CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

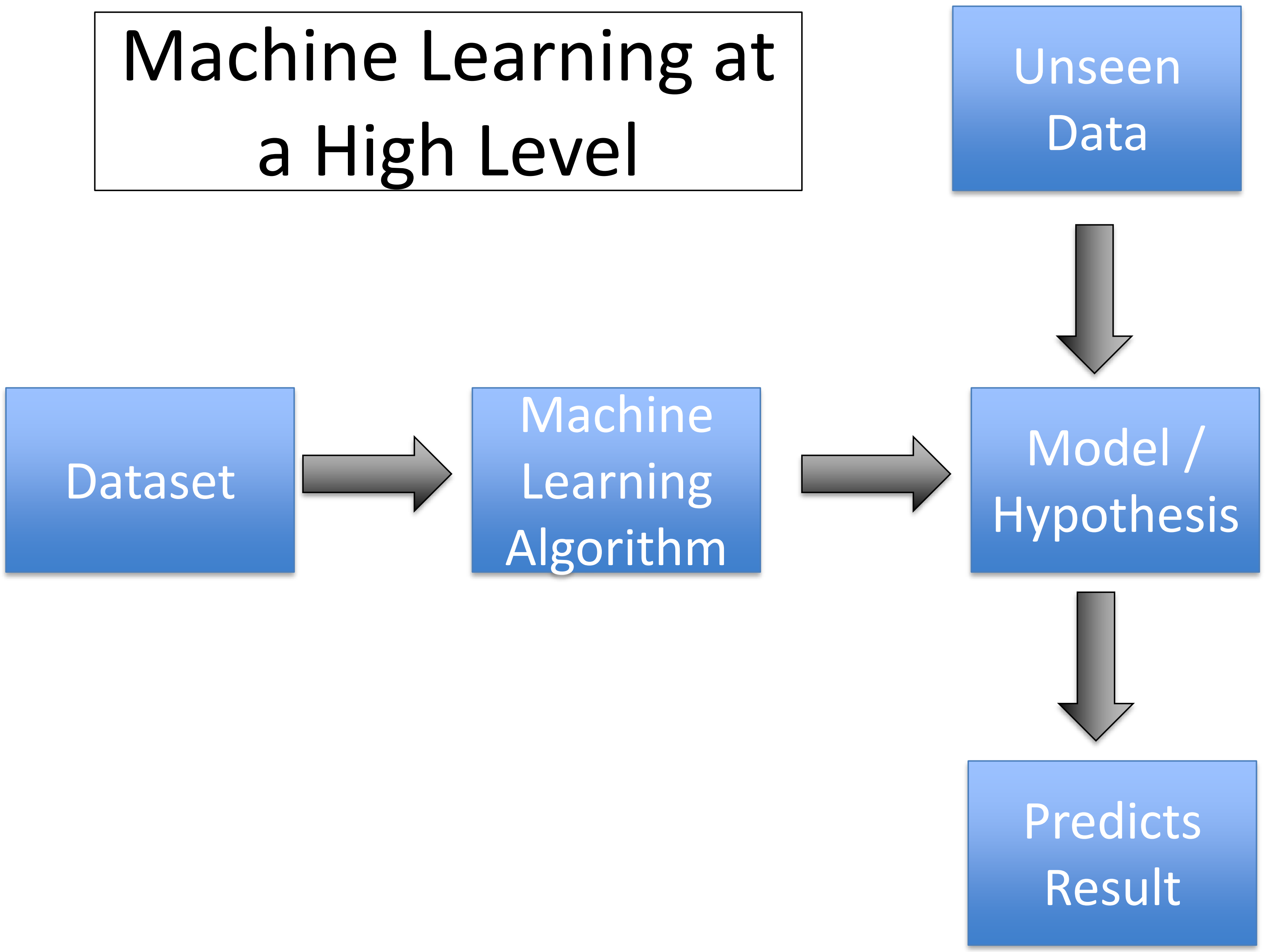
Machine Learning

Machine Learning

Lecture: A basic overview of Scikit-Learn

Ted Scully

Machine Learning at a High Level



Machine Learning Process

- The machine learning process is much more involved than the high level work-flow depicted in the previous slide.

- The stages can be broadly defined as follows:
  1. Data exploration
     - Understand the feature data (data types, missing values, outliers, etc)
     - Correlations between features and the target
     - Visualization (boxplots, scatter plots, correlation matrix, etc)
     - Feature engineering (aggregate features)

Machine Learning Process

- The machine learning process is much more involved than the high level work-flow depicted in the previous slide.

- The stages can be broadly defined as follows:

2. Data preparation
    - Dealing with outliers
    - Dealing with missing values
    - Feature encoding (encoding categorical features)
    - Feature selection
    - Feature scaling
    - Handling Imbalance

Machine Learning Process

- The machine learning process is much more involved than the high level work-flow depicted in the previous slide.

- The stages can be broadly defined as follows:

3. Building and Evaluating Models
    - Train many models from different categories (e.g., linear, naïve Bayes, SVM, kNN, decision trees, Random Forest, etc.) using standard parameters.
    - Measure and compare their performance.
    - Debug ML models and analyse the types of errors the models make.

4. Fine Tuning and Optimization
    - Perform hyper-parameter optimization
    - Incorporate transformation choices from part 2 as part of the hyper-parameter optimization
    - Try Ensemble methods
    - Finally assess the generalization capability of your model on the test set.

Machine Learning Process

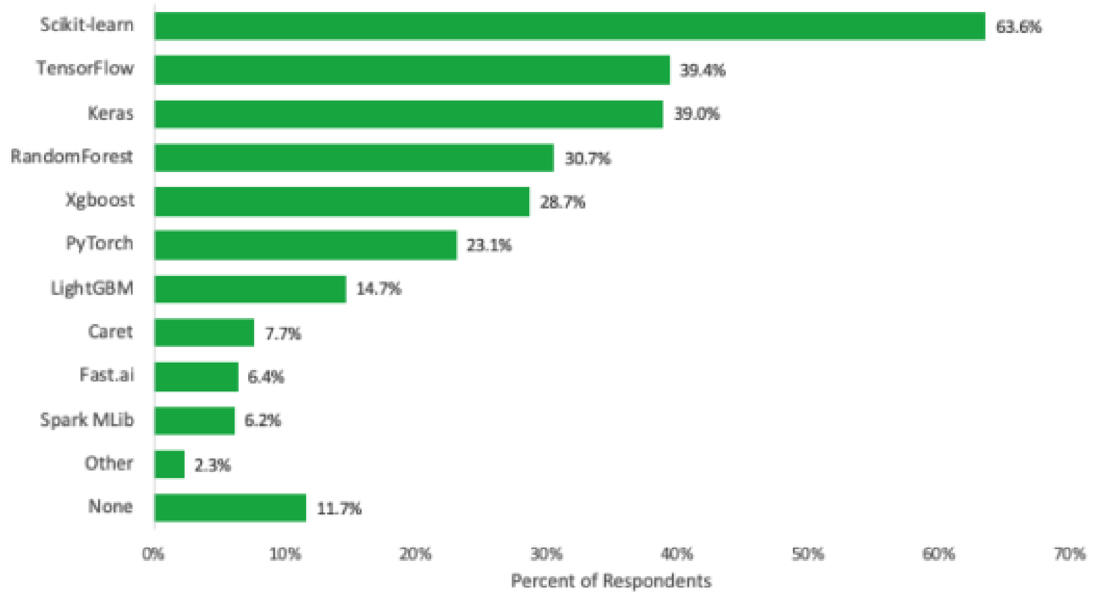
- We will be looking at the steps 2-4 outlined in the previous two slides using Scikit Learn.

- However, before we delve into this process in detail we will will first briefly introduce Scikit learn and how to use it to perform **basic classification and regression modelling**.

- Therefore, the structure of what we cover will be:
  - Introduction to Scikit Learn
  - Data Preparation
  - Build and Evaluating Models
  - Fine Tuning and Optimization

Part 1 - Introduction to SciKit Learn

- Scikit-learn provides a range of supervised and unsupervised learning algorithms in Python. The library is largely written in Python but makes extensive use of **NumPy**. It is also designed to be used easily with **Matplotlib** for visualization.
- The library is focused on modelling data. It also has a collection of functionality to support pre-processing steps.



- The most recent release was in May 2019 (scikit-learn 0.21.0). We will be using 0.21.x throughout this module.

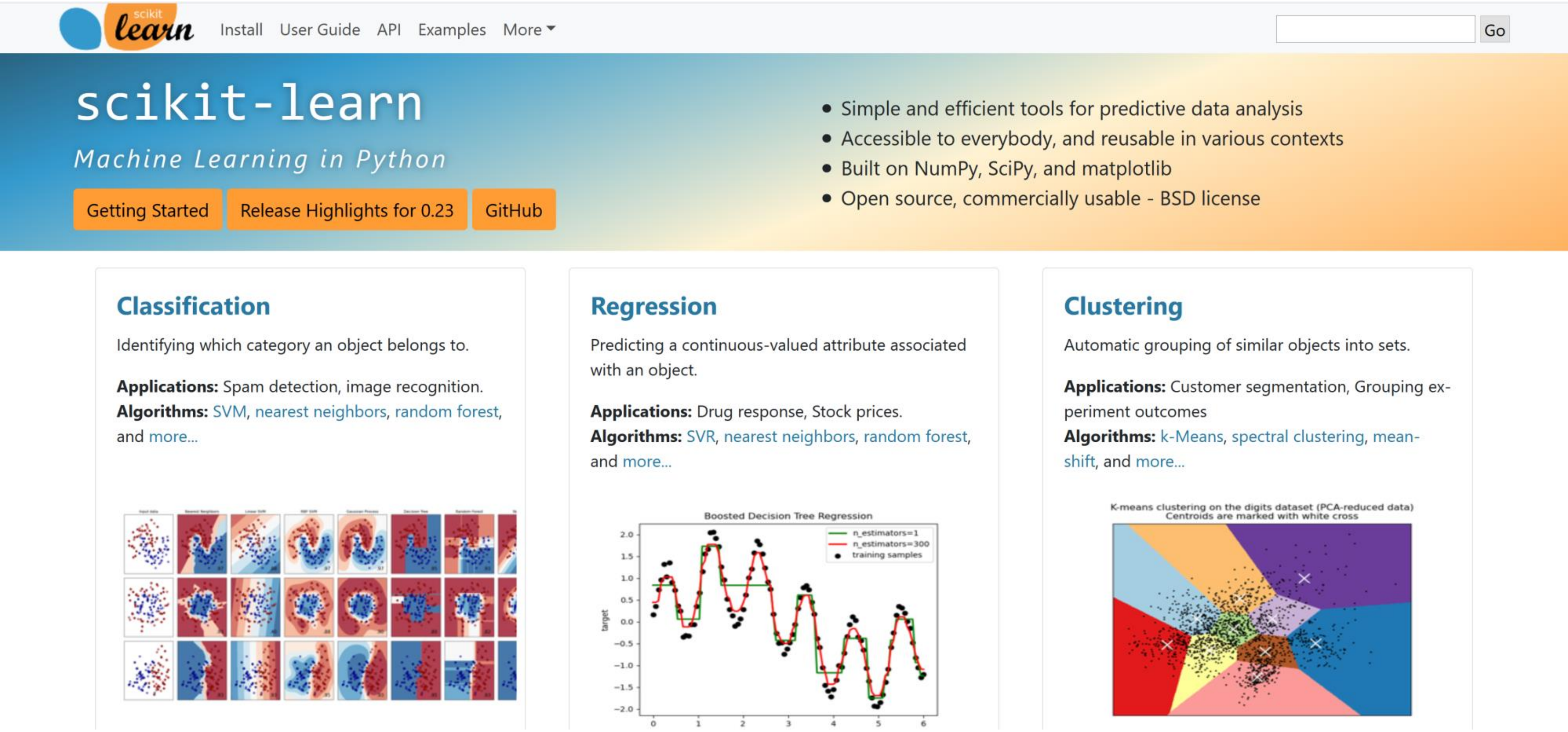
```
import sklearn
print(sklearn.__version__)
```

Introduction to Scikit Learn

- **Scikit Learn is well organized and there are extensive tutorials and API pages, which can be accessed here.**

- **The functionally offered by Scikit can be broken into the following :**

1. **Classification**: a large collection of learning algorithms such as naive bayes, kNNs, support vector machines, decision trees, ensembles, logistic regression etc.
2. **Clustering**: for grouping unlabelled data such as Kmeans, DBScan, etc.
3. **Regression**: libraries for predicting real-valued attributes such as multiple linear regression, ridge regression, etc.
4. **Pre-processing**: Outlier detection, normalization, encoding categorical features, feature selection, etc.
5. **Dimensionality Reduction**: Reduces the number of features that you need to consider in your dataset.
6. **Model Selection**: Cross- validation, metrics, hyper-parameter optimization.



scikit-learn

Machine Learning in Python

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...

Regression

Predicting a continuous-valued attribute associated with an object.

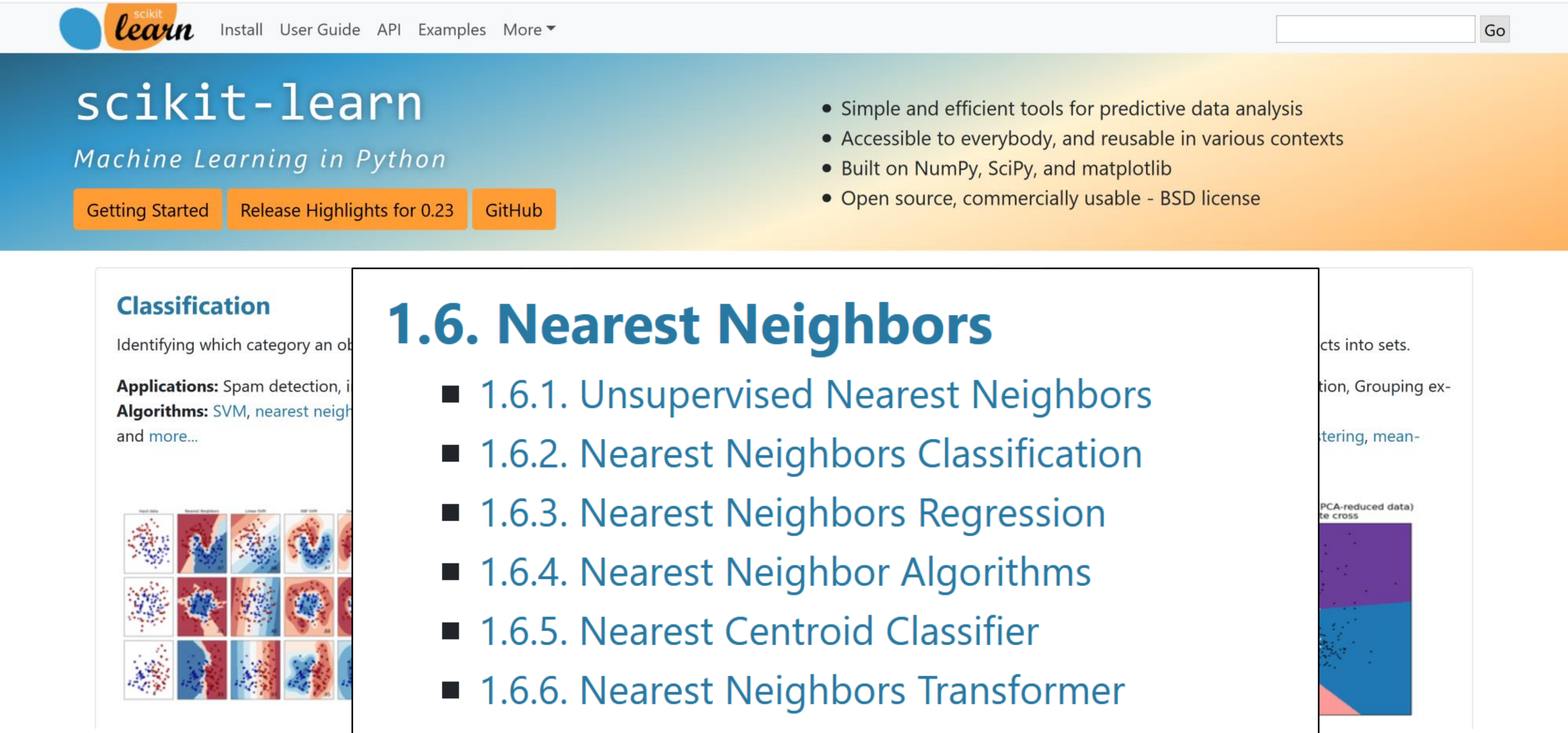
Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...

