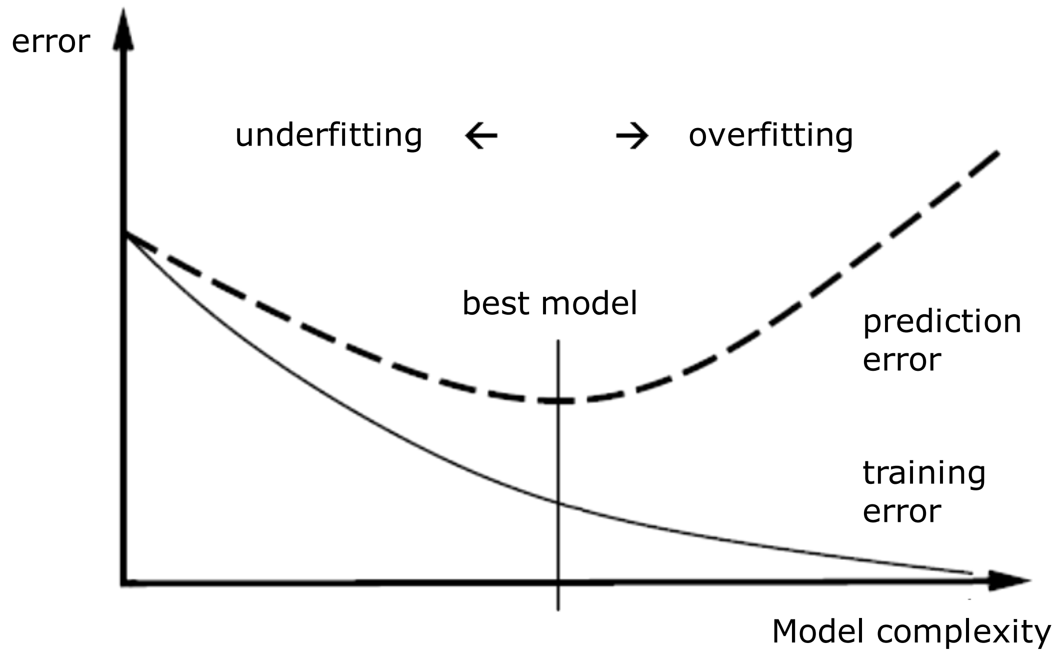
7

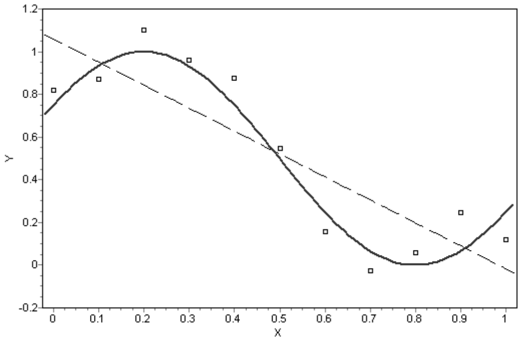
Subset selection, model building, state space, heuristic search

The topic

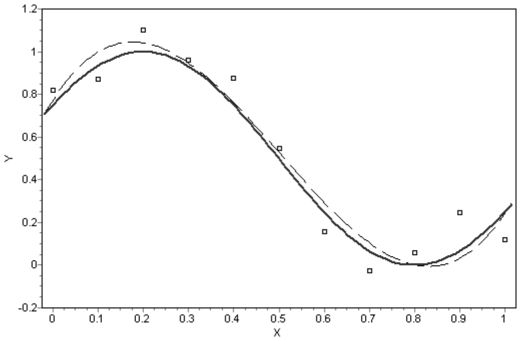
- This lecture continues and expands the topic on finding a good model for your data



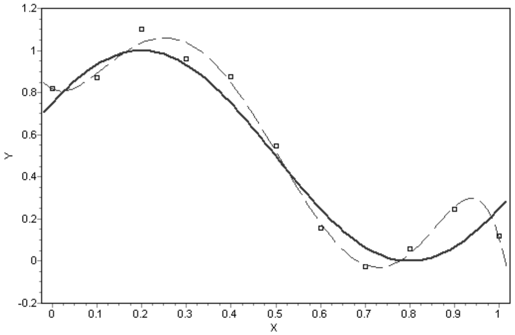
Model complexity



1st degree polynomial



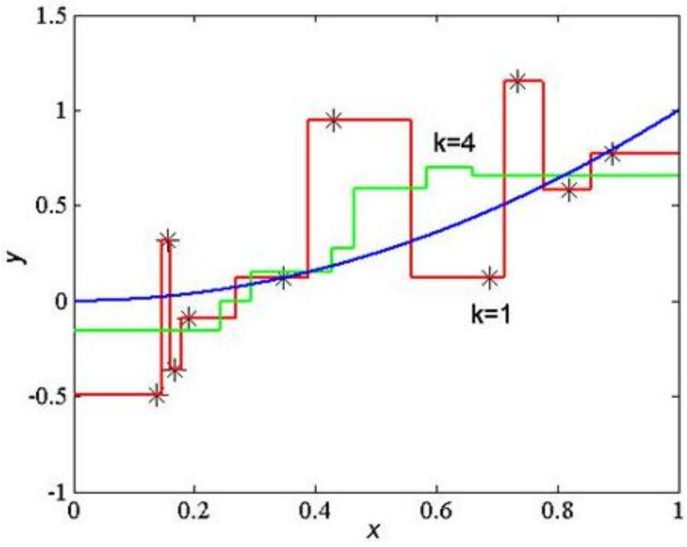
3rd degree polynomial



7th degree polynomial

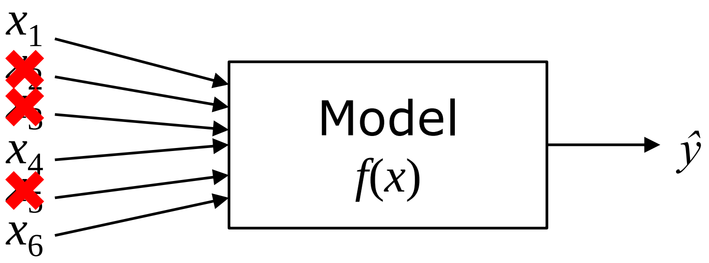
Finding a good model, by balancing between *underfitting* and *overfitting*, is not only about finding some best value for some hyperparameter, for example,

- selecting best degree for the polynomial in linear regression;
- selecting best *k* value for *k*-NN method;
- etc.



