Computational Statistics and Data Analysis (MVComp2)

Solutions to exercise 8

Lecturer Tristan Bereau

Semester Wi23/24
Due Dec. 14, 2023, 23:59

1 Partial derivative of the residual sum of squares (2 points)

Recall the residual sum of squares, $\text{RSS} = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2$.

Show that

$$\frac{\partial}{\partial w_k}\text{RSS}(\boldsymbol{w}) = a_k w_k - c_k$$
$$a_k = 2\|x_{:,k}\|^2$$
$$c_k = 2\sum_{i=1}^{n} x_{ik}(y_i - \boldsymbol{w}_{-k}^\top \boldsymbol{x}_{i,-k}) = 2\boldsymbol{x}_{:,k}^\top \boldsymbol{r}_k,$$

where $\boldsymbol{w}_{-k}$ corresponds to $\boldsymbol{w}$ *without* component k, $\boldsymbol{x}_{i,-k}$ is $\boldsymbol{x}_i$ without component k, and $\boldsymbol{r}_k = \boldsymbol{y} - \boldsymbol{w}_{-k}^\top \boldsymbol{x}_{:,-k}$ is the residual due to using all the features except feature k.

Hint: Partition the weights into those involving k and those not involving k.

1.1 Solution

Write the derivative of the RSS and split the weights into those involving k and those not involving k to yield the desired answer

$$\begin{aligned}
\frac{\partial \text{RSS}}{\partial w_k} &= -2\sum_{i=1}^{n}(y_i - (\boldsymbol{x}_i^\top \boldsymbol{w}))x_{ik} \\
&= -2\sum_{i=1}^{n}(y_i - (x_{ik}w_k + \boldsymbol{x}_{:,-k}^\top w_{-k}))x_{ik} \\
&= -2\sum_{i=1}^{n}(r_{ik} - x_{ik}w_k)x_{ik} \\
&= 2\sum_{i=1}^{n} x_{ik}^2 w_{ik} - 2\sum_{i=1}^{n} r_{ik}x_{ik} \\
&= a_k w_k - c_k.
\end{aligned}$$

2 Lasso regression (2 points)

Recall the result of problem 1, where the RSS provides the smooth part of the lasso objective

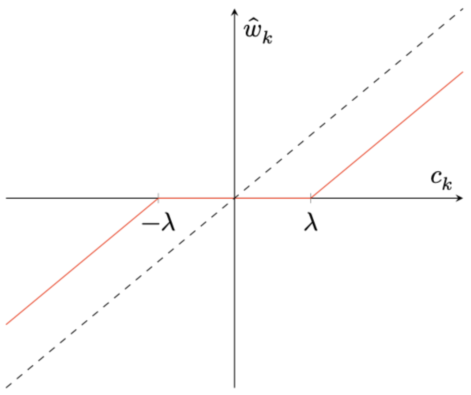
$$\mathcal{L}(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_1.$$

(a) Find the optimal parameters, $\hat{w}_k$ as a function of c_k. You should find a relation as illustrated in the Figure.

(b) Instead of using the optimal parameters derived in (a), it helps to approximate it with a so-called soft-threshold function

$$\mathrm{SoftThreshold}(x, \delta) = \mathrm{sign}(x)\max(|x| - \delta, 0).$$

Sketch $\hat{w}_d = \mathrm{SoftThreshold}(\frac{c_d}{a_d}, \frac{\lambda}{a_d})$ and compare it with both the optimal parameters derived in (a), as well as the vanilla RSS from problem 1. Interpret how the two Lasso methods affect the optimal parameters.



Hint: We can generalize the notion of a derivative for certain functions with local discontinuities. Such functions are called subdifferentiable. A famous example is the absolute value function, $f(x) = |x|$. Its subdifferential is given by

$$\partial f(x) = \begin{cases} -1, \text{ if } x < 0 \\ [-1, 1], \text{ if } x = 0 \\ +1, \text{ if } x > 0 \end{cases}$$

where the notation $[-1, 1]$ means any value between -1 and 1 inclusive.

