PS4-generative-models

April 26, 2017

In [1]:
```
from __future__ import division
from ps4_utils import load_data,load_experiment
from ps4_utils import AbstractGenerativeModel
from ps4_utils import save_submission
from scipy.misc import logsumexp
import numpy as np
data_fn = "datasets-hw4.h5"
MAX_OUTER_ITER = 15
```

In [2]:
```
class MixtureModel(AbstractGenerativeModel):
    def __init__(self, CLASSES, NUM_FEATURES, NUM_MIXTURE_COMPONENTS, MAX_I
        AbstractGenerativeModel.__init__(self, CLASSES, NUM_FEATURES)
        self.num_mixture_components = NUM_MIXTURE_COMPONENTS # list of num_
        self.max_iter = MAX_ITER # max iterations of EM
        self.epsilon = EPS # help with stability, to be used according to h
        self.params = { # lists of length CLASSES
            'pi': [np.repeat(1/k,k) for k in self.num_mixture_components],
            'theta': [np.zeros((self.num_features,k)) for k in self.num_mix
        }
    def pack_params(self, X, class_idx):
        pi,theta = self.fit(X[class_idx],class_idx) # fit parameters
        self.params['pi'][class_idx] = pi # update member variable pi
        self.params['theta'][class_idx] = theta #update member variable the

    #make classification based on which mixture model gives higher probabil
    def classify(self, X):
        P = list()
        pi = self.params['pi']
        theta = self.params['theta']
        for c in range(self.num_classes):
            _,Pc = self.findP(X, pi[c], theta[c])
            P.append(Pc)
        return np.vstack(P).T.argmax(-1) # np.array of class predictions fo

    # --- E-step
    def updateLatentPosterior(self, X, pi, theta, num_mixture_components):
        # YOUR CODE HERE
```

```
        # --- gamma: responsibilities (probabilities), np.array (matrix)
        # ---        shape: number of data points in X (where X consists of
        # note: can use output of findP here (with care taken to return gam
        t, sumlogt = self.findP(X, pi, theta)
        gamma = np.zeros((X.shape[0], num_mixture_components))
        for i in range(X.shape[0]):
            for c in range(num_mixture_components):
                log_gamma = t[i,c]-(sumlogt[i])
                gamma[i,c] = np.exp(log_gamma)

        return gamma
    # --- M-step (1)
    @staticmethod
    def updatePi(gamma): #update the pi component using the posteriors (gam
        # YOUR CODE HERE
        # --- pi_c: class specific pi, np.array (vector)
        # ---        shape: NUM_MIXTURE_COMPONENTS[c]
        pi_c = np.sum(gamma, axis = 0) / gamma.shape[0]
        return pi_c
    # -- M-step (2)
    @staticmethod
    def updateTheta(X, gamma): #update theta component using posteriors (ga
        # YOUR CODE HERE
        # --- theta_c: class specific theta, np.array matrix
        # ---        shape: NUM_FEATURES by NUM_MIXTURE_COMPONENTS[c]
        theta_c = np.dot(X.T, gamma) / np.sum(gamma, axis = 0)
        eps = 10**(-7)
        for i in range(theta_c.shape[0]):
            for j in range(theta_c.shape[1]):
                if theta_c[i][j] < eps:
                    theta_c[i][j] = eps
                elif theta_c[i][j] > 1-eps:
                    theta_c[i][j] = eps

        return theta_c

