

## PSY9511: Seminar 3

### Regularization and variable selection

---

Esten H. Leonardsen

07.09.23

## 1. Introduction

- Pragmatic reflections on different models
- Python
- Coding tips: Separation of concerns

## 2. Regularization

- Variable selection
- Shrinkage (+ live coding 🤖)
- Dimensionality reduction

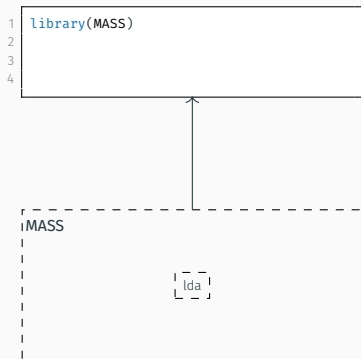
Which models does you *really* need to know?

```
1  
2  
3  
4
```



```
MASS  
  
lda
```





```
1 library(MASS)
2
3 lda_fit <- lda(display ~ age + fb,
4               data = display)
```

MASS

[ lda ]

# Python: Imports

```
1 library(MASS)
2
3 lda_fit <- lda(display ~ age + fb,
4               data = display)
```

MASS

[lda]

```
In[1]: from sklearn import *

lda = discriminant_analysis.
       LinearDiscriminantAnalysis()
lda.fit(display[['age', 'fb']],
        display['display'])
```

sklearn

discriminant\_analysis

LinearDiscriminantAnalysis

# Python: Imports

```
1 library(MASS)
2
3 lda_fit <- lda(display ~ age + fb,
4               data = display)
```

MASS

lda

```
In[1]: import sklearn
```

```
lda = sklearn.discriminant_analysis.
      LinearDiscriminantAnalysis()
lda.fit(display[['age', 'fb']],
        display['display'])
```

sklearn

discriminant\_analysis

LinearDiscriminantAnalysis



# Python: Imports

```
1 library(MASS)
2
3 lda_fit <- lda(display ~ age + fb,
4               data = display)
```

MASS

[lda]

```
In[1]: from sklearn.discriminant_analysis \
import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis()
lda.fit(display[['age', 'fb']],
        display['display'])
```

sklearn

discriminant\_analysis

LinearDiscriminantAnalysis

```
1 path <- '/Users/esten/Downloads/Auto.csv'  
2 df <- read.csv(path)  
3 head(df, 10)
```

|    | mpg | cylinders | displacement | horsepower |
|----|-----|-----------|--------------|------------|
| 1  | 18  | 8         | 307.0        | 130        |
| 2  | 15  | 8         | 350.0        | 165        |
| 3  | 18  | 8         | 318.0        | 150        |
| 4  | 16  | 8         | 304.0        | 150        |
| 5  | 17  | 8         | 302.0        | 140        |
| 6  | 15  | 8         | 429.0        | 198        |
| 7  | 14  | 8         | 454.0        | 220        |
| 8  | 14  | 8         | 440.0        | 215        |
| 9  | 14  | 8         | 455.0        | 225        |
| 10 | 15  | 8         | 390.0        | 190        |

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

```
path = '/Users/esten/Downloads/Auto.csv'  
df = pd.read_csv(path)  
df.head(10)
```

```
Out[1]:
```

|   | mpg | cylinders | displacement | horsepower |
|---|-----|-----------|--------------|------------|
| 0 | 18  | 8         | 307.0        | 130        |
| 1 | 15  | 8         | 350.0        | 165        |
| 2 | 18  | 8         | 318.0        | 150        |
| 3 | 16  | 8         | 304.0        | 150        |
| 4 | 17  | 8         | 302.0        | 140        |
| 5 | 15  | 8         | 429.0        | 198        |
| 6 | 14  | 8         | 454.0        | 220        |
| 7 | 14  | 8         | 440.0        | 215        |
| 8 | 14  | 8         | 455.0        | 225        |
| 9 | 15  | 8         | 390.0        | 190        |

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

```
In[2]: np.random.seed(42)
```

```
In[3]: np.arange(0, 10, 1)
```

```
Out[1]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In[4]: np.isnan([0, 1, np.nan, 3])
```

```
Out[2]: array([False, False,  True, False])
```

```
In[5]: np.amin([1, 0, 3, 2])
```

```
Out[3]: 0
```

```
Out[6]: np.argmin([1, 0, 3, 2])
```

```
Out[4]: 1
```

```
In[7]: np.nanmin([1, 0, 3, np.nan])
```

```
Out[5]: 0
```

# Python: statsmodels

```
1 path <- '/Users/esten/Downloads/Auto.csv'
2 data <- read.csv(path)
3
4 model <- lm(mpg ~ cylinders + displacement +
5             horsepower + weight +
6             acceleration + year,
7             data=data)
8 summary(model)
```

Coefficients:

|              | Estimate   | Std. Error | Pr(> t )    |
|--------------|------------|------------|-------------|
| (Intercept)  | -1.454e+01 | 4.764e+00  | 0.00244 **  |
| cylinders    | -3.299e-01 | 3.321e-01  | 0.32122     |
| displacement | 7.678e-03  | 7.358e-03  | 0.29733     |
| horsepower   | -3.914e-04 | 1.384e-02  | 0.97745     |
| weight       | -6.795e-03 | 6.700e-04  | < 2e-16 *** |
| acceleration | 8.527e-02  | 1.020e-01  | 0.40383     |
| year         | 7.534e-01  | 5.262e-02  | < 2e-16 *** |