2.1 Solution

(a) Take the derivative wrt the parameter, w_d, using the definition of the subdifferential

$$\begin{aligned} \partial_{w_d}\mathcal{L}(\boldsymbol{w}) &= (a_d w_d - c_d) + \lambda \partial_{w_d}\|w\|_1 \\ &= \begin{cases} a_d w_d - c_d - \lambda, & w_d < 0 \\ [-c_d - \lambda, -c_d + \lambda], & w_d = 0 \\ a_d w_d - c_d + \lambda, & w_d > 0 \end{cases} \end{aligned}$$

This leads to three cases

1. $w_d = \frac{c_d + \lambda}{a_d}$ for $w_d < 0$, leading to the condition $c_d < -\lambda$
2. $w_d = 0$ with the values in the subdifferential leading to the interval $c_d \in [-\lambda, \lambda]$
3. $w_d = \frac{c_d - \lambda}{a_d}$ for $w_d > 0$, leading to the condition $c_d > \lambda$.

These cases exactly describe what's on the Figure.

(b) The soft threshold will look like the figure in (a). It keeps the defining feature in the interval $[-\lambda, \lambda]$, which consists of setting the parameters to 0 in that interval, at the expense of being a biased estimator away from it. This is the shrinkage operator that removes small contributions and sets parameters to 0 instead.

3 Prostate cancer (6 points)

We will analyze a prostate cancer dataset, which you can download here: prostate.csv. It contains 97 datapoints, each with the following features:

Variable	Description
`lcavol`	(log) Cancer volume
`lweight`	(log) Weight
`age`	Patient age
`lbph`	(log) Vening Prostatic Hyperplasia
`svi`	Seminal Vesicle Invasion
`lcp`	(log) Capsular Penetration
`gleason`	Gleason score
`pgg45`	Percent of Gleason score 4 or 5
`lpsa`	(log) Prostate Specific Antigen

The objective is to build a predictive model for `lpsa`. We will denote the vector of labels of `lpsa`, $\boldsymbol{y}$. The rest of the dataset will consist of the feature matrix, $\boldsymbol{X}$, made of the abovementioned variables, as well as an extra column $[1, \dots, 1]^\top$ to account for the intercept of your linear models.

(a) Plot the correlation between `lpsa` and each one of the three features that correlate the strongest with `lpsa`.

(b) Check that your dataset matrix, $\boldsymbol{X}$, is full rank.

(c) Split your data into a training and a test set. Keep 30 datapoints for the test set. (Feel free to use existing libraries, like `scikit-learn`.)

(d) Implement yourself an ordinary least squares solver. Do not use existing statistics / machine learning libraries (though feel free to use linear-algebra libraries). To simplify your task, consider filling in the following template:

```python
import numpy as np
from dataclasses import dataclass, field

@dataclass
class OrdinaryLeastSquares:
    model_params: np.ndarray = field(init=False)

    def __post_init__(self):
        self.model_params = None

    def fit(self, X_train_: np.ndarray, y_train_: np.ndarray) -> None:
      # self.model_params = ... (fit your model)
      pass

    def predict(self, X_test_: np.ndarray) -> np.ndarray:
      # make predictions for X_test_
      pass

    def rmse(self, X_test_: np.ndarray, y_test_: np.ndarray) -> float:
      # compute the root-mean-squared-error
      pass
```

Print out the set of optimal parameters for your training set, as well as the RMSE of the test set.

(e) Implement yourself ridge regression. Consider extending the following template:

```
@dataclass
class RidgeRegression(OrdinaryLeastSquares):
    ridge_penalty: float

    def fit(self, X_train_: np.ndarray, y_train_: np.ndarray) -> None:
        # self.model_params = ... (fit your model)
        pass
```

where `ridge_penalty` is the coefficient in front of the ℓ_2 regularization term. Exclude the intercept from the regularization procedure. Plot the values of all model parameters (except the intercept) as a function of the regularization penalty term. Have the penalty term vary from 0 to 100. On a separate figure, plot the RMSE of the test set as a function of the penalty term. Indicate at what penalty value your best model is.

