regularized-regression-sample-solution

June 19, 2023

Ullrich Köthe: Machine Learning Essentials, Summer Semester 2023

Sample solution for Exercise 5

1) Bias and variance of ridge regression (8 points)

Ridge regression solves:

$$\begin{aligned}\hat{\beta}_\tau =& \text{argmin}_\beta ||X\beta - y||_2^2 + \tau||\beta||_2^2 && (1)\\ =& \text{argmin}_\beta \left((X\beta - y)^T(X\beta - y) + \tau\beta^T\beta\right) && (2)\end{aligned}$$

The task is to prove, that:

$$\begin{aligned}\mathbb{E}[\hat{\beta}_\tau] &= S_\tau^{-1}S\beta^* && (3)\\ \text{Cov}[\hat{\beta}_\tau] &= S_\tau^{-1}SS_\tau^{-1}\sigma^2 && (4)\end{aligned}$$

with *scatter matrices* $S = X^TX$ and $S_\tau = X^TX + \tau\mathbb{1}$

Proof: Find the minimal loss with respect to β by calculating $\partial_\beta\text{Loss} \overset{!}{=} 0$:

$$\begin{aligned}\partial_\beta\left((X\beta - y)^T(X\beta - y) + \tau\beta^T\beta\right) &= 0 && (5)\\ \partial_\beta\left((X\beta)^TX\beta - (X\beta)^Ty - y^TX\beta - y^Ty + \tau\beta^T\beta\right) &= 0 && (6)\\ \partial_\beta\left((X\beta)^TX\beta - 2\beta^TX^Ty - y^Ty + \tau\beta^T\beta\right) &= 0 && (7)\\ 2X^TX\hat{\beta} - 2X^Ty + 2\tau\hat{\beta} &= 0 && (8)\\ (X^TX + \tau\mathbb{1})\hat{\beta} &= X^Ty && (9)\\ \hat{\beta} &= S_\tau^{-1}X^Ty && (10)\\ & && (11)\end{aligned}$$

where we abbreviate $\hat{\beta}_\tau$ with $\hat{\beta}$. Replace y with the true model: $y = X\beta^* + \epsilon$

$$\begin{aligned}\hat{\beta} &= S_\tau^{-1}X^T(X\beta^* + \epsilon) && (12)\\ \hat{\beta} &= S_\tau^{-1}S\beta^* + S_\tau^{-1}X^T\epsilon && (13)\end{aligned}$$

Now, take the expectation with respect to ϵ. Use that the expectation is linear and that $\mathbb{E}[\epsilon] = 0$ as ϵ has a 0 mean:

$$\begin{aligned}\mathbb{E}[\hat{\beta}] &= \mathbb{E}[S_\tau^{-1}S\beta^*] + \mathbb{E}[S_\tau^{-1}X^T\epsilon] && (14)\\ &= S_\tau^{-1}S\beta^* + S_\tau^{-1}X^T\mathbb{E}[\epsilon] && (15)\\ &= S_\tau^{-1}S\beta^* \quad \square && (16)\end{aligned}$$

Calculate Covariance:

$$\begin{aligned}\mathrm{Cov}[\hat{\beta}] &= \mathbb{E}\big[(\hat{\beta} - \mathbb{E}[\hat{\beta}]) \cdot (\hat{\beta} - \mathbb{E}[\hat{\beta}])^T\big] && (17)\\ &= \mathbb{E}\big[(\hat{\beta} - S_\tau^{-1}S\beta^*) \cdot (\hat{\beta} - S_\tau^{-1}S\beta^*)^T\big] && (18)\\ &= \mathbb{E}\big[(S_\tau^{-1}S\beta^* + S_\tau^{-1}X^T\epsilon - S_\tau^{-1}S\beta^*) \cdot (S_\tau^{-1}S\beta^* + S_\tau^{-1}X^T\epsilon - S_\tau^{-1}S\beta^*)^T\big] && (19)\\ &= \mathbb{E}\big[(S_\tau^{-1}X^T\epsilon) \cdot (S_\tau^{-1}X^T\epsilon)^T\big] && (20)\\ &= \mathbb{E}\big[S_\tau^{-1}X^T\epsilon\epsilon^T X(S_\tau^{-1})^T\big] && (21)\\ &= S_\tau^{-1}S(S_\tau^T)^{-1}\mathbb{E}[\epsilon\epsilon^T] && (22)\\ &= S_\tau^{-1}SS_\tau^{-1}\sigma^2 \quad \square && (23)\end{aligned}$$

In the last step we used that S_τ is symmetric, because $S_\tau^T = (X^TX + \tau\mathbb{1})^T = (X^TX)^T + \tau\mathbb{1}^T = X^TX + \tau\mathbb{1} = S_\tau$.

