Chap6-Exercises_LuisCorreia-745724_v1

September 20, 2025

1 MAP5935 - Statistical Learning (Chapter 6 - Linear Model Selection and Regilarization)

Prof. Christian Jäkel

https://www.statlearning.com/

[30]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import subplots
import statsmodels.api as sm

from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer, make_column_selector as selector
from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import LassoCV
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression

from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

1.1 *Conceptual Exercises*

1.1.1 (5) It is well-known that ridge regression tends to give similar coefficient values to correlated variables, whereas the lasso may give quite different coefficient values to correlated variables. We will now explore this property in a very simple setting.

1.1.2 Suppose that $n = 2$, $p = 2$, $x_{11} = x_{12}$, $x_{21} = x_{22}$. Furthermore, suppose that $y1 + y2 = 0$ and $x_{11} + x_{21} = 0$ and $x_{12} + x_{22} = 0$, so that the estimate for the intercept in a least squares, ridge regression, or lasso model is zero: $\hat{\beta}_0 = 0$.

1.1.3 (a) Write out the ridge regression optimization problem in this setting.

Solution: Supposing that $n = 2$, $p = 2$, and

$$x_{11} = x_{12}, \qquad x_{21} = x_{22}.$$

… and that:

$$y_1 + y_2 = 0, \qquad x_{11} + x_{21} = 0, \qquad x_{12} + x_{22} = 0,$$

… with the intercept in least squares, ridge, or lasso is zero: $\hat{\beta}_0 = 0$.

In this sense, we have *ridge regression* problem by solving the minimization problem considering the equation, as seen in this chapter:

$$\min_{\hat{\beta}_1, \hat{\beta}_2} \sum_{i=1}^{2} \left(y_i - x_{i1}\hat{\beta}_1 - x_{i2}\hat{\beta}_2 \right)^2 \; + \; \lambda \left(\hat{\beta}_1^2 + \hat{\beta}_2^2 \right).$$

with λ being the regularization parameter.

Using the facts $x_{11} = x_{12}$ and $x_{21} = x_{22}$, this can rewrite the optimization problem as follows:

$$\boxed{\min_{\hat{\beta}_1, \hat{\beta}_2} \; \left(y_1 - x_{11}(\hat{\beta}_1 + \hat{\beta}_2) \right)^2 + \left(y_2 - x_{21}(\hat{\beta}_1 + \hat{\beta}_2) \right)^2 \; + \; \lambda \left(\hat{\beta}_1^2 + \hat{\beta}_2^2 \right).}$$

1.1.4 (b) Argue that in this setting, the ridge coefficient estimates satisfy $\hat{\beta}_1 = \hat{\beta}_2$.

Solution: From part (a), with $x_{11} = x_{12}$ and $x_{21} = x_{22}$, the ridge objective is

$$L(\hat{\beta}_1, \hat{\beta}_2) = \left(y_1 - x_{11}(\hat{\beta}_1 + \hat{\beta}_2)\right)^2 + \left(y_2 - x_{21}(\hat{\beta}_1 + \hat{\beta}_2)\right)^2 + \lambda(\hat{\beta}_1^2 + \hat{\beta}_2^2).$$

The data-fit term depends only on the sum $s = \hat{\beta}_1 + \hat{\beta}_2$.

For any $(\hat{\beta}_1, \hat{\beta}_2)$, let $m = (\hat{\beta}_1 + \hat{\beta}_2)/2$ and $d = (\hat{\beta}_1 - \hat{\beta}_2)/2$.

Replacing $(\hat{\beta}_1, \hat{\beta}_2)$ by (m, m) keeps s and the fit unchanged, but changes the penalty to

$$\hat{\beta}_1^2 + \hat{\beta}_2^2 = (m + d)^2 + (m - d)^2.$$

Expanding,

$$= 2m^2 + 2d^2 = 2m^2 + \frac{(\hat{\beta}_1 - \hat{\beta}_2)^2}{2}.$$

Since $\frac{(\hat{\beta}_1-\hat{\beta}_2)^2}{2} \geq 0$, we have

$$\hat{\beta}_1^2 + \hat{\beta}_2^2 \ \geq \ 2m^2,$$

and, when equality holds $\implies \hat{\beta}_1 = \hat{\beta}_2$.