---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*'  
0.05

```
In[1]: import statsmodels.formula.api as smf

path = '/Users/esten/Downloads/Auto.csv'
df = pd.read_csv(path)

model = smf.ols(
    formula='mpg ~ cylinders + displacement +
            horsepower + weight +
            acceleration + year',
    data=df
)
fit = model.fit()
print(fit.summary())
```

```
Out[1]:
```

|              | coef     | std err | P> t  | [0.025 | 0.975] |
|--------------|----------|---------|-------|--------|--------|
| Intercept    | -14.5353 | 4.764   | 0.002 | -23.90 | -5.16  |
| cylinders    | -0.3299  | 0.332   | 0.321 | -0.98  | 0.32   |
| displacement | 0.0077   | 0.007   | 0.297 | -0.00  | 0.02   |
| horsepower   | -0.0004  | 0.014   | 0.977 | -0.02  | 0.02   |
| weight       | -0.0068  | 0.001   | 0.000 | -0.00  | -0.00  |
| acceleration | 0.0853   | 0.102   | 0.404 | -0.11  | 0.28   |
| year         | 0.7534   | 0.053   | 0.000 | 0.65   | 0.85   |

```
In[1]: from sklearn.linear_model import LinearRegression

path = '/Users/esten/Downloads/Auto.csv'
df = pd.read_csv(path)

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

model = LinearRegression()
model.fit(df[predictors], df[target])
model.summary()
```

```
In[1]: from sklearn.linear_model import LinearRegression

path = '/Users/esten/Downloads/Auto.csv'
df = pd.read_csv(path)

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

model = LinearRegression()
model.fit(df[predictors], df[target])
model.summary()
```

```
Out[1]: -----
AttributeError                                Traceback (most recent call last)
Cell In[52], line 13
     11 model = LinearRegression()
     12 model.fit(df[predictors], df[target])
--> 13 model.summary()

AttributeError: 'LinearRegression' object has no attribute 'summary'
```

```
In[1]: from sklearn.linear_model import LinearRegression

path = '/Users/esten/Downloads/Auto.csv'
df = pd.read_csv(path)

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

model = LinearRegression()
model.fit(df[predictors], df[target])

# Print model coefficients
print(f'Intercept: {model.intercept_}')
print(f'Coefficients: {model.coef_}')

# Print model residuals
predictions = model.predict(df[predictors])
residuals = df[target] - predictions
print(f'Residuals: {residuals.values[:5]}...')
```

```
Out[1]: Intercept: -14.53525048050604
Coefficients: [-3.29859089e-01  7.67843024e-03 -3.91355574e-04 -6.79461791e-03
              8.52732469e-02  7.53367180e-01]
Residuals: [2.91708096  0.92742531  2.46368456  0.46552549  1.71359255]...
```

## APIs of scikit-learn objects

To have a uniform API, we try to have a common basic API for all the objects. In addition, to avoid the proliferation of framework code, we try to adopt simple conventions and limit to a minimum the number of methods an object must implement.

Elements of the scikit-learn API are described more definitively in the [Glossary of Common Terms and API Elements](#).

### Different objects

The main objects in scikit-learn are (one class can implement multiple interfaces):

**Estimator:** The base object, implements a `fit` method to learn from data, either:

```
estimator = estimator.fit(data, targets)
```

or:

```
estimator = estimator.fit(data)
```

**Predictor:** For supervised learning, or some unsupervised problems, implements:

```
prediction = predictor.predict(data)
```

Classification algorithms usually also offer a way to quantify certainty of a prediction, either using `decision_function` or `predict_proba`:

```
probability = predictor.predict_proba(data)
```

**Transformer:** For filtering or modifying the data, in a supervised or unsupervised way, implements:

```
new_data = transformer.transform(data)
```

When fitting and transforming can be performed much more efficiently together than separately, implements:

```
new_data = transformer.fit_transform(data)
```

**Model:** A model that can give a *goodness of fit* measure or a likelihood of unseen data, implements (higher is better):

```
score = model.score(data)
```

<https://scikit-learn.org/stable/developers/develop.html>



```
In[1]: from sklearn.linear_model import LinearRegression

path = '/Users/esten/Downloads/Auto.csv'
df = pd.read_csv(path)

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

model = LinearRegression()
model.fit(df[predictors], df[target])

# Print model coefficients
print(f'Intercept: {model.intercept_}')
print(f'Coefficients: {model.coef_}')

# Print model residuals
predictions = model.predict(df[predictors])
residuals = df[target] - predictions
print(f'Residuals: {residuals.values[:5]}...')
```

```
Out[1]: Intercept: -14.53525048050604
Coefficients: [-3.29859089e-01  7.67843024e-03 -3.91355574e-04 -6.79461791e-03
               8.52732469e-02  7.53367180e-01]
Residuals: [2.91708096  0.92742531  2.46368456  0.46552549  1.71359255]...
```

```
In[1]: from sklearn.svm import SVR

path = '/Users/esten/Downloads/Auto.csv'
df = pd.read_csv(path)

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

model = SVR(kernel='linear')
model.fit(df[predictors], df[target])

# Print model coefficients
print(f'Intercept: {model.intercept_}')
print(f'Coefficients: {model.coef_}')

# Print model residuals
predictions = model.predict(df[predictors])
residuals = df[target] - predictions
print(f'Residuals: {residuals.values[:5]}...')
```

```
Out[1]: Intercept: [-35.38646279]
Coefficients: [[-1.0526357  0.05910105 -0.03667206 -0.00831565  0.56218046
  0.96851648]]
Residuals: [3.0266171  0.62154228 3.10666275 1.34695011 3.07475274]...
```

