

## PSY9511: Seminar 3

### Regularization and variable selection

---

Esten H. Leonardsen

11.09.24

1. Assignment 1
2. Assignment 2
3. Regularization
  - Variable selection
  - Shrinkage (+ live coding 🤖)

## Assignment 1

---



UNIVERSITETET  
I OSLO

- Create a vector of 100 standard normally distributed numbers and visualize them with a histogram.
- Show rows 5, 8, 9, and 10 of the Auto dataset.
- Show the last three columns of the Auto dataset.
- Show all cars with five cylinders in the Auto dataset.

- Create a vector of 100 standard normally distributed numbers and visualize them with a histogram.
- Show rows 5, 8, 9, and 10 of the Auto dataset.
- Show the last three columns of the Auto dataset.
- Show all cars with five cylinders in the Auto dataset.

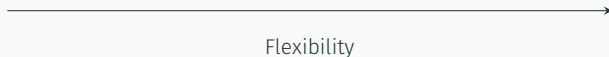
<http://localhost:8889/notebooks/notebooks%2FAssignment%201.ipynb>



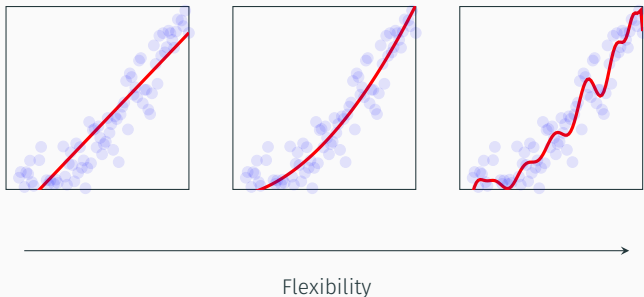
## Assignment 2: Bias-variance trade-off



## Assignment 2: Bias-variance trade-off

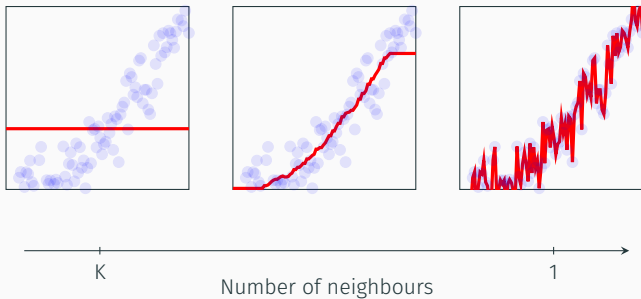


## Assignment 2: Bias-variance trade-off

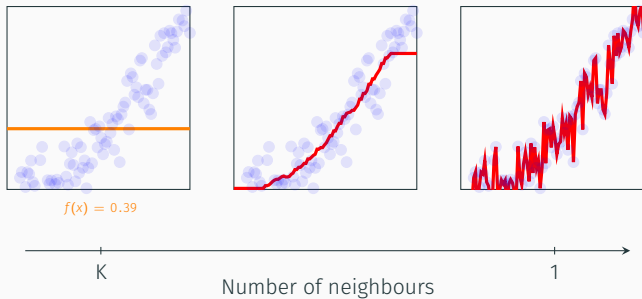




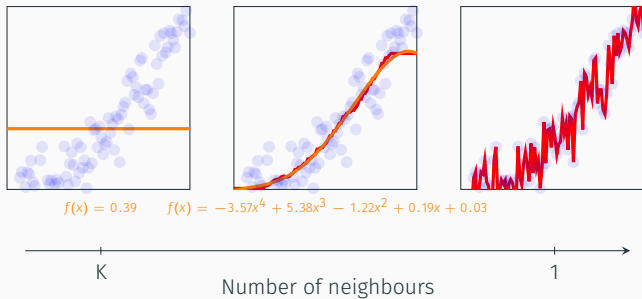
## Assignment 2: Bias-variance trade-off



## Assignment 2: Bias-variance trade-off



## Assignment 2: Bias-variance trade-off



## Assignment 2: Bias-variance trade-off

$$f(x) = 0.39 \quad f(x) = -3.57x^4 + 5.38x^3 - 1.22x^2 + 0.19x + 0.03$$



## Assignment 2: Bias-variance trade-off

$$f(x) = 0.39 \quad f(x) = -3.57x^4 + 5.38x^3 - 1.22x^2 + 0.19x + 0.03$$

↑  
1



## Assignment 2: Bias-variance trade-off

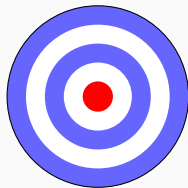
$$f(x) = 0.39 \quad f(x) = -3.57x^4 + 5.38x^3 - 1.22x^2 + 0.19x + 0.03$$

↑                    ↑        ↑        ↑        ↑        ↑  
1                    1        2        3        4        5



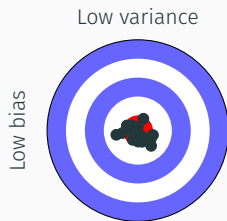
Model flexibility: Denotes the complexity of the approximated function  $\hat{y} = \hat{f}(x)$ .

- Informally: Wigglyness of the line
- Formally: Number of parameters in the function (degrees of freedom)

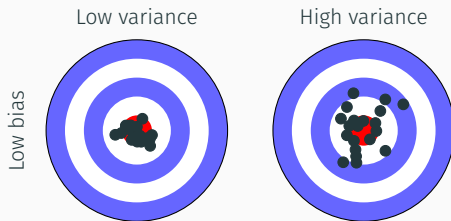




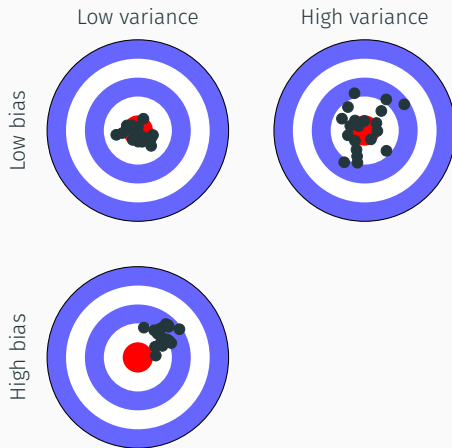
## Assignment 2: Bias-variance trade-off



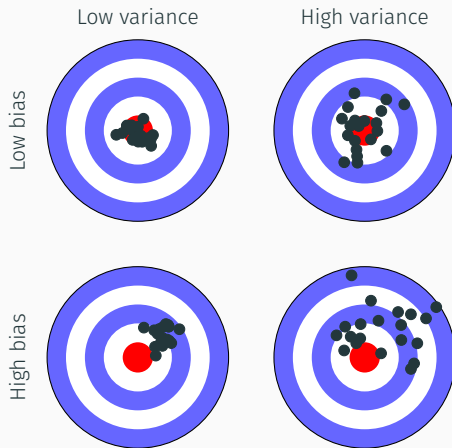
## Assignment 2: Bias-variance trade-off