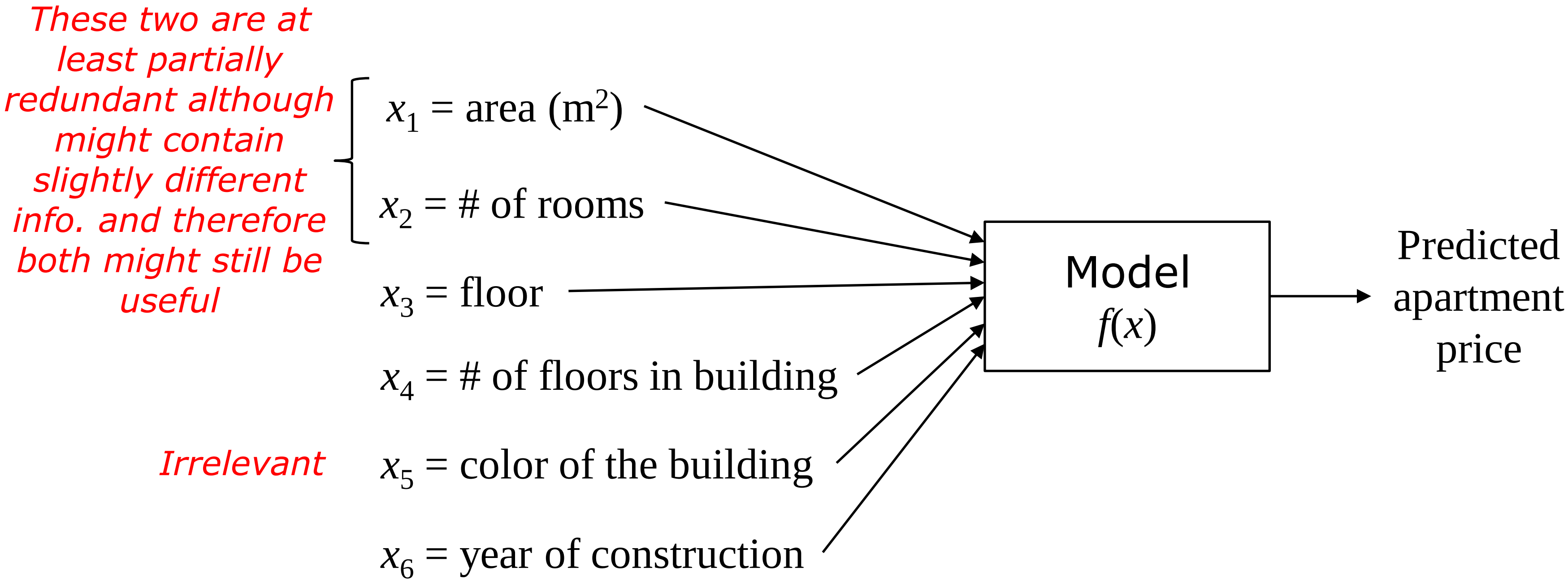
Not all features might be useful

- Some of the features might be irrelevant or redundant. Such features make the models unnecessary complex, resulting in potentially lower predictive performance
  - Irrelevant features don’t give any useful information for prediction purposes (*for example, car color when you want to estimate fuel consumption of the car*)
  - Redundant features strongly correlate with each other and so it might be that only one of them is actually useful (*for example volume of engine (engine displacement) in cubic centimeters vs. cubic inches, or for example product’s price vs. VAT of the price*)



$$f(x_1, \cancel{x_2,} \cancel{x_3,} x_4, \cancel{x_5,} x_6)$$

Not all features might be useful



These two are at least partially redundant although might contain slightly different info. and therefore both might still be useful

x_1 = area (m^2)

x_2 = # of rooms

x_3 = floor

x_4 = # of floors in building

Irrelevant

x_5 = color of the building

x_6 = year of construction

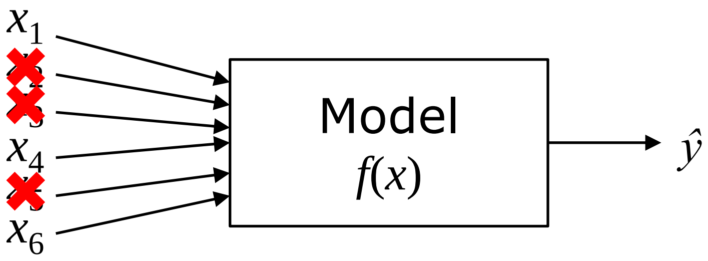
Model
$f(x)$

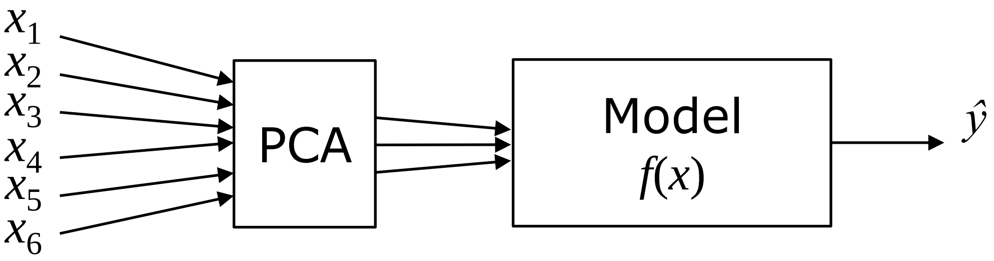
Predicted apartment price

Also not sure about the relevance of x_3 and x_4.
In either case some kind of selection procedure is needed to remove the excess complexity and noise.

Dimensionality reduction

- In order to obtain a statistically reasonable result, the amount of data needed to support the result grows fast, it can grow even exponentially, with the dimensionality.
- Furthermore, it is possible that even if you keep only the useful features, you still have too many of them.
- Solutions to these problems are:
  - Remove features that are the least useful
  - Use methods that transforms the data in high-dimensional space to a space of fewer dimensions, for example, *Principal Component Analysis* (PCA) and others.



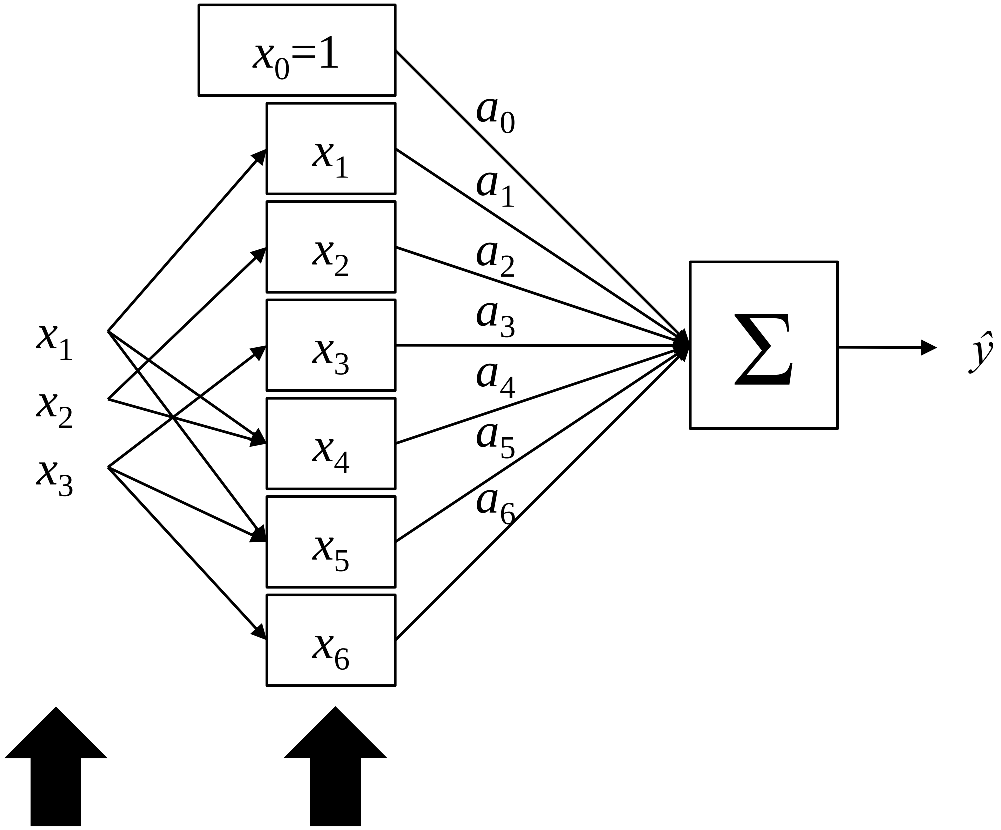


$$f(x_1, \cancel{x_2}, \cancel{x_3}, x_4, \cancel{x_5}, x_6)$$

$$f(x'_1, x'_2, x'_3)$$

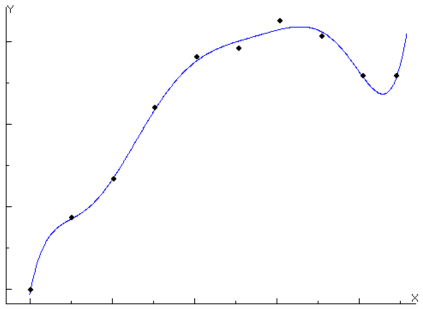
In this lecture we talk about the first solution.

