AASD 4000
Machine Learning - I

Applied AI Solutions Developer Program

Module 8
ML Algorithms - I

Vejey Gandyer

Agenda

- Model Building Template
- Linear Regression
- Support Vector Machine
- Decision Tree
- Random Forest
- K-Means

ML Algorithm

What is it?

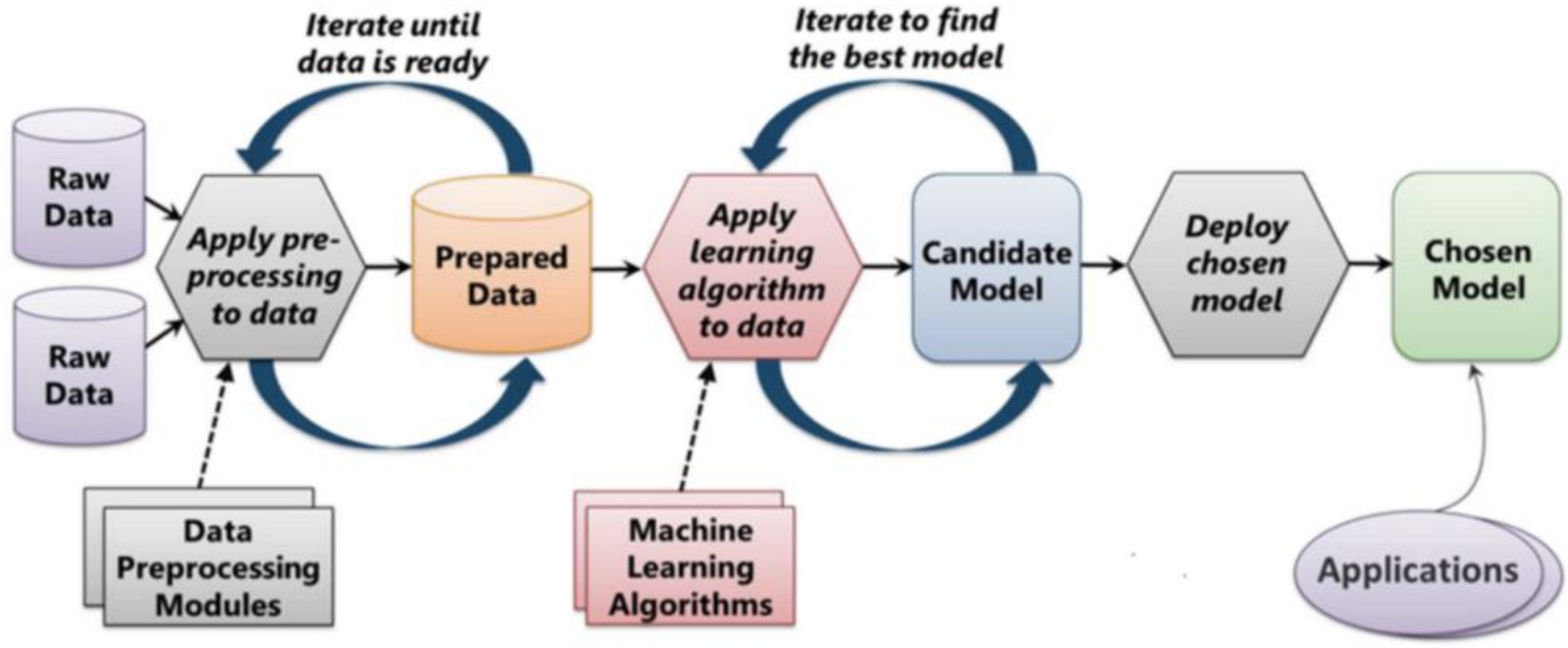
What is ML Algorithm ?

ML Algorithm is a series of steps that is used to learn a mapping function that converts raw data into set of rules.

Top ML Algorithms you should be familiar with

- Linear Regression
- Logistic Regression
- Naïve Bayes
- Gaussian Mixture Models
- Support Vector Machine
- Decision Tree
- Random Forest
- Principal Component Analysis
- K-Means
- XGBoost
- LightGBM

Machine Learning Process



From "Introduction to Microsoft Azure" by David Chappell

Model Building Template

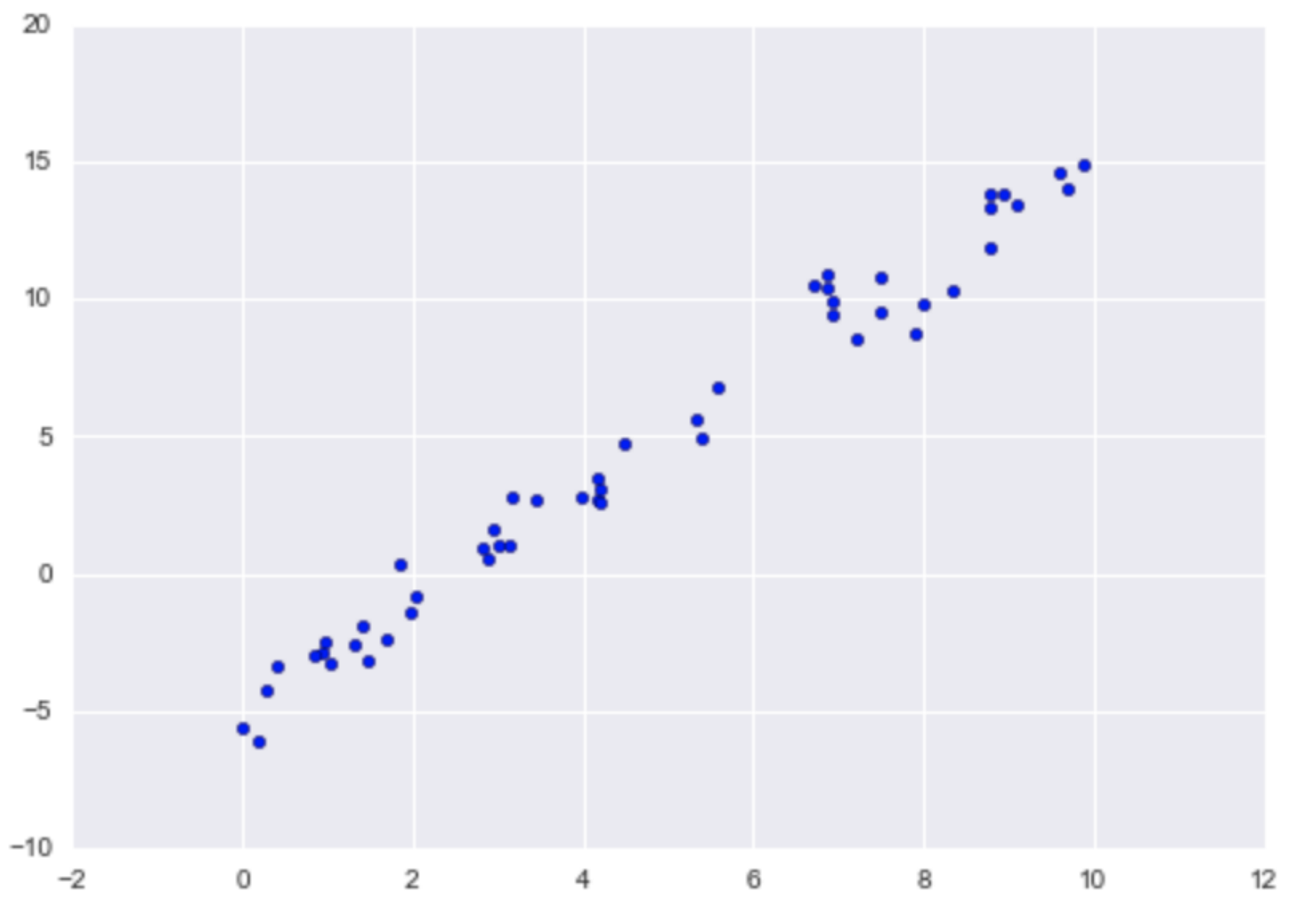
Linear Regression

Linear Regression

Linear Regression: Straight-line fit to data

$$y = ax + b$$

```
rng = np.random.RandomState(1)
x = 10 * rng.rand(50)
y = 2 * x - 5 + rng.randn(50)
plt.scatter(x, y)
```



Slope: 2
Intercept: -5

Linear Regression

```
from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True)
```

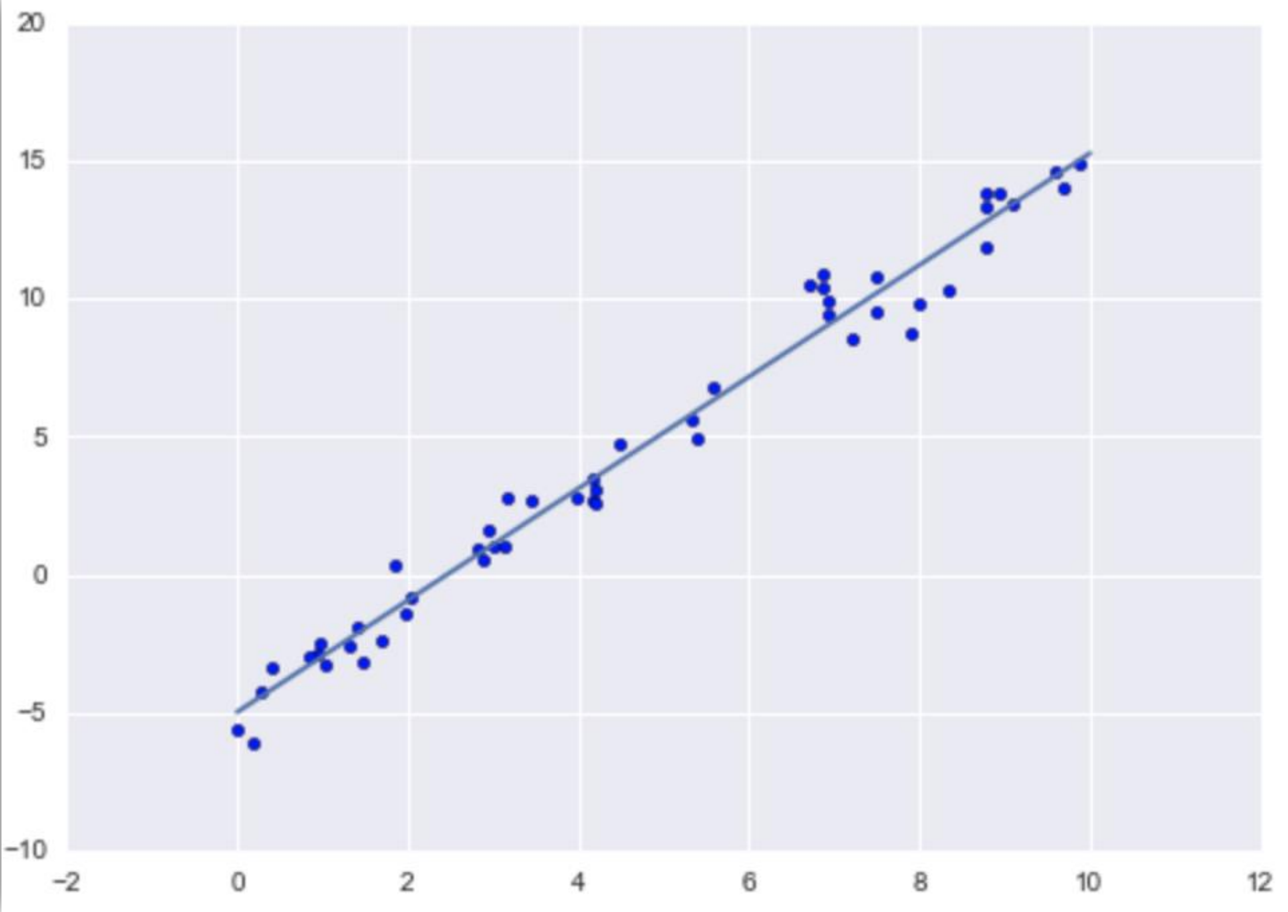
```
model.fit(x[:, np.newaxis], y)

xfit = np.linspace(0, 10, 1000)
yfit = model.predict(xfit[:, np.newaxis])

plt.scatter(x, y)
plt.plot(xfit, yfit);
```

```
print("Model slope:    ", model.coef_[0])
print("Model intercept:", model.intercept_)
```

```
Model slope:     2.02720881036
Model intercept: -4.99857708555
```



