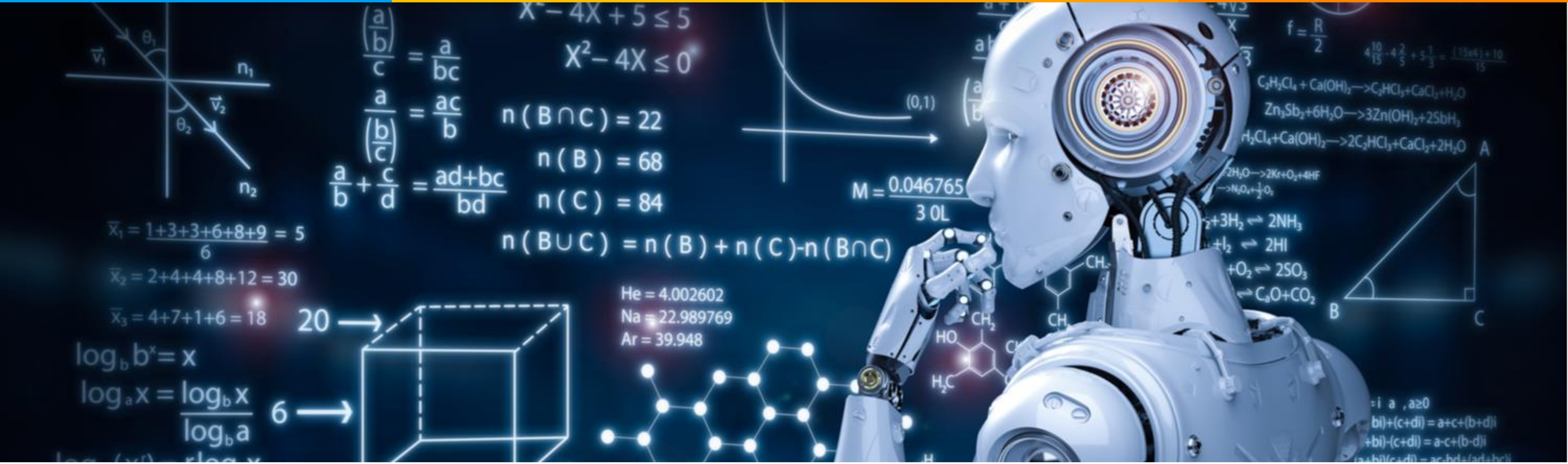
Duration: 120 min | Srikanth Shenoy | AML 5102 | M.E. AI & ML



Lecture 26 & 27: Feature Selection Part 1

Recap

- All about Linear Regression

A question to ponder

- Suppose you are running sklearn algorithm with a large dataset. It is running slow. What will you think of optimizing?
  - App level
    - Fixing Logic, Grid search CV (reducing range)
  - System level: Multi threading/Multi-processing
  - Hardware level: Using GPU
  - Architecture level
    - Vertical scaling – Get a bigger machine
    - Horizontal scaling – Spread the load across machines
- What obvious thing have you missed?

Some facts about Python concurrency/parallelism

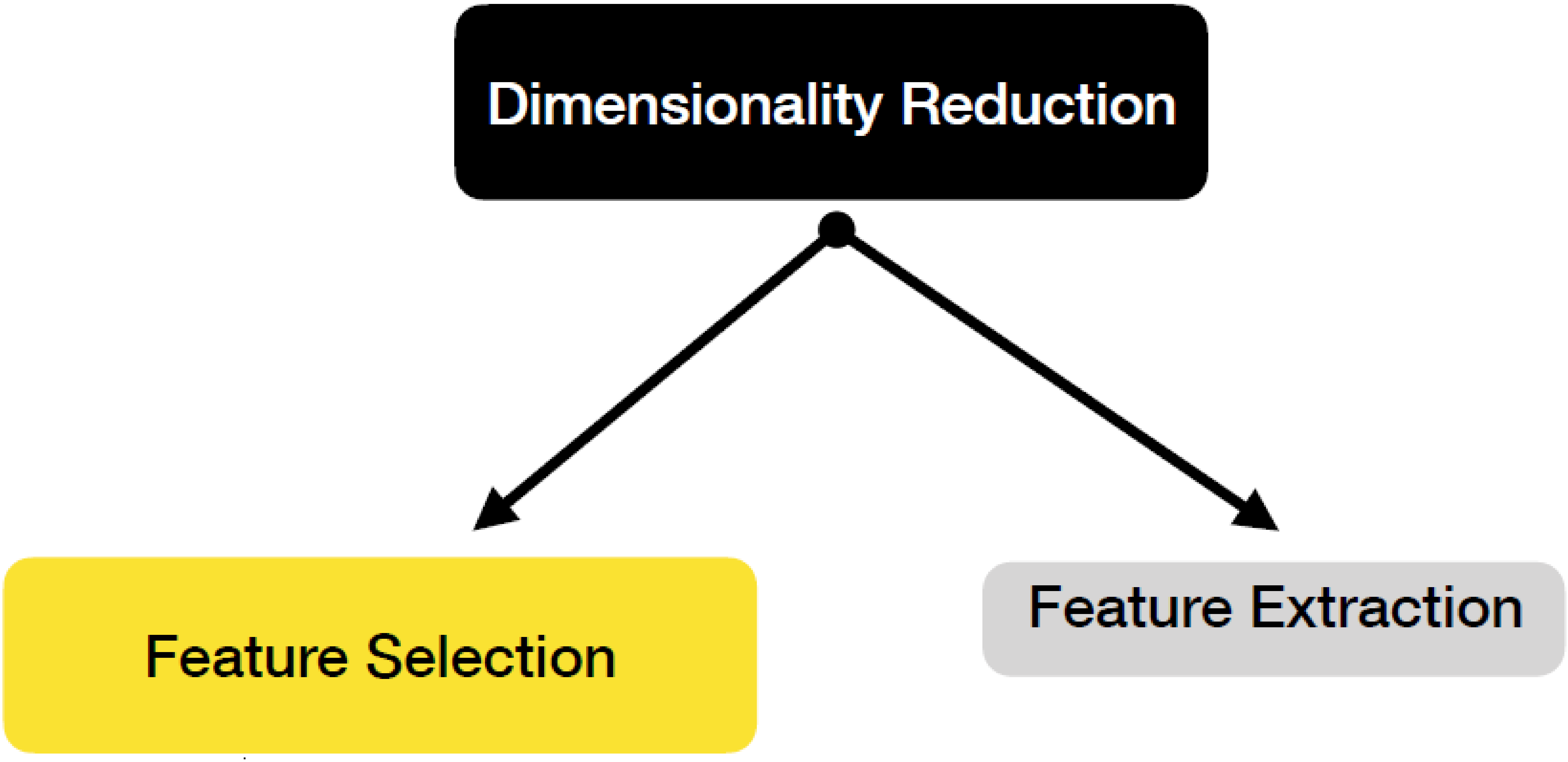
- No real multi threading in Python
  - Blame it on GIL
- No multi core computing in Python foundation
  - Not a big deal in other languages
- In python
  - Multi-processing libraries are available as add-on
  - Needs explicit constructs in coding to achieve
  - Process is always bound to CPU on which python runs

Speeding up, Scaling up sklearn

- System Level: Set n_jobs=-1
- Can sklearn use GPU?
  - If yes, why?
  - If not, why not?
- Horizontal scaling for sklearn
  - Use joblib
  - Use Ray for distributed training
- Choose a different ML programming paradigm
  - Spark Mlib
  - H2O

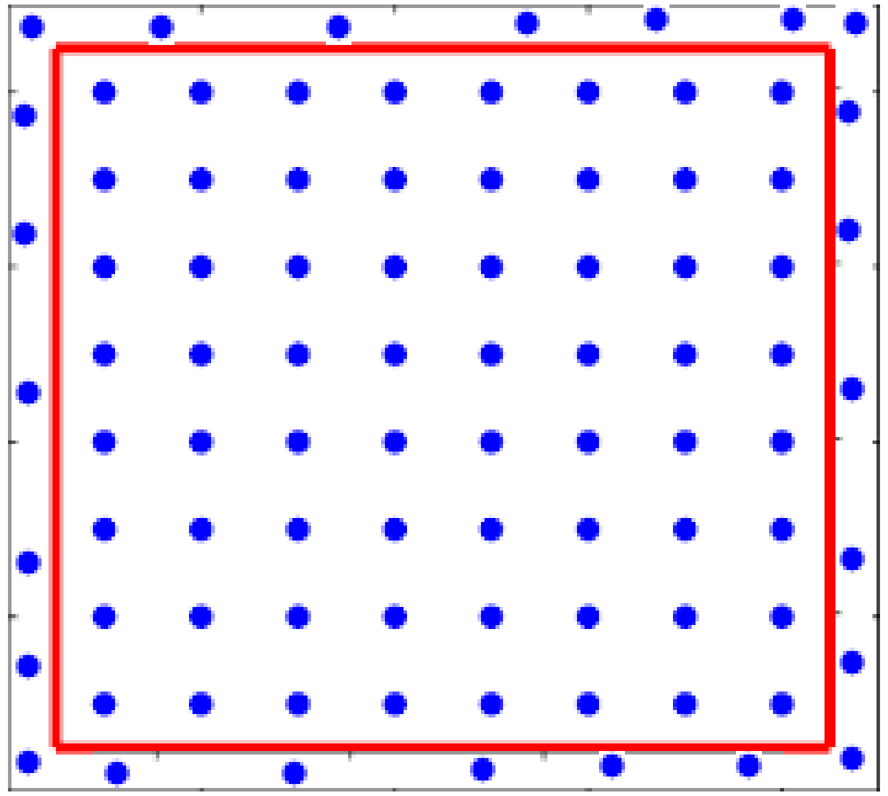
Hold on a second

- What obvious performance improvement have you missed?
  - Dimensionality reduction
    - Feature selection & Feature extraction



Why Feature Selection?