# 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']]
```

# Getting 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 = []
```

Setup

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

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

Selection

```
        # 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)
```

Modelling

```
        # 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']]
```

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(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_model(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']]
```

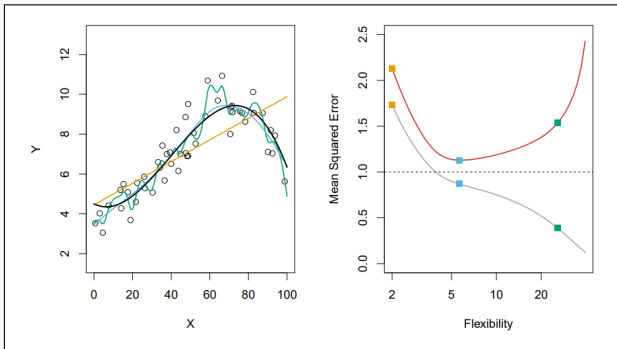
Modelling

## Regularization

---

$$y \sim \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \beta_3 * x_3$$

# Regularization: Motivation





$$y \sim \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \beta_3 * x_3$$

# Regularization: Out-of-sample testing

```
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
  - a. Principal Component Regression
  - b. Partial Least Squares

## Variable selection

---

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  $y$ .

# Variable selection: Outline

## 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  $y$ .

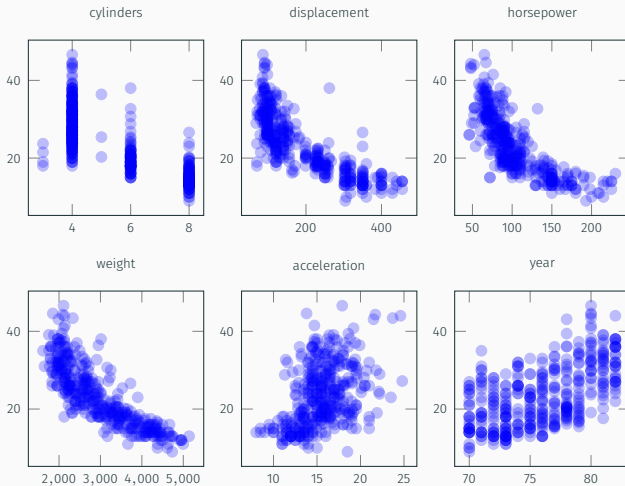
## Motivation

1. Reduce model complexity (overfitting)
2. Simplify interpretation

# Variable selection: Outline

## 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  $y$ .





## 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

## 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

```
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