Linear Basis Functions

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_D x_D$$

Polynomial Basis Functions

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

Gaussian Basis Functions

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

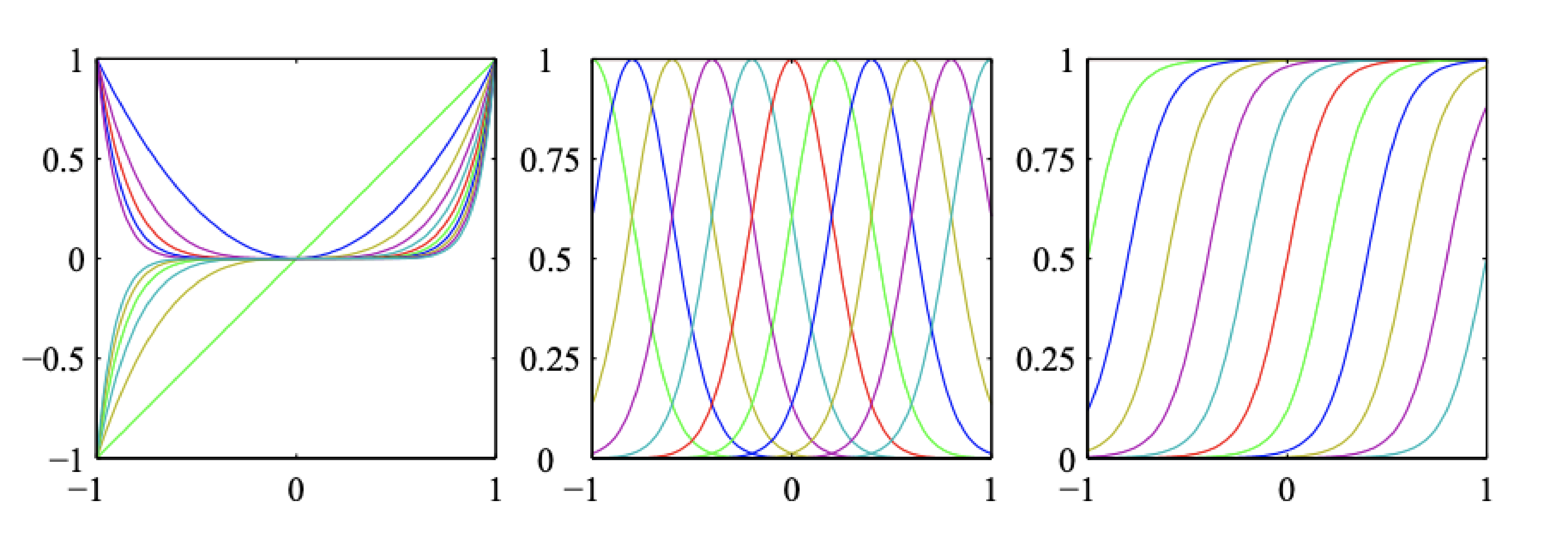
$$\phi_j(x) = \exp\left(- \frac{(x - \mu_j)^2}{2s^2} \right)$$

Sigmoidal Basis Functions

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

$$\phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right)$$

Basis Functions



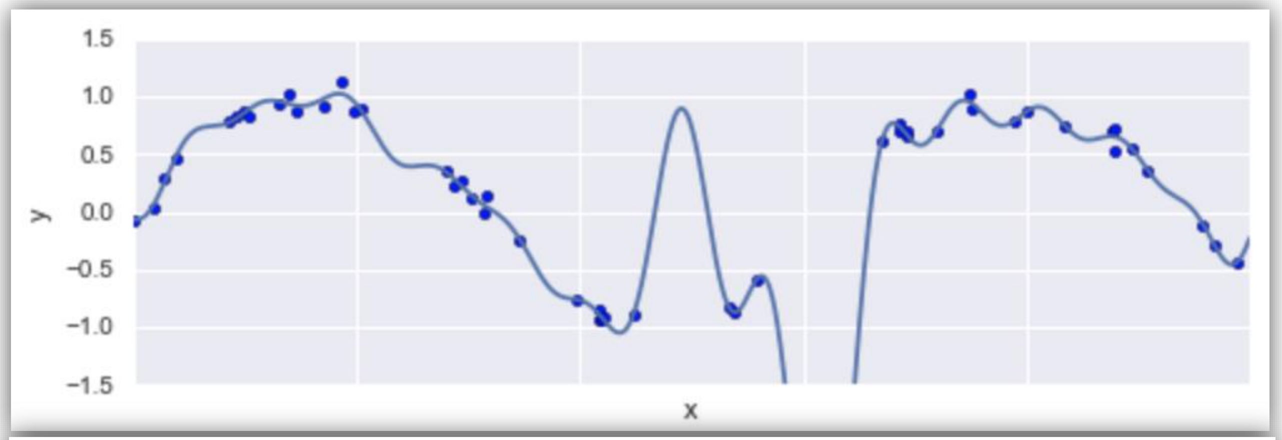
Regularization

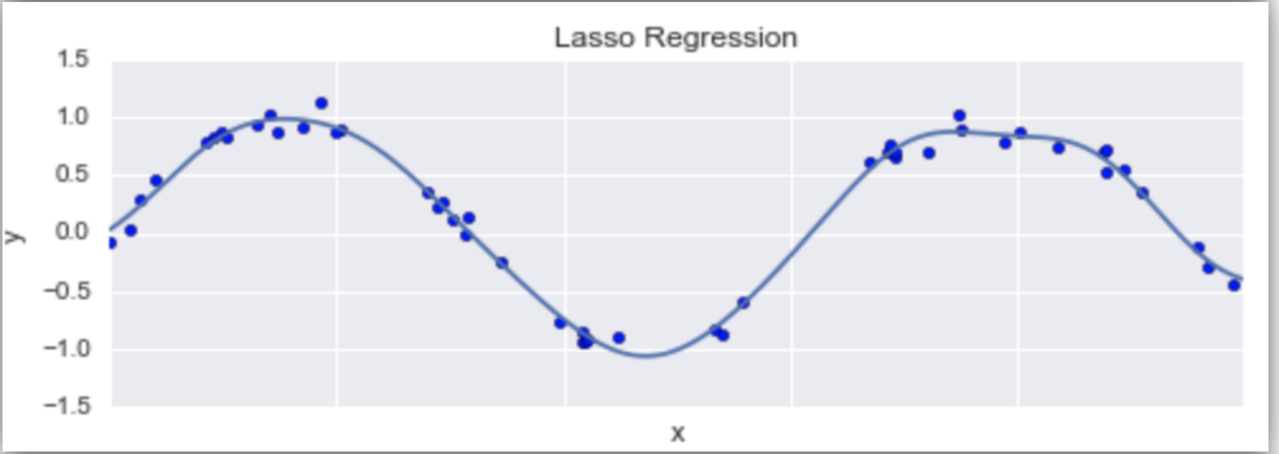
Lasso Regression (L_1 Regularization)

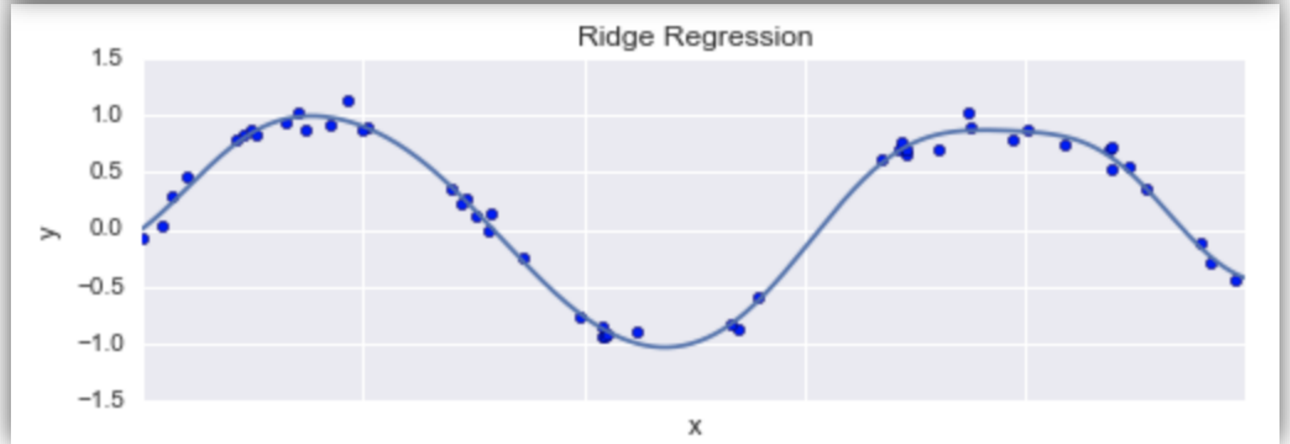
$$P = \alpha \sum_{n=1}^{N} |\theta_n|$$

Ridge Regression (L_2 Regularization)

$$P = \alpha \sum_{n=1}^{N} \theta_n^2$$







Bike Traffic Prediction

Dataset: *curl -o FremontBridge.csv https://data.seattle.gov/api/views/65db-xm6k/rows.csv?accessType=DOWNLOAD*
Weather data: http://www.ncdc.noaa.gov/cdo-web/search?datasetid=GHCND
Cleaned data: https://github.com/subashgandyer/datasets/blob/main/seattle_bike_data.csv

