COSC 3337 : Data Science I

N. Rizk

College of Natural and Applied Sciences

Department of Computer Science

University of Houston

Dimensionality reduction in Supervised learning: feature selection

Model selection:

Regularization (shrinkage)

↔

Subset selection (reduce the variance)

Why feature selection is important?

- May Improve performance of classification algorithm
- Classification algorithm may not scale up to the size of the full feature set either in sample or time
- Allows us to better understand the domain
- Cheaper to collect a reduced set of predictors
- Safer to collect a reduced set of predictors

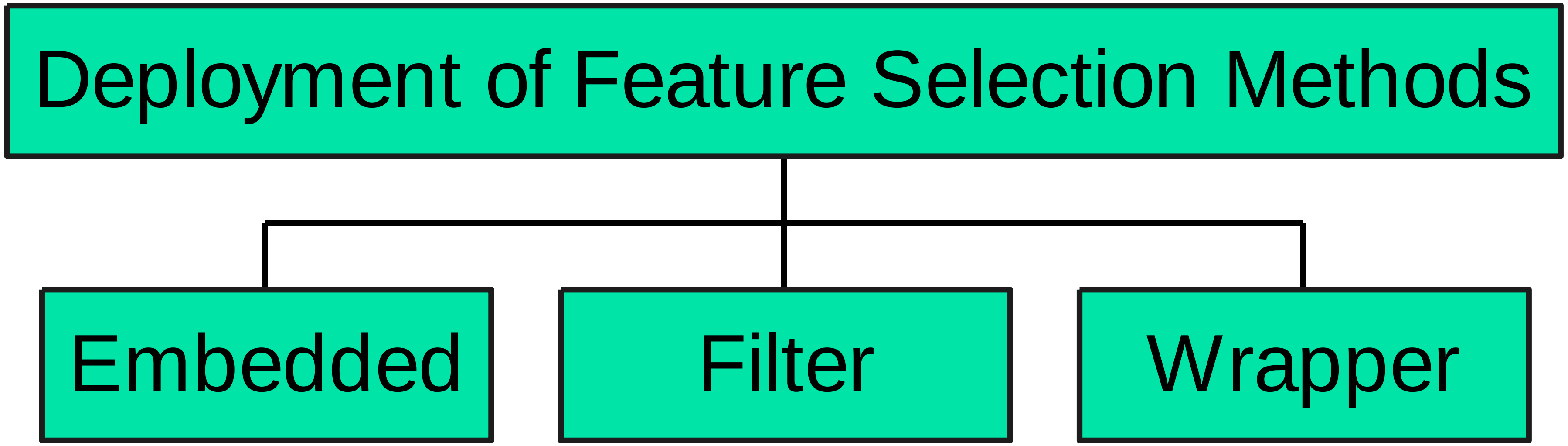
Feature Selection

- Definition
  - A process that chooses an optimal subset of features according to a objective function
- Objectives
  - To reduce dimensionality and remove noise
  - To improve mining performance
    - Speed of learning
    - Predictive accuracy
    - Simplicity and comprehensibility of mined results

Deployment of Feature Selection Methods

- Based on their relation to the induction algorithm feature selection methods can be grouped as:
  - Embedded: They are a part of induction algorithms
  - Filter: They are separate processes from the induction algorithms
  - Wrapper: They are also separate processes from induction algorithm but they use induction algorithm as a subroutine

