Week 6.1
Feature Construction
Part 1

Feature Construction

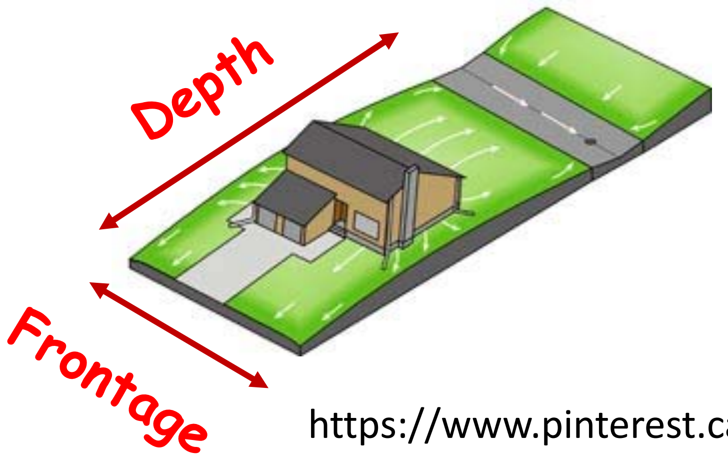
- How we can create, manipulate, rank and select relevant features from raw data?
- Construct new good features that are relevant and ready to be used in our machine learning model.
- **Example:**
  - Objective: Build a model to predict the price of a house.
    - A set of inputs have been given: the square footage of the house, the size of the lot, number of rooms, how much was sold in the past, location, and the number of concrete blocks in the driveway, etc.
    - Do we take a filed from the raw data and map to a feature in the features vector? And then use it in our ML model for training. How do we know what features to use? Or what makes even a good feature?

Feature Construction

- Feature construction examples:
  - Lots of data has timestamp associated with it. There are a number of features that can be extracted from a timestamp that might improve the model performance. What was the month? The day of the week? The hour of the day? Was it a weekend or a holiday?
  - Another example has to do with text data. Counting the number of times certain words occur in a text is one technique. This is usually combined with normalization techniques like Term Frequency -Inverse Document Frequency.

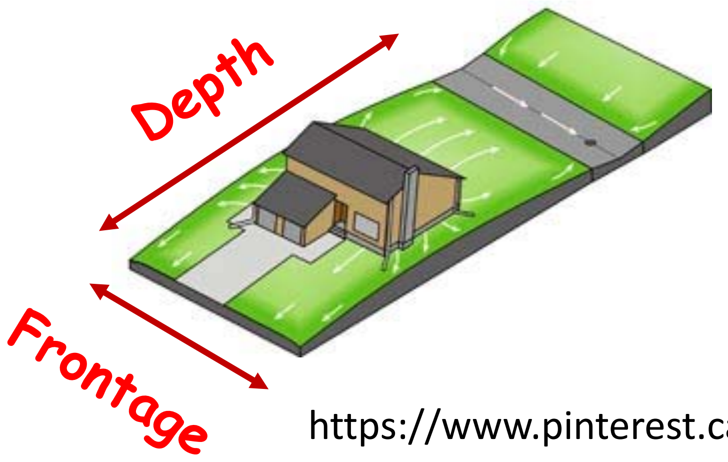
Feature Construction

- **House Price Example:**
  - Objective: Build a model to predict the price of a house.
    - Suppose that we are given two features. The frontage of the house and the depth of the house.



https://www.pinterest.ca/

Feature Construction



https://www.pinterest.ca/

1) Linear Regression Model:

- $\hat{y}_i = b_0 + b_1 \times Frontage_i + b_2 \times Depth_i$

2) Construct a new feature:

- $Area_i = Frontage_i \times Depth_i$
- Linear Regression Model: $\hat{y}_i = \theta_0 + \theta_1 \times Area_i$

interaction term.

Week 6.2
Feature Construction
Part 2

Feature Construction (ESLII- 5.1)

- Linear Regression: $\hat{y} = \sum \beta_j x_j$

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot\cdot \\ \cdot\cdot \\ x_p \end{bmatrix}$$

- We can move beyond linearity.
  - We can replace/augment the vector of input features **X** with additional features, which are transformation of **X**, and then use linear models in this new space of derived input features

Feature Construction (ESLII- 5.1)

- We can move beyond linearity.
  - $\underset{\text{Assumed}}{\underline{h_m(X)}} : \underbrace{\mathbb{R}^p \mapsto \mathbb{R}}_{\uparrow}$ the m^{th} transformation **X**, *m=1, …, M.*

    turn polynomial space to real space.

- $h_m(X)$ are sometimes called basis functions or feature functions. They define new features
  - The Feature space is typically more high-dimensional than the data space

$$\hat{y} = \sum_{m=1}^{M} \beta_m h_m(\boldsymbol{X})$$

Feature Construction

- Some simple and widely used examples of the h_m are the following:
  - $h_m(X) = \underline{x_m}$, m = 1, . . . , p recovers the original linear model. identity. the interaction point.
  - $h_m(X) = x_m^2$ or $h_m(X) = \underline{\overset{2}{x_m x_{m+1}}}$ → Allow us to augment the inputs with polynomial terms
  - $h_m(X) = \log(x_m)$, $\sqrt{x_m}$, …. → permits nonlinear transformations of single inputs

Example 1: Polynomial Expansion

- **Single input:**

  Input: x

  Transformation:

  $h_0(x) = 1, \quad h_1(x) = x, \quad h_2(x) = x^2, \dots\dots, h_d(x) = x^d$

- **Multiple inputs:**

  Input: $\mathbf{X} = [x_1, x_2, x_3]$

1st order - 3 features:	x_1	x_2	x_3			
2nd order - 6 features:	x_1^2	$x_1 x_2$	x_2^2	$x_2 x_3$	x_3^2	$x_1 x_3$
3rd order - 10 polynomial features:	x_1^3	$x_1^2 x_2$	$x_1 x_2^2$	…………………		$x_1 x_2 x_3$

……………

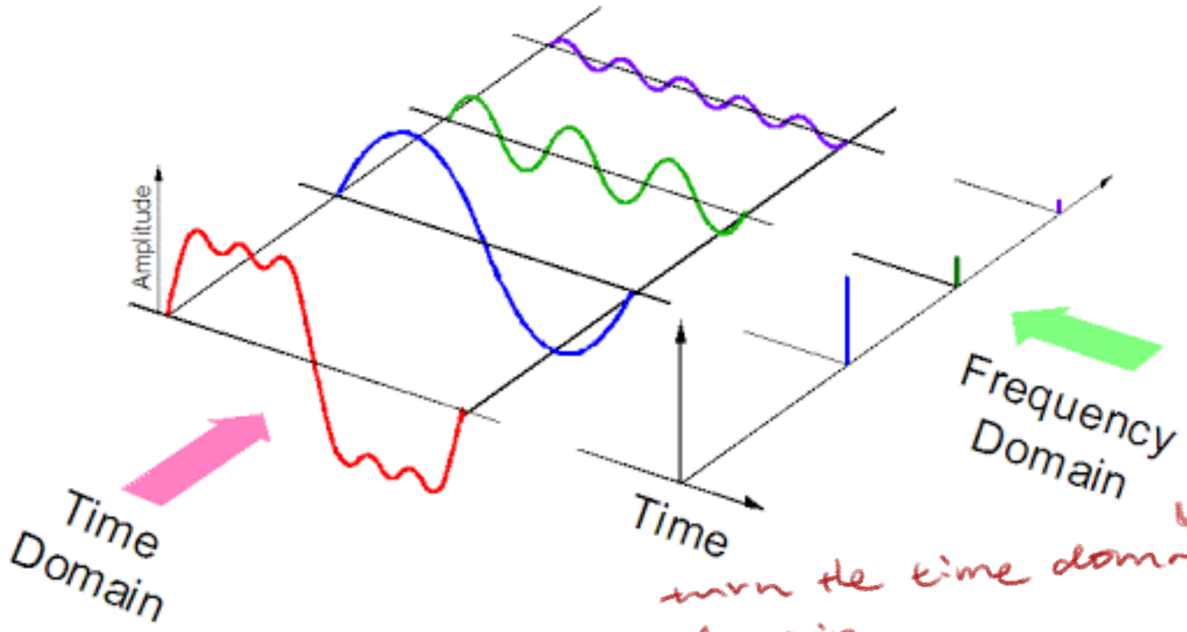
polynomial expans. every function could be [illegible] into polynomial.
2

- **Observations:**
  - Helps to capture non-linearities in regression models
  - May introduce high variance, especially near the boundaries of the data

Example 2: Fourier Series

- Fourier Basis: Reasonable option for periodic data or data with know boundaries
- Example:

$$h_0(x) = 1 \qquad h_j(x) = \cos(\omega_j x + \psi_j) \qquad \text{for } j > 0$$



turn the time domain to a freq domain

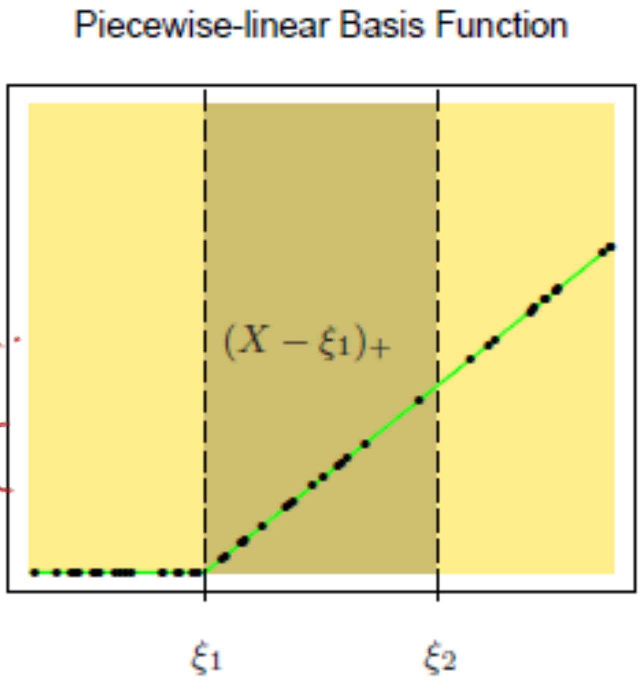
https://math.stackexchange.com/

Other Important Base Functions

Feature selection is adopted when:

- **Piecewise Polynomials** (ESLII- Fig. 5.1)

Building feature based on the data you have.
But be careful not dropping any useful data.
Do not remove the attribute unless you have a certain reason to do so.



https://towardsdatascience.com/non-linear-regression-basis-expansion-polynomials-splines-2d7adb2cc226

Week 6.3
Feature Selection
Part 1

Feature selection (JWHT 6.1)

	House	# of Bedrooms	# of Concrete Blocks	Sq. ft
House A $\longrightarrow$	A	3	20	2200
House B $\longrightarrow$	B	1	12	1350
	…	…	…	…

- Why might you want to perform feature selection?
  - Some or many of the features used in a multiple regression model are in fact not associated with the response.
  - Irrelevant features leads to unnecessary complexity in the resulting model.
    - 10 M features: each prediction is expensive

10 variables

$\Rightarrow 10 + 10 \times 9 + \cdots + 1 \times 10$ combination

$\Rightarrow 2^{10}$ variable

$$\hat{y} = \sum \beta_j x_j$$

- Which features are relevant to the prediction?

Sparsity: Housing application

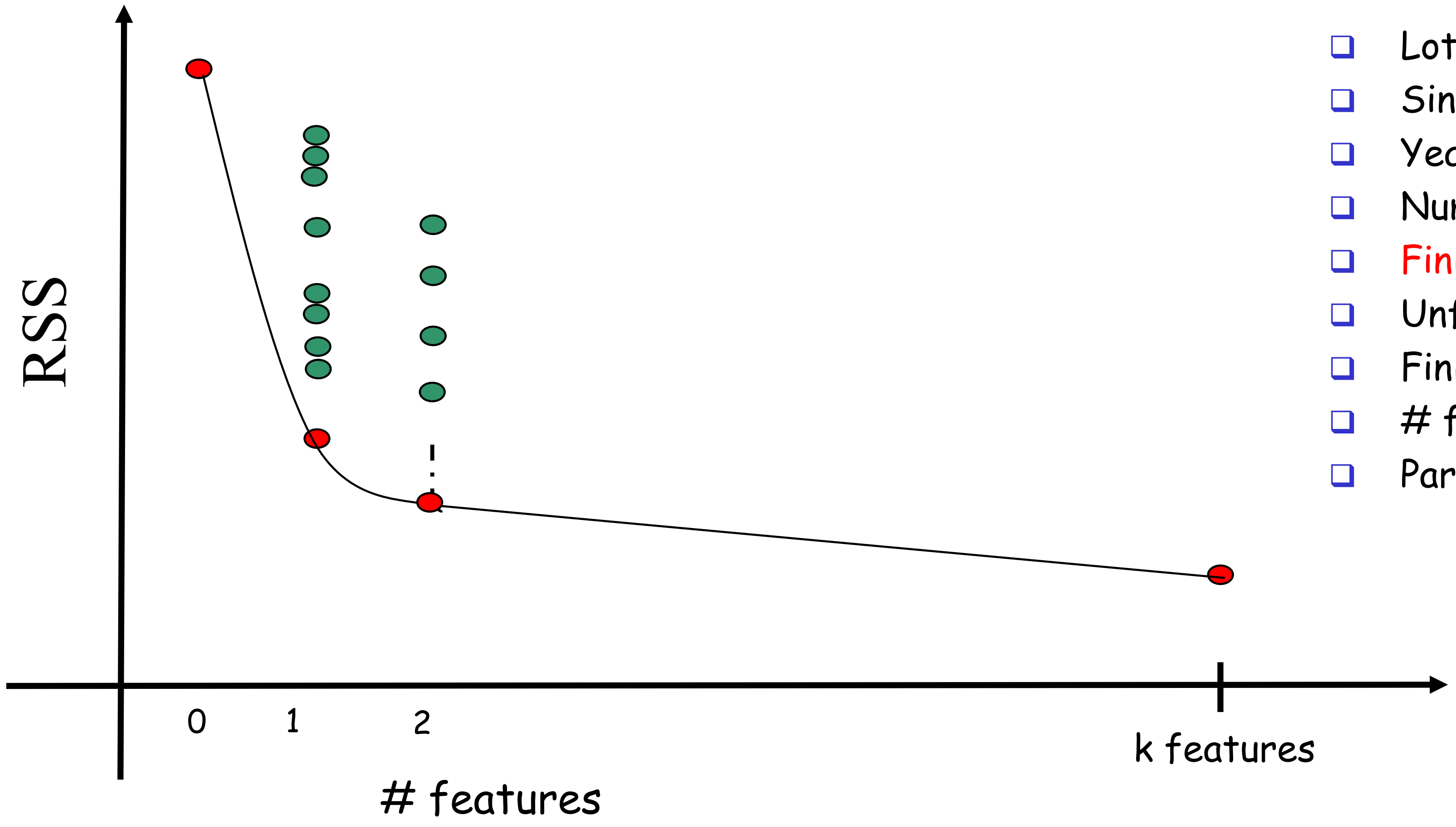
- Lot size
- Single Family
- Year built
- Last sold price
- Last sale price
- Finished sqft
- Unfinished sqft
- Finished basement sqft
- # floors
- Flooring types
- Parking type
- Parking amount
- Cooling
- Heating
- Exterior materials
- Roof type
- Structure style
- Dishwasher
- Garbage disposal
- Microwave
- Range/Oven
- Refrigerator
- Washer
- Dryer
- Laundry location
- Heating type
- Jetted Tub
- Deck
- Fenced Yard
- Lawn
- Garden
- Sprinkler System
- ……

Feature Selection

- **Subset Selection**: This approach involves identifying a subset of the *k* predictors that we believe to be related to the response.
- **Shrinkage** (also known as **regularization**): This approach involves fitting a model involving all *p* predictors. However, the estimated coefficients are shrunken towards zero relative to the least squares estimates. This shrinkage has the effect of reducing variance. automatically remove redundent
- **Dimension Reduction**: This approach involves projecting the *k* predictors into a M-dimensional subspace, where $M < k$.

