housing_regularization_cv

March 13, 2025

1 Regularization example IN3050

1.1 Preamble and dataset

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import sklearn
```

[2]:
```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
```

In the previous years, IN3050 used the `Boston` dataset for this notebook, but this dataset is now removed from `scikit-learn` due to an ethical problem. That's why in 2024, we switched to the `California housing` dataset.

[3]:
```
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
print(housing.DESCR)
```

```
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

:Number of Instances: 20640

:Number of Attributes: 8 numeric, predictive attributes and the target

:Attribute Information:
    - MedInc        median income in block group
    - HouseAge      median house age in block group
    - AveRooms      average number of rooms per household
    - AveBedrms     average number of bedrooms per household
    - Population    block group population
    - AveOccup      average number of household members
    - Latitude      block group latitude
    - Longitude     block group longitude
```

```
:Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

The target variable is the median house value for California districts,
expressed in hundreds of thousands of dollars ($100,000).

This dataset was derived from the 1990 U.S. census, using one row per census
block group. A block group is the smallest geographical unit for which the U.S.
Census Bureau publishes sample data (a block group typically has a population
of 600 to 3,000 people).

A household is a group of people residing within a home. Since the average
number of rooms and bedrooms in this dataset are provided per household, these
columns may take surprisingly large values for block groups with few households
and many empty houses, such as vacation resorts.

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. rubric:: References

- Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
  Statistics and Probability Letters, 33 (1997) 291-297
```

[4]:
```python
X = housing.data
t = housing.target
```

[5]:
```python
X.shape, t.shape
```

[5]:
```
((20640, 8), (20640,))
```

[6]:
```python
X[0, :]
```

[6]:
```
array([   8.3252    ,   41.        ,    6.98412698,    1.02380952,
        322.        ,    2.55555556,   37.88      , -122.23      ])
```

[7]:
```python
t[:10]
```

[7]:
```
array([4.526, 3.585, 3.521, 3.413, 3.422, 2.697, 2.992, 2.414, 2.267,
       2.611])
```

[8]:
```python
from sklearn.model_selection import train_test_split
X_train, X_val, t_train, t_val = train_test_split(X, t, random_state=2025)
```

[9]:
```
# train_test_split?
```

[10]:
```
X_train.shape, X_val.shape, t_train.shape, t_val.shape
```

[10]: ((15480, 8), (5160, 8), (15480,), (5160,))

1.2 Linear Regression

[11]:
```
# LinearRegression?
```

[12]:
```
lr =LinearRegression()
lr.fit(X_train, t_train)
```

[12]: LinearRegression()

[14]:
```
round(lr.score(X_train, t_train), 4)
```

[14]: 0.6075

[15]:
```
round(lr.score(X_val, t_val), 4)
```

[15]: 0.6021

[16]:
```
# lr.score?
```

[17]:
```
from sklearn.metrics import mean_squared_error as sk_mse
```

[18]:
```
round(sk_mse(lr.predict(X_val), t_val), 4)
```

[18]: 0.5347

1.2.1 So far

Similar results for train and val. No overfitting. But can we do better?

1.3 Polynomial features

We add second order polynomial features

[19]:
```
from sklearn.preprocessing import PolynomialFeatures
```

[20]:
```
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly_train = poly.fit_transform(X_train)
X_poly_val = poly.transform(X_val)
```

[21]:
```
X_poly_train.shape
```

[21]: (15480, 44)

Comment

- 8 original features
- 8 squares of original features
- 28 polynomial combinations of the features with degree ≤ 2

[22]:
```
lr_poly = LinearRegression()
lr_poly.fit(X_poly_train, t_train)
```

[22]:
```
LinearRegression()
```

[23]:
```
round(lr_poly.score(X_poly_train, t_train), 4)
```

[23]:
```
0.6872
```

[24]:
```
round(lr_poly.score(X_poly_val, t_val), 4)
```

[24]:
```
-5.398
```

[25]:
```
round(sk_mse(lr_poly.predict(X_poly_train), t_train), 4)
```

[25]:
```
0.4152
```

[26]:
```
round(sk_mse(lr_poly.predict(X_poly_val), t_val), 4)
```

[26]:
```
8.5982
```

1.3.1 So far

Large improvement on *train*. The oposite on *val*. Large difference between *train* and *val*. Overfitting!

1.4 Ridge regularization

[27]:
```
ridge_poly = Ridge()
ridge_poly.fit(X_poly_train, t_train)
```

```
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=2.53832e-19): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

[27]:
```
Ridge()
```

Scikit-learn complains that the data contains values which are too high or too low, leading to possibly inaccurate results. We will deal with this soon.