Deployment of Feature Selection Methods

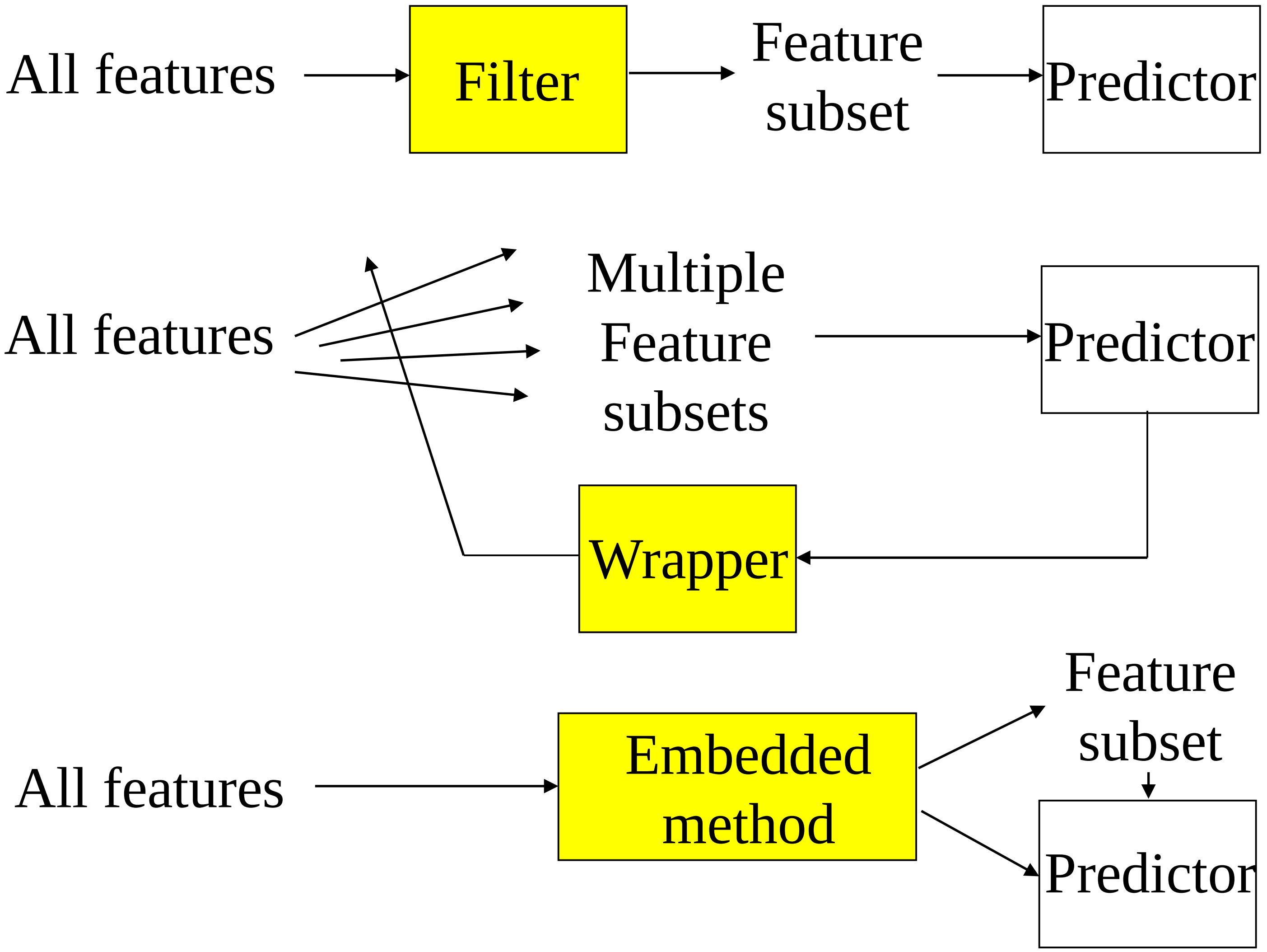


Motivation

The objective of feature selection is three-fold:

- Improving the prediction performance of the predictors
- Providing a faster and more cost-effective predictors
- Providing a better understanding of the underlying process that generated the data

Filters,Wrappers, and Embedded methods



Feature Subset Selection

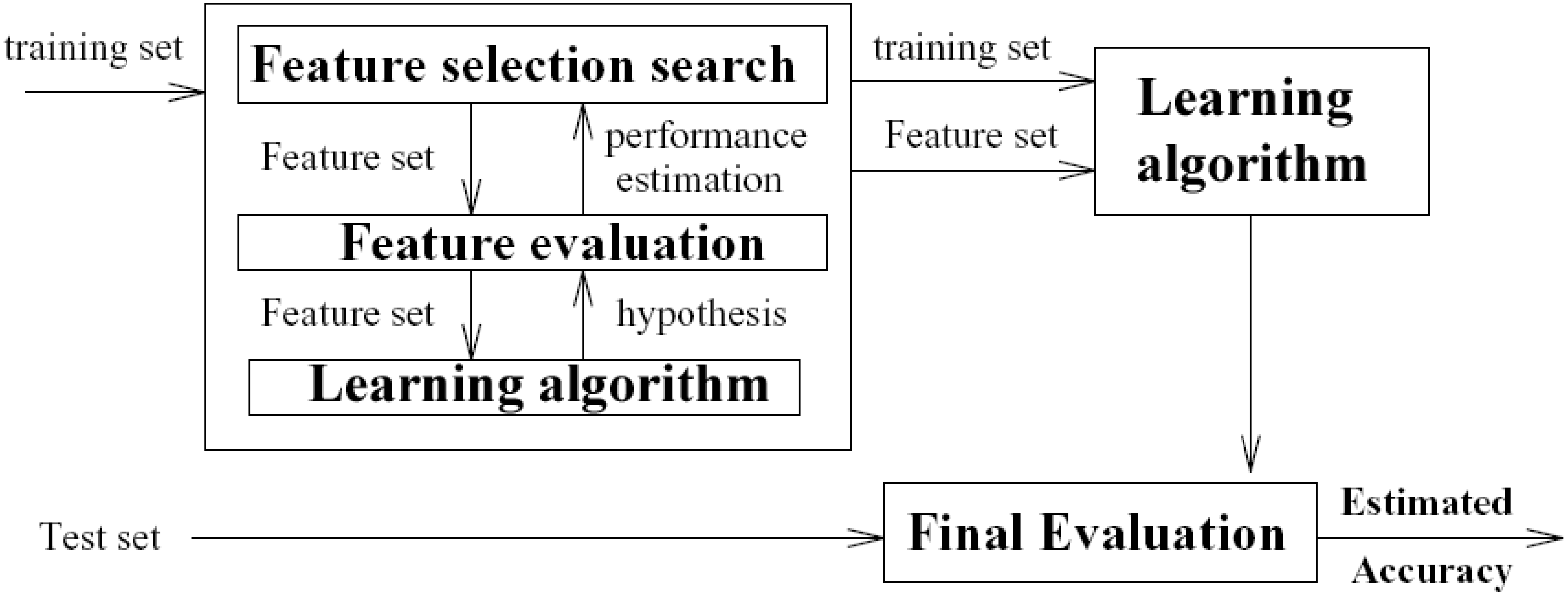
Filter Methods

- Select subsets of variables as a pre-processing step,
  independently of the used classifier!!