All subsets ((JWHT 6.1))

- Search over every possible combination of features we might want to include in our model and look at the performance of each of those models



- Lot size
- Single Family
- Year built
- Number of rooms
- Finished sqft
- Unfinished sqft
- Finished basement sqft
- # floors
- Parking type

Find the best number of features

Best Subset Selection (Exhaustive Search over all the subsets)

- To perform best subset selection, we fit a separate least squares regression best subset for each possible combination of the *k* predictors.
  - we fit all *k* models selection that contain exactly one predictor
  - Models that contain exactly two predictors,

$$\binom{k}{2} = \frac{k!}{2!\,(k-2)!} = \frac{k(k-1)}{2}$$

  - And so forth

$$\binom{k}{n} = \frac{k!}{n!\,(k-n)!}$$

The set of all *n*-combinations of a set of size k

Best Subset Selection (Exhaustive Search over all the subsets)

- *K* features → we are 2^k possibilities of combining them
  - $1 + k + \frac{k(k-1)}{2} + \binom{k}{2} + \binom{k}{2} + \ .... + 1 \approx 2^k$

- We then look at all of the resulting models, with the goal of identifying the one that is best.

All subsets ((JWHT 6.1))

- Search over every possible combination of features we might want to include in our model and look at the performance of each of those models
- Choosing model complexity?
- Complexity of "all subsets"
  - How many models did we have to evaluate?
  - each indexed by features included (to indicate a feature is included or not)

$$\text{Complexity} \rightarrow 2^k$$

$k = 8 \rightarrow 256$ We have to search over 256 models

$k = 30 \rightarrow 1{,}073{,}741{,}824$

$k = 100 \rightarrow \; > \; 10^{30}$

It is computationally prohibitive to do this all subset search!

Option 2: Stepwise Selections

- Stepwise methods explore a far more restricted set of models: attractive alternatives to best subset selection
  - Forward Stepwise Selection:
  - Backward Stepwise Selection
  - Add one/two/three, then remove 1, like an iterating
- Both consider a much smaller set of models compared to the best subset selection

rank the selection until
1) we no longer gain
2) items run out
It's a greedy search.

However, this ignore the interaction between features.

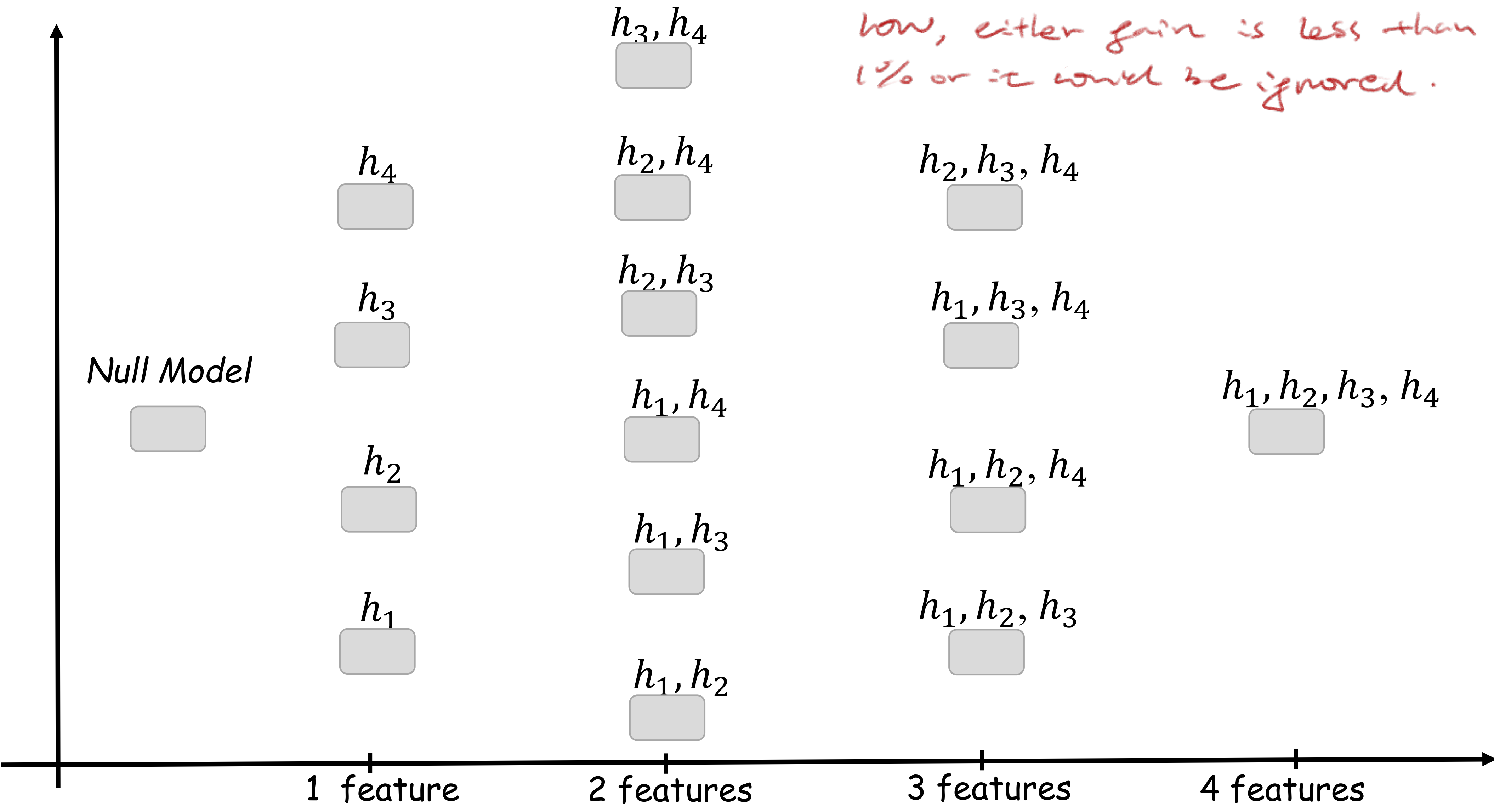
So, it is Bad!

Forward Stepwise Selection

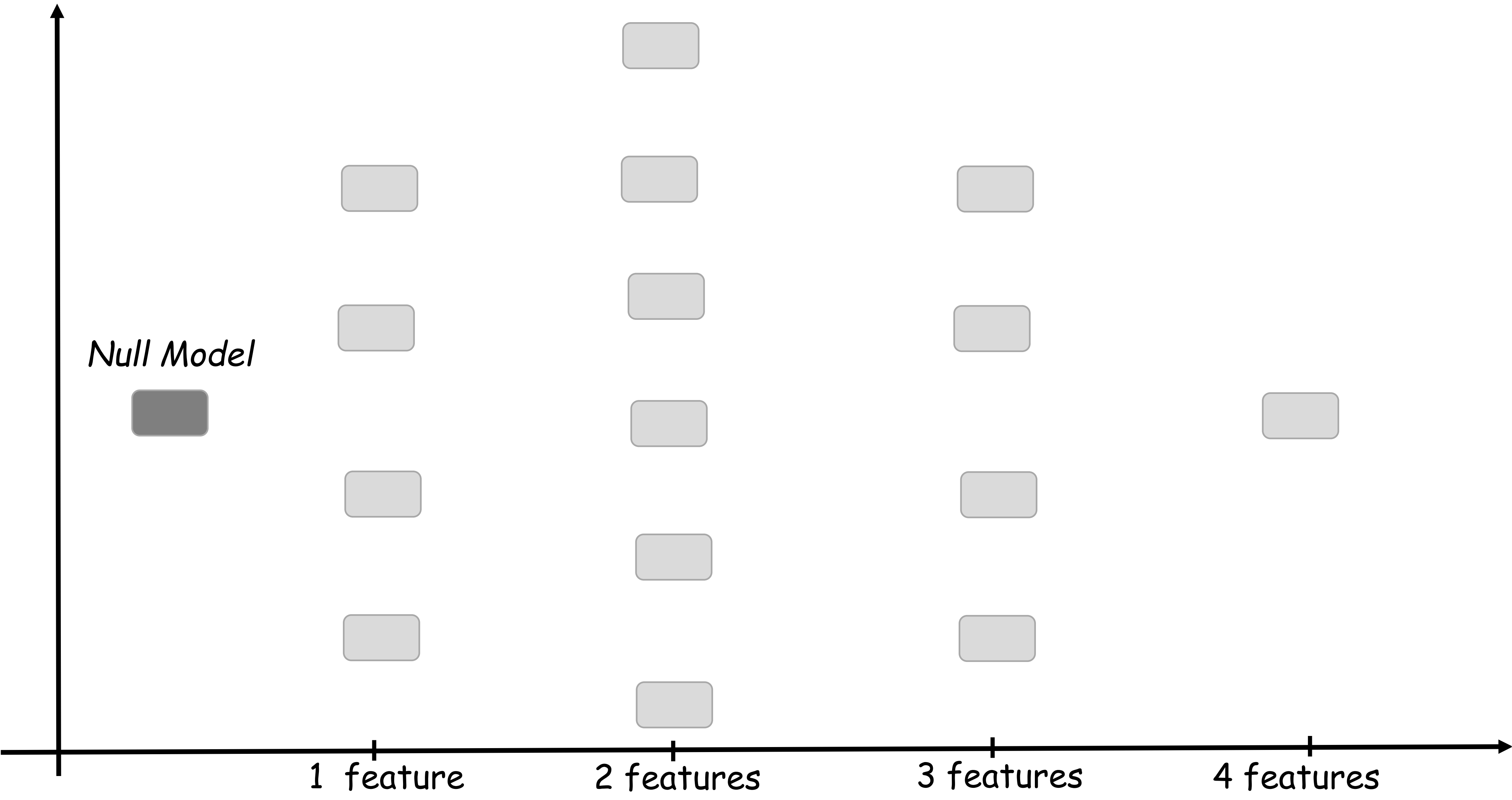
- Forward stepwise selection begins with a model containing no features, and then adds features to the model, one-at-a-time, until all of the features are in the model.
- At each step the feature that gives the greatest additional improvement to the fit is added to the model

Feature Selection

we stop until the gain is low, either gain is less than 1% or it would be ignored.