Steps:
Download the dataset
Explore the dataset
Prepare the dataset
Build the model
Predict on testing data
Report insights

Notebook:
ML01_Linear_Regression_Seattle_Bike.ipynb

Support Vector Machine

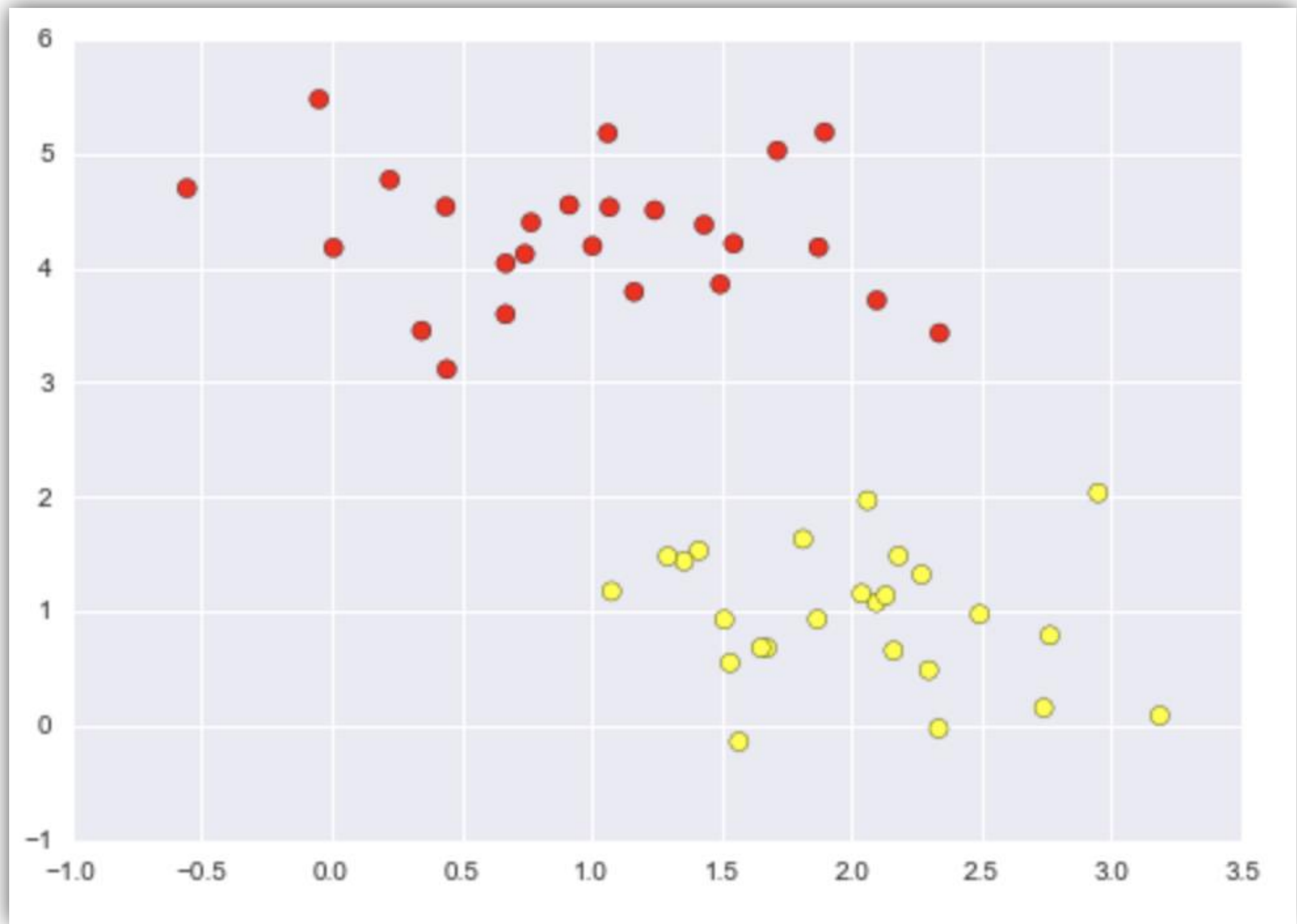
Support Vector Machine (SVM)

SVMs are powerful and flexible class of supervised algorithms
Classification
Regression
Discriminative classifier: Rather than modeling each class, simply find a line or curve or manifold that divides the classes from each other

Support Vector Machine (SVM)

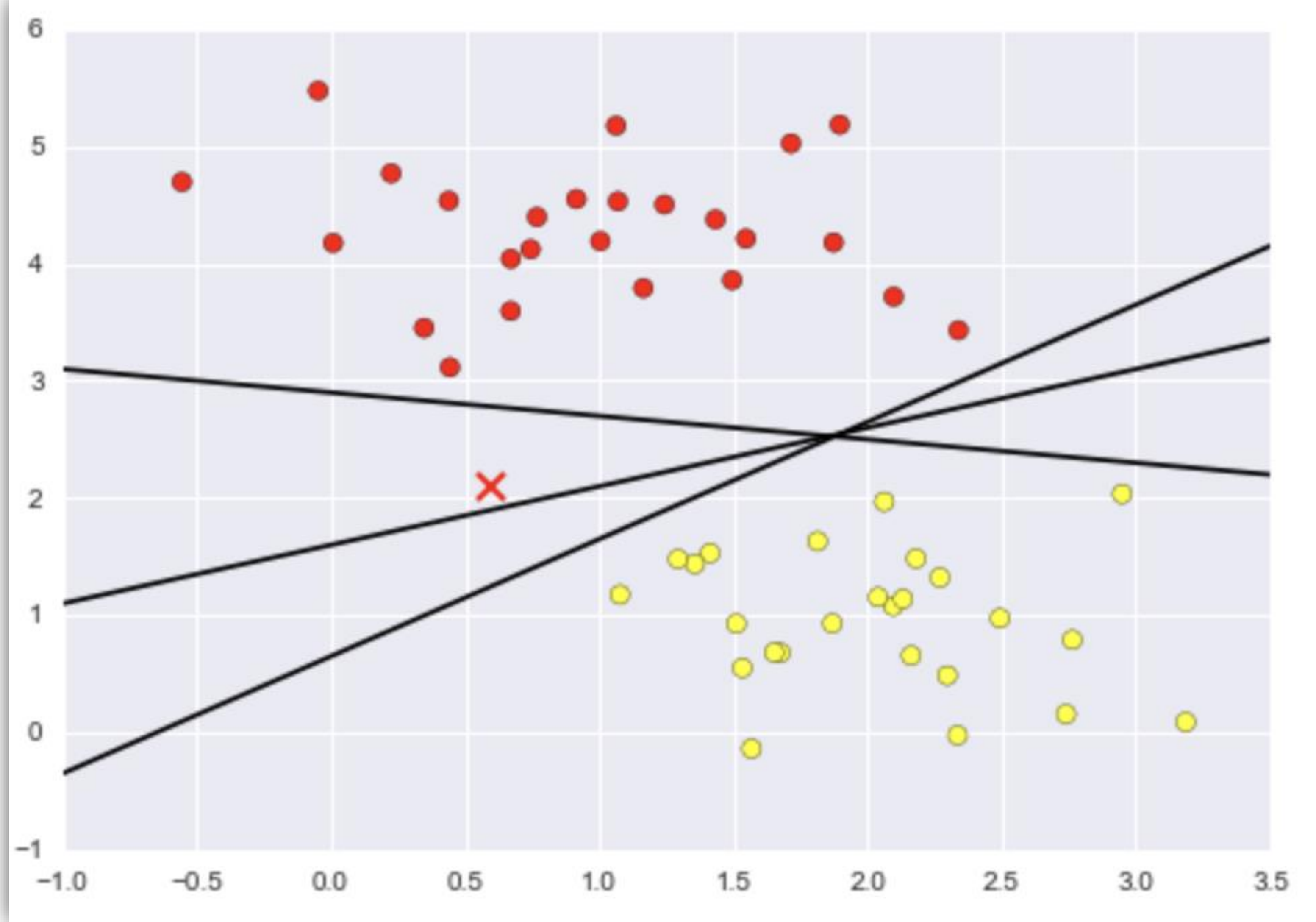
SVMs are powerful and flexible class of supervised algorithms

Lets draw a decision boundary



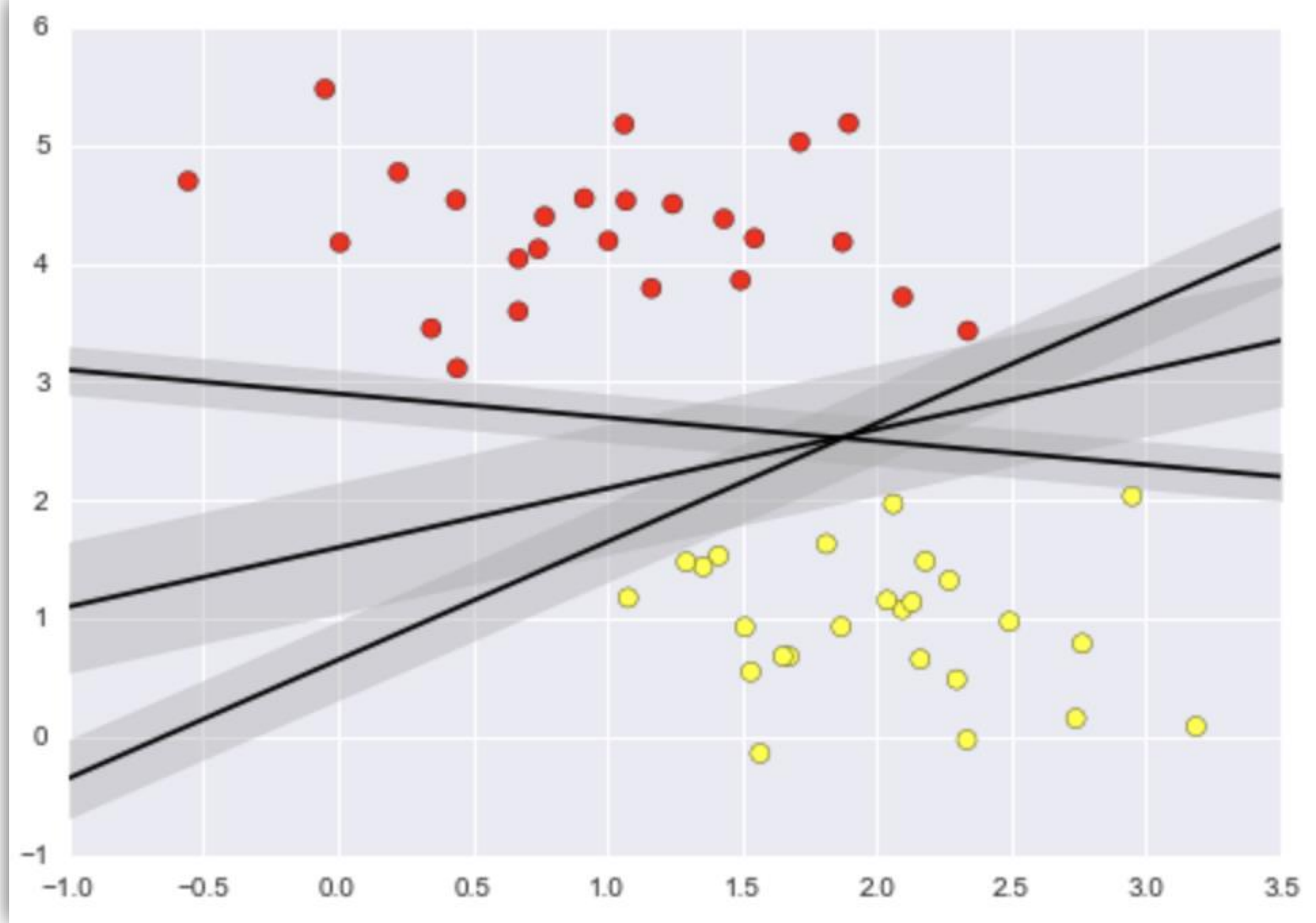
Support Vector Machine (SVM)

Which line to choose?