- Note that Variable Ranking-FS is a filter method
- Feature selection is independent of the prediction model (Information Gain, Minimum Redundancy Maximum Relevance, etc.)

Feature Subset Selection

Wrapper Methods



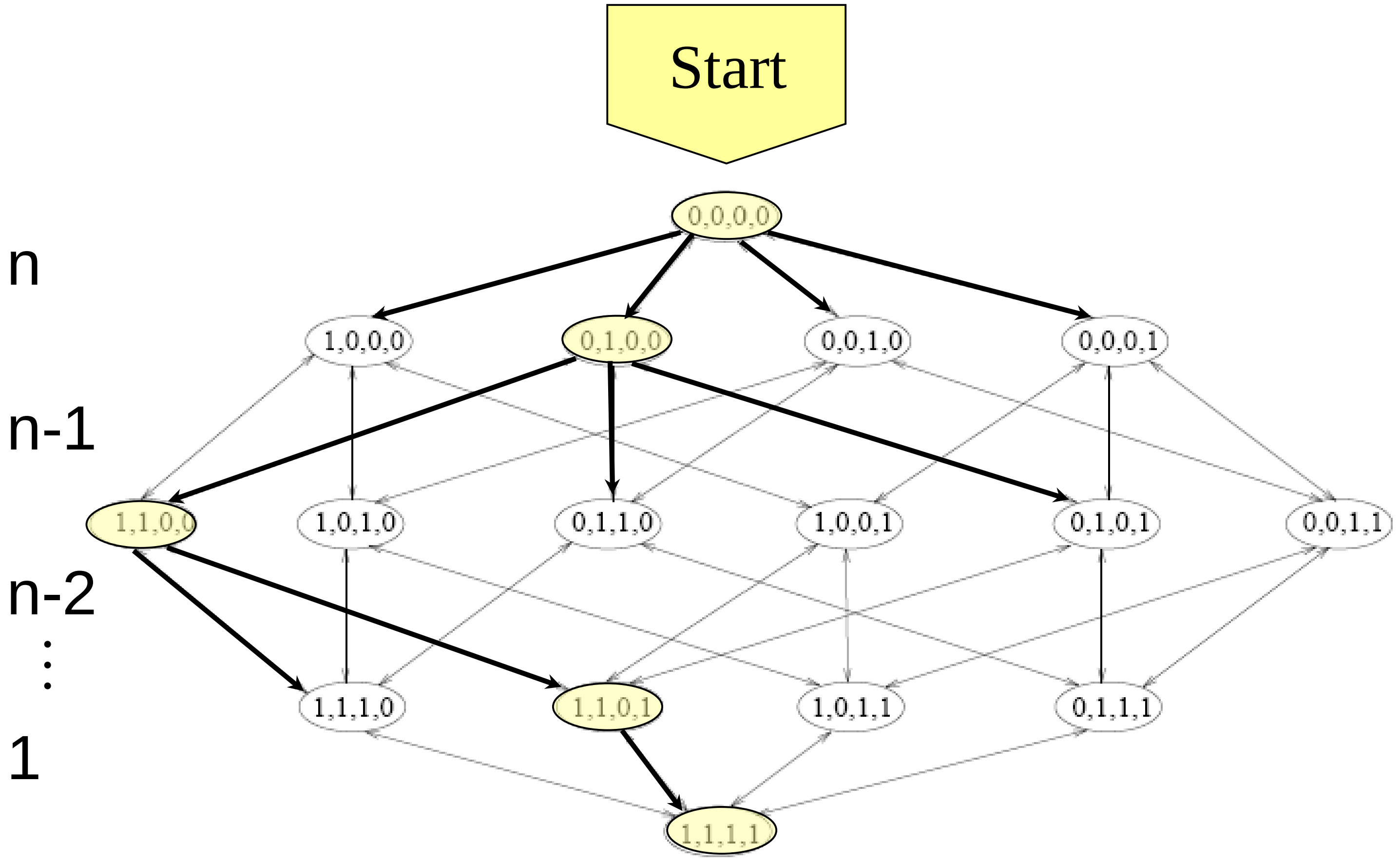
Feature selection is integrated in the prediction model, typically very slow, not able to incorporate domain knowledge (LASSO, SVM-RFE…)

Filters vs Wrappers: Filters

In the filter approach we do not rely on running a particular classifier and searching in the space of feature subsets; instead we select features on the basis of statistical properties. A classic example is univariate associations:

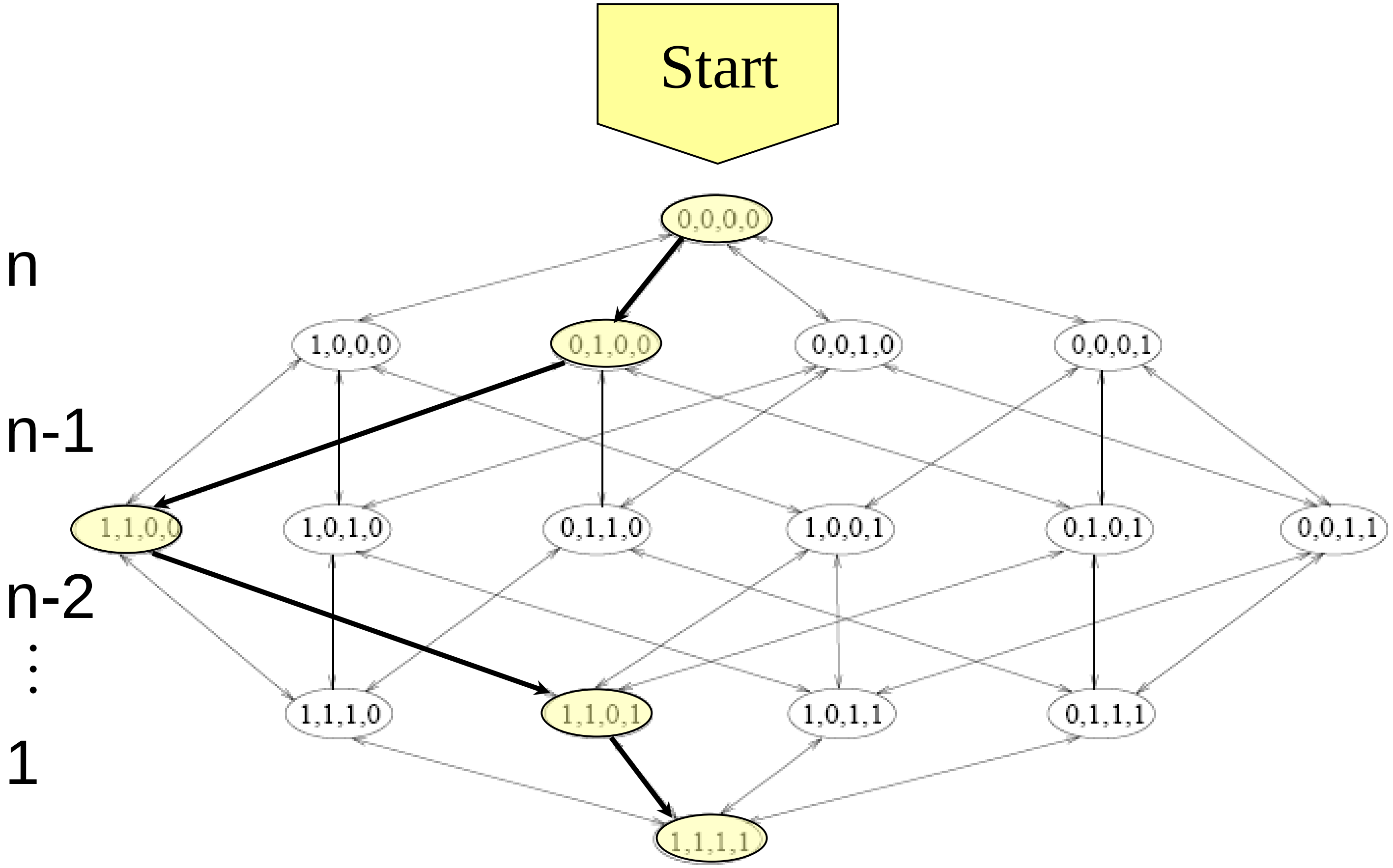
FEATURE	ASSOCIATION WITH TARGET	
{A}	91%	Threshold gives suboptimal solution
{B}	90%	Threshold gives optimal solution
{C}	89%	Threshold gives suboptimal solution

Forward Selection (wrapper)



Also referred to as SFS: Sequential Forward Selection

Forward Selection (embedded)



Guided search: we do not consider alternative paths.

Forward Selection with GS

Stoppiglia, 2002. Gram-Schmidt orthogonalization.