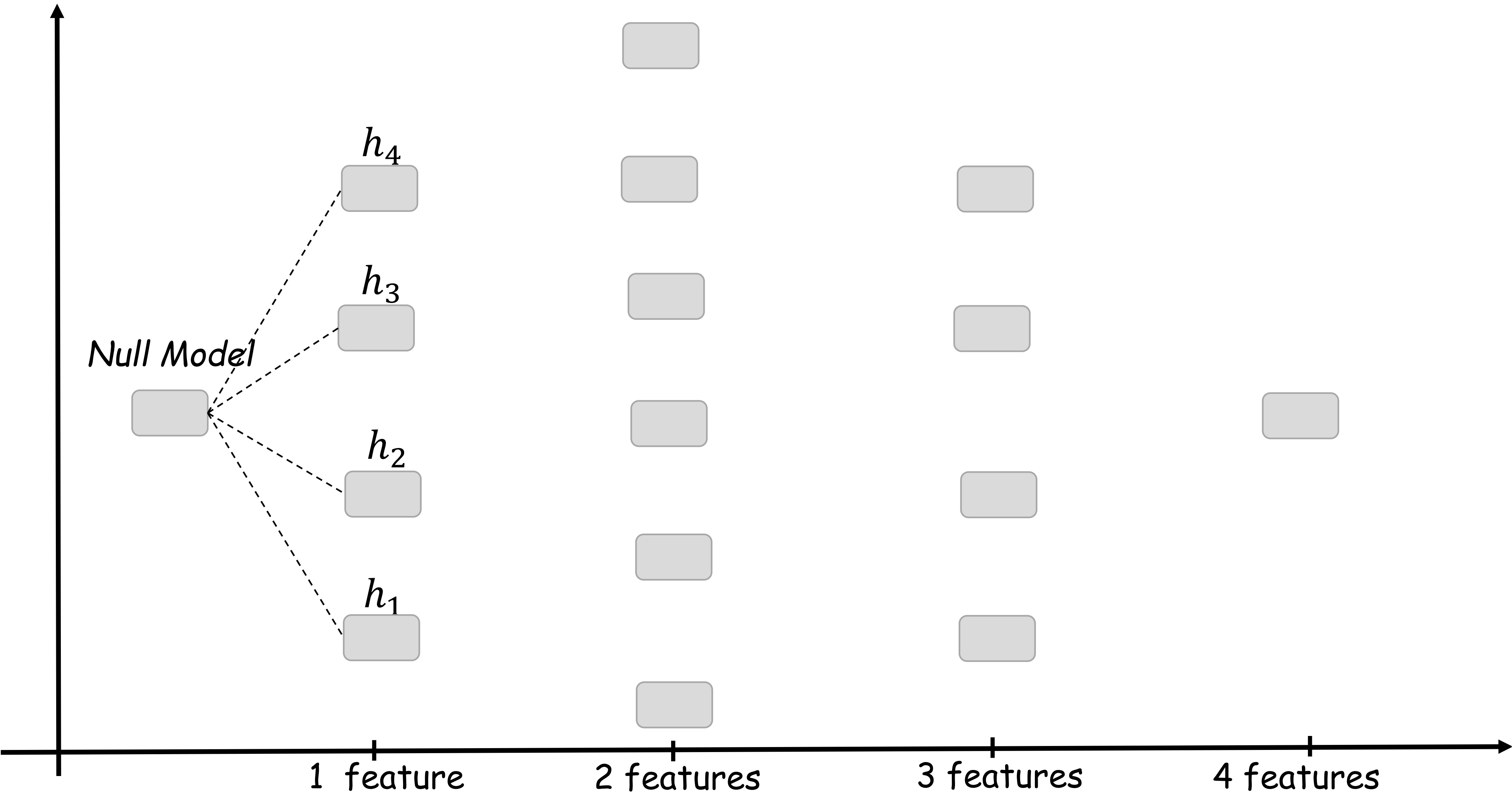


Forward stepwise selection



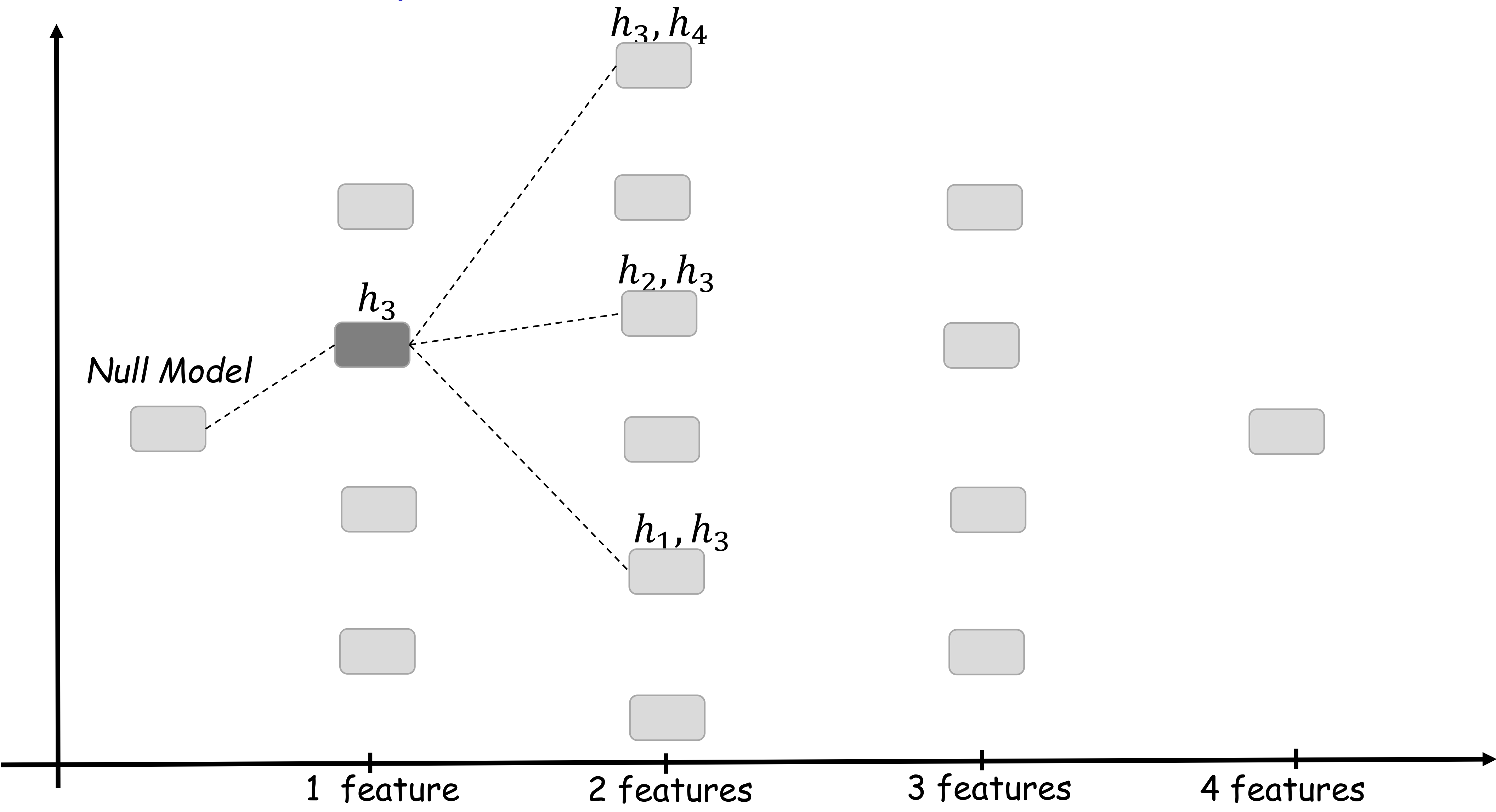
- Consider the *Null Model*, which contains no features

Forward stepwise selection



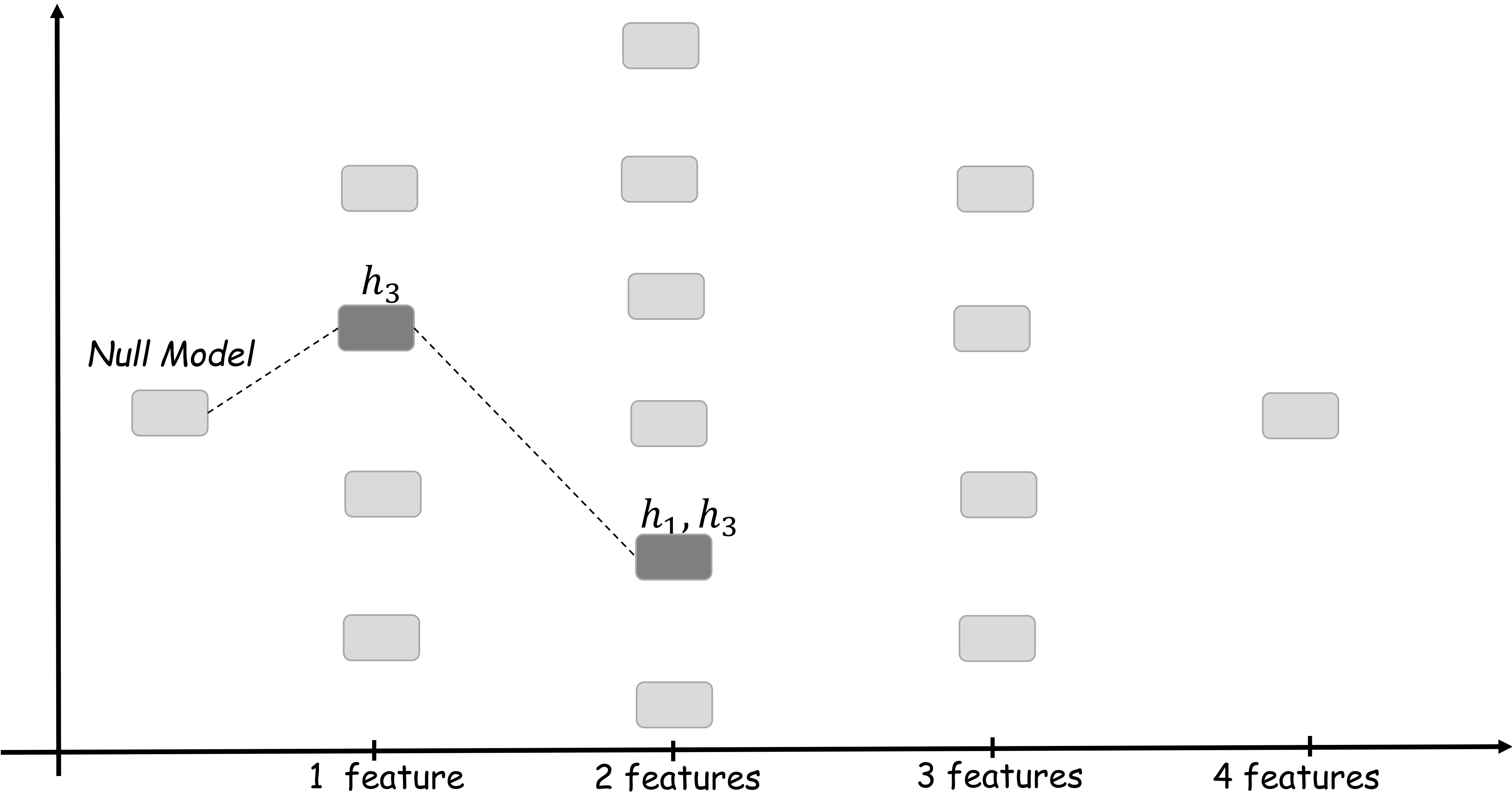
- ❑ Consider the M_0, M_1, M_2, and M_4 models - Each contains 1 feature
- ❑ Select the best model (best CV error)

Forward stepwise selection



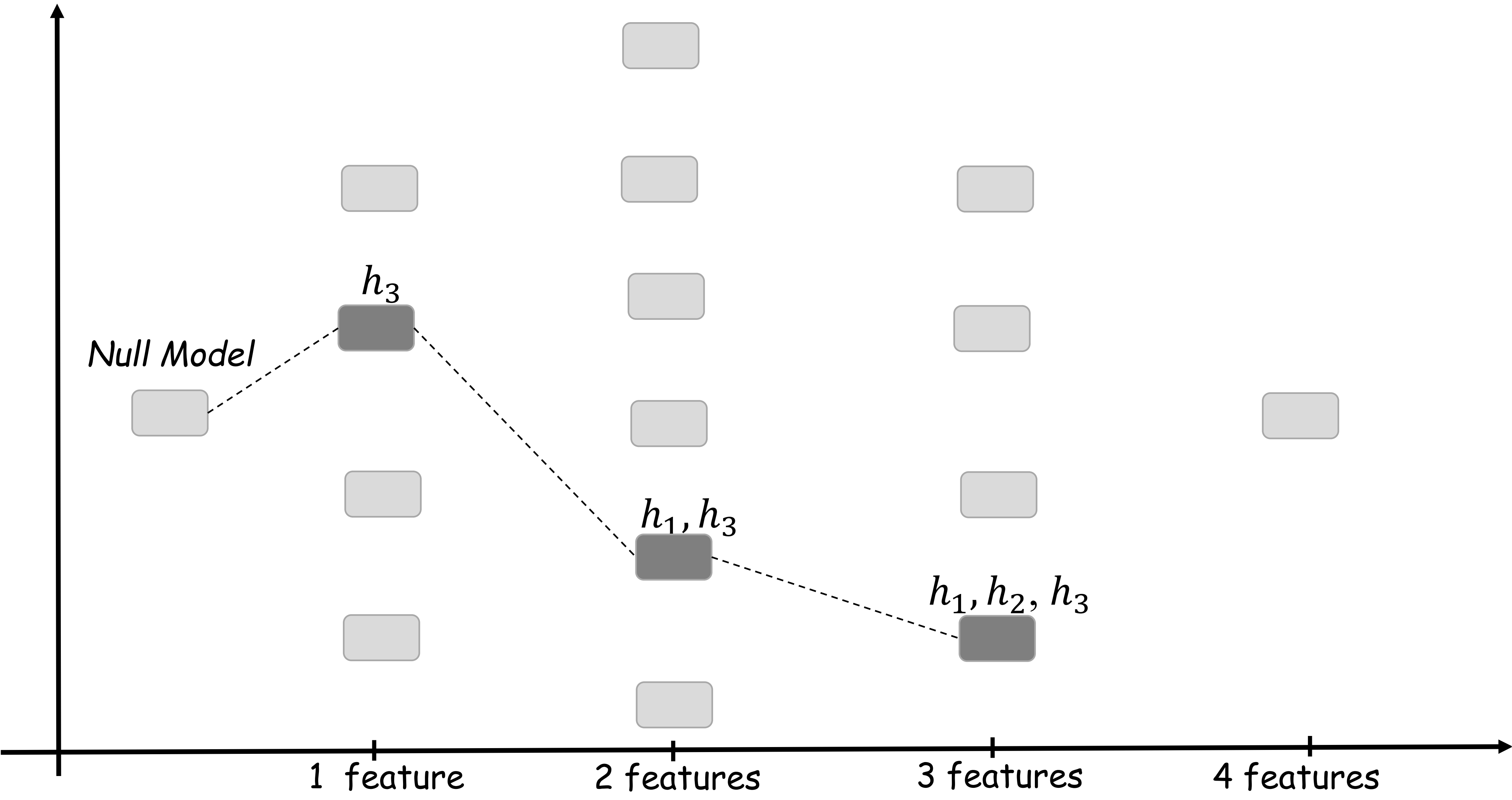
- Consider models with 2 features - These models include only the chosen feature from the previous iteration (they augment the Model selected in the previous iteration with one feature)
- Select the best model (best CV error)

