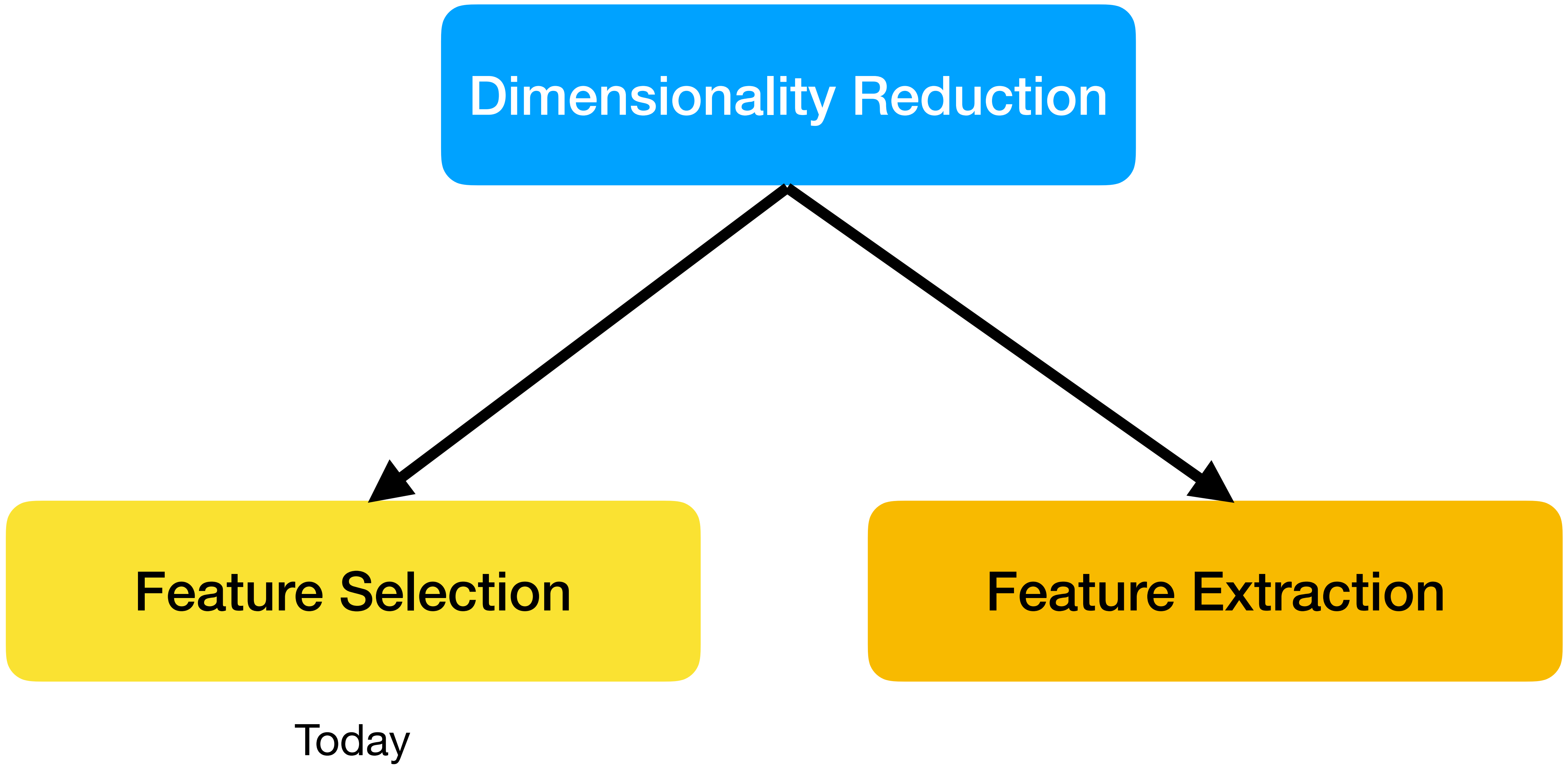
Lecture 13

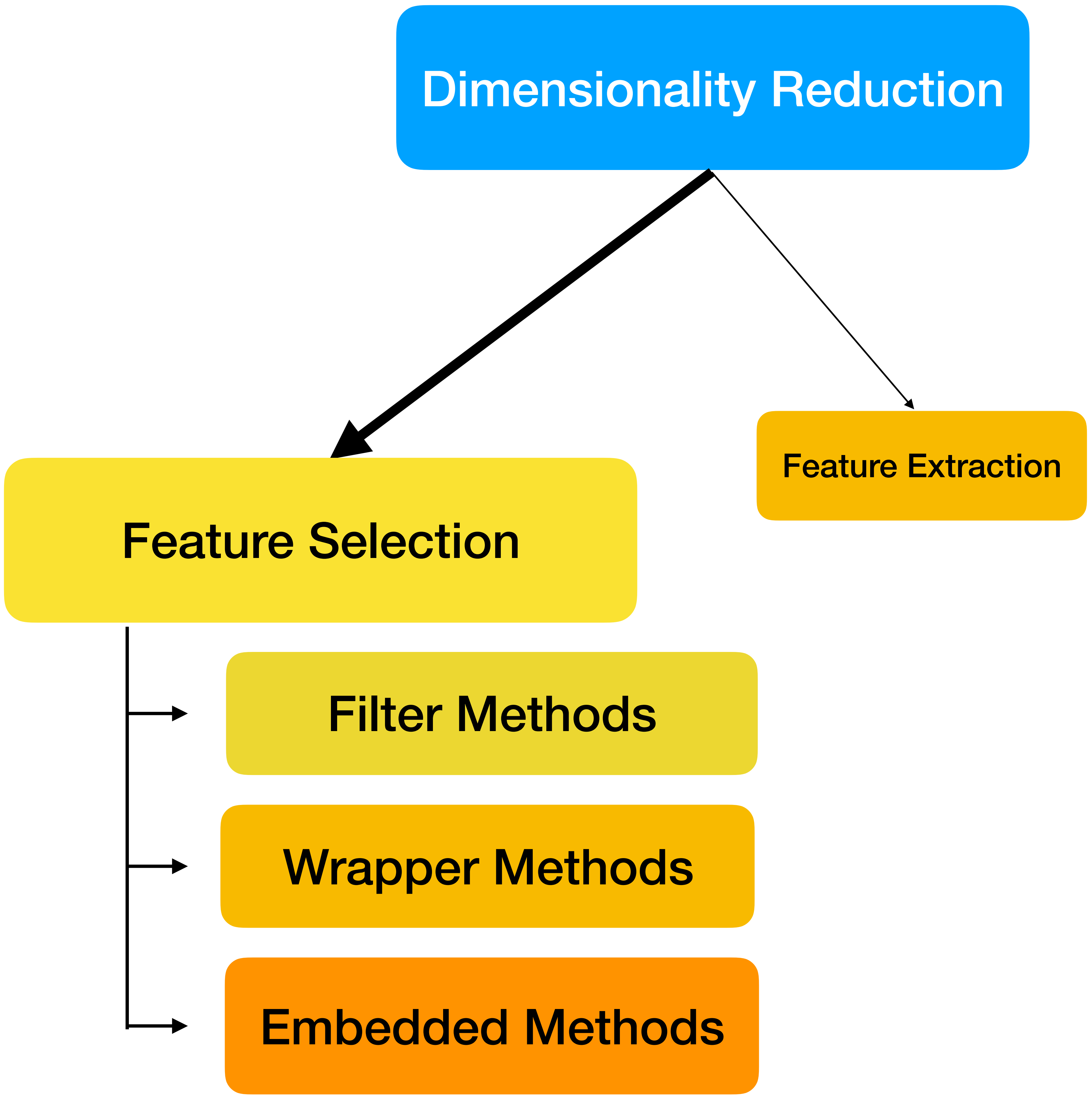
Dimensionality Reduction I: Feature Selection

