

Machine Learning for Computer Vision

Exercise 5

Kodai Matsuoka, Yuyan Li

26 May, 2017

1 Features

Our unary and Potts features are computed with the code in Listing 1.

Listing 1: Feature functions (lines 30 - 63) in StructLearn.py

```
def get_unary_features(raw):
    features = []

    #####
    # ADD YOUR CODE HERE
    #####

    raw = norm01(raw)
    sigmas = [0.5, 1.0, 1.5, 2.0, 3.0]
    for sigma in sigmas:
        feat = skimage.filters.gaussian(raw, sigma=sigma)
        features.append(feat[:, :, None])

    features.append(numpy.ones(raw.shape)[:, :, None])
    return numpy.concatenate(features, axis=2)

def get_potts_features(raw):
    features = []

    #####
    # ADD YOUR CODE HERE
    #####

    raw = norm01(raw)
    sigmas = [0.5, 1.0, 1.5, 2.0, 3.0]
    for sigma in sigmas:
        smooth = skimage.filters.gaussian(raw, sigma=sigma)
        edge = skimage.filters.laplace(smooth)
        feat = numpy.exp(-1.0*numpy.abs(edge))
```

```

        features.append(feats[:, :, None])

# a constant feature is needed
        features.append(numpy.ones(raw.shape)[:, :, None])
    return numpy.concatenate(features, axis=2)

```

2 Test set performance with GraphCut

The full code is at the bottom in Listing 3. We have a variable `mode` for the different exercises.

For this part we chose `mode='gp'` to use graph cut.

The computed loss values for different noises and regularizers are:

```

[[[ 7 15  1 10  8]
  [ 7 13  2  8  8]
  [ 7 13  2  9  9]
  [ 7 14  1 10  9]
  [ 7 14  1 10  9]]

[[16 14 17 15 19]
 [15 15 17 19 20]
 [16 15 12 19 20]
 [16 16 13 20 20]
 [16 15 13 20 20]]

[[10 12 16 10 11]
 [17 14 22 10 15]
 [17 14 23 10 15]
 [17 14 22 10 15]
 [19 14 22 10 15]]

[[22 18 16 21 15]
 [20 21 19 24 16]
 [20 23 19 22 18]
 [20 23 19 24 18]
 [20 23 19 24 18]]

[[40 25 18 32 22]
 [40 30 18 41 32]
 [37 28 18 40 38]
 [41 30 17 42 43]
 [41 30 17 42 43]]

```

The content of this list structure is:

- every row is the loss of the five test images of a certain regularizer C and noise
- every list of five rows belong to a certain noise

The noises and regularizer values are sorted as given in the exercise.

As expected, a bigger noise leads to worse predictions and therefore higher losses.

3 Test set performance with ICM

The same is done using an ICM solver instead of GraphCut by setting `mode='icm'`.

```
[[[ 5  5  6  9  5]
   [ 5  5  6  9  5]
   [ 5  5  6  8  5]
   [ 5  5  6  8  5]
   [ 5  5  6  8  5]]]
```