Forward stepwise selection



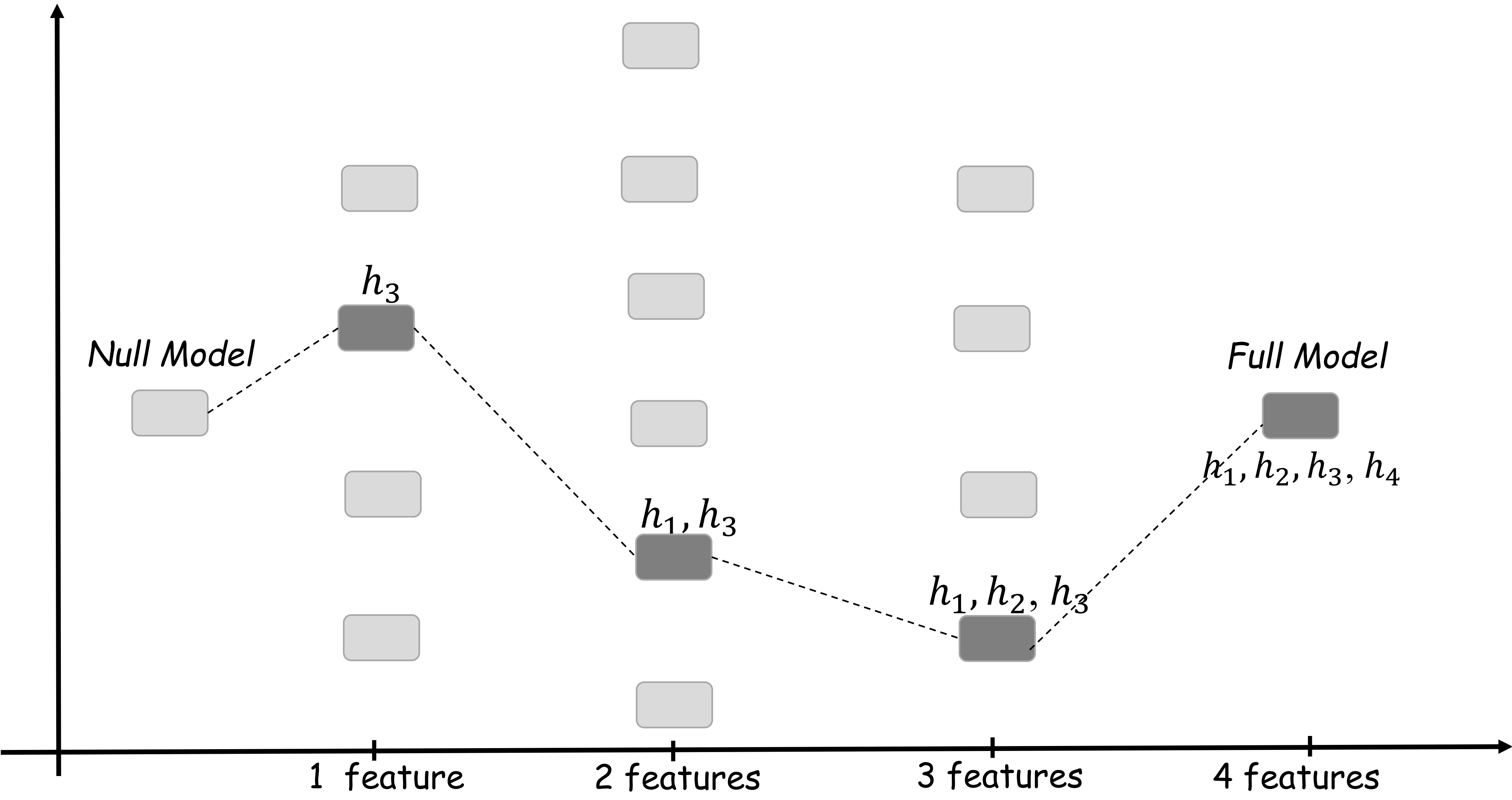
- Select the best model (best CV error)

Forward stepwise selection



- Select the best model (best CV error)

Forward stepwise selection



- Select the best model (best CV error)

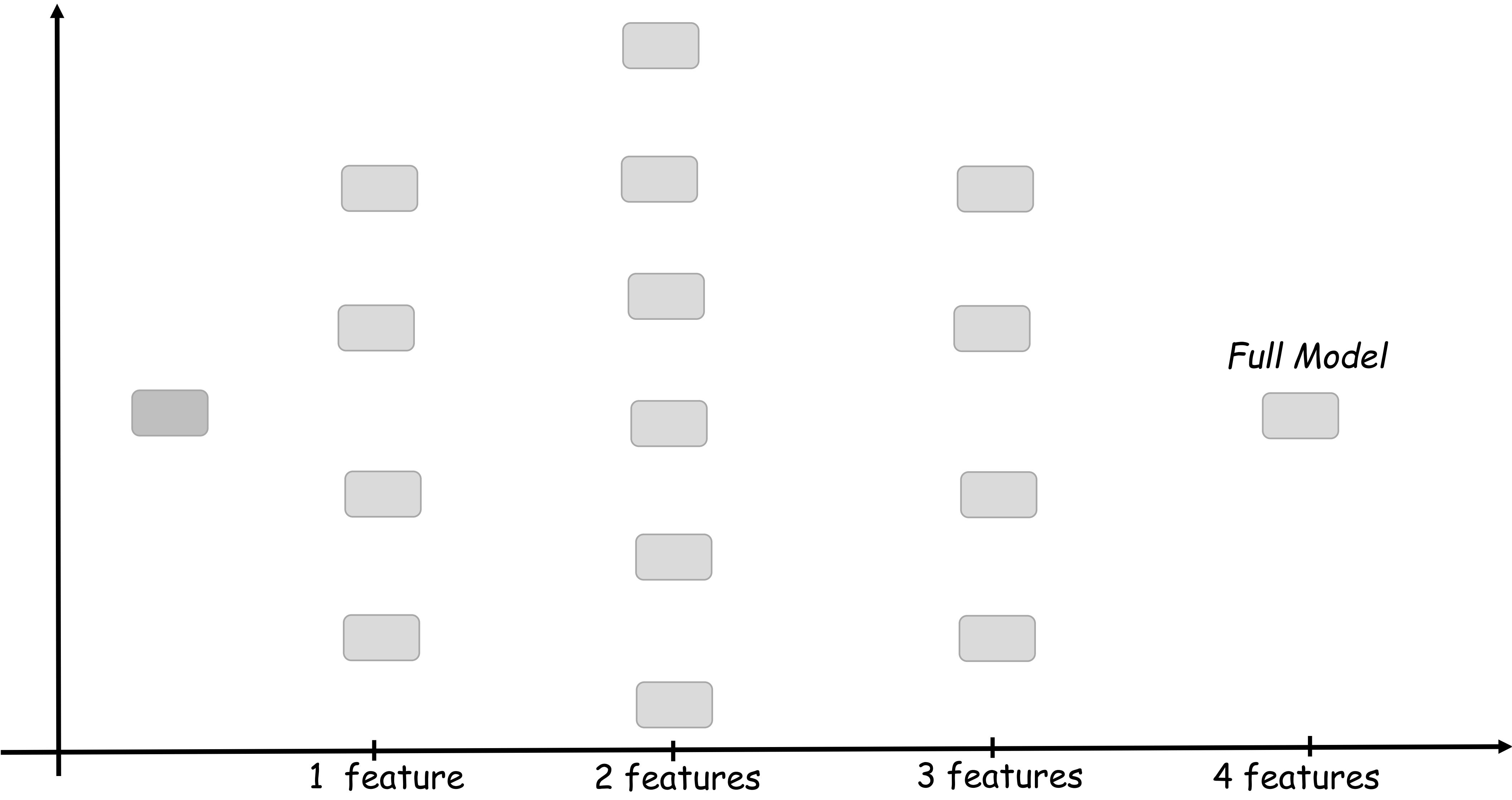
Forward stepwise selection

- Select a single best model using cross-validated prediction error
- Best subset selection algorithm: considers 2^k possibilities
- Forward stepwise selection:
  - Considers: $(k - j)$ models in the j^{th} iteration
  - Number of possibilities:

$$1 + \sum_{j=0}^{k} (k-j) = 1 + k + (k-1) + ..+1 \; = \; 1 + \frac{k(k+1)}{2}$$

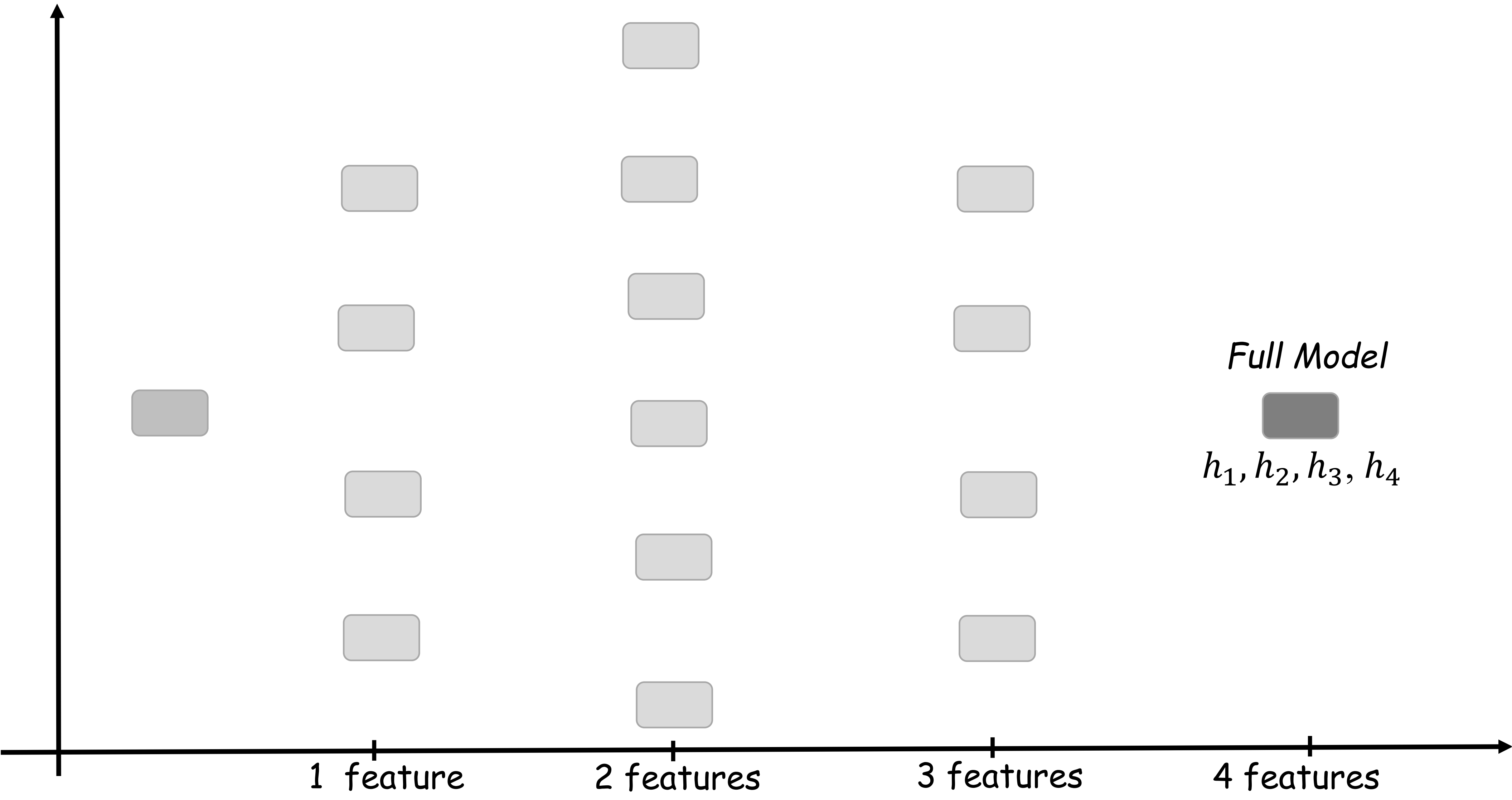
- If $k = 30$:
  - The best subset selection algorithm requires to fit/consider **1,073,741,824** models (2^{30})
  - The forward stepwise selection model requires fitting **466** models!

Backward stepwise selection



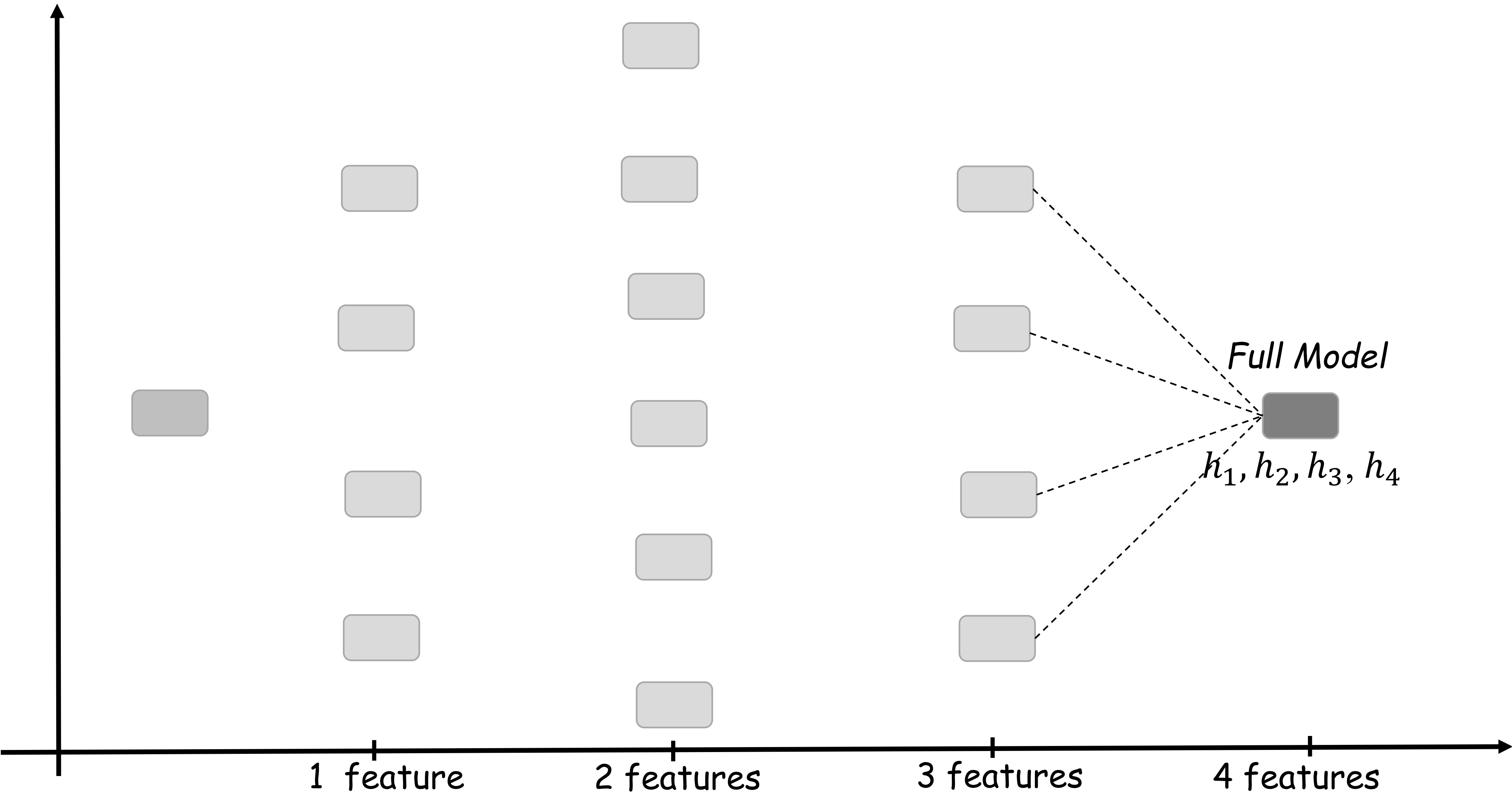
- Consider the *Full Model first*, which contains all features