K-means clustering on the digits dataset (PCA-reduced data)
Centroids are marked with white cross



scikit-learn

Machine Learning in Python

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an ob

Applications: Spam detection, i
Algorithms: SVM, nearest neigh
and more...

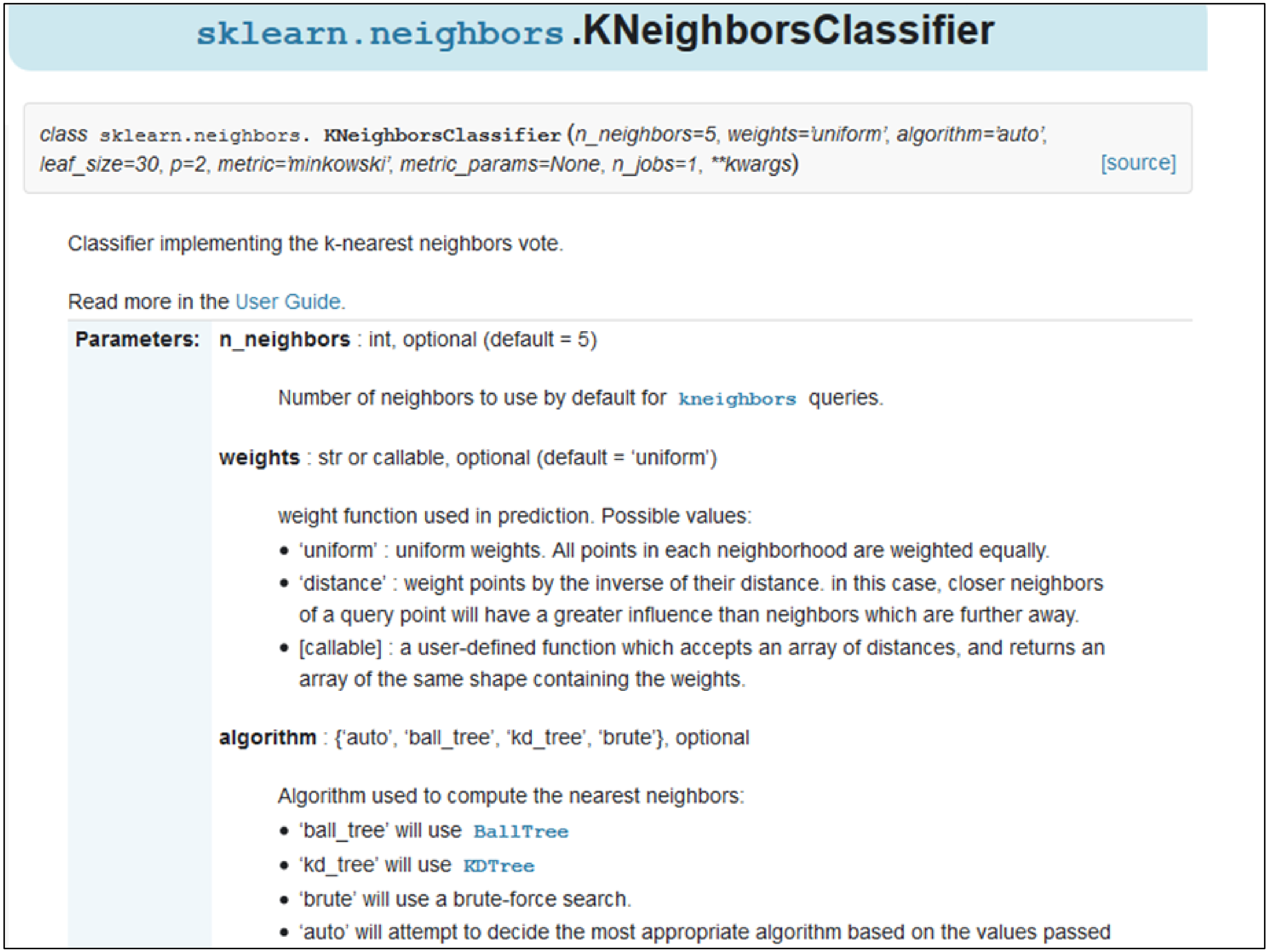
cts into sets.

tion, Grouping ex-

tering, mean-

1.6. Nearest Neighbors

- 1.6.1. Unsupervised Nearest Neighbors
- 1.6.2. Nearest Neighbors Classification
- 1.6.3. Nearest Neighbors Regression
- 1.6.4. Nearest Neighbor Algorithms
- 1.6.5. Nearest Centroid Classifier
- 1.6.6. Nearest Neighbors Transformer
- 1.6.7. Neighborhood Components Analysis



sklearn.neighbors.KNeighborsClassifier

class `sklearn.neighbors.` **`KNeighborsClassifier`**(*n_neighbors=5*, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=1*, ***kwargs*) [source]

Classifier implementing the k-nearest neighbors vote.

Read more in the User Guide.

Parameters: **n_neighbors** : int, optional (default = 5)

Number of neighbors to use by default for `kneighbors` queries.

weights : str or callable, optional (default = 'uniform')

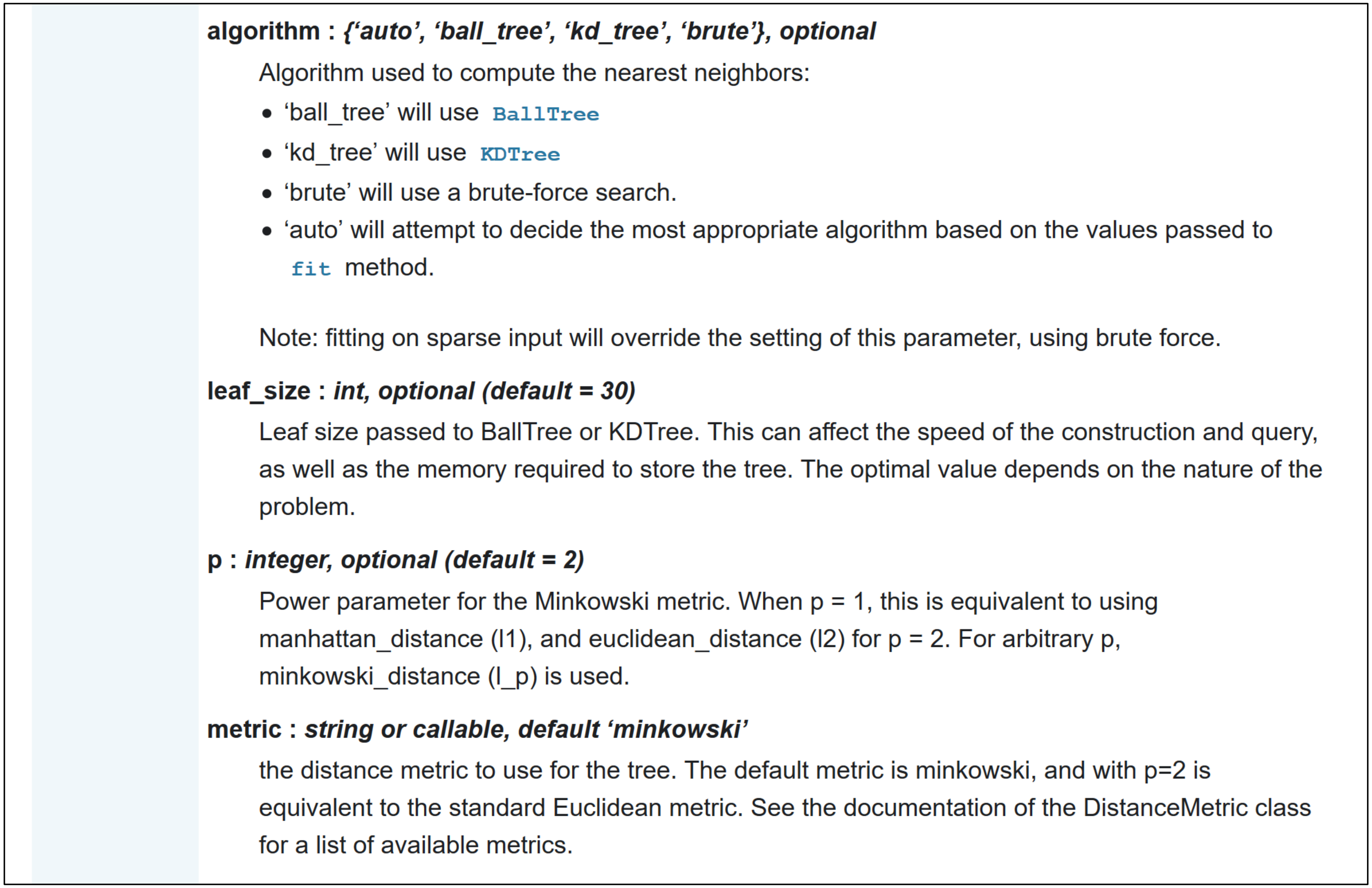
weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

algorithm : {'auto', 'ball_tree', 'kd_tree', 'brute'}, optional

Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use `BallTree`
- 'kd_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed



algorithm : *{‘auto’, ‘ball_tree’, ‘kd_tree’, ‘brute’}, optional*

Algorithm used to compute the nearest neighbors:

- ‘ball_tree’ will use `BallTree`
- ‘kd_tree’ will use `KDTree`
- ‘brute’ will use a brute-force search.
- ‘auto’ will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

Note: fitting on sparse input will override the setting of this parameter, using brute force.

leaf_size : *int, optional (default = 30)*

Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

p : *integer, optional (default = 2)*

Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

metric : *string or callable, default ‘minkowski’*

the distance metric to use for the tree. The default metric is minkowski, and with p=2 is equivalent to the standard Euclidean metric. See the documentation of the DistanceMetric class for a list of available metrics.

A few important notes about Scikit Learn

- The following are some important requirements that you should keep in mind when working with Scikit learn.

  1. Features and classes/target values are **separate** objects (data structures)
  2. Features and classes/ target values should be **numerical**
  3. Features and classes/ target values should be **NumPy** arrays
  4. Features and classes should have a **specific shape**
     - Features should be 2D (Columns correspond to numbers of features and rows are number of data instances)
     - Class array or regression target values should be one dimensional with same number of instances as there are data instances in the features array

```
import numpy as np


dataset = np.genfromtxt("training.csv",
delimiter=',’)

features = dataset[:, :-1]

labels = dataset[:, -1]
```

Using Datasets

- Scikit-learn comes with a number of standard example datasets. These are broken into toy datasets and real-world datasets.

- Toy datasets include the iris dataset and digits datasets for classification and the Boston house prices dataset for regression.
- Real-world datasets include Olivetti faces dataset, newsgroups, California housing dataset, etc.

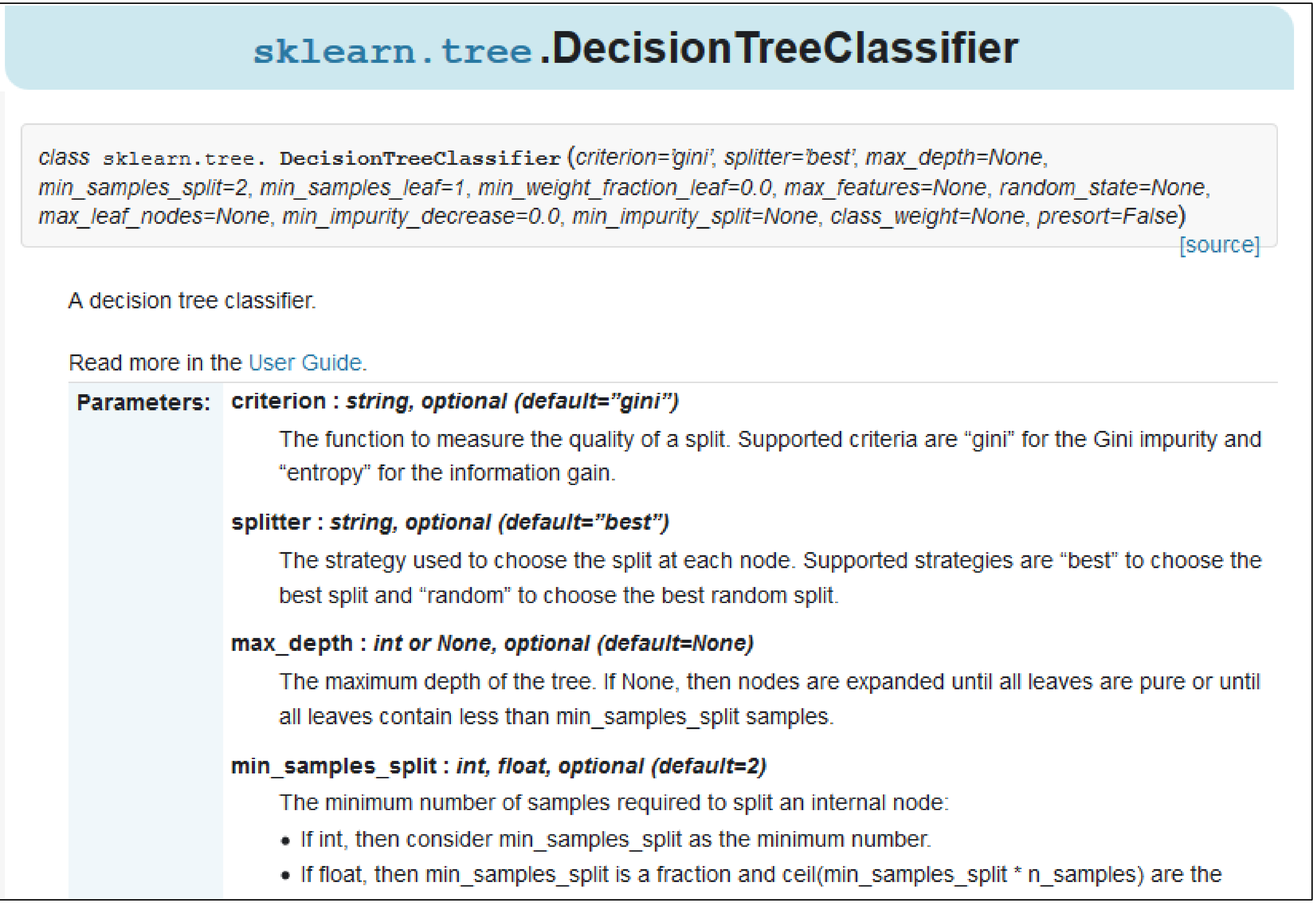
- These datasets are **dictionary-like** objects holding at least two items:
  - A NumPy array of shape *n_samples* * *n_features* with the key ***data***
  - A NumPy array of length *n_samples*, containing the class values, with key ***target***

```
from sklearn import datasets

iris = datasets.load_iris()

print ( iris.data.shape )
print (iris.target.shape)
```

Outputs the dimensions of the data (150, 4) and labels (150,)

Using a Decision Tree Classifier in SciKit Learn

- Decision trees are a family of supervised learning methods (estimators) used for classification and regression.
- The class we will be using is **sklearn.tree.DecisionTreeClassifier**
- The API documentation can be found at **DecisionTreeClassifer.**



sklearn.tree.DecisionTreeClassifier

class `sklearn.tree.DecisionTreeClassifier`(*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False*) [source]

A decision tree classifier.