Thus, the ridge penalty is minimized when $\hat{\beta}_1 = \hat{\beta}_2$.

Since the loss part depends only on s, the overall minimizer must satisfy

$$\boxed{\hat{\beta}_1 = \hat{\beta}_2}.$$

1.1.5 (c) Write out the lasso optimization problem in this setting.

Solution: With $\hat{\beta}_0 = 0$, $x_{11} = x_{12}$, and $x_{21} = x_{22}$, the *lasso objective* is

$$\min_{\beta_1,\beta_2} \sum_{i=1}^{2} \Big(y_i - x_{i1}\beta_1 - x_{i2}\beta_2\Big)^2 \ + \ \lambda(|\beta_1| + |\beta_2|).$$

Using $x_{11} = x_{12}$ and $x_{21} = x_{22}$, this can be written equivalently as

$$\boxed{\min_{\beta_1,\beta_2} \ \Big(y_1 - x_{11}(\beta_1+\beta_2)\Big)^2 + \Big(y_2 - x_{21}(\beta_1+\beta_2)\Big)^2 \ + \ \lambda(|\beta_1| + |\beta_2|).}$$

1.1.6 (d) Argue that in this setting, the lasso coefficients $\hat{\beta}_1$ and $\hat{\beta}_2$ are not unique—in other words, there are many possible solutions to the optimization problem in (c). Describe these solutions.

Solution: From (c), with $x_{11} = x_{12}$ and $x_{21} = x_{22}$,

$$\min_{\beta_1,\beta_2} \ \Big(y_1 - x_{11}(\beta_1+\beta_2)\Big)^2 + \Big(y_2 - x_{21}(\beta_1+\beta_2)\Big)^2 + \lambda(|\beta_1| + |\beta_2|).$$

Let $s = \beta_1 + \beta_2$. The data-fit term depends **only** on s, so the problem reduces to

$$\min_{s} \ \underbrace{\sum_{i=1}^{2} \big(y_i - z_i s\big)^2}_{\mathrm{RSS}(s)} + \lambda \ \underbrace{\min_{\beta_1+\beta_2=s} \big(|\beta_1| + |\beta_2|\big)}_{\phi(s)},$$

where $z_i = x_{i1} + x_{i2}$ (here $z_i = 2x_{i1}$).

For a fixed sum s, the inner minimization

$$\phi(s) = \min_{\beta_1+\beta_2=s} \big(|\beta_1| + |\beta_2|\big)$$

By the triangle inequality (see reference below) we have:

$$\phi(s) = |s| \quad \text{and it is attained by every pair} \ \begin{cases} \beta_1 \in [0,s], \ \ \beta_2 = s - \beta_1, & \text{if } s \geq 0, \\ \beta_1 \in [s,0], \ \ \beta_2 = s - \beta_1, & \text{if } s \leq 0. \end{cases}$$

i.e., by any split of s between β_1 and β_2 that keeps them with the **same sign as** s (zeros allowed).

Indeed, for $s \geq 0$, $|\beta_1| + |\beta_2| = |\beta_1| + |s - \beta_1| = s$ for all $\beta_1 \in [0, s]$; outside this interval the value is strictly larger than s. An analogous statement holds for $s \leq 0$.

Therefore the full problem is equivalent to the **univariate lasso** in s:

$$\min_s \ \mathrm{RSS}(s) + \lambda|s|.$$

Let supose exists $s^\star$ being its (unique) minimizer (obtained by soft-thresholding the OLS estimate for s).