[28]:
```
# Ridge?
```

[29]:
```
round(sk_mse(ridge_poly.predict(X_poly_val), t_val), 4)
```

[29]: 2.5762

[30]:
```
round(sk_mse(ridge_poly.predict(X_poly_train), t_train), 4)
```

[30]: 0.4204

[31]:
```
round(ridge_poly.score(X_poly_train, t_train), 4)
```

[31]: 0.6833

[32]:
```
round(ridge_poly.score(X_poly_val, t_val), 4)
```

[32]: -0.917

1.4.1 So far

Best score on val so far. Still much better on train. Is the regularization optimal? And can we do anything with the warnings?

1.5 Tuning regularization

An instance of parameter tuning

[33]:
```
for a in [0, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]:
    ridge_poly = Ridge(alpha=a)
    ridge_poly.fit(X_poly_train, t_train)
    train_score = ridge_poly.score(X_poly_train, t_train)
    val_score = ridge_poly.score(X_poly_val, t_val)
    train_mse = sk_mse(ridge_poly.predict(X_poly_train), t_train)
    val_mse = sk_mse(ridge_poly.predict(X_poly_val), t_val)
    print(f"Alpha: {a:.5f}, train_score: {train_score:.3f}, val_score:
  ↪{val_score:.3f}, train_mse: {train_mse:.4f}, val_mse: {val_mse:.4f}")
```

```
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=7.23758e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=7.26104e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=7.47217e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=9.58809e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

```
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=3.08655e-20): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=2.53832e-19): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=2.81128e-18): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

```
Alpha: 0.00000, train_score: 0.687, val_score:-5.398, train_mse: 0.4152,
val_mse: 8.5982
Alpha: 0.00010, train_score: 0.687, val_score:-5.399, train_mse: 0.4152,
val_mse: 8.5992
Alpha: 0.00100, train_score: 0.687, val_score:-5.404, train_mse: 0.4152,
val_mse: 8.6069
Alpha: 0.01000, train_score: 0.687, val_score:-5.415, train_mse: 0.4153,
val_mse: 8.6210
Alpha: 0.10000, train_score: 0.686, val_score:-4.564, train_mse: 0.4164,
val_mse: 7.4776
Alpha: 1.00000, train_score: 0.683, val_score:-0.917, train_mse: 0.4204,
val_mse: 2.5762
Alpha: 10.00000, train_score: 0.675, val_score:0.303, train_mse: 0.4310,
val_mse: 0.9372
Alpha: 100.00000, train_score: 0.671, val_score:0.371, train_mse: 0.4369,
val_mse: 0.8451
Alpha: 1000.00000, train_score: 0.668, val_score:0.386, train_mse: 0.4412,
val_mse: 0.8246
Alpha: 10000.00000, train_score: 0.665, val_score:0.466, train_mse: 0.4450,
val_mse: 0.7181
```

```
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=2.87186e-17): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

[35]:
```python
for a in range(10):
    a = .5 + 0.1 * a
    ridge_poly = Ridge(alpha=a)
    ridge_poly.fit(X_poly_train, t_train)
    train_score = ridge_poly.score(X_poly_train, t_train)
    val_score = ridge_poly.score(X_poly_val, t_val)
    train_mse = sk_mse(ridge_poly.predict(X_poly_train), t_train)
    val_mse = sk_mse(ridge_poly.predict(X_poly_val), t_val)
```

```
    print(f"Alpha: {a:.4f}, train_score: {train_score:.3f}, val_score:
↪{val_score:.3f}, train_mse: {train_mse:.4f}, val_mse: {val_mse:.4f}")
```

```
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=1.2775e-19): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=1.52583e-19): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=1.77625e-19): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=2.02857e-19): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

```
Alpha: 0.5000, train_score: 0.685, val_score:-2.025, train_mse: 0.4185, val_mse:
4.0658
Alpha: 0.6000, train_score: 0.684, val_score:-1.704, train_mse: 0.4189, val_mse:
3.6343
Alpha: 0.7000, train_score: 0.684, val_score:-1.447, train_mse: 0.4193, val_mse:
3.2878
Alpha: 0.8000, train_score: 0.684, val_score:-1.236, train_mse: 0.4197, val_mse:
3.0053
```

```
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=2.28264e-19): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=2.53832e-19): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=2.79549e-19): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=3.05402e-19): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=3.75664e-19): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

