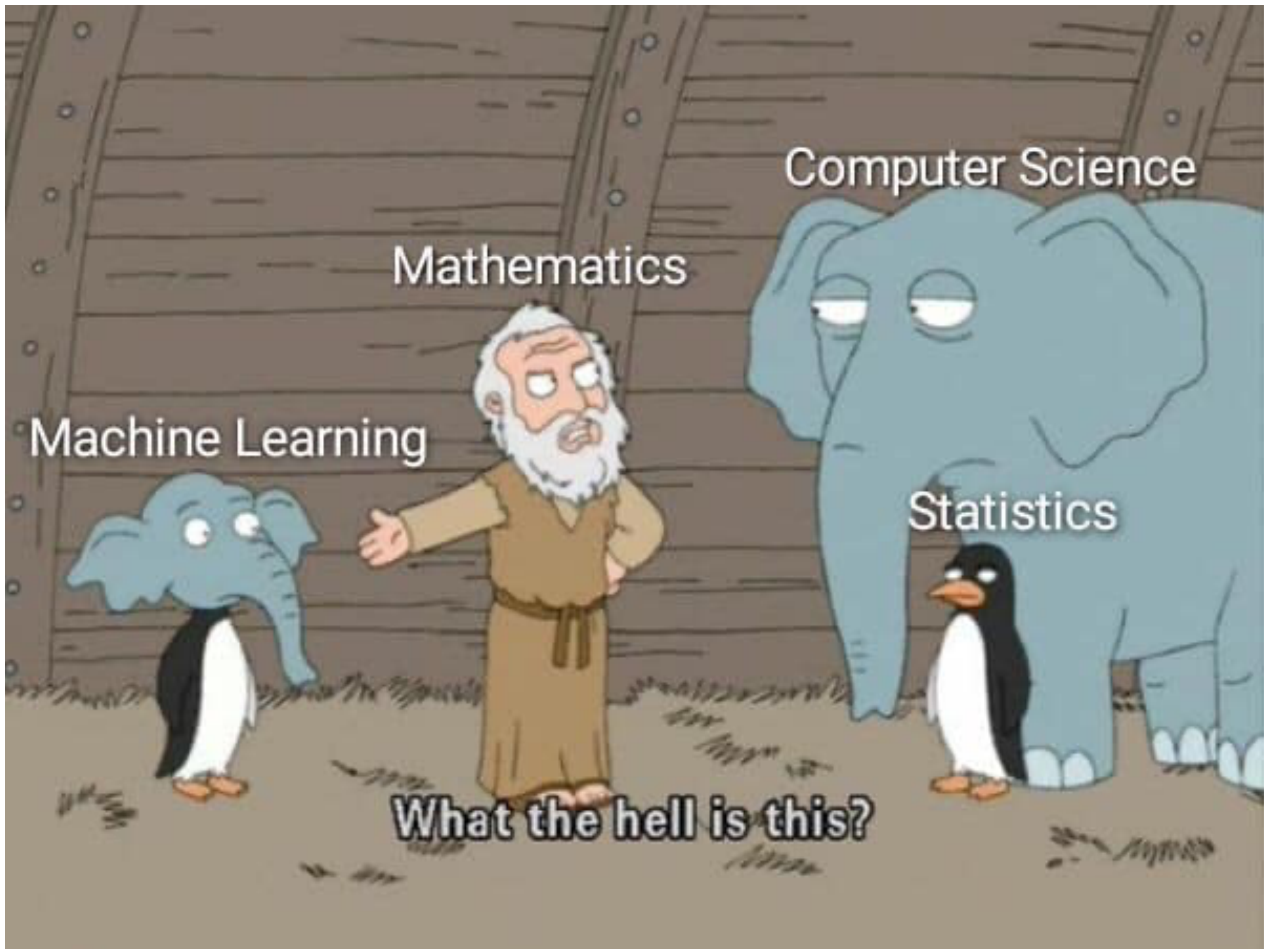
Lecture 13

Dimensionality Reduction I: Feature Selection

STAT 479: Machine Learning, Fall 2019

Sebastian Raschka

http://stat.wisc.edu/~sraschka/teaching/stat479-fs2019/



https://twitter.com/rvcraiu/status/1194787514604765192?s=20

r/MachineLearning

Posts

Posted by u/etienne_ben 20 hours ago

56

[D] What stupid things did you use to do?

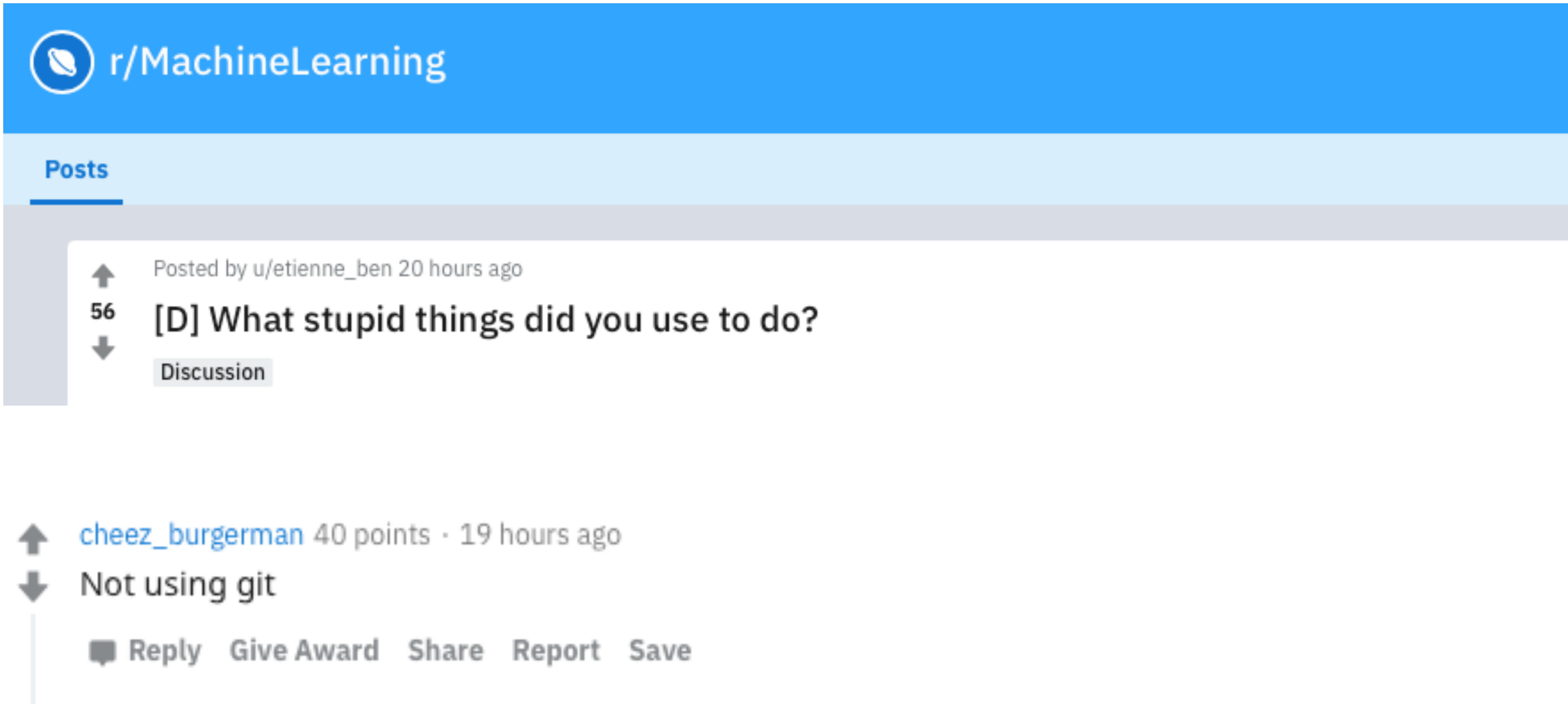
Discussion

When I was a student, I used to spend weeks tweaking hyperparameters to improve my models for Kaggle competitions by 1 or 2%. Not once did I look at the data to see what my model was doing wrong.

Share your shame.

69 Comments Give Award Share Save Hide Report

https://www.reddit.com/r/MachineLearning/comments/dvyi48/d_what_stupid_things_did_you_use_to_do/



https://www.reddit.com/r/MachineLearning/comments/dvyi48/d_what_stupid_things_did_you_use_to_do/

Posted by u/etienne_ben 20 hours ago

56

[D] What stupid things did you use to do?

Discussion

probablyuntrue 26 points · 19 hours ago

data leakage, so much data leakage

Reply Give Award Share Report Save

coffeecoffeecoffeee 6 points · 11 hours ago

Yep. I once spent half a day debugging why a model had like 99.98% AUC. Turned out that I was inadvertently concatenating the class label to the block of text I was training on. For eight hours.

Reply Give Award Share Report Save

purplecramps 6 points · 19 hours ago

what's data leakage?

Reply Give Award Share Report Save

TopsyMitoTurvy 14 points · 19 hours ago

Most frequently it's a standarization before doing splitting

Reply Give Award Share Report Save

WERE_CAT 6 points · 15 hours ago

Care to elaborate on that ? Give some ressource that lists such errors ?

Reply Give Award Share Report Save

MrTwiggy 2 points · 2 hours ago

In the broadest sense, I would define data leakage as this: Some of your data points in your training set contain information that would not be available in the future when deploying the model into the real problem space.

https://www.reddit.com/r/MachineLearning/comments/dvyi48/d_what_stupid_things_did_you_use_to_do/

Posted by u/etienne_ben 20 hours ago

56

[D] What stupid things did you use to do?

Discussion

ThomasAger 1 point · 10 hours ago

Im a bit confused on what that means, to me standardization would be making sure all the data has a similar format to some degree. Why would that result in data leakage if done before splitting?

Reply Give Award Share Report Save

Xylon- 4 points · 9 hours ago

Standardization is scaling your data so that it has a mean (μ) of 0 and std (σ) of 1. You do this with `z = (x-μ)/σ`, where `x` is your data you want to standardize, and μ and σ are the mean and std of this data.

You generally want your test/validation set to be completely unseen, but if you're standardizing before splitting, you're including these data points into the calculation of μ and σ.