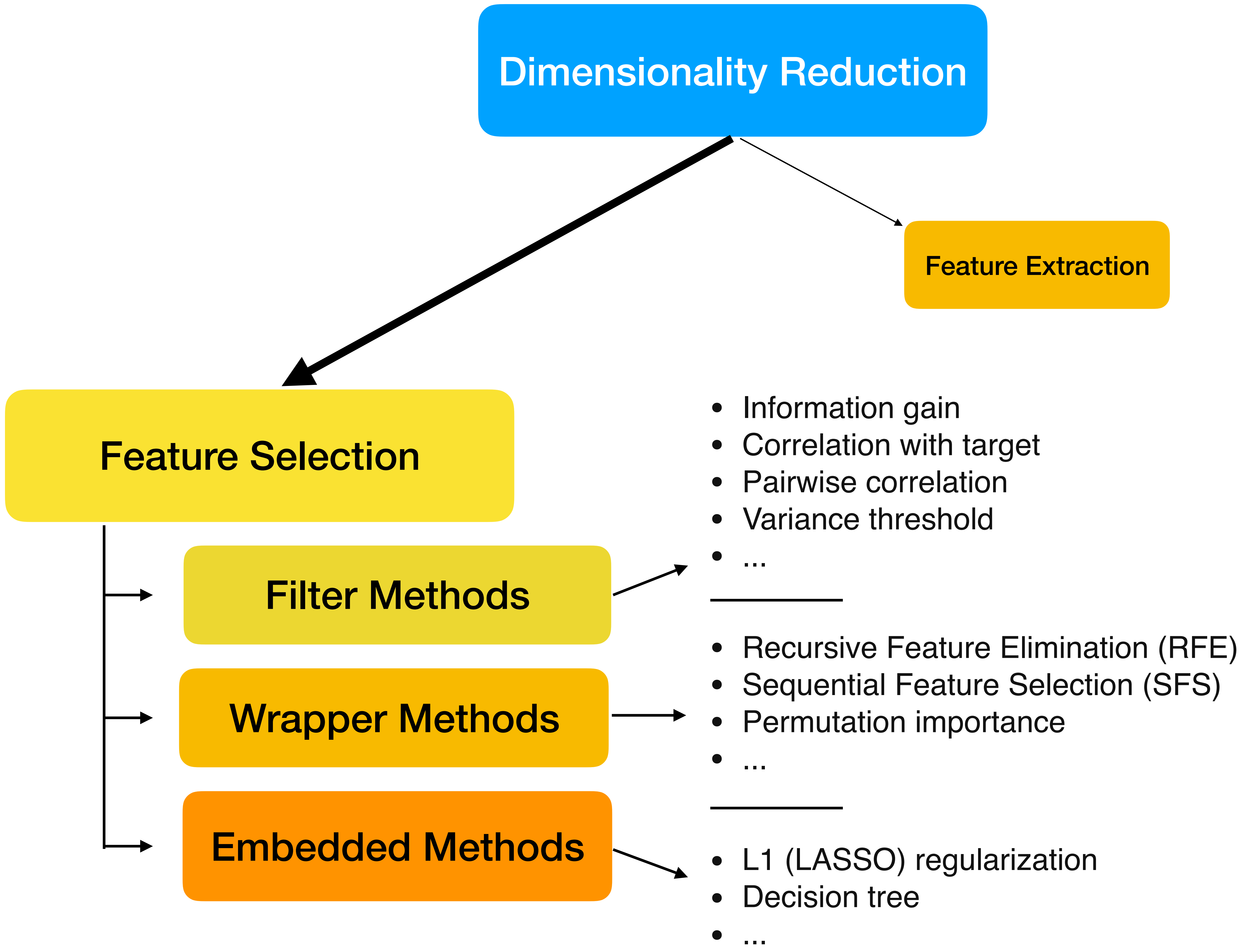
STAT 479: Machine Learning, Fall 2018

Sebastian Raschka

http://stat.wisc.edu/~sraschka/teaching/stat479-fs2018/


Dimensionality Reduction
Feature Selection
Feature Extraction
Today


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



- Information gain
- Correlation with target
- Pairwise correlation
- Variance threshold
- ...

- Recursive Feature Elimination (RFE)
- Sequential Feature Selection (SFS)
- Permutation importance
- ...