```
Alpha: 0.9000, train_score: 0.684, val_score:-1.062, train_mse: 0.4200, val_mse:
2.7716
Alpha: 1.0000, train_score: 0.683, val_score:-0.917, train_mse: 0.4204, val_mse:
2.5762
Alpha: 1.1000, train_score: 0.683, val_score:-0.794, train_mse: 0.4207, val_mse:
2.4109
Alpha: 1.2000, train_score: 0.683, val_score:-0.689, train_mse: 0.4210, val_mse:
2.2698
Alpha: 1.3000, train_score: 0.683, val_score:-0.599, train_mse: 0.4213, val_mse:
2.1483
Alpha: 1.4000, train_score: 0.682, val_score:-0.520, train_mse: 0.4216, val_mse:
2.0429
```

```
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=4.0218e-19): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

1.5.1 So far

The regularization factor of 10.0 seems optimal. It gives 0.303 on val. Further increasing it ruins train performance.

1.6 Scaling

Let's make our data nice and beautiful and get rid of the warnings

[36]:
```python
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

[37]:
```python
for a in [0, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000,␣
 ↪10000]:
    ridge_poly = make_pipeline(StandardScaler(with_mean=False), Ridge(alpha=a))
    ridge_poly.fit(X_poly_train, t_train)
    train_score = ridge_poly.score(X_poly_train, t_train)
    val_score = ridge_poly.score(X_poly_val, t_val)
    train_mse = sk_mse(ridge_poly.predict(X_poly_train), t_train)
    val_mse = sk_mse(ridge_poly.predict(X_poly_val), t_val)
    print(f"Alpha: {a:.4f}, train_score: {train_score:.3f}, val_score:
 ↪{val_score:.3f}, train_mse: {train_mse:.4f}, val_mse: {val_mse:.4f}")
```

```
Alpha: 0.0000, train_score: 0.687, val_score:-5.398, train_mse: 0.4152, val_mse:
8.5982
Alpha: 0.0000, train_score: 0.687, val_score:-5.393, train_mse: 0.4152, val_mse:
8.5916
Alpha: 0.0000, train_score: 0.687, val_score:-5.349, train_mse: 0.4152, val_mse:
8.5320
Alpha: 0.0001, train_score: 0.687, val_score:-4.931, train_mse: 0.4152, val_mse:
7.9710
```

```
Alpha: 0.0010, train_score: 0.687, val_score:-2.420, train_mse: 0.4155, val_mse:
4.5959
Alpha: 0.0100, train_score: 0.685, val_score:0.310, train_mse: 0.4182, val_mse:
0.9271
Alpha: 0.1000, train_score: 0.680, val_score:0.523, train_mse: 0.4251, val_mse:
0.6417
Alpha: 1.0000, train_score: 0.669, val_score:0.413, train_mse: 0.4396, val_mse:
0.7890
Alpha: 10.0000, train_score: 0.662, val_score:0.336, train_mse: 0.4485, val_mse:
0.8926
Alpha: 100.0000, train_score: 0.648, val_score:0.524, train_mse: 0.4676,
val_mse: 0.6403
Alpha: 1000.0000, train_score: 0.616, val_score:0.573, train_mse: 0.5095,
val_mse: 0.5738
Alpha: 10000.0000, train_score: 0.540, val_score:0.527, train_mse: 0.6100,
val_mse: 0.6353
```

[39]:
```
for a in [0.1, 0.2, 0.5, 0.8, 1.0, 1.5, 2, 5]:
    ridge_poly = make_pipeline(StandardScaler(with_mean=False), Ridge(alpha=a))
    ridge_poly.fit(X_poly_train, t_train)
    train_score = ridge_poly.score(X_poly_train, t_train)
    val_score = ridge_poly.score(X_poly_val, t_val)
    train_mse = sk_mse(ridge_poly.predict(X_poly_train), t_train)
    val_mse = sk_mse(ridge_poly.predict(X_poly_val), t_val)
    print(f"Alpha: {a:.4f}, train_score: {train_score:.3f}, val_score:
 ↪{val_score:.3f}, train_mse: {train_mse:.4f}, val_mse: {val_mse:.4f}")
```