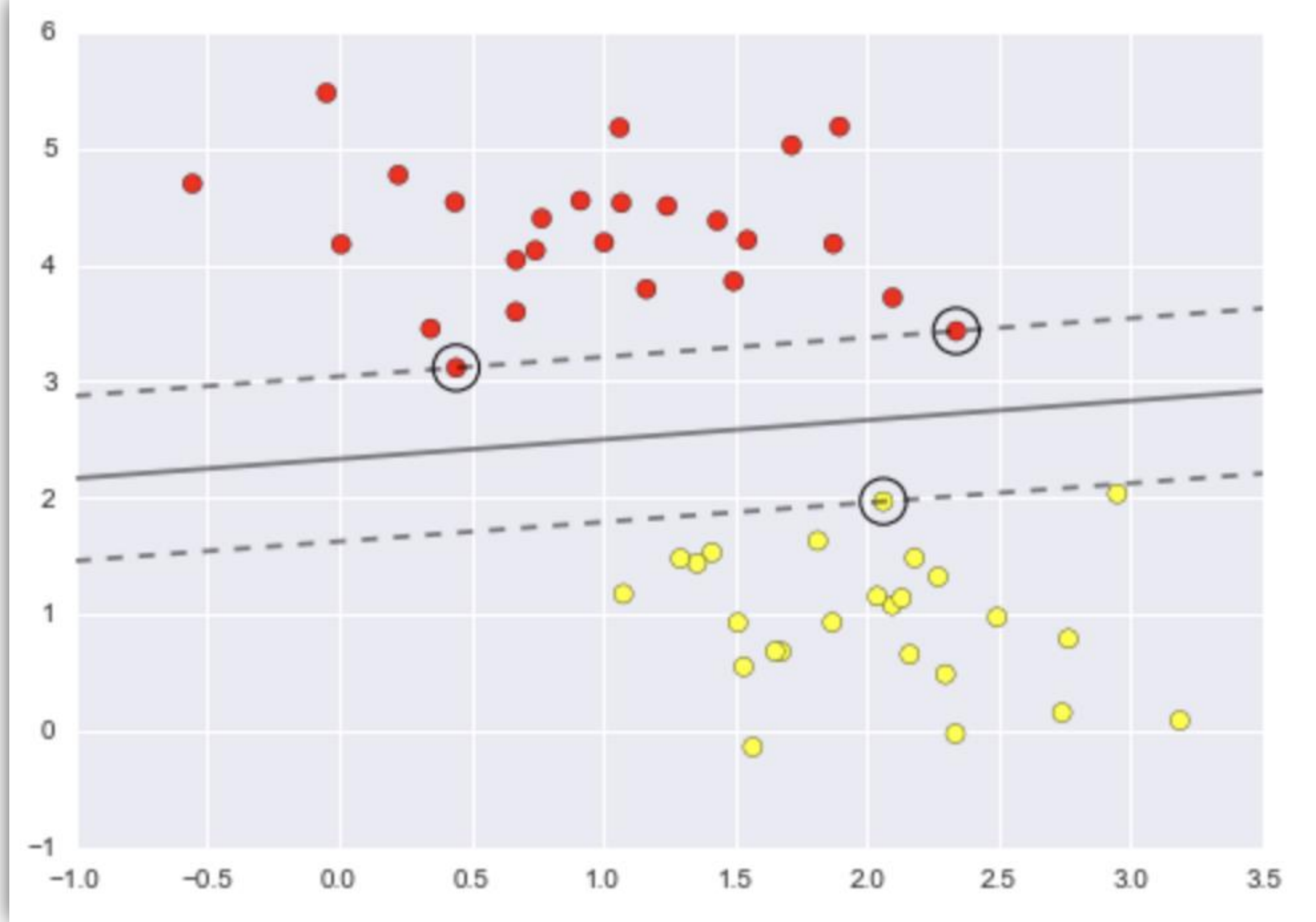
Support Vector Machine (SVM)

Margins



Support Vector Machine (SVM)

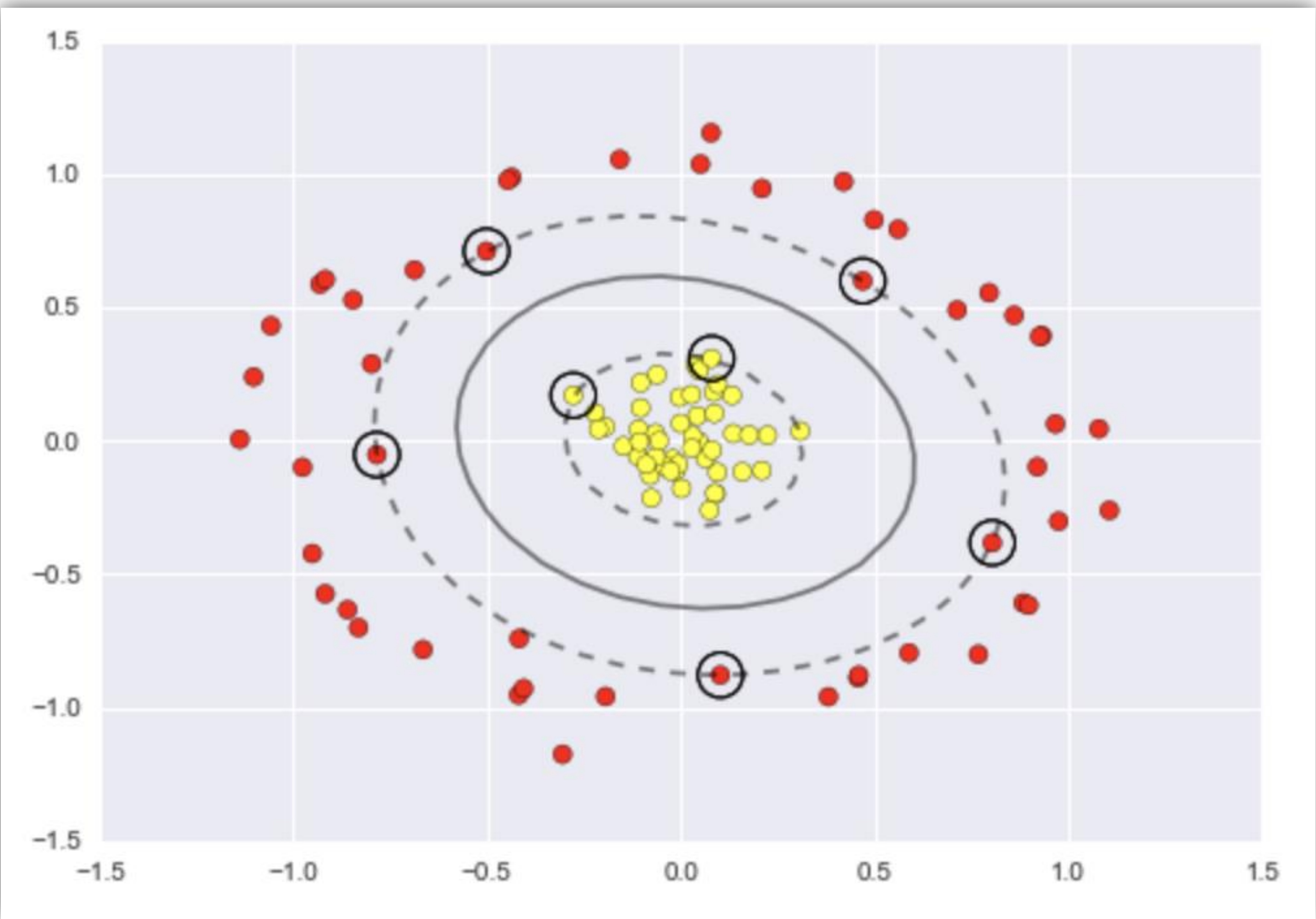
Support Vectors



Radial Basis Functions

Non-linear decision boundary

Kernel trick
https://en.wikipedia.org/wiki/Kernel_trick



Support Vector Machine (SVM)

```
from sklearn.svm import SVC
model = SVC(kernel='linear', C=1E10)
model.fit(X, y)
```

```
model.support_vectors_
```

```
array([[ 0.44359863,  3.11530945],
       [ 2.33812285,  3.43116792],
       [ 2.06156753,  1.96918596]])
```

