CHULA ENGINEERING
Foundation toward Innovation

COMPUTER

Data Mining Group
Machine Intelligence and Knowledge Discovery Lab
Chulalongkorn University

Practical Machine Learning

Assoc. Prof. Peerapon Vateekul, Ph.D.
Department of Computer Engineering,
Faculty of Engineering, Chulalongkorn University
Peerapon.v@chula.ac.th
www.cp.eng.chula.ac.th/~peerapon/

Outlines

- Introduction
- Supervised Learning
- Unsupervised Learning
- Special Tasks
- Demo (colab)

Introduction

Retail's Adapt-Or-Die Moment: How Artificial Intelligence Is Reshaping Commerce

https://www.cbinsights.com/research/artificial-intelligence-reshaping-commerce/

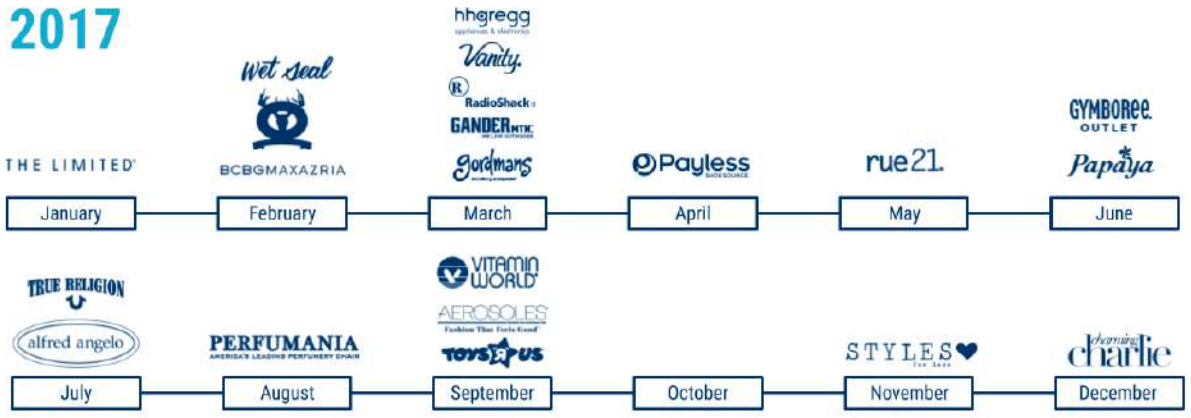
July 18, 2019

Artificial Intelligence Retail

Traditional and new-school retailers alike are using AI and robotics to automate various parts of the retail chain, from manufacturing to last-mile delivery.

Retail is under pressure to crack the AI code.

After all, corporations in every industry are scrambling to adapt and integrate artificial intelligence into their products — and retail is no exception. Between Q1'13 and Q2'19, retail AI startups looking to address retailers' AI needs raised a total of $2.67B across 564 deals.

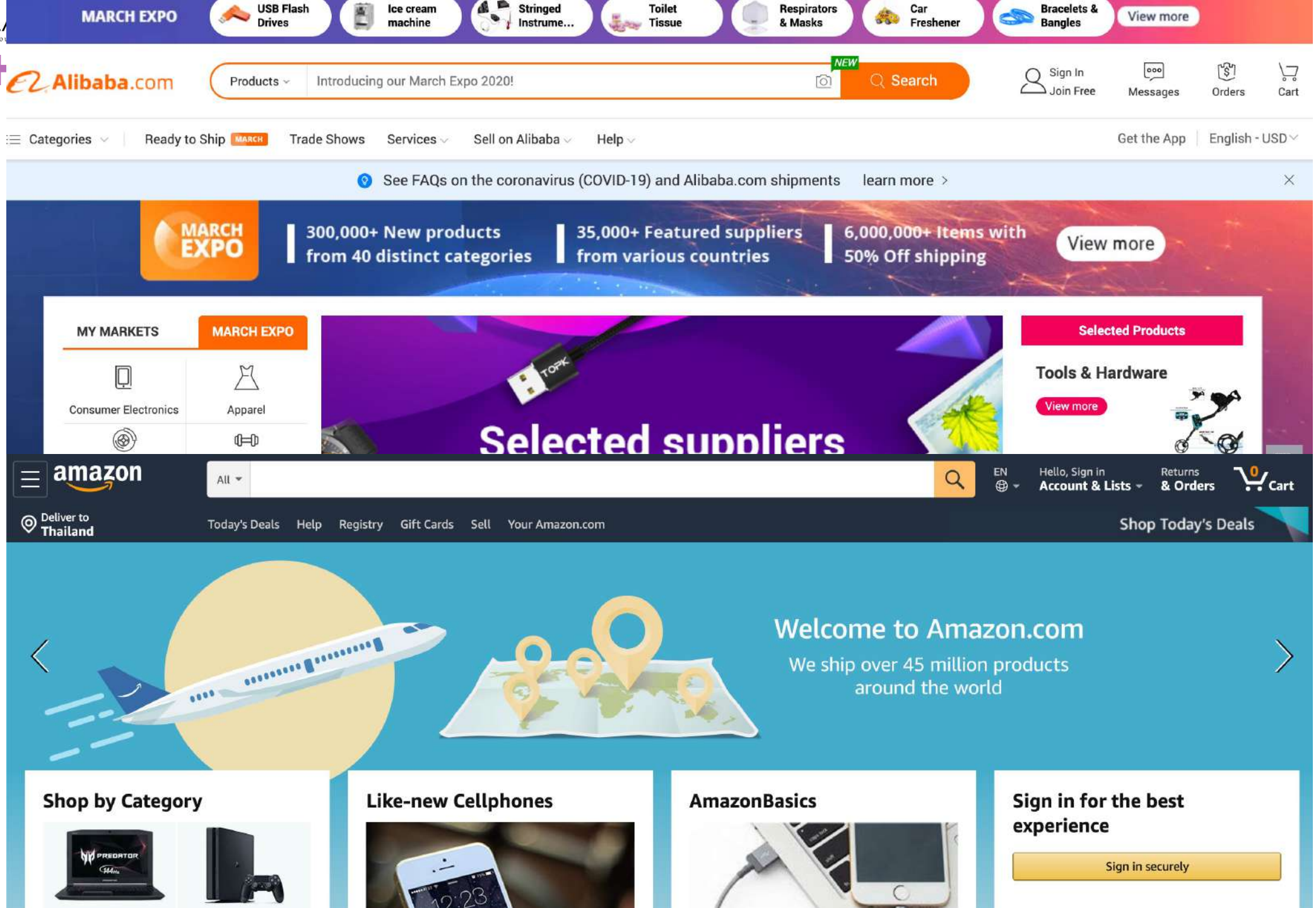


CBINSIGHTS

Bankruptcy!

Consumer channels: selling online vs. in-store

personalized experience & direct targeted marketing

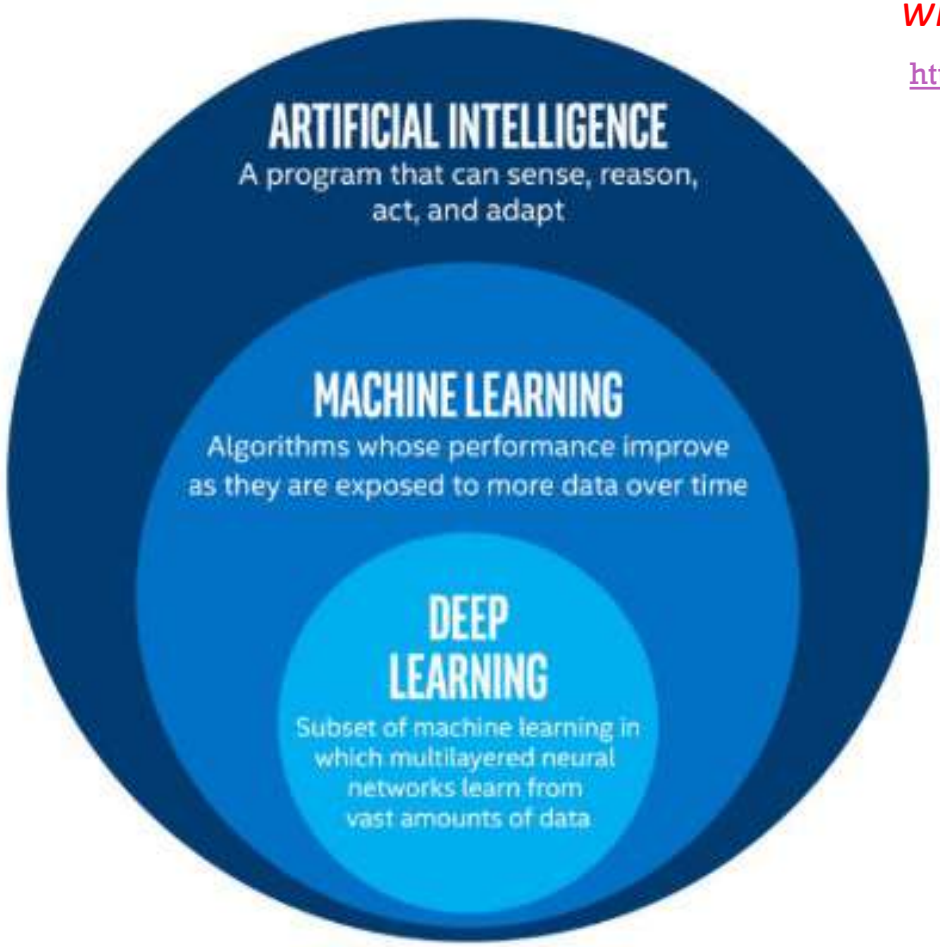

MARCH EXPO
USB Flash Drives
Ice cream machine
Stringed Instrume...
Toilet Tissue
Respirators & Masks
Car Freshener
Bracelets & Bangles
View more
Alibaba.com
Products
Introducing our March Expo 2020!
NEW
Search
Sign In
Join Free
Messages
Orders
Cart
Categories
Ready to Ship
MARCH
Trade Shows
Services
Sell on Alibaba
Help
Get the App
English - USD
See FAQs on the coronavirus (COVID-19) and Alibaba.com shipments
learn more
MARCH EXPO
300,000+ New products from 40 distinct categories
35,000+ Featured suppliers from various countries
6,000,000+ Items with 50% Off shipping
View more
MY MARKETS
MARCH EXPO
Consumer Electronics
Apparel
Selected suppliers
Selected Products
Tools & Hardware
View more
amazon
All
EN
Hello, Sign in
Account & Lists
Returns
& Orders
Cart
Deliver to Thailand
Today's Deals
Help
Registry
Gift Cards
Sell
Your Amazon.com
Shop Today's Deals
Welcome to Amazon.com
We ship over 45 million products around the world
Shop by Category
Like-new Cellphones
AmazonBasics
Sign in for the best experience
Sign in securely

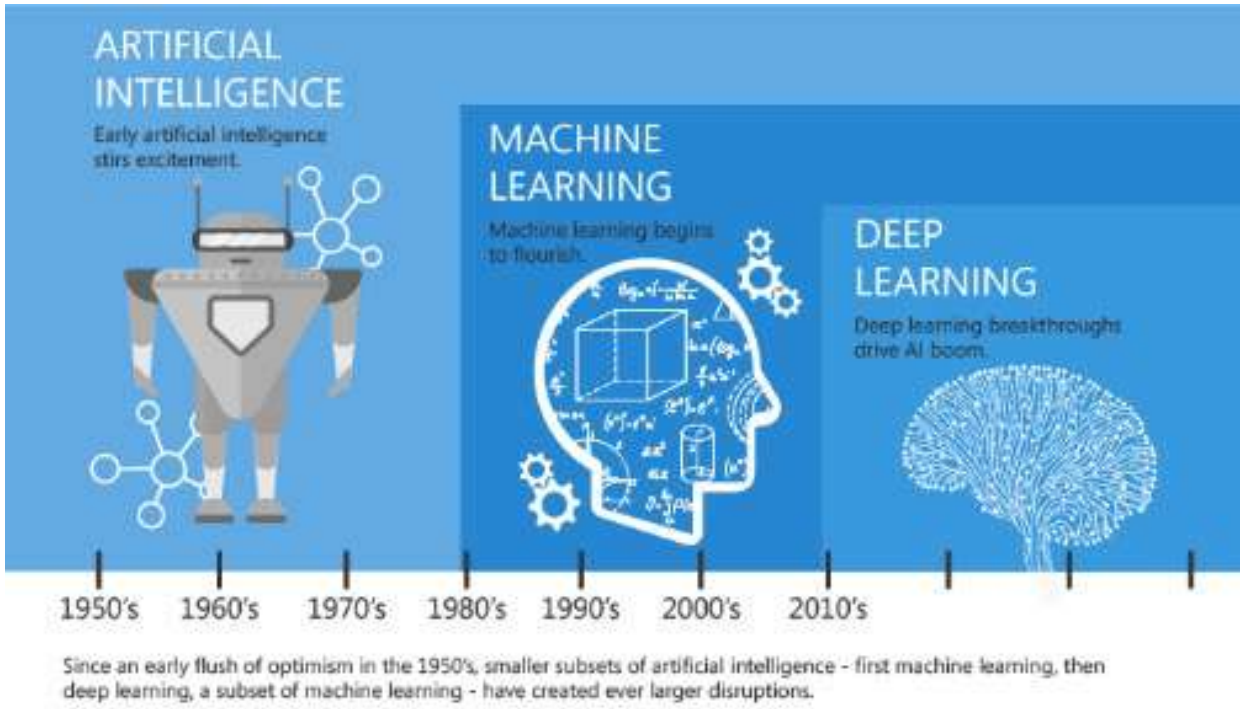
AI, Machine Learning, and Deep Learning

- **Machine Learning (ML)** is a subfield in AI focusing on making to learn by itself **without human intervention**.

"Machine learning is the sc*ience of getting computers to act without being explicitly programmed*." — *Stanford University*

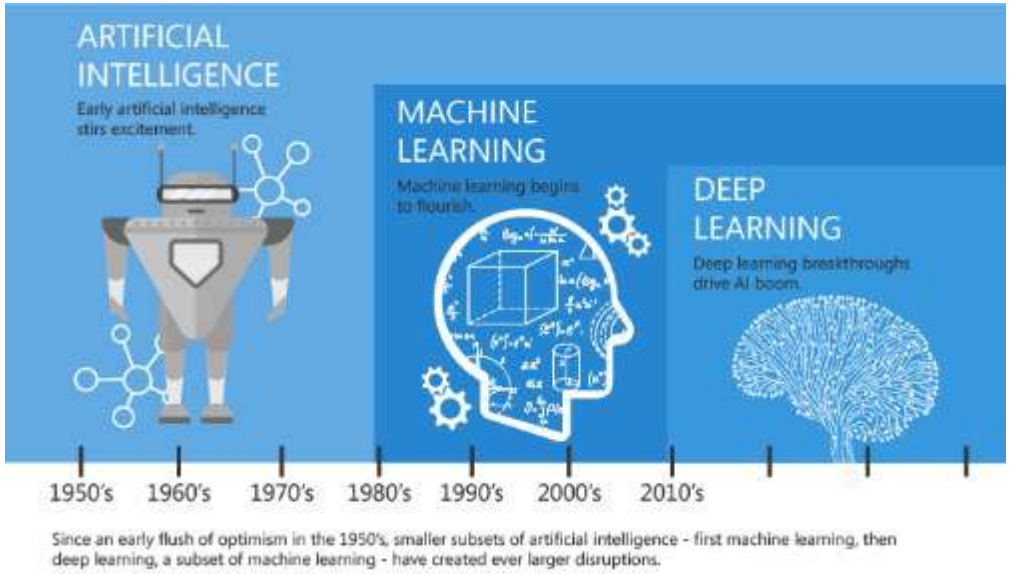
https://towardsdatascience.com/cousins-of-artificial-intelligence-dda4edc27b55

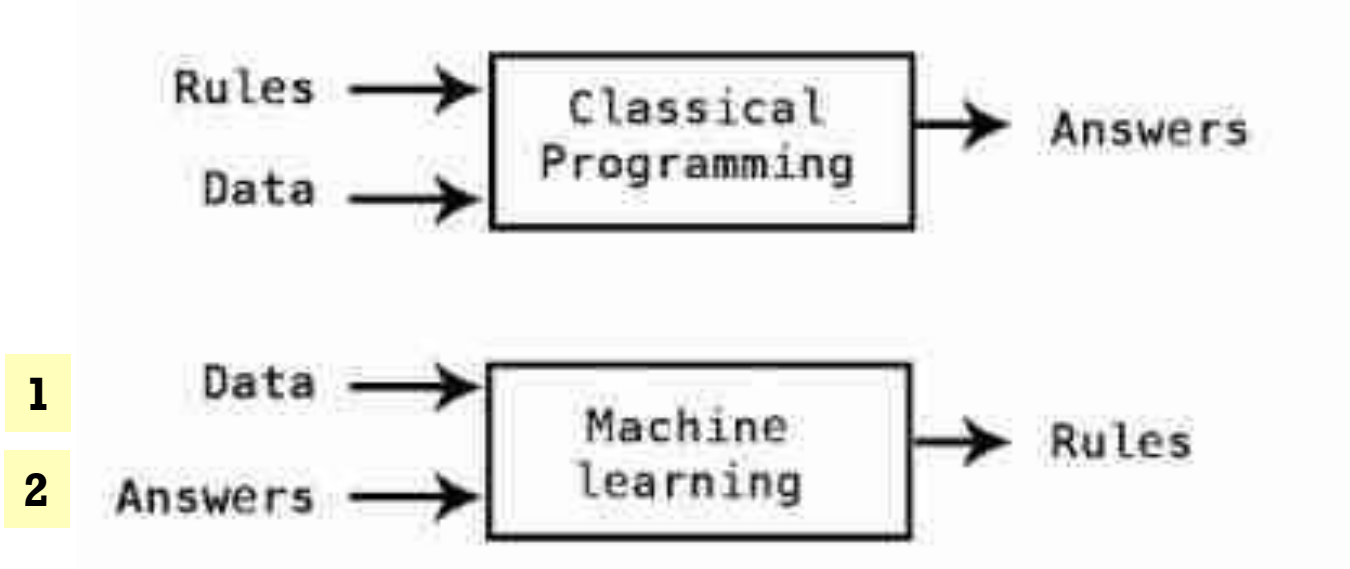




AI = Automation

- 1) Rule-based AI (Symbolic AI)
- 2) Machine Learning





https://mc.ai/machine-learning-basics-artificial-intelligence-machine-learning-and-deep-learning/

Sample of ML Application: Finding Waldo

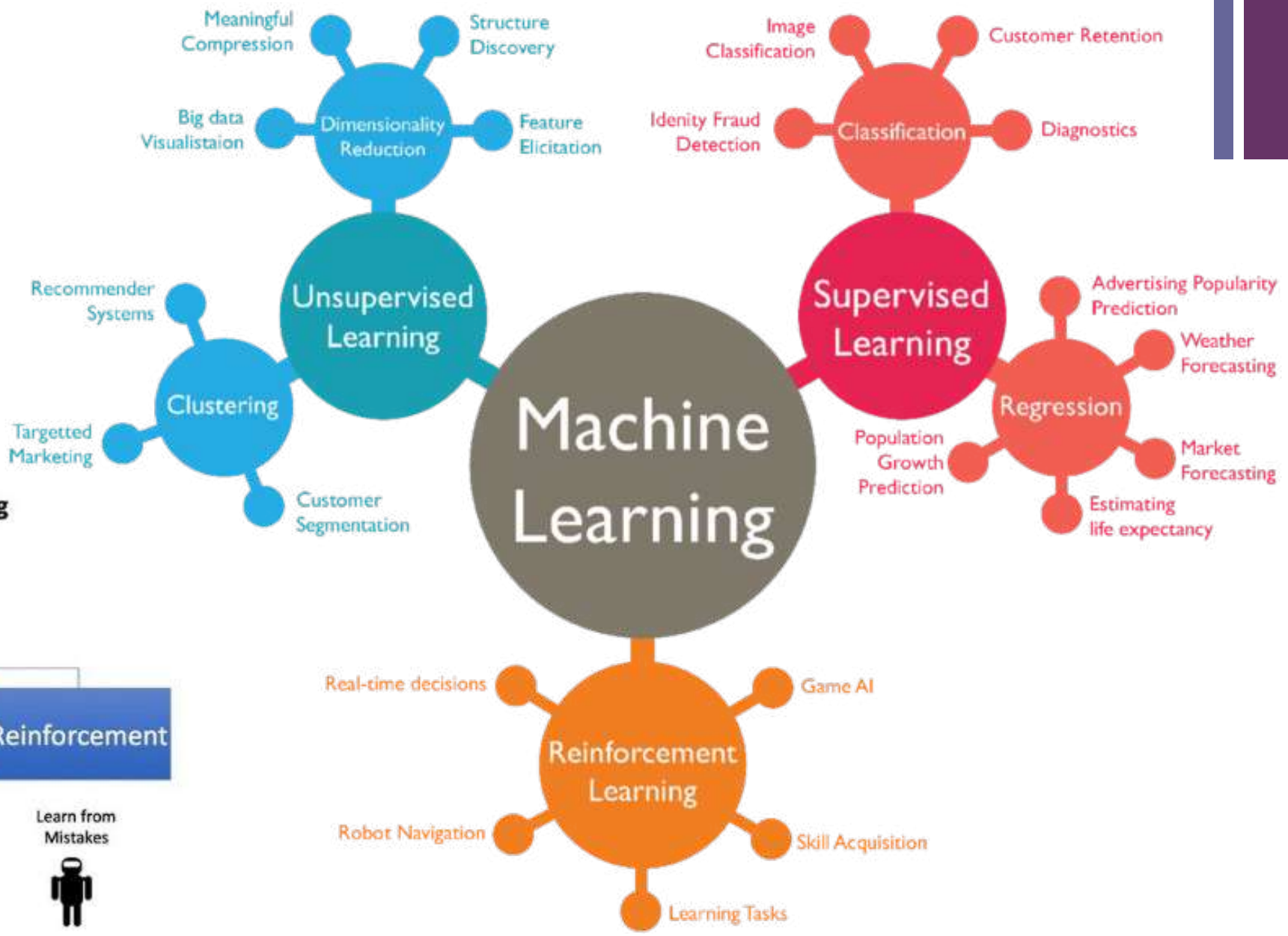
THERE'S WALDO

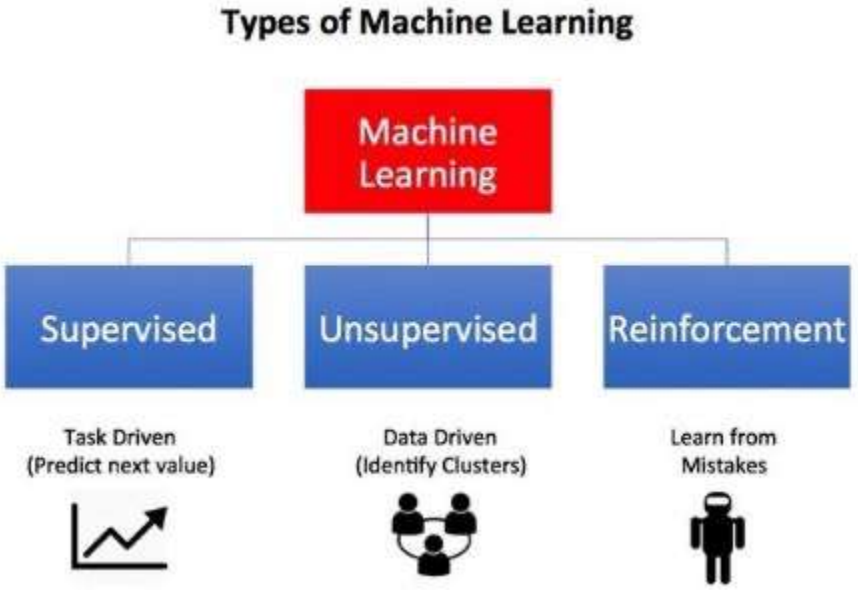
FINDING WALDOS WITH GOOGLE AUTOML VISION

PROTOTYPE v0.1
redpepper.land/innovation

rp

Machine Learning (cont.)





scikit-learn

Machine Learning in Python

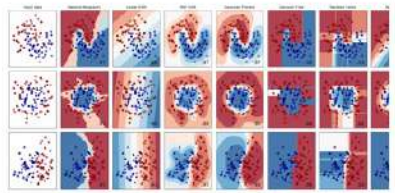
Getting Started | What's New in 0.22.2 | GitHub

- 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...

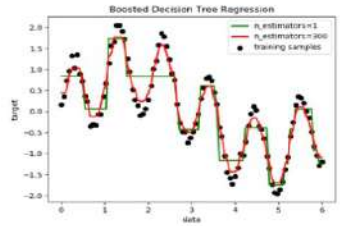


Examples

Regression

Predicting a continuous-valued attribute associated with an object.

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

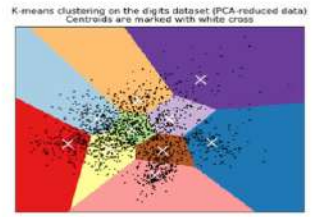


Examples

Clustering

Automatic grouping of similar objects into sets.

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

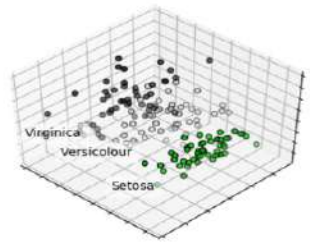


Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: k-Means, feature selection, non-negative matrix factorization, and more...

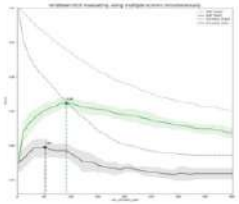


Examples

Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning
Algorithms: grid search, cross validation, metrics, and more...

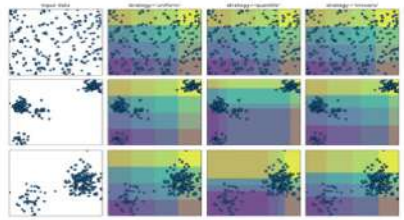


Examples

Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: preprocessing, feature extraction, and more...



Examples

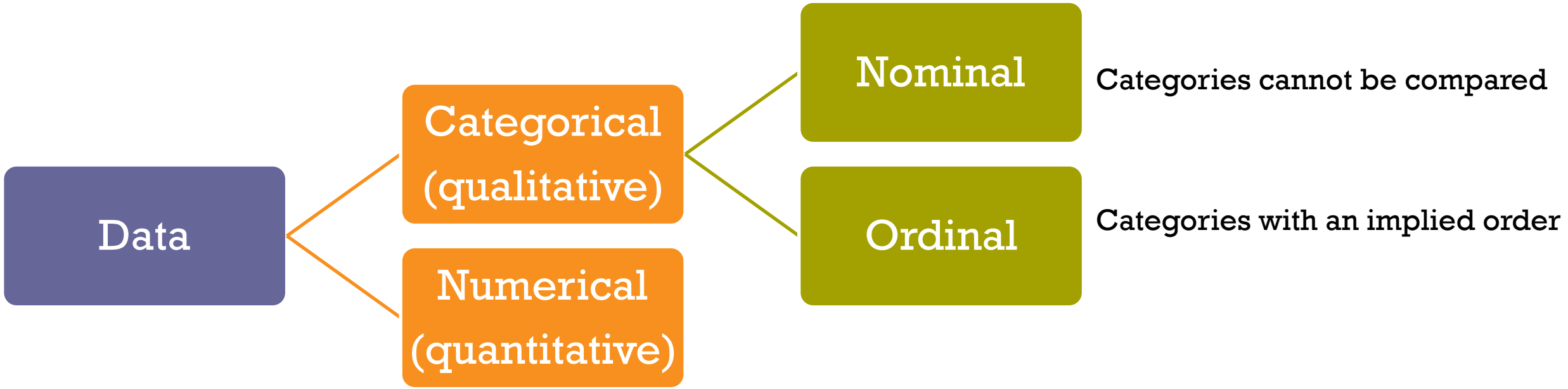
Terminology: Data table

inputs | target

Age	Income	Gender	Province	Purchase
25	25,000	Female	Bangkok	Yes
35	50,000	Female	Nontaburi	Yes
32	35,000	Male	Bangkok	No

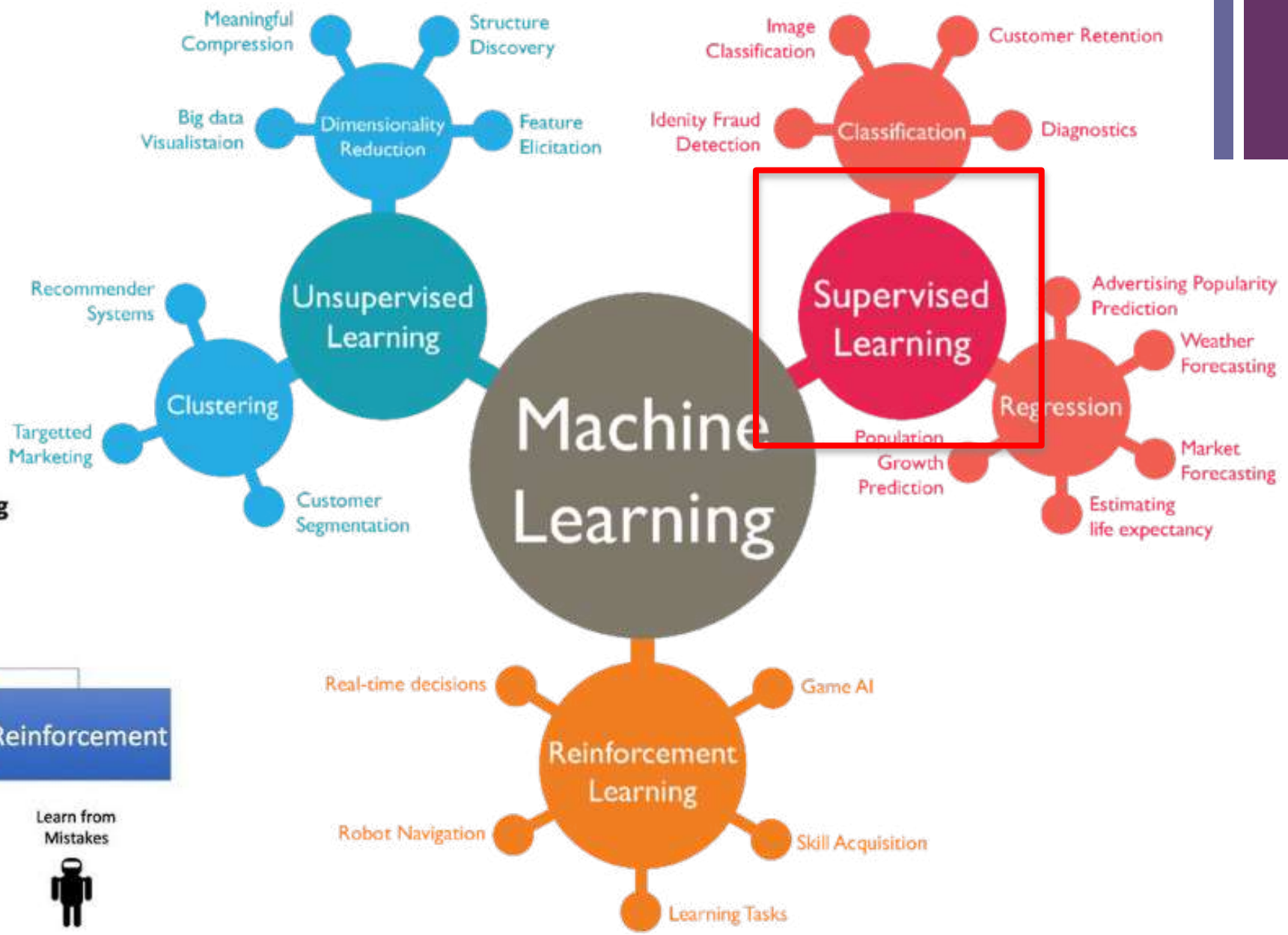
- Row
  - Example, instance, case, observation, subject
- Column
  - Feature, variable, attribute
- Input
  - Predictor, independent, explanatory variable
- Target
  - Output, outcome, response, dependent variable

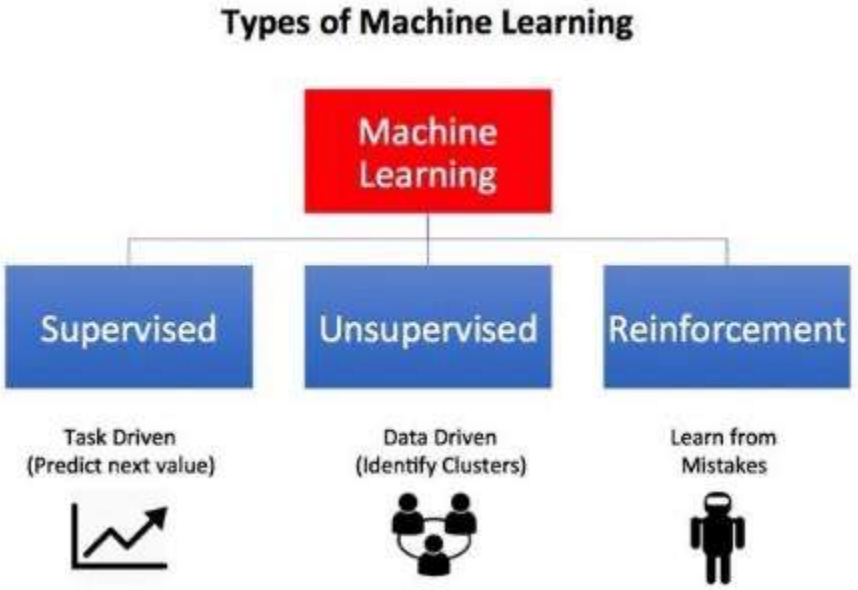
Terminology: Kinds of data



Supervised Learning (Predictive Task)

Machine Learning (cont.)





Task1: Supervised learning

Handcrafted features

Training Data

inputs | target

Age	Income	Gender	Province	Purchase
25	25,000	Female	Bangkok	Yes
35	50,000	Female	Nontaburi	Yes
32	35,000	Male	Bangkok	No

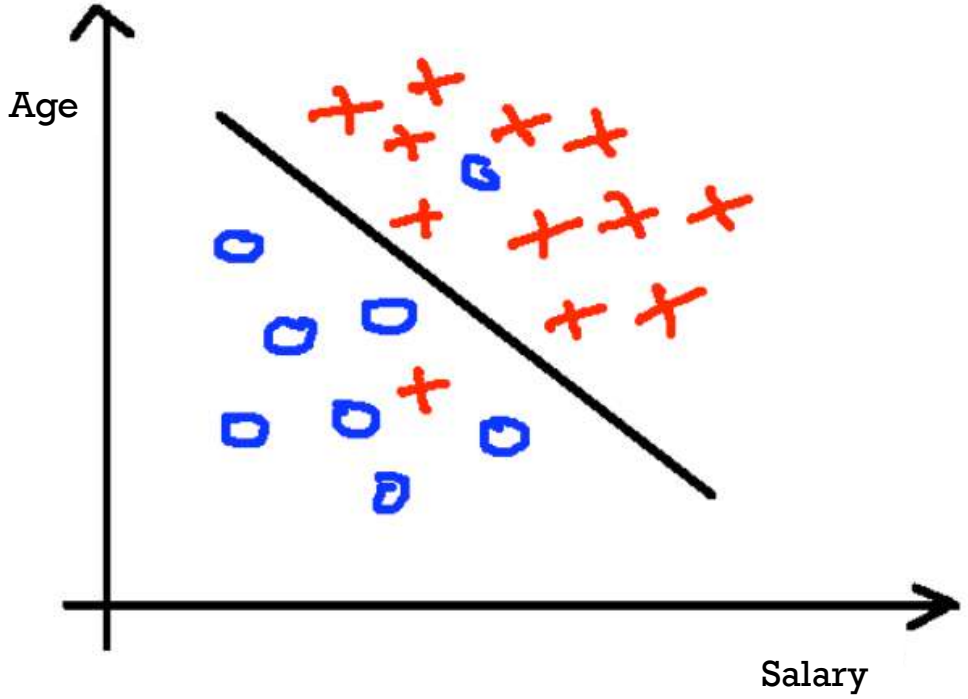
Testing Data

Age	Income	Gender	Province	Purchase
25	25,000	Female	Bangkok	?

Application: Direct Target Marketing

Classification: predicting a category

Logistic Regression

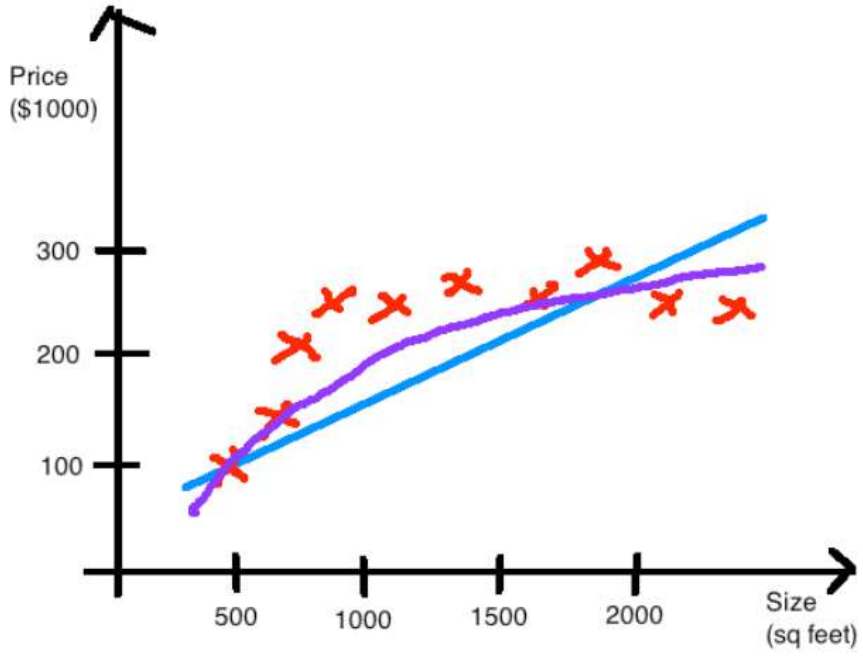


Predict targeted customers who tend to buy our product (yes/no)

- **Some techniques:**
  - Naïve Bayes
  - Decision Tree
  - Logistic Regression
  - Support Vector Machines
  - Neural Network
  - Ensembles
- **Sample Applications**
  - Database marketing
  - Fraud detection
  - Pattern detection
  - Churn customer detection

Regression: predict a continuous value

Linear Regression



Predict a sale price of each house

- **Some techniques:**
  - Linear Regression / GLM
  - Decision Trees
  - Support vector regression
  - Neural Network
  - Ensembles
- **Sample Applications**
  - Financial risk management
  - Revenue forecasting

Scikit-learn: Machine learning library in Python

- Provides many machine learning tools with a common Estimator interface
- Built in helpers for common ML tasks (e.g., metrics, preprocessing)
- Easily combine algorithms to make a complex pipeline
- Relies heavily on numpy and scipy, often used with pandas

How do you pronounce the project name?

sy-kit learn. sci stands for science!

Why scikit?

There are multiple scikits, which are scientific toolboxes built around SciPy. You can find a list at https://scikits.appspot.com/scikits. Apart from scikit-learn, another popular one is scikit-image.

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, ... — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ... — Examples

Clustering

Automatic grouping of similar objects into sets.

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

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics. — Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction. — Examples

http://scikit-learn.org/stable/index.html

Estimator Interface

Decision Trees

We'll start just by training a single decision tree.

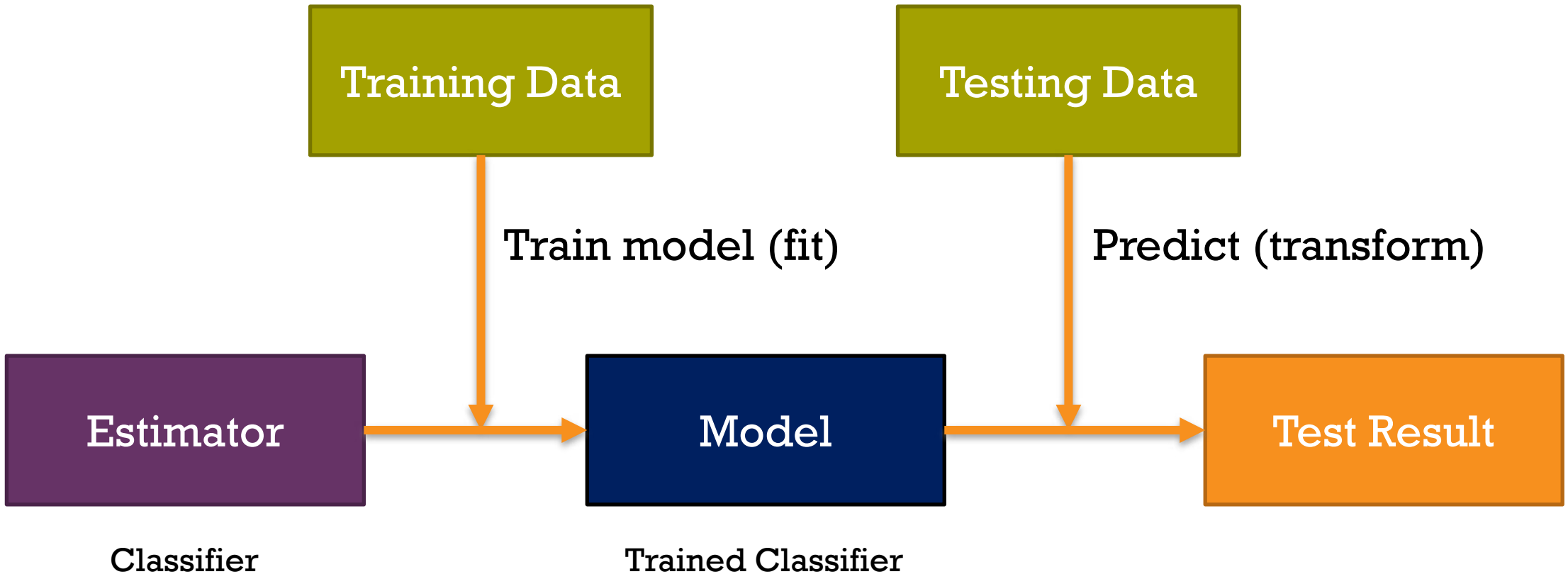
```
In [8]: from sklearn.tree import DecisionTreeClassifier

In [9]: dtree = DecisionTreeClassifier(min_samples_leaf=10, criterion='entropy')

In [10]: dtree.fit(X_train,y_train)

Out[10]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
                                max_features=None, max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=10, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                                splitter='best')
```

Prediction and Evaluation

Let's evaluate our decision tree.

```
[11]: predictions = dtree.predict(X_test)

[12]: from sklearn.metrics import classification_report,confusion_matrix

[13]: print(classification_report(y_test,predictions))
```

	precision	recall	f1-score	support
absent	0.85	0.85	0.85	20
present	0.40	0.40	0.40	5
avg / total	0.76	0.76	0.76	25



Training Phase

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()      # Get some data
X_train, X_test, y_train, y_test = train_test_split(
        cancer.data, cancer.target,
        stratify=cancer.target, random_state=1337)

tree = DecisionTreeClassifier(random_state=7331)
tree.fit(X_train, y_train)  # Learn a Decision Function
```

Ref: C. Travis Johnson, Mines Linux Users Group on April 27, 2017

Testing Phase (Prediction)

```
y_pred = tree.predict(X_test)
```

```
>>> from sklearn.metrics import classification_report
>>> y_true = [0, 1, 2, 2, 2]
>>> y_pred = [0, 0, 2, 2, 1]
>>> target_names = ['class 0', 'class 1', 'class 2']
>>> print(classification_report(y_true, y_pred, target_names=target_names))
              precision    recall  f1-score   support

     class 0       0.50      1.00      0.67         1
     class 1       0.00      0.00      0.00         1
     class 2       1.00      0.67      0.80         3

   micro avg       0.60      0.60      0.60         5
   macro avg       0.50      0.56      0.49         5
weighted avg       0.70      0.60      0.61         5
```

Prediction Algorithms

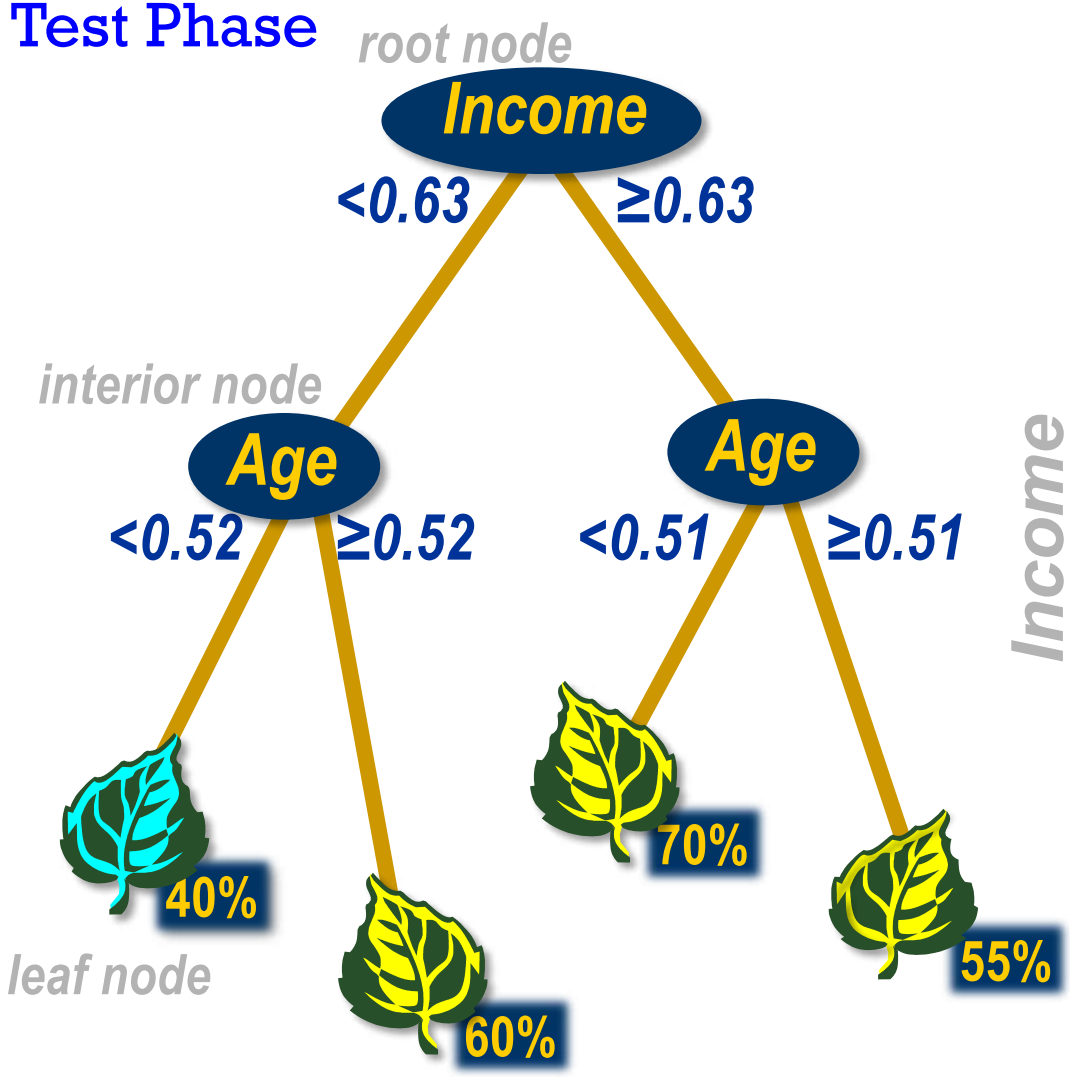
- Decision Tree
- (Logistic) Regression
- Neural Networks (NN)
- kNN
- Support Vector Machine
- Deep Learning

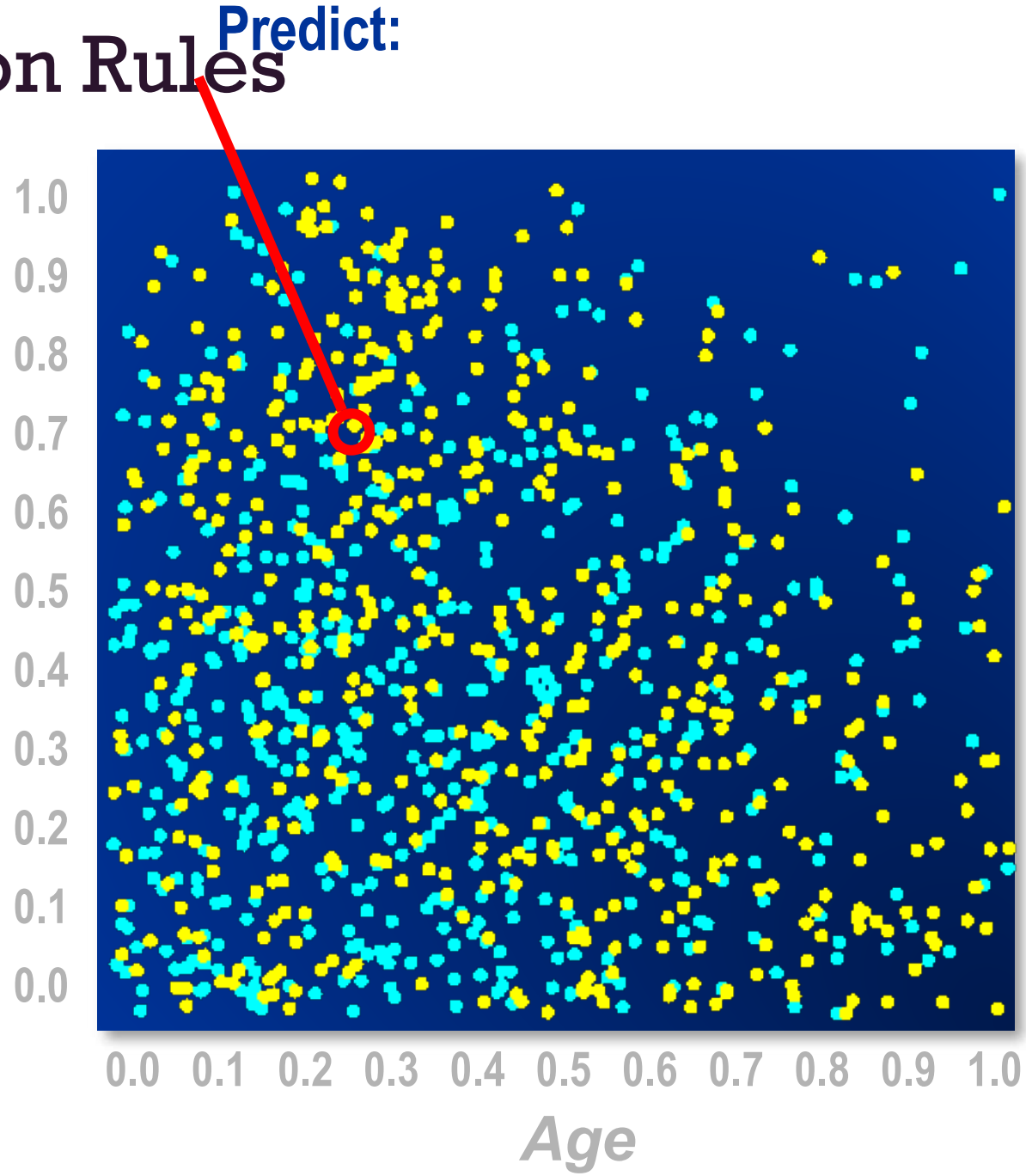
+

1) Decision Tree

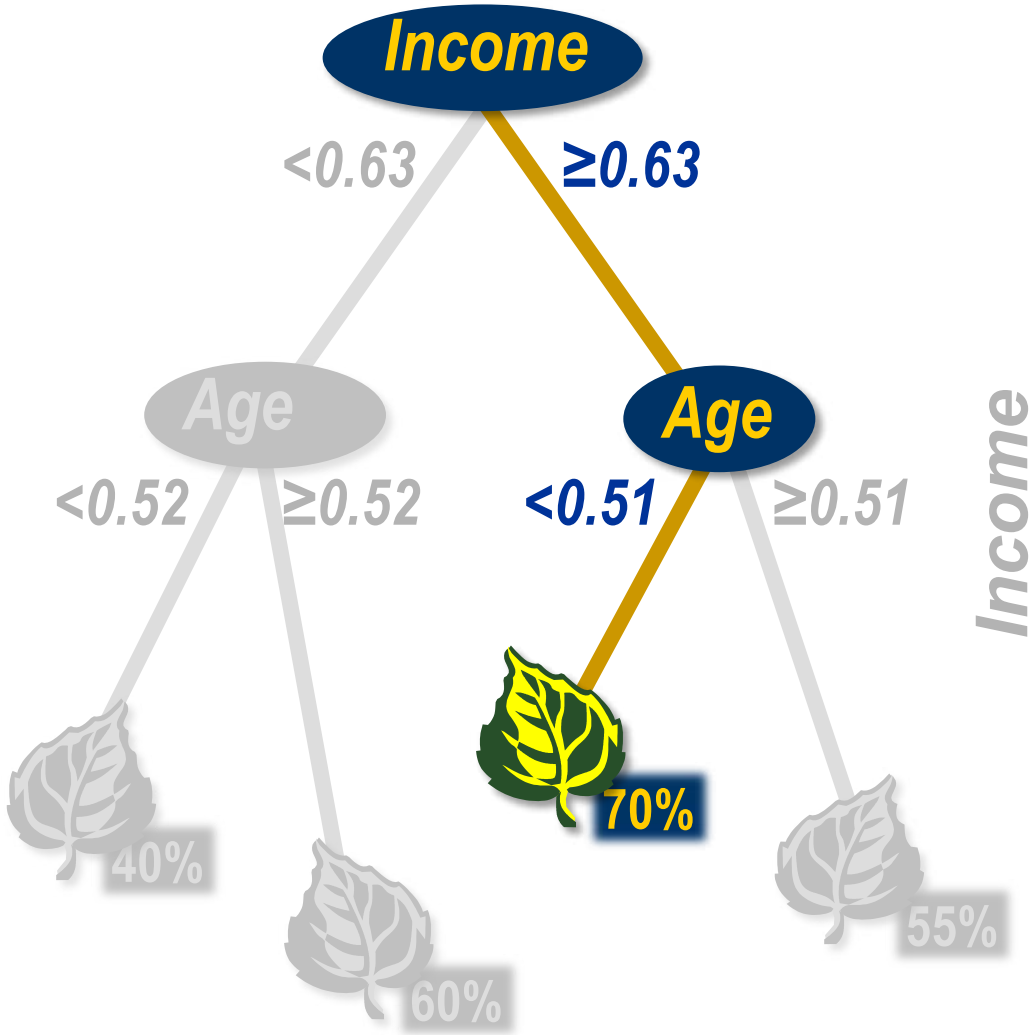
1) Decision Tree Prediction Rules

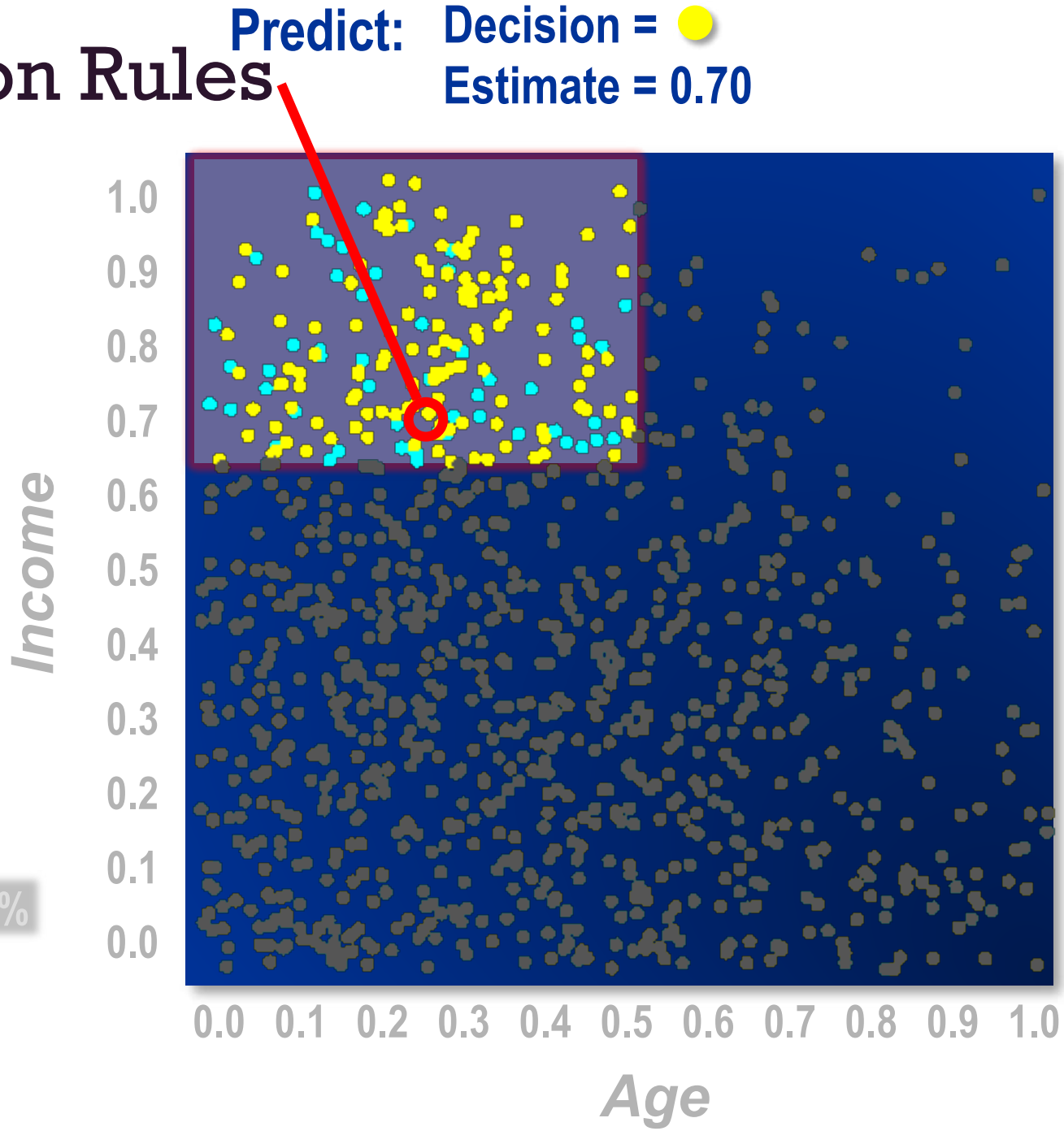
Test Phase





Decision Tree Prediction Rules





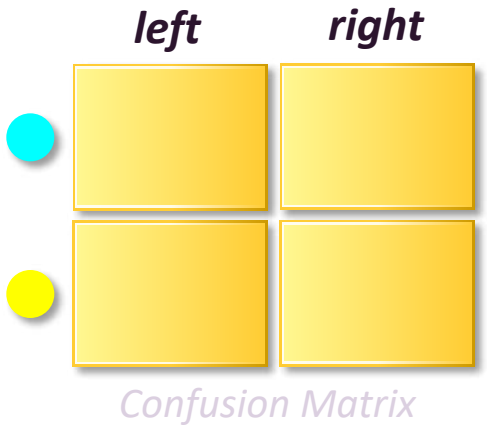
Model Essentials: Decision Trees

- Predict cases.
- Select useful predictors.

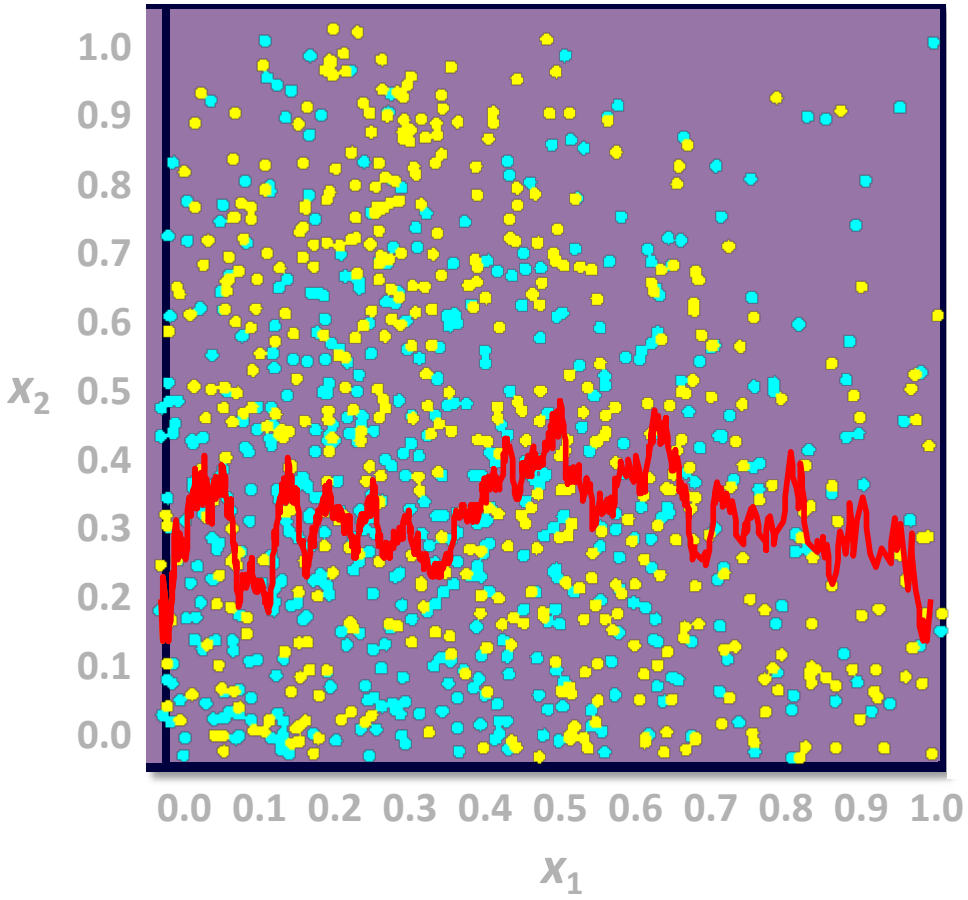
Prediction rules

Split search

Decision Tree Split Search

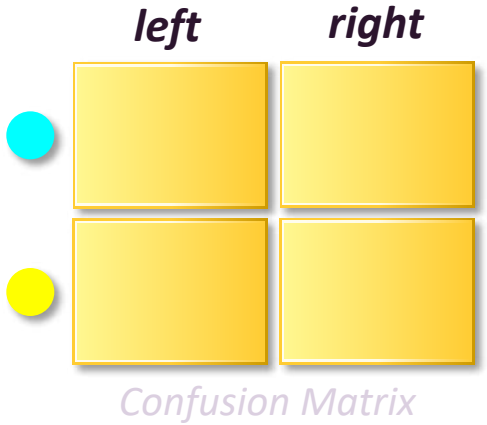


Calculate information gain on partitions on input x_1.

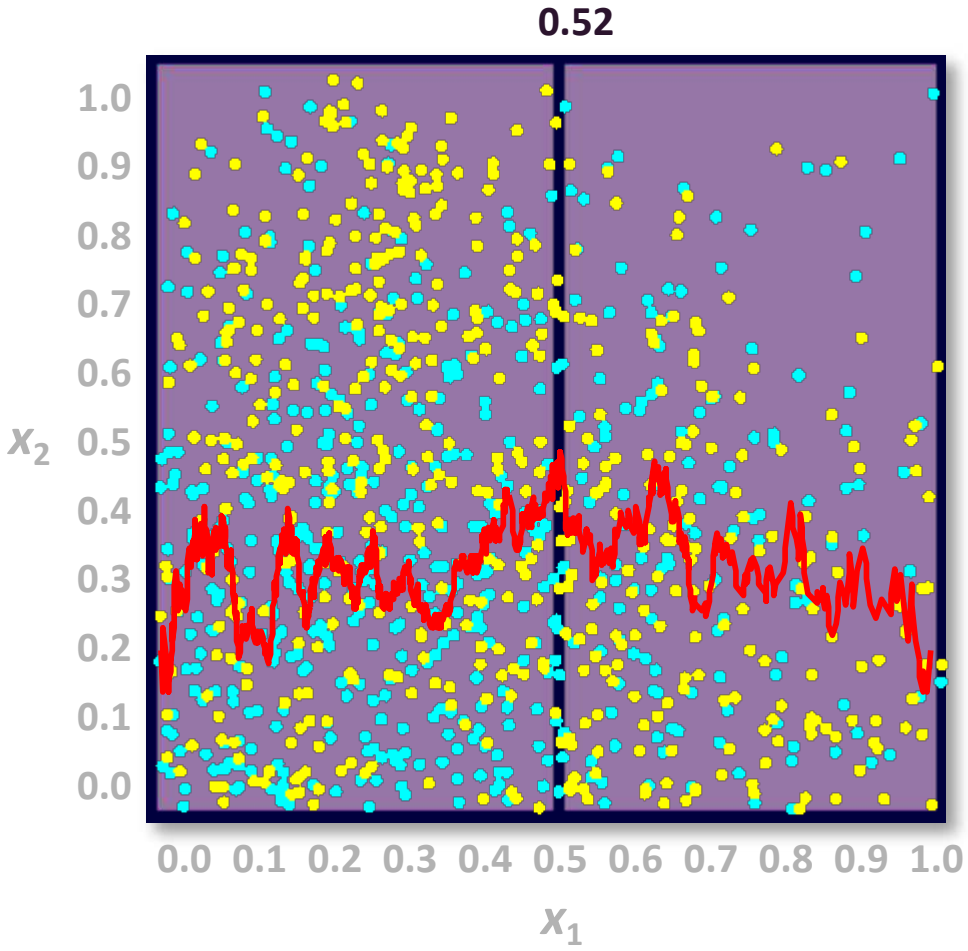


...

Decision Tree Split Search



Calculate the gain of every partition on input x_1.

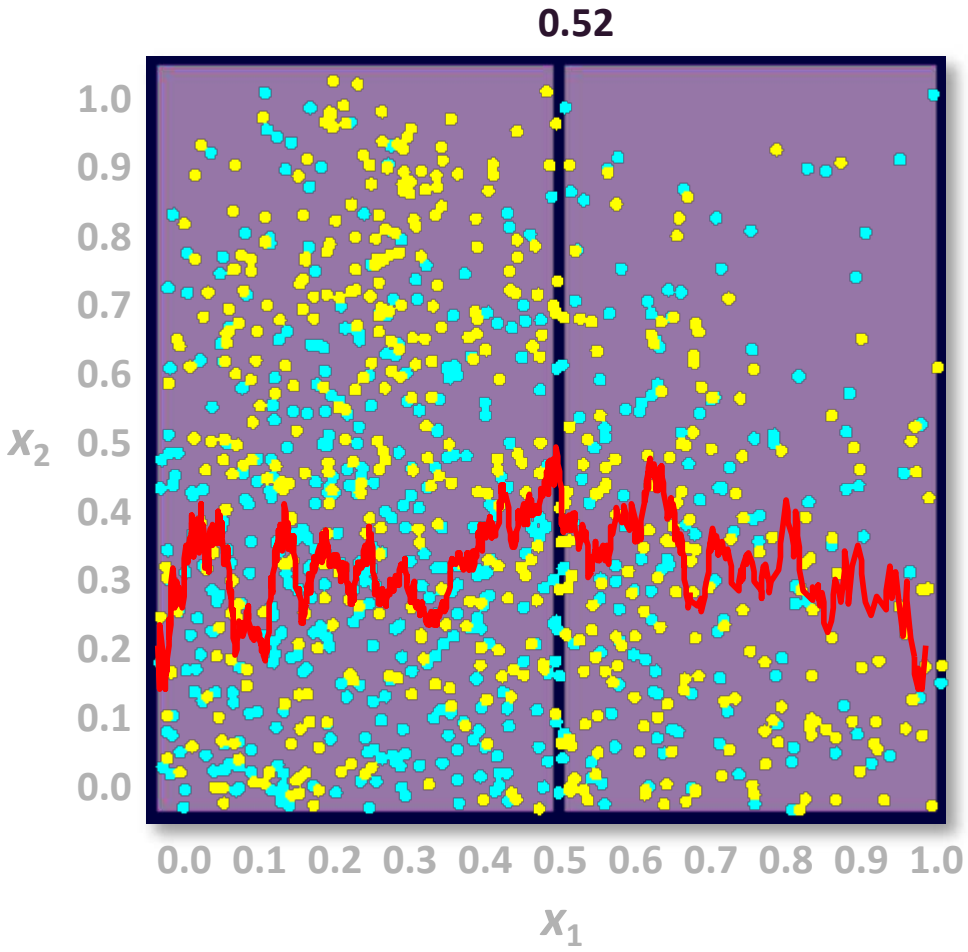


...

Decision Tree Split Search

	left	*right*
●	53%	42%
●	47%	58%

max gain(x_1)
0.0112



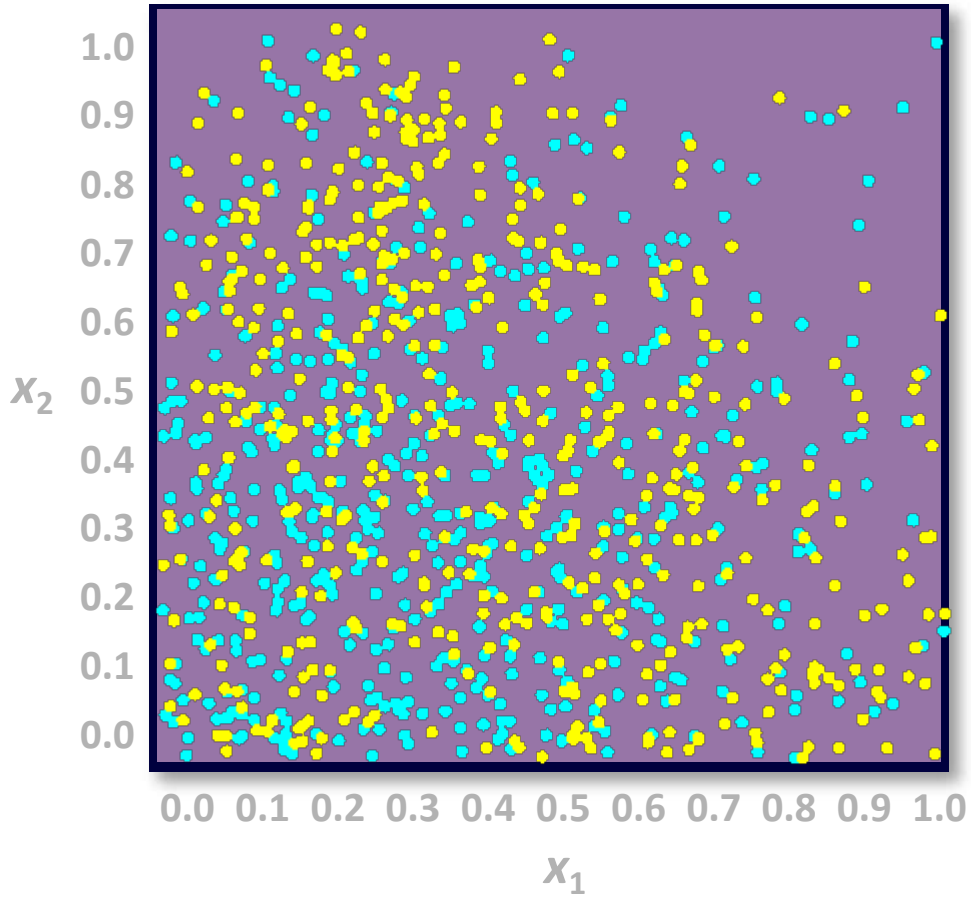
Select the partition with the maximum gain.

...

Decision Tree Split Search

	left	*right*
	53%	42%
	47%	58%

max Gain(x_1) 0.0112



Repeat for input x_2.

...

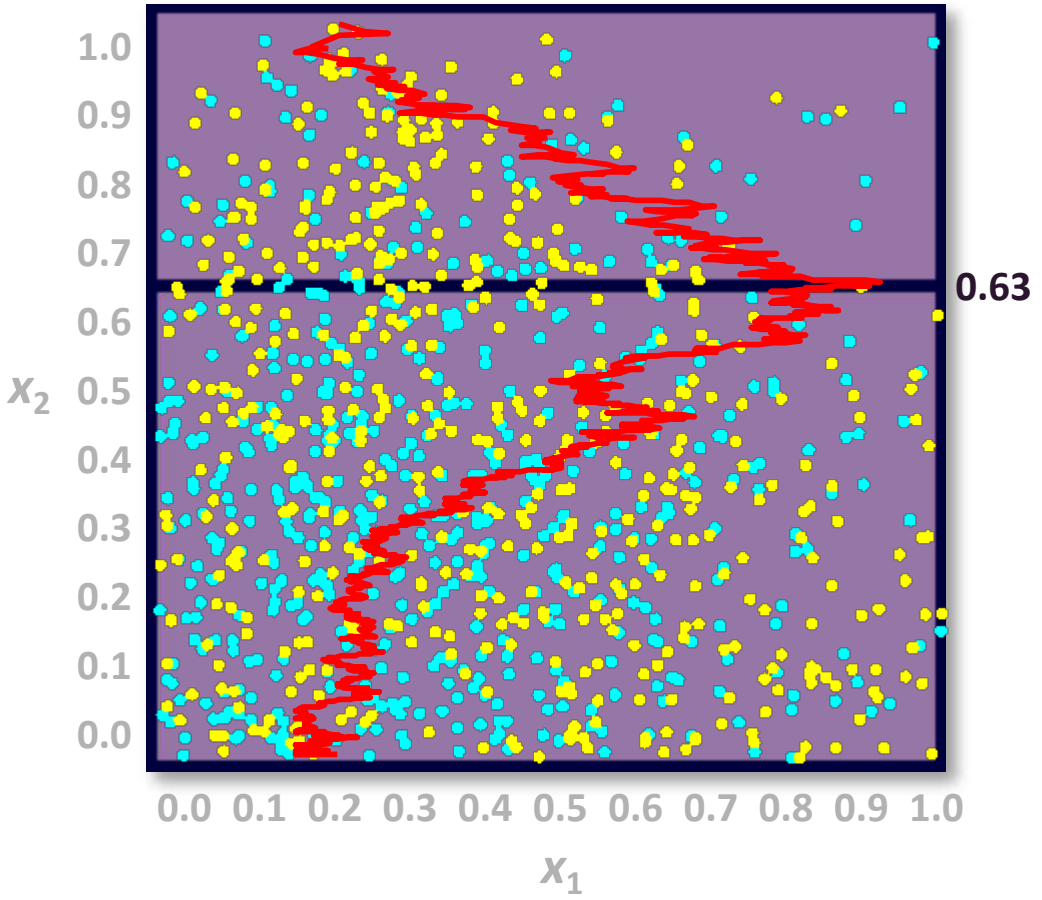
Decision Tree Split Search

	left	*right*
●	53%	42%
●	47%	58%

max gain(x_1) 0.0112

	bottom	*top*
●	54%	35%
●	46%	65%

max gain(x_2) 0.0273



...

Decision Tree Split Search

	left	*right*
●	53%	42%
●	47%	58%

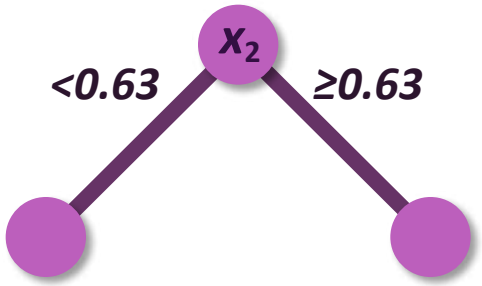
max gain(x_1)
0.0112

	bottom	*top*
●	54%	35%
●	46%	65%

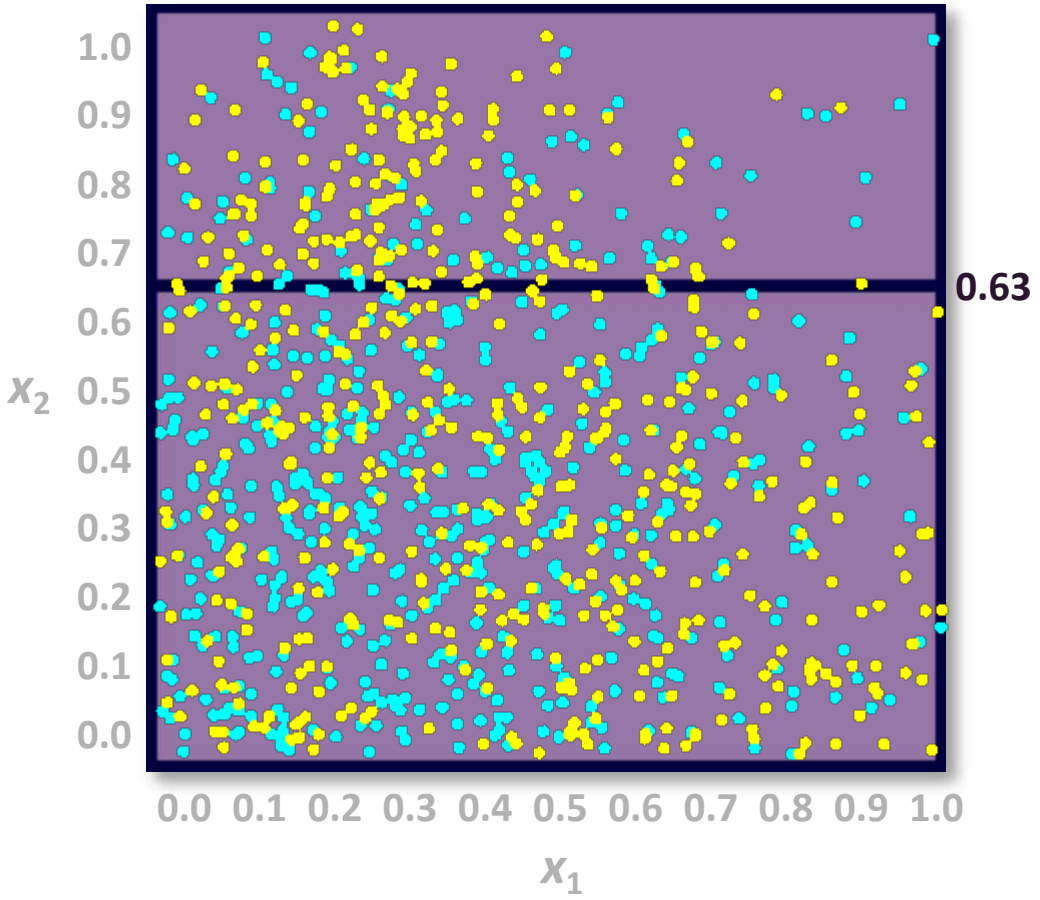
max gain(x_2)
0.0273

...

Decision Tree Split Search

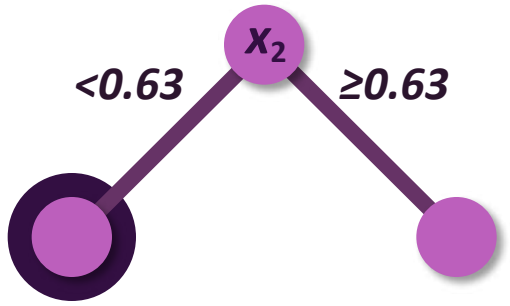


Create a partition rule from the best partition across all inputs.

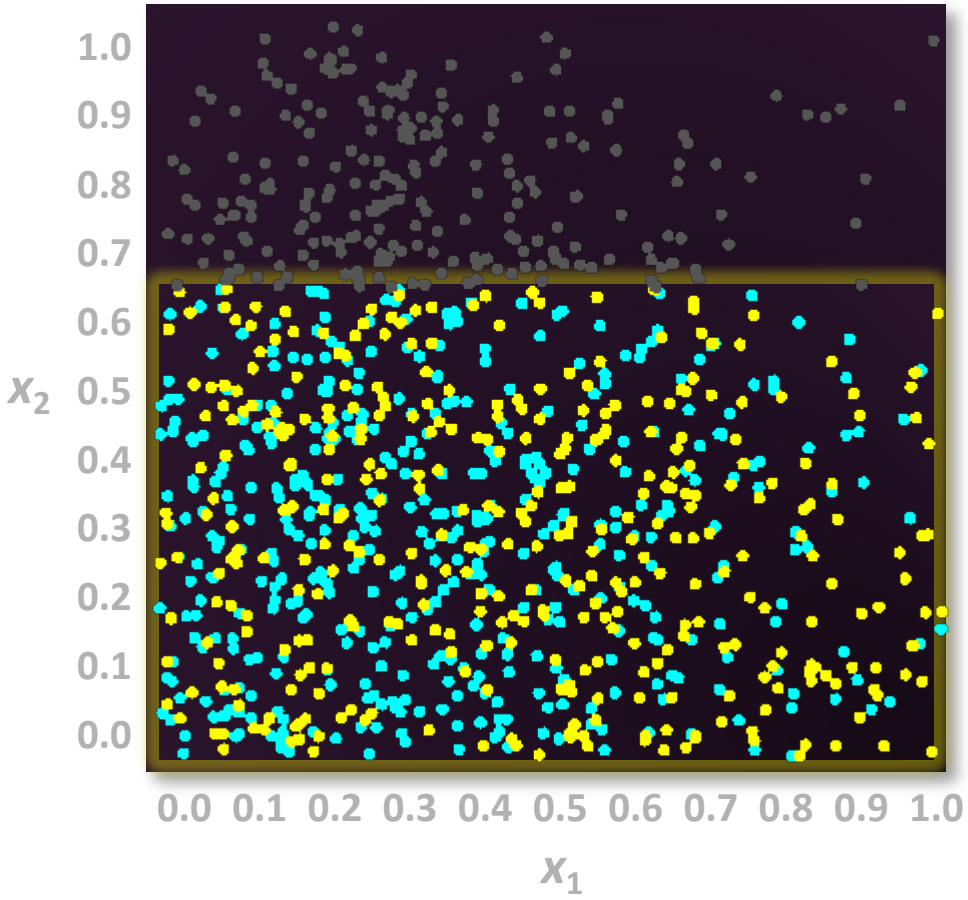


...

Decision Tree Split Search



Repeat the process in each subset.

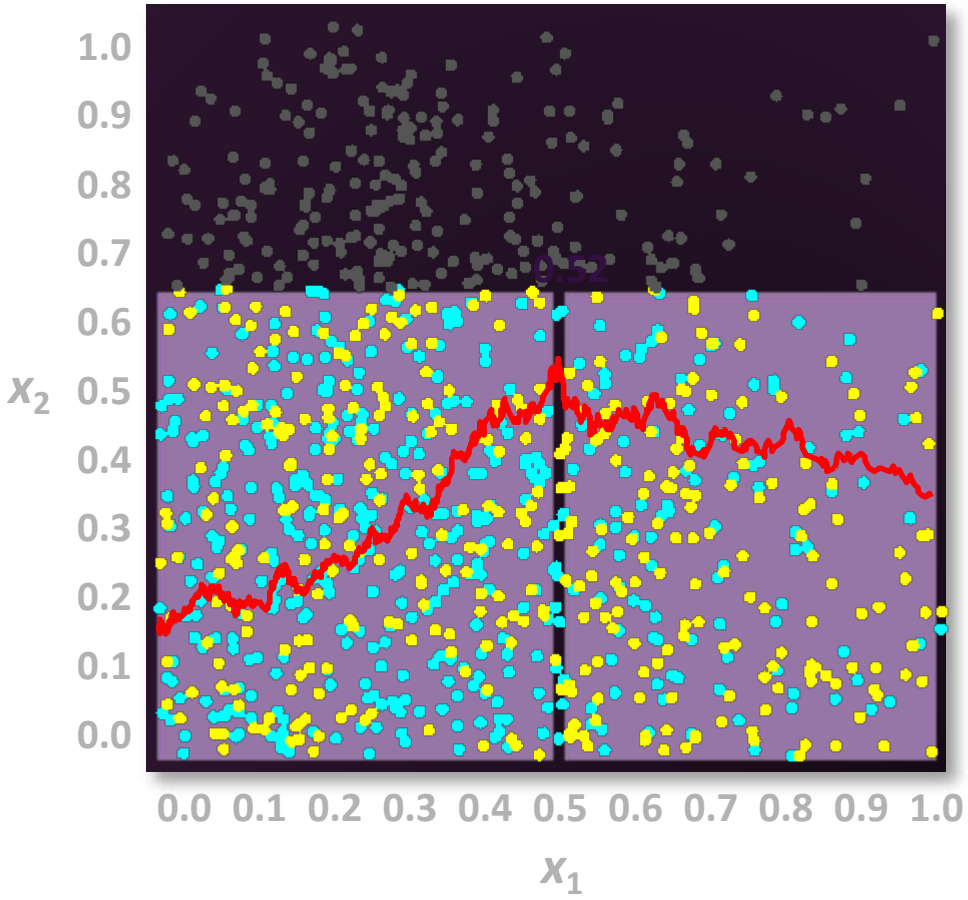


...

Decision Tree Split Search

	left	*right*
●	61%	55%
●	39%	45%

max gain(x_1)
0.0203



...

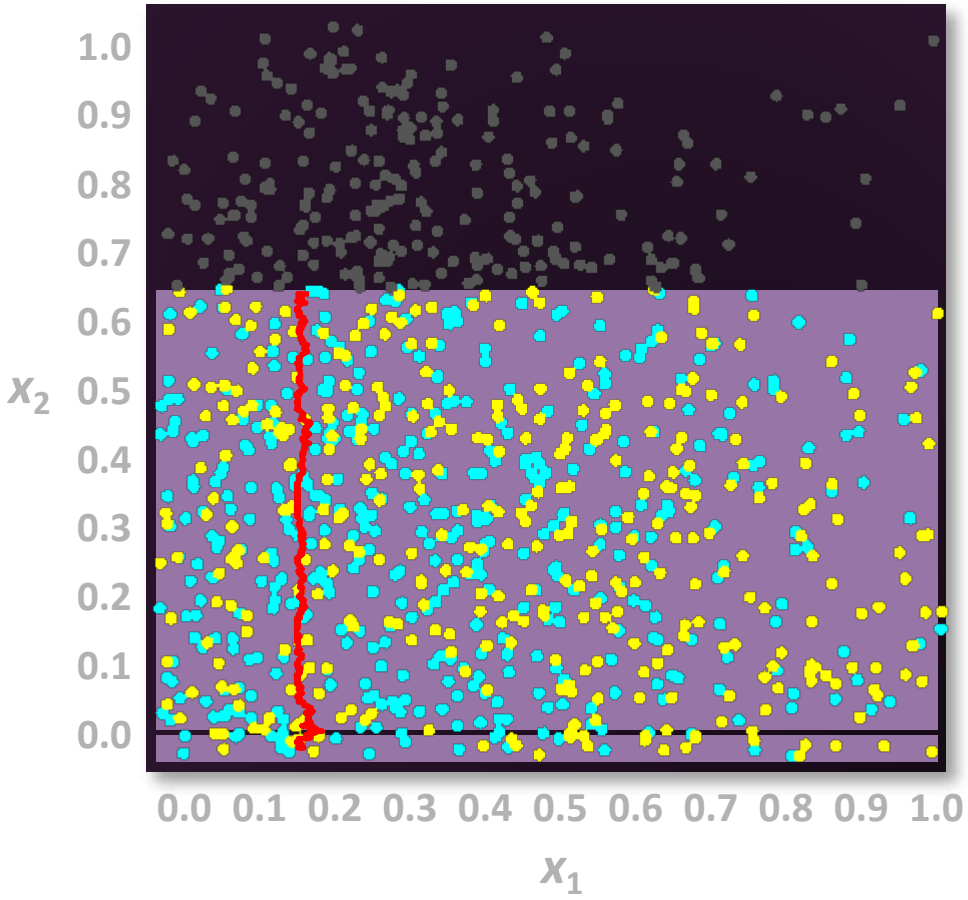
Decision Tree Split Search

	left	*right*
●	61%	55%
●	39%	45%

max gain(x_1) 0.0203

	bottom	*top*
●	38%	55%
●	62%	45%

max gain(x_2) 0.019



...

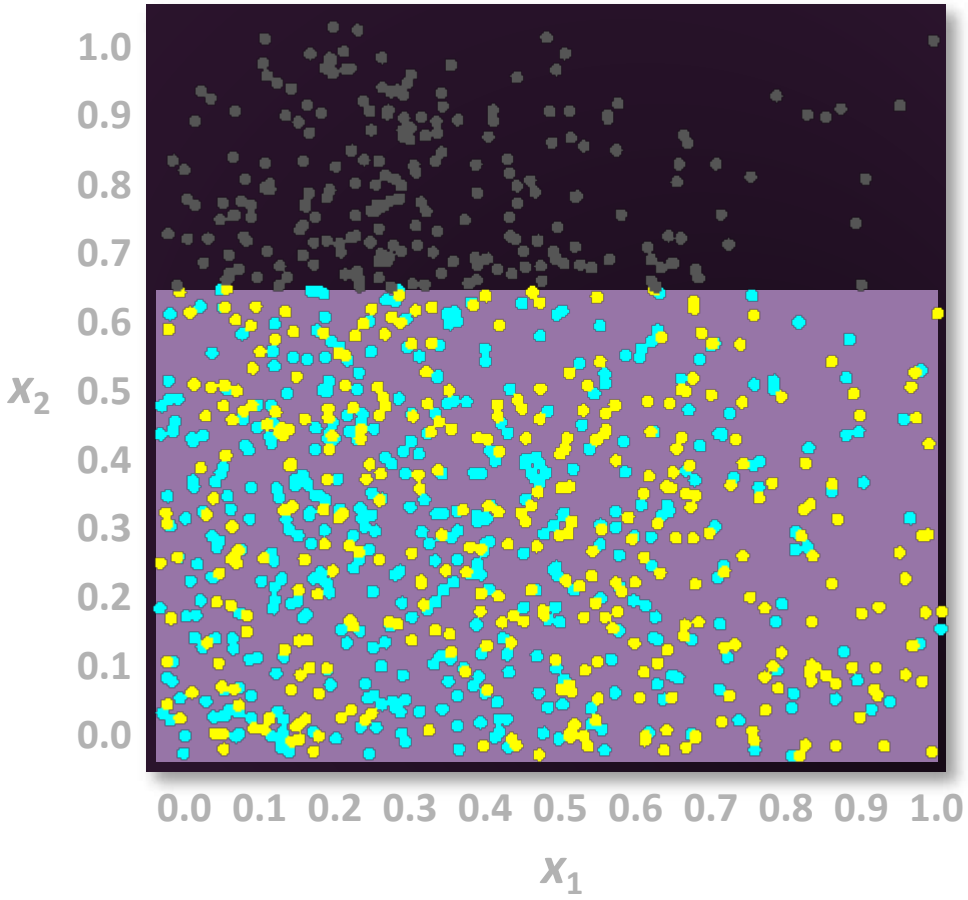
Decision Tree Split Search

	left	*right*
●	61%	55%
●	39%	45%

max gain(x_1) 0.0203

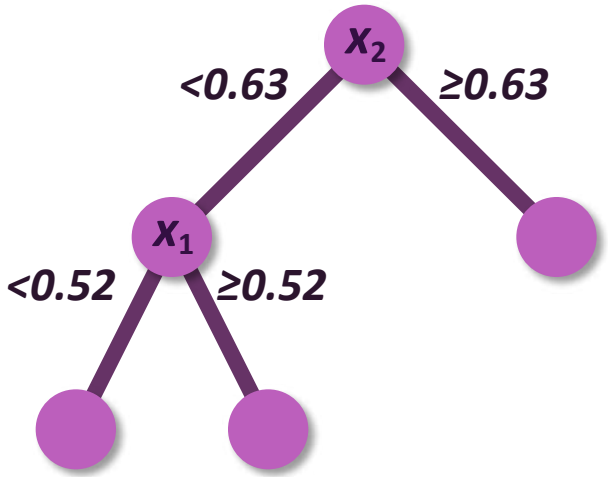
	bottom	*top*
●	38%	55%
●	62%	45%

max gain(x_2) 0.019

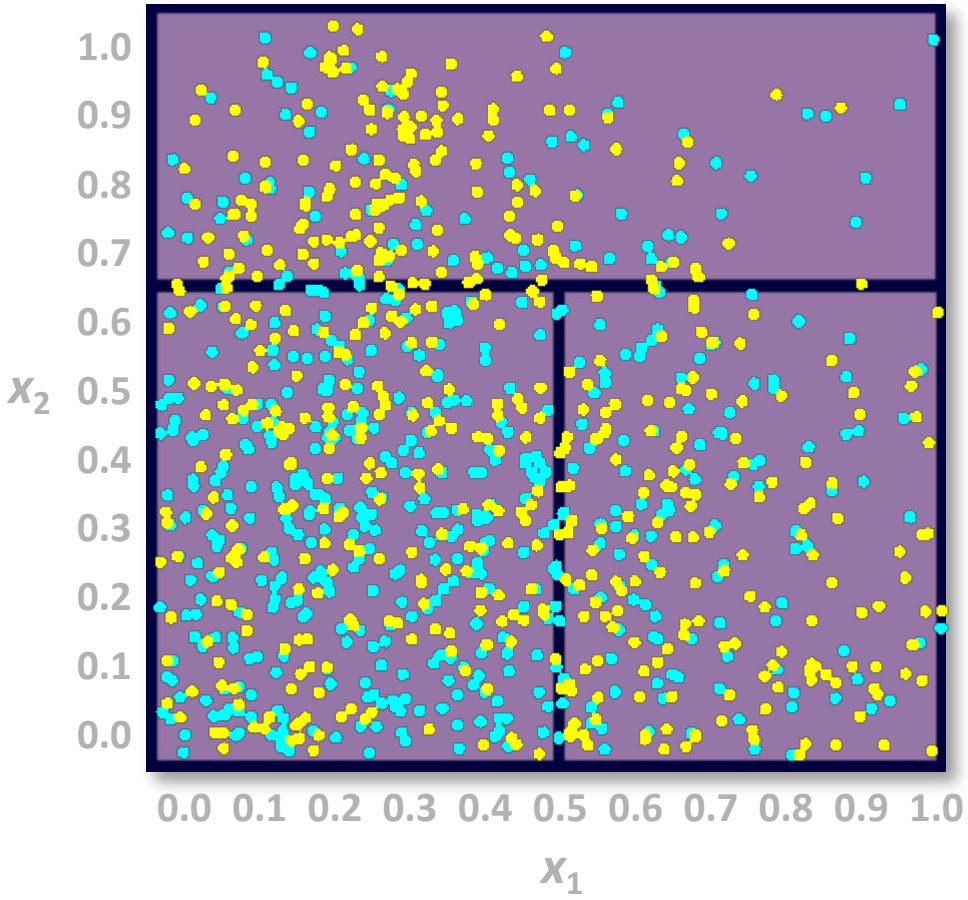


...

Decision Tree Split Search

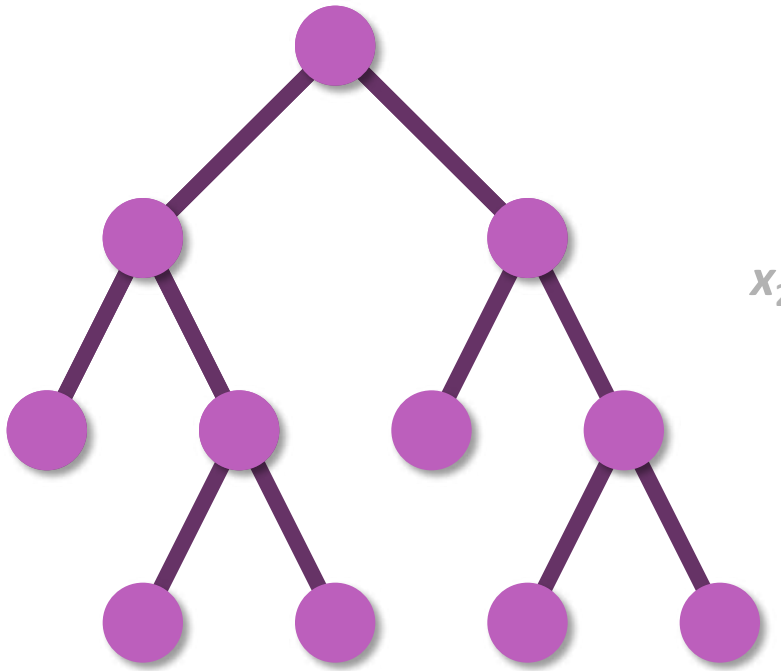


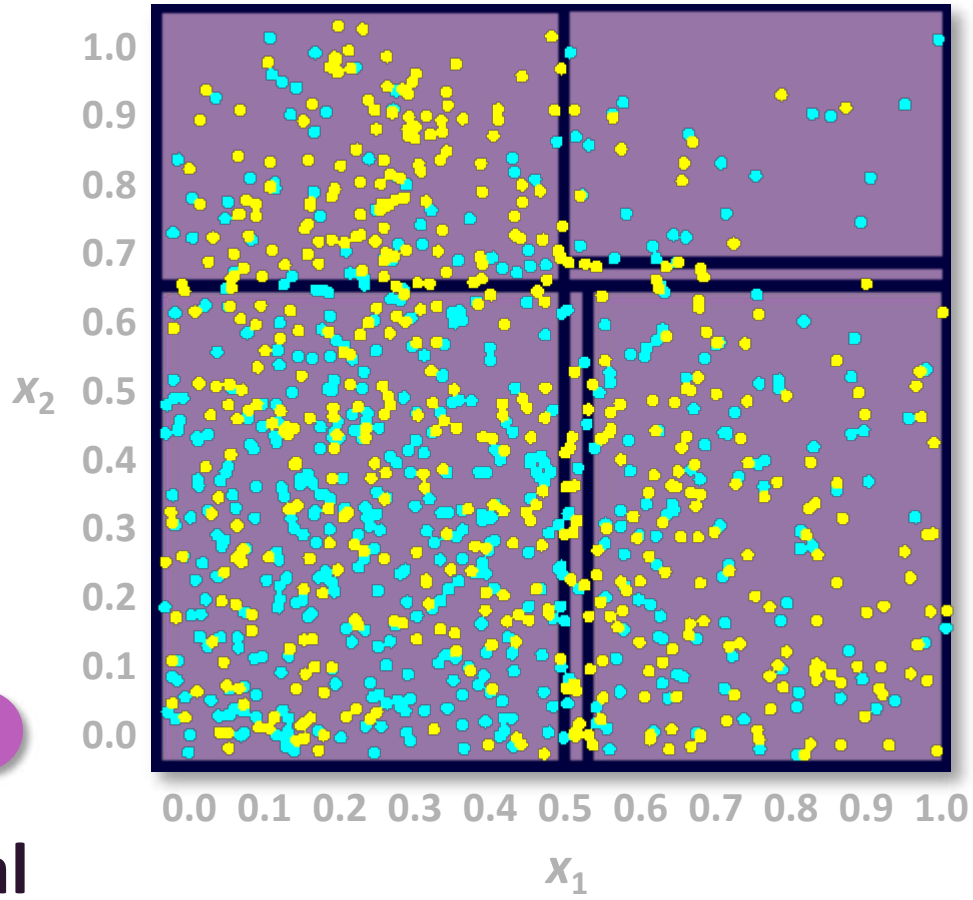
Create a second partition rule.



...

Decision Tree Split Search





Repeat to form a maximal tree.

1) Entropy (impurity)

- **Entropy** is a measure of disorder or uncertainty and the goal of machine learning models and general is to reduce uncertainty.

$$Entropy = \sum_{i=1}^{n} -p_i log_2 p_i$$

scipy.stats.entropy

scipy.stats.**entropy**(*pk*, *qk=None*, *base=None*, *axis=0*)

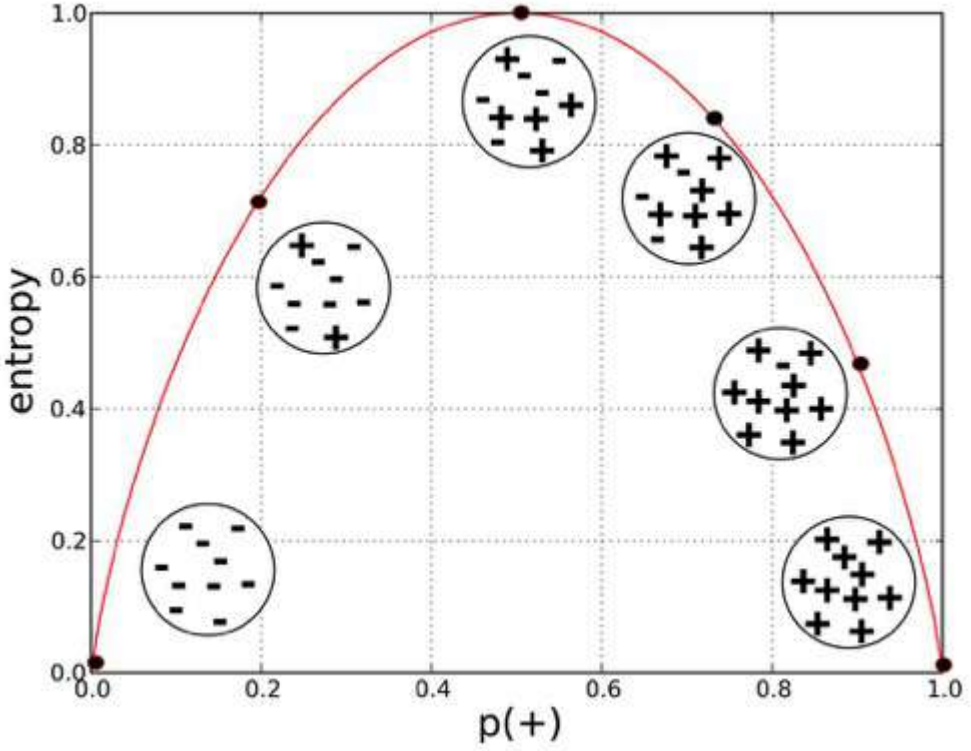
Calculate the entropy of a distribution for given probability values.

If only probabilities *pk* are given, the entropy is calculated as `S = -sum(pk * log(pk), axis=axis)`.

If *qk* is not None, then compute the Kullback-Leibler divergence `S = sum(pk * log(pk / qk), axis=axis)`.

This routine will normalize *pk* and *qk* if they don't sum to 1.

SciPy.org



Source: Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking

Information Gain
Before - After

- Which one is Better ?

Split w/ Age: 70
∑ Entropy: 0.350

Split w/ Age: 50
∑ Entropy: 0.348

- Information Gain: measure the reduction of this disorder in our target variable/class given additional information

$$InformationGain = Entropy(before) - \sum Entropy(after)$$

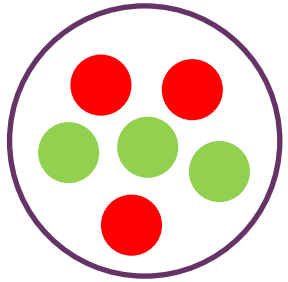
- "Before" = Entropy of Parent Node
  "After" = Entropy of Child Nodes

2) Gini Impurity
Gini Reduction = Before - After

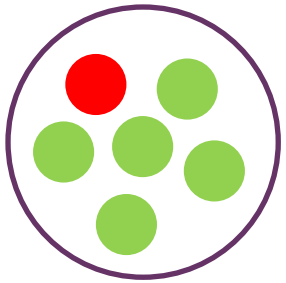
- Another way to measure how well a splitting feature.

$$Gini = 1 - \sum_{i=1}^{n} (P_i)^2$$

When P is the probability of class i in data-set.



$$Gini = 1 - ((3/6)^2 + (3/6)^2)$$
$$= 0.5$$

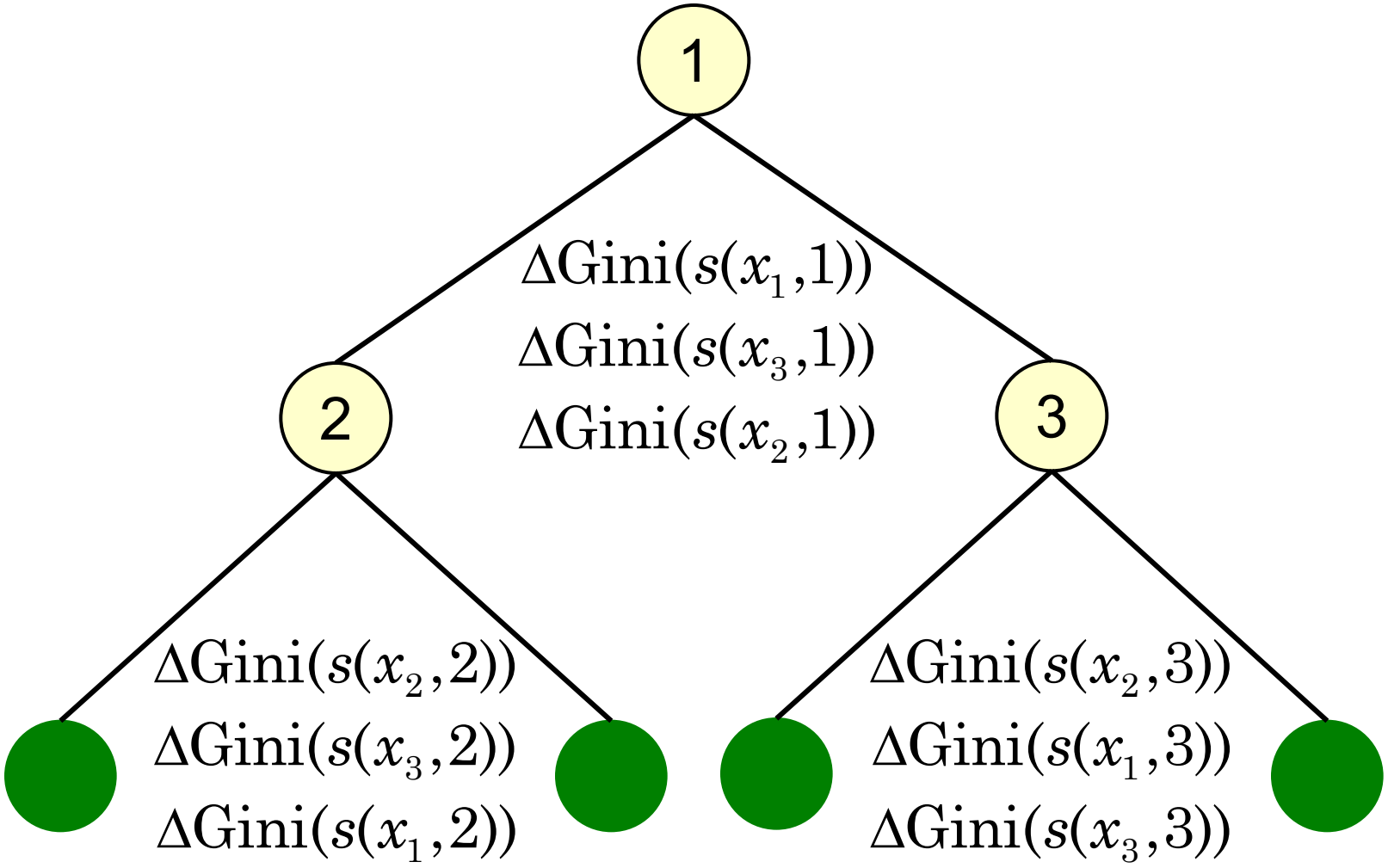


$$Gini = 1 - ((5/6)^2 + (1/6)^2)$$
$$= 0.28$$

- Easy to calculation, may take less time to build in large dataset.

Source: https://towardsdatascience.com/understanding-decision-tree-classification-with-scikit-learn-2ddf272731bd

Variable Importance



Types of Decision Tree

Algorithm	Splitting Measure
ID3	Entropy
C4.5	Gain Ratio
CART	Gini index
CHAID	Chi-squared test

Prev Up Next

scikit-learn 0.22.2
Other versions

Please cite us if you use the software.

sklearn.tree.DecisionTreeClassifier
Examples using sklearn.tree.DecisionTreeClassi

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='deprecated', ccp_alpha=0.0*) [source]

A decision tree classifier.

Read more in the User Guide.

Parameters:

criterion : *{"gini", "entropy"}, 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 : *{"best", "random"}, 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, 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 or float, default=2*
The minimum number of samples required to split an internal node:

Decision Tree with Regression

sklearn.tree.DecisionTreeRegressor

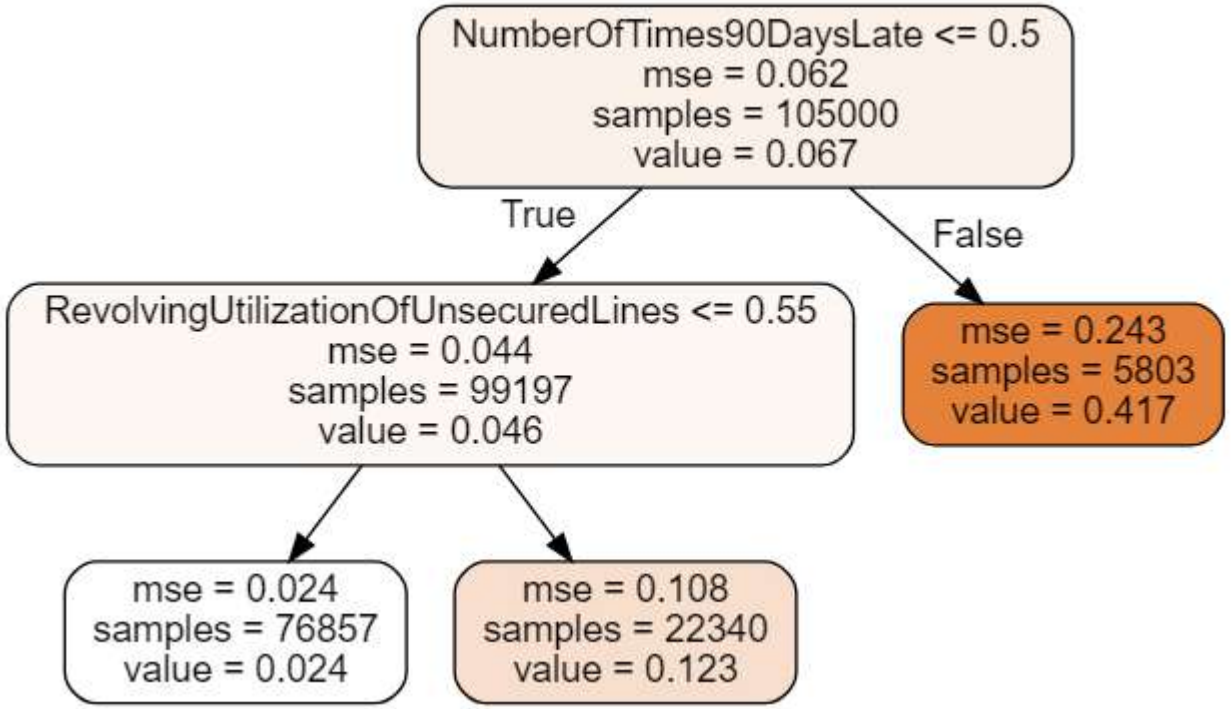
class `sklearn.tree.` **`DecisionTreeRegressor`**(*criterion='mse', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, presort='deprecated', ccp_alpha=0.0*) [source]

A decision tree regressor.

Read more in the User Guide.

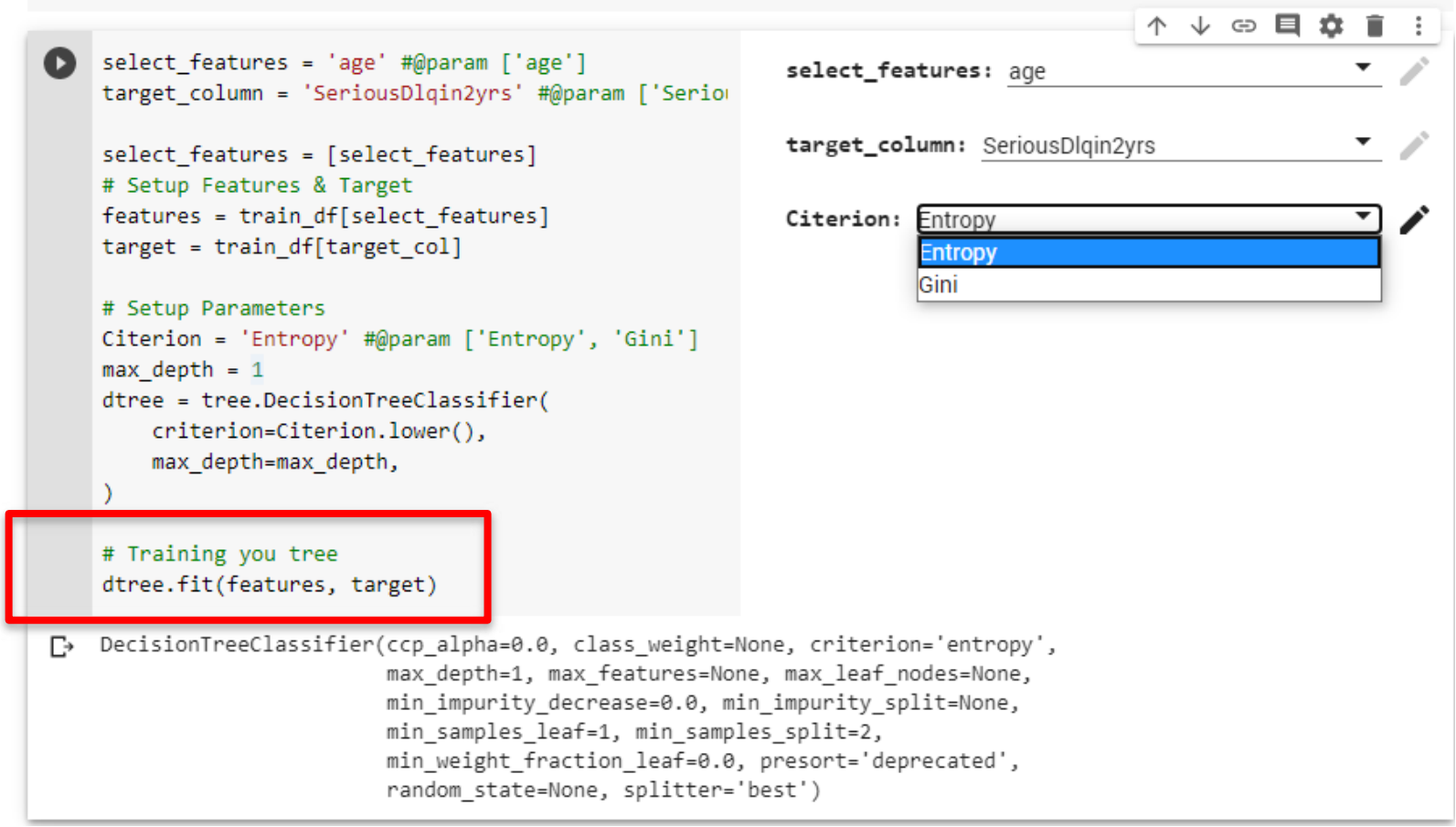
Parameters: **criterion : *{"mse", "friedman_mse", "mae"},***

MSE: Mean Square Error
MAE: Mean Absolute Error



Build your Decision Tree (cont.)

- Code Section 2.4



```
select_features = 'age' #@param ['age']
target_column = 'SeriousDlqin2yrs' #@param ['Serio

select_features = [select_features]
# Setup Features & Target
features = train_df[select_features]
target = train_df[target_col]

# Setup Parameters
Citerion = 'Entropy' #@param ['Entropy', 'Gini']
max_depth = 1
dtree = tree.DecisionTreeClassifier(
    criterion=Citerion.lower(),
    max_depth=max_depth,
)

# Training you tree
dtree.fit(features, target)
```

select_features: age

target_column: SeriousDlqin2yrs

Citerion: Entropy

Entropy

Gini

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=1, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Model Visualization (read the model)

Basic Description

1. get_params : Get all model parameters
2. get_depth : Get depth of trained tree
3. get_n_leaves : Get number of nodes
4. tree_.impurity : Get Gini/Entropy values

1
```
Parameters: {
 "ccp_alpha": 0.0,
 "class_weight": null,
 "criterion": "entropy",
 "max_depth": 1,
 "max_features": null,
 "max_leaf_nodes": null,
 "min_impurity_decrease": 0.0,
 "min_impurity_split": null,
 "min_samples_leaf": 1,
 "min_samples_split": 2,
 "min_weight_fraction_leaf": 0.0,
 "presort": "deprecated",
 "random_state": null,
 "splitter": "best"
}
```
2
```
Tree Depth: 1
```
3
```
Number of Node: 2
```
4
```
Entropy of Nodes: [0.35401168 0.42690946 0.2175608 ]
```

Model Visualization (cont.)

sklearn.tree.plot_tree

sklearn.tree.plot_tree(*decision_tree*, *, *max_depth=None*, *feature_names=None*, *class_names=None*, *label='all'*, *filled=False*, *impurity=True*, *node_ids=False*, *proportion=False*, *rounded=False*, *precision=3*, *ax=None*, *fontsize=None*) [source]

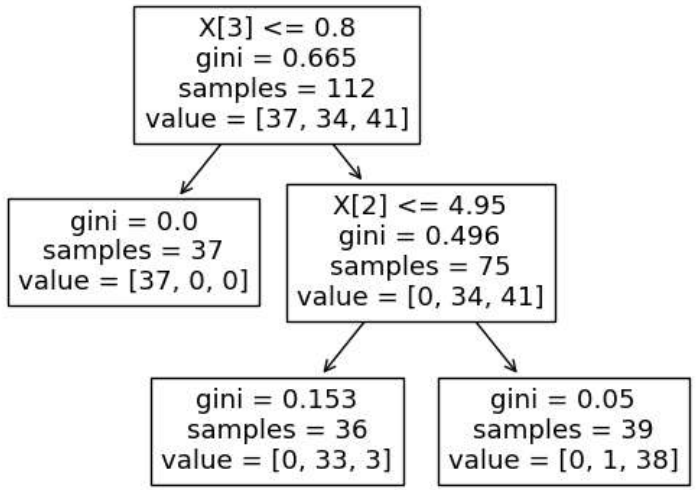
Plot a decision tree.

The sample counts that are shown are weighted with any sample_weights that mig

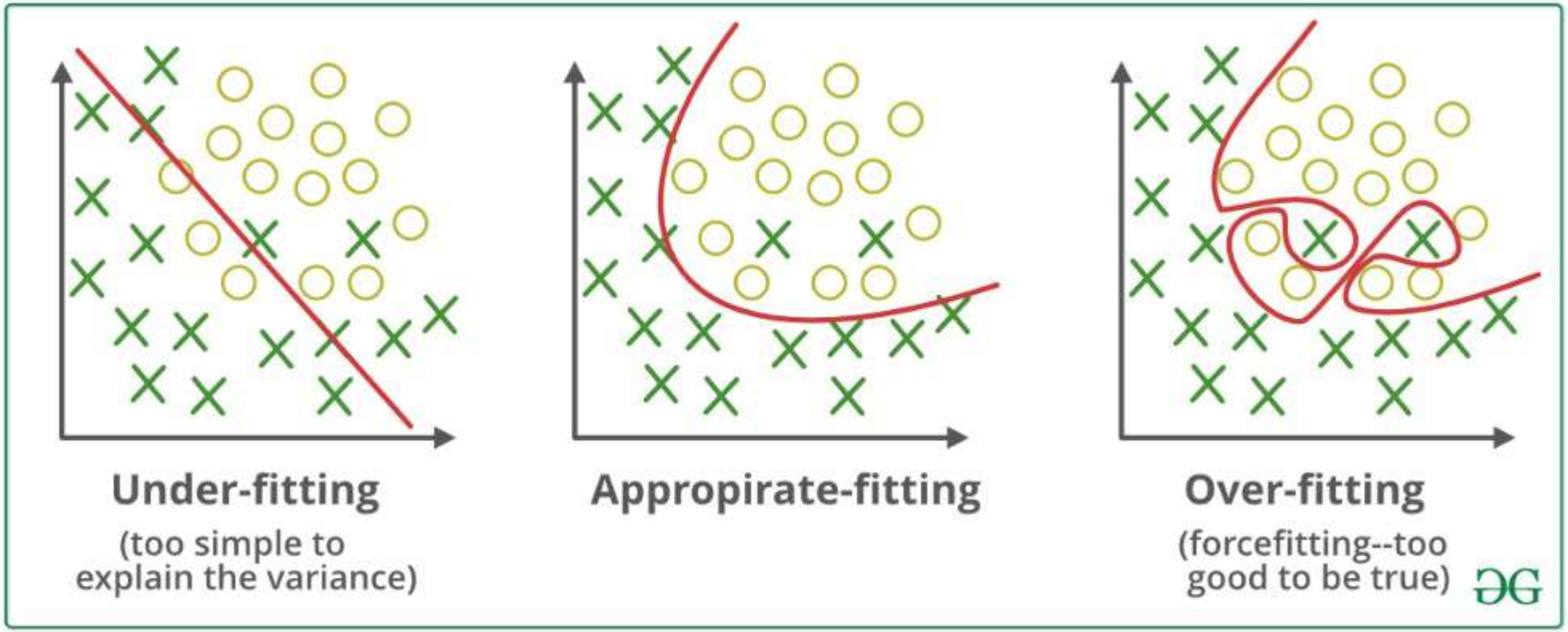
The visualization is fit automatically to the size of the axis. Use the `figsize` or `dp` size of the rendering.

Read more in the User Guide.

New in version 0.21.



Overfitting



Source: https://towardsdatascience.com/underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6fe4a8a49dbf

Pruning Tree

https://scikit-learn.org/stable/modules/tree.html#minimal-cost-complexity-pruning/

ccp_alpha : *non-negative float, default=0.0*

Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than `ccp_alpha` will be chosen. By default, no pruning is performed. See Minimal Cost-Complexity Pruning for details.

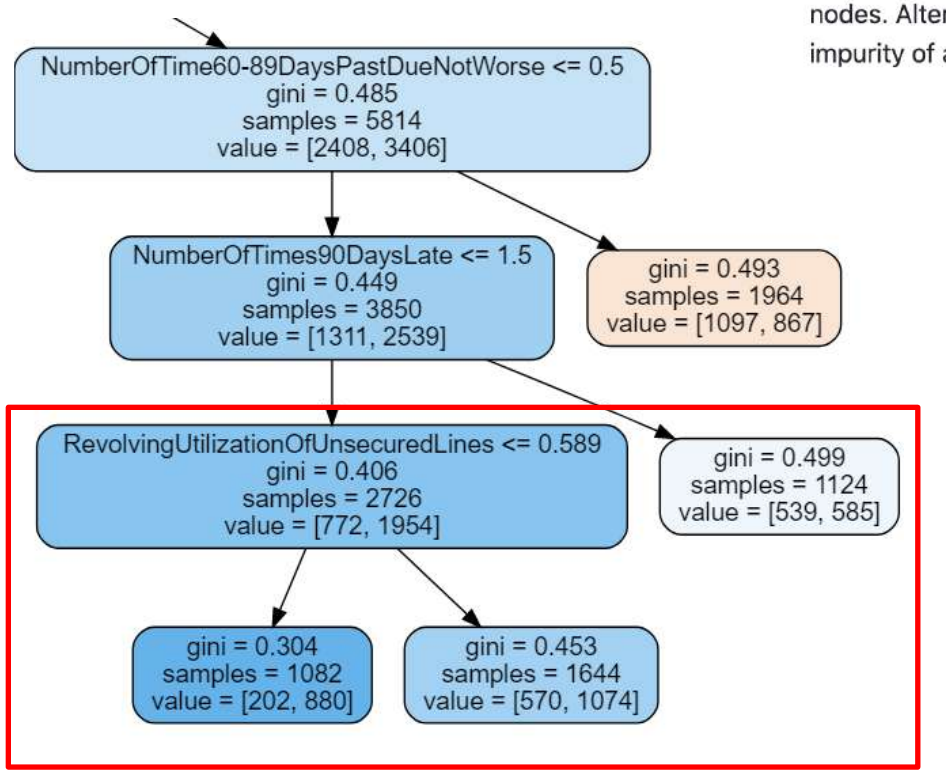
New in version 0.22.

$$R_\alpha(T) = R(T) + \alpha|T|$$

where $|T|$ is the number of terminal nodes in T and $R(T)$ is traditionally defined as the total misclassification rate of the terminal nodes. Alternatively, scikit-learn uses the total sample weighted impurity of the terminal nodes for $R(T)$. As shown above, the impurity of a node depends on the criterion. Minimal cost-complexity pruning finds the subtree of T that minimizes $R_\alpha(T)$.

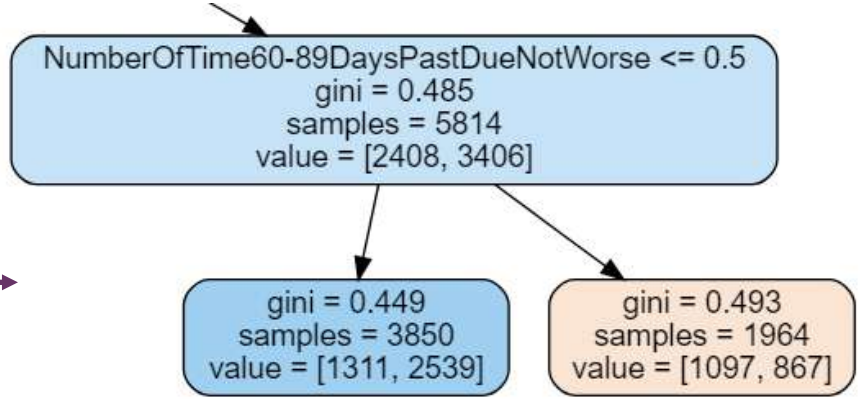
More

Impurity



Less

Prune



When impurity is too low, it cause more overfitting

Test pruning parameter by .cost_complexity_pruning_path

```
dtree.cost_complexity_pruning_path(features, target)
```

```
{'ccp_alphas': array([0.00000000e+00, 8.91619910e-05, 1.17039010e-04, 1.29787452e-04,
       1.94525732e-04, 3.18283592e-04, 4.15157131e-04, 5.84363721e-04,
       6.81710606e-04, 1.05951846e-03, 1.17765555e-03, 1.83295138e-03,
       3.24166605e-03, 1.41432990e-02]),
 'impurities': array([0.10075641, 0.10084557, 0.10096261, 0.1010924 , 0.10128692,
       0.10160521, 0.10202036, 0.10260473, 0.10328644, 0.10434596,
       0.10552361, 0.10735656, 0.11059823, 0.12474153])}
```

ccp_alphas: Values which affect pruning (use lower bound value)
Impurities: Impurity after pruning (default=0; no pruning)

Important Parameters in Decision Tree

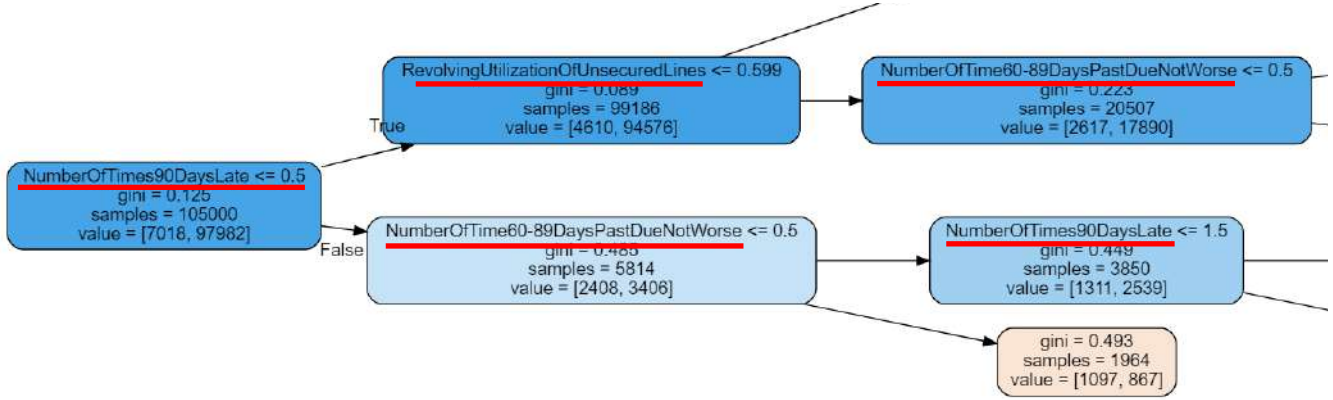
- Splitting measure (criterion) : gini / entropy
- Maximum depth : ~5-10 (depend on number of feature)
- Maximum leaf nodes : depend on number of class (target) and feature
- Minimum sample split : 5 – 20% (depend on number of data)
- Minimum impurity decrease : (default 0)
- Pruning adjustment (ccp_alpha) : 0.001 - 0.01 (depend on size of tree)

Which features are important ?

- Check how important by . feature_importances_
- The importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature. It is also known as the gini importance.

```
dtree.feature_importances_
```

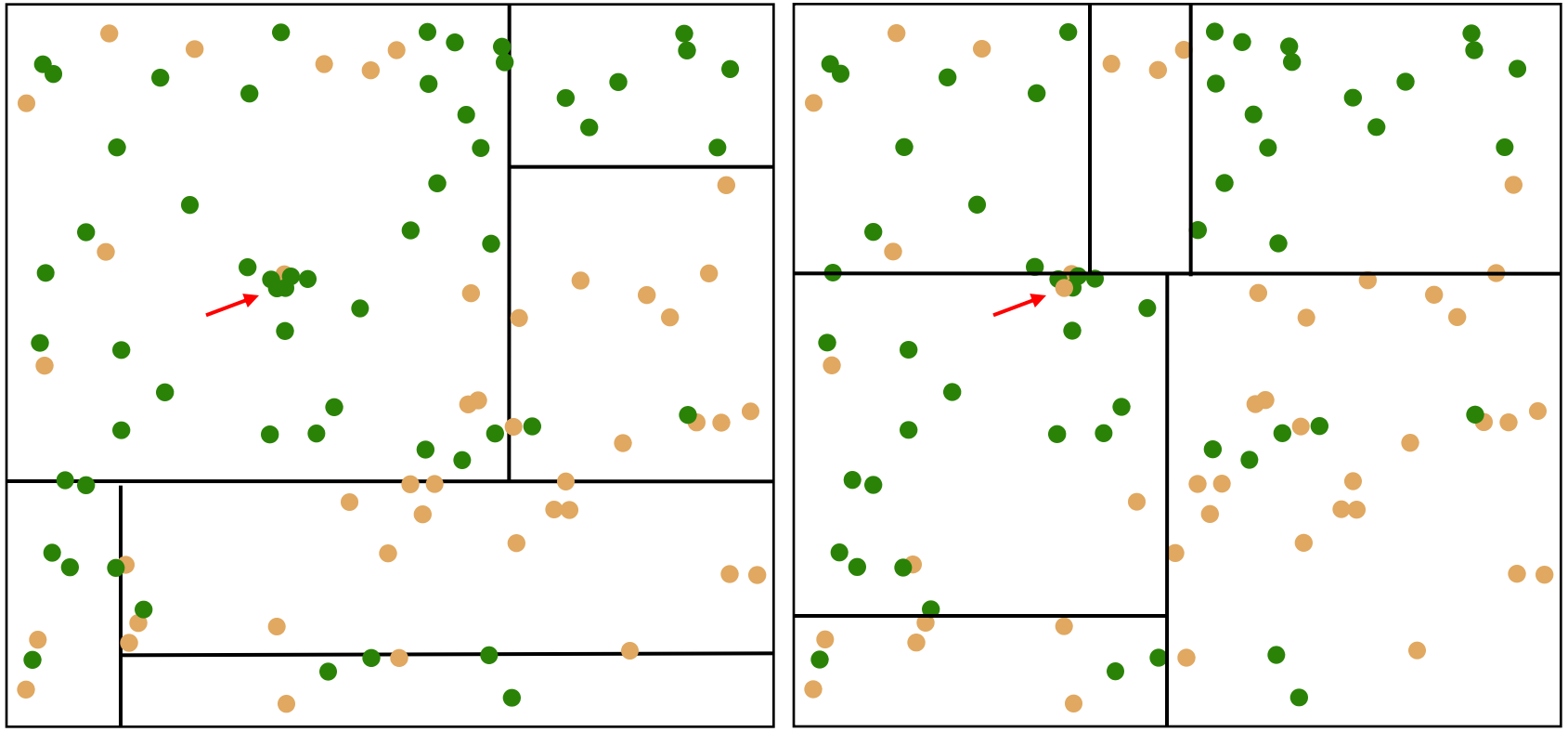
```
array([0.15330291, 0.00371739, 0.07500437, 0.        , 0.        ,
       0.        , 0.61403332, 0.        , 0.15394201, 0.        ])
```




```
======= Features Important =======
0.614 : Numberoftimes90dayslate
0.154 : Numberoftime60-89dayspastduenotworse
0.153 : Revolvingutilizationofunsecuredlines
0.075 : Numberoftime30-59dayspastduenotworse
0.004 : Age
0.000 : Debtratio
0.000 : Monthlyincome
0.000 : Numberofopencreditlinesandloans
0.000 : Numberrealestateloansorlines
0.000 : Numberofdependents
```

Instability

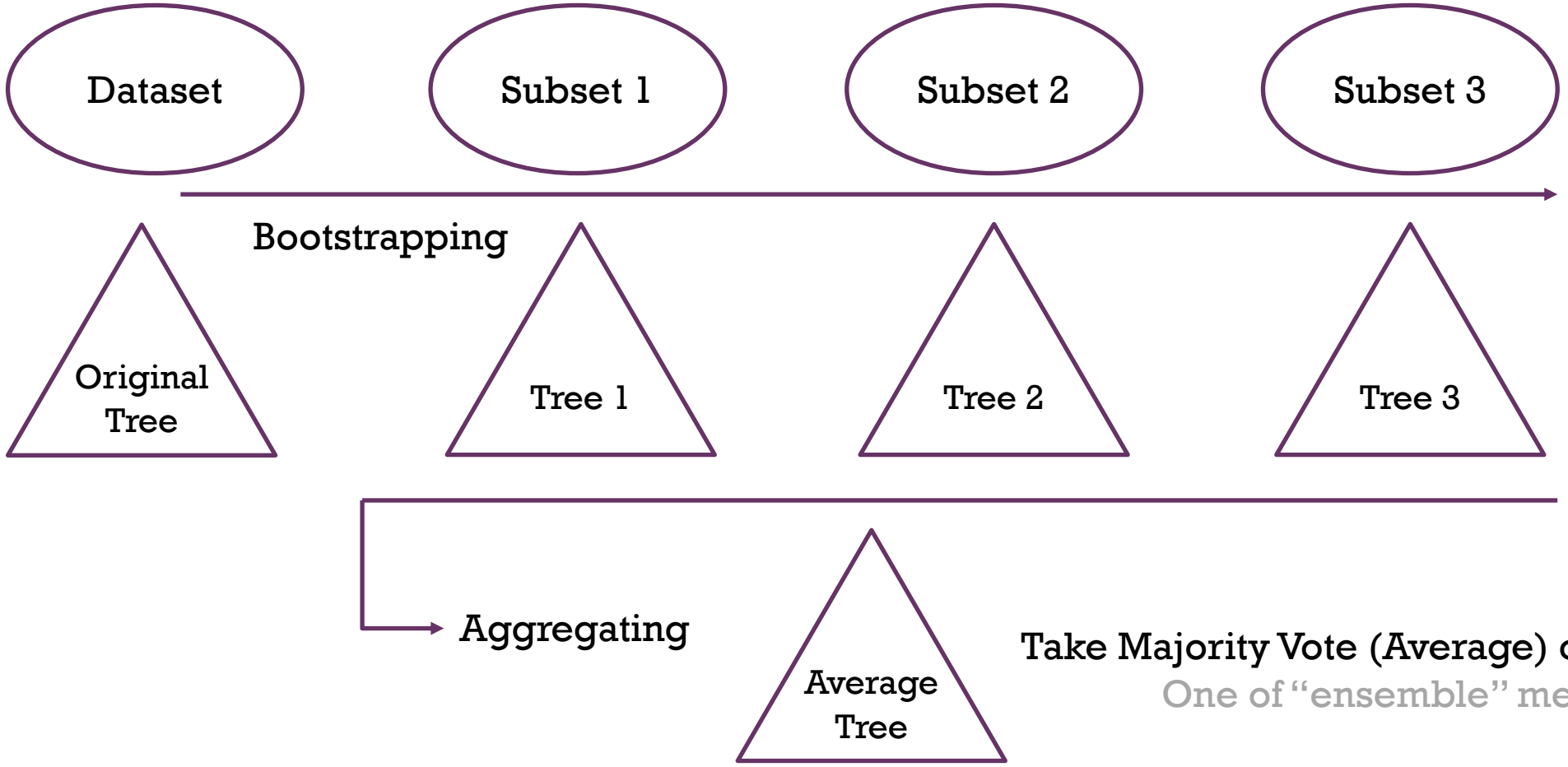
One reversal



Accuracy = 81%

Accuracy = 80%

Bagging (Bootstrap Aggregation)



Take Majority Vote (Average) of Prediction

One of "ensemble" method

Bagging Method in sklearn

sklearn.ensemble.BaggingClassifier

class sklearn.ensemble.**BaggingClassifier**(*base_estimator=None, n_estimators=10, *, max_samples=1.0, max_features=1.0, bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=None, verbose=0*)

[source]

sklearn.ensemble.BaggingRegressor¶

class sklearn.ensemble.**BaggingRegressor**(*base_estimator=None, n_estimators=10, *, max_samples=1.0, max_features=1.0, bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=None, verbose=0*)

[source]

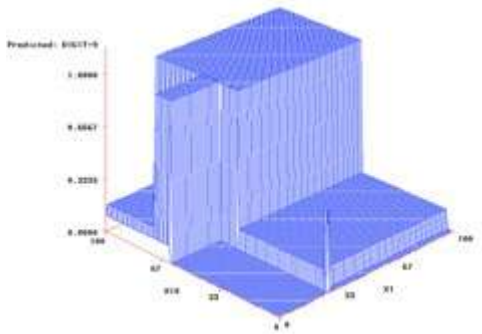
Example

```
>>> from sklearn.ensemble import BaggingClassifier
>>> from sklearn.tree import DecisionTreeClassifier
>>> bagging = BaggingClassifier(DecisionTreeClassifier(),
...                             max_samples=0.5, max_features=0.5)
```

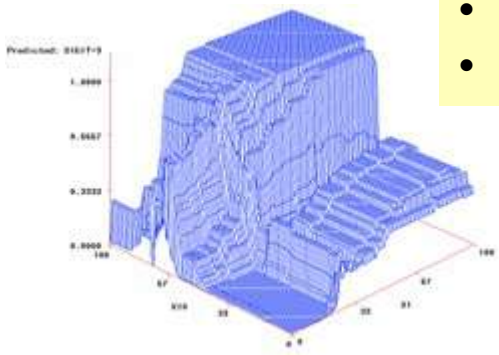
Random Forest Algorithm (RF)

- Forest algorithm samples the rows and the columns at each step.
- Forest takes bootstrap samples of the rows in training data (sampling with replacement)
- At each step, a set of variables (columns) is sampled.
- This increases variation among trees in the **ensemble** often leads to improved prediction accuracy.

Important Params:
- #Tree
- #Columns (variables)
- #Rows (examples)

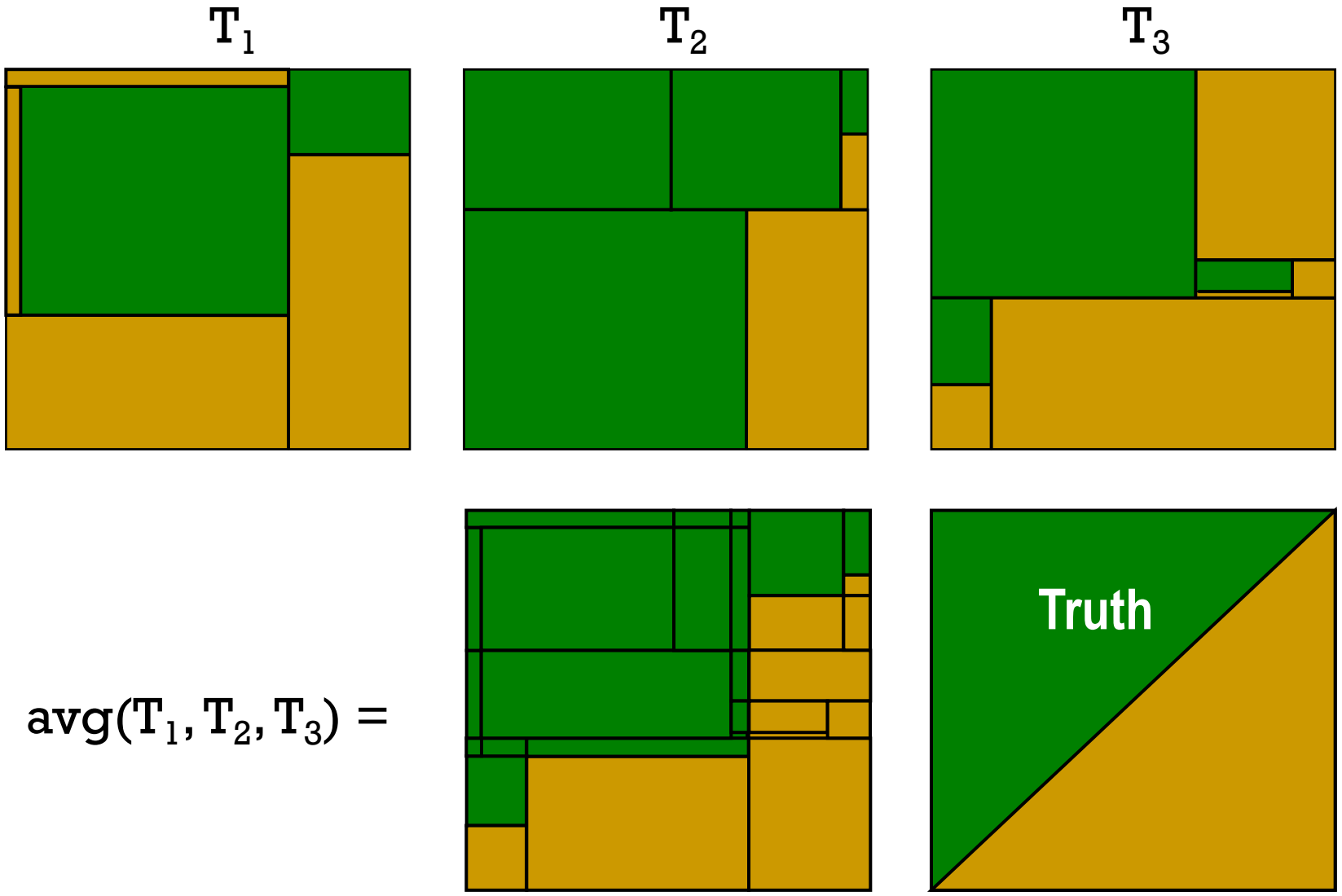


Single Tree



Random Forest

Combine



...

3.2.4.3.1. sklearn.ensemble.RandomForestClassifier

class `sklearn.ensemble.` **`RandomForestClassifier`**(*n_estimators=100*, *criterion='gini'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features='auto'*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *min_impurity_split=None*, *bootstrap=True*, *oob_score=False*, *n_jobs=None*, *random_state=None*, *verbose=0*, *warm_start=False*, *class_weight=None*, *ccp_alpha=0.0*, *max_samples=None*) [source]

A random forest classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if `bootstrap=True` (default).

Read more in the User Guide.

Parameters:

n_estimators : *integer, optional (default=100)*

The number of trees in the forest.

Changed in version 0.22: The default value of `n_estimators` changed from 10 to 100 in 0.22.

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. Note: this parameter is tree-specific.

3.2.4.3.2. sklearn.ensemble.RandomForestRegressor

class `sklearn.ensemble.` **`RandomForestRegressor`**(*n_estimators=100, criterion='mse', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, ccp_alpha=0.0, max_samples=None*) ¶ [source]

A random forest regressor.

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if `bootstrap=True` (default).

Read more in the User Guide.

Parameters:

n_estimators : *integer, optional (default=10)*

The number of trees in the forest.

Changed in version 0.22: The default value of `n_estimators` changed from 10 to 100 in 0.22.

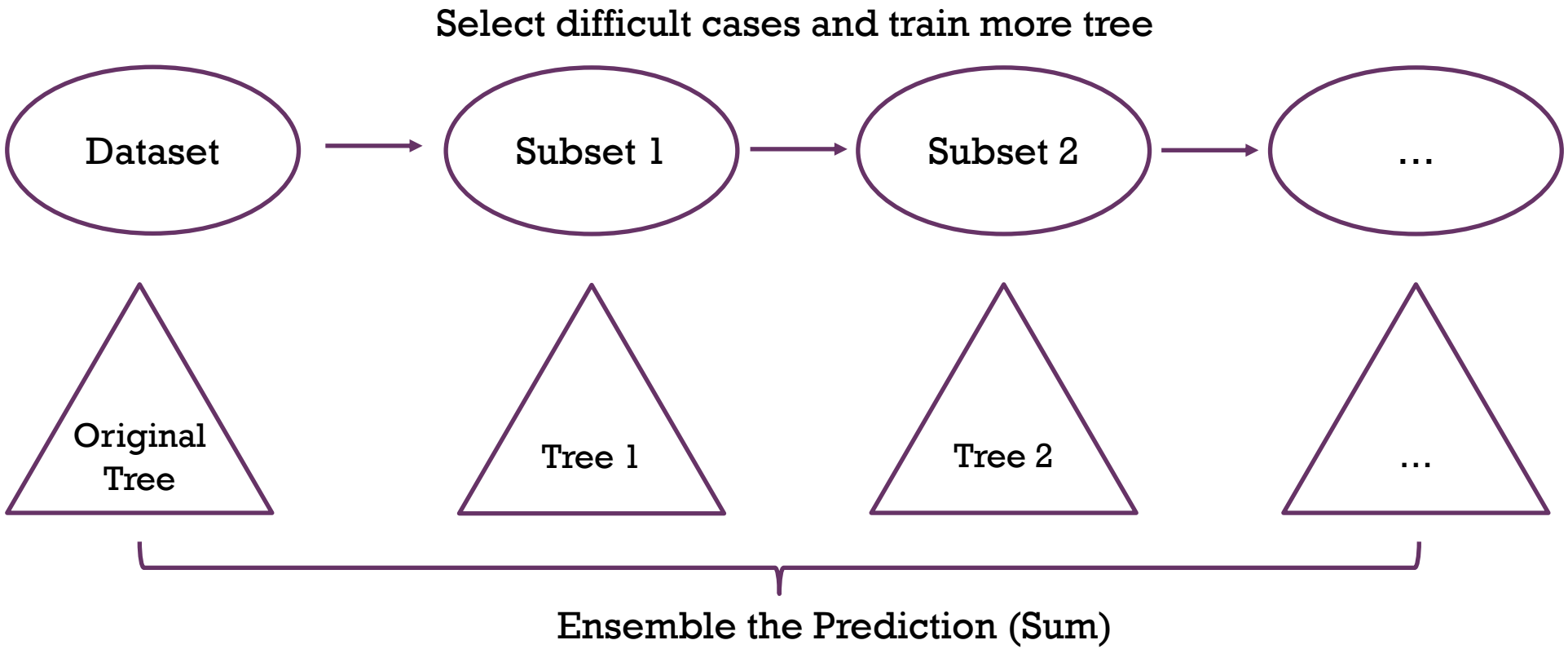
criterion : *string, optional (default="mse")*

The function to measure the quality of a split. Supported criteria are "mse" for the mean squared error, which is equal to variance reduction as feature selection criterion, and "mae" for the mean absolute error.

New in version 0.18: Mean Absolute Error (MAE) criterion.

Boosting

- Boosting is a method of converting weak learners into strong learners.
- In boosting, each new tree is a fit on a modified version of the original data set.



Boosting (cont.)

	k=1		k=2		k=3		k=4 …
case	freq	m	freq	m	freq	m	freq
1	1	1	1.5	1	.5	2	.97
2	1	0	.75	0	.25	0	.06
3	1	1	1.5	2	4.25	3	4.69
4	1	0	.75	1	.5	1	.11
5	1	0	.75	0	.25	0	.06
6	1	0	.75	0	.25	1	.11

*Shown is Arc-x4, one method of boosting

Boosting in sklearn

- AdaBoost, short for "Adaptive Boosting"

sklearn.ensemble.AdaBoostClassifier¶

class sklearn.ensemble.**AdaBoostClassifier**(*base_estimator=None, *, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None*) [source]

sklearn.ensemble.AdaBoostRegressor

class sklearn.ensemble.**AdaBoostRegressor**(*base_estimator=None, *, n_estimators=50, learning_rate=1.0, loss='linear', random_state=None*) [source]

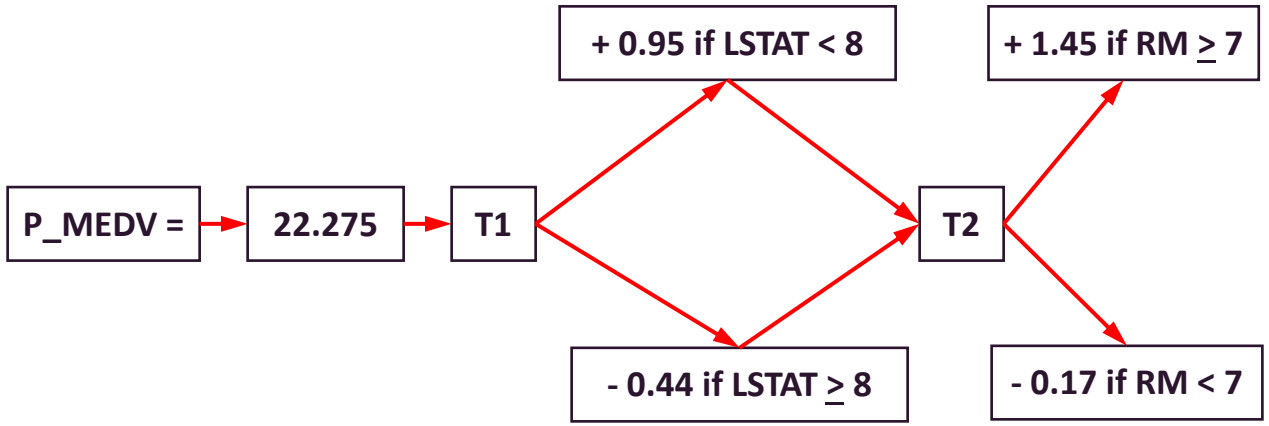
Example

```
>>> from sklearn.ensemble import AdaBoostClassifier
>>> from sklearn.tree import DecisionTreeClassifier
>>> boosting = AdaBoostClassifier(DecisionTreeClassifier(),
...                               max_samples=0.5, max_features=0.5)
```

https://towardsdatascience.com/boosting-algorithm-adaboost-b6737a9ee60c#:~:text=AdaBoost,classification%20can%20be%20represented%20as

Gradient Boosting Tree

- Boosting the performance by using ensemble trees
- Each tree in the next step aims to predict *error*.
- So, the prediction result is the summation of all trees (not average any more).
- In the final step, weights are assigned to all leaf nodes using gradient descent in order to finally reduce errors.

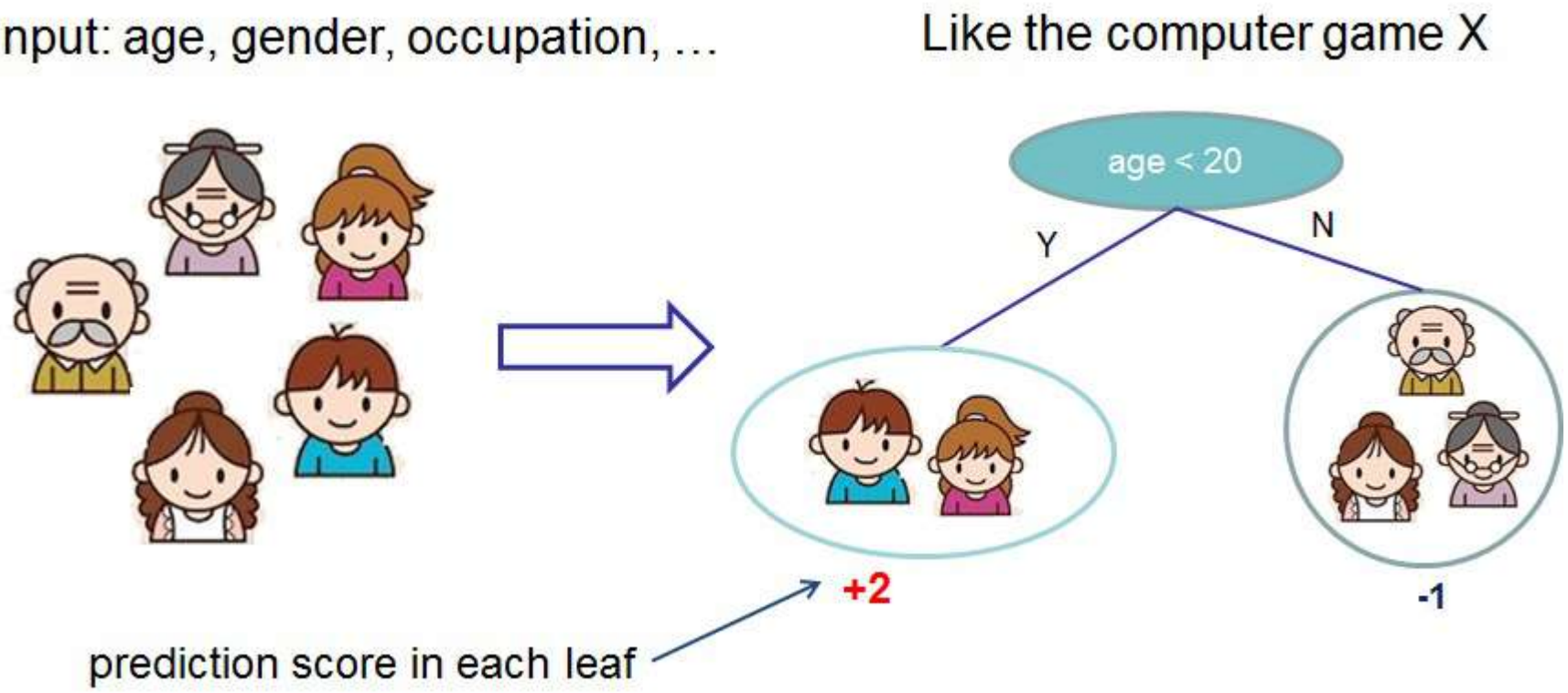


Boston House Price Prediction

- 'RM' is the average number of rooms among homes in the neighborhood.
- 'LSTAT' is the percentage of homeowners in the neighborhood considered "lower class" (working poor).

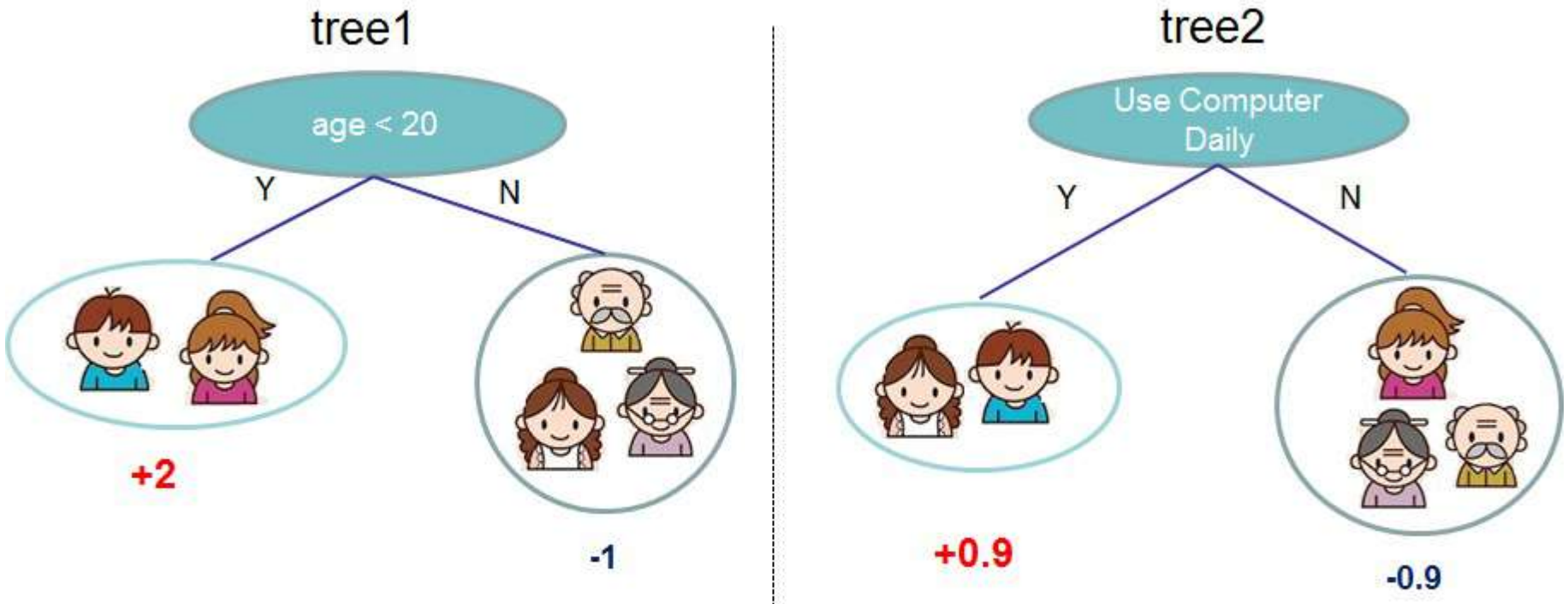
Intro to Gradient Boosting Tree

- Decision Tree



Intro to Gradient Boosting Tree (cont.)

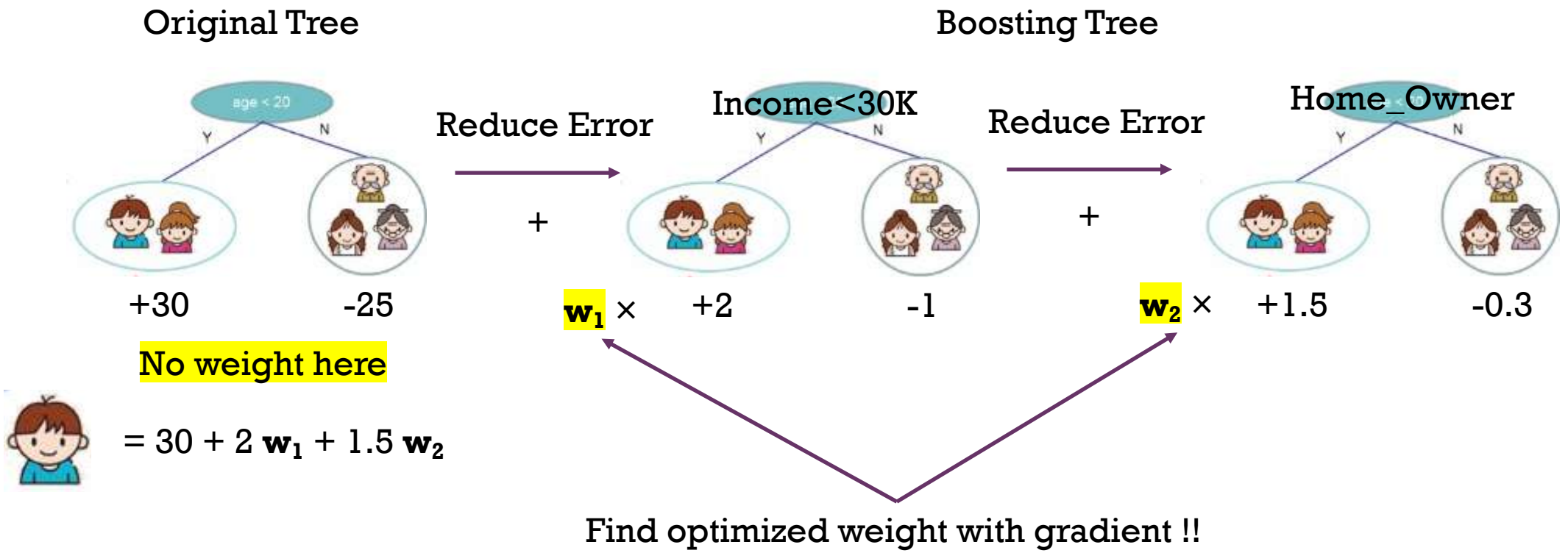
- Ensemble by weight on leaf nodes



f( ) = 2 + 0.9= 2.9 f( )= -1 - 0.9= -1.9

Intro to Gradient Boosting Tree (cont.)

- Optimize weight of leaf nodes by gradient descent



Gradient Boosting Tree in sklearn

3.2.4.3.5. sklearn.ensemble.GradientBoostingClassifier

class sklearn.ensemble.**GradientBoostingClassifier**(**, loss='deviance', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0, min_impurity_split=None, init=None, random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, presort='deprecated', validation_fraction=0.1, n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0*) [source]

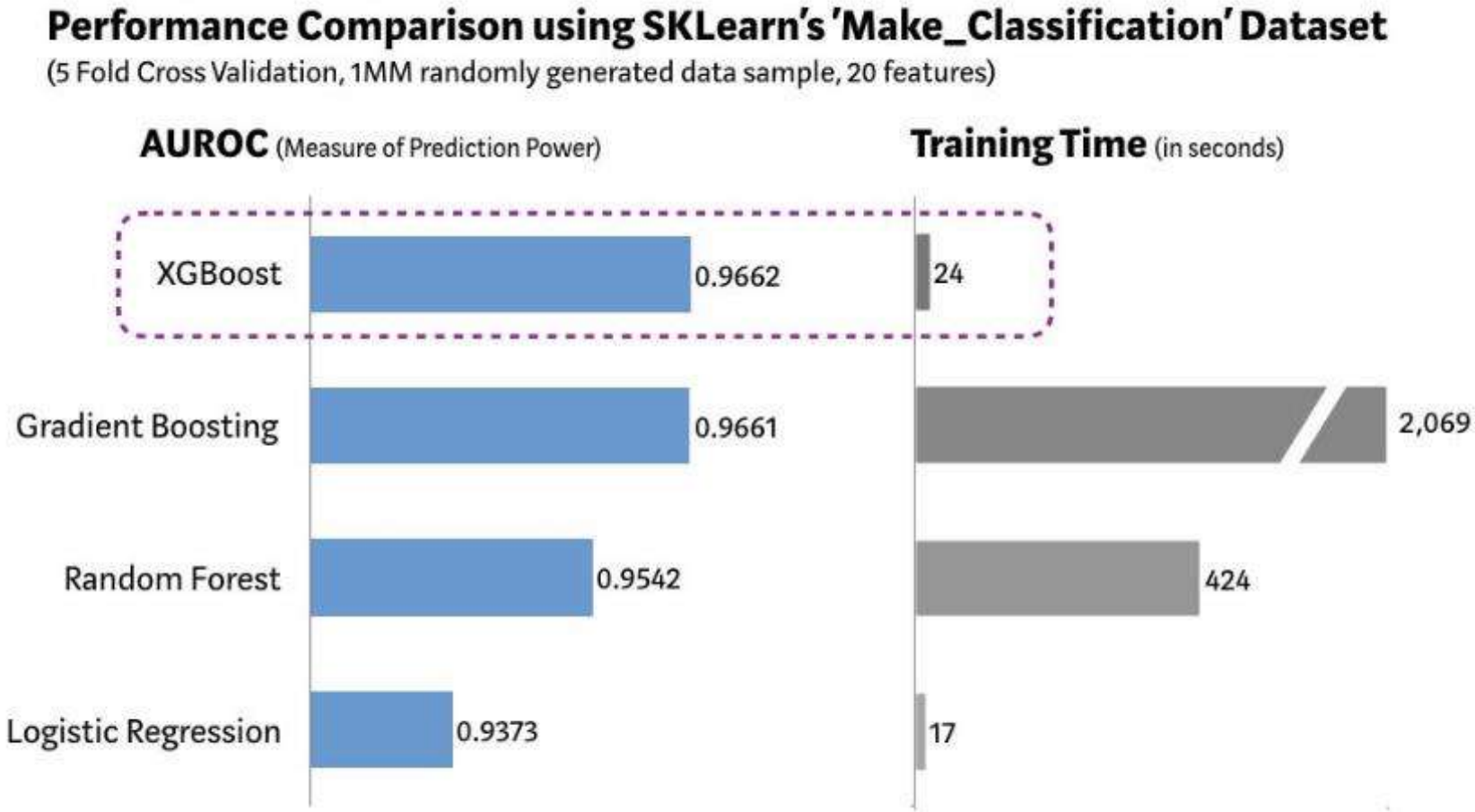
eXtreme Gradient Boosting Tree (XGBoost)

- XGBoost stands for "Extreme Gradient Boosting", where the term "Gradient Boosting" originates from the paper *Greedy Function Approximation: A Gradient Boosting Machine*, by Friedman.

- XGBoost is exactly a tool motivated by the formal principle introduced in this tutorial! More importantly, it is developed with both deep consideration in terms of **systems optimization** and **principles in machine learning**. The goal of this library is to push the extreme of the computation limits of machines to provide a **scalable**, **portable** and **accurate** library

dmlc XGBoost

Models Performance



XGBoost Package

- Multiple Language Support

- Python package
- R package
- JVM package
- Ruby package
- Swift package
- Julia package
- C Package

- Distributed Model on Cloud
  - AWS
  - Kubernetes
  - Spark
  - Dask
- Compatible with sklearn interface

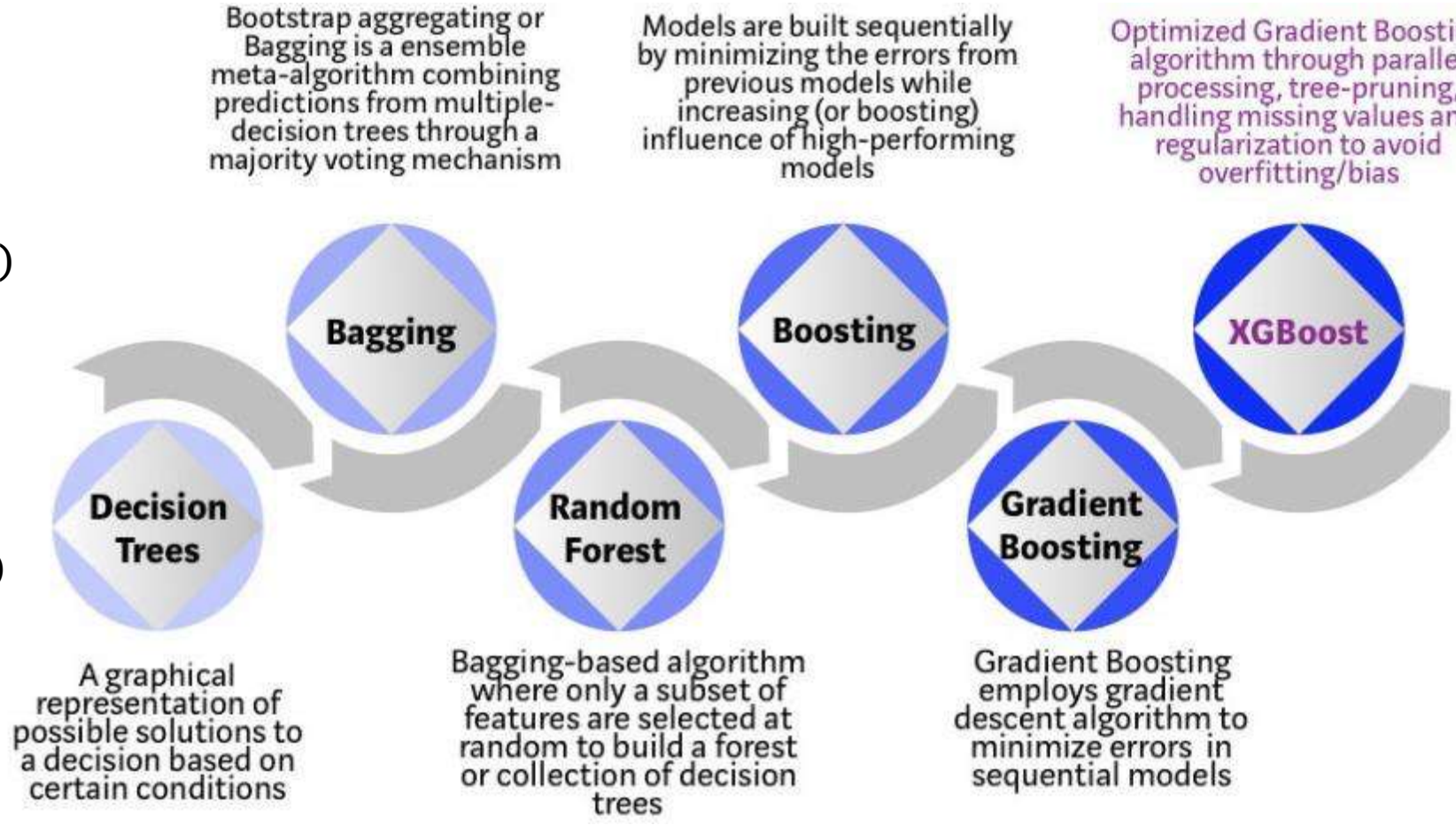
XGBoost Important Parameter

- n_jobs : Number of parallel tasks (-1 = all cores)
- objective : Type of Prediction
  - for Classification `binary:hinge`, `binary:logistic` (prob) or `multi:softmax` (multi-class)
  - for Regression `reg:squarederror`
- learning_rate : Step size used in update weights
- n_estimators : Number of trees
- subsample (0, 1]: Number of training data to growing trees, will prevent overfitting
- importance_type : Get feature importance of each feature.
  - 'weight': the number of times a feature is used to split the data across all trees.
  - 'gain': the average gain across all splits the feature is used in.
  - 'cover': the average coverage across all splits the feature is used in.
  - 'total_gain': the total gain across all splits the feature is used in.
  - 'total_cover': the total coverage across all splits the feature is used in.

Evolution of Decision Trees

Performance (Accuracy, Time)

Method (Based method)



Source: https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

Prev Up Next

scikit-learn 1.2.1
Other versions

Please cite us if you use the software.

sklearn.feature_selection.SelectFromModel ¶

class `sklearn.feature_selection.`**`SelectFromModel`**(*estimator*, ***, *threshold=None*, *prefit=False*, *norm_order=1*, *max_features=None*, *importance_getter='auto'*) [source]

Meta-transformer for selecting features based on importance weights.

New in version 0.17.

Read more in the User Guide.

Parameters:

estimator : *object*

The base estimator from which the transformer is built. This can be both a fitted (if `prefit` is set to True) or a non-fitted estimator. The estimator should have a `feature_importances_` or `coef_` attribute after fitting. Otherwise, the `importance_getter` parameter should be used.

threshold : *str or float, default=None*

The threshold value to use for feature selection. Features whose absolute importance value is greater or equal are kept while the others are discarded. If “median” (resp. “mean”), then the `threshold` value is the median (resp. the mean) of the feature importances. A scaling factor (e.g., “1.25*mean”) may also be used. If None and if the estimator has a parameter penalty set to l1, either explicitly or implicitly (e.g, Lasso), the threshold used is 1e-5. Otherwise, “mean” is used by default.

prefit : *bool, default=False*

Whether a prefit model is expected to be passed into the constructor directly or not. If `True`, `estimator`

https://www.datatechnotes.com/2021/04/selectfrommodel-feature-selection.html

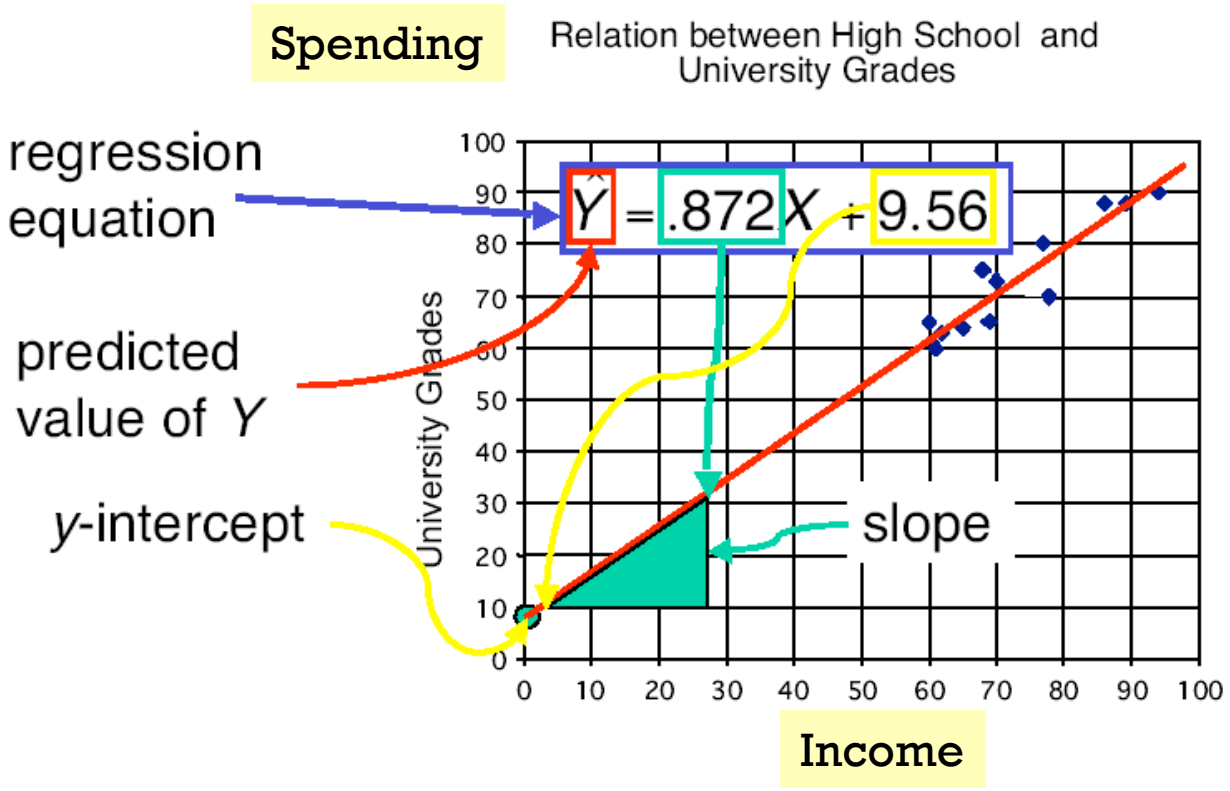

```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import AdaBoostRegressor
from sklearn.datasets import load_boston
from numpy import array

boston = load_boston()
x = boston.data
y = boston.target

print("Feature data dimension: ", x.shape)

estimator = AdaBoostRegressor(random_state=0, n_estimators=50
selector = SelectFromModel(estimator)
selector = selector.fit(x, y)

status = selector.get_support()
print("Selection status: ", status)

features = array(boston.feature_names)
print("All features:")
print(features)

print("Selected features:")
print(features[status])
selector.transform(x)
```

```
status = selector.get_support()
print("Selection status: ", status)

Selection status:  [False False False False False  True False  True False False False False
  True]

features = array(boston.feature_names)
print("All features:")
print(features)

print("Selected features:")
print(features[filter])
selector.transform(x)


All features:
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']

Selected features:
['RM' 'DIS' 'LSTAT']
array([[6.575 , 4.09  , 4.98  ],
       [6.421 , 4.9671, 9.14  ],
       [7.185 , 4.9671, 4.03  ],
       ...,
       [6.976 , 2.1675, 5.64  ],
       [6.794 , 2.3889, 6.48  ],
       [6.03  , 2.505 , 7.88  ]])
```

+

2) Regression

2) Regression



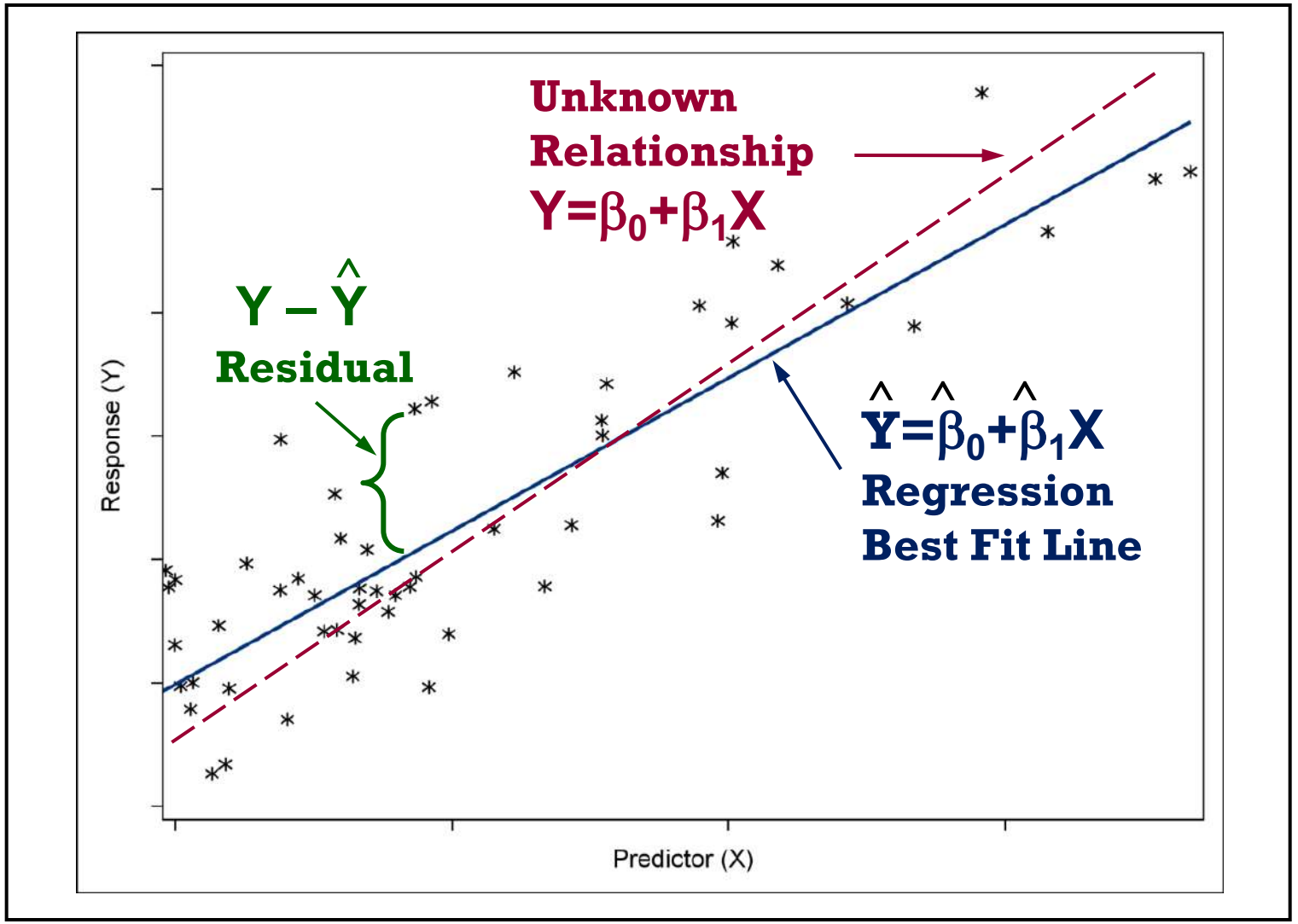
weight, coefficient

$$\underset{\text{target}}{\hat{y}} = \underset{\text{intercept}}{\hat{w}_0} + \hat{w}_1 x_1 + \hat{w}_2 \underset{\text{input}}{x_2}$$

- The least square method aims to minimize the following term

$$\sum_{\substack{training \\ data}} (y_i - \hat{y}_i)^2$$

Ordinary Least Squares (OLS) Regression



Multiple Linear Regression Model Matrix Multiplication Approach

inputs		target
Age	**Income**	**Spending**
25	25,000	400
35	50,000	500
32	35,000	550

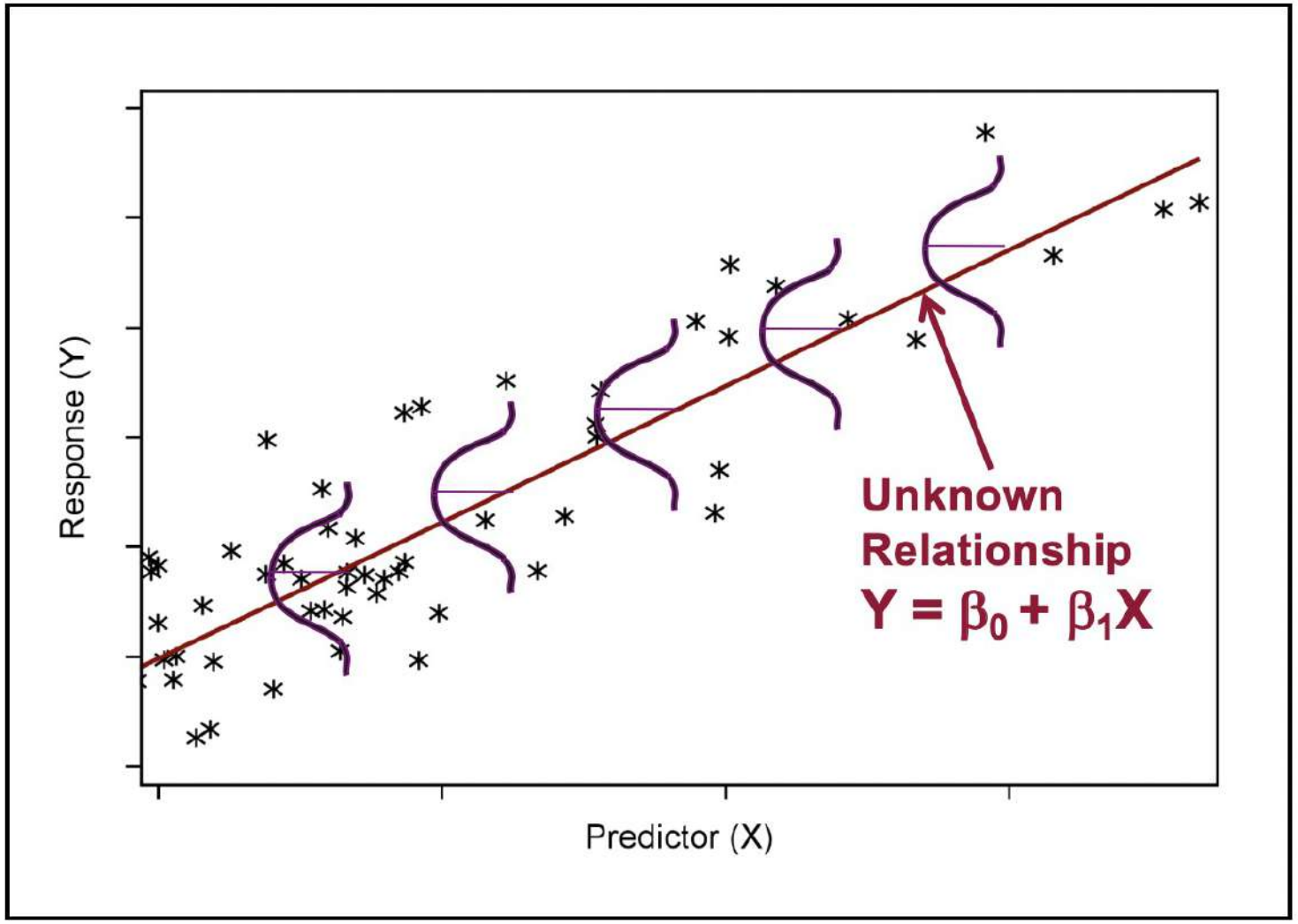
$$Y=\beta_0(1)+\beta_1X_1+\beta_2X_2+\varepsilon$$

$$[Y] = [X][\beta]$$
$$[\beta]=[X]^{-1}[Y]$$

https://sites.google.com/site/mystatistics01/regression-correlation-analysis/multiple-linear-regression-step

Linear Regression Assumption

Spending = 500 + 10*Income10K + 2*Age



- Linear relationship between (y, x_i). [Pearson correlation]
- Error is normal distributed. [remove outliers, log-transformation]
- Error has equal variance (homoscedasticity) [remove outliers, log-transformation]
- Errors are independent from each other. [design new data correction]

sklearn.linear_model.LinearRegression

class sklearn.linear_model.**LinearRegression**(*fit_intercept=True*, *normalize=False*, *copy_X=True*, *n_jobs=None*) [source]

Ordinary least squares Linear Regression.

LinearRegression fits a linear model with coefficients w = (w1, ..., wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

Parameters:

fit_intercept : *bool, optional, default True*
Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).

normalize : *bool, optional, default False*
This parameter is ignored when `fit_inte`
regression by subtracting the mean and di
sklearn.preprocessing.StandardScal

normalize : *bool, default=False*
This parameter is ignored when `fit_intercept` is set to False. If True, the regressors X will be normali
before regression by subtracting the mean and dividing by the l2-norm. If you wish to standardize, plea
use `StandardScaler` before calling `fit` on an estimator with `normalize=False`.

Deprecated since version 1.0: `normalize` was deprecated in version 1.0 and will be removed in 1.2.

Attributes::

coef_ : *array of shape (n_features,) or (n_targets, n_features)*
Estimated coefficients for the linear regression problem. If multiple targets are passed during the fit (y 2D),
only one target is passed, this is a 1D array of

intercept_ : *float or array of shape (n_targets,)*
Independent term in the linear model. Set to 0.0 if `fit_intercept = False`.

n_features_in_ : *int*
Number of features seen during fit.

New in version 0.24.

feature_names_in_ : *ndarray of shape (`n_features_in_`,)*
Names of features seen during fit. Defined only when `X` has feature names that are all strings.

New in version 1.0.

for n_targets > 1 and sufficient

one means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors.

more details.

Regression with Regularization

- Lasso Regression (L1 Regularization)
- Ridge Regression (L2 Regularization)

Lasso Regression (Least Absolute Shrinkage and Selection Operator) adds "*absolute value of magnitude*" of coefficient as penalty term to the loss function.

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Cost function

Ridge regression adds "*squared magnitude*" of coefficient as penalty term to the loss function. Here the *highlighted* part represents L2 regularization element.

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Cost function

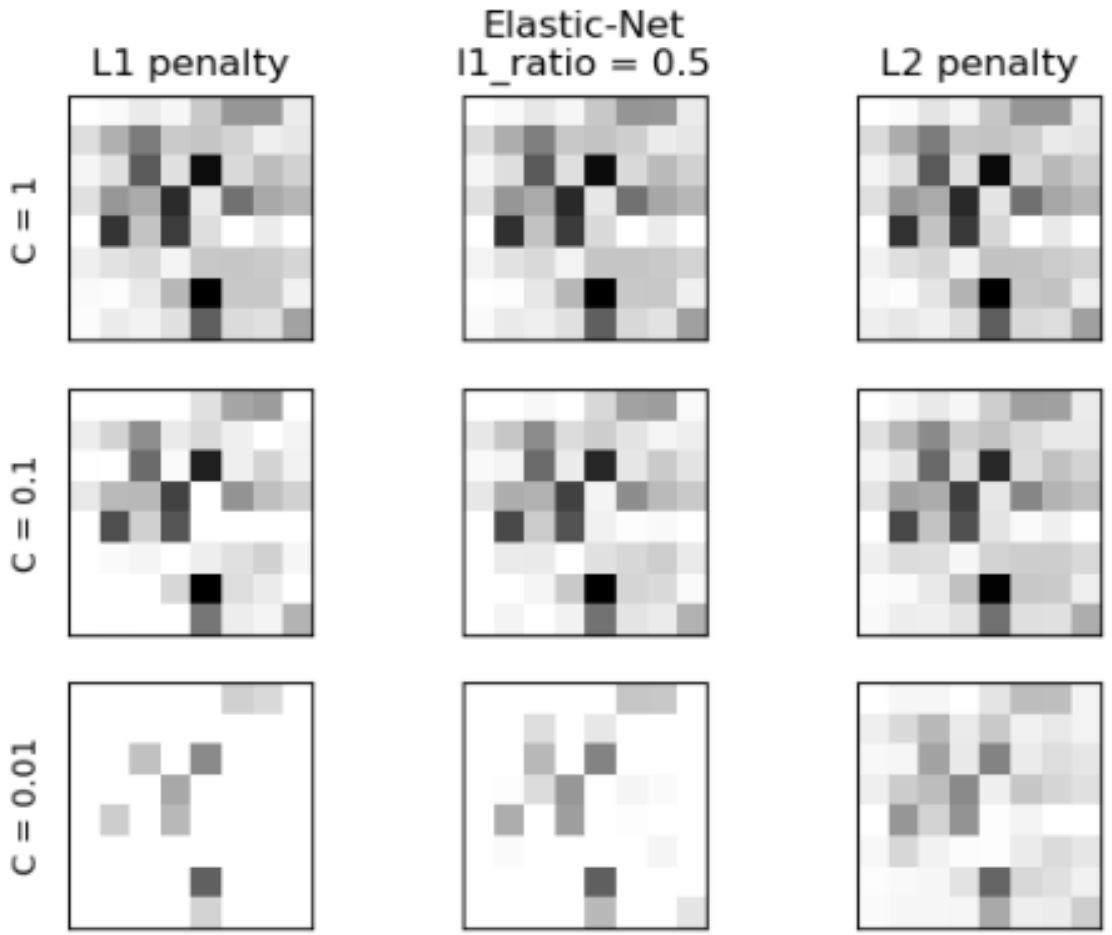
The function of both the regularization methods are almost the same. The **key difference** between these techniques is that Lasso shrinks the less important feature's coefficient to zero thus, removing some feature altogether. So, this works well for **feature selection** in case we have a huge number of features.

https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c

ElasticNet

l1_ratio : *float, default=0.5*

The ElasticNet mixing parameter, with `0 <= l1_ratio <= 1`. For `l1_ratio = 0` the penalty is an L2 penalty. `For l1_ratio = 1` it is an L1 penalty. For `0 < l1_ratio < 1`, the penalty is a combination of L1 and L2.



https://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_l1_l2_sparsity.html#:~:text=Comparison%20of%20the%20sparsity%20(percentage,C%20constrain%20the%20model%20more.

See also:

Ridge
Ridge regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of the coefficients with l2 regularization.

Lasso
The Lasso is a linear model that estimates sparse coefficients with l1 regularization.

ElasticNet
Elastic-Net is a linear regression model trained with both l1 and l2 -norm regularization of the coefficients.

sklearn.linear_model.Ridge

class `sklearn.linear_model.`**`Ridge`**(*alpha=1.0, *, fit_interce*
max_iter=None, tol=0.001, solver='auto', positive=False, random

Linear least squares with l2 regularization.

Minimizes the objective function:

```
||y - Xw||^2_2 + alpha * ||w||^2_2
```

This model solves a regression model where the loss function i
the l2-norm. Also known as Ridge Regression or Tikhonov regu
regression (i.e., when y is a 2d-array of shape (n_samples, n_t

Read more in the User Guide.

Parameters::

alpha : *{float, ndarray of shape (n_targets,*
Constant that multiplies the L2 term, contro
float i.e. in `[0, inf)`.

When `alpha = 0`, the objective is equivale
object. For numerical reasons, using `alpha`
use the `LinearRegression` object.

If an array is passed, penalties are assumec
number.

fit_intercept : *bool, default=True*
Whether to fit the intercept for this model. I
`y` are expected to be centered).

normalize : *bool, default=False*
This parameter is ignored when `fit_inter`
before regression by subtracting the mean

sklearn.linear_model.Lasso

class `sklearn.linear_model.`**`Lasso`**(*alpha=1.0, *, fit_intercept=True, normalize='deprecated', precon*
copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selecti

Linear Model trained with L1 prior as regularizer (aka the Lasso).

The optimization objective for Lasso is:

```
(1 / (2 * n_samples)) * ||y - Xw||^2_2 + alpha * ||w||_1
```

Technically the Lasso model is optimizing the same objective function as the Elastic Net with `l1_ratio=`

Read more in the User Guide.

Parameters::

alpha : *float, default=1.0*
Constant that multiplies the L1 term, controlling regularization strength. `alpha` must
float i.e. in `[0, inf)`.

When `alpha = 0`, the objective is equivalent to ordinary least squares, solved by the
object. For numerical reasons, using `alpha = 0` with the `Lasso` object is not advisec
use the `LinearRegression` object.

fit_intercept : *bool, default=True*
Whether to calculate the intercept for this model. If set to False, no intercept will be u
(i.e. data is expected to be centered).

normalize : *bool, default=False*
This parameter is ignored when `fit_intercept` is set to False. If True, the regressor
before regression by subtracting the mean and dividing by the l2-norm. If you wish t
use `StandardScaler` before calling `fit` on an estimator with `normalize=False`.

Deprecated since version 1.0: `normalize` was deprecated in version 1.0 and will be

sklearn.linear_model.ElasticNet

class `sklearn.linear_model.`**`ElasticNet`**(*alpha=1.0, *, l1_ratio=0.5, fit_intercept=True, normalize='deprecated', precompute=False, max_iter=1000, copy_X=True, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic'*) [source]

Linear regression with combined L1 and L2 priors as regularizer.

Minimizes the objective function:

```
1 / (2 * n_samples) * ||y - Xw||^2_2
+ alpha * l1_ratio * ||w||_1
+ 0.5 * alpha * (1 - l1_ratio) * ||w||^2_2
```

If you are interested in controlling the L1 and L2 penalty separately, keep in mind that this is equivalent to:

```
a * ||w||_1 + 0.5 * b * ||w||_2^2
```

where:

```
alpha = a + b and l1_ratio = a / (a + b)
```

The parameter l1_ratio corresponds to alpha in the glmnet R package while alpha corresponds to the lambda parameter in glmnet. Specifically, l1_ratio = 1 is the lasso penalty. Currently, l1_ratio <= 0.01 is not reliable, unless you supply your own sequence of alpha.

Read more in the User Guide.

Parameters::

alpha : *float, default=1.0*
Constant that multiplies the penalty terms. Defaults to 1.0. See the notes for the exact mathematical meaning of this parameter. `alpha = 0` is equivalent to an ordinary least square, solved by the `LinearRegression` object. For numerical reasons, using `alpha = 0` with the `Lasso` object is not advised. Given this, you should use the `LinearRegression` object.

l1_ratio : *float, default=0.5*
The ElasticNet mixing parameter, with `0 <= l1_ratio <= 1`. For `l1_ratio = 0` the penalty is an L2 penalty. `For l1_ratio = 1` it is an L1 penalty. For `0 < l1_ratio < 1`, the penalty is a combination of L1 and L2.

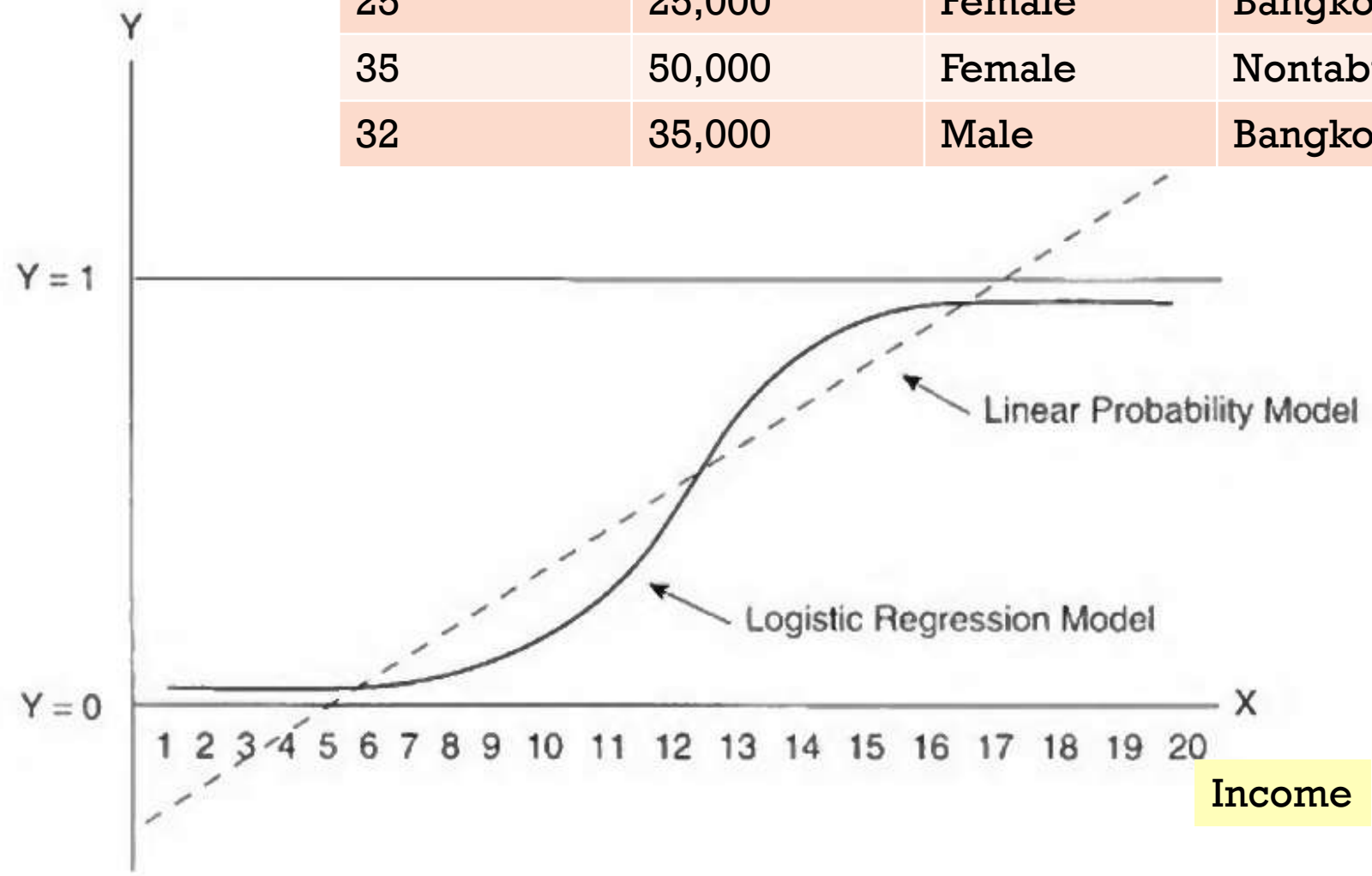
fit_intercept : *bool, default=True*
Whether the intercept should be estimated or not. If `False`, the data is assumed to be already centered.

normalize : *bool, default=False*

Classification Problem

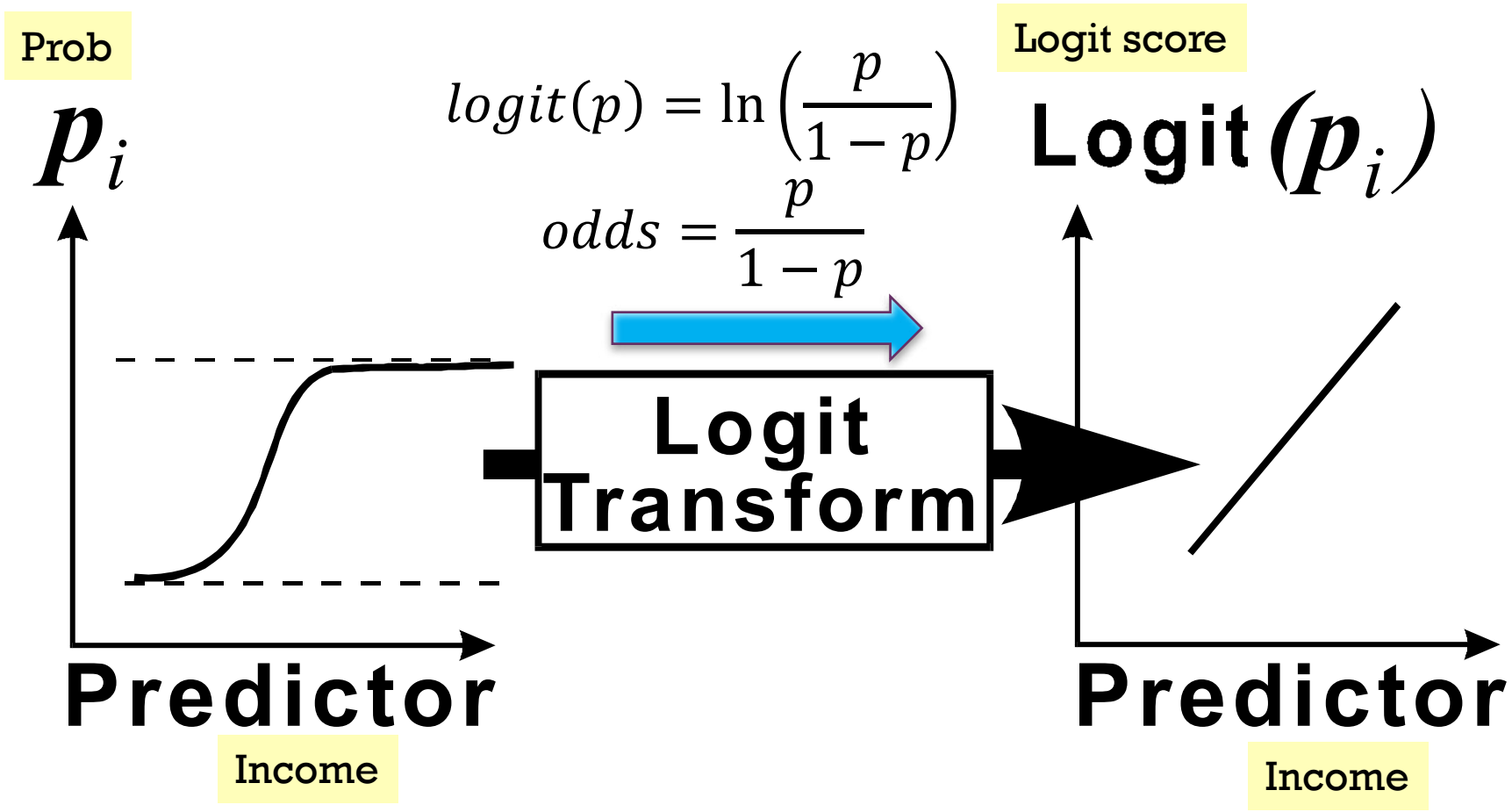
Prob. of purchase

Age	Income	Gender	Province	Purchase
25	25,000	Female	Bangkok	Yes (1)
35	50,000	Female	Nontaburi	Yes (1)
32	35,000	Male	Bangkok	No (0)



Logistic function
Sigmoid function

Logistic Regression (forward pass)
Logit link function: Non-linear to Linear



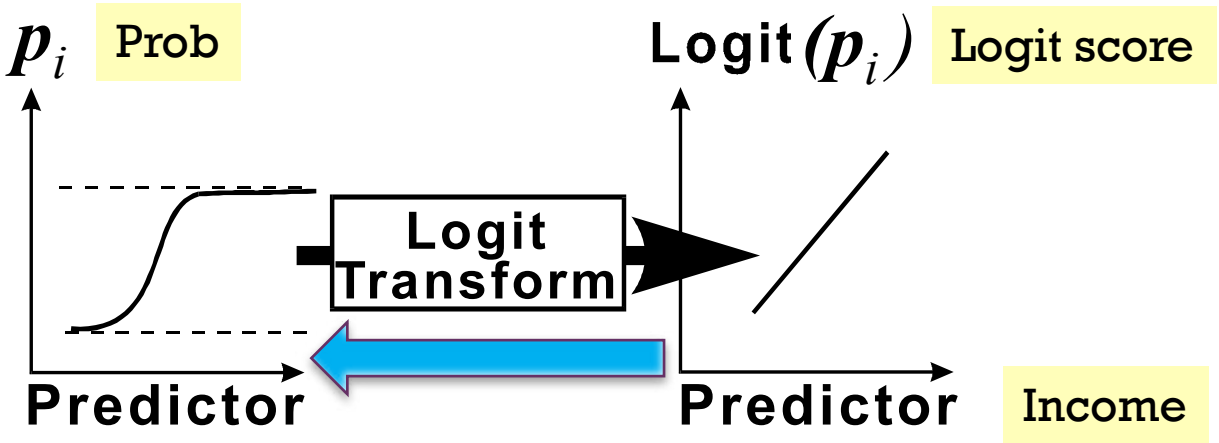
Logit Link Function (backward pass)

$$\log\left(\frac{\hat{p}}{1-\hat{p}}\right) = \hat{w}_0 + \hat{w}_1 x_1 + \hat{w}_2 x_2 = \text{logit}(\hat{p})$$

- Maximum likelihood estimates

$$\hat{p} = \frac{1}{1 + e^{-\text{logit}(\hat{p})}}$$

To obtain prediction estimates, the logit equation is solved for $\hat{p}$.



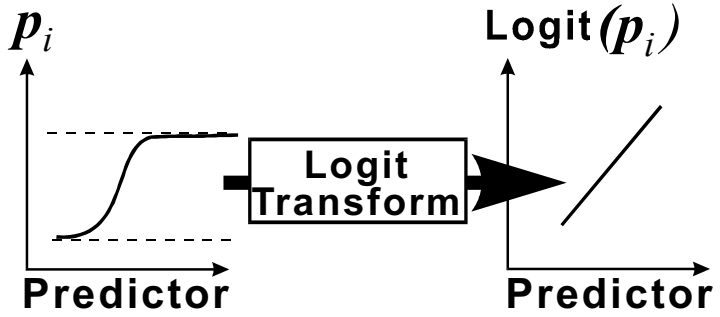
...

Assumptions for Logistic Regression

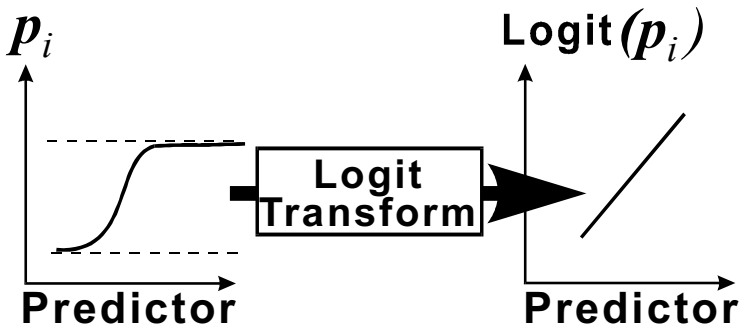
- The mean of the logit is accurately modeled by a linear function of the X_i.
  - (logit, x_i) = linear relationship

- ~~The random error term, ε, is assumed to have a normal distribution with a mean of zero.~~
- ~~The random error term, ε, is assumed to have a constant variance, σ^2.~~
  - ~~Not skew~~

- The errors are independent.



Quiz



- What is the logistic regression prediction for the indicated point?

a. 0.243

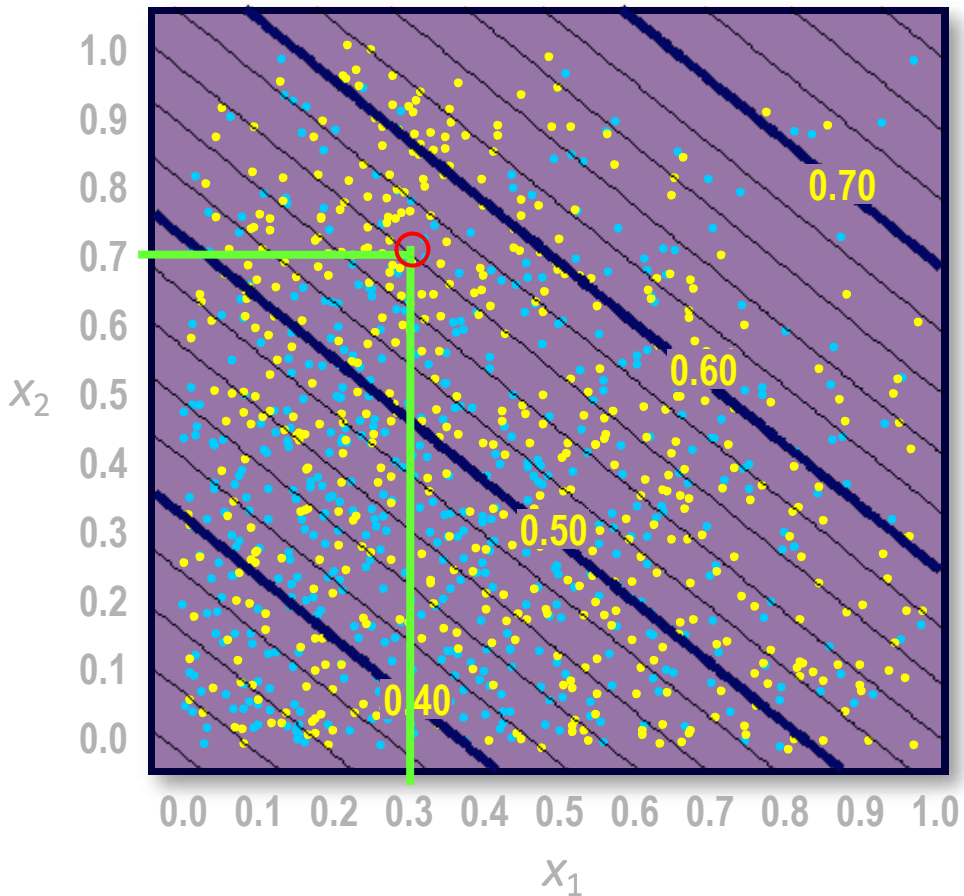
b. 0.56

c. yellow

d. It depends.

$$\text{logit}(\hat{p}) = -0.81 + 0.92\,x_1 + 1.11\,x_2$$

$$\hat{p} = \frac{1}{1 + e^{-\text{logit}(\hat{p})}}$$



sklearn.linear_model.LogisticRegression

class sklearn.linear_model.**LogisticRegression**(*penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None*) ¶ [source]

Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi_class' option is set to 'multi
'sag', 'saga' and 'newton-cg' solvers.)

This class implements regularized logistic regression using
regularization is applied by default. It can handle both de
64-bit floats for optimal performance; any other input form

The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 r
solver supports both L1 and L2 regularization, with a dual f
supported by the 'saga' solver.

Read more in the User Guide.

Examples

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(random_state=0).fit(X, y)
>>> clf.predict(X[:2, :])
array([0, 0])
>>> clf.predict_proba(X[:2, :])
array([[9.8...e-01, 1.8...e-02, 1.4...e-08],
       [9.7...e-01, 2.8...e-02, ...e-08]])
>>> clf.score(X, y)
0.97...
```

Parameters:

penalty : *{'l1', 'l2', 'elasticnet', 'none'}, default='l2'*

Used to specify the norm used in the penalization. The 'newton-cg', 'sag' and 'lbfgs' solvers support only l2 penalties. 'elasticnet' is only supported by the 'saga' solver. If 'none' (not supported by the liblinear solver), no regularization is applied.

New in version 0.19: l1 penalty with SAGA solver (allowing 'multinomial' + L1)

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

l1_ratio : *float, default=None*

The Elastic-Net mixing parameter, with `0 <= l1_ratio <= 1`. Only used if `penalty='elasticnet'`. Setting `l1_ratio=0` is equivalent to using `penalty='l2'`, while setting `l1_ratio=1` is equivalent to using `penalty='l1'`. For `0 < l1_ratio <1`, the penalty is a combination of L1 and L2.

C : *float, default=1.0*

Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

Here, we are going to check how sparsity increases as we increase lambda (or decrease C, as C= 1/ λ) when L1 Regularizer is used.

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C= 1000, penalty= 'l1')
clf.fit(x_train,y_train)
pred = clf.predict(x_test)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 50470
```

```
clf = LogisticRegression(C= 1, penalty= 'l1'
clf.fit(x_train,y_train)
pred = clf.predict(x_test)
print("Non Zero weights:",np.count_nonzer
```

```
Non Zero weights: 8228
```

```
clf = LogisticRegression(C= 0.01, penalty= 'l1')
clf.fit(x_train,y_train)
pred = clf.predict(x_test)
print("Non Zero weights:",np.count_nonzero(clf.coef_)
```

```
Non Zero weights:396
```

https://medium.com/@aditya97p/l1-and-l2-regularization-237438a9caa6

class_weight : *dict or 'balanced', default=None*

Weights associated with classes in the form `{class_label: weight}`. If not given, all classes are supposed to have weight one.

The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as `n_samples / (n_classes * np.bincount(y))`.

Note that these weights will be multiplied with sample_weight (passed through the fit method) if sample_weight is specified.

New in version 0.17: class_weight='balanced'

random_state : *int, RandomState instance, default=None*

Used when `solver` == 'sag', 'saga' or 'liblinear' to shuffle the data. See Glossary for details.

Examples

```
>>> np.bincount(np.arange(5))
array([1, 1, 1, 1, 1])
>>> np.bincount(np.array([0, 1, 1, 3, 2, 1, 7]))
array([1, 3, 1, 1, 0, 0, 0, 1])
```

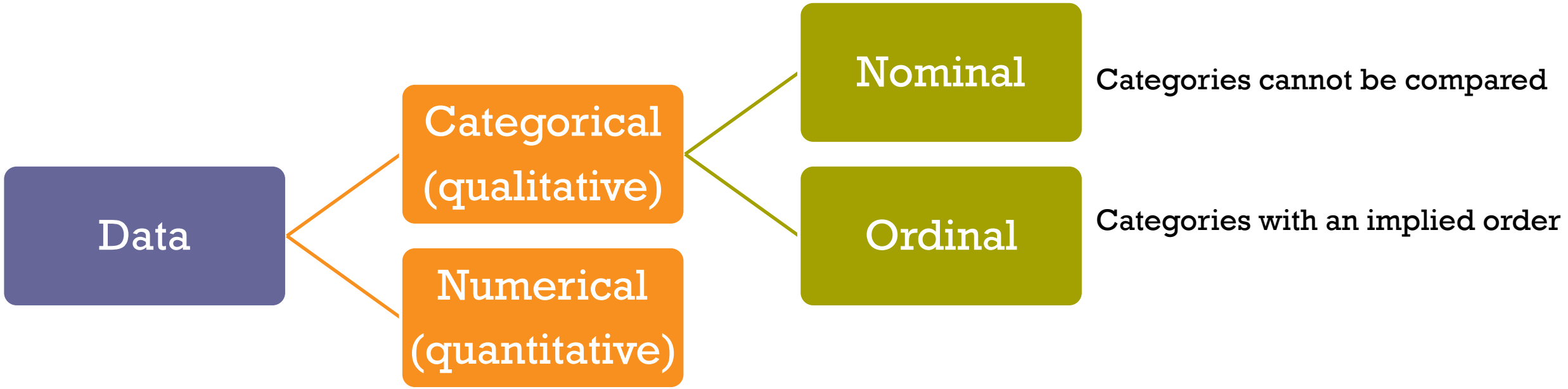
- 100 (20:80)
- W_c1 = 100 / (2 * 20) = 2.5
- W_c2 = 100 / (2 * 80) = 0.625

Linear Regression Limitation

- Manage missing value
- Handle outliers (skewness)
- Handle categorical variables (dummy codes)
- Variable selection:
  - Forward, Backward, Stepwise (not exist in scikit-learn)
- Accounted for nonlinearities

Important Params:
- Require good data preparation
- Variable selection

Terminology: Kinds of data (recap)



Ordinal: Recode

Grade	GradeNum
A	4.00
B+	3.50
B	3.00
C+	2.50
C	2.00
D+	1.50
D	1.00
F	0.00

Size	SizeNum
XL	5
L	4
M	3
S	2
XS	1

Categorical

- Dummy coding = (n-1) dummy codes

Branch	BranchNum	D_BKK	B_Patum	D_Non	D_BKK	B_Patum	
BKK	1	1	0	0	1	0	
Patumtani	2	0	1	0	0	1	
Nontaburi	3	0	0	1	0	0	reference

Search the docs ...

Input/output
General functions
Series
DataFrame
Pandas arrays
Panel
Index objects
Date offsets
Window
GroupBy
Resampling
Style

pandas.get_dummies

pandas.**get_dummies**(*data, prefix=None, prefix_sep='_', dummy_na=False, columns=None, sparse=False, drop_first=False, dtype=None*) → 'DataFrame' [source]

Convert categorical variable into dummy/indicator variables.

Parameters: **data** : *array-like, Series, or DataFrame*

Data of which to get dummy indicators.

prefix : *str, list of str, or dict of str, default None*

String to append DataFrame column names. Pass a list with length equal to the number of columns when calling get_dummies on a DataFrame. Alternatively, *prefix* can be a dictionary mapping column names to prefixes.

prefix_sep : *str, default '_'*

If appending prefix, separator/delimiter to use. Or pass a list or dictionary as with *prefix*.

dummy_na : *bool, default False*

Add a column to indicate NaNs, if False NaNs are ignored.

ke, default None

s in the DataFrame to be encoded. If *columns* is None then all the columns with *object* or *category* converted.

efault False

dummy-encoded columns should be backed by a `SparseArray` (True) or a regular NumPy array (False).

```
>>> pd.get_dummies(pd.Series(list('abcaa')), drop_first=True)
   b  c
0  0  0
1  1  0
2  0  1
3  0  0
4  0  0
```

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html

sklearn.preprocessing.OneHotEncoder

class `sklearn.preprocessing.OneHotEncoder`(*, *categories='auto'*, *drop=None*, *sparse='deprecated'*, *sparse_output=True*, *dtype=<class 'numpy.float64'>*, *handle_unknown='error'*, *min_frequency=None*, *max_categories=None*) [source]

Encode categorical features as a one-hot numeric array.

The input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features. The features are encoded using a one-hot (aka 'one-of-K' or 'dummy') encoding scheme. This creates a binary column for each category and returns a sparse matrix or dense array (depending on the `sparse_output` parameter)

By default, the encoder derives the categories based on the unique values in each feature. Alternatively, the `categories` manually.

This encoding is needed for feeding categorical data to many scikit-learn estimators, notably linear mo standard kernels.

Note: a one-hot encoding of y labels should use a LabelBinarizer instead.

drop : *{'first', 'if_binary'} or an array-like of shape (n_features,), default=None*

Specifies a methodology to use to drop one of the categories per feature. This is useful in situations w perfectly collinear features cause problems, such as when feeding the resulting data into an unregular linear regression model.

However, dropping one category breaks the symmetry of the original representation and can therefore induce a bias in downstream models, for instance for penalized linear classification or regression mod

- None : retain all features (the default).
- 'first' : drop the first category in each feature. If only one category is present, the feature will be dropped entirely.
- 'if_binary' : drop the first category in each feature with two categories. Features with 1 or more than 2 categories are left intact.
- array : `drop[i]` is the category in feature `X[:, i]` that should be dropped.

New in version 0.21: The parameter `drop` was added in 0.21.

Changed in version 0.23: The option `drop='if_binary'` was added in 0.23.

Changed in version 1.1: Support for dropping infrequent categories.

```
>>> X = [['Male', 1], ['Female', 3], ['Female', 2]]
```

One can always drop the first column for each feature:

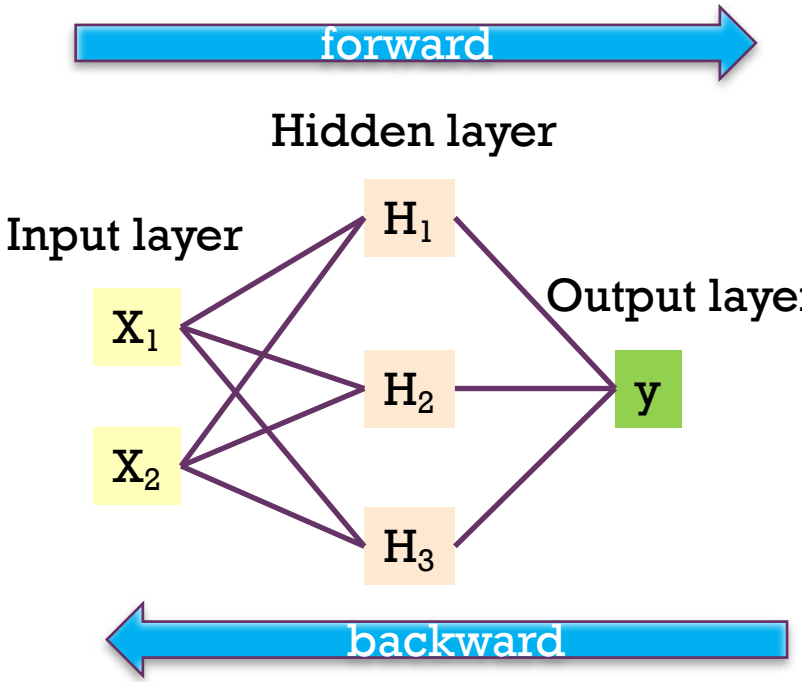
```
>>> drop_enc = OneHotEncoder(drop='first').fit(X)
>>> drop_enc.categories_
[array(['Female', 'Male'], dtype=object), array([1, 2, 3], dtype=object)]
>>> drop_enc.transform([['Female', 1], ['Male', 2]]).toarray()
array([[0., 0., 0.],
       [1., 1., 0.]])
```

Or drop a column for feature only having 2 categories:

```
>>> drop_binary_enc = OneHotEncoder(drop='if_binary').fit(X)
>>> drop_binary_enc.transform([['Female', 1], ['Male', 2]]).toarray()
array([[0., 1., 0., 0.],
       [1., 0., 1., 0.]])
```

3) Neural Networks

3) Neural Networks (universal approximator)

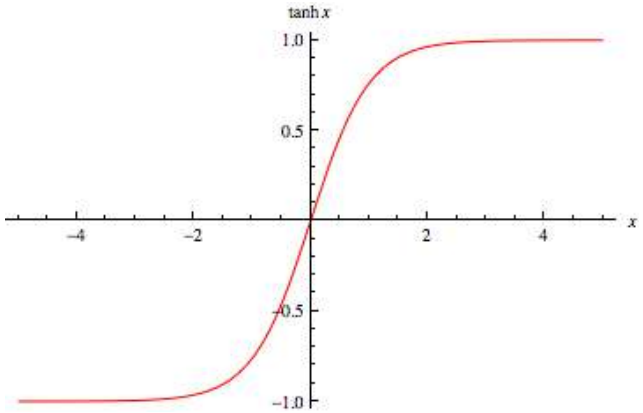


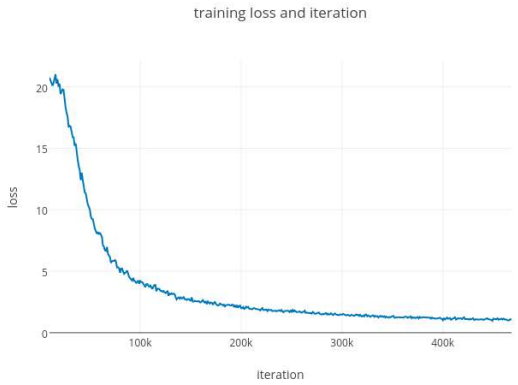
$$log\left(\frac{\hat{p}}{1-\hat{p}}\right) = \hat{w}_0 + \hat{w}_1 H_1 + \hat{w}_2 H_2 + \hat{w}_3 H_3$$

$$H_1 = tanh(\hat{w}_{10} + \hat{w}_{11} x_1 + \hat{w}_{12} x_2)$$

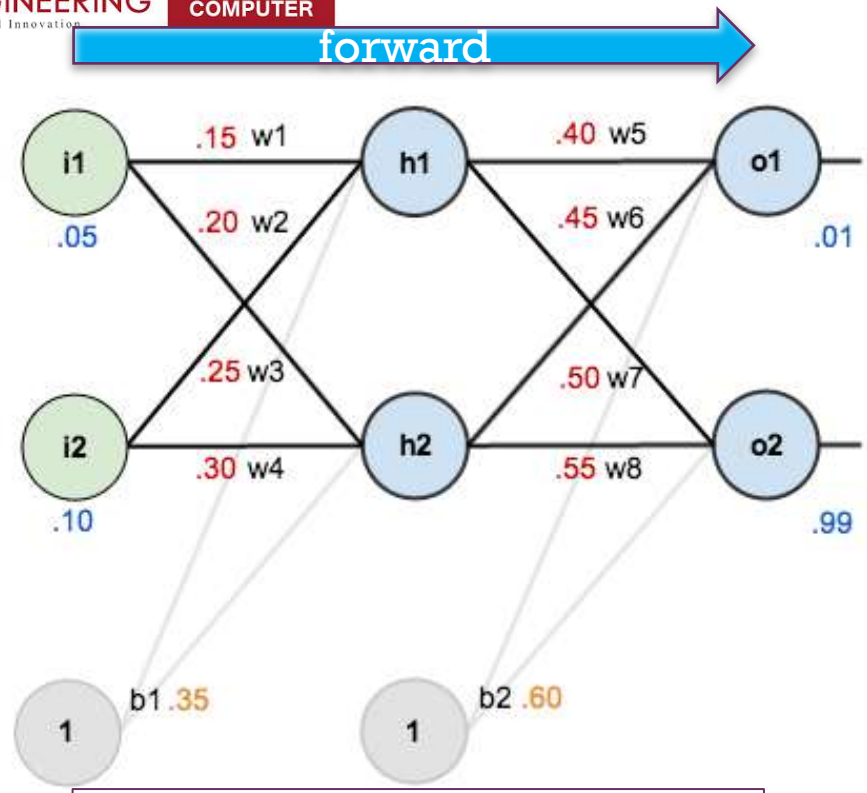
$$H_2 = tanh(\hat{w}_{20} + \hat{w}_{21} x_1 + \hat{w}_{22} x_2)$$

$$H_3 = tanh(\hat{w}_{30} + \hat{w}_{31} x_1 + \hat{w}_{32} x_2)$$



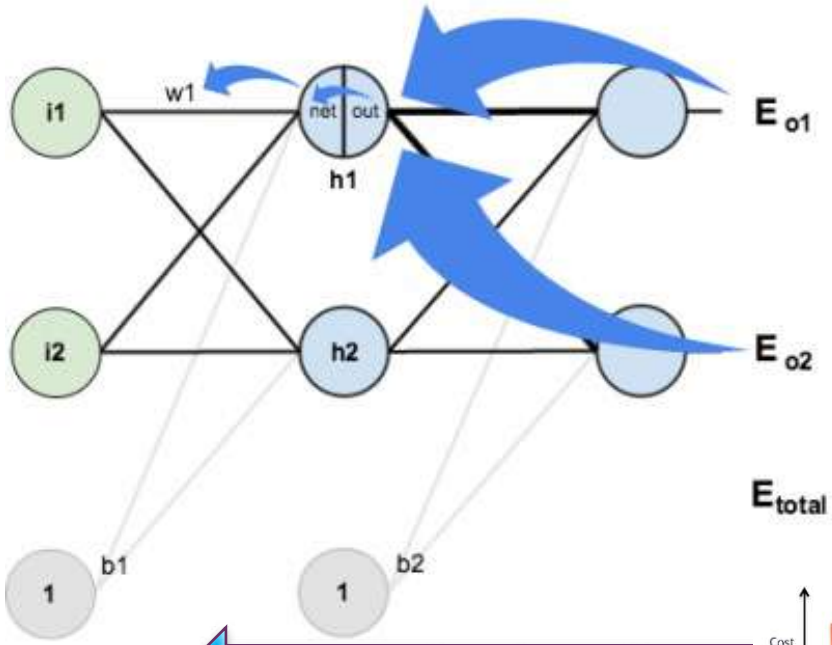


How to update weight
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

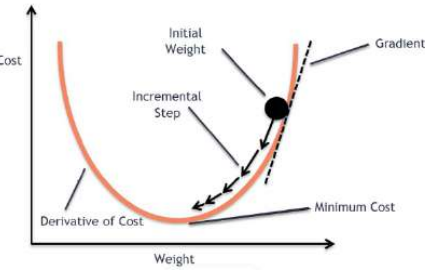


$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



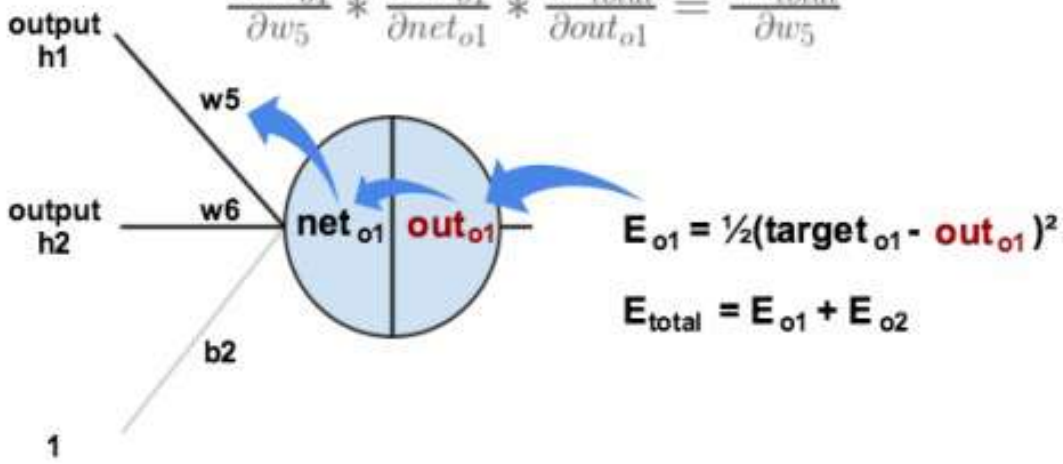
$E_{total} = E_{o1} + E_{o2}$



backward

Putting it all together:

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

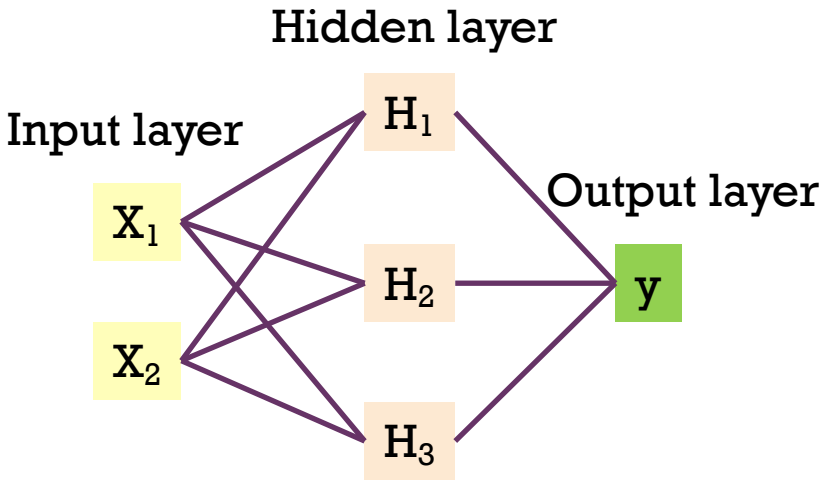
$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

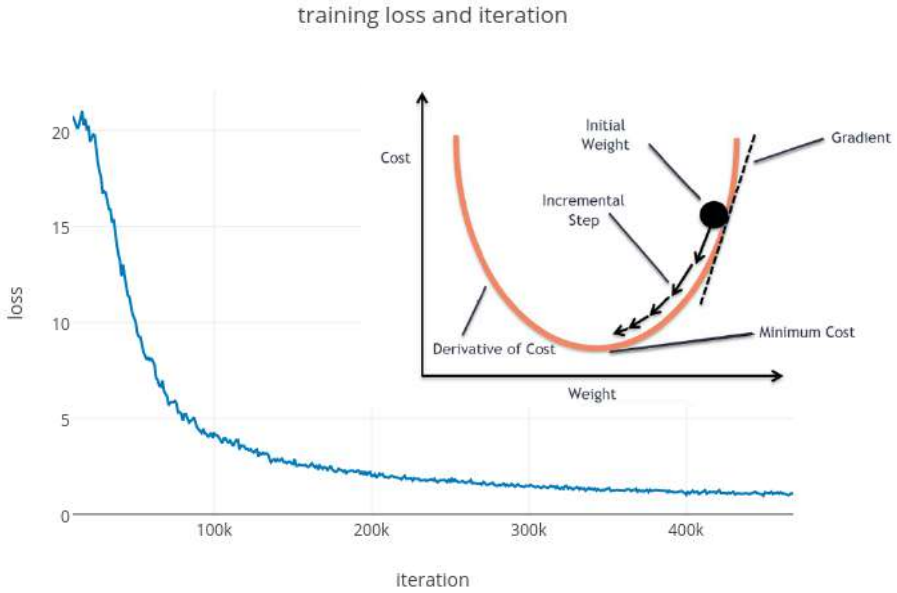
$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

Loss function

- Mean Squared Error (MSE)
- Binary Crossentropy (BCE)
- Categorical Crossentropy (CC)
- Sparse Categorical Crossentropy (SCC)

- https://shiva-verma.medium.com/understanding-different-loss-functions-for-neural-networks-dd1ed0274718
- https://programmathically.com/an-introduction-to-neural-network-loss-functions/





Stop when?
- Converge (no change in loss)
- Max epochs

Important Params:
- #hidden units, #hidden layers
- Learning rate, momentum, decay
- Seed number, etc.

cce vs. scce

Simply:

- `categorical_crossentropy` (`cce`) produces a one-hot array containing the proba match for each category,
- `sparse_categorical_crossentropy` (`scce`) produces a category index of the *mos* matching category.

Many categorical models produce `scce` output because you save space, but lose A LOT of information (for example, in the 2nd example, index 2 was also very close.) I generally prefer `cce` output for model reliability.

- Categorical Cross Entropy [Doc]:

> Use this crossentropy loss function when there are two or more label classes. We expect labels to be provided in a one_hot representation.

One Hot

```
>>> y_true = [[0, 1, 0], [0, 0, 1]]
>>> y_pred = [[0.05, 0.95, 0], [0.1, 0.8, 0.1]]
>>> # Using 'auto'/'sum_over_batch_size' reduction type.
>>> cce = tf.keras.losses.CategoricalCrossentropy()
>>> cce(y_true, y_pred).numpy()
1.177
```

- Sparse Categorical Cross Entropy [Doc]:

> Use this crossentropy loss function when there are two or more label classes. We expect labels to be provided as integers.

running number

```
>>> y_true = [1, 2]
>>> y_pred = [[0.05, 0.95, 0], [0.1, 0.8, 0.1]]
>>> # Using 'auto'/'sum_over_batch_size' reduction type.
>>> scce = tf.keras.losses.SparseCategoricalCrossentropy()
>>> scce(y_true, y_pred).numpy()
1.177
```

One good example of the sparse-categorical-cross-entropy is the fasion-mnist dataset.

```
import tensorflow as tf
from tensorflow import keras

fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()

print(y_train_full.shape) # (60000,)
print(y_train_full.dtype) # uint8

y_train_full[:10]
# array([9, 0, 0, 3, 0, 2, 7, 2, 5, 5], dtype=uint8)
```

https://stackoverflow.com/questions/58565394/what-is-the-difference-between-sparse-categorical-crossentropy-and-categorical-c

Prev Up Next

scikit-learn 0.22.2
Other versions

Please cite us if you use the software.

sklearn.neural_network.MLPClassifier
Examples using sklearn.neural_network.MLPClass

sklearn.neural_network.MLPClassifier

class `sklearn.neural_network.` **`MLPClassifier`**(*hidden_layer_sizes=(100,), activation='relu', solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*) ¶ [source]

Multi-layer Perceptron classifier.

This model optimizes the log-loss function using LBFGS or stochastic gradient descent.

New in version 0.18.

Parameters:

hidden_layer_sizes : *tuple, length = n_layers - 2, default=(100,)*
The ith element represents the number of neurons in the ith hidden layer.

activation : *{'identity', 'logistic', 'tanh', 'relu'}, default='relu'*
Activation function for the hidden layer.

- 'identity', no-op activation, useful to implement linear bottleneck, returns f(x) = x
- 'logistic', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- 'tanh', the hyperbolic tan function, returns f(x) = tanh(x).
- 'relu', the rectified linear unit function, returns f(x) = max(0, x)

solver : *{'lbfgs', 'sgd', 'adam'}, default='adam'*
The solver for weight optimization.

sklearn.neural_network.MLPRegressor

class `sklearn.neural_network.` **`MLPRegressor`**(*hidden_layer_sizes=(100,), activation='relu', solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*) ¶ [source]

Multi-layer Perceptron regressor.

This model optimizes the squared-loss using LBFGS or stochastic gradient descent.

New in version 0.18.

Parameters:

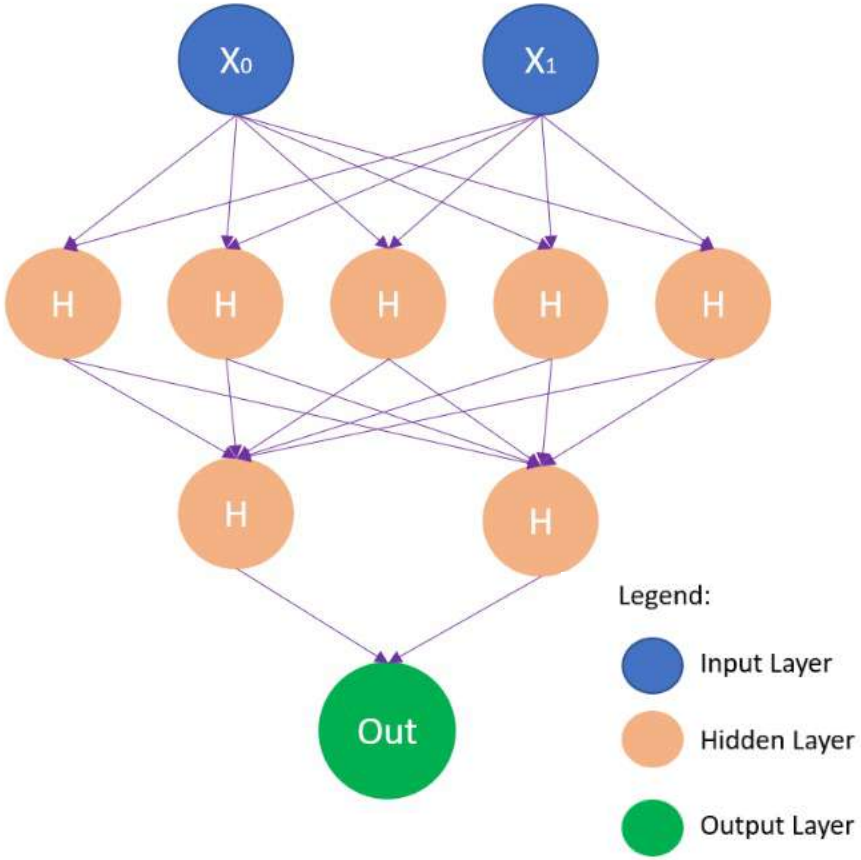
hidden_layer_sizes : *tuple, length = n_layers - 2, default=(100,)*
The ith element represents the number of neurons in the ith hidden layer.

activation : *{'identity', 'logistic', 'tanh', 'relu'}, default='relu'*
Activation function for the hidden layer.

- 'identity', no-op activation, useful to implement linear bottleneck, returns f(x) = x
- 'logistic', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- 'tanh', the hyperbolic tan function, returns f(x) = tanh(x).
- 'relu', the rectified linear unit function, returns f(x) = max(0, x)

solver : *{'lbfgs', 'sgd', 'adam'}, default='adam'*
The solver for weight optimization.

Exmaple



batch_size : *int, default='auto'*

Size of minibatches for stochastic optimizers. If the solver is 'lbfgs', the classifier will not use minibatch. When set to "auto", `batch_size=min(200, n_samples)`.

learning_rate : *{'constant', 'invscaling', 'adaptive'}, default='constant'*

Learning rate schedule for weight updates.

```
mlp_clf = MLPClassifier(hidden_layer_sizes=(5,2),
                        max_iter = 300,activation = 'relu',
                        solver = 'adam')
```

```
param_grid = {
    'hidden_layer_sizes': [(150,100,50), (120,80,40), (100,5(
    'max_iter': [50, 100, 150],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
```

```
grid = GridSearchCV(mlp_clf, param_grid, n_jobs= -1, cv=5)
grid.fit(trainX_scaled, trainY)


print(grid.best_params_)
```

https://michael-fuchs-python.netlify.app/2021/02/03/nn-multi-layer-perceptron-classifier-mlpclassifier/

4) kNN

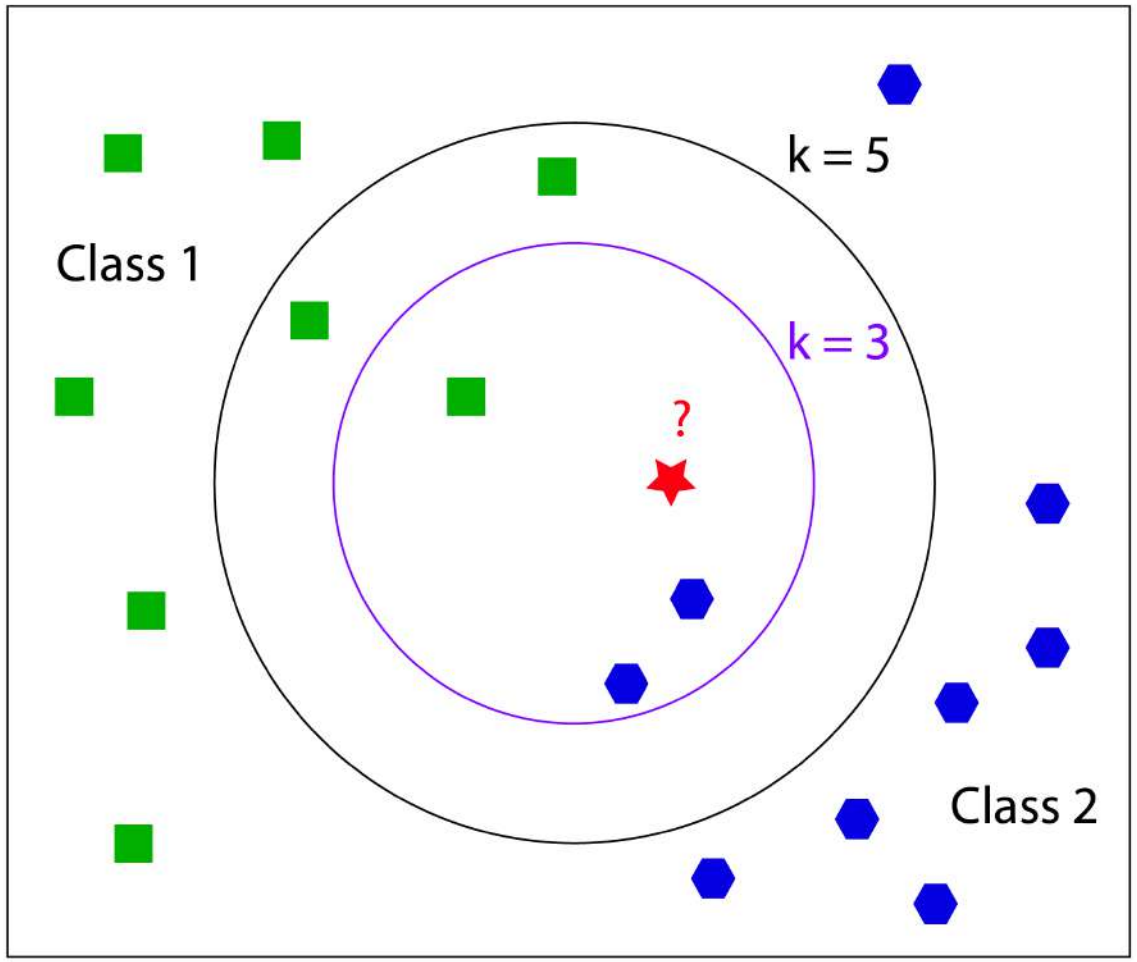
4) k-Nearest Neighbors (kNN)

Important Params:
- k
- Distance function

$Age' = \frac{Age - \mu}{\sigma}$ | $Inc' = \frac{Inc - \mu}{\sigma}$

- Memory based learning
- Suitable for small data sets
- Merge
  - Voting
  - Average
  - Maximum prob
- Cautions:
  - Support only numerical variables
  - Need to adjust variable range

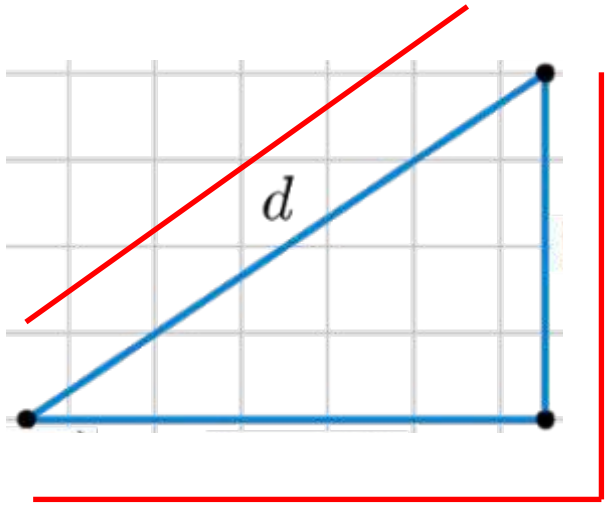
* ก่อนจะ distance ต้อง scaling ก่อน



Distance Function

- Euclidean Distance

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$



- Manhattan Distance

$$\sum_{i=1}^{k}|x_i - y_i|$$

- Minkowski Distance

$$\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$$

q = 1 is Manhattan Distance
q = 2 is Euclidean Distance

Distance Function (cont.)

- Hamming distance: same (0) vs diff (1)
- It still supports only numeric variable.

$$D_H = \sum_{i=1}^{k} |x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

X	Y	Distance
Male	Male	0
Male	Female	1

Nearest Neighbors

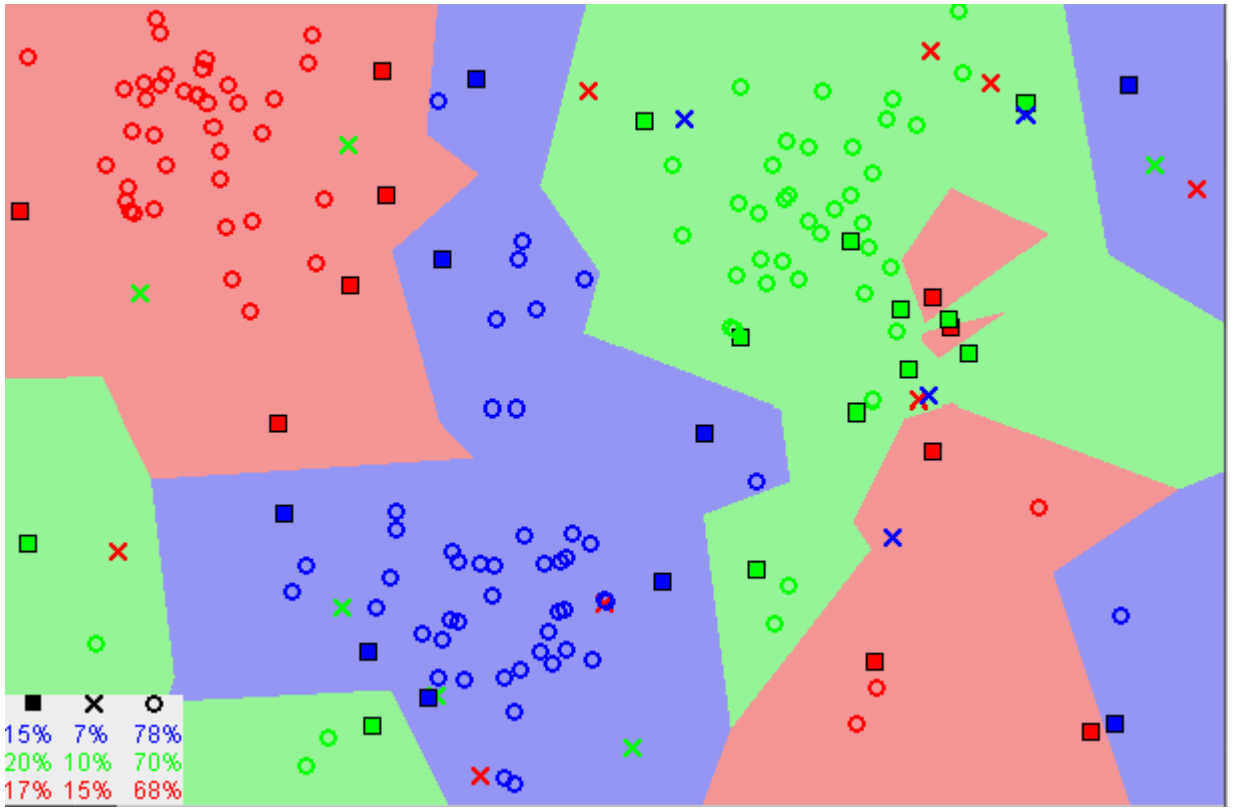


Image showing how similar data points typically exist close to each other

Source : https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

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=None, **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.

ยิ่งใกล้ weight ยิ่งเยอะ

function ที่เราเขียนขึ้นเอง

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.

sklearn.neighbors.KNeighborsRegressor

class sklearn.neighbors.**KNeighborsRegressor**(*n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs*) [sourc

Regression based on k-nearest neighbors.

The target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set.

Read more in the User Guide.

New in version 0.9.

Parameters:

n_neighbors : *int, optional (default = 5)*
Number of neighbors to use by default for kneighbors queries.

weights : *str or callable*
weight function used in prediction. Possible values:

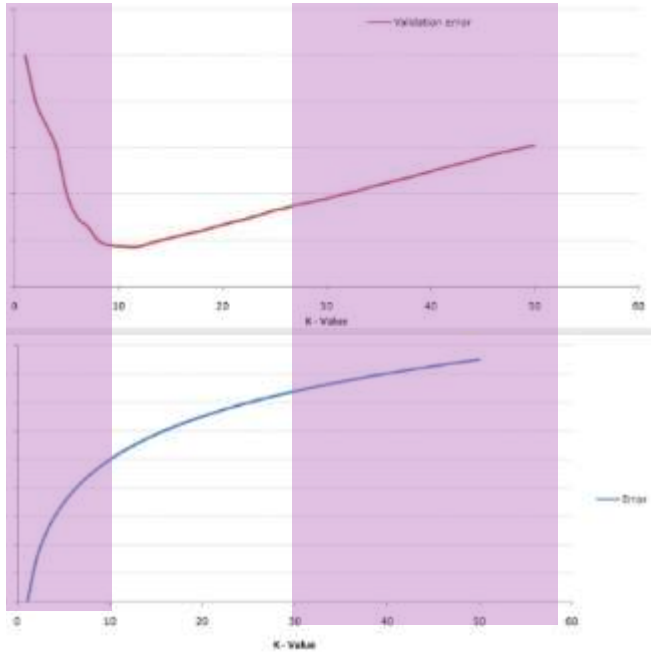
- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function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

Uniform weights are used by default.

Choosing the right value of K

Too Underfitted

Too Overfitted



Search the best parameters with …

sklearn.model_selection.GridSearchCV

class sklearn.model_selection.**GridSearchCV**(*estimator, param_grid, *, scoring=None, n_jobs=None, iid='deprecated', refit=True, cv=None, verbose=0, pre_dispatch='2*n_jobs', error_score=nan, return_train_score=False*) [source]

sklearn.model_selection.RandomizedSearchCV

class sklearn.model_selection.**RandomizedSearchCV**(*estimator, param_distributions, *, n_iter=10, scoring=None, n_jobs=None, iid='deprecated', refit=True, cv=None, verbose=0, pre_dispatch='2*n_jobs', random_state=None, error_score=nan, return_train_score=False*) [source]

```
param_grid=dict(
    n_neighbors=[1,2,3,4,5,6,7,8,9,10],
    weights=['uniform', 'distance'],
),
```

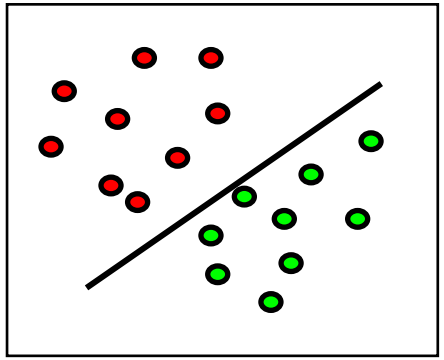
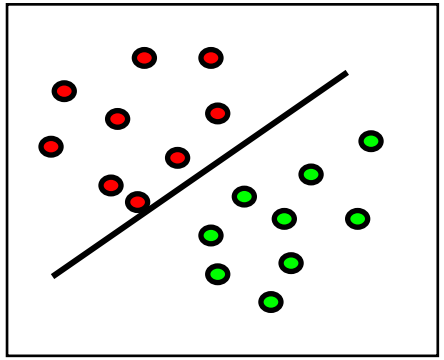
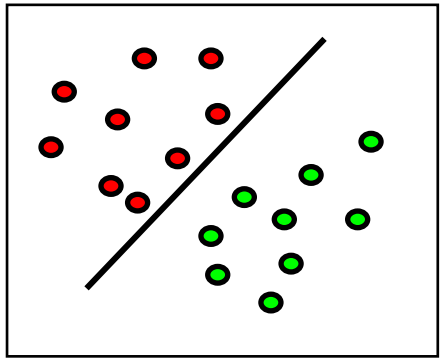
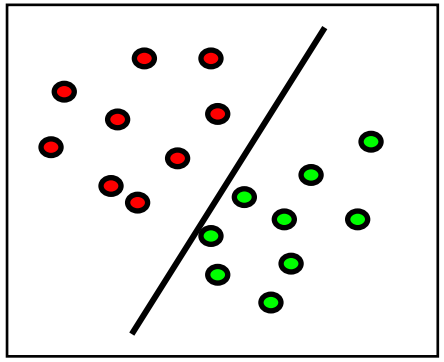
Search with k: 1-10
on uniform/distance weights

5) Support Vector Machine

Adapt from ML Class by David Sontag, New York University
https://people.csail.mit.edu/dsontag/courses/ml16/slides/

Linear Separators

- If training data is linearly separable, perceptron is guaranteed to find some linear separator
- Which of these is optimal?

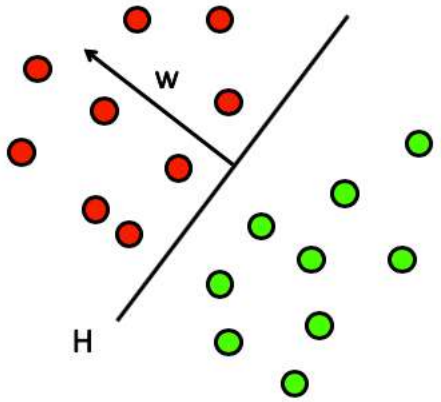





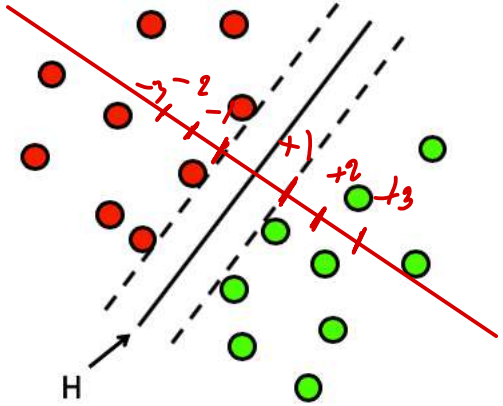
Support Vector Machine (SVM)

- SVMs (Vapnik, 1990's) choose the linear separator with the largest margin

$$H = \{x \mid \langle w, x \rangle + b = 0\}$$

Dot product





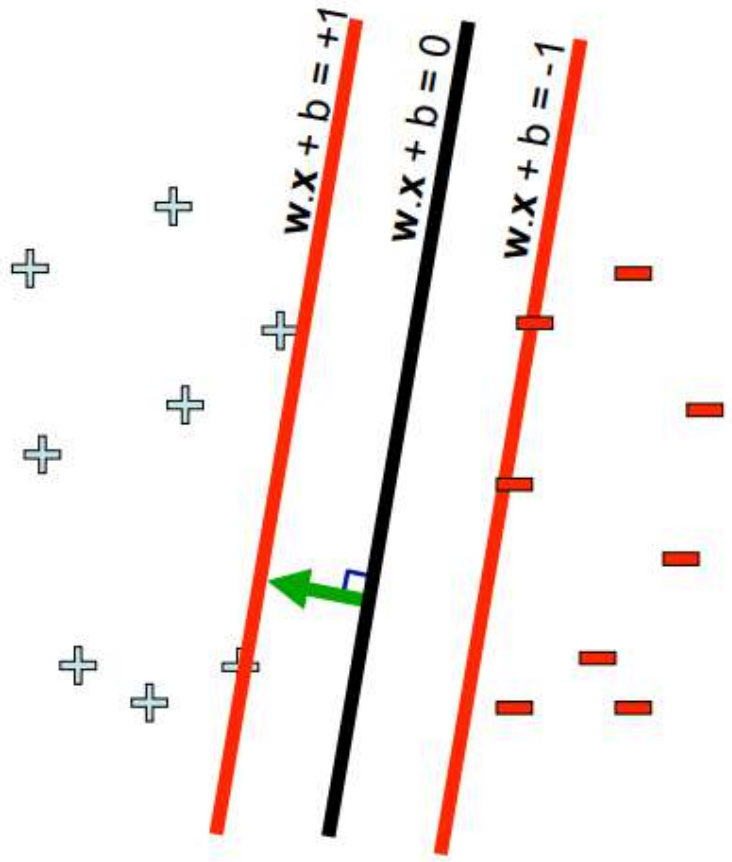
- Good according to intuition, theory, practice
- SVM became famous when, using images as input, it gave accuracy comparable to neural-network with hand-designed features in a handwriting recognition task

$wx+b$

Support vector machines: 3 key ideas

1. Use **optimization** to find solution (i.e., a hyperplane) with few errors
2. Seek **large margin** separator to improve generalization
3. Use **kernel trick** to make large feature spaces computationally efficient

Key 1: Optimization



For every data point (x_t, y_t), enforce the constraint

for $y_t = +1$, $w \cdot x_t + b \geq 1$

and for $y_t = -1$, $w \cdot x_t + b \leq -1$

Equivalently, we want to satisfy all of the linear constraints

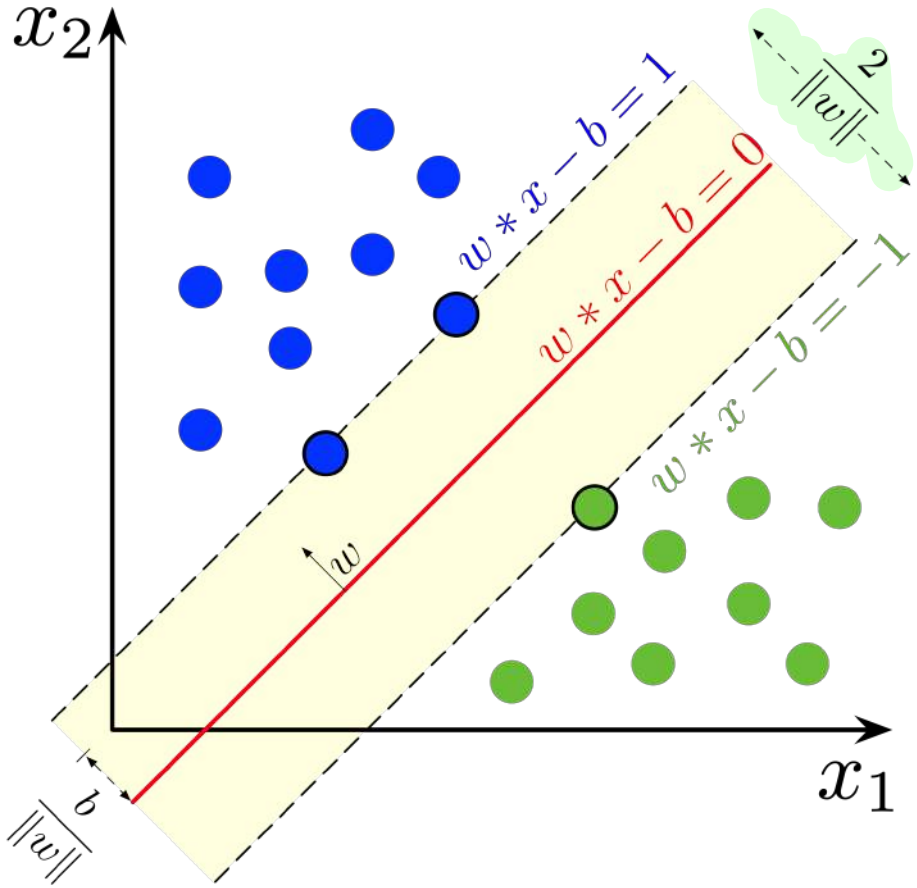
$$y_t \left(w \cdot x_t + b \right) \geq 1 \quad \forall t$$

This *linear program* can be efficiently solved using algorithms such as simplex, interior point, or ellipsoid

Key 2: Seek Large Margin Hard-margin SVM

- Allow for slack to improve generalization.

margin = $\frac{2}{\|w\|}$



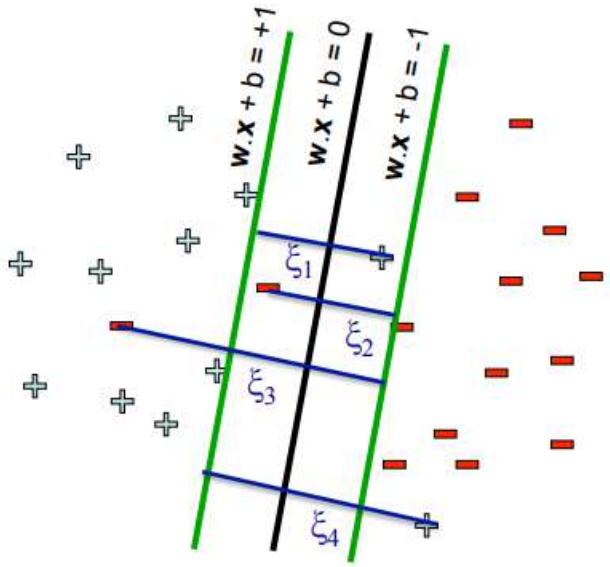
$$y_t \left(w \cdot x_t + b \right) \geq 1 \quad \forall t$$

Minimize $\| w \|^2$

Key 2: Seek Large Margin
Soft-margin SVM

- Allow for slack to improve generalization.

C↑ → margin↓
C↓ → margin↑



$$\text{minimize}_{\mathbf{w},b,\xi} \; \Sigma_j \xi_j \quad \text{(error)}$$

$$(\mathbf{w}.\mathbf{x}_j + b)\, y_j \geq 1 - \xi_j \;, \forall j \;\; \xi_j \geq 0$$

"slack variables"

We now have a linear program again, and can efficiently find its optimum

C เยอะ: ⇒ เน้น Hard margin
C น้อย ⇒ เน้น Soft margin

For each data point:

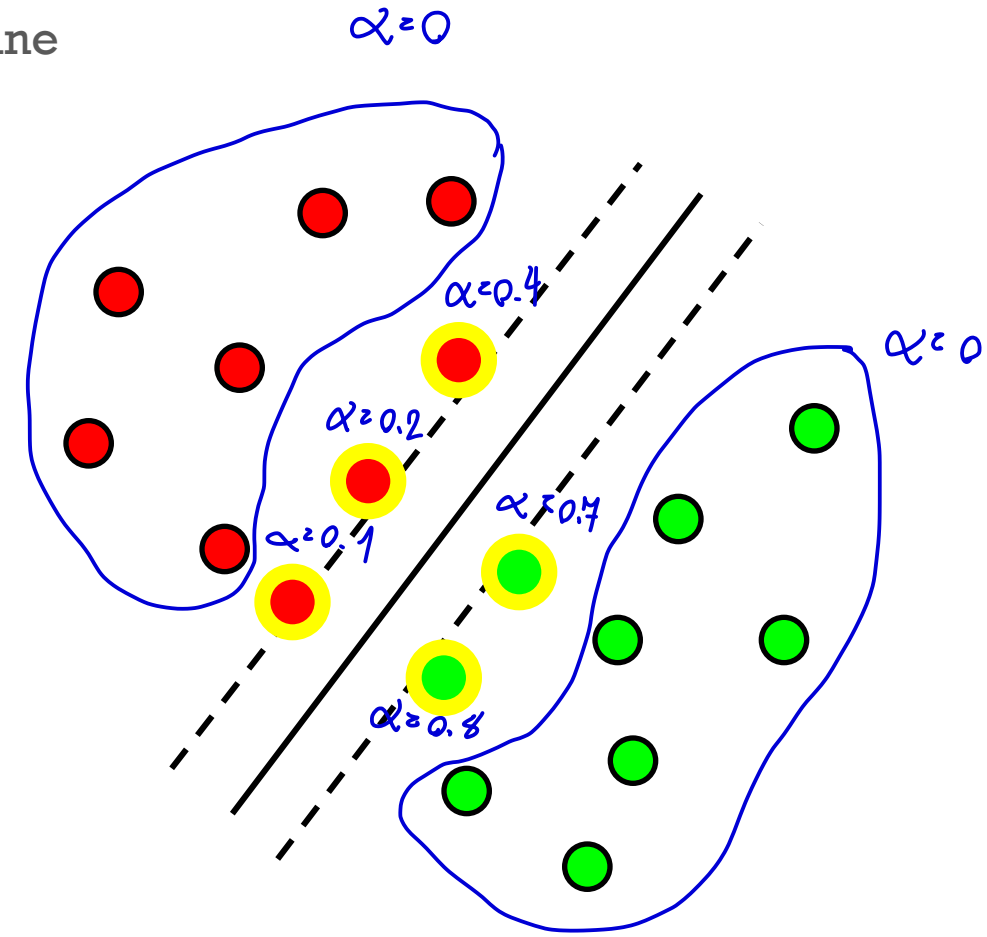
- If functional margin ≥ 1, don't care
- If functional margin < 1, pay linear penalty

Minimize $\| w \|^2 + C \cdot \sum_i \xi_i$

What Are the Support Vectors?

- "Carrying vectors"
- The points, located closest to the hyperplane
- Determining the location of the hyperplane
- All other data points have $\alpha_i = 0$.

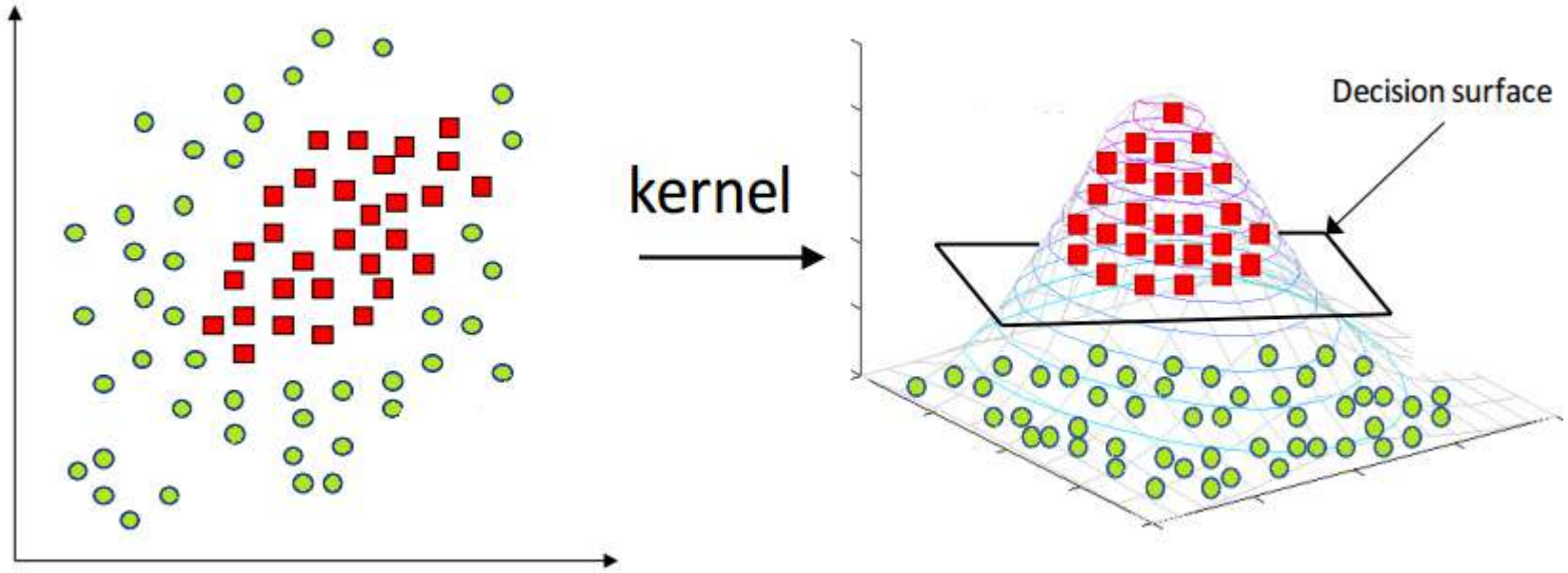
$$w = \sum_{i=1}^{\#sv} \alpha_i y_i x_i^{sv}$$



Prediction = sign(w*x + b)

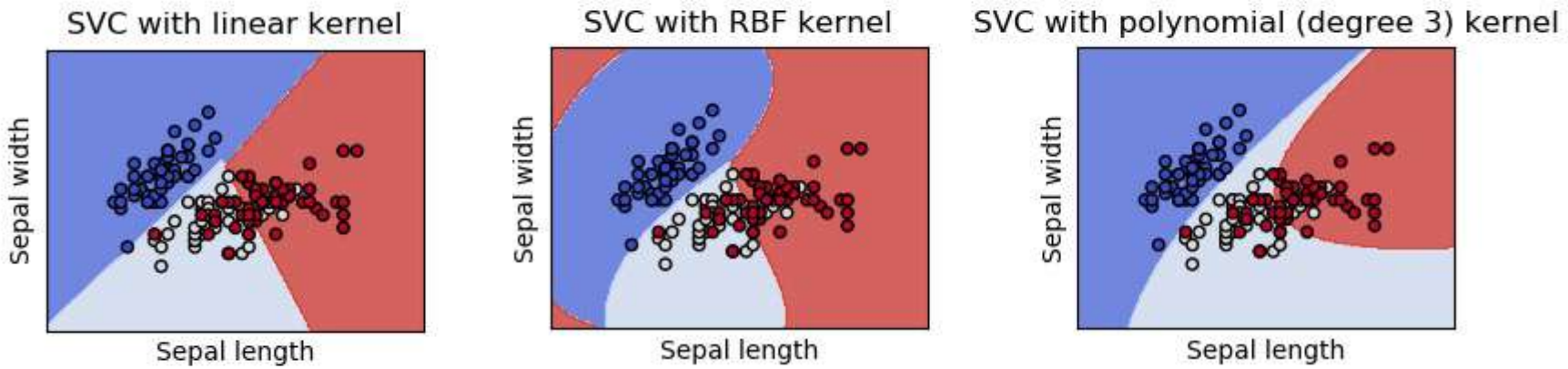
Key 3: Kernel

- Kernel allows us to operate in the original feature space without computing the coordinates of the data in a higher dimensional space.



Source : https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d

Different Kernel on iris dataset



$$w = \sum_{i=1}^{\#sv} \alpha_i y_i x_i^{sv}$$

RBF: Radial Basis Function

$$K(x_i, x_j) = e^{-\gamma(x_i - x_j)^2}$$

Polynomial

$$K(x_i, x_j) = (x_i \cdot x_j + 1)^p$$

Prediction = sign(w*x + b)

https://scikit-learn.org/stable/auto_examples/svm/plot_iris_svc.html

SVM in sklearn (Classifier)

sklearn.svm.SVC

class `sklearn.svm.` **SVC**(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None) [source]

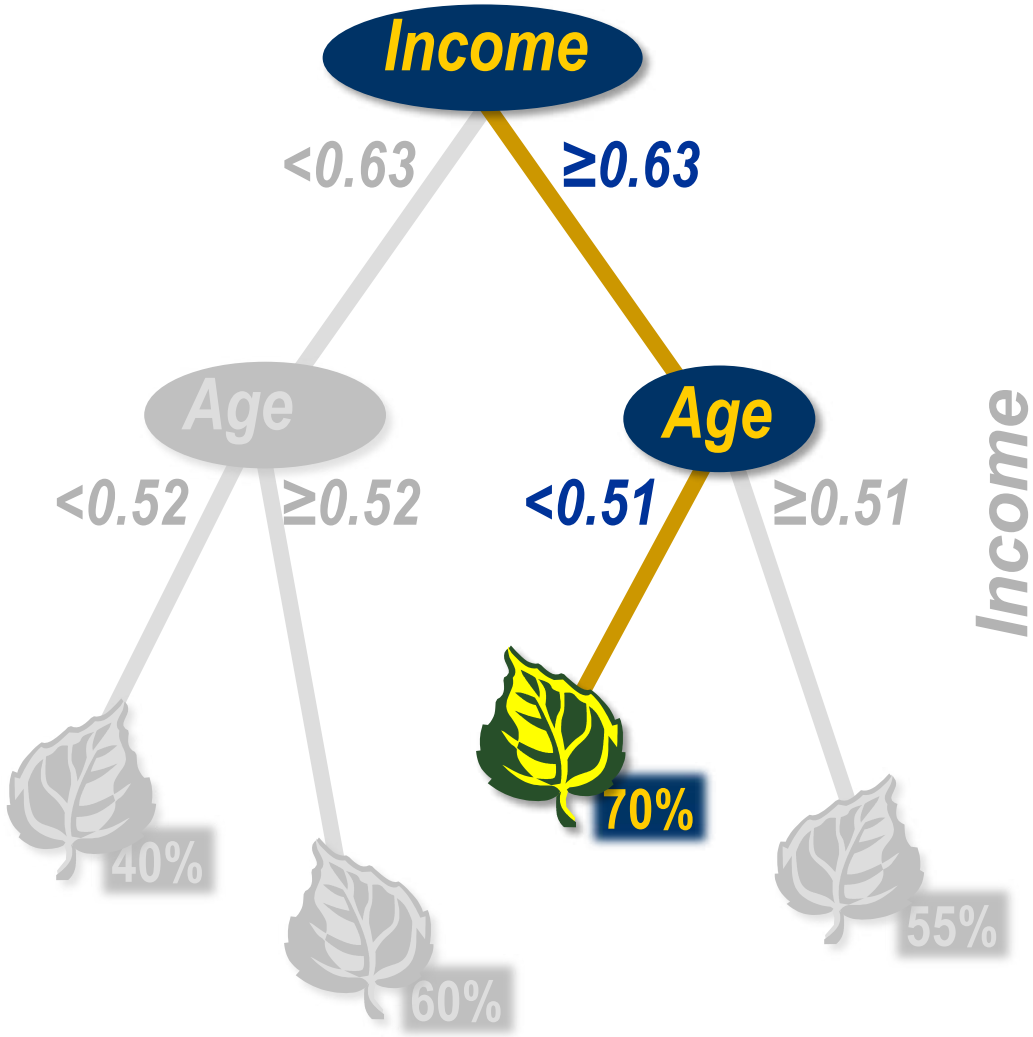
C-Support Vector Classification.

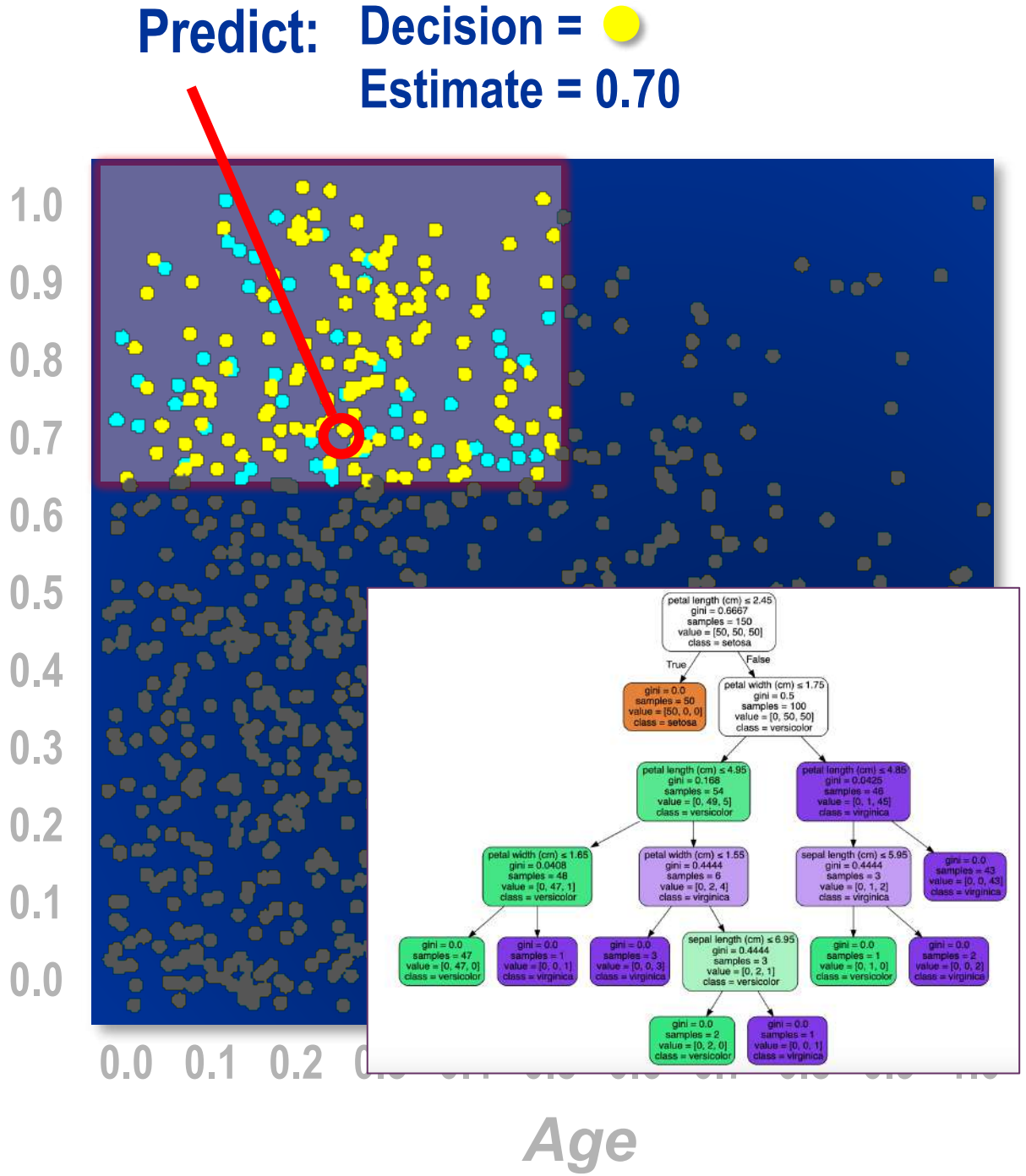
- Parameters
  - **kernel** : Can be 'linear', 'poly', 'rbf', 'sigmoid' or callable function (custom).
  - **C** : For regularization. A larger value creates a larger of margin hyperplane.
  - **gamma** : Determine how model fit to training data.

+

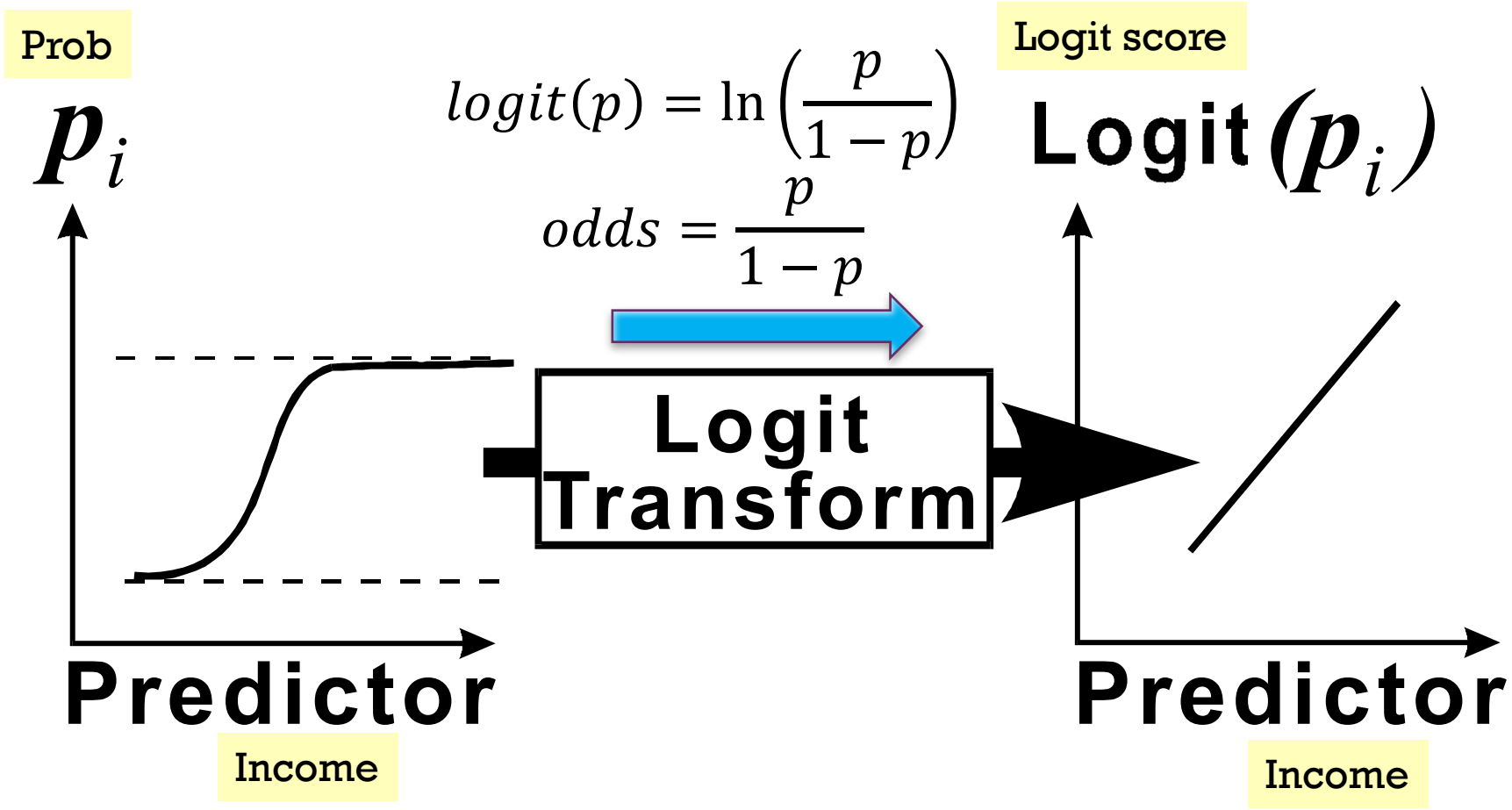
Multiclass & Multi-Label

1) Decision Tree (recap)

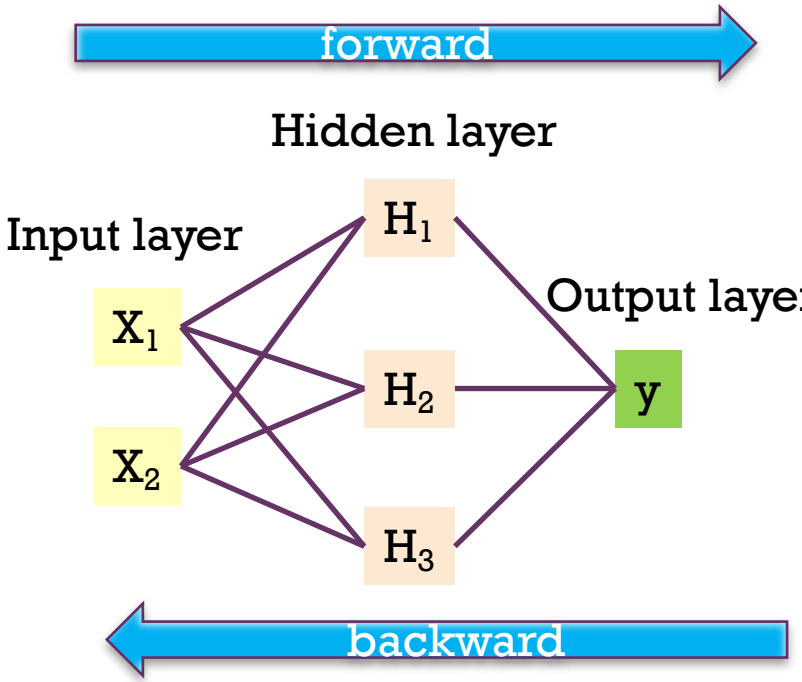




2) Regression (recap)



3) Neural Networks (recap)

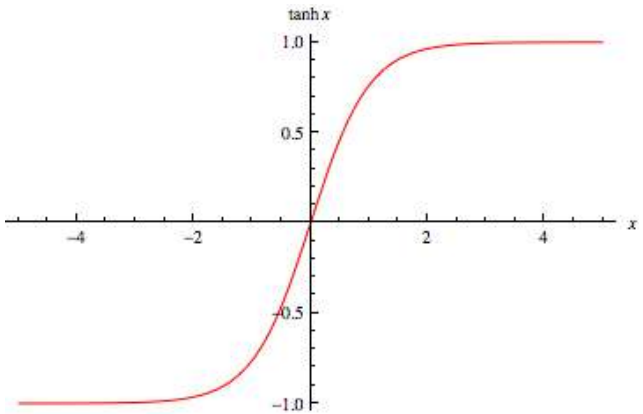


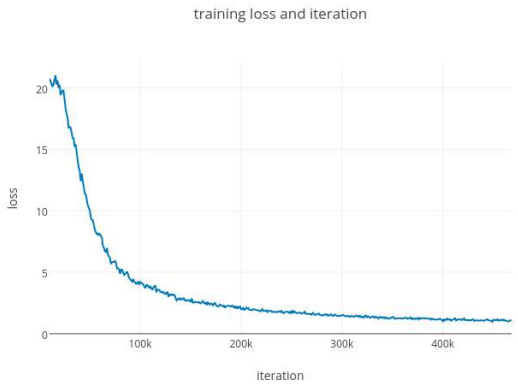
$$log\left(\frac{\hat{p}}{1-\hat{p}}\right) = \hat{w}_0 + \hat{w}_1 H_1 + \hat{w}_2 H_2 + \hat{w}_3 H_3$$

$$H_1 = tanh(\hat{w}_{10} + \hat{w}_{11} x_1 + \hat{w}_{12} x_2)$$

$$H_2 = tanh(\hat{w}_{20} + \hat{w}_{21} x_1 + \hat{w}_{22} x_2)$$

$$H_3 = tanh(\hat{w}_{30} + \hat{w}_{31} x_1 + \hat{w}_{32} x_2)$$

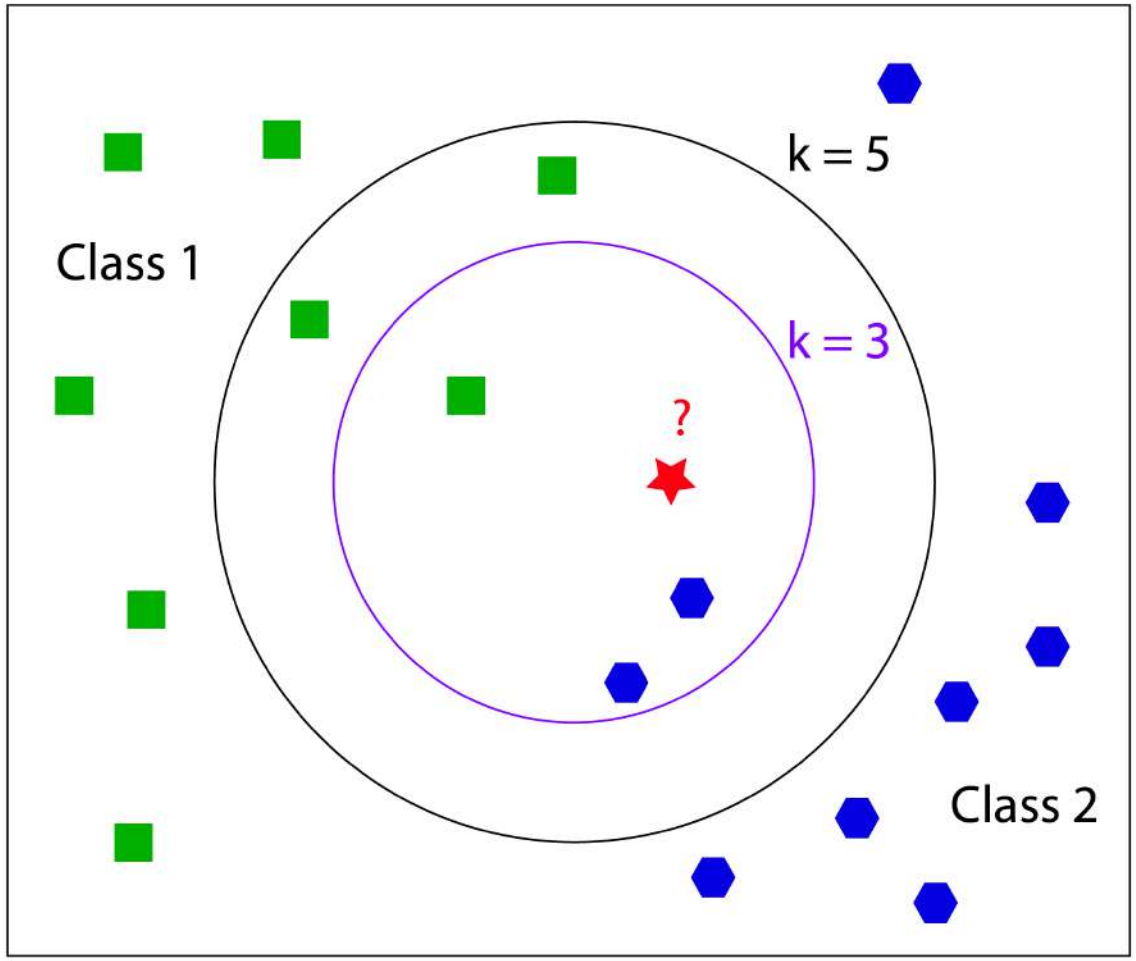




How to update weight
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

4) k-Nearest Neighbors (recap)

- Memory based learning
- Suitable for small data sets
- Merge
  - Voting
  - Average
  - Maximum prob
- Cautions:
  - Support only numerical variables
  - Need to adjust variable range

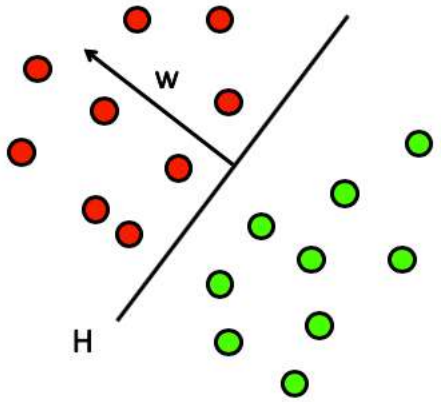


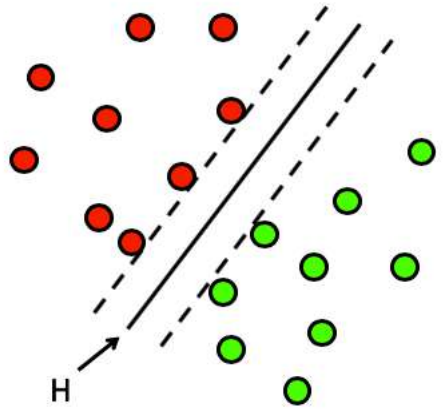
5) SVM (recap)

- SVMs (Vapnik, 1990's) choose the linear separator with the largest margin

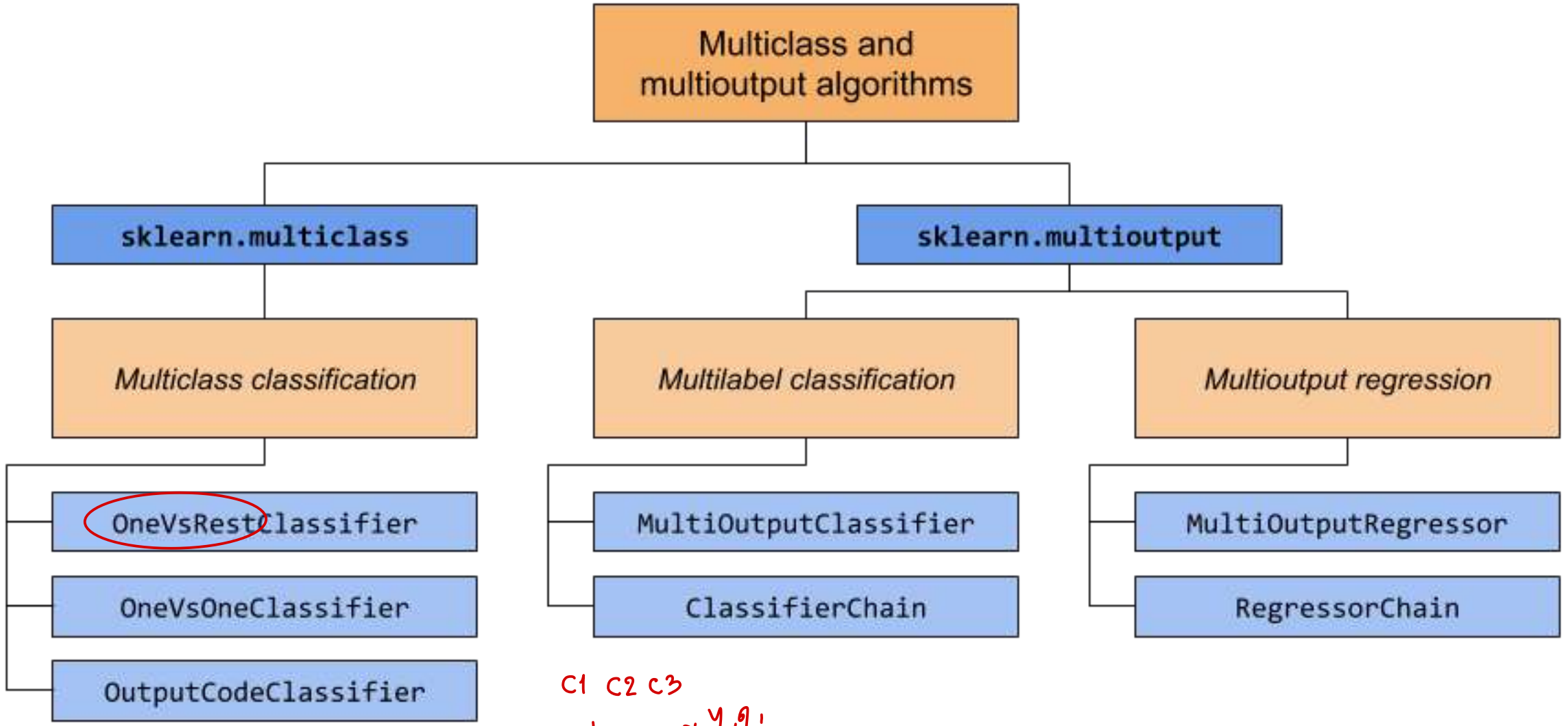
$$H = \{x \mid \langle w, x \rangle + b = 0\}$$

Dot product





- Good according to intuition, theory, practice
- SVM became famous when, using images as input, it gave accuracy comparable to neural-network with hand-designed features in a handwriting recognition task



C1 C2 C3

One Vs Rest: C1 vs 2, 3 C1

One Vs One: C1 vs C2, C1 vs C3, C2 vs C3

Multiclass

- **Inherently multiclass:**
  - `naive_bayes.BernoulliNB`
  - `tree.DecisionTreeClassifier`
  - `tree.ExtraTreeClassifier`
  - `ensemble.ExtraTreesClassifier`
  - `naive_bayes.GaussianNB`
  - `neighbors.KNeighborsClassifier`
  - `semi_supervised.LabelPropagation`
  - `semi_supervised.LabelSpreading`
  - `discriminant_analysis.LinearDiscriminantAnalysis`
  - `svm.LinearSVC` (setting multi_class="crammer_singer")
  - `linear_model.LogisticRegression` (setting multi_class="multinomial")
  - `linear_model.LogisticRegressionCV` (setting multi_class="multinomial")
  - `neural_network.MLPClassifier`
  - `neighbors.NearestCentroid`
  - `discriminant_analysis.QuadraticDiscriminantAnalysis`
  - `neighbors.RadiusNeighborsClassifier`
  - `ensemble.RandomForestClassifier`
  - `linear_model.RidgeClassifier`
  - `linear_model.RidgeClassifierCV`
- **Multiclass as One-Vs-One:**
  - `svm.NuSVC`
  - `svm.SVC`.
  - `gaussian_process.GaussianProcessClassifier` (setting multi_class = "one_vs_one")
- **Multiclass as One-Vs-The-Rest:**
  - `ensemble.GradientBoostingClassifier`
  - `gaussian_process.GaussianProcessClassifier` (setting multi_class = "one_vs_rest")
  - `svm.LinearSVC` (setting multi_class="ovr")
  - `linear_model.LogisticRegression` (setting multi_class="ovr")
  - `linear_model.LogisticRegressionCV` (setting multi_class="ovr")
  - `linear_model.SGDClassifier`
  - `linear_model.Perceptron`
  - `linear_model.PassiveAggressiveClassifier`
- **Support multilabel:**
  - `tree.DecisionTreeClassifier`
  - `tree.ExtraTreeClassifier`
  - `ensemble.ExtraTreesClassifier`
  - `neighbors.KNeighborsClassifier`
  - `neural_network.MLPClassifier`
  - `neighbors.RadiusNeighborsClassifier`
  - `ensemble.RandomForestClassifier`
  - `linear_model.RidgeClassifier`
  - `linear_model.RidgeClassifierCV`
- **Support multiclass-multioutput:**
  - `tree.DecisionTreeClassifier`
  - `tree.ExtraTreeClassifier`
  - `ensemble.ExtraTreesClassifier`
  - `neighbors.KNeighborsClassifier`
  - `neighbors.RadiusNeighborsClassifier`
  - `ensemble.RandomForestClassifier`

Multiclass Logistic Regression

sklearn.linear_model.LogisticRegression

class `sklearn.linear_model.`**`LogisticRegression`**(*penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None*) [source]

Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag', 'saga' and 'newton-cg' solvers.)

This class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers. **Note that regularization is applied by default**. It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).

multi_class : ***{'auto', 'ovr', 'multinomial'}, default='auto'***

If the option chosen is 'ovr', then a binary problem is fit for each label. For 'multinomial' the loss minimised is the multinomial loss fit across the entire probability distribution, *even when the data is binary*. 'multinomial' is unavailable when solver='liblinear'. 'auto' selects 'ovr' if the data is binary, or if solver='liblinear', and otherwise selects 'multinomial'.

New in version 0.18: Stochastic Average Gradient descent solver for 'multinomial' case.

Multiclass SVM

sklearn.svm.SVC

class `sklearn.svm.`**SVC**(**, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using `LinearSVC` or `SGDClassifier` instead, possibly after a `Nystroem` transformer or other Kernel Approximation.

The multiclass support is handled according to a one-vs-one scheme.

For details on the precise mathematical formulation of the provided kernel functions and how `gamma`, `coef0` and `degree` affect each other, see the corresponding section in the narrative documentation: Kernel functions.

Read more in the User Guide.

Wrapper

sklearn.multiclass.OneVsRestClassifier

class `sklearn.multiclass.`**`OneVsRestClassifier`**(*estimator*, ***, *n_jobs=None*, *verbose=0*) [source]

One-vs-the-rest (OvR) multiclass strategy.

Also known as one-vs-all, this strategy consists in fitting one classifier per class. For each classifier, the class is fitted against all the other classes. In addition to its computational efficiency (only `n_classes` classifiers are needed), one advantage of this approach is its interpretability. Since each class is represented by one and one classifier only, it is possible to gain knowledge about the class by inspecting its corresponding classifier. This is the most commonly used strategy for multiclass classification and is a fair default choice.

OneVsRestClassifier can also be used for multilabel classification. To use this feature, provide an indicator matrix for the target `y` when calling `.fit`. In other words, the target labels should be formatted as a 2D binary (0/1) matrix, where [i, j] == 1 indicates the presence of label j in sample i. This estimator uses the binary relevance method to perform multilabel classification, which involves training one binary classifier independently for each label.

Read more in the User Guide.

Parameters:

estimator : *estimator object*
A regressor or a classifier that implements fit. When a classifier is passed, decision_function will be used in priority and it will fallback to predict_proba if it is not available. When a regressor is passed, predict is used.

n_jobs : *int, default=None*
The number of jobs to use for the computation: the `n_classes` one-vs-rest problems are computed in parallel.

`None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See Glossary

See also:

`OneVsOneClassifier`
One-vs-one multiclass strategy.

`OutputCodeClassifier`
(Error-Correcting) Output-Code multiclass strategy.

`sklearn.multioutput.MultiOutputClassifier`
Alternate way of extending an estimator for multilabel classification.

`sklearn.preprocessing.MultiLabelBinarizer`
Transform iterable of iterables to binary indicator matrix.

Examples

```
>>> import numpy as np
>>> from sklearn.multiclass import OneVsRestClassifier
>>> from sklearn.svm import SVC
>>> X = np.array([
...     [10, 10],
...     [8, 10],
...     [-5, 5.5],
...     [-5.4, 5.5],
...     [-20, -20],
...     [-15, -20]
... ])
>>> y = np.array([0, 0, 1, 1, 2, 2])
>>> clf = OneVsRestClassifier(SVC()).fit(X, y)
>>> clf.predict([[-19, -20], [9, 9], [-5, 5]])
array([2, 0, 1])
```

+

Evaluation Measures

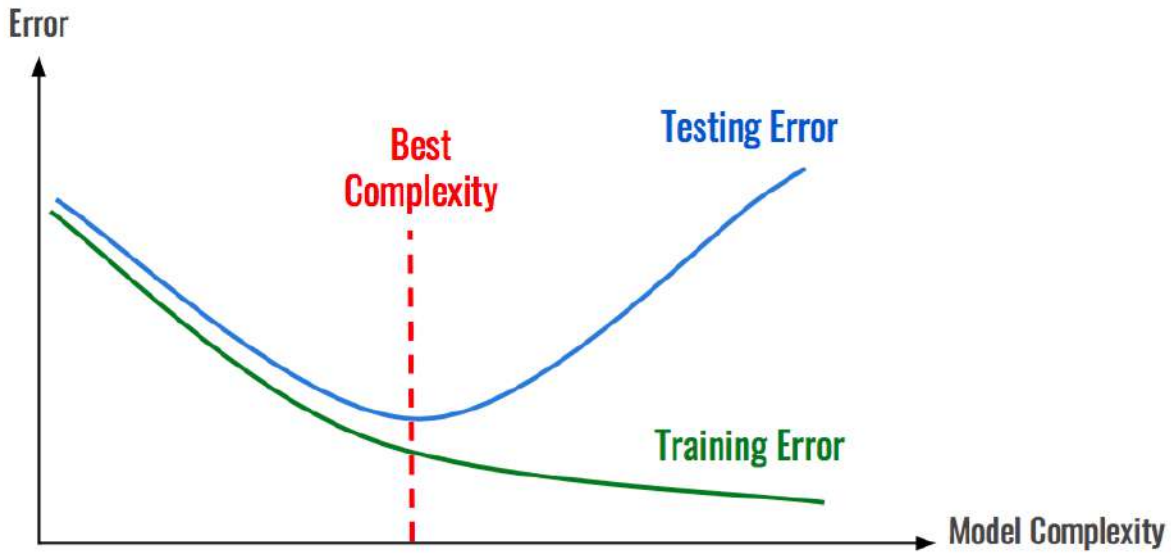
Evaluation (Train/Test Split)

Training Data

Age	Income	Purchase
25	25,000	Yes
35	50,000	Yes
32	35,000	No

Testing Data

Age	Income	Purchase
27	35,000	Yes
23	20,000	No
45	34,000	No



sklearn.model_selection.train_test_split

sklearn.model_selection.**train_test_split**(**arrays*, ***options*) [source]

Split arrays or matrices into random train and test subsets

Quick utility that wraps input validation and `next(ShuffleSplit().split(X, y))` and application to input data into a single call for splitting (and optionally subsampling) data in a oneliner.

Read more in the User Guide.

Parameters:

arrays : *sequence of indexables with same length / shape[0]
Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.

test_size : *float, int or None, optional (default=None)*
If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is set to the complement of the train size. If `train_size` is also None, it will be set to 0.25.

train_size : *float, int, or None, (default=None)*
If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the train split. If int, represents the absolute number of train samples. If None, the value is automatically set to the complement of the test size.

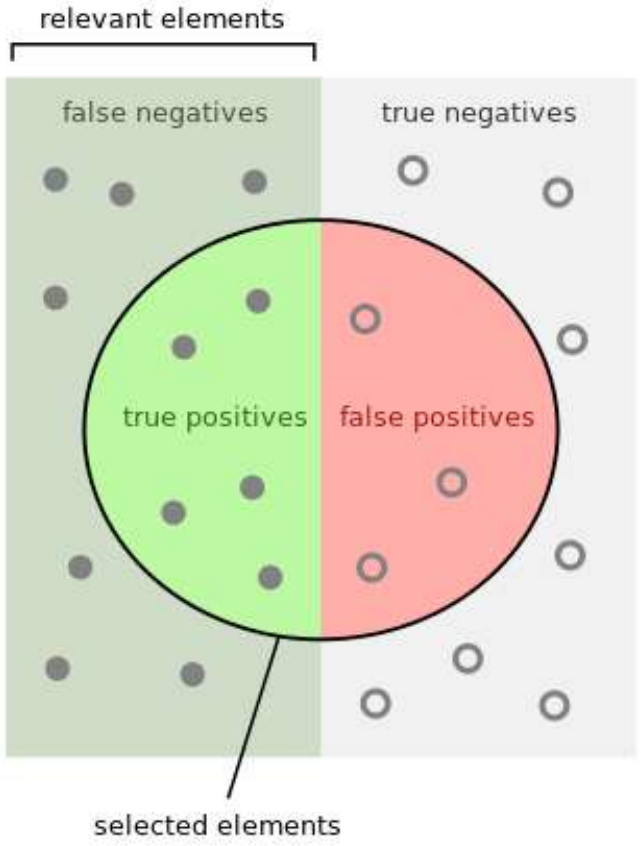
random_state : *int, RandomState instance or None, optional (default=None)*
If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by `np.random`.

Model Evaluation

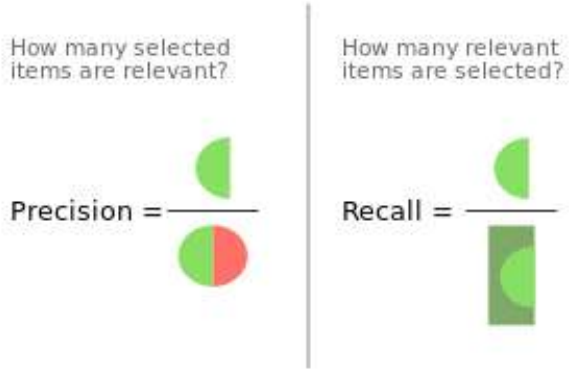
- Regression
  - Sum Squared Error (SSE)
  - Average Squared Error (ASE)
- Classification
  - **Accuracy**
  - Misclassification
  - Precision
  - Recall
  - F1
- Graph
  - ROC Curve
  - Area Under ROC (c-statistic)
  - Lift
  - Gain
  - Response

Precision, Recall, F1

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	



- Precision = correctly predict = TP / (TP + FP)
- Recall = coverage = TP / (TP + FN)
- F1 = (2*pre*rec) / (pre + rec)



3.5.1. The `scoring` parameter: defining model evaluation rules

Model selection and evaluation using tools, such as `grid_search.GridSearchCV` and `cross_validation.cross_val_score`, take a `scoring` parameter that controls what metric they apply to estimators evaluated.

3.5.1.1. Common cases: predefined values

For the most common usecases, you can simply provide a string as the `scoring` parameter. Possible values are:

Scoring	Function
Classification	
'accuracy'	`sklearn.metrics.accuracy_score`
'average_precision'	`sklearn.metrics.average_precision_score`
'f1'	`sklearn.metrics.f1_score`
'precision'	`sklearn.metrics.precision_score`
'recall'	`sklearn.metrics.recall_score`
'roc_auc'	`sklearn.metrics.roc_auc_score`
Clustering	
'adjusted_rand_score'	`sklearn.metrics.adjusted_rand_score`
Regression	
'mean_absolute_error'	`sklearn.metrics.mean_absolute_error`
'mean_squared_error'	`sklearn.metrics.mean_squared_error`
'r2'	`sklearn.metrics.r2_score`

Classification

Prev Up Next

scikit-learn 0.22.1
Other versions

Please cite us if you use the software.

sklearn.metrics.confusion_matrix
Examples using
sklearn.metrics.confusion_matri

sklearn.metrics.confusion_matrix

sklearn.metrics.**confusion_matrix**(*y_true*, *y_pred*, *labels=None*, *sample_weight=None*, *normalize=None*) [source]

Compute confusion matrix to evaluate the accuracy of a classification.

By definition a confusion matrix C is such that $C_{i,j}$ is equal to the number of observations known to be in group i and predicted to be in group j.

Thus in binary classification, the count of true negatives is $C_{0,0}$, false negatives is $C_{1,0}$, true positives is $C_{1,1}$ and false positives is $C_{0,1}$.

Read more in the User Guide.

Parameters:

y_true : *array-like of shape (n_samples,)*
Ground truth (correct) target values.

f shape (n_samples,)
s returned by a classifier.

shape (n_classes), default=None
ex the matrix. This may be used to reorder or select a subset of labels. If `None` is given, those once in `y_true` or `y_pred` are used in sorted order.

y-like of shape (n_samples,), default=None

normalize : *{'true', 'pred', 'all'}, default=None*
Normalizes confusion matrix over the true (rows), predicted (columns) conditions or all the population. If None, confusion matrix will not be normalized.

```
>>> from sklearn.metrics import confusion_matrix
>>> y_true = [2, 0, 2, 2, 0, 1]
>>> y_pred = [0, 0, 2, 2, 0, 2]
>>> confusion_matrix(y_true, y_pred)
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

Classification

scikit learn | Install | User Guide | API | Examples | More

Prev | Up | Next

scikit-learn 0.22.1
Other versions

Please cite us if you use the software.

sklearn.metrics.classification_report
Examples using sklearn.metrics.classification_

sklearn.metrics.classification_report

sklearn.metrics.**classification_report**(*y_true*, *y_pred*, *labels=None*, *target_names=None*, *sample_weight=None*, *digits=2*, *output_dict=False*, *zero_division='warn'*) [source]

Build a text report showing the main

Read more in the User Guide.

Parameters:

y_true : *1d array*
Ground truth (c

y_pred : *1d array*
Estimated targe

labels : *array, sh*
Optional list of

target_names : *l*
Optional displa

sample_weight :
Sample weight

digits : *int*
Number of digi
the returned va

Examples

```
>>> from sklearn.metrics import classification_report
>>> y_true = [0, 1, 2, 2, 2]
>>> y_pred = [0, 0, 2, 2, 1]
>>> target_names = ['class 0', 'class 1', 'class 2']
>>> print(classification_report(y_true, y_pred, target_names=target_names))
              precision    recall  f1-score   support
<BLANKLINE>
     class 0       0.50      1.00      0.67         1
     class 1       0.00      0.00      0.00         1
     class 2       1.00      0.67      0.80         3
<BLANKLINE>
    accuracy                           0.60         5
   macro avg       0.50      0.56      0.49         5
weighted avg       0.70      0.60      0.61         5
<BLANKLINE>
>>> y_pred = [1, 1, 0]
>>> y_true = [1, 1, 1]
>>> print(classification_report(y_true, y_pred, labels=[1, 2, 3]))
              precision    recall  f1-score   support
<BLANKLINE>
           1       1.00      0.67      0.80         3
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
<BLANKLINE>
   micro avg       1.00      0.67      0.80         3
   macro avg       0.33      0.22      0.27         3
weighted avg       1.00      0.67      0.80         3
<BLANKLINE>
```

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report

Regression

sklearn.metrics.mean_absolute_error

sklearn.metrics.**mean_absolute_error**(*y_true*, *y_pred*, *sample_weight=None*) ¶

Mean absolute error regression loss

Parameters:

y_true : array-like of shape = [n_samples] or [n_samples, n_outputs]

Ground truth (correct) target values.

y_pred : array-like of shape = [n_samples] or [n_samples, n_outputs]

Estimated target values.

sample_weight : array-like of

Sample weights.

Returns:

loss : float

A positive floating point

Examples

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2, 7]
>>> y_pred = [2.5, 0.0, 2, 8]
>>> mean_absolute_error(y_true, y_pred)
0.5
>>> y_true = [[0.5, 1], [-1, 1], [7, -6]]
>>> y_pred = [[0, 2], [-1, 2], [8, -5]]
>>> mean_absolute_error(y_true, y_pred)
0.75
```

Regression

sklearn.metrics.mean_squared_error

```
sklearn.metrics.mean_squared_error(y_true, y_pred, sample_weight=None)
```

Mean squared error regression loss

Parameters:

y_true : array-like of shape = [n_samples] or [n_samples, n_outputs]

Ground truth (correct) target values.

y_pred : array-like of shape = [n_samples] or [n_samples, n_outputs]

Estimated target values.

sample_weight : array-like of shape =

Sample weights.

Returns:

loss : float

A positive floating point value (th

Examples

```
>>> from sklearn.metrics import mean_squared_error
>>> y_true = [3, -0.5, 2, 7]
>>> y_pred = [2.5, 0.0, 2, 8]
>>> mean_squared_error(y_true, y_pred)
0.375
>>> y_true = [[0.5, 1],[-1, 1],[7, -6]]
>>> y_pred = [[0, 2],[-1, 2],[8, -5]]
>>> mean_squared_error(y_true, y_pred)
0.708...
```

`sklearn.metrics.r2_score`

`sklearn.metrics.`**`r2_score`**(*y_true*, *y_pred*, *sample_weight=None*)

R^2 (coefficient of determination) regression score function.

Best possible score is 1.0, lower values are worse.

Parameters:

y_true : array-like of shape = [n_samples] or [n_samples, n_outputs]

Ground truth (correct) target values.

y_pred : array-like of shape = [n_samples] or [n_samples, n_outputs]

Estimated target values.

sample_weight : array-like of sh

Sample weights.

Returns:

z : float

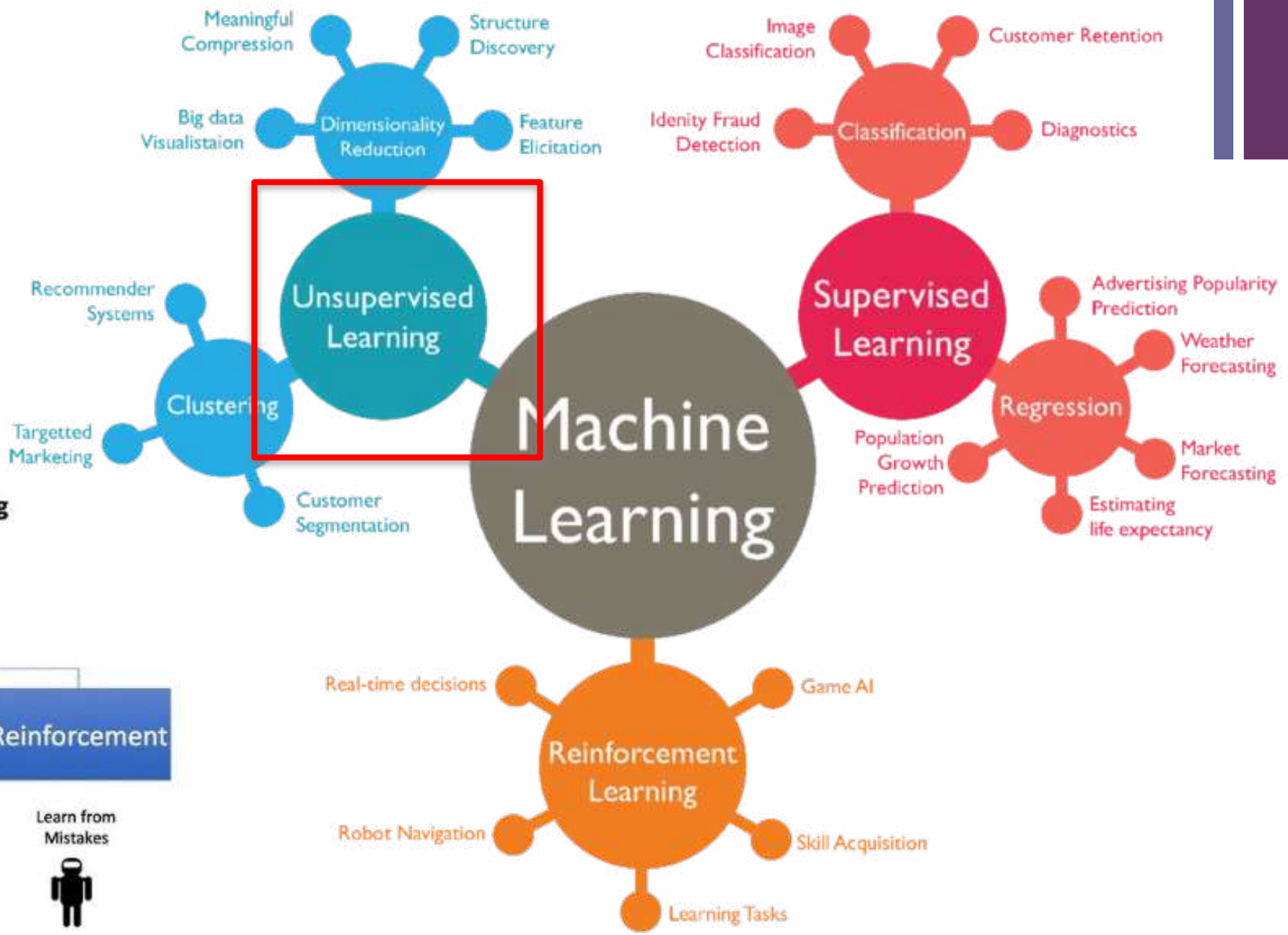
The R^2 score.

Examples

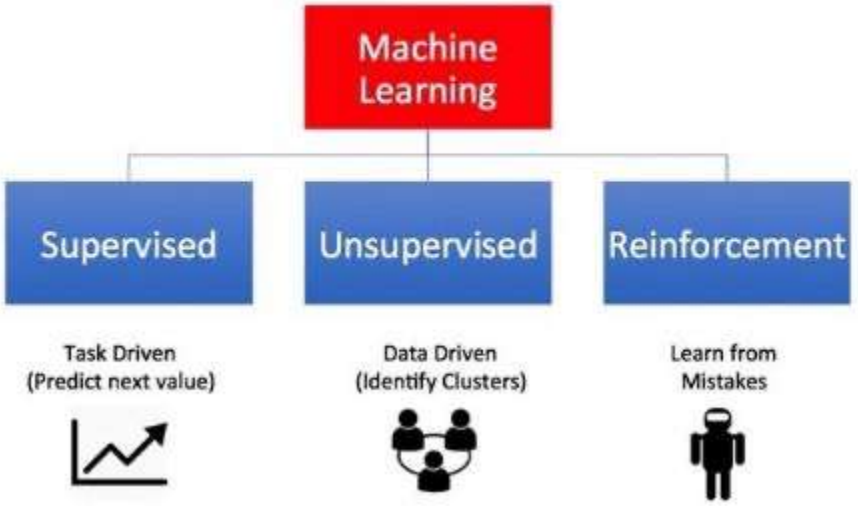
```
>>> from sklearn.metrics import r2_score
>>> y_true = [3, -0.5, 2, 7]
>>> y_pred = [2.5, 0.0, 2, 8]
>>> r2_score(y_true, y_pred)
0.948...
>>> y_true = [[0.5, 1], [-1, 1], [7, -6]]
>>> y_pred = [[0, 2], [-1, 2], [8, -5]]
>>> r2_score(y_true, y_pred)
0.938...
```

Part3: Unsupervised Learning (Descriptive Task)

Machine Learning (cont.)



Types of Machine Learning



Task2: Unsupervised learning (descriptive task)

Training Data

inputs

target

Age	Income	Gender	Province	Purchase
25	25,000	Female	Bangkok	Yes
35	50,000	Female	Nontaburi	Yes
32	35,000	Male	Bangkok	

Testing Data

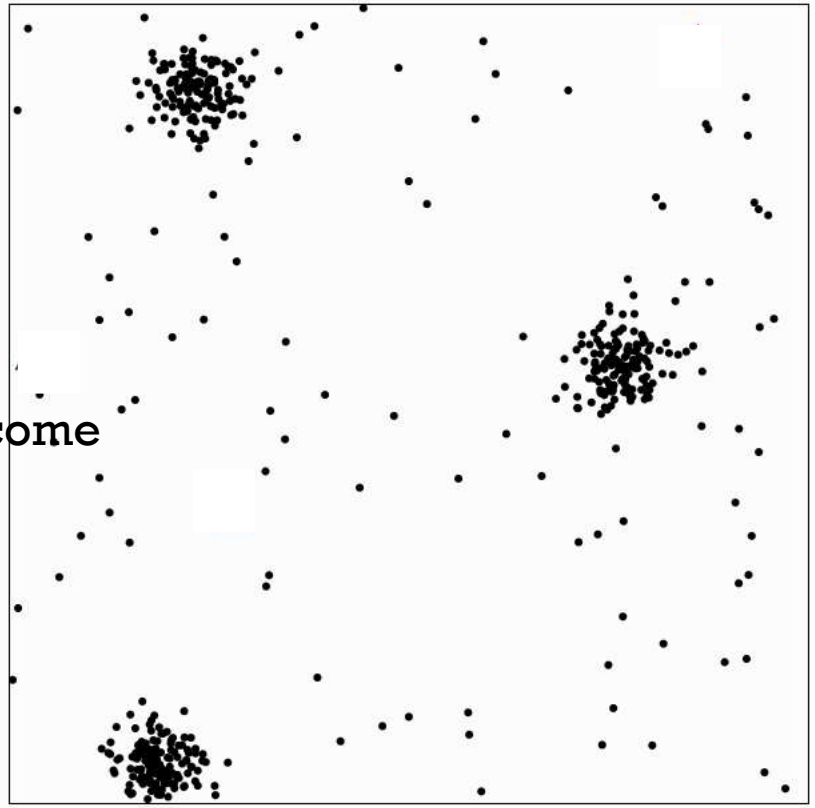
Age	Income		Province	Purchase
25	25,000		Bangkok	?

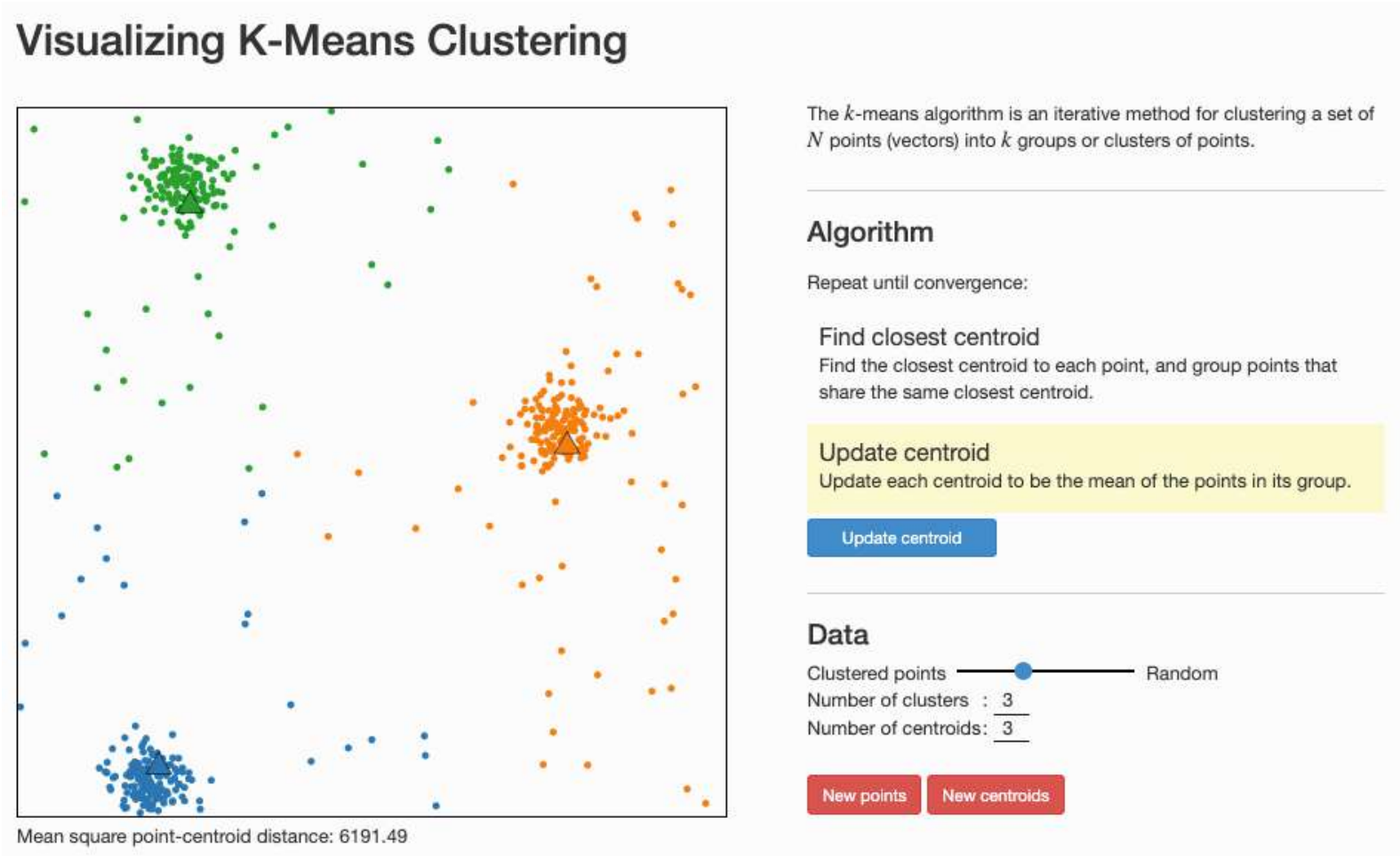
Clustering

- In our class, there are many participants. Should we teach them using the same method?
- May be not! Since they may have different learning behaviors and backgrounds.
- Inputs
  - Education field
  - Level of English communication
  - Level of computer skills
  - Age range
  - Gender

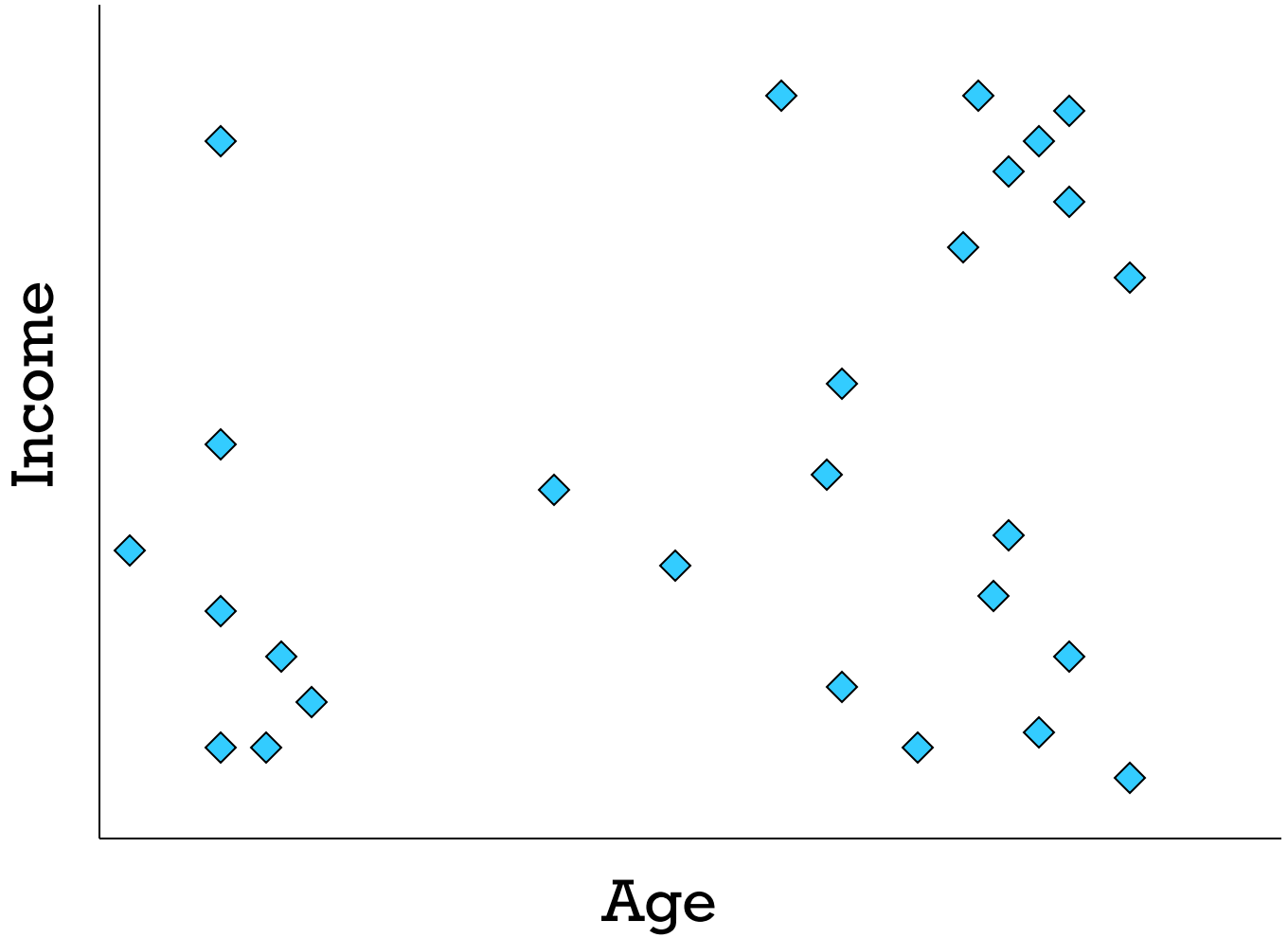
K-means Clustering

- http://web.stanford.edu/class/ee103/visualizations/kmeans/kmeans.html



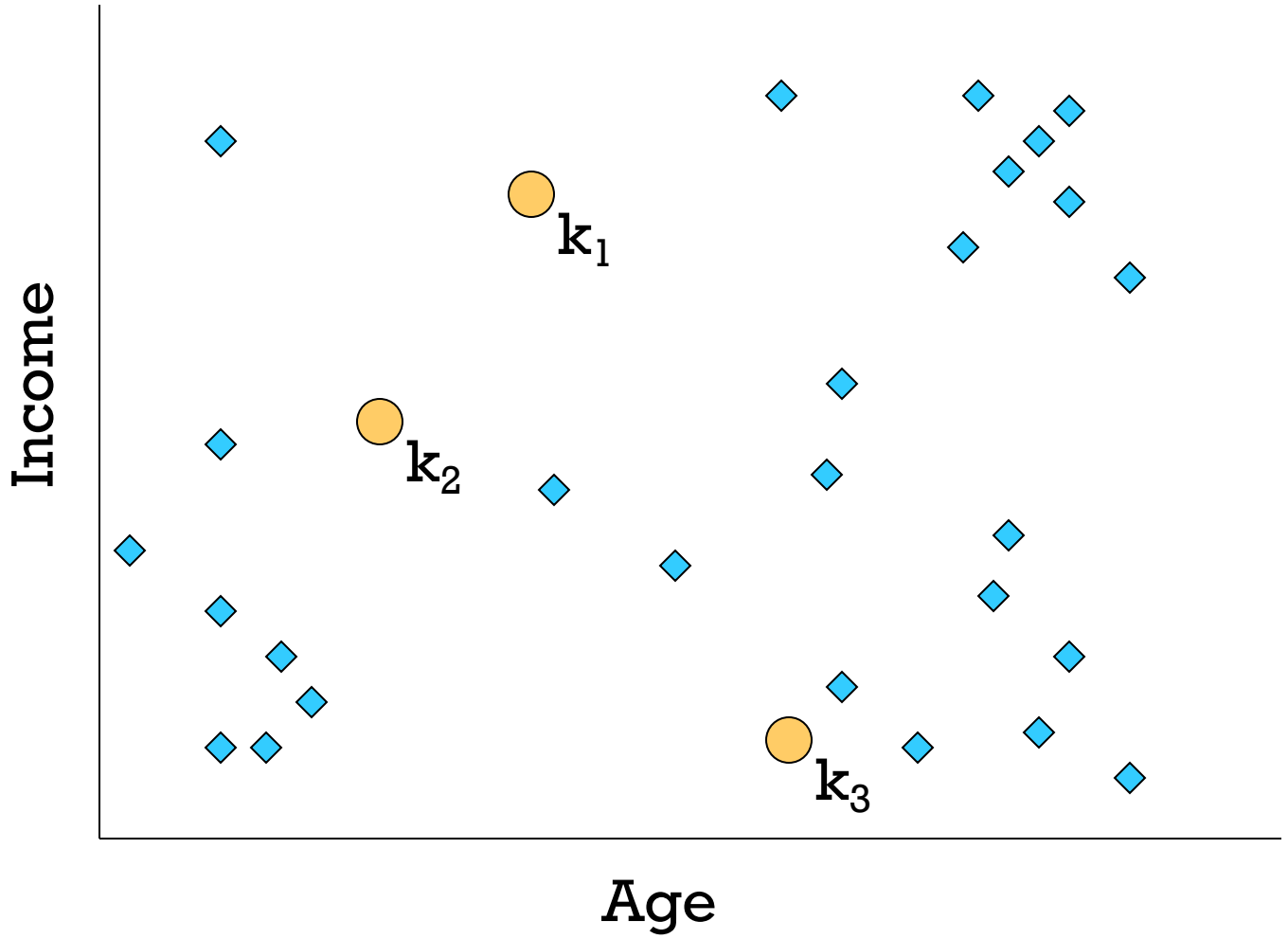


K-means: Step0



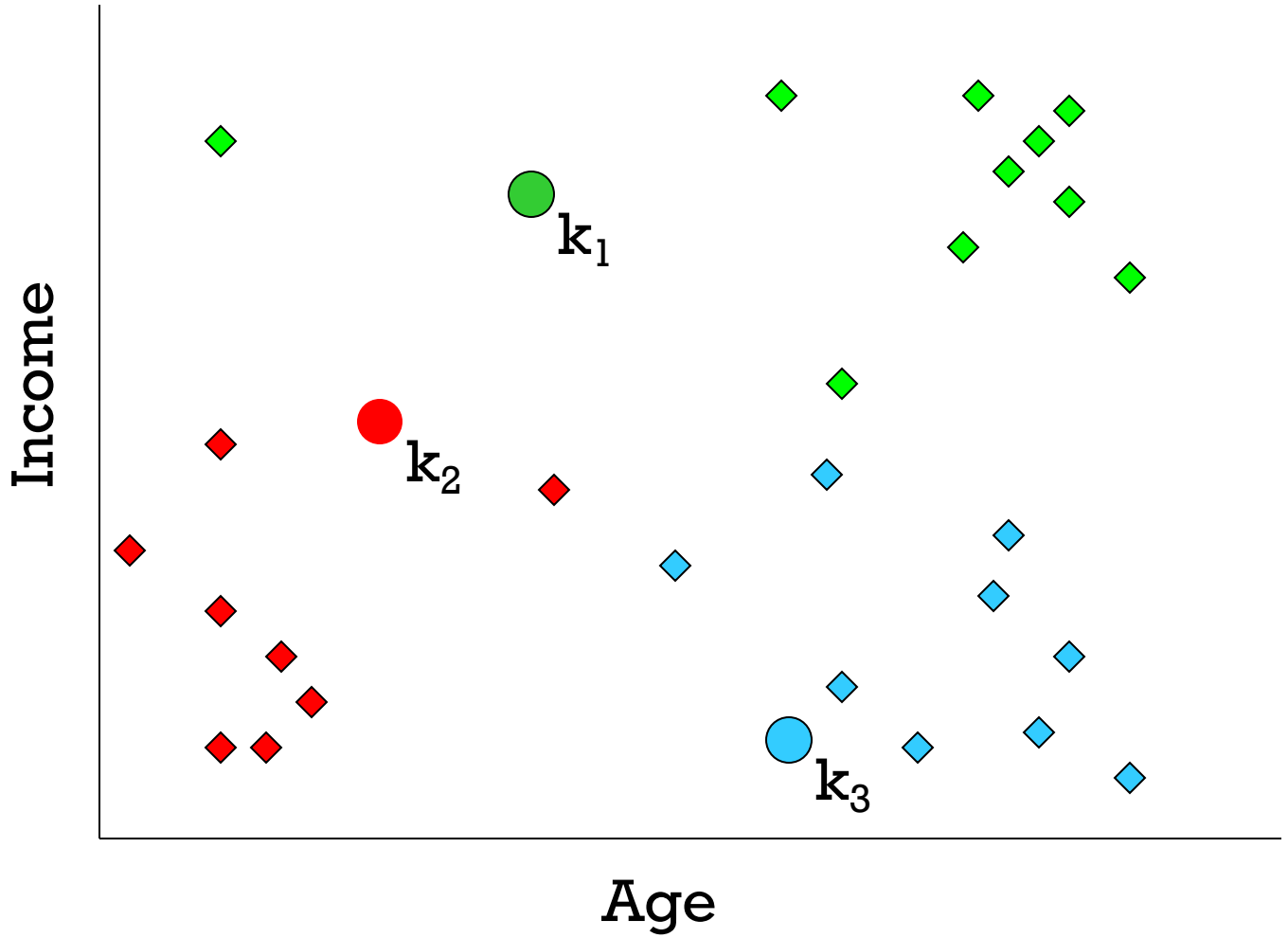
K-means: Step 1

Pick 3 initial cluster centers (randomly)



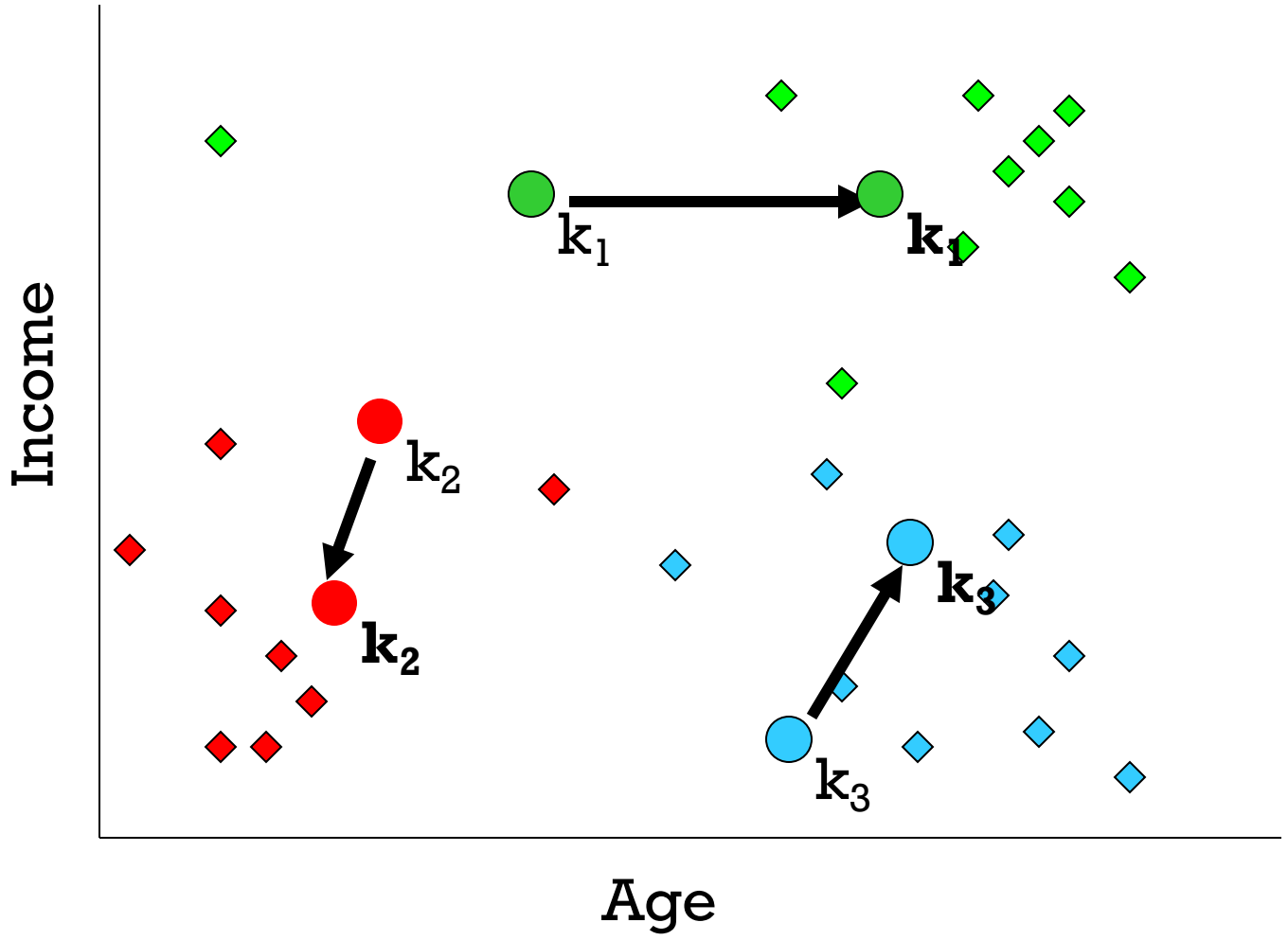
K-means: Step2

Assign each point to the closest cluster center



K-means: Step3

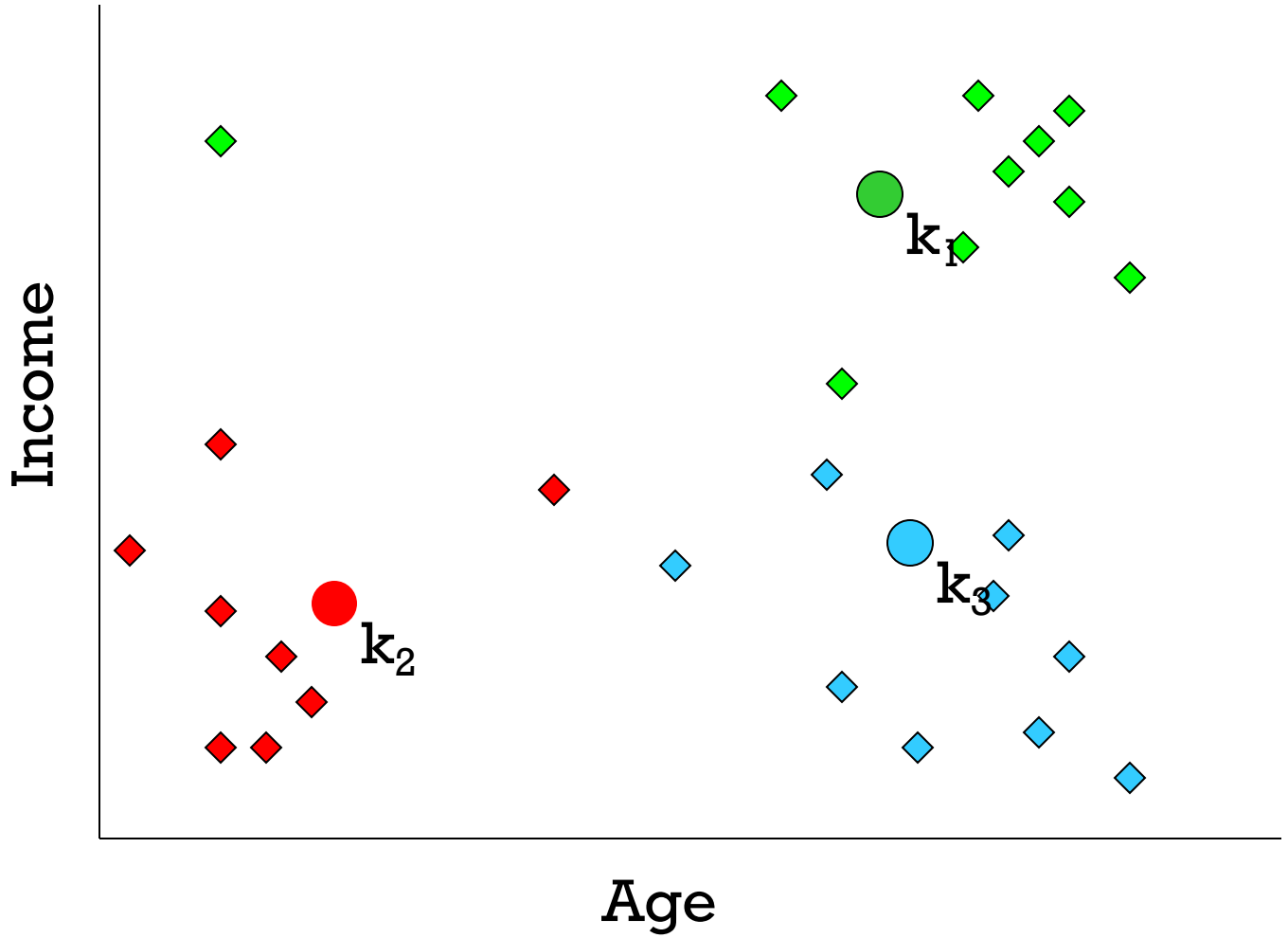
Move
each cluster
center
to the mean
of each cluster



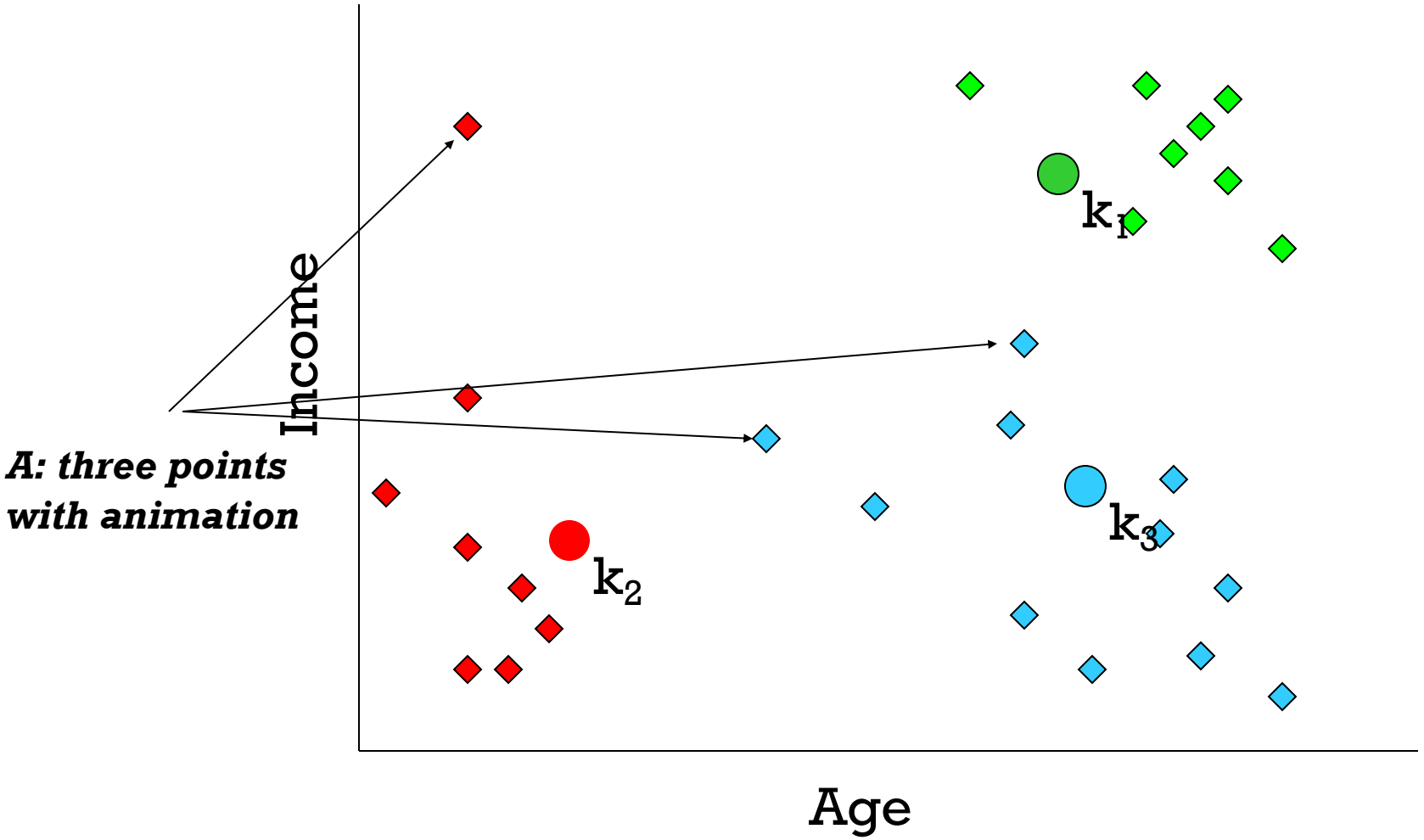
K-means: Step4

Reassign points closest to a different new cluster center

Q: Which points are reassigned?

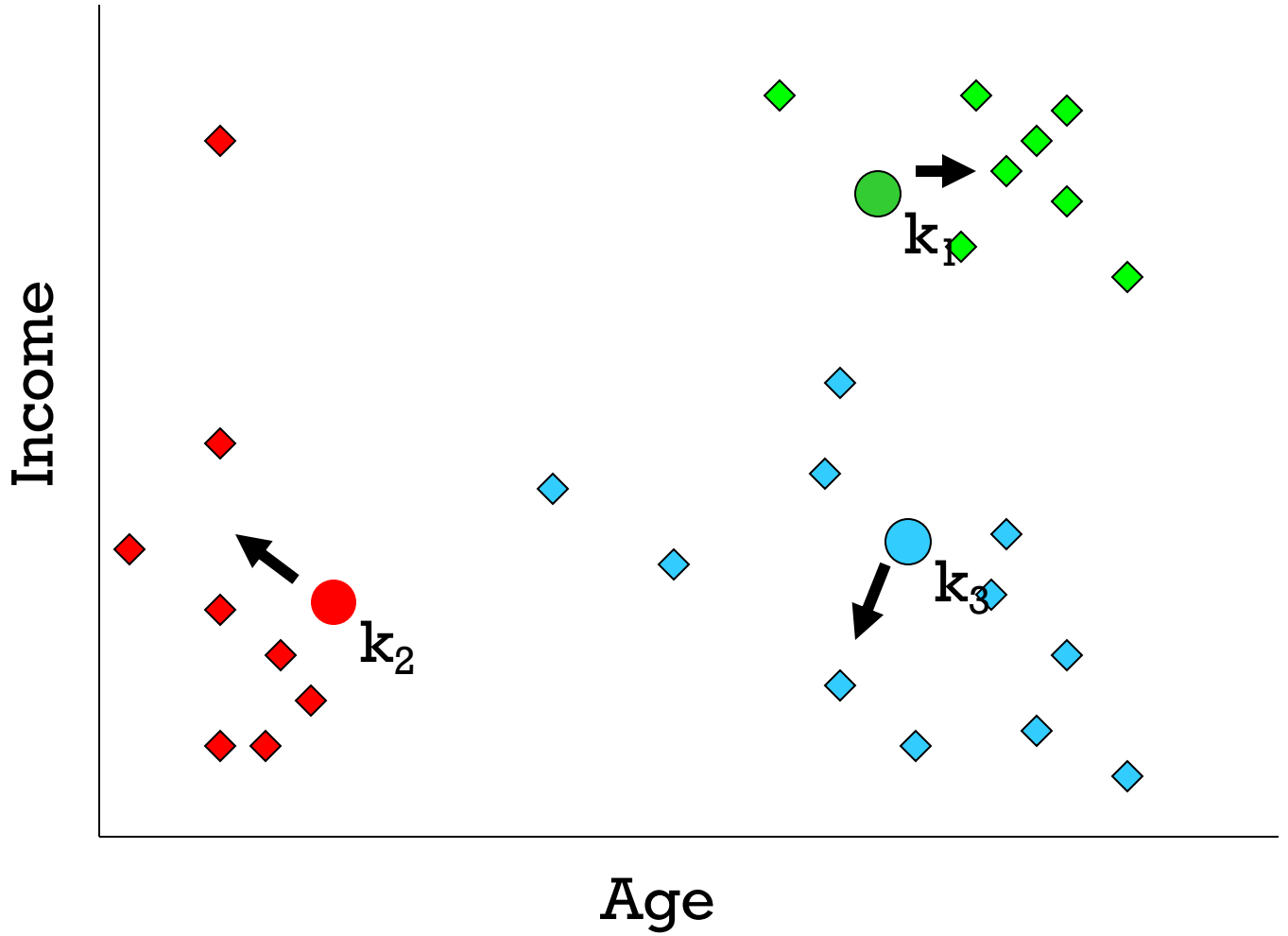


K-means: Step4(a)



K-means: Step5

re-compute
cluster means

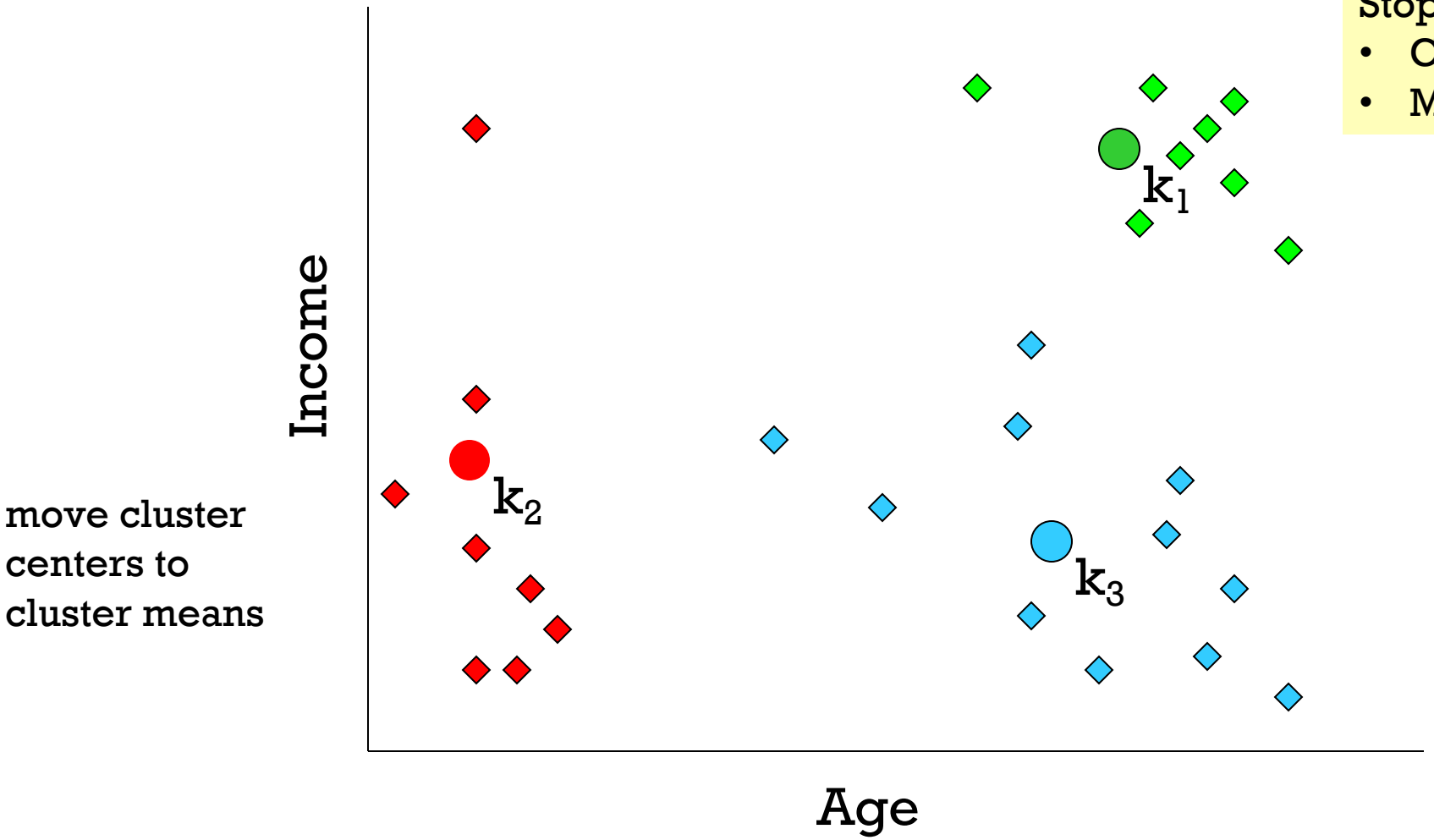


K-means: Step5(a)

Cautions:
- Support only numerical variables
- Need to adjust variable range

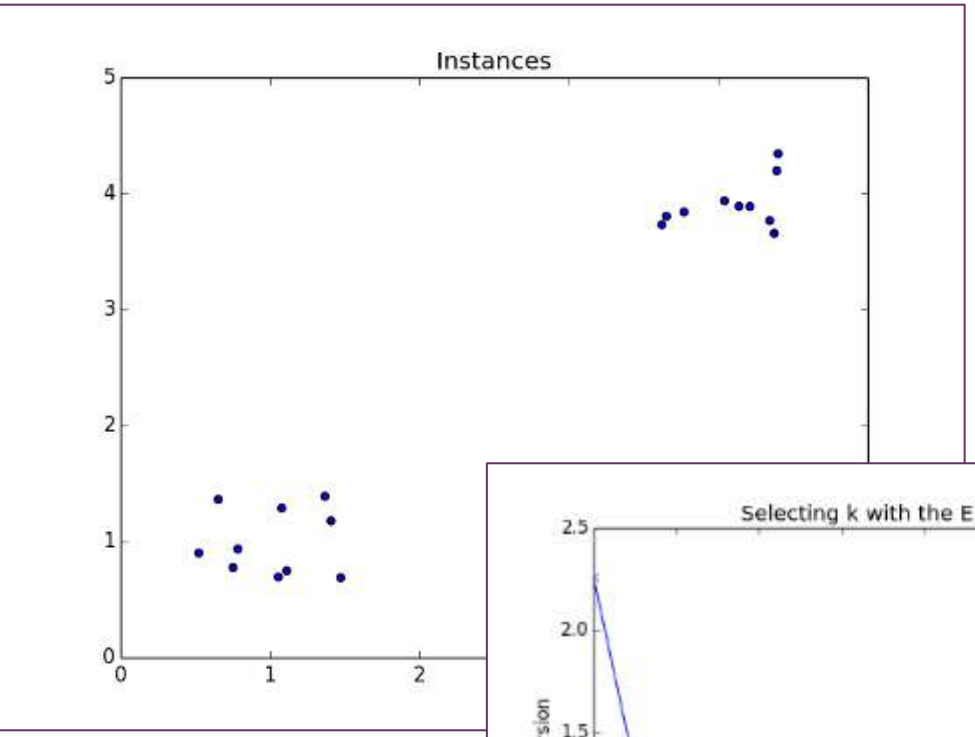
Important Params:
- k, Distance function
- Maximum epochs

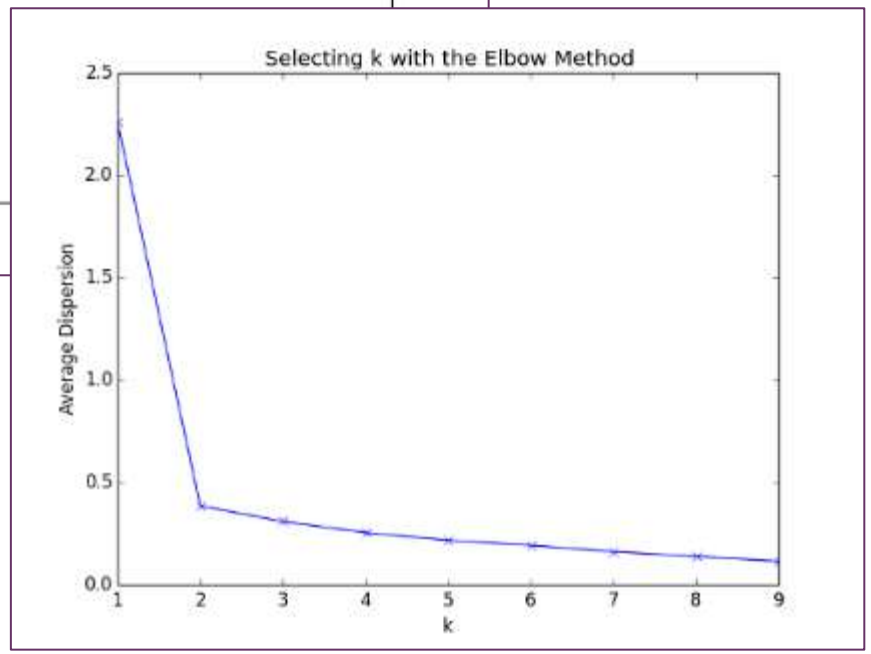
Stop
- Converge (no change)
- Maximum epochs

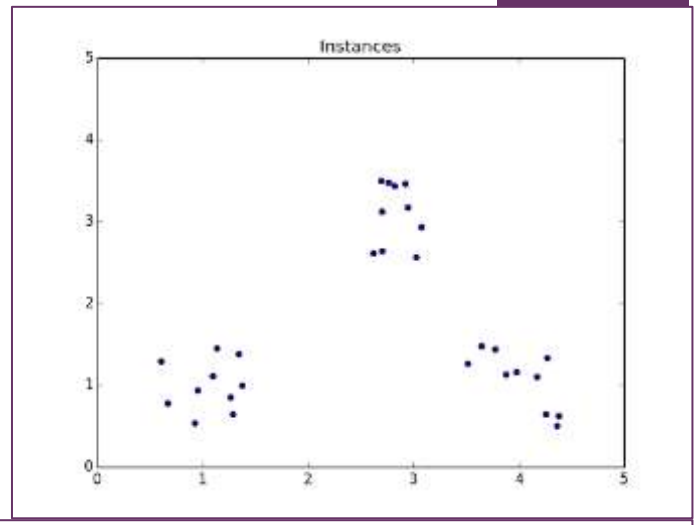


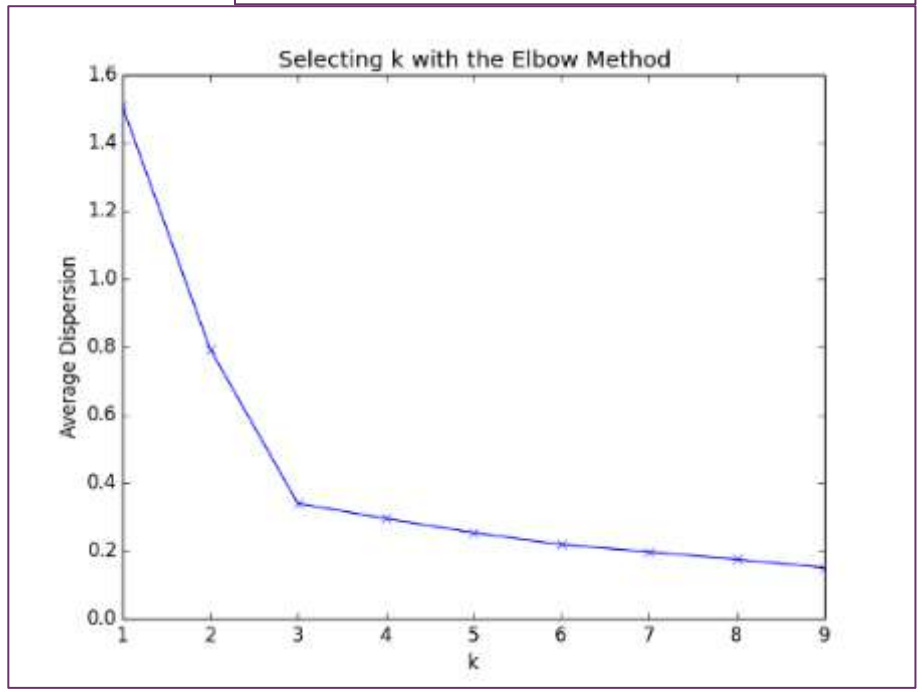
Determine the number of *k*: Elbow Method

https://www.packtpub.com/books/content/clustering-k-means









sklearn.preprocessing.StandardScaler

class sklearn.preprocessing.**StandardScaler**(*copy=True*, *with_mean=True*, *with_std=True*) [source]

Standardize features by removing the mean and scaling to unit variance

The standard score of a sample `x` is calculated as:

z = (x - u) / s

where `u` is the mean of the training samples or zero if `with_mean=False`, and `s` is the standard deviation of the training samples or one if `with_std=False`.

Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using `transform`.

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data (e.g. Gaussian with 0 mean and unit variance).

For instance many elements used in the objective function of a learning algorithm (such as the RBF kernel of Support Vector Machines or the L1 and L2 regularizers of linear models) assume that all features are centered around 0 and have variance in the same order. If a feature has a variance that is orders of magnitude larger that others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

This scaler can also be applied to sparse CSR or CSC matrices by passing `with_mean=False` to avoid breaking the sparsity structure of the data.

Clustering - K-means

```
>>> from sklearn.cluster import KMeans
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [10, 2], [10, 4], [10, 0]])
>>> kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
>>> kmeans.labels_
array([1, 1, 1, 0, 0, 0], dtype=int32)
>>> kmeans.predict([[0, 0], [12, 3]])
array([1, 0], dtype=int32)
>>> kmeans.cluster_centers_
array([[10.,  2.],
       [ 1.,  2.]])
```

sklearn.cluster.KMeans

class `sklearn.cluster.KMeans`(*n_clusters=8, *, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='deprecated', verbose=0, random_state=None, copy_x=True, n_jobs='deprecated', algorithm='auto'*) [source]

K-Means clustering.

Read more in the User Guide.

Parameters:

n_clusters : *int, default=8*
The number of clusters to form as well as the number of centroids to generate.

init : *{'k-means++', 'random', ndarray, callable}, default='k-means++'*
Method for initialization:

'k-means++' : selects initial cluster centers for k-mean clustering in a smart way to speed up convergence. See section Notes in k_init for more details.

'random': choose `n_clusters` observations (rows) at random from data for the initial centroids.

If an ndarray is passed, it should be of shape (n_clusters, n_features) and gives the initial centers.

If a callable is passed, it should take arguments X, n_clusters and a random state and return an initialization.

n_init : *int, default=10*
Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.

max_iter : *int, default=300*
Maximum number of iterations of the k-means algorithm for a single run.

Reference : https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

sklearn.metrics.silhouette_score

`sklearn.metrics.silhouette_score(X, labels, *, metric='euclidean', sample_size=None, random_state=None, **kwds)` [source]

Compute the mean Silhouette Coefficient of all samples.

The Silhouette Coefficient is calculated using the mean intra-cluster distance (`a`) and the mean nearest-cluster distance (`b`) for each sample. The Silhouette Coefficient for a sample is `(b - a) / max(a, b)`. To clarify, `b` is the distance between a sample and the nearest cluster that the sample is not a part of. Note that Silhouette Coefficient is only defined if number of labels is `2 <= n_labels <= n_samples - 1`.

This function returns the mean Silhouette Coefficient over all samples. To obtain the values for each sample, use `silhouette_samples`.

The best value is 1 and the worst value is -1. Values near 0 indicate overlapping clusters. Negative values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar.

Read more in the User Guide.

Parameters::

X : *array-like of shape (n_samples_a, n_samples_a) if metric == "precomputed" or (n_samples_a, n_features) otherwise*
An array of pairwise distances between samples, or a feature array.

labels : *array-like of shape (n_samples,)*
Predicted labels for each sample.

metric : *str or callable, default='euclidean'*
The metric to use when calculating distance between instances in a feature array. If metric is a string, it must be one of the options allowed by `metrics.pairwise.pairwise_distances`. If `X` is the distance array itself, use `metric="precomputed"`.

sample_size : *int, default=None*
The size of the sample to use when computing the Silhouette Coefficient on a random subset of the data. If `sample_size is None`, no sampling is used.

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html

Clustering - DBSCAN

K-mean : Distance base

- Density based algorithm
- **D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise
- It groups points that are closely packed together
- Then expanding clusters in any direction where there are nearby points
- this algorithm does not requires the number of clusters to be specified
- Can identify outliers as noises
- Doesn't perform well when the clusters are of varying density
- Can form any shape of cluster
- Expensive computation to high-dimensional data

Clustering - DBSCAN

Step

1. Initial starting point that has not been visited

2. All points which are within the r distance (eps) are neighborhood points

3. If (number of neighborhood points > minPoints)
   - Assign all neighborhood points to same cluster
   - Repeat step 2 – 3 on all neighborhood points
   - marked all of these point as "visited"

Else
   - the point will be labeled as noise
   - marked all of these point as "visited"

4. Repeat step 1-3 until all points are visited

Clustering - DBSCAN

Step

minPoints = 3

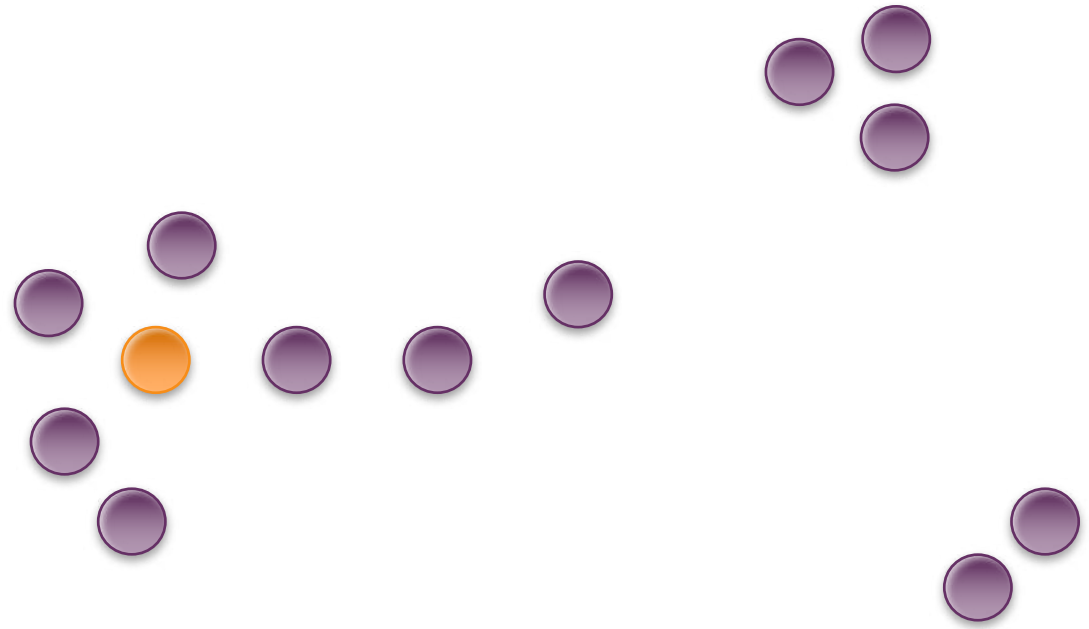
1. Initial starting point that has not been visited

2. All points which are within the r distance are neighborhood points

3. If (number of neighborhood points > minPoints)
   - Assign all neighborhood points to same cluster
   - Repeat step 2 – 3 on all neighborhood points
   - marked all of these point as "visited"

Else
   - the point will be labeled as noise
   - marked all of these point as "visited"

4. Repeat step 1-3 until all points are visited



Clustering - DBSCAN

Step

minPoints = 3

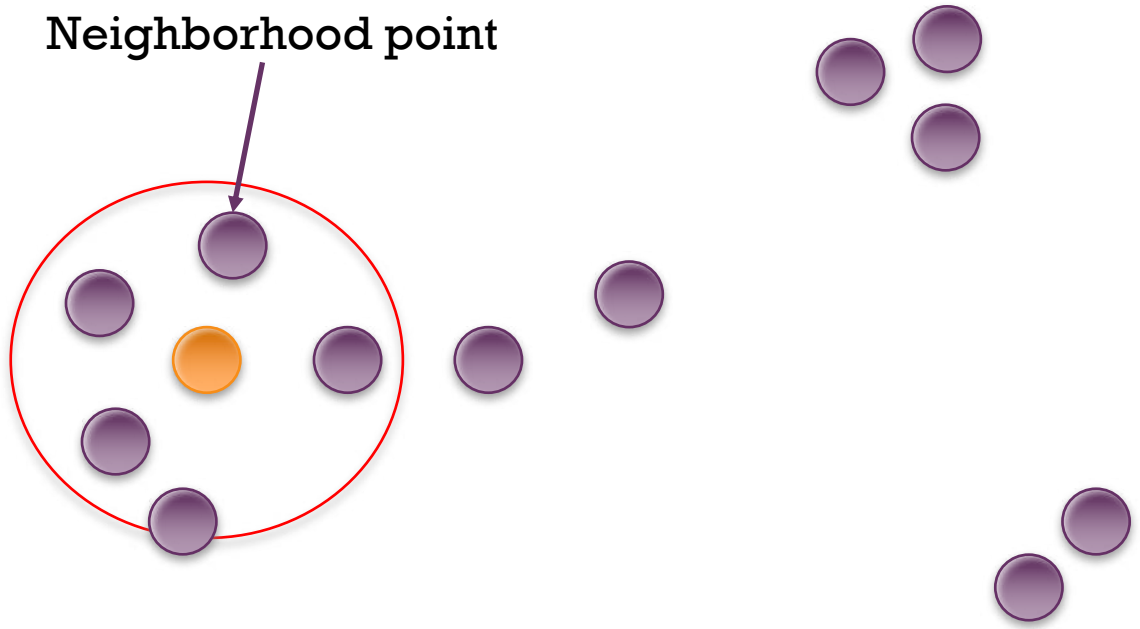
1. Initial starting point that has not been visited

2. All points which are within the r distance are neighborhood points

3. If (number of neighborhood points > minPoints)
   - Assign all neighborhood points to same cluster
   - Repeat step 2 – 3 on all neighborhood points
   - marked all of these point as "visited"

Else
   - the point will be labeled as noise
   - marked all of these point as "visited"

4. Repeat step 1-3 until all points are visited



Clustering - DBSCAN

Step

minPoints = 3

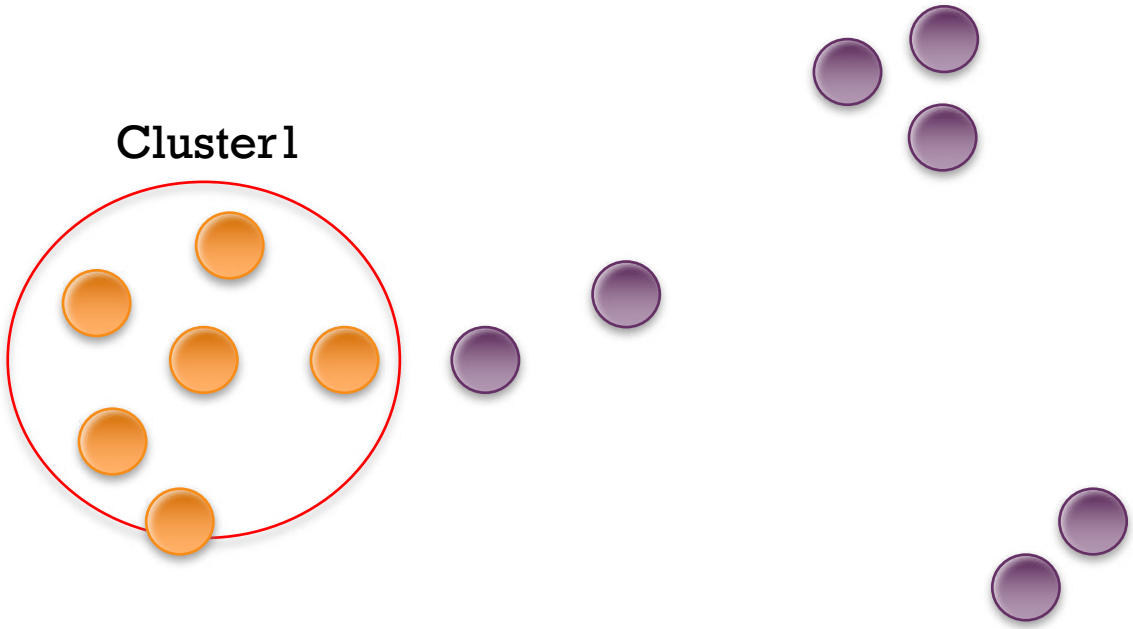
1. Initial starting point that has not been visited

2. All points which are within the r distance are neighborhood points

3. If (number of neighborhood points > minPoints)
   - Assign all neighborhood points to same cluster
   - Repeat step 2 – 3 on all neighborhood points
   - marked all of these point as "visited"

Else
   - the point will be labeled as noise
   - marked all of these point as "visited"

4. Repeat step 1-3 until all points are visited



Clustering - DBSCAN

Step

1. Initial starting point that has not been visited

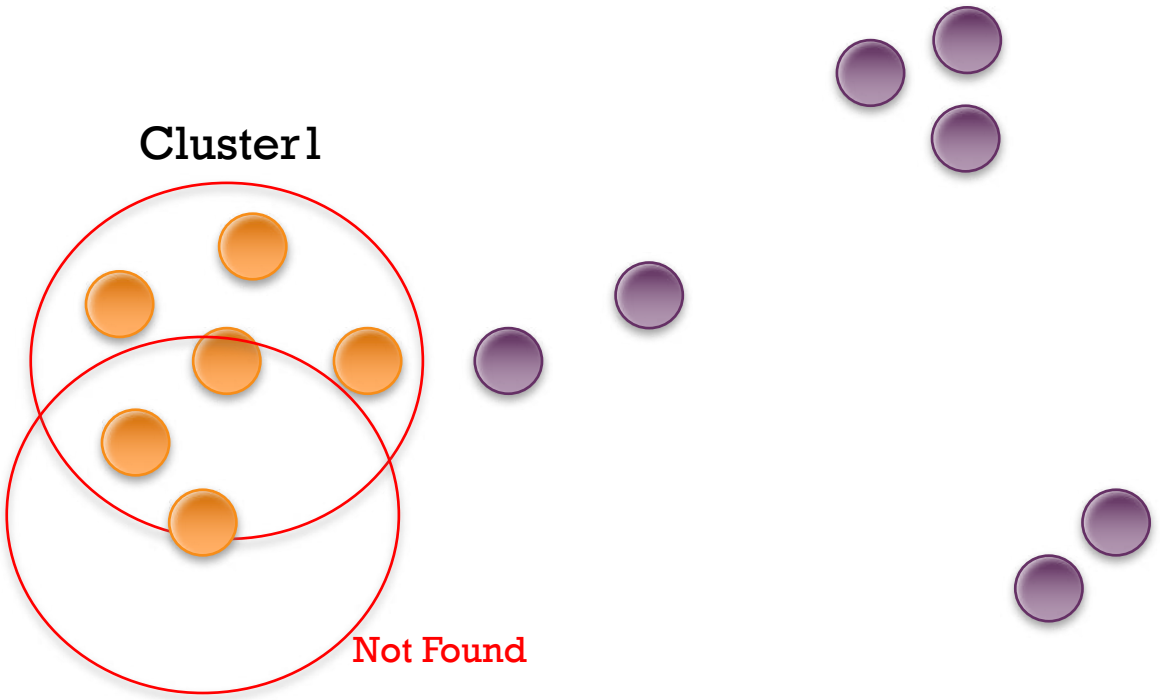
2. All points which are within the r distance are neighborhood points

3. If (number of neighborhood points > minPoints)
   - Assign all neighborhood points to same cluster
   - Repeat step 2 – 3 on all neighborhood points
   - marked all of these point as "visited"

Else
   - the point will be labeled as noise
   - marked all of these point as "visited"

4. Repeat step 1-3 until all points are visited

minPoints = 3



Clustering - DBSCAN

Step

minPoints = 3

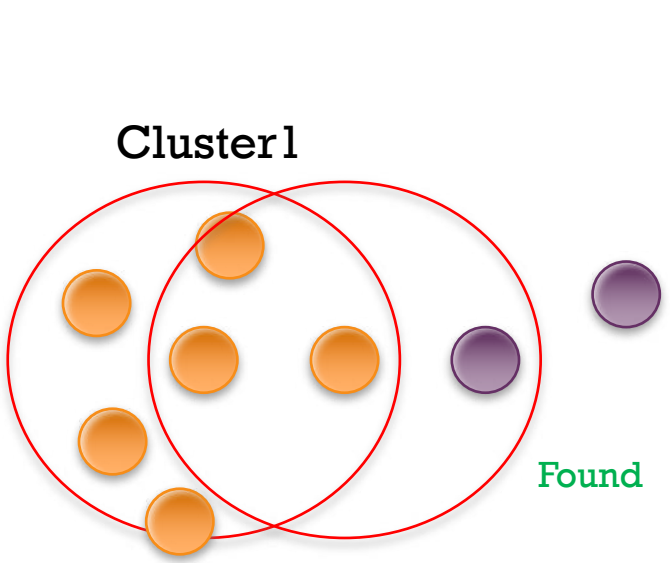
1. Initial starting point that has not been visited

2. All points which are within the r distance are neighborhood points

3. If (number of neighborhood points > minPoints)
   - Assign all neighborhood points to same cluster
   - Repeat step 2 – 3 on all neighborhood points
   - marked all of these point as "visited"

Else
   - the point will be labeled as noise
   - marked all of these point as "visited"

4. Repeat step 1-3 until all points are visited



Clustering - DBSCAN

Step

minPoints = 3

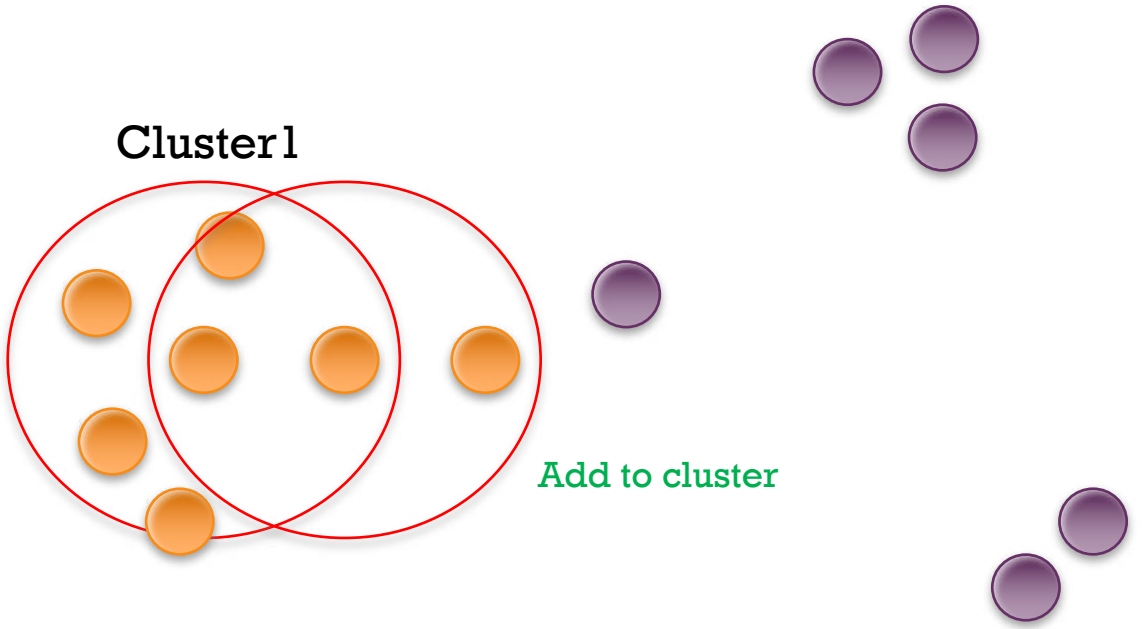
1. Initial starting point that has not been visited

2. All points which are within the r distance are neighborhood points

3. If (number of neighborhood points > minPoints)
   - Assign all neighborhood points to same cluster
   - Repeat step 2 – 3 on all neighborhood points
   - marked all of these point as "visited"

Else
   - the point will be labeled as noise
   - marked all of these point as "visited"

4. Repeat step 1-3 until all points are visited



Clustering - DBSCAN

Step

minPoints = 3

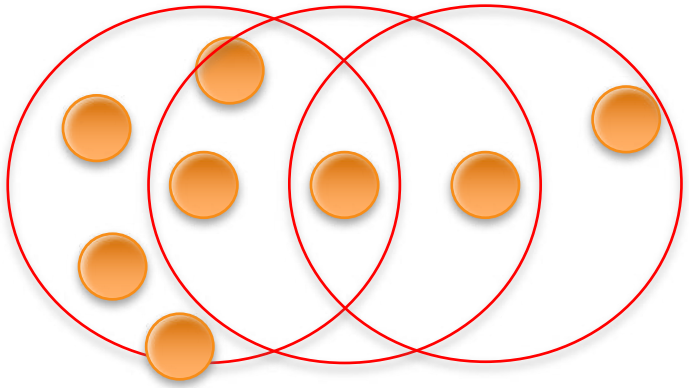
1. Initial starting point that has not been visited

2. All points which are within the r distance are neighborhood points

3. If (number of neighborhood points > minPoints)
   - Assign all neighborhood points to same cluster
   - Repeat step 2 – 3 on all neighborhood points
   - marked all of these point as "visited"

Else
   - the point will be labeled as noise
   - marked all of these point as "visited"

4. Repeat step 1-3 until all points are visited



Clustering - DBSCAN

Step

1. Initial starting point that has not been visited

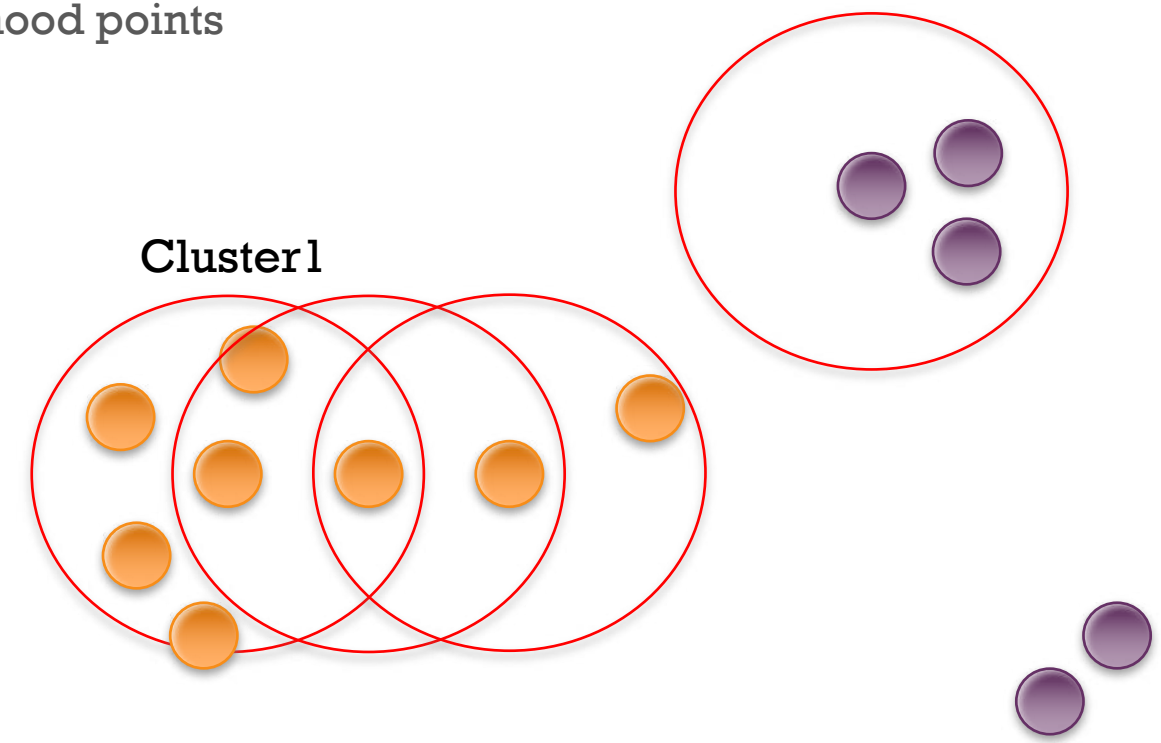
2. All points which are within the r distance are neighborhood points

3. If (number of neighborhood points > minPoints)
   - Assign all neighborhood points to same cluster
   - Repeat step 2 – 3 on all neighborhood points
   - marked all of these point as "visited"

Else
   - the point will be labeled as noise
   - marked all of these point as "visited"

4. Repeat step 1-3 until all points are visited

minPoints = 3



Clustering - DBSCAN

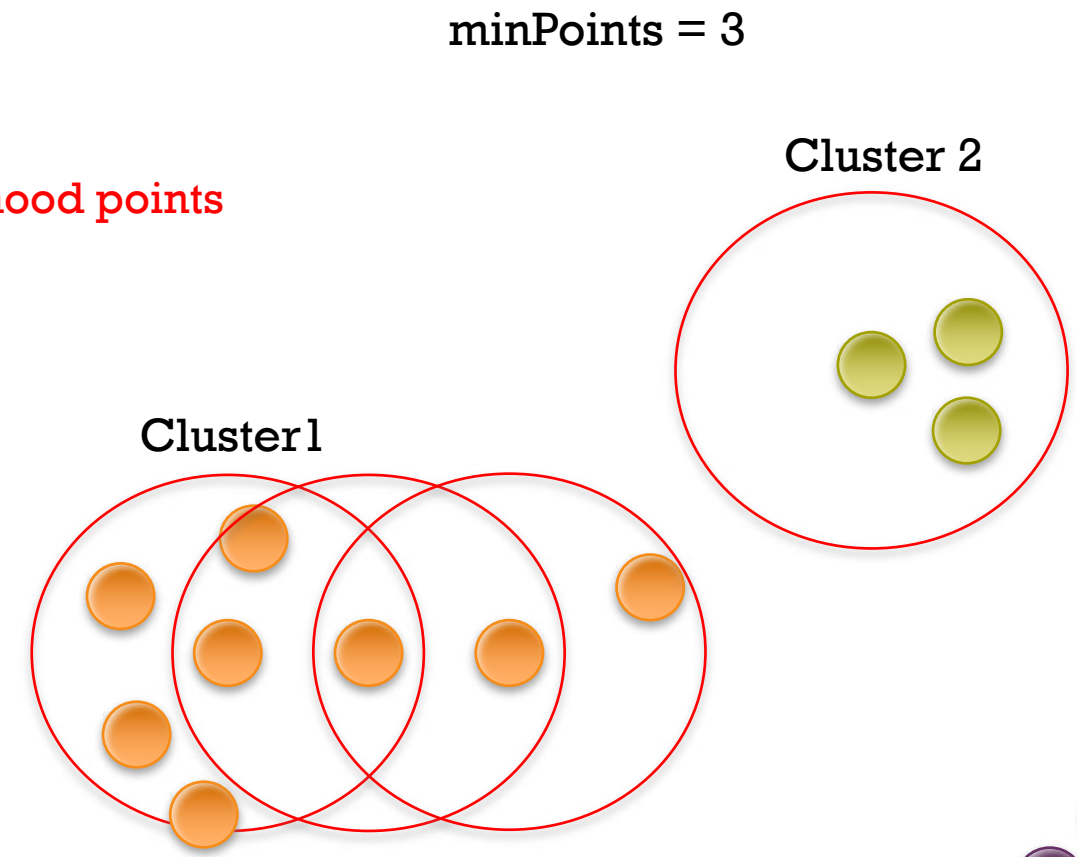
Step

1. Initial starting point that has not been visited
2. All points which are within the r distance are neighborhood points
3. If (number of neighborhood points > minPoints)
   - Assign all neighborhood points to same cluster
   - Repeat step 2 – 3 on all neighborhood points
   - marked all of these point as "visited"

Else

- the point will be labeled as noise
- marked all of these point as "visited"

4. Repeat step 1-3 until all points are visited



Clustering - DBSCAN

Step

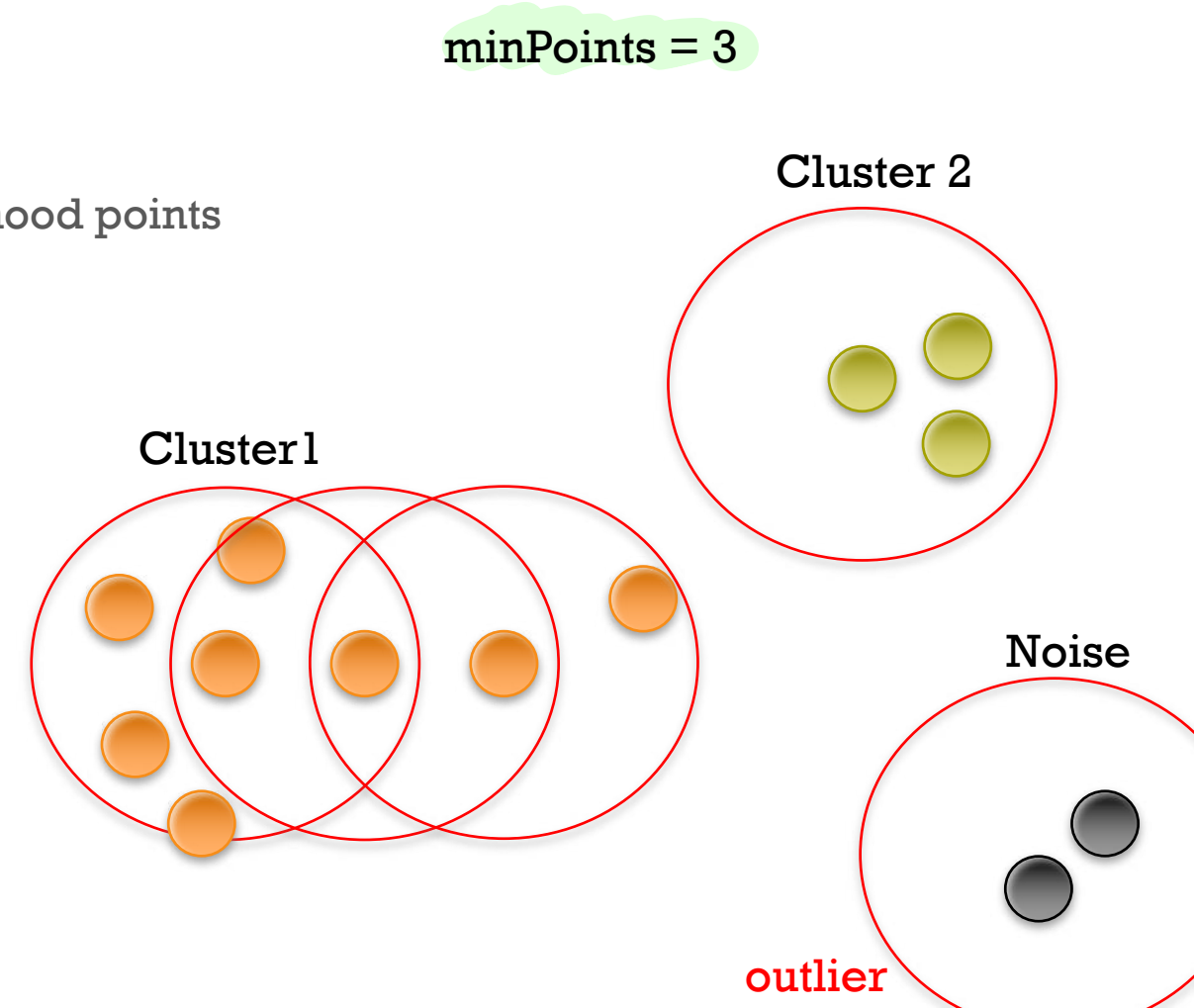
1. Initial starting point that has not been visited

2. All points which are within the r distance are neighborhood points

3. If (number of neighborhood points > minPoints)
   - Assign all neighborhood points to same cluster
   - Repeat step 2 – 3 on all neighborhood points
   - marked all of these point as "visited"

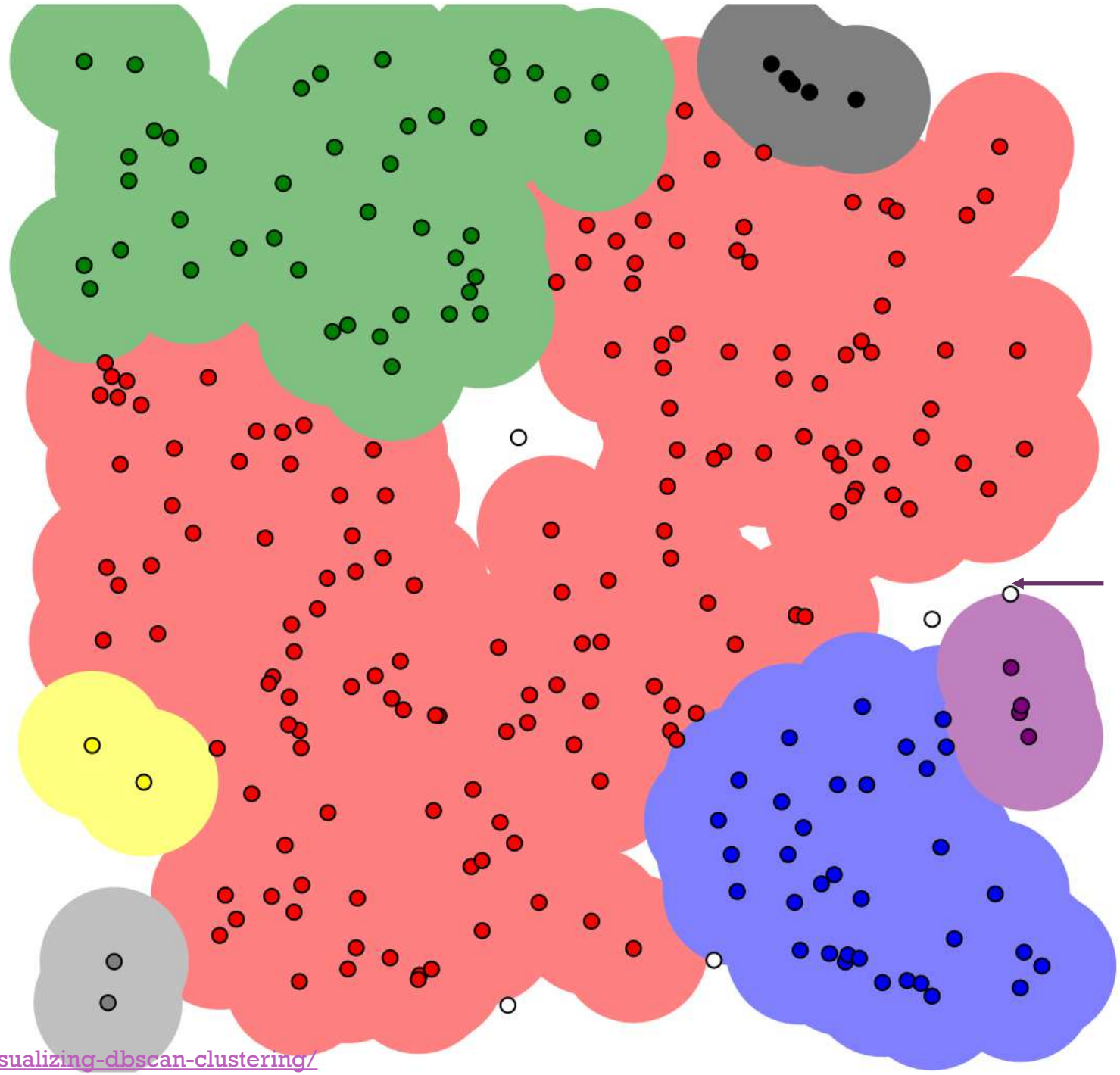
Else
   - the point will be labeled as noise
   - marked all of these point as "visited"

4. Repeat step 1-3 until all points are visited



Clustering - DBSCAN

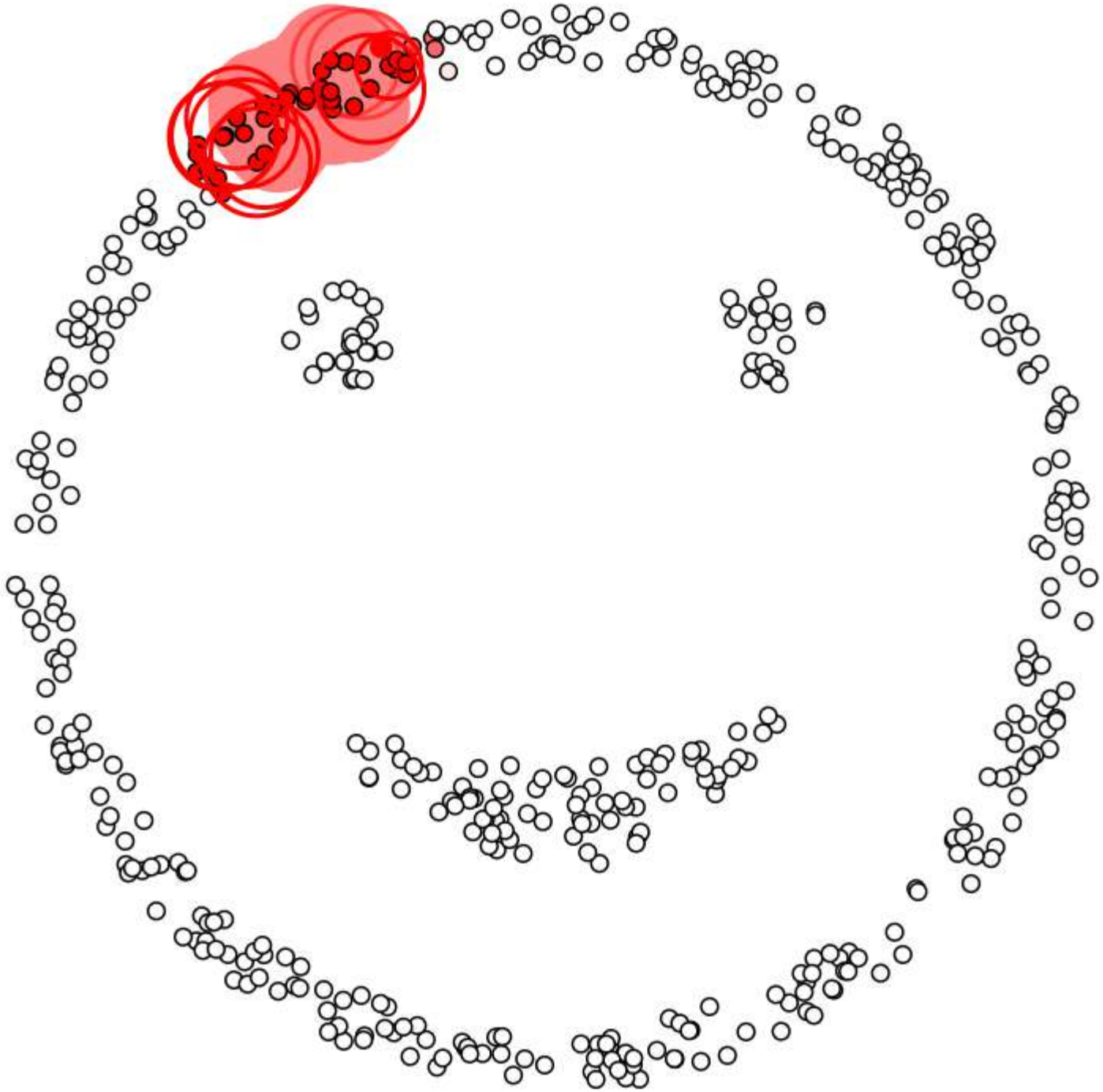
- Try yourself
- Round 1
  - Epsilon =1
  - minPoints = 4
- Round 1
  - Epsilon =1
  - minPoints = 2
- Round 1
  - Epsilon =1.5
  - minPoints = 4



Can't form cluster, outlier

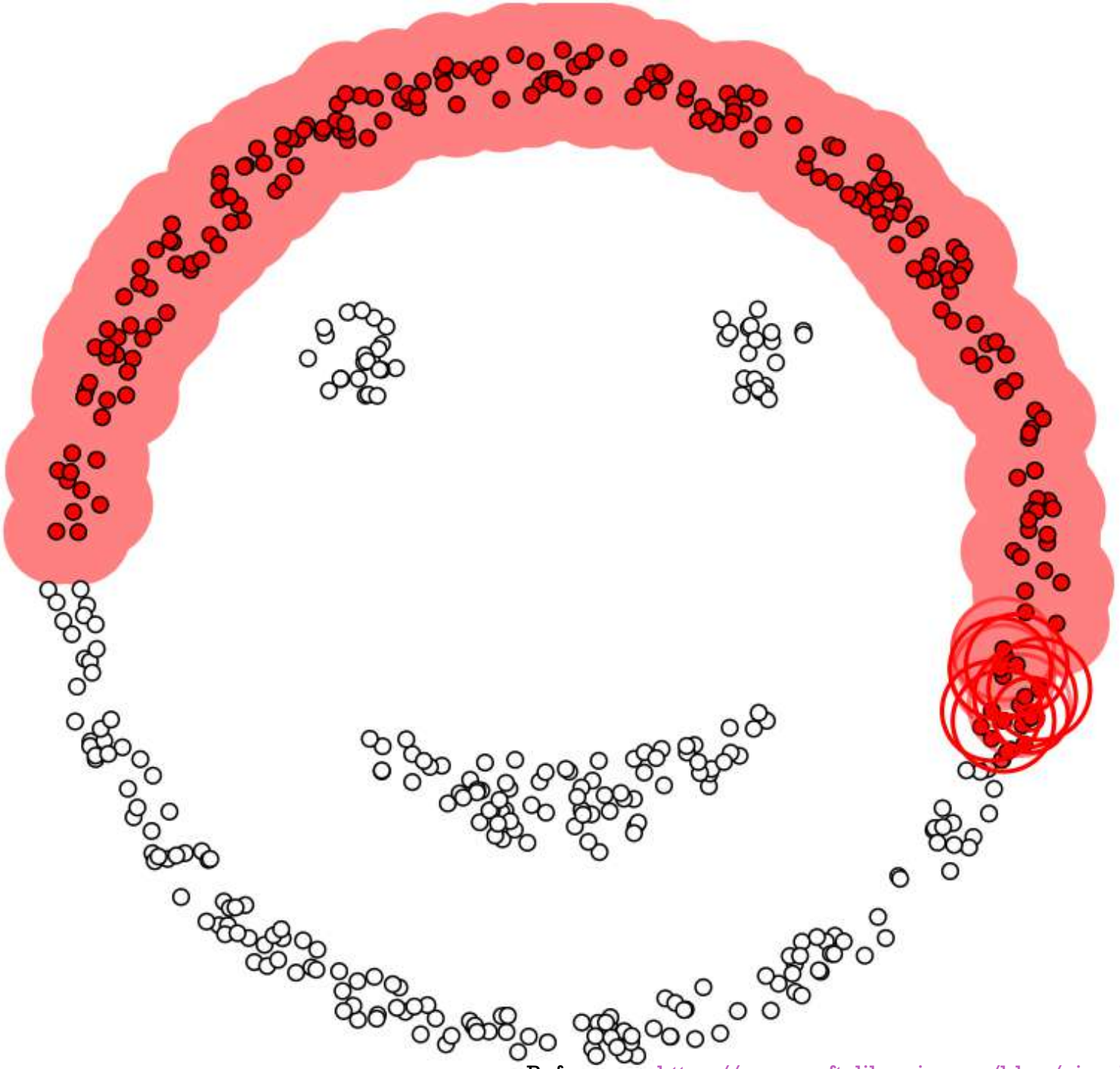
Reference : https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/

Clustering - DBSCAN



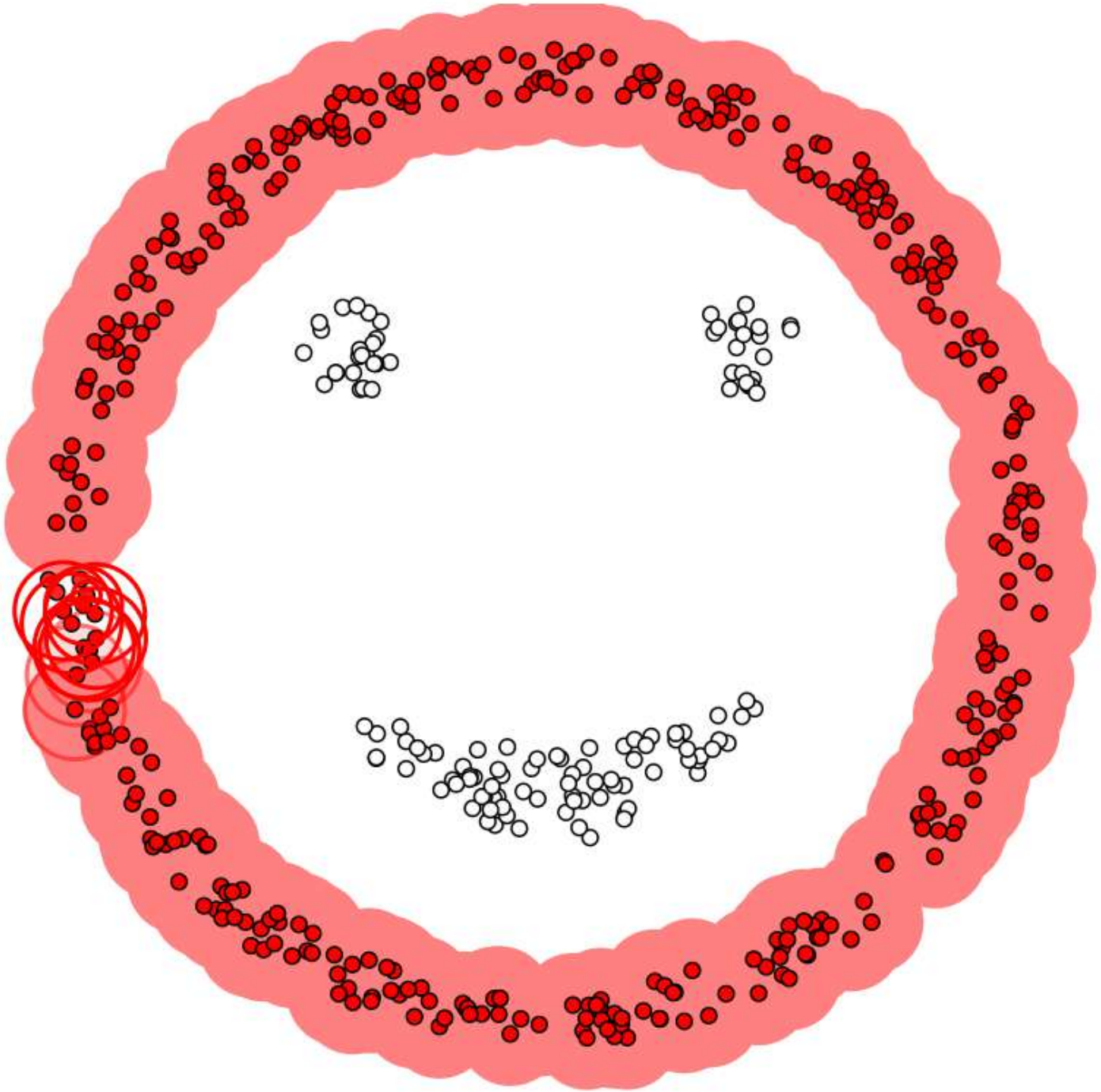
Reference : https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/

Clustering - DBSCAN



Reference : https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/

Clustering - DBSCAN

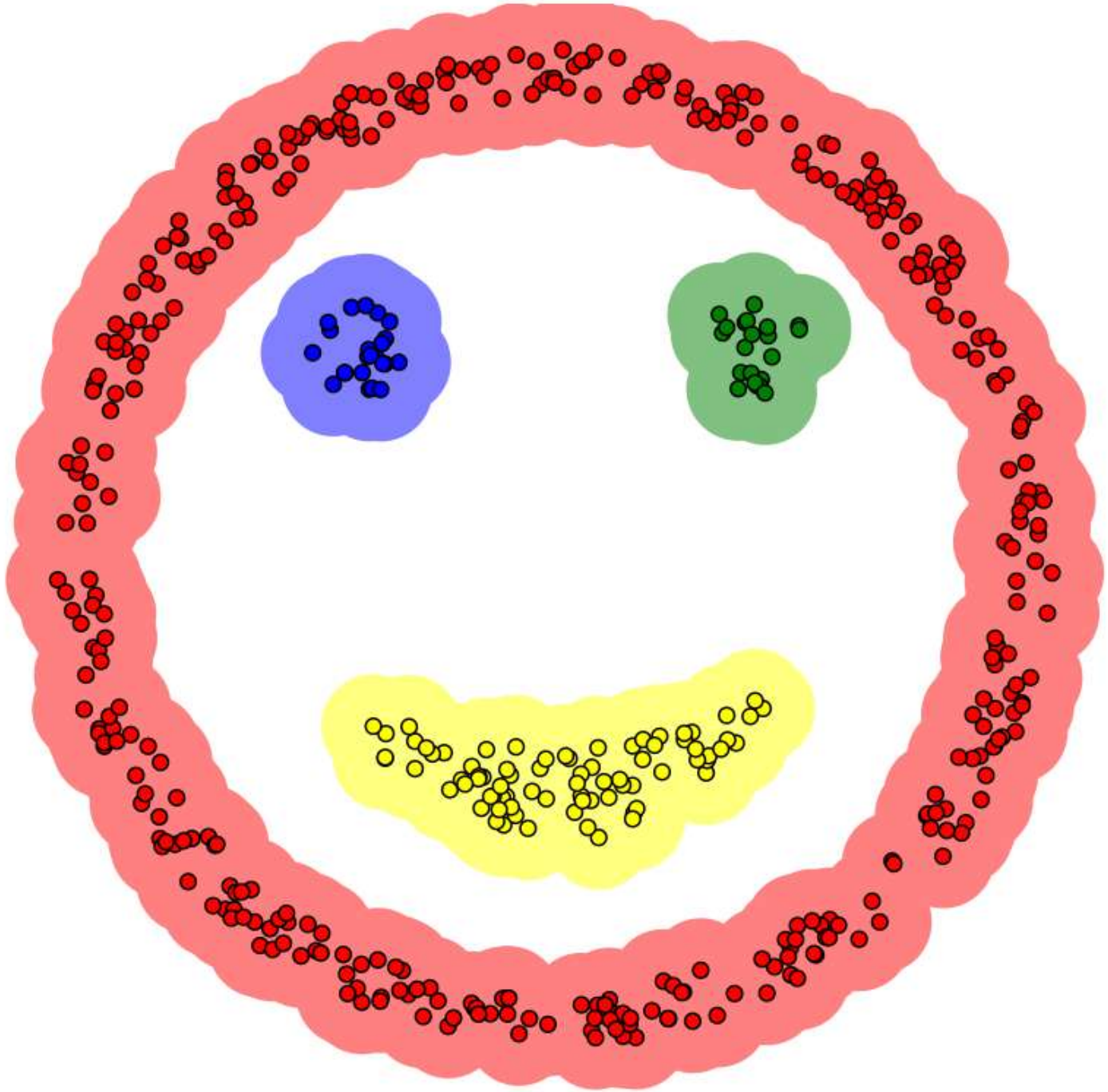


Reference : https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/

Clustering - DBSCAN

1.) cluster
- geo location

2.) anomaly detection
- fraud



Reference : https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/

Clustering - DBSCAN

sklearn.cluster.DBSCAN

class `sklearn.cluster.` **`DBSCAN`**(*eps=0.5, *, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None*) [source]

Perform DBSCAN clustering from vector array or distance matrix.

DBSCAN - Density-Based Spatial Clustering of Applications with Noise. Finds core samples of high density and expands clusters from them. Good for data which contains clusters of similar density.

Read more in the User Guide.

Parameters:

eps : *float, default=0.5*
The maximum distance between two samples for one to be considered as in the neighborhood of the other. This is not a maximum bound on the distances of points within a cluster. This is the most important DBSCAN parameter to choose appropriately for your data set and distance function.

min_samples : *int, default=5*
The number of samples (or total weight) in a neighborhood for a point to be considered as a core point. This includes the point itself.

metric : *string, or callable, default='euclidean'*
The metric to use when calculating distance between instances in a feature array. If metric is a string or callable, it must be one of the options allowed by `sklearn.metrics.pairwise_distances` for its metric parameter. If metric is "precomputed", X is assumed to be a distance matrix and must be square. X may be a Glossary, in which case only "nonzero" elements may be considered neighbors for DBSCAN.

New in version 0.17: metric *precomputed* to accept precomputed sparse matrix.

metric_params : *dict, default=None*
Additional keyword arguments for the metric function.

New in version 0.19.

algorithm : *{'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'*
The algorithm to be used by the NearestNeighbors module to compute pointwise distances and find nearest neighbors. See NearestNeighbors module documentation for details.

Market Basket Analysis

Association Rule mining

- Discover associations between items
- Count combinations of items that occur together frequently in transactions

$$Support(\{X\} \rightarrow \{Y\}) = \frac{Transactions\ containing\ both\ X\ and\ Y}{Total\ number\ of\ transactions}$$

$$Confidence(\{X\} \rightarrow \{Y\}) = \frac{Transactions\ containing\ both\ X\ and\ Y}{Transactions\ containing\ X}$$

$$Lift(\{X\} \rightarrow \{Y\}) = \frac{(Transactions\ containing\ both\ X\ and\ Y)/(Transactions\ containing\ X)}{Fraction\ of\ transactions\ containing\ Y}$$

Market Basket Analysis (Association Rule Mining)

ความบ่อยของกฎที่ใช้ได้ → Support ข่อย

มี Donut 2 ห:กร้า

Rule	Support	Confidence
Apple => Donut	2/5	2/3 → มี apple 3 ห:กร้า
Coconut => Apple	2/5	2/4
Apple => Coconut	2/5	2/3
Banana & Coconut => Donut	1/5	1/3

Association Rule Mining (cont.)

กฎจะใช้ได้ เมื่อผ่าน minimum support และ minimum confidence

- Wal-Mart customers who purchase Barbie dolls have a 60% likelihood of also purchasing one of three types of candy bars [Forbes, Sept 8, 1997]

- Strategies?
  1. Put them closer together in the store.
  2. Put them far apart in the store.
  3. Package candy bars with the dolls.
  4. Package Barbie + candy + poorly selling item.
  5. Raise the price on one and lower it on the other.
  6. Offer Barbie accessories for proofs of purchase.
  7. Do not advertise candy and Barbie together.
  8. Offer candies in the shape of a Barbie doll.

Caution in Association Rule Mining

1. Basket size: per bill, customer, day
2. Item level: SKU, product category

mlxtend version: 0.17.2

{coconut, donut}

apriori

apriori(df, min_support=0.5, use_colnames=False, max_len=None, verbose=0, low_memory=False)

Get frequent itemsets from a one-hot DataFrame

Parameters

- `df` : pandas DataFrame

  pandas DataFrame the encoded format. Also supports DataFrames with sparse data; for more info, please see (https://pandas.pydata.org/pandas-docs/stable/ user_guide/sparse.html#sparse-data-structures)

  Please note that the old pandas SparseDataFrame format is no longer supported in mlxtend >= 0.17.2.

  The allowed values are either 0/1 or True/False. For example,

```
   Apple  Bananas   Beer  Chicken   Milk   Rice
0   True    False   True     True  False   True
1   True    False   True    False  False   True
2   True    False   True    False  False  False
```

{c,d}

c → d

d → c

association_rules

association_rules(df, metric='confidence', min_threshold=0.8, support_only=False)

Generates a DataFrame of association rules including the metrics 'score', 'confidence', and 'lift'

Parameters

- `df` : pandas DataFrame

  pandas DataFrame of frequent itemsets with columns ['support', 'itemsets']

- `metric` : string (default: 'confidence')

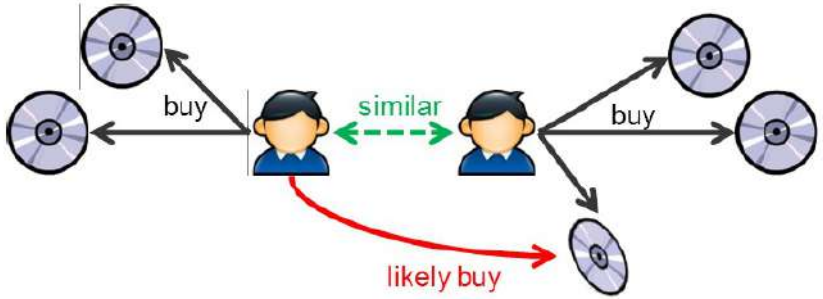
  Metric to evaluate if a rule is of interest. **Automatically set to 'support' if `support_only=True` .** Otherwise, supported metrics are 'support', 'confidence', 'lift',

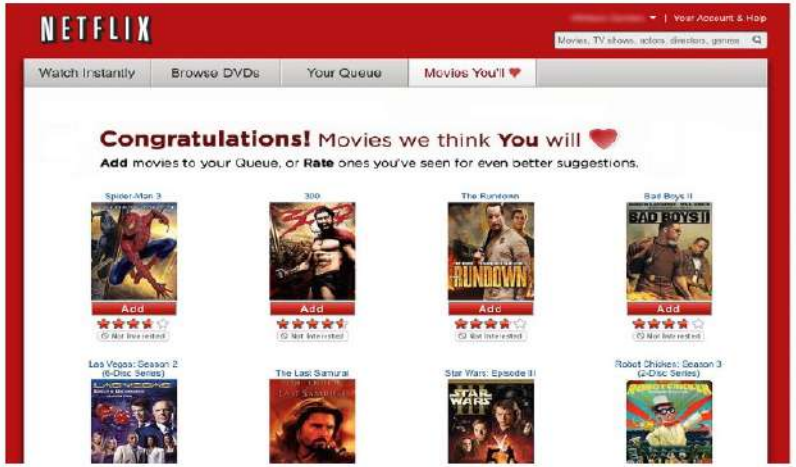
'leverage', and 'conviction' These metrics are computed as follows:

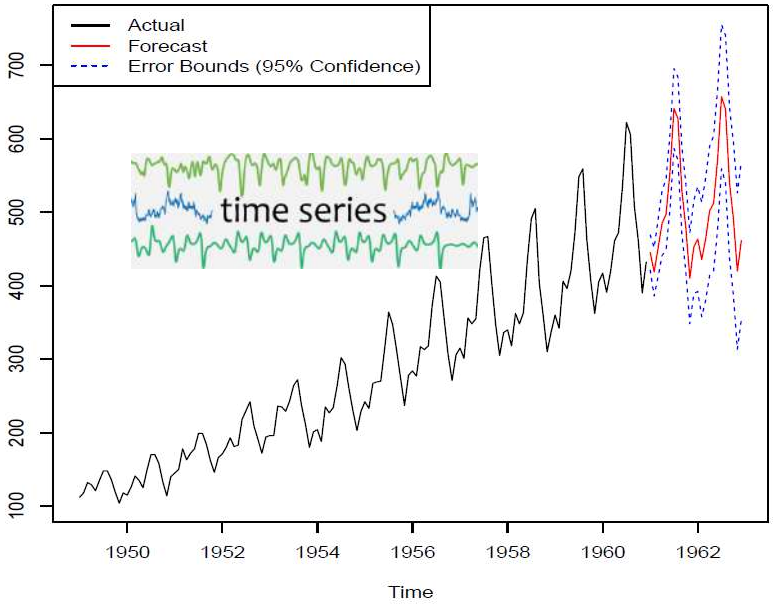
```
- support(A->C) = support(A+C) [aka 'support'], range: [0, 1]

- confidence(A->C) = support(A+C) / support(A), range: [0, 1]

- lift(A->C) = confidence(A->C) / support(C), range: [0, inf]

- leverage(A->C) = support(A->C) - support(A)*support(C),
range: [-1, 1]

- conviction = [1 - support(C)] / [1 - confidence(A->C)],
```

Part4: Special Tasks

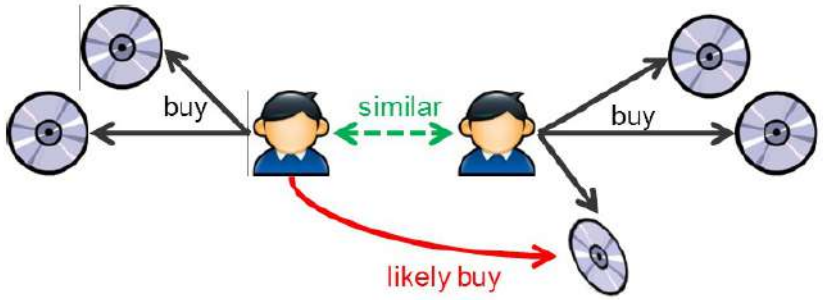
Special task

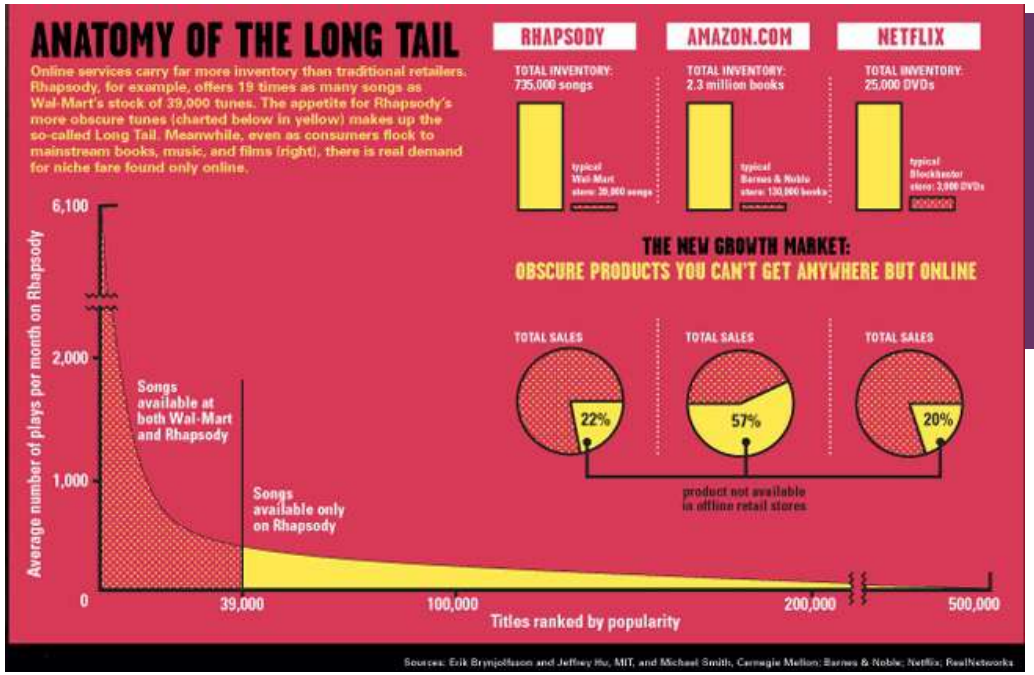


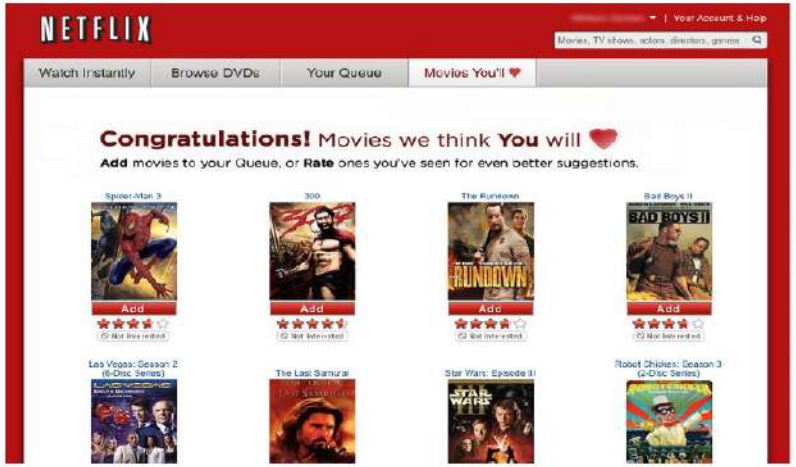




Recommendation system



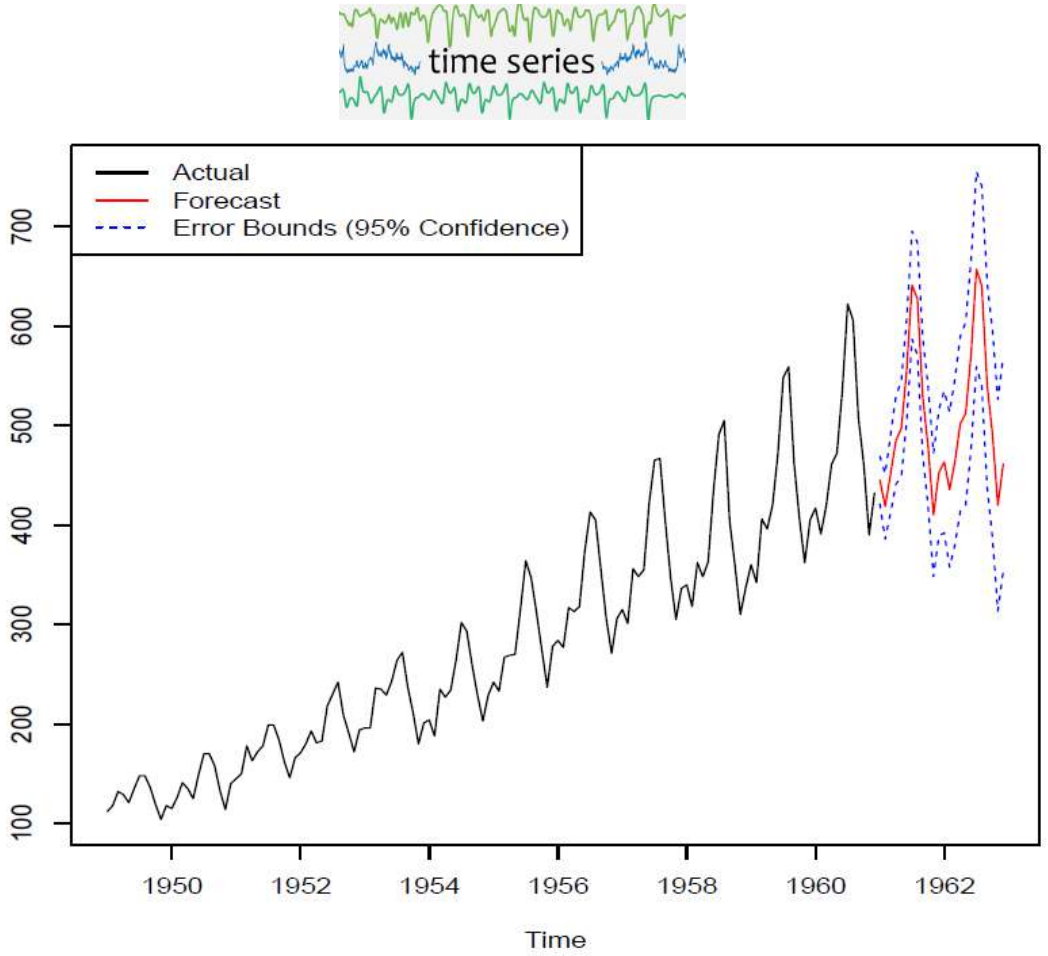




	Harry potter	X-Men	Hobbit	Argo	Pirates
101	5	2	4	?	?
102	?	?	5	2	?
103	1	2	?	?	3
104					
105					
…					

	Harry potter	X-Men	Hobbit	Argo	Pirates
101	5	2	4	1	3
102	4	1	5	2	3
103	1	2	4	1	3
104					
105					
…					

Time Series Analysis (Trend Forecasting)



- Techniques
  - **ARIMA (Autoregressive integrated moving average)**
  - Exponential Smoothing
  - Neural Networks
  - Deep Learning
- Sample Applications
  - Customer trend forecasting
  - Revenue trend forecasting
  - Rainfall forecasting
  - Remaining useful life forecasting (preventive maintenance)

Text Mining

https://ischool.syr.edu/infospace/2013/04/23/what-is-text-mining/

- Text mining, which is sometimes referred to "text analytics" is one way to make qualitative or "unstructured" data **usable by a computer**.
- Convert from unstructured to structured data

Parse → Stop → Stem

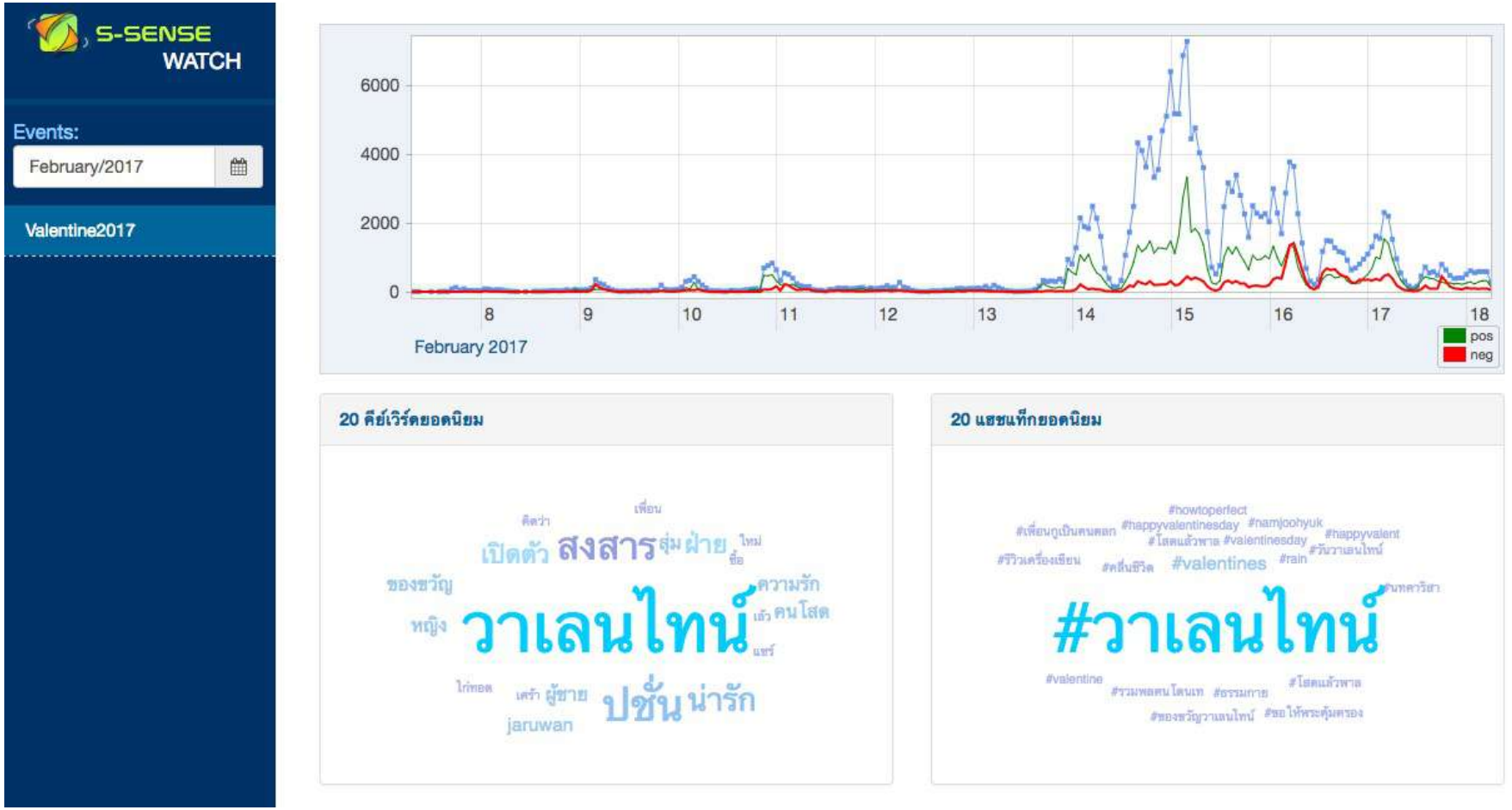
Comments	Good	Like	Hate	Sentiment
Tweet1	7	8	0	😁
Tweet2	1	0	10	😡
Tweet3	2	9	1	😁

Text Mining (cont.): Sentiment Analysis

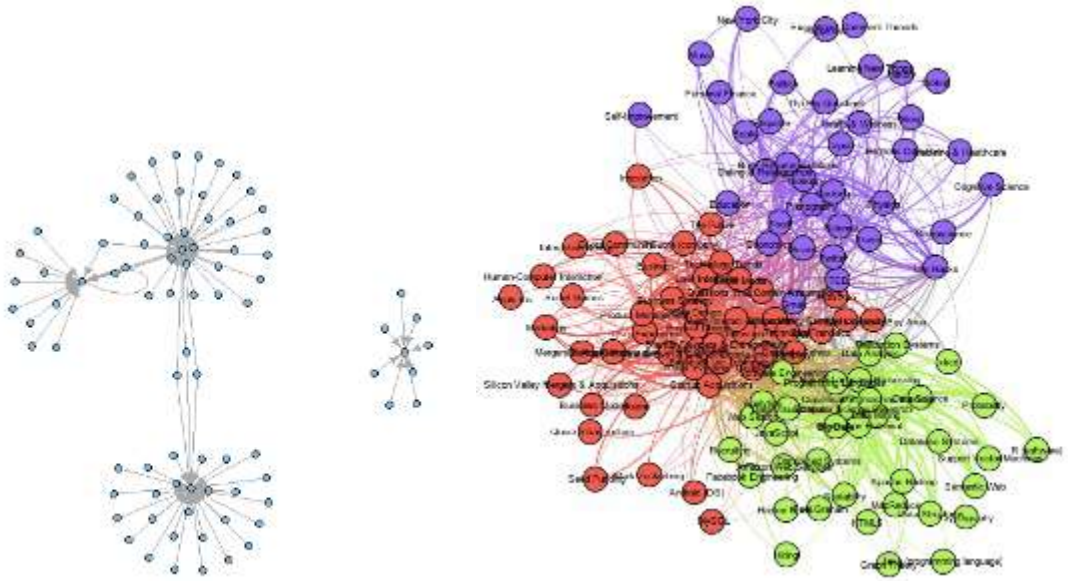
http://pop.ssense.in.th/

Text Mining (cont.): Emerging Trend Analysis

http://www.ssense.in.th/watch/



Social Network Analysis



- Techniques
  - Centrality: degree, closeness, betweenness, transitivity
  - Community detection
  - Graph Clustering
- Sample Applications
  - Influencer detection
  - Community detection

+

Part5: Demo

Demo

- 1_Decision-Trees_Random-Forests.ipynb
- 2_Linear-Regression.ipynb
- 3_Logistic-Regression.ipynb
- 4_Neural-Network.ipynb
- 5_K_Nearest_Neighbors
- 6_Support_Vector_Machine
- 7_Save_Load_Model

- 8_K-Means-Clustering.ipynb
- 9_Market_Basket_Intro.ipynb

Exercise

Target=y - has the client subscribed a term deposit? (binary: 'yes', 'no')

Bank Direct Target Marketing

M1: Simple Logistic Regression

1) data prep

2) Model

- Logistic Regression
- print model (coef, intercept)

3) Evaluation

- confusion matrix
- classification report

M2: Categorical

- dummy code
- recode - ordinal

M3: Model Selection

- backward selection

The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe a term deposit (variable y).

Data Set Information: The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

Attribute Information:

Bank client data:

- Age (numeric)
- Job : type of job (categorical: 'admin.', 'blue-collar', 'entrepreneur', 'housemaid', 'management', 'retired', 'self-employed', 'services', 'student', 'technician', 'unemployed', 'unknown')
- Marital : marital status (categorical: 'divorced', 'married', 'single', 'unknown' ; note: 'divorced' means divorced or widowed)

Thank you
& any questions