What you should do is calculate the μ and σ purely based on the train data, and then use these to scale the test and validation data.

https://www.reddit.com/r/MachineLearning/comments/dvyi48/d_what_stupid_things_did_you_use_to_do/

r/MachineLearning

Posts

56

Posted by u/etienne_ben 20 hours ago

[D] What stupid things did you use to do?

Discussion

ckatem 17 points · 18 hours ago

Im extremely embarrassed to admit this. I used to exclusively use SAS.

Reply Give Award Share Report Save

WERE_CAT 6 points · 15 hours ago

ew

Reply Give Award Share Report Save

https://www.reddit.com/r/MachineLearning/comments/dvyi48/d_what_stupid_things_did_you_use_to_do/

Part I: Introduction

- Lecture 1: What is Machine Learning? An Overview.
- Lecture 2: Intro to Supervised Learning: KNN

Part II: Computational Foundations

- Lecture 3: Using Python, Anaconda, IPython, Jupyter Notebooks
- Lecture 4: Scientific Computing with NumPy, SciPy, and Matplotlib
- Lecture 5: Data Preprocessing and Machine Learning with Scikit-Learn

Part III: Tree-Based Methods

- Lecture 6: Decision Trees
- Lecture 7: Ensemble Methods

Part IV: Evaluation

- Lecture 8: Model Evaluation 1: Introduction to Overfitting and Underfitting
- Lecture 9: Model Evaluation 2: Uncertainty Estimates and Resampling
- Lecture 10: Model Evaluation 3: Model Selection and Cross-Validation
- Lecture 11: Model Evaluation 4: Algorithm Selection and Statistical Tests
- Lecture 12: Model Evaluation 5: Performance Metrics