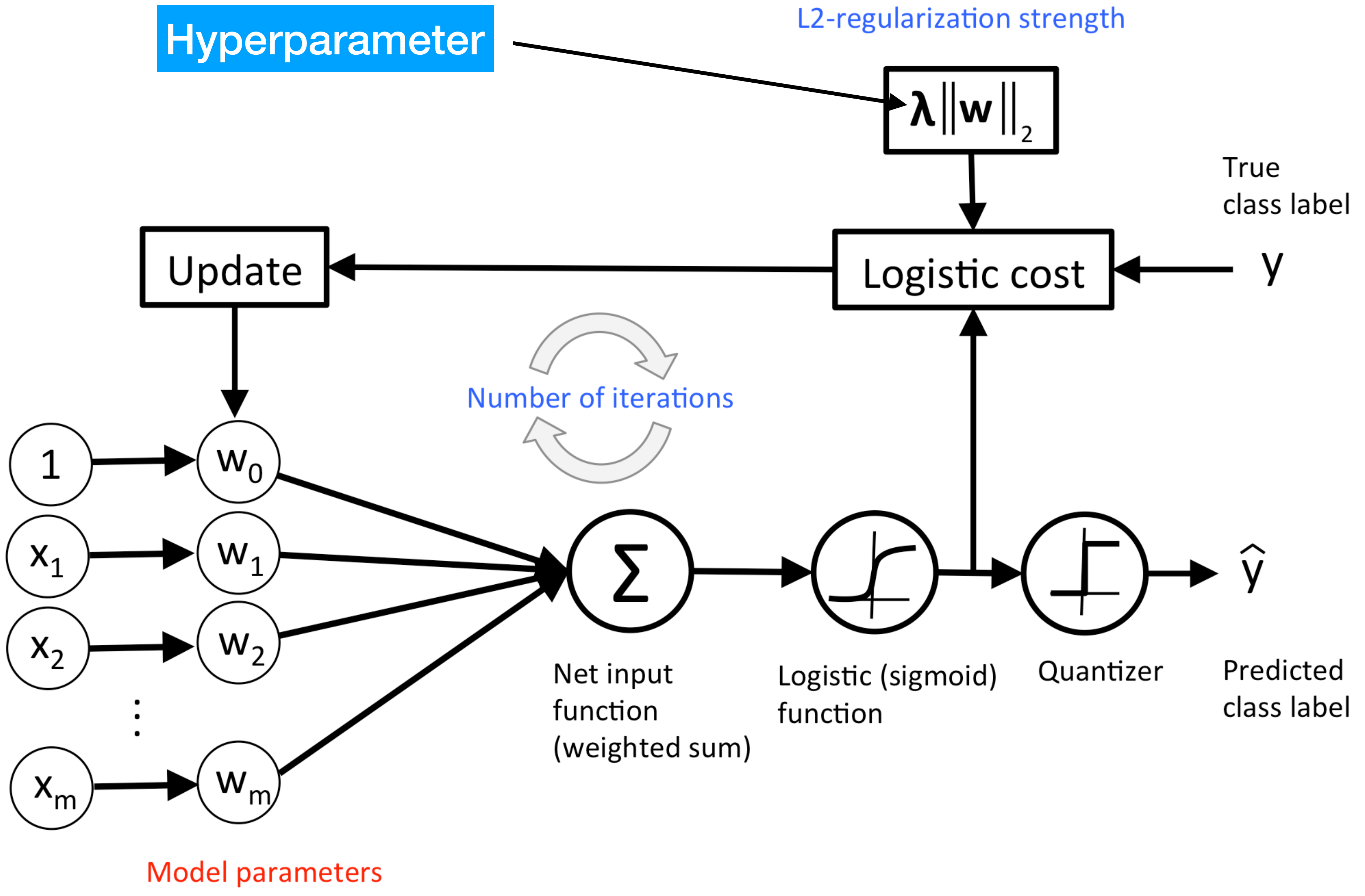
- L1 (LASSO) regularization
- Decision tree
- ...

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

Hyperparameters

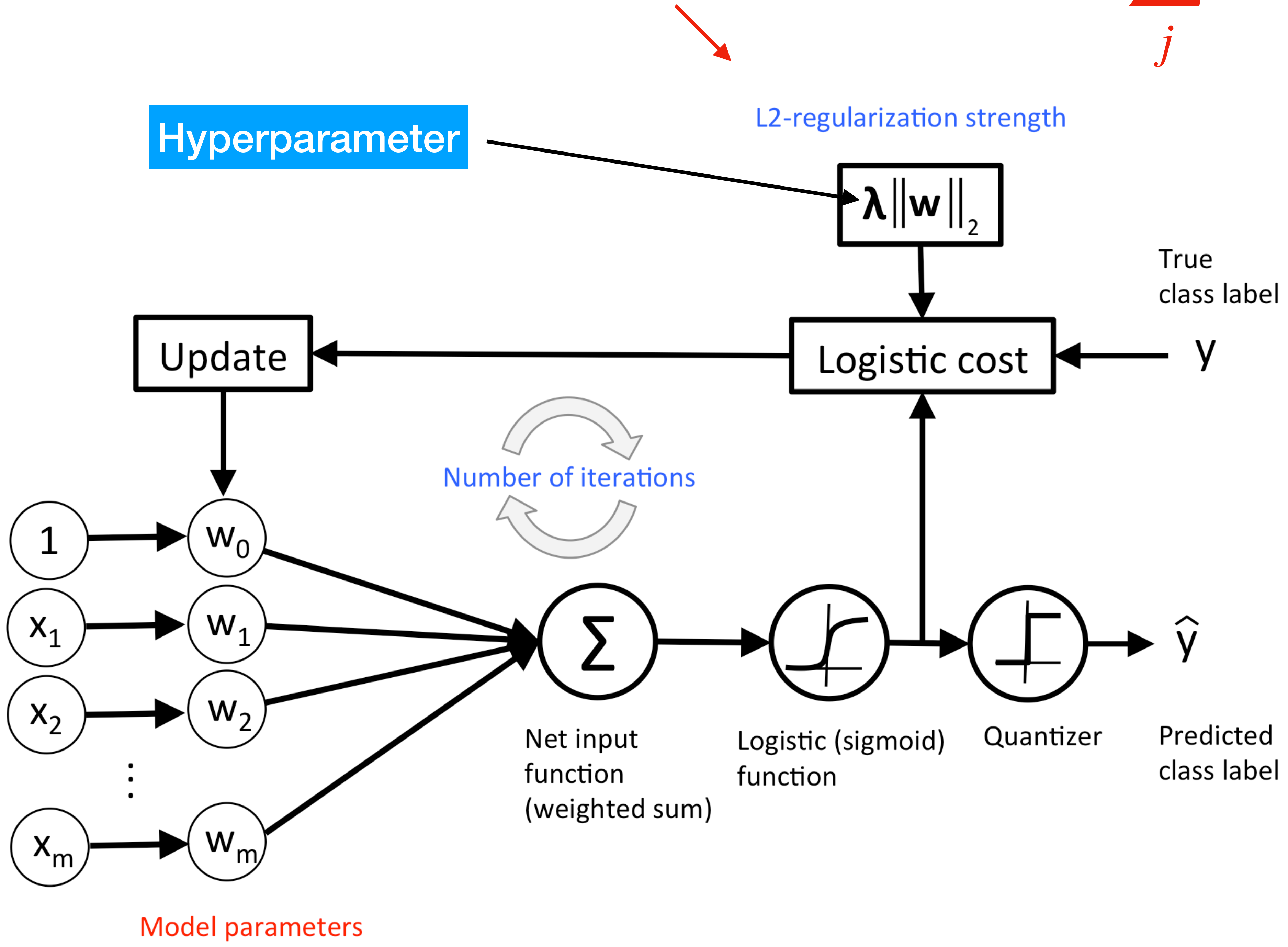
parametric model: logistic regression



L1 Regularization / LASSO (Embedded)