Then **every** pair $(\hat{\beta}_1, \hat{\beta}_2)$ satisfying

$$\hat{\beta}_1 + \hat{\beta}_2 = s^\star \quad \text{and} \quad \hat{\beta}_1, \hat{\beta}_2 \text{ have the same sign as } s^\star$$

is a lasso solution. Geometrically, the solution set is the **entire line segment**

$$\{(\hat{\beta}_1, \hat{\beta}_2) : \ (\hat{\beta}_1, \hat{\beta}_2) = (t, \ s^\star - t), \ t \in [0, s^\star]\} \quad \text{if } s^\star \geq 0,$$

and the segment $\{t \in [s^\star, 0]\}$ if $s^\star \leq 0$.

(If $s^\star = 0$, the unique solution is $\hat{\beta}_1 = \hat{\beta}_2 = 0$.)

Hence, in this setting the lasso coefficients are **not unique** whenever $s^\star \neq 0$; there are infinitely many optimal splits of the common effect across the two perfectly collinear predictors.

References

- Boyd, S., & Vandenberghe, L. (2004). *Convex Optimization.* Cambridge University Press (page 634)

1.2 *Applied Exercises*

1.2.1 (9) In this exercise, we will predict the number of applications received using the other variables in the `College` data set.

1.2.2 (a) Split the data set into a training set and a test set.

[31]:
```
College = pd.read_csv("../Data/College.csv")
```

[32]:
```
College.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 777 entries, 0 to 776
Data columns (total 19 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Unnamed: 0   777 non-null    object
 1   Private      777 non-null    object
 2   Apps         777 non-null    int64
 3   Accept       777 non-null    int64
 4   Enroll       777 non-null    int64
```

```
 5   Top10perc    777 non-null    int64
 6   Top25perc    777 non-null    int64
 7   F.Undergrad  777 non-null    int64
 8   P.Undergrad  777 non-null    int64
 9   Outstate     777 non-null    int64
 10  Room.Board   777 non-null    int64
 11  Books        777 non-null    int64
 12  Personal     777 non-null    int64
 13  PhD          777 non-null    int64
 14  Terminal     777 non-null    int64
 15  S.F.Ratio    777 non-null    float64
 16  perc.alumni  777 non-null    int64
 17  Expend       777 non-null    int64
 18  Grad.Rate    777 non-null    int64
dtypes: float64(1), int64(16), object(2)
memory usage: 115.5+ KB
```

[33]:
```python
College.head()
```

[33]:
```
                     Unnamed: 0 Private  Apps  Accept  Enroll  Top10perc  \
0  Abilene Christian University     Yes  1660    1232     721         23
1            Adelphi University     Yes  2186    1924     512         16
2                Adrian College     Yes  1428    1097     336         22
3           Agnes Scott College     Yes   417     349     137         60
4     Alaska Pacific University     Yes   193     146      55         16

   Top25perc  F.Undergrad  P.Undergrad  Outstate  Room.Board  Books  Personal  \
0         52         2885          537      7440        3300    450      2200
1         29         2683         1227     12280        6450    750      1500
2         50         1036           99     11250        3750    400      1165
3         89          510           63     12960        5450    450       875
4         44          249          869      7560        4120    800      1500

   PhD  Terminal  S.F.Ratio  perc.alumni  Expend  Grad.Rate
0   70        78       18.1           12    7041         60
1   29        30       12.2           16   10527         56
2   53        66       12.9           30    8735         54
3   92        97        7.7           37   19016         59
4   76        72       11.9            2   10922         15
```

We'll fit a logistic regression using **income** and **balance** to predict **default**. The response needs to be binary, so we map "Yes"→1, "No"→0.

