=====DATSCIW261 ASSIGNMENT #6=====

MIDS UC Berkeley, Machine Learning at Scale DATSCIW261 ASSIGNMENT #6

Hetal Chandaria, Patrick Ng, Marjorie Sayer

W261 - 2 , ASSIGNMENT #6

Submission Date: Feb 27, 2016

Group: 4

In [ ]:
```
%matplotlib inline
```

HW6.0.

In mathematics, computer science, economics, or management science what is mathematical optimization? Give an example of a optimization problem that you have worked with directly or that your organization has worked on. Please describe the objective function and the decision variables. Was the project successful (deployed in the real world)? Describe.

Answer

Mathematical optimization is the selection of a best element (with regard to some criteria) from some set of available alternatives. In the simplest case, an optimization problem consists of maximizing or minimizing a real function by systematically choosing input values from within an allowed set and computing the value of the function.

HW6.1

Optimization theory: For unconstrained univariate optimization what are the first order Necessary Conditions for Optimality (FOC). What are the second order optimality conditions (SOC)? Give a mathematical defintion. Also in python, plot the univartiate function X^3 -12x^2-6 defined over the real domain -6 to +6.

Also plot its corresponding first and second derivative functions. Eyeballing these graphs, identify candidate optimal points and then classify them as local minimums or maximums. Highlight and label these points in your graphs. Justify your responses using the FOC and SOC.

For unconstrained multi-variate optimization what are the first order Necessary Conditions for Optimality (FOC). What are the second order optimality conditions (SOC)? Give a mathematical defintion. What is the Hessian matrix in this context?

Answer

Unconstrained univariate optimization

1. FOC : The first-order derivatives must equal zero when evaluated at the same point, called a critical point
2. Take the first derivative of a function and find the function for the slope.
3. Set dy/dx equal to zero, and solve for x to get the critical point or points.
4. Take the second derivative of the original function.
5. SOC: The second-order direct derivatives must have the same sign when evaluated at the critical point(s).

   Substitute the x from step 3 into the second derivative and solve, paying particular attention to the sign of the second derivative.
6. Use the following characteristics to determine whether the function evaluated at the critical point or points is a relative maximum or minimum:

Relative Maximum	**Relative Minimum**
f'(x=a) =0	f'(x=a) = 0
f"(x=a) <0	f"(x=a) > 0

Refrence : http://www.columbia.edu/itc/sipa/math/calc_econ_interp_u.html (http://www.columbia.edu/itc/sipa/math/calc_econ_interp_u.html)

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
from sympy import *


x,f = symbols('x,f')
expr = x**3-12*(x**2)-6
# display(expr)


plt.figure(figsize=(18,5))

plt.subplot(131)
xvals = np.arange(-6, 6, 0.01)
yvals = (xvals**3)-(12*(xvals**2))-6
plt.plot(xvals, yvals)
plt.annotate('max', xy=(0, 0), xytext=(0, -100),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
plt.axvline(x=0,color='green',linestyle='dashed')
plt.title('f(x):Optimiation function')



expr = 3*x**2 - 24*x
# display(expr)
plt.subplot(132)
markerson =[0]
yvals = (3*xvals**2)-(24*xvals)
plt.plot(xvals, yvals)
plt.plot([0], [0], 'ro')
plt.axhline(y=0,color='green',linestyle='dashed')
plt.title('FOC')


expr = 6*x - 24
# display(expr)
plt.subplot(133)
yvals = (6*xvals)-(24)
plt.plot(xvals, yvals)
plt.axhline(y=-24,color='green',linestyle='dashed')
plt.axhline(y=24,color='green',linestyle='dashed')
plt.axhline(y=0,color='red',linestyle='dashed')
plt.axvline(x=0,color='green',linestyle='dashed')
plt.plot([0], [-24], 'ro')
plt.title('SOC')