Read more in the User Guide.

Parameters:

criterion : *string, optional (default="gini")*

The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

splitter : *string, optional (default="best")*

The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth : *int or None, optional (default=None)*

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : *int, float, optional (default=2)*

The minimum number of samples required to split an internal node:

- If int, then consider min_samples_split as the minimum number.
- If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the

Building a Decision Tree

```
from sklearn import tree
from sklearn import datasets

iris = datasets.load_iris()

# creates a new decision tree object classifier
clf = tree.DecisionTreeClassifier()

# train the classifier pass it the training data and classes
clf.fit(iris.data, iris.target)

# predict the class for an unseen example
print (clf.predict( [[4.5, 2.9, 3.0, 2.0]]  ))
```

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
-- Iris Setosa
-- Iris Versicolour
-- Iris Virginica

When we run this code it will predict the output for this unseen instance as being 2, which corresponds to virginica

Using kNN on Iris Dataset

- In the next example we will apply k-nearest neighbour to the iris dataset.

```
from sklearn import datasets
from sklearn import neighbors

iris = datasets.load_iris()


knn = neighbors.KNeighborsClassifier(n_neighbors = 8)

knn.fit(iris.data, iris.target)

print (knn.predict([[3, 5, 4, 2]]))
```

We can provide many different parameters to machine learning algorithms in Scikit Learn. However, most have default values allows us to get started very quickly.

Basics of using Scikit Learn

- In this section we are just focused on getting up and running with the basic of Scikit Learn.

- In the following slides we look at two separate basic scenarios for evaluation (we will cover more <u>**advanced**</u> and <u>**realistic**</u> techniques in a later):

1. There is a separate **training** and **test** dataset that can be used.

2. A **single** dataset split into a training and test data (holdout method).

Basics of using Scikit Learn

1. There is a separate **training** and **test** dataset that can be used.

f1	f2	f3	…	fn	class

Training Set

f1	f2	f3	…	fn	class

Testing Set

```
from sklearn import metrics
from sklearn import tree

train = np.genfromtxt("trainingData.csv", delimiter=',')
test = np.genfromtxt("testData.csv", delimiter=‘,’)

features_train = train[:, :-1]
labels_train = train[:, -1]

features_test = test[:, :-1]
labels_test = test[:, -1]

# creates a new decision tree object classifier
clf = tree.DecisionTreeClassifier()

# trains the classifier pass it the training data and classes
clf = clf.fit(features_train, labels_train)

# predict the class for an unseen example
results= clf.predict(features_test)

print (   metrics.accuracy_score(results, labels_test)   )
```

Notice we start with two separate datasets. A train dataset and a test dataset. We have to split each into feature and class labels. Notice when we call predict we are passing it the test feature data (a 2D NumPy array).

```
from sklearn import metrics
from sklearn import tree

train = np.genfromtxt("trainingData.csv", delimiter=',')
test = np.genfromtxt("testData.csv", delimiter=‘,’)

features_train = train[:, :-1]
labels_train = train[:, -1]

features_test = test[:, :-1]
labels_test = test[:, -1]

# creates a new decision tree object classifier
clf = tree.DecisionTreeClassifier()

# trains the classifier pass it the training data and classes
clf = clf.fit(features_train, labels_train)

# predict the class for an unseen example
results= clf.predict(features_test)

print (   metrics.accuracy_score(results, labels_test)   )
```

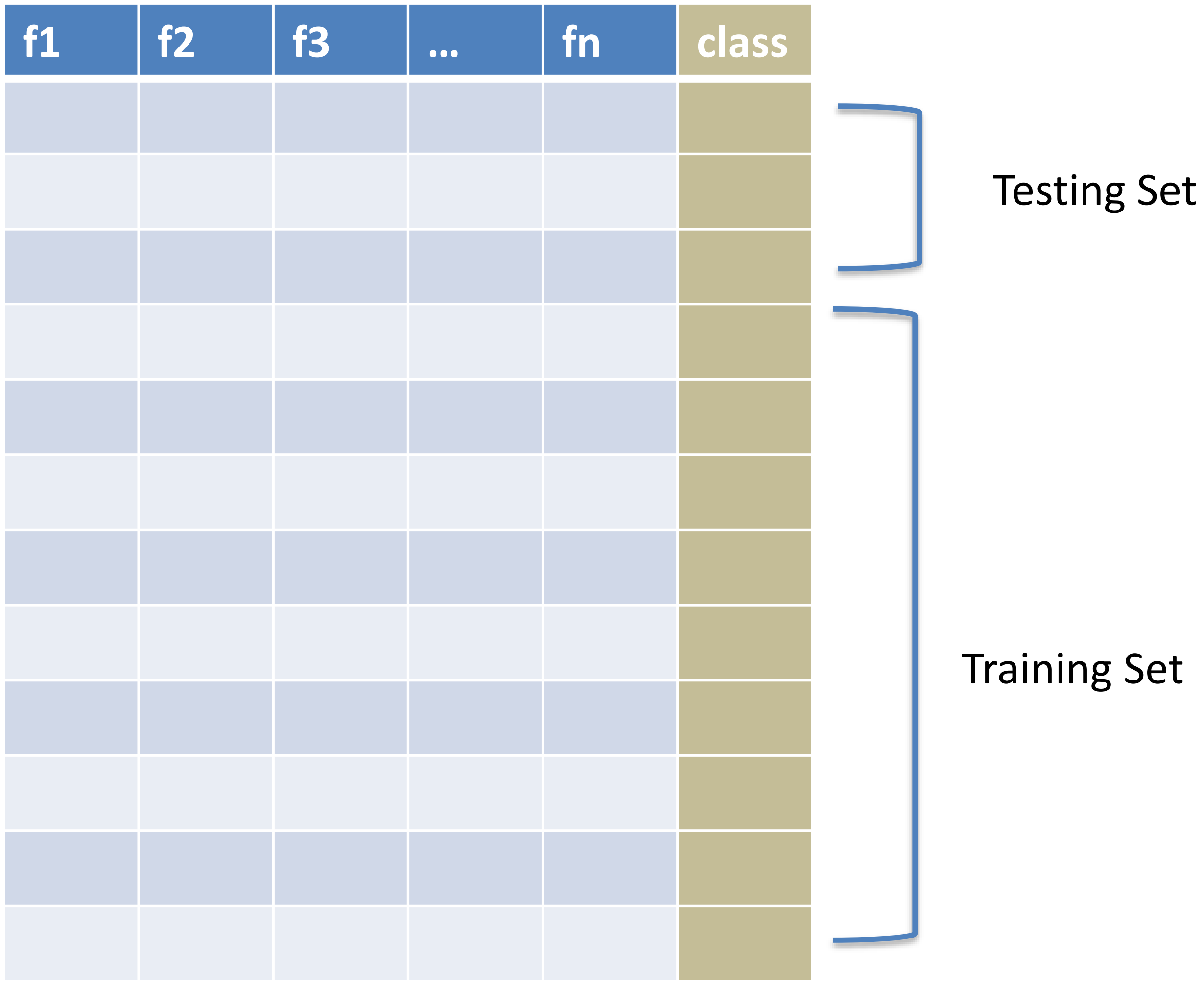
The **accuracy_score is** a simple function (available in the metrics module) will count the number of classes we correctly predicated and express that as a **percentage** of the total number of test data instances.

features_train, labels_train

features_test, labels_test

Basics of using Scikit Learn

1. In this scenario we have a single dataset, which we then split into training and test data.



We split a single dataset into test and train data we select the rows ***randomly***.

Assessing Accuracy (Splitting Training Data)

- In scikit-learn a random split into training and test sets can be quickly computed with the ***train_test_split*** helper function.
- As arguments we pass it the **original data** and **target** as well as the **percentage of the original data** we want for the training data. We also pass it a random seed.

```
from sklearn import tree
from sklearn import datasets
from sklearn import model_selection


wine = datasets.load_wine()

train_features, test_features, train_labels, test_labels = model_selection.train_test_split(
wine.data, wine.target, test_size=0.2, random_state=0)



print (wine.data.shape, wine.target.shape)
print (train_features.shape, train_labels.shape)

print (test_features.shape, test_labels.shape)
```

```
(178, 13) (178,)

(142, 13) (142,)
(36, 13) (36,)
```

```
from sklearn import tree
from sklearn import datasets
from sklearn import model_selection
from sklearn import metrics


wine = datasets.load_wine()
train_features, test_features, train_labels, test_labels =
 model_selection.train_test_split( wine.data, wine.target, test_size=0.2, random_state=0)

clf = tree.DecisionTreeClassifier()
# trains the classifier pass it the training data and classes
clf = clf.fit(train_features, train_labels)


# predict the class for an unseen example
results= clf.predict(test_features)
print  (metrics.accuracy_score(results, test_labels))

# we can replace the above two lines with the line below as
# a shortcut
print (clf.score(test_features, test_labels))
```

The slide shows the full program, where we take in the training data. We split into a training and test set. We then assess its accuracy

```
0.972222222222
0.972222222222
```

Notice the score function can be used instead of the predict and accuracy score functions

```
from sklearn import datasets
from sklearn import model_selection



iris = datasets.load_iris()

train_features, test_features, train_labels, test_labels =  model_selection.train_test_split(
iris.data, iris.target, test_size=0.2, random_state=4)

from collections import Counter

print (Counter(iris.target))
print (Counter(train_labels))
```

```
Counter({0: 50, 1: 50, 2: 50})

Counter({1: 45, 2: 41, 0: 34})
```

```
from sklearn import datasets
from sklearn import model_selection



iris = datasets.load_iris()

train_features, test_features, train_labels, test_labels =  model_selection.train_test_split(
iris.data, iris.target, test_size=0.2, random_state=4)

from collections import Counter

print (Counter(iris.target))
print (Counter(train_labels))
```

```
Counter({0: 50, 1: 50, 2: 50})

Counter({1: 45, 2: 41, 0: 34})
```

```
from sklearn import datasets
from sklearn import model_selection



iris = datasets.load_iris()
train_features, test_features, train_labels, test_labels =  model_selection.train_test_split(
iris.data, iris.target, test_size=0.2, random_state=4, stratify = iris.target)

from collections import Counter
print (Counter(iris.target))
print (Counter(train_labels))
```

```
Counter({0: 50, 1: 50, 2: 50})

Counter({2: 40, 1: 40, 0: 40})
```

Notice the use of **stratify** above means that the split between the training and test set is done randomly but in such a way that the distribution of class labels in the new training and test sets reflect the distribution of classes in the original dataset.

Using a Regression in Scikit Learn

- In the following slide we apply a k nearest neighbour regression algorithm to a regression dataset.
- Specifically we apply the KNeighborsRegressor algorithm to this dataset.

```
from sklearn import metrics
from sklearn import model_selection
from sklearn.neighbors import KNeighborsRegressor
from sklearn.datasets.samples_generator import make_regression

data, target = make_regression(n_samples=10000, n_features=20, noise = 0.2,
random_state = 10)

train_features, test_features, train_labels, test_labels = model_selection.train_test_split(
data, target, test_size=0.2)

reg = KNeighborsRegressor()
reg = reg.fit(train_features, train_labels)

results= reg.predict(test_features)
print (metrics.r2_score(test_labels,results))

print (reg.score(test_features, test_labels))
```

Above I generate a sample dataset using **make_regression**. This is often really useful if you want to explore certain functionality in Scikit as you can generate dataset and give them specific characteristics. (Note there is also a **make_classification** function)

```
from sklearn import metrics
from sklearn import model_selection
from sklearn.neighbors import KNeighborsRegressor
from sklearn.datasets.samples_generator import make_regression

data, target = make_regression(n_samples=10000, n_features=20, noise = 0.2,
random_state = 10)

train_features, test_features, train_labels, test_labels = model_selection.train_test_split(
data, target, test_size=0.2)

reg = KNeighborsRegressor()
reg = reg.fit(train_features, train_labels)

results= reg.predict(test_features)
print (metrics.r2_score(test_labels,results))

print (reg.score(test_features, test_labels))
```

Notice, just we again use the **score** function (as we did with classification). However, the default metric in the score value for regression models is r^2. Alternatively we can call the r2_score function from the metrics module. If you look at the metrics module it provide a much broad range of metrics (more on this later)

Machine Learning

Machine Learning

Part 1 – Data Preparation

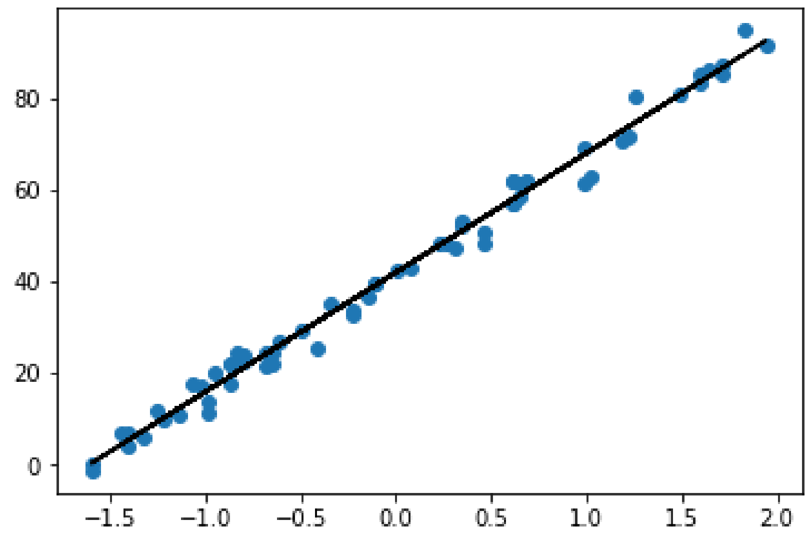
Ted Scully

Data Pre-processing for Scikit Learn

- Dealing with Outliers
- Dealing with Missing Values
- Handling Categorical Data
- Scaling Data
- Handling Imbalance
- Feature Selection