Dimensionality reduction



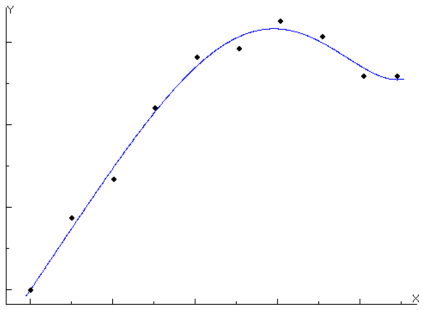
In linear regression, selecting the necessary features can be done either on the level of original input variables or on the level of the transformations of those variables.

Selecting combination of transformations



Here the question is about the choice of combination of transformations but it can be considered to be the same dimensionality reduction problem

$$\hat{y} = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$$



(Here the highest degree (7th) term stays in the model but some of lower degree terms are removed)

$$\hat{y} = a_0 + a_1 x + \cancel{a_2 x^2} + \cancel{a_3 x^3} + a_4 x^4 + \cancel{a_5 x^5} + \cancel{a_6 x^6} + a_7 x^7$$

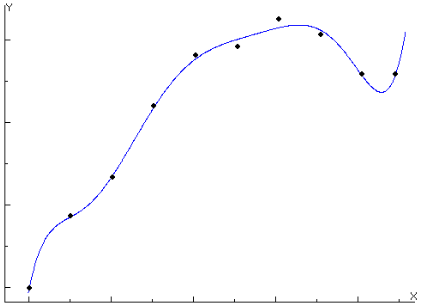
Feature subset selection

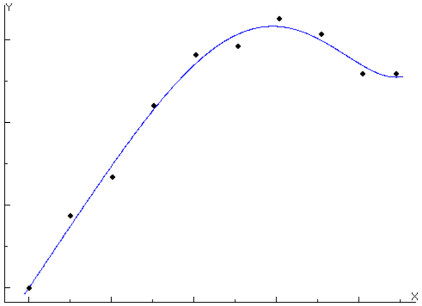
- But how to find which features to use in my model?
- This problem is called:
    - Feature subset selection because we are searching for the “best” subset of the full set of features
    - Model selection because we are choosing one “best” model from a number of possible models
    - Model building because we are building a model from the available components (features, their transformations)

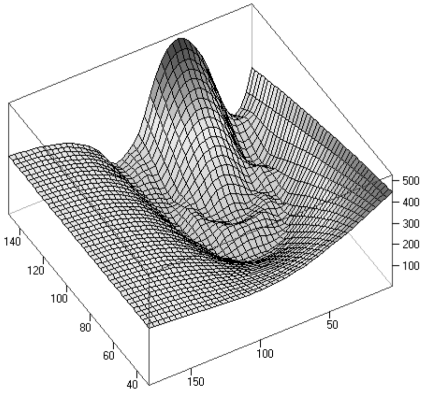
- *The lower is the number of examples in your dataset (compared to the number of features) the more important is the feature selection problem*

“Manual” selection

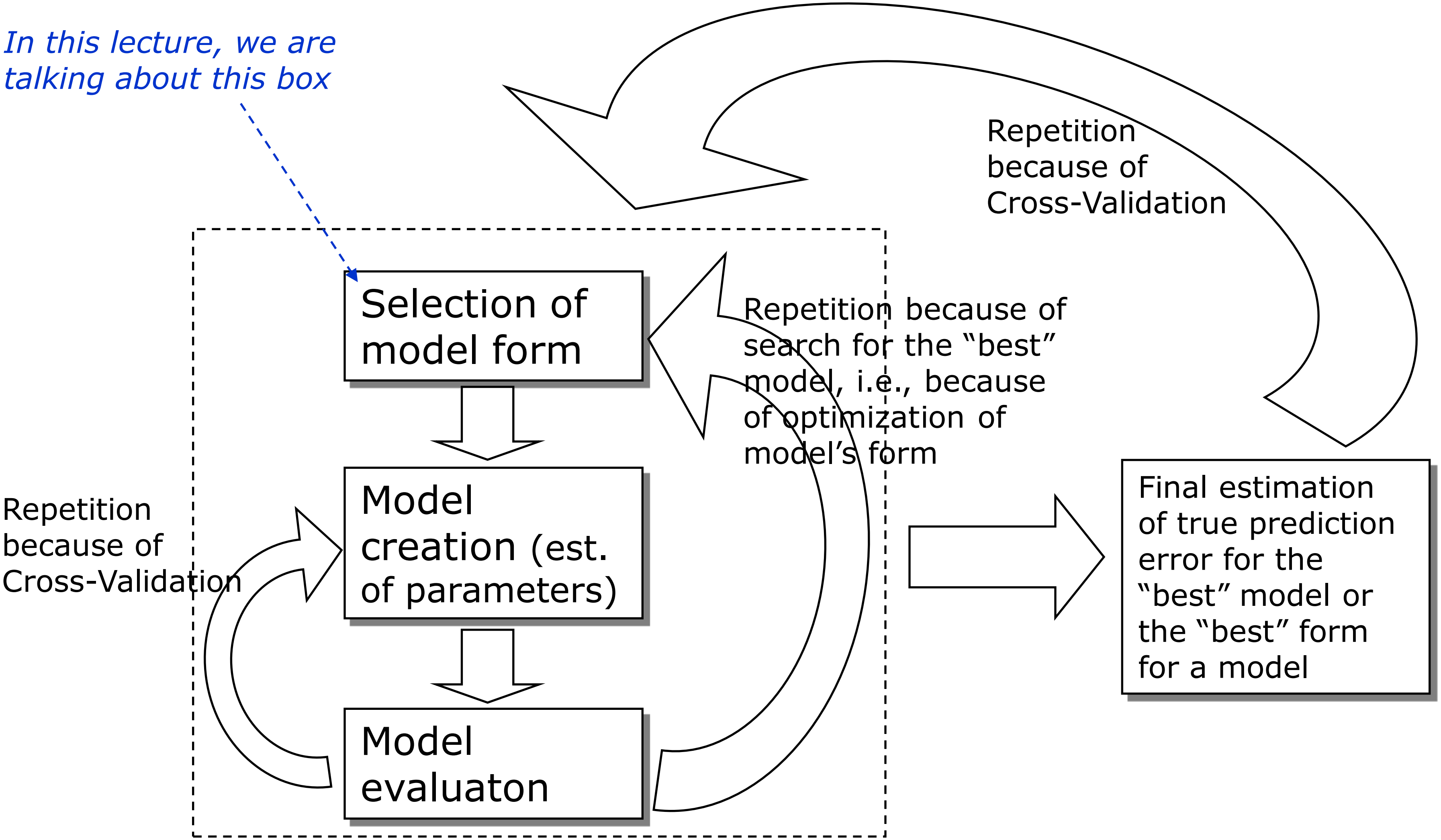
- “Manual” feature selection is practical only with small number of features and/or features that can be easily understood
  - While working with just a few features, the data and the model can be visualized, interpreted, and evaluated relatively easily
  - In low dimensionality, humans have impressive ability to recognize complex patterns
- But in the general case, we need to formalize this process and make it automatic







General scheme



In this lecture, we are talking about this box

Selection of model form

Model creation (est. of parameters)