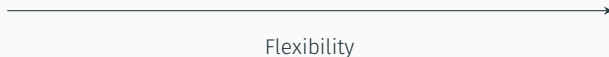
## Assignment 2: Bias-variance trade-off



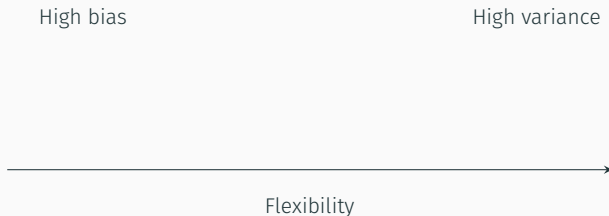
## Assignment 2: Bias-variance trade-off



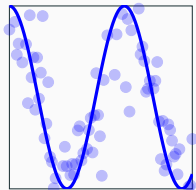
## Assignment 2: Bias-variance trade-off



## Assignment 2: Bias-variance trade-off

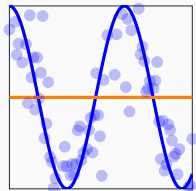


## Assignment 2: Bias-variance trade-off



→  
Flexibility

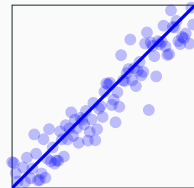
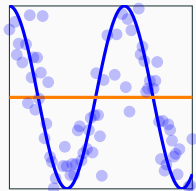
## Assignment 2: Bias-variance trade-off



→  
Flexibility

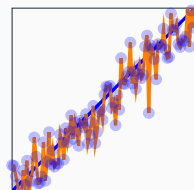
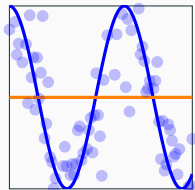


## Assignment 2: Bias-variance trade-off



Flexibility

## Assignment 2: Bias-variance trade-off

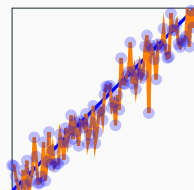
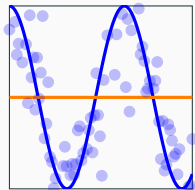


Flexibility

Bias and variance: Two ways the model can be bad

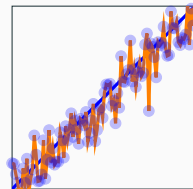
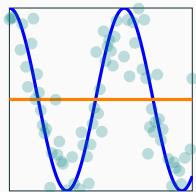
- High bias: The model misses in *systematic* ways
- High variance: The model misses in *unsystematic* ways

## Assignment 2: Bias-variance trade-off



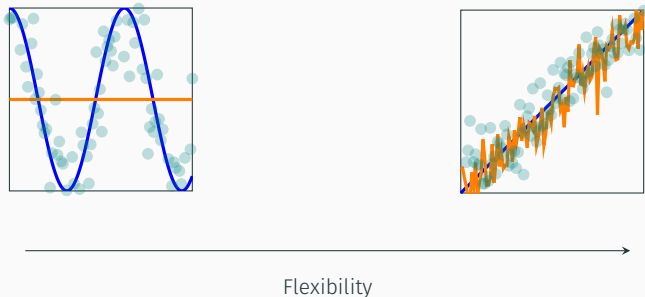
Flexibility

## Assignment 2: Bias-variance trade-off



Flexibility

## Assignment 2: Bias-variance trade-off



Underfitting and overfitting: Bias-variance trade-off in practice

- Underfitting: The model is equally bad on training and test data *due to not having captured the true relationship between inputs and outputs*
- Overfitting: The model is good on training data, but bad on test data *because it has found patterns in the noise during training*

## Assignment 2

---



UNIVERSITETET  
I OSLO



- Download the Auto.csv dataset from the ISLP website.
- Read the Auto.csv-dataset into memory.
- In the horsepower-column, some values are missing. These are encoded with '?'. Remove these rows from the dataset.
- Create a new column 'muscle'. This column should contain a 1 for all muscle cars (e.g. cars that have above average horsepower) and 0 for the rest.
- Split the dataset into a training set and a test set, by randomly drawing 80% of the rows for the former and 20% of the rows for the latter.

<http://localhost:8890/notebooks/notebooks/Assignment%202.ipynb>



- Fit a simple linear regression model using horsepower as the predictor and mpg as the outcome using the training data.
- Create a scatter plot with horsepower on the x-axis and mpg on the y-axis using the testing data. Plot the regression line found by the model in the plot.
- Use the model to generate predictions for the training set. Calculate and report the mean absolute error (MAE) of these predictions.
- Use the model to generate predictions for the test set. Calculate and report the MAE of the predictions.
- Reflection: Is the training and testing MAE is lower? Does this match your expectation? What would be the general pattern we expect here (e.g. one is lower than the other, they are the same, etc.), and why do we expect that?

<http://localhost:8890/notebooks/notebooks/Assignment%202.ipynb>



- Fit a multivariate linear regression model using horsepower, weight, displacement, and year as predictors and mpg as the outcome.
- Print the intercept and coefficients of the model .
- Use the model to generate predictions for the training set. Calculate and report the MAE of these predictions.
- Use the model to generate predictions for the test set. Calculate and report the MAE of the predictions.
- Reflection: Is the training MAE lower or higher than in the simple linear regression model? Does it have to be this way, or could it have been otherwise? What about the testing MAE?

<http://localhost:8890/notebooks/notebooks/Assignment%202.ipynb>



- Fit a logistic regression model using weight, displacement and year as predictors and our newly created muscle-column as the outcome. Why don't we use horsepower as a predictor in this model?
- Use the model to generate predictions for the training set. Calculate and report the accuracy of these predictions.
- Use the model to generate predictions for the testing set. Calculate and report the accuracy of these predictions.