Backward stepwise selection



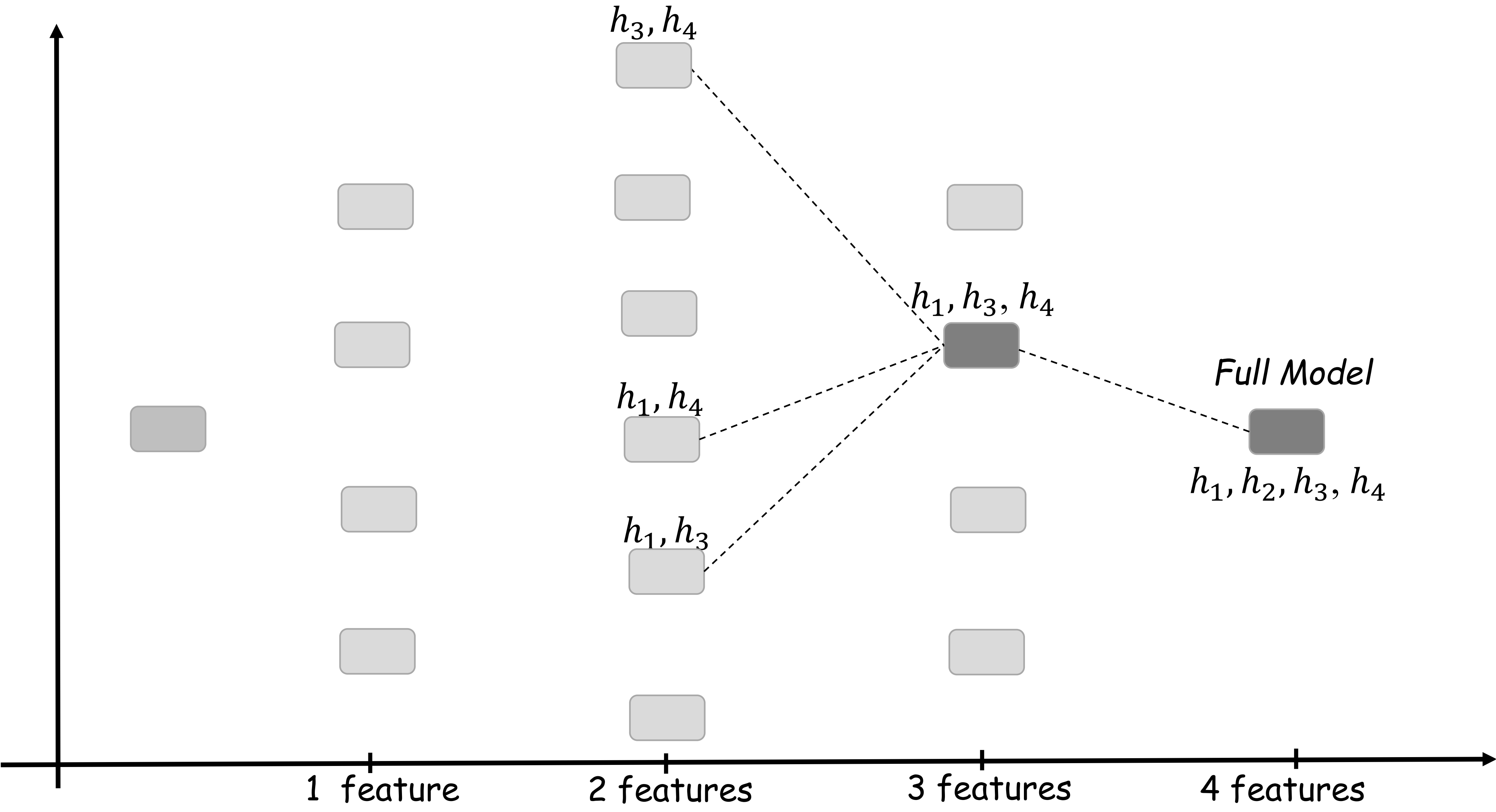
- Consider the *Full Model first*, which contains all features

Backward stepwise selection



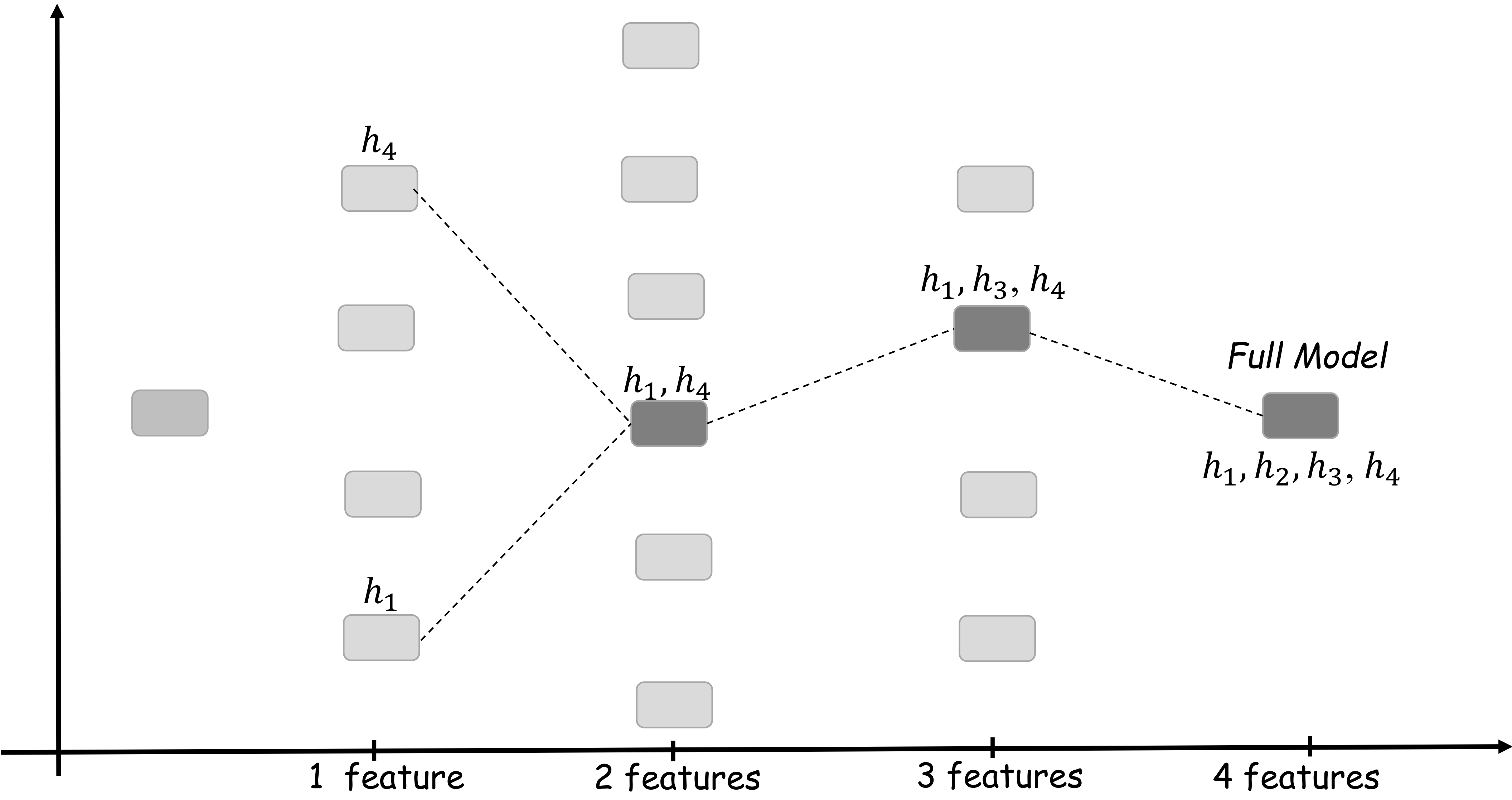
- ❑ Consider the *Full Model first*, which contains all features
- ❑ Then iteratively removes the least useful feature, one-at-a-time.
- ❑ Keep the best among the Models

Backward stepwise selection



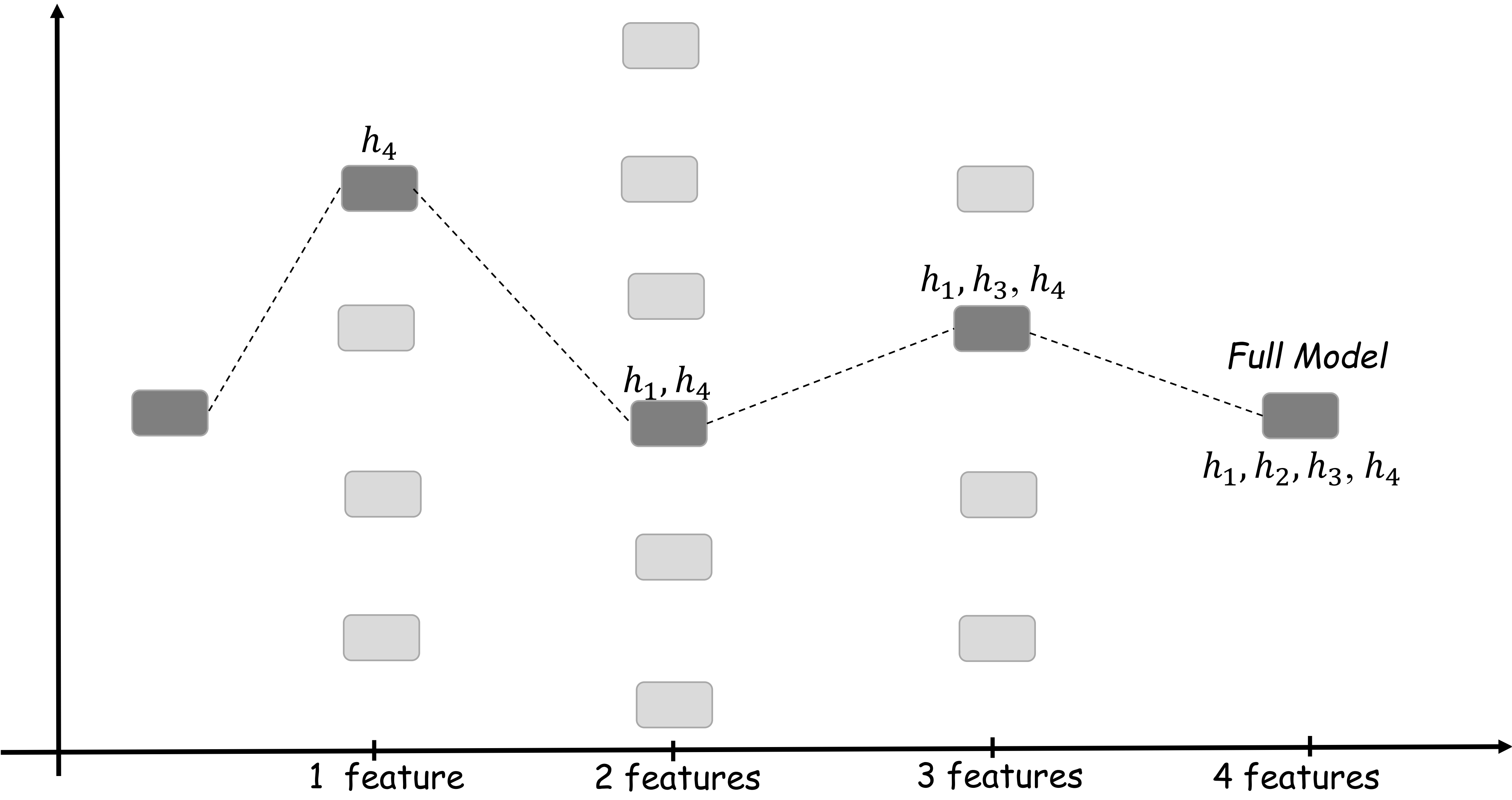
- Consider the *Full Model first*, which contains all features
- Then iteratively removes the least useful feature, one-at-a-time.
- Keep the best among the Models

Backward stepwise selection



- Consider the *Full Model first*, which contains all features
- Then iteratively removes the least useful feature, one-at-a-time.
- Keep the best among the Models

Backward stepwise selection



- ❑ Consider the *Full Model first*, which contains all features
- ❑ Then iteratively removes the least useful feature, one-at-a-time.
- ❑ Keep the best among the Models

Backward stepwise selection

- Select a single best model using cross-validated prediction error

minimizing lasso

$\downarrow$

Best shape of points .

Week 6.5
Regularization
Part 1

Regularization (JWHT 6.2)

- Feature Selections (All Subset, forward and backward selections) → set of models each contains a subset of the *k* features.
  - Use the CV error to select the best model
- Regularization: (Alternative)
  - Keep all features – Consider all features
  - Constrains (regularizes) the coefficient estimates of the features – or may shrink the coefficient estimates towards zero
  - It turns out that shrinking the coefficient estimates can significantly reduce their variance.
- Two best-known regularization techniques (shrinking the regression coefficients):
  - Ridge regression
  - Lasso regression
  - elastic net ← most usually use, good for almost all models.

Regularization (JWHT 6.2)

- Work with one Model space
- Change the optimization criterion where we balance:
  - How well the function fits data
  - Magnitude of estimated coefficients

- Total cost = **measure of fit** + **measure of magnitude of coefficients**
  - This is a new measure of the quality of the model

Ridge Regression

- **Ridge Regression: Regularization example**
  - β_j are selected to minimize

estimate from linear regression.

loss

$$J_{ridge} = \underbrace{\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{k}\beta_j x_{ij}\right)^2}_{\text{RSS}} + \underbrace{\lambda \sum_{j=1}^{k}\beta_j^2}_{\text{Regularization Term}}$$

coefficient square.

change the bias term.

- λ is the regularization coefficient
  - If λ =0: no regularization
  - If λ = ∞: $\beta_j = 0 \quad, \forall\ j$

find a balance between bias and variance.

