ex06_nico

January 2, 2018

Ullrich Köthe: Fundamentals of Machine Learning, Winter Semester 2017/18 Notebook created by Nicolas Roth

1 Solutions for Exercise 6

```
In [1]: import numpy as np
        %matplotlib inline
        import matplotlib.pyplot as plt

        import pandas as pd
        pd.options.display.float_format = '{:,.2f}'.format
        def fade_zeros(s):
            return ['color: lightgray' if v else 'color: black' for v in (0 == s)]
        from IPython.display import display
```

2 1.) Bias and variance of ridge regression

Probably on blackboard so everyone can follow the calculations...

Ridge regression solves:

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

Prove, that:

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

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

Find the minimum by calculating $\partial_\beta \text{Loss} \stackrel{!}{=} 0$:

$$\begin{aligned} 2X^T X\hat{\beta}_\tau - 2X^T y + 2\tau\hat{\beta}_\tau &= 0 && (5) \\ (X^T X + \tau\mathbb{1})\hat{\beta}_\tau &= X^T y && (6) \\ \hat{\beta}_\tau &= S_\tau^{-1} X^T y && (7) \\ & && (8) \end{aligned}$$

True model: $y = X\beta^* + \epsilon$

$$\hat{\beta}_\tau = S_\tau^{-1} X^T (X\beta^* + \epsilon) \tag{9}$$
$$\hat{\beta}_\tau = S_\tau^{-1} S\beta^* + S_\tau^{-1} X^T \epsilon \tag{10}$$

Now, take the expectancy value:

$$\begin{aligned}
\mathbb{E}[\hat{\beta}] &= \mathbb{E}[S_\tau^{-1} S\beta^*] + \mathbb{E}[S_\tau^{-1} X^T \epsilon] && (11)\\
&= S_\tau^{-1} S\beta^* + S_\tau^{-1} X^T \mathbb{E}[\epsilon] && (12)\\
&= S_\tau^{-1} S\beta^* \quad \square && (13)
\end{aligned}$$

Calculate Covariance:

$$\begin{aligned}
\mathrm{Cov}[\hat{\beta}_\tau] = \mathrm{Cov}[\hat{\beta}_\tau, \hat{\beta}_\tau] &= \mathbb{E}\big[(\hat{\beta}_\tau - \mathbb{E}[\hat{\beta}_\tau]) \cdot (\hat{\beta}_\tau - \mathbb{E}[\hat{\beta}_\tau])^T\big] && (14)\\
&= \mathbb{E}\big[(\hat{\beta}_\tau - S_\tau^{-1} S\beta^*) \cdot (\hat{\beta}_\tau - S_\tau^{-1} S\beta^*)^T\big] && (15)\\
&= \mathbb{E}\big[(S_\tau^{-1} X^T \epsilon) \cdot (S_\tau^{-1} X^T \epsilon)^T\big] && (16)\\
&= \mathbb{E}\big[S_\tau^{-1} X^T \epsilon\epsilon^T X S_\tau^{-1T}\big] && (17)\\
&= S_\tau^{-1} S S_\tau^{-1T} \mathbb{E}[\epsilon\epsilon^T] && (18)\\
&= S_\tau^{-2} S \sigma^2 \quad \square && (19)
\end{aligned}$$

BUT: Why is $[S, S_\tau^{-1T}] = 0$?

$$\begin{aligned}
[S, S_\tau] = [S, S] + \tau[S, \not{1}] &= 0 && (20)\\
[S, \not{1}] = [S, S_\tau S_\tau^{-1}] &= 0 && (21)\\
\Rightarrow \quad [S, S_\tau^{-1}] &= 0 && (22)\\
\overset{sym}{\Rightarrow} \quad [S, S_\tau^{-1T}] &= 0 && (23)
\end{aligned}$$