2) LDA-Derivation from the Least Squares Error (16 points)

Using the definitions from the task, we set the derivative of the loss with respect to β to zero:

$$\frac{\partial}{\partial\beta}\sum_{i=1}^{N}(y_i^* - X_i \cdot \beta)^2 = \sum_{i=1}^{N} -2X_i^\top(y_i^* - X_i \cdot \beta) \overset{!}{=} 0 \qquad (24)$$

$$\Leftrightarrow \sum_{i=1}^{N} X_i^\top X_i \cdot \beta = \sum_{i=1}^{N} X_i^\top y_i^* \qquad (25)$$

We first transform the RHS of the equation, using the fact that each observation belongs to one class and the definition of the class means:

$$\begin{aligned}\sum_{i=1}^{N} X_i^\top y_i^* &= \sum_{i:y_i^*=1} X_i^\top - \sum_{i:y_i^*=-1} X_i^\top && (26)\\ &= N_1 \cdot \mu_1^\top - N_{-1} \cdot \mu_{-1}^\top && (27)\\ &= \frac{N}{2}(\mu_1 - \mu_{-1})^\top && (28)\end{aligned}$$

Next, we can turn to the LHS of the equation (see below for an alternative solution):

$$\sum_{i=1}^{N} X_i^\top X_i \cdot \beta = \left(\sum_{i=1}^{N} X_i^\top X_i\right) \cdot \beta \qquad (29)$$

We transform the sum (omitting β for now) to obtain the two terms, one proportional to Σ and the other proportional to $(\mu_1 - \mu_{-1})^\top(\mu_1 - \mu_{-1})$. We make use of the fact that the data are centered and the classes are balanced, so $\frac{1}{N}\sum_{i=1}^N X_i = \frac{1}{N}\sum_{i:y_i^*=1} X_i + \frac{1}{N}\sum_{i:y_i^*=-1} X_i = \frac{1}{2}(\mu_1 + \mu_{-1}) = 0$:

$$\begin{aligned}
\sum_{i=1}^N X_i^\top X_i &= \sum_{i=1}^N X_i^\top \left(X_i - \underbrace{\frac{1}{2}(\mu_1 + \mu_{-1})}_{=0} \right) && (30) \\
&= \sum_{i:y_i^*=1} X_i^\top \left(X_i \underbrace{- \mu_1 + \frac{1}{2}(\mu_1 - \mu_{-1})}_{=-\frac{1}{2}(\mu_1+\mu_{-1})=0} \right) + \sum_{i:y_i^*=-1} X_i^\top \left(X_i \underbrace{- \mu_{-1} - \frac{1}{2}(\mu_1 - \mu_{-1})}_{=-\frac{1}{2}(\mu_1+\mu_{-1})=0} \right) && (31) \\
&= \underbrace{\sum_{i=0}^N X_i^\top (X_i - \mu_{y_i^*})}_{(a)} + \underbrace{\left[\sum_{i:y_i^*=1} X_i^\top - \sum_{i:y_i^*=-1} X_i^\top \right] \frac{1}{2}(\mu_1 - \mu_{-1})}_{(b)} && (32)
\end{aligned}$$

As

$$0 = \sum_{i=1}^N -\mu_{y_i^*}^\top (X_i - \mu_{y_i^*}), \text{ because } \sum_{i=1}^N X_i = \sum_{i=1}^N \mu_{y_i^*}$$

we can write

$$\begin{aligned}
(a) = (a) + 0 &= \sum_{i=1}^N X_i^\top (X_i - \mu_{y_i^*}) + \sum_{i=1}^N -\mu_{y_i^*}^\top (X_i - \mu_{y_i^*}) && (33) \\
&= \sum_{i=1}^N (X_i - \mu_{y_i^*})^\top (X_i - \mu_{y_i^*}) = N\Sigma && (34)
\end{aligned}$$

For (b):

$$\left[\sum_{i:y_i^*=1} X_i^\top - \sum_{i:y_i^*=-1} X_i^\top \right] \frac{1}{2}(\mu_1 - \mu_{-1}) = [N_1 \cdot \mu_1^\top - N_{-1} \cdot \mu_{-1}^\top] \frac{1}{2}(\mu_1 - \mu_{-1}) = N\frac{1}{2}(\mu_1 - \mu_{-1})^\top \frac{1}{2}(\mu_1 - \mu_{-1}) \quad (35)$$

Putting it all together, we find equation (2) from the exercise:

$$\begin{aligned}
N\Sigma \cdot \beta + N\frac{1}{4}(\mu_1 - \mu_{-1})^\top(\mu_1 - \mu_{-1}) \cdot \beta &= \frac{N}{2}(\mu_1 - \mu_{-1})^\top && (36) \\
\Leftrightarrow \Sigma \cdot \beta + \frac{1}{4}(\mu_1 - \mu_{-1})^\top(\mu_1 - \mu_{-1}) \cdot \beta &= \frac{1}{2}(\mu_1 - \mu_{-1})^\top && (37)
\end{aligned}$$

Alternative solution for the LHS:

$$\sum_{i=1}^{N} X_i^\top X_i \cdot \beta = \left(\sum_{i=1}^{N} X_i^\top X_i\right) \cdot \beta \tag{38}$$

As each case only belongs to one class, we can split up the sum (ignoring β, which we will reintroduce in the end):

$$\sum_{i=1}^{N} X_i^\top X_i = \sum_{i:y_i^*=-1} X_i^\top X_i + \sum_{i:y_i^*=1} X_i^\top X_i \tag{39}$$

We can expand the factors with the corresponding means to obtain the shared covariance matrix, subtracting the newly introduced terms so that the result doesn't change. Here, we demonstrate it for $y_i^* = -1$, the case for $y_i^* = 1$ can be obtained in the same way.

$$\begin{aligned}
\sum_{i:y_i^*=-1} X_i^\top X_i &= \sum_{i:y_i^*=-1} [(X_i - \mu_{-1})^\top \cdot (X_i - \mu_{-1}) + X_i^\top \mu_{-1} + \mu_{-1}^\top X_i - \mu_{-1}^\top \mu_{-1}] && (40)\\
&= \sum_{i:y_i^*=-1} (X_i - \mu_{-1})^\top \cdot (X_i - \mu_{-1}) + \left(\sum_{i:y_i^*=-1} X_i^\top\right) \mu_{-1} + \mu_{-1}^\top \sum_{i:y_i^*=-1} X_i - \sum_{i:y_i^*=-1} \mu_{-1}^\top \mu_{-1} && (41)\\
&= \sum_{i:y_i^*=-1} (X_i - \mu_{-1})^\top \cdot (X_i - \mu_{-1}) + N_{-1} \cdot \mu_{-1}^\top \mu_{-1} + \underbrace{N_{-1} \cdot \mu_{-1}^\top \mu_{-1} - N_{-1} \cdot \mu_{-1}^\top \mu_{-1}}_{=0} && (42)\\
&= \sum_{i:y_i^*=-1} (X_i - \mu_{-1})^\top \cdot (X_i - \mu_{-1}) + N_{-1} \cdot \mu_{-1}^\top \mu_{-1} && (43)
\end{aligned}$$

Combining the results we obtain

$$\begin{aligned}
\sum_{i=1}^{N} X_i^\top X_i &= \sum_{i:y_i^*=-1} (X_i - \mu_{-1})^\top \cdot (X_i - \mu_{-1}) + \sum_{i:y_i^*=1} (X_i - \mu_1)^\top \cdot (X_i - \mu_1) + N_{-1} \cdot \mu_{-1}^\top \mu_{-1} + N_1 \cdot \mu_1^\top \mu_1 && (44)\\
&= N\Sigma + \frac{N}{2}(\mu_{-1}^\top \mu_{-1} + \mu_1^\top \mu_1). && (45)
\end{aligned}$$

To get the desired result, we have to change the second term to be proportional to $(\mu_{-1} - \mu_1)^\top (\mu_{-1} - \mu_1)$. To get there, we can use the fact that the data are centered and the classes are balanced, so $\frac{1}{N}\sum_{i=1}^{N} X_i = \frac{1}{N}\sum_{i:y_i^*=1} X_i + \frac{1}{N}\sum_{i:y_i^*=-1} X_i = \frac{1}{2}(\mu_1 + \mu_{-1}) = 0$.

$$\frac{N}{2}(\mu_{-1}^\top\mu_{-1}+\mu_1^\top\mu_1) = \frac{N}{2}\left[(\mu_{-1}^\top\mu_{-1}+\mu_1^\top\mu_1) - \underbrace{\frac{1}{2}\mu_{-1}^\top(\mu_{-1}+\mu_1)}_{=0} - \underbrace{\frac{1}{2}\mu_1^\top(\mu_{-1}+\mu_1)}_{=0}\right] \tag{46}$$

$$= \frac{N}{2}\left[\mu_{-1}^\top\left(\mu_{-1}-\frac{1}{2}(\mu_{-1}+\mu_1)\right)+\mu_1^\top\left(\mu_1-\frac{1}{2}(\mu_{-1}+\mu_1)\right)\right] \tag{47}$$

$$= \frac{N}{2}\left[\frac{1}{2}\mu_{-1}^\top(\mu_{-1}-\mu_1)-\frac{1}{2}\mu_1^\top(\mu_{-1}-\mu_1)\right] \tag{48}$$

$$= \frac{N}{4}(\mu_{-1}-\mu_1)^\top(\mu_{-1}-\mu_1) \tag{49}$$

$$\tag{50}$$

Putting it all together, we find equation (2) from the exercise:

$$N\Sigma\cdot\beta + \frac{N}{4}(\mu_1-\mu_{-1})^\top(\mu_1-\mu_{-1})\cdot\beta = \frac{N}{2}(\mu_1-\mu_{-1})^\top \tag{51}$$

$$\Leftrightarrow \Sigma\cdot\beta + \frac{1}{4}(\mu_1-\mu_{-1})^\top(\mu_1-\mu_{-1})\cdot\beta = \frac{1}{2}(\mu_1-\mu_{-1})^\top \tag{52}$$

3) Automatic feature selection for LDA as regression

3.1) Implement Orthogonal Matching Pursuit (8 points)

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

import sklearn
from sklearn.datasets import load_digits
from sklearn import model_selection
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[2]:
```python
def omp_regression(X, y, T):
    """Orthogonal Matching Pursuit.

    Parameters:
    X: N x D matrix
    y: N-dimensional vector
    T: > 0, desired number of non-zero elements in the final solution.
    """
    N, D = X.shape
    # Initialize the sets
    A = []  # active columns
    B = list(range(D))  # inactive columns
    # Initialize the residual
    r = y.copy()
    # Store beta hats in matrix:
    beta = np.zeros((D, T))
    for t in range(T):
        # 1. Calculate the correlations
        corr = np.abs(X[:,B].transpose() @ r)
        # find index of maximum correlation
        corr_max_ind = np.argmax(corr)
        # relate index to column
        j_t = B[corr_max_ind]
        # 2. Move j_t from the inactive set B to the active set A
        A.append(j_t)
        B.remove(j_t)
        # 3. Form the active matrix X_t consisting of all currently active␣
↪columns of X
        X_t = X[:,A]
        # 4. Solve the least squares problem
        beta[A,t] = np.linalg.inv(X_t.transpose() @ X_t) @ X_t.transpose() @ y
        # 5. Update the residual
        r = y - X @ beta[:,t]
    return beta, A
```