Part V: Dimensionality Reduction

- Lecture 13: Feature Selection
- Lecture 14: Feature Extraction

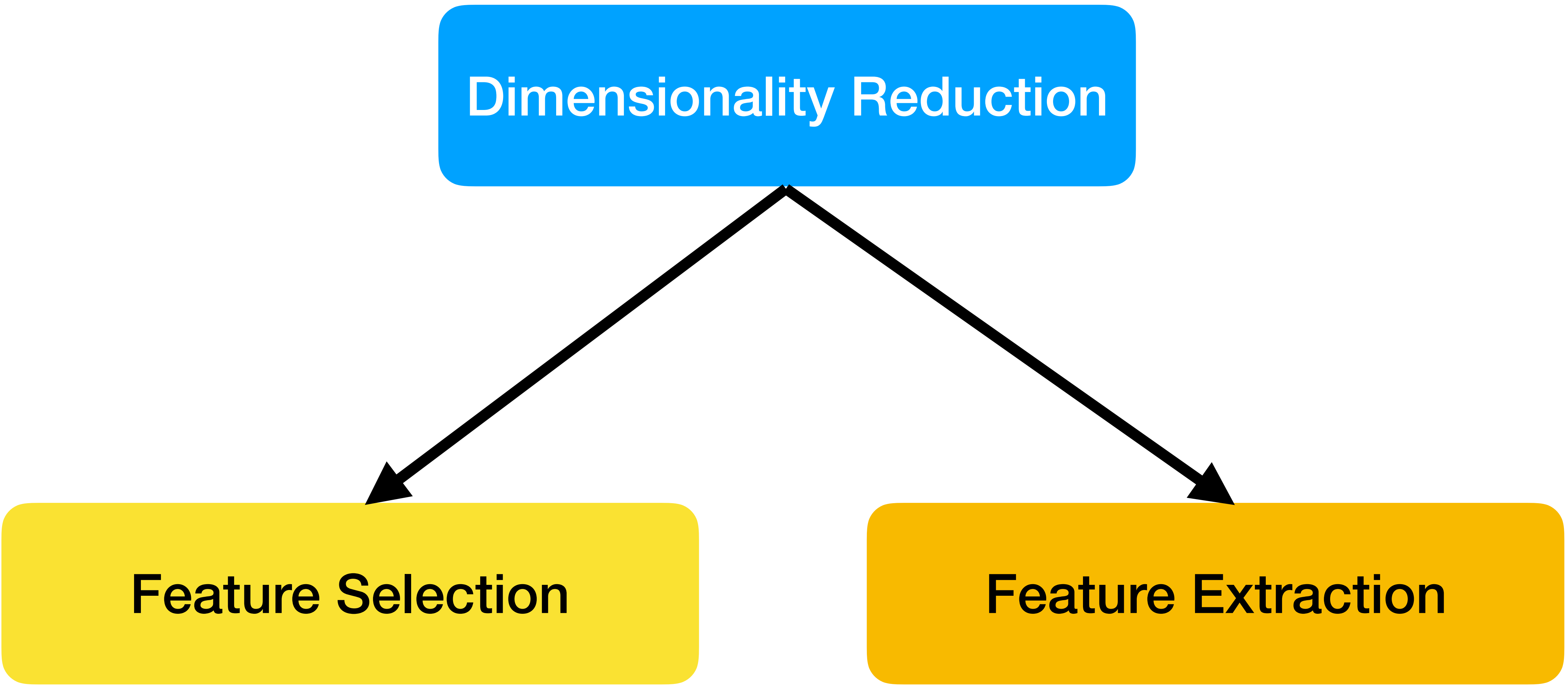
Lecture 13

Dimensionality Reduction I: Feature Selection

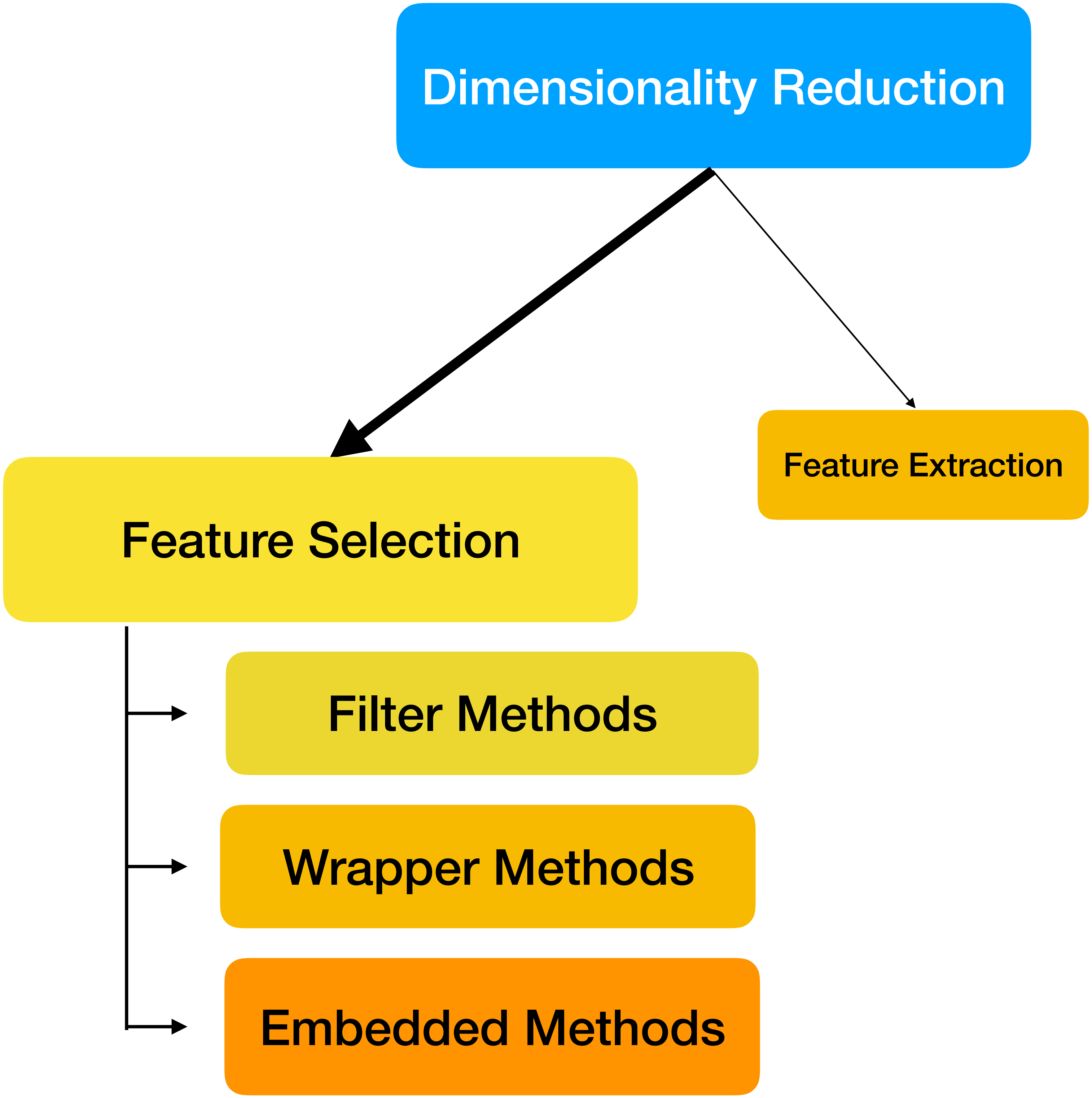
STAT 479: Machine Learning, Fall 2019

Sebastian Raschka

http://stat.wisc.edu/~sraschka/teaching/stat479-fs2019/


Dimensionality Reduction
Feature Selection
Feature Extraction

Today


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

Dimensionality Reduction

Feature Extraction

Feature Selection

Filter Methods

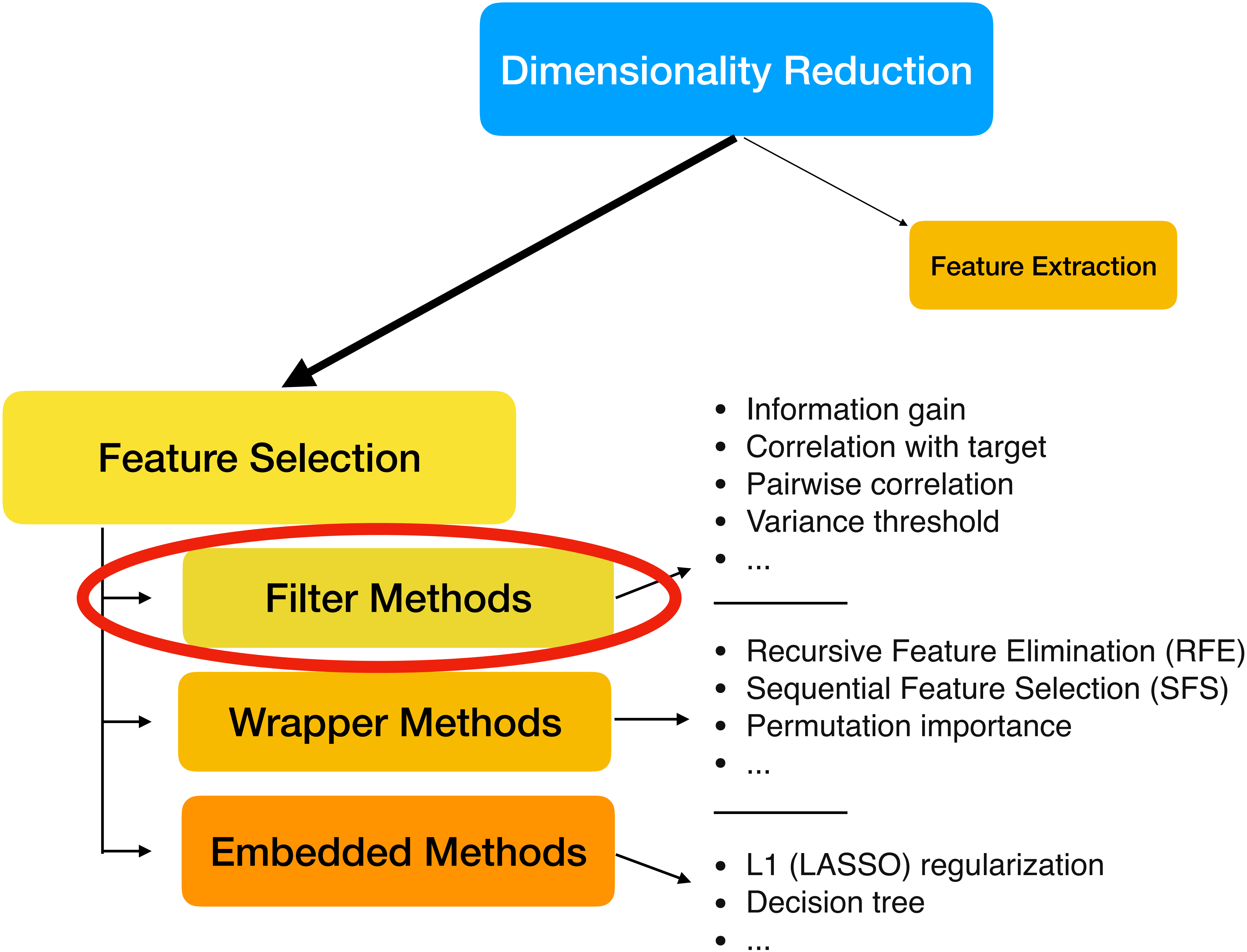
- Information gain
- Correlation with target
- Pairwise correlation
- Variance threshold
- ...

Wrapper Methods

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

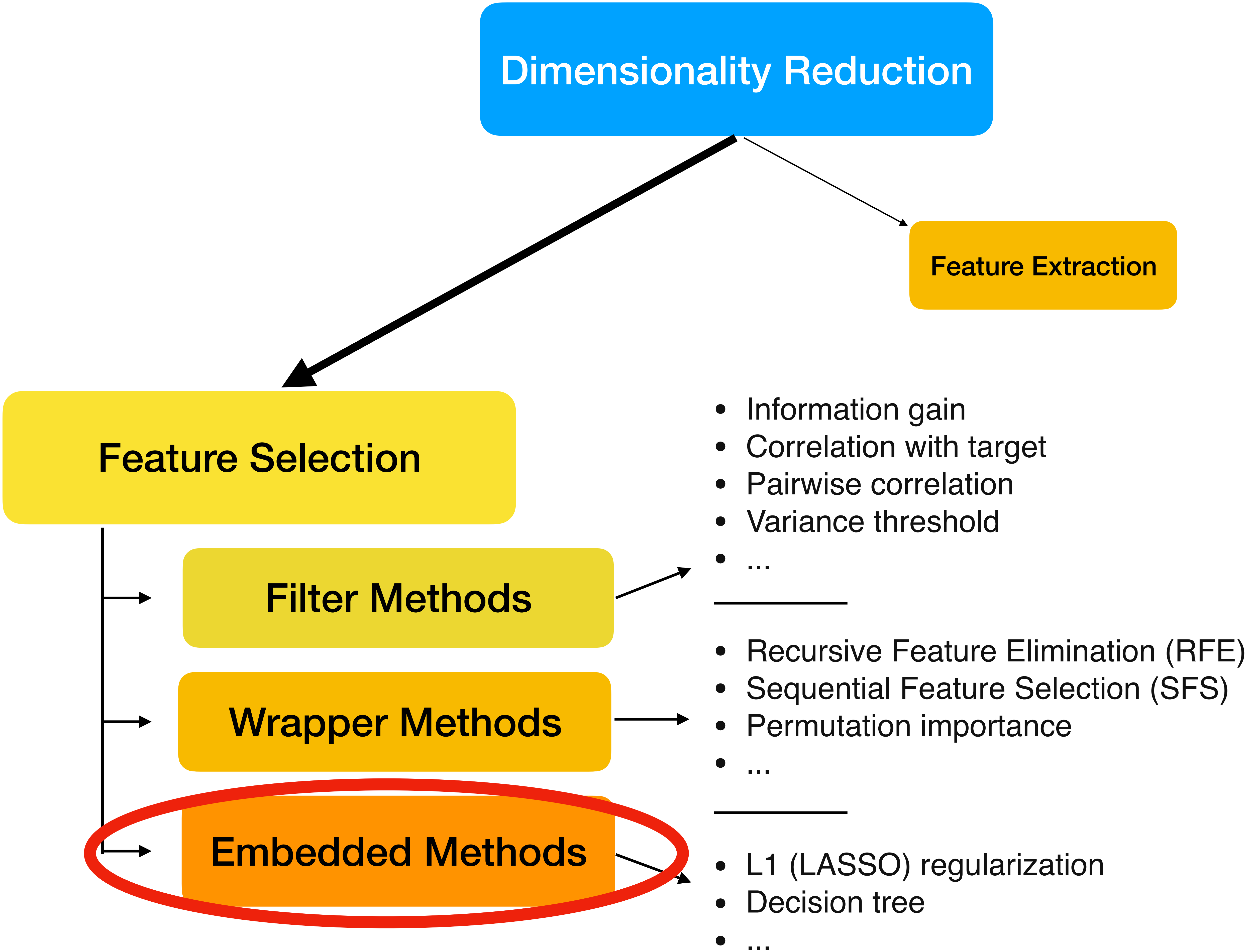
Embedded Methods

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

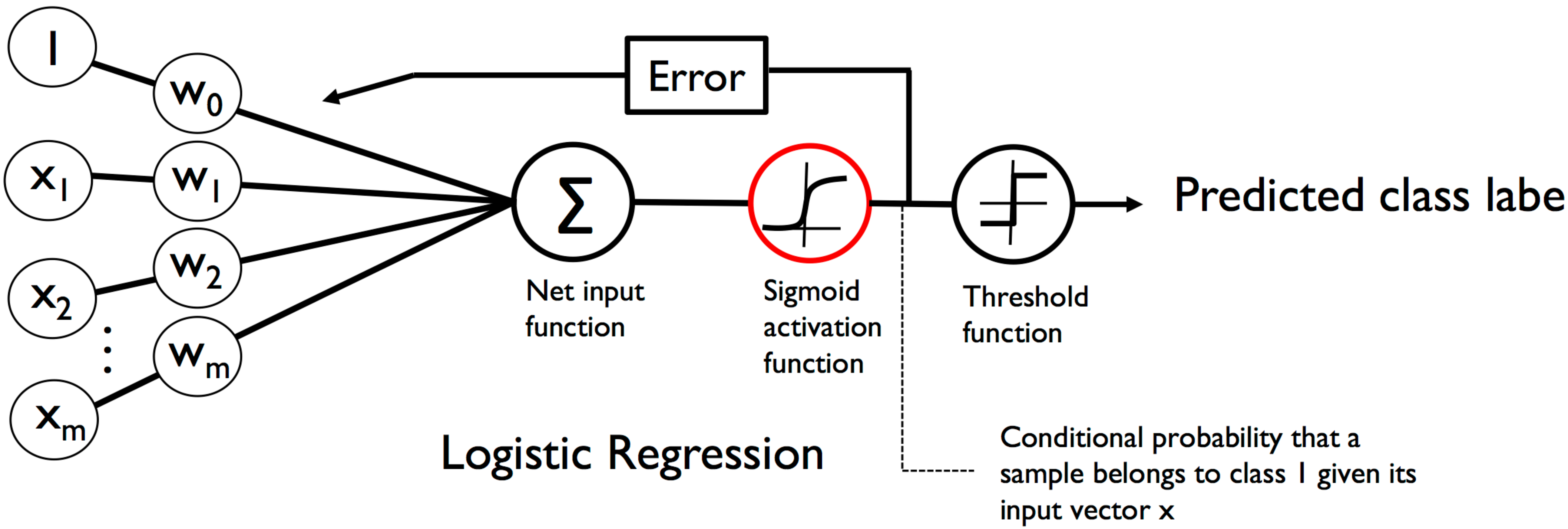

Dimensionality Reduction
Feature Extraction
Feature Selection
Filter Methods
Wrapper Methods
Embedded Methods
• Information gain
• Correlation with target
• Pairwise correlation
• Variance threshold
• ...
• Recursive Feature Elimination (RFE)
• Sequential Feature Selection (SFS)
• Permutation importance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


Dimensionality Reduction
Feature Extraction
Feature Selection
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
• ...