    @staticmethod
    def findP(X, pi, theta):
        # YOUR CODE HERE
        # NOTE: you can also use t as a probability, just change "logsumexp
        # --- t: logprobabilities of x given each component of mixture
        # ---        shape: number of data points in X (where X consists of
        # --- logsumexp(t,axis=1): (for convenience) once exponentiated, gi
        # ---        shape: number of data points in X (where X consists of
        eps = 10**(-7)
        t = np.dot(X, np.log(theta + eps)) + np.dot((1 - X), np.log(1 - the
        return t,logsumexp(t,axis=1)
```

```
        # --- execute EM procedure
        def fit(self, X, class_idx):
            max_iter = self.max_iter
            eps = self.epsilon
            N = X.shape[0]
            pi = self.params['pi'][class_idx]
            theta = self.params['theta'][class_idx]
            num_mixture_components = self.num_mixture_components[class_idx]
            # INITIALIZE theta, note theta is currently set to zeros but needs
            for i in range(num_mixture_components):
                theta[:,i] = np.sum(X[range(i,N,num_mixture_components),:], axi


            for i in range(theta.shape[0]):
                for j in range(theta.shape[1]):
                    if theta[i][j] < self.epsilon:
                        theta[i][j] = self.epsilon
                    elif theta[i][j] > 1 - self.epsilon:
                        theta[i][j] = 1 - self.epsilon

            for i in range(max_iter):
                gamma = self.updateLatentPosterior(X, pi, theta, num_mixture_co
                pi = self.updatePi(gamma)
                theta = self.updateTheta(X, gamma)


            return pi,theta #pi and theta, given class_idx

In [3]: class NaiveBayesModel(AbstractGenerativeModel):
        def __init__(self, CLASSES, NUM_FEATURES, EPS=10**(-12)):
            AbstractGenerativeModel.__init__(self, CLASSES, NUM_FEATURES)
            self.epsilon = EPS # help with stability
            self.params = {
                'p': [np.zeros((NUM_FEATURES))] * self.num_classes # estimated
            }
        def pack_params(self, X, class_idx):
            p = self.fit(X[class_idx])
            self.params['p'][class_idx] = p
        def classify(self, X): # naive bayes classifier
            # YOUR CODE HERE
            # --- predictions: predictions for data points in X (where X consis
            # ---         shape: number of data points
            pred = []
            for i in range(X.shape[0]):
                p_x_i = [0]*len(self.params['p'])
                x_i = X[i,:]
                for c, p_c in enumerate(self.params['p']):
                    for j in range(X.shape[1]):
```

```
                        p_x_i[c] += np.log((x_i[j] * p_c[j] + (1 - x_i[j]) * (1
                    pred.append(np.argmax(p_x_i))

                pred = np.array(pred)
                return pred
            def fit(self, X):
                # YOUR CODE HERE
                # --- estimated_p: estimated p's of features for input X (where X c
                # ---         shape: NUM_FEATURES
                estimated_p = np.sum(X, axis = 0) / X.shape[0]
                return np.array(estimated_p)
```