Case Study: Face Recognition

Dataset: Sklearn built-in dataset

Notebook: ML02_SVM_Face_Recognition.ipynb

Decision Tree

Decision Tree

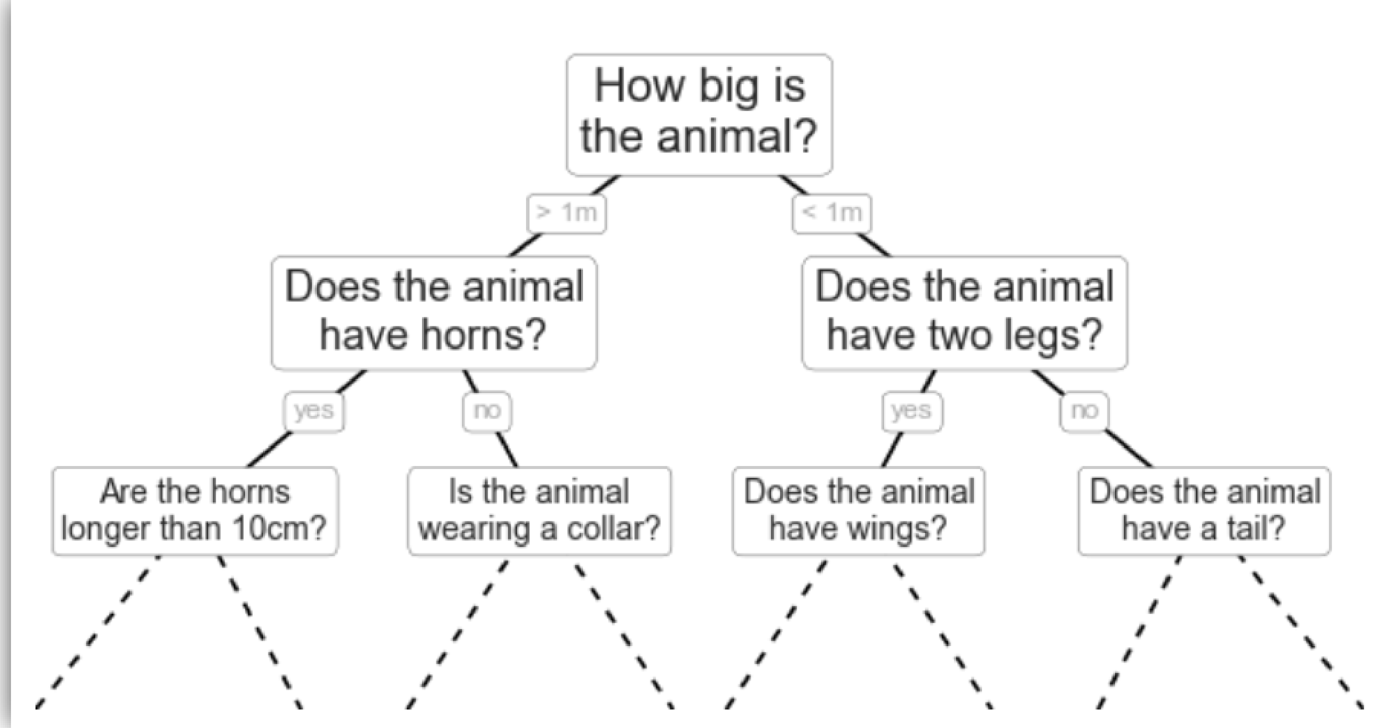
Non-parametric algorithm

Ask a series of questions designed to zero-in on the classification

Eg: To classify an animal, ask these questions to narrow down the decision process of classifying it

Each question will cut the number of options by half, quickly narrowing down the options.

Which questions to ask at each step???

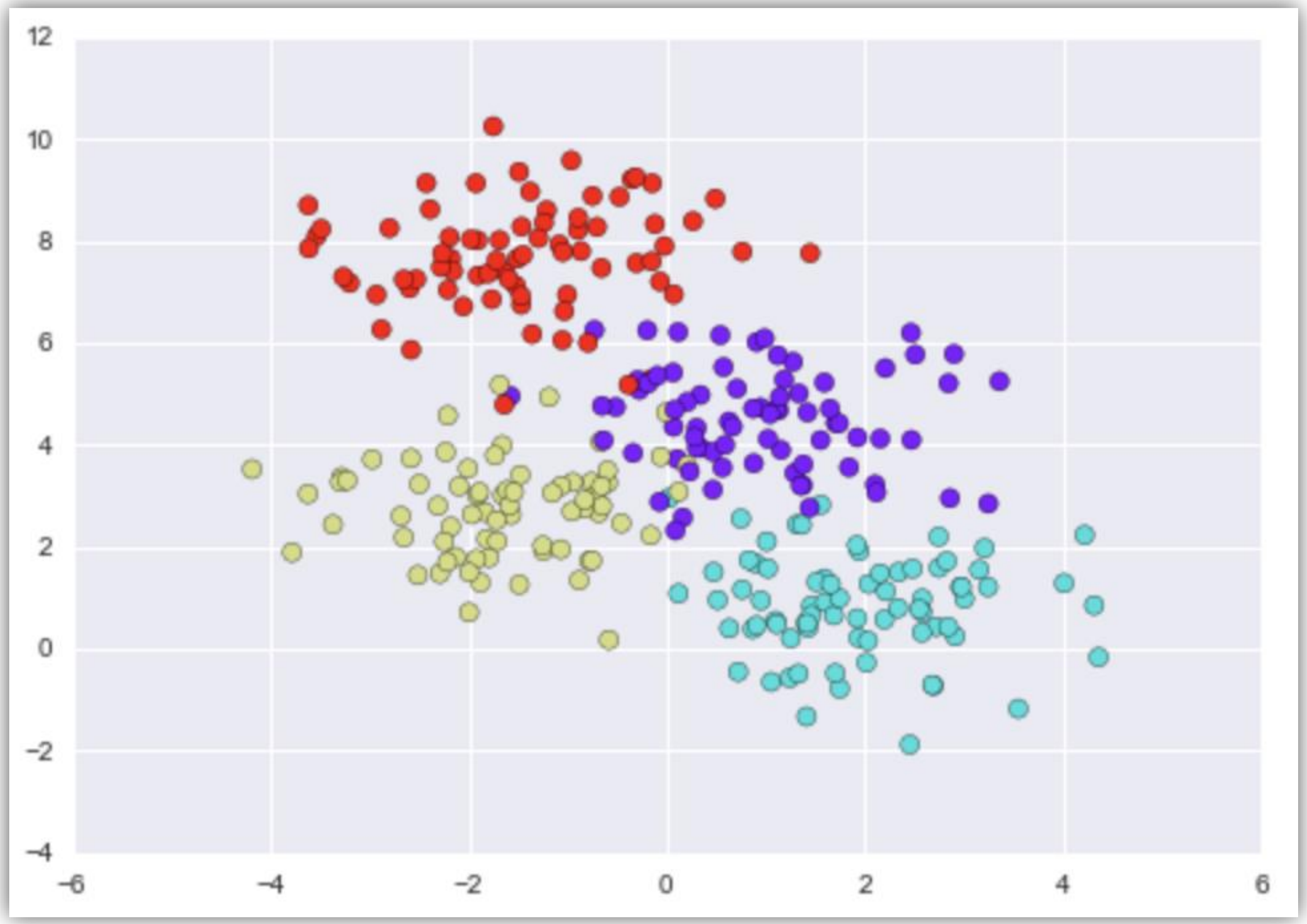


Decision Tree

Use make_blobs()

Iteratively split the data along one or the other axis according to some quantitative criterion

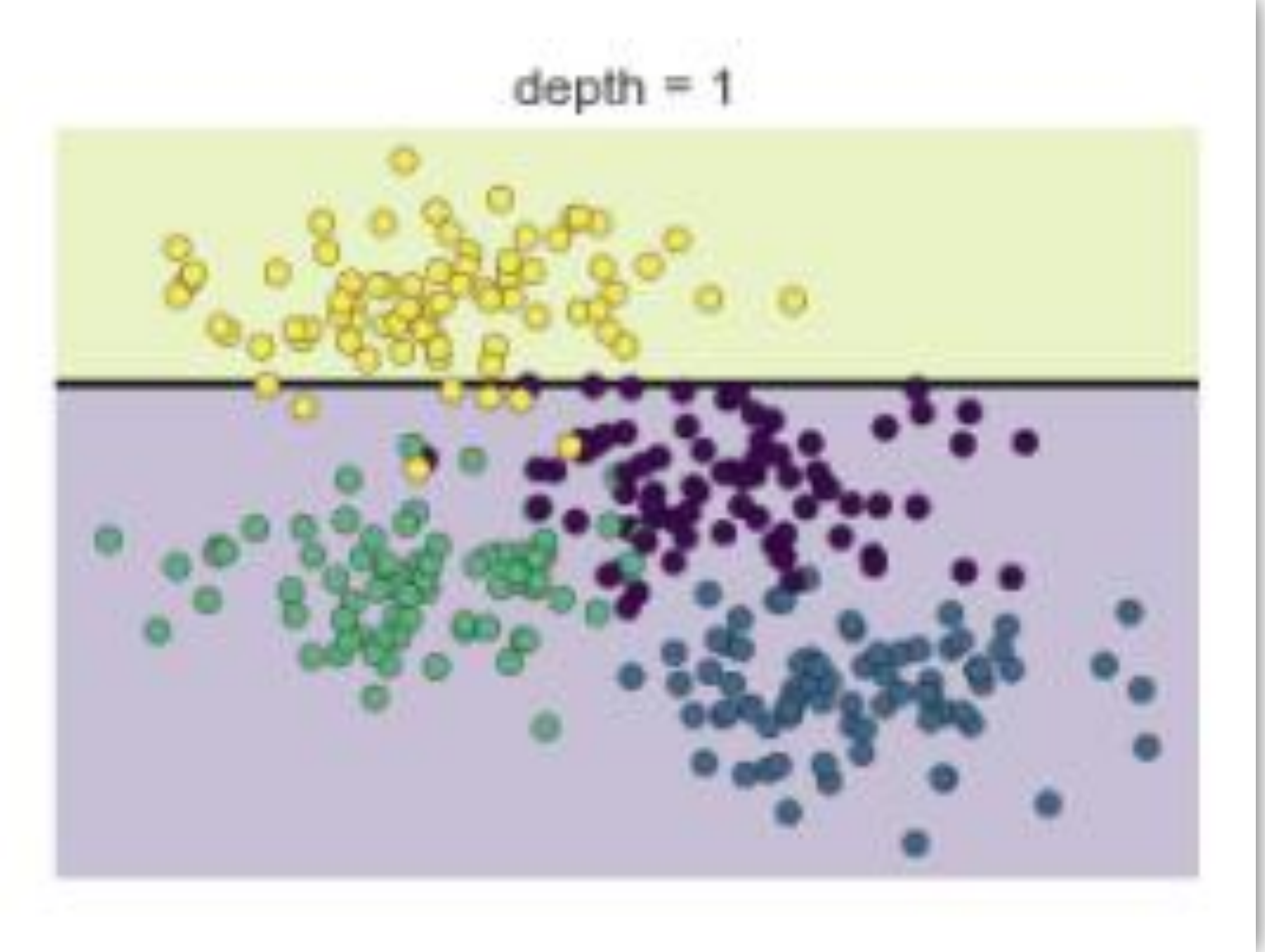
At each level assign the label of the new region according to a majority vote of points within it



Decision Tree

After first split, every point int the upper branch remains unchanged, so there is no need to further subdivide this branch