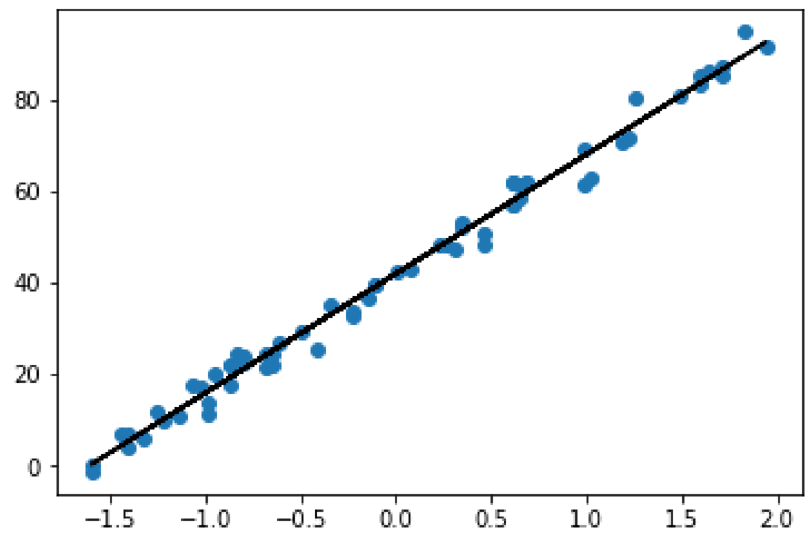
Outlier Detection

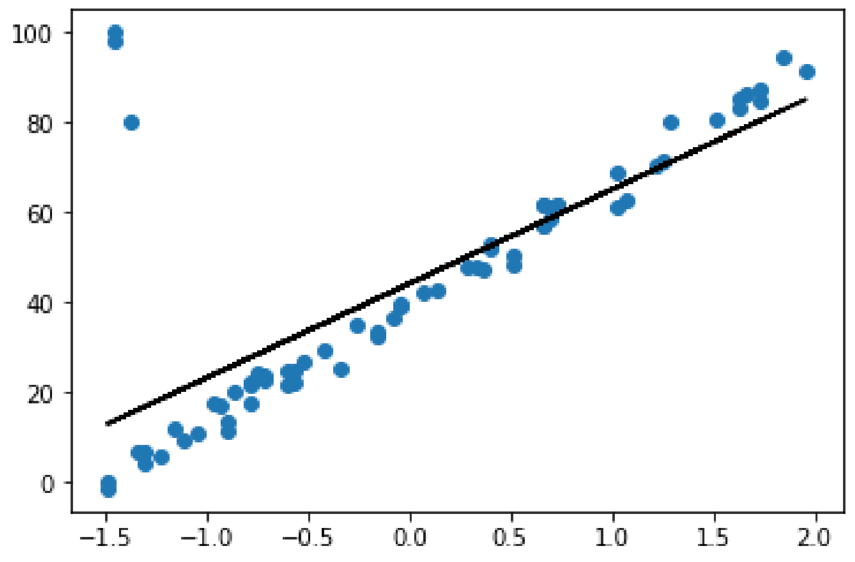
- Outliers are data that differ significantly from other data in a sample.
- Outliers **skew your data distributions** and impact your basic statistical measures and can be responsible for **underperformance** of **certain** algorithms.
- Outliers might be caused by faulty equipment, human error such as data entry, transcribing results, data processing error, etc



Outlier Detection

- Outliers are data that differ significantly from other data in a sample.
- Outliers **skew your data distributions** and impact your basic statistical measures and can be responsible for **underperformance** of **certain** algorithms.
- Outliers might be caused by faulty equipment, human error such as data entry, transcribing results, data processing error, etc





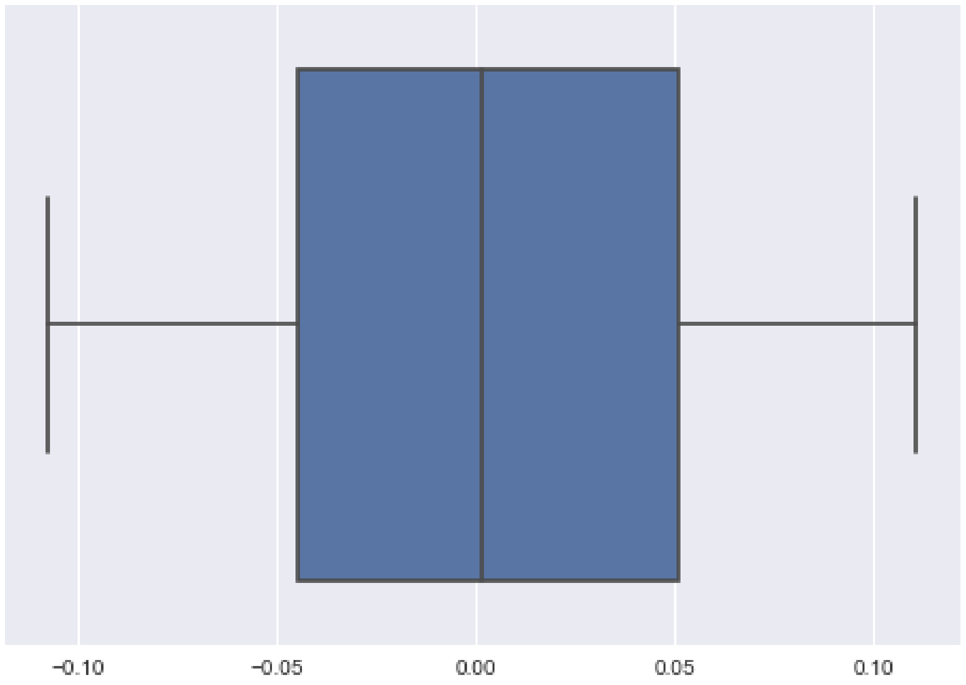
Univariate Outlier Detection

- There are many different methods used for identifying and dealing with outliers.
- When considering **univariate outlier detection** we look at a specific feature in isolation and attempt to identify an outlier data point amongst the feature data.
- For example, the following rules are often applied:
  - Removal of data points that occur three or more standard deviations away from mean are considered outlier.
  - Values outside the range -1.5 x IQR to 1.5 x IQR removed.

However, visualizations are often used as an aid in order to try to identify outliers.

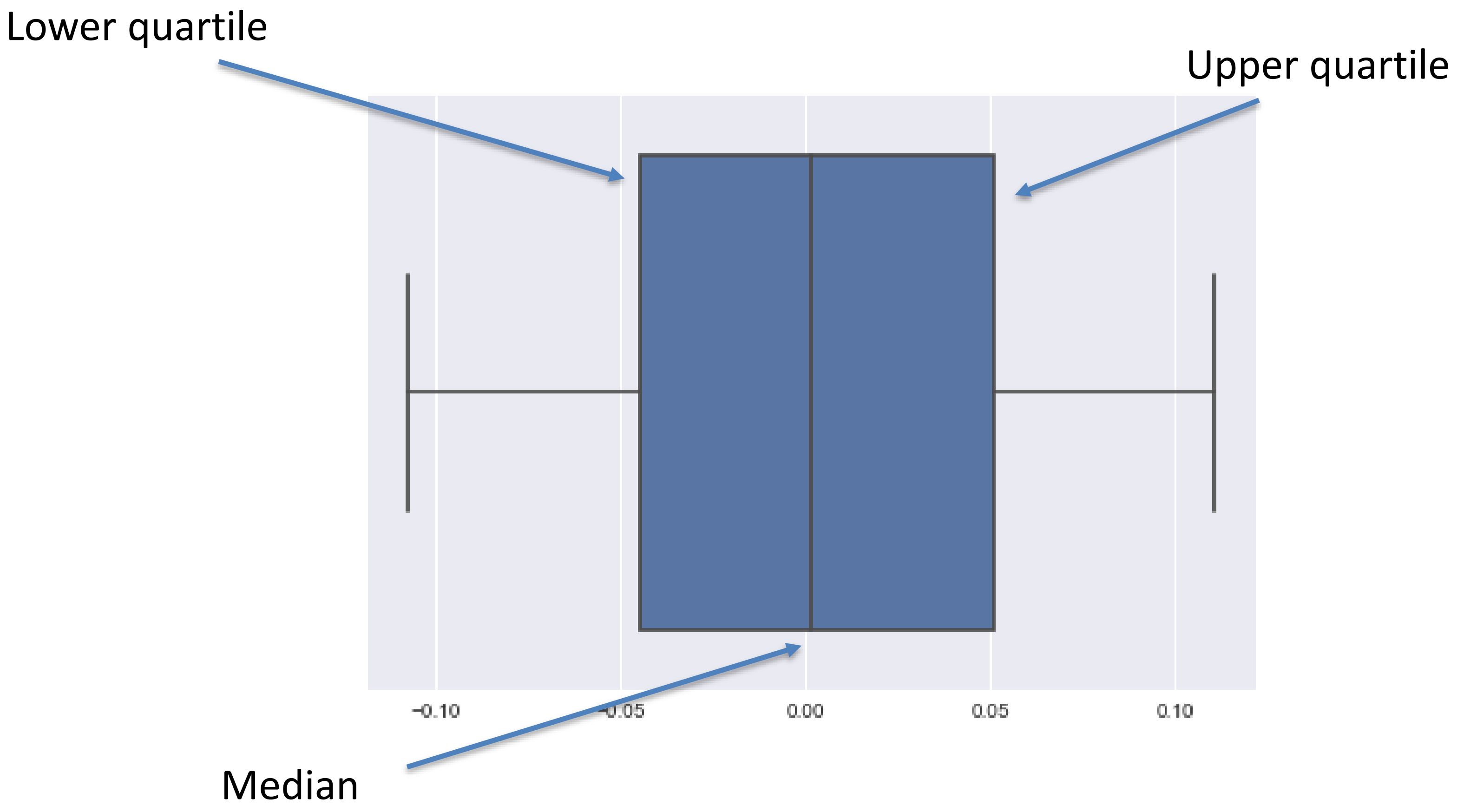
Visualization for Outliers

- A boxplot is a common visualization used for depicting the distribution of data feature based on it’s **quartiles**.
- You can spot the problematic feature values by looking at the **extremities** of the distribution.
- A boxplot in Seaborn has what are called whiskers, which by default are set to **plus or minus 1.5 IQR** (Inter quartile range is the difference between the upper and low quartiles).
- Any data points outside these whiskers are possible outliers and are candidates for removal.



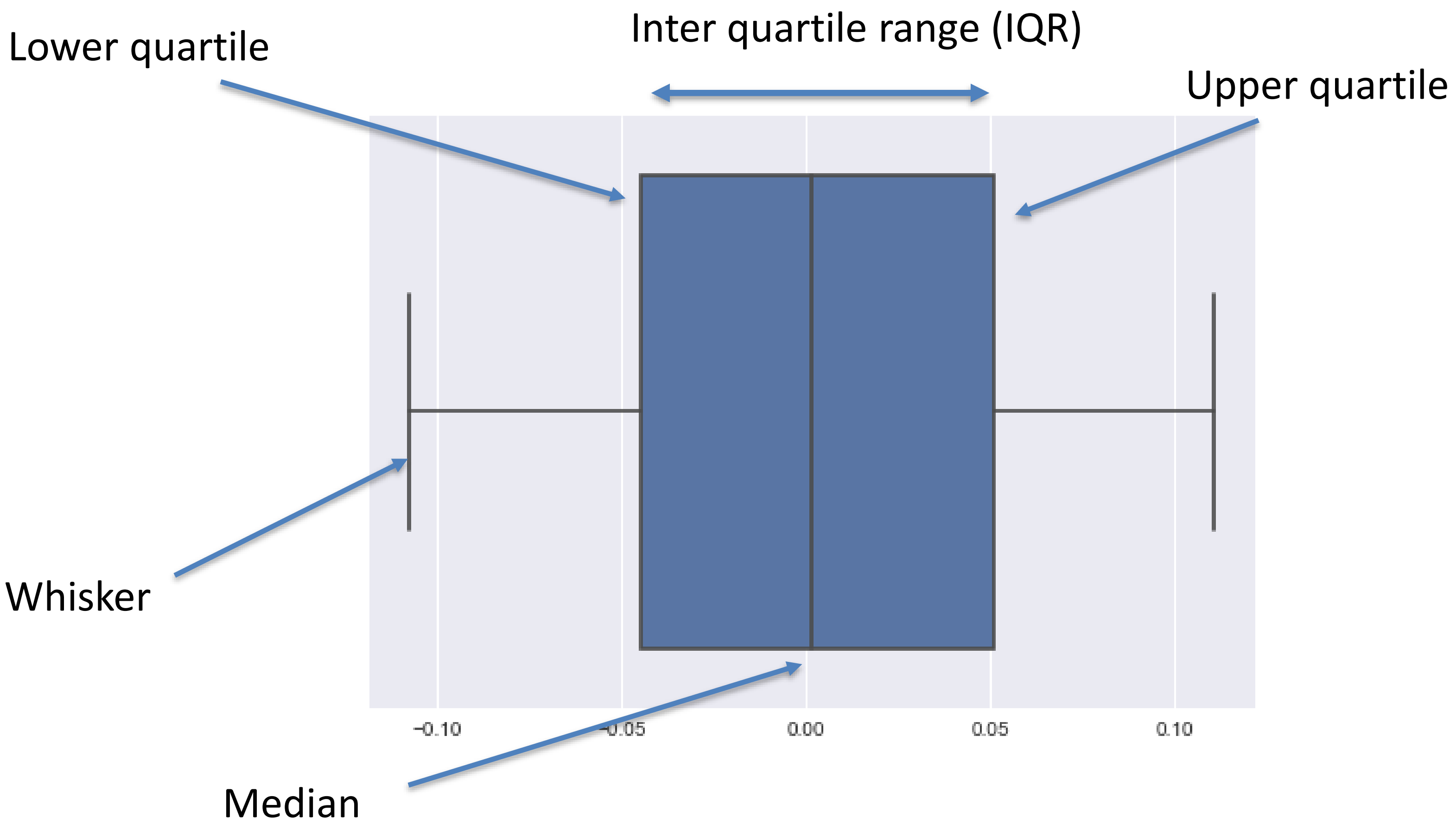
Visualization for Outliers

- A boxplot in Seaborn has what are called whiskers, which by default are set to **plus or minus 1.5 IQR** (Inter quartile range is the difference between the upper and low quartiles).
- Any data points outside these whiskers are possible outliers and are candidates for removal.