In [4]:
```
experiment_name = "sentiment_analysis"
# --- SENTIMENT ANALYSIS setup
Xtrain,Xval,num_classes,num_features = load_experiment(data_fn, experiment_

# -- build naive bayes model for sentiment analysis
print("SENTIMENT ANALYSIS -- NAIVE BAYES MODEL:")
nbm = NaiveBayesModel(num_classes, num_features)
nbm.train(Xtrain)
print("ACCURACY ON VALIDATION: " + str(nbm.val(Xval)))

# -- build mixture model for sentiment analysis
print("SENTIMENT ANALYSIS -- MIXTURE MODEL:")
for i in range(MAX_OUTER_ITER):
    num_mixture_components =  np.random.randint(2,15,num_classes)
    print("COMPONENTS: " + " ".join(str(i) for i in num_mixture_components)
    mm = MixtureModel(num_classes, num_features, num_mixture_components)
    mm.train(Xtrain)
    print("ACCURACY ON VALIDATION: " + str(mm.val(Xval)))

# submit to kaggle
Xkaggle = load_data(data_fn, experiment_name, "kaggle")
save_submission("mm-{}-submission.csv".format(experiment_name), mm.classify
```

```
SENTIMENT ANALYSIS -- NAIVE BAYES MODEL:
ACCURACY ON VALIDATION: 0.74
SENTIMENT ANALYSIS -- MIXTURE MODEL:
COMPONENTS: 3 2
ACCURACY ON VALIDATION: 0.718
COMPONENTS: 12 14
ACCURACY ON VALIDATION: 0.726
COMPONENTS: 2 10
ACCURACY ON VALIDATION: 0.676
COMPONENTS: 13 7
ACCURACY ON VALIDATION: 0.712
COMPONENTS: 5 12
ACCURACY ON VALIDATION: 0.712
```

```
COMPONENTS: 12 8
ACCURACY ON VALIDATION: 0.714
COMPONENTS: 14 14
ACCURACY ON VALIDATION: 0.732
COMPONENTS: 10 6
ACCURACY ON VALIDATION: 0.728
COMPONENTS: 2 2
ACCURACY ON VALIDATION: 0.712
COMPONENTS: 8 9
ACCURACY ON VALIDATION: 0.706
COMPONENTS: 7 13
ACCURACY ON VALIDATION: 0.714
COMPONENTS: 14 2
ACCURACY ON VALIDATION: 0.696
COMPONENTS: 10 12
ACCURACY ON VALIDATION: 0.738
COMPONENTS: 4 11
ACCURACY ON VALIDATION: 0.718
COMPONENTS: 3 10
ACCURACY ON VALIDATION: 0.696
Saved: mm-sentiment_analysis-submission.csv
```

```
In [5]: experiment_name = "mnist"
        # --- MNIST DIGIT CLASSIFICATION setup
        Xtrain,Xval,num_classes,num_features = load_experiment(data_fn, experiment_

        # -- build naive bayes model for mnist digit classification
        print("MNIST DIGIT CLASSIFICATION -- NAIVE BAYES MODEL:")
        nbm = NaiveBayesModel(num_classes, num_features)
        nbm.train(Xtrain)
        print("ACCURACY ON VALIDATION: " + str(nbm.val(Xval)))

        # -- build mixture model for mnist digit classification
        print("MNIST DIGIT CLASSIFICATION -- MIXTURE MODEL:")
        for i in range(MAX_OUTER_ITER):
            num_mixture_components =  np.random.randint(2,15,num_classes)
            print("COMPONENTS: " + " ".join(str(i) for i in num_mixture_components)
            mm = MixtureModel(num_classes, num_features, num_mixture_components)
            mm.train(Xtrain)
            print("ACCURACY ON VALIDATION: " + str(mm.val(Xval)))

        # submit to kaggle
        Xkaggle = load_data(data_fn, experiment_name, "kaggle")
        save_submission("mm-{}-submission.csv".format(experiment_name), mm.classify

MNIST DIGIT CLASSIFICATION -- NAIVE BAYES MODEL:
ACCURACY ON VALIDATION: 0.7355
```

```
MNIST DIGIT CLASSIFICATION -- MIXTURE MODEL:
COMPONENTS: 9 8 4 10 14 2 7 9 9 9
ACCURACY ON VALIDATION: 0.7715
COMPONENTS: 11 3 4 5 4 9 10 5 8 6
ACCURACY ON VALIDATION: 0.776
COMPONENTS: 13 3 4 9 2 8 4 7 5 4
ACCURACY ON VALIDATION: 0.789
COMPONENTS: 7 6 5 2 12 8 7 5 5 13
ACCURACY ON VALIDATION: 0.775
COMPONENTS: 5 13 12 3 8 9 6 6 10 10
ACCURACY ON VALIDATION: 0.7775
COMPONENTS: 8 13 14 3 2 4 8 8 14 10
ACCURACY ON VALIDATION: 0.7685
COMPONENTS: 3 6 13 7 3 10 9 10 8 10
ACCURACY ON VALIDATION: 0.7915
COMPONENTS: 2 7 9 4 4 4 3 9 9 8
ACCURACY ON VALIDATION: 0.773
COMPONENTS: 14 13 2 3 7 7 2 13 10 7
ACCURACY ON VALIDATION: 0.772
COMPONENTS: 3 13 13 9 14 5 3 11 2 10
ACCURACY ON VALIDATION: 0.771
COMPONENTS: 7 10 7 9 5 4 12 12 2 3
ACCURACY ON VALIDATION: 0.7675
COMPONENTS: 3 14 12 11 14 12 13 13 4 14
ACCURACY ON VALIDATION: 0.766
COMPONENTS: 13 11 3 3 13 10 6 12 14 6
ACCURACY ON VALIDATION: 0.767
COMPONENTS: 10 7 13 8 13 9 7 7 4 8
ACCURACY ON VALIDATION: 0.779
COMPONENTS: 10 14 11 6 5 3 5 8 2 6
ACCURACY ON VALIDATION: 0.7865
Saved: mm-mnist-submission.csv
```

In [ ]:

In [ ]: