

Visualization -

- Matplotlib Scatter
- Basemap
- Fusion Table’s Map

https://www.google.com/fusiontables/embedviz?q=select+col7+from+1m9rD4onw6XZD_mwaqPHq5CC-aUrIq4ZuuIJSrtdr+where+col9+%3E%3D+0+and+col9+%3C%3D+1757&viz=MAP&h=false&lat=31.859404045459222&lng=-34.5449375&t=1&z=3&l=col7&y=2&tmplt=2&hml=TWO_COL_LAT_LNG

https://www.google.com/fusiontables/DataSource?docid=1m9rD4onw6XZD_mwaqPHq5CC-aUrIq4ZuuIJSrtdr

Mapping

Basemap

Pros

- Robust plotting toolkit
- Works with matplotlib
- Good documentation

Cons

- Difficult to install on Windows
- Poor installation instructions

Fusion Tables

Pros

- Much better looking
- …seriously, did you see that?

Cons

- Difficult to import data

Distance Function

- Measuring similarity or distance between two data points is very fundamental to many Machine Learning algorithms such as K-Nearest-Neighbor, Clustering ... etc.
- Depends on the nature of the data point, various measurement can be used.
- When the dimension of data point is numeric, the general form is called **Minkowski distance**

$$((x_1 - x_2)^p + (y_1 - y_2)^p)^{1/p}$$

- When p = 2, this is equivalent to **Euclidean distance**.
- When p = 1, this is equivalent to **Manhattan distance**.

http://horicky.blogspot.com/2012/08/measuring-similarity-and-distance.html

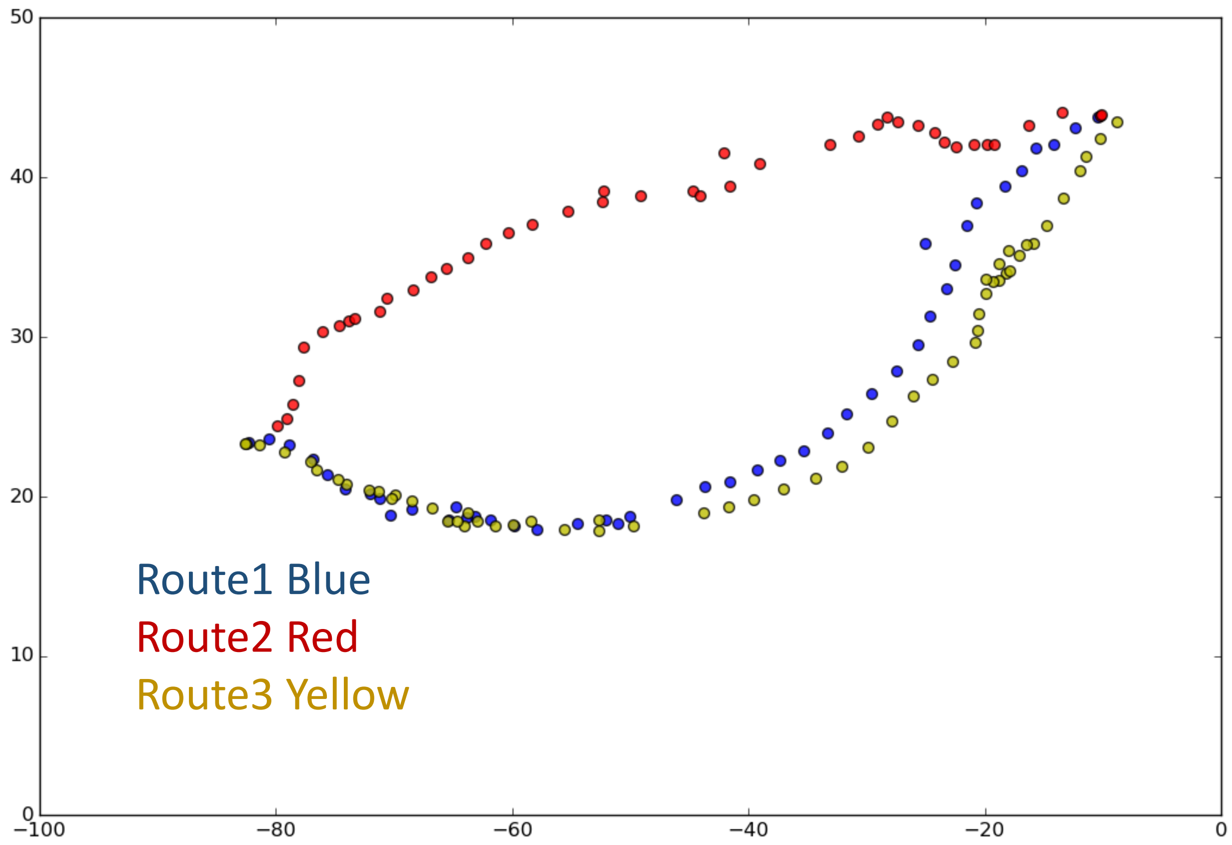
Distance Function

- The **earth mover's distance (EMD)** is a measure of the distance between two probability distributions over a region *D*. In mathematics, this is known as the Wasserstein metric.

pyemd 0.2.0 Usage Sample

```
>>> from pyemd import emd
>>> import numpy as np
>>> first = np.array([0.0, 1.0])
>>> second = np.array([5.0, 3.0])
>>> emd(first, second)
```

EMD(Route1,Route2) = 11.8620852635
EMD(Route1,Route3) = 3.66804849504
EMD(Route1,Route3) < EMD(Route1,Route2)



https://pypi.python.org/pypi/pyemd/0.2.0

EMD

Pros

- Simple to use
- Calculates differences that we need

Cons

- Poor documentation
- Cannot run on Windows