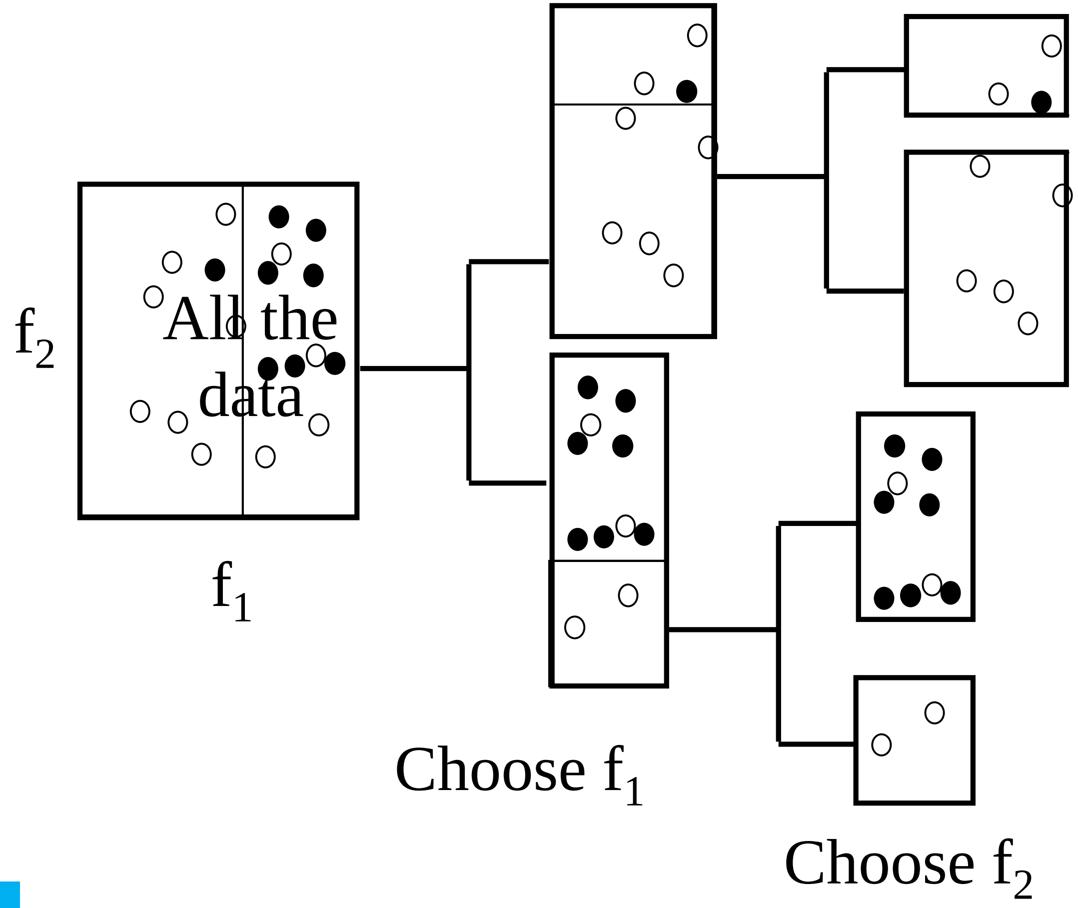
- Select a first feature $X_{v(1)}$ with maximum cosine with the target $\cos(\mathbf{x}_i, \mathbf{y}) = \mathbf{x}.\mathbf{y}/||\mathbf{x}||\ ||\mathbf{y}||$
- For each remaining feature X_i
  - Project X_i and the target Y on the null space of the features already selected
  - Compute the cosine of X_i with the target in the projection
- Select the feature $X_{v(k)}$ with maximum cosine with the target in the projection.

Embedded method for the linear least square predictor

Forward Selection w. Trees

- Tree classifiers,

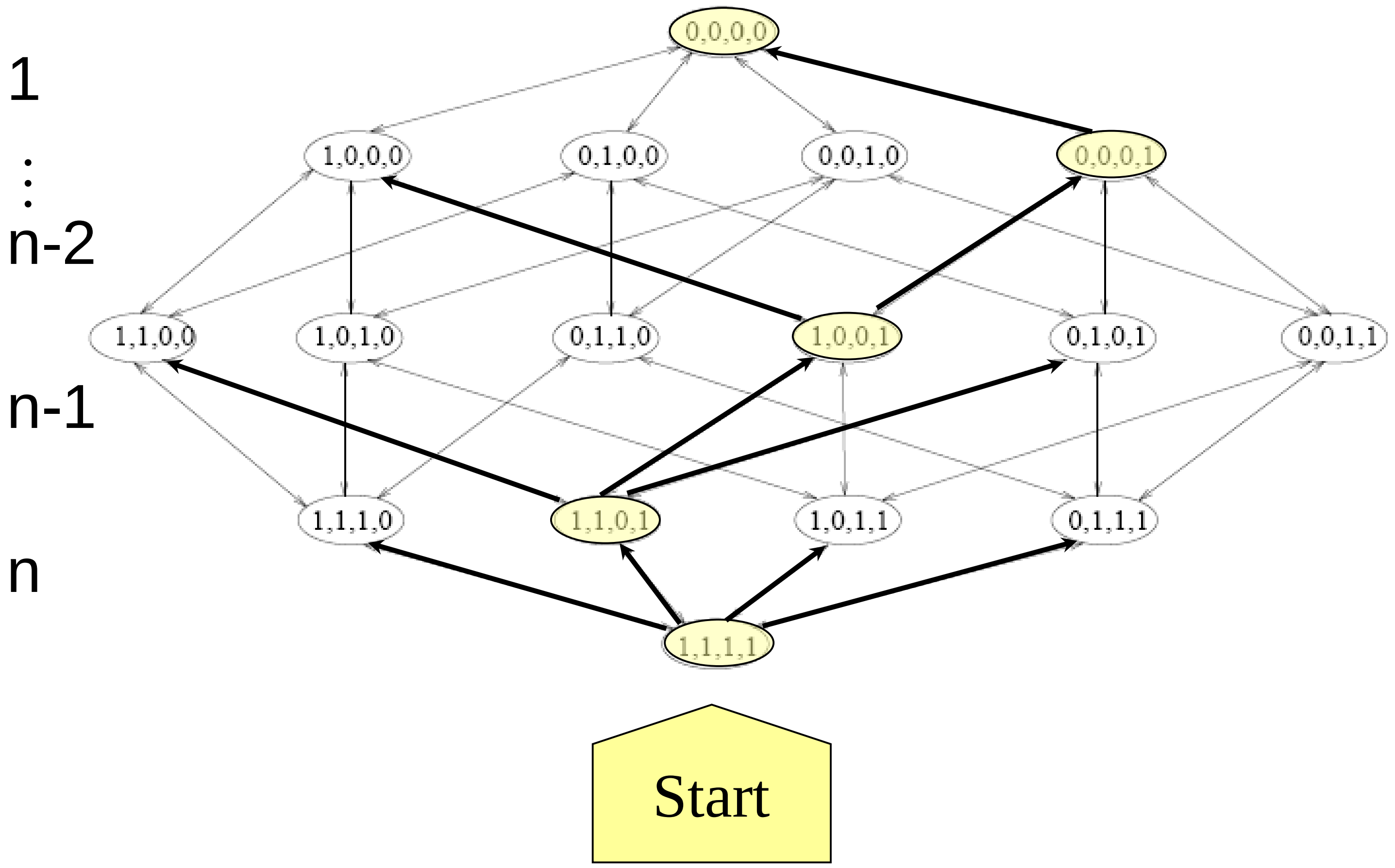
  like *CART (Breiman, 1984)* or *C4.5 (Quinlan, 1993)*



