GMM model for $P(h)$

1. Update Training Pages Layout Data $->$ Done.
2. Code general XML parser $->$ Done see 1.1.
3. Train GMM model using EM $->$ Done see 1.1.
4. Plot results over sample page $->$ Done see 1 and 2.

1 Results

Model is separated into two independent models: Upper corner model and Bottom corner Model. Each model is trained using the data from the 22 train images, both restricted to a diagonal covariance matrix in order keep the Mixtures axis-aligned as much as possible [1] (under axis independence assumption).

First, a 2 mixtures $\boldsymbol{u}$ and 3 mixtures $\boldsymbol{b}$ models are shown in Figure 1, notice that since only one page of the training set had a main paragraph that end in the middle of the page, the tendency of the model to move to that position is lower that the expected (issue will be fixed using more training images).

Now in order to see the tendency of each model, a 5 mixtures $\boldsymbol{u}$ and 8 mixtures $\boldsymbol{b}$ models are shown in Figure 2.

1.1 Some notes about the implementation

- Implemented in Python 2.7
- Dependencies: `argparse, numpy, scipy, glob, os, sklearn, matplotlib, [pickle | cPickle]`
- Execution time (22 images, $\boldsymbol{u} \approx 5$ mixtures, $\boldsymbol{b} \approx 8$ mixtures):
  - Getting Data Points ≈ 0.05908 seconds
  - Training GMM ≈ 0.08245 seconds
  - Total Time ≈ 1.73153 seconds (most of the time consumed by plotting the results, because each GMM is tested over all the image coordinates)

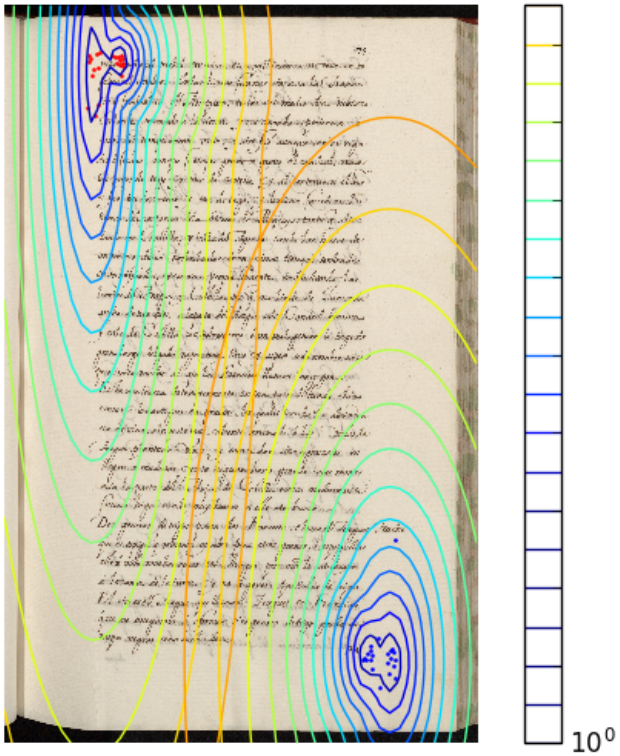

Figure 1: Learned GMM model; Red Points = $\boldsymbol{u}' \sim \mathcal{N}^2(\mu, \Sigma_{diag})$; Blue Points = $\boldsymbol{b}' \sim \mathcal{N}^3(\mu, \Sigma_{diag})$

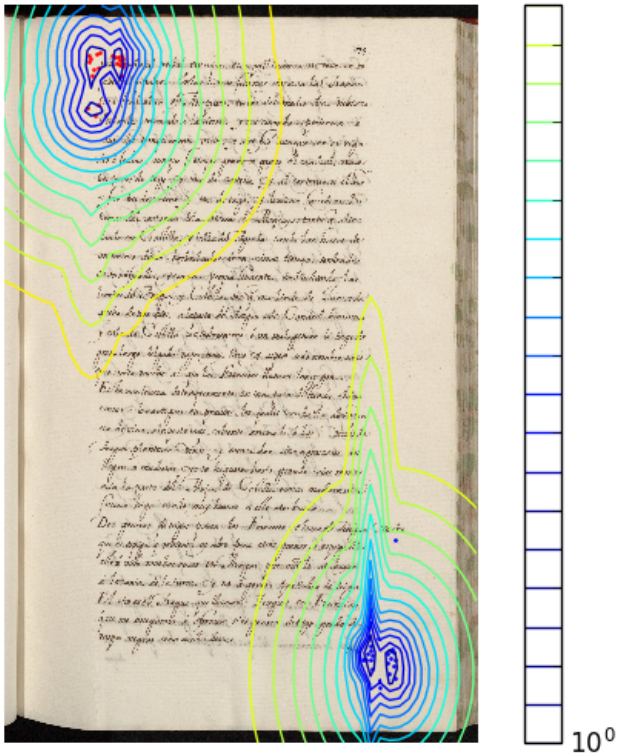

Figure 2: Learned GMM model; Red Points = $\boldsymbol{u}' \sim \mathcal{N}^5(\mu, \Sigma_{diag})$; Blue Points = $\boldsymbol{b}' \sim \mathcal{N}^8(\mu, \Sigma_{diag})$