<http://localhost:8890/notebooks/notebooks/Assignment%202.ipynb>



## Assignment 2: Data splitting

```
In[1]: import pandas as pd

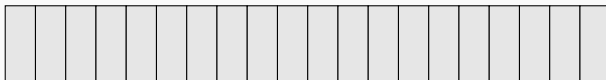
df = pd.DataFrame(...)
train = df.sample(frac=0.8)
test = df.sample(frac=0.2)
```



## Assignment 2: Data splitting

```
In[1]: import pandas as pd

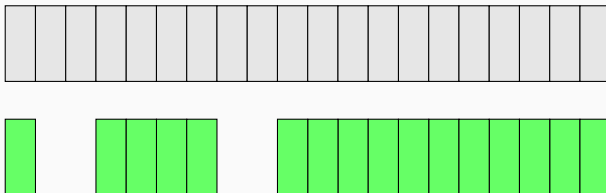
df = pd.DataFrame(...)
train = df.sample(frac=0.8)
test = df.sample(frac=0.2)
```



## Assignment 2: Data splitting

```
In[1]: import pandas as pd

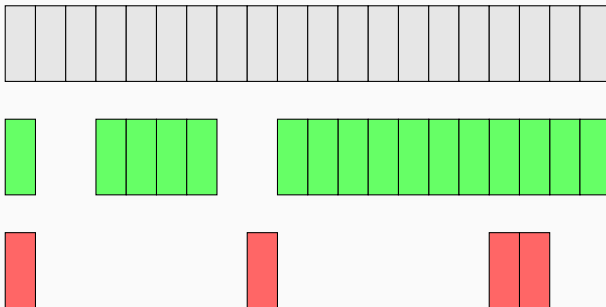
df = pd.DataFrame(...)
train = df.sample(frac=0.8)
test = df.sample(frac=0.2)
```



## Assignment 2: Data splitting

```
In[1]: import pandas as pd

df = pd.DataFrame(...)
train = df.sample(frac=0.8)
test = df.sample(frac=0.2)
```

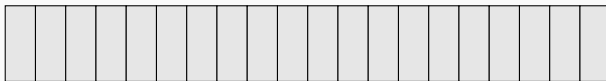




## Assignment 2: Data splitting

```
In[1]: import pandas as pd

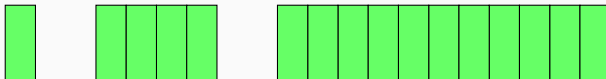
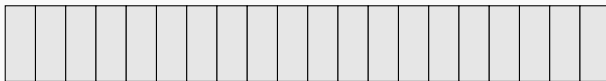
df = pd.DataFrame(...)
train = df.sample(frac=0.8)
test = df.drop(train.index)
```



## Assignment 2: Random seeds

```
In[1]: import pandas as pd

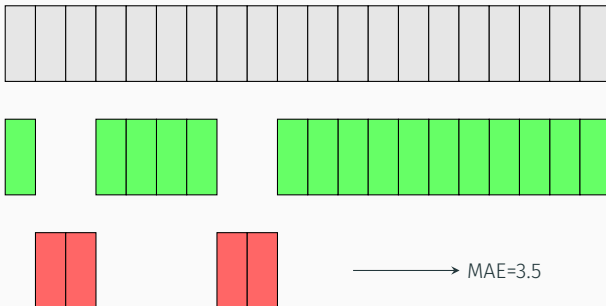
df = pd.DataFrame(...)
train = df.sample(frac=0.8)
test = df.drop(train.index)
```



## Assignment 2: Random seeds

```
In[1]: import pandas as pd

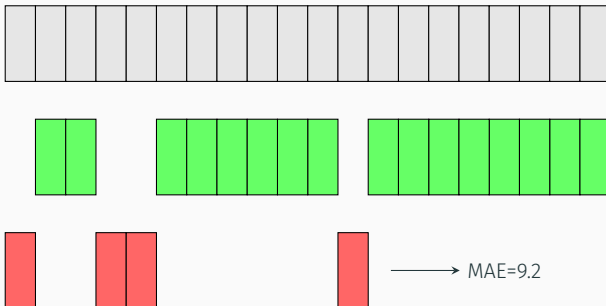
df = pd.DataFrame(...)
train = df.sample(frac=0.8)
test = df.drop(train.index)
```



## Assignment 2: Random seeds

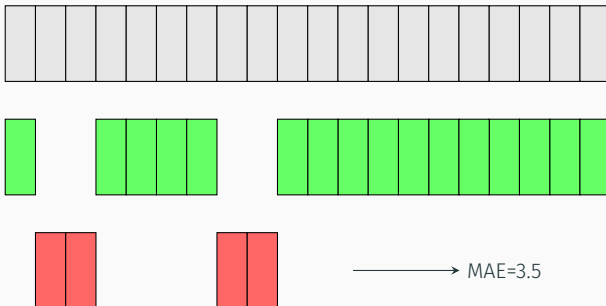
```
In[1]: import pandas as pd

df = pd.DataFrame(...)
train = df.sample(frac=0.8)
test = df.drop(train.index)
```



## Assignment 2: Random seeds

```
In[1]: import pandas as pd
import numpy as np
np.random.seed(42)
df = pd.DataFrame(...)
train = df.sample(frac=0.8)
test = df.drop(train.index)
```



## Assignment 2: Log-odds vs probability vs class

```
model <- glm(muscle ~ year + weight, data=df, family='binomial')  
predict(model, df)
```



## Assignment 2: Log-odds vs probability vs class

```
model <- glm(muscle ~ year + weight, data=df, family='binomial')  
predict(model, df)
```

| 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | 10     | 11     |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 1.2460 | 1.9245 | 1.0019 | 0.9911 | 1.0485 | 4.2506 | 4.1465 | 4.5522 | 2.4889 | 1.4578 | 1.6223 |



## Assignment 2: Log-odds vs probability vs class

```
model <- glm(muscle ~ year + weight, data=df, family='binomial')  
predict(model, df)
```

| 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | 10     | 11     |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 1.2460 | 1.9245 | 1.0019 | 0.9911 | 1.0485 | 4.2506 | 4.1465 | 4.5522 | 2.4889 | 1.4578 | 1.6223 |

```
predict(model, df, type="response")
```

| 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | 10     | 11     |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 0.7766 | 0.8726 | 0.7314 | 0.7293 | 0.7405 | 0.9853 | 0.9844 | 0.9895 | 0.9233 | 0.8112 | 0.8352 |





## Assignment 2: Log-odds vs probability vs class

```
In[1]: model = LogisticRegression()  
model.fit(df[['year', 'weight']], df['muscle'])  
model.predict(df[['year', 'weight']])
```

```
Out[1]: array([0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0])
```

```
In[1]: model.predict_proba(df[['year', 'weight']])
```

```
Out[1]: array([[0.14, 0.86], [0.08, 0.92], [0.17, 0.83], [0.18, 0.82]])
```



## Assignment 2: Eye test

```
model <- glm(muscle ~ year + weight, data=df, family='binomial')  
predict(model, df)
```

| 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | 10     | 11     |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 1.2460 | 1.9245 | 1.0019 | 0.9911 | 1.0485 | 4.2506 | 4.1465 | 4.5522 | 2.4889 | 1.4578 | 1.6223 |



