PSY9511: Seminar 3

Variable selection and regularization

Esten H. Leonardsen

07.09.23

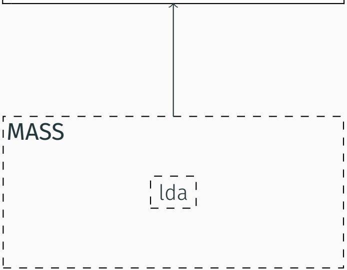
Outline

1. Introduction
    - Python
    - Coding tips: Separation of concerns
2. Variable selection
    - Best subset selection
    - Forward stepwise selection
    - Backward stepwise selection
3. Regularization
    - Ridge regression
    - Lasso
    - Elastic net
4. Dimensionality reduction
    - Principal component regression
    - Partial least squares

Python: Imports

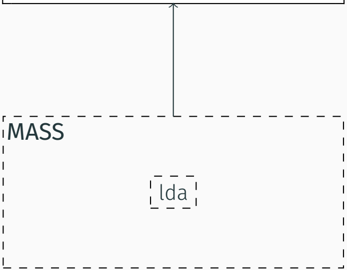
Python: Imports

```r
library(MASS)
```



Python: Imports

```r
library(MASS)

lda_fit <- lda(display ~ age + fb,
               data = display)
```



Python: Imports

```r
library(MASS)

lda_fit <- lda(display ~ age + fb,
               data = display)
```

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

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

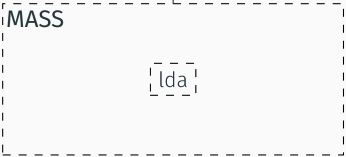
Python: Imports

```r
library(MASS)

lda_fit <- lda(display ~ age + fb,
               data = display)
```

In[1]:
```python
import sklearn

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

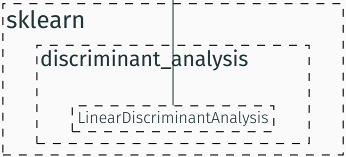
Python: Imports

```r
library(MASS)

lda_fit <- lda(display ~ age + fb,
               data = display)
```



In[1]:

```python
from sklearn.discriminant_analysis \
    import LinearDiscriminantAnalysis

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



Python: pandas

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

In[1]:

```python
import pandas as pd

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

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

Python: numpy

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

```r
path <- '/Users/esten/Downloads/Auto.csv'
data <- read.csv(path)

model <- lm(mpg ~ cylinders + displacement +
                  horsepower + weight +
                  acceleration + year,
            data=data)
summary(model)
```

In[1]:

```python
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())
```

```
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
```

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
==============================================
```

Python: scikit-learn

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()
```

Python: scikit-learn

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'
```

Python: scikit-learn

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]...
```

Python: scikit-learn

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

Python: scikit-learn

In[1]:

```python
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]...
```

Python: scikit-learn

In[1]:

```python
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']]
```

Coding tips: Separation of concerns

In[1]:

Setup

```
# 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 = []
```

Selection

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

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

Modelling

```
        # 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}
```

Housekeeping

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

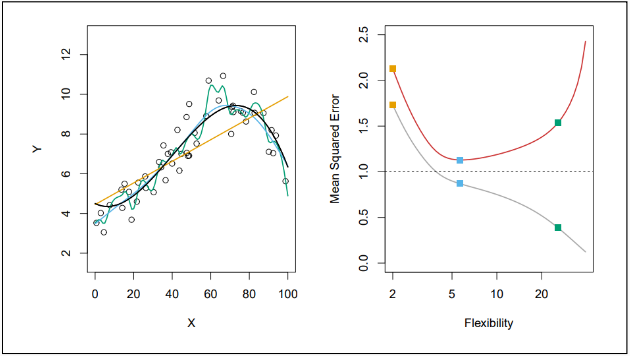
Coding tips: Separation of concerns

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 = []

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 = LinearRegression()
    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: Motivation



Regularization: Out-of-sample testing

Regularization: Methods

1. Variable selection
    a. Best subset selection
    b. Forward stepwise selection
    c. Backward stepwise selection
2. Shrinkage
    a. LASSO
    b. Ridge Regression
    c. Elastic net
3. Dimensionality reduction
    a. Principal Component Regression
    b. Partial Least Squares

Variable selection: Outline

Problem

We have a set of predictors $P = \{x_0, x_1, ...\}$ 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 $P = \{x_0, x_1, ...\}$ 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. Simplify interpretation
2. Reduce model complexity (overfitting)

Problem

We have a set of predictors $P = \{x_0, x_1, ...\}$ 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 $P = \{x_0, x_1, ...\}$ 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 (best subset selection)

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

Variable selection: Best subset selection

Problem

We have a set of predictors $P = \{x_0, x_1, ...\}$ 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 (best subset selection)

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

Variable selection: Best subset selection