Model evaluaton

Repetition because of Cross-Validation

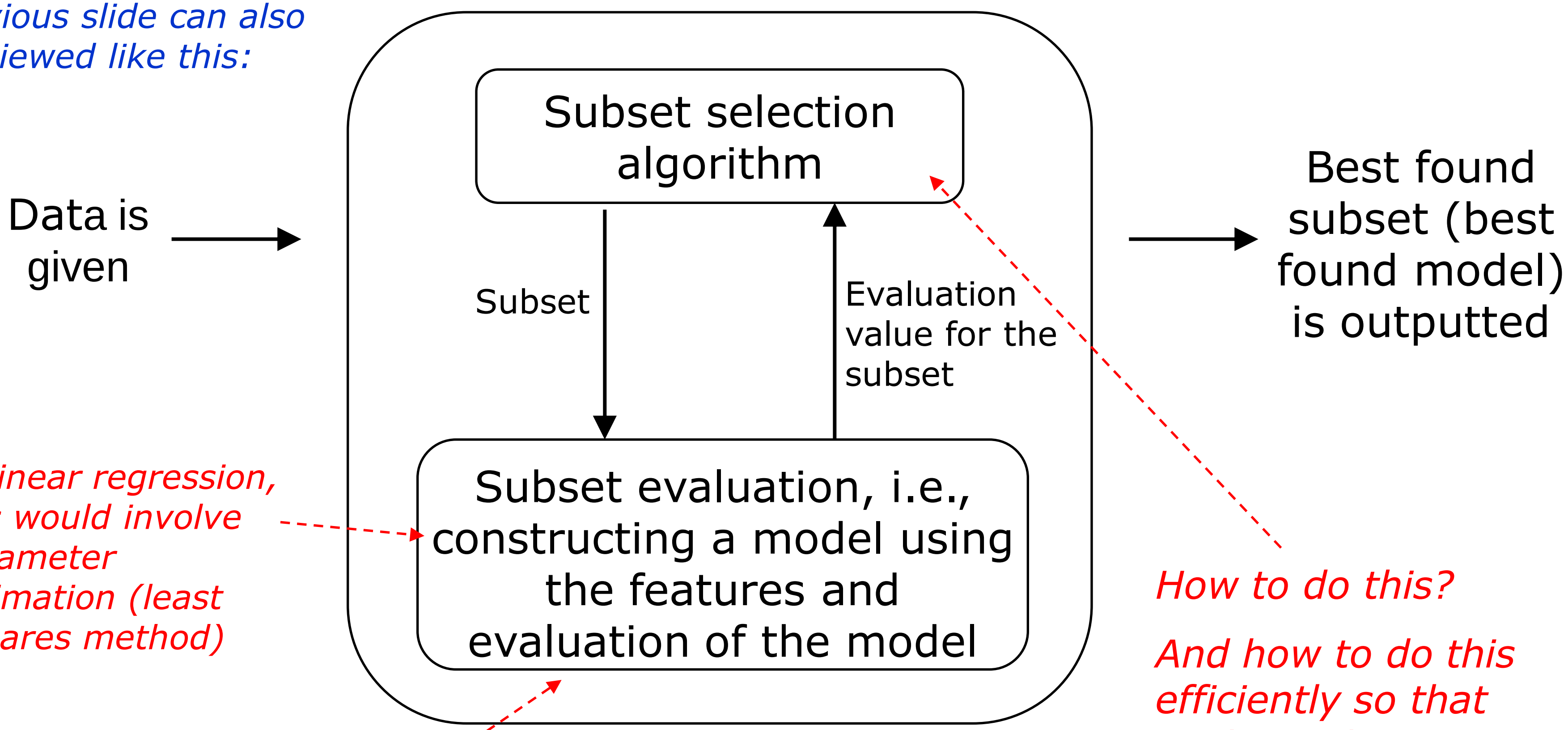
Repetition because of search for the "best" model, i.e., because of optimization of model's form

Repetition because of Cross-Validation

Final estimation of true prediction error for the "best" model or the "best" form for a model

Subset selection

The main loop in the previous slide can also be viewed like this:



Data is given

Subset selection algorithm

Subset

Subset evaluation, i.e., constructing a model using the features and evaluation of the model

Evaluation value for the subset

Best found subset (best found model) is outputted

In linear regression, this would involve parameter estimation (least squares method)

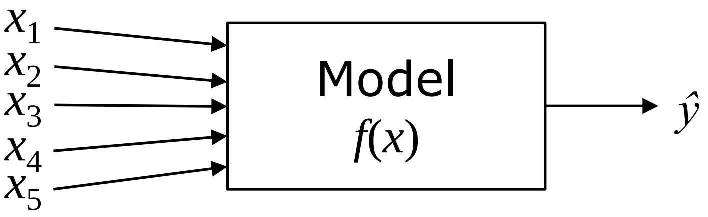
For example, Hold-Out, Cross-Validation, complexity-penalization criteria

How to do this?

And how to do this efficiently so that we keep the number of evaluated subsets/models to a minimum?

Bit representation

Example with 5 features.
$f(x)$ is lin. reg. model or
k-NN or other



In case of linear regression, full model would look like, e.g.:

$$\hat{y} = a_0 + a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4 + a_5x_5$$

- We must find which combination (subset) of the features would allow us to make the best model for our data
- Lets indicate inclusion/exclusion of features with one bit
  - The inclusion of the free parameter a_0 is usually fixed
  - For 5 features we need 5 bits
- We want to find a combination of bit values so that the corresponding model has the best evaluation (e.g., Hold-Out, Cross-Validation, complexity-penalization criteria etc.)
  - This is a search problem and a (combinatorial) optimization problem

00000
00001
00010
00011
00100
00101
00110
00111
01000
01001
...
11111

Selection of the best combination

- Search through all combinations (*exhaustive search*, *brute force*)
  - Evaluate all possible combinations (all possible models) and choose the best
  - But evaluating one combination is very time consuming – must create the model and evaluate it
  - Big problem: the number of combinations of features grows exponentially

The number of all possible combinations is 2^m, where m is the total number of features defined.

If $m = 5$ then $2^m = 32$

If $m = 10$ then $2^m = 1024$

If $m = 20$ then $2^m = 1048576$

If $m = 30$ then $2^m = 1073741824$

If $m = 70$ then $2^m = 1180591620717411303424$

If $m = 270$, we exceeded the number of atoms in the known universe

If this takes about 30 min then this takes about a month

And this takes more than the age of the universe

Heuristic search

- We need a type of search that enables us to find good combinations/models without requiring huge computational resources
- Solution – use heuristics
  - *Heuristics are based on previous experience and readily available, though loosely applicable, information. They enable reaching the goal by sacrificing optimality.*
- The basic idea – consider only the most promising solutions (feature combinations / models) thus significantly saving time
  - Advantage – significant savings of time (e.g., not days, months, or years but seconds, minutes, or hours)
  - Disadvantage – such algorithms do not guarantee optimality of the results, instead they can give us good solutions in within a reasonable time. Good news – this is usually good enough.

Heuristic search in subset selection

- In short: start with some combination and, by modifying it and evaluating the modifications, go in direction of the most promising combinations.
- Heuristic search in subset selection can be characterized using these five basic components:

1. Initial state

    The combination with which we begin our search.

2. State-transition operators

    The available ways to modify the combination.

3. Search strategy

    Which combinations to evaluate and where to go next.

4. State evaluation criterion

    Criterion for evaluation of the created combinations.

5. Termination condition

    When to stop the search.

Heuristic search in subset selection

- Typical variations for the basic components:

  1. Initial state

     Empty set (no features included, "00000"), full set (all features included, "11111"), random subset.

  2. State-transition operators

     The two typical operators: addition of one feature (0 -> 1), deletion of one feature (1 -> 0). There can be also other operators, e.g., genetic algorithms use crossover and mutation

  3. Search strategy

     Hill Climbing, Beam Search, Floating Search, Simulated Annealing, imitation of evolution (in Genetic Algorithms) etc.

  4. State evaluation criterion

     In our case: Hold-out, Cross-Validation, MDL, AIC, AICC etc.

  5. Termination condition

     When none of the operators can find a better state (local minimum found). When none of the operators are applicable anymore. When a predefined number of iterations are done. When a predefined error rate is reached. When a predefined time is elapsed. Etc.

SFS – concrete algorithm

- One of the simplest subset selection methods is Sequential Forward Selection (SFS):

  1. Initial state

     Empty set (no features included, “00000”).

  2. State-transition operators

     Addition of one feature to the model (0 -> 1)

  3. Search strategy

     A variation of Hill Climbing

  4. State evaluation criterion

     (*We can use any suitable criterion*)

  5. Termination condition

     a) When none of the operators can find a better state (local minimum found).

     b) When none of the operators are applicable (i.e., for SFS this means that the state with all bits equal to 1 is reached).