## Assignment 2: Eye test

```
model <- glm(muscle ~ year + weight, data=df, family='binomial')
predict(model, df)
```

| 1      | 2      | 3      | 4      | 5      | 6      | 7      | 8      | 9      | 10     | 11     |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 1.2460 | 1.9245 | 1.0019 | 0.9911 | 1.0485 | 4.2506 | 4.1465 | 4.5522 | 2.4889 | 1.4578 | 1.6223 |

```
model <- lm(mpg ~ horsepower, df)
summary(model)
```

```
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 19.59412  0.96187 20.371 < 2e-16 ***
horsepower102 0.40588  4.08087  0.099  0.920840
horsepower103 0.70588  4.08087  0.173  0.862789
horsepower105 0.90588  1.49529  0.606  0.545091
```



## Regularization

---

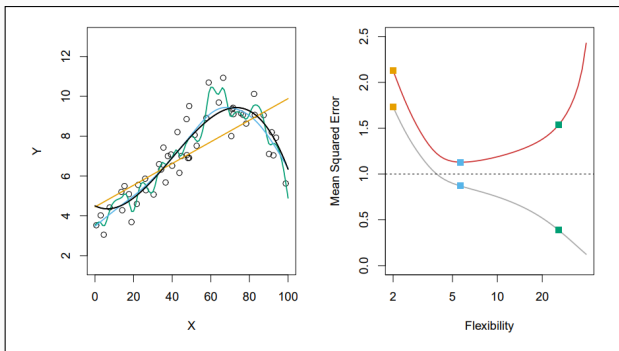


UNIVERSITETET  
I OSLO

$$y \sim \beta_0 + \beta_1 * x_1 + \beta_2 * x_2$$

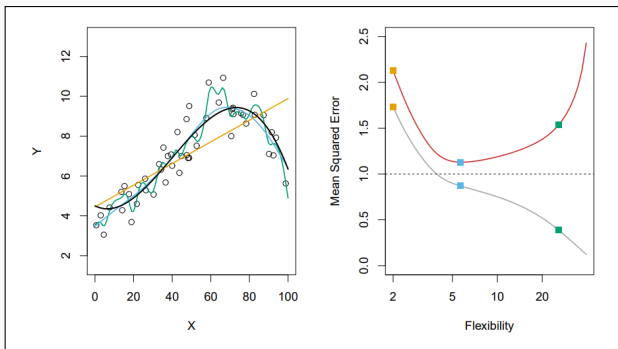
# Regularization: Preparations

$$y \sim \beta_0 + \beta_1 * x_1 + \beta_2 * x_2$$



# Regularization: Preparations

$$y \sim \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \beta_3 * x_3 + \beta_4 * x_4 + \beta_5 * x_5 + \beta_6 * x_6$$



```
In[1]: import pandas as pd

df = pd.read_csv('/Users/esten/Downloads/Auto.csv')
train = df.iloc[:int(len(df) * 0.8)]
validation = df.iloc[int(len(df) * 0.8):]

print(f'Using len(train) samples for training')
print(f'Using len(validation) samples for validation')
```

```
Out[1]: Using 317 samples for training
Using 80 samples for validation
```



1. Variable selection
  - a. Best subset selection
  - b. Forward stepwise selection
  - c. Backward stepwise selection
2. Shrinkage
  - a. LASSO
  - b. Ridge Regression
3. Dimensionality reduction: Lecture 6 and self-study

1. Variable selection
  - a. Best subset selection
  - b. Forward stepwise selection
  - c. Backward stepwise selection
2. Shrinkage
  - a. **LASSO**
  - b. Ridge Regression
3. Dimensionality reduction: Lecture 6 and self-study

## Variable selection

---



UNIVERSITETET  
I OSLO

The number of predictors we are using in our model directly impacts model complexity.

## Problem

We have a set of predictors  $P = \{x_0, x_1, \dots\}$  and a target variable  $y$ , and we want to find the subset  $p \subseteq P$  that yields the best (linear) model for predicting  $y$ .

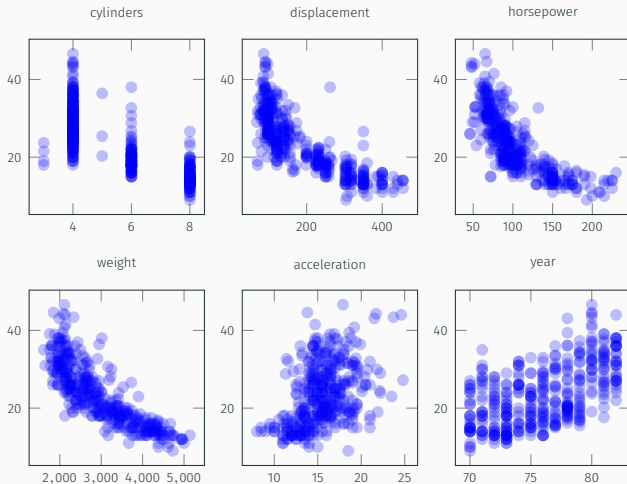
## Problem

We have a set of predictors  $P = \{x_0, x_1, \dots\}$  and a target variable  $y$ , and we want to find the subset  $p \subseteq P$  that yields the best (linear) model for predicting  $y$ .

## Motivation

To reduce model complexity (and therefore risk of overfitting), and to simplify subsequent interpretations.

# Variable selection



# Variable selection: Best subset selection

## Problem

We have a set of predictors  $P = \{x_0, x_1, \dots\}$  and a target variable  $y$ , and we want to find the subset  $p \subseteq P$  that yields the best (linear) model for predicting  $y$ .

## Solution

Train models on all subsets  $p$  and select the best one.





# Variable selection: Best subset selection

```
In[1]: import numpy as np

from itertools import chain, combinations
from sklearn.linear_model import LinearRegression

subsets = list(chain.from_iterable(combinations(predictors, r)
                                   for r in range(len(predictors)+1)))

best = {'mse': float('inf'), 'subset': None}

for subset in subsets:
    if len(subset) == 0:
        continue

    model = LinearRegression()
    model.fit(train[list(subset)], train[target])
    predictions = model.predict(validation[list(subset)])
    mse = np.mean((predictions - validation[target]) ** 2)

    if mse < best['mse']:
        best = {'mse': mse, 'subset': subset}

print(f'MSE: {best["mse"]:.2f}, predictors: {best["subset"]}')
```

```
Out[1]: MSE: 29.68, predictors: ('cylinders', 'displacement', 'horsepower', 'weight', 'year')
```



# Variable selection: Best subset selection

## Problem

We have a set of predictors  $P = \{x_0, x_1, \dots\}$  and a target variable  $y$ , and we want to find the subset  $p \subseteq P$  that yields the best (linear) model for predicting  $y$ .

## Solution

Train models on all subsets  $p$  and select the best one.

### + Positives

Guaranteed to find the optimal solution.