[34]:
```python
# 1) Get the DataFrame (works whether it's named `college` or `College`)
df = College.copy()

# 2) Drop non-informative ID column if present
if "Unnamed: 0" in df.columns:
```

```python
    df = df.drop(columns=["Unnamed: 0"])

# 3) Define target and features
y = df["Apps"]
X = df.drop(columns=["Apps"])

# 4) Optional stratification by the `Private` flag (helps keep balance)
stratifier = X["Private"] if "Private" in X.columns else None

# 5) Split: 80% train / 20% test
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.20,
    random_state=42,
    stratify=stratifier
)

# 6) Quick checks
print("Shapes:")
print("  X_train:", X_train.shape, "  y_train:", y_train.shape)
print("  X_test :", X_test.shape,  "  y_test :", y_test.shape)

if "Private" in X_train.columns:
    print("\nPrivate proportion (train):")
    print(X_train["Private"].value_counts(normalize=True).round(3).to_string())
    print("\nPrivate proportion (test):")
    print(X_test["Private"].value_counts(normalize=True).round(3).to_string())
```

```
Shapes:
  X_train: (621, 17)   y_train: (621,)
  X_test : (156, 17)   y_test : (156,)

Private proportion (train):
Private
Yes    0.728
No     0.272

Private proportion (test):
Private
Yes    0.724
No     0.276
```

1.2.3 (b) Fit a linear model using least squares on the training set, and report the test error obtained.

[35]:
```python
# Reuse X_train, X_test, y_train, y_test created in part (a)

# Preprocessor: one-hot encode object columns; passthrough numerics
preprocess = ColumnTransformer(
    transformers=[
        ("cat", OneHotEncoder(drop="first", handle_unknown="ignore"),
         selector(dtype_include=object)),
        ("num", "passthrough",
         selector(dtype_include=["int64", "float64"]))
    ],
    remainder="drop"
)

# Pipeline: preprocessing + OLS
ols = Pipeline(steps=[
    ("preprocess", preprocess),
    ("model", LinearRegression())
])

# Fit on training data
ols.fit(X_train, y_train)

# Predict on test data
y_pred = ols.predict(X_test)

# Metrics
test_mse  = mean_squared_error(y_test, y_pred)
test_rmse = np.sqrt(test_mse)
test_r2   = r2_score(y_test, y_pred)
test_mae  = mean_absolute_error(y_test, y_pred)

print('\nLinear Model:\n-------------')
print(f"Test MSE : {test_mse:,.2f}")
print(f"Test RMSE: {test_rmse:,.2f}")
print(f"Test MAE : {test_mae:,.2f}")
print(f"Test R^2 : {test_r2:,.3f}")
```

```
Linear Model:
-------------
Test MSE : 1,085,881.76
Test RMSE: 1,042.06
Test MAE : 646.07
Test R^2 : 0.912
```

1.2.4 (c) Fit a ridge regression model on the training set, with λ chosen by cross-validation. Report the test error obtained.

[36]:
```
alphas = np.logspace(-4, 4, 81)
cv = KFold(n_splits=10, shuffle=True, random_state=42)

ridge_cv = RidgeCV(
    alphas=alphas,
    cv=cv,
    scoring="neg_mean_squared_error"  # keep your metric
)

ridge = Pipeline(steps=[
    ("preprocess", preprocess),
    ("scale", StandardScaler(with_mean=False)),
    ("model", ridge_cv)
])

ridge.fit(X_train, y_train)
best_alpha = ridge.named_steps["model"].alpha_
y_pred = ridge.predict(X_test)

test_mse  = mean_squared_error(y_test, y_pred)
test_rmse = np.sqrt(test_mse)
test_r2   = r2_score(y_test, y_pred)
test_mae  = mean_absolute_error(y_test, y_pred)

print('\nRidge Regression:\n-----------------')
print(f"Selected lambda (alpha): {best_alpha:.6g}")
print(f"Test MSE : {test_mse:,.2f}")
print(f"Test RMSE: {test_rmse:,.2f}")
print(f"Test MAE : {test_mae:,.2f}")
print(f"Test R^2 : {test_r2:,.3f}")
```

```
Ridge Regression:
-----------------
Selected lambda (alpha): 0.0001
Test MSE : 1,085,880.52
Test RMSE: 1,042.06
Test MAE : 646.07
Test R^2 : 0.912
```

1.2.5 (d) Fit a lasso model on the training set, with λ chosen by crossvalidation. Report the test error obtained, along with the number of non-zero coefficient estimates.