To avoid curse of dimensionality



- P(point < 0.01 units from border) =
  - 1 - P(point inside 0.99)
- P(point inside 0.99) = 0.99 * 0.99
- 1- 0.9801 = 0.0199

- P(point < 0.01 units from border) =

$$= 1 - (0.99)^3 = 1 - 0.9703 = 0.029$$

- For 1000 dimensions, P(point in 0.01 border) =

$$= 1 - (0.99)^{1000} = 1 - 0.00004317 = 0.99995683$$

Unnecessary training & deployment effort

- Impacts Training
  - Computational performance - more time for fitting
  - Slower calculation of distances etc. (kNN, Clustering)
  - Longer dot product (Generalized Linear Models)
  - More decision nodes at lower end (Decision Trees)
- Impacts Storage – Additional feature storage
- Impacts deployment/ML lifecycle
  - More feature preprocessing overhead
  - Additional ETL overhead
  - More features for managing drift

Impact on model

- Predictive Performance
  - More is not merrier
  - Adds uncertainty
  - Can potentially make model performance worse
- Interpretability becomes complicated
- Affects model robustness
  - Adds interacting features
  - Add unwanted uncertainty
- Breaks rule of Occam’s razor

A flavour of unsupervised feature selection methods

Remove Incomplete features

- Recall Imputation
- Find features with very high % of missing values
- Remove Features

Remove Constant/Quasi-Constant feature

Constant

F1	F2	F3	F4
1	0	1	2
2	8	1	2
4	5	1	2
6	6	1	0
9	4	1	2

2
1
0
0
/
2

Quasi-Constant
80% feature values are identical

- Pandas

```
quasi_constant_features = [feat for feat in X_train.columns
if X_train_normalized[feat].var() <= 0.03]
print(quasi_constant_features)
```

Remove Constant/Quasi-Constant feature

- Scikit Learn

```
from sklearn.feature_selection import VarianceThreshold

vt = VarianceThreshold(threshold=0.003)
vt.fit_transform(X_train_normalized)

mask = vt.get_support()
mask # mask tells which column to retain or remove
```

```
array([ True, False, False,  True, False, False, False, False,  True
```

```python
from sklearn.preprocessing import OneHotEncoder

X = [['blue'], ['green'], ['blue'], ['blue'],
     ['green'], ['red'], ['blue'], ['green']]
y = [0, 0, 1, 0, 0, 1, 0, 0]

enc = OneHotEncoder(drop='first')
enc.fit(X)
X_ohe = enc.transform(X)
X_ohe.toarray()
```

```python
from sklearn.feature_selection import VarianceThreshold

sel = VarianceThreshold(threshold=(.8 * (1 - .8)))

sel.fit(X_ohe)
sel.transform(X_ohe).toarray()
```

Remove Constant/Quasi-Constant feature

- Feature Engine

```
transformer = DropConstantFeatures(tol=0.7)
transformer.fit(X_train)
transformer.features_to_drop_
```

✓ 0.0s

```
['parch', 'cabin', 'embarked', 'body']
```

```
X_train_vars = transformer.transform(X_train)
X_test_vars = transformer.transform(X_test)
```

✓ 0.0s

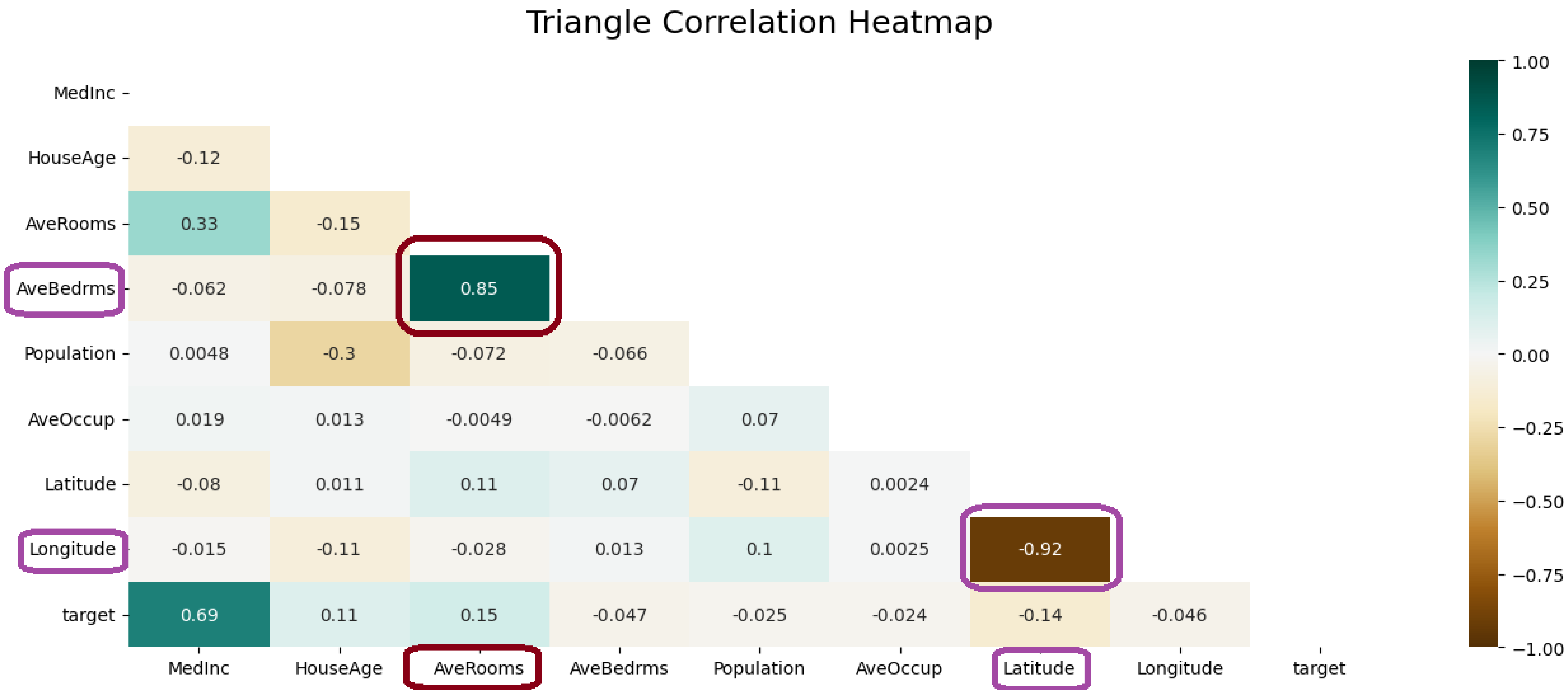
Remove Duplicate features

- Feature Engine
  - `DropDuplicateFeatures`

Remove highly correlated features

- Pearson's correlation coefficient (-1 0 +1)

$$\rho = \frac{\sum_{i=1}^{m}(x^{(i)} - \bar{x})(y^{(i)} - \bar{y})}{\sigma_x \sigma_y}$$



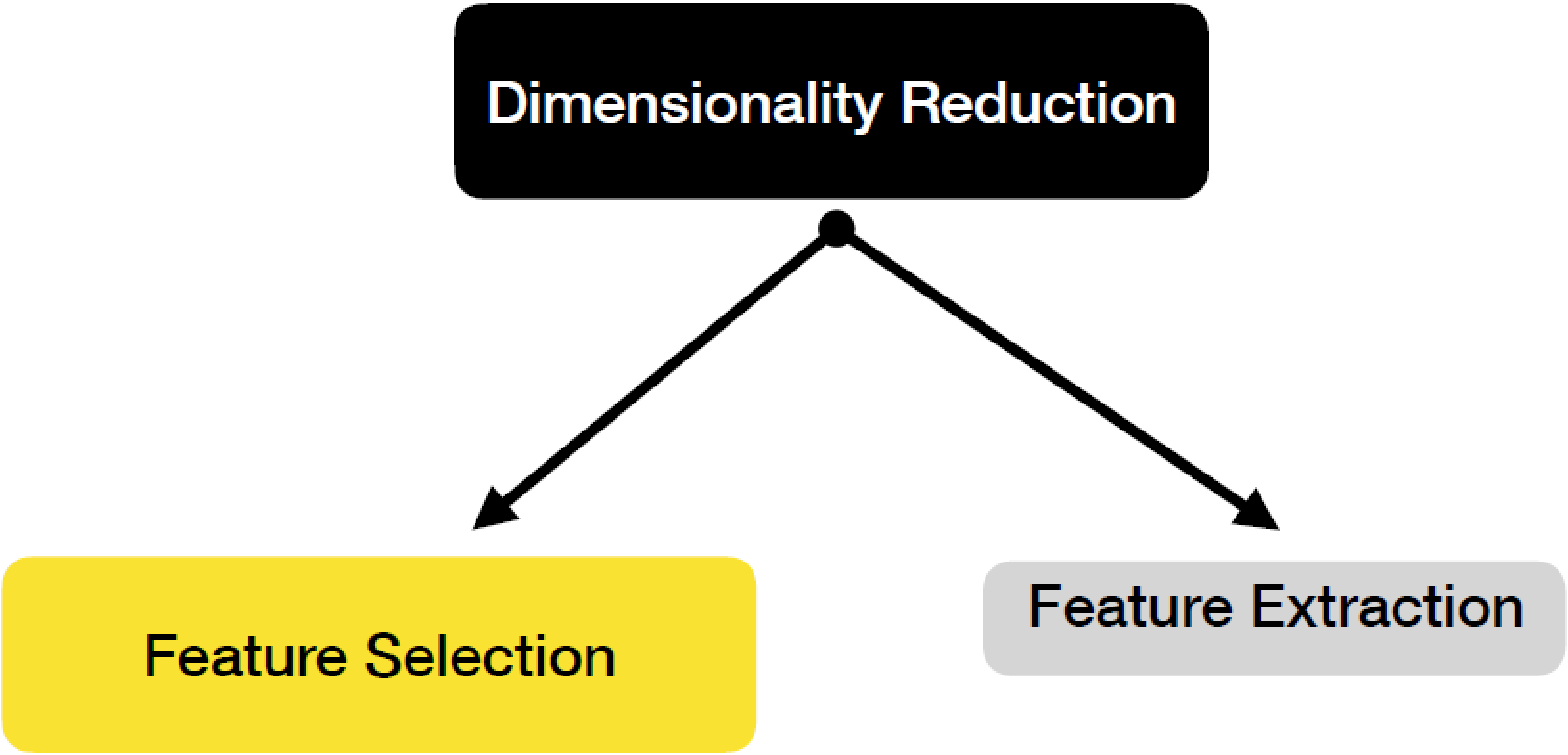
Remove highly correlated features programmatically

- Feature Engine

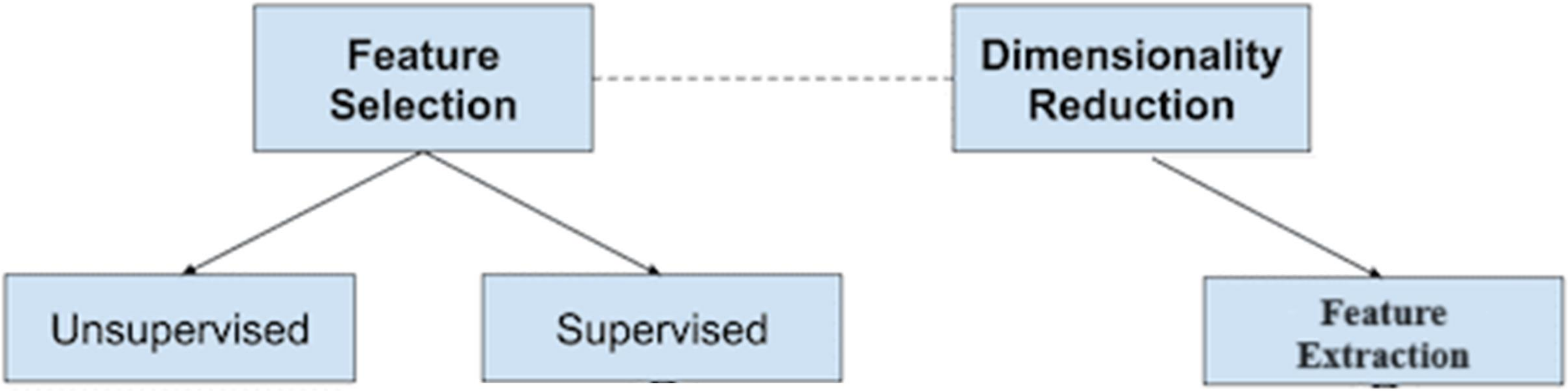
- Unsupervised
  - `DropCorrelatedFeatures`
  - Drop all but one from group of correlated features

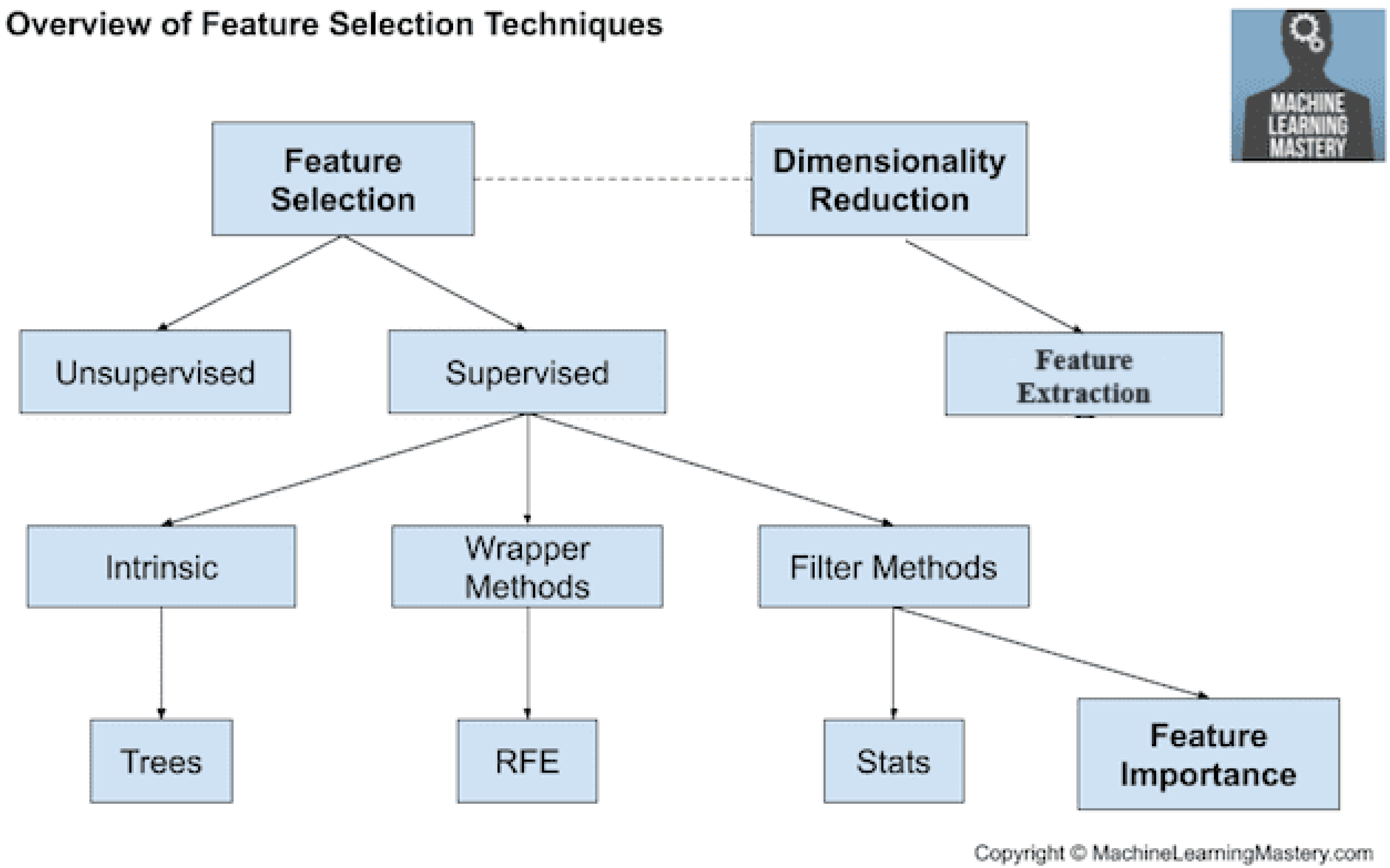
- Supervised
  - Retain feature that has best prediction power
  - `SmartCorrelatedSelection`

Feature Selection overview

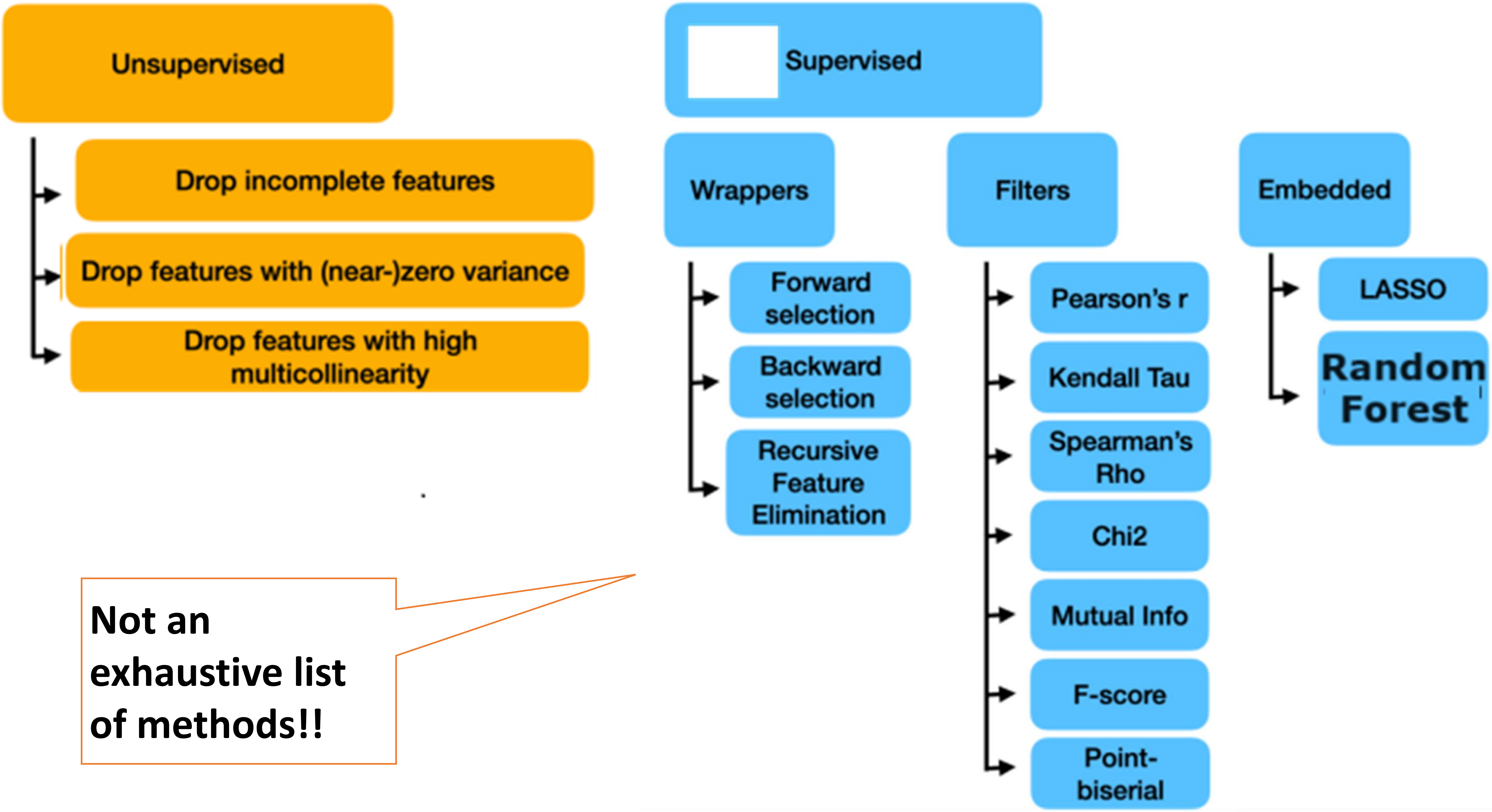

Dimensionality Reduction
Feature Selection
Feature Extraction

Overview of Feature Selection Techniques




Overview of Feature Selection Techniques
MACHINE LEARNING MASTERY
Feature Selection
Dimensionality Reduction
Unsupervised
Supervised
Feature Extraction
Intrinsic
Wrapper Methods
Filter Methods
Trees
RFE
Stats
Feature Importance
Copyright © MachineLearningMastery.com

Feature selection methods

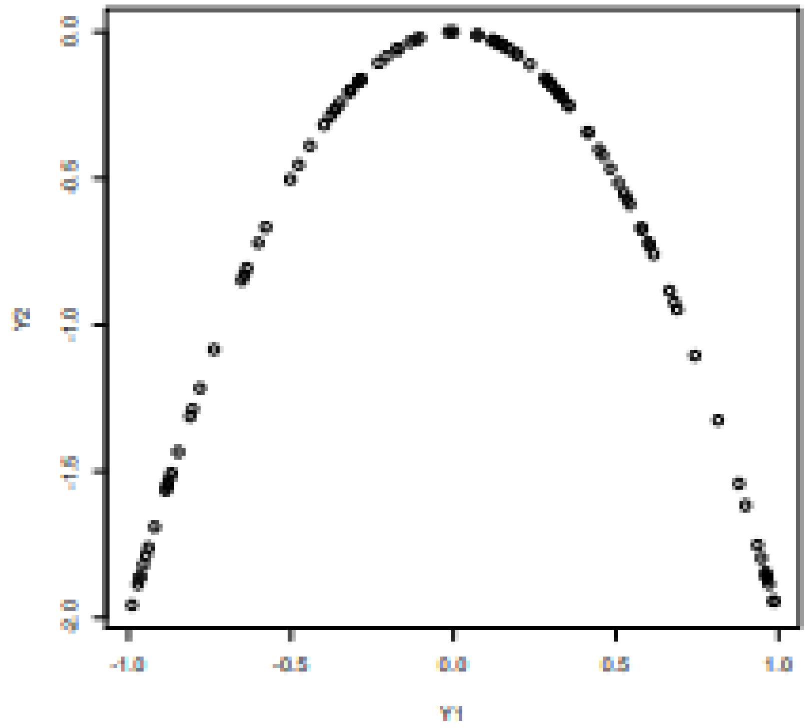


Not an exhaustive list of methods!!

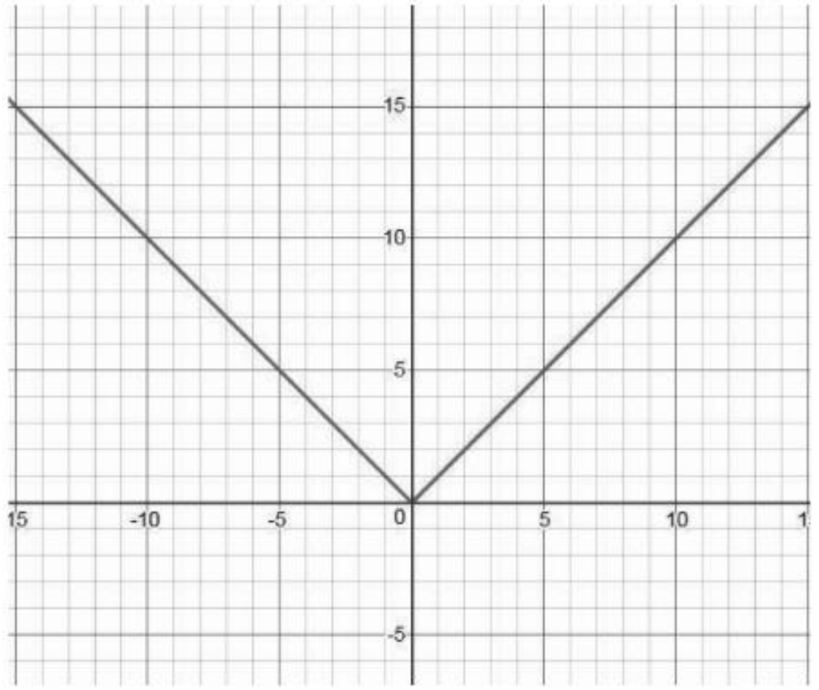
Guiding principles for Feature Selection

- Features should be as less correlated among themselves
- Features and target variable should be highly correlated

$$\rho = \frac{\mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]}{\sigma_x \sigma_y}$$

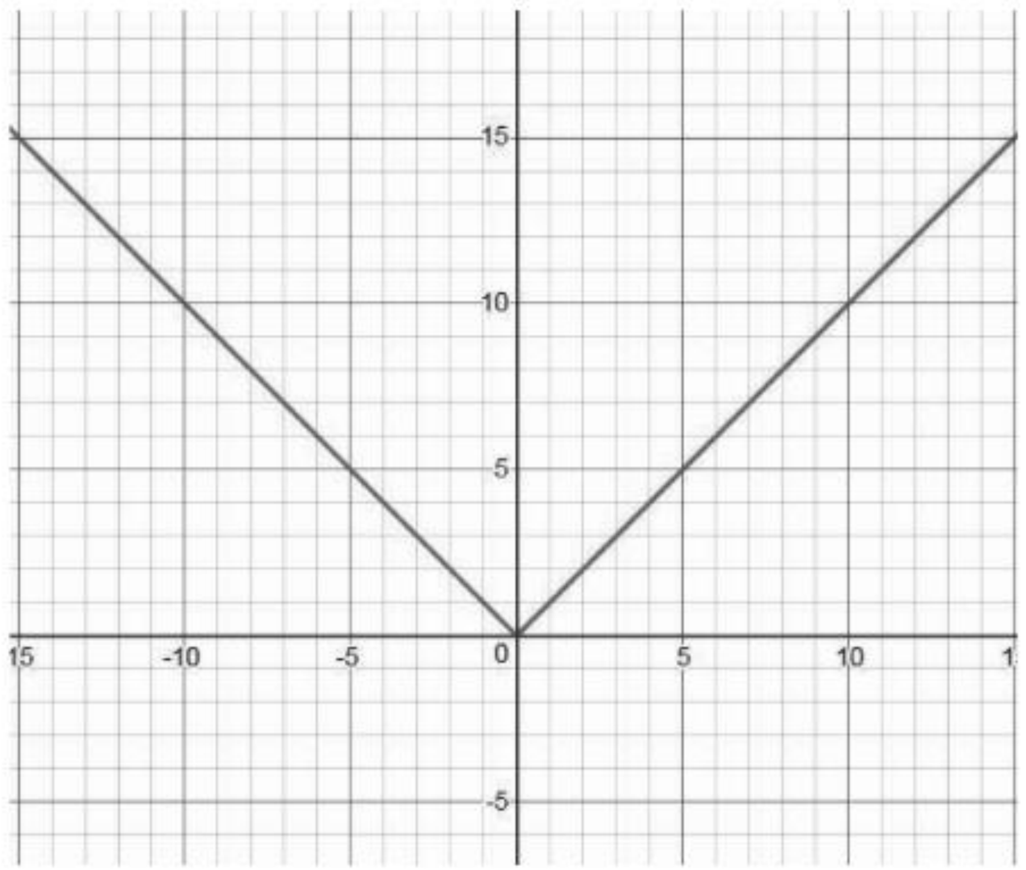


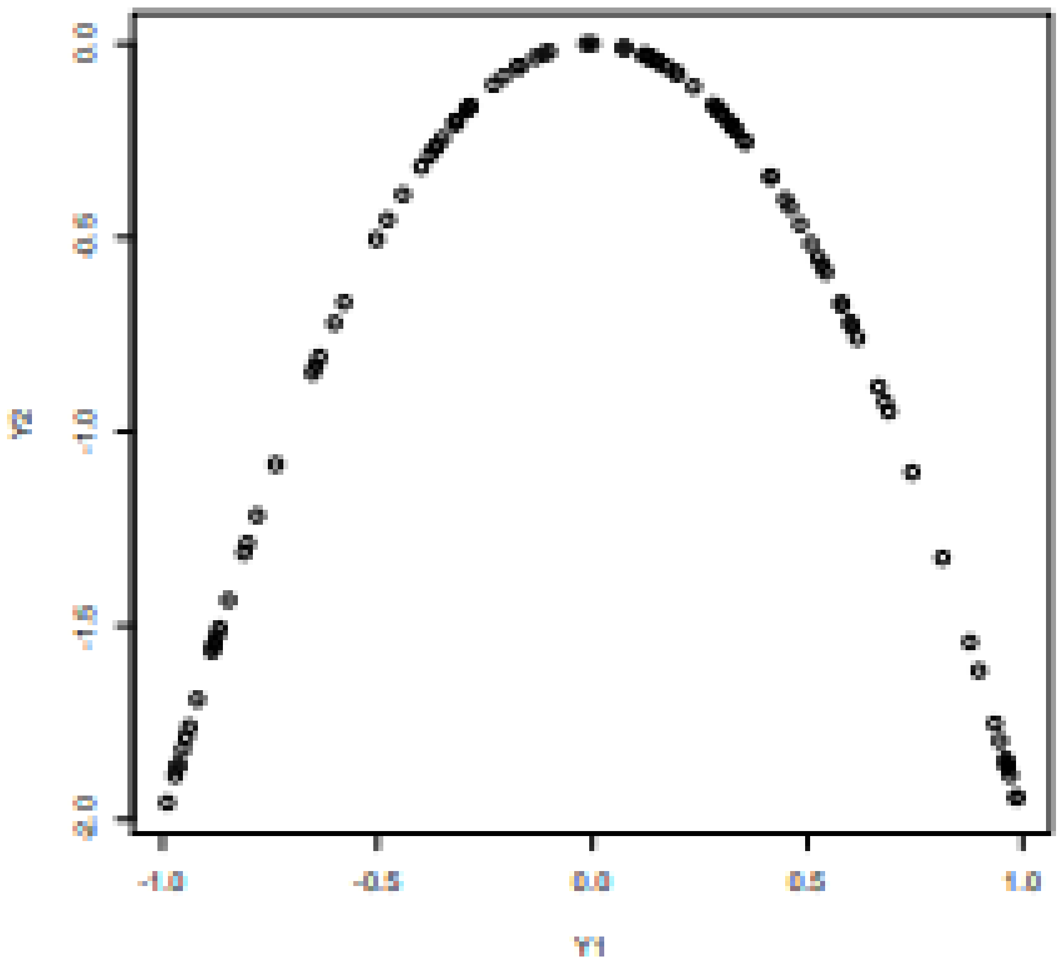
- Correlation captures linear relation
- What about non-linear association?
- Consider the plots
  - Uncorrelated but NOT independent
- Moral of the story
  - No correlation doesn't imply independence



Guiding principles for Feature Selection

- Uncorrelated but NOT independent





- There is predictive power in X beyond linearity
- Independent random variables are always uncorrelated
- Uncorrelated random variables NEED NOT be independent

Feature Selection: Correlation methods

- Features should be as less correlated among themselves
- Features and target variable should be highly correlated
- Individual (p-value of coefficients) and/or collective (p-value of F-statistic) linear predictive power of independent variables

Feature Selection – Independence methods

- Features should be independent among themselves
- Features and target variable should be highly dependent
- Statistical Independence $P(X, Y) = P(X)P(Y)$

$$\mathbb{E}[XY] = \mathbb{E}[Y]\mathbb{E}[Y]$$

- Chi-Square test for statistical independence
- ANOVA for within group and between group relationship