3 2.) Denoising of a CT image

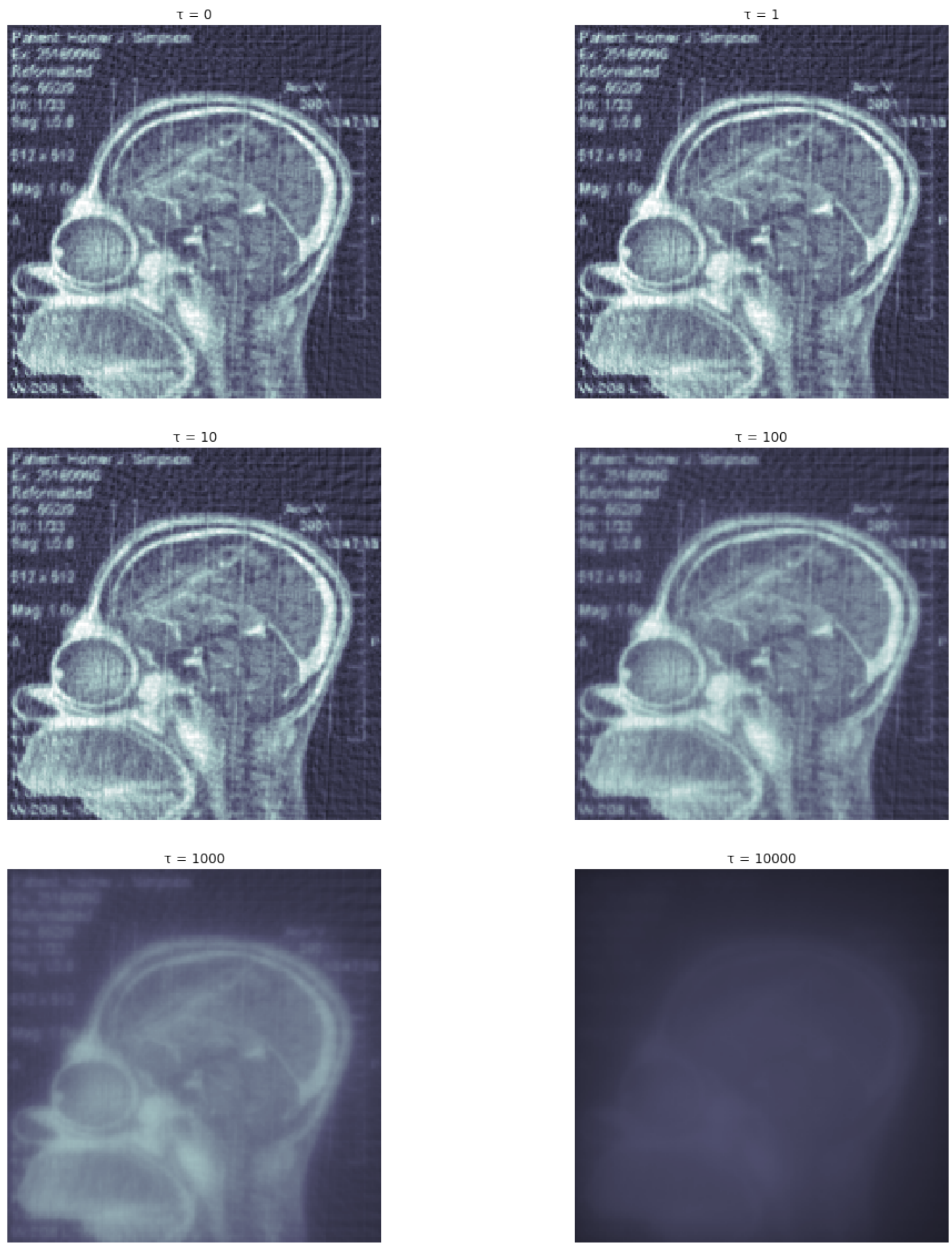
Use solution from last week - with only one small tweak. . .

```
In [2]: from scipy.sparse.linalg import lsqr
        from scipy.sparse import dok_matrix, coo_matrix, vstack, eye

        def construct_X(M, alphas, Np = None, tau=0):
            D = M*M
            # define sensor size
            if Np is None:
                Np = int(np.ceil(np.sqrt(2) * M))
                if Np % 2 == 0: Np += 1
            # number of angles
```

```python
No = len(alphas)

# flattened output coordinates
j = np.mgrid[0:D].astype(np.int32)
# coordinate matrix for the output pixels
M2 = (M-1) / 2
grid = np.mgrid[-M2:M-M2,-M2:M-M2].swapaxes(1,2).reshape(2,D)

# collect indices and corresponding values for all iterations
i_indices = []
j_indices = []
weights = []

for k, alpha in enumerate(alphas):
    # convert angle and prepare projection vector
    alph_rad = np.radians(alpha)
    proj_vec = np.array([np.cos(alph_rad), -np.sin(alph_rad)])
    # project coordinates
    proj = np.dot(proj_vec, grid) + Np // 2
    # compute sensor indices and weights below the projected points
    i = np.floor(proj)
    w = (i+1) - proj

    # make sure rays falling outside the sensor are not counted
    clip = np.logical_and(0 <= i, i < Np-1)

    i_indices.append((i + k*Np)[clip])
    j_indices.append(j[clip])
    weights.append(w[clip])
    # compute sensor indices and weights above the projected points
    w = proj - i
    i_indices.append((i+1 + k*Np)[clip])
    j_indices.append(j[clip])
    weights.append(w[clip])

# construct matrix X
i = np.concatenate(i_indices).astype(np.int32)
j = np.concatenate(j_indices).astype(np.int32)
w = np.concatenate(weights)
X = coo_matrix((w, (i,j)), shape = (No*Np, D), dtype = np.float32)

# append diag(sqrt(tau)) for regularization
if tau > 0:
    reg = np.sqrt(tau)*eye(M*M)
    X = vstack([X,reg])
return X
```

Reconstruct the tomogram for 64 angles and variations of τ

```
In [3]: M = 195
        Np = 275
        y = np.load('hs_tomography/y_195.npy')
        alphas = np.load('hs_tomography/alphas_195.npy')
        # reconstruct for 64 angles
        index = [int(np.ceil(len(alphas) * p/64)) for p in range(64)]
        alphas_sub = alphas[index]
        y_sub = []
        for j in index:
            y_sub.extend(y[j*Np : (j+1)*Np])
        y_sub = np.asarray(y_sub)
        y_sub_tau = np.hstack((y_sub,np.zeros(M**2)))

        tau = [0,1,10,100,1000,10000]
        fig, axes = plt.subplots(3, 2, figsize = (16,16))

        for i in range(len(tau)):
            X = construct_X(M, alphas_sub, Np, tau=tau[i]).tocsc()
            beta = lsqr(X, y_sub if tau[i] == 0 else y_sub_tau, atol = 1e-5, btol =
            axes.flat[i].imshow(beta, vmin = 0, vmax = 255, interpolation = 'neares
            axes.flat[i].set_title('τ = {}'.format(tau[i]))
            axes.flat[i].axis('off')
            print('τ = {}, shape of X = {}, mean GV = {}'.format(tau[i], X.shape, r

        fig.tight_layout()
        plt.show()

τ = 0, shape of X = (17600, 38025), mean GV = 119.92763181263976
τ = 1, shape of X = (55625, 38025), mean GV = 119.91782488461713
τ = 10, shape of X = (55625, 38025), mean GV = 119.82967990942062
τ = 100, shape of X = (55625, 38025), mean GV = 118.95303321615249
τ = 1000, shape of X = (55625, 38025), mean GV = 110.7951105526207
τ = 10000, shape of X = (55625, 38025), mean GV = 65.61257007577262
```

