3_ex_regularized-regression-commented

June 27, 2023

1 3 Automatic feature selection for LDA as regression

1.1 3.1 Implement Orthogonal Matching Pursuit (8 points)

[1]:
```
import copy
import numpy as np
from scipy.linalg import lstsq
```

Essentially the same solution. However we don't return A. We also preallocate A and B and return early if there is no residual left. We compute the residual and print it at every step. We also use scipy to solve the least squares problem. We compute the inactive betas explicitly.

[2]:
```
def omp_regression(X, y, T):
    """Orthogonal Matching Pursuit is a simple greedy sparse regression␣
↪algorithm. It approximates
    the exact algorithms for least squares under L1 regularization

    :param X: Input Matrix of shape R^NxD
    :param y: Output Vector of shape R^N
    :param T: The desired number of non zero elements in the final solution
    :return beta_ts, optimal_t: The solution weights at each time t and the t␣
↪where the solution is closest to the real solution.
    """

    assert T > 0, "Number of non zero Elements is smaller than 0"

    N = X.shape[0]
    D = X.shape[1]
    assert N == y.shape[0], "Dimension of Inputs and output does not match."

    # axtive matrix X
    X_active = np.zeros(X.shape)
    # inactive matrix
    X_inactive = copy.deepcopy(X)
    # zero matrix for generating inactive matrix
    zero_matrix = np.zeros(X.shape)
    # beta solution list
    beta_ts = np.zeros((D, T))
```

```
    A = np.array(D*[False])
    B = np.array(D*[True])
    r = y
    residual_norms = []

    for t in range(T):
        # 1
        # Find most active column or
        # maximal correlation with the current residual
        correlation = np.abs(np.dot(X.T, r))
        j = np.argmax(correlation)

        # 2
        # set active index to 1
        A[j] = True
        # remove active index from B and set to 0
        B[j] = False

        # 3
        # Select active A
        X_active[:,A] = X[:,A]
        X_inactive[:,A] = zero_matrix[:,A]

        # 4
        # Calculate least squares
        beta_t, residue, rank, singular_value = lstsq(X_active, y)

        # 5
        # Update the residual
        r = y - np.dot(X_active, beta_t)
        residual_norms.append(np.linalg.norm(r))

        # Stop early if solution is found
        if np.sum(r) == 0:
            break

        beta_ts[:, t] = beta_t
        print(f"Distance to optimal solution on train after {t+1} steps:␣
↪{residual_norms[-1]}")
        print("#"*40)

    return beta_ts
```

[3]:
```
X = np.random.randint(5, size=(3,10))
y = np.random.randint(5, size=(3))
T = 3
```

```
[4]: solutions = omp_regression(X, y, T)
```

```
Distance to optimal solution on train after 1 steps: 1.1602387022306428
########################################
Distance to optimal solution on train after 2 steps: 0.5015568278463086
########################################
Distance to optimal solution on train after 3 steps: 1.2947314098277875e-15
########################################
```

1.2 3.2 Classifcation with sparse LDA (8 points)

```
[5]: from sklearn.datasets import load_digits
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

We redistribute the data rather than balance it from the get go. Thus our dataset is complete compared to the sample solution.

```
[6]: def balance_training_split(X_train, X_test, y_train, y_test):
    """
    From a given train test split of binary data, balance training split.
    The split has to contain both labels for this algorithm to work.

    :param X_train: M x D, the training feature instances
    :param X_test: M x D, the test feature instances
    :param y_train: M x 1, the training labels
    :param y_test: M x 1, the test features
    """
    labels = np.unique(y_train)
    assert len(labels) == 2, "The split has to contain both labels for this␣
↪algorithm to work."