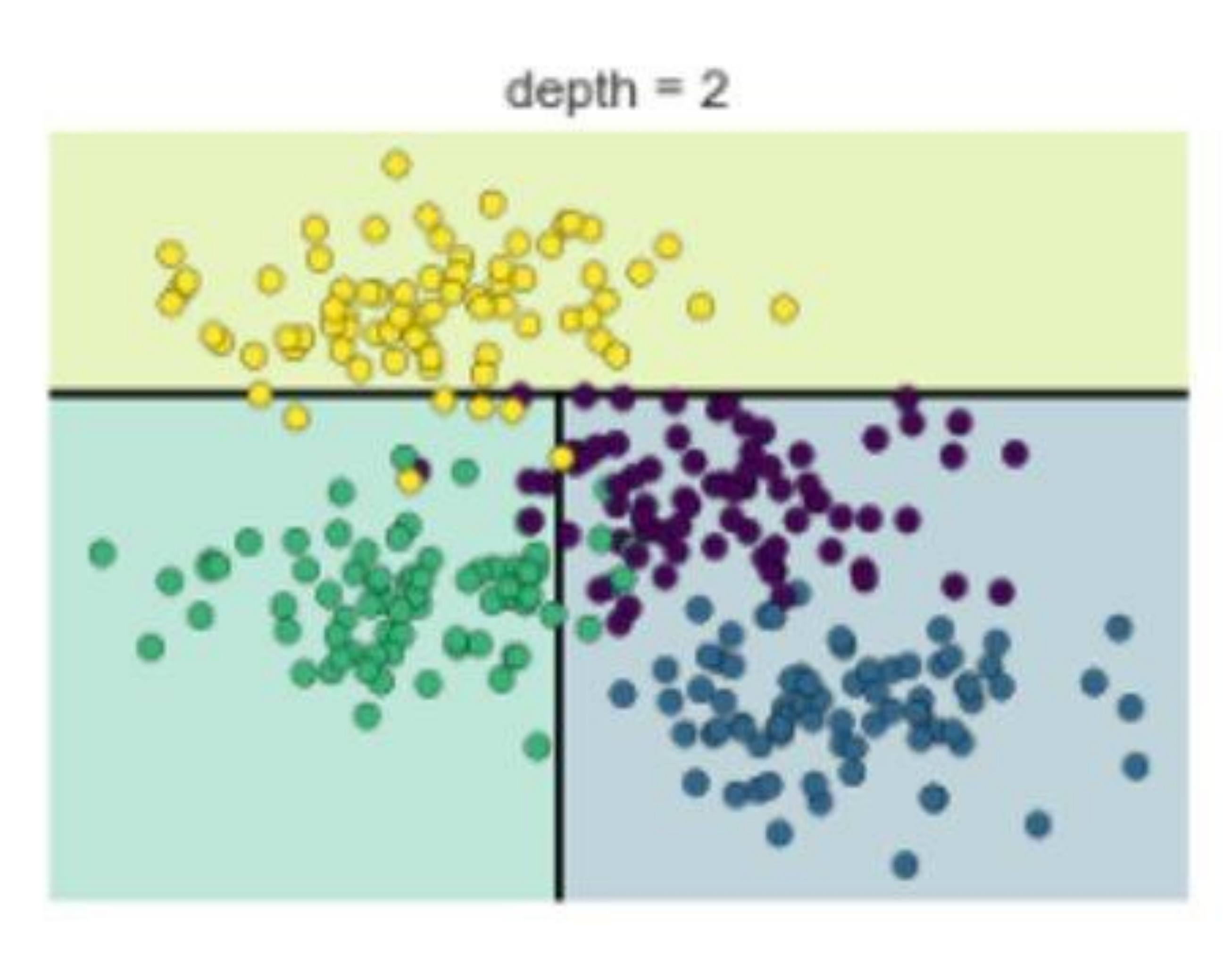
Only split further if all nodes are not of one colour (class)



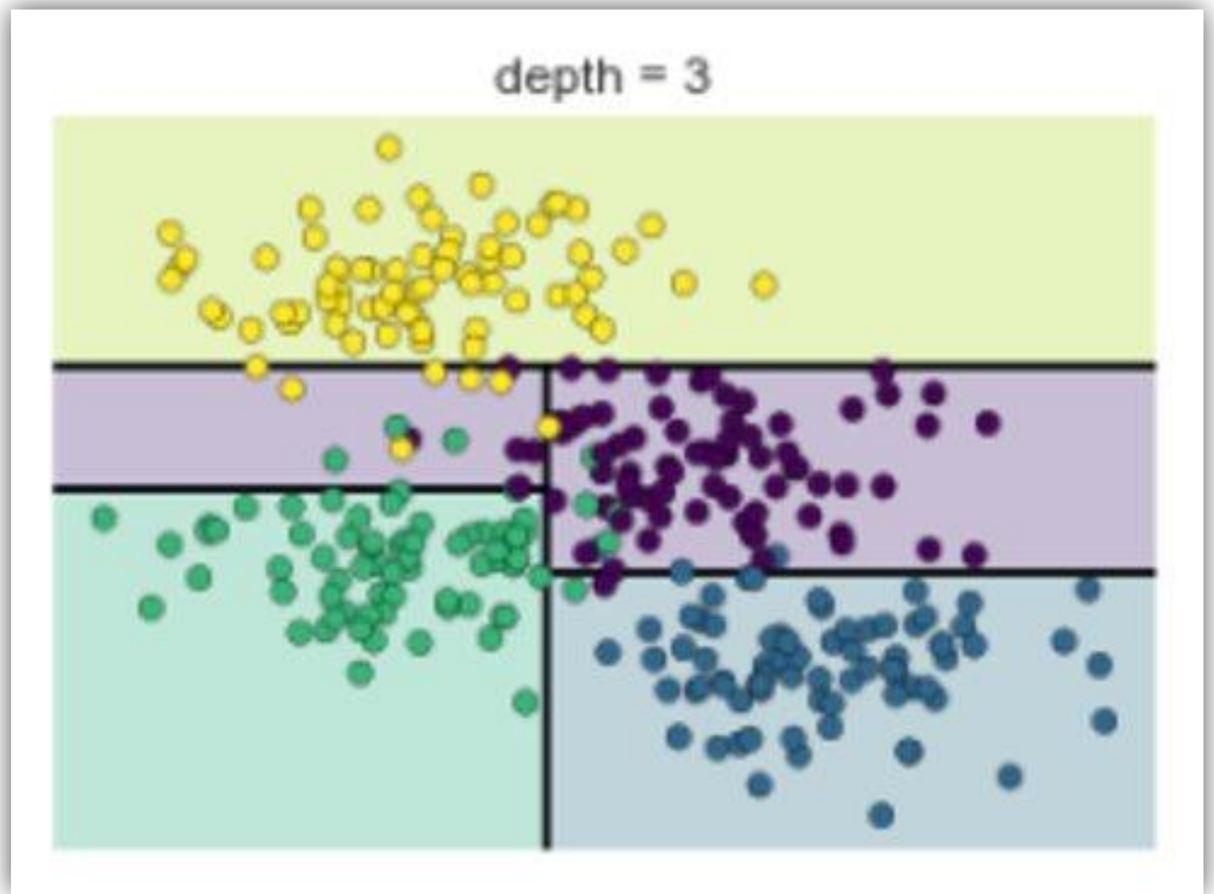
Decision Tree

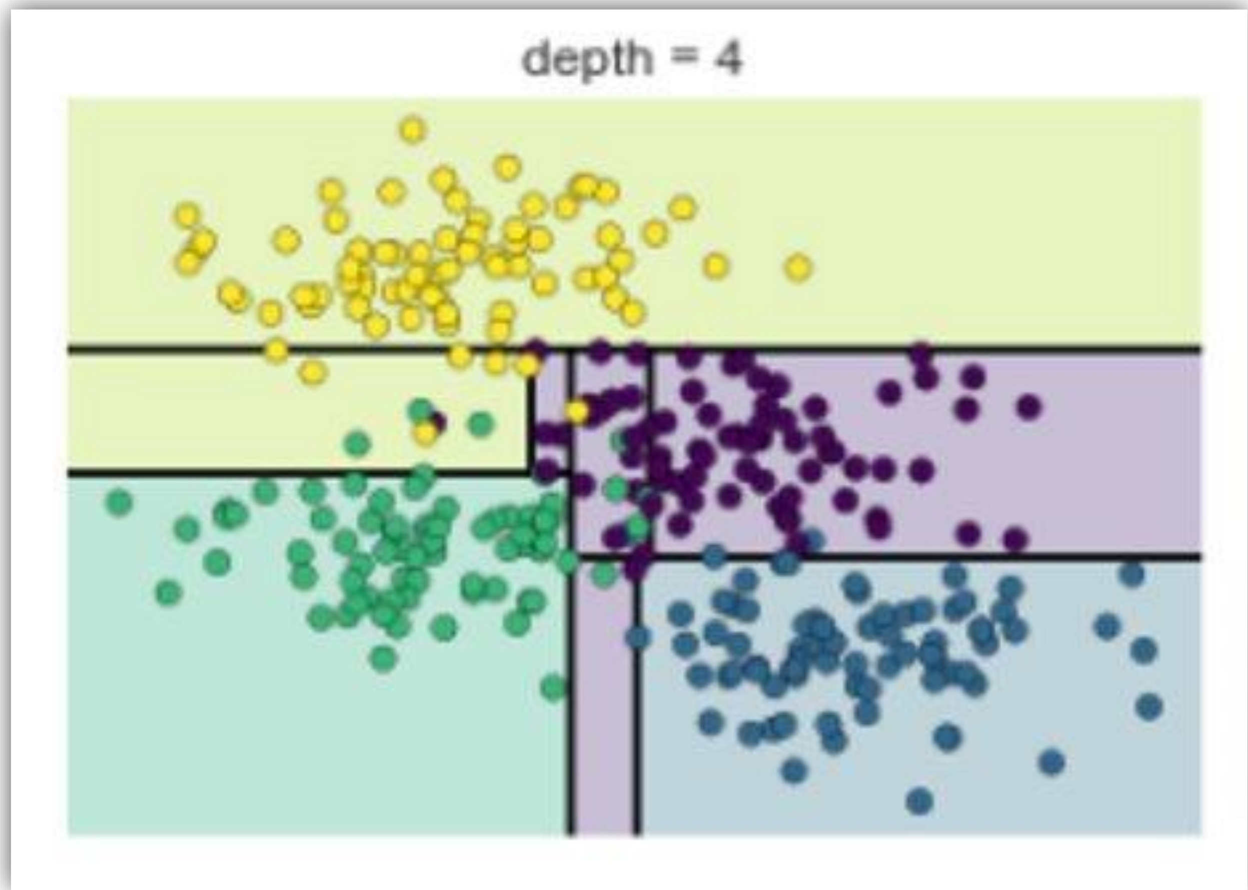
After second split, every point in the lower left branch remains unchanged, so there is no need to further subdivide this branch even though it has one or two outliers from other classes

