Lecture 13

Dimensionality Reduction I: Feature Selection

STAT 451: Intro to Machine Learning, Fall 2021

Sebastian Raschka

https://sebastianraschka.com/teaching/stat451-fs2021/

Dimensionality Reduction

Why do we care?

Dimensionality Reduction

Why do we care?

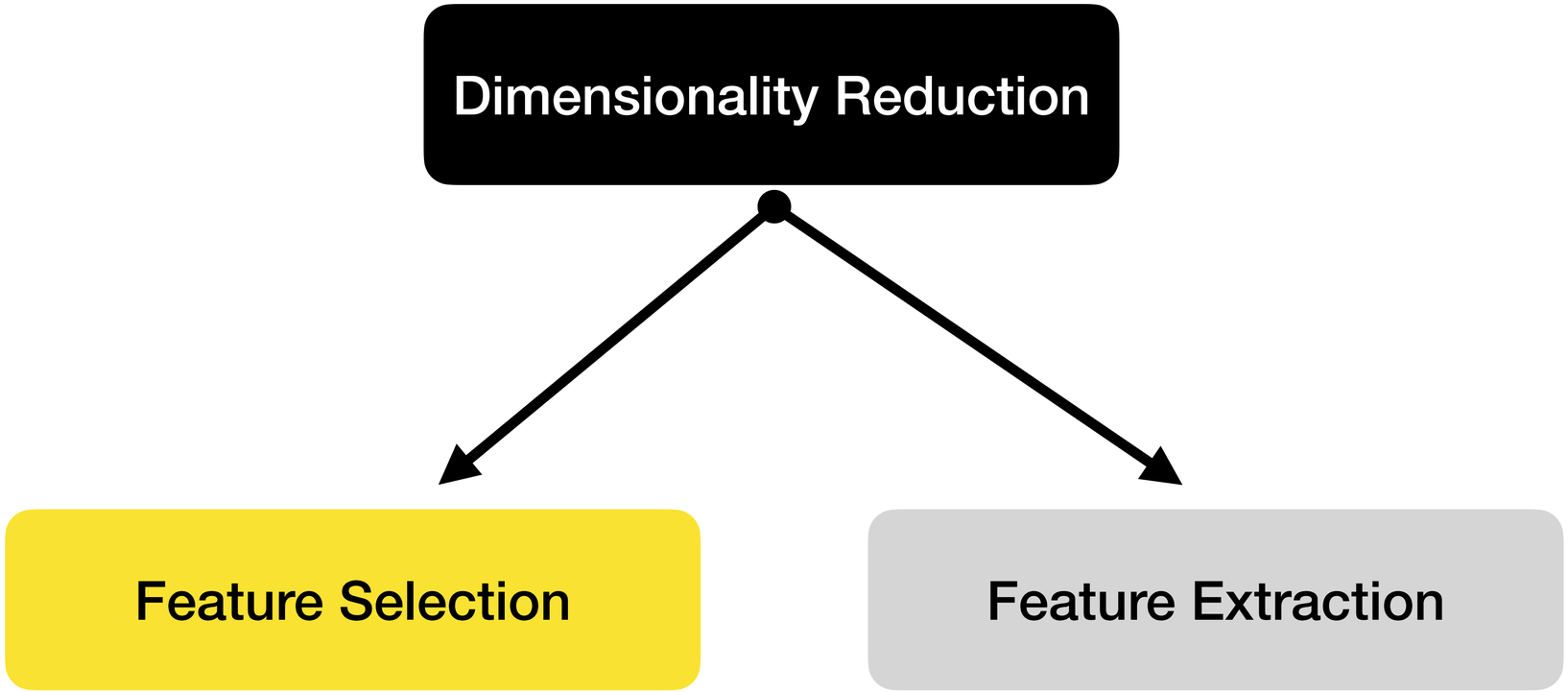
Curse of dimensionality

Computational efficiency

Easier data collection

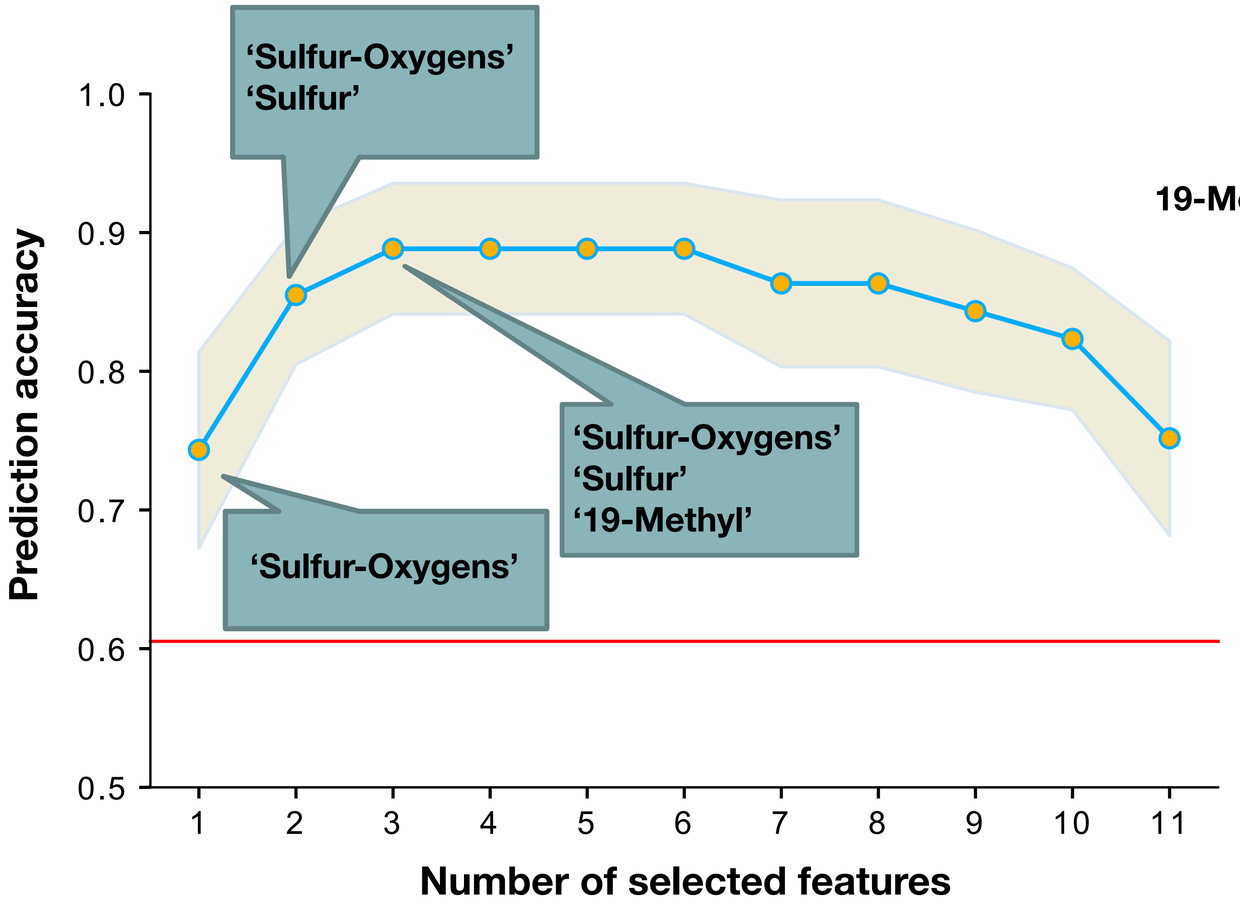
Storage space

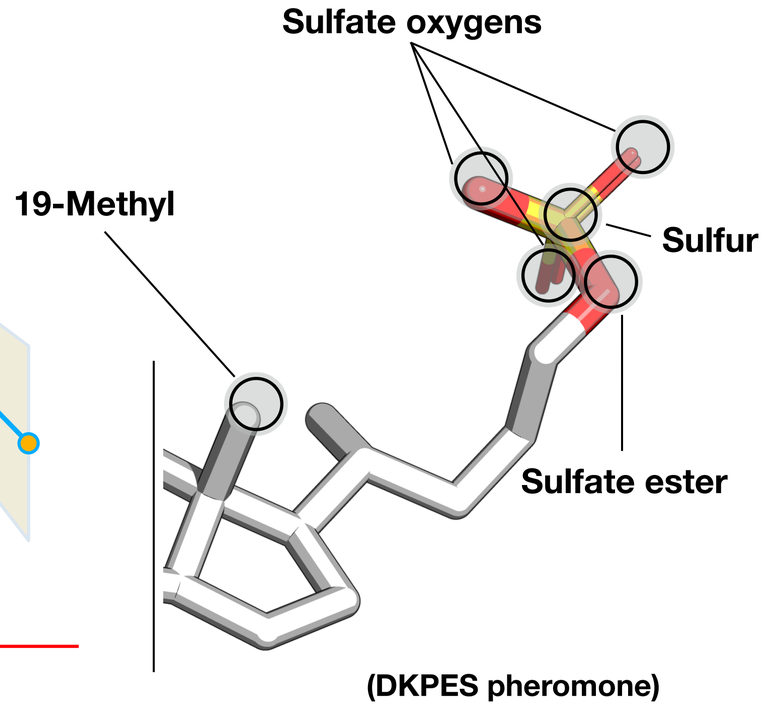
Interpretability


Dimensionality Reduction
Feature Selection
Feature Extraction
Today!
Next lecture

Discovery of a pheromone receptor inhibitor for invasive species control (sea lamprey) in the Great Lakes

https://en.wikipedia.org/wiki/Sea_lamprey#/media/File:Sea_lamprey_on_brown_trout_flipped.jpg

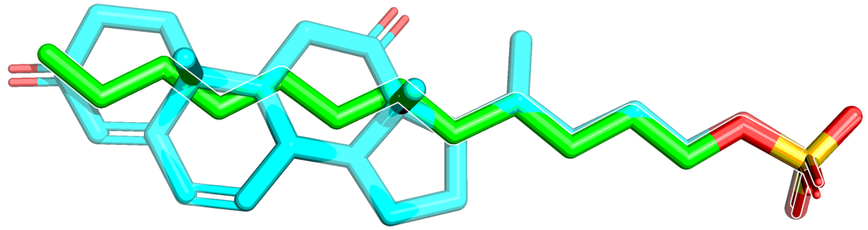




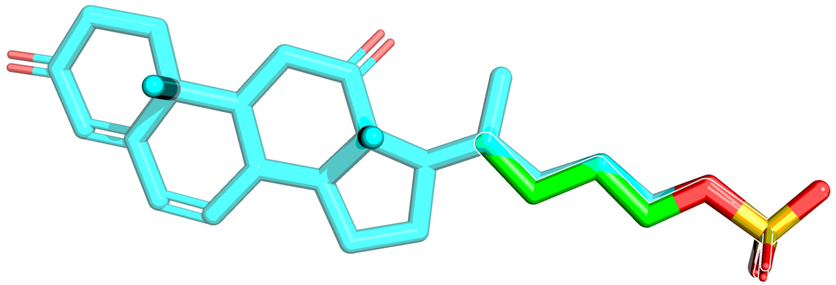
22

Sebastian Raschka, Nan Liu, Santosh Gunturu, Anne M. Scott, Mar Huertas, Weiming Li, and Leslie A. Kuhn (2018) *Facilitating the Hypothesis-driven Prioritization of Small Molecules in Large Databases: Screenlamp and its Application to GPCR Inhibitor Discovery.* Journal of Computer-Aided Molecular Design, 32(3), 415-433.

"Sulfate-tail" sufficient for bioactivity



69% signal inhibition

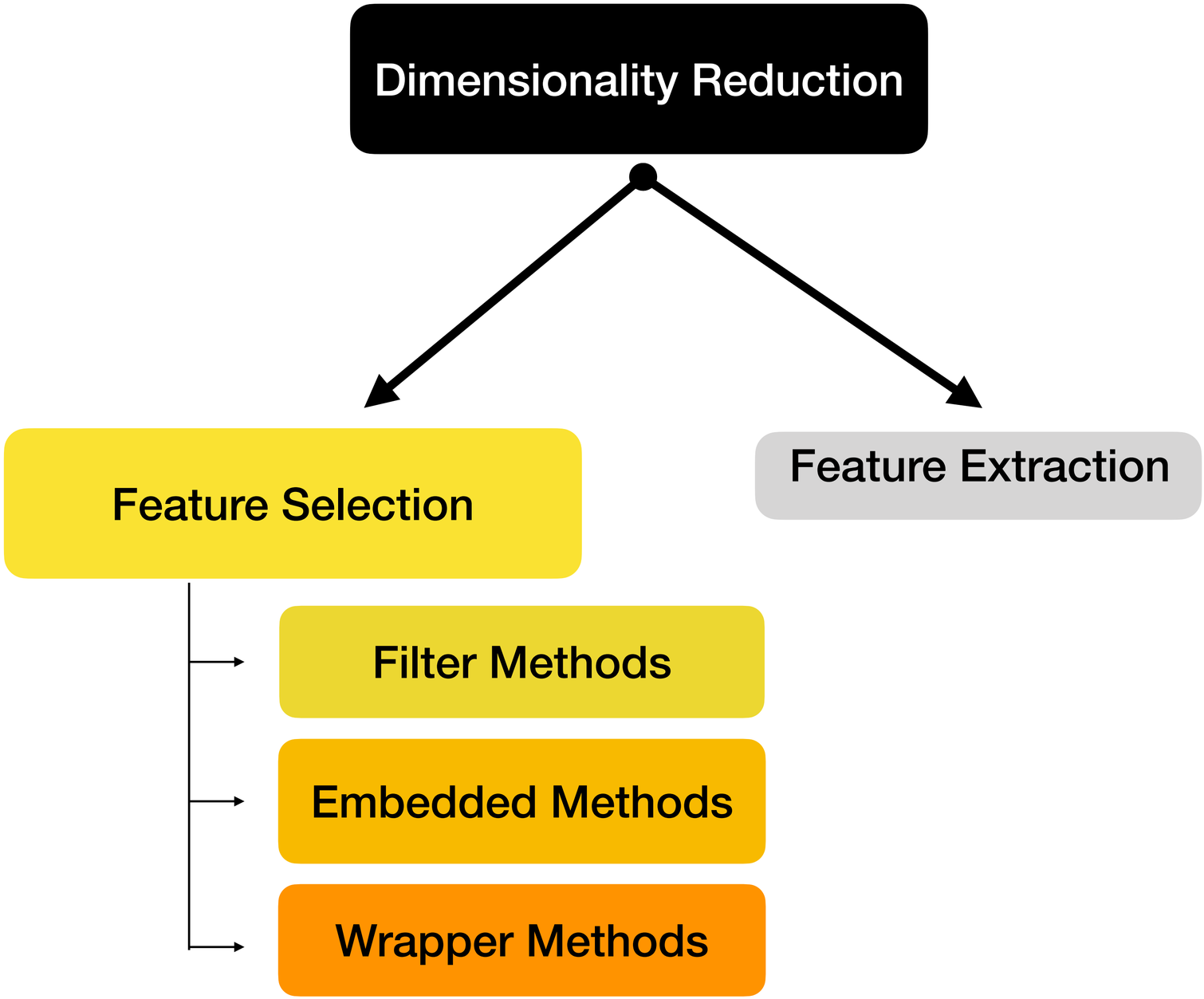


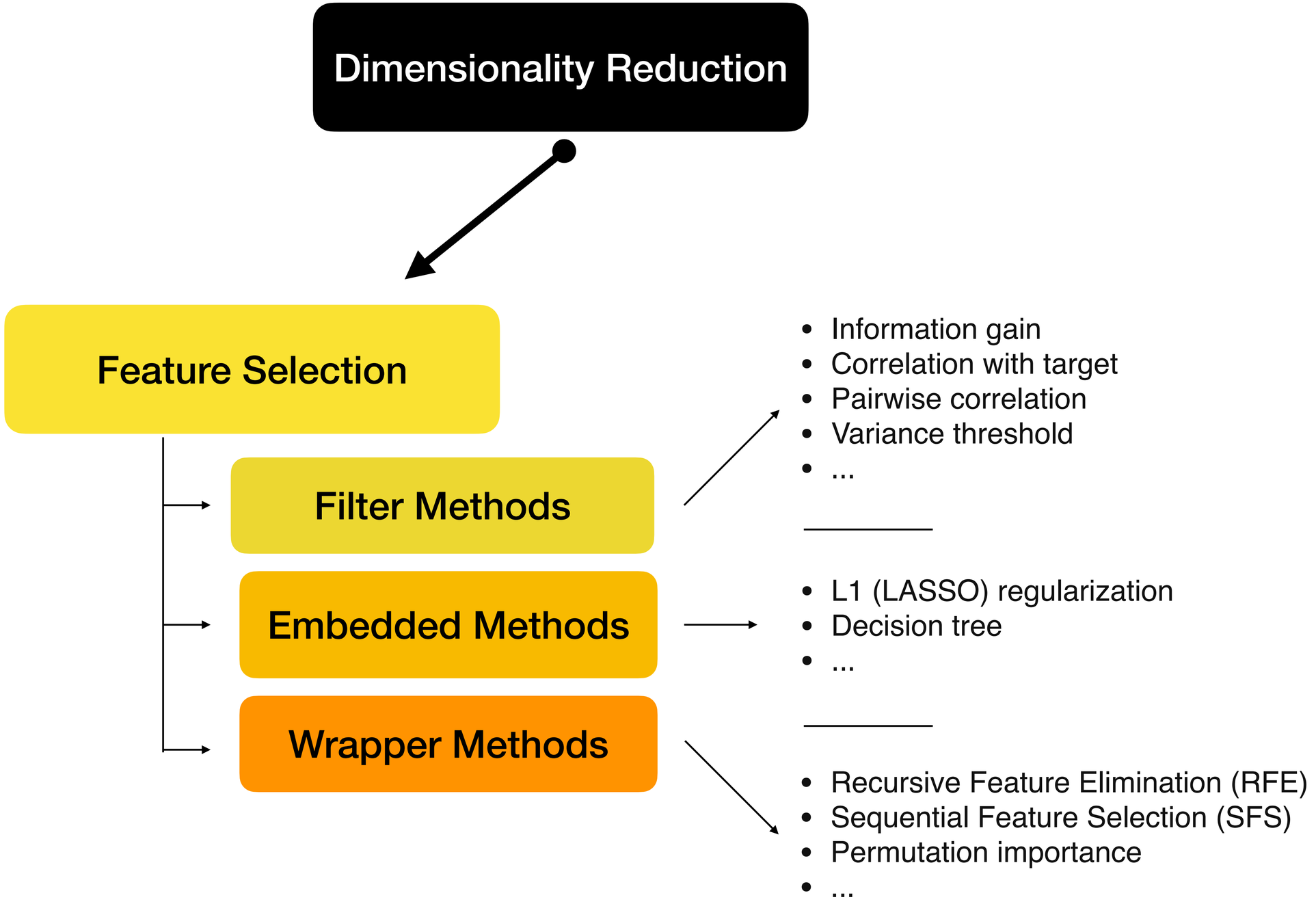
62% signal inhibition

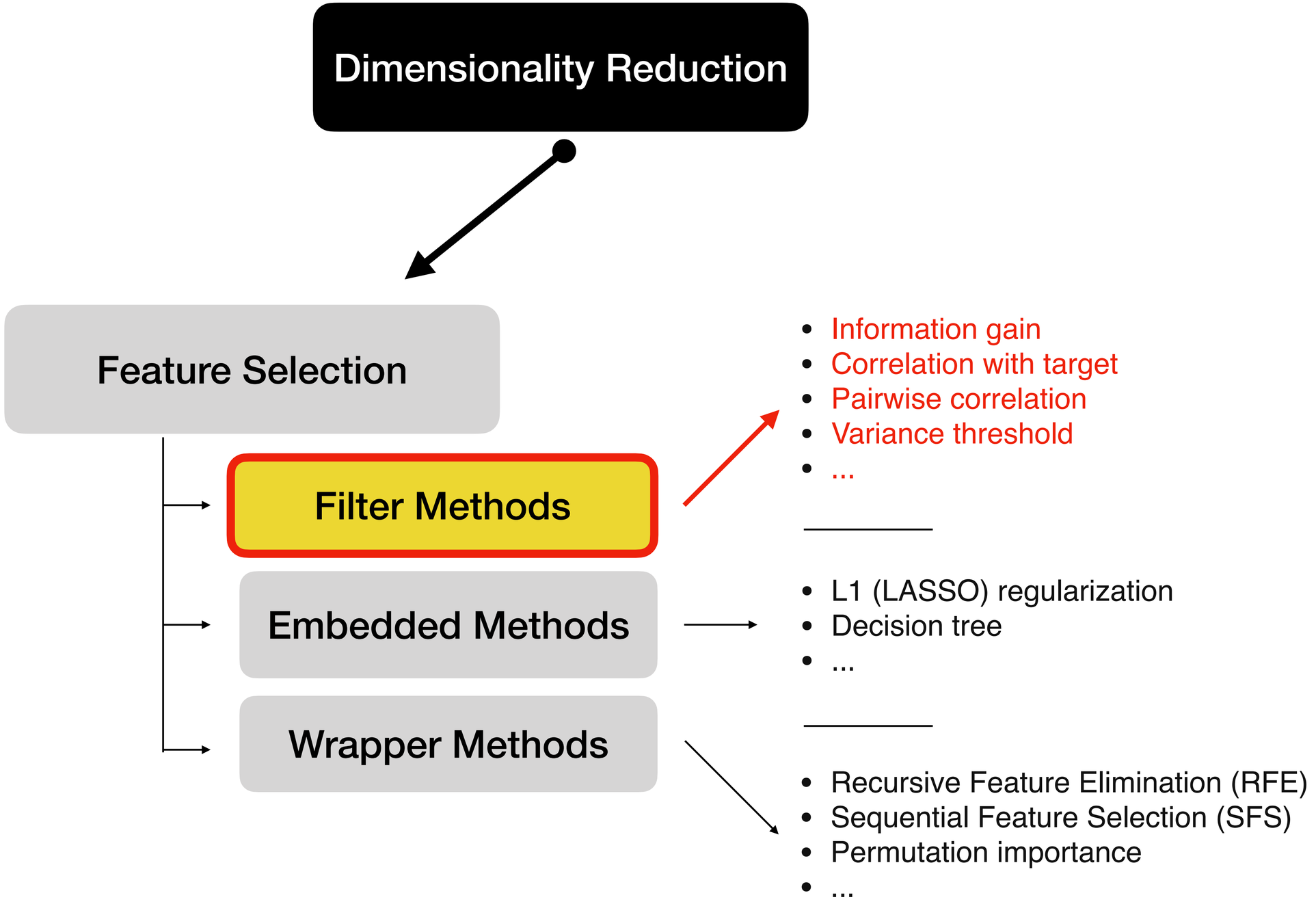
23

1. Different categories of feature selection
2. Filter methods
3. Embedded methods
    3.1. L1-regularized logistic regression
    3.2. Random forest feature importance
4. Wrapper methods
    4.1. Recursive feature elimination
    4.2. Permutation importance
    4.3. Permutation importance code example
    4.4. Sequential feature selection
    4.5. Sequential feature selection code example