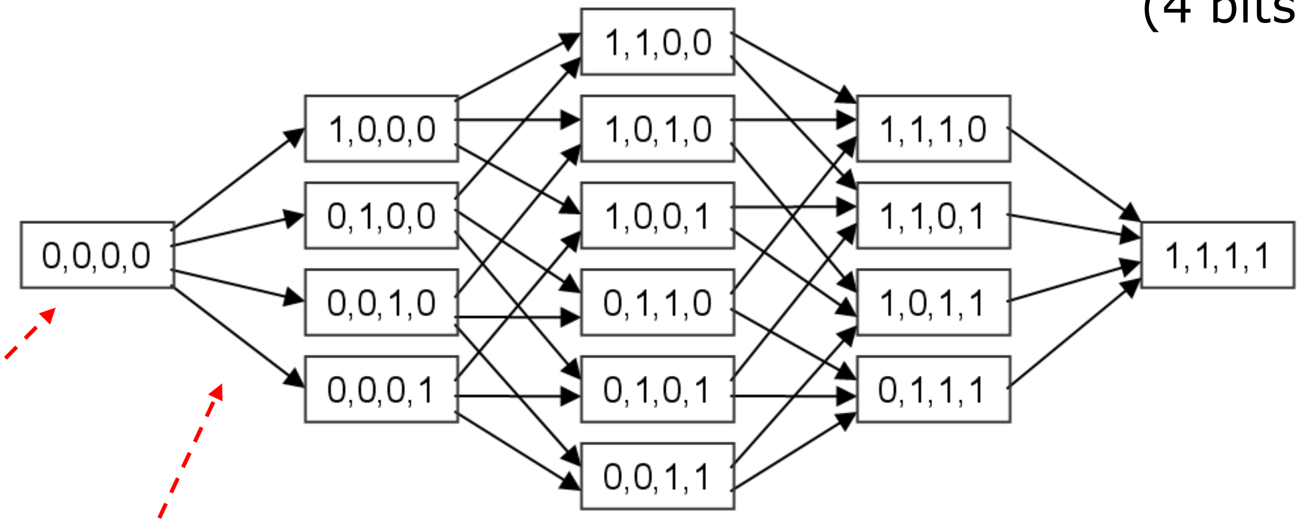
     [In next slides, the (b) version of the algorithm is explained.]

Initial state together with state-transition operators define the state space.

State space of SFS

- Initial state together with state-transition operators define the state space.

A tiny example with 4 features (4 bits)



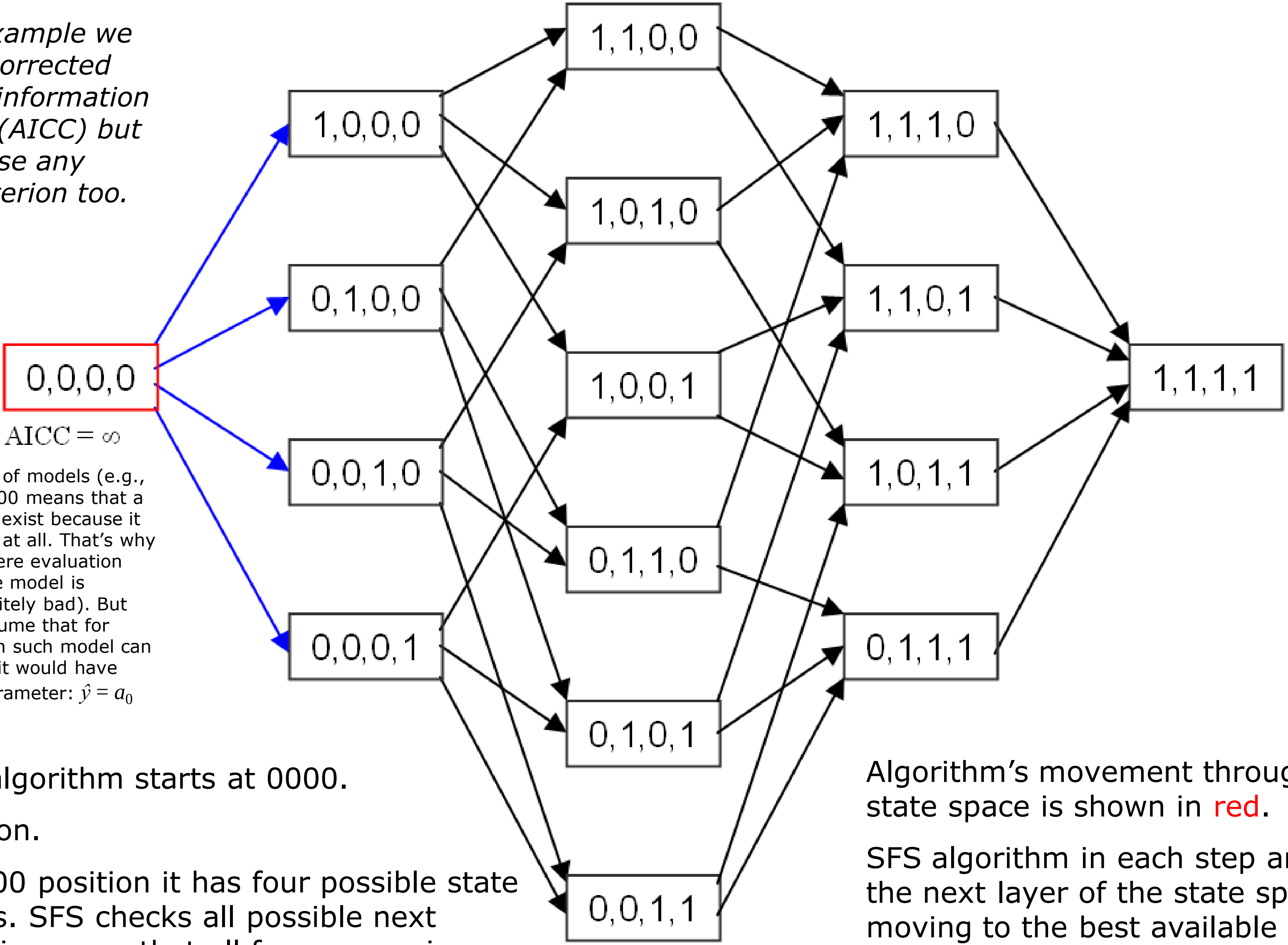
Initial state

State-transitions

To evaluate a state, we must create the corresponding model (estimate its parameters etc.) and estimate prediction error using the chosen criterion (furthermore, if Cross-Validation is used, the time is multiplied). This also means that evaluation is slow.

SFS example

In this example we use the corrected Akaike's information criterion (AICC) but we can use any other criterion too.



For many types of models (e.g., *k*-NN), state 0000 means that a model does not exist because it has no features at all. That's why this state has here evaluation *infinity* (i.e., the model is considered infinitely bad). But we can also assume that for linear regression such model can be created and it would have just the free parameter: $\hat{y} = a_0$

The SFS algorithm starts at 0000.

1st iteration.