    train_class_0_idx = np.where(y_train == labels[0])[0]
    train_class_1_idx = np.where(y_train == labels[1])[0]
    test_class_0_idx = np.where(y_test == labels[0])[0]
    test_class_1_idx = np.where(y_test == labels[1])[0]
    diff = np.abs(len(train_class_0_idx) - len(train_class_1_idx))

    to_test_idx = []
    to_train_idx = []
    if diff % 2 != 0:
        if len(train_class_0_idx) > len(train_class_1_idx):
            to_test_idx.append(train_class_0_idx[0])
            train_class_0_idx = np.delete(train_class_0_idx, 0)
        else:
            to_test_idx.append(train_class_1_idx[0])
            train_class_1_idx = np.delete(train_class_1_idx, 0)
```

```
        diff -= 1

    if diff > 0:
        switched_elements = list(range(int(diff/2)))
        if len(train_class_0_idx) > len(train_class_1_idx):
            to_test_idx += train_class_0_idx[switched_elements].tolist()
            train_class_0_idx = np.delete(train_class_0_idx, switched_elements)
            to_train_idx = test_class_1_idx[switched_elements].tolist()
            test_class_1_idx = np.delete(test_class_1_idx, switched_elements)
        else:
            to_test_idx += train_class_1_idx[switched_elements].tolist()
            train_class_1_idx = np.delete(train_class_1_idx, switched_elements)
            to_train_idx = test_class_0_idx[switched_elements].tolist()
            test_class_0_idx = np.delete(test_class_0_idx, switched_elements)


    y_to_test = y_train[to_test_idx]
    X_to_test = X_train[to_test_idx]
    y_to_train = y_test[to_train_idx]
    X_to_train = X_test[to_train_idx]

    y_train = np.delete(y_train, to_test_idx)
    X_train = np.delete(X_train, to_test_idx, axis=0)
    y_test = np.delete(y_test, to_train_idx)
    X_test = np.delete(X_test, to_train_idx, axis=0)

    y_train = np.concatenate([y_train, y_to_train])
    X_train = np.concatenate([X_train, X_to_train], axis=0)
    y_test = np.concatenate([y_test, y_to_test])
    X_test = np.concatenate([X_test, X_to_test], axis=0)

    return X_train, X_test, y_train, y_test
```

[7]:
```
# load and select data
digits = load_digits()

data = digits["data"]
images = digits["images"]
target = digits["target"]
target_names = digits["target_names"]

# filter data to images with label 3 or 9
idx = (target == 3) | (target == 9)
data_cleaned = data[idx]
images_cleaned = images[idx]
```

```python
target_cleaned = target[idx]

# relabel labels 3 and 9 to -1, 1
idx_target_3 = (target_cleaned == 3)
idx_target_9 = (target_cleaned == 9)
target_cleaned[idx_target_3] = -1
target_cleaned[idx_target_9] = 1

# split the data
X_train, X_test, y_train, y_test = train_test_split(data_cleaned,␣
 ↪target_cleaned)

# balance training data
X_train, X_test, y_train, y_test = balance_training_split(X_train, X_test,␣
 ↪y_train, y_test)
```

The data standardization is basically the same with the exception that we fill NaN values with 0.

```python
[8]: def standardize_data(data: np.ndarray):
         """
         Do data standardization: x_standardized = (x - mean(x)) / std(x)

         :param data: N x D ndarray containing the data to be standarized.
         """
         centralized_data = data - np.mean(data, axis=0)
         std_dev = np.std(data, axis=0)
         standardized_data = centralized_data / std_dev
         # For dimensions with std_dev = 0 choose standardized_data = 0
         # For these entries the Z-score is defined as 0
         zero_std_dev = std_dev == 0
         standardized_data[:, zero_std_dev] = 0
         return standardized_data
```

```python
[9]: data_cleaned_standardized = standardize_data(data_cleaned)

     scaler = StandardScaler()
     data_cleaned_sklearn_standardized = scaler.fit_transform(data_cleaned)

     # verify if our solution is correct
     print(f"Our standard scaling standard deviation and mean:\n{np.
      ↪std(data_cleaned_standardized)}, {np.mean(data_cleaned_standardized)}")
     print(f"Sklearn's standard scaling standard deviation and mean:\n{np.
      ↪std(data_cleaned_sklearn_standardized)}, {np.
      ↪mean(data_cleaned_sklearn_standardized)}")
```