Consider specific total cost

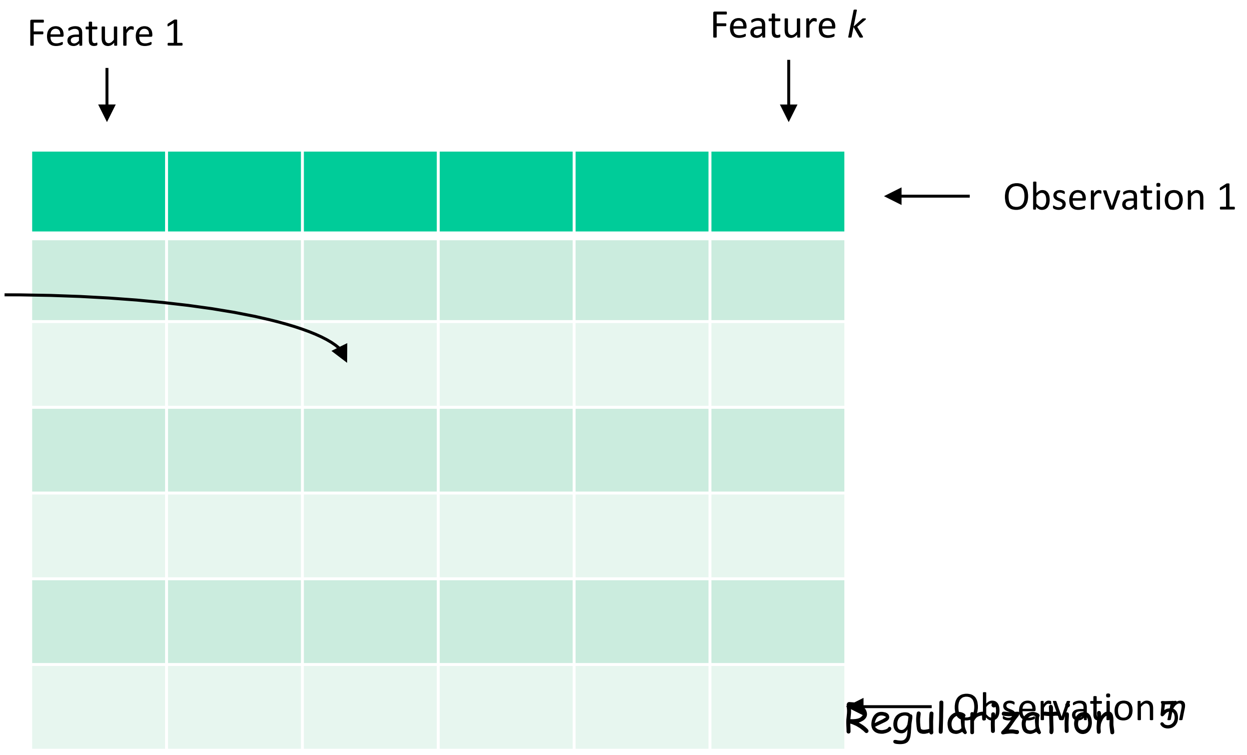
relaxation of the problem.
=> adding λ, adding constraint.

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ . \\ . \\ \beta_k \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ . \\ . \\ y_n \end{bmatrix}$$

$$J_{ridge} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{k} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{k} \beta_j^2$$

x_{ij} : value of feature *j* in observation *i*

$\mathbf{X} =$



Ridge Regression (JWHT 6.2)

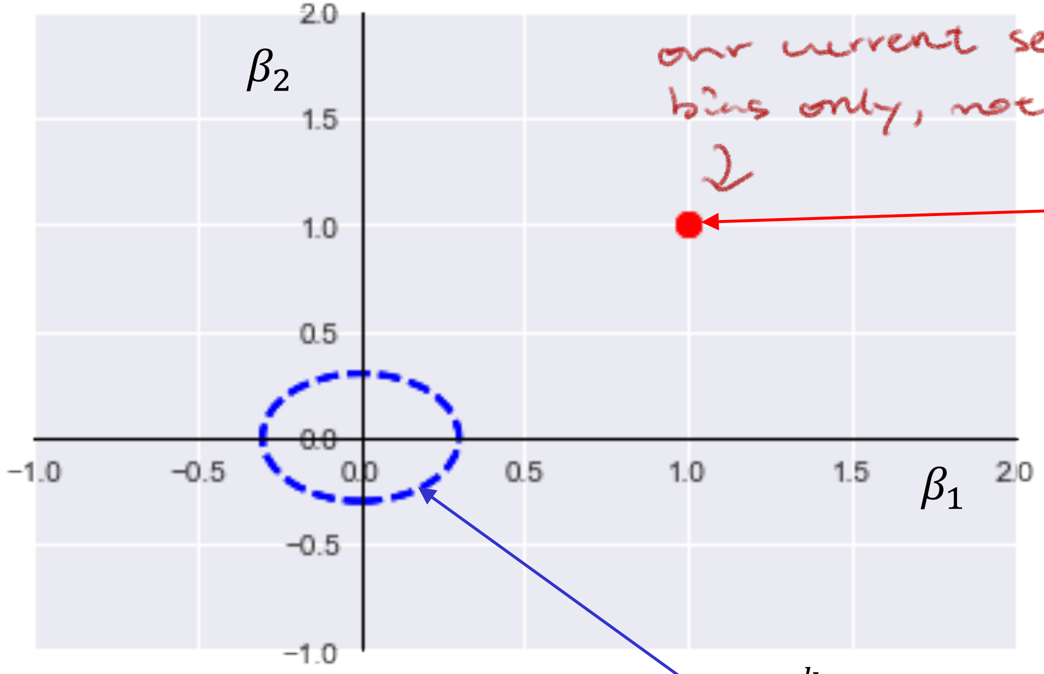
- Another Formulation for Ridge Regression

$$\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{k} \beta_j x_{ij} \right)^2 \right\}$$

$$\text{subject to}: \sum_{j=1}^{k} \beta_j^2 \le s$$

- For every value of λ there is some s such the equation above will give the ridge regression coefficient estimates

Ridge Regression: Contour of the constraint function



our current selection.
bias only, not consider varience.

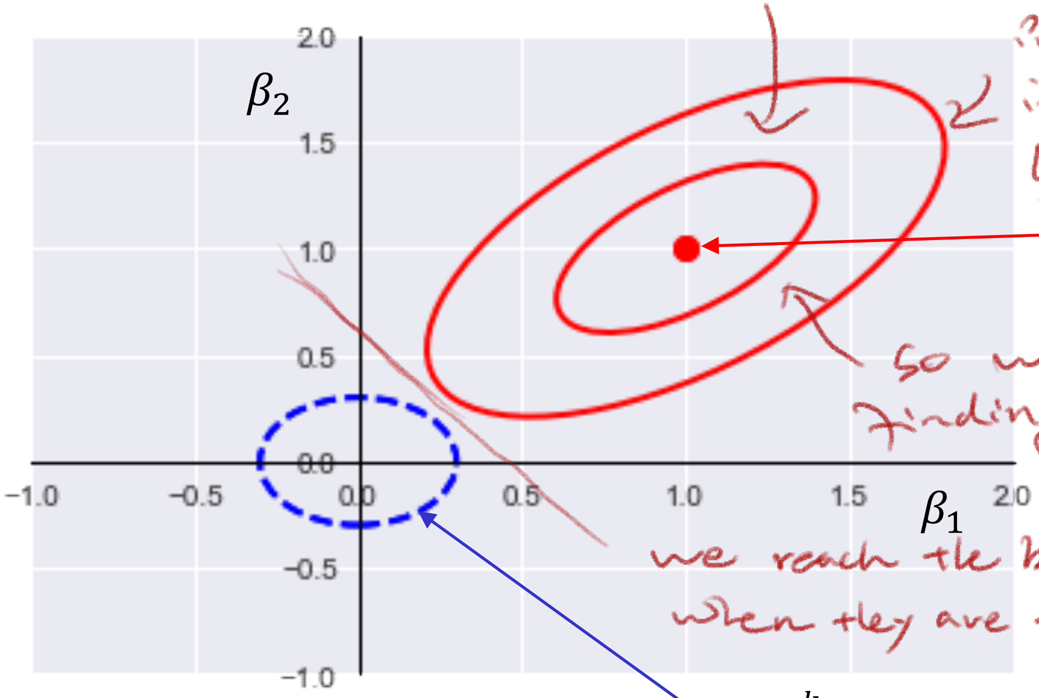
$\hat{\beta}$ The vector of least squares coefficient estimates

$$\sum_{j=1}^{k} \beta_j^2 \leq s$$

Ridge Regression: Contours of the error and constraint function

modifying λ.
the circle getting larger.

λ multiplies variance.



increase in λ, more important variance is, so less important bias. less importance of error.

$\hat{\beta}$ The vector of least square coefficient estimates

so we're finding optimal size of it that has smallest coefficient with least error.

we reach the best case when they are tangent.

$$\sum_{j=1}^{k} \beta_j^2 \le s$$

$$\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{k} \beta_j x_{ij} \right)^2 \right\}$$

$$\text{subject to}: \sum_{j=1}^{k} \beta_j^2 \le s$$

Regularization: Scaling

- Un-regularized Regression:
  - Multiplying a feature by a constant c simply leads to a scaling of the least square coefficient estimates by a factor of 1/c.
  - Regardless of what jth feature is called, $\sum_{i=1}^{n} x_{ij}\beta_j$ will remain the same
  - Scale equivariant
- Regularization:
  - Ridge regression coefficient estimates can change substantially when multiplying a given feature by a constant
  - $J_{ridge} = \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{k}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{k}\beta_j^2$

Regularization

- Since scaling changes the result, one strategy is to z-standardize features before submitting them to the model

$$x = \frac{x - mean(x)}{std(x)}$$

- Another possibility is to have different regularization coefficients (λ) for different features (sometimes discussed as Tikhonov regularization)

Week 6.6

Regularization:

Lasso and the hybrid Elastic Net

Part 3

L1 Regularization: Lasso

effective in variable selection and regularization
not robust against correlation
so a transform/selection is needed, harder to optimize

- Lasso penalizes with the L1-norm:

$$J_{Lasso} = \underbrace{\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{k}\beta_j x_{ij}\right)^2}_{\text{RSS}} + \underbrace{\lambda\sum_{j=1}^{k}|\beta_j|}_{\text{Regularization Term } (L_1 \text{ penalty})}$$

- Ridge Regression will always generate a model involving all features → increasing the value of λ will tend to reduce the magnitudes of the coefficients, but will not result in exclusion of any of the variables
- Lasso: the L1 penalty has the effect of forcing some of the coefficients estimates to be exactly equal to zero when the tuning parameter is sufficiently large.

Lasso (JWHT 6.2)

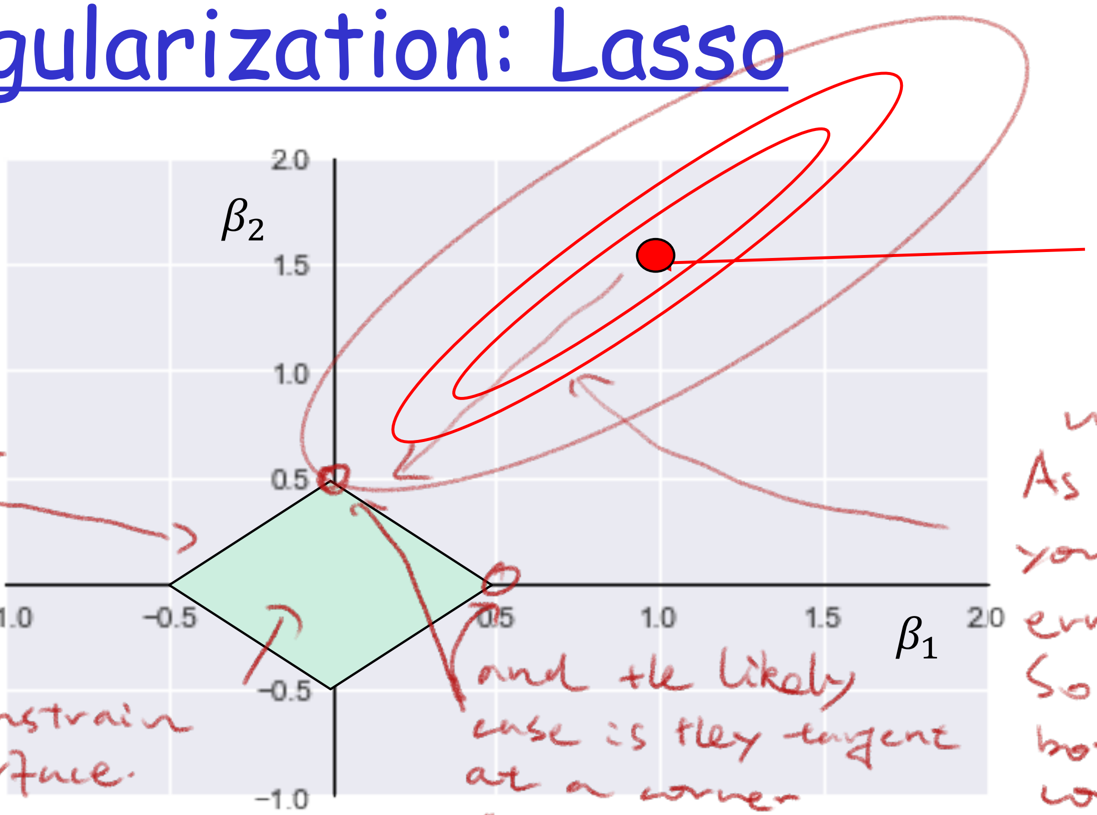
- Another Formulation for Lasso

$$\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{k} \beta_j x_{ij} \right)^2 \right\}$$

$$\text{subject to} : \sum_{j=1}^{k} \left| \beta_j \right| \le s$$

- For every value of λ there is some s such the equation above will give the ridge regression coefficient estimates

L1 Regularization: Lasso



The vector of least square coefficient estimates

optimal selection .

it is also the answer without constrain.

As you are constraining, you are getting higher error

So the best case to both minimal error and constrain is the tangent situation .

in ridge we have a circle.

constrain serface.

and the likely case is they tangent at a corner

And at this case, some variable would be serial.