#used http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.axvline
```

$f(x) = x^3 - 12x^2 - 6$

$f'(x) = \frac{d}{dx}(x^3 - 12x^2 - 6) = 3x^2 - 24x$

Solving f'(x) = 0

$3x^2 - 24x = 0$

$3x(x - 8) = 0$

$3x = 0$ or $x - 8 = 0$

$x = 0$ or $x = 8$

$f\text{ '' }(x) = \frac{d^2}{dx^2}(x^3 - 12x^2 - 6) = \frac{d}{dx}(3x^2 - 24x) = 6x - 24$

- Looking at the graph of f(x): Optimization function we see that the function is maximum when x =0
- Based on the FOC graph we can see the graph hits x at 0 when x =0
- Based on the SOC graph we can see that when x = 0, y = -24 which proves that the point x=0 is the local maxima of the function f(x)

Unconstrained multi-variate optimization

The conditions for relative maxima and minima for multivariate functions are very similar to those for univariate functions, with one additional requirement.

FOC

All first-order partial derivatives must equal zero when evaluated at the same point, called a critical point. If we are considering a function z with two independent variables x and y, then the three-dimensional shape taken by the function z reaches a high or low point when evaluated at specific values of x and y; these values are determined by setting the first derivatives equal to zero, and then solving the resulting system of equations for the two variables.

SOC

The second-order direct partial derivatives must both be the same sign when evaluated at the critical point(s). For a maximum, they must both be negative and for a minimum, both positive. This condition serves the same purpose as the second-order derivative condition in univariate optimization. It guarantees that the point where the slope is zero is indeed a high point, in the direction of the x variable and also in the direction of the y variable.

Relative Maximum	Relative Minimum
fx,fy=0	fx,fy=0
fxx,fyy <0	fxx,fyy >0

Hessian Matrix:

The hessian matrix is a symmetric matrix, that is $\frac{\partial^2 f(x)}{\partial x_i \partial x_j} = \frac{\partial^2 f(x)}{\partial x_j \partial x_i}$

Hessian matrix is used to discover the nature of a stationary point for a function of several variables.

Our task is equivalent to working out whether the Hessian matrix is positive definite, negative definite, or indefinite.

For an n x n symmetric matrix, a kth order leading principal minor is the determinant of the matrix obtained by deleting the last (n-k) rows and columns

(a) If and only if all leading principal minors of the matrix are positive, then the matrix is positive definite. For the Hessian, this implies the stationary point is a minimum.

(b) If and only if the kth order leading principal minor of the matrix has sign (-1)k, then the matrix is negative definite. For the Hessian, this implies the stationary point is a maximum.

(c) If none of the leading principal minors is zero, and neither (a) nor (b) holds, then the matrix is indefinite. For the Hessian, this implies the stationary point is a saddle point.

Otherwise the test is inconclusive. This implies that, at a local minimum (resp. a local maximum), the Hessian is positive-semi-definite (resp. negative semi-definite).

Example :

Let $f(x_1, x_2) = x_1^2 + 2x_1x_2 + 3x_2^2 + 4x_1 + 5x_2 + 6$

This function is differentiable and thus taking parital derivative we get below. Thus extrema can occur at points x* such that $\nabla f(x) = 0$

$\nabla f(x) = \begin{pmatrix} 2x_1+2x_2+4 \\ 2x_1+6x_2+5 \end{pmatrix}$

Solving the above we get $x_1 = \frac{-7}{4}$ and $x_2 = \frac{-1}{4}$

$H(x) = \begin{pmatrix} 2 & 2 \\ 2 & 6 \end{pmatrix}$

$H(-7/4, -1/4) = \begin{pmatrix} 2 & 2 \\ 2 & 6 \end{pmatrix}$ Its first principal minor has $det(H_1) = 2 > 0$ and its second principal minor has $det(H_2) = (2 * 6) - (2 * 2) = 12 - 4 = 8$

In [ ]:

HW6.2

Taking x=1 as the first approximation(xt1) of a root of X^3 + 2x -4 = 0, use the Newton-Raphson method to calculate the second approximation (denoted as xt2) of this root. (Hint the solution is xt2=1.2)

Answer

The Newton Raphson Method finds the tangent to the function f(x) at $x = x_0$ and extrapolates it to intersect the x-axis to get x_1

Newton Raphson iteration formula is given as $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$

$f(x) = x^3 + 2x - 4 = 0$

$\frac{\mathrm{d}(x^3+2x-4)}{\mathrm{d}x} = 3x^2 + 2$

$xt2 = xt1 - \frac{f(x=1)}{f'(x=1)}$

In [ ]:
```
from __future__ import division
def nr(x):
    fx = x**3+2*x-4
    der_fx = 3*x**2+2
    return (x- (fx/der_fx))
nr(1)
```

HW6.3 Convex optimization