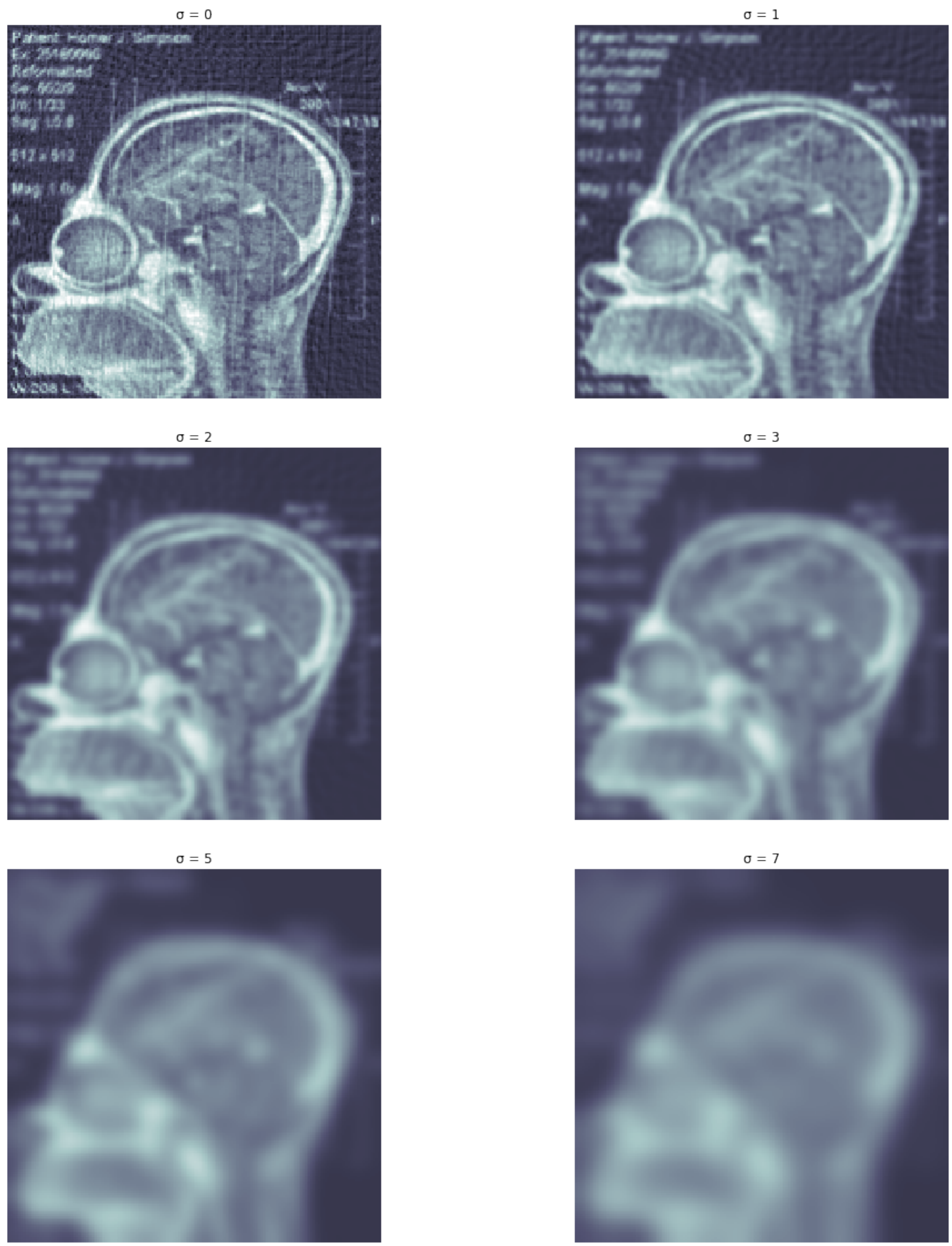


Compare this with gaussian filtering of different standard deviations:

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\frac{x^2}{2\sigma^2}} \tag{24}$$

Apply 1d-filter in all dimensions (truncate filter after 4σ)

In [4]:
```
from scipy.ndimage.filters import gaussian_filter

X = construct_X(M, alphas_sub, Np, tau=0).tocsc()
beta = lsqr(X, y_sub, atol = 1e-5, btol = 1e-5)[0].reshape(195,195)

sigma = [0,1,2,3,5,7]
fig, axes = plt.subplots(3, 2, figsize = (16,16))

for i in range(len(sigma)):
    beta_smooth = gaussian_filter(beta,sigma[i])
    axes.flat[i].imshow(beta_smooth, vmin = 0, vmax = 255, interpolation =
    axes.flat[i].set_title('σ = {}'.format(sigma[i]))
    axes.flat[i].axis('off')
    #print(np.mean(beta_smooth))
fig.tight_layout()
plt.show()
```


σ = 0
Patient: Homer J. Simpson
Reformatted
512 x 512
σ = 1
σ = 2
σ = 3
σ = 5
σ = 7

4 3.) Automatic feature selection for regression

4.1 3.1) Orthogonal Matching Pursuit

Implement OMP according to algorithm given in the lecture:

```
In [5]: from sklearn.datasets import load_digits
        from sklearn.model_selection import train_test_split
        from numpy.linalg import lstsq

        def omp_regression(X, y, T):
            """
            Orthogonal Matching Pursuit iterates T times to automatically find the
            1...T most relevant features in the training set X (standardized!) with
            It returns the optimal weight vector for every iteration with dimension
            """
            # initialization
            dim = X.shape[1]
            beta_hat = np.zeros((dim,T))
            A = []
            B = list(range(X.shape[1]))
            res = y
            # iteration
            for t in range(T): # do iteration
                cor = [np.abs(np.dot(X[:,j].T,res)) for j in B] # 1a.) correlations
                j_max = np.argmax(np.array(cor)) # 1b.) find maximum
                A.append(B.pop(j_max)) # 2.) move most important dim/feature to act
                X_active = X[:,A] # 3.) form the active matrix
                #print(A,len(B))
                beta = lstsq(X_active,y) # 4.) solve least squares problem n
                res = y - np.dot(X_active,beta[0]) # 5.) update the residual
                beta_hat[A,t] = beta[0]
                #print(y)

            return beta_hat
```