References

[1] Duda, R. O., Hart, P. E., and Stork, D. G. *Pattern Classification (2Nd Edition)*. Wiley-Interscience, 2000.

```
from __future__ import division
import numpy as np #--- To handle math processing
import scipy.ndimage as ndi #--- To handle image processing
from scipy import misc
import os #--- To handle OS callbacks
import xml.etree.ElementTree as et #--- To handle XML data


class imgPage(object):
    """imgPage object compiles all data of some page"""
    def __init__(self, filePointer):
        super(imgPage, self).__init__()
        self.imgPointer = filePointer
        self.dir = os.path.dirname(filePointer)
        self.name = os.path.splitext(os.path.basename(filePointer)
    ↪ )[0]
        self.xmlPointer = self.dir + '/page/' + self.name + '.xml'

    def readImage(self, full = False):
        """

    ↪ #-------------------------------------------------------------------------------#
    ↪
        #--                                readImage
    ↪             --#

    ↪ #-------------------------------------------------------------------------------#
    ↪
        Description:
            Read input image and stores it on numpy array
        Inputs:
            self
        Outputs:
            self + image array
        Author:
            Quiros Diaz, Lorenzo
        Date:
            Jun/19/2016

    ↪ #-------------------------------------------------------------------------------#
    ↪
        """
        if (full):
            self.img = ndi.imread(self.imgPointer)
        else:
            self.img = ndi.imread(self.imgPointer, mode='I')
        self.imgShape = self.img.shape

    def parseXML(self):
```

```
42          """
43
    ↪ #--------------------------------------------------------------------------#
    ↪
44          #--                                parseXML
    ↪                --#
45
    ↪ #--------------------------------------------------------------------------#
    ↪
46          Description:
47              parse XML file related to img
48          Inputs:
49              self
50          Outputs:
51              self + XML data
52          Author:
53              Quiros Diaz, Lorenzo
54          Date:
55              Jun/19/2016
56
    ↪ #--------------------------------------------------------------------------#
    ↪
57          """
58          tree = et.parse(self.xmlPointer)
59          self.rootXML = tree.getroot()
60          self.baseXML = self.rootXML.tag.rsplit('}',1)[0] + '}'
61
62      def getMainParagraph(self):
63          #mainParag = self.rootXML[1][3][0].attrib.get('points').
    ↪ split()
64          mainParag = self.rootXML.findall('./' + self.baseXML + '
    ↪ Page' +
65              '/*[@type="paragraph"]')[0].findall('./' + self.baseXML
    ↪  +
66              'Coords')[0].attrib.get('points').split()
67          return np.array([i.split(',') for i in mainParag]).astype(
    ↪ np.int)
68
69      def getUpperPoints(self):
70          orig = self.getMainParagraph()
71          return np.array([orig[0][0], orig[0][1]])
72
73      def getBottomPoints(self):
74          orig = self.getMainParagraph()
75          return np.array([orig[2][0], orig[2][1]])
76
77      def getGroundTrueMask(self):
78          to_return = np.zeros(self.imgShape, dtype='uint8')
79          Points = self.getMainParagraph()
```

```
80          parPos = np.ix_(np.arange(Points[0][1],Points[2][1]),
81                          np.arange(Points[0][0], Points[2][0]))
82          to_return[parPos] = 255
83
84          return to_return
85
86
87  def test_module():
88      imgP = "/Users/lquirosd/Documents/MsC_MIARFID/MIARFID/TFM/
        ↪ MILA/DataCorpus/test/Mss_003357_0958_pag-825[857].jpg"
89
90      imgData = imgPage(imgP)
91      imgData.readImage()
92      imgData.parseXML()
93      mask = imgData.getGroundTrueMask()
94      misc.imsave('mask.jpg', mask)
95      #---- sudo ln /dev/null /dev/raw1394
96
97  if __name__ == '__main__':
98      test_module()
```

```
from __future__ import division
import sys, argparse #--- To handle console arguments
import numpy as np #--- To handle math processing
import scipy.ndimage as ndi #--- To handle image processing
from scipy import misc
import glob, os #--- To handle OS callbacks
import utils
from sklearn import mixture
import time
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
try:
   import cPickle as pickle
except:
   import pickle


def main():
   """

    ↪ #-------------------------------------------------------------------------------#
    ↪
   #--                                main
    ↪    --#