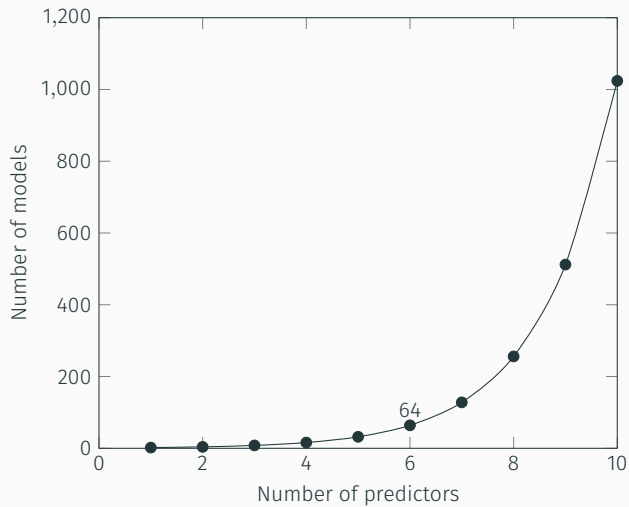
Guaranteed to find the optimal solution.

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.

|  |
|--|
| $\begin{aligned} y &\sim 1 \\ mse &= 146.47 \end{aligned}$ |
|--|

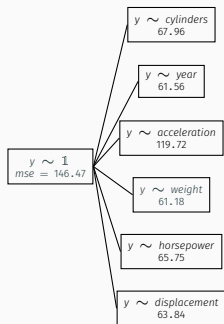
# Variable selection: Forward stepwise selection

## 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.



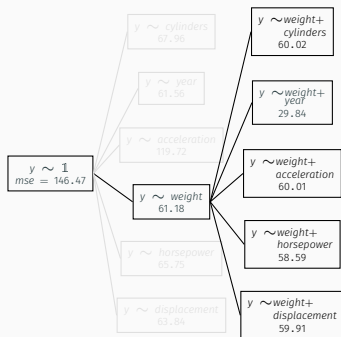
# Variable selection: Forward stepwise selection

## 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.





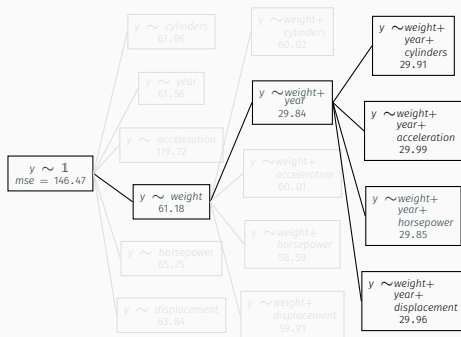
# Variable selection: Forward stepwise selection

## 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.



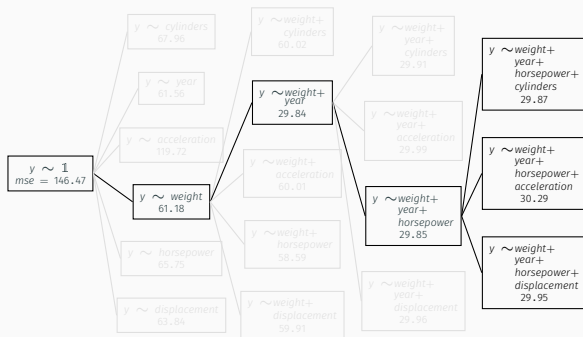
# Variable selection: Forward stepwise selection

## 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.



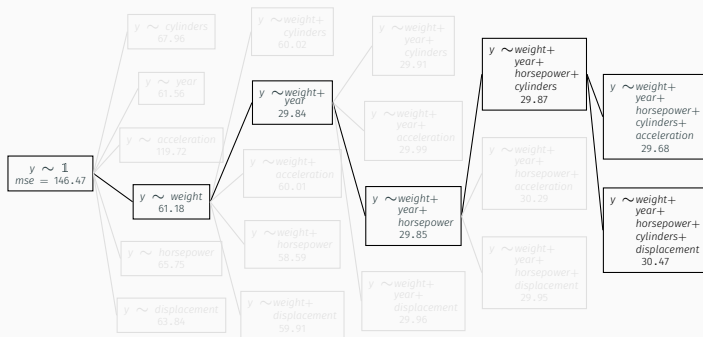
# Variable selection: Forward stepwise selection

## 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.



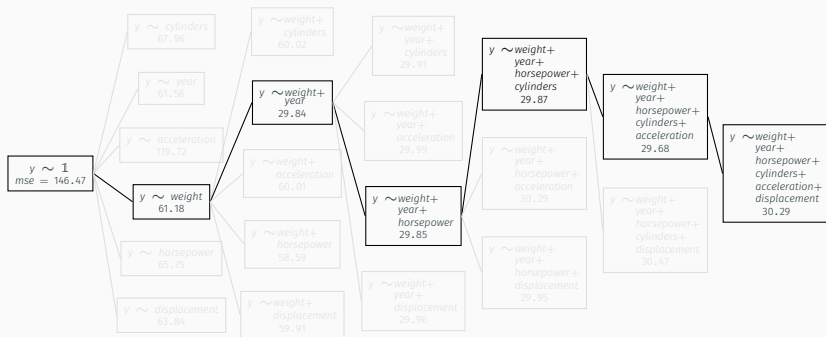
# Variable selection: Forward stepwise selection

## Problem

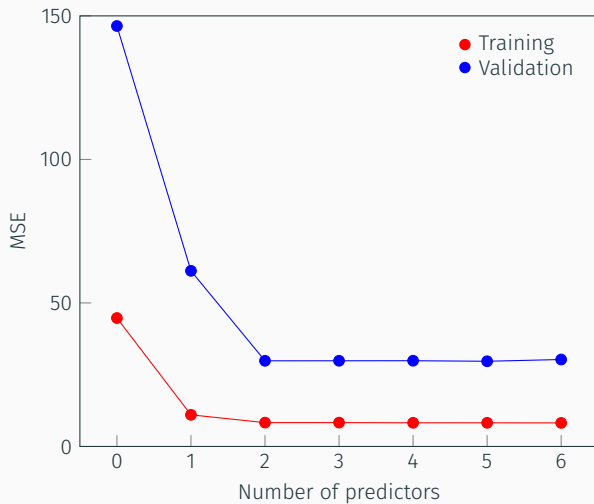
We have a set of predictors  $P = \{x_0, x_1, \dots\}$  and a target variable  $y$ , and we want to find the subset  $p \subseteq P$  that yields the best (linear) model for predicting  $y$ .

## Solution

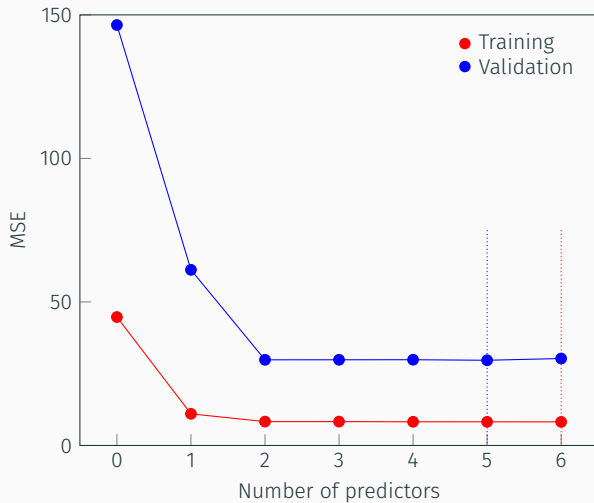
Start with no predictors. Iteratively add the predictor that yields the best model until all are included.



## Variable selection: Forward stepwise selection



## Variable selection: Forward stepwise selection



# 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

