# 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 ends in the middle of the page, the tendency of the model to move to that position is weaker 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 $\approx 0.05908$ seconds
  - Training GMM $\approx 0.08245$ seconds
  - Total Time $\approx 1.73153$ seconds (most of the time consumed by plotting the results, because each GMM is tested over all the image coordinates)

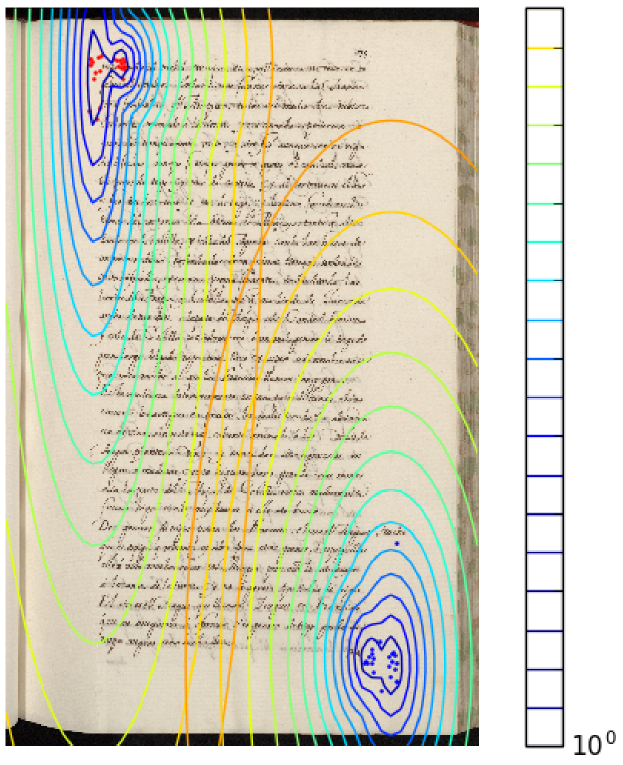

Figure 1: Learned GMM model; Red Points = $\boldsymbol{u'} \sim \mathcal{N}^2(\mu, \Sigma_{diag})$; Blue Points = $\boldsymbol{b'} \sim \mathcal{N}^3(\mu, \Sigma_{diag})$

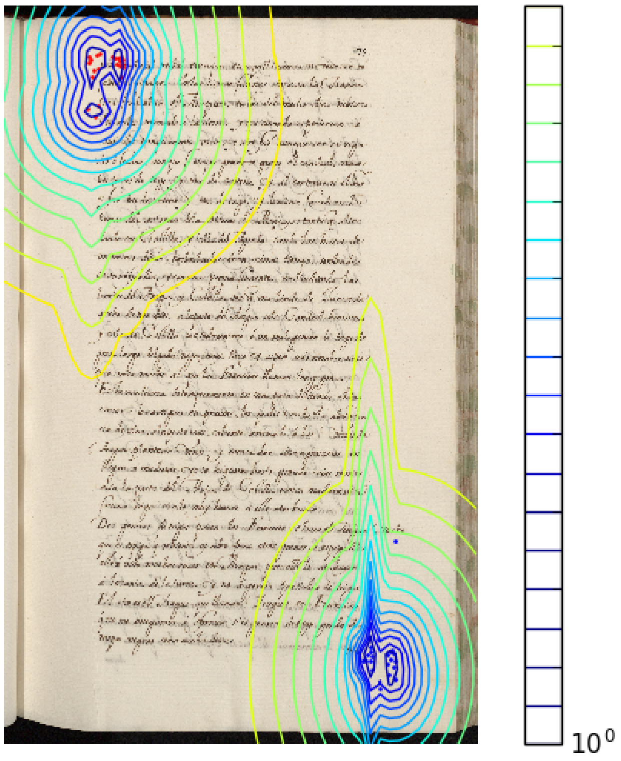

Figure 2: Learned GMM model; Red Points = $\boldsymbol{u'} \sim \mathcal{N}^5(\mu, \Sigma_{diag})$; Blue Points = $\boldsymbol{b'} \sim \mathcal{N}^8(\mu, \Sigma_{diag})$

## References

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

```python
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):
```

```
    """
↪ #------------------------------------------------------------------------#
↪
    #---                     parseXML
↪           ---#
↪ #------------------------------------------------------------------------#
↪
    Description:
        parse XML file related to img
    Inputs:
        self
    Outputs:
        self + XML data
    Author:
        Quiros Diaz, Lorenzo
    Date:
        Jun/19/2016
↪ #---------------------------------------------------------------------------#
↪
    """
    tree = et.parse(self.xmlPointer)
    self.rootXML = tree.getroot()
    self.baseXML = self.rootXML.tag.rsplit('}',1)[0] + '}'

def getMainParagraph(self):
    #mainParag = self.rootXML[1][3][0].attrib.get('points').
↪ split()
    mainParag = self.rootXML.findall('./' + self.baseXML + '
↪ Page' +
        '/*[@type="paragraph"]')[0].findall('./' + self.baseXML
↪  +
        'Coords')[0].attrib.get('points').split()
    return np.array([i.split(',') for i in mainParag]).astype(
↪ np.int)

def getUpperPoints(self):
    orig = self.getMainParagraph()
    return np.array([orig[0][0], orig[0][1]])

def getBottomPoints(self):
    orig = self.getMainParagraph()
    return np.array([orig[2][0], orig[2][1]])

def getGroundTrueMask(self):
    to_return = np.zeros(self.imgShape, dtype='uint8')
    Points = self.getMainParagraph()
```

```
        parPos = np.ix_(np.arange(Points[0][1],Points[2][1]),
                np.arange(Points[0][0], Points[2][0]))
        to_return[parPos] = 255

        return to_return


def test_module():
    imgP = "/Users/lquirosd/Documents/MsC_MIARFID/MIARFID/TFM/
    ↪ MILA/DataCorpus/test/Mss_003357_0958_pag-825[857].jpg"

    imgData = imgPage(imgP)
    imgData.readImage()
    imgData.parseXML()
    mask = imgData.getGroundTrueMask()
    misc.imsave('mask.jpg', mask)
    #---- sudo ln /dev/null /dev/raw1394

if __name__ == '__main__':
    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]")
   parser.add_argument('-nB', '--nBottom', type=int, default=3,
    ↪ action="store", help="Number of Mixtures for Bottom Model
    ↪ [Default=3]")
   parser.add_argument('-s', '--statistics', action="store_true"
    ↪ , help="Print some statistics about script execution")
   parser.add_argument('--debug', action="store_true", help="Run
    ↪  script on Debugging mode")
   args = parser.parse_args()
   if (args.debug): print args
   if (args.statistics): init = time.clock()
   #--- Validate arguments
   if (not os.path.isdir(args.trDir)):
      print "Folder: %s does not exists\n" %args.trDir
      parser.print_help()
      sys.exit(2)
   if (not os.path.isdir(args.out)):
      print "Folder: %s does not exists\n" %args.out
      parser.print_help()
      sys.exit(2)

   #--- Read images
   allImgs = glob.glob(args.trDir + "/*.jpg")
   nImgs = len(allImgs)
   if  nImgs <= 0:
      print "Folder: %s contains no images\n" %args.trDir
      parser.print_help()
      sys.exit(2)

   if (args.statistics): GPinit = time.clock()
   #--- keep all image data, just to check memory usage
   #--- TODO: remove unnecessary data on each iteration
   imgData = np.empty(nImgs, dtype=object)
   #--- Array of Upper corners
   U = np.zeros((nImgs, 2), dtype=np.int)
   #--- Array of Bottom corners
   B = np.zeros((nImgs, 2), dtype=np.int)
   #--- get U & B corners from all TR dataSet
   for i, file in enumerate(allImgs):
      imgData[i] = utils.imgPage(file)
      #imgData[i].readImage()
      imgData[i].parseXML()
      U[i] = imgData[i].getUpperPoints()
      B[i] = imgData[i].getBottomPoints()

   if (args.statistics): print 'Getting Data Points: {0:.5f}
    ↪ seconds'.format(time.clock() - GPinit)
   if (args.statistics): TGinit = time.clock()
```

```python
    #--- Train GMM Models
    #--- Upper GMM
    uGMM = mixture.GMM(n_components = args.nUpper)
    uGMM.fit(U)
    #--- Bottom GMM
    bGMM = mixture.GMM(n_components = args.nBottom, covariance_type='diag')
    bGMM.fit(B)

    GMM_models = {'Upper': uGMM, 'Bottom': bGMM}
    #--- Save Models to file
    #--- Out File Name
    outFile = args.out + 'GMM_tr' + str(nImgs) + '_u' + str(args.nUpper) + '_b' + str(args.nBottom)
    fh = open(outFile + '.model', 'w')
    pickle.dump(GMM_models, fh)
    fh.close()
    if (args.statistics): print 'Training GMM: {0:.5f} seconds'.format(time.clock() - TGinit)

    #--- Plot Mixtures and Data
    m=9
    imgData[m].readImage(full=True)
    fig, axs = plt.subplots(1,1)
    axs.scatter(U[:, 0], U[:, 1], .8, color='red')
    axs.scatter(B[:, 0], B[:, 1], .8, color='blue')

    x = np.linspace(0, imgData[m].imgShape[1])
    y = np.linspace(0, imgData[m].imgShape[0])
    X, Y = np.meshgrid(x, y)
    XX = np.array([X.ravel(), Y.ravel()]).T

    uZ = -uGMM.score_samples(XX)[0]
    uZ = uZ.reshape(X.shape)
    bZ = -bGMM.score_samples(XX)[0]
    bZ = bZ.reshape(X.shape)

    CSu = axs.contour(X, Y, uZ, norm=LogNorm(vmin=np.min(uZ), vmax=np.max(uZ)),
                levels=np.logspace(0, 3, 20))
    CSb = axs.contour(X, Y, bZ, norm=LogNorm(vmin=np.min(bZ), vmax=np.max(bZ)),
                levels=np.logspace(0, 3, 20))
    #axs.clabel(CS, inline=1, fontsize=10)
    CB = plt.colorbar(CSu, ax=axs, extend='both')

    axs.imshow(imgData[m].img, cmap='gray')
    plt.axis('off')
    fig.savefig(outFile + '.png', bbox_inches='tight')
    if (args.statistics): print 'Total Time: {0:.5f} seconds'.
    ↪ format(time.clock() - init)
    plt.show()

if __name__ == '__main__':
    main()
```