4.2 3.2) Classifcation with sparse LDA

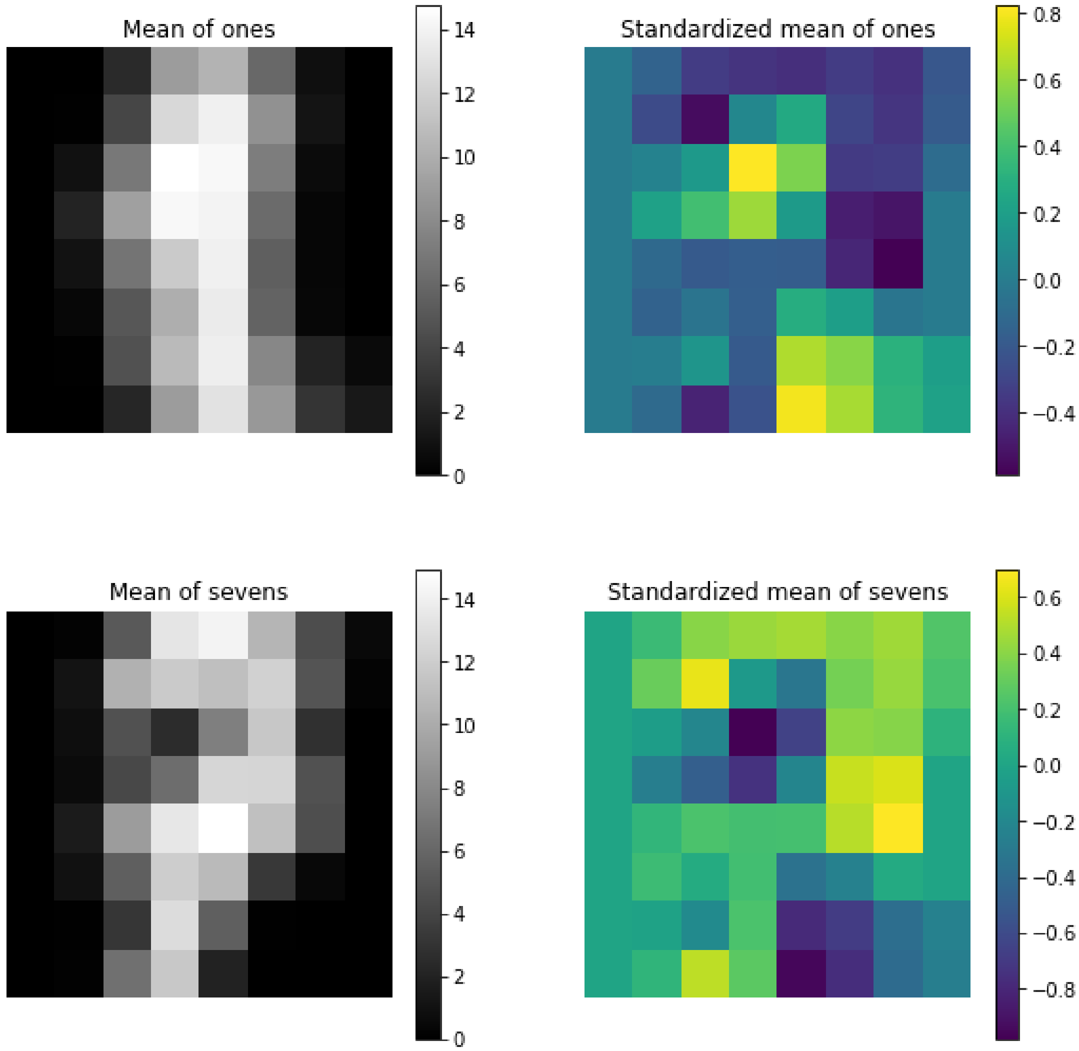
Use results of OMP for linear discrimination - again with the digits 1&7 of the **digits** data set.

```
In [6]: digits = load_digits()
        data = digits["data"]
        target = digits["target"]
        target_names = digits["target_names"]

        def LDA_data(data, target, num_1 = 1, num_2 = 7, test_percentage = 0.33, ra
            """
            This function filters two digits from the dataset, standardizes it and
            """
```

```
    # Data filering
    mask = np.logical_or(target == num_1, target == num_2)
    data = data[mask]#/data.max()
    # data_std = (data -np.mean(data))/np.std(data) # standardize matrix BU
    target = target[mask]

    # Relabel targets
    target[target == num_1] = 1
    target[target == num_2] = -1

    # Random Split
    x_training, x_test, y_training, y_test = train_test_split(data, target,
                                                                rando
    train_std = np.std(x_training, axis=0)+1e-99
    train_mean = np.mean(x_training, axis=0)
    # standardize training data (for every dimension, respectively)
    x_training_std = (x_training - train_mean)/train_std
    # standardize test data in SAME WAY
    x_test_std = (x_test - train_mean)/train_std

    return x_training, x_training_std, x_test, x_test_std, y_training, y_te

x_training, x_training_std, x_test, x_test_std, y_training, y_test = LDA_da
```

Take a look at standardized data

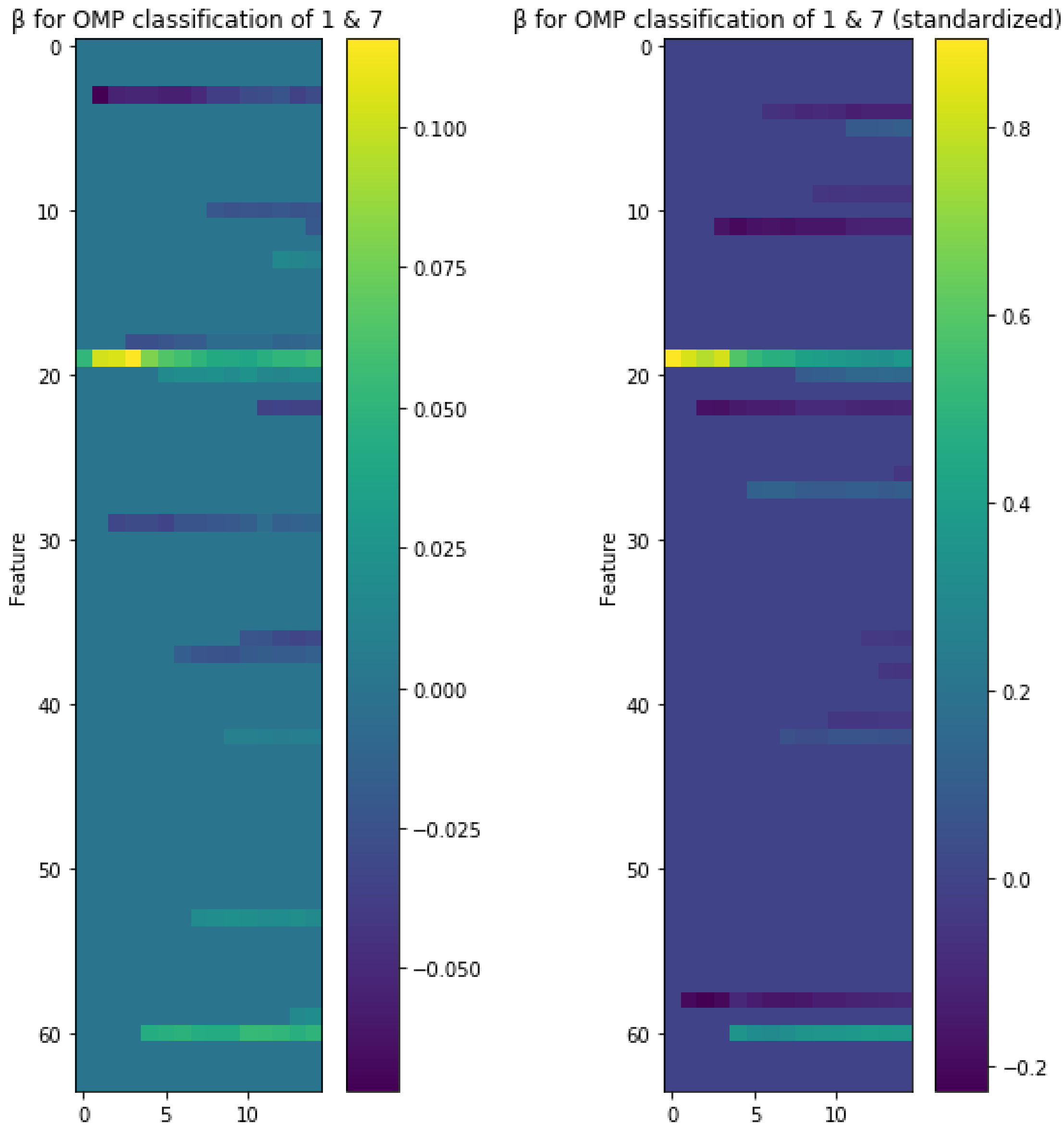
```
In [7]: plt.figure(figsize=(10,10))
        plt.subplot(221)
        plt.imshow(np.array(np.mean(x_training[y_training==1],axis=0)).reshape((8,8
        plt.title('Mean of ones')
        plt.axis('off'); plt.colorbar()
        plt.subplot(222)
        plt.title('Standardized mean of ones')
        plt.imshow(np.array(np.mean(x_training_std[y_training==1],axis=0)).reshape(
        plt.axis('off'); plt.colorbar()
        plt.subplot(223)
        plt.title('Mean of sevens')
        plt.imshow(np.array(np.mean(x_training[y_training==-1],axis=0)).reshape((8,
        plt.axis('off'); plt.colorbar()
        plt.subplot(224)
        plt.title('Standardized mean of sevens')
        plt.imshow(np.array(np.mean(x_training_std[y_training==-1],axis=0)).reshape
        plt.axis('off'); plt.colorbar()
        plt.show()
```