(f) Implement yourself lasso regression. Use the objective function and soft threshold function from Problem 2 (b) to iteratively update your model parameters until convergence (controlled by `num_iter` in the template code below)

```
@dataclass
class LassoRegression(OrdinaryLeastSquares):
    lasso_penalty: float

    def fit(self, X_train_: np.ndarray, y_train_: np.ndarray, num_iter: int = 1000) -> None:
        # self.model_params = ... (fit your model)
        pass
```

where `lasso_penalty` is the coefficient in front of the ℓ_1 regularization term. Exclude the intercep from the regularization procedure. Plot the values of all model parameters (except the intercept) as a function of the regularization penalty term. Have the penalty term vary from 10^{-1} to 10^{2}. On a separate figure, plot the RMSE of the test set as a function of the penalty term. Indicate at what penalty value your best model is.

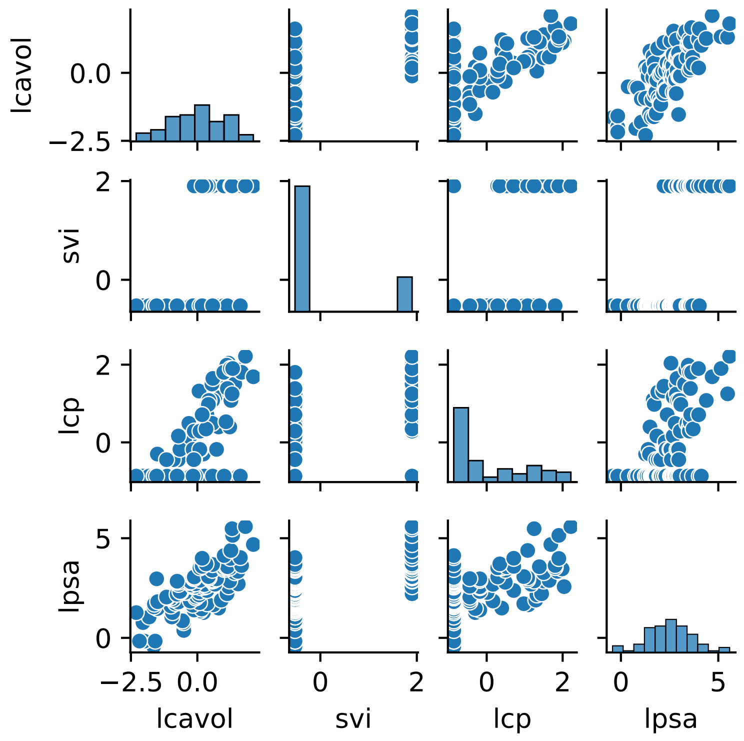
(g) Compare your sets of best parameters across the three solvers, and comment on the impact of ridge and lasso regression.

3.1 Solution

(a) Inspection of the data shows that the three features correlating most strongly `lpsa` are: `lcavol`, `svi`, `lcp`. The `seaborn.pairplot` function allows us to plot the desired correlations.

```
import pandas as pd
import numpy as np
import seaborn as sns

df = pd.read_csv("data_08_prostate.csv")
column_names = df.columns
g = sns.pairplot(df[["lcavol", "svi", "lcp", "lpsa"]], height=1)
```



(b)

```
def extract_X_y(df_: pd.DataFrame):
    X = df_.to_numpy()[:,:-1]
    X = np.hstack((np.ones((X.shape[0], 1)), X))
    y = df_.to_numpy()[:,-1]
    return X, y

def has_full_rank(X_: np.array) -> bool:
    return np.linalg.matrix_rank(X_) == X_[0].size

X, y = extract_X_y(df)
has_full_rank(X)
```

```
True
```

(c)

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
  X, y, test_size=0.30, random_state=42
)
X_train.shape, X_test.shape
```

```
((67, 9), (30, 9))
```

(d)