[37]:
```
# After concatenation, scale all features (now dense)
scaler = StandardScaler(with_mean=True)

# Alpha (lambda) grid on log-scale (covers near-OLS to strong shrinkage)
alphas = np.logspace(-4, 2, 61)  # 1e-4 … 1e2

cv = KFold(n_splits=10, shuffle=True, random_state=42)

lasso_cv = LassoCV(
    alphas=alphas,
    cv=cv,
    max_iter=20000,
    n_jobs=-1,
    random_state=42,   # used if selection='random'; harmless otherwise
)

lasso = Pipeline(steps=[
    ("preprocess", preprocess),
    ("scale", scaler),
    ("model", lasso_cv)
])

# Fit
lasso.fit(X_train, y_train)

# Selected lambda (alpha)
best_alpha = lasso.named_steps["model"].alpha_

# Predict on test
y_pred = lasso.predict(X_test)

# Metrics
test_mse  = mean_squared_error(y_test, y_pred)
test_rmse = np.sqrt(test_mse)
test_r2   = r2_score(y_test, y_pred)
test_mae  = mean_absolute_error(y_test, y_pred)

# Sparsity: number of non-zero coefficients (tolerate tiny numerical noise)
coefs = lasso.named_steps["model"].coef_
nnz = int(np.sum(np.abs(coefs) > 1e-8))
p_total = coefs.size

print('\nLasso Regression:\n-----------------')
print(f"Selected lambda (alpha): {best_alpha:.6g}")
```

```python
print(f"Test MSE : {test_mse:,.2f}")
print(f"Test RMSE: {test_rmse:,.2f}")
print(f"Test MAE : {test_mae:,.2f}")
print(f"Test R^2 : {test_r2:,.3f}")
print(f"Non-zero coefficients: {nnz} out of {p_total}")
```

```
Lasso Regression:
-----------------
Selected lambda (alpha): 10
Test MSE : 1,058,263.23
Test RMSE: 1,028.72
Test MAE : 630.88
Test R^2 : 0.915
Non-zero coefficients: 14 out of 17
```

1.2.6 (e) Fit a PCR model on the training set, with M chosen by crossvalidation. Report the test error obtained, along with the value of M selected by cross-validation.

[38]:
```python
Xtr_proc = preprocess.fit_transform(X_train)
p_total = Xtr_proc.shape[1]

# 3) Build PCR pipeline: preprocess -> scale -> PCA(M) -> OLS
pcr = Pipeline(steps=[
    ("preprocess", preprocess),
    ("scale", StandardScaler(with_mean=True)),
    ("pca", PCA(svd_solver="full")),   # deterministic for given data
    ("model", LinearRegression())
])

# CV over M = 1..p_total
M_list = list(range(1, p_total + 1))
cv = KFold(n_splits=10, shuffle=True, random_state=42)

grid = GridSearchCV(
    estimator=pcr,
    param_grid={"pca__n_components": M_list},
    scoring="neg_mean_squared_error",
    cv=cv,
    n_jobs=-1
)

# Fit on training
grid.fit(X_train, y_train)

best_M = grid.best_params_["pca__n_components"]
best_pcr = grid.best_estimator_
```

```python
# Evaluate on test
y_pred = best_pcr.predict(X_test)

test_mse  = mean_squared_error(y_test, y_pred)
test_rmse = np.sqrt(test_mse)
test_r2   = r2_score(y_test, y_pred)
test_mae  = mean_absolute_error(y_test, y_pred)

print('\nPCR Model:\n---------')
print(f"Total encoded predictors (p): {p_total}")
print(f"Selected number of PCs (M): {best_M}")
print(f"Test MSE : {test_mse:,.2f}")
print(f"Test RMSE: {test_rmse:,.2f}")
print(f"Test MAE : {test_mae:,.2f}")
print(f"Test R^2 : {test_r2:,.3f}")
```

```
PCR Model:
---------
Total encoded predictors (p): 17
Selected number of PCs (M): 17
Test MSE : 1,085,881.76
Test RMSE: 1,042.06
Test MAE : 646.07
Test R^2 : 0.912
```