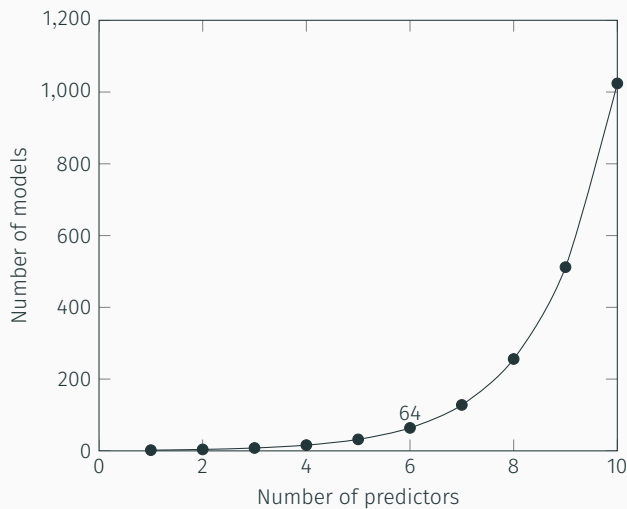
Simple implementation

### - Drawbacks

Need to train many ( $2^{|P|}$ ) models.



## Variable selection: Best subset selection



# Variable selection: Forward stepwise selection

## Problem

We have a set of predictors  $P = \{x_0, x_1, \dots\}$  and a target variable  $y$ , and we want to find the subset  $p \subseteq P$  that yields the best (linear) model for predicting  $y$ .

## Solution

Start with no predictors. Iteratively add the predictor that yields the best model until all are included.



# Variable selection: Forward stepwise selection

## Problem

We have a set of predictors  $P = \{x_0, x_1, \dots\}$  and a target variable  $y$ , and we want to find the subset  $p \subseteq P$  that yields the best (linear) model for predicting  $y$ .

## Solution

Start with no predictors. Iteratively add the predictor that yields the best model until all are included.

$$\begin{array}{l} y \sim \mathbf{1} \\ mse = 146.47 \end{array}$$



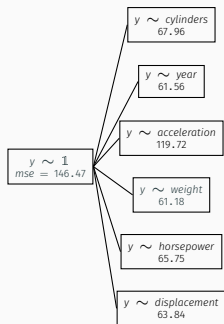
# Variable selection: Forward stepwise selection

## Problem

We have a set of predictors  $P = \{x_0, x_1, \dots\}$  and a target variable  $y$ , and we want to find the subset  $p \subseteq P$  that yields the best (linear) model for predicting  $y$ .

## Solution

Start with no predictors. Iteratively add the predictor that yields the best model until all are included.



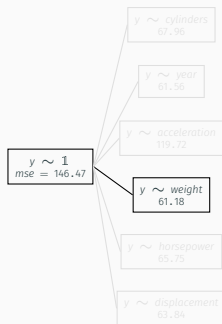
# Variable selection: Forward stepwise selection

## Problem

We have a set of predictors  $P = \{x_0, x_1, \dots\}$  and a target variable  $y$ , and we want to find the subset  $p \subseteq P$  that yields the best (linear) model for predicting  $y$ .

## Solution

Start with no predictors. Iteratively add the predictor that yields the best model until all are included.



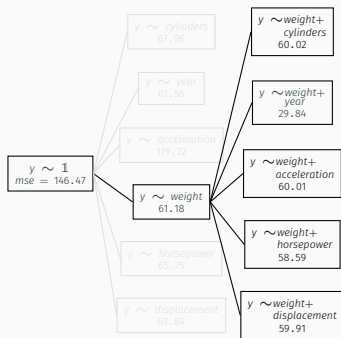
# Variable selection: Forward stepwise selection

## Problem

We have a set of predictors  $P = \{x_0, x_1, \dots\}$  and a target variable  $y$ , and we want to find the subset  $p \subseteq P$  that yields the best (linear) model for predicting  $y$ .

## Solution

Start with no predictors. Iteratively add the predictor that yields the best model until all are included.





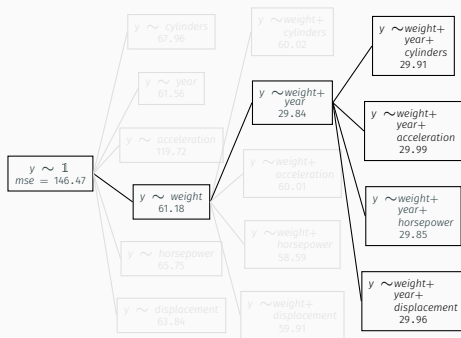
# Variable selection: Forward stepwise selection

## Problem

We have a set of predictors  $P = \{x_0, x_1, \dots\}$  and a target variable  $y$ , and we want to find the subset  $p \subseteq P$  that yields the best (linear) model for predicting  $y$ .

## Solution

Start with no predictors. Iteratively add the predictor that yields the best model until all are included.



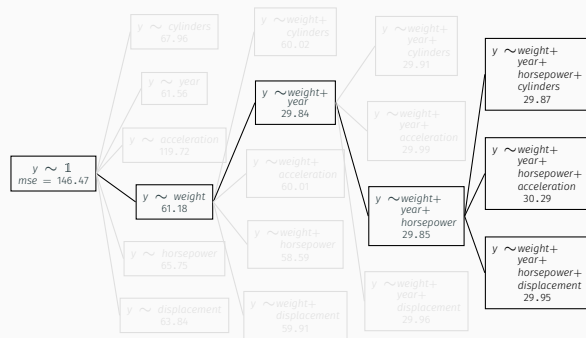
# Variable selection: Forward stepwise selection

## Problem

We have a set of predictors  $P = \{x_0, x_1, \dots\}$  and a target variable  $y$ , and we want to find the subset  $p \subseteq P$  that yields the best (linear) model for predicting  $y$ .

## Solution

Start with no predictors. Iteratively add the predictor that yields the best model until all are included.



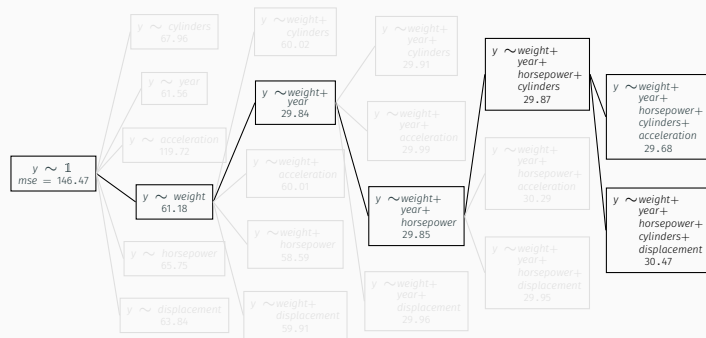
# Variable selection: Forward stepwise selection

## Problem

We have a set of predictors  $P = \{x_0, x_1, \dots\}$  and a target variable  $y$ , and we want to find the subset  $p \subseteq P$  that yields the best (linear) model for predicting  $y$ .

## Solution

Start with no predictors. Iteratively add the predictor that yields the best model until all are included.



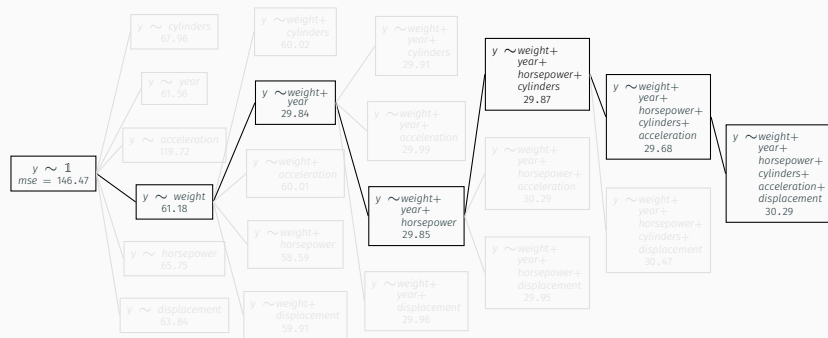
# Variable selection: Forward stepwise selection

## Problem

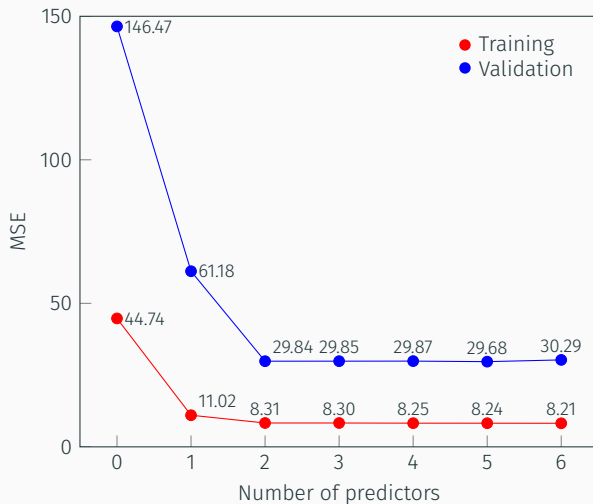
We have a set of predictors  $P = \{x_0, x_1, \dots\}$  and a target variable  $y$ , and we want to find the subset  $p \subseteq P$  that yields the best (linear) model for predicting  $y$ .

## Solution

Start with no predictors. Iteratively add the predictor that yields the best model until all are included.



## Variable selection: Forward stepwise selection



# Variable selection: Forward stepwise selection

```
In[1]: def fit_and_evaluate(train: pd.DataFrame, validation: pd.DataFrame,
    predictors: List[str], target: str):
    model = LinearRegression()
    model.fit(train[predictors], train[target])

    train_predictions = model.predict(train[predictors])
    validation_predictions = model.predict(validation[predictors])

    return np.mean((train_predictions - train[target]) ** 2),
           np.mean((validation_predictions - validation[target]) ** 2)

predictors = ['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year']
target = 'mpg'

train['intercept'] = 1
validation['intercept'] = 1
train_mse, validation_mse = fit_and_evaluate(train, validation, predictors=['intercept'], target=target)
print(f'[]: {validation_mse:.2f} ({train_mse:.2f})')

chosen_predictors = []

while len(chosen_predictors) < len(predictors):
    best_predictor = {'train_mse': None, 'validation_mse': float('inf'),
                     'predictor': None}

    for predictor in set(predictors) - set(chosen_predictors):
        train_mse, validation_mse = fit_and_evaluate(train, validation, predictors=chosen_predictors + [predictor], target=target)

        if validation_mse < best_predictor['validation_mse']:
            best_predictor = {'train_mse': train_mse, 'validation_mse': validation_mse, 'predictor': predictor}

    chosen_predictors.append(best_predictor['predictor'])

print(f'{chosen_predictors}: {best_predictor["validation_mse"]:.2f} ({best_predictor["train_mse"]:.2f})')
```



# Variable selection: Forward stepwise selection

## Problem

We have a set of predictors  $P = \{x_0, x_1, \dots\}$  and a target variable  $y$ , and we want to find the subset  $p \subseteq P$  that yields the best (linear) model for predicting  $y$ .

## Solution

Start with no predictors. Iteratively add the predictor that yields the best model until all are included.

### + Positives

Need to train fewer models.

### - Drawbacks

Not guaranteed to find the optimal solution.



# Variable selection: Backward stepwise selection

## Problem

We have a set of predictors  $P = \{x_0, x_1, \dots\}$  and a target variable  $y$ , and we want to find the subset  $p \subseteq P$  that yields the best (linear) model for predicting  $y$ .

## Solution

Start with all predictors. Iteratively remove the predictor that yields the best model until all you have none left.





# Variable selection: Backward stepwise selection

## Problem

We have a set of predictors  $P = \{x_0, x_1, \dots\}$  and a target variable  $y$ , and we want to find the subset  $p \subseteq P$  that yields the best (linear) model for predicting  $y$ .

## Solution

Start with all predictors. Iteratively remove the predictor that yields the best model until all you have none left.

### + Positives

Need to train fewer models.

### - Drawbacks

Not guaranteed to find the optimal solution.



## Shrinkage

---



UNIVERSITETET  
I OSLO

$$y \sim \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6$$

$$y \sim \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6$$

$$y \sim \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6$$
$$\beta_n \rightarrow 0$$

$$y \sim \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6$$
$$\beta_n \rightarrow 0$$

1.  $\beta_1 = 0 \implies$  One less degree of freedom in our function

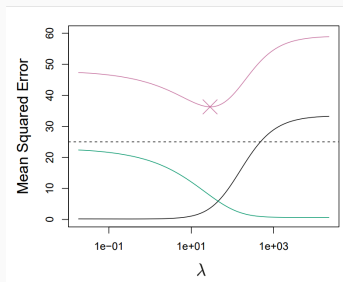
$$y \sim \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6$$
$$\beta_n \rightarrow 0$$

1.  $\beta_1 = 0 \implies$  One less degree of freedom in our function

$$mse = bias^2 + variance + irreducible\ error$$

$$y \sim \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \beta_5 X_5 + \beta_6 X_6$$
$$\beta_n \rightarrow 0$$

1.  $\beta_1 = 0 \implies$  One less degree of freedom in our function



$$\text{mse} = \text{bias}^2 + \text{variance} + \text{irreducible error}$$



$$y \sim \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6$$
$$\beta_n \rightarrow 0$$