```
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

---

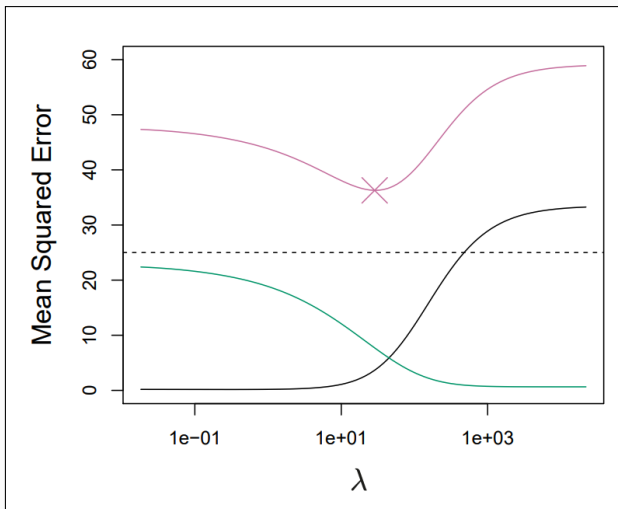
$$y \sim \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6$$

```
Out[1]:
```

|              | coef     | std err | P> t  | [0.025 | 0.975] |
|--------------|----------|---------|-------|--------|--------|
| Intercept    | -14.5353 | 4.764   | 0.002 | -23.90 | -5.16  |
| cylinders    | -0.3299  | 0.332   | 0.321 | -0.98  | 0.32   |
| displacement | 0.0077   | 0.007   | 0.297 | -0.00  | 0.02   |
| horsepower   | -0.0004  | 0.014   | 0.977 | -0.02  | 0.02   |
| weight       | -0.0068  | 0.001   | 0.000 | -0.00  | -0.00  |
| acceleration | 0.0853   | 0.102   | 0.404 | -0.11  | 0.28   |
| year         | 0.7534   | 0.053   | 0.000 | 0.65   | 0.85   |

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

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



$$\text{salary} \sim \beta_0 + \beta_1 * \text{age}$$



$$salary \sim \beta_0 + \beta_1 * age$$

$$salary \sim 300000 + 10000 * age$$

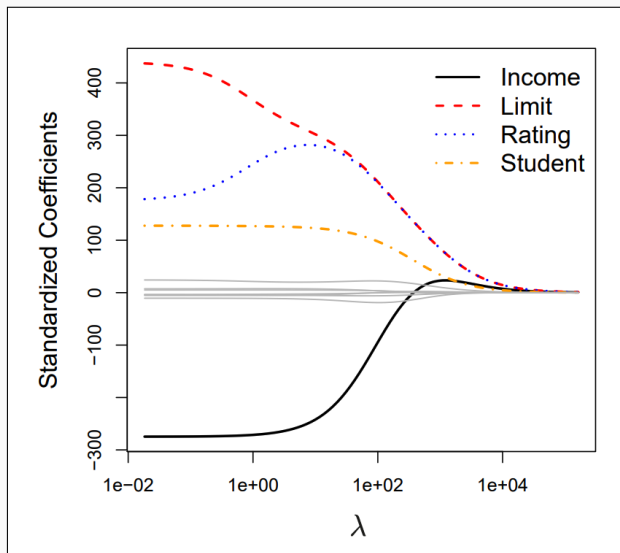
$$salary \sim 600000 + 0 * age$$

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

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

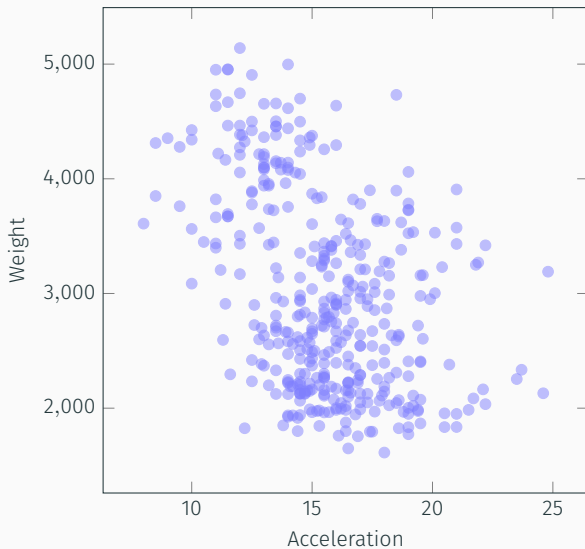
$$\text{loss}_{\text{ridge}} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^p \beta_j^2$$

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



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

## Shrinkage: Feature standardization



z-score standardization



## z-score standardization

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

## z-score standardization

$$X = \frac{x - \mu_X}{\sigma_X^2}$$

```
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})')
```

## z-score standardization

$$X = \frac{x - \mu_x}{\sigma_x^2}$$

```
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)
weight: 2977.58 (848.32)
acceleration: 15.54 (2.76)
year: 75.98 (3.68)
cylinders after: -0.00 (1.00)
displacement after: -0.00 (1.00)
horsepower after: -0.00 (1.00)
weight after: -0.00 (1.00)
acceleration after: 0.00 (1.00)
year after: -0.00 (1.00)
```

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

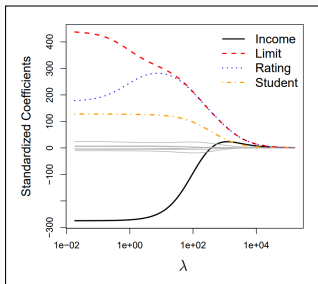
$$loss_{ridge} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^p \beta_j^2$$

Regularization through shrinking the model covariates towards zero.