Logistic Regression



Source: Raschka & Mirjalili. *Python Machine Learning 2nd ed., Ch 3*

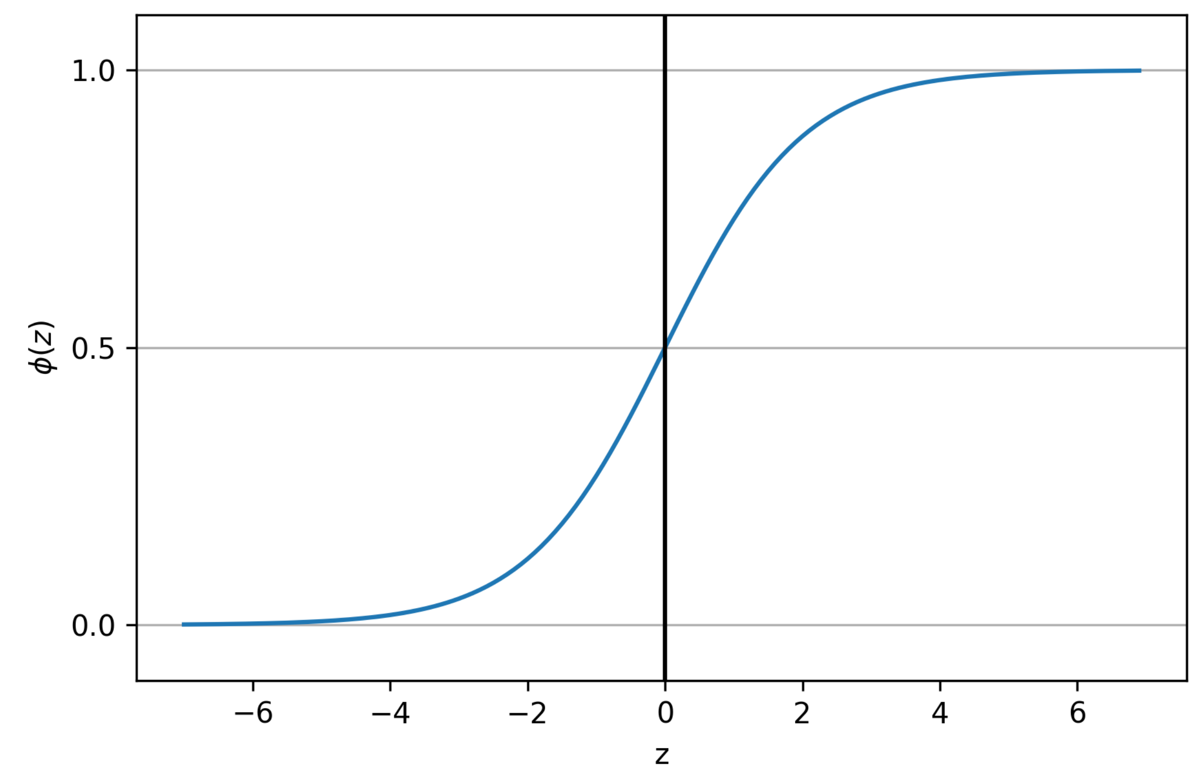
Logistic Regression

1) Weighted inputs ("net inputs", "logits")

$$z := \text{logit}(p(y = 1 \mid x)) = w_0x_0 + w_1x_1 + \cdots + w_mx_m = \sum_{i=0}^{m} w_ix_i = \boldsymbol{w}^T\boldsymbol{x}$$

2) Nonlinear function (logistic sigmoid)

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



3) Threshold for predicting class label

$$\hat{y} = \begin{cases} 1 & \text{if } \phi(z) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Logistic Regression

4) Loss function to minimize during training (negative log-likelihood)

$$J(\mathbf{w}) = \sum_{i=1}^{n} \left[-y^{(i)} \log\left(\phi\left(z^{(i)}\right)\right) - \left(1 - y^{(i)}\right) \log\left(1 - \phi\left(z^{(i)}\right)\right) \right]$$

"True" class label

$$\boxed{\phi(z) = \frac{1}{1 + e^{-z}}}$$

predicted probability p(y=1|x)

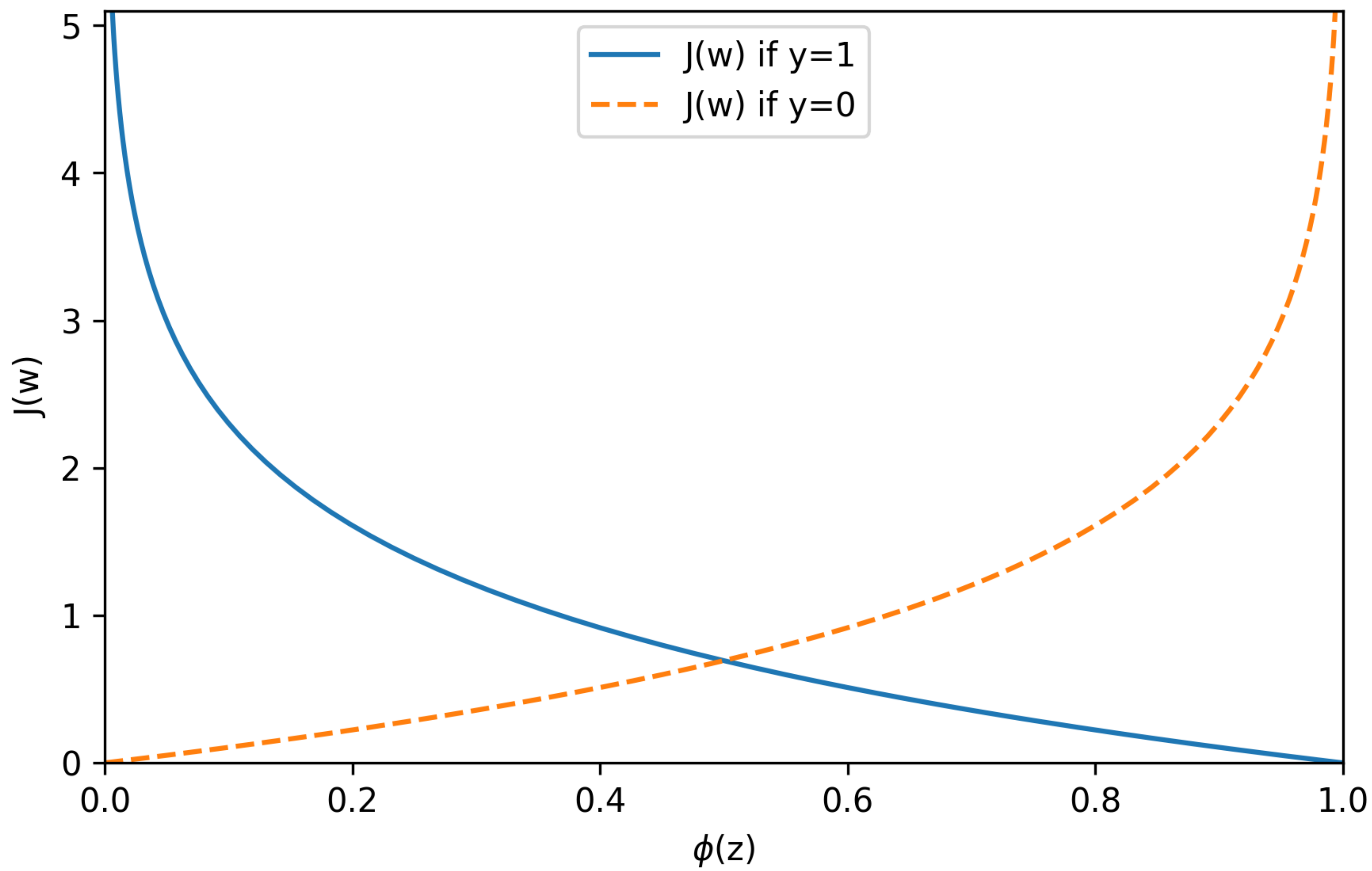
Equivalent to

$$J(\phi(z), y; w) = \begin{cases} -\log(\phi(z)) & \text{if } y = 1 \\ -\log(1 - \phi(z)) & \text{if } y = 0 \end{cases}$$

(More details in Stat 453: Deep Learning)