[39]:
```python
# 1) See the CV curve across M (uses the `grid` object from earlier)
mean_mse = -grid.cv_results_["mean_test_score"]           # neg MSE -> MSE
M_values = grid.cv_results_["param_pca__n_components"].data.astype(int)
for M, mse in zip(M_values, mean_mse):
    print(f"M={M:2d}  CV-MSE={mse:,.1f}")

# 2) Check cumulative explained variance (on training PCs from the best␣
↪estimator)
pca = grid.best_estimator_.named_steps["pca"]
cumvar = np.cumsum(pca.explained_variance_ratio_)
for i, v in enumerate(cumvar, 1):
    if i in (1,2,3,5,10,17):
        print(f"M={i:2d}  cum var={v:.3f}")
```

```
M= 1  CV-MSE=15,129,434.9
M= 2  CV-MSE=4,570,138.0
M= 3  CV-MSE=4,544,848.4
M= 4  CV-MSE=4,060,801.1
M= 5  CV-MSE=2,728,122.4
M= 6  CV-MSE=2,740,609.7
M= 7  CV-MSE=2,690,164.5
```

```
M= 8  CV-MSE=2,628,453.7
M= 9  CV-MSE=2,540,386.1
M=10  CV-MSE=2,543,341.7
M=11  CV-MSE=2,556,853.9
M=12  CV-MSE=2,549,640.9
M=13  CV-MSE=2,562,233.5
M=14  CV-MSE=2,575,949.2
M=15  CV-MSE=2,423,158.0
M=16  CV-MSE=1,463,514.9
M=17  CV-MSE=1,355,313.0
M= 1  cum var=0.316
M= 2  cum var=0.573
M= 3  cum var=0.643
M= 5  cum var=0.753
M=10  cum var=0.928
M=17  cum var=1.000
```

1.2.7 (f) Fit a PLS model on the training set, with M chosen by crossvalidation. Report the test error obtained, along with the value of M selected by cross-validation

[40]:
```python
# Fit preprocess once to know total encoded dimensionality
Xtr_proc = preprocess.fit_transform(X_train)
p_total = Xtr_proc.shape[1]

# Build PLS pipeline: preprocess -> scale -> PLS(M)
#    We scale here, so set PLS(scale=False) to avoid double-scaling.
pls_pipe = Pipeline(steps=[
    ("preprocess", preprocess),
    ("scale", StandardScaler(with_mean=True)),
    ("pls", PLSRegression(scale=False))
])

# CV over M = 1..p_total
M_list = list(range(1, p_total + 1))
cv = KFold(n_splits=10, shuffle=True, random_state=42)

grid = GridSearchCV(
    estimator=pls_pipe,
    param_grid={"pls__n_components": M_list},
    scoring="neg_mean_squared_error",
    cv=cv,
    n_jobs=-1
)

# Fit on training
grid.fit(X_train, y_train)
```

```python
best_M = grid.best_params_["pls__n_components"]
best_pls = grid.best_estimator_

# Evaluate on test
y_pred = best_pls.predict(X_test).ravel()  # PLS returns (n, 1); flatten to (n,)

test_mse  = mean_squared_error(y_test, y_pred)
test_rmse = np.sqrt(test_mse)
test_r2   = r2_score(y_test, y_pred)
test_mae  = mean_absolute_error(y_test, y_pred)

print("PLS Model:")
print("---------")
print(f"Total encoded predictors (p): {p_total}")
print(f"Selected number of components (M): {best_M}")
print(f"Test MSE : {test_mse:,.2f}")
print(f"Test RMSE: {test_rmse:,.2f}")
print(f"Test MAE : {test_mae:,.2f}")
print(f"Test R^2 : {test_r2:,.3f}")
```

```
PLS Model:
---------
Total encoded predictors (p): 17
Selected number of components (M): 17
Test MSE : 1,085,881.76
Test RMSE: 1,042.06
Test MAE : 646.07
Test R^2 : 0.912
```

1.2.8 (g) Comment on the results obtained. How accurately can we predict the number of college applications received? Is there much difference among the test errors resulting from these five approaches?