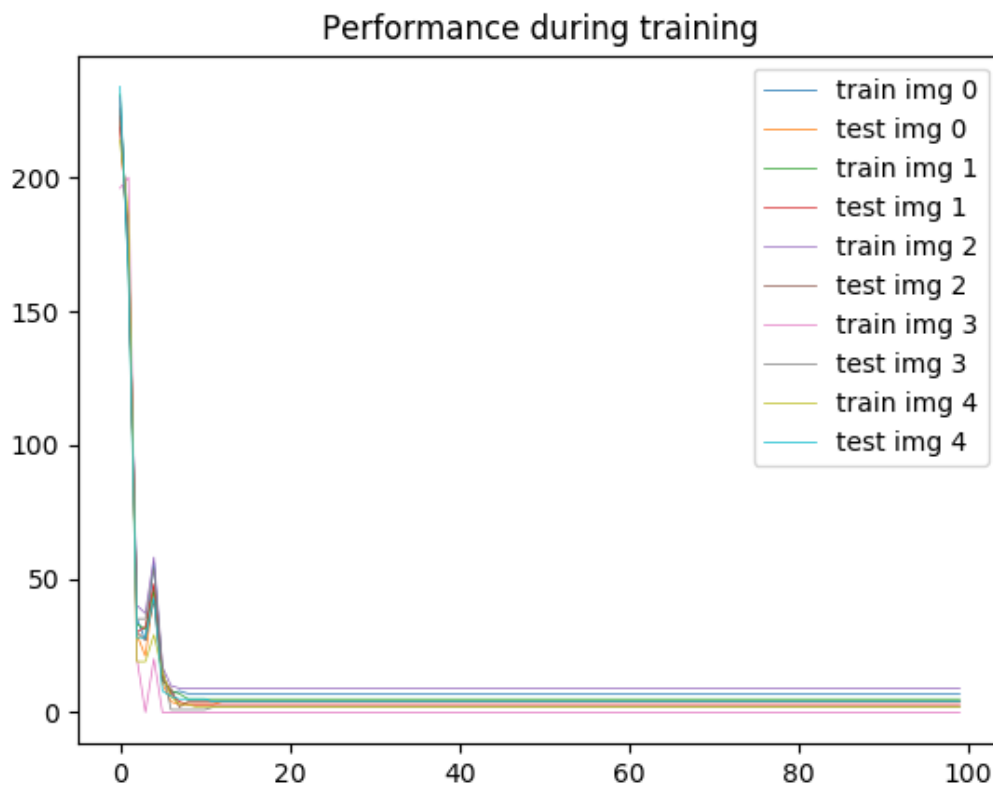
```
[[ 1 10 12 14  0]
 [ 5 11 12 15  0]
 [ 4 12 11 14  0]
 [ 5 11 13 15  0]
 [ 5 11 13 14  0]]]
```

```
[[16 12 12 17 15]
 [17 13 22 18 14]
 [17 13 21 19 14]
 [17 13 22 20 14]
 [17 13 22 20 14]]]
```

```
[[21 22 22 17 16]
 [22 25 20 22 17]
 [23 27 20 22 16]
 [22 24 19 22 16]
 [22 24 18 22 16]]]
```

```
[[56 46 21 26 34]
 [40 24 34 14 60]
 [46 24 43 17 60]
 [46 24 55 19 60]
 [44 26 44 21 60]]]
```

Qualitatively, there does not seem to be a difference to the GraphCut results.



4 Bonus: show performance during training

With the code in Listing 2 (part of `subgradient_ssvm`) the training and test set performances are shown after every iteration during training.

If `mode='bonus'` then after every training a plot with the losses is shown. One exemplary plot can be seen in Section 4.

Listing 2: Code to show performance after every iteration during training (lines 243 - 273 in `StrucLearn.py`).

```
# =====
# show performance during training
# =====

losses = []
for gm, _, loss_function in dataset.models_train:
    gm.change_weights(weights)
    graphcut = GraphCut(model=gm)
    y_pred = graphcut.optimize()
    losses.append(loss_function(y_pred))
loss_train.append(losses)
print('training set performance', losses)
```

```

losses = []
for gm,_,loss_function in dataset.models_test:
    gm.change_weights(weights)
    graphcut = GraphCut(model=gm)
    y_pred = graphcut.optimize()
    losses.append(loss_function(y_pred))
loss_test.append(losses)
print('test set performance', losses)

# in bonus mode: make a plot of performance
if mode=='bonus':
    plottrain = numpy.array(loss_train).T
    plottest = numpy.array(loss_test).T
    for i in range(plottrain.shape[0]):
        plt.plot(range(n_iter),plottrain[i], lw=0.5, label=f'train img {i}')
        plt.plot(range(n_iter),plottest[i], lw=0.5, label=f'test img {i}')
    plt.title('Performance during training')
    plt.legend()

```

Listing 3: StructLearn.py

```

1  # Structured Learning:
2  # =====
3  #
4  # In this exercise we will implement a structured learning system for
5  # foreground background segmentation.
6  # We will learn the weights of a CRF Potts model.
7  #
8  # The first step is to import all needed modules
9
10 # misc
11 import numpy
12 import sys
13
14 # visualization
15 import matplotlib.pyplot as plt
16 import pylab
17
18 # features
19 import skimage.filters
20
21 # discrete graphical model package
22 from dgm.models import *
23 from dgm.solvers import *
24 from dgm.value_tables import *
25 # misc. tools
26 from tools import make_toy_dataset, norm01
27
28 from skimage.morphology import disk
29
30 def get_unary_features(raw):
31     features = []
32
33     #####
34     # ADD YOUR CODE HERE
35     #####
36
37     raw = norm01(raw)
38     sigmas = [0.5, 1.0, 1.5, 2.0, 3.0]
39     for sigma in sigmas:
40         feat = skimage.filters.gaussian(raw, sigma=sigma)
41         features.append(feat[:, :, None])
42
43     features.append(numpy.ones(raw.shape)[:, :, None])
44     return numpy.concatenate(features, axis=2)
45
46
47 def get_potts_features(raw):
48     features = []
49
50     #####
51     # ADD YOUR CODE HERE

```

```

52 #####
53 raw = norm01(raw)
54 sigmas = [0.5, 1.0, 1.5, 2.0, 3.0]
55 for sigma in sigmas:
56     smooth = skimage.filters.gaussian(raw, sigma=sigma)
57     edge = skimage.filters.laplace(smooth)
58     feat = numpy.exp(-1.0*numpy.abs(edge))
59     features.append(feat[:, :, None])
60
61     # a constant feature is needed
62     features.append(numpy.ones(raw.shape)[:, :, None])
63     return numpy.concatenate(features, axis=2)
64
65
66 class HammingLoss(object):
67     def __init__(self, y_true):
68         self.y_true = y_true.copy()
69
70     def __call__(self, y_pred):
71         """total loss"""
72         return numpy.sum(self.y_true!=y_pred)
73
74
75 def build_model(raw_data, gt_image, weights):
76     shape = raw_data.shape
77     n_var = shape[0] * shape[1]
78     n_labels = 2
79     variable_space = numpy.ones(n_var)*n_labels
80
81     # lets compute some filters for the unary features
82     unary_features = get_unary_features(raw_data)
83
84     # lets compute some filters for the potts features
85     potts_features = get_potts_features(raw_data)
86
87     n_weights = potts_features.shape[2] + unary_features.shape[2]
88
89     #assert n_weights == len(weights)
90
91     # both graphical models
92     gm = WeightedDiscreteGraphicalModel(variable_space=variable_space,
93         weights=weights)
94     loss_augmented_gm = WeightedDiscreteGraphicalModel(variable_space=
95         variable_space, weights=weights)
96
97     # convert coordinates to scalar
98     def vi(x0,x1):
99         return x1 + x0*shape[1]
100
101     # weight ids for the unaries (just plain numbers to remeber which
102     weights are associated with the unary features)
103     weight_ids = numpy.arange(unary_features.shape[2])

```

```

101 for x0 in range(shape[0]):
102     for x1 in range(shape[1]):
103
104         pixel_val = raw_data[x0, x1]
105         gt_label = gt_image[x0, x1]
106         features = unary_features[x0, x1, :]
107
108         unary_function = WeightedTwoClassUnary(features=features,
109                                                 weight_ids=weight_ids,
110                                                 weights=weights)
111
112         if gt_label == 0:
113             loss = numpy.array([0,1])
114         else:
115             loss = numpy.array([1,0])
116
117         loss_augmented_unary_function = WeightedTwoClassUnary(features
118                                                                 =features, weight_ids=weight_ids,
119                                                                 weights=weights,
120                                                                 const_terms=-1.0*
121                                                                 loss)
122
123         variables = vi(x0,x1)
124         gm.add_factor(variables=variables, value_table=unary_function)
125         loss_augmented_gm.add_factor(variables=variables, value_table=
126                                     loss_augmented_unary_function)
127
128     # average over 2 coordinates to extract feature vectors for potts
129     funcsins
130     def get_potts_feature_vec(coord_a, coord_b):
131
132         fa = potts_features[coord_a[0],coord_a[1],:]
133         fb = potts_features[coord_b[0],coord_b[1],:]
134         return (fa+fb)/2.0
135
136     # weight ids for the potts functions (just plain numbers to remeber
137     which weights are associated with the potts features)
138     weight_ids = numpy.arange(potts_features.shape[2]) + unary_features.
139     shape[2]
140
141     for x0 in range(shape[0]):
142         for x1 in range(shape[1]):
143
144             # horizontal edge
145             if x0 + 1 < shape[0]:
146                 variables = [vi(x0,x1),vi(x0+1,x1)]
147                 features = get_potts_feature_vec((x0,x1), (x0+1,x1))
148                 # the weighted potts function
149                 potts_function = WeightedPottsFunction(shape=[2,2],
150                                                         features=features,
151                                                         weight_ids=
152                                                         weight_ids,