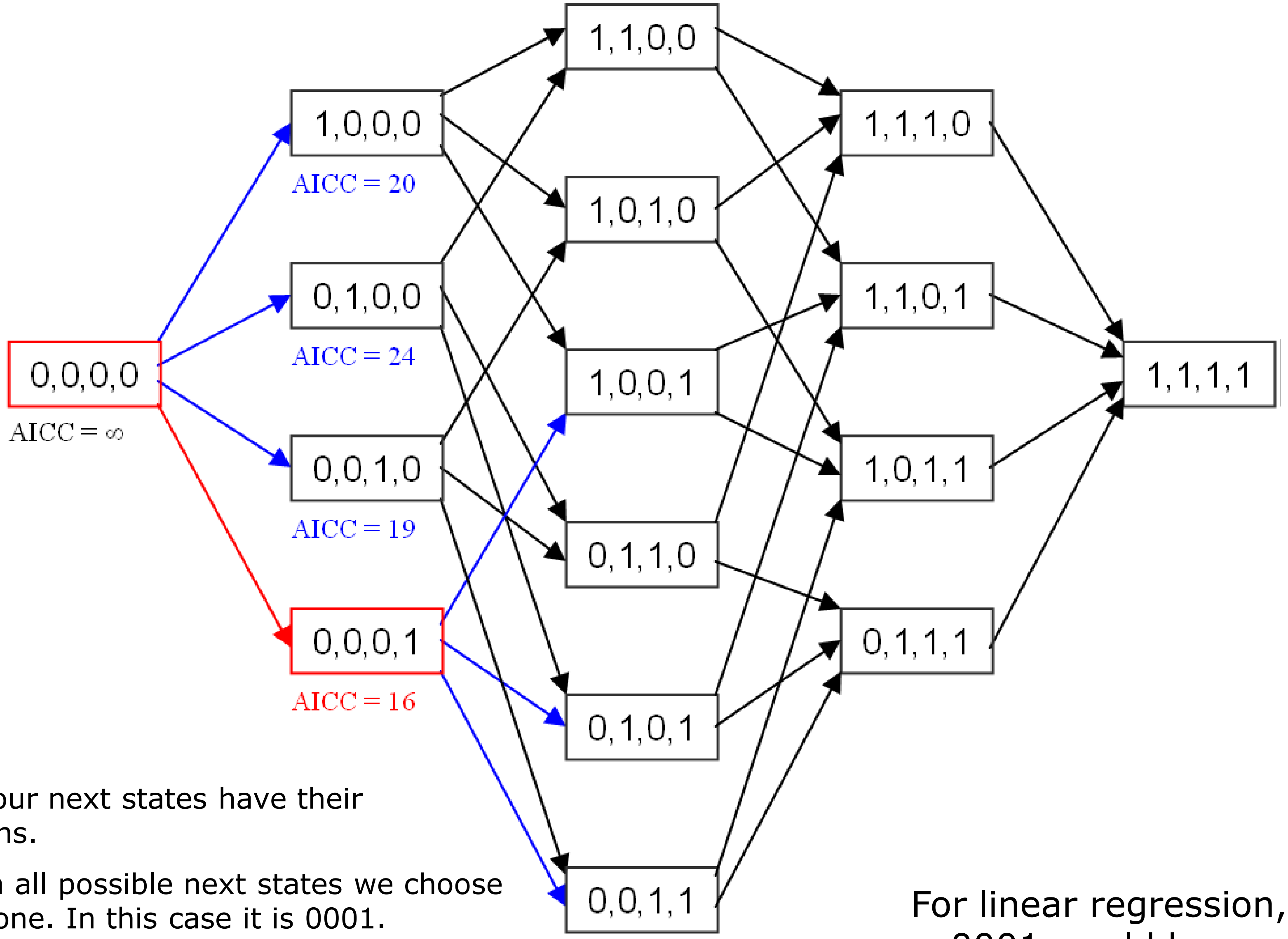
At the 0000 position it has four possible state transitions. SFS checks all possible next states. This means that all four regression models will be created and evaluated so that all those four states have their evaluations.

Algorithm's movement through the state space is shown in red.

SFS algorithm in each step arrives in the next layer of the state space by moving to the best available state.

(If the "a" version of the termination condition is used, the algorithm moves to next layer only if the next state is better than the current state.)

SFS example



Now all four next states have their evaluations.

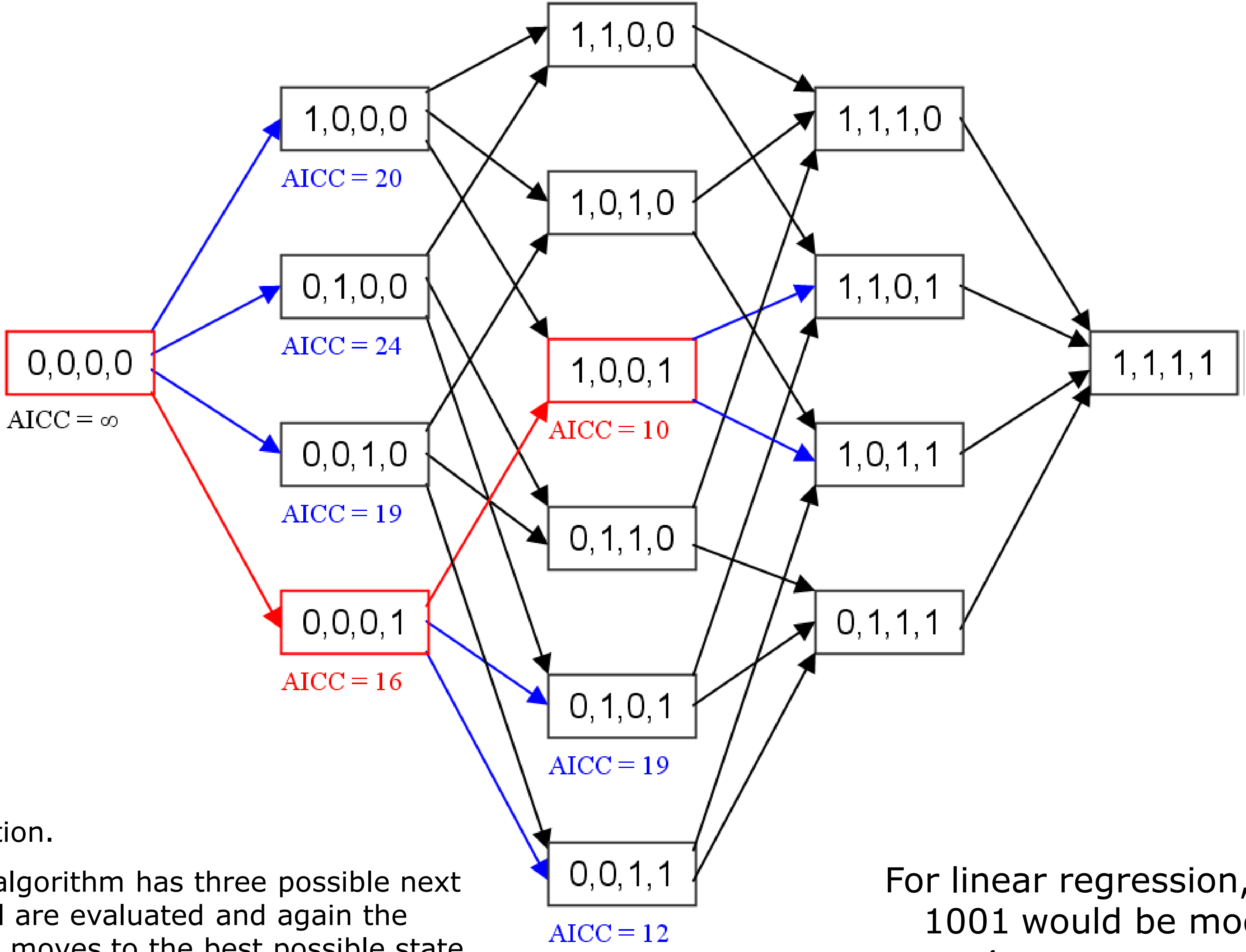
Now from all possible next states we choose the best one. In this case it is 0001.

0001 becomes the new current state. And we proceed to the next iteration.