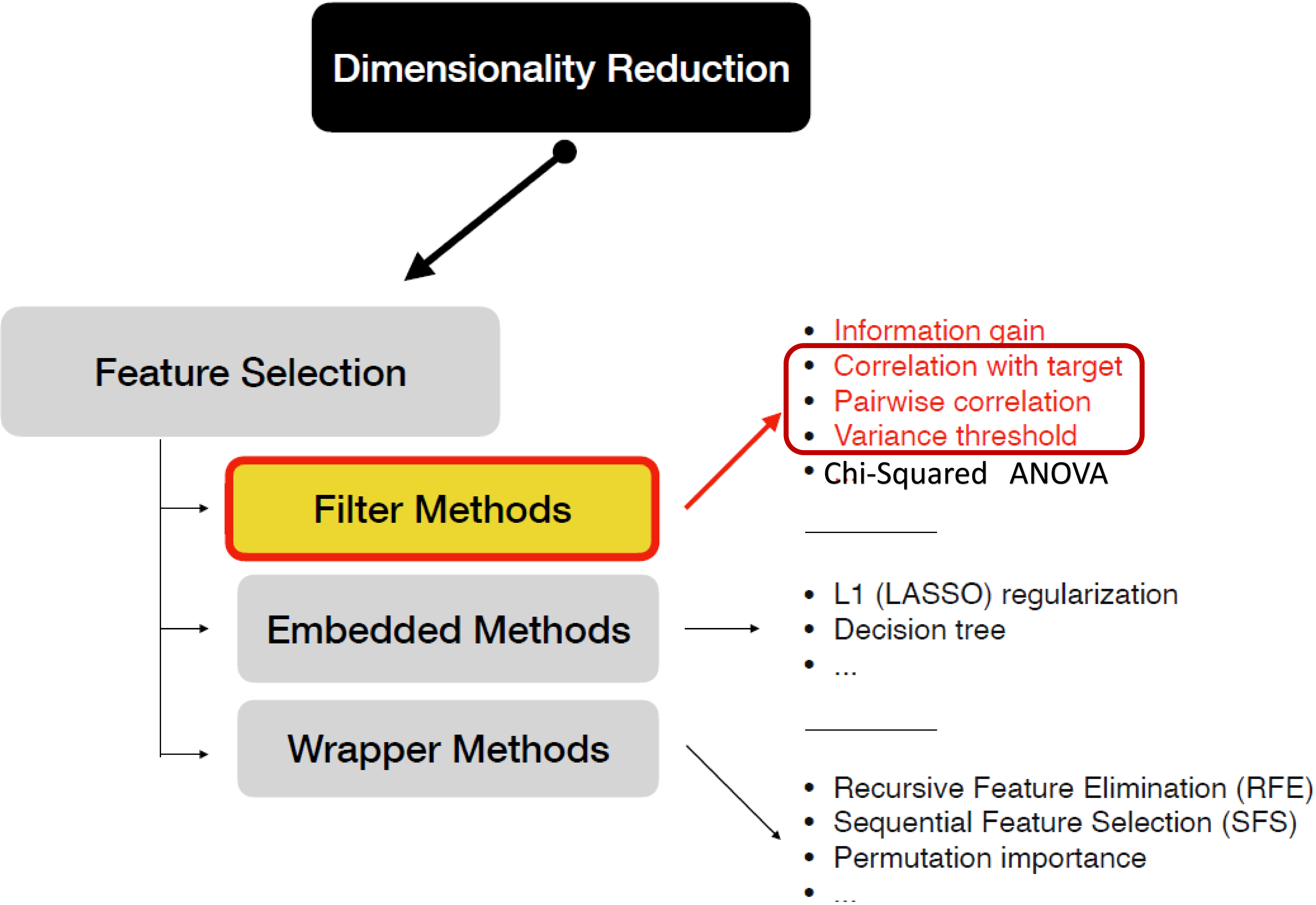
Feature Selection: Information methods

- Features should have as less mutual information among themselves but max mutual information with target
  - Entropy (general)
  - Gini Impurity (Tree specific)

Helpful libraries

- Numpy, Pandas
- Scikit Learn
- Feature Engine

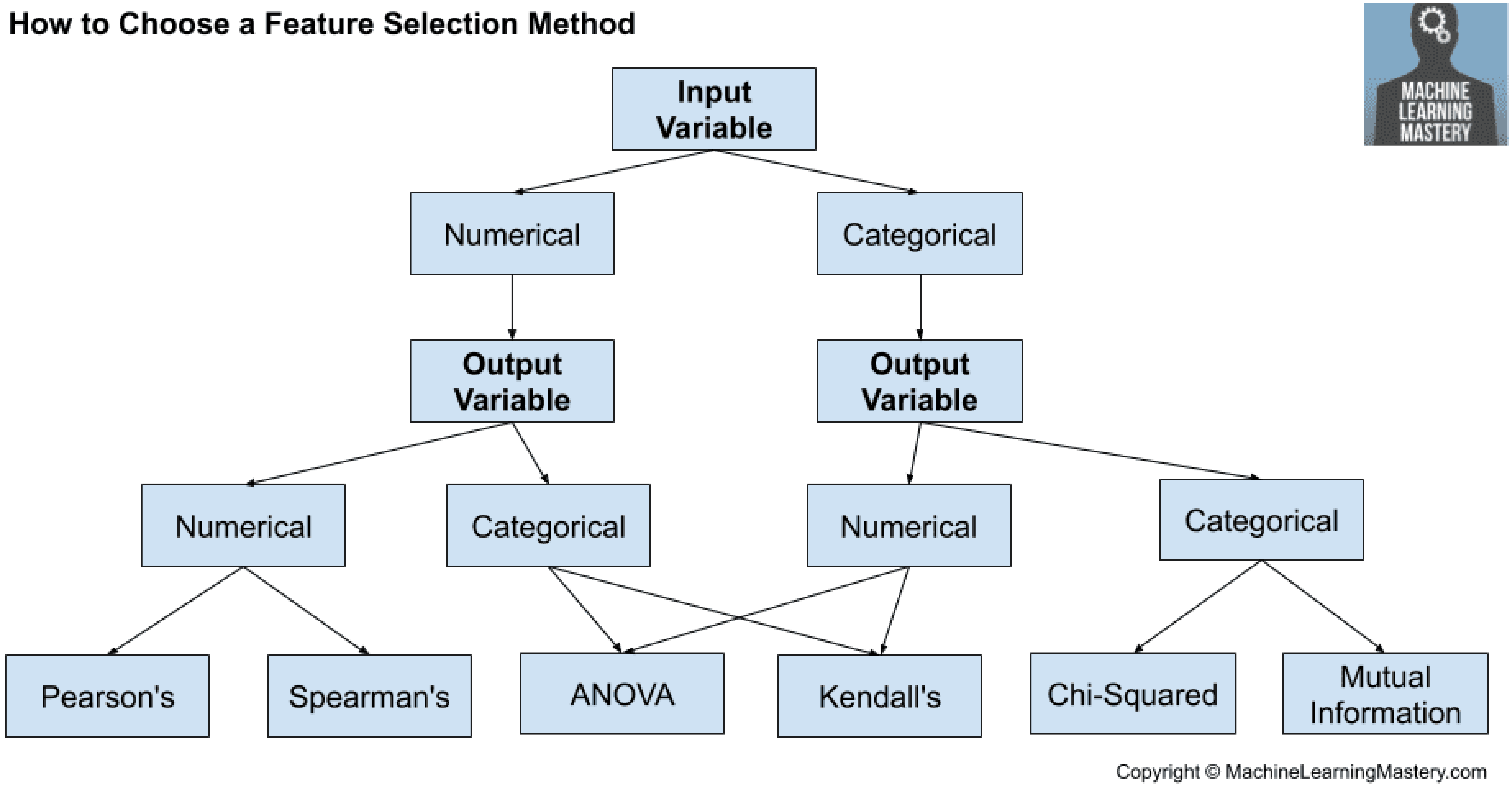
Feature Selection – Filter methods


Dimensionality Reduction
Feature Selection
Filter Methods
Embedded Methods
Wrapper Methods
• Information gain
• Correlation with target
• Pairwise correlation
• Variance threshold
• Chi-Squared ANOVA
• L1 (LASSO) regularization
• Decision tree
• ...
• Recursive Feature Elimination (RFE)
• Sequential Feature Selection (SFS)
• Permutation importance
• ...

Feature analysis: Statistical methods

- Incomplete Features (Unsupervised)
  - Features with lots of missing values (say > 50%)
- Irrelevant Features
  - Constant/Quasi-Constant Feature (Unsupervised)
  - No relation with target variable (independent of y)
    - Numeric-Numeric: Correlation, Mutual Information
    - Numerical-Categorical: Mutual Information, ANOVA
    - Categorical-Categorical: Chi-Squared Independence Test
- Redundant Features (Unsupervised)
  - Two or more features share the same information. One or more can be discarded

Supervised Feature Selection methods



Output → Input ↓	Numeric	Categorical
Numeric	1. Mutual Information 2. Pearson’s Correlation 3. Spearman’s Correlation	1. Mutual Information 2. ANOVA 3. Kendall’s 4. Chi-Squared (after discretizing input)
Categorical	1. Mutual Information 2. ANOVA 3. Kendall’s 4. Chi-Squared (after discretizing output)	1. Mutual Information 2. Chi-Squared

Filter methods

- Simple and fast
- Need an in-depth understanding of Statistics
- Supervised methods
  - Bivariate
    - Between one feature and target variable
    - Between two features
  - Multivariate – many features and target variable

Filter methods – Scikit Learn implementation

- SelectKBest
- SelectKBest(scoring="")
  - Scoring=chi2
  - Scoring=mutual_information_classif
  - Scoring=mutual_information_regression
  - f_regression
  - Many more

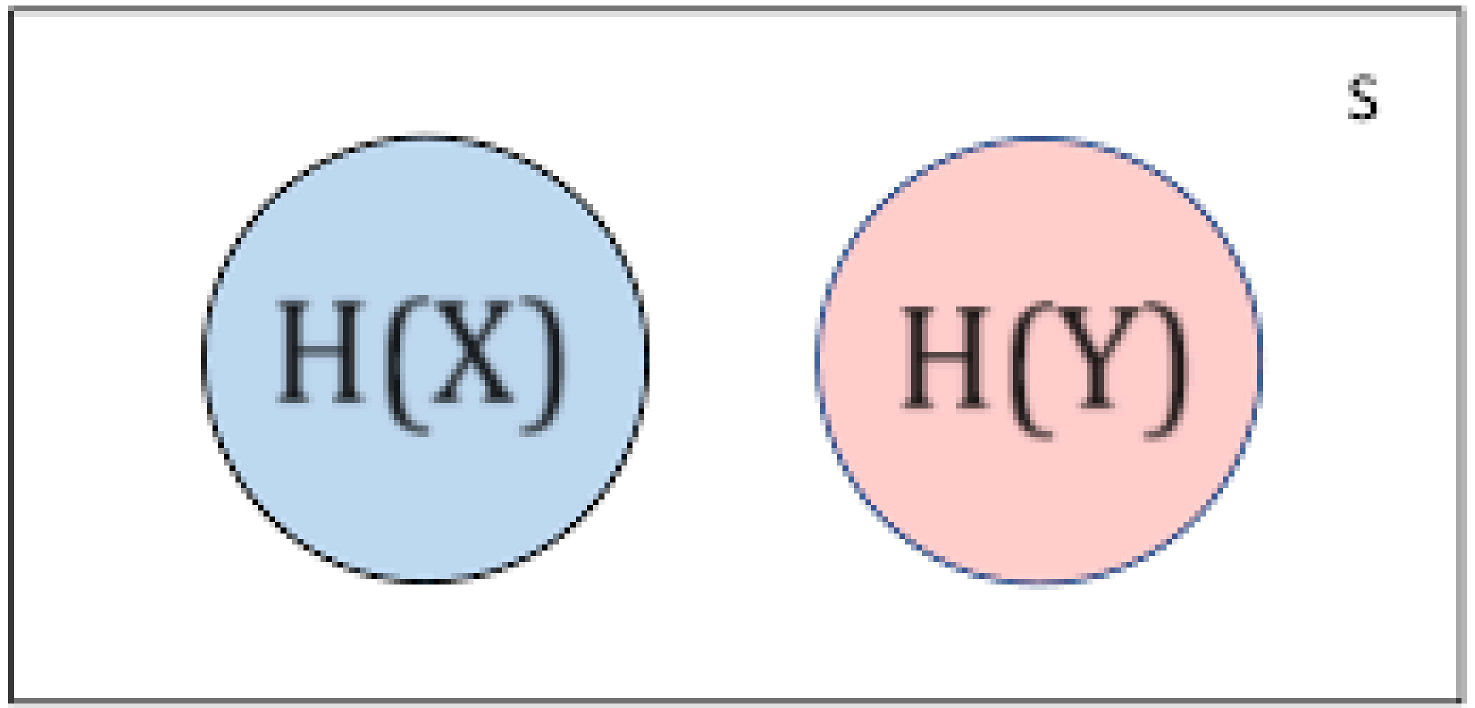
Mutual Information based Feature Selection