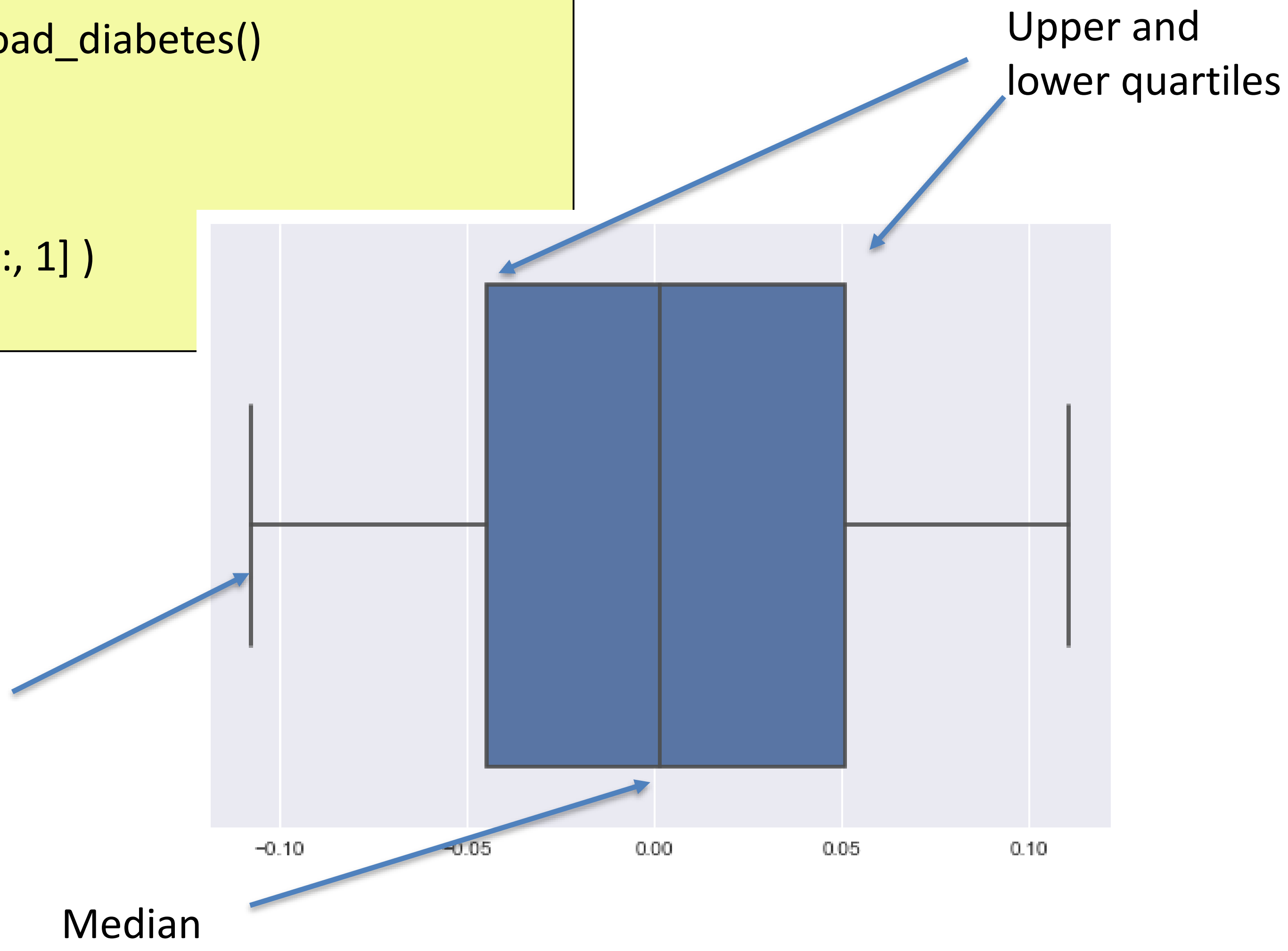


Visualization for Outliers

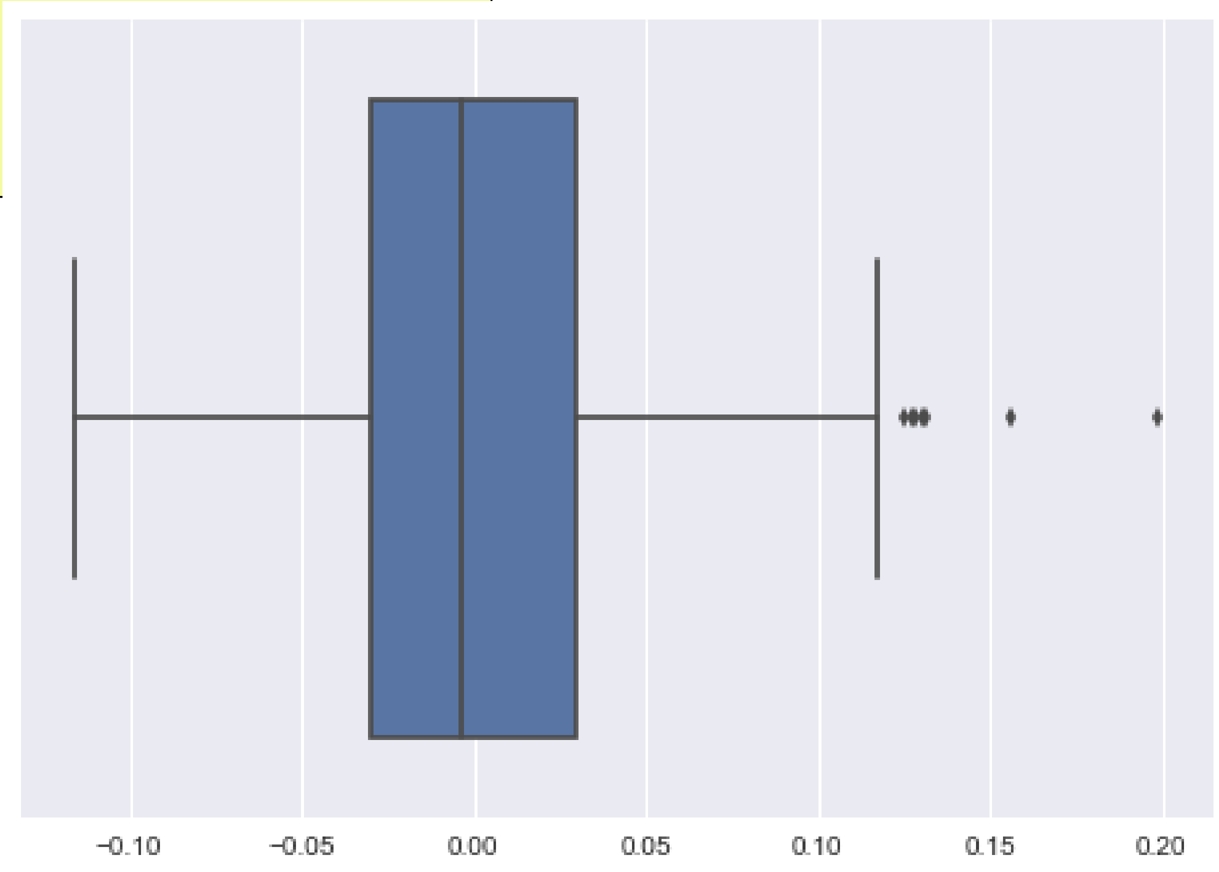
- A boxplot in Seaborn has what are called whiskers, which by default are set to **plus or minus 1.5 IQR** (Inter quartile range is the difference between the upper and low quartiles).
- Any data points outside these whiskers are possible outliers and are candidates for removal.



```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import datasets

diab = datasets.load_diabetes()
X = diab.data
y = diab.target

sns.boxplot( x=X[:, 1] )
plt.show()
```

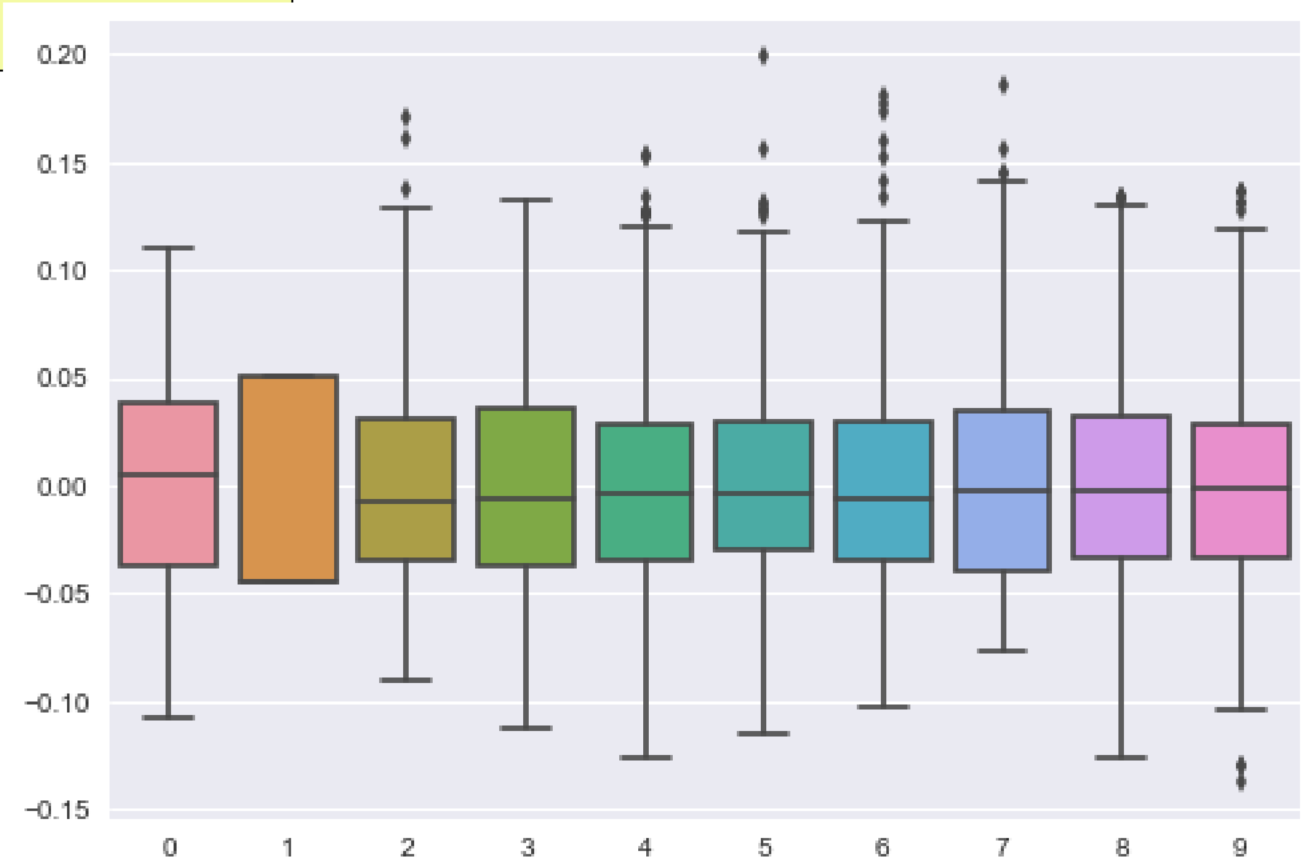


```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import datasets

diab = datasets.load_diabetes()
X = diab.data
y = diab.target

sns.boxplot( x=X[:, 5] )
plt.show()
```



```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import datasets

diab = datasets.load_diabetes()
X = diab.data
y = diab.target

sns.boxplot(data= pd.DataFrame(X))
plt.show()
```

- It is often useful to **plot all features together** to get a overview of entire dataset (clearly this is only practical if all features have been normalized or standardized in advance or appear within the same range).
- We can do this by passing in a dataframe containing all the features from our dataset.



What to do with the outliers

- **Recommendation**: Recommend that you "err on the side of caution" when dealing with outliers.

- If data points appear just outside the whisker boundaries or appear in significant quantity I would not recommend classifying them as outliers. Remember outliers are really datapoints that are significantly isolated and removed from the main distribution of data points.

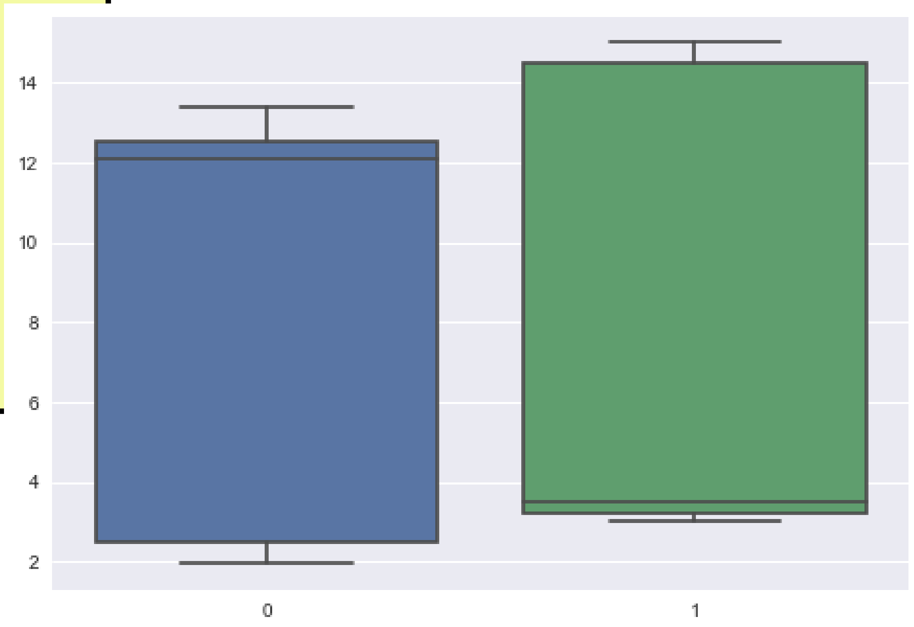
- There are a number of different methods for dealing with outliers.
  - Deletion of the associated instance from the dataset
  - Clamping outlier values. Resetting the outlier value to some upper or lower boundary value (for example, upper outlier values can be set to median + 1.5*IQR, lower outlier values will be set to median - 1.5*IQR, lower)
  - Set the outlier as a missing value and impute

Visualization for Outliers

- Consider the following dataset where we have two features.
- The boxplot doesn't reveal any outliers in this data.

```
import seaborn as sns
import matplotlib.pyplot as plt

data = np.array([ [2.5, 3.5] , [2.1, 3.3], [2.9, 3.1],
[1.95, 3.0], [12.5, 14.5], [13.1, 14.3], [13.4, 14.7],
[12.1, 15], [12.5, 3.2]])

print (data)

df = pd.DataFrame(data)

sns.boxplot(data= pd.DataFrame(data))
plt.show()
```

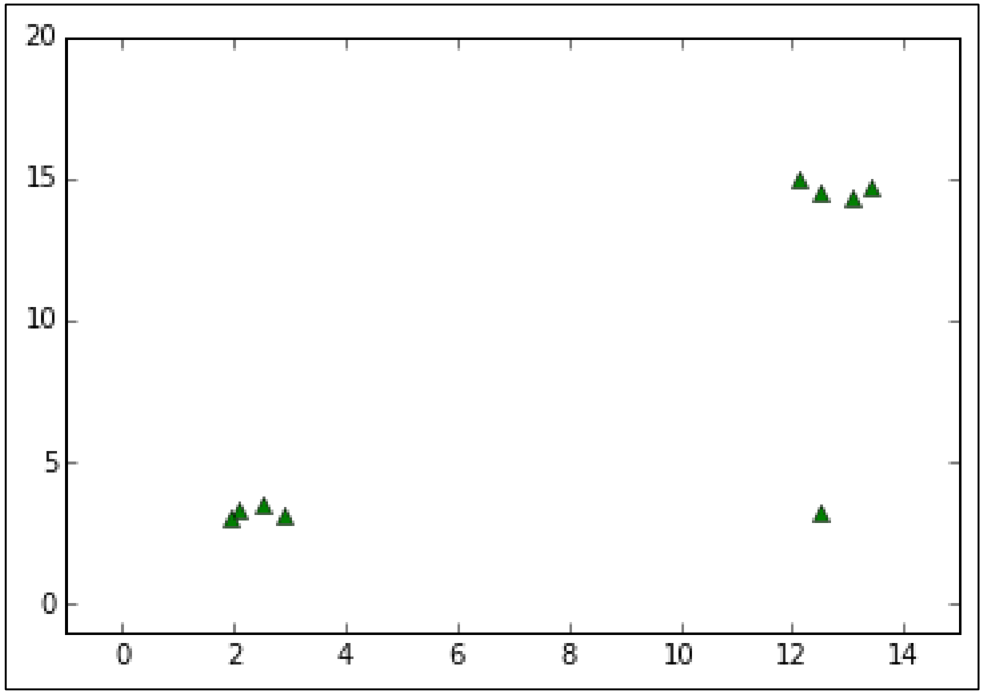
```
[[  2.5    3.5 ]
 [  2.1    3.3 ]
 [  2.9    3.1 ]
 [  1.95   3.  ]
 [ 12.5    3.2 ]
 [ 12.5   14.5 ]
 [ 13.1   14.3 ]
 [ 13.4   14.7 ]
 [ 12.1   15.  ]]
```