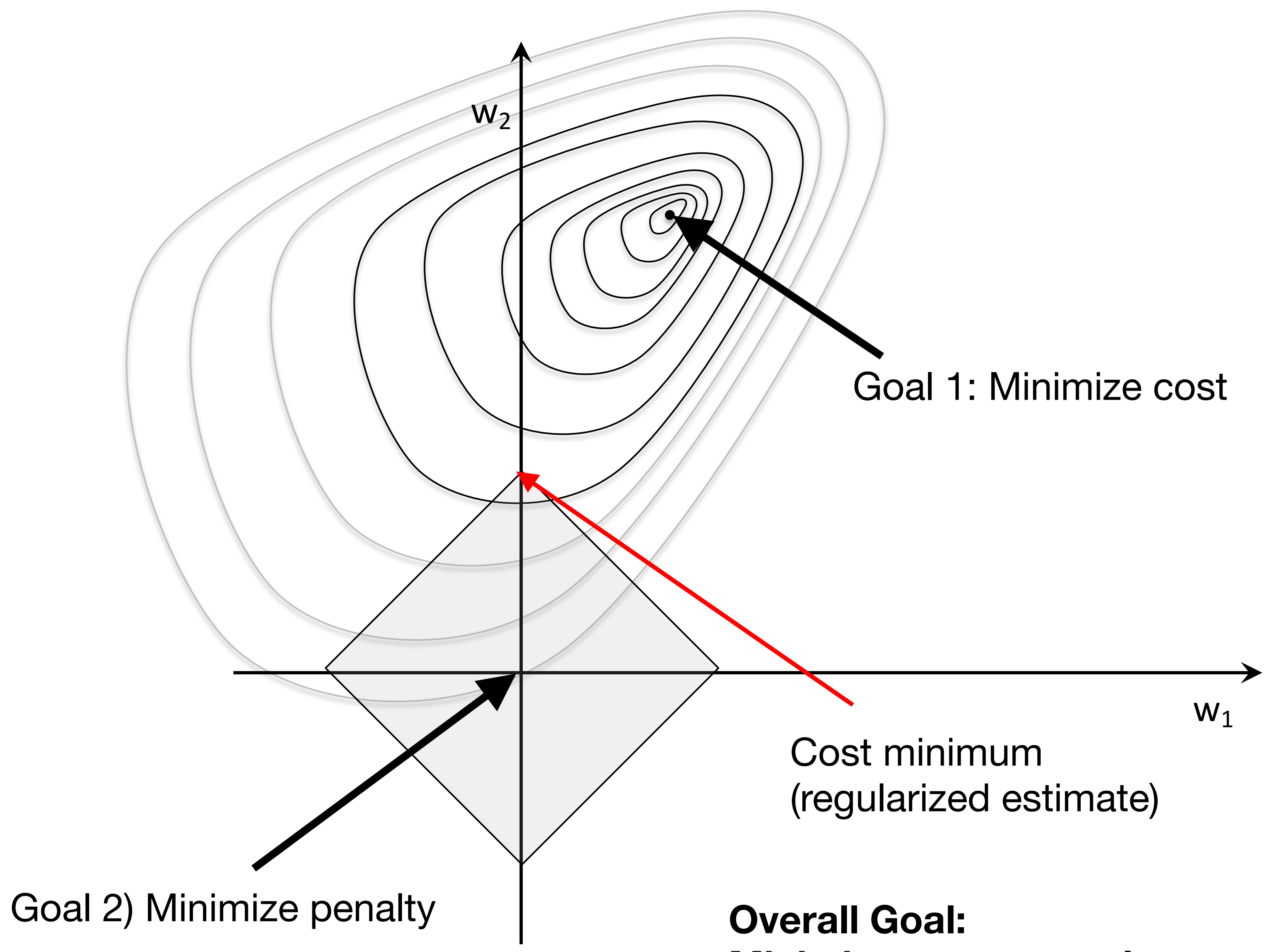
Least Absolute Shrinkage and Selection Operator

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



L1 Regularization / LASSO (Embedded)

Least Absolute Shrinkage and Selection Operator



Overall Goal:
Minimize cost + penalty

L1 Regularization / LASSO (Embedded)