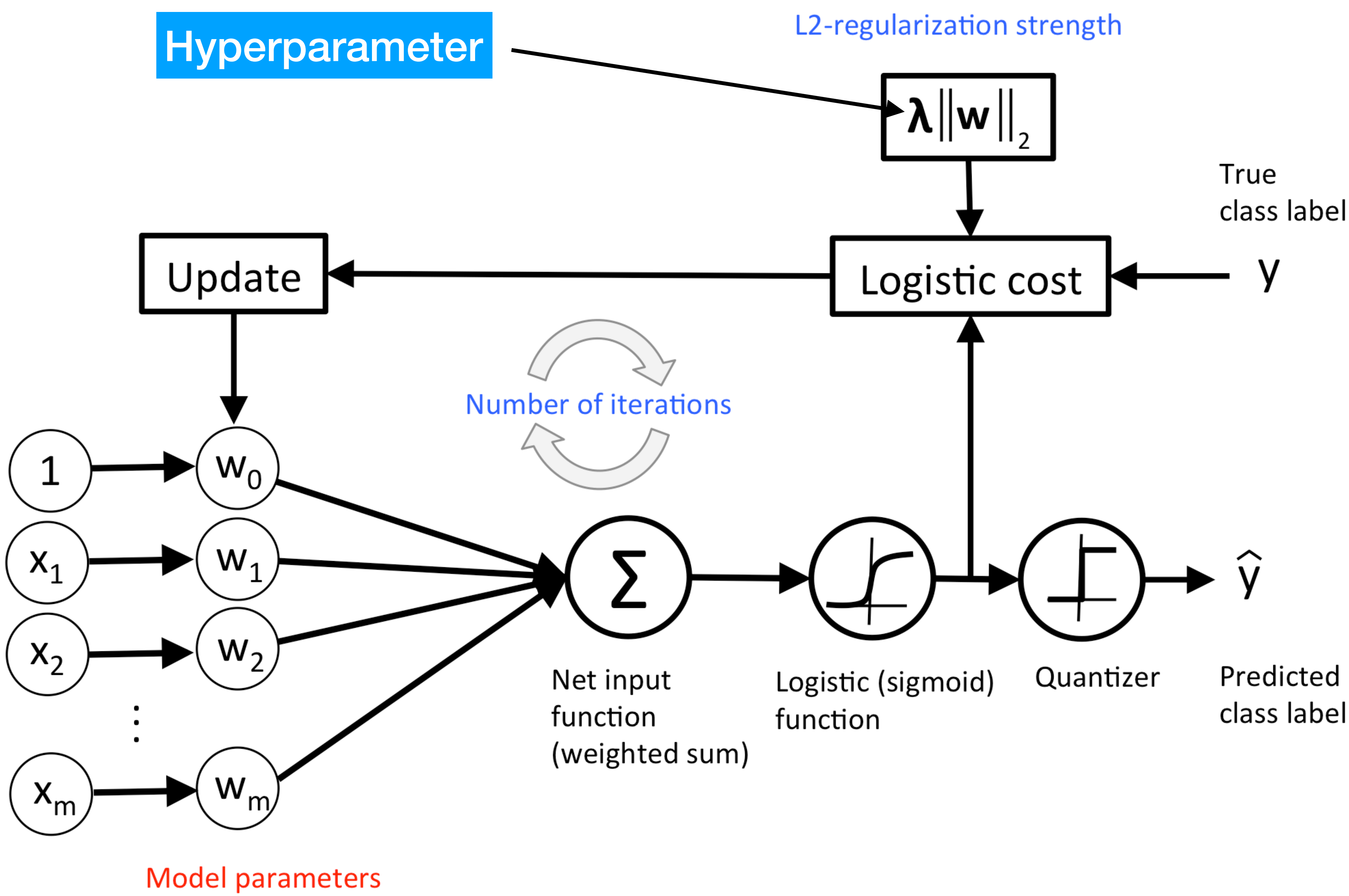
Logistic Regression

4) Loss function to minimize during training (negative log-likelihood)



$$J(\phi(z), y; w) = \begin{cases} -\log(\phi(z)) & \text{if } y = 1 \\ -\log(1 - \phi(z)) & \text{if } y = 0 \end{cases}$$

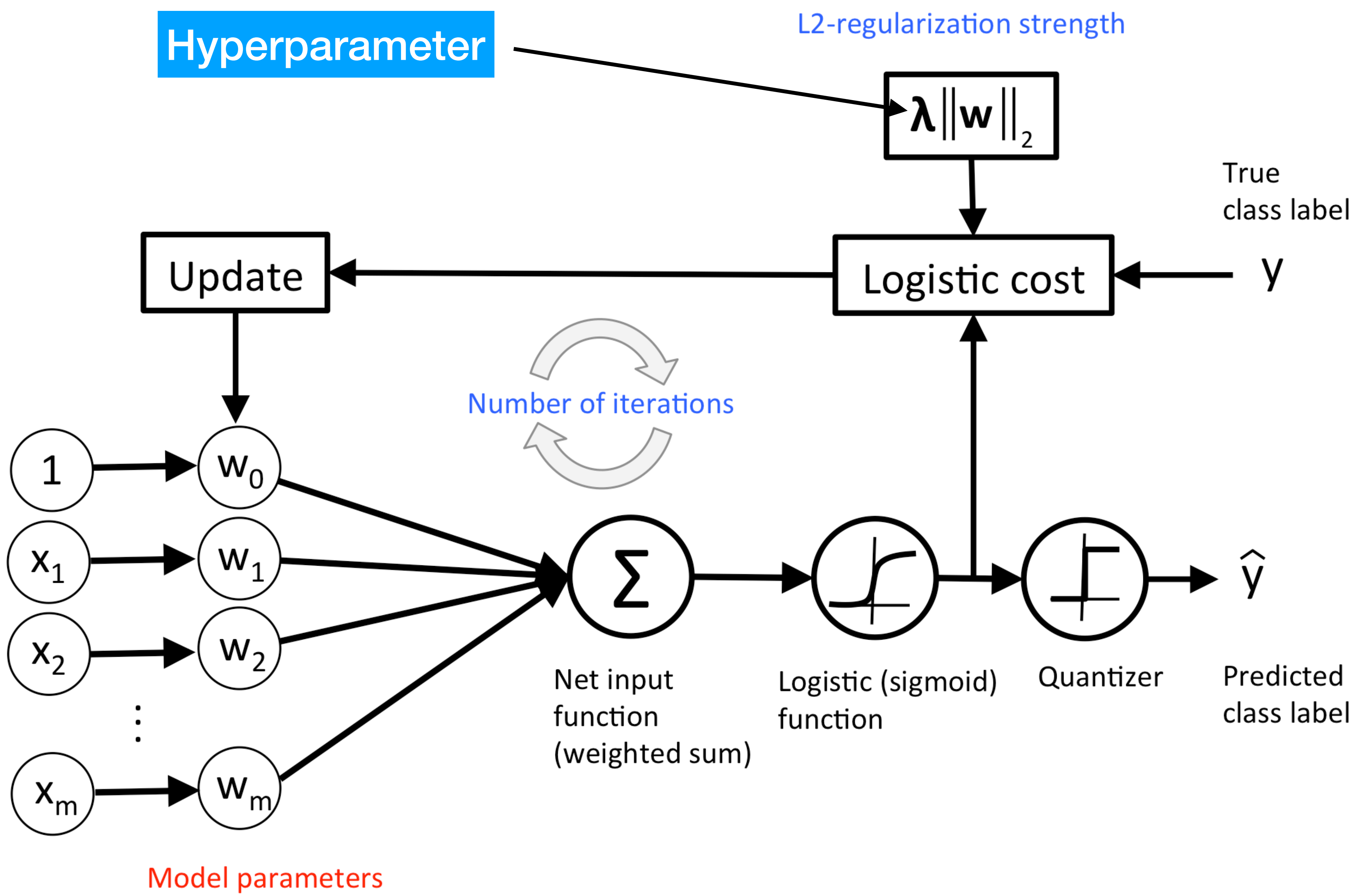
Logistic Regression Hyperparameters



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

L1 penalty against complexity

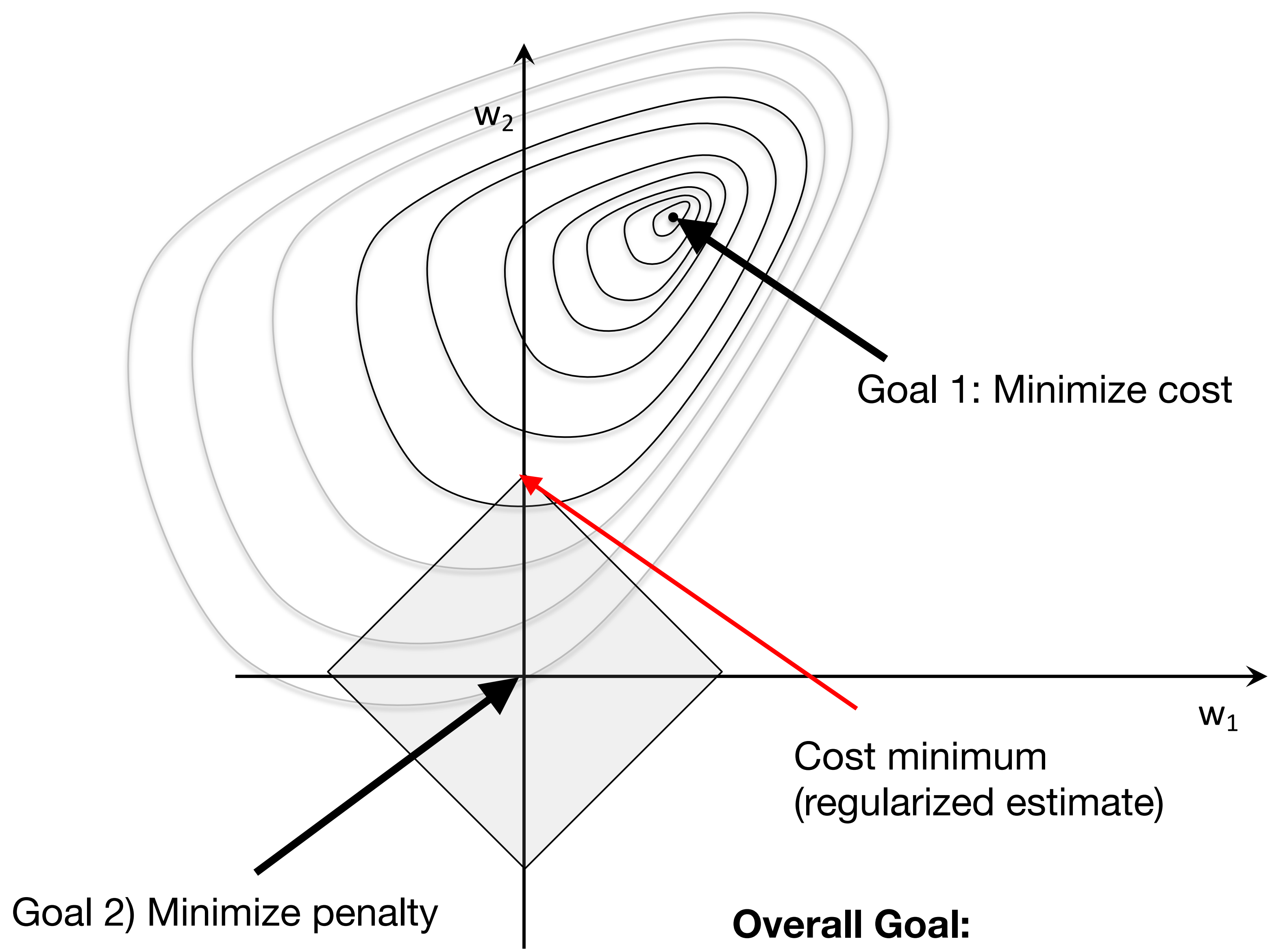
$$L1 : \|\boldsymbol{w}\|_1 = \sum_{j=1}^{m} \left| w_j \right|$$