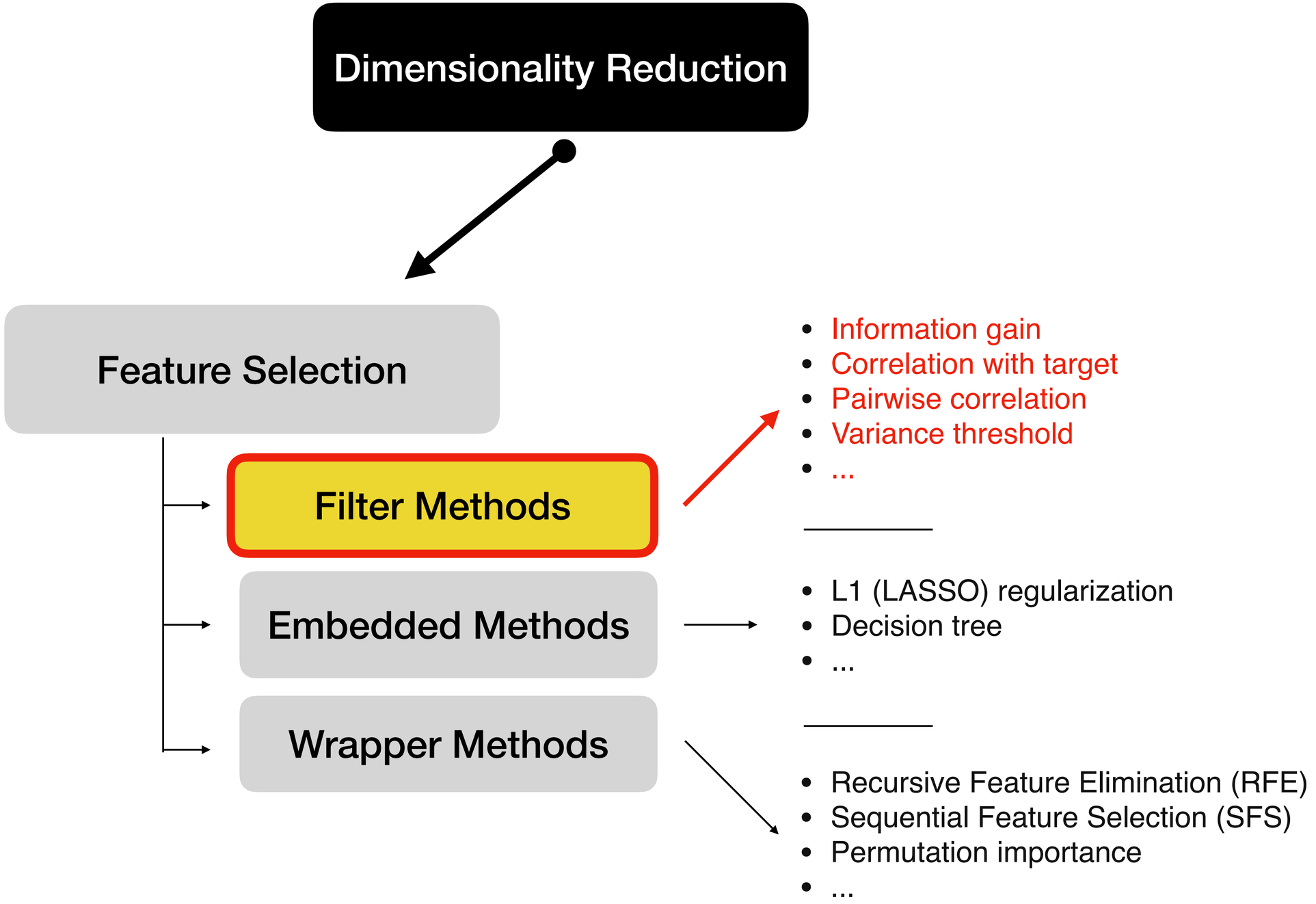
1. **Different categories of feature selection**
2. Filter methods
3. Embedded methods
    3.1. L1-regularized logistic regression
    3.2. Random forest feature importance
4. Wrapper methods
    4.1. Recursive feature elimination
    4.2. Permutation importance
    4.3. Permutation importance code example
    4.4. Sequential feature selection
    4.5. Sequential feature selection code example


Dimensionality Reduction
Feature Selection
Feature Extraction
Filter Methods
Embedded Methods
Wrapper 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
• ...

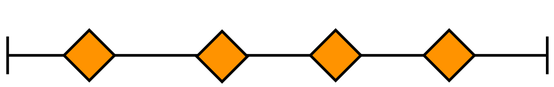

Dimensionality Reduction
Feature Selection
Filter Methods
Embedded Methods
Wrapper Methods
Information gain
Correlation with target
Pairwise correlation
Variance threshold
...
L1 (LASSO) regularization
Decision tree
...
Recursive Feature Elimination (RFE)
Sequential Feature Selection (SFS)
Permutation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

1. Different categories of feature selection
2. **Filter methods**
3. Embedded methods
    - 3.1. L1-regularized logistic regression
    - 3.2. Random forest feature importance
4. Wrapper methods
    - 4.1. Recursive feature elimination
    - 4.2. Permutation importance
    - 4.3. Permutation importance code example
    - 4.4. Sequential feature selection
    - 4.5. Sequential feature selection code example

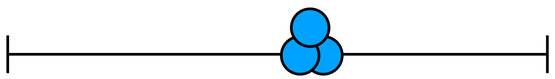

Dimensionality Reduction
Feature Selection
Filter Methods
Embedded Methods
Wrapper Methods
Information gain
Correlation with target
Pairwise correlation
Variance threshold
...
L1 (LASSO) regularization
Decision tree
...
Recursive Feature Elimination (RFE)
Sequential Feature Selection (SFS)
Permutation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

Variance Threshold (Filter)

- Compute the variance of each feature
- Assume that features with a higher variance may contain more useful information



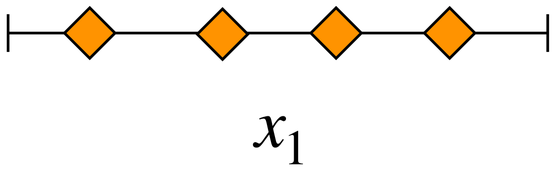
x_1

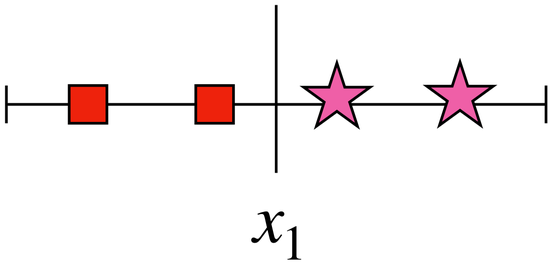


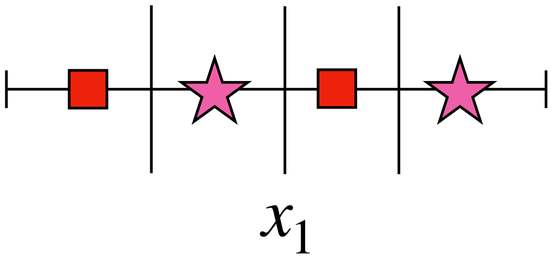
x_2

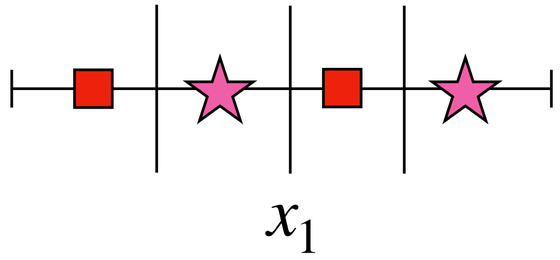
Variance Threshold (Filter)

- Compute the variance of each feature
- Assume that features with a higher variance may contain more useful information





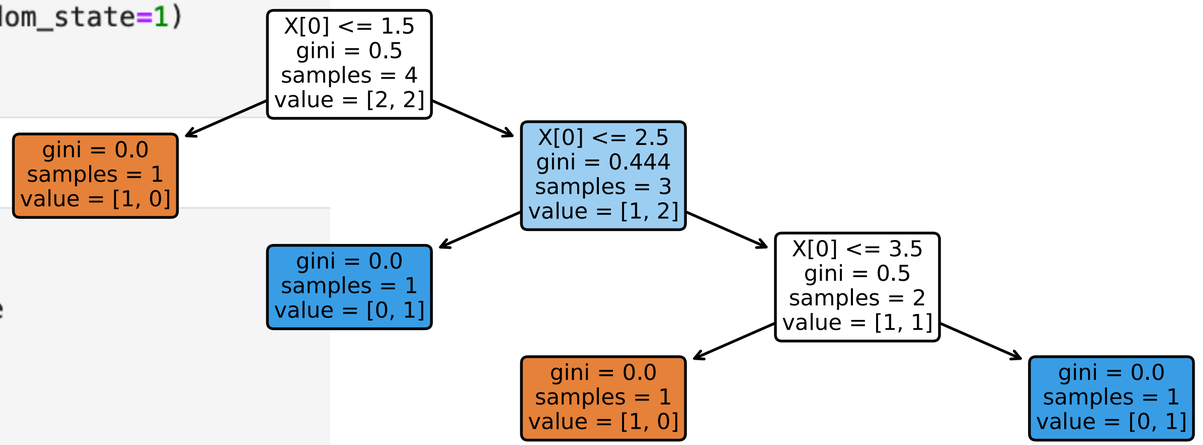





```
import numpy as np
from sklearn.tree import DecisionTreeClassifier

X = np.array([[1.], [2.], [3.], [4.]])
y = np.array([0, 1, 0, 1])


tree = DecisionTreeClassifier(random_state=1)
tree.fit(X, y)
tree.score(X, y)
```

```
1.0
```

```
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree


plt.figure(figsize=(10, 3))

plot_tree(tree,
          filled=True,
          rounded=True)

plt.show()
```



Variance

Variance of discrete random variable:

$$\text{Var}(X) = \sum_{i=1}^{n} p_i \cdot \left(x_i - \mu\right)^2$$

E.g., dataset with n datapoints (for sample variance, *n-1*)

$$\text{Var}(X) = \frac{1}{n} \sum_{i=1}^{n} \left(x_i - \mu\right)^2$$

E.g., dataset with n datapoints (for sample variance, *n-1*)

$$\text{Var}(X) = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - \mu\right)^2$$

Variance of Bernoulli variable (Boolean feature, e.g., after one-hot encoding)

$$\text{Var}(X) = p(1-p)$$

```
data_var = np.var(50*[0] + 50*[1]) # i.e., p = 0.5
data_var
```

```
0.25
```

```
0.5 * 0.5
```

```
0.25
```

More Filter Methods

`sklearn.feature_selection`: Feature Selection

The `sklearn.feature_selection` module implements feature selection algorithms. It currently includes univariate filter selection methods and the recursive feature elimination algorithm.

User guide: See the Feature selection section for further details.

`feature_selection.GenericUnivariateSelect`([...])	Univariate feature selector with configurable strategy.
`feature_selection.SelectPercentile`([...])	Select features according to a percentile of the highest scores.
`feature_selection.VarianceThreshold`([threshold])	Feature selector that removes all low-variance features.
`feature_selection.chi2`(X, y)	Compute chi-squared stats between each non-negative feature and class.
`feature_selection.f_classif`(X, y)	Compute the ANOVA F-value for the provided sample.
`feature_selection.f_regression`(X, y, *[, center])	Univariate linear regression tests returning F-statistic and p-values.
`feature_selection.r_regression`(X, y, *[, center])	Compute Pearson's r for each features and the target.
`feature_selection.mutual_info_classif`(X, y, *)	Estimate mutual information for a discrete target variable.
`feature_selection.mutual_info_regression`(X, y, *)	Estimate mutual information for a continuous target variable.

https://scikit-learn.org/stable/modules/classes.html?highlight=feature%20selection#module-sklearn.feature_selection

VarianceThreshold $0.8 \times (1 - 0.8) = 0.16$

```
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()
```

```
array([[0., 0.],
       [1., 0.],
       [0., 0.],
       [0., 0.],
       [1., 0.],
       [0., 1.],
       [0., 0.],
       [1., 0.]])
```

```
from sklearn.feature_selection import VarianceThreshold

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

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

```
array([[0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.]])
```

Be aware of feature scaling!

E.g., dataset with n datapoints (for sample variance, *n-1*)

$$\text{Var}(X) = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - \mu\right)^2$$

```
np.random.seed(123)

data = np.random.random_sample(100)
np.var(data)
```

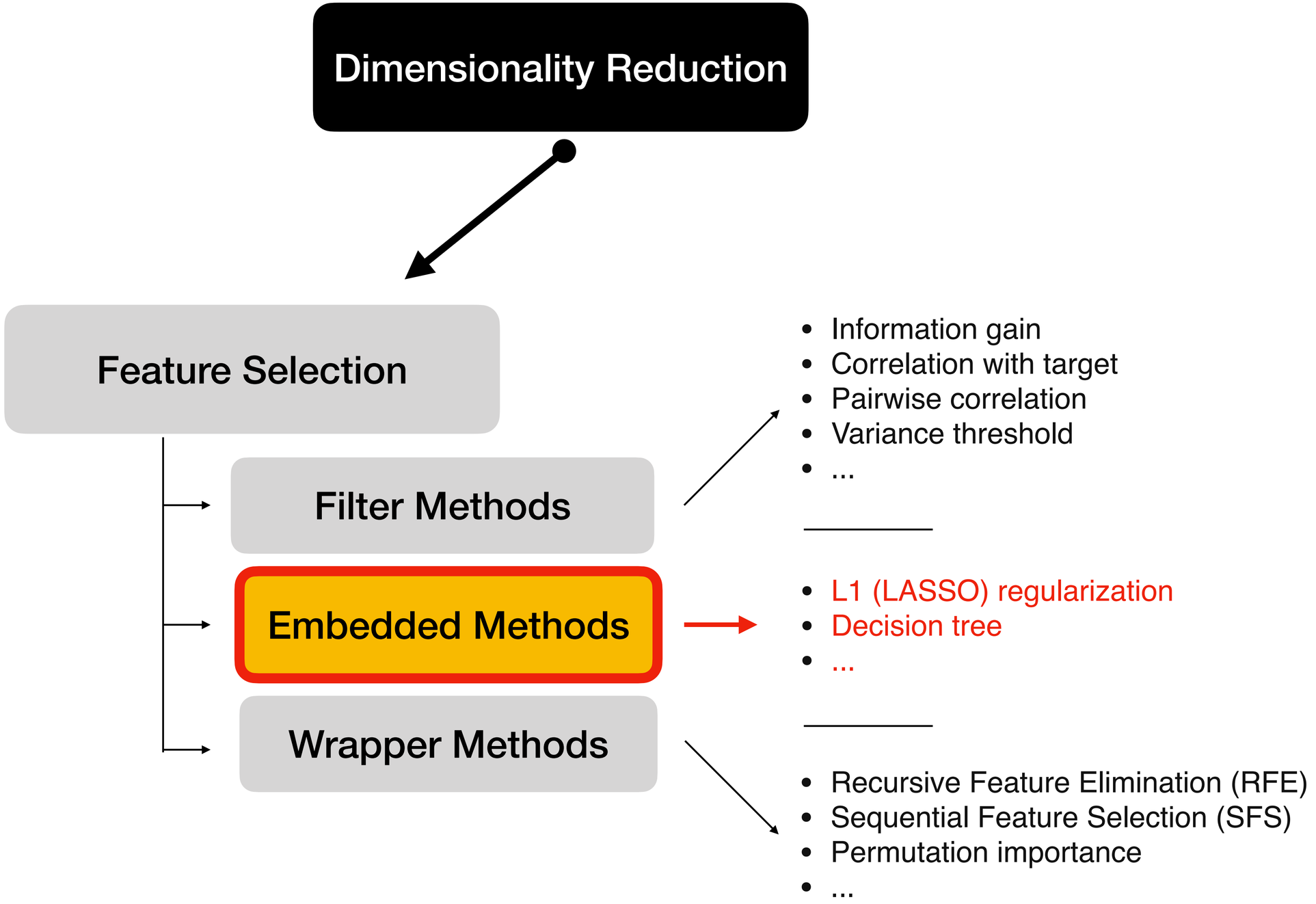
```
0.060211775685055 76
```