Least Absolute Shrinkage and Selection Operator

Wine Dataset

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

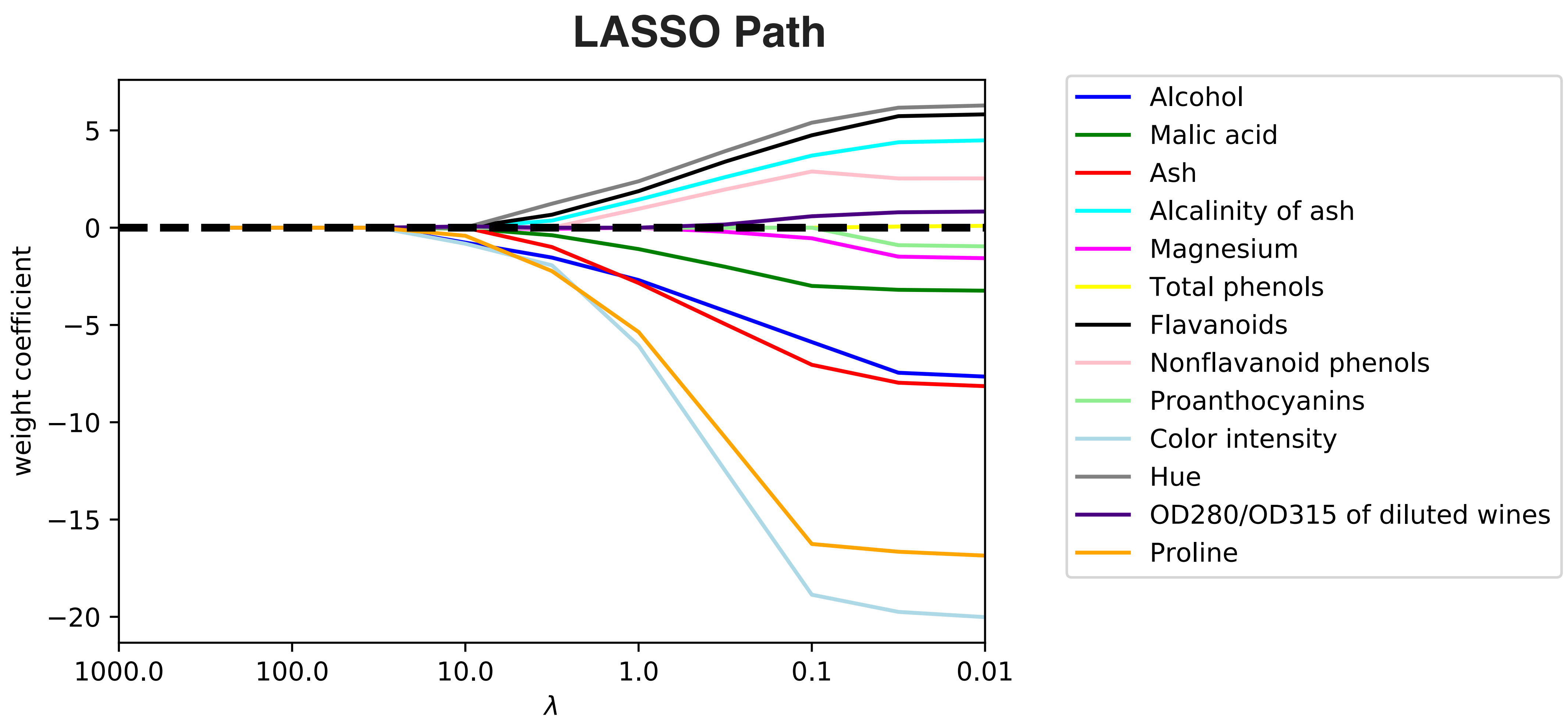
'Class label'	'Alcohol'	'Malic acid'	'Ash'	Alcalinity of ash'	'Magnesium'	'Total phenols'	'Flavanoids'	'Nonflavanoid phenols'	'Proanthocyanins'	'Color intensity'	'Hue'	OD280/OD315 of diluted wines'	'Proline'
1	14.23	1.71	2.43	15.6	127	2.8	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	13.2	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.4	1050
1	13.16	2.36	2.67	18.6	101	2.8	3.24	0.3	2.81	5.68	1.03	3.17	1185
3	13.27	4.28	2.26	20	120	1.59	0.69	0.43	1.35	10.2	0.59	1.56	835
3	13.17	2.59	2.37	20	120	1.65	0.68	0.53	1.46	9.3	0.6	1.62	840
3	14.13	4.1	2.74	24.5	96	2.05	0.76	0.56	1.35	9.2	0.61	1.6	560

L1 Regularization / LASSO (Embedded)

Least Absolute Shrinkage and Selection Operator

Wine Dataset

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



(don't forget to normalize/standardize features)

Recursive Feature Elimination (Wrapper)

Consider a (generalized) linear model:

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

Random Forest Feature Importance "Method A"

- Usually measured as
  - impurity decrease (Gini, Entropy) for a given node/feature decision
  - weighted by number of examples at that node
  - averaged over all trees
  - then normalize so that sum of feature importances sum to 1

(used in scikit-learn)

- (Unfair for variables with many vs few values)

Permutation Test (Interlude)

- A nonparametric test procedure to test the null hypothesis that two different groups come from the same distribution

- Can be used for significance or hypothesis testing w/o requiring to make any assumptions about the sampling distribution (e.g., it doesn't require the samples to be normal distributed).

- Under the null hypothesis (treatment = control), any permutations are equally likely

- Note that there are (n+m)! permutations, where *n* is the number of records in the treatment sample, and *m* is the number of records in the control sample

- For a two-sided test, we define the alternative hypothesis that the two samples are different (e.g., treatment != control)

Permutation Test

1. Compute the difference (here: mean) of sample x (size *n*) *and* sample y (size *m*)

2. Combine all measurements into a single dataset

3. Draw a permuted dataset from all possible permutations of the dataset in 2.

4. Divide the permuted dataset into two datasets x' and y' of size *n* and *m*, respectively

5. Compute the difference (here: mean) of sample x' and sample y' and record this difference

6. Repeat steps 3-5 until all permutations are evaluated

7. Return the p-value as the number of times the recorded differences were more extreme than the original difference from 1., then divide this number by the total number of permutations

Here, the p-value is defined as the probability, given the null hypothesis (no difference between the samples) is true, that we obtain results that are at least as extreme as the results we observed (i.e., the sample difference from 1.).

Permutation Test

1. Compute the difference (here: mean) of sample x (size *n*) *and* sample y (size *m*)

2. Combine all measurements into a single dataset

3. Draw a permuted dataset from all possible permutations of the dataset in 2.

4. Divide the permuted dataset into two datasets x' and y' of size *n* and *m*, respectively

5. Compute the difference (here: mean) of sample x' and sample y' and record this difference

6. Repeat steps 3-5 until all permutations are evaluated

7. Return the p-value as the number of times the recorded differences were more extreme than the original difference from 1., then divide this number by the total number of permutations

Here, the p-value is defined as the probability, given the null hypothesis (no difference between the samples) is true, that we obtain results that are at least as extreme as the results we observed (i.e., the sample difference from 1.).

Permutation Test

Here, the p-value is defined as the probability, given the null hypothesis (no difference between the samples) is true, that we obtain results that are at least as extreme as the results we observed (i.e., the sample difference from 1.).

$$p(t > t_0) = \frac{1}{(n+m)!} \sum_{j=1}^{(n+m)!} I(t_j > t_0),$$

where t_0 is the observed value of the test statistic, and
t is the t-value, the statistic computed from the resamples, and I is the indicator function.