What makes an optimization problem convex? What are the first order Necessary Conditions for Optimality in convex optimization. What are the second order optimality conditions for convex optimization? Are both necessary to determine the maximum or minimum of candidate optimal solutions?

Fill in the BLANKS here: Convex minimization, a subfield of optimization, studies the problem of minimizing BLANK functions over BLANK sets. The BLANK property can make optimization in some sense "easier" than the general case - for example, any local minimum must be a global minimum.

Answer

An optimization problem is convex if

1. Its objective is a convex function
2. inequality constraints f_j are convex
3. the equality constraints h_j are affine

A convex function is one where the line segment connecting two points □(x□, f(□x))□□ and (□y,□f(□y)) □lies above the function□ Mathematically□ a function f is convex if□ for all (x□, y) and all □$0 < \alpha < 1$□ □□

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y)$$

First Order Optimality condition

A function f(x), which is differentiable, is convex if and only if its domain is a convex set and if the following inequality condition is satisfied: $f\,(y) \;\geq\; f\,(x) + (\nabla_x f\,(x))^T\,(y - x)\,; \quad \forall x, y \in Domain(f)$

The first order condition for convexity says that f is convex if and only if the tangent line is a global underestimator of the function f. In other words, if we take our function and draw a tangent line at any point, then every point on this line will lie below the corresponding point on f.

Second Order Optimality condition

There is an easy way to check for convexity when f is twice di□fferentiable□,the function f is convex on domain [a,b] if (and only if) f"(x) >= 0 for all x in the domain.

If f(x) is convex, then any local minimum is also global minimum.

Fill in the blanks

Convex minimization, a subfield of optimization, studies the problem of minimizing **convex** functions over **convex** sets. The **convexity** property can make optimization in some sense "easier" than the general case - for example, any local minimum must be a global minimum.

HW 6.4

The learning objective function for weighted ordinary least squares (WOLS) (aka weight linear regression) is defined as follows:

0.5 *sumOverTrainingExample i (weight_i* (W * X_i - y_i)^2)

Where training set consists of input variables X (in vector form) and a target variable y, and W is the vector of coefficients for the linear regression model.

Derive the gradient for this weighted OLS by hand; showing each step and also explaining each step.

Answer

1. X = Training set of Input vairables in vector form
2. y = Target variable
3. W = coefficients of the linear regression model
4. w = weight vector
5. n = number of features
6. m = total number of records

Now we will define the cost or learning objective function for Weighted ordinary least squares as

$J(W) = \frac{1}{2} \sum_i w_i (WX_i - y_i)^2$

Derivation of Gradient descent:

1. Start with an initial value of W and repeatedly perform the update as below until we have a value of W such that J(W) is minimized

$W_j = W_j - \alpha \frac{\partial J(W)}{\partial W}$

This update is simultaneously performed for all values of j = 0,...,n

1. To implement the above algorithm we need to derive the partial derivative .

$\frac{\partial J(W)}{\partial W} = \frac{\partial}{\partial W} \frac{1}{2} \sum_i w_i (WX_i - y_i)^2$

$= 2\frac{1}{2} w_i (WX_i - y_i) \frac{\partial}{\partial W} \sum_i (WX_i - y_i)$

$= w_i (WX_i - y_i) * X_i$

$= (WX_i - y_i) w_i X_i$

1. Repeat until convergence {

$W_j = W_j - \alpha \sum_i (WX_i - y_i) w_i X_i$

}

HW 6.5

Write a MapReduce job in MRJob to do the training at scale of a weighted OLS model using gradient descent.

Generate one million datapoints just like in the following notebook: http://nbviewer.ipython.org/urls/dl.dropbox.com/s/kritdm3mo1daolj/MrJobLinearRegressionGD.ipynb (http://nbviewer.ipython.org/urls/dl.dropbox.com/s/kritdm3mo1daolj/MrJobLinearRegressionGD.ipynb)

Weight each example as follows:

weight(x)= abs(1/x)

Sample 1% of the data in MapReduce and use the sampled dataset to train a (weighted if available in SciKit-Learn) linear regression model locally using SciKit-Learn (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html))

Plot the resulting weighted linear regression model versus the original model that you used to generate the data. Comment on your findings.

Answer

Data Information:

Sizes: 1000000 points True model: y = 1.0 * x - 4 Noise：Normal Distributed mean = 0, var = 0.5

In [ ]:
```
import numpy as np
import pylab
import random