```
from sklearn.metrics import mean_squared_error

@dataclass
class OrdinaryLeastSquares:
    model_params: np.ndarray = field(init=False)

    def __post_init__(self):
        self.model_params = None

    def fit(self, X_train_: np.ndarray, y_train_: np.ndarray) -> None:
        self.model_params = (
            np.linalg.inv(X_train_.T @ X_train_) @ X_train_.T @ y_train_
        )

    def predict(self, X_test_: np.ndarray) -> np.ndarray:
        return X_test_ @ self.model_params

    def rmse(self, X_test_: np.ndarray, y_test_: np.ndarray) -> float:
        y_pred = self.predict(X_test_)
        return np.sqrt(mean_squared_error(y_test_, y_pred))

ols = OrdinaryLeastSquares()
ols.fit(X_train, y_train)
ols.model_params, ols.rmse(X_test, y_test)
```

```
(array([ 2.45462916,  0.75271916,  0.24358154, -0.19201857,  0.11429899,
         0.3585389 , -0.23917874,  0.06634662,  0.13186029]),
 0.6941798374525127)
```

(e)

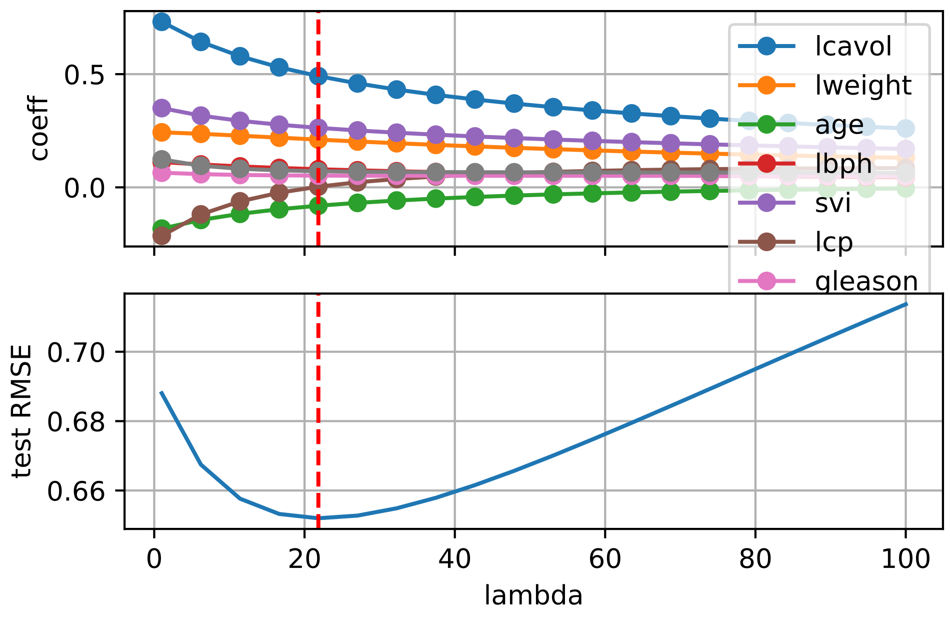
```
import matplotlib.pyplot as plt

@dataclass
```

```
class RidgeRegression(OrdinaryLeastSquares):
    ridge_penalty: float

    def fit(self, X_train_: np.ndarray, y_train_: np.ndarray) -> None:
        I = np.eye(X_train_.shape[1])
        # Exclude intercept from regularization
        I[0, 0] = 0
        self.model_params = (
            np.linalg.inv(X_train_.T @ X_train_ + self.ridge_penalty * I)
            @ X_train_.T @ y_train_
        )