1.  $\beta_1 = 0 \implies$  One less degree of freedom in our function
2. A little more bias  $\implies$  A lot less variance

$$y \sim \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6$$
$$\beta_n \rightarrow 0$$

1.  $\beta_1 = 0 \implies$  One less degree of freedom in our function
2. A little more bias  $\implies$  A lot less variance

$$y \sim \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6$$

↑  
↓

$$y \sim \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6$$
$$\beta_n \rightarrow 0$$

1.  $\beta_1 = 0 \implies$  One less degree of freedom in our function
2. A little more bias  $\implies$  A lot less variance
3. Parameters depend on each other  $\implies$   
Fewer degrees of freedom

$$loss_{mse} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2$$

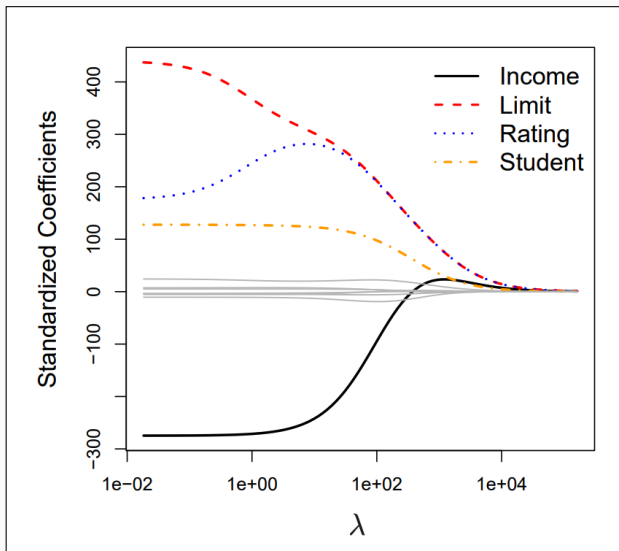
$$loss_{ridge} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^p \beta_j^2$$

$$loss_{ridge} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^p \beta_j^2$$

$\Downarrow$

$$\lambda \rightarrow \infty \Rightarrow \beta \rightarrow 0$$

## Shrinkage: Ridge regression

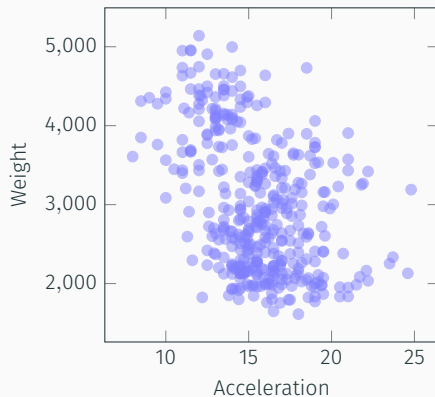


$$loss_{ridge} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^p \beta_j^2$$

$$y \sim \beta_0 + \beta_1 x_1 + \beta_2 x_2, x_1 \in [0, 1], x_2 \in [0, 1000]$$



## Shrinkage: Feature standardization



z-score standardization

## z-score standardization

$$x = \frac{x - \mu_x}{\sigma_x}$$

## z-score standardization

$$x = \frac{x - \mu_x}{\sigma_x}$$

```
In[1]: for col in predictors:
        print(f'{col}: {np.mean(df[col]):.2f} ({np.std(df[col]):.2f})')

        # z-score standardization
        for col in predictors:
            df[col] = (df[col] - np.mean(df[col])) / np.std(df[col])

        for col in predictors:
            print(f'{col} after: {np.mean(df[col]):.2f} ({np.std(df[col]):.2f})')
```

```
Out[1]: cylinders: 5.47 (1.70)
displacement: 194.41 (104.51)
horsepower: 104.47 (38.44)
cylinders after: -0.00 (1.00)
displacement after: -0.00 (1.00)
horsepower after: -0.00 (1.00)
```



$$loss_{ridge} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^p \beta_j^2$$

$$\text{loss}_{\text{ridge}} = \sum_{i=0}^n \left( y_{ij} - \sum_{j=0}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^p \beta_j^2$$

Blackboard!

$$loss_{ridge} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^p \beta_j^2$$

<http://localhost:8888/notebooks/notebooks/Live%20coding.ipynb>



$$\text{loss}_{\text{ridge}} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^p \beta_j^2$$

Regularization by shrinking the model covariates towards zero.



$$loss_{ridge} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^p \beta_j^2$$

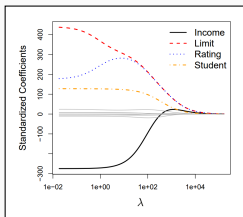
$$loss_{lasso} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^p |\beta_j|$$

$$loss_{ridge} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^p \beta_j^2$$

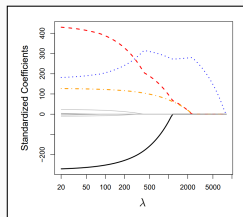
$$loss_{lasso} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^p |\beta_j|$$

# Shrinkage: LASSO

Ridge

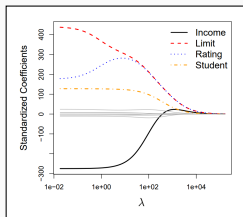


LASSO

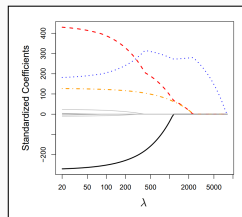


# Shrinkage: LASSO

Ridge



LASSO

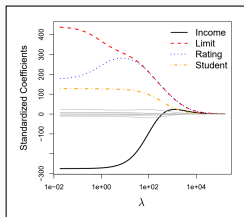


| Predictor    | Ridge | LASSO |
|--------------|-------|-------|
| Intercept    | 23.44 | 23.44 |
| Weight       | -5.59 | -4.78 |
| Year         | 2.75  | 2.00  |
| Acceleration | 0.19  | 0     |
| Displacement | 0.66  | 0     |

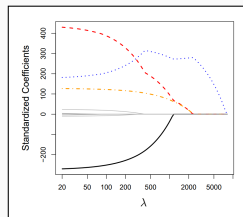


# Shrinkage: LASSO

Ridge



LASSO

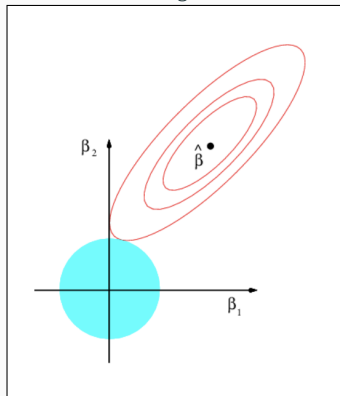


A coefficient of 0 does not mean the predictor has  
no association with the outcome!