    ↪ #-------------------------------------------------------------------------------#
    ↪
   Description:
      main module
   Inputs:
      #--- To be updated
   Outputs:
      #--- To be updated
   Author:
      Quiros Diaz, Lorenzo
   Date:
      Jun/20/2016

    ↪ #-------------------------------------------------------------------------------#
    ↪
   """
   #--- processing arguments
   parser = argparse.ArgumentParser(description='K-NN classifier
    ↪ ')
   parser.add_argument('-trDir', required=True, action="store",
    ↪ help="Pointer to Training images folder")
   parser.add_argument('-o', '--out', required=True, default="."
    ↪ , action="store", help="Folder to save Out files")
   parser.add_argument('-nU', '--nUpper', type=int, default=2,
```

```
    ↪ action="store", help="Number of Mixtures for Upper Model [
    ↪ Default=2]")
40  parser.add_argument('-nB', '--nBottom', type=int, default=3,
    ↪ action="store", help="Number of Mixtures for Bottom Model
    ↪ [Default=3]")
41  parser.add_argument('-s', '--statistics', action="store_true"
    ↪ , help="Print some statistics about script execution")
42  parser.add_argument('--debug', action="store_true", help="Run
    ↪  script on Debugging mode")
43  args = parser.parse_args()
44  if (args.debug): print args
45  if (args.statistics): init = time.clock()
46  #--- Validate arguments
47  if (not os.path.isdir(args.trDir)):
48     print "Folder: %s does not exists\n" %args.trDir
49     parser.print_help()
50     sys.exit(2)
51  if (not os.path.isdir(args.out)):
52     print "Folder: %s does not exists\n" %args.out
53     parser.print_help()
54     sys.exit(2)
55
56  #--- Read images
57  allImgs = glob.glob(args.trDir + "/*.jpg")
58  nImgs = len(allImgs)
59  if  nImgs <= 0:
60     print "Folder: %s contains no images\n" %args.trDir
61     parser.print_help()
62     sys.exit(2)
63
64  if (args.statistics): GPinit = time.clock()
65  #--- keep all image data, just to check memory usage
66  #--- TODO: remove unnecessary data on each iteration
67  imgData = np.empty(nImgs, dtype=object)
68  #--- Array of Upper corners
69  U = np.zeros((nImgs, 2), dtype=np.int)
70  #--- Array of Bottom corners
71  B = np.zeros((nImgs, 2), dtype=np.int)
72  #--- get U & B corners from all TR dataSet
73  for i, file in enumerate(allImgs):
74     imgData[i] = utils.imgPage(file)
75     #imgData[i].readImage()
76     imgData[i].parseXML()
77     U[i] = imgData[i].getUpperPoints()
78     B[i] = imgData[i].getBottomPoints()
79
80  if (args.statistics): print 'Getting Data Points: {0:.5f}
    ↪ seconds'.format(time.clock() - GPinit)
81  if (args.statistics): TGinit = time.clock()
```

```
82    #--- Train GMM Models
83    #--- Upper GMM
84    uGMM = mixture.GMM(n_components = args.nUpper)
85    uGMM.fit(U)
86    #--- Bottom GMM
87    bGMM = mixture.GMM(n_components = args.nBottom,
      ↪ covariance_type='diag')
88    bGMM.fit(B)
89
90    GMM_models = {'Upper': uGMM, 'Bottom': bGMM}
91    #--- Save Models to file
92    #--- Out File Name
93    outFile = args.out + 'GMM_tr' + str(nImgs) + '_u' + str(args.
      ↪ nUpper) + '_b' + str(args.nBottom)
94    fh = open(outFile + '.model', 'w')
95    pickle.dump(GMM_models, fh)
96    fh.close()
97    if (args.statistics): print 'Training GMM: {0:.5f} seconds'.
      ↪ format(time.clock() - TGinit)
98
99    #--- Plot Mixtures and Data
100   m=9
101   imgData[m].readImage(full=True)
102   fig, axs = plt.subplots(1,1)
103   axs.scatter(U[:, 0], U[:, 1], .8, color='red')
104   axs.scatter(B[:, 0], B[:, 1], .8, color='blue')
105
106   x = np.linspace(0, imgData[m].imgShape[1])
107   y = np.linspace(0, imgData[m].imgShape[0])
108   X, Y = np.meshgrid(x, y)
109   XX = np.array([X.ravel(), Y.ravel()]).T
110
111   uZ = -uGMM.score_samples(XX)[0]
112   uZ = uZ.reshape(X.shape)
113   bZ = -bGMM.score_samples(XX)[0]
114   bZ = bZ.reshape(X.shape)
115
116   CSu = axs.contour(X, Y, uZ, norm=LogNorm(vmin=np.min(uZ),
      ↪ vmax=np.max(uZ)),
117                     levels=np.logspace(0, 3, 20))
118   CSb = axs.contour(X, Y, bZ, norm=LogNorm(vmin=np.min(bZ),
      ↪ vmax=np.max(bZ)),
119                     levels=np.logspace(0, 3, 20))
120   #axs.clabel(CS, inline=1, fontsize=10)
121   CB = plt.colorbar(CSu, ax=axs, extend='both')
122
123   axs.imshow(imgData[m].img, cmap='gray')
124   plt.axis('off')
125   fig.savefig(outFile + '.png', bbox_inches='tight')
```

```
126     if (args.statistics): print 'Total Time: {0:.5f} seconds'.
        ↪ format(time.clock() - init)
127     plt.show()
128 
129 if __name__ == '__main__':
130     main()
```