L1-penalized loss

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

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

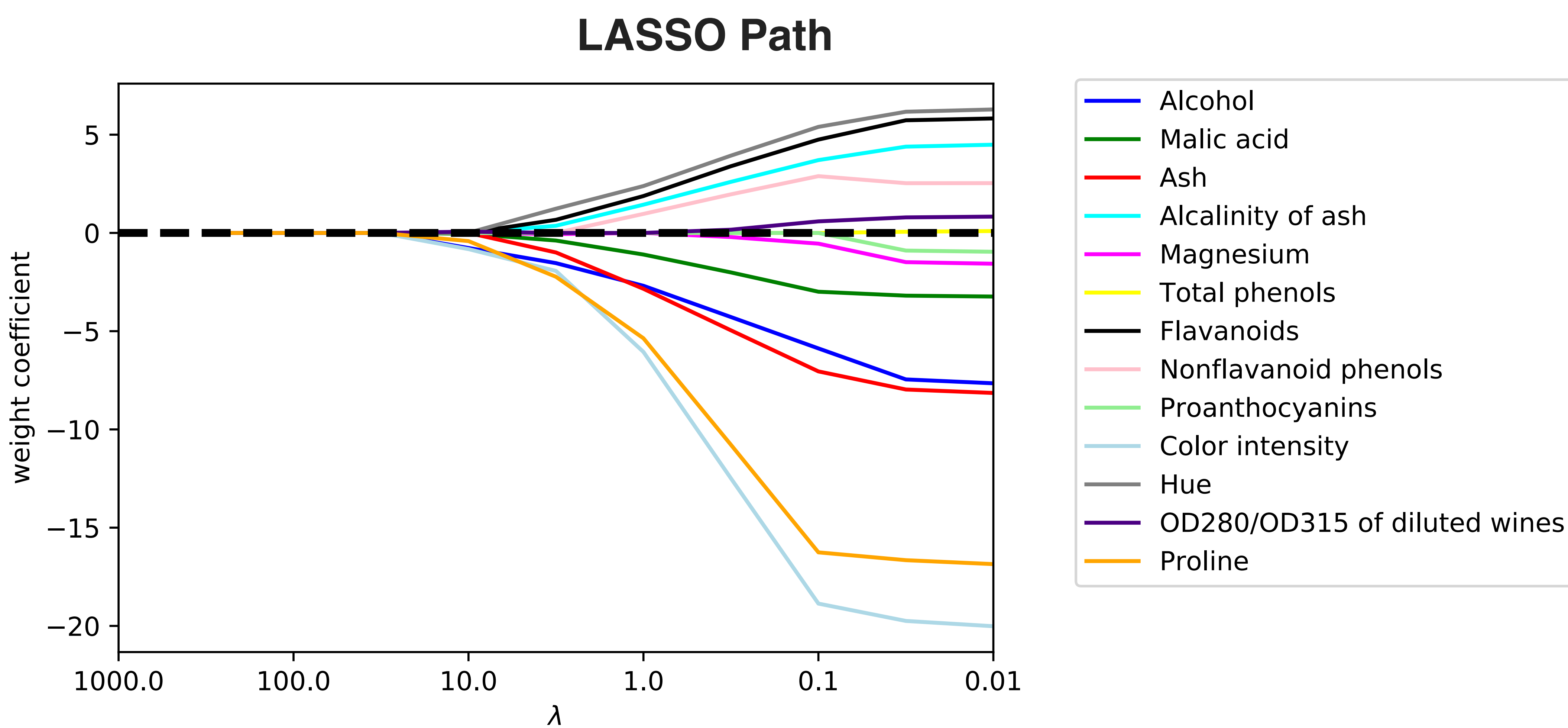
'Class label'	'Alcohol'	'Malic acid'	'Ash'	Alcalinity of ash'	'Magnesium'	'Total phenols'	'Flavanoids'	'Nonflavanoid phenols'	'Proanthocyanins'	'Color intensity'	'Hue'	OD280/OD315 of diluted wines'	'Proline'
1	14.23	1.71	2.43	15.6	127	2.8	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	13.2	1.78	2.14	11.2	100	2.65	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)

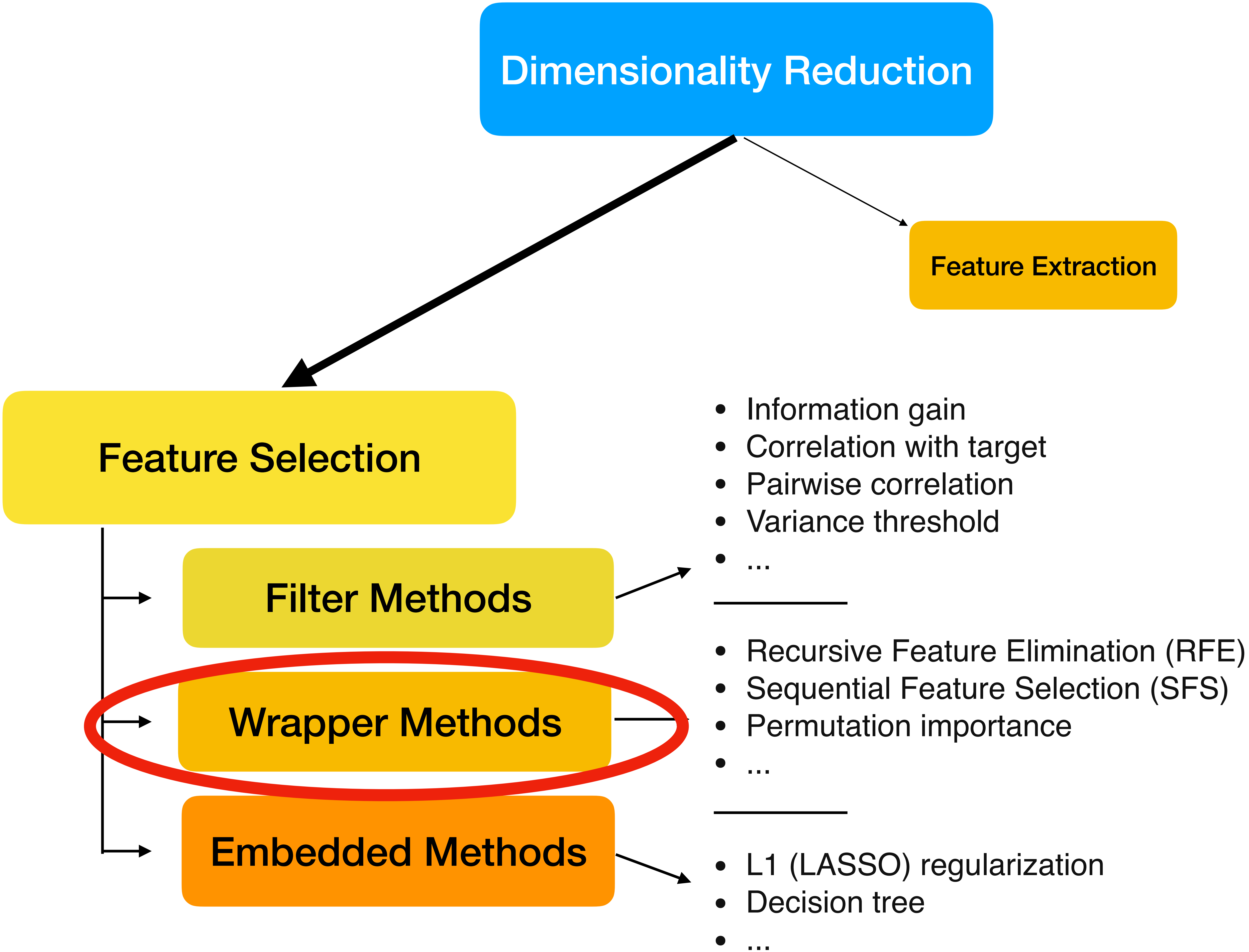
Least Absolute Shrinkage and Selection Operator

Wine Dataset

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



(don't forget to normalize/standardize features)


Dimensionality Reduction
Feature Extraction
Feature Selection
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
• ...

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

Random Forest Feature Importance

"Method A" (this is used in scikit-learn)

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

- (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

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.

mlxtend Home User Guide API Installation About Search GitHub

Permutation Test

An implementation of a permutation test for hypothesis testing -- testing the null hypothesis that two different groups come from the same distribution.

```
from mlxtend.evaluate import permutation_test
```

http://rasbt.github.io/mlxtend/user_guide/evaluate/permutation_test/

Example 1 -- Two-sided permutation test

Perform a two-sided permutation test to test the null hypothesis that two groups, "treatment" and "control" come from the same distribution. We specify alpha=0.01 as our significance level.

```
treatment = [ 28.44,  29.32,  31.22,  29.58,  30.34,  28.76,  29.21,  30.4 ,
              31.12,  31.78,  27.58,  31.57,  30.73,  30.43,  30.31,  30.32,
              29.18,  29.52,  29.22,  30.56]
control = [ 33.51,  30.63,  32.38,  32.52,  29.41,  30.93,  49.78,  28.96,
            35.77,  31.42,  30.76,  30.6 ,  23.64,  30.54,  47.78,  31.98,
            34.52,  32.42,  31.32,  40.72]
```

Since evaluating all possible permutations may take a while, we will use the approximation method (see the introduction for details):

```
from mlxtend.evaluate import permutation_test

p_value = permutation_test(treatment, control,
                           method='approximate',
                           num_rounds=10000,
                           seed=0)
print(p_value)
```

```
0.0066
```

Since p-value < alpha, we can reject the null hypothesis that the two samples come from the same distribution.