For linear regression, state 0001 would be model:

$$\hat{y} = a_0 + a_4 x_4$$

SFS example



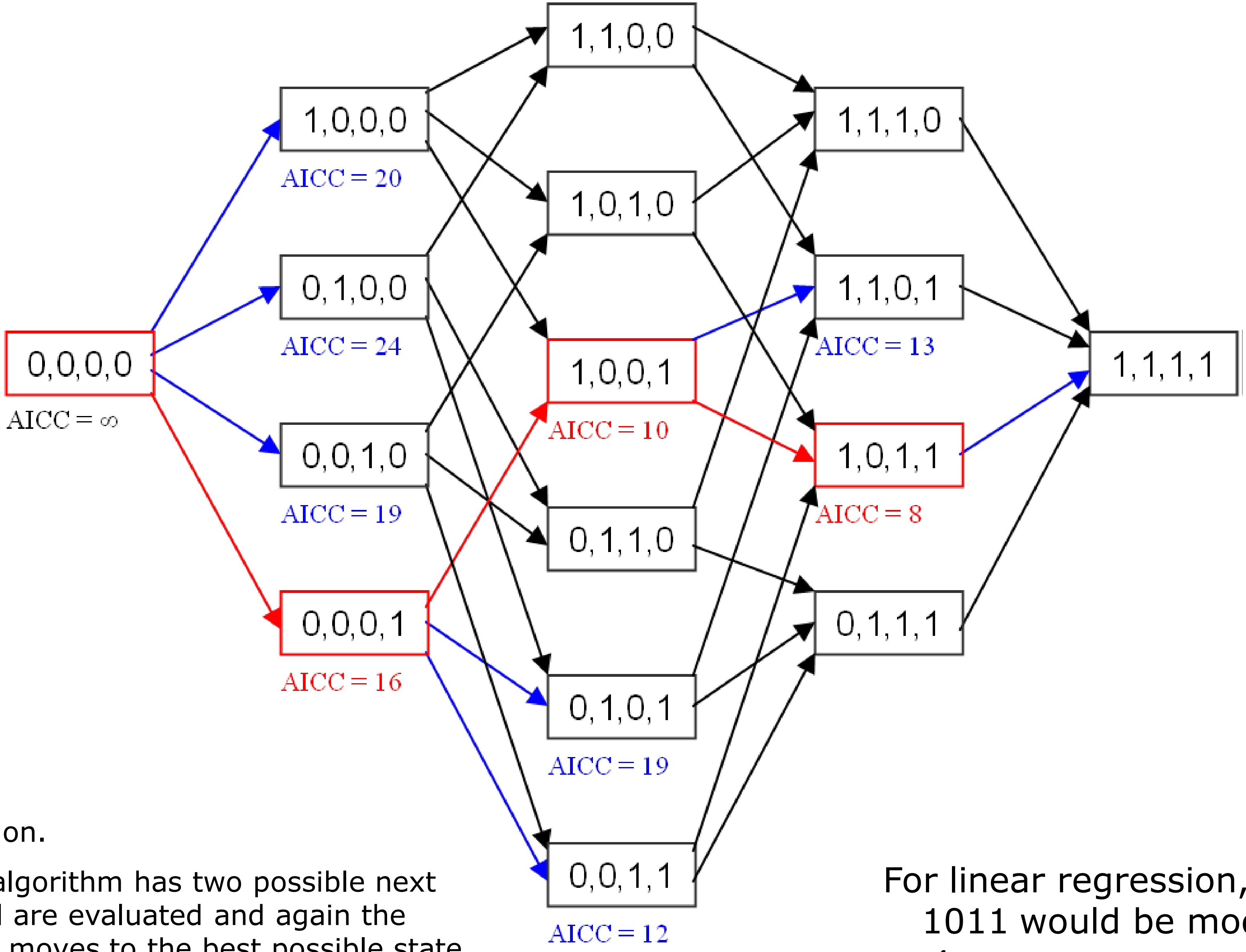
2nd iteration.

Now the algorithm has three possible next states. All are evaluated and again the algorithm moves to the best possible state.

For linear regression, state 1001 would be model:

$$\hat{y} = a_0 + a_1 x_1 + a_4 x_4$$

SFS example



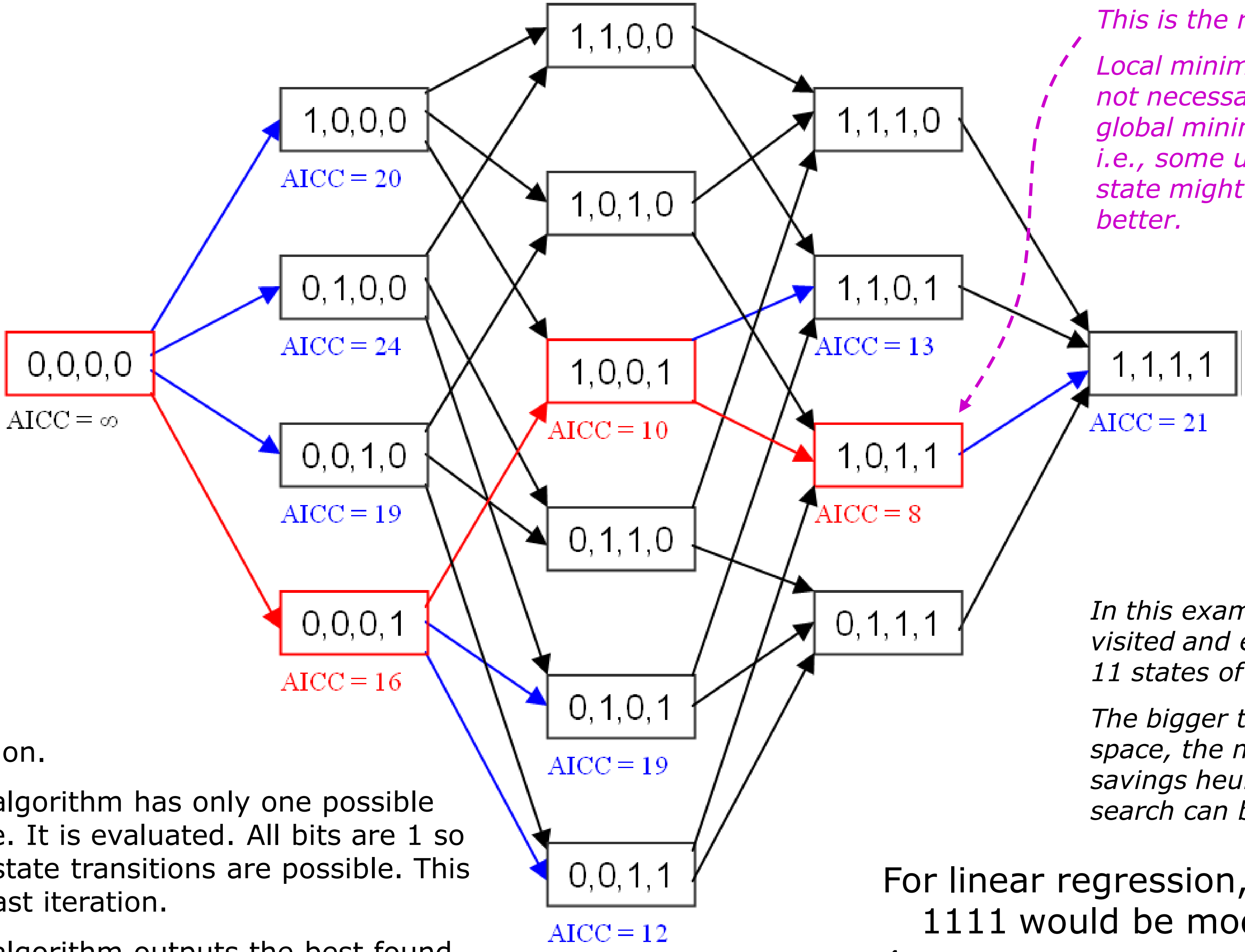
3rd iteration.

Now the algorithm has two possible next states. All are evaluated and again the algorithm moves to the best possible state.

For linear regression, state 1011 would be model:

$$\hat{y} = a_0 + a_1x_1 + a_3x_3 + a_4x_4$$

SFS example



0,0,0,0

AICC = ∞

1,0,0,0

AICC = 20

0,1,0,0

AICC = 24

0,0,1,0

AICC = 19

0,0,0,1

AICC = 16

1,1,0,0

1,0,1,0

1,0,0,1

AICC = 10

0,1,1,0

0,1,0,1

AICC = 19

0,0,1,1

AICC = 12

1,1,1,0

1,1,0,1

AICC = 13

1,0,1,1

AICC = 8

0,1,1,1

1,1,1,1

AICC = 21

This is the result.