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

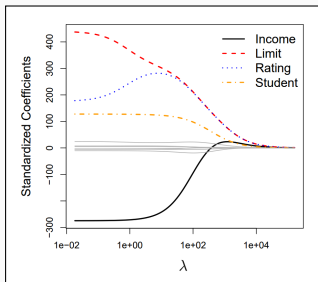
$$loss_{lasso} = \sum_{i=0}^n \left( y_i - \sum_{j=0}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^p |\beta_j|$$

Ridge

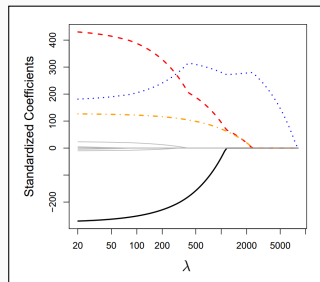


# Shrinkage: LASSO

Ridge



LASSO



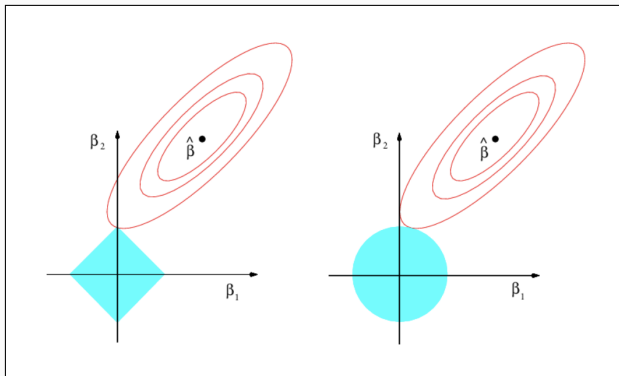


| Predictor    | Ridge | LASSO |
|--------------|-------|-------|
| Intercept    | 23.44 | 23.44 |
| Weight       | -5.59 | -4.78 |
| Year         | 2.75  | 2.00  |
| Horsepower   | -0.07 | -0.09 |
| Cylinders    | -0.54 | 0     |
| Acceleration | 0.19  | 0     |
| Displacement | 0.66  | 0     |

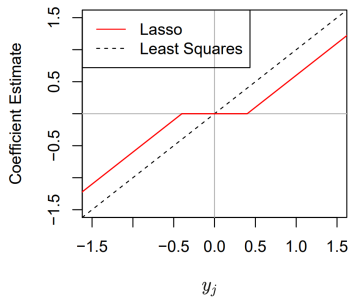
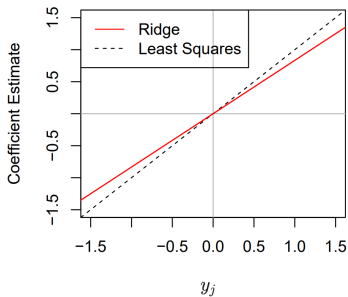
| Predictor    | Ridge | LASSO |
|--------------|-------|-------|
| Intercept    | 23.44 | 23.44 |
| Weight       | -5.59 | -4.78 |
| Year         | 2.75  | 2.00  |
| Horsepower   | -0.07 | -0.09 |
| Cylinders    | -0.54 | 0     |
| Acceleration | 0.19  | 0     |
| Displacement | 0.66  | 0     |

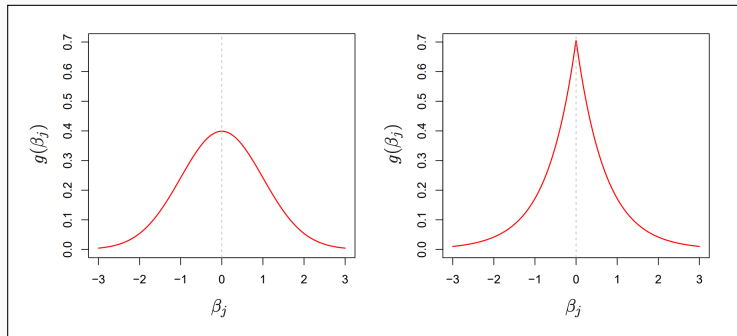
A coefficient of 0 does not mean the predictor has no predictive value for the outcome!

# Shrinkage: LASSO



Whiteboard! 🤖





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

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.

## Shrinkage: Summary

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

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**.

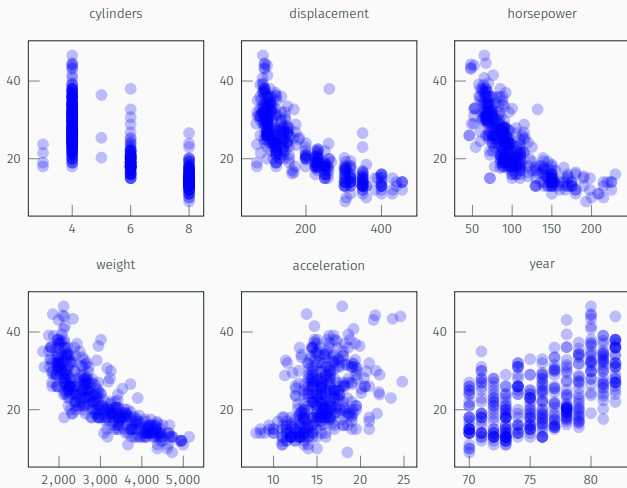


## Dimensionality reduction

---

Although we have  $p$  predictors, there are actually  $q < p$  dimensions of variability in our data, and using  $q$  instead of  $p$  is going to reduce model complexity.