```



```

144                                     weights=weights)
145     # add factors to both models
146     gm.add_factor(variables=variables, value_table=
147         potts_function)
148     loss_augmented_gm.add_factor(variables=variables,
149         value_table=potts_function)
150
151     # vertical edge
152     if x1 + 1 < shape[1]:
153         variables = [vi(x0,x1),vi(x0, x1+1)]
154         features = get_potts_feature_vec((x0,x1), (x0,x1+1))
155         # the weighted potts function
156         potts_function = WeightedPottsFunction(shape=[2,2],
157             features=features,
158             weight_ids=
159                 weight_ids,
160                 weights=weights)
161
162     # add factors to both models
163     gm.add_factor(variables=variables, value_table=
164         potts_function)
165     loss_augmented_gm.add_factor(variables=variables,
166         value_table=potts_function)
167
168     # gm, loss augmented and the loss
169     return gm, loss_augmented_gm, HammingLoss(gt_image.ravel())
170
171 # very simple helper class to combine things
172 class Dataset(object):
173     def __init__(self, models_train, models_test, weights):
174         self.models_train = models_train
175         self.models_test = models_test
176         self.weights = weights
177
178 # Subgradient SSVM
179 # =====
180 #
181 # Instead of a cutting plane approach, we use a subgradient decent to find
182 # the optimal weights
183 #
184 def subgradient_ssvm(dataset, mode='gp', n_iter=20, learning_rate=1.0, c
185     =0.5, lower_bounds=None, upper_bounds=None, convergence=0.001):
186
187     loss_train = []
188     loss_test = []
189
190     weights = dataset.weights
191     n = len(dataset.models_train)
192
193     if lower_bounds is None:
194         lower_bounds = numpy.ones(len(weights))*-1.0*float('inf')
195
196

```

```

189     if upper_bounds is None:
190         upper_bounds = numpy.ones(len(weights))*float('inf')
191
192     do_opt = True
193     for iteration in range(n_iter):
194
195         effective_learning_rate = learning_rate*float(learning_rate)/(1.0+
196             iteration)
197
198         # compute gradient
199         diff = numpy.zeros(weights.shape)
200         for gm, gm_loss_augmented, loss_function in dataset.models_train:
201
202             # update the weights to the current weight vector
203             gm.change_weights(weights)
204             gm_loss_augmented.change_weights(weights)
205
206             # the gt vector
207             y_true = loss_function.y_true
208
209             # optimize loss augmented / find most violated constraint
210             if mode == 'icm':
211                 icm = IteratedConditionalModes(model=gm_loss_augmented)
212                 y_hat = icm.optimize()
213             else:
214                 graphcut = GraphCut(model=gm_loss_augmented)
215                 y_hat = graphcut.optimize()
216
217             # compute joint feature vector
218             phi_y_hat = gm.phi(y_hat)
219             phi_y_true = gm.phi(y_true)
220
221             diff += phi_y_true - phi_y_hat
222
223
224
225         new_weights = weights - effective_learning_rate*(c/n)*diff
226
227         # project new weights
228         where_to_large = numpy.where(new_weights>upper_bounds)
229         new_weights[where_to_large] = upper_bounds[where_to_large]
230         where_to_small = numpy.where(new_weights<lower_bounds)
231         new_weights[where_to_small] = lower_bounds[where_to_small]
232
233
234         delta = numpy.abs(new_weights-weights).sum()
235         if(delta<convergence):
236             print("converged")
237             break
238     print('iter',iteration, 'delta',delta," ",numpy.round(new_weights
239         ,3))