```
Alpha: 0.1000, train_score: 0.680, val_score:0.523, train_mse: 0.4251, val_mse:
0.6417
Alpha: 0.2000, train_score: 0.677, val_score:0.511, train_mse: 0.4289, val_mse:
0.6570
Alpha: 0.5000, train_score: 0.672, val_score:0.464, train_mse: 0.4352, val_mse:
0.7198
Alpha: 0.8000, train_score: 0.670, val_score:0.430, train_mse: 0.4383, val_mse:
0.7662
Alpha: 1.0000, train_score: 0.669, val_score:0.413, train_mse: 0.4396, val_mse:
0.7890
Alpha: 1.5000, train_score: 0.667, val_score:0.384, train_mse: 0.4417, val_mse:
0.8283
Alpha: 2.0000, train_score: 0.666, val_score:0.365, train_mse: 0.4430, val_mse:
0.8529
Alpha: 5.0000, train_score: 0.664, val_score:0.330, train_mse: 0.4463, val_mse:
0.9006
```

[40]:
```
for b in range(11):
    a = 0.1 * b
    ridge_poly = make_pipeline(StandardScaler(with_mean=False), Ridge(alpha=a))
    ridge_poly.fit(X_poly_train, t_train)
```

```
    train_score = ridge_poly.score(X_poly_train, t_train)
    val_score = ridge_poly.score(X_poly_val, t_val)
    train_mse = sk_mse(ridge_poly.predict(X_poly_train), t_train)
    val_mse = sk_mse(ridge_poly.predict(X_poly_val), t_val)
    print(f"Alpha: {a:.4f}, train_score: {train_score:.3f}, val_score:
↪{val_score:.3f}, train_mse: {train_mse:.4f}, val_mse: {val_mse:.4f}")
```

```
Alpha: 0.0000, train_score: 0.687, val_score:-5.398, train_mse: 0.4152, val_mse:
8.5982
Alpha: 0.1000, train_score: 0.680, val_score:0.523, train_mse: 0.4251, val_mse:
0.6417
Alpha: 0.2000, train_score: 0.677, val_score:0.511, train_mse: 0.4289, val_mse:
0.6570
Alpha: 0.3000, train_score: 0.675, val_score:0.495, train_mse: 0.4316, val_mse:
0.6787
Alpha: 0.4000, train_score: 0.673, val_score:0.479, train_mse: 0.4336, val_mse:
0.7002
Alpha: 0.5000, train_score: 0.672, val_score:0.464, train_mse: 0.4352, val_mse:
0.7198
Alpha: 0.6000, train_score: 0.671, val_score:0.451, train_mse: 0.4364, val_mse:
0.7372
Alpha: 0.7000, train_score: 0.670, val_score:0.440, train_mse: 0.4374, val_mse:
0.7526
Alpha: 0.8000, train_score: 0.670, val_score:0.430, train_mse: 0.4383, val_mse:
0.7662
Alpha: 0.9000, train_score: 0.669, val_score:0.421, train_mse: 0.4390, val_mse:
0.7783
Alpha: 1.0000, train_score: 0.669, val_score:0.413, train_mse: 0.4396, val_mse:
0.7890
```

1.6.1 So far

The optimal score on *val* (not the best, but without losing the performance on train) is 0.523, achieved with

- polynomial features
- regularization
- $\alpha = 0.1$
- scaling

1.7 Cross-validation experiments

[41]:
```
from sklearn.model_selection import cross_val_score
```

[42]:
```
cvs = cross_val_score(LinearRegression(), X_train, t_train, cv=4)
print(cvs)
print(f"Mean score: {np.sum(cvs)/len(cvs):.4f}")
print(f"Standard deviation: {np.std(cvs):.4f}")
```

```
[0.32557717 0.60987968 0.6149328  0.60697934]
Mean score: 0.5393
Standard deviation: 0.1235
```

1.7.1 Observations

- Some variation in results.
- The conclusions from using one dev-set are less firm
- Hopefully, the mean is a better measure than the individual experiments
- This set seems too small
- (Each training set is slightly smaller than earlier, 75%)

[43]:
```python
cvs = cross_val_score(Ridge(), X_train, t_train, cv=4)
print(cvs)
print(f"Mean score: {np.sum(cvs)/len(cvs):.4f}")
print(f"Standard deviation: {np.std(cvs):.4f}")
```