```
Our standard scaling standard deviation and mean:
0.9354143466934854, -1.5292328162880945e-19
Sklearn's standard scaling standard deviation and mean:
```

```
0.9354143466934854, -1.5292328162880945e-19

/tmp/ipykernel_17617/2403962991.py:9: RuntimeWarning: invalid value encountered
in true_divide
  standardized_data = centralized_data / std_dev
```

[10]:
```
# Scale the test and training data
X_standardized_train = standardize_data(X_train)
X_standardized_test = standardize_data(X_test)
```

```
/tmp/ipykernel_17617/2403962991.py:9: RuntimeWarning: invalid value encountered
in true_divide
  standardized_data = centralized_data / std_dev
```

[11]:
```
# for non-standardized images
solutions_digits = omp_regression(X_train, y_train, X_train.shape[1])
```

```
Distance to optimal solution on train after 1 steps: 13.915025701524401
########################################
Distance to optimal solution on train after 2 steps: 8.870110705921896
########################################
Distance to optimal solution on train after 3 steps: 8.540165900419137
########################################
Distance to optimal solution on train after 4 steps: 7.78826891685411
########################################
Distance to optimal solution on train after 5 steps: 7.278822350683555
########################################
Distance to optimal solution on train after 6 steps: 7.002903782491763
########################################
Distance to optimal solution on train after 7 steps: 6.4968017600884105
########################################
Distance to optimal solution on train after 8 steps: 6.263334102419298
########################################
Distance to optimal solution on train after 9 steps: 6.12400397794607
########################################
Distance to optimal solution on train after 10 steps: 5.969128638107378
########################################
Distance to optimal solution on train after 11 steps: 5.787048695539513
########################################
Distance to optimal solution on train after 12 steps: 5.647405689998397
########################################
Distance to optimal solution on train after 13 steps: 5.552547880539769
########################################
Distance to optimal solution on train after 14 steps: 5.488524793585701
########################################
Distance to optimal solution on train after 15 steps: 5.385644913883237
########################################
Distance to optimal solution on train after 16 steps: 5.2882567450780105
########################################
```

```
Distance to optimal solution on train after 17 steps: 5.232814856977235
########################################
Distance to optimal solution on train after 18 steps: 5.1550932048944516
########################################
Distance to optimal solution on train after 19 steps: 5.11441615513987
########################################
Distance to optimal solution on train after 20 steps: 5.0360981650891565
########################################
Distance to optimal solution on train after 21 steps: 4.99392270821011
########################################
Distance to optimal solution on train after 22 steps: 4.958227629304
########################################
Distance to optimal solution on train after 23 steps: 4.904588542992632
########################################
Distance to optimal solution on train after 24 steps: 4.890791791310587
########################################
Distance to optimal solution on train after 25 steps: 4.873410209340849
########################################
Distance to optimal solution on train after 26 steps: 4.844963600900127
########################################
Distance to optimal solution on train after 27 steps: 4.731437635099091
########################################
Distance to optimal solution on train after 28 steps: 4.703736258738175
########################################
Distance to optimal solution on train after 29 steps: 4.6676105825416485
########################################
Distance to optimal solution on train after 30 steps: 4.657558773804667
########################################
Distance to optimal solution on train after 31 steps: 4.653762182764904
########################################
Distance to optimal solution on train after 32 steps: 4.641261631402614
########################################
Distance to optimal solution on train after 33 steps: 4.627736432511158
########################################
Distance to optimal solution on train after 34 steps: 4.612911525714947
########################################
Distance to optimal solution on train after 35 steps: 4.60272483077835
########################################
Distance to optimal solution on train after 36 steps: 4.59984155259062
########################################
Distance to optimal solution on train after 37 steps: 4.59459907238186
########################################
Distance to optimal solution on train after 38 steps: 4.589251670975272
########################################
Distance to optimal solution on train after 39 steps: 4.58889902802033
########################################
Distance to optimal solution on train after 40 steps: 4.57761834864538
########################################
```