OMP regression for t = 1 . . . 15

```
In [8]: T = 15
        beta = omp_regression(x_training, y_training, T)
        beta_std = omp_regression(x_training_std, y_training, T)

        plt.figure(figsize=(10,10))
        plt.subplot(121)
        plt.ylabel('Feature')
        plt.title('β for OMP classification of 1 & 7')
        plt.imshow(beta)
        plt.colorbar()

        plt.subplot(122)
        plt.ylabel('Feature')
```

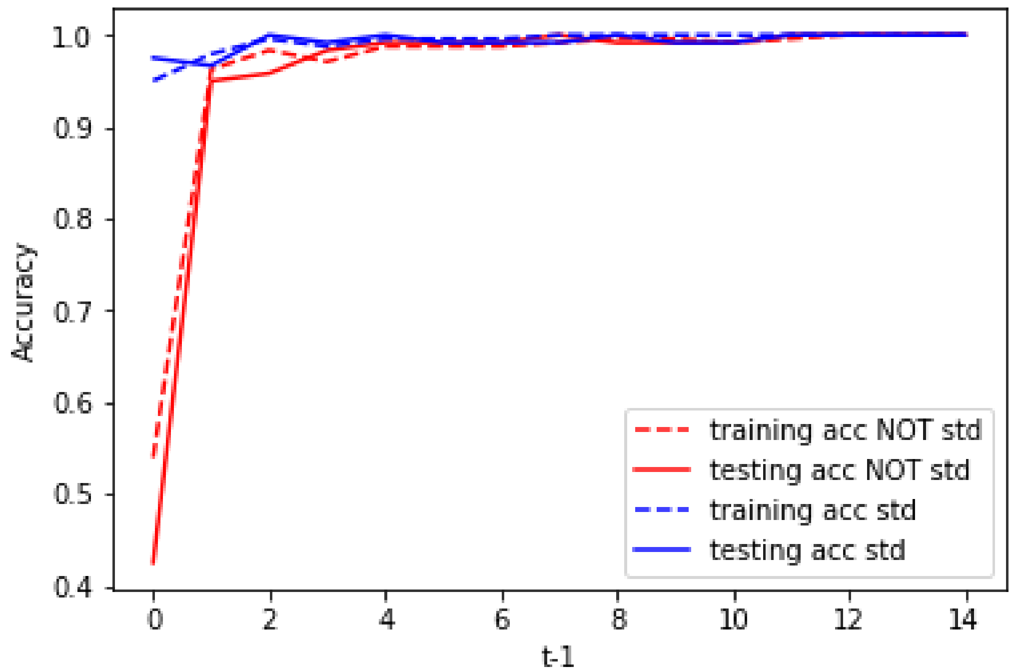
```
plt.imshow(beta_std)
plt.title('$\beta$ for OMP classification of 1 & 7 (standardized)')
plt.colorbar()
plt.show()
```




```
In [9]: def prediction_acc(testset, beta_vector, label):
            #print(testset.shape, beta_vector.shape)
            pred = np.dot(testset,beta_vector)
            pred[np.where(pred < 0)] = -1
            pred[np.where(pred > 0)] = 1
            return np.mean(pred == label)
```

```
accuracy = np.zeros((2,T))
accuracy_std = np.zeros((2,T))
for t in range(T):
    #print(str(t) +' Test acc: '+ str(prediction_acc(x_test,beta[:,t],y_tes
    accuracy[0,t] = prediction_acc(x_training,beta[:,t],y_training)
    accuracy[1,t] = prediction_acc(x_test,beta[:,t],y_test)
    accuracy_std[0,t] = prediction_acc(x_training_std,beta_std[:,t],y_trair
    accuracy_std[1,t] = prediction_acc(x_test_std,beta_std[:,t],y_test)

display(pd.DataFrame(
        data = accuracy,
        index = ['err_train', 'err_test'],
        columns = list(range(1,T+1)))
        .rename_axis('not standard. | t =', axis = 'columns'))

display(pd.DataFrame(
        data = accuracy_std,
        index = ['err_train', 'err_test'],
        columns = list(range(1,T+1)))
        .rename_axis('standardized | t =', axis = 'columns'))

plt.plot(accuracy[0,:],'r',linestyle='--', label='training acc NOT std')
plt.plot(accuracy[1,:], 'r', label='testing acc NOT std')
plt.plot(accuracy_std[0,:],'b',linestyle='--', label='training acc std')
plt.plot(accuracy_std[1,:],'b', label='testing acc std')
plt.ylabel('Accuracy')
plt.xlabel('t-1')
plt.legend()
plt.show()
```

```
not standard. | t =    1     2     3     4     5     6     7     8     9    10    11  \
err_train           0.54  0.96  0.98  0.97  0.99  0.99  0.99  0.99  1.00  1.00  0.99
err_test            0.42  0.95  0.96  0.98  0.99  0.99  0.99  1.00  0.99  0.99  0.99

not standard. | t =   12    13    14    15
err_train           1.00  1.00  1.00  1.00
err_test            1.00  1.00  1.00  1.00


standardized | t =    1     2     3     4     5     6     7     8     9    10    11  \
err_train          0.95  0.98  1.00  0.99  1.00  1.00  1.00  1.00  1.00  1.00  1.00
err_test           0.97  0.97  1.00  0.99  1.00  0.99  0.99  0.99  1.00  0.99  0.99

standardized | t =   12    13    14    15
err_train          1.00  1.00  1.00  1.00
err_test           1.00  1.00  1.00  1.00
```