Feature Importance Through Permutation (Wrapper)

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 *i*:
    a. randomly permute feature column *i* 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

Feature Importance Through Permutation (Wrapper)

intuitive & model-agnostic

Column-Drop variant:

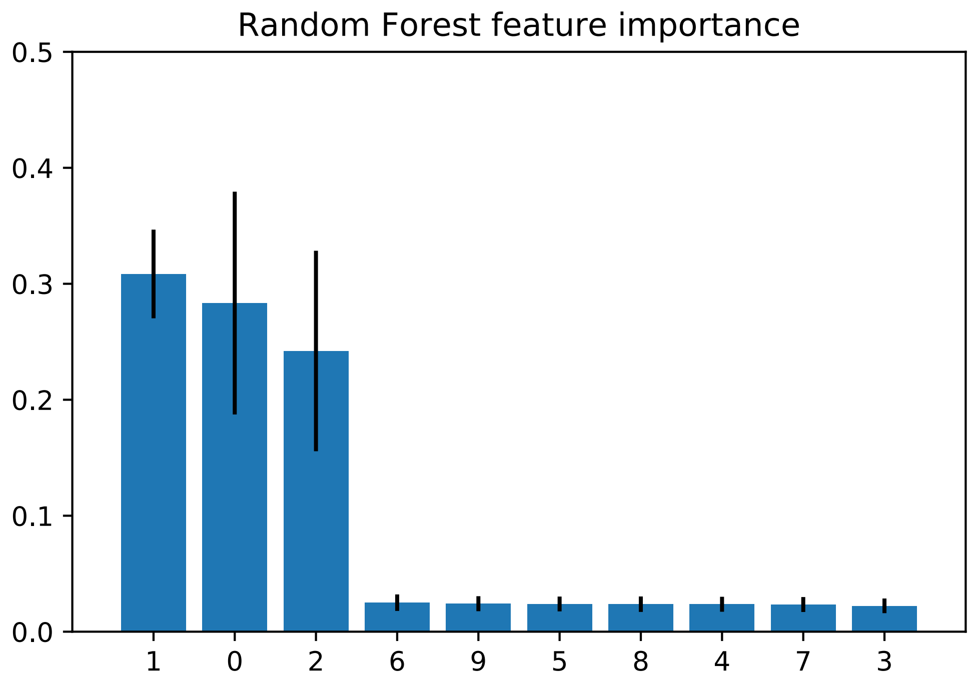
For each feature column *i:*

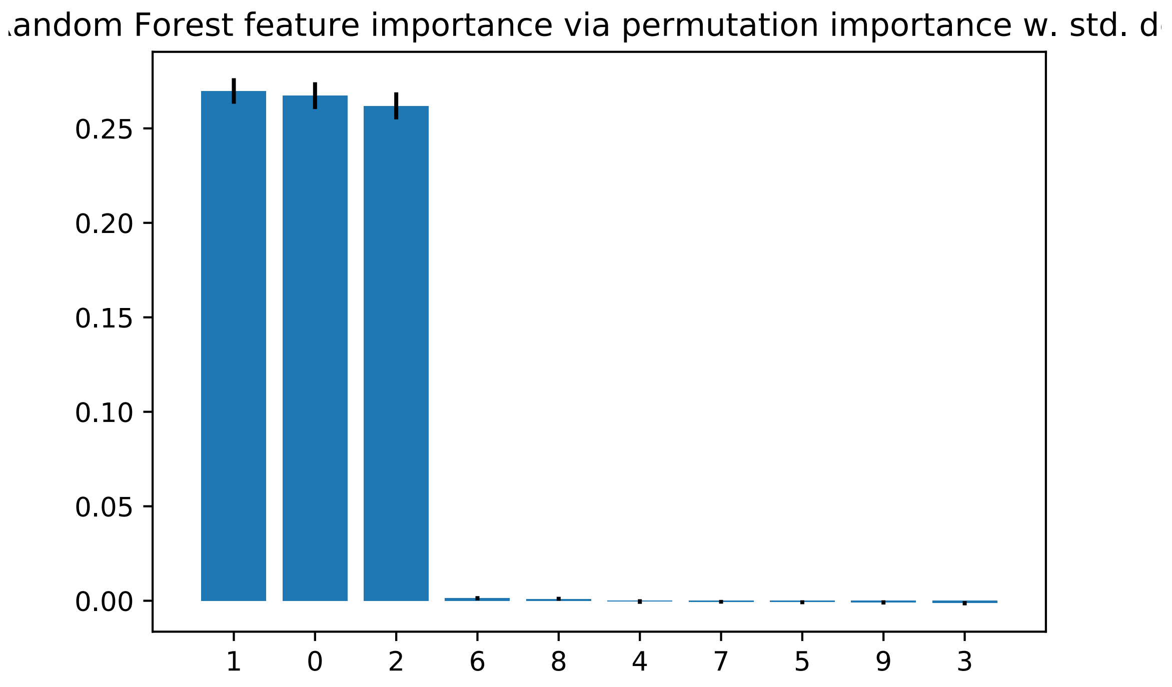
1. temporarily remove column
2. fit model to reduced dataset
3. compute validation set performance and compare to before

Random Forest Importance vs Permutation

```
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier

# Build a classification task using 3 informative features
X, y = make_classification(n_samples=10000,
                           n_features=10,
                           n_informative=3,
                           n_redundant=0,
                           n_repeated=0,
                           n_classes=2,
                           random_state=0,
                           shuffle=False)
```





- Permutation performance is a **universal** method, RF just shown as an example
- Permutation performance much more expensive, and in case of RF, usually computationally wasteful but can be more robust
- In the special case of RF, we can also use the OOB examples instead of a validation set (next slide)

Random Forest Feature Importance

"Method B"

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
- The proportion over all examples where the majority vote is wrong is the OOB accuracy estimate

Out-of-bag feature importance via permutation:

- 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

Sequential Forward Selection (Wrapper)

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):

$x^+ = \text{ arg max } J(x_k + x), \text{ where } x \in Y - X_k$
$X_{k+1} = X_k + x^+$
$k = k + 1$
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*.

Sequential Backward Selection

Input: the set of all features, $Y = \{y_1, y_2, \ldots, y_d\}$

- The SBS algorithm takes the whole feature set as input.

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

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

Initialization: $X_0 = Y$, $k = d$

- We initialize the algorithm with the given feature set so that the $k = d$.

Step 1 (Exclusion):

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

- In this step, we remove a feature, x^- from our feature subset X_k.
- x^- is the feature that maximizes our criterion function upon re,oval, that is, the feature that is associated with the best classifier performance if it is removed from 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*.

Sequential Floating Forward Selection

Input: the set of all features, $Y = \{y_1, y_2, \ldots, y_d\}$

- The ***SFFS*** algorithm takes the whole feature set as input, if our feature space consists of, e.g. 10, if our feature space consists of 10 dimensions (***d = 10***).

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

- The returned output of the algorithm is a subset of the feature space of a specified size. E.g., a subset of 5 features from a 10-dimensional feature space (***k = 5, d = 10***).