Lasso

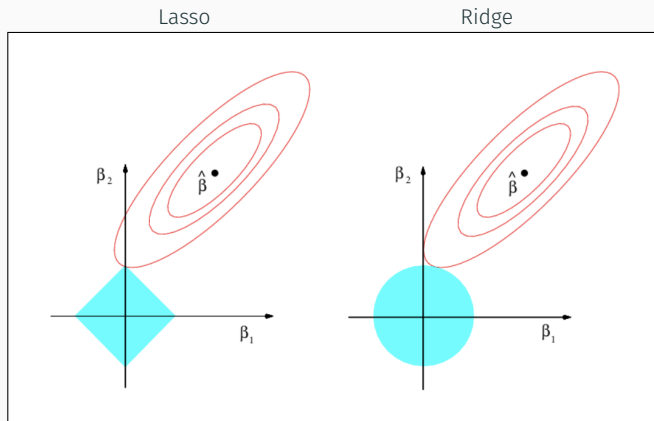
?

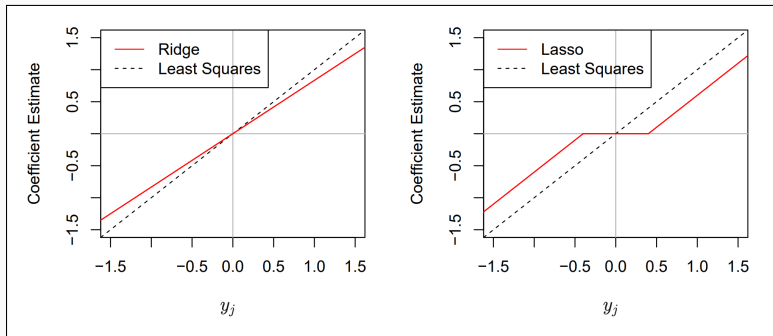
Ridge



Whiteboard! 🤖

# Shrinkage: LASSO







$$loss_{mse} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2$$

Fits the **best** model  
to the data.

$$loss_{mse} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2$$

Fits the **best** model to the data.

$$loss_{ridge} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^p \beta_j^2$$

Fits the **best** model to the data while **shrinking** coefficients towards zero.

$$loss_{mse} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2$$

Fits the **best** model to the data.

$$loss_{ridge} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^p \beta_j^2$$

Fits the **best** model to the data while **shrinking** coefficients towards zero.

$$loss_{lasso} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^p |\beta_j|$$

Fits the **best** model to the data while **shrinking** coefficients towards zero such that some variables are effectively **removed**.

## Assignment 3

---



UNIVERSITETET  
I OSLO

[https://uio.instructure.com/courses/53357/assignments/118667?module\\_item\\_id=962921](https://uio.instructure.com/courses/53357/assignments/118667?module_item_id=962921)

# Coding tips: Separation of concerns

```
In[1]: # Read and clean data
path = '/Users/esten/Downloads/Auto.csv'
df = pd.read_csv(path)

# Split data
train = df.iloc[:int(len(df) * 0.8)]
validation = df.iloc[int(len(df) * 0.8):]

# Define input and output variables
predictors = ['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year']
target = 'mpg'

# Define necessary data structures for state
chosen_predictors = []
mses = []

while len(predictors) > 0:
    best_predictor = {'mse': float('inf'), 'predictor': None}

    for predictor in set(predictors) - set(chosen_predictors):
        potential_predictors = chosen_predictors + [predictor]

        # Fit and evaluate model
        model = LinearRegression()
        model.fit(train[potential_predictors], train[target])
        predictions = model.predict(validation[potential_predictors])
        test_mse = np.mean((validation[target] - predictions) ** 2)

        # Compare model with previous best
        if test_mse < best_predictor['mse']:
            best_predictor = {'mse': test_mse, 'predictor': predictor}

    # Update state
    chosen_predictors.append(best_predictor['predictor'])
    mses.append(best_predictor['mse'])
    predictors = [p for p in predictors if p != best_predictor['predictor']]
```



# Coding tips: Separation of concerns

In[1]:

```
# Read and clean data
path = '/Users/esten/Downloads/Auto.csv'
df = pd.read_csv(path)

# Split data
train = df.iloc[:int(len(df) * 0.8)]
validation = df.iloc[int(len(df) * 0.8):]

# Define input and output variables
predictors = ['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year']
target = 'mpg'

# Define necessary data structures for state
chosen_predictors = []
mses = []

while len(predictors) > 0:
    best_predictor = {'mse': float('inf'), 'predictor': None}

    for predictor in set(predictors) - set(chosen_predictors):
        potential_predictors = chosen_predictors + [predictor]

        # Fit and evaluate model
        model = LinearRegression()
        model.fit(train[potential_predictors], train[target])
        predictions = model.predict(validation[potential_predictors])
        test_mse = np.mean((validation[target] - predictions) ** 2)

        # Compare model with previous best
        if test_mse < best_predictor['mse']:
            best_predictor = {'mse': test_mse, 'predictor': predictor}

    # Update state
    chosen_predictors.append(best_predictor['predictor'])
    mses.append(best_predictor['mse'])
    predictors = [p for p in predictors if p != best_predictor['predictor']]
```

Setup

Selection

Modelling

Housekeeping



# Coding tips: Separation of concerns

```
In[1]: # Read and clean data
path = '/Users/esten/Downloads/Auto.csv'
df = pd.read_csv(path)

# Split data
train = df.iloc[:int(len(df) * 0.8)]
validation = df.iloc[int(len(df) * 0.8):]

# Define input and output variables
predictors = ['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year']
target = 'mpg'

# Define necessary data structures for state
chosen_predictors = []
mses = []

def fit_and_evaluate(model: LinearRegression, train: pd.DataFrame, validation: pd.DataFrame, variables: List[str], target: str):
    """ Fit a given model on a training dataset using a given set of variables and return MSE from a validation dataset. """
    model.fit(train[potential_predictors], train[target])
    predictions = model.predict(validation[potential_predictors])

    return np.mean((validation[target] - predictions) ** 2)

while len(predictors) > 0:
    best_predictor = {'mse': float('inf'), 'predictor': None}

    for predictor in set(predictors) - set(chosen_predictors):
        potential_predictors = chosen_predictors + [predictor]
        test_mse = fit_and_evaluate(LinearRegression(), train, validation, variables=potential_predictors, target=target)

        # Compare model with previous best
        if test_mse < best_predictor['mse']:
            best_predictor = {'mse': test_mse, 'predictor': predictor}

    # Update state
    chosen_predictors.append(best_predictor['predictor'])
    mses.append(best_predictor['mse'])
    predictors = [p for p in predictors if p != best_predictor['predictor']]
```

Modelling