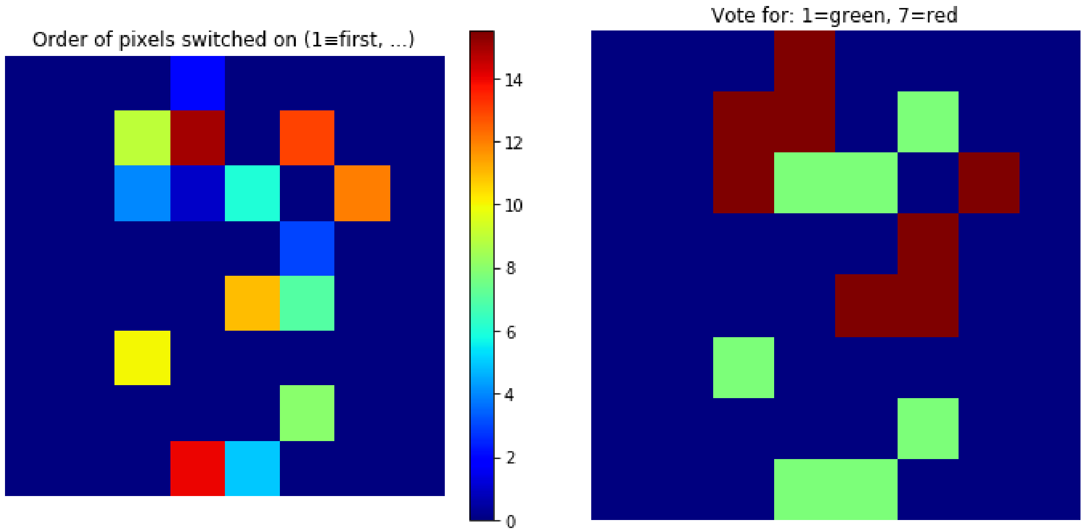


Standardization only makes a difference for small t, especially for $t = 1$ (...which makes sense!)

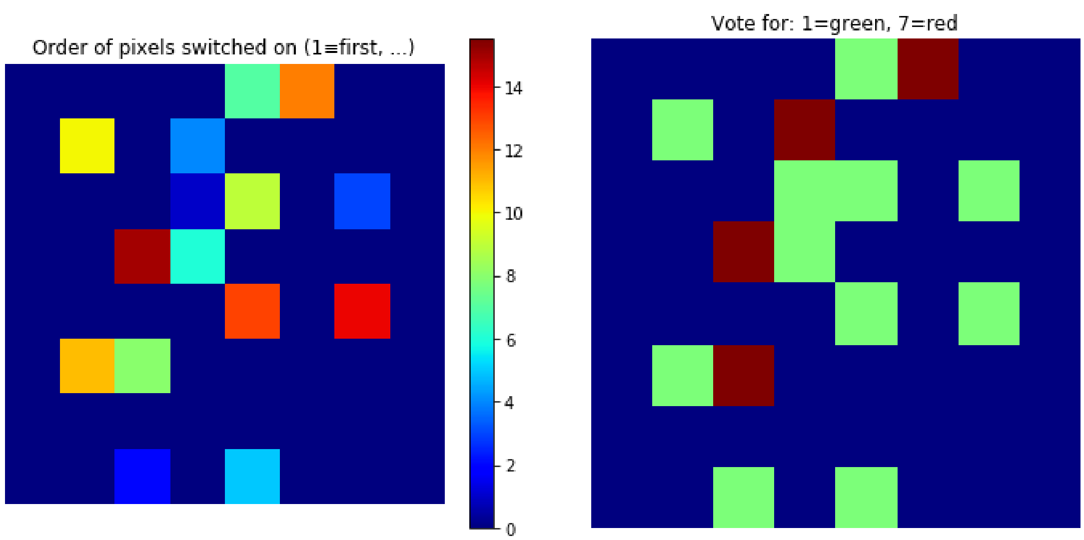
Order of the pixels switched to “active”:

```
In [10]: def vis_pixel(beta, mean=np.ones(64)):
             '''
             Visualizes the order of pixels getting activated and for what number t
             'mean' only has to be given, if the data was standardized (otherwise i
             '''
             old_idx = []
             im = np.zeros((8,8))
             im_v = np.zeros((8,8))
             for j in range(beta.shape[1]):
                 idx = np.where(beta[:,j]!=0)
                 for i in idx[0]:
                     if i not in old_idx:
                         new = i
                         iu = np.unravel_index(i, (8,8))
                         im[iu] = beta.shape[1]+1-j
                         old_idx.append(i)
                         if beta[i,j]*mean[i] > 0:
                             vote = 1
                             im_v[iu] = 1
                         else:
                             vote = 7
```

```
                    im_v[iu] = 2
        plt.figure()
        plt.subplot(121); plt.axis('off')
        plt.title('Add pixel {} ≡ {}'.format(new,iu))
        plt.imshow(im,vmin=0, vmax=beta.shape[1]+1, cmap='jet')
        plt.subplot(122); plt.axis('off')
        plt.title('Votes for {}'.format(vote))
        plt.imshow(im_v,vmin=0,vmax=2, cmap='jet')

def vis_pixel_singleoutput(beta, mean=np.ones(64)):
    '''
    Same as vis_pixel() but only one output image is generated
    (mostly for exporting to pdf)
    '''
    old_idx = []
    im = np.zeros((8,8))
    im_v = np.zeros((8,8))
    for j in range(beta.shape[1]):
        idx = np.where(beta[:,j]!=0)
        for i in idx[0]:
            if i not in old_idx:
                new = i
                iu = np.unravel_index(i, (8,8))
                im[iu] = j+1
                old_idx.append(i)
                if beta[i,j]*mean[i] > 0:
                    vote = 1
                    im_v[iu] = 1
                else:
                    vote = 7
                    im_v[iu] = 2
    plt.figure(figsize=(12,12))
    plt.subplot(121); plt.axis('off')
    plt.title('Order of pixels switched on (1≡first, ...)')
    plt.imshow(im,vmin=0, vmax=beta.shape[1]+0.5, cmap='jet')
    plt.colorbar(fraction=0.05)
    plt.subplot(122); plt.axis('off')
    plt.title('Vote for: 1=green, 7=red')
    plt.imshow(im_v,vmin=0,vmax=2, cmap='jet')

vis_pixel_singleoutput(beta)
```




```
In [11]: vis_pixel_singleoutput(beta_std, np.array(np.mean(x_training_std[y_trainin
```