Over-fitting happens when there is too much sub-divisions



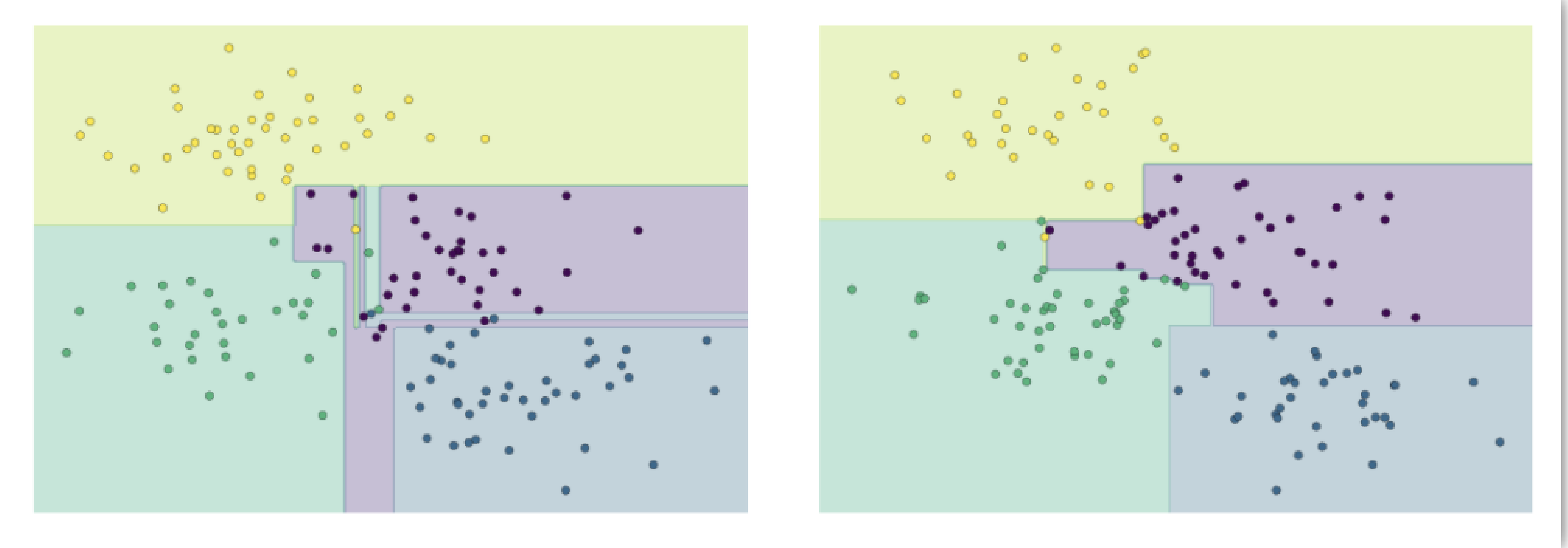
Decision Tree Splitting





Decision Tree – Over-fitting

Combining two trees' results would help !!



Case Study:

Dataset:

Notebook: ML03_Decision_Tree_Example.ipynb

Random Forests

Random Forests

Ensemble - Bagging Algorithm

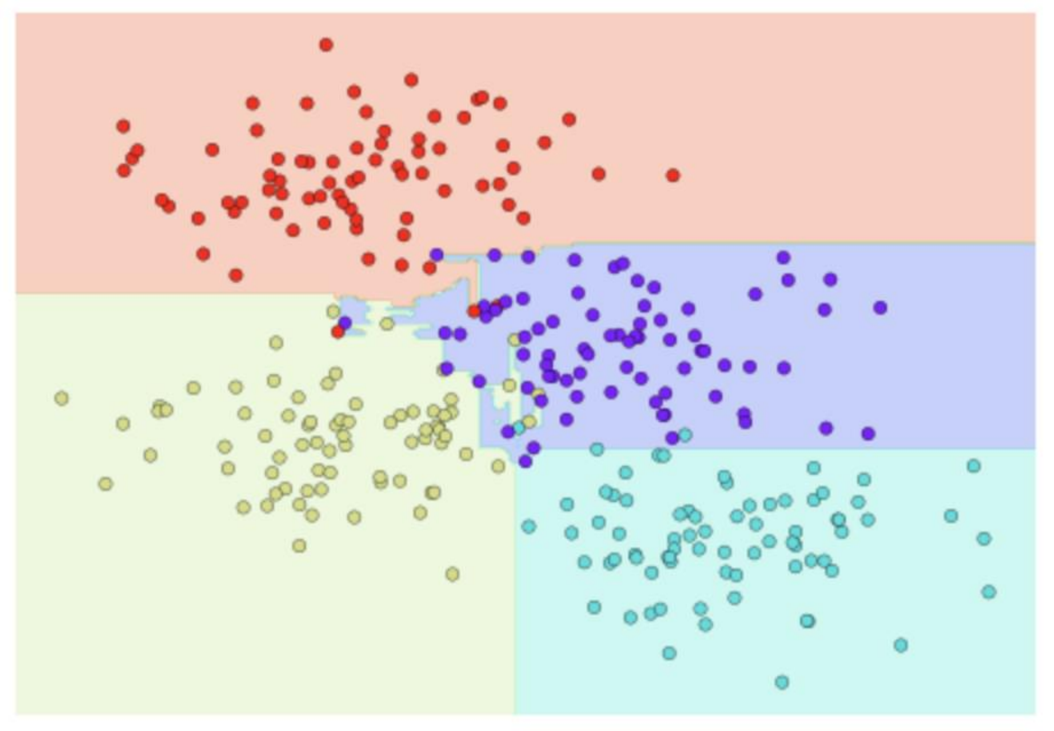
Multiple overfitting decision tree estimators can be combined to reduce the effect of overfitting

Ensemble of randomized decision trees

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier

tree = DecisionTreeClassifier()
bag = BaggingClassifier(tree, n_estimators=100, max_samples=0.8,
                        random_state=1)

bag.fit(X, y)
visualize_classifier(bag, X, y)
```

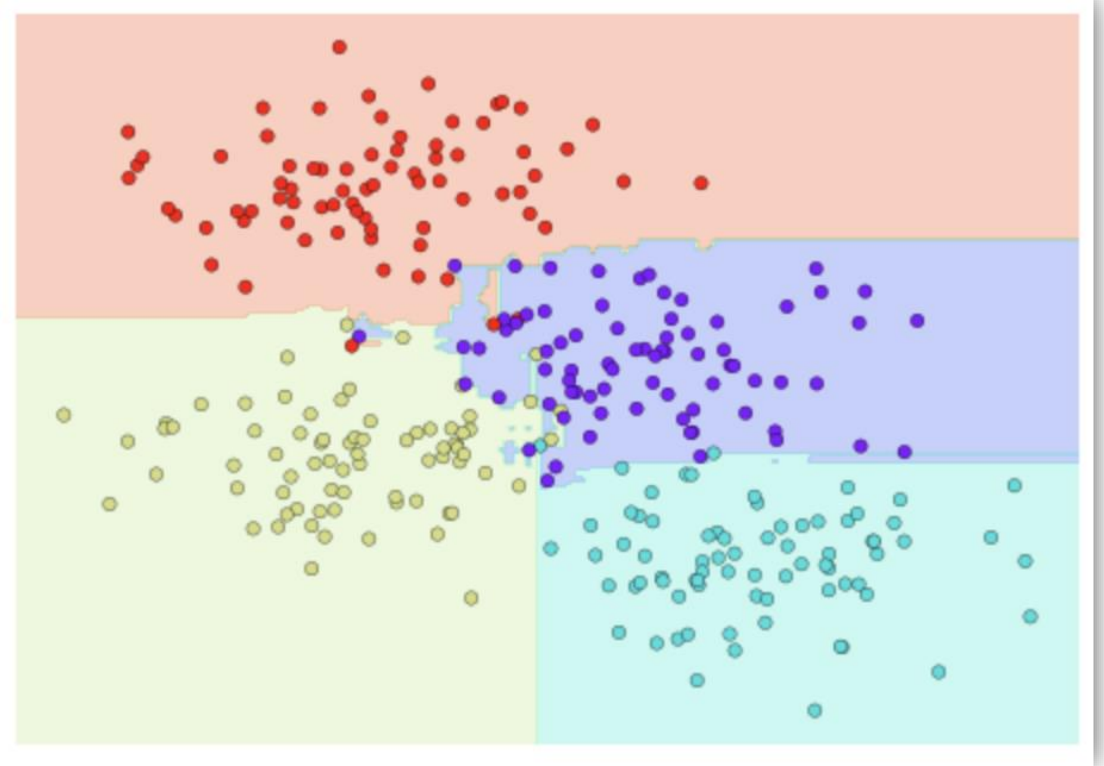


Random Forests

RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100, random_state=0)
visualize_classifier(model, X, y);
```



Random Forests

Fast Training

Fast Inference

Parallel computation can happen for decision trees

Explainability of the models is not that straight

Case Study: Best Random Forest

Dataset:

Notebook:
ML04_Random_Forests_Theory.ipynb

Assignment:
Task8_Best_Random_Forest.ipynb

K-Means

K-Means

Unsupervised algorithm

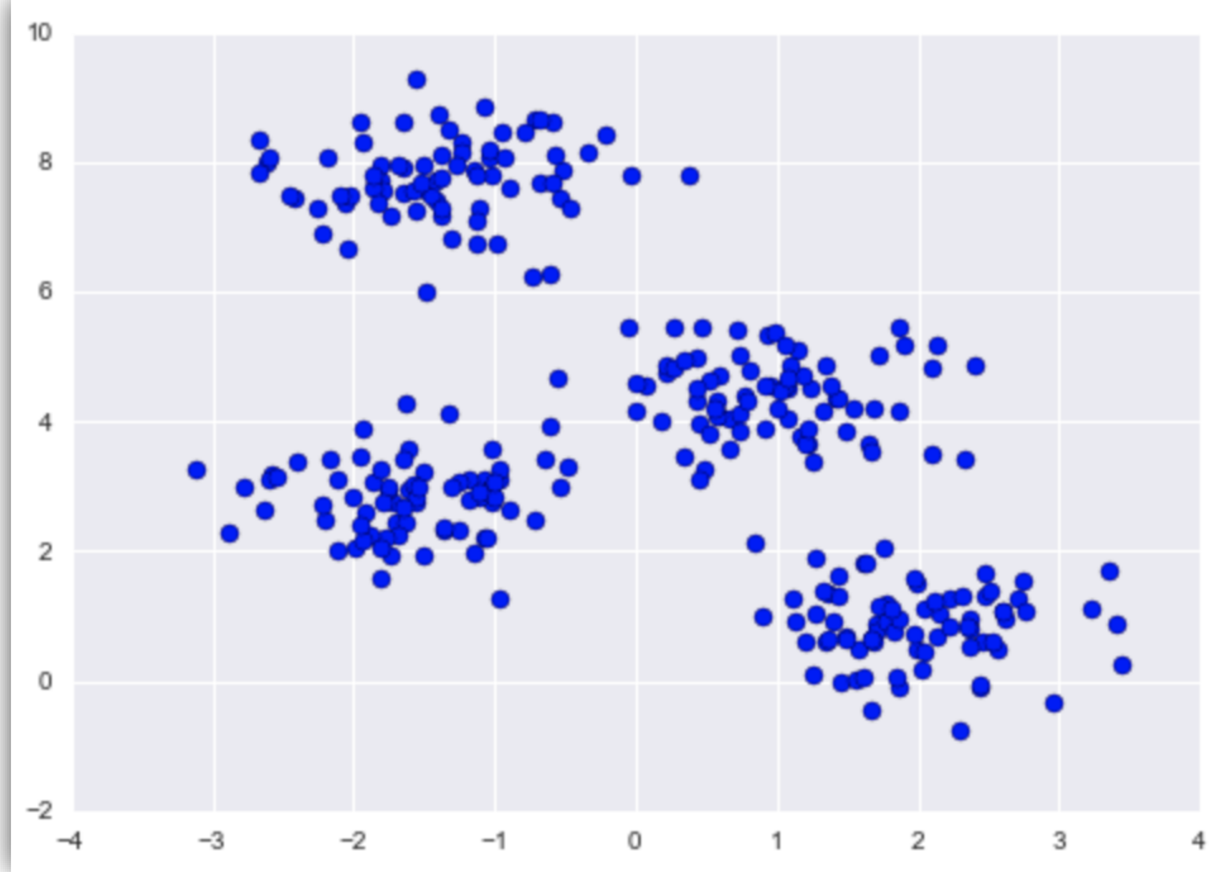
Searches for a pre-determined number of clusters within an unlabeled multidimensional dataset