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 mlxtend Home User Guide API Installation About Search GitHub

Feature Importance Permutation
Overview
Example 1 -- Feature Importance for Classifiers
Example 1 -- Feature Importance for Regressors
API

Feature Importance Permutation

A function to estimate the feature importance of classifiers and regressors based on *permutation importance*.

```
from mlxtend.evaluate import feature_importance_permutation
```

http://rasbt.github.io/mlxtend/user_guide/evaluate/feature_importance_permutation/

Feature Importance Through Permutation (Wrapper)

intuitive & model-agnostic

Column-Drop variant:

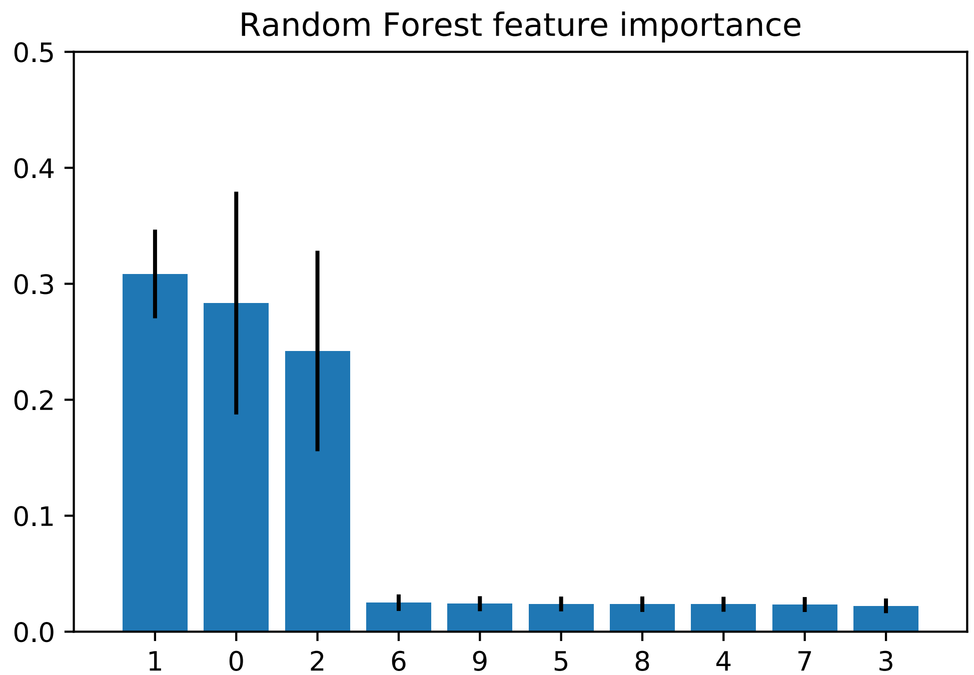
For each feature column *i:*

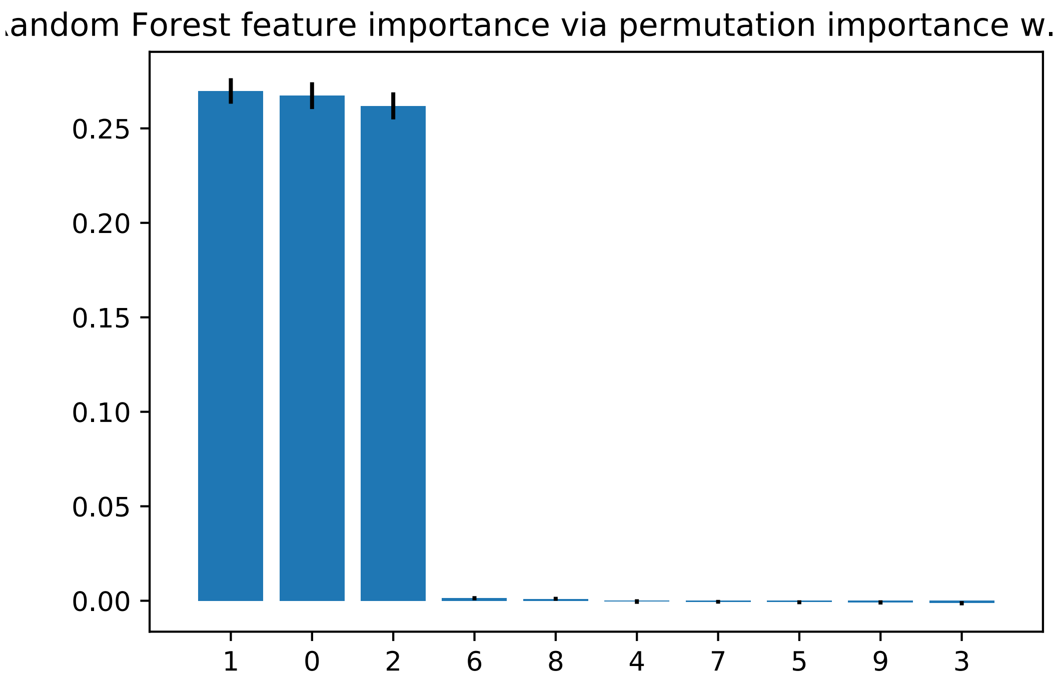
1. temporarily remove column
2. fit model to reduced dataset
3. compute validation set performance and compare to before

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"

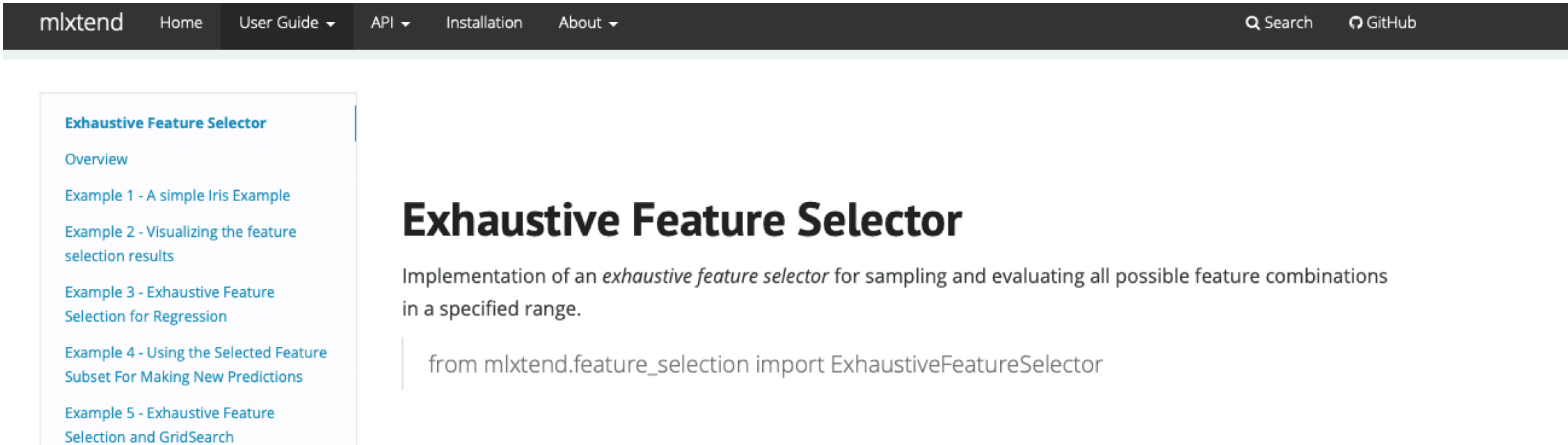
Out-of-bag accuracy:

- 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 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:

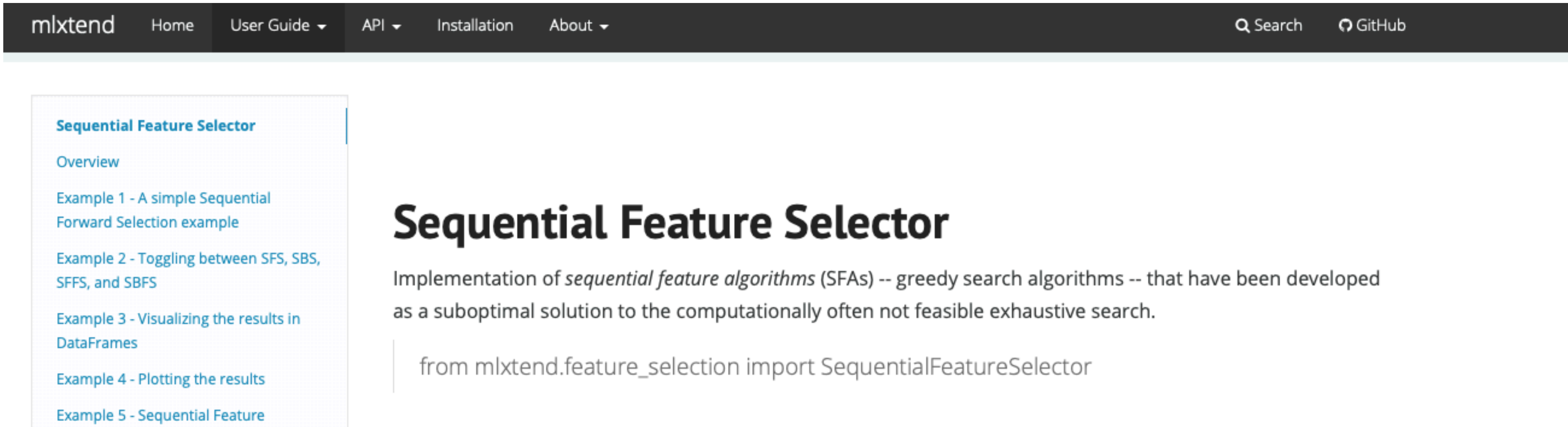
- Count votes for correct class
- Given feature *i,* permute this feature in OOB examples of a tree
- Compute the number of correct votes after permutation from the number of votes before permutation for given tree
- Repeat for all trees in the random forest and average the importance
- Repeat for other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

Exhaustive Feature Selection (Wrapper)