- The lasso coefficients estimates are given by the first point an ellipse contacts the constraint region
- The green diamond represents the lasso constraints.
- The lasso constraint has corners at each of the axes → the ellipse will often intersect the constraint region at the an axis → one of the coefficient will be zero

Lasso also works well when the number of variables (b) is greater than the number of samples (m), it is consistent when you get more variable (Overspecifize model)

Lasso vs. Ridge Regression

for all regression, regularize first.

- Often neither one is overall better.
- Lasso can set some coefficients to zero → performing feature selection, while ridge regression cannot.
- Both methods: as λ increases, the variance decreases and the bias increases.
- Lasso tends to do well if there are a small number of significant parameters and the others are close to zero.
- Ridge works well if there are many large parameters of about the same value (ergo: when most predictors impact the response).
- lasso performs variable selection, and hence results in models that are easier to interpret

Lasso is good for variable selection
But bad for correlation.

- Note: in general we regularize: $||\boldsymbol{\beta}||^q$

Elastic Net

: combine both penalities into one model.

- Combine the penalties of ridge regression and lasso to get the best of both worlds.

tendency in model. Lasso Ridge.

$$J_{Elastic\ Net} = \underbrace{\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{k}\beta_j x_{ij}\right)^2}_{\text{Bias.}} + \lambda\left(\alpha\sum_{j=1}^{k}|\beta_j| + (1-\alpha)\sum_{j=1}^{k}\beta_j^2\right)$$

importance of variance over bias, like before.

- α is the mixing parameter between ridge (α = 0) and lasso (α = 1).
- Two parameters to tune

make sure the model runs better.

when looking for values: lower Lasso. (serial, reduce the coefficient for useless variables).

the ridge part here is to dealing with the correlation (not-Linear function) to improve Lasso's performance.

Week 6. 7
Regularization
Part 3

Ho do we select λ? cross validation!

- Implementing ridge regression and the lasso requires a method for selecting a value for the tuning parameter λ
- Cross-validation: A simple method to tackle this problem. We choose a grid of λ values, and compute the cross-validation error for each value of λ.
  - The tuning parameter with the smallest cross-validation error is chosen.

Ridge Regression vs. Least Squares (JWHT 6.2)

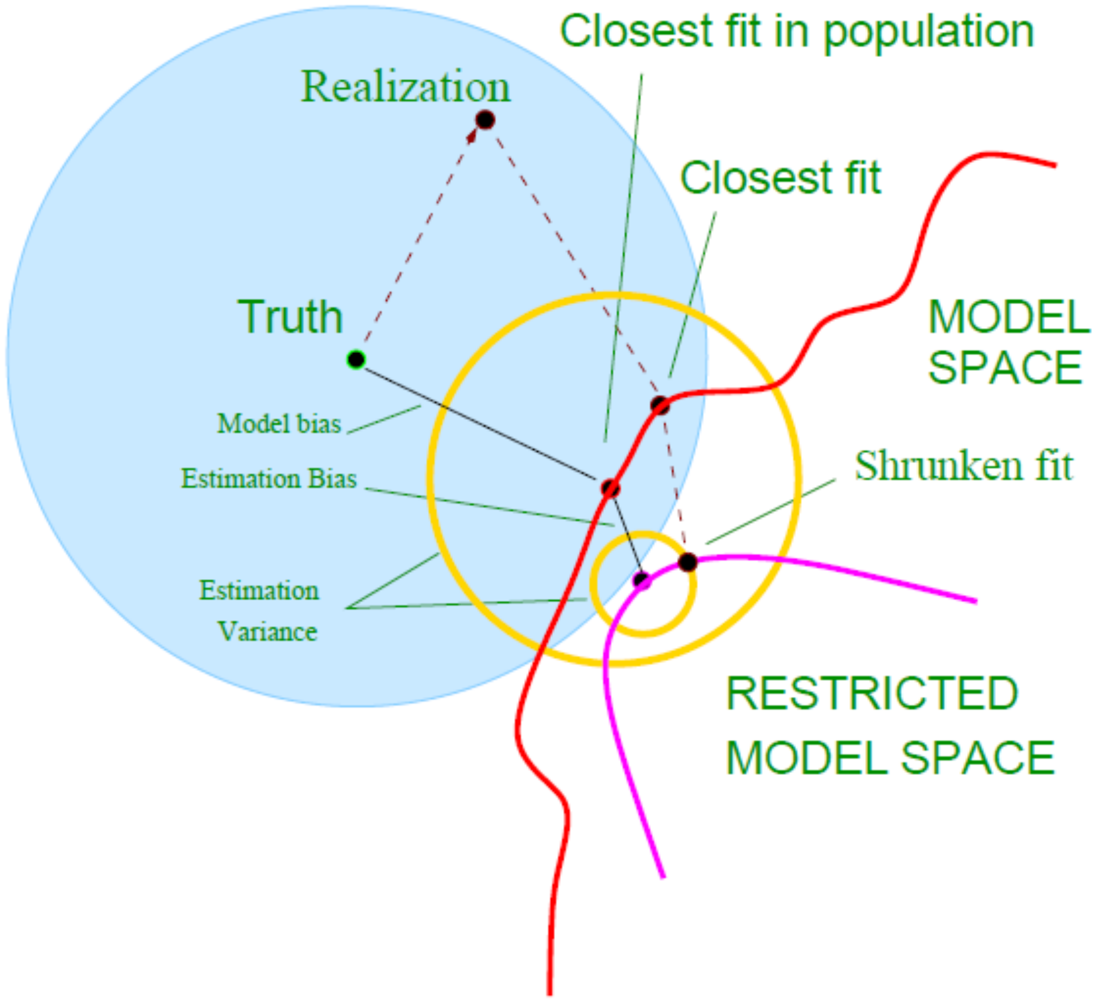
- Ridge regression's advantage: the *bias-variance trade-off*.
- λ increases → the flexibility of the ridge regression fit decreases, leading to *decreased variance* but *increased bias*
- In general, if the relationship between the response and the features is close to linear → the least squares estimates will have low bias but may have high variance
- Ridge regression works best in situations where the least squares estimates have high variance
- Ridge regression also has substantial computational advantages over best subset selection, which requires searching through 2^k models.
- Ridge regression (for any fixed value of λ) only fits a single model, and the model-fitting procedure can be performed quite quickly.

- *For more details please the Section 6.2 from JWHT*

Regularization: Summary

- If a linear model contains large number of features or if these features are correlated → the standard OLS method may introduce large variance → the model unreliable.
- To counter this, regularization techniques can be used:
  - A technique allowing to decrease this variance at the cost of introducing some bias. Finding a good bias-variance trade-off allows to minimize the model's total error.
- There are three popular regularization techniques, each of them aiming at decreasing the size of the coefficients:
  - Ridge Regression, which penalizes sum of squared coefficients (L2 penalty).
  - Lasso Regression, which penalizes the sum of absolute values of the coefficients (L1 penalty).
  - Elastic Net, a convex combination of Ridge and Lasso.
- The size of the respective penalty terms can be tuned via cross-validation to find the model's best fit.

Regularization: Summary (HTF p. 225)



The Impact of Regularization

The following notebook will highlight the impact of using Ridge Regularization. The required imports are shown in the following cell. The random seed has been set to 42 for reproducibility. Some stylistic settings have been activated to improve the presentation of the graphs.

In [18]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
sns.set_context('paper')
plt.style.use('seaborn')
%matplotlib inline
np.random.seed(42)
```

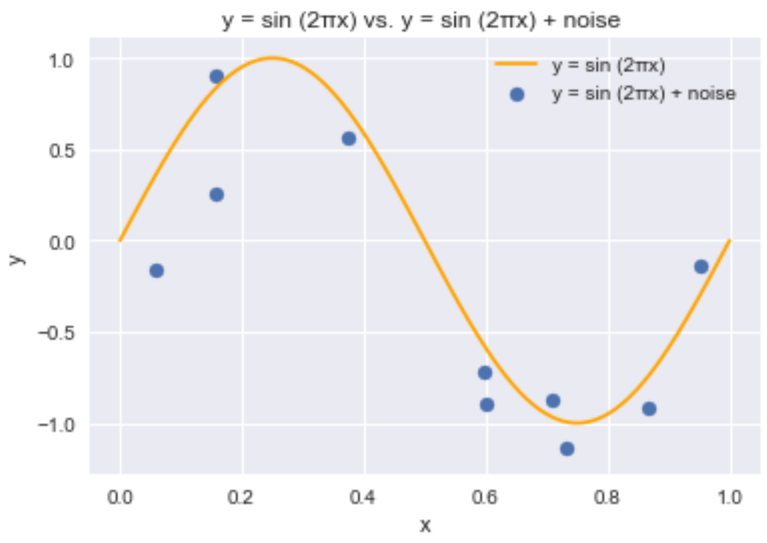
The first step is to generate two sets of data. The premise of this exercise is to mimic a real-world situation where the relationship between input and output is affected by noise. To do this, we will add Gaussian noise to a sample of 10 data points from the function y = sin(2πx). To visualize the effect of the noise on the data points, we also generate 100 data points for the continuous function y = sin(2πx)

In [19]:
```
#Generate datapoints for y = sin (2πx) + noise
x = np.random.random_sample(10)
y_orig = np.sin(2*math.pi*x)
noise = np.random.normal(0,0.3,10)
y_noise = np.sin(2*math.pi*x) + noise

#Generate Curve for y = sin (2πx)
x2 = np.linspace(0,1,100)
y2 = np.sin(2*math.pi*x2)
```

Below is the visualization of the original function (orange) along with the noisy datapoints (blue). A low number of data points was selected to clearly demonstrate how fitting regression models with low amounts of data leads to the manifestation of over and under fitting

In [20]:
```
plt.plot(x2,y2, label = 'y = sin (2πx)', color='orange')
plt.scatter(x, y_noise, label = 'y = sin (2πx) + noise ')
plt.legend()
plt.xlabel("x")
plt.ylabel('y')
plt.title('y = sin (2πx) vs. y = sin (2πx) + noise ')
```

Out[20]:
```
Text(0.5, 1.0, 'y = sin (2πx) vs. y = sin (2πx) + noise ')
```

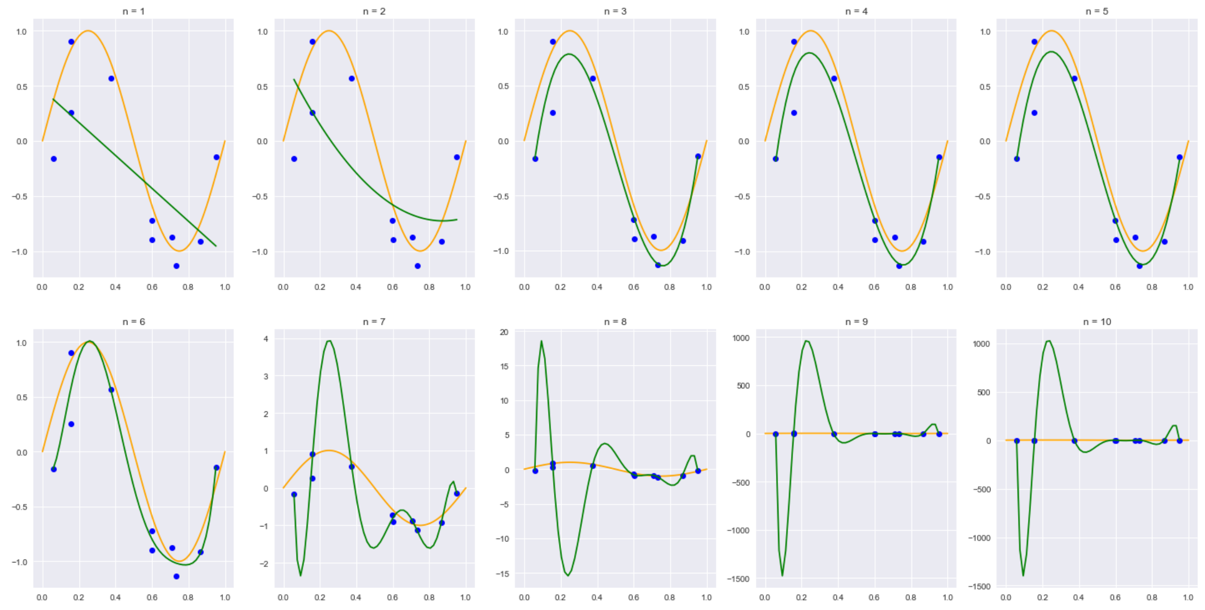


For our first model, we will be using linear regression with polynomial features, effectively making a polynomial regression model. The first stage in creating this model is to create the polynomial features. As each data point currently contains an x and y value, the 9 more features are generated by raising the x value to the power of 2, 3, ... 10.

In [21]:
```
#generate polynomial features up to degree 10
data = pd.DataFrame(x, columns = ['x'])   ## These data points will be using t
o train the model
for i in range(2,11):
    colname = 'x_%d'%i
    data[colname] = data['x']**i