```
np.var(data*10)
```

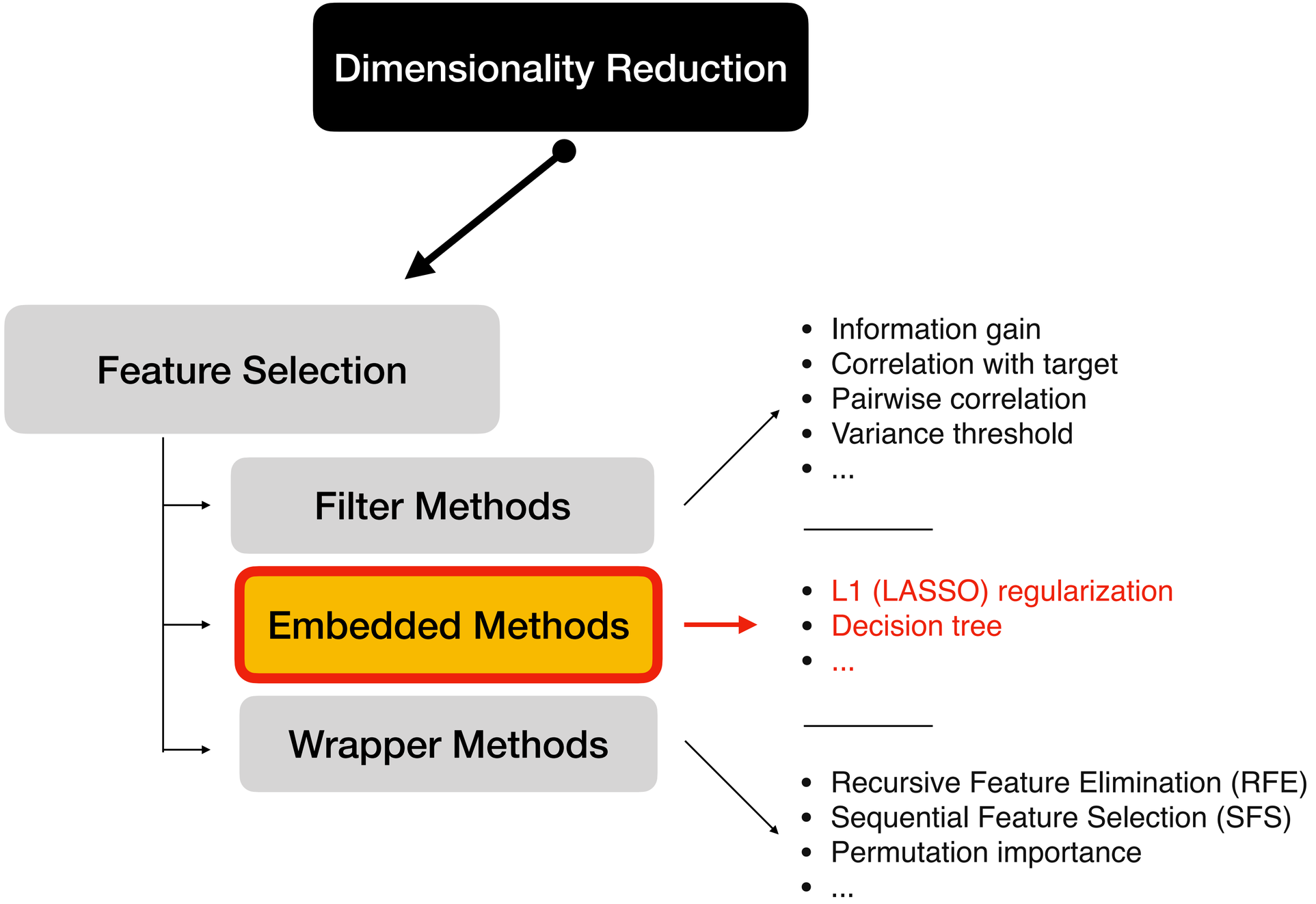
```
6.021177568505576
```

Variance Threshold (Filter)

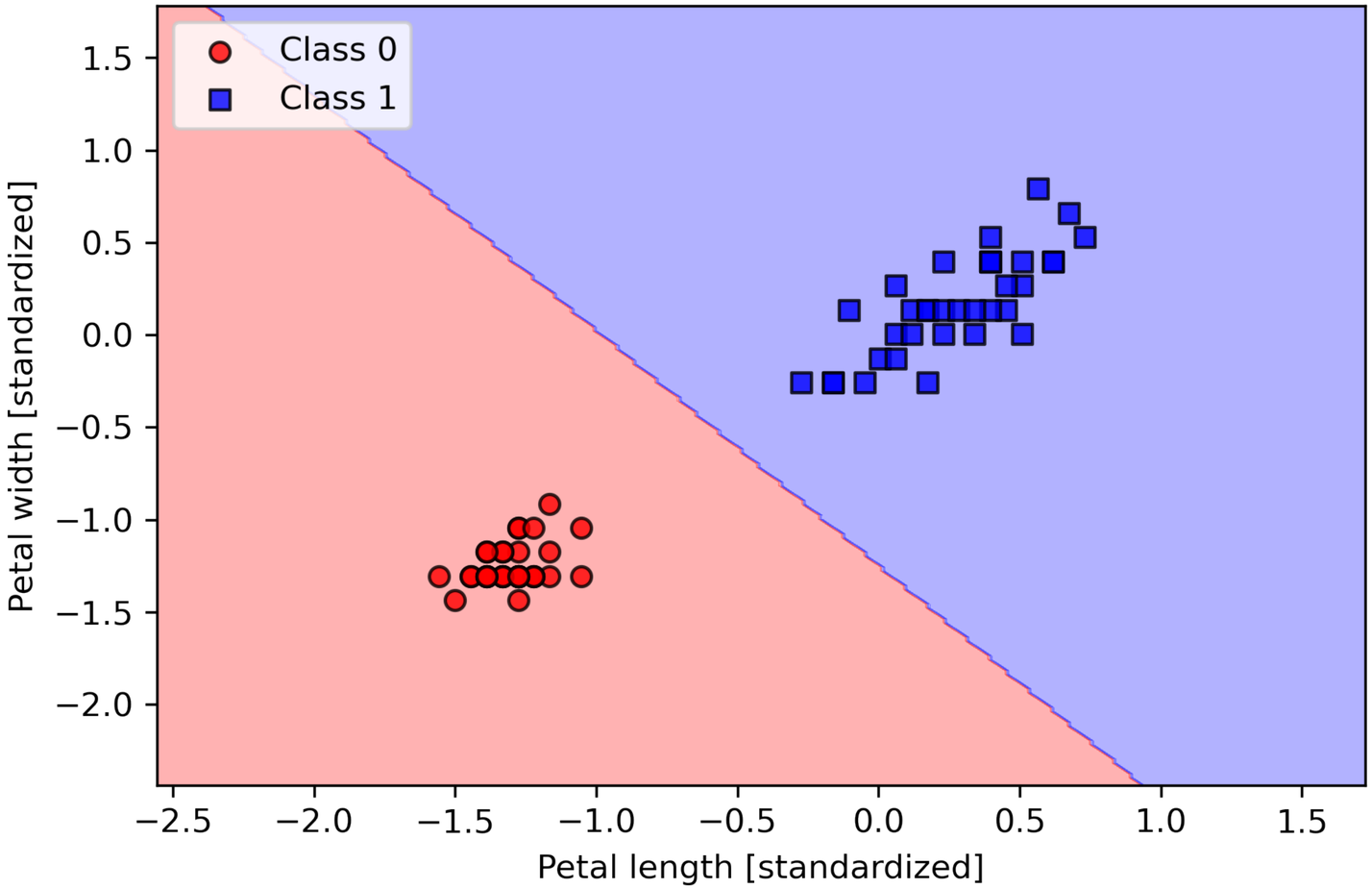
- Compute the variance of each feature
- Assume that features with a higher variance may contain more useful information
- Select the subset of features based on a user-specified threshold ("keep if greater or equal to *x*" or "keep the the top *k* features with largest variance")
- **Good:** fast!
- **Bad:** does not take the relationship among features into account


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
...

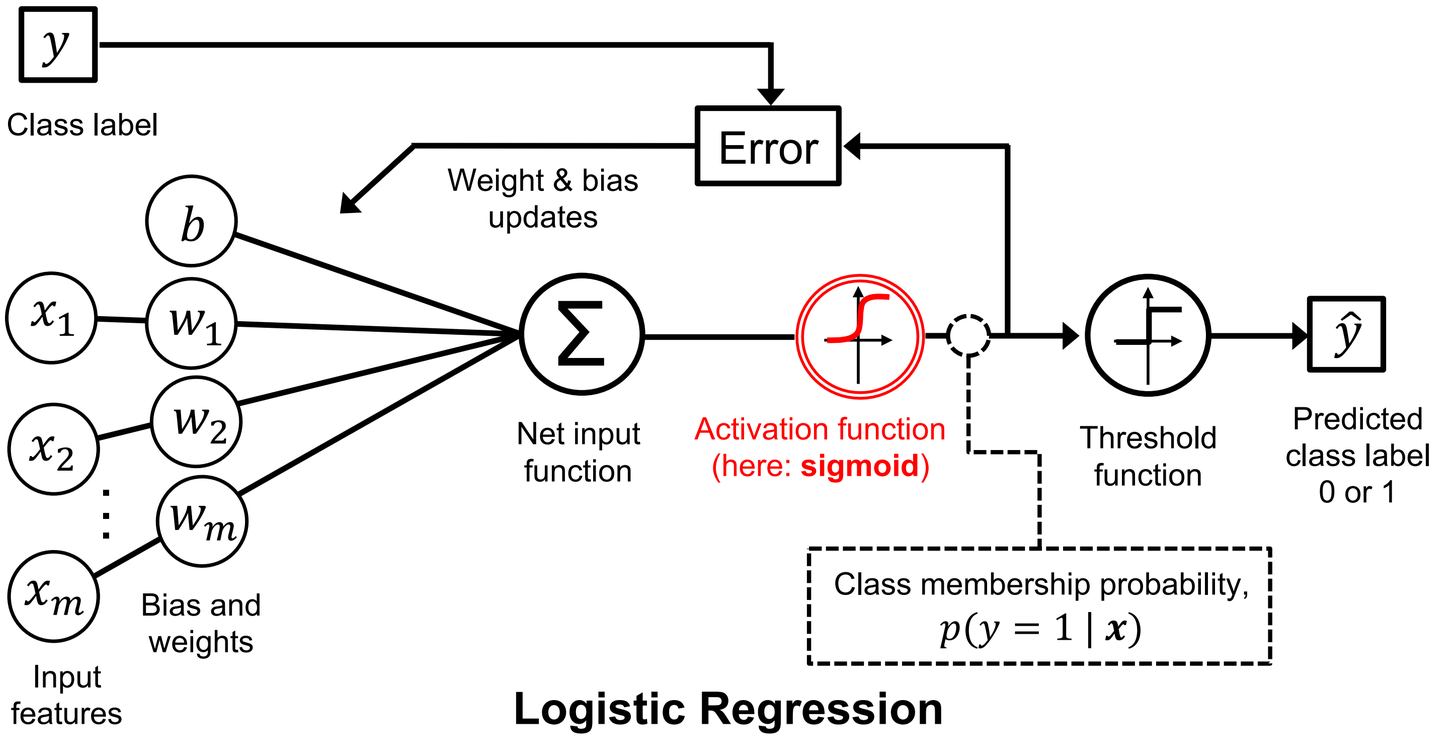
1. Different categories of feature selection
2. Filter methods
3. **Embedded methods**
    - **3.1. L1-regularized logistic regression**
    - 3.2. Random forest feature importance
4. Wrapper methods
    - 4.1. Recursive feature elimination
    - 4.2. Permutation importance
    - 4.3. Permutation importance code example
    - 4.4. Sequential feature selection
    - 4.5. Sequential feature selection code example


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
...

Logistic Regression



Logistic Regression



Source: Raschka, Liu, and Mirjalili. M*achine Learning with PyTorch and Scikit-Learn, Ch 3*

Logistic Regression Hyperparameters

Regular loss function to minimize during training

$$L(\mathbf{w}, b \mid \mathbf{x}) = -\sum_{i=1} \left[y^{(i)} \log\left(\sigma\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - \sigma\left(z^{(i)}\right)\right) \right]$$

L1 Norm /
LASSO (Least Absolute Shrinkage and Selection Operator)

L1 norm: $\lambda \, ||\mathbf{w}||_1 = \lambda \sum_{j=1}^{m} |w_j|$

L1 Regularization / LASSO (Embedded)

Least Absolute Shrinkage and Selection Operator

L1 penalty against complexity

$$\lambda \, ||\mathbf{w}||_1 = \lambda \sum_{j=1}^{m} |w_j|$$

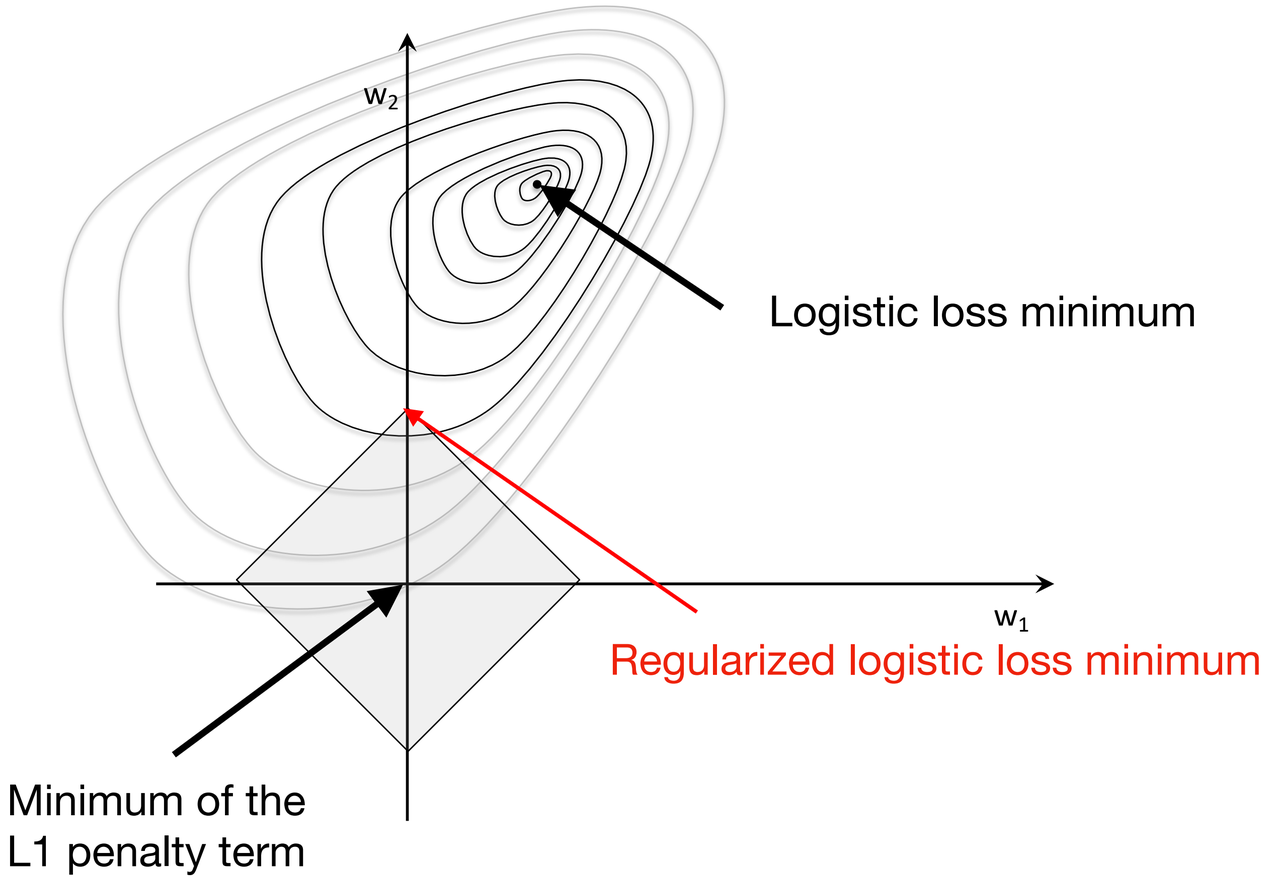
hyperparameter

L1-penalized loss

$$L_{L1}(\mathbf{w}, b \mid \mathbf{x}) = -\sum_{i=1} \left[y^{(i)} \log\left(\sigma\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - \sigma\left(z^{(i)}\right)\right) \right] + \lambda \|w\|_1$$

L1 Regularization / LASSO (Embedded)

Least Absolute Shrinkage and Selection Operator



For more details, see Tibshirani, Ryan, and L. Wasserman. "A closer look at sparse regression." *Lecture notes* (2016).

L1 Regularization / LASSO (Embedded)

Least Absolute Shrinkage and Selection Operator

Wine Dataset

https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data

'Class lab	'Alcoho	'Malic aci	'Ash	Alcalinit of ash'	'Magnesiu	'Total phenc	'Flavanoid	'Nonflavan phenols'	'Proanthocyan	'Color intens	'Hue	OD280/OD31 of diluted win	'Proline

```
from sklearn.model_selection import train_test_split

X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values

X_train, X_test, y_train, y_test =\
    train_test_split(X, y,
                     test_size=0.3,
                     random_state=0,
                     stratify=y)
```

```
from sklearn.preprocessing import StandardScaler


sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(penalty='l1', C=1.0, solver='liblinear', multi_class='ovr')
# Note that C=1.0 is the default. You can increase
# or decrease it to make the regulariztion effect
# stronger or weaker, respectively.
lr.fit(X_train_std, y_train)
print('Training accuracy:', lr.score(X_train_std, y_train))
print('Test accuracy:', lr.score(X_test_std, y_test))
```