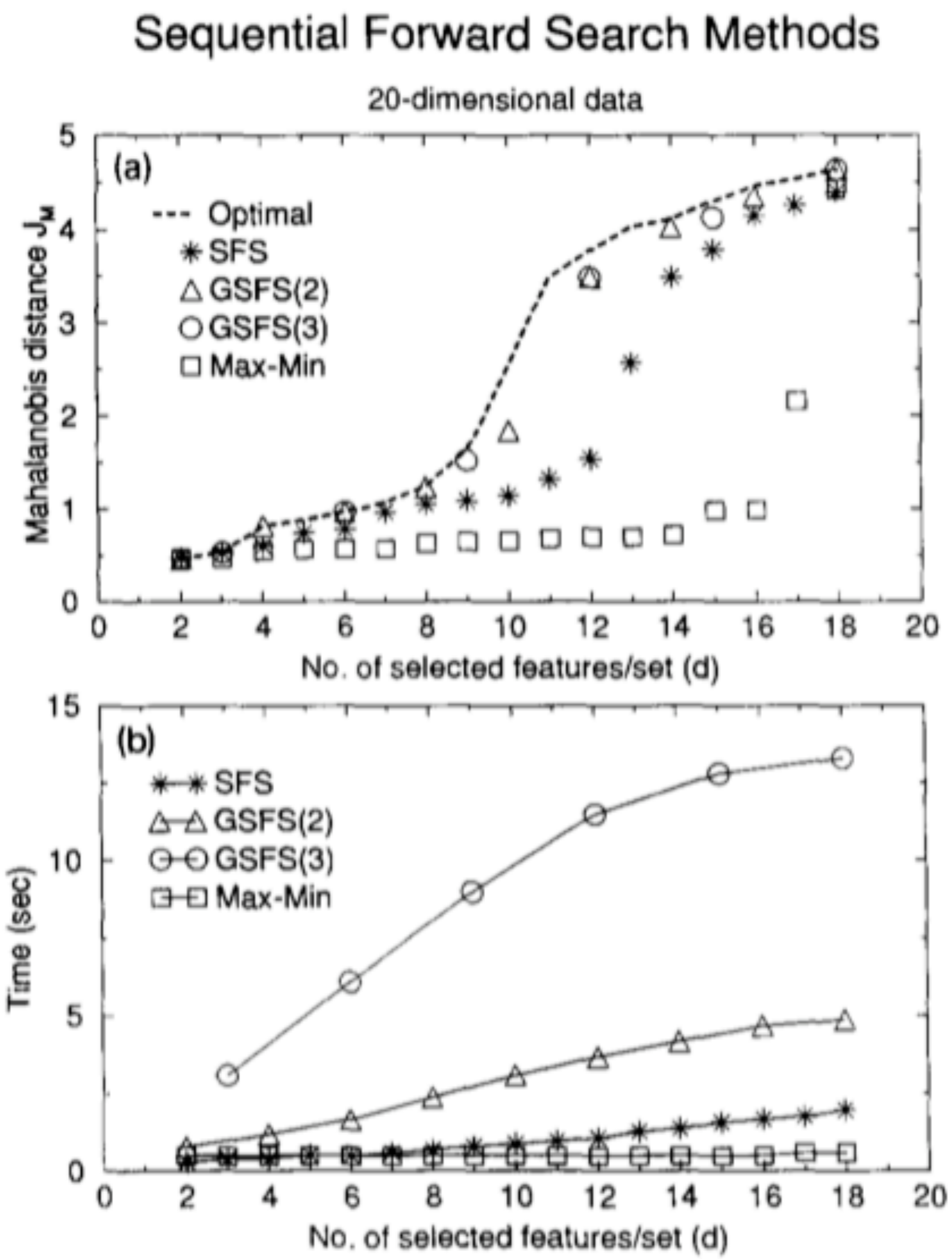


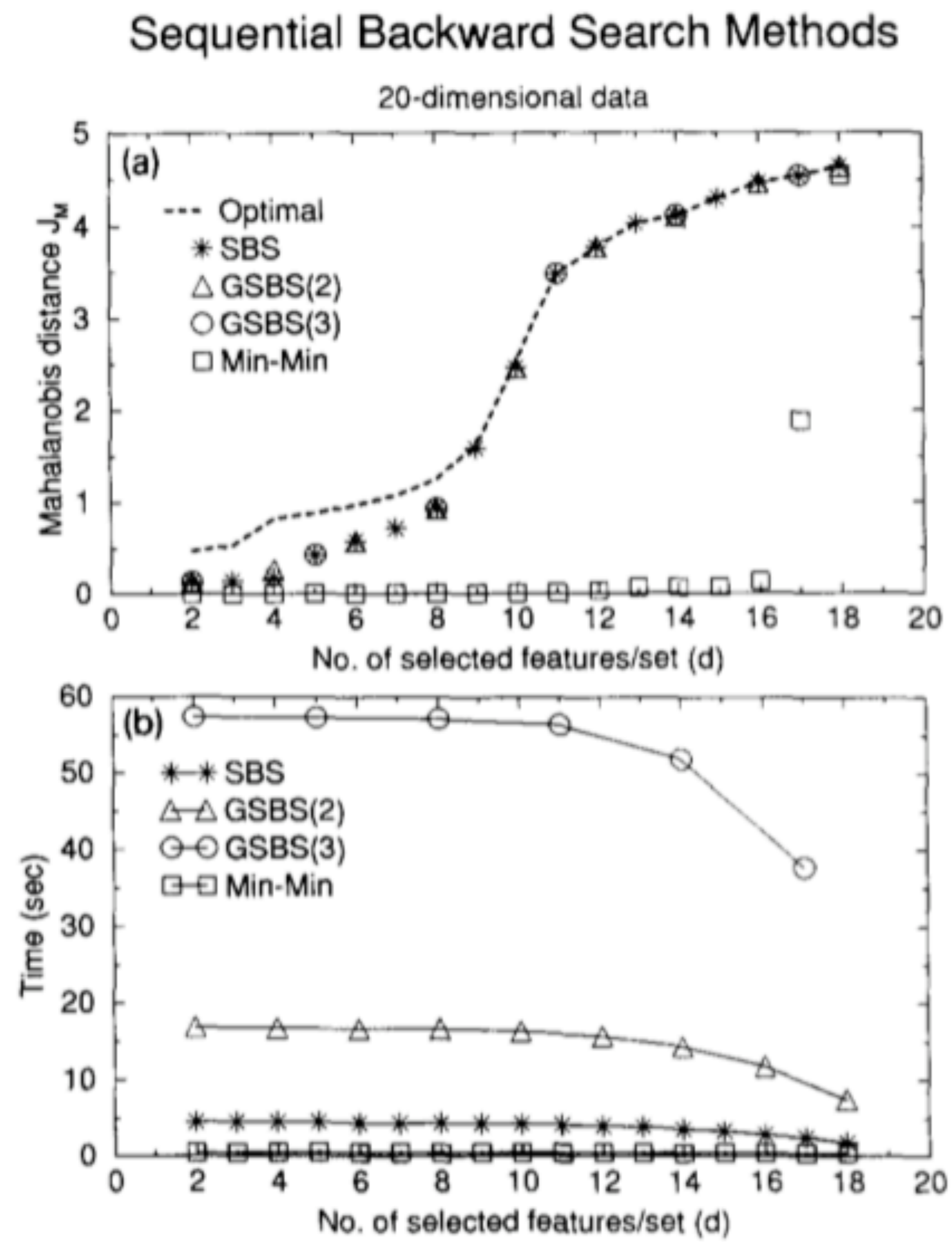
http://rasbt.github.io/mlxtend/user_guide/feature_selection/ExhaustiveFeatureSelector/

Sequential Forward/Backward Selection (Wrapper)



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





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).

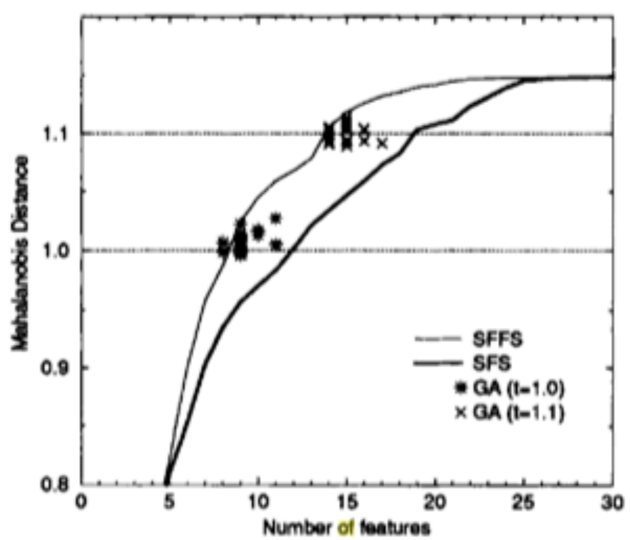


Figure 5. Results of Feature Selection obtained by SFS and SFFS methods for the $D = 30$ experiment. Crosses and asterisks show the results corresponding to different runs of the GA with two different values of the threshold parameter.

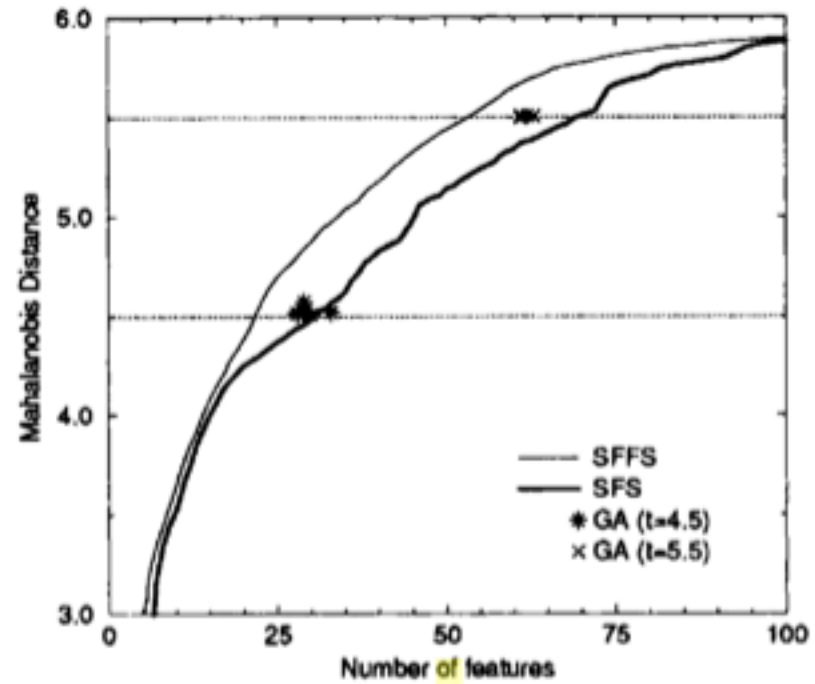


Figure 7. Results of Feature Selection obtained by SFS and SFFS methods for the $D = 120$ experiment. Crosses and asterisks show the results corresponding to different runs of the GA with two different values of the threshold parameter.

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.

Code Examples

https://github.com/rasbt/stat479-machine-learning-fs19/tree/master/13_feat-sele/code