Test_Data = np.linspace(np.sort(x)[0], np.sort(x)[-1], num =50)   # 50 data po
ints that will be used to demonstrate the relationship built by the model (ext
reme case)
Test_Data = pd.DataFrame(Test_Data, columns = ['x'])
for i in range(2,11):
    colname = 'x_%d'%i
    Test_Data[colname] = Test_Data['x']**i
```

The following code creates 10 subplots, one for each of the polynomial degrees. For each degree, only the associated polynomial terms are used to build the linear regression model (ex. a polynomial degree of 4 uses 4 terms: x1, x2, x3, x4). As seen in these figures, the first two degrees show underfitting whereas degrees 7+ show clear signs of overfitting.

In [22]:
```
coefs = []  ## Python list
rss = []
fig, axs = plt.subplots(2,5, figsize = (25,12.5))
for i in range(0,2):
    for j in range (0,5):
        LeastSquaresModel = LinearRegression(normalize=True)
        LeastSquaresModel.fit(data.iloc[:,0:5*i+j+1], y_noise)
        Test_Data_pred_curve = LeastSquaresModel.predict(Test_Data.iloc[:,0:5*
i+j+1]) ## we applied the model to the 50 data  points
        y_pred_points = LeastSquaresModel.predict(data.iloc[:,0:5*i+j+1])  ##
 prediciting of the training data points
        rss.append(np.sum(np.square(y_noise - y_pred_points))) ## Training RSS
        coefs.append(LeastSquaresModel.coef_)  ## Model Coefficients
        axs[i,j].plot(x2,y2, label = 'y = sin (2πx)', color='orange') ## the s
in function
        axs[i,j].plot(x, y_noise,"o", color='b') # the 10 training points (wit
h noise)
        axs[i,j].plot(Test_Data.x, Test_Data_pred_curve, color='g')  ## visuli
zation of the model
        axs[i,j].title.set_text("n = " + str(5*i+j+1))
```



As seen above, as the model complexity increases, so too does the level of overfitting. To further illustrate a consequence of the higher-order polynomials, we will explore the coefficients of each model.

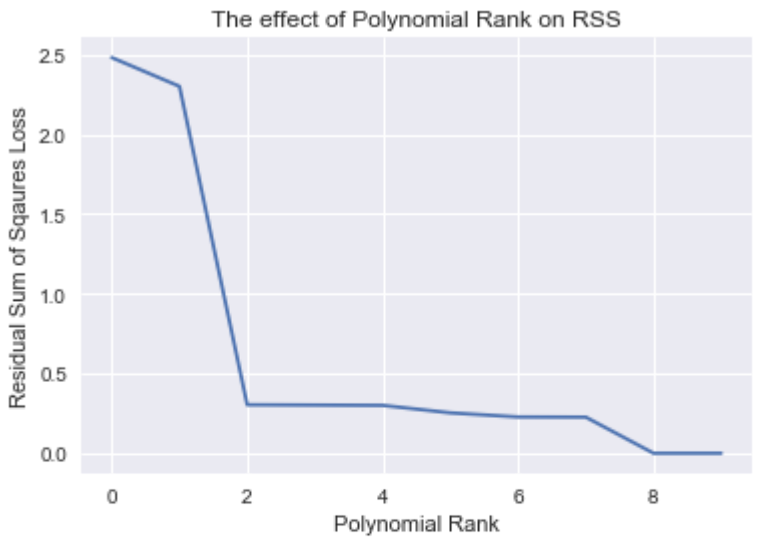
In [23]:
```python
#Visualizing Size of Coefficients
coef_mat = pd.DataFrame(coefs)  ## coefficent matrix
pd.options.display.float_format = '{:,.2g}'.format
coef_mat.index.name = 'polynomial rank'
coef_mat.columns = ['x1','x2','x3','x4','x5','x6','x7','x8','x9','x10']
coef_mat
```

Out[23]:

	x1	**x2**	**x3**	**x4**	**x5**	**x6**	**x7**	**x8**	**x9**
polynomial rank									
0	-1.5	nan	nan	nan	nan	nan	nan	nan	nan
1	-3.4	1.9	nan	nan	nan	nan	nan	nan	nan
2	16	-43	28	nan	nan	nan	nan	nan	nan
3	17	-47	35	-3.6	nan	nan	nan	nan	nan
4	15	-34	0.31	36	-16	nan	nan	nan	nan
5	-13	2.1e+02	-9.4e+02	1.7e+03	-1.5e+03	4.9e+02	nan	nan	nan
6	-7.1e+02	7.4e+03	-3.4e+04	8.1e+04	-1e+05	6.9e+04	-1.8e+04	nan	nan
7	6.6e+03	-7.4e+04	4e+05	-1.2e+06	2e+06	-2e+06	1.1e+06	-2.6e+05	nan
8	-6.5e+05	8e+06	-4.8e+07	1.7e+08	-3.7e+08	4.9e+08	-4.1e+08	1.9e+08	-3.8e+07
9	-5.5e+05	6.3e+06	-3.5e+07	1.1e+08	-1.7e+08	1.2e+08	6e+07	-1.8e+08	1.2e+08

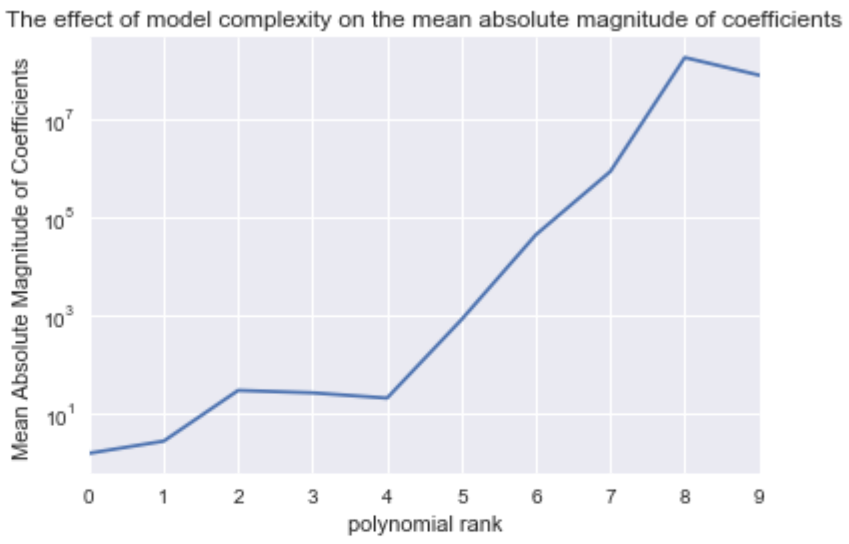
What we have observed above is known as coefficient explosion (magnitude of coefficients increases exponentially as model complexity increases) and is a key indication of overfitting. To further verify this, we will plot the Residual Sum of Squares (RSS) for each of the polynomial degrees. As seen in this plot, the highest rank polynomials have an RSS of 0 as they fit each point.

```
In [24]: plt.plot(rss)
         plt.xlabel('Polynomial Rank')
         plt.ylabel('Residual Sum of Sqaures Loss')
         plt.title("The effect of Polynomial Rank on RSS")

Out[24]: Text(0.5, 1.0, 'The effect of Polynomial Rank on RSS')
```



To illustrate coefficient explosion, we plot the mean absolute magnitude of the coefficients for each of the polynomial ranks. As seen, the average magnitude of the coefficient drastically increases with model complexity.

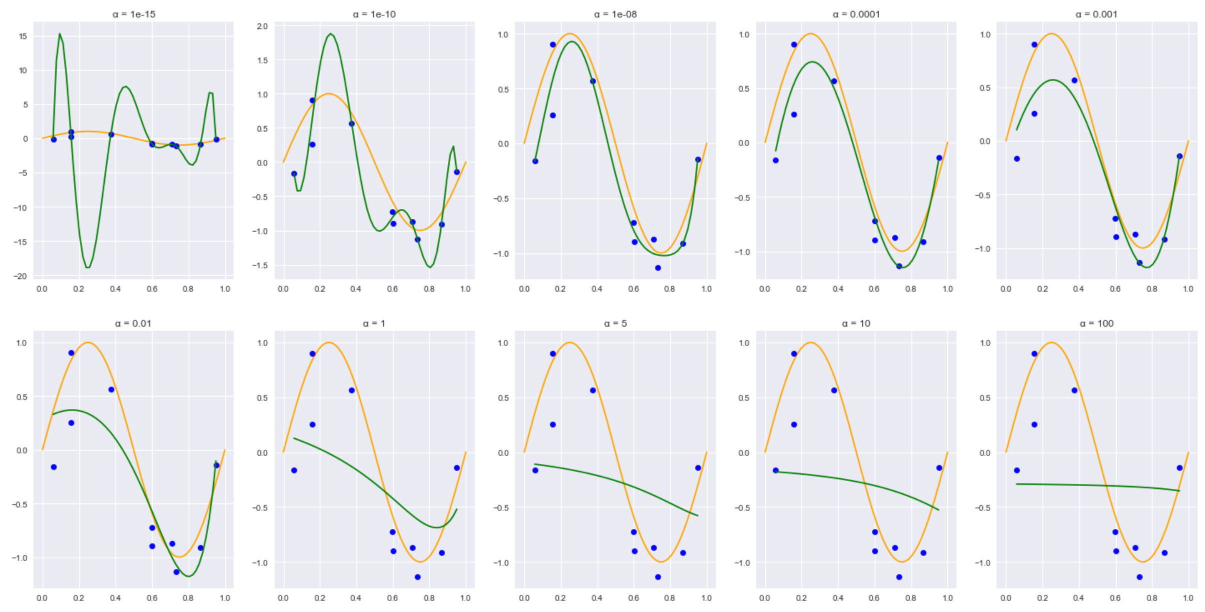
```
In [25]: abs(coef_mat).mean(axis=1).plot(logy=True)
         plt.ylabel("Mean Absolute Magnitude of Coefficients")
         plt.title("The effect of model complexity on the mean absolute magnitude of co
         efficients")

Out[25]: Text(0.5, 1.0, 'The effect of model complexity on the mean absolute magnitude
         of coefficients')
```



To mitigate the effect of overfitting with respect to polynomial rank, Ridge Regression is used. As you know, in terms of Ridge Regression, the α parameter defines the level of regularization; an α of 0 indicates no regularization whereas an α approaching ∞ indicates full regularization and reduces all coefficients to 0. The following code illustrates the effect of this regularization on the 10 term polynomial developed above.

In [27]:
```
coefs = []
rss = []
fig, axs = plt.subplots(2,5, figsize = (25,12.5))
alphas = [1e-15, 1e-10, 1e-8, 1e-4, 1e-3,1e-2, 1, 5, 10, 100]
for i in range(0,2):
    for j in range (0,5):
        RidgeModel = Ridge(normalize=True, alpha = alphas[5*i+j])
        RidgeModel.fit(data, y_noise)  # training the model using the 10 data
 points
        TestData_pred_curve = RidgeModel.predict(Test_Data) ## Model applied t
o the test data
        y_pred_points = RidgeModel.predict(data)  ## Model applied on the trai
ng data
        rss.append(np.sum(np.square(y_noise - y_pred_points))) ## training RSS
        coefs.append(RidgeModel.coef_)
        axs[i,j].plot(x2,y2, label = 'y = sin (2πx)', color='orange') ## the s
in function
        axs[i,j].plot(x, y_noise,"o" , color='b') # plot the 10 training data
 points
        axs[i,j].plot(Test_Data.x, TestData_pred_curve, color = 'g') # plot th
e predicted ouput of the 50 test data points
        axs[i,j].title.set_text("α = " + str(alphas[5*i+j]))
```

```
C:\Users\ashami2\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\linear_model\ridge.py:147: LinAlgWarning: Ill-conditioned matrix (rcond=6.33
133e-17): result may not be accurate.
  overwrite_a=True).T
```



As seen above, as the value of α increases, there is a transition from overfitting to underfitting. This shows the power of regression to reverse the effects of overfitting but also highlights the importance of appropriately affecting the α value as too high of a value results in severe underfitting. We can see the effect through the size of coefficients for the various values of α below.

```
In [10]: coef_mat = pd.DataFrame(coefs)
         pd.options.display.float_format = '{:,.2g}'.format
         coef_mat.index = [1e-15, 1e-10, 1e-8, 1e-4, 1e-3,1e-2, 1, 5, 10, 100]
         coef_mat.index.name = 'alpha'
         coef_mat.columns = ['x1','x2','x3','x4','x5','x6','x7','x8','x9','x10']
         coef_mat
```

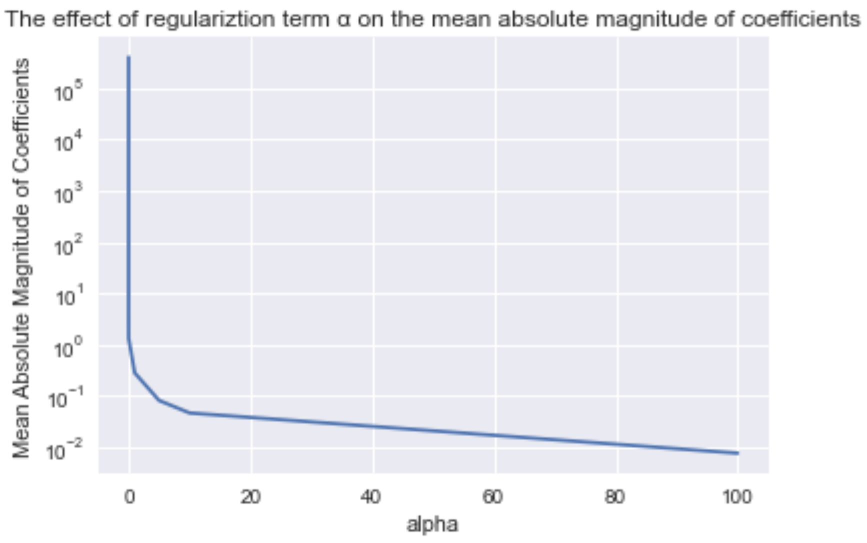
Out[10]:

alpha	x1	x2	x3	x4	x5	x6	x7	x8	x9	
1e-15	3.7e+03	-3.3e+04	1.1e+05	-7.3e+04	-3.7e+05	7.8e+05	-8.3e+04	-1.1e+06	1.2e+06	-3.
1e-10	-1.1e+02	1.1e+03	-3.4e+03	3.4e+03	2.2e+03	-3.2e+03	-3.8e+03	1.7e+03	5.9e+03	-3.
1e-08	4.4	46	-2.1e+02	1.6e+02	1.6e+02	-93	-1.9e+02	-11	1.9e+02	
0.0001	11	-20	-4.7	7.9	9.7	4.9	-1.2	-4.9	-3.9	
0.001	5.7	-9.2	-5.4	0.35	3.8	4.7	3.8	1.7	-0.92	
0.01	1.1	-2.8	-2.4	-1	0.2	0.98	1.3	1.4	1.2	
1	-0.55	-0.46	-0.34	-0.2	-0.063	0.053	0.16	0.25	0.33	
5	-0.2	-0.16	-0.13	-0.09	-0.054	-0.022	0.0047	0.029	0.05	
10	-0.11	-0.098	-0.079	-0.06	-0.042	-0.026	-0.012	-0.00018	0.01	
1e+02	-0.014	-0.013	-0.011	-0.0094	-0.0077	-0.0062	-0.0049	-0.0038	-0.0029	-

The following is a visualization of the ridge where we can see the size of the coefficients drastically dropping and approaching zero as α approaches ∞.

In [11]:
```
abs(coef_mat).mean(axis=1).plot(logy=True)
plt.ylabel("Mean Absolute Magnitude of Coefficients")
plt.title("The effect of regulariztion term α on the mean absolute magnitude o
f coefficients")
```

Out[11]:
```
Text(0.5, 1.0, 'The effect of regulariztion term α on the mean absolute magni
tude of coefficients')
```


The effect of regulariztion term α on the mean absolute magnitude of coefficients
Mean Absolute Magnitude of Coefficients
10^5
10^4
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
alpha