Initialization: $X_0 = Y$, $k = d$

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

Step 1 (Inclusion):

```
$x^+ = \text{arg max } J(x_k + x), \text{ where } x \in Y - X_k$
$X_{k+1} = X_k + x^+$
$k = k + 1$
Go to Step 2
```

Step 2 (Conditional Exclusion):

```
$x^- = \text{arg max } J(x_k - x), \text{ where } x \in X_k$
$if\ J(x_k - x) > J(x_k - x):$
    $X_{k-1} = X_k - x^-$
    $k = k - 1$
Go to Step 1
```

- In step 1, we include the feature from the ***feature space*** that leads to the best performance increase for our ***feature subset*** (assessed by the ***criterion function***). Then, we go over to step 2
- In step 2, we only remove a feature if the resulting subset would gain an increase in performance. If $k = 2$ or an improvement cannot be made (i.e., such feature x^+ cannot be found), go back to step 1; else, repeat this step.
- Steps 1 and 2 are repeated until the **Termination** criterion is reached.

Sequential Floating Backward Selection

Input: the set of all features, $Y = \{y_1, y_2, \ldots, y_d\}$

- The SBFS algorithm takes the whole feature set as input.

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

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

Initialization: $X_0 = Y, k = d$

- We initialize the algorithm with the given feature set so that the $k = d$.

Step 1 (Exclusion):

```
$x^- = \text{arg max } J(x_k - x),\ \text{where } x \in X_k$
$X_{k-1} = X_k - x^-$
$k = k - 1$
Go to Step 2
```

- In this step, we remove a feature, x^- from our feature subset X_k.
- x^- is the feature that maximizes our criterion function upon re,oval, that is, the feature that is associated with the best classifier performance if it is removed from X_k.

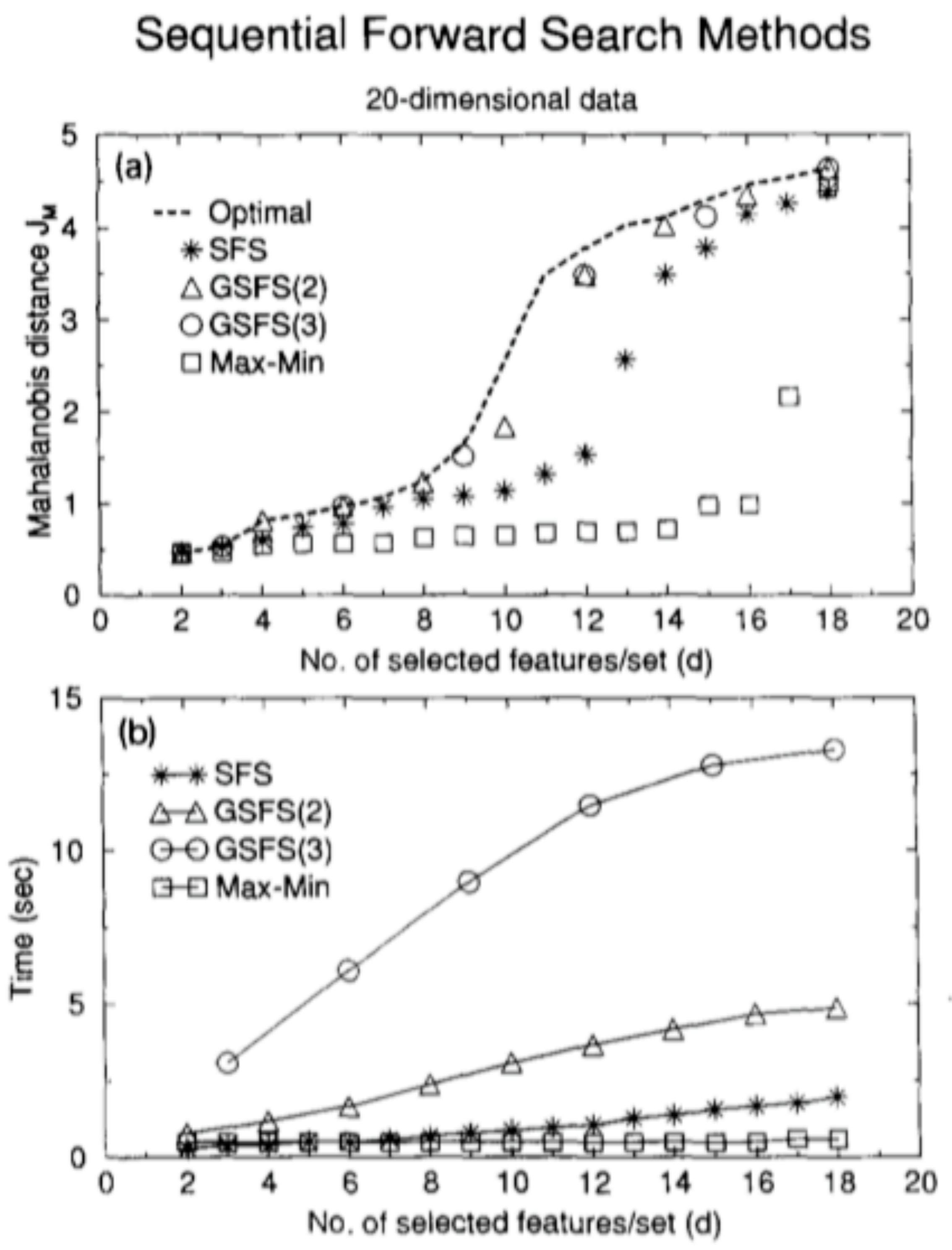
Step 2 (Conditional Inclusion):