3.2) Classification with sparse LDA (8 points)

We reuse the code from Exercise 1 to load the data:

[3]:
```
# Load data
digits = load_digits()
data = digits["data"]
target = digits["target"]
# Data filering
num_1, num_2 = 3, 9
mask = np.logical_or(target == num_1, target == num_2)
data = data[mask]/data.max()  # normalize
target = target[mask]
# Relabel targets
target[target == num_1] = 1
target[target == num_2] = -1
```

To obtain a balanced training and test set, we have to make sure that we have the same number of samples for each class:

[4]:
```
print(np.sum(target))
```

```
3
```

The number of threes (label 1) is higher, so we remove 3 threes from the data set to make it balanced:

[5]:
```
rm_ind = np.nonzero(target == 1)[0][-3:]
target = np.delete(target, rm_ind, axis=0)
data = np.delete(data, rm_ind, axis=0)
```

Now we can split the data into train and test set, using `stratify=y_all` to ensure that train and test set are balanced as well:

[6]:
```
X_all = data
y_all = target
X_train, X_test, y_train, y_test = \
    model_selection.train_test_split(X_all, y_all, test_size=0.4 ,␣
  ↪random_state=0, stratify=y_all)
# We have to make sure that the
print(f"X_train.shape = {X_train.shape}")
print(f"X_test.shape = {X_test.shape}")
print(f"y_train.shape = {y_train.shape}")
print(f"y_test.shape = {y_test.shape}")
print(f"y_test[:10] = {y_test[:10]}")
print(f"sum(y_train) = {np.sum(y_train)}")
print(f"sum(y_test) = {np.sum(y_test)}")
```

```
X_train.shape = (216, 64)
X_test.shape = (144, 64)
y_train.shape = (216,)
y_test.shape = (144,)
y_test[:10] = [-1 -1 -1 -1 -1 -1 -1  1  1  1]
sum(y_train) = 0
sum(y_test) = 0
```

We first do the task for non-standardized data:

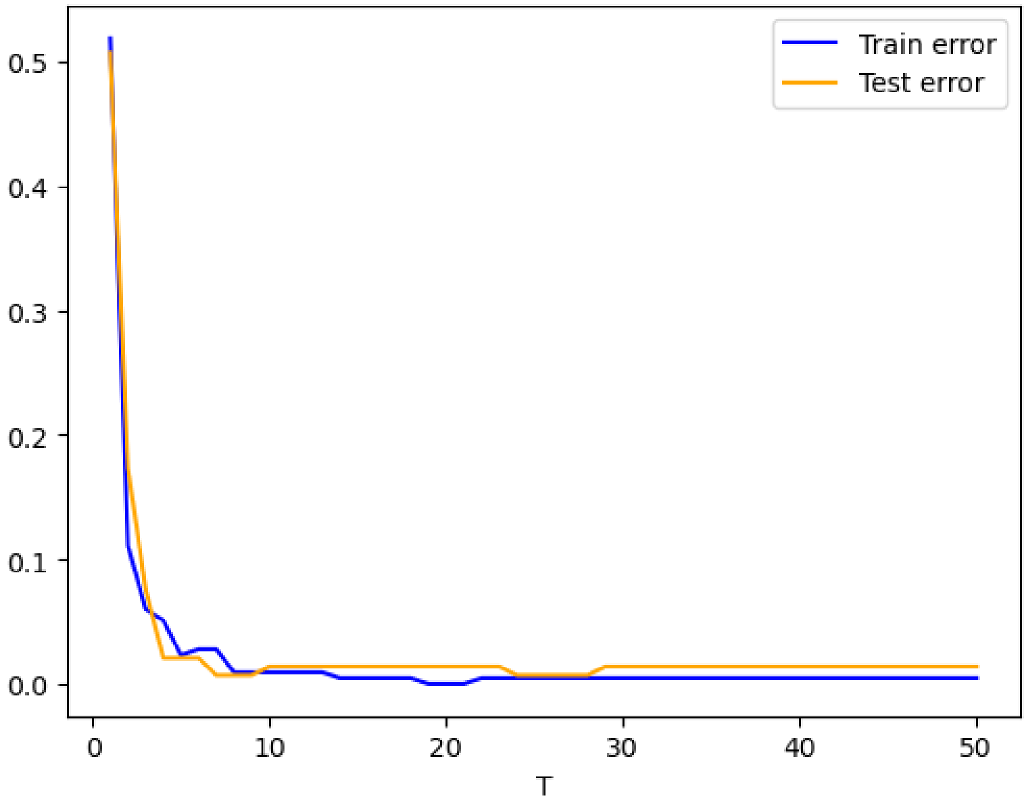
[7]:
```
beta, A = omp_regression(X_train, y_train, 50)
```

[8]:
```
def error_rate(X, y, beta):
    pred = np.sign(X @ beta)
    true = np.stack(beta.shape[1]*[y], axis=1)
    return np.mean(pred != true, axis=0)
```

[9]:
```
error_rate_train = error_rate(X_train, y_train, beta)
error_rate_test = error_rate(X_test, y_test, beta)
```

[10]:
```
plt.plot(range(1, len(error_rate_train)+1), error_rate_train, label="Train␣
  ↪error", color="blue")
plt.plot(range(1, len(error_rate_test)+1), error_rate_test, label="Test error",␣
  ↪color="orange")
plt.xlabel("T")
```

```
_ = plt.legend()
```



For acceptable error rates, at least 5 to 10 pixels should be used.