At each step, choose the feature that "reduces entropy" most. Work towards "node purity".

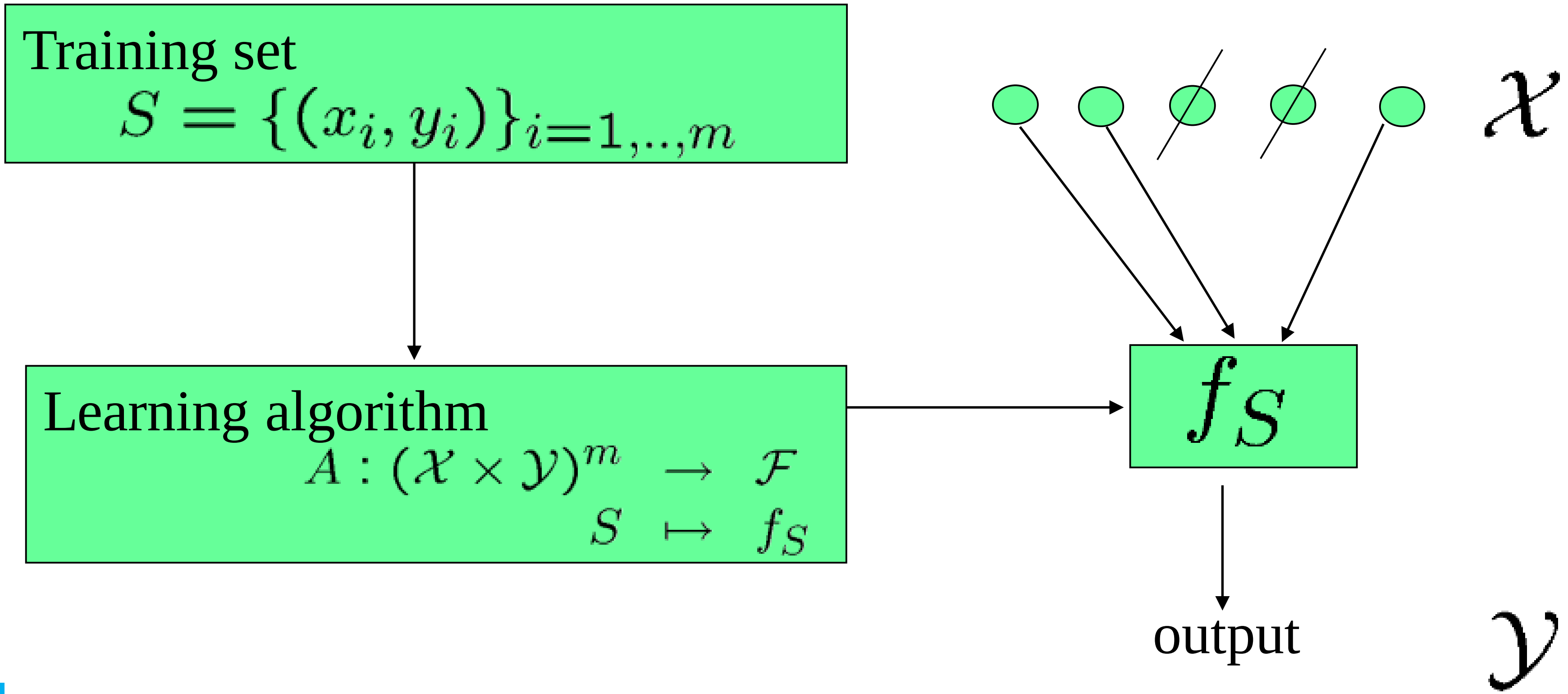
Backward Elimination (wrapper)