lmb_range = np.linspace(1, 100, 20)
model_params = np.zeros((len(lmb_range), len(column_names)-1))
rmse_test = np.zeros((len(lmb_range)))
for i, lmb in enumerate(lmb_range):
    rr = RidgeRegression(ridge_penalty=lmb)
    rr.fit(X_train, y_train)
    rmse_test[i] = rr.rmse(X_test, y_test)
    model_params[i] = rr.model_params[1:]
fig, ax = plt.subplots(2, 1, sharex=True)
for col in range(len(column_names)-1):
    ax[0].plot(lmb_range, model_params.T[col], "o-", label=f"{column_names[col]}")
best_rr_params = model_params[rmse_test.argmin()]
ax[0].axvline(lmb_range[rmse_test.argmin()], linestyle="--", c="r")
ax[0].legend()
ax[0].grid()
ax[0].set_ylabel("coeff")
ax[1].plot(lmb_range, rmse_test)
ax[1].axvline(lmb_range[rmse_test.argmin()], linestyle="--", c="r")
ax[1].set_ylabel("test RMSE")
ax[1].set_xlabel("lambda")
ax[1].grid();
```



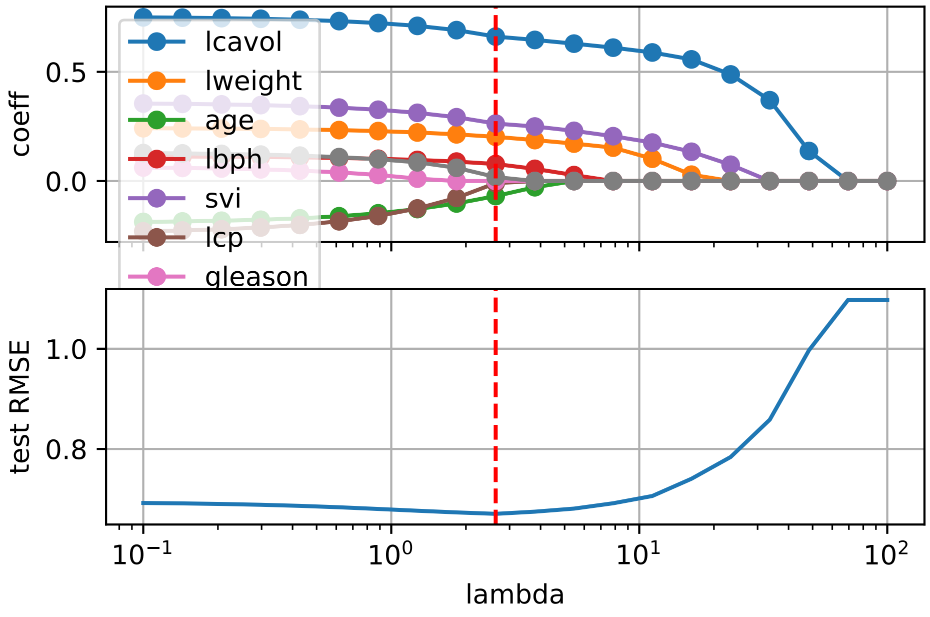
(f)

```
@dataclass
class LassoRegression(OrdinaryLeastSquares):
    lasso_penalty: float

    def fit(self, X_train_: np.ndarray, y_train_: np.ndarray, num_iter: int = 1000) -> None:
        def soft_thresholding(x_: float, penalty: float) -> float:
            return np.sign(x_) * max(abs(x_) - penalty, 0)

        n, m = X_train_.shape
        w = np.zeros(m)
        for _ in range(num_iter):
            for j in range(m):
                residual = y_train_ - (X_train_ @ w - X_train_[:, j] * w[j])
                rho = X_train_[:, j].dot(residual)
                norm = np.linalg.norm(X_train_[:, j]) ** 2
                w[j] = (
                    soft_thresholding(rho / norm, self.lasso_penalty / norm)
                    if j != 0 else rho / norm
                )
        self.model_params = w