```
[0.3254875  0.609871   0.61501522 0.60696442]
Mean score: 0.5393
Standard deviation: 0.1235
```

1.7.2 With polynomial features

[44]:
```python
cvs = cross_val_score(Ridge(), X_poly_train, t_train, cv=4)
print(cvs)
print(f"Mean score: {np.sum(cvs)/len(cvs):.4f}")
print(f"Standard deviation: {np.std(cvs):.4f}")
```

```
[0.58055433 0.67157301 0.69887793 0.50789235]
Mean score: 0.6147
Standard deviation: 0.0757
```

```
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=3.14658e-19): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=3.07154e-19): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=3.10294e-19): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=1.33003e-18): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

1.7.3 Observations

- The variation is lower than without polynomial features

What if we remove regularization (alpha) from Linear regression?

[46]:
```
cvs = cross_val_score(Ridge(alpha=0), X_poly_train, t_train, cv=4)
print(cvs)
print(f"Mean score: {np.sum(cvs)/len(cvs):.4f}")
print(f"Standard deviation: {np.std(cvs):.4f}")
```

```
[-4.86395083  0.67589949  0.63670265  0.5131771 ]
Mean score: -0.7595
Standard deviation: 2.3704
```

```
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=5.79581e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=5.60619e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=4.91813e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/home/andrei/my_python/lib/python3.12/site-
packages/sklearn/linear_model/_ridge.py:215: LinAlgWarning: Ill-conditioned
matrix (rcond=2.49242e-20): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

1.7.4 Observations

- Large variation, one split is really unlucky
- The mean is of course not impressive
- The polynomial features *need* regularization.

1.7.5 With normalization

[47]:
```
for b in range(11):
    a = 0.1 * b
    ridge_poly = make_pipeline(StandardScaler(with_mean=False), Ridge(alpha=a))
    ridge_poly.fit(X_poly_train, t_train)
    train_score = ridge_poly.score(X_poly_train, t_train)
    val_score = ridge_poly.score(X_poly_val, t_val)
    train_mse = sk_mse(ridge_poly.predict(X_poly_train), t_train)
    val_mse = sk_mse(ridge_poly.predict(X_poly_val), t_val)
    print(f"Alpha: {a:.4f}, train_score: {train_score:.3f}, val_score:
 ↪{val_score:.3f}, train_mse: {train_mse:.4f}, val_mse: {val_mse:.4f}")
```

```
Alpha: 0.0000, train_score: 0.687, val_score:-5.398, train_mse: 0.4152, val_mse:
8.5982
Alpha: 0.1000, train_score: 0.680, val_score:0.523, train_mse: 0.4251, val_mse:
0.6417
Alpha: 0.2000, train_score: 0.677, val_score:0.511, train_mse: 0.4289, val_mse:
0.6570
Alpha: 0.3000, train_score: 0.675, val_score:0.495, train_mse: 0.4316, val_mse:
0.6787
Alpha: 0.4000, train_score: 0.673, val_score:0.479, train_mse: 0.4336, val_mse:
0.7002
Alpha: 0.5000, train_score: 0.672, val_score:0.464, train_mse: 0.4352, val_mse:
0.7198
Alpha: 0.6000, train_score: 0.671, val_score:0.451, train_mse: 0.4364, val_mse:
0.7372
Alpha: 0.7000, train_score: 0.670, val_score:0.440, train_mse: 0.4374, val_mse:
0.7526
Alpha: 0.8000, train_score: 0.670, val_score:0.430, train_mse: 0.4383, val_mse:
0.7662
Alpha: 0.9000, train_score: 0.669, val_score:0.421, train_mse: 0.4390, val_mse:
0.7783
Alpha: 1.0000, train_score: 0.669, val_score:0.413, train_mse: 0.4396, val_mse:
0.7890
```

[52]:

```
cvs = cross_val_score(
    make_pipeline(StandardScaler(with_mean=False),
                  Ridge(alpha=0.1)),
    X_poly_train, t_train, cv=4)
print(cvs)
print(f"Mean score: {np.sum(cvs)/len(cvs):.4f}")
print(f"Standard deviation: {np.std(cvs):.4f}")
```

```
[-4.20902366  0.66736505  0.69693884  0.58101288]
Mean score: -0.5659
Standard deviation: 2.1038
```