```

```

239
240     weights = new_weights
241
242
243     # =====
244     # show performance during training
245     # =====
246
247     losses = []
248     for gm,_,loss_function in dataset.models_train:
249         gm.change_weights(weights)
250         graphcut = GraphCut(model=gm)
251         y_pred = graphcut.optimize()
252         losses.append(loss_function(y_pred))
253     loss_train.append(losses)
254     print('training set performance', losses)
255
256     losses = []
257     for gm,_,loss_function in dataset.models_test:
258         gm.change_weights(weights)
259         graphcut = GraphCut(model=gm)
260         y_pred = graphcut.optimize()
261         losses.append(loss_function(y_pred))
262     loss_test.append(losses)
263     print('test set performance', losses)
264
265     # in bonus mode: make a plot of performance
266     if mode=='bonus':
267         plottrain = numpy.array(loss_train).T
268         plottest = numpy.array(loss_test).T
269         for i in range(plottrain.shape[0]):
270             plt.plot(range(n_iter),plottrain[i], lw=0.5, label=f'train img {
                i}')
271             plt.plot(range(n_iter),plottest[i], lw=0.5, label=f'test img {
                i}')
272         plt.title('Performance during training')
273         plt.legend()
274         plt.show()
275
276
277     return weights
278
279     noises = [1.5, 2.0, 2.5, 3.0, 3.5]
280     regularizers = [0.1, 0.5, 0.9, 5., 10.]
281     shape = (20,20)
282     # noise = 2.0
283
284     loss_train = []
285     loss_test = []
286
287     # =====
288     # CHOOSE MODE

```

```

289 # gp: use graphcut
290 # icm: use icm
291 # bonus: show a plot of the loss at every step of learning phase
292 # =====
293 modes = [ 'gp', 'icm', 'bonus' ]
294 mode = modes[2]
295
296 for noise in noises:
297
298     l_noises = []
299
300     # The Dataset
301     # =====
302
303     x_train, y_train = make_toy_dataset(shape=shape, n_images=5, noise=
        noise)
304     x_test, y_test = make_toy_dataset(shape=shape, n_images=5, noise=
        noise)
305
306     # Build the weighted models:
307     # =====
308
309     unary_features_0 = get_unary_features(x_train[0])
310     potts_features_0 = get_potts_features(x_train[0])
311     n_unary_features = unary_features_0.shape[2]
312     n_potts_features = potts_features_0.shape[2]
313
314     n_weights = n_unary_features + n_potts_features
315
316     for c in regularizers:
317         weights = numpy.zeros(n_weights)
318
319         # build the graphical models
320         models_train = [build_model(x,y, weights) for x,y in zip(x_train,
            y_train)]
321         models_test = [build_model(x,y, weights) for x,y in zip(x_test,
            y_test)]
322
323
324         # combine things in a dataset
325         dset = Dataset(models_train, models_test, weights)
326
327
328         # we want the regularizer 'beta' to be positive
329         lower_bounds = numpy.ones(n_weights)*(-1.0*float('inf'))
330         lower_bounds[n_unary_features:n_unary_features+n_potts_features] =
            0
331
332         weights = subgradient_ssvm(dset, mode=mode, c=c, learning_rate=1.0,
            lower_bounds=lower_bounds, n_iter=100)
333
334

```

```

335     # Training Set Performance:
336     # =====
337
338     print("learned weights", weights)
339
340     for i, (gm, _, loss_function) in enumerate(models_train):
341         gm.change_weights(weights)
342
343         if mode == 'icm':
344             icm = IteratedConditionalModes(model=gm)
345             y_pred = icm.optimize()
346         else:
347             graphcut = GraphCut(model=gm)
348             y_pred = graphcut.optimize()
349         print('loss of train img ', i, ':', loss_function(y_pred))
350
351
352     # Test set performance:
353     # =====
354
355     # losses for this c
356     l_c = []
357
358     for i, (gm, _, loss_function) in enumerate(models_test):
359         gm.change_weights(weights)
360
361         if mode == 'icm':
362             icm = IteratedConditionalModes(model=gm)
363             y_pred = icm.optimize()
364         else:
365             graphcut = GraphCut(model=gm)
366             y_pred = graphcut.optimize()
367
368         print('loss of test img', i, '=', loss_function(y_pred))
369         l_c.append(loss_function(y_pred))
370
371     l_noises.append(l_c)
372
373     loss_test.append(l_noises)
374
375     print(numpy.array(loss_test))

```