Also referred to as SBS: Sequential Backward Selection

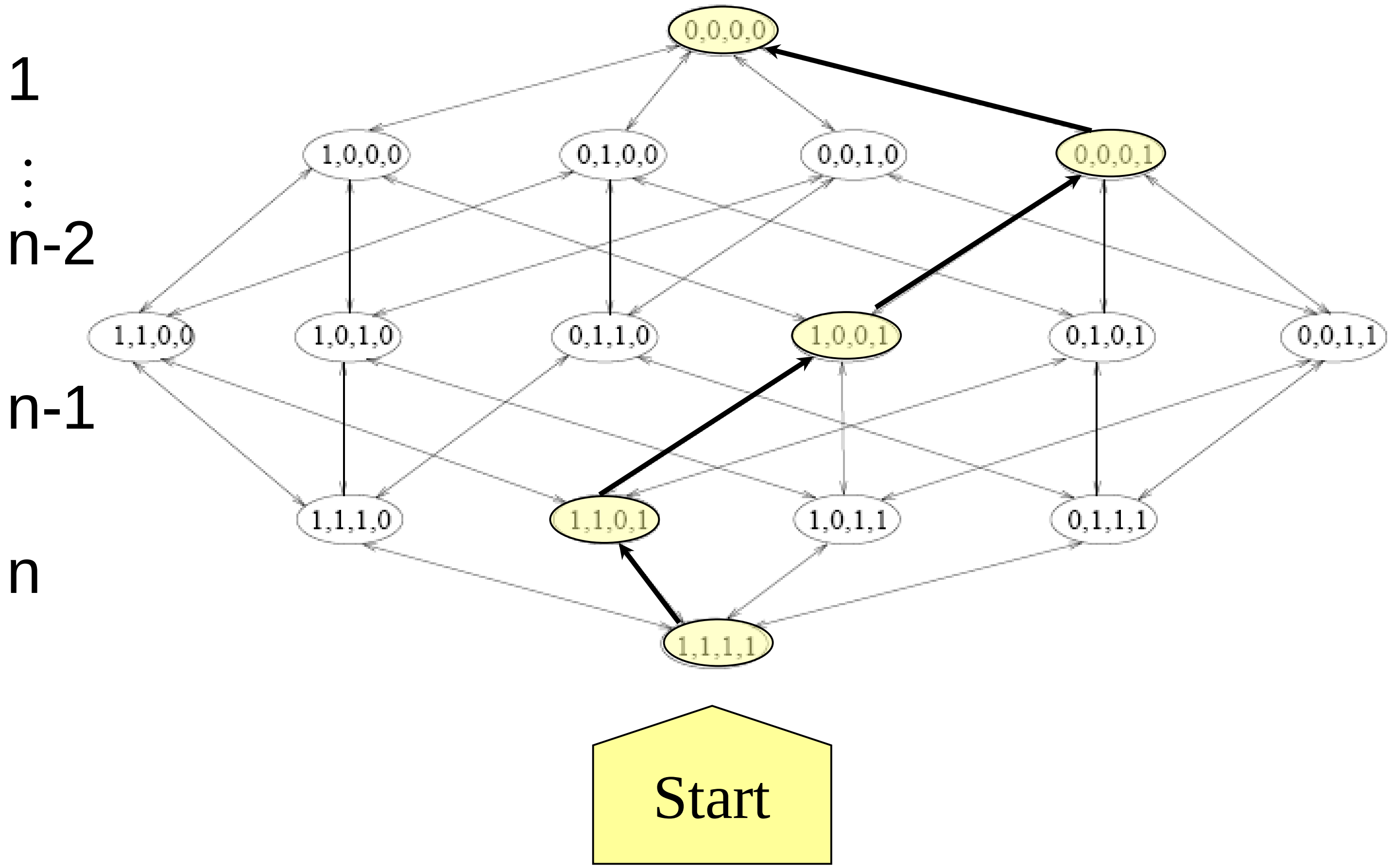


Embedded

- Definition: an embedded feature selection method is a *machine learning algorithm* that returns a model using a limited number of features.



Backward Elimination (embedded)



Backward Elimination: RFE
Recursive Feature Elimination

RFE-SVM, Guyon, Weston, et al, 2002

Start with all the features.

- Train a learning machine f on the current subset of features by minimizing a risk functional J[f].
- For each (remaining) feature X_i, estimate, without retraining f, the change in J[f] resulting from the removal of X_i.
- Remove the feature $X_{v(k)}$ that results in improving or least degrading J.

Embedded method for SVM, kernel methods, neural nets.

RFE works on feature ranking system

1- The model is fit on linear regression based on all variables.

2-Then it calculates variable coefficients and their importance.