Now we want to standardize our data. To do so, we have to set the standard deviation of constant pixels to zero. Then we standardize both training and test set with mean and standard deviation from the training set.

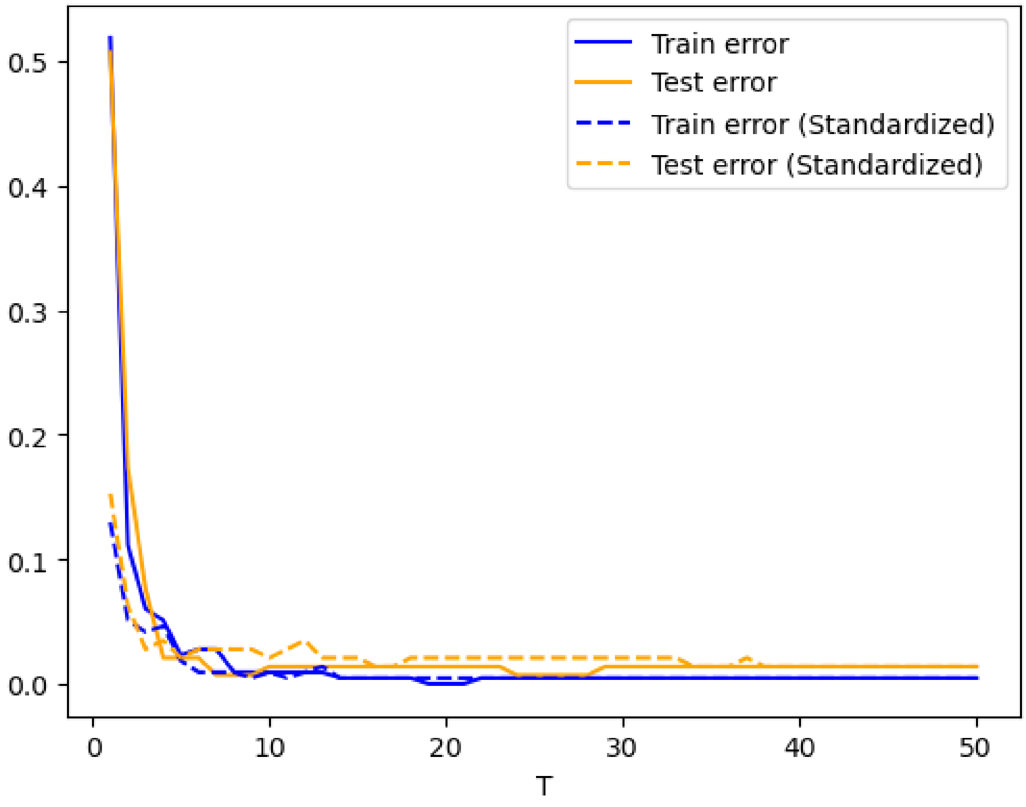
[13]:
```
std_dev = X_train.std(axis=0)
std_dev = np.where(std_dev == 0, 1, std_dev)
mean = X_train.mean(axis=0)

X_train_std = (X_train - mean) / std_dev
X_test_std = (X_test - mean) / std_dev
beta_std, A_std = omp_regression(X_train_std, y_train, 50)
```

[14]:
```
error_rate_train_std = error_rate(X_train_std, y_train, beta_std)
error_rate_test_std = error_rate(X_test_std, y_test, beta_std)
```

[15]:
```
plt.plot(range(1, len(error_rate_train)+1), error_rate_train, label="Train␣
  ↪error", color="blue")
```

```
plt.plot(range(1, len(error_rate_test)+1), error_rate_test, label="Test error",␣
 ↪color="orange")
plt.plot(range(1, len(error_rate_train_std)+1), error_rate_train_std,␣
 ↪label="Train error (Standardized)", linestyle="--", color="blue")
plt.plot(range(1, len(error_rate_test_std)+1), error_rate_test_std, label="Test␣
 ↪error (Standardized)", linestyle="--", color="orange")
plt.xlabel("T")
_ = plt.legend()
```