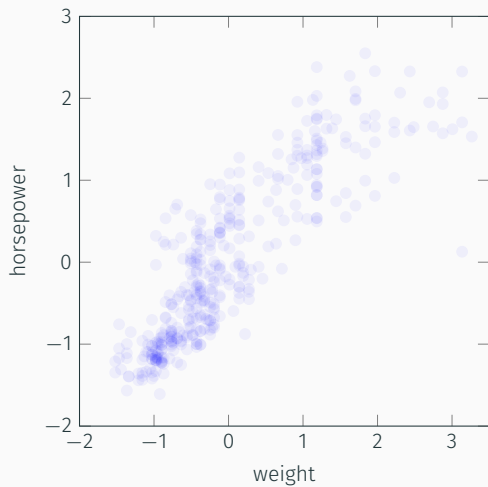
# Dimensionality reduction: Outline



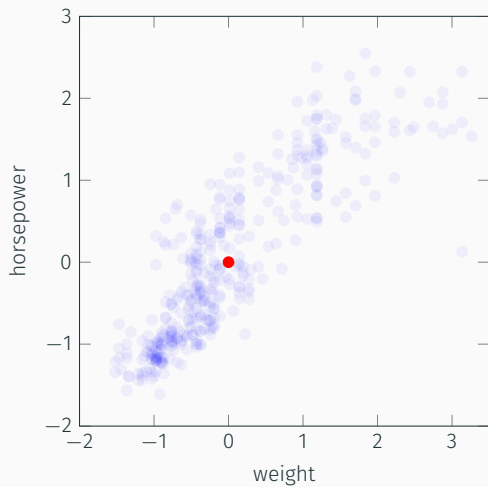
## Dimensionality reduction: Principal component analysis

|      |      |      |      |      |      |
|------|------|------|------|------|------|
| 1    | 0.30 | 0.86 | 0.89 | 0.41 | 0.93 |
| 0.30 | 1    | 0.41 | 0.34 | 0.29 | 0.36 |
| 0.86 | 0.41 | 1    | 0.84 | 0.68 | 0.89 |
| 0.89 | 0.34 | 0.84 | 1    | 0.50 | 0.95 |
| 0.41 | 0.29 | 0.68 | 0.50 | 1    | 0.54 |
| 0.93 | 0.36 | 0.89 | 0.95 | 0.54 | 1    |

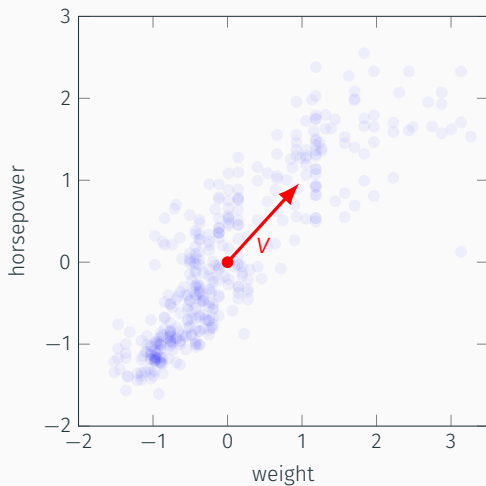
## Dimensionality reduction: Principal component analysis



## Dimensionality reduction: Principal component analysis

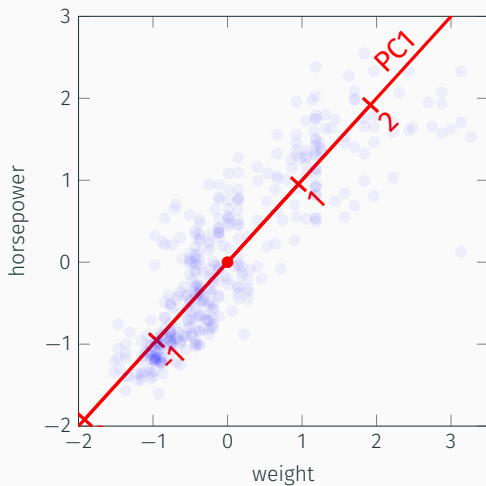


## Dimensionality reduction: Principal component analysis



$v \implies$  direction of greatest variance in  $X$

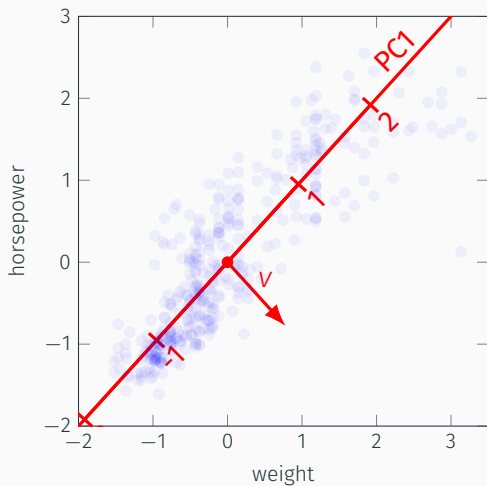
## Dimensionality reduction: Principal component analysis