Local minimum. But not necessarily global minimum, i.e., some unvisited state might be better.

In this example, we visited and evaluated 11 states of 16 total.

The bigger the state space, the more time savings heuristic search can bring.

4th iteration.

Now the algorithm has only one possible next state. It is evaluated. All bits are 1 so no more state transitions are possible. This was the last iteration.

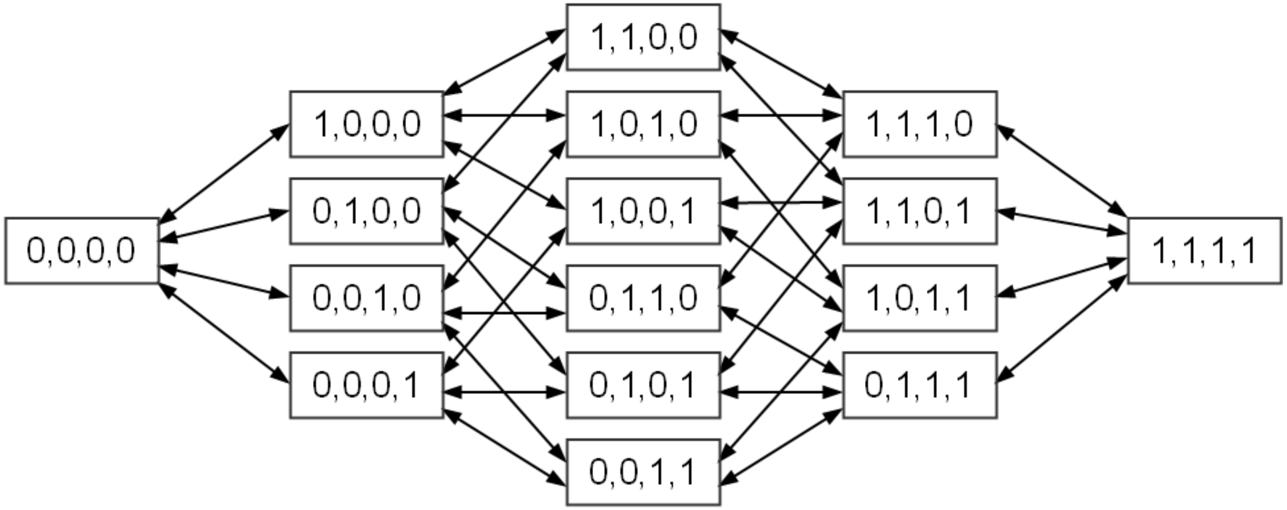
Now the algorithm outputs the best found state which in this case is 1011 with AICC=8.

For linear regression, state 1111 would be model:

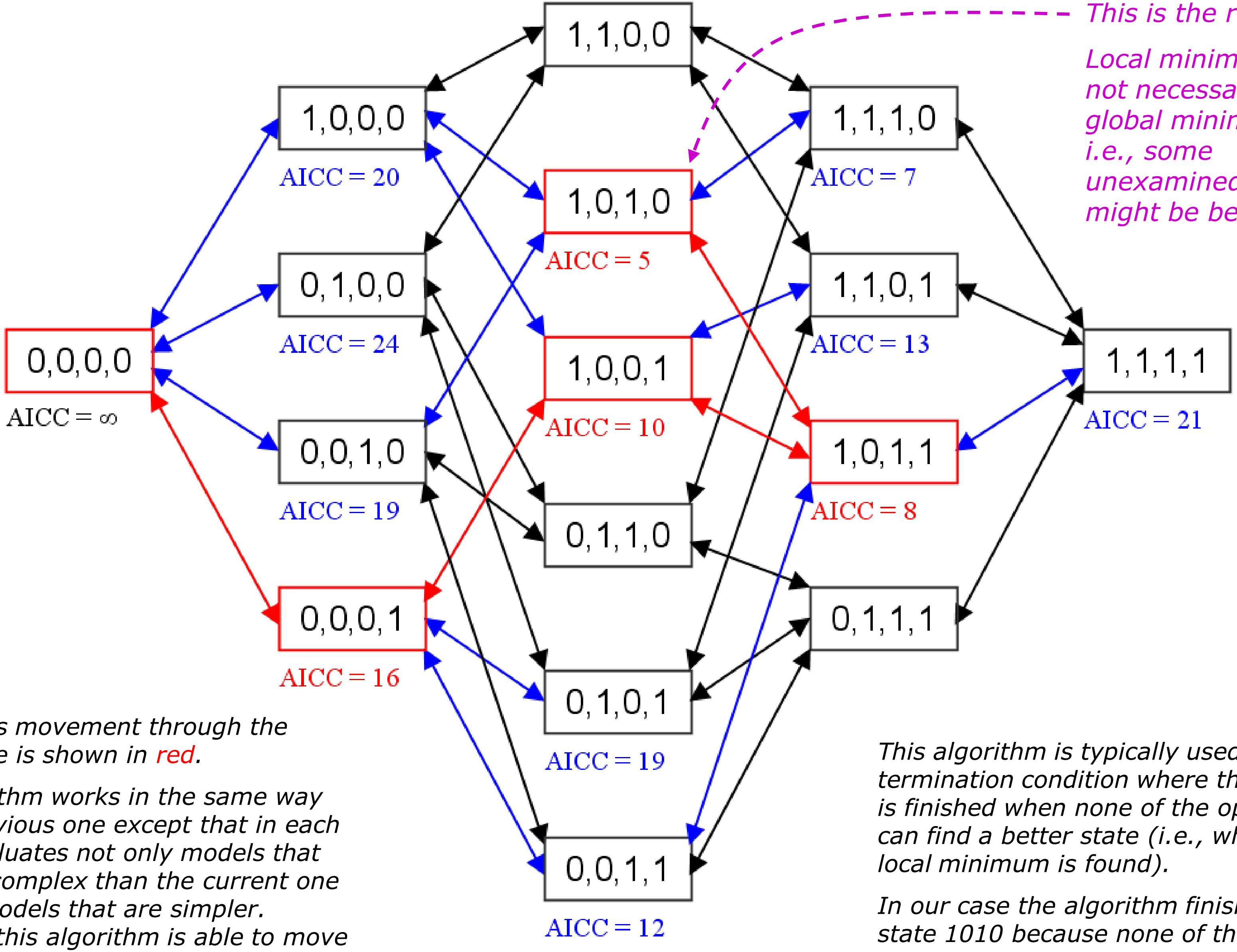
$$\hat{y} = a_0 + a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4$$

What if we use both basic operators?

- Hill climbing using both basic operators – addition as well as deletion of features
  - Now the models can also be simplified, not just made more complex



Example with both operators



This is the result.

Local minimum. But not necessarily global minimum, i.e., some unexamined state might be better.

Algorithm's movement through the state space is shown in red.

This algorithm works in the same way as the previous one except that in each step it evaluates not only models that are more complex than the current one but also models that are simpler. Therefore this algorithm is able to move also in the direction of simpler models.

This algorithm is typically used with the termination condition where the search is finished when none of the operators can find a better state (i.e., when a local minimum is found).

In our case the algorithm finishes at state 1010 because none of the states directly around it are better.

Algorithms

- There are many search/optimization algorithms developed that can be used in subset selection, for example:
  - Sequential Forward Selection
  - Sequential Backward Selection
  - Steepest Descent/Ascent Hill Climbing
  - Random Restart Hill Climbing
  - Random Mutation Hill Climbing
  - Sequential Floating Forward Selection
  - Sequential Floating Backward Selection
  - Beam Search
  - Best-first search
  - Simulated Annealing
  - Evolutionary Strategies
  - Genetic Algorithms
  - Memetic Algorithms
  - *and many others*

All these are developed for solving search/optimization problems and can be used for many different search/optimization applications, not just for feature subset selection.