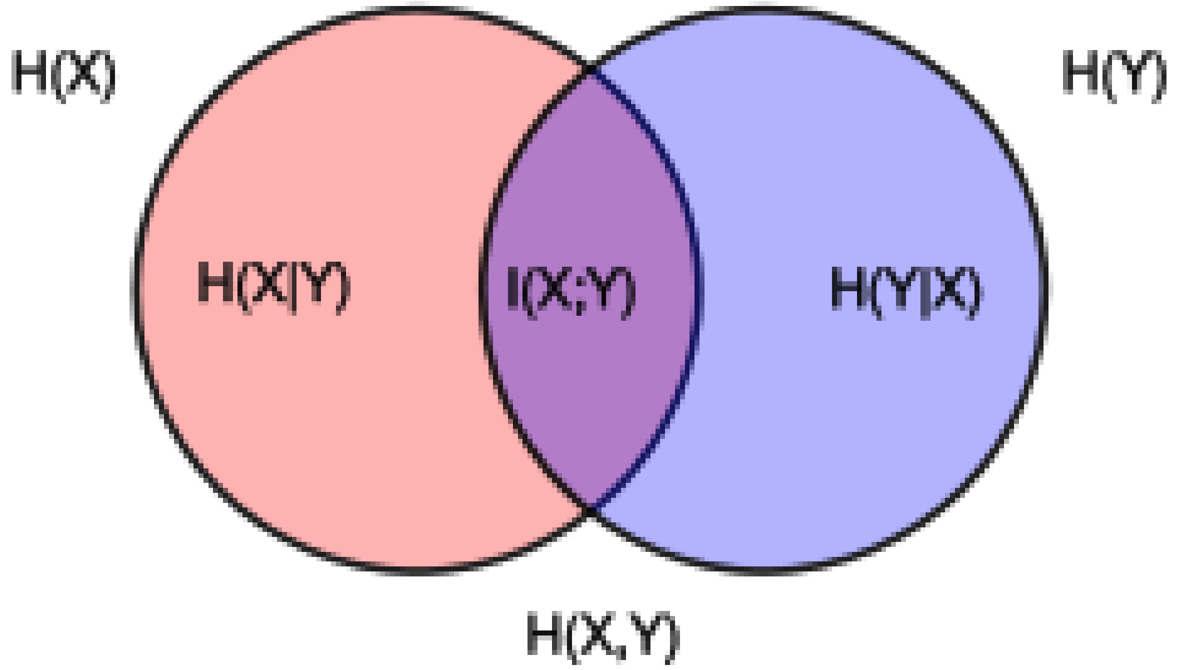
Mutual Information

- H(X,Y) = H(X) + H(Y)

- H(X,Y) = H(Y) + H(X|Y)
- I(X,Y) = = H(Y) – H(Y|X)

$$I(X,Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) log\left(\frac{p(x,y)}{p(x)p(y)}\right)$$





Mutual Information

- Both X and Y are categorical

$$I(X,Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) log\left(\frac{p(x,y)}{p(x)p(y)}\right)$$

- One of X or Y is numerical

$$I(X,Y) = \sum_{x \in X} \int_y p(x,y) log\left(\frac{p(x,y)}{p(x)p(y)}\right) \qquad I(X,Y) = \int_x \sum_{y \in Y} p(x,y) log\left(\frac{p(x,y)}{p(x)p(y)}\right)$$

- Both are numerical

$$I(X,Y) = \int_x \int_y p(x,y) log\left(\frac{p(x,y)}{p(x)p(y)}\right)$$

Mutual Information for categorical variables X and Y

- Titanic dataset
  - Sex & Survived (x & y)

$$I(X,Y) = \sum_{x\in X}\sum_{y \in Y} p(x,y) log\left(\frac{p(x,y)}{p(x)p(y)}\right)$$

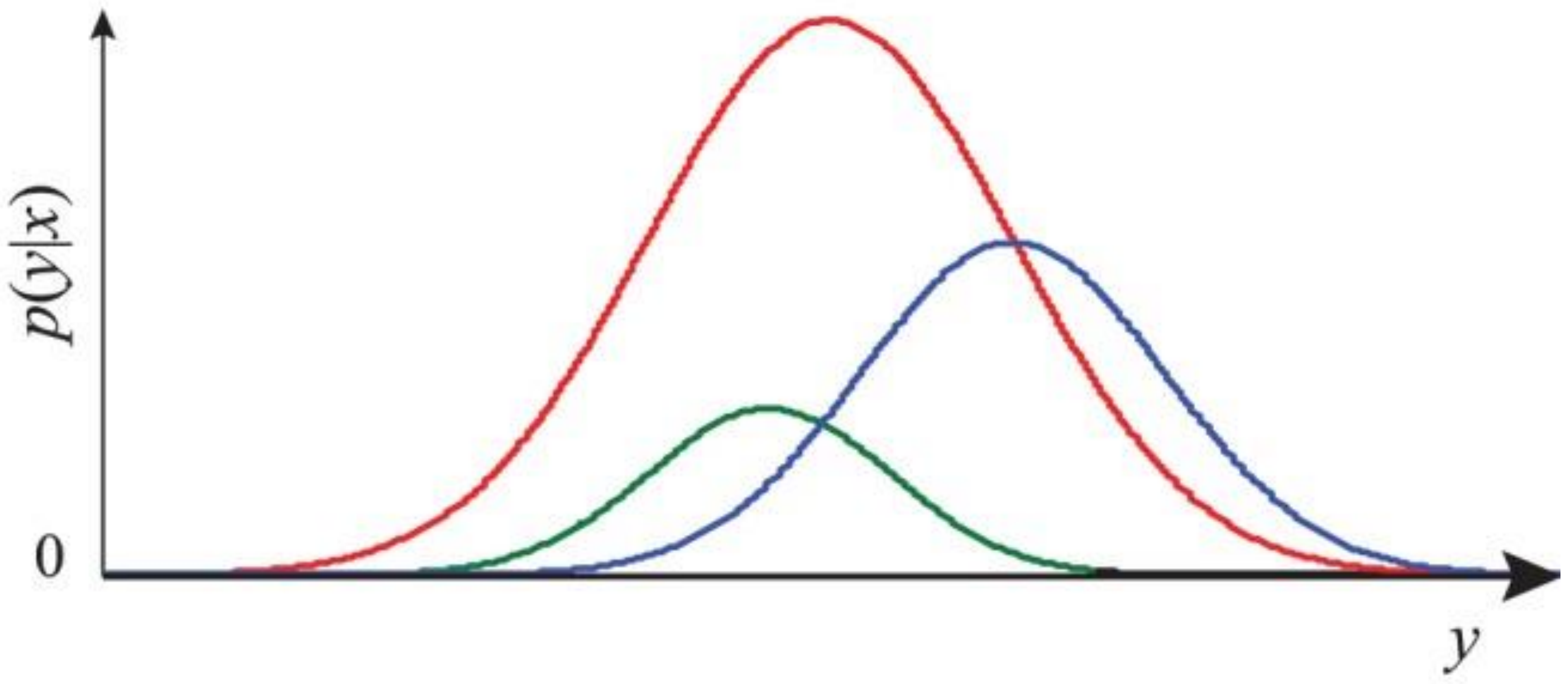
sex	female	male	
survived			
0	91	479	570
1	235	109	344
Total	326	588	914

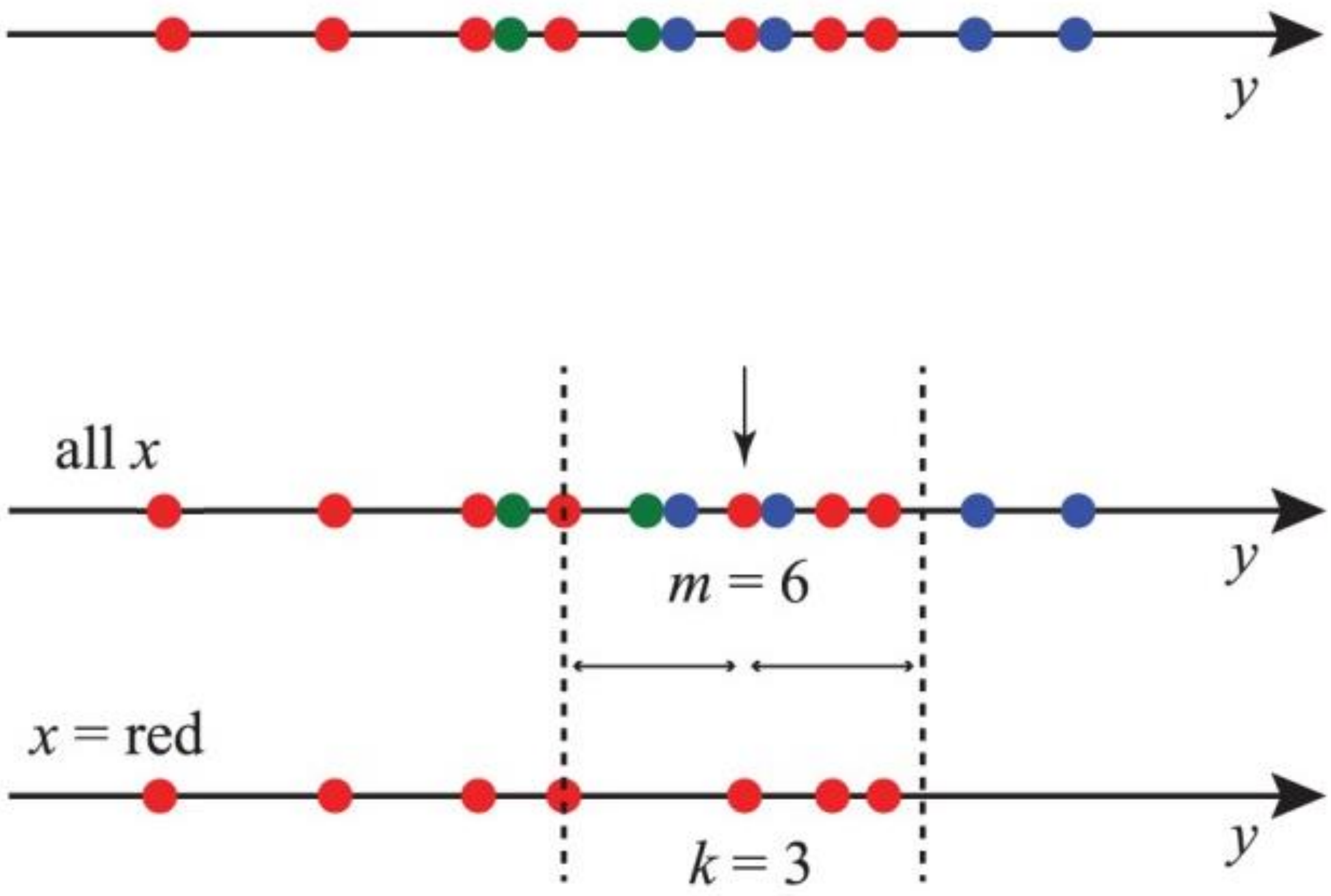
$$0.1 \times log\left(\frac{0.1}{0.62 \times 0.36}\right) + 0.52 \times log\left(\frac{0.52}{0.62 \times 0.64}\right) +$$
$$0.26 \times log\left(\frac{0.26}{0.38 \times 0.36}\right) + 0.12 \times log\left(\frac{0.12}{0.38 \times 0.64}\right)$$