$$PC1 = 0.69 * horsepower + 0.71 * weight$$

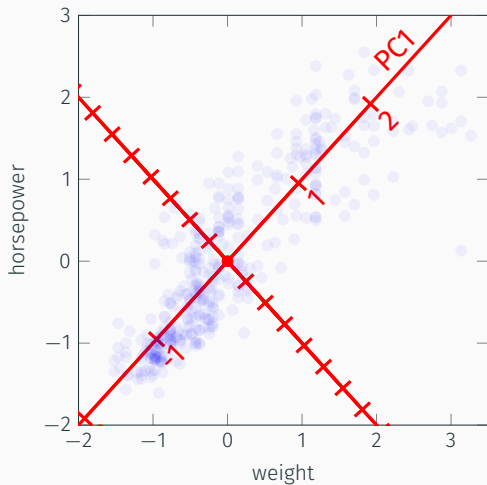


## Dimensionality reduction: Principal component analysis



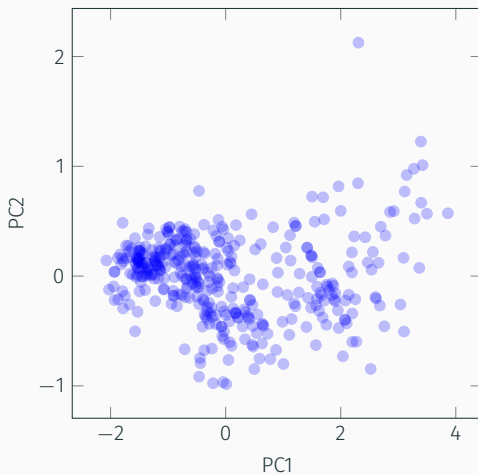
$v \implies$  direction of greatest variance in  $X$  **after regressing out PC1**

## Dimensionality reduction: Principal component analysis



$$PC2 = 0.69 * horsepower + 0.71 * weight$$

## Dimensionality reduction: Principal component analysis



## Dimensionality reduction: Principal component regression

| mpg | horsepower | weight | PC1   | PC2   |
|-----|------------|--------|-------|-------|
| 18  | 130        | 3504   | 0.908 | 0.303 |
| 15  | 165        | 3693   | 1.709 | 0.517 |
| 18  | 150        | 3436   | 1.219 | 0.455 |
| 16  | 150        | 3433   | 1.217 | 0.457 |
| 17  | 140        | 3449   | 1.046 | 0.260 |
| 15  | 198        | 4341   | 2.856 | 0.583 |
| 14  | 220        | 4354   | 3.272 | 0.977 |

## Dimensionality reduction: Principal component regression

| mpg | horsepower | weight | PC1   | PC2   |
|-----|------------|--------|-------|-------|
| 18  | 130        | 3504   | 0.908 | 0.303 |
| 15  | 165        | 3693   | 1.709 | 0.517 |
| 18  | 150        | 3436   | 1.219 | 0.455 |
| 16  | 150        | 3433   | 1.217 | 0.457 |
| 17  | 140        | 3449   | 1.046 | 0.260 |
| 15  | 198        | 4341   | 2.856 | 0.583 |
| 14  | 220        | 4354   | 3.272 | 0.977 |

$$mpg \sim \beta_0 + \beta_1 * horsepower + \beta_2 * weight$$

## Dimensionality reduction: Principal component regression

| mpg | horsepower | weight | PC1   | PC2   |
|-----|------------|--------|-------|-------|
| 18  | 130        | 3504   | 0.908 | 0.303 |
| 15  | 165        | 3693   | 1.709 | 0.517 |
| 18  | 150        | 3436   | 1.219 | 0.455 |
| 16  | 150        | 3433   | 1.217 | 0.457 |
| 17  | 140        | 3449   | 1.046 | 0.260 |
| 15  | 198        | 4341   | 2.856 | 0.583 |
| 14  | 220        | 4354   | 3.272 | 0.977 |

$$mpg \sim \beta_0 + \beta_1 * horsepower + \beta_2 * weight$$

$$mpg \sim \beta_0 + \beta_1 * PC1 + \beta_2 * PC2$$

## Principal component regression

1. Fit a PCA to transform your  $p$  predictors into  $p$  principal components.
2. Fit a linear regression model using a subset of the principal components.

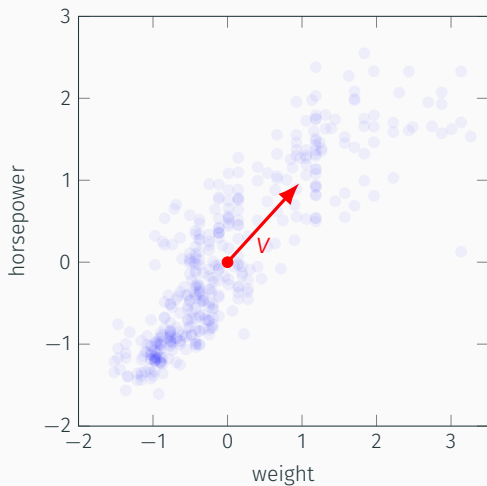
### + Positives

- The principal components are orthogonal, so there is no issue of collinearity.
- The principal components are ordered by the amount of variance they explain, so we can the early principal components are (probably) the best predictors.

### - Drawbacks

- Have to select number of principal components to use.
- What if the principal components that explain the largest amount of variance are not related to the outcome?

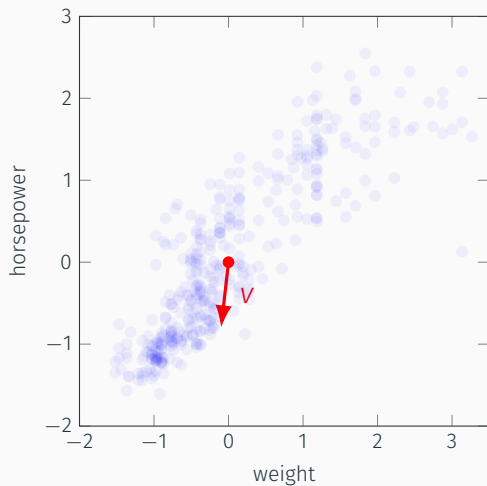
## Dimensionality reduction: Partial least squares



$v \implies$  direction of greatest variance in  $X$



## Dimensionality reduction: Partial least squares



$v \implies$  direction of greatest covariance between  $X$  and  $Y$