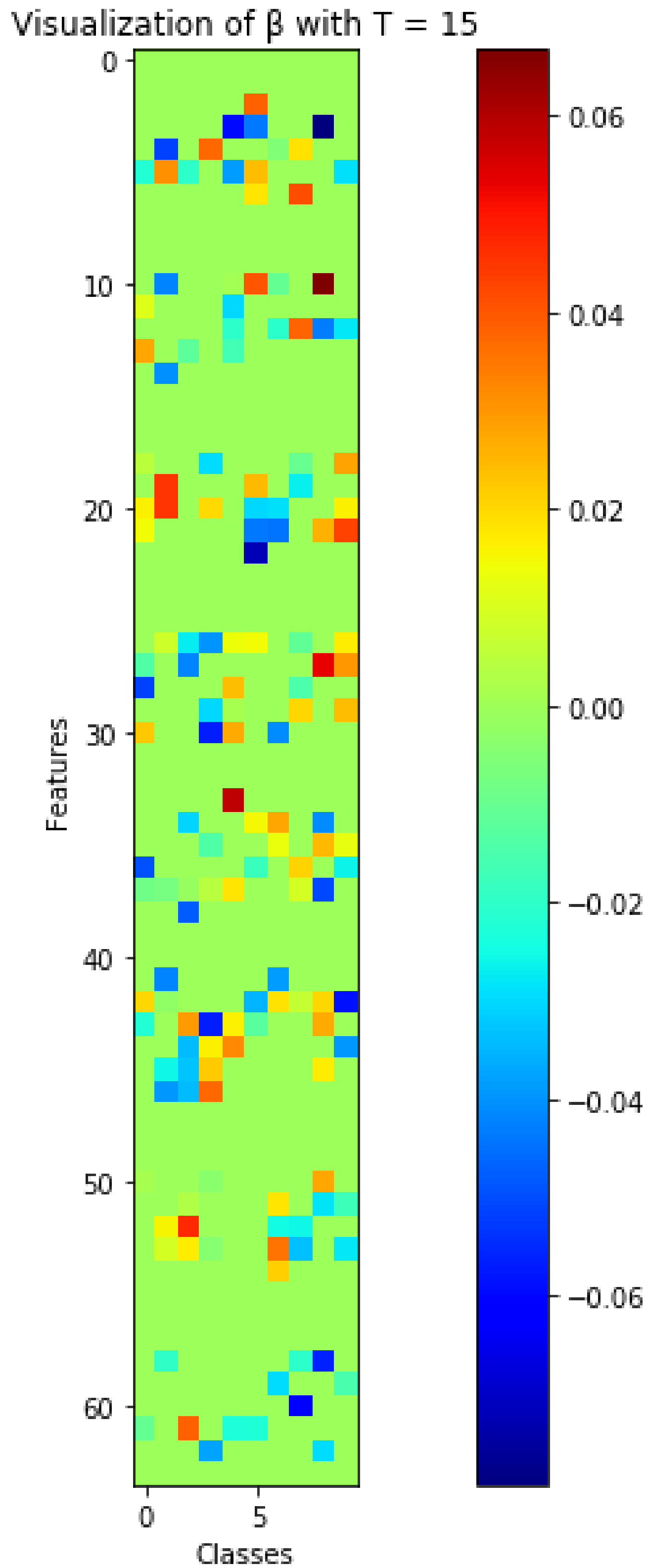
4.3 3.3) One-against-the-rest classifcation

Train C classifiers, trained on C auxillary training sets. Every classifier gives a score for the one class he is trained on. The test instance is assigned the class with the highest score.

```
In [12]: from sklearn.utils import shuffle # shuffle arrays in consistent way
```

```
def OAR_trainingdata(num, train_data, train_target):
    """
    This function filters one digit from the trainingset (label = 1)
    and combines it with the same amount of other digits (label = -1)
    to the auxilary training set for "class = num"
    """
    train_data, train_target = shuffle(train_data, train_target, random_st
    # Data filering for num
    data_num = train_data[train_target == num]
    target_num = train_target[train_target == num]
    # Data filering for other classes
    data_other = train_data[train_target != num]
    data_other = data_other[:data_num.shape[0],:] # slize for balanced tra
    target_other = train_target[train_target != num]
    target_other = target_other[:data_num.shape[0]]

    data_train = np.concatenate((data_num,data_other))
    target_train = np.concatenate((target_num,target_other))

    target_train[target_train != num] = -1
    target_train[target_train == num] = 1

    # Random shuffle
    x_training, y_training = shuffle(data_train, target_train, random_stat
    return x_training, y_training
```

In [13]:
```
T_OAR = 15 #also show for T_OAR = 2, 10, 15

C = target_names # =[0 1 2 3 4 5 6 7 8 9]
X_training, X_test, Y_training, Y_test = train_test_split(data, target, te

beta_classes = np.zeros((data.shape[1],len(C))) #  (D x #classes) matrix
# train every classifier with corresponding auxillary training set
for k in C:
    x_temp, y_temp = OAR_trainingdata(k, X_training, Y_training)
    beta_OAR = omp_regression(x_temp, y_temp, T_OAR)
    beta_classes[:,k] = beta_OAR[:,T_OAR-1]

plt.figure(figsize=(16,10))
plt.imshow(beta_classes,cmap='jet')
plt.title('Visualization of $\beta$ with T = '+str(T_OAR))
plt.xlabel('Classes'); plt.ylabel('Features')
plt.colorbar()
plt.show()
```