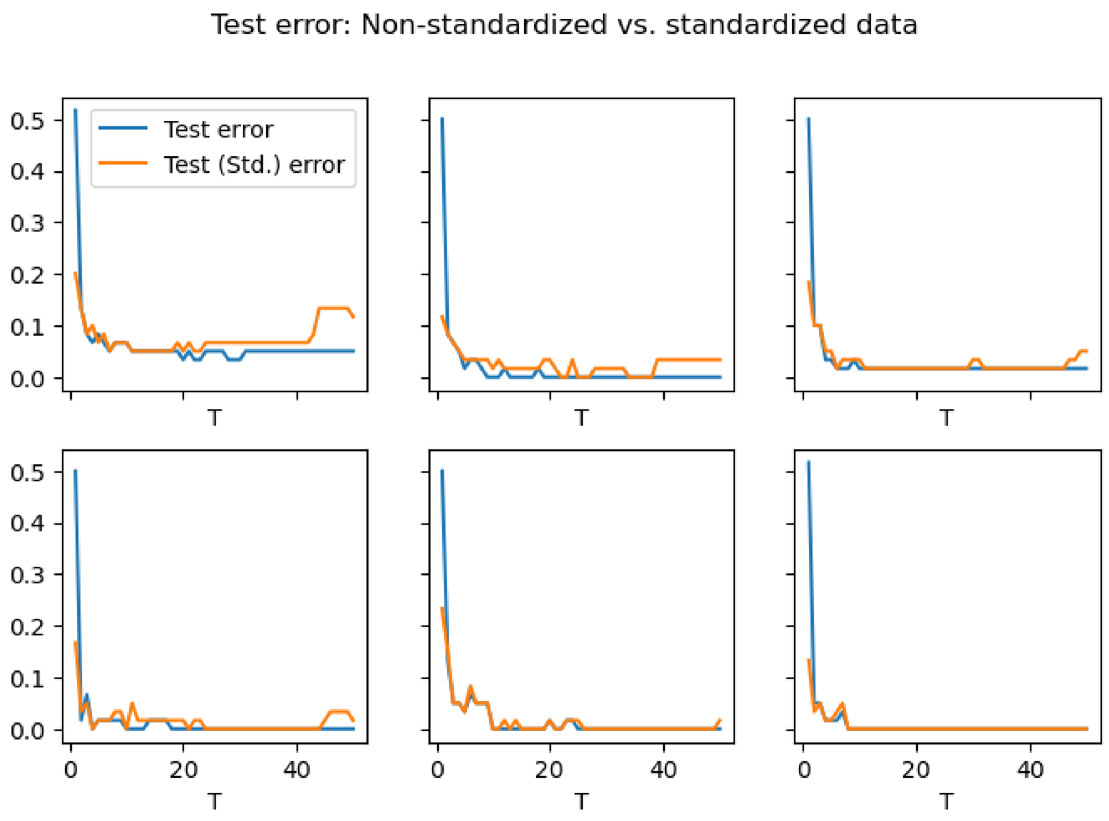


As we can see, standardizing seems to hurt the test error in this case. To check whether this appears to be a general pattern, we can do multiple runs using cross-validation:

[59]:
```
cv = sklearn.model_selection.StratifiedKFold(n_splits=6, shuffle=True,␣
 ↪random_state=2)
```

[60]:
```
error_rates_cv = {'test': [], 'test_std': []}
for i, (train_index, test_index) in enumerate(cv.split(X_all, y_all, y_all)):
    beta_cv, _ = omp_regression(X_all[train_index], y_all[train_index], 50)
    error_rates_cv['test'].append(error_rate(X_all[test_index],␣
 ↪y_all[test_index], beta_cv))
    std_dev_cv = X_all[train_index].std(axis=0)
    std_dev_cv = np.where(std_dev_cv == 0, 1, std_dev_cv)
    mean_cv = X_all[train_index].mean(axis=0)
    beta_cv_std, _ = omp_regression(X_all[train_index], y_all[train_index], 50)
    error_rates_cv['test_std'].append(
        error_rate((X_all[test_index]-mean_cv) / std_dev_cv, y_all[test_index],␣
 ↪beta_cv_std)
    )
```

```
[81]: fig, axs = plt.subplots(2, 3, figsize=(8, 5), sharex=True, sharey=True)
fig.suptitle("Test error: Non-standardized vs. standardized data")
for i, ax in enumerate(axs.flat):
    ax.plot(range(1, len(error_rates_cv['test'][i])+1),␣
 ↪error_rates_cv['test'][i],
            label="Test error")
    ax.plot(range(1, len(error_rates_cv['test_std'][i])+1),␣
 ↪error_rates_cv['test_std'][i],
            label="Test (Std.) error")
    ax.set(xlabel="T")
_ = axs.flat[0].legend(loc='upper right')
```

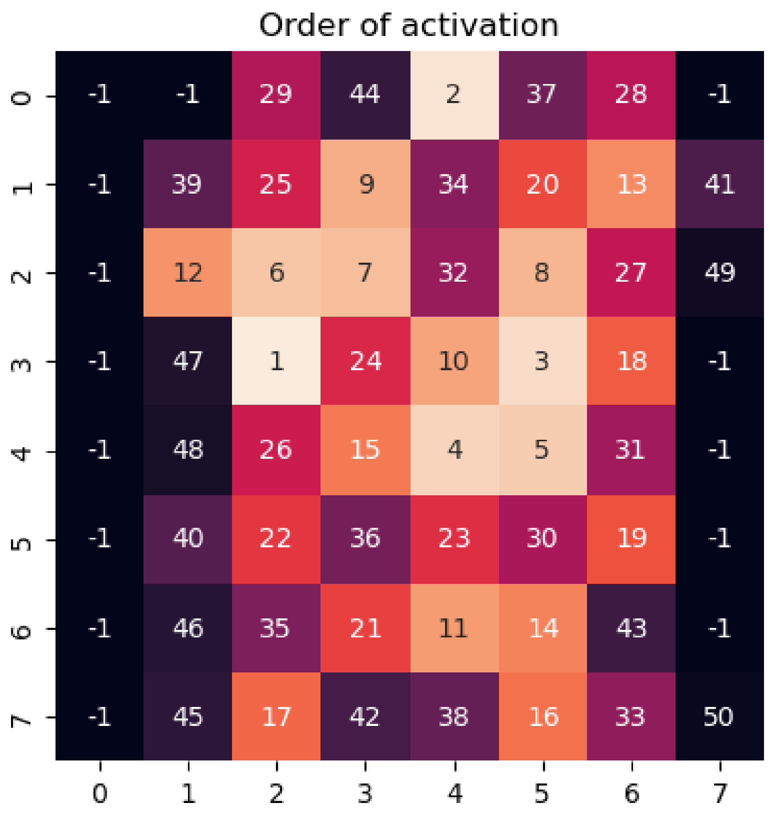


We see the same pattern, so it seems that standardization in fact hurts the performance here, especially for larger T. The reason for this might be that in step 1 of the OMP implementation, we use the (non-standardized) inner product and not the (standardized) correlation. This favors features with higher mean, which in this case are the more reliable features in the center, and suppresses features with low mean, which in our case are probably uninformative and would lead to overfitting. This effect is lost when we standardize the features.

To further investigate this, we can plot the order in which the pixels are switched to active:

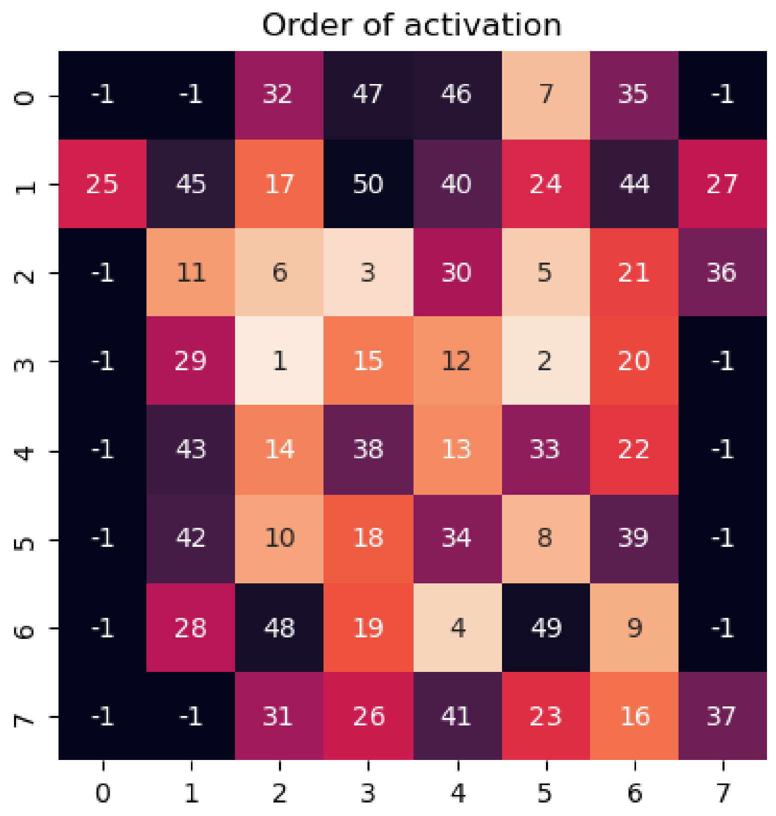
[82]:
```
def plot_activation_order(A):
    brightness = np.zeros(64)
    label = np.zeros(64) - 1
    for i, pixel in enumerate(A):
        brightness[pixel] = len(A)-i  # set brightness (first is brightest)
        label[pixel] = i + 1  # use position as label
    sns.heatmap(np.reshape(brightness, (8, 8)), annot=np.reshape(label, (8,␣
↪8)), cbar=False, square=True)
    _ = plt.title("Order of activation")
```

[85]:
```
plot_activation_order(A)
```



As one might intuitively expect, the pixels on the top left and in the center are the most important. The uninformative pixels at the left and right borders are ignored.

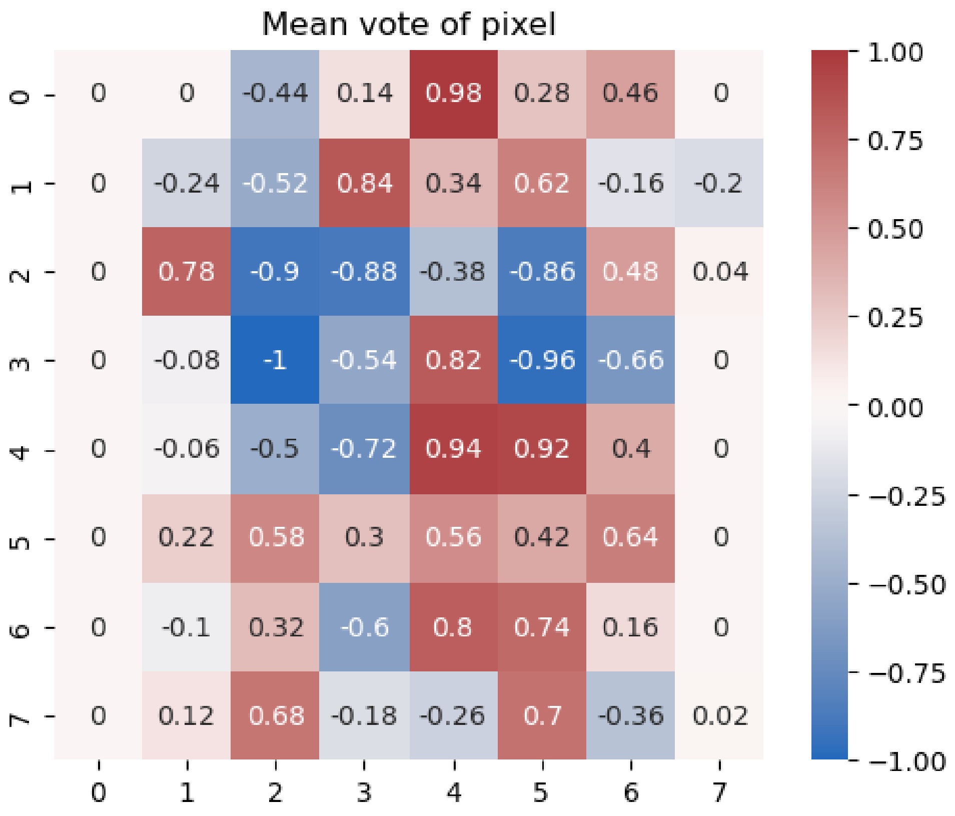
```
[86]: plot_activation_order(A_std)
```



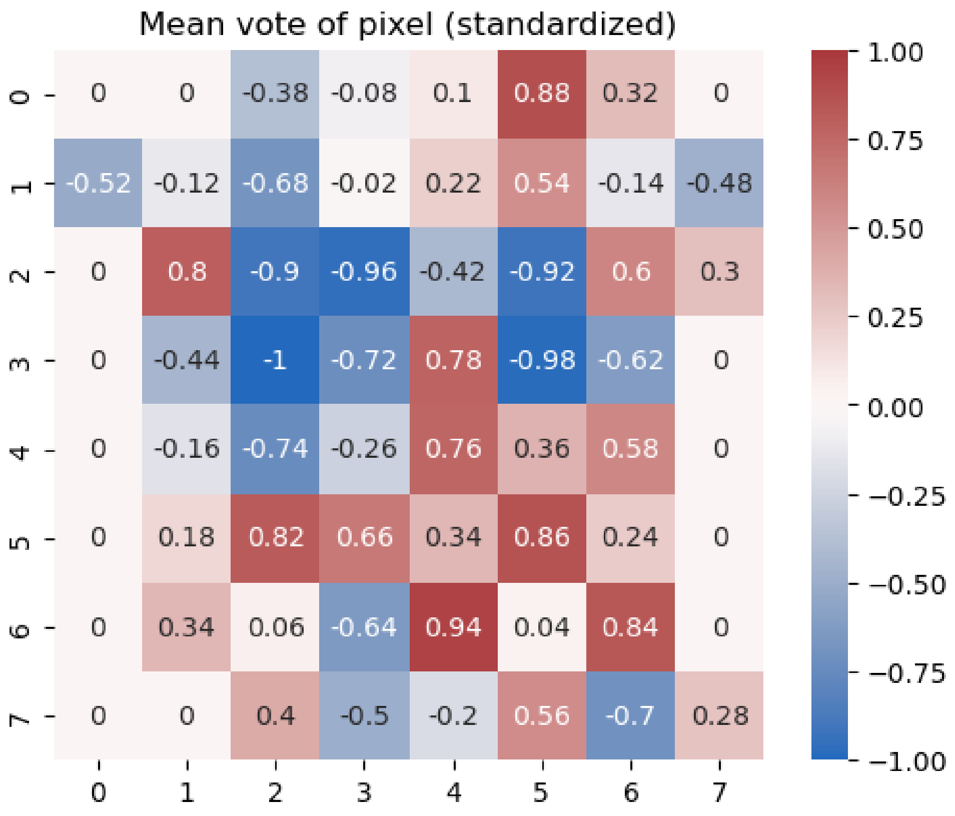
For the standardized data, we can see that peripheral features tend to be added earlier. Also, more pixels from the uninformative left and right borders are included.

A pixel can be considered to vote for ‘3’ (label 1), if the corresponding weight is larger than zero. To summarize the votes for all different T, we can take the mean of the votes. A pixel that voted for ‘3’ for many values of T will have a value close to 1, a pixel that voted for ‘9’ a close to -1. As expected, the line on the left which is only present for nines consistently votes for ‘9’.

```
[89]: sns.heatmap(np.reshape(np.mean(np.sign(beta), axis=1), (8, 8)), annot=True,␣
       ↪cbar=True, square=True, cmap="vlag", vmin=-1, vmax=1)
      _ = plt.title("Mean vote of pixel")
```



[90]:
```
sns.heatmap(np.reshape(np.mean(np.sign(beta_std), axis=1), (8, 8)), annot=True,
            cbar=True, square=True, cmap="vlag", vmin=-1, vmax=1)
_ = plt.title("Mean vote of pixel (standardized)")
```



We can see a similar voting pattern for both non-standardized and standardized data.