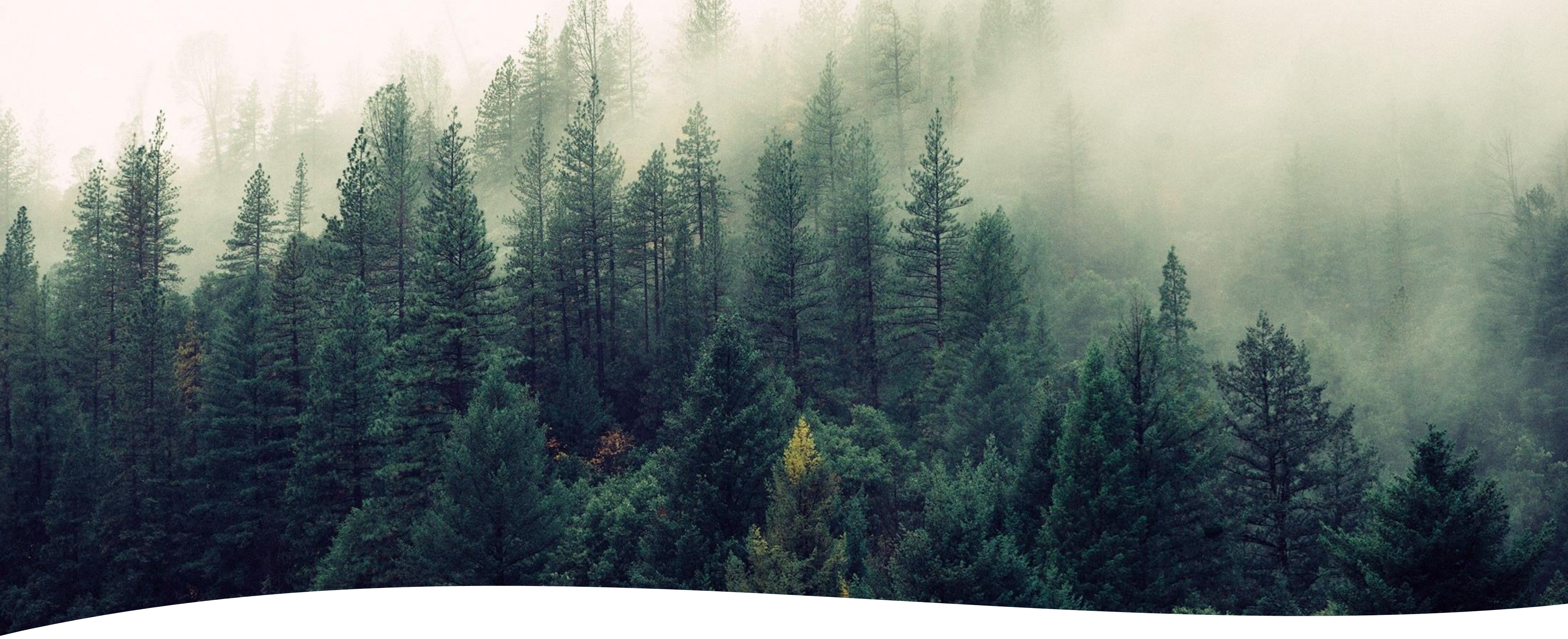


Visualization for Outliers

- However, notice if we plot one feature against another we can see that one data point is significantly removed from another data point.

- Univariate outlier detection is limited in this regard as you aren't considering unusual combinations of multiple variables





Isolation Forests for Multi-variate Outlier Detection

- Isolation forest are an **unsupervised learning** approach to identify isolated instances (outliers or anomalies) in feature space.
- The fundamental rationale behind an isolation forest (which consists of many isolation trees) is that it is much easier to isolate an outlier data point in feature space compare to isolating a data point that is part of an existing cluster.

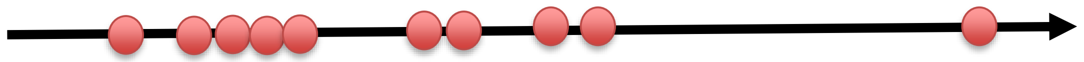
Isolation Forests for Multi-variate Outlier Detection

Consider the values from a single feature represented graphically below (there are just 10 values for this feature).

We will **randomly pick** a point to partition this feature (a point randomly chosen between the **max and min** value for the feature.

Let's assume the partition divides the data in into two groups. In one group we will only have a single instance (data point) and in the other group we will have the remaining nine in another partition.

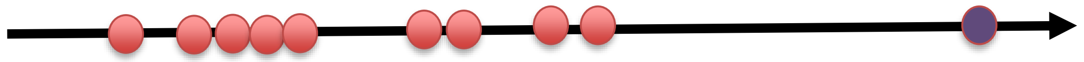
Which data point is most likely to be the single data point.



Isolation Forests for Multi-variate Outlier Detection

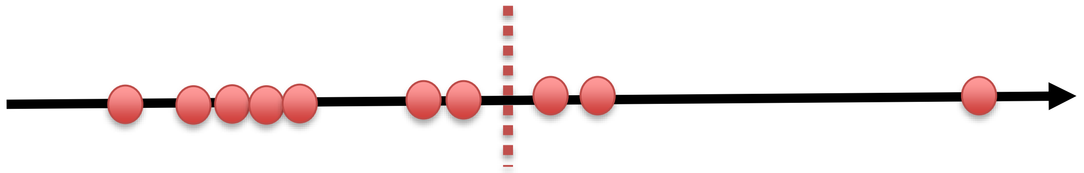
Which data point is most likely to be the single data point.

The data point on the right hand side is most likely to be the isolated instance as it is so far from the other data points.



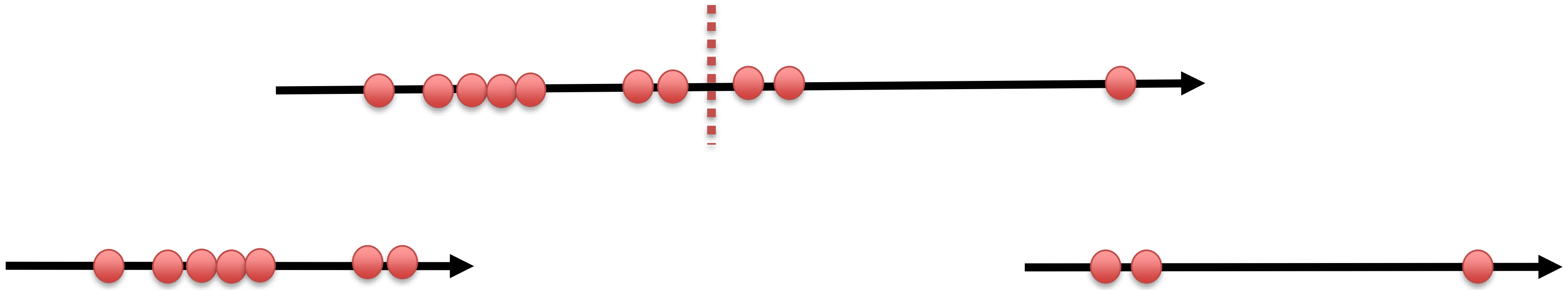
Isolation Forests for Multi-variate Outlier Detection

Now let’s say we were to repeat the process until we isolated one instance …. clearly the instance on the right in still the most likely to be isolated.



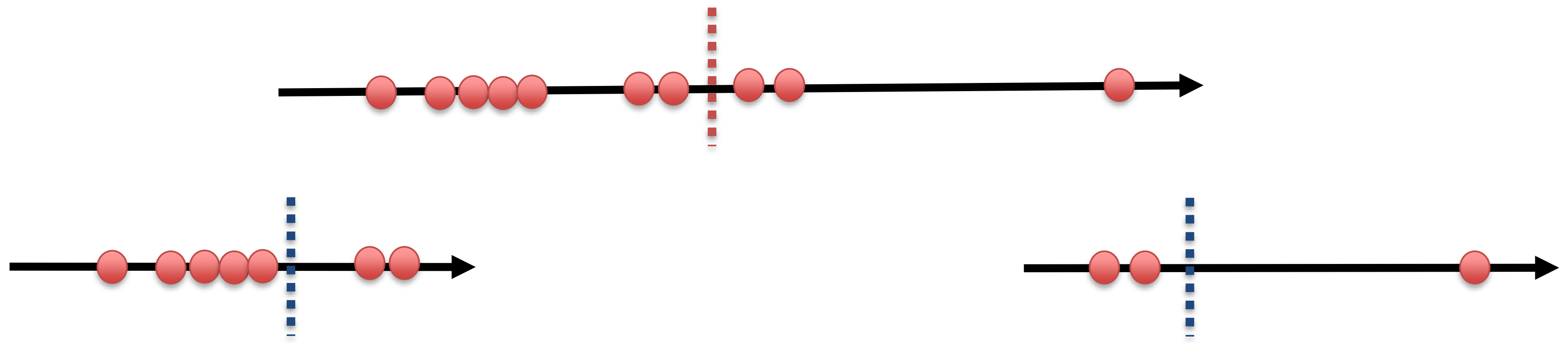
Isolation Forests for Multi-variate Outlier Detection

Now let's say we were to repeat the process until we isolated one instance …. clearly the instance on the right in still the most likely to be isolated.



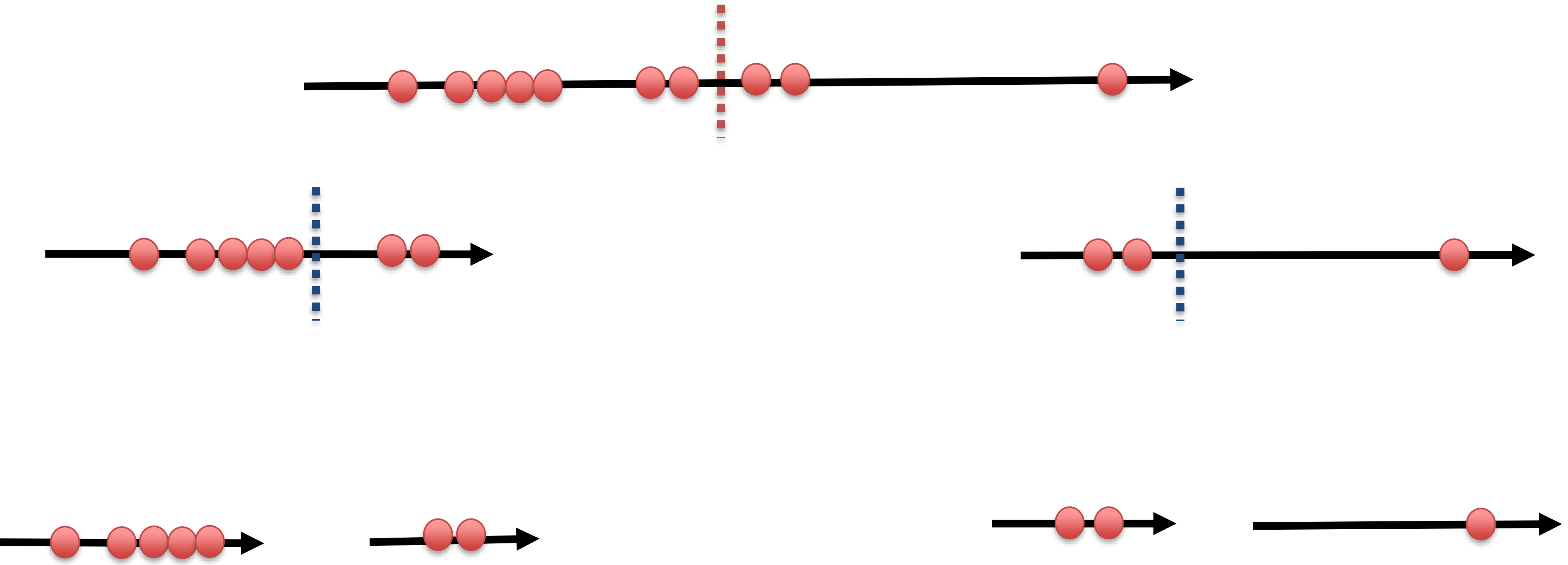
Isolation Forests for Multi-variate Outlier Detection

Now let’s say we were to repeat the process until we isolated one instance ….
clearly the instance on the right in still the most likely to be isolated.



Isolation Forests for Multi-variate Outlier Detection

Now let’s say we were to repeat the process until we isolated one instance …. clearly the instance on the right in still the most likely to be isolated.

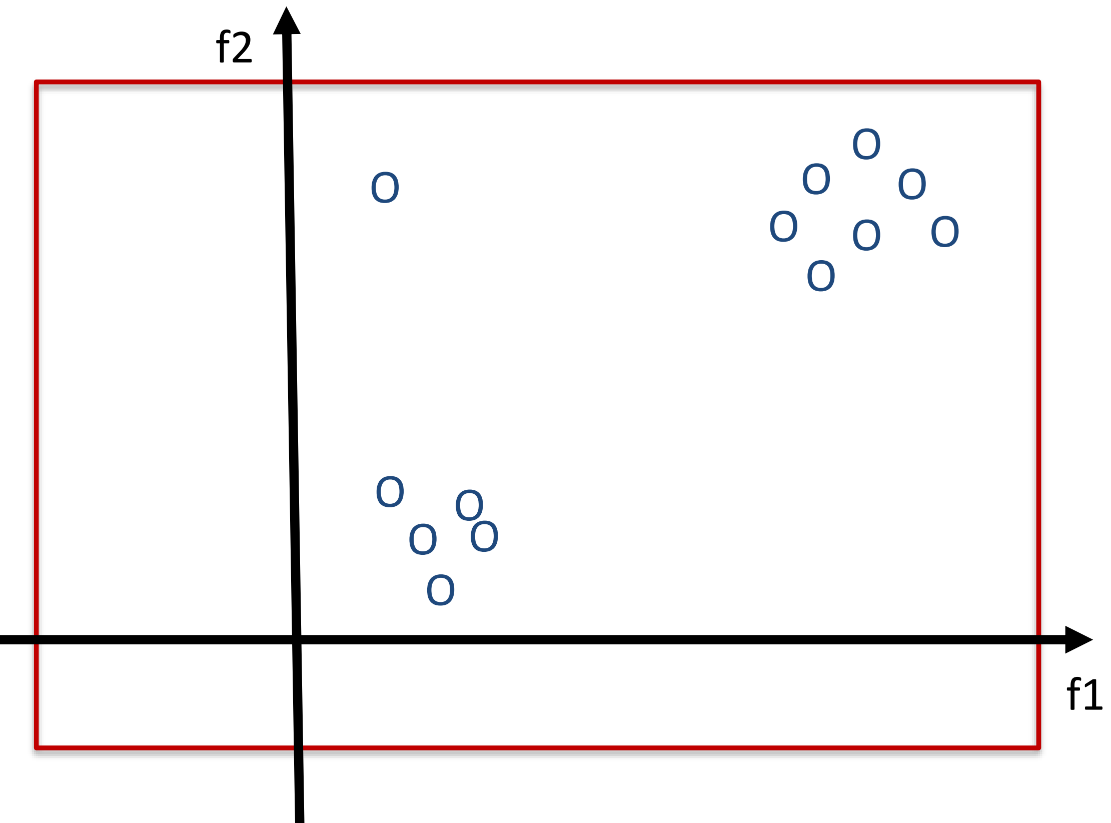


Isolation Forests for Multi-variate Outlier Detection

Now let's consider a feature space with 2 features. An isolation forest will build many **isolation trees**.

To build each tree we will (1) **randomly select a feature** and will (2) **randomly select a value to partition** that feature. It will continue this process iteratively.

Below f1 is feature 1 and f2 is feature 2 from our dataset.

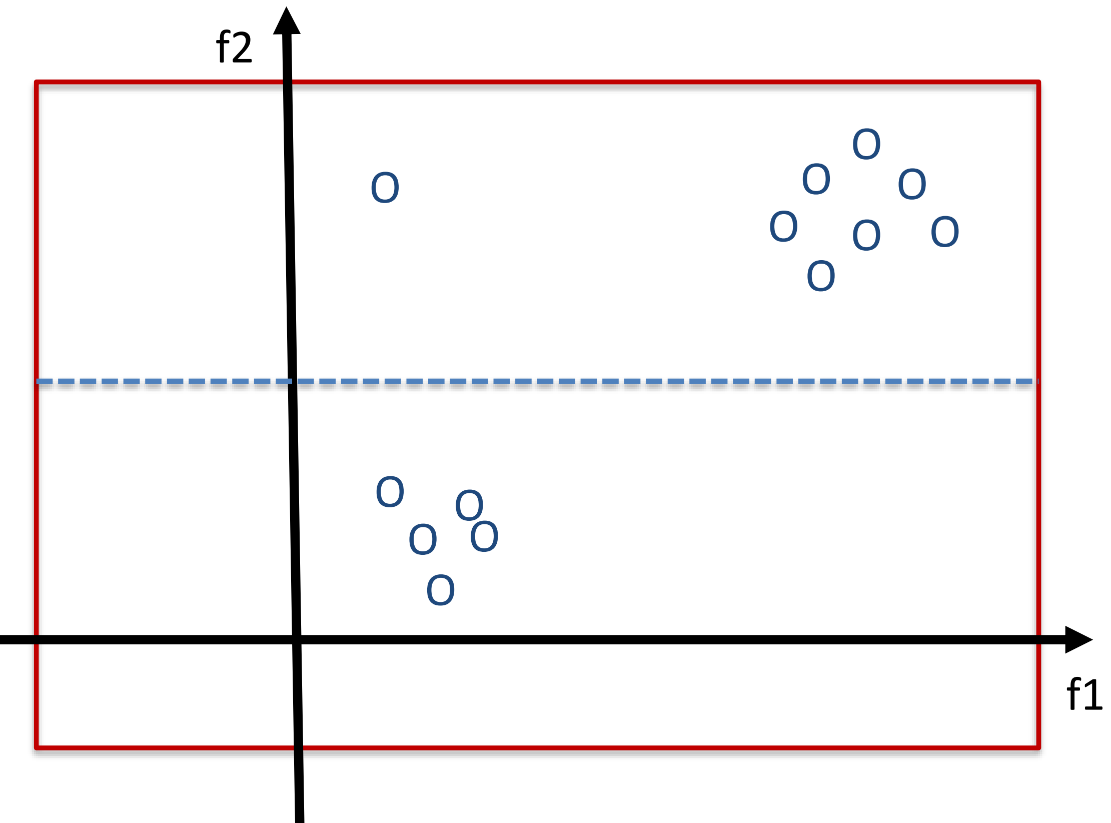


Isolation Forests for Multi-variate Outlier Detection

Now let's consider a feature space with 2 features. An isolation forest will build many **isolation trees**.

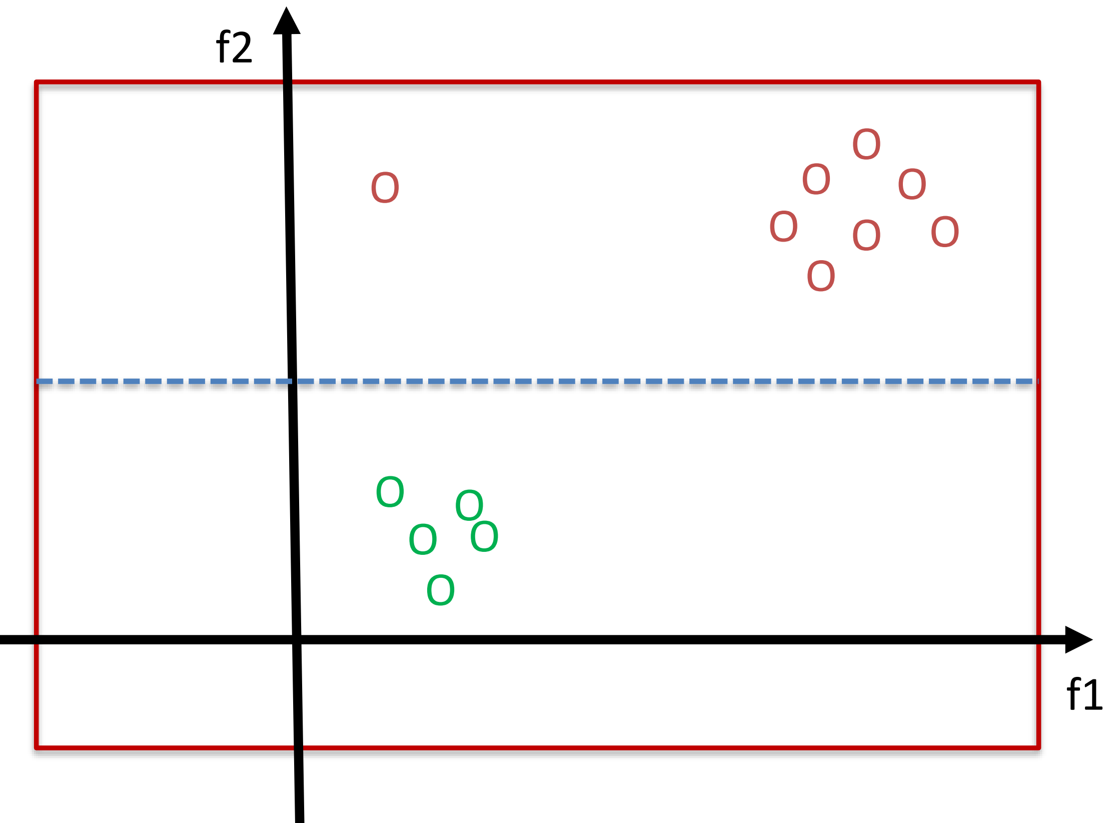
To build each tree we will (1) **randomly select a feature** and will (2) **randomly select a value to partition** that feature. It will continue this process iteratively.

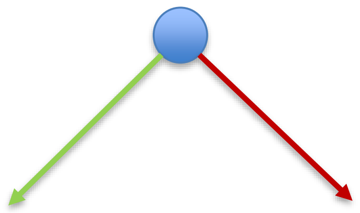
Below f1 is feature 1 and f2 is feature 2 from our dataset.



Isolation Forests for Multi-variate Outlier Detection

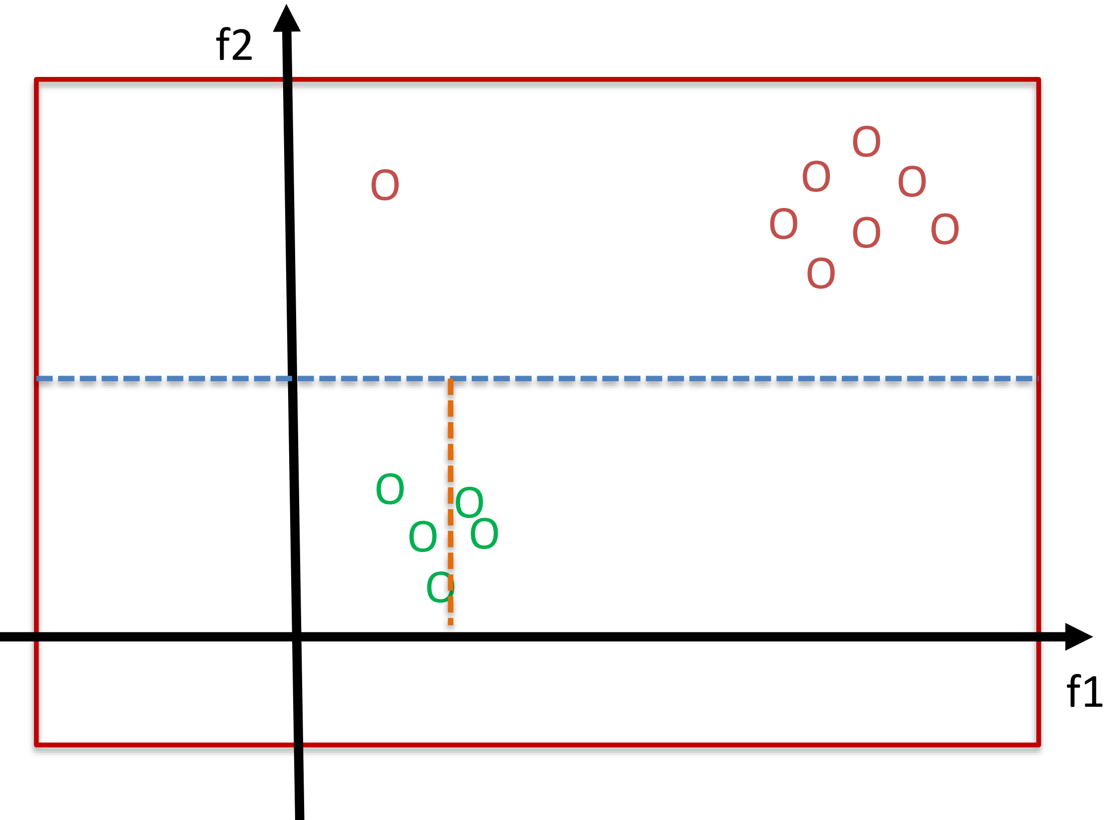
We can visualize this process as a binary tree. Every instance that has a f2 value less than the select partition flows down one branch and every instance with an f2 value greater than the partition points flows down the other branch

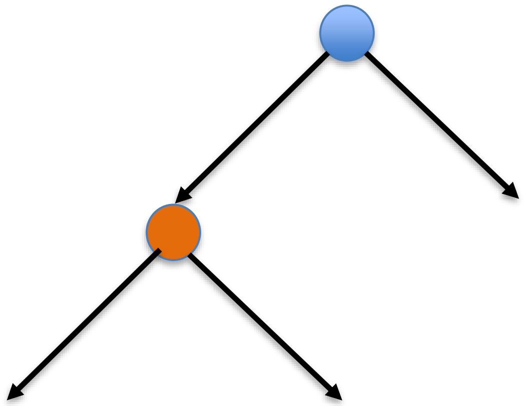




Isolation Forests for Multi-variate Outlier Detection

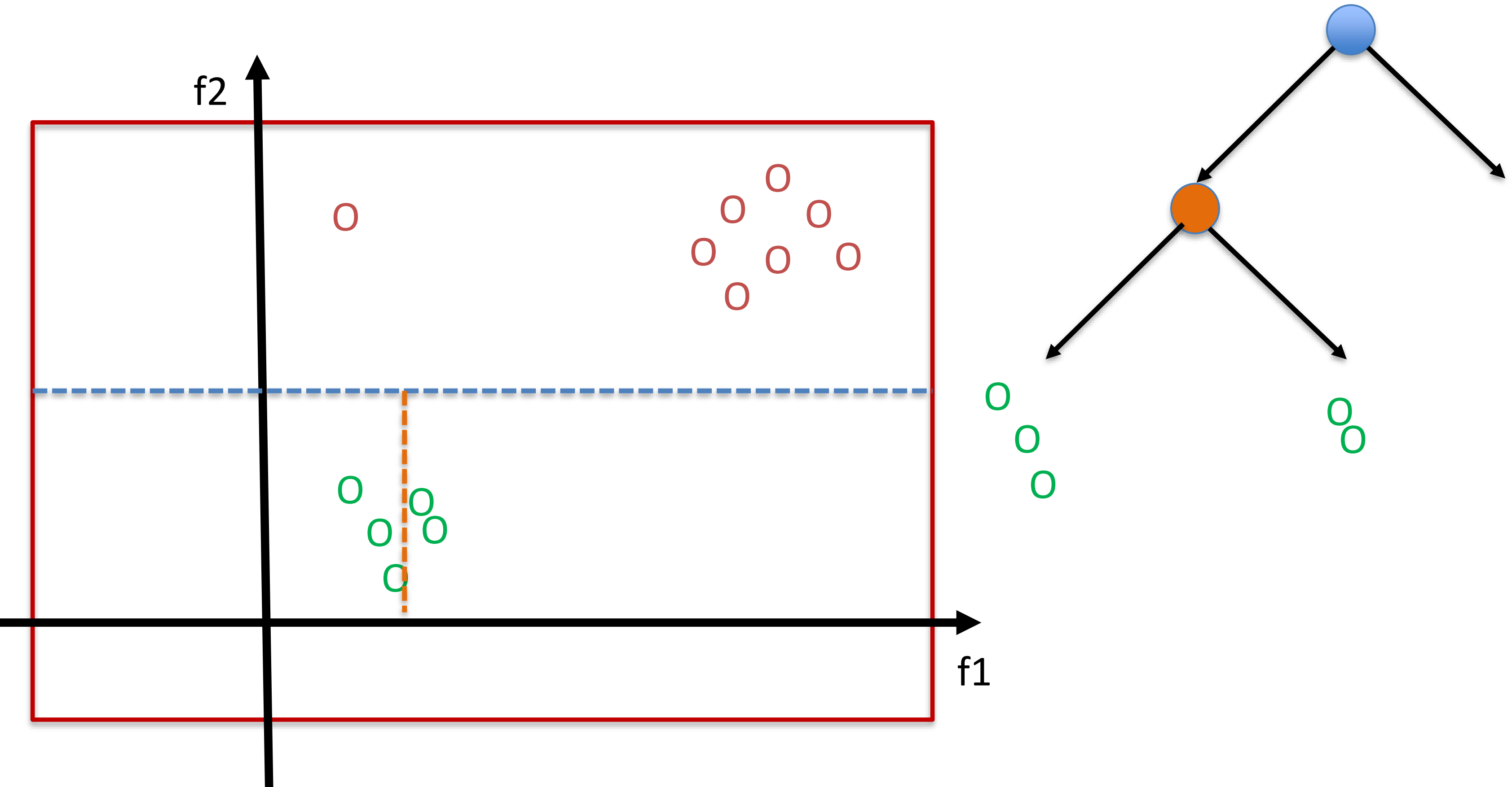
This iterative process then continues. For all those instances that flowed down the left hand side of our graph, we again randomly select a feature (for simplicity we assume it is f1). Then we randomly partition the data using f1.





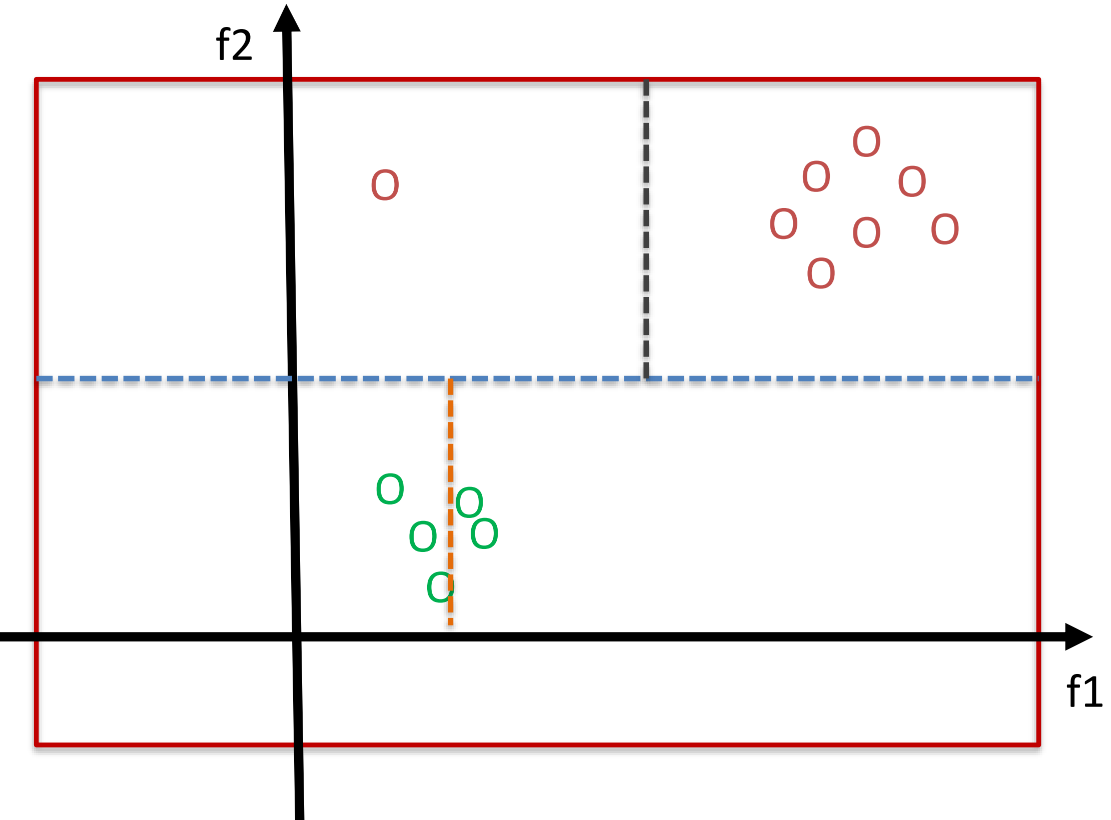
Isolation Forests for Multi-variate Outlier Detection

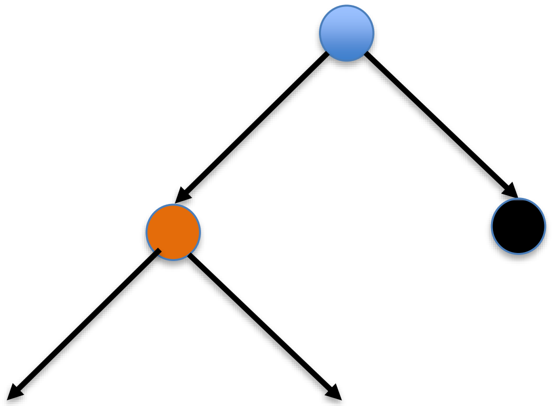
This iterative process then continues. For all those instances that flowed down the left hand side of our graph, we again randomly select a feature (for simplicity we assume it is f1). Then we randomly partition the data using f1.



Isolation Forests for Multi-variate Outlier Detection

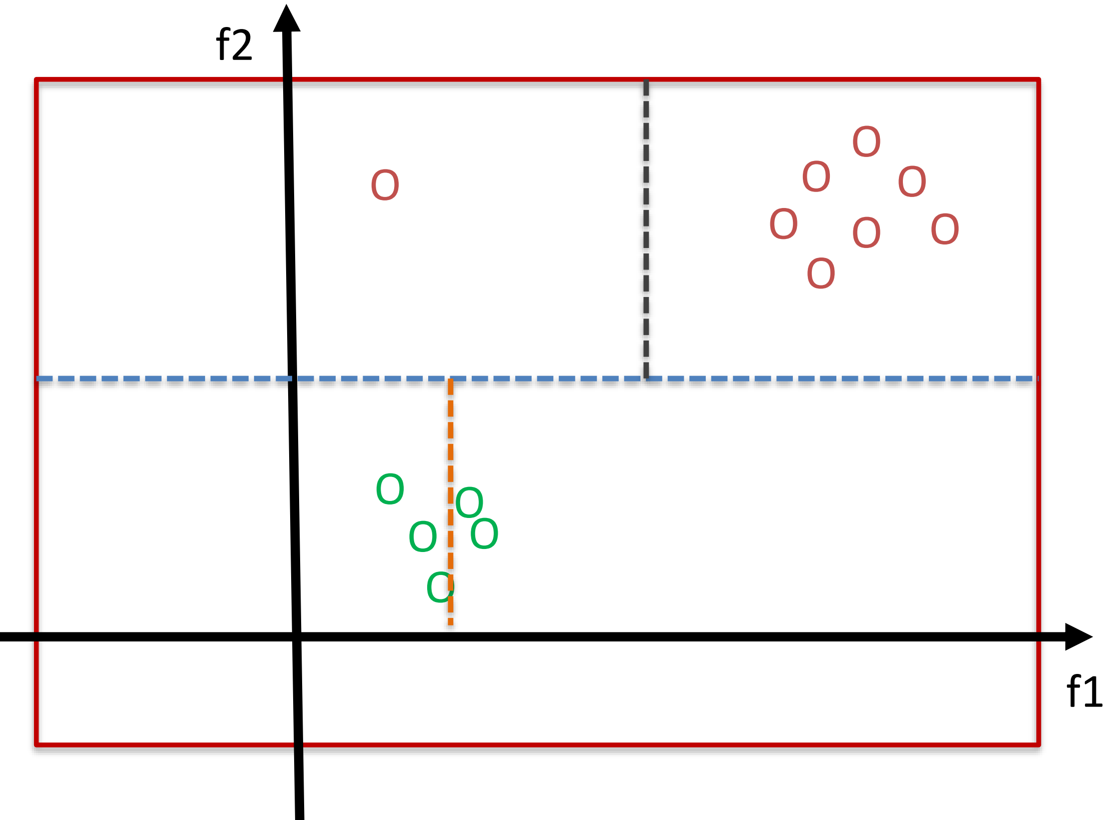
We repeat the same process for all those that flow down the right hand side of the original partition. We again pick a random a feature (f1 selected) and partition the feature space randomly by f1.

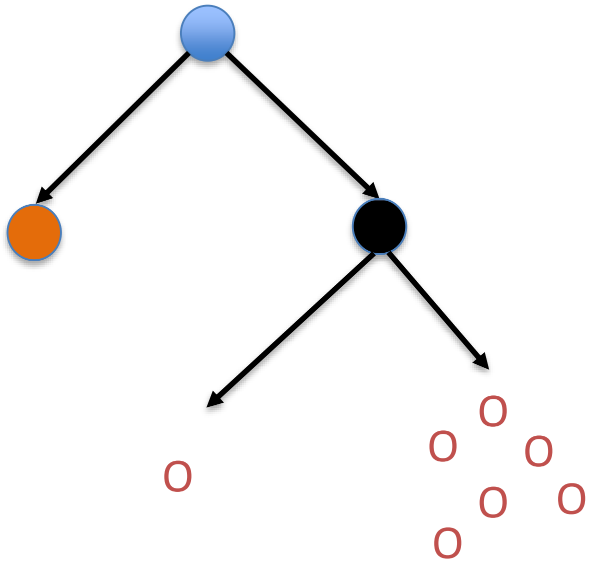




Isolation Forests for Multi-variate Outlier Detection

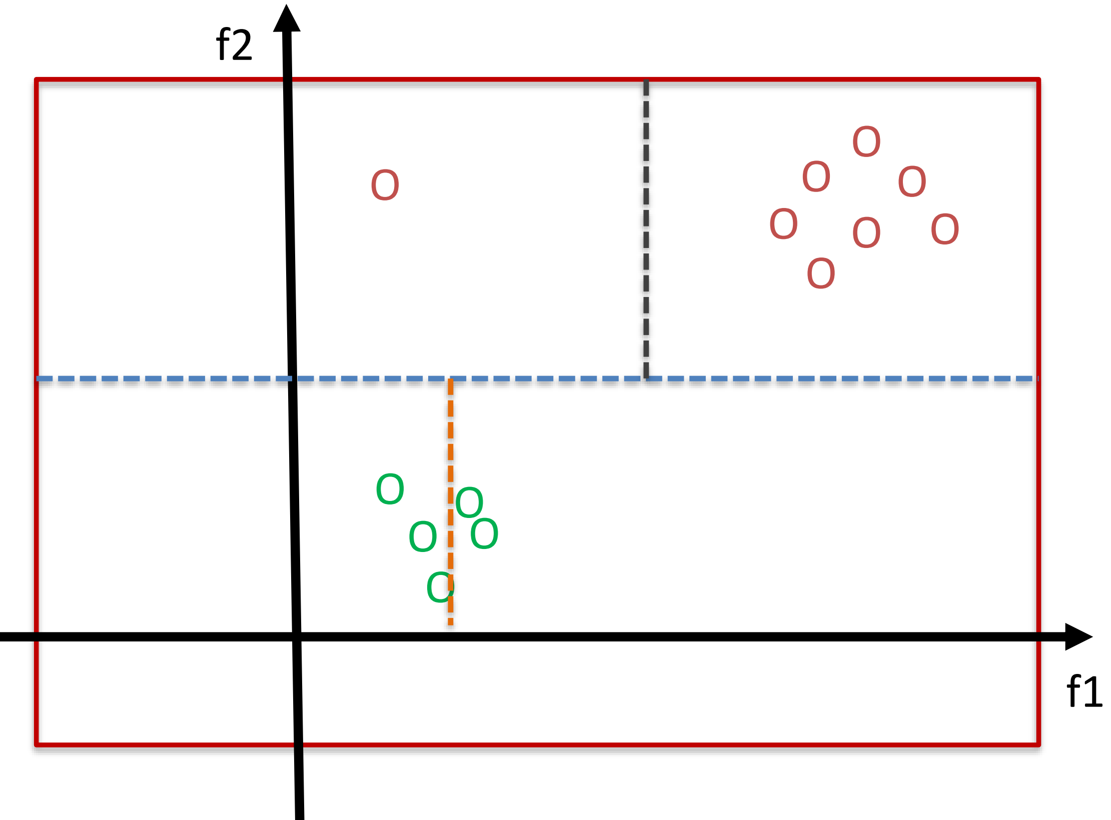
Notice we have now reached a point in the tree, where a single instance is on it's own (notice that this instance is the outlier). The process of building the tree continues until we reach a point where a particular depth of the tree is reached or until everyone single instance is alone at the end of the branch.

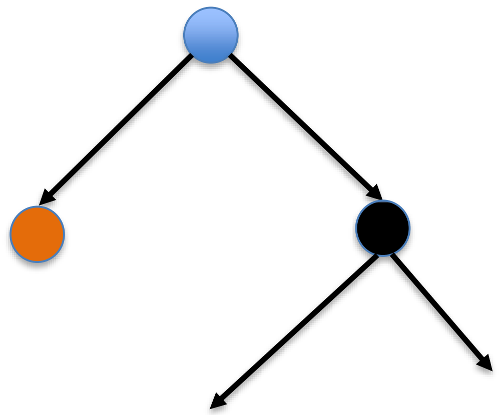




Isolation Forests for Multi-variate Outlier Detection

Notice we have now reached a point in the tree, where a single instance is on it's own (notice that this instance is the outlier). The process of building the tree continues until we reach a point where a particular depth of the tree is reached or until everyone single instance is alone at the end of the branch.

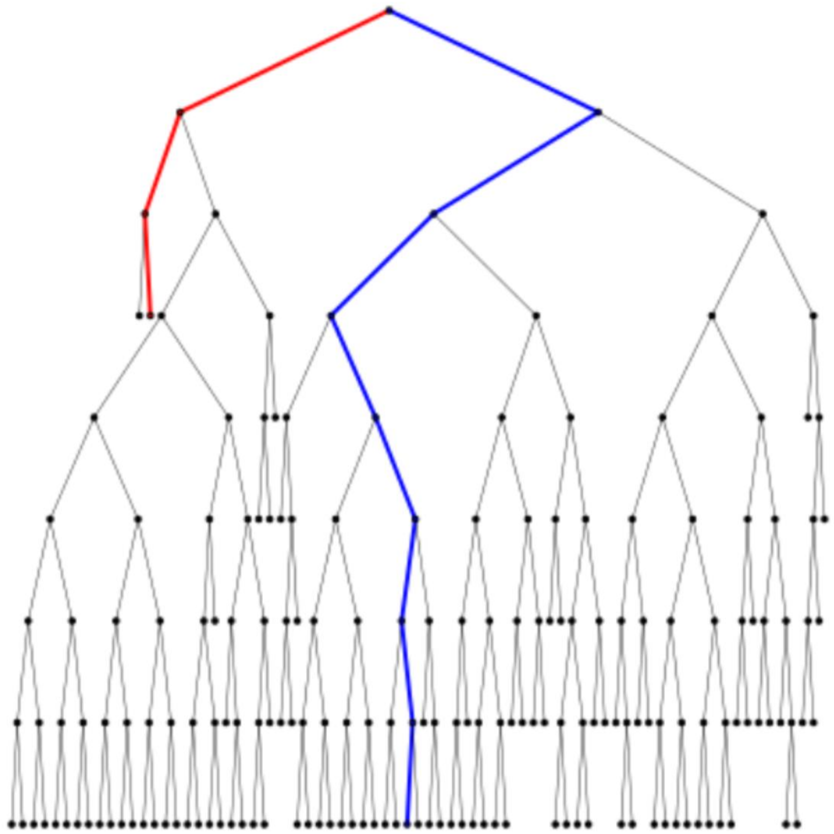




We refer to the tree above as an isolation tree. We continue growing the tree until all data points are isolated or until we have reached a particular maximum depth.

Isolation Forests for Multi-variate Outlier Detection

- At the core of the Isolation Forest algorithm is the observation that outlier instances in a dataset are easier to separate (isolate) from the rest of the sample (isolate).
- Consider the single isolation tree on the right.
- The number of edges on the red path is 3, whereas the number of edges along the blue path is 8. The data point at the terminal of the red path is more likely to be an outlier compared to the data point at the terminal of the blue path.



https://arxiv.org/pdf/1811.02141.pdf