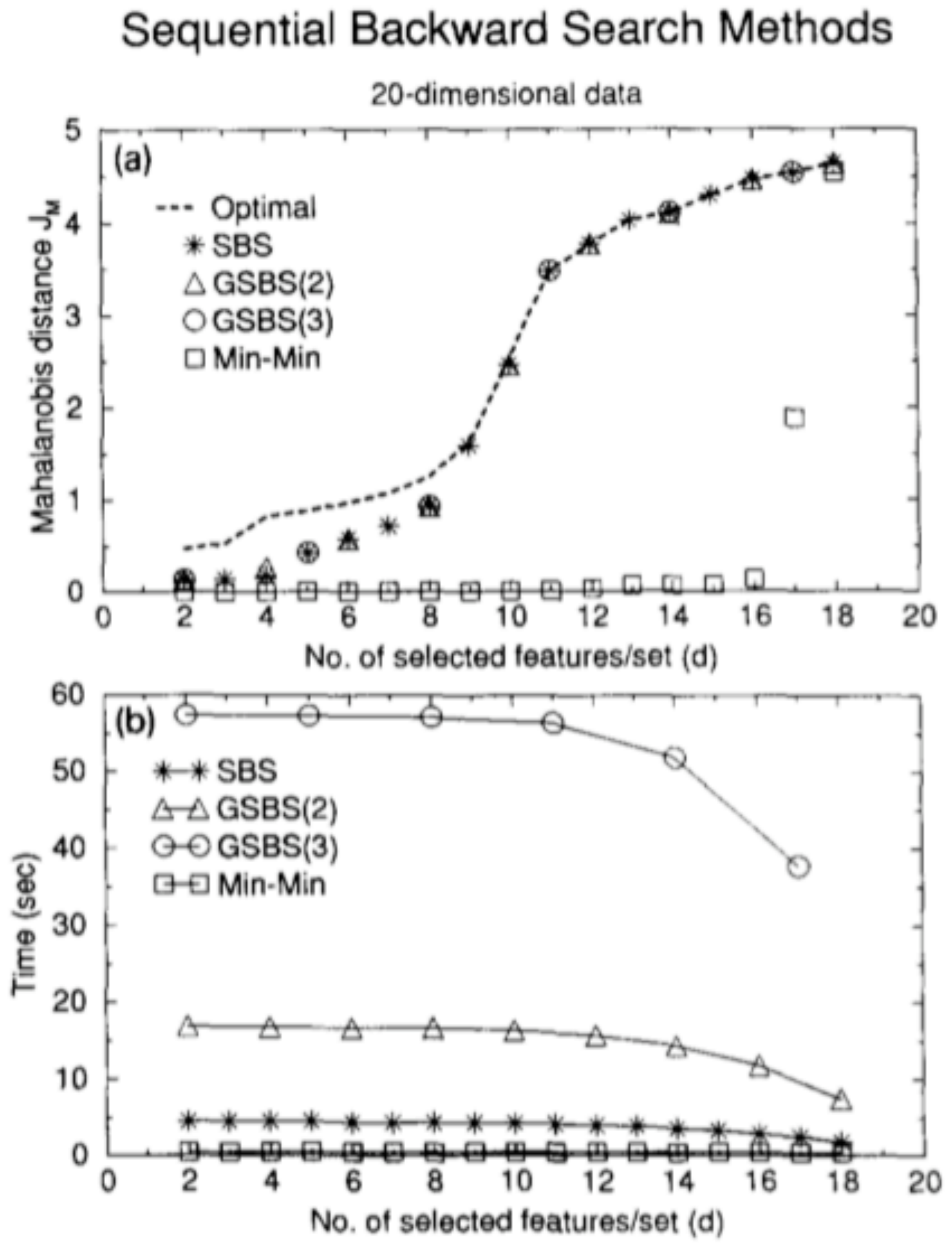
```
$x^+ = \text{arg max } J(x_k + x),\ \text{where } x \in Y - X_k$
if J(x_k + x) > J(x_k + x):
    $X_{k+1} = X_k + x^+$
    $k = k + 1$
Go to Step 1
```

- In Step 2, we search for features that improve the classifier performance if they are added back to the feature subset. If such features exist, we add the feature x^+ for which the performance improvement is maximized. If $k = 2$ or an improvement cannot be made (i.e., such feature x^+ cannot be found), go back to step 1; else, repeat this step.

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*.





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

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).

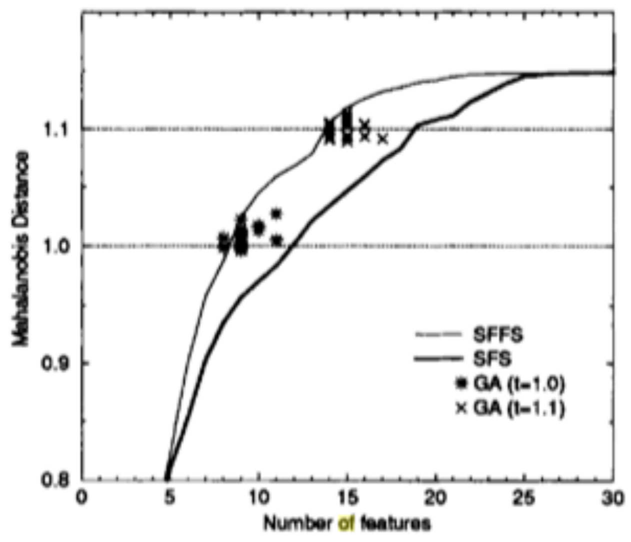


Figure 5. Results of Feature Selection obtained by SFS and SFFS methods for the $D = 30$ experiment. Crosses and asterisks show the results corresponding to different runs of the GA with two different values of the threshold parameter.

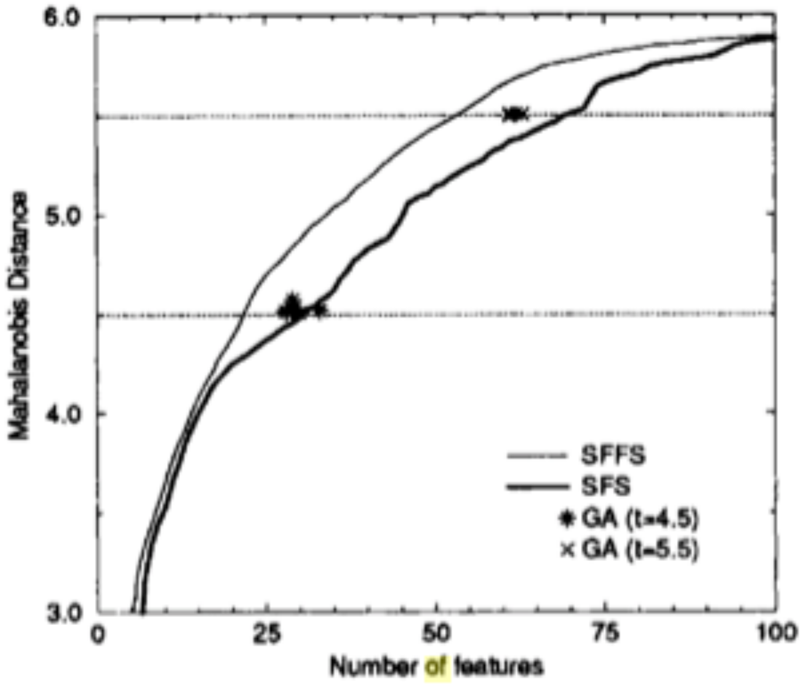


Figure 7. Results of Feature Selection obtained by SFS and SFFS methods for the $D = 120$ experiment. Crosses and asterisks show the results corresponding to different runs of the GA with two different values of the threshold parameter.

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.