```
Distance to optimal solution on train after 41 steps: 4.5728637276361335
#########################################
Distance to optimal solution on train after 42 steps: 4.570672315828692
#########################################
Distance to optimal solution on train after 43 steps: 4.566472907087987
#########################################
Distance to optimal solution on train after 44 steps: 4.566261261848185
#########################################
Distance to optimal solution on train after 45 steps: 4.562445536738479
#########################################
Distance to optimal solution on train after 46 steps: 4.561209584873174
#########################################
Distance to optimal solution on train after 47 steps: 4.560878101173517
#########################################
Distance to optimal solution on train after 48 steps: 4.560733083907693
#########################################
Distance to optimal solution on train after 49 steps: 4.556604320867482
#########################################
Distance to optimal solution on train after 50 steps: 4.551765630197875
#########################################
Distance to optimal solution on train after 51 steps: 4.542792822192767
#########################################
Distance to optimal solution on train after 52 steps: 4.542666652769874
#########################################
Distance to optimal solution on train after 53 steps: 4.5426350748404865
#########################################
Distance to optimal solution on train after 54 steps: 4.542630982972832
#########################################
Distance to optimal solution on train after 55 steps: 4.542630982972832
#########################################
Distance to optimal solution on train after 56 steps: 4.542630982972832
#########################################
Distance to optimal solution on train after 57 steps: 4.542630982972832
#########################################
Distance to optimal solution on train after 58 steps: 4.542630982972832
#########################################
Distance to optimal solution on train after 59 steps: 4.542630982972832
#########################################
Distance to optimal solution on train after 60 steps: 4.542630982972832
#########################################
Distance to optimal solution on train after 61 steps: 4.542630982972832
#########################################
Distance to optimal solution on train after 62 steps: 4.542630982972832
#########################################
Distance to optimal solution on train after 63 steps: 4.542630982972832
#########################################
Distance to optimal solution on train after 64 steps: 4.542630982972832
#########################################
```

[12]:
```
# for standardized images
solutions_standardized_digits = omp_regression(X_standardized_train, y_train,␣
↪X_standardized_train.shape[1])
```

```
Distance to optimal solution on train after 1 steps: 10.881641649376581
########################################
Distance to optimal solution on train after 2 steps: 8.854083917667381
########################################
Distance to optimal solution on train after 3 steps: 7.856843624497157
########################################
Distance to optimal solution on train after 4 steps: 7.338428098819996
########################################
Distance to optimal solution on train after 5 steps: 6.828241236680242
########################################
Distance to optimal solution on train after 6 steps: 6.32592592713819
########################################
Distance to optimal solution on train after 7 steps: 6.16284441377525
########################################
Distance to optimal solution on train after 8 steps: 5.9979524347077104
########################################
Distance to optimal solution on train after 9 steps: 5.897293046572178
########################################
Distance to optimal solution on train after 10 steps: 5.685591858717865
########################################
Distance to optimal solution on train after 11 steps: 5.597920668477057
########################################
Distance to optimal solution on train after 12 steps: 5.4794353322083635
########################################
Distance to optimal solution on train after 13 steps: 5.382611982858704
########################################
Distance to optimal solution on train after 14 steps: 5.261317280688126
########################################
Distance to optimal solution on train after 15 steps: 5.210579188393709
########################################
Distance to optimal solution on train after 16 steps: 5.145102336830046
########################################
Distance to optimal solution on train after 17 steps: 5.065430893356197
########################################
Distance to optimal solution on train after 18 steps: 5.023569972568772
########################################
Distance to optimal solution on train after 19 steps: 4.980650998596544
########################################
Distance to optimal solution on train after 20 steps: 4.915051077477012
########################################
Distance to optimal solution on train after 21 steps: 4.826260971175179
########################################
Distance to optimal solution on train after 22 steps: 4.769694454372308
########################################
```