	Female	Male	Total
Survived	0.1	0.52	0.62
Not Survived	0.26	0.12	0.38
Total	0.36	0.64	1

Mutual Information for categorical & numerical

- Binning
  - Different bins give different MI
  - Not a robust way for MI calc with numeric variables
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3929353/ (PLOS One 2014)





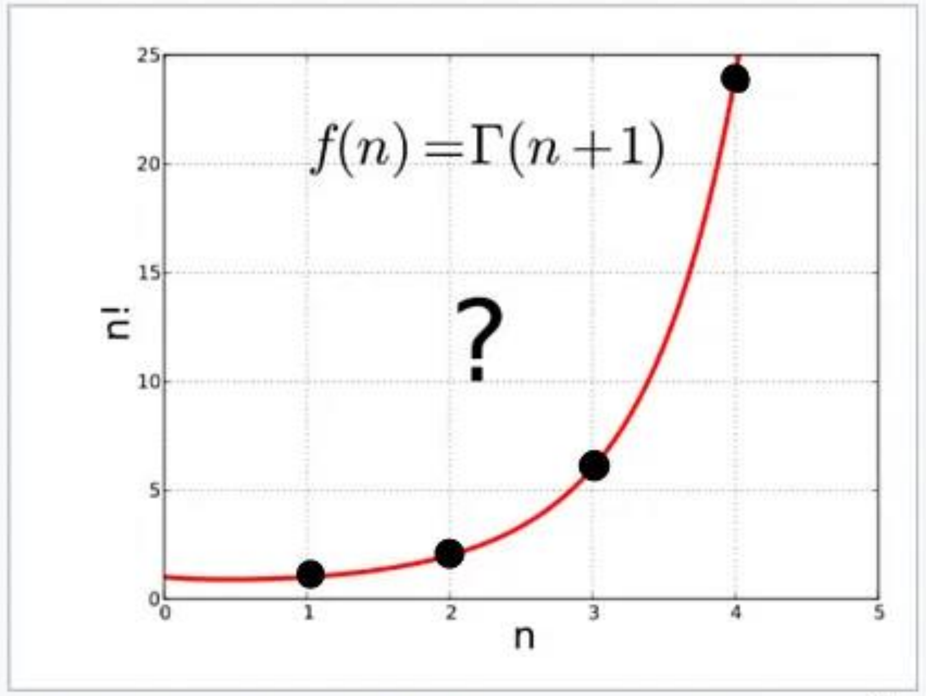
$$I_{x^{(i)}} = \varphi(N) - \varphi(N_{x^{(i)}}) + \varphi(k) - \varphi(m_i)$$

$$MI = \frac{1}{m} \sum_i I_{x^{(i)}}$$

- Nearest Neighbors
- K = Num Neighbors of same type
- M = Num entries to get k same type neighbors
- Nxi = Total number of entries of same type as xi
- N = Total entries in dataset
- Digamma function

Digamma function

- Digamma function = Derivative of log of gamma function
- Gamma function
  - Factorial of a non integer
- Used in many probability distributions
  - Gamma distribution
  - Beta distribution
  - T-distribution
  - Chi-squared distribution



The gamma function interpolates the factorial function to non-integer values.

The gamma function can be seen as a solution to the following interpolation problem:

> "Find a smooth curve that connects the points (x, y) given by $y = (x-1)!$ at the positive integer values for x."

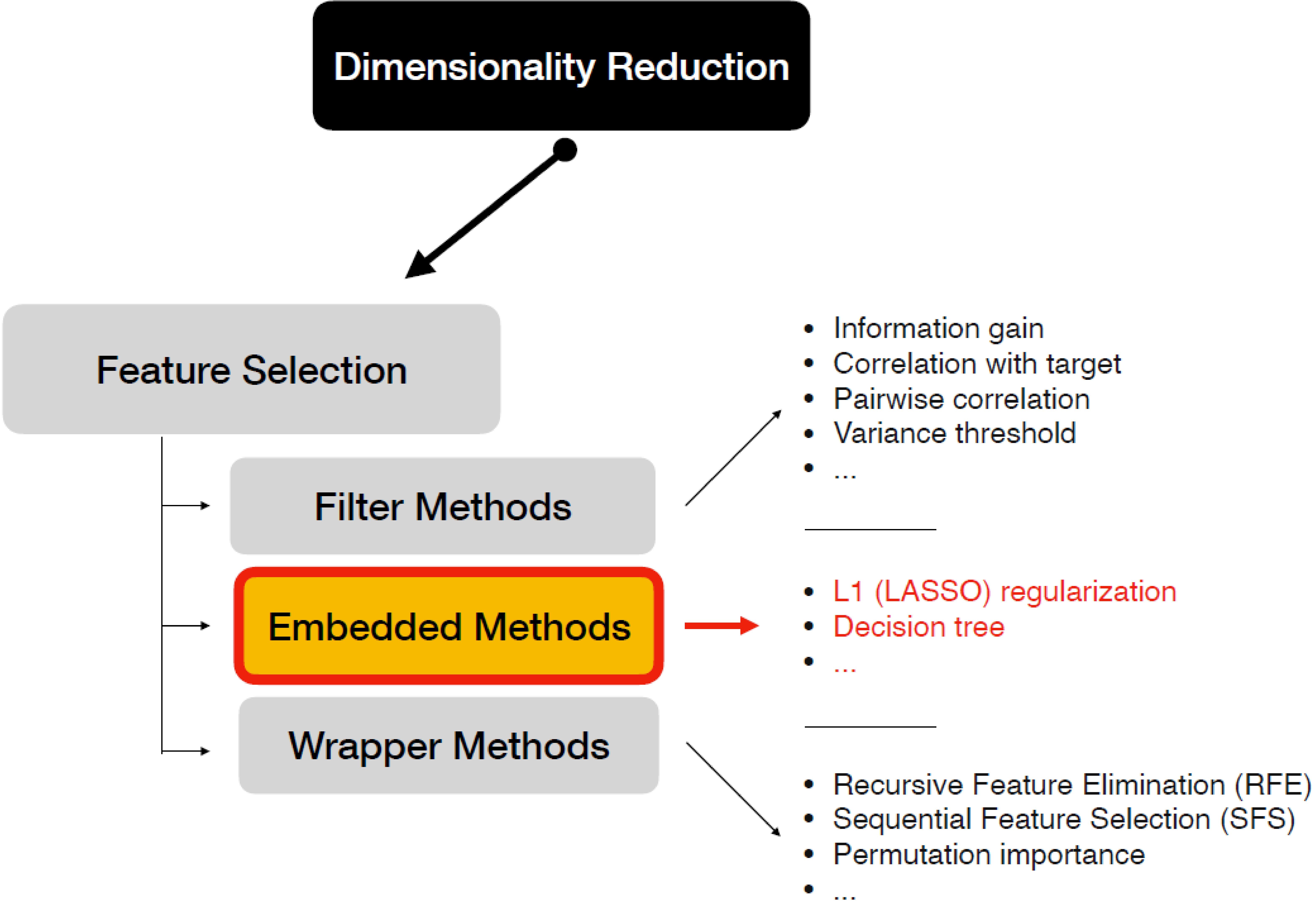
Sklearn implementation

- Categorical+numerical features & categorical target
  - SelectKBest(scoring=mutual_info_classif, ….)
- Categorical+numerical features & numerical target
  - SelectKBest(scoring=mutual_info_regression, ….)