3-Then It ranks the variable on the basis on linear regression fit and

4-Then remove low ranking variable in each iteration.

scikit package can do this automatically by defining the number of features needed to reduce to

```
from sklearn.datasets import make_friedman1
from sklearn.feature_selection import RFE
from sklearn.svm import SVR
X, y = make_friedman1(n_samples=50, n_features=10, random_state=0)
estimator = SVR(kernel="linear")
selector = RFE(estimator, 5, step=1)
selector = selector.fit(X, y)
selector.support_


selector.ranking_
```

Filter Methods	Wrapper Methods
Usually fast	Learner is considered a black-box
Provide generic selection of features, not tuned by given learner (universal)	Interface of the black-box is used to score subsets of variables according to the predictive power of the learner when using the subsets.
This is also often criticised (feature set not optimized for used classifier)	Results vary for different learners
Sometimes used as a preprocessing step for other methods	One needs to define: • How to search the space of all possible variable subsets ? • How to assess the prediction performance of a learner ?

Add/Remove feature summary

- Many algorithms can be turned into embedded methods for feature selections by using the following approach:

1. Choose an objective function that measure how well the model returned by the algorithm performs
2. "Differentiate" (or sensitivity analysis) this objective function according to the σ parameter (i.e. how does the value of this function change when one feature is removed and the algorithm is rerun)
3. Select the features whose removal (resp. addition) induces the desired change in the objective function (i.e. minimize error estimate, maximize alignment with target, etc.)

What makes this method an 'embedded method' is the use of the structure of the learning algorithm to compute the gradient and to search/weight relevant features.

Model selection: choosing estimators and their parameters

```python
from sklearn import datasets, svm
digits = datasets.load_digits()
X_digits = digits.data
y_digits = digits.target
svc = svm.SVC(C=1, kernel='linear')
svc.fit(X_digits[:-100], y_digits[:-100]).score(X_digits[-100:],
y_digits[-100:])
```

```python
import numpy as np
X_folds = np.array_split(X_digits, 3)
y_folds = np.array_split(y_digits, 3)
scores = list()
for k in range(3):
    # We use 'list' to copy, in order to 'pop' later on
    X_train = list(X_folds)
    X_test  = X_train.pop(k)
    X_train = np.concatenate(X_train)
    y_train = list(y_folds)
    y_test  = y_train.pop(k)
    y_train = np.concatenate(y_train)
    scores.append(svc.fit(X_train, y_train).score(X_test,
y_test))
print(scores)
```

```python
from sklearn.model_selection import KFold, cross_val_score
 X = ["a", "a", "a", "b", "b", "c", "c", "c", "c", "c"]
k_fold = KFold(n_splits=5)
 for train_indices, test_indices in k_fold.split(X):
              print('Train: %s | test: %s' % (train_indices, test_indices))
```

```python
[svc.fit(X_digits[train], y_digits[train]).score(X_digits[test], y_digits[test])
for train, test in k_fold.split(X_digits)]
```

```python
cross_val_score(svc, X_digits, y_digits, cv=k_fold, n_jobs=-1)

cross_val_score(svc, X_digits, y_digits, cv=k_fold, scoring='precision_macro')
```

```
import numpy as np
from sklearn.model_selection
import cross_val_score
from sklearn
import datasets, svm
```

```
digits = datasets.load_digits()
X = digits.data
y = digits.target
svc = svm.SVC(kernel='linear')
C_s = np.logspace(-10, 0, 10)
scores = list()
scores_std = list()
for C in C_s:
    svc.C = C
    this_scores = cross_val_score(svc, X, y, cv=5, n_jobs=1)
    scores.append(np.mean(this_scores))
    scores_std.append(np.std(this_scores))
# Do the plotting
import matplotlib.pyplot as plt
plt.figure(1, figsize=(4, 3))
plt.clf()
plt.semilogx(C_s, scores)
plt.semilogx(C_s, np.array(scores) + np.array(scores_std), '
plt.semilogx(C_s, np.array(scores) - np.array(scores_std), '
locs, labels = plt.yticks()
plt.yticks(locs, list(map(lambda x: "%g" % x, locs)))
plt.ylabel('CV score')
plt.xlabel('Parameter C')
plt.ylim(0, 1.1)
plt.show()
```

```
from sklearn.model_selection import LeaveOneGroupOut
X = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
y = np.array([1, 2, 1, 2])
groups = np.array([1, 1, 2, 2])
logo = LeaveOneGroupOut()
logo.get_n_splits(X, y, groups)

logo.get_n_splits(groups=groups)
# 'groups' is always required

print(logo)

for train_index, test_index in logo.split(X, y, groups):
   print("TRAIN:", train_index, "TEST:", test_index)
   X_train, X_test = X[train_index], X[test_index]
   y_train, y_test = y[train_index], y[test_index]
   print(X_train, X_test, y_train, y_test)
```

```
from sklearn.model_selection
import KFold, cross_val_score
from sklearn.preprocessing
import PolynomialFeatures
from sklearn.linear_model
import LinearRegression
from sklearn.pipeline import
Pipeline
```

```
x = pd.DataFrame(Auto.horsepower)
y = Auto.mpg

model = LinearRegression()
model.fit(x, y)
print model.intercept_
print model.coef_
```

```
k_fold = KFold(n_splits=x.shape[0])
test = cross_val_score(model, x, y, cv=k_fold,
scoring = 'neg_mean_squared_error', n_jobs=-1)
print np.mean(-test)
```