lmb_range = np.logspace(-1, 2, 20, base=10)
model_params = np.zeros((len(lmb_range), len(column_names)-1))
rmse_test = np.zeros((len(lmb_range)))
for i, lmb in enumerate(lmb_range):
    lasso = LassoRegression(lasso_penalty=lmb)
    lasso.fit(X_train, y_train)
    rmse_test[i] = lasso.rmse(X_test, y_test)
    model_params[i] = lasso.model_params[1:]
fig, ax = plt.subplots(2, 1, sharex=True)
for col in range(len(column_names)-1):
    ax[0].plot(lmb_range, model_params.T[col], "o-", label=f"{column_names[col]}")
best_lasso_params = model_params[rmse_test.argmin()]
ax[0].axvline(lmb_range[rmse_test.argmin()], linestyle="--", c="r")
ax[0].legend()
ax[0].grid()
ax[0].set_ylabel("coeff")
ax[1].plot(lmb_range, rmse_test)
ax[1].axvline(lmb_range[rmse_test.argmin()], linestyle="--", c="r")
ax[1].set_ylabel("test RMSE")
ax[1].set_xlabel("lambda")
ax[1].set_xscale("log")
ax[1].grid();
```



(g)

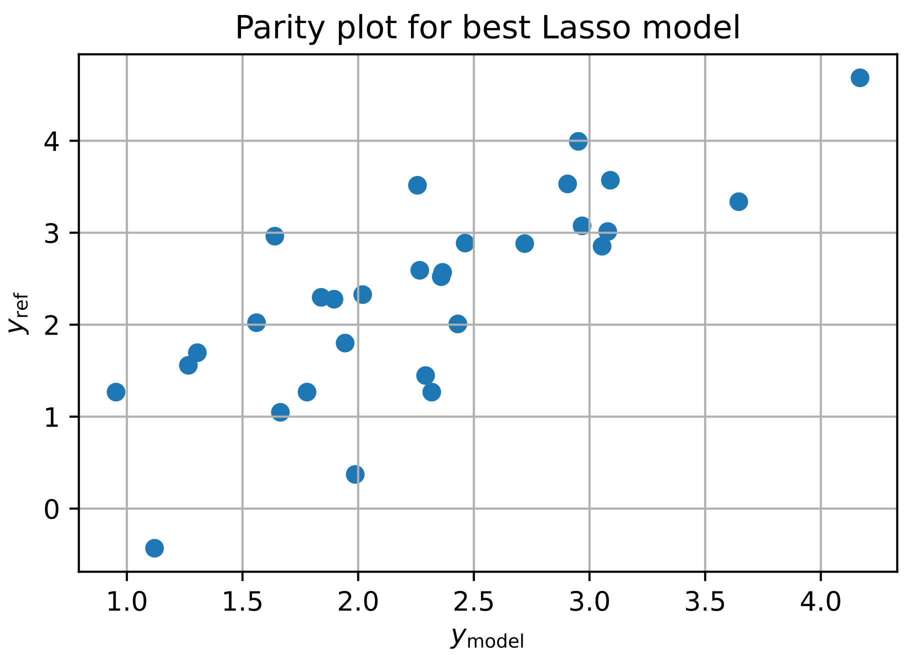
```
df_params = pd.DataFrame(
    data=np.array([ols.model_params[1:], best_rr_params, best_lasso_params]).T,
    columns=["OLS", "Ridge", "LASSO"],
)
df_params["param"] = column_names[:-1]
df_params = df_params.set_index("param")
df_params
```

	OLS	Ridge	LASSO
param			
lcavol	0.752719	0.491252	0.661498

	OLS	Ridge	LASSO
param			
lweight	0.243582	0.210797	0.202488
age	-0.192019	-0.081104	-0.068675
lbph	0.114299	0.079937	0.078088
svi	0.358539	0.262819	0.263272
lcp	-0.239179	0.002996	-0.009506
gleason	0.066347	0.051870	0.000000
pgg45	0.131860	0.071212	0.018324

All RMSEs are close. Ridge and LASSO provide some significantly smaller coefficients. LASSO overall removes the `gleason` parameter entirely.

```python
lasso = LassoRegression(lasso_penalty=rmse_test.argmin())
lasso.fit(X_train, y_train)
y_pred = lasso.predict(X_test)
plt.title("Parity plot for best Lasso model")
plt.scatter(y_pred, y_test)
plt.xlabel(r"$y_\text{model}$")
plt.ylabel(r"$y_\text{ref}$")
plt.grid();
```


Parity plot for best Lasso model
y_ref
y_model