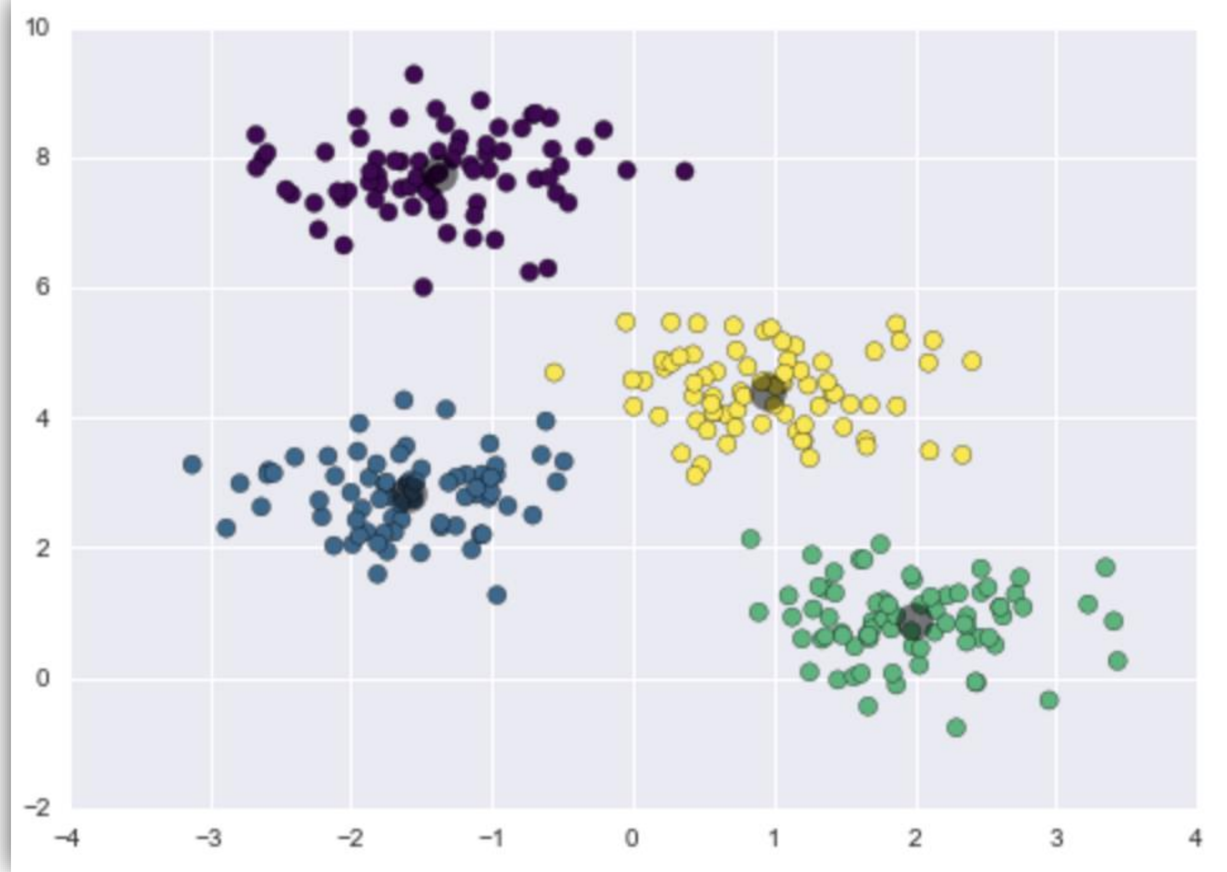
Two main assumptions

- Cluster center is arithmetic mean of all points belonging to the cluster
- Each point is closer to its own cluster center than other cluster centers

K-Means

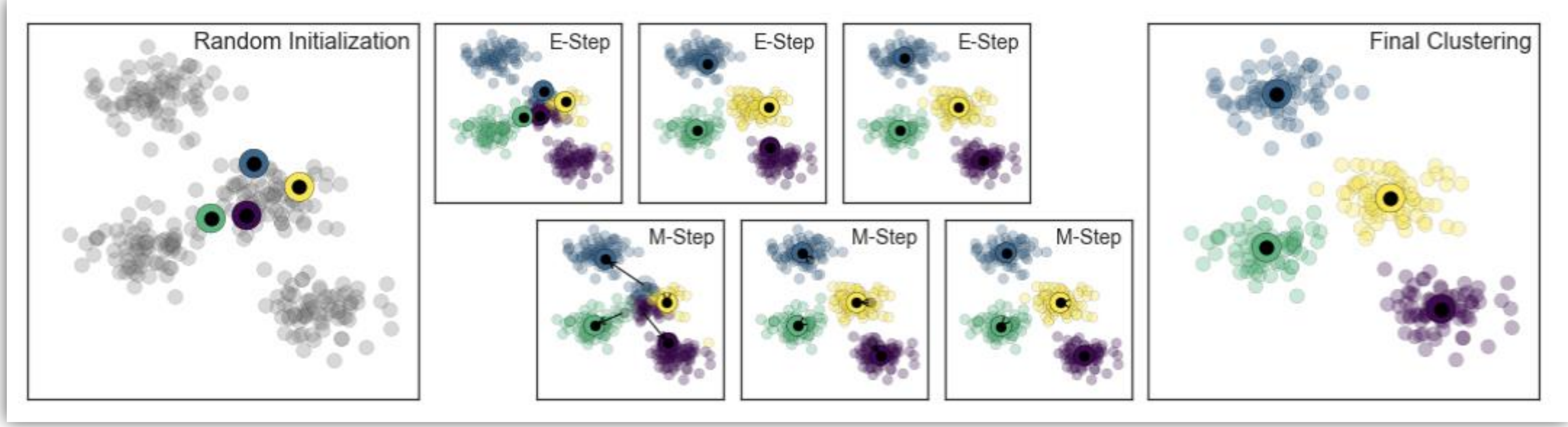
```python
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
```





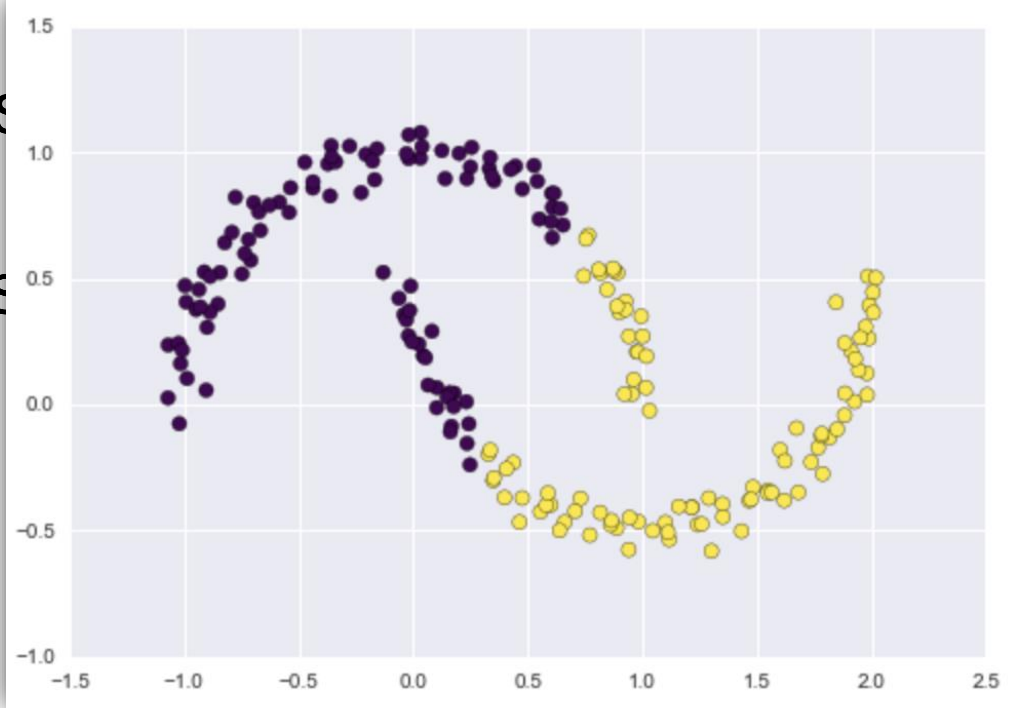
K-Means: Expectation-Maximization (EM)

1. Guess some cluster centers
2. Repeat until converged
    1. E-Step: assign points to the nearest cluster center
    2. M-Step: set the cluster centers to the mean



Weaknesses of K-Means

- Globally optimal result may not be achieved with different random seeds
  - Solution: Many runs
- Number of clusters should be selected beforehand
  - Solution: Silhouette Analysis
- Limited to only linear cluster boundaries
  - Solution: SpectralClustering, GMM
- Pretty slow for large number of samples
  - Solution: MiniBatchKMeans



Case Study: Handwritten Digits

Dataset:

Notebook: ML05_KMeans_Example.ipynb

Further Reading

Scikit-learn documentation

https://scikit-learn.org/

Slides and other references made from

Python Data Science Handbook by *Jake Vanderplas*