```
Distance to optimal solution on train after 23 steps: 4.73078877023645
########################################
Distance to optimal solution on train after 24 steps: 4.708621032120979
########################################
Distance to optimal solution on train after 25 steps: 4.680238620073461
########################################
Distance to optimal solution on train after 26 steps: 4.656316678505915
########################################
Distance to optimal solution on train after 27 steps: 4.648475013990084
########################################
Distance to optimal solution on train after 28 steps: 4.6427648908548
########################################
Distance to optimal solution on train after 29 steps: 4.636121621717053
########################################
Distance to optimal solution on train after 30 steps: 4.6251783550768435
########################################
Distance to optimal solution on train after 31 steps: 4.6164504833679185
########################################
Distance to optimal solution on train after 32 steps: 4.604081184697056
########################################
Distance to optimal solution on train after 33 steps: 4.592523984487908
########################################
Distance to optimal solution on train after 34 steps: 4.587910092535218
########################################
Distance to optimal solution on train after 35 steps: 4.5719292908842935
########################################
Distance to optimal solution on train after 36 steps: 4.566867782449537
########################################
Distance to optimal solution on train after 37 steps: 4.558725211229966
########################################
Distance to optimal solution on train after 38 steps: 4.554547762484517
########################################
Distance to optimal solution on train after 39 steps: 4.5384370717818365
########################################
Distance to optimal solution on train after 40 steps: 4.534983873427219
########################################
Distance to optimal solution on train after 41 steps: 4.531974965873104
########################################
Distance to optimal solution on train after 42 steps: 4.528527368837843
########################################
Distance to optimal solution on train after 43 steps: 4.527532655321474
########################################
Distance to optimal solution on train after 44 steps: 4.526831740680015
########################################
Distance to optimal solution on train after 45 steps: 4.525647142647113
########################################
Distance to optimal solution on train after 46 steps: 4.5250904141956285
########################################
```

```
Distance to optimal solution on train after 47 steps: 4.524540391313629
########################################
Distance to optimal solution on train after 48 steps: 4.5243221637308535
########################################
Distance to optimal solution on train after 49 steps: 4.524213414000305
########################################
Distance to optimal solution on train after 50 steps: 4.649242084123436
########################################
Distance to optimal solution on train after 51 steps: 4.649242084123436
########################################
Distance to optimal solution on train after 52 steps: 4.649242084123436
########################################
Distance to optimal solution on train after 53 steps: 4.649242084123436
########################################
Distance to optimal solution on train after 54 steps: 4.649242084123436
########################################
Distance to optimal solution on train after 55 steps: 4.649242084123436
########################################
Distance to optimal solution on train after 56 steps: 4.649242084123436
########################################
Distance to optimal solution on train after 57 steps: 4.649242084123436
########################################
Distance to optimal solution on train after 58 steps: 4.649242084123436
########################################
Distance to optimal solution on train after 59 steps: 4.649242084123436
########################################
Distance to optimal solution on train after 60 steps: 4.649242084123436
########################################
Distance to optimal solution on train after 61 steps: 4.649242084123436
########################################
Distance to optimal solution on train after 62 steps: 4.649242084123436
########################################
Distance to optimal solution on train after 63 steps: 4.649242084123436
########################################
Distance to optimal solution on train after 64 steps: 4.649242084123436
########################################
```

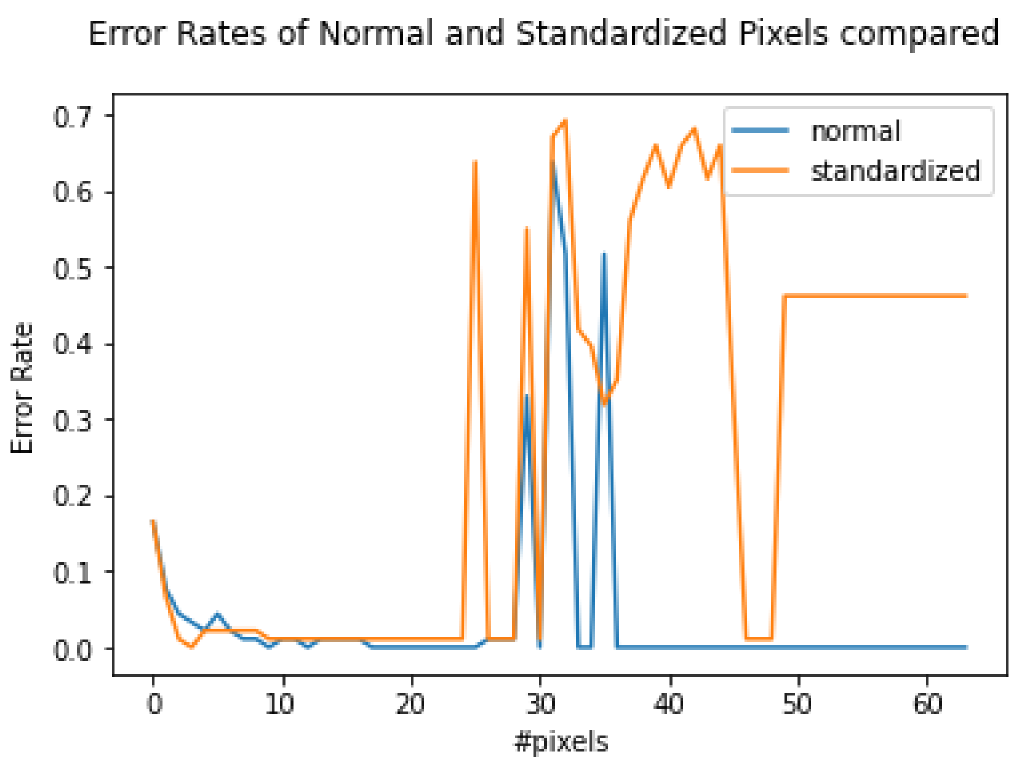
[13]:
```python
def predict(solutions, X_train, X_test):
    # we can assume that the mean over all is the same as the mean over each␣
 ↪class
    # since there are balanced in train
    mu = np.mean(X_train, axis=0)
    lhs = np.dot(X_test, solutions)
    rhs = np.dot(mu, solutions)
    return np.sign(lhs - rhs)

def error_rates(predictions, y_test):
```

```
    errors = np.abs((predictions.T - y_test)/2)
    return np.mean(errors, axis=1)
```

[14]:
```
errors = error_rates(predict(solutions_digits, X_train, X_test), y_test)
error_rates_standardized = error_rates(predict(solutions_standardized_digits,␣
 ↪X_standardized_train, X_standardized_test), y_test)
```

Our prediction method is wrong therefore our error rates look very different to sample solution. Activation order missing.

[15]:
```
plt.plot(list(range(len(errors))), errors, label="normal")
plt.plot(list(range(len(error_rates_standardized))), error_rates_standardized,␣
 ↪label="standardized")
plt.legend()
plt.suptitle("Error Rates of Normal and Standardized Pixels compared")
plt.xlabel("#pixels")
plt.ylabel("Error Rate")
plt.show()
```



Between 10 and 20 pixels we have a high probability to get a perfect classifier. With 3 to 5 pixels we see an error rate of at or below 5 percent.

Standardization did not give us signifcant improvements in classification quality though we suspect that it could be more stable in some scenarios.

Unfortunately, we were not able to finish the pixel visualization in time. We would propose an animated image which activates the pixels according if the respective dimensions in beta_t are nonzero. Then we would look at an averaged 3 image and decide whether the respective beta votes for or against 3 by looking if the average brightness is smaller or larger than some threshold (e.g. 0.2 of the max brightness).

Depending on how well we chose the pixels in exercise 1 we would see overlap with the solution here since OMP will choose the most optimal pixels for the classification task first.