Filter method with chi-squared & ANOVA will be covered in next lecture

Lots of statistical theory needed for implementation with 1-2 lines of code

Feature Selection with Embedded (Intrinsic) methods

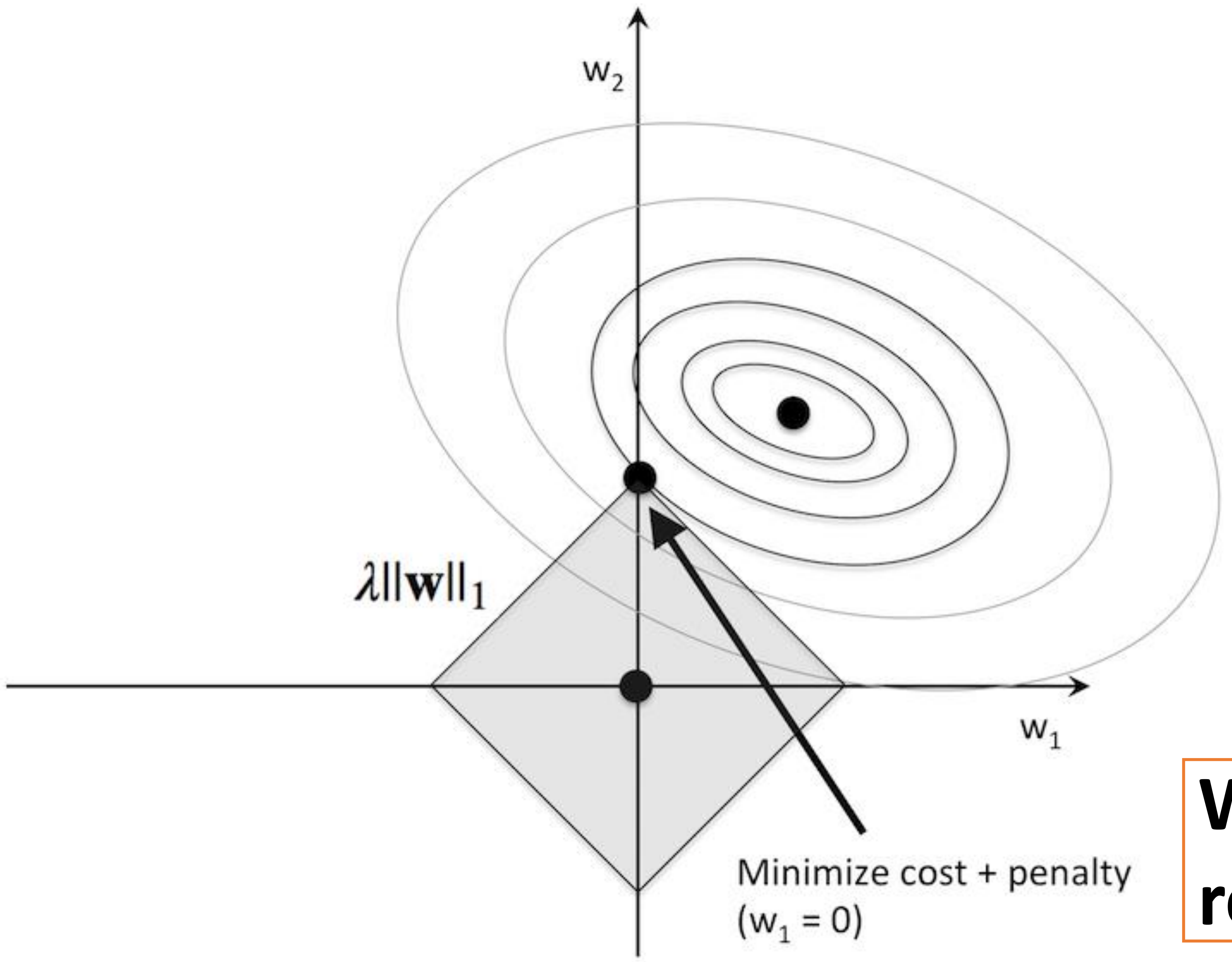

Dimensionality Reduction
Feature Selection
Filter Methods
Embedded Methods
Wrapper Methods
• Information gain
• Correlation with target
• Pairwise correlation
• Variance threshold
• ...
• L1 (LASSO) regularization
• Decision tree
• ...
• Recursive Feature Elimination (RFE)
• Sequential Feature Selection (SFS)
• Permutation importance
• ...

Embedded methods

- Always supervised
- No separate feature selection
- Feature Selection happens as part of model training
- E.g.:
  - LASSO
  - Feature Importance with Random Forest
- Returns coeff_ or feature_importances
- Other non parametric methods do not augur well

Feature Selection with LASSO

Cost function adjusted for L1 Regularization



$$\arg\min_{w} \nabla_w \mathcal{J} + \lambda \nabla_w \|w\|_1$$

$$\nabla_w \mathcal{J} = \frac{2}{m} X^T (Xw - y) \qquad \nabla_w \|w\|_1 = \mathbf{1}$$

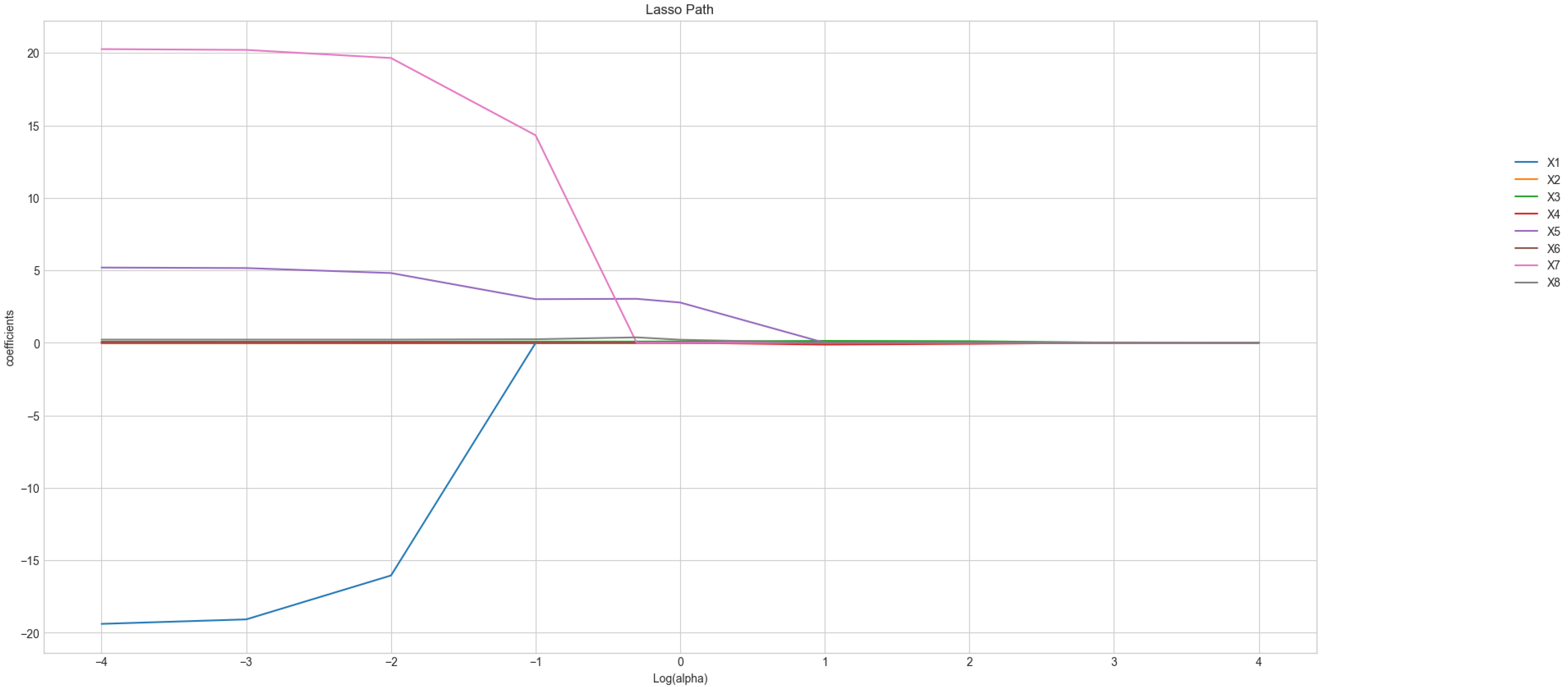
$$\mathbf{w} = \mathbf{w} - \eta \nabla_w \mathcal{J} \qquad \mathbf{w} = \mathbf{w} - \eta \nabla_w \mathcal{J} - \eta\lambda$$

Without regularization

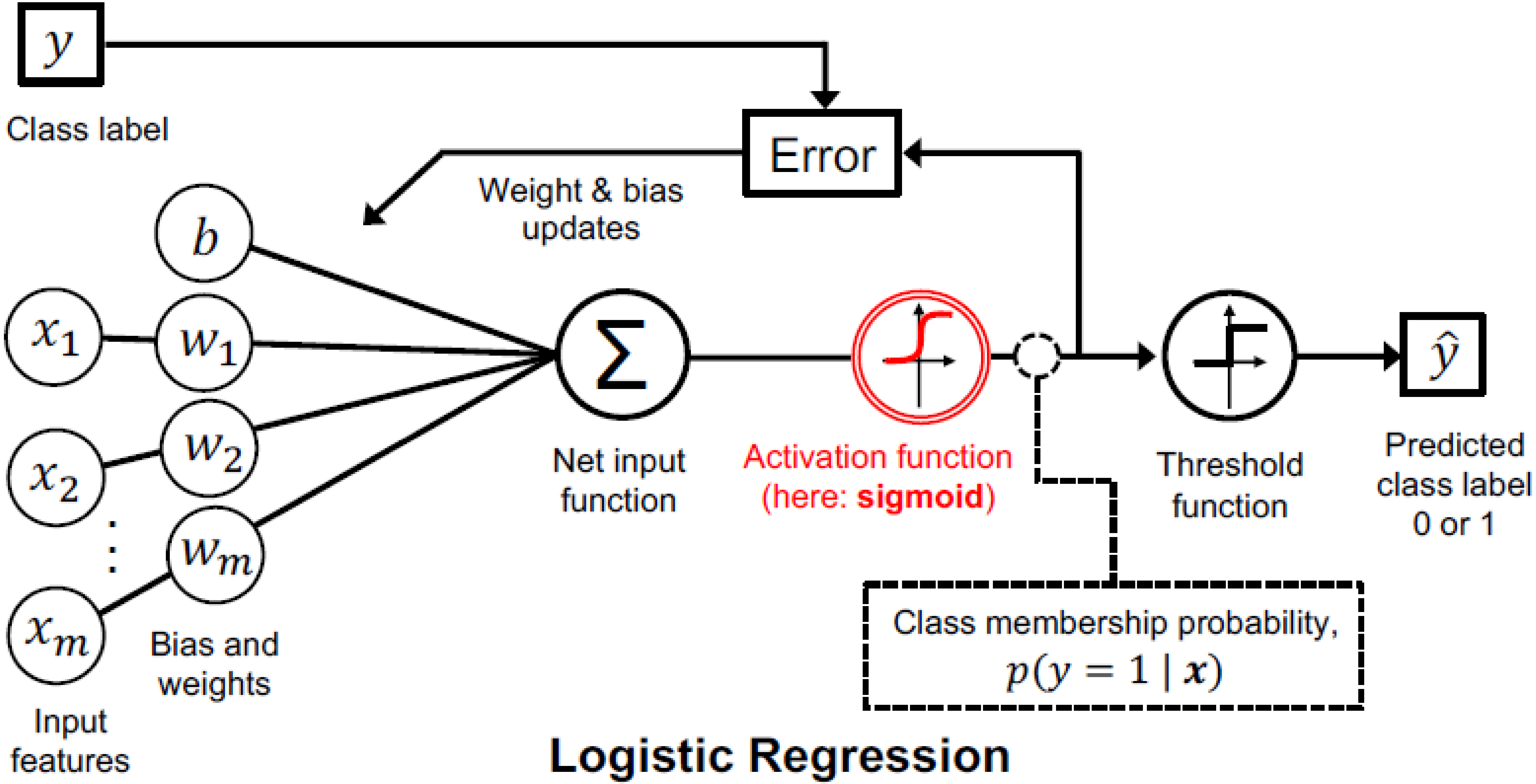
$$\mathbf{w} = (\mathbf{w} - \eta\lambda) - \eta \nabla_w \mathcal{J}$$

A FIXED small number keeps getting subtracting from a small w. Net effect w becomes 0

Lasso path



$$\arg\min_w \nabla_w \mathcal{J} + \lambda \nabla_w \|w\|_1$$



Logistic Regression

$$-\sum_{i=1}\left[y^{(i)}\log\left(\sigma\left(z^{(i)}\right)\right)+\left(1-y^{(i)}\right)\log\left(1-\sigma\left(z^{(i)}\right)\right)\right] + \lambda\|w\|_1$$


QUESTIONS