How accurate are the predictions?
- All methods achieve **high accuracy**: $R^2 \approx 0.91$, meaning ~91% of the variance in `Apps` is explained by the predictors.
- The typical absolute error on the test set is ~**630–650 applications** (MAE), with **RMSE 1,030–1,040**.

Any meaningful differences among methods?
- **OLS, Ridge, PCR, and PLS** all yield **essentially the same** test error (MSE 1,085,882; RMSE 1,042; (R^2 0.912)).
- Ridge chose a **tiny** λ (1e-4) → effectively **OLS**.
- **PCR/PLS** selected $M = p = 17$ → they replicate the **OLS** fit. - **Lasso** is slightly better: **MSE 1,058,263; RMSE 1,028.7; MAE 630.9;** $R^2 = 0.915$ with **14/17** non-zero coefficients.
- Gains vs OLS: **MSE ↓ ~2.54%**, **RMSE ↓ ~1.28%**, **MAE ↓ ~2.35%**, (R^2) +0.003.
- These are **small but consistent** improvements; practically modest, and likely within the noise one often sees across different splits.

Why this pattern?
- (p=17 n): OLS is already **stable**, so heavy shrinkage/dimension reduction doesn't help much.
- Predictors like **`Accept/Enroll`** are highly informative about **`Apps`**; discarding PCs or enforcing too much shrinkage tends to **drop signal**.
- Lasso's mild sparsity can trim a few redundant/noisy directions, yielding a **slight** edge without losing key signal.

Comments
- You can predict `Apps` **quite well** with these features.
- **No large differences** among methods for this split; **Lasso** offers a **small** improvement and a **simpler** model (14/17 coefficients).
- If you want more reduction (smaller (M) or more sparsity) or a fairer "forecasting" setup, consider **removing near-tautological predictors** (e.g., `Accept`, possibly `Enroll`) and re-running; then PCR/PLS and regularization typically show larger benefits.

Apendix

[41]:
```
# 1) How flat is the CV curve vs M?
mean_mse = -grid.cv_results_["mean_test_score"]      # from your PLS GridSearchCV
M_vals = grid.cv_results_["param_pls__n_components"].data.astype(int)
for M, mse in zip(M_vals, mean_mse):
    print(f"M={M:2d}  CV-MSE={mse:,.1f}")
```

```
M= 1  CV-MSE=3,809,605.1
M= 2  CV-MSE=2,801,109.8
M= 3  CV-MSE=2,305,470.4
M= 4  CV-MSE=2,041,009.1
M= 5  CV-MSE=1,631,588.8
M= 6  CV-MSE=1,443,624.2
M= 7  CV-MSE=1,391,610.0
M= 8  CV-MSE=1,380,315.1
M= 9  CV-MSE=1,360,507.0
M=10  CV-MSE=1,358,676.4
M=11  CV-MSE=1,356,148.3
M=12  CV-MSE=1,357,355.8
M=13  CV-MSE=1,357,752.8
M=14  CV-MSE=1,356,312.4
M=15  CV-MSE=1,355,519.8
M=16  CV-MSE=1,355,363.8
M=17  CV-MSE=1,355,313.0
```

[42]:
```
# 2) Confirm PLS(M=p) ~ OLS predictions
y_pred_pls = best_pls.predict(X_test).ravel()

# reuse the OLS pipeline from part (b): 'ols'
y_pred_ols = ols.predict(X_test)

import numpy as np
print("Max abs diff (PLS vs OLS):", np.max(np.abs(y_pred_pls - y_pred_ols)))
```

```
Max abs diff (PLS vs OLS): 1.709850039333105e-10
```