Visualization of β with T = 15
Features
Classes
0
10
20
30
40
50
60
0
5
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06

If all scores for a test image are negative, assign it the class “unknown” (“unkn”) to reduce the amount of false positives.

```
In [14]: C_u = np.hstack((C,10))
         class_scores = np.zeros((X_test.shape[0],len(C_u)))

         for k in C:
             class_scores[:,k] = np.dot(X_test,beta_classes[:,k])

         y_predict_u = np.argmax(class_scores,axis=1)
         y_predict = np.argmax(class_scores[:,:len(C)],axis=1)

         acc = np.mean(y_predict == Y_test)
         acc_u = np.mean(y_predict_u == Y_test)

         confusion = np.zeros((len(C_u),len(C)))
         confusion_u = np.zeros((len(C_u),len(C)))
         for i in C_u:
             for j in C:
                 confusion_u[i,j] = np.sum((Y_test == j) * (y_predict_u == i)) / np
                 confusion[i,j] = np.sum((Y_test == j) * (y_predict == i)) / np.sum


         print('Accuracy of OAR classifier (T = {}) without introduction of "unknow
         display(
             pd.DataFrame(data = confusion[:len(C),:], index = C, columns = C)
             .rename_axis('w/o "unkn"', axis = 'columns')
             .style.apply(fade_zeros)
             .format('{0:.2f}%')
         )

         print('Accuracy of OAR classifier (T = {}) with introduction of "unknown":
         display(
             pd.DataFrame(data = confusion_u, index = np.hstack((C,np.nan)), column
             .rename_axis('w/ "unkn"', axis = 'columns')
             .style.apply(fade_zeros)
             .format('{0:.2f}%')
         )
```

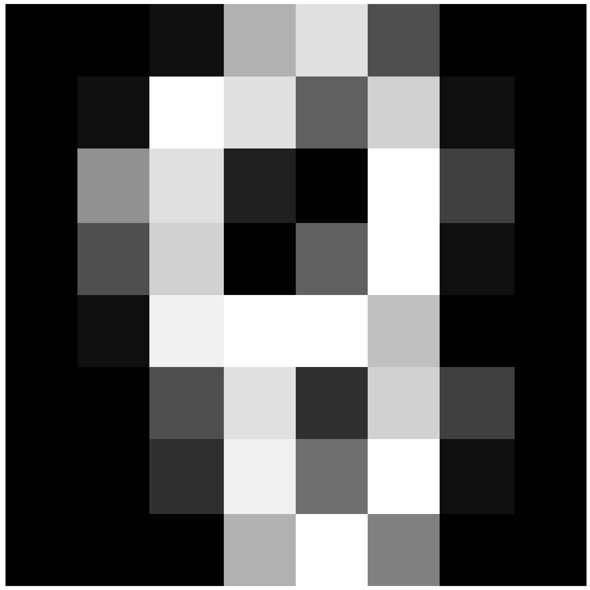
```
Accuracy of OAR classifier (T = 15) without introduction of "unknown": 0.9276094276


<pandas.io.formats.style.Styler at 0x7f71819117b8>


Accuracy of OAR classifier (T = 15) with introduction of "unknown": 0.9276094276094


<pandas.io.formats.style.Styler at 0x7f716e61fdd8>
```

w/o "unkn"	0	1	2	3	4	5	6	7	8	9
0	1.00%	0.00%	0.03%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	0.00%	0.95%	0.00%	0.00%	0.00%	0.00%	0.00%	0.02%	0.07%	0.00%
2	0.00%	0.00%	0.92%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
3	0.00%	0.00%	0.02%	0.91%	0.00%	0.00%	0.00%	0.00%	0.06%	0.02%
4	0.00%	0.00%	0.00%	0.00%	0.94%	0.00%	0.00%	0.04%	0.00%	0.00%
5	0.00%	0.00%	0.00%	0.00%	0.00%	0.92%	0.00%	0.00%	0.01%	0.00%
6	0.00%	0.02%	0.00%	0.00%	0.00%	0.02%	1.00%	0.00%	0.01%	0.00%
7	0.00%	0.00%	0.00%	0.00%	0.02%	0.00%	0.00%	0.91%	0.01%	0.02%
8	0.00%	0.03%	0.03%	0.07%	0.02%	0.00%	0.00%	0.00%	0.82%	0.05%
9	0.00%	0.00%	0.00%	0.02%	0.02%	0.06%	0.00%	0.04%	0.00%	0.92%

w/ "unkn"	0	1	2	3	4	5	6	7	8	9
0.0	1.00%	0.00%	0.03%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1.0	0.00%	0.95%	0.00%	0.00%	0.00%	0.00%	0.00%	0.02%	0.07%	0.00%
2.0	0.00%	0.00%	0.92%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
3.0	0.00%	0.00%	0.02%	0.91%	0.00%	0.00%	0.00%	0.00%	0.06%	0.02%
4.0	0.00%	0.00%	0.00%	0.00%	0.94%	0.00%	0.00%	0.04%	0.00%	0.00%
5.0	0.00%	0.00%	0.00%	0.00%	0.00%	0.92%	0.00%	0.00%	0.01%	0.00%
6.0	0.00%	0.02%	0.00%	0.00%	0.00%	0.02%	1.00%	0.00%	0.00%	0.00%
7.0	0.00%	0.00%	0.00%	0.00%	0.02%	0.00%	0.00%	0.91%	0.01%	0.02%
8.0	0.00%	0.03%	0.03%	0.05%	0.02%	0.00%	0.00%	0.00%	0.82%	0.05%
9.0	0.00%	0.00%	0.00%	0.02%	0.02%	0.06%	0.00%	0.04%	0.00%	0.92%
nan	0.00%	0.00%	0.00%	0.02%	0.00%	0.00%	0.00%	0.00%	0.01%	0.00%

In [15]:
```
unknowns = list(np.where(y_predict_u == C_u[-1])[0])

for u in unknowns:
    plt.figure()
    plt.imshow(X_test[u,:].reshape(8,8), cmap='gray')
    plt.axis('off')
    plt.title('Test image {}, if not "unkn": prediction = {}, true label =
    plt.show()
```

Test image 156, if not "unkn": prediction = 6, true label = 8



Test image 509, if not "unkn": prediction = 8, true label = 3