```
Training accuracy: 1.0
Test accuracy: 1.0
```

```
lr.intercept_
```

```
array([-1.26363107, -1.21610924, -2.37035486])
```

```
np.set_printoptions(8)
```

```
lr.coef_[lr.coef_!=0].shape
```

```
(23,)
```

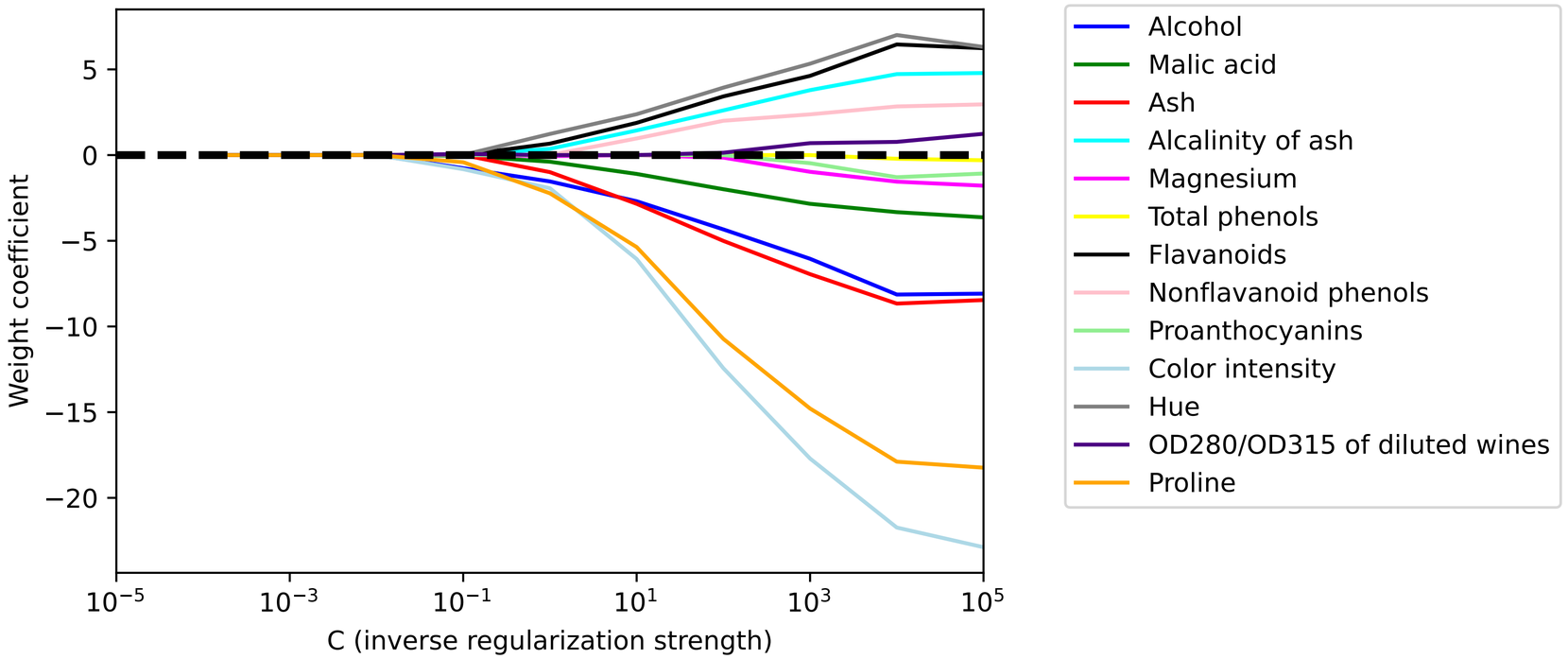
```
lr.coef_
```

```
array([[ 1.2455466 ,  0.18072432,  0.74213192, -1.15948629,  0.        ,
         0.        ,  1.17434899,  0.        ,  0.        ,  0.        ,
         0.        ,  0.54353185,  2.51127873],
       [-1.53786975, -0.38667962, -0.9954337 ,  0.36456123, -0.05923747,
         0.        ,  0.66763266,  0.        ,  0.        , -1.93321837,
         1.23529768,  0.        , -2.23229101],
       [ 0.13570425,  0.16821283,  0.35724291,  0.        ,  0.        ,
         0.        , -2.43809231,  0.        ,  0.        ,  1.56377541,
        -0.81940109, -0.49234846,  0.        ]])
```

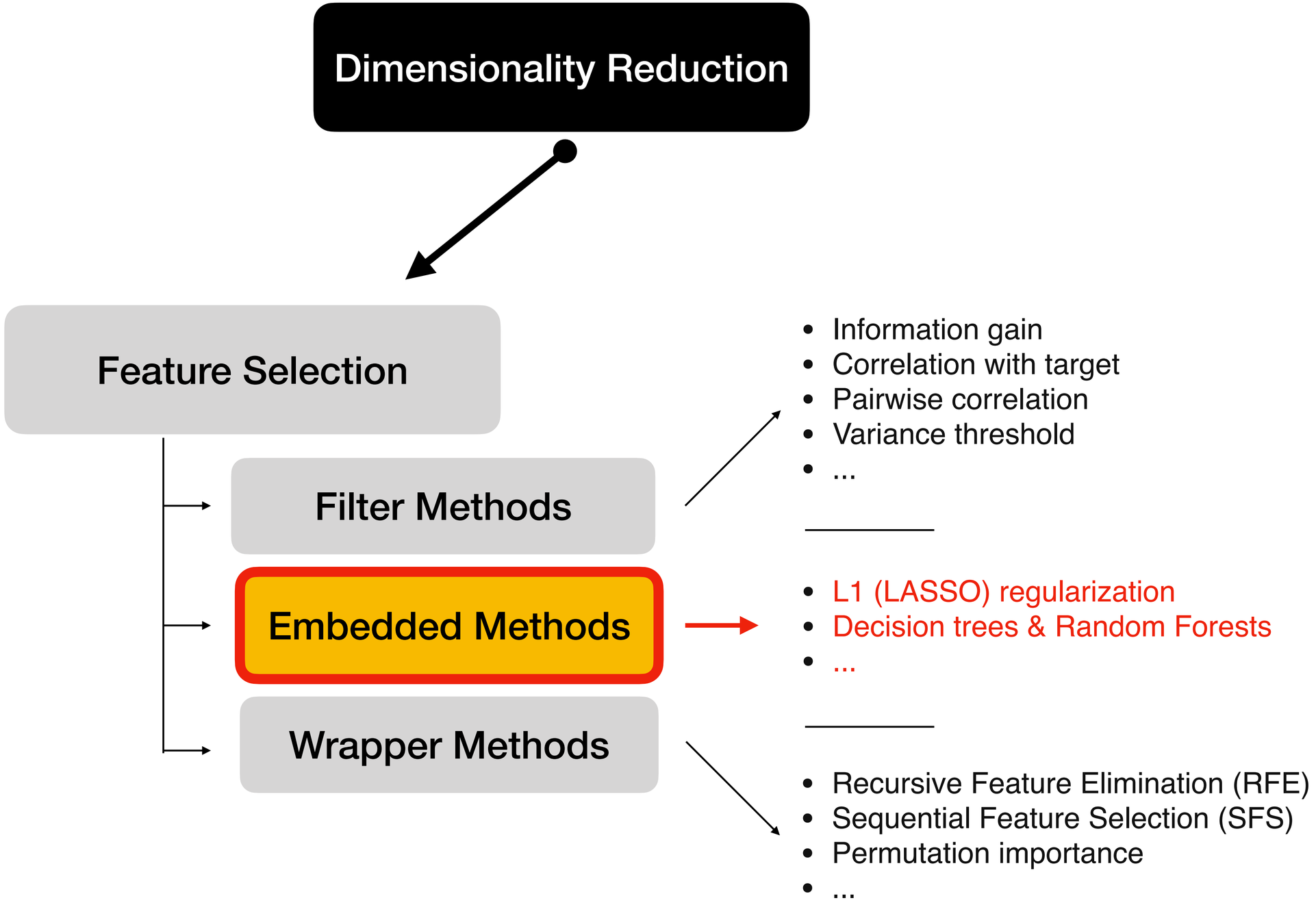
L1 Regularization / LASSO (Embedded)

Least Absolute Shrinkage and Selection Operator

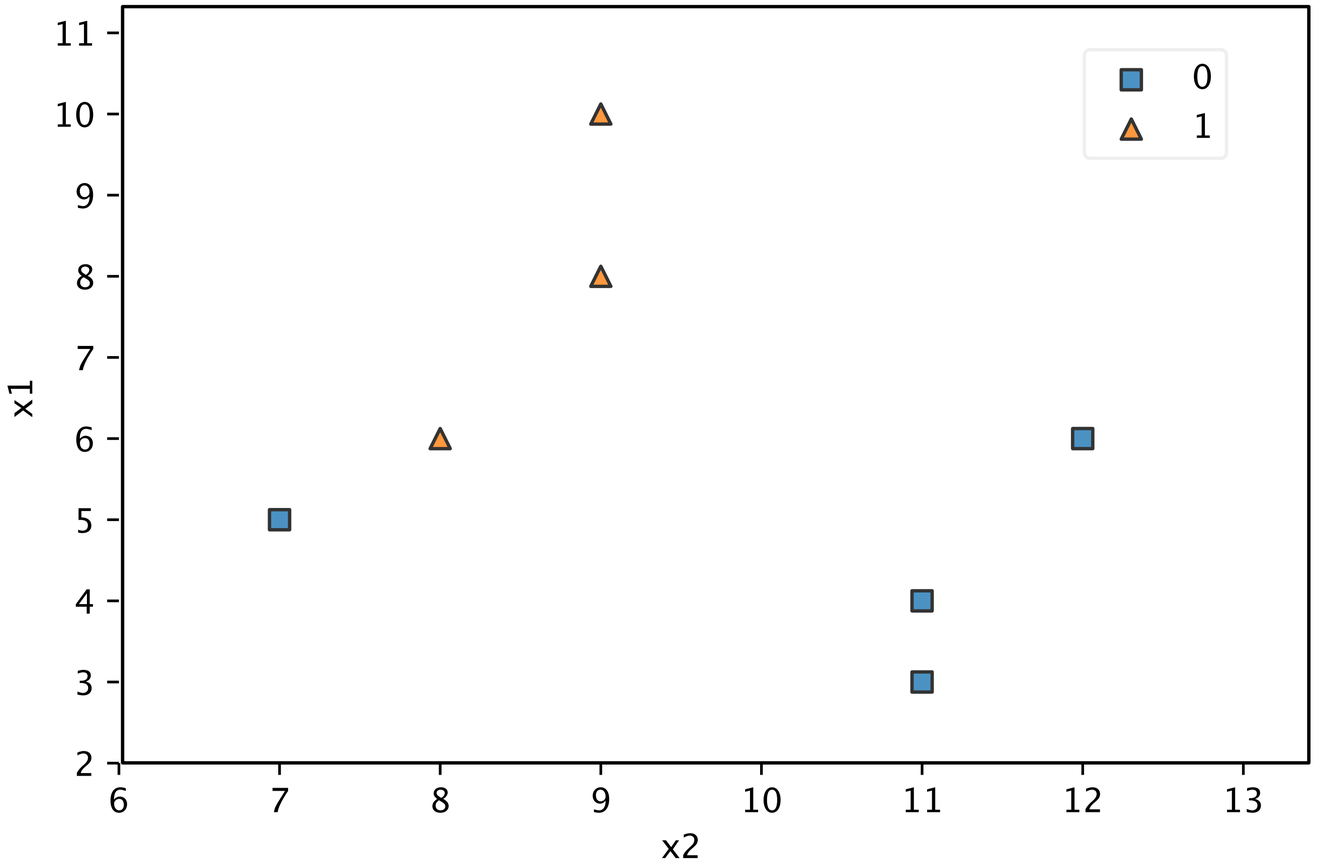
LASSO Path



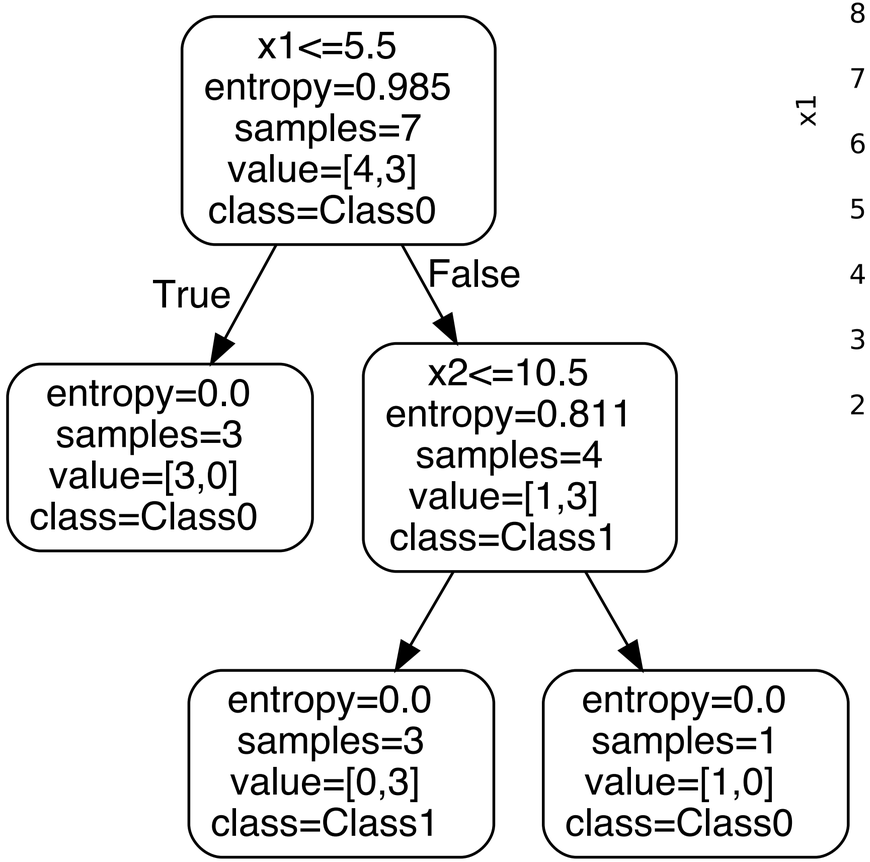
1. Different categories of feature selection
2. Filter methods
3. Embedded methods
    3.1. L1-regularized logistic regression
    3.2. Decision trees & random forest feature importance
4. Wrapper methods
    4.1. Recursive feature elimination
    4.2. Permutation importance
    4.3. Permutation importance code example
    4.4. Sequential feature selection
    4.5. Sequential feature selection code example

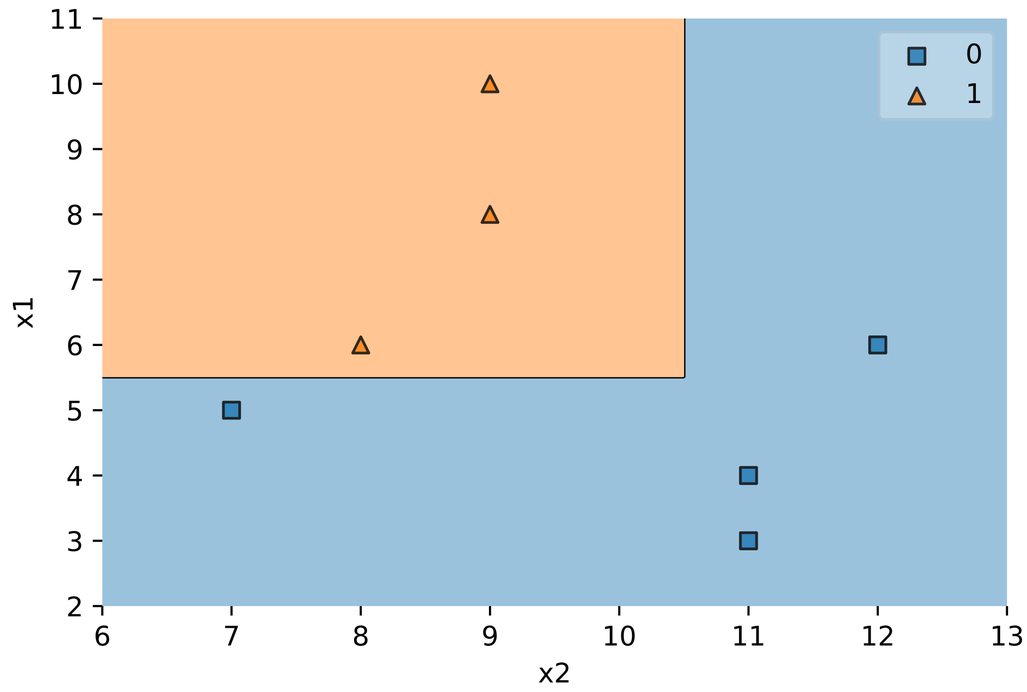

Dimensionality Reduction
Feature Selection
Filter Methods
Embedded Methods
Wrapper Methods
• Information gain
• Correlation with target
• Pairwise correlation
• Variance threshold
• ...
• L1 (LASSO) regularization
• Decision trees & Random Forests
• ...
• Recursive Feature Elimination (RFE)
• Sequential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

Feature Selection in Decision Trees (1)

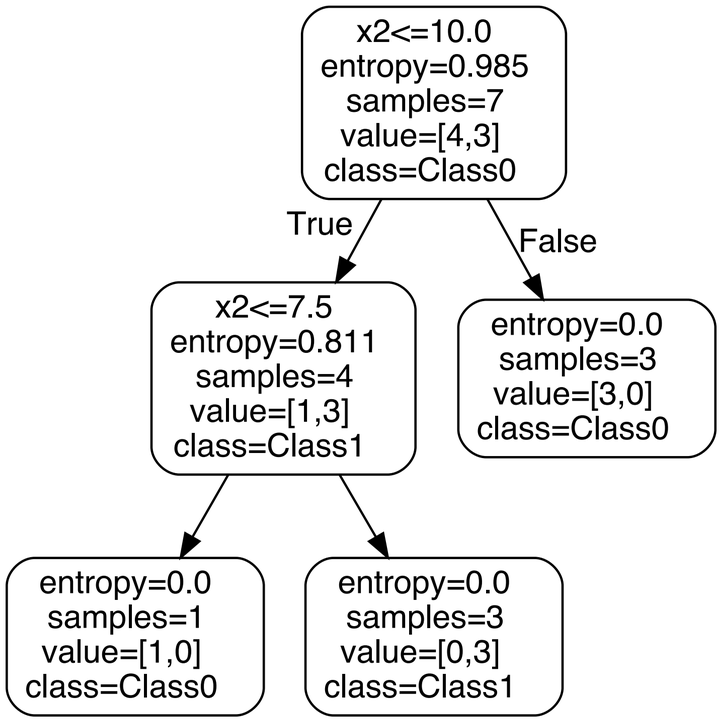


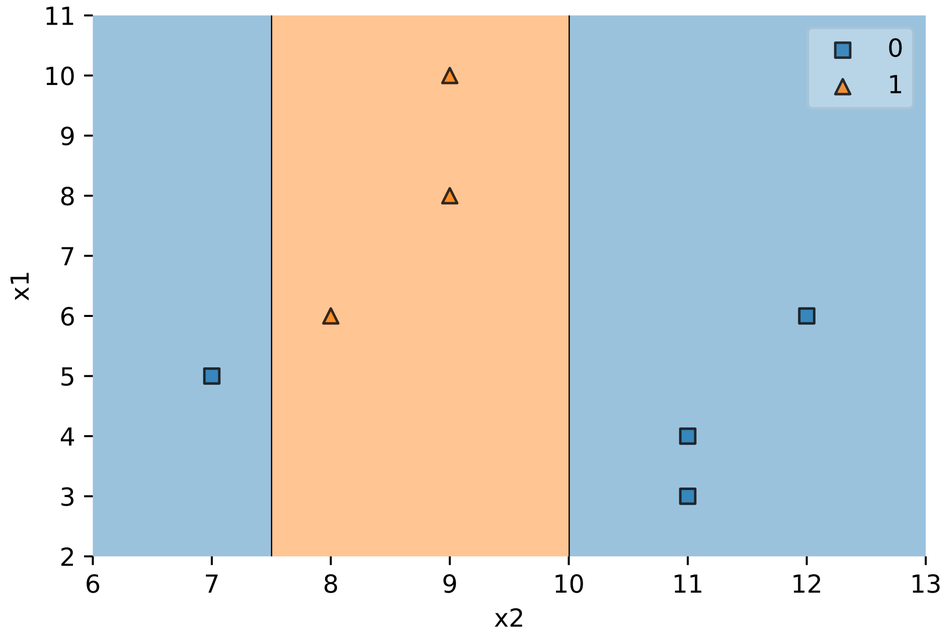
Feature Selection in Decision Trees (2)





Feature Selection in Decision Trees (3)



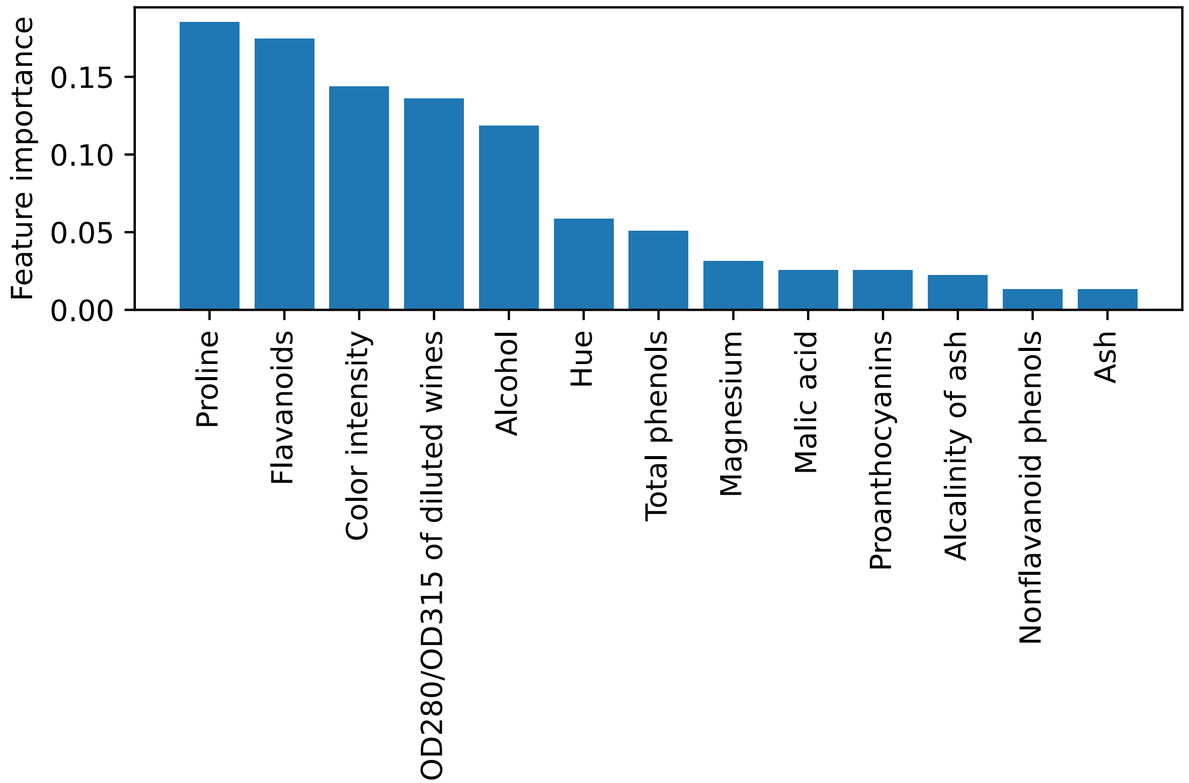


```
import pandas as pd
import numpy as np


df_wine = pd.read_csv('https://archive.ics.uci.edu/'
                      'ml/machine-learning-databases/wine/wine.data',
                      header=None)


df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
                   'Alcalinity of ash', 'Magnesium', 'Total phenols',
                   'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
                   'Color intensity', 'Hue', 'OD280/OD315 of diluted wines',
                   'Proline']

print('Class labels', np.unique(df_wine['Class label']))
df_wine.head()
```

```
Class labels [1 2 3]
```

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

```
from sklearn.ensemble import RandomForestClassifier



forest = RandomForestClassifier(n_estimators=500,
                                random_state=1)

forest.fit(X_train, y_train)
importances = forest.feature_importances_

indices = np.argsort(importances)[::-1]

plt.ylabel('Feature importance')
plt.bar(range(X_train.shape[1]),
        importances[indices],
        align='center')

feat_labels = df_wine.columns[1:]
plt.xticks(range(X_train.shape[1]),
           feat_labels[indices], rotation=90)

plt.xlim([-1, X_train.shape[1]])
```



```
forest.estimators_
```

```
[DecisionTreeClassifier(max_features='auto', random_state=1791095845),
 DecisionTreeClassifier(max_features='auto', random_state=2135392491),
 DecisionTreeClassifier(max_features='auto', random_state=946286476),
 DecisionTreeClassifier(max_features='auto', random_state=1857819720),
 DecisionTreeClassifier(max_features='auto', random_state=491263),
 DecisionTreeClassifier(max_features='auto', random_state=550290313),
 DecisionTreeClassifier(max_features='auto', random_state=1298508491),
 DecisionTreeClassifier(max_features='auto', random_state=2143362693),
 DecisionTreeClassifier(max_features='auto', random_state=630311759),
 DecisionTreeClassifier(max_features='auto', random_state=1013994432),
 DecisionTreeClassifier(max_features='auto', random_state=396591248),
 DecisionTreeClassifier(max_features='auto', random_state=1703301249),
```

max_features : *{"auto", "sqrt", "log2"}, int or float, default="auto"*

The number of features to consider when looking for the best split:

- If int, then consider `max_features` features at each split.
- If float, then `max_features` is a fraction and `round(max_features * n_features)` features are considered at each split.
- If "auto", then `max_features=sqrt(n_features)`.
- If "sqrt", then `max_features=sqrt(n_features)` (same as "auto").
- If "log2", then `max_features=log2(n_features)`.
- If None, then `max_features=n_features`.

Random Forest Feature Importance

Method A: Impurity-based feature importance

(this is used in scikit-learn)

Usually measured as follows:

- for a given feature
  - for each tree
    - compute impurity decrease (Gini, Entropy)
    - weight by number of examples at that node
    - averaged over all trees
- normalize importances so that sum of feature importances sum to 1

Random Forest Feature Importance

Caveats

- Impurity-based feature importance are inflated for categorical features with lots of unique values (we will cover permutation-based performance later, which addresses this)
- Correlated features share importance

Bootstrap Sampling

Random Forest Feature Importance

Method B: Permutation Importance

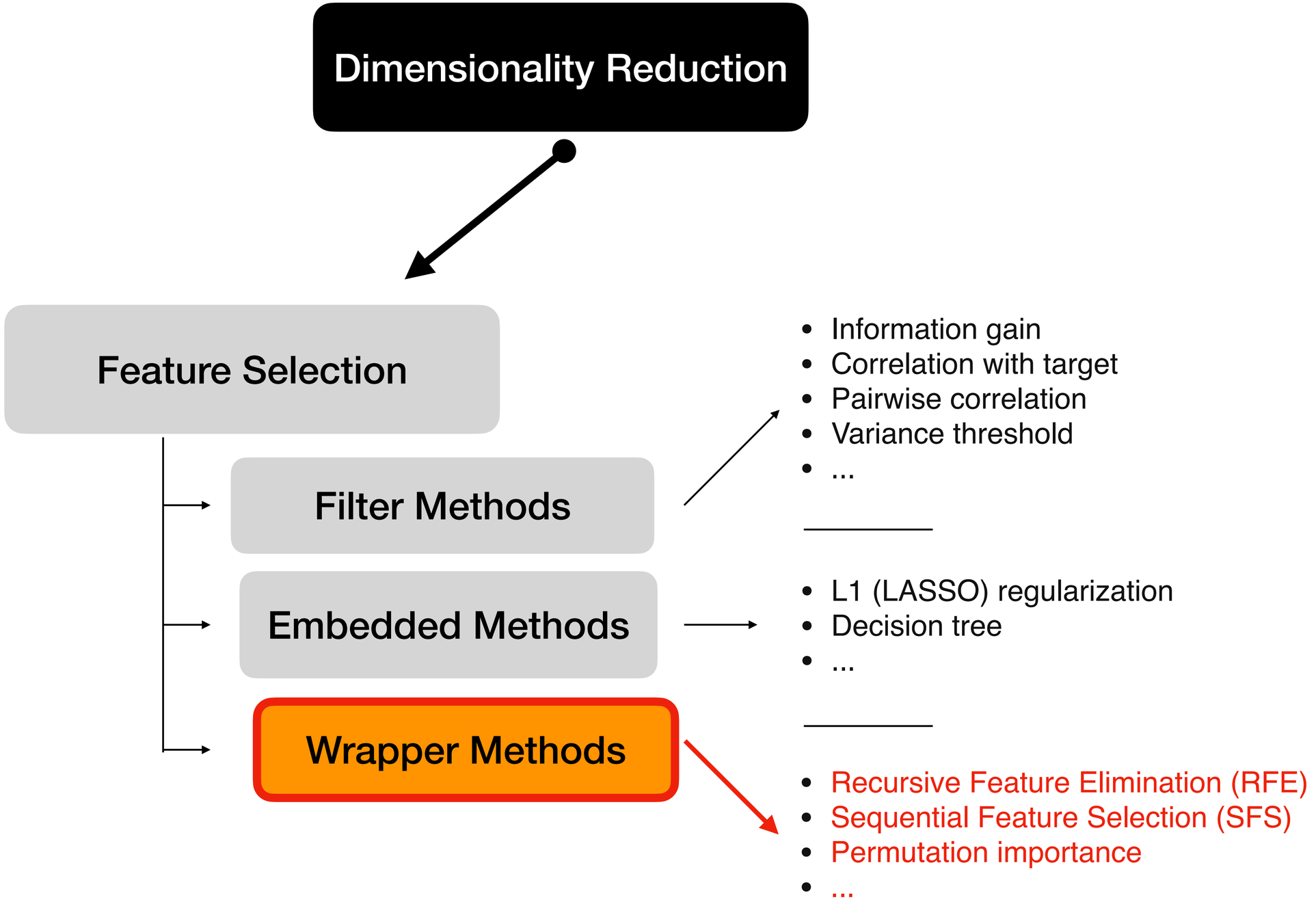
Out-of-bag accuracy:

- During training, for each tree, make prediction for OOB sample (~1/3 of the training data)
- Based on those predictions where example *i* was OOB, compute label via majority vote among the trees that did not use example *i* during model fitting
- The proportion over all examples where the prediction (by majority vote) is correct is the OOB accuracy estimate

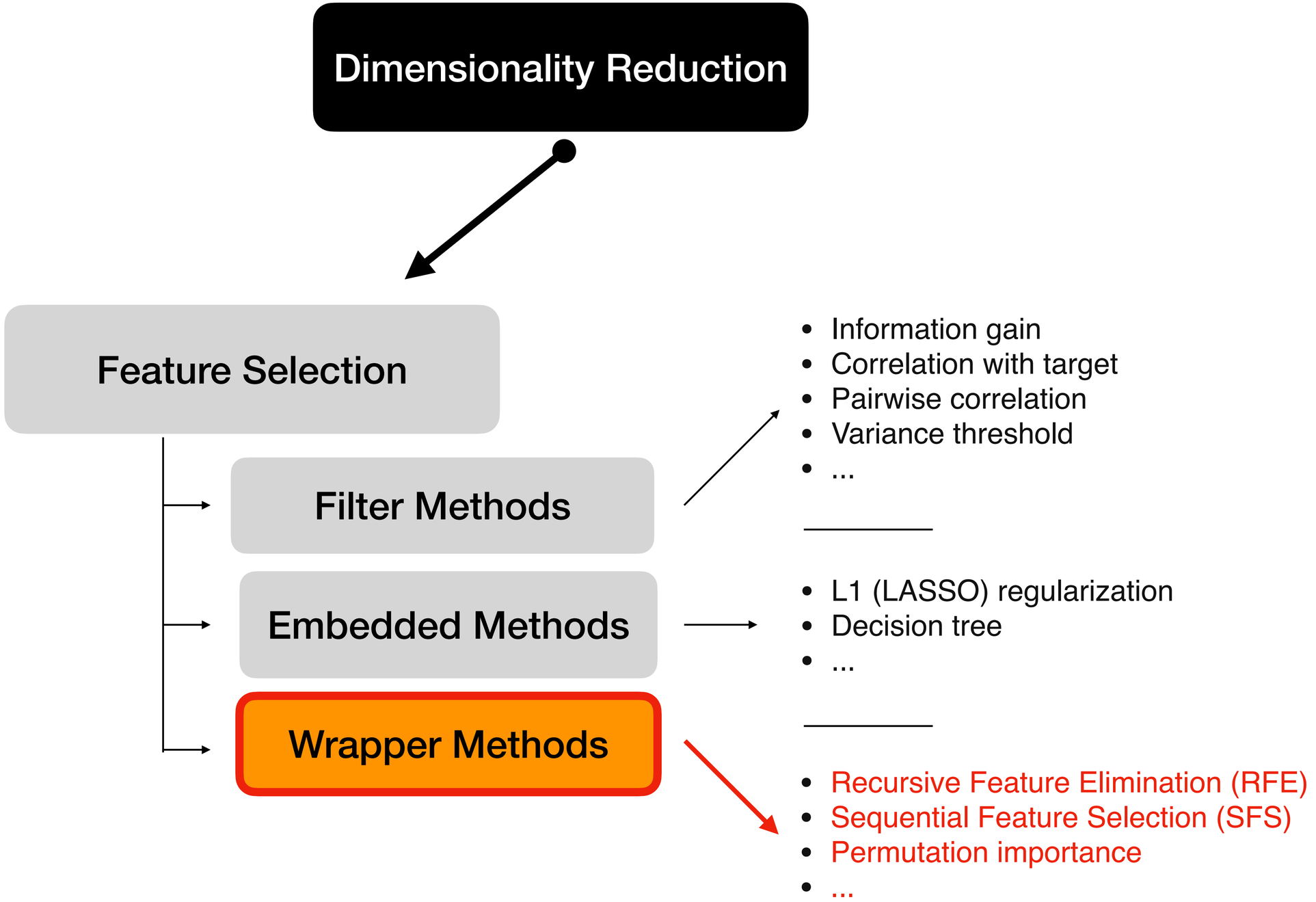
Out-of-bag feature importance via permutation:

(we will also cover a generalized version with a hold out set later)

- Count votes for correct class
- Given feature *i,* permute this feature in OOB examples of a tree
- Compute the number of correct votes after permutation from the number of votes before permutation for given tree
- Repeat for all trees in the random forest and average the importance
- Repeat for other features


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
• ...

1. Different categories of feature selection
2. Filter methods
3. Embedded methods
    3.1. L1-regularized logistic regression
    3.2. Random forest feature importance
4. **Wrapper methods**
    4.1. Recursive feature elimination
    4.2. Permutation importance
    4.3. Permutation importance code example
    4.4. Sequential feature selection
    4.5. Sequential feature selection code example

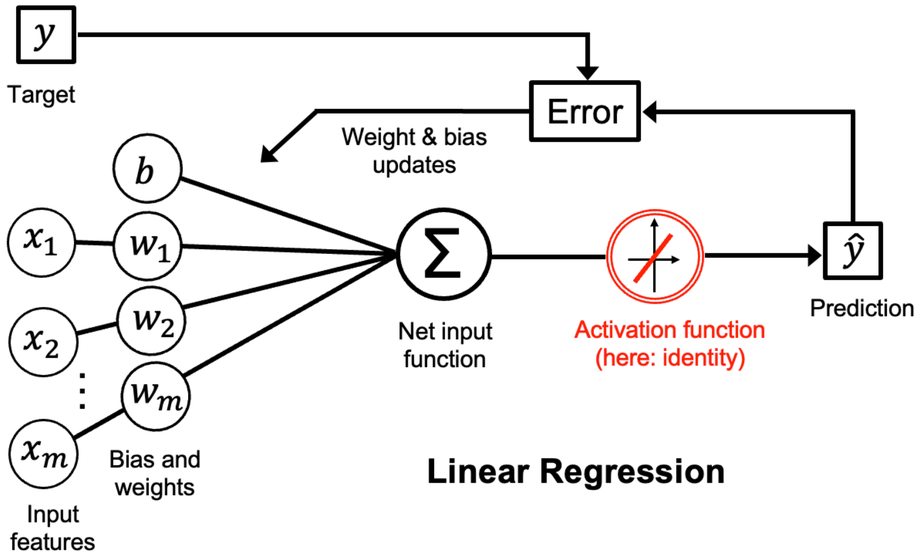

Dimensionality Reduction
Feature Selection
Filter Methods
Embedded Methods
Wrapper Methods
Information gain
Correlation with target
Pairwise correlation
Variance threshold
...
L1 (LASSO) regularization
Decision tree
...
Recursive Feature Elimination (RFE)
Sequential Feature Selection (SFS)
Permutation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

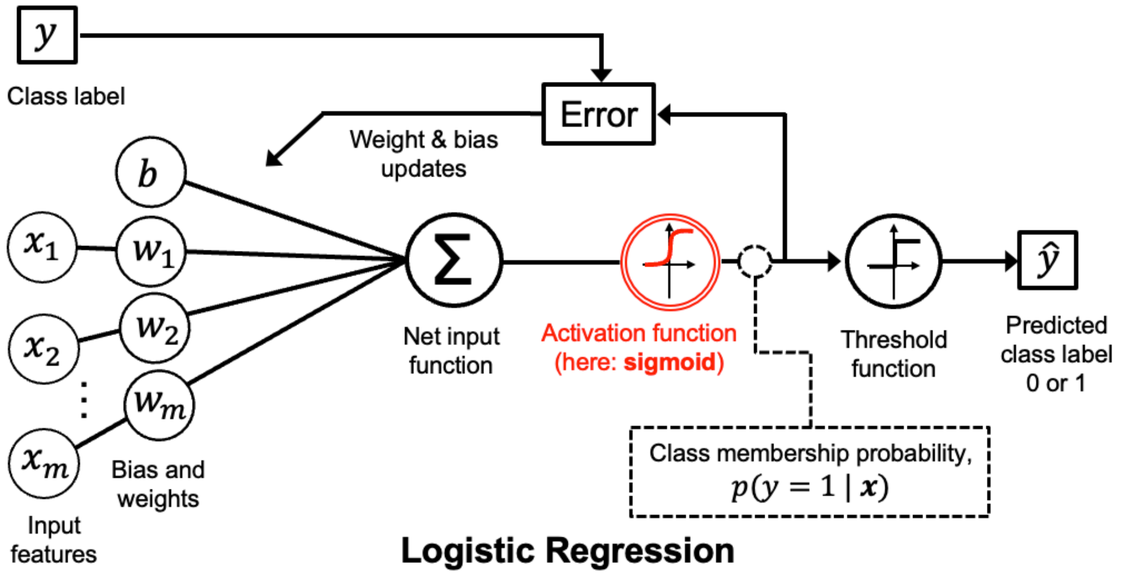
Recursive Feature Elimination (Wrapper)

Consider a (generalized) linear model (like linear or logistic regression):

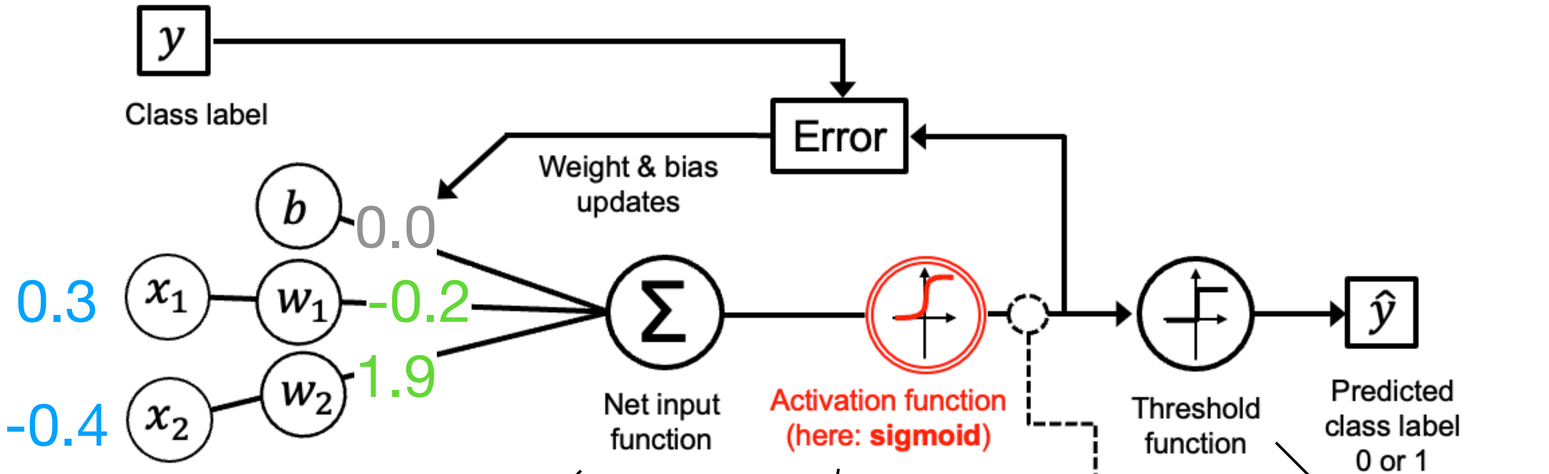
1. Fit model to dataset
2. Eliminate feature with the smallest coefficient ("most unimportant")
3. Repeat steps 1-2 until desired number of features is reached

\


y
Target
Error
Weight & bias updates
b
x1
w1
x2
w2
wm
xm
Σ
Net input function
Activation function (here: identity)
ŷ
Prediction
Bias and weights
Input features
Linear Regression


y
Class label
Error
Weight & bias updates
b
x1
w1
x2
w2
wm
xm
Σ
Net input function
Activation function (here: sigmoid)
Threshold function
ŷ
Predicted class label 0 or 1
Bias and weights
Input features
Class membership probability, p(y = 1 | x)
Logistic Regression

Logistic Regression



$0.3 \times (-0.2) + (-0.4) \times 1.9 + 0.0 = -0.82$

$$\frac{1}{1 + e^{-(-0.82)}} = 0.305764...$$

$$\hat{y} = \begin{cases} 1 & \text{if activation} \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

RFE Code example (1)

Dataset Preparation

[3]:
```
import pandas as pd
import numpy as np


df_wine = pd.read_csv('https://archive.ics.uci.edu/'
                      'ml/machine-learning-databases/wine/wine.data',
                      header=None)


df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
                   'Alcalinity of ash', 'Magnesium', 'Total phenols',
                   'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
                   'Color intensity', 'Hue', 'OD280/OD315 of diluted wines',
                   'Proline']

print('Class labels', np.unique(df_wine['Class label']))
df_wine.head()
```

```
Class labels [1 2 3]
```

[3]:

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

RFE Code example (2)

[4]:
```
from sklearn.model_selection import train_test_split

X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values

X_train, X_test, y_train, y_test =\
    train_test_split(X, y,
                     test_size=0.3,
                     random_state=0,
                     stratify=y)
```

[5]:
```
from sklearn.preprocessing import StandardScaler


sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

RFE Code example (3)

Recursive Feature Selection

[6]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE



lr = LogisticRegression(solver='liblinear', random_state=123)

rfe = RFE(estimator=lr, n_features_to_select=5, step=1)
rfe.fit(X_train_std, y_train)

X_train_sub = rfe.transform(X_train_std)
```

Which features are selected?

[7]:
```
rfe.support_
```

[7]:
```
array([ True, False, False, False, False, False,  True, False, False,
        True,  True, False,  True])
```

[8]:
```
df_wine.columns[1:][rfe.support_]
```

[8]:
```
Index(['Alcohol', 'Flavanoids', 'Color intensity', 'Hue', 'Proline'], dtype='object')
```

RFE Code example (4)

RFE as Part of a Pipeline

[9]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline


pipe = make_pipeline(RFE(estimator=lr, step=1),
                     KNeighborsClassifier())


parameters = {'rfe__n_features_to_select': range(1, 13),
              'kneighborsclassifier__n_neighbors': range(1, 10) }

grid = GridSearchCV(pipe, param_grid=parameters, cv=10, n_jobs=-1)
grid.fit(X_train_std, y_train)

print('Best params:', grid.best_params_)
print('Best accuracy:', grid.best_score_)
```

```
Best params: {'kneighborsclassifier__n_neighbors': 3, 'rfe__n_features_to_select': 5}
Best accuracy: 0.9916666666666666
```

[10]:
```
# Reduced feature set from grid search

grid.best_estimator_.score(X_test_std, y_test)
```

[10]:
```
1.0
```

[11]:
```
# Full feature set for reference

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_std, y_train)
knn.score(X_test_std, y_test)
```

[11]:
```
0.9629629629629629
```

RFE with other models

sklearn.feature_selection.RFE ¶

class sklearn.feature_selection.RFE(*estimator*, ***, *n_features_to_select=None*, *step=1*, *verbose=0*, *importance_getter='auto'*) [source]

Parameters: **estimator : *`Estimator` instance***

A supervised learning estimator with a `fit` method that provides information about feature importance (e.g. `coef_`, `feature_importances_`).

https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html

Recursive Feature Elimination Pros and Cons

(+) Can explicitly select number of features

(+) Not super expensive (if linear model is used)

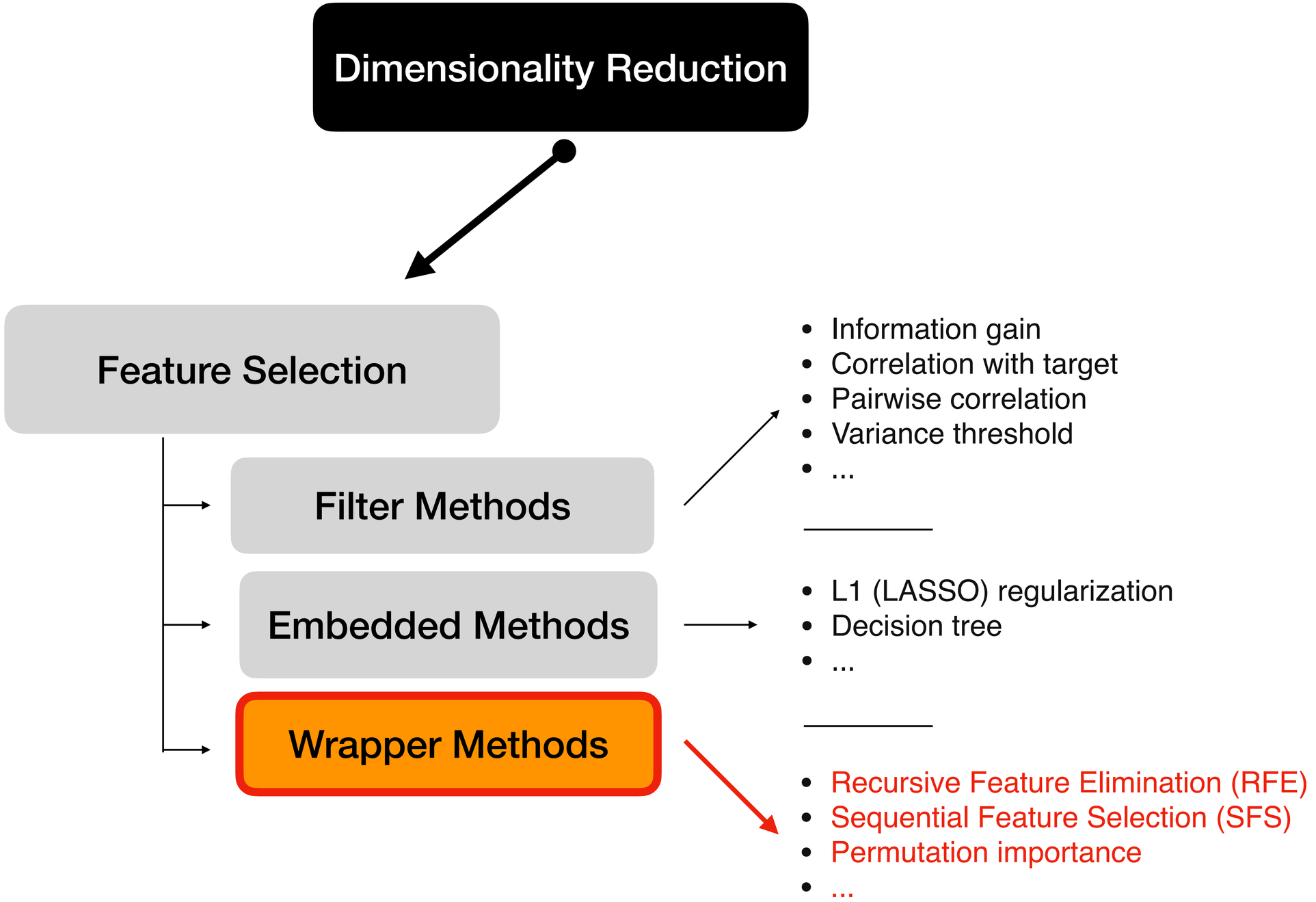
(+) Takes feature interaction into account

(-) Assumes linear separability (if linear model is used)

(-) Does not optimize performance metric directly

(-) Needs search method to find good number of features

1. Different categories of feature selection
2. Filter methods
3. Embedded methods
    3.1. L1-regularized logistic regression
    3.2. Random forest feature importance
4. Wrapper methods
    4.1. Recursive feature elimination
    4.2. Permutation importance
    4.3. Permutation importance code example
    4.4. Sequential feature selection
    4.5. Sequential feature selection code example


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
...

Permutation Importance

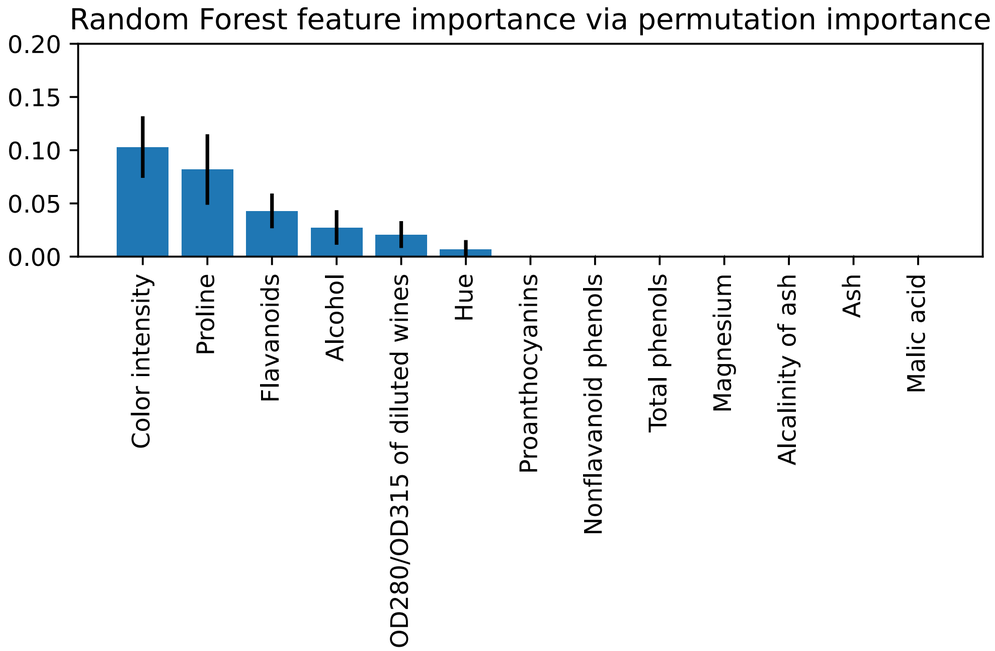
For each feature column:

1. shuffle feature column
2. observe performance and compare to original

Permutation Importance

For each feature column:

1. shuffle feature column
2. observe performance and compare to original



Permutation Importance

Permutation importance often gives similar results as
random forest impurity-based importance but it is model agnostic

Permutation Importance

Permutation importance often gives similar results as
random forest impurity-based importance but it is model agnostic

Also, permutation importance is not strictly feature selection, but it tells us which features a model relies on the most

Permutation Importance

Permutation importance often gives similar results as
random forest impurity-based importance but it is model agnostic

Also, permutation importance is not strictly feature selection, but it tells
us which features a model relies on the most

You can think of permutation importance as a Generalization of Method B in the
random forest video but hold out set instead of OOB samples

Permutation Importance

intuitive & model-agnostic

1. Take a model that was fit to the training set

```
>>> randomforest.fit(X_train, y_train)
```

Permutation Importance

intuitive & model-agnostic

1. Take a model that was fit to the training set
2. Estimate the predictive performance of the model on an independent dataset (e.g., validation dataset) and record it as the baseline performance

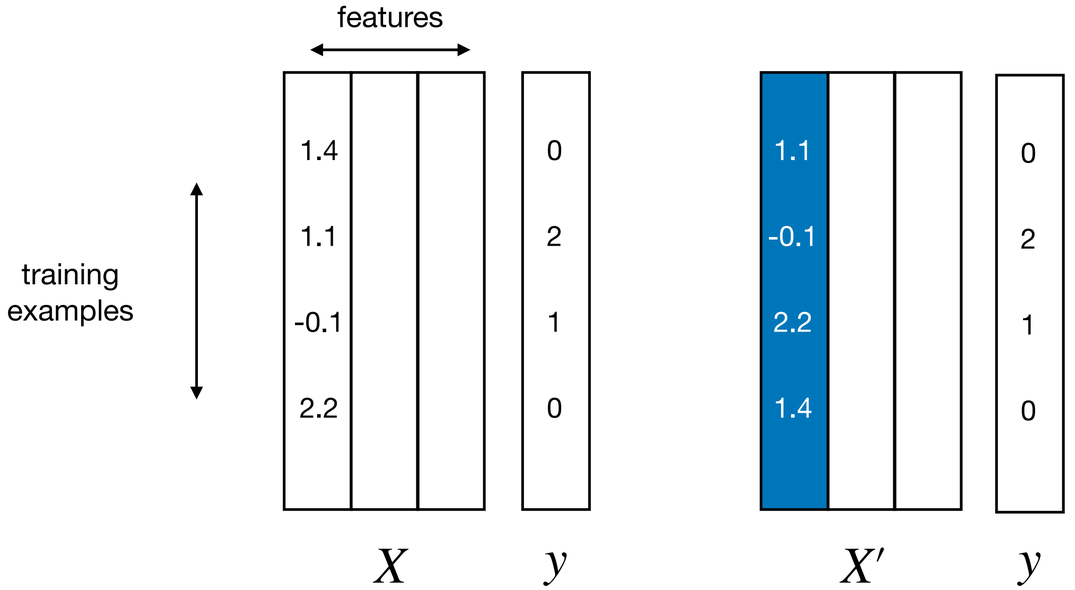
```
>>> randomforest.fit(X_train, y_train)

>>> acc = randomforest.score(X_val, y_val)
>>> print(acc)
0.99
```

Permutation Importance

intuitive & model-agnostic

1. Take a model that was fit to the training set
2. Estimate the predictive performance of the model on an independent dataset (e.g., validation dataset) and record it as the baseline performance
3. For each feature *j*
    a. randomly permute feature column *j* in the original dataset

```
>>> randomforest.fit(X_train, y_train)

>>> acc = randomforest.score(X_val, y_val)
>>> print(acc)
0.99
```

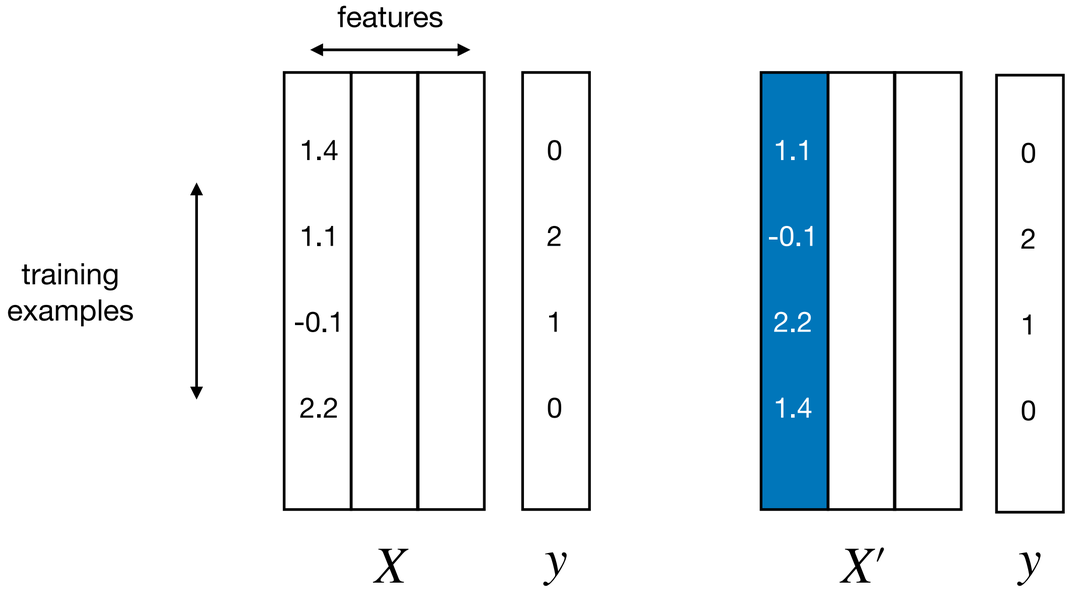


Permutation Importance

intuitive & model-agnostic

1. Take a model that was fit to the training set
2. Estimate the predictive performance of the model on an independent dataset (e.g., validation dataset) and record it as the baseline performance
3. For each feature *j*
    a. randomly permute feature column *j* in the original dataset
    b. record the predictive performance of the model on the dataset with the permuted column

```
>>> randomforest.fit(X_train, y_train)

>>> acc = randomforest.score(X_val, y_val)
>>> print(acc)
0.99
```




```
>>> acc = randomforest.score(
...     X_val_perm, y_val_perm)
>>> print(acc)
0.85
```

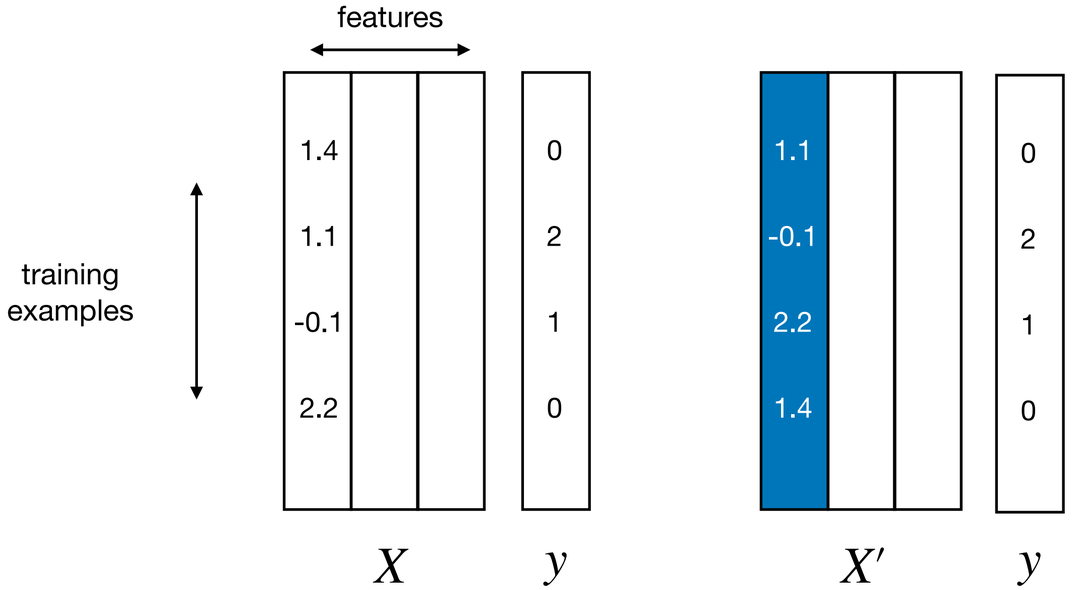
Permutation Importance

intuitive & model-agnostic

1. Take a model that was fit to the training set
2. Estimate the predictive performance of the model on an independent dataset (e.g., validation dataset) and record it as the baseline performance
3. For each feature *j*
    a. randomly permute feature column *j* in the original dataset
    b. record the predictive performance of the model on the dataset with the permuted column
    c. compute the feature importance as the difference between the baseline performance (step 2) and the performance on the permuted dataset

Repeat a-c exhaustively (all combinations) or a large number of times and compute the feature importance as the average difference

```
>>> randomforest.fit(X_train, y_train)

>>> acc = randomforest.score(X_val, y_val)
>>> print(acc)
0.99
```




```
>>> acc = randomforest.score(
...     X_val_perm, y_val_perm)
>>> print(acc)
0.85
```

Column-Drop variant:

For each feature column *j:*

1. temporarily remove column
2. fit model to reduced dataset
3. compute validation set performance and compare to before

More accurate but more expensive (and not for 1 particular model)

(will adopt something similar for SFS)

Permutation Importance Pros and Cons

(+) Model agnostic

(+) Based on metric of choice

(+) Easy to understand

(+/-) Feature importance is for that particular model (feature might be more/less important to another model)

(+) Unlike impurity-based random forest importance, it does not suffer from "overfitting" since an independent dataset is used

(-) Like in impurity-based random forest importance, the importance is undervalued if two features are highly correlated

1. Different categories of feature selection
2. Filter methods
3. Embedded methods
    3.1. L1-regularized logistic regression
    3.2. Random forest feature importance
4. Wrapper methods
    4.1. Recursive feature elimination
    4.2. Permutation importance
    4.3. Permutation importance code example
    4.4. Sequential feature selection
    4.5. Sequential feature selection code example

Permutation Importance -- Dataset Preparation (1)

```
import pandas as pd
import numpy as np


df_wine = pd.read_csv('https://archive.ics.uci.edu/'
                      'ml/machine-learning-databases/wine/wine.data',
                      header=None)


df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
                   'Alcalinity of ash', 'Magnesium', 'Total phenols',
                   'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
                   'Color intensity', 'Hue', 'OD280/OD315 of diluted wines',
                   'Proline']

print('Class labels', np.unique(df_wine['Class label']))
df_wine.head()
```

```
Class labels [1 2 3]
```

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32

Dataset Preparation (2)

```
from sklearn.model_selection import train_test_split

X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values

X_train, X_test, y_train, y_test =\
    train_test_split(X, y,
                     test_size=0.3,
                     random_state=0,
                     stratify=y)
```

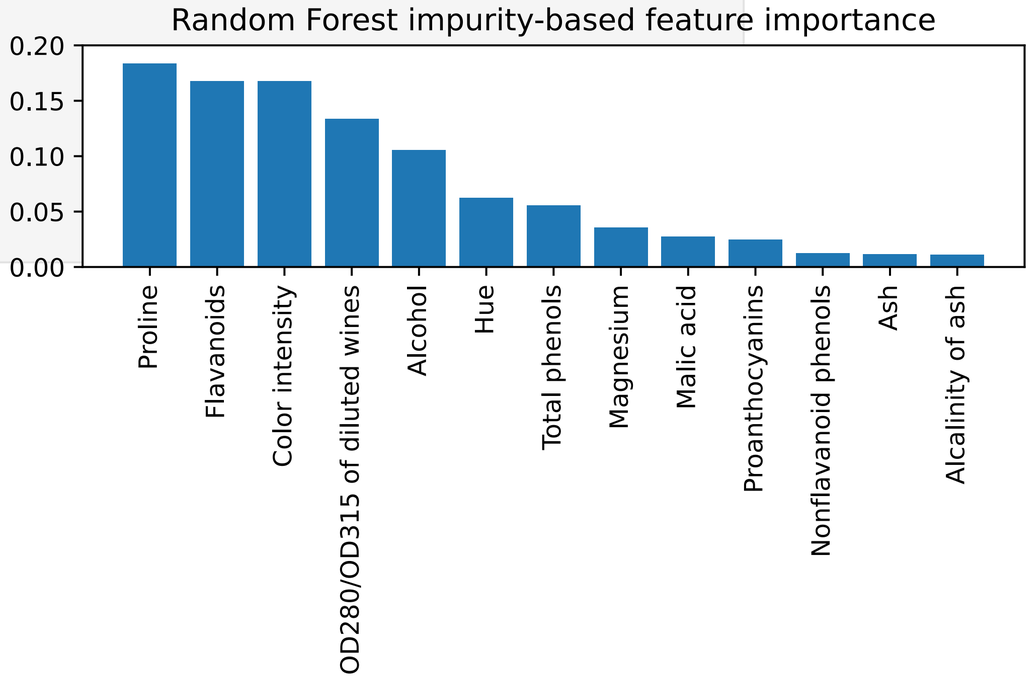
Random Forest Model (1)

```
from sklearn.ensemble import RandomForestClassifier


forest = RandomForestClassifier(n_estimators=100,
                                random_state=0)

forest.fit(X_train, y_train)

print('Training accuracy:', np.mean(forest.predict(X_train) == y_train)*100)
print('Test accuracy:', np.mean(forest.predict(X_test) == y_test)*100)
```

```
Training accuracy: 100.0
Test accuracy: 100.0
```

Random Forest Model (2)

```
importance_vals = forest.feature_importances_
indices = np.argsort(importance_vals)[::-1]

# Plot the feature importances of the forest
plt.figure()
plt.title("Random Forest impurity-based feature importance")
plt.bar(range(X_train.shape[1]), importance_vals[indices])

plt.xticks(range(X_train.shape[1]), df_wine.columns[1:][indices], rotation=90)
plt.xlim([-1, X_train.shape[1]])
plt.ylim([0, 0.2])

plt.tight_layout()
plt.savefig('1.pdf')
plt.show()
```

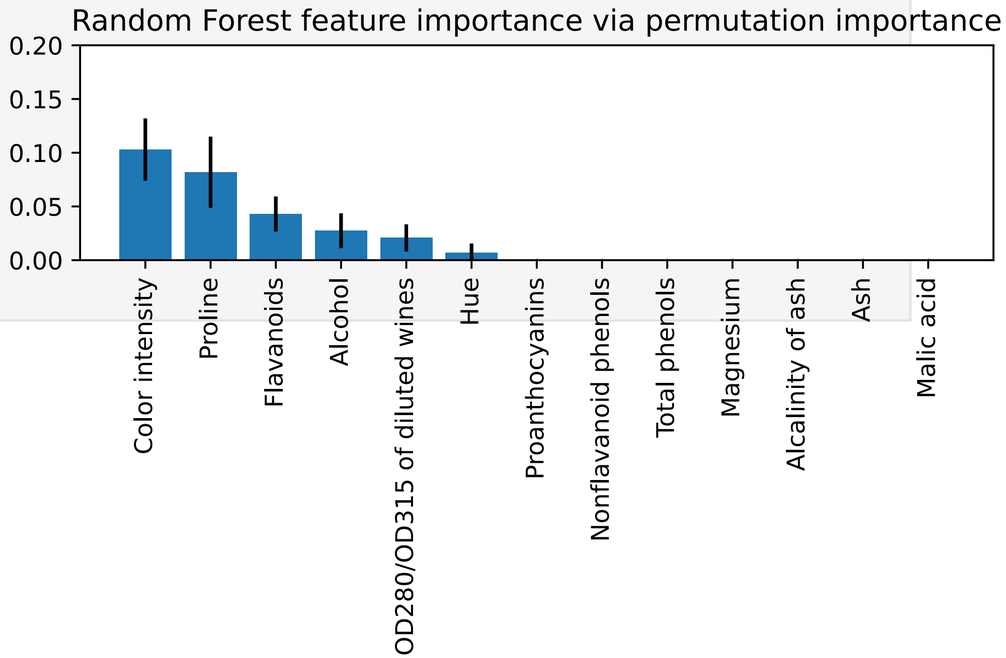


Random Forest Model (3)

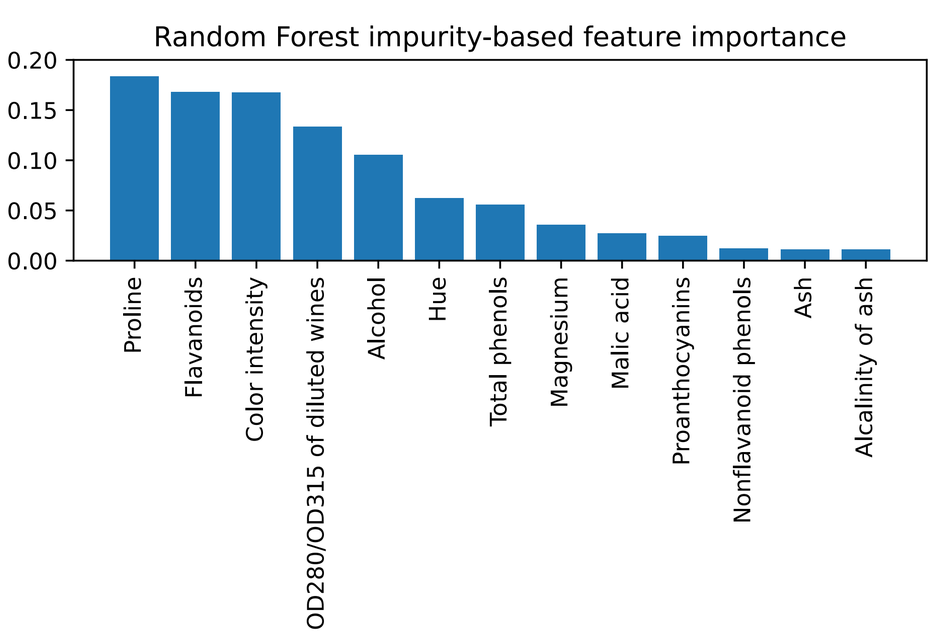
```
from mlxtend.evaluate import feature_importance_permutation


imp_vals, imp_all = feature_importance_permutation(
    predict_method=forest.predict,
    X=X_test,
    y=y_test,
    metric='accuracy',
    num_rounds=50,
    seed=0)
```

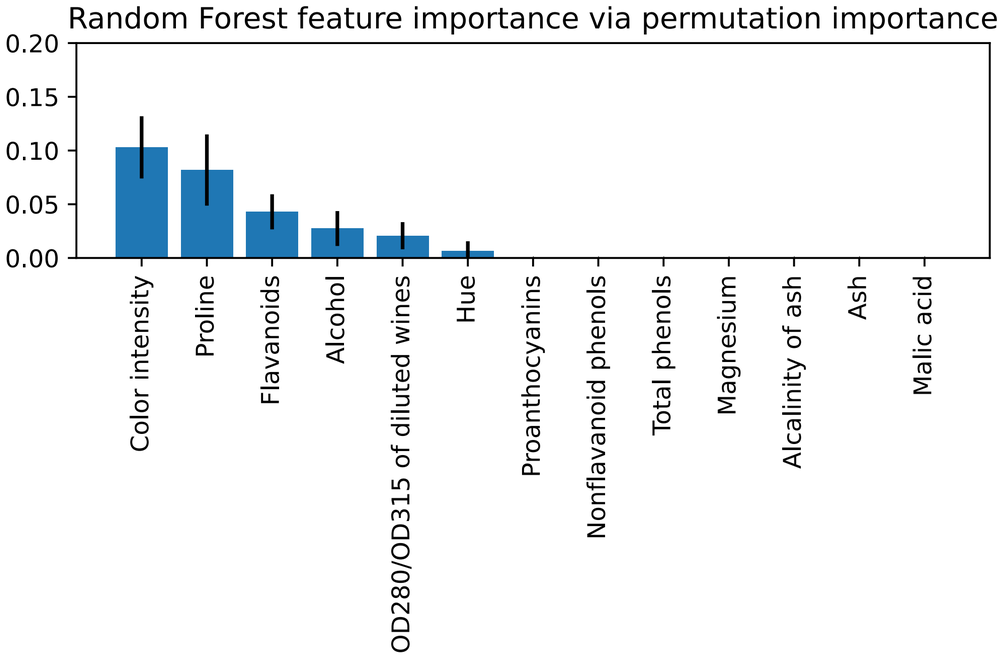
Random Forest Model (4)

```
std = np.std(imp_all, axis=1)
indices = np.argsort(imp_vals)[::-1]

plt.figure()
plt.title("Random Forest feature importance via permutation importance")
plt.bar(range(X_train.shape[1]), imp_vals[indices], yerr=std[indices])

plt.xticks(range(X_train.shape[1]), df_wine.columns[1:][indices], rotation=90)
plt.xlim([-1, X_train.shape[1]])
plt.ylim([0, 0.2])

plt.tight_layout()
plt.savefig('2.pdf')
plt.show()
```



Random Forest Model (4)

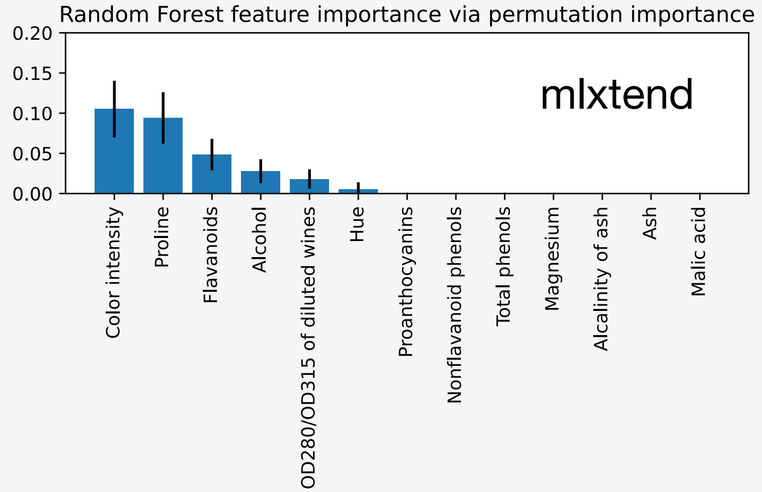


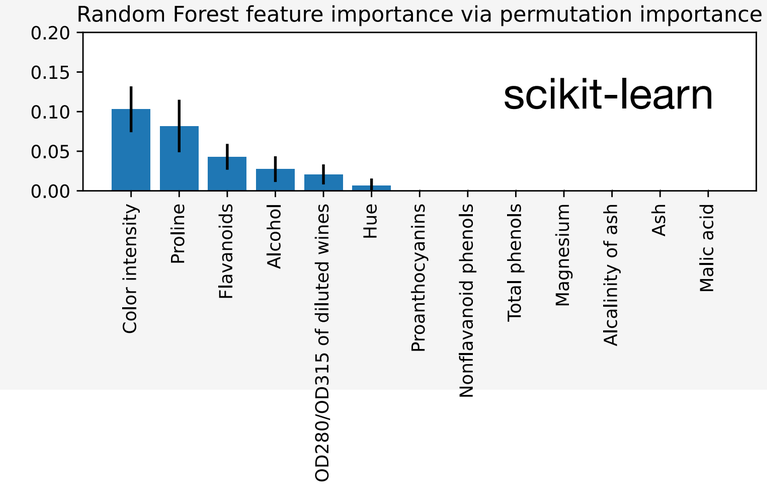


More useful not to scale the feature importance as we can read the accuracy drop from it

mlxtend vs scikit-learn

```
from sklearn.inspection import permutation_importance


forest = RandomForestClassifier(n_estimators=100,
                                random_state=0)

forest.fit(X_train, y_train)

result = permutation_importance(
    estimator=forest,
    X=X_test,
    y=y_test,
    scoring='accuracy',
    n_repeats=50,
    random_state=0
)
```





Random Feature as Control (1)

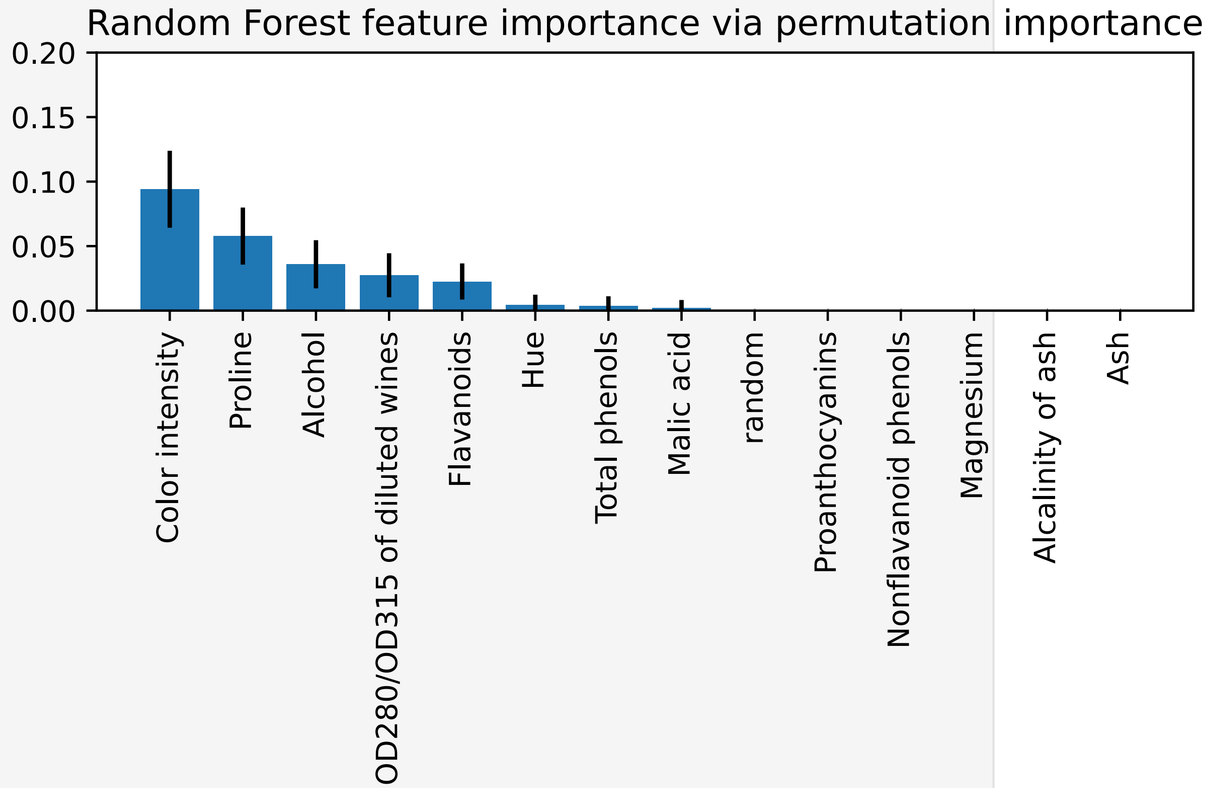
```
import numpy as np


np.random.seed(123)
x1 = np.random.randn(X_train.shape[0]).reshape(-1, 1)
x2 = np.random.randn(X_test.shape[0]).reshape(-1, 1)
```

```
X_train_r = np.hstack((X_train, x1))
X_test_r = np.hstack((X_test, x2))
X_test_r.shape
```

```
(54, 14)
```

Random Feature as Control (2)

```
from sklearn.inspection import permutation_importance
from sklearn.ensemble import RandomForestClassifier


forest = RandomForestClassifier(n_estimators=100,
                                random_state=0)

forest.fit(X_train_r, y_train)

result = permutation_importance(
    estimator=forest,
    X=X_test_r,
    y=y_test,
    scoring='accuracy',
    n_repeats=50,
    random_state=0
)


indices = np.argsort(result['importances_mean'])[::-1]

plt.figure()
plt.title("Random Forest feature importance via permutation importance")
plt.bar(
    range(X_train_r.shape[1]),
    result['importances_mean'][indices],
    yerr=result['importances_std'][indices]
)

feature_names = np.array(list(df_wine.columns[1:])+['random'])
plt.xticks(range(X_train_r.shape[1]), feature_names[indices], rotation=90)
```



Correlated Features (1)

```
np.random.seed(123)

y = np.zeros(1000)
y[:500] = 1

x1 = np.random.randn(1000)

x2 = np.empty(1000)
x2[:500] = np.random.randn(500)
x2[500:] = np.random.randn(500)+4

x3 = x2

X = np.vstack((x3, x2, x1)).swapaxes(1, 0)
X.shape
```

```
(1000, 3)
```

Correlated Features (2)

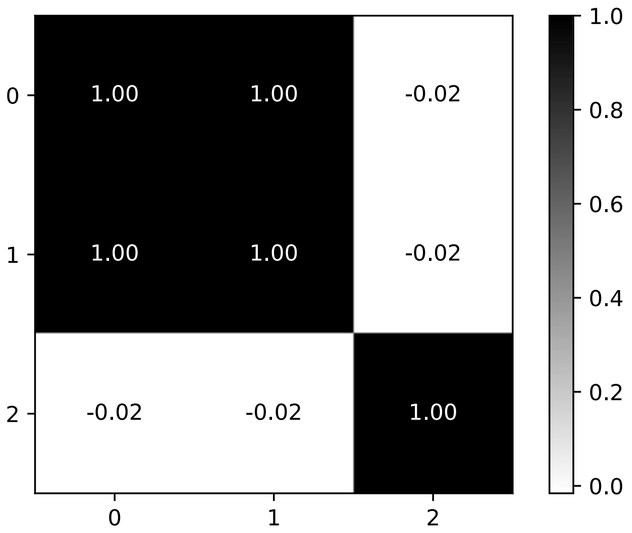
```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test =\
    train_test_split(X, y,
                     test_size=0.3,
                     random_state=123,
                     stratify=y)
```

```
from sklearn.ensemble import RandomForestClassifier


forest = RandomForestClassifier(n_estimators=10,
                                random_state=123,
                                max_features=2)

forest.fit(X_train, y_train)

print('Training accuracy:', np.mean(forest.predict(X_train) == y_train)*100)
print('Test accuracy:', np.mean(forest.predict(X_test) == y_test)*100)
```

```
Training accuracy: 99.71428571428571
Test accuracy: 97.66666666666667
```

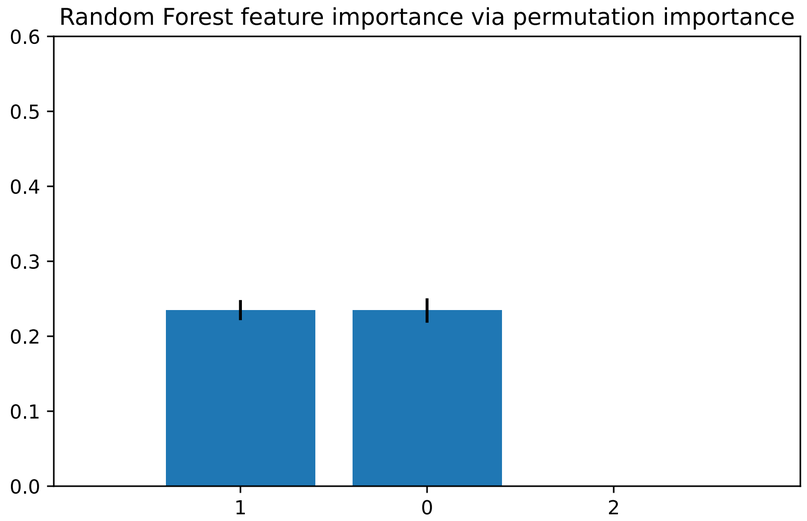
Correlated Features (3)

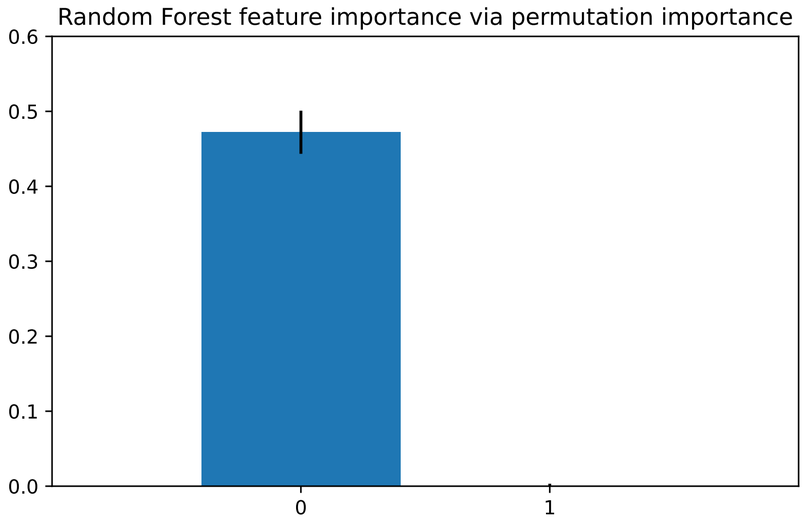
```
from mlxtend.plotting import heatmap


heatmap(corr, cmap='binary')
plt.savefig('5.pdf')
plt.show()
```



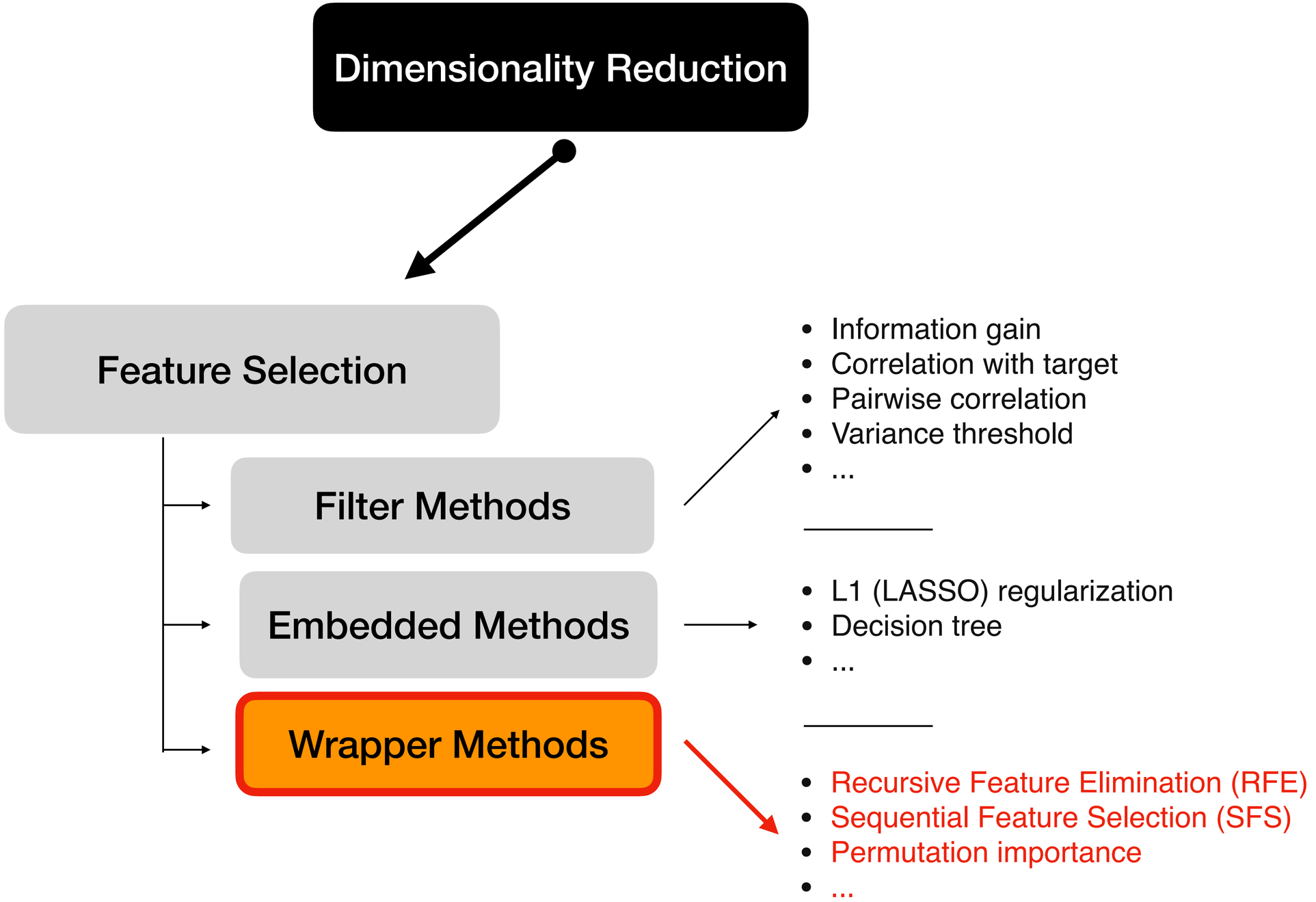
Correlated Features (4)

```
from mlxtend.evaluate import feature_importance_permutation


imp_vals, imp_all = feature_importance_permutation(
    predict_method=forest.predict,
    X=X_test,
    y=y_test,
    metric='accuracy',
    num_rounds=50,
    seed=123)
```





1. Different categories of feature selection
2. Filter methods
3. Embedded methods
    3.1. L1-regularized logistic regression
    3.2. Random forest feature importance
4. Wrapper methods
    4.1. Recursive feature elimination
    4.2. Permutation importance
    4.3. Permutation importance code example
    4.4. Sequential feature selection
    4.5. Sequential feature selection code example


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
...

Which feature selection would guarantee optimal model performance?

Which feature selection would guarantee optimal model performance?

Trying all possible feature combinations --> **exhaustive** feature selection

(1) Sepal length
(2) Sepal width
(3) Petal length
(4) Petal width

→

1. {0}
2. {1}
3. {2}
4. {3}
5. {0, 1}
6. {0, 2}
7. {0, 3}
8. {1, 2}
9. {1, 3}
10. {2, 3}
11. {0, 1, 2}
12. {0, 1, 3}
13. {0, 2, 3}
14. {1, 2, 3}
15. {0, 1, 2, 3}

Which feature selection would guarantee optimal model performance?

Trying all possible feature combinations --> **exhaustive** feature selection

(1) Sepal length
(2) Sepal width
(3) Petal length
(4) Petal width

→

$$\sum_{i=1}^{m} \binom{m}{i}$$ Combinations!

$$\binom{4}{1} + \binom{4}{2} + \binom{4}{3} + \binom{4}{4} = 15$$

Which feature selection would guarantee optimal model performance?

Trying all possible feature combinations --> **exhaustive** feature selection

(1) Alcohol
(2) Malic acid
...
(13) Color intensity

→

$$\sum_{i=1}^{m} \binom{m}{i}$$ Combinations!

$$\binom{13}{1} + \binom{13}{2} + \ldots + \binom{13}{13} = 8191$$

Which feature selection would guarantee optimal model performance?

Trying all possible feature combinations --> **exhaustive** feature selection

- Very expensive!
- We will look at an approximation called *Sequential Feature Selection*

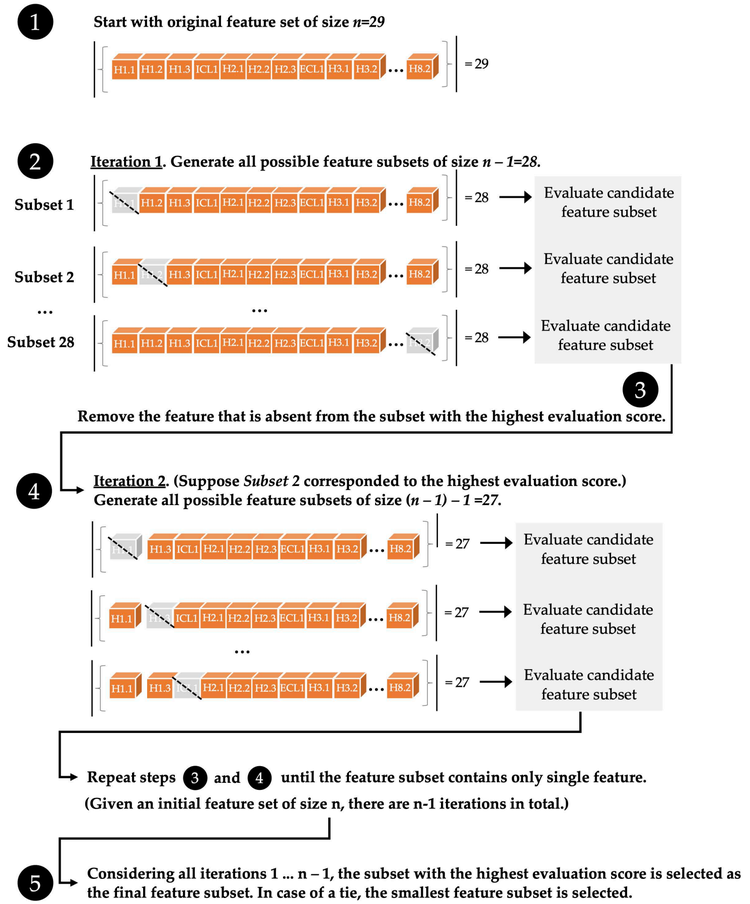
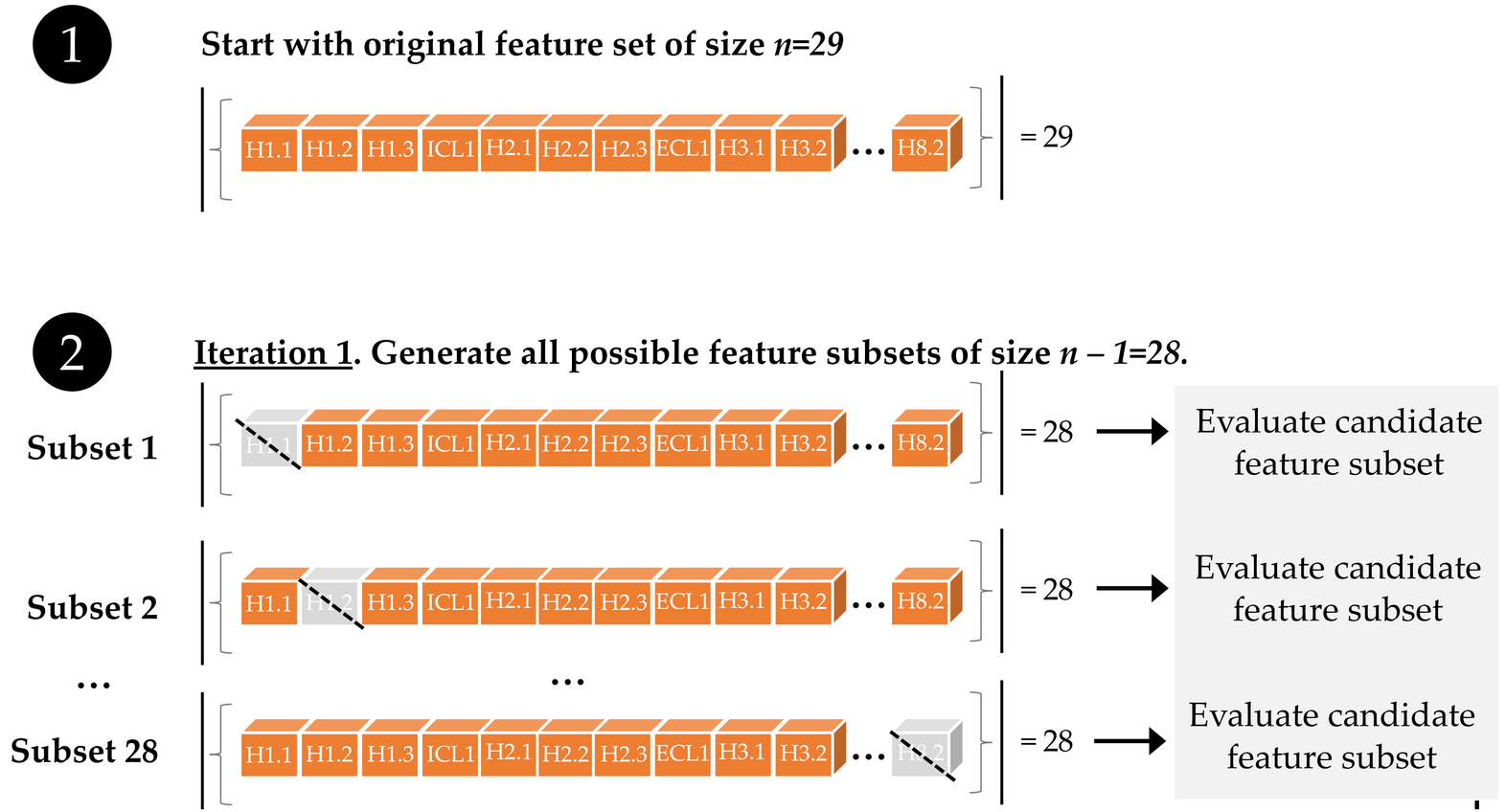


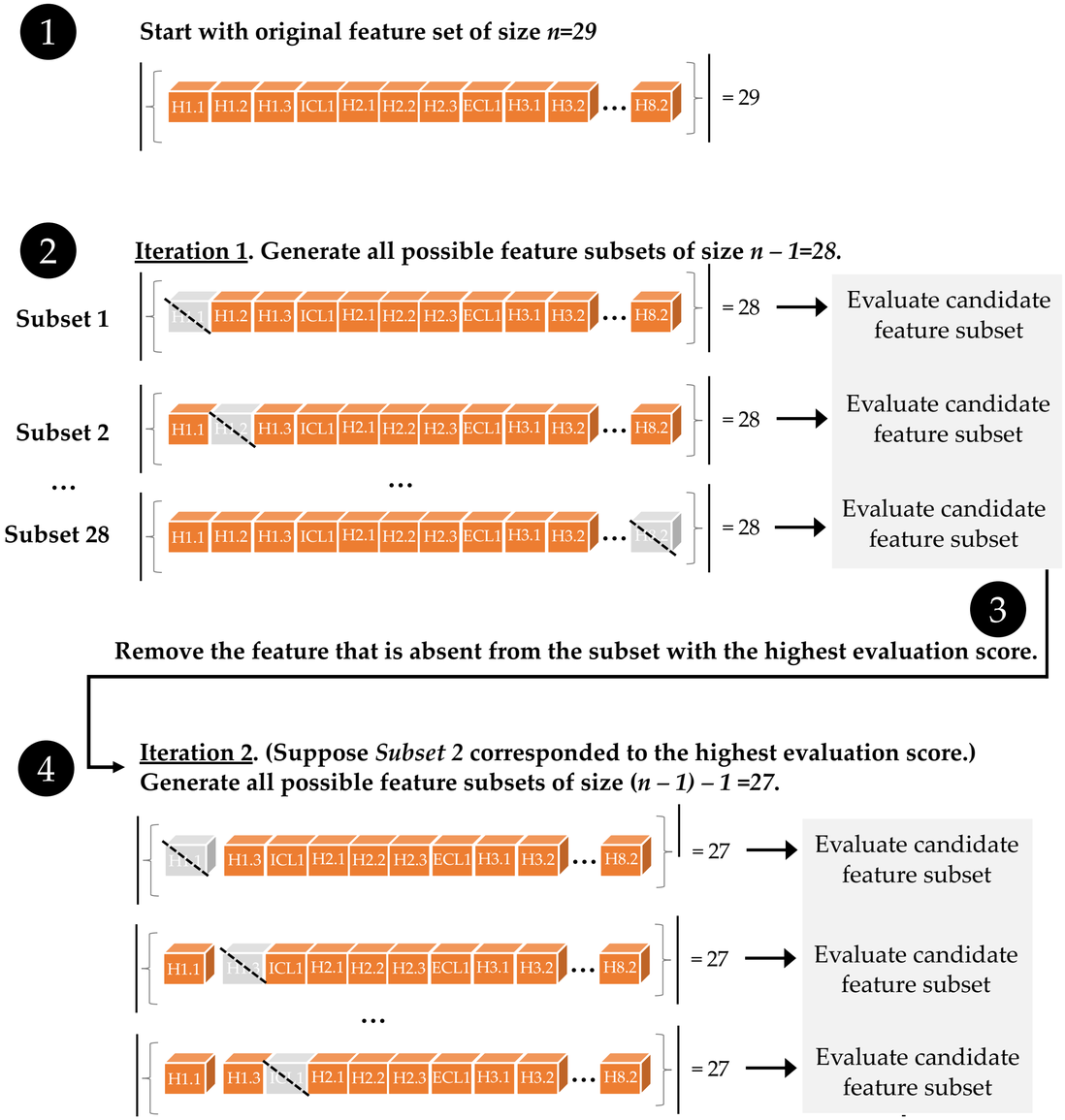
Figure 4. Illustration of backward sequential feature selection for identifying feature subsets that maximize the performance of a predictive model. In this study, the candidate feature subsets were evaluated by using leave-one-out cross-validation and the out-of-bag bootstrap method with a three-nearest neighbor classifier. The classifier accuracy in predicting active/inactive cases in the GPCR held-out test data was used to evaluate each feature subset, as detailed in Table 2.

Joe Bemister-Buffington, Alex J. Wolf, Sebastian Raschka, and Leslie A. Kuhn (2020)
Machine Learning to Identify Flexibility Signatures of Class A GPCR Inhibition
Biomolecules 2020, 10, 454. (https://www.mdpi.com/2218-273X/10/3/454#)

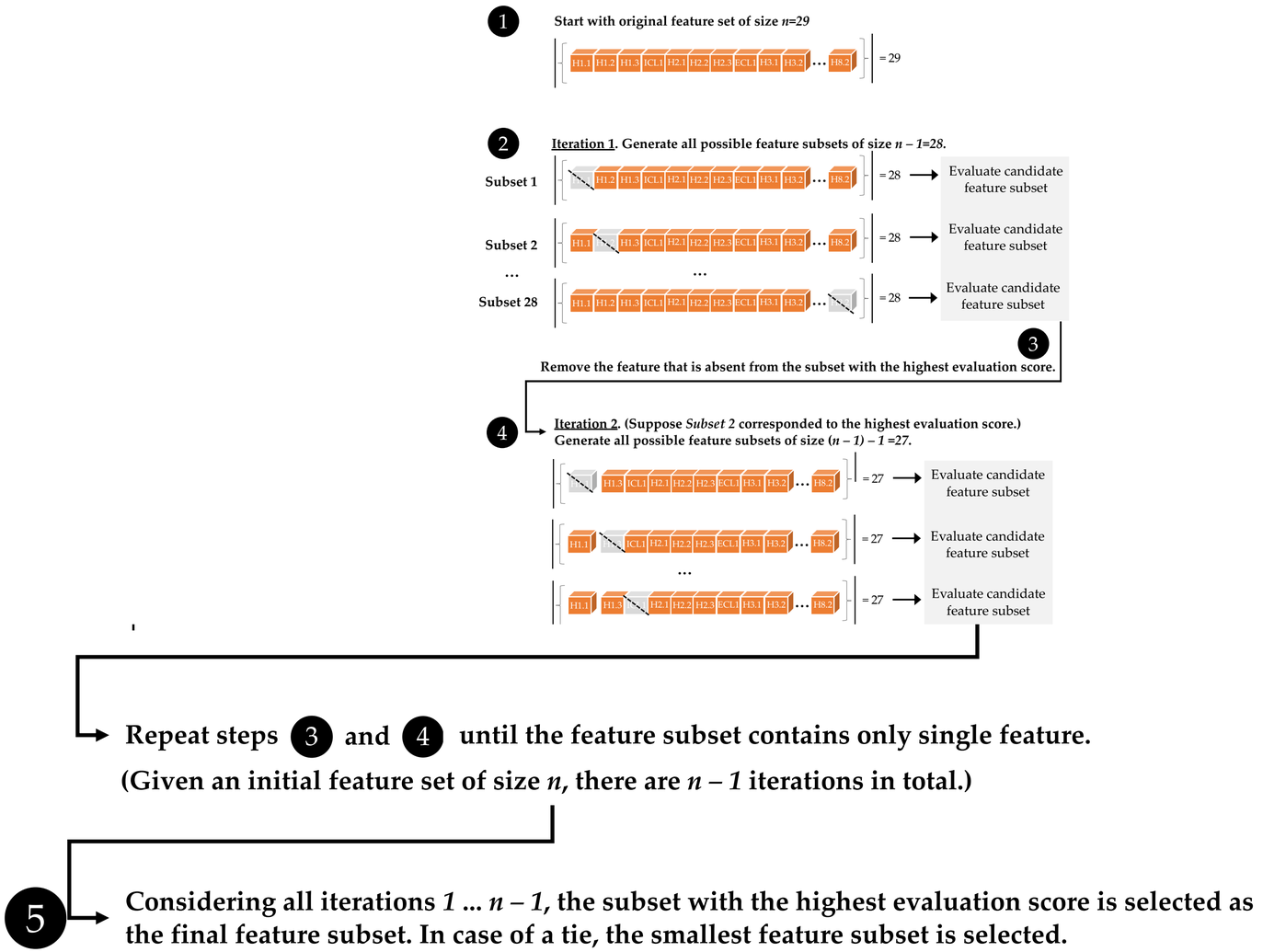
Sequential Backward Selection (1)



Sequential Backward Selection (2)



Sequential Backward Selection (2)



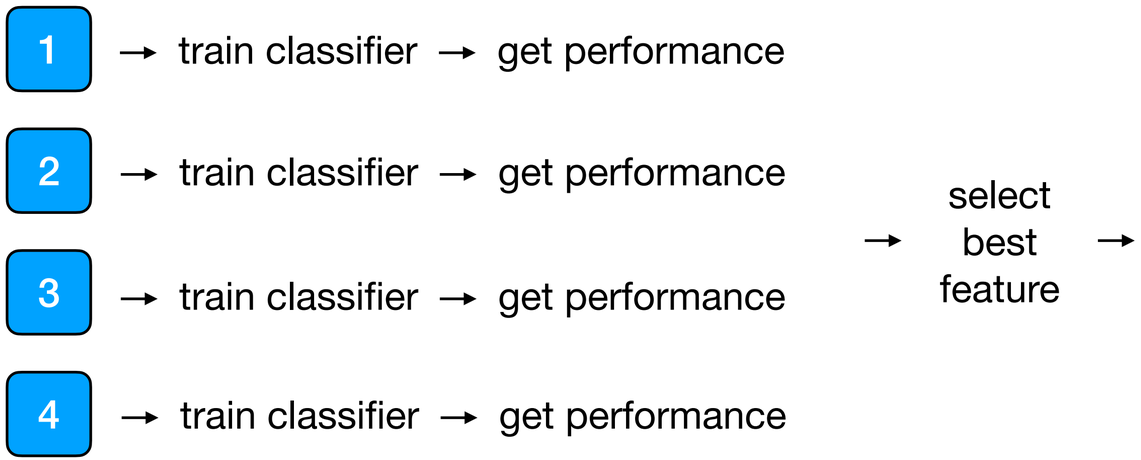
Different Flavors of Sequential Feature Selection

- ☑ Sequential Backward Selection
- Sequential Forward Selection
- Sequential Floating Forward Selection
- Sequential Floating Backward Selection

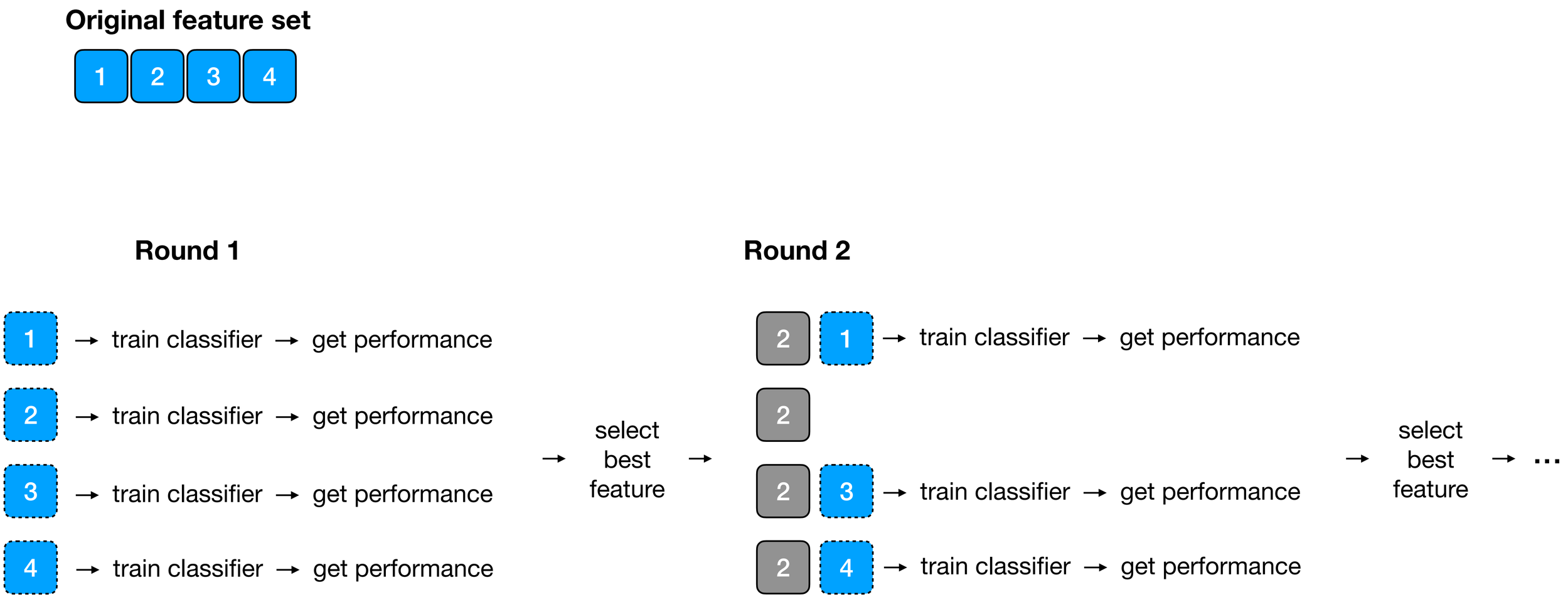
Sequential Forward Selection

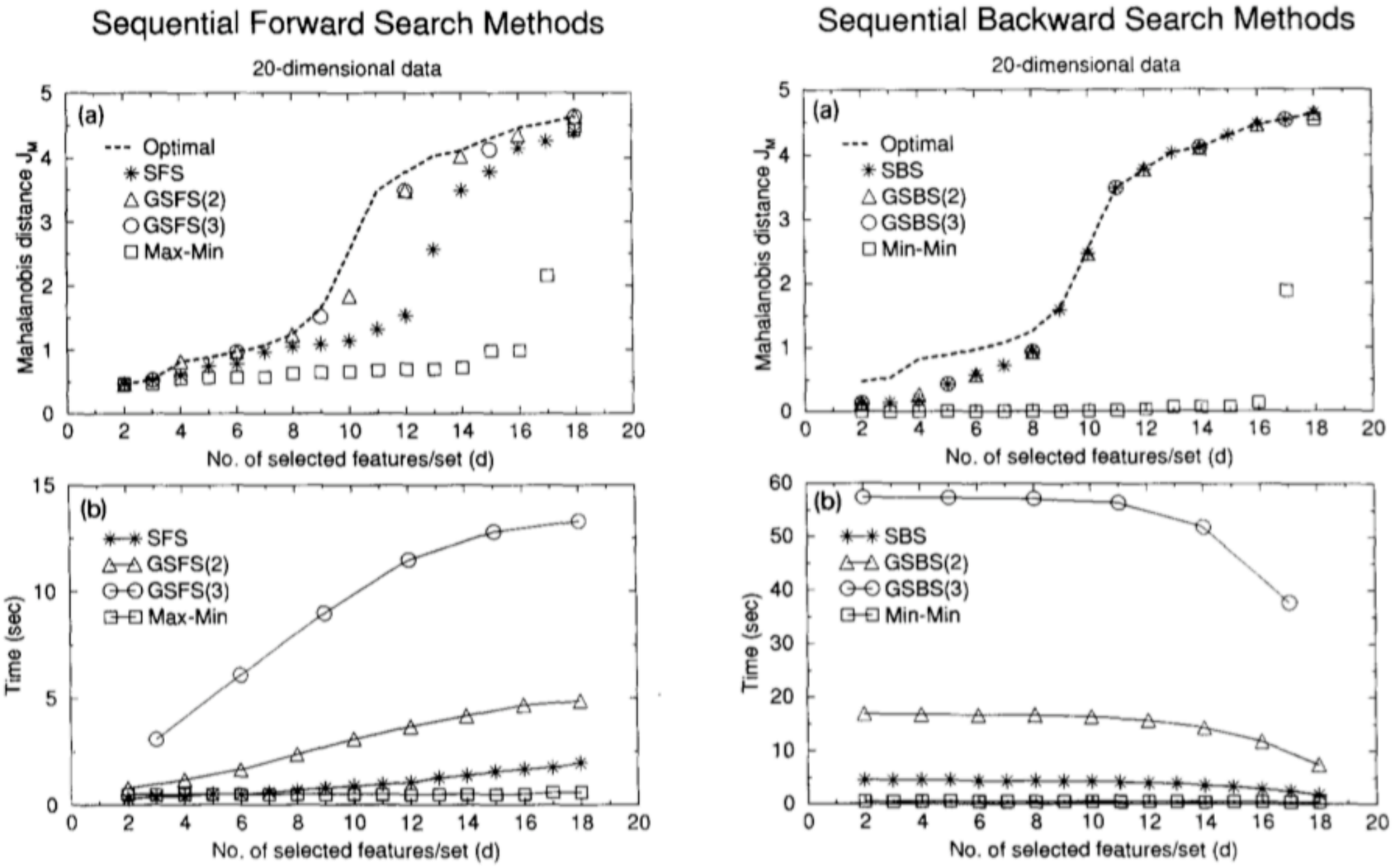
Original feature set

Round 1



Sequential Forward Selection





Pudil, P., Novovičová, J., & Kittler, J. (1994). *"Floating search methods in feature selection."* Pattern recognition letters 15.11 (1994): 1119-1125.

When To Use Forward/Backward Selection

Different Flavors of Sequential Feature Selection

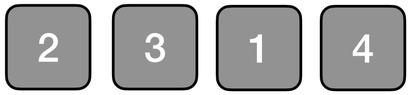
- ☑ Sequential Backward Selection
- ☑ Sequential Forward Selection
- Sequential Floating Forward Selection
- Sequential Floating Backward Selection

Sequential Floating Forward Selection (1)

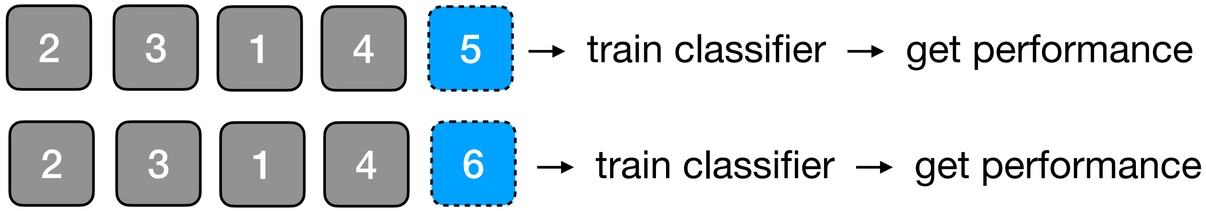
Original feature set

For reference; no floating selection here

After Round 4



<u>Regular</u> Round 5

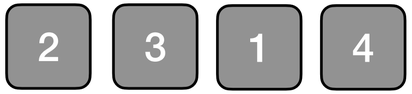


Sequential Floating Forward Selection (1)

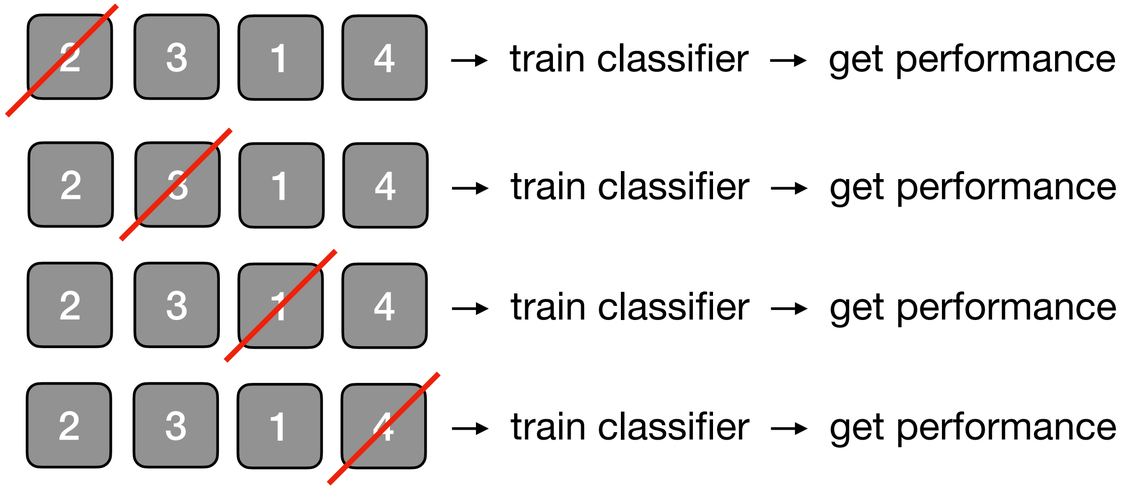
Original feature set

Remove feature if it improves performance

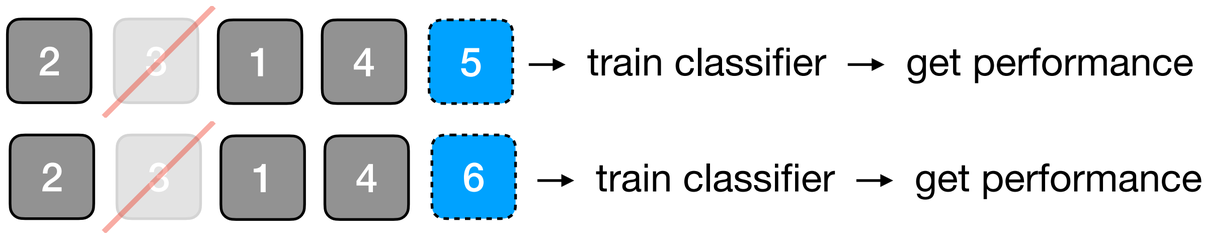
After Round 4



Floating Round 5



Regular Round 5



Different Flavors of Sequential Feature Selection

- ✅ Sequential Backward Selection
- ✅ Sequential Forward Selection
- ✅ Sequential Floating Forward Selection
- Sequential Floating Backward Selection

both approaches obtained similar results. Note, that the GA led to the optimal solution in comparable time (about 1500 subset evaluations) even taking into account the need to run it a number of times to achieve good performance. In this experiment, the GA was run 10 times for each value of t and, in more than half of the cases the GA obtained better or the same results than the SFFS ones (the figure can be misleading in this sense because each plotted symbol may represent more than one result).

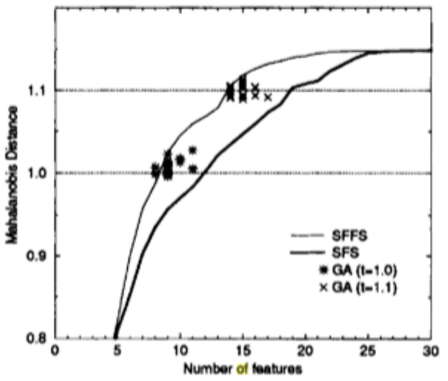


Figure 5. Results of Feature Selection obtained by SFS and SFFS methods for the $D = 30$ experiment. Crosses and asterisks show the results corresponding to different runs of the GA with two different values of the threshold parameter.

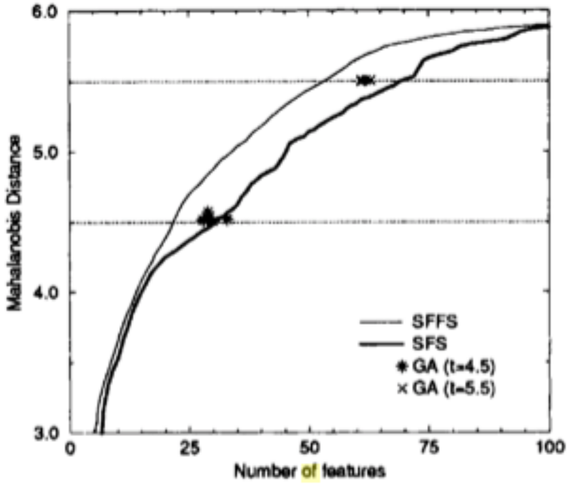


Figure 7. Results of Feature Selection obtained by SFS and SFFS methods for the $D = 120$ experiment. Crosses and asterisks show the results corresponding to different runs of the GA with two different values of the threshold parameter.

Ferri, F. J., Pudil P., Hatef, M., Kittler, J. (1994). "Comparative study of techniques for large-scale feature selection." Pattern Recognition in Practice IV : 403-413.

Sequential Forward Selection (SFS)

Input: $Y = \{y_1, y_2, \ldots, y_d\}$

- The ***SFS*** algorithm takes the whole d-dimensional feature set as input.

Output: $X_k = \{x_j \mid j = 1, 2, \ldots, k;\ x_j \in Y\}$, where $k = (0, 1, 2, \ldots, d)$

- SFS returns a subset of features; the number of selected features k, where $k < d$, has to be specified *a priori*.

Initialization: $X_0 = \emptyset, k = 0$

- We initialize the algorithm with an empty set $\emptyset$ ("null set") so that $k = 0$ (where k is the size of the subset).

Step 1 (Inclusion):

$x^+ = \arg\max J(X_k + x),\ \text{where } x \in Y - X_k$
$X_{k+1} = X_k + x^+$
$k = k + 1$
Go to Step 1

- in this step, we add an additional feature, x^+, to our feature subset X_k.
- x^+ is the feature that maximizes our criterion function, that is, the feature that is associated with the best classifier performance if it is added to X_k.
- We repeat this procedure until the termination criterion is satisfied.

Termination: $k = p$

- We add features from the feature subset X_k until the feature subset of size k contains the number of desired features p that we specified *a priori*.

http://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/

1. Different categories of feature selection
2. Filter methods
3. Embedded methods
    3.1. L1-regularized logistic regression
    3.2. Random forest feature importance
4. Wrapper methods
    4.1. Recursive feature elimination
    4.2. Permutation importance
    4.3. Permutation importance code example
    4.4. Sequential feature selection
    4.5. Sequential feature selection code example