size = 1000000
sample_per = 1
x = np.random.uniform(-4, 4, size)
y = x * 1.0 - 4 + np.random.normal(0,0.5,size)
data = zip(y,x)
np.savetxt('LinearRegression.csv',data,delimiter = ",")


# pylab.plot(x, y,'*')
# pylab.title("Original Model")
# pylab.show()
```

In [ ]:
```
%%writefile MrJobBatchGD_LinearReg.py
from __future__ import division
from mrjob.job import MRJob
from mrjob.step import MRStep

# This MrJob calculates the gradient of the entire training set
#     Mapper: calculate partial gradient for each example

class MRJob_GD_LR(MRJob):
    def steps(self):
        return [MRStep(mapper_init=self.read_weights,
                       mapper=self.partial_gradient,
                       mapper_final=self.partial_gradient_emit,
                       reducer=self.gradient_accumulater,
                       jobconf={
                        "mapred.map.tasks":4,
                        "mapred.reduce.tasks":1
                        }
                      )]

    def read_weights(self):
        # Read initial_seed file
        with open('initial_seed.txt', 'r') as f:
            self.seed = [float(v) for v in f.readline().split(',')]
        # Initialze gradient for this iteration
        self.partial_Gradient = [0]*len(self.seed)
        self.partial_count = 0

    def partial_gradient(self,_,line):
        y_l,x_l = map(float,line.split(','))
        #assume equation is of the form y=b0+b1x. Get values of b0 and b1 from
the intial seed
        b0,b1 = self.seed

        #calculate weight as abs(1/x)
        w_l = abs(1/x_l)

        # y_hat is the predicted value given current weights
        y_hat = b0+b1*x_l

        # Update parial gradient vector with gradient form current example
        self.partial_Gradient =  [self.partial_Gradient[0]+ (y_l-y_hat)*w_l, se
lf.partial_Gradient[1]+(y_l-y_hat)*x_l*w_l]
        self.partial_count = self.partial_count + 1
        #yield None, (D[0]-y_hat,(D[0]-y_hat)*D[1],1)

    # Finally emit in-memory partial gradient and partial count
    def partial_gradient_emit(self):
        yield None, (self.partial_Gradient,self.partial_count)

    # Accumulate partial gradient from mapper and emit total gradient
    # Output: key = None, Value = gradient vector
    def gradient_accumulater(self, _, partial_Gradient_Record):
        total_gradient = [0]*2
        total_count = 0
        for partial_Gradient,partial_count in partial_Gradient_Record:
            total_count = total_count + partial_count
            total_gradient[0] += partial_Gradient[0]
            total_gradient[1] +=partial_Gradient[1]
        yield None, [v/total_count for v in total_gradient]

if __name__ == '__main__':
    MRJob_GD_LR.run()
```

In [ ]:
```
%load_ext autoreload
%autoreload 2

from numpy import random,array
from MrJobBatchGD_LinearReg import MRJob_GD_LR

np.random.seed(0)

learning_rate = 0.05
stop_criteria = 0.000005

# Generate random values as inital seed for b0 and b1
weights = array([random.uniform(-3,3),random.uniform(-3,3)])
# Write the weights to the files
with open('initial_seed.txt', 'w+') as f:
    f.writelines(','.join(str(j) for j in weights))

# create a mrjob instance for batch gradient descent update over all data
mr_job = MRJob_GD_LR(args=['LinearRegression.csv','--file','initial_seed.txt'])
# Update centroids iteratively
i = 0
while(1):
    print "iteration ="+str(i)+"  weights =",weights
    # Save weights from previous iteration
    weights_old = weights
    with mr_job.make_runner() as runner:
        runner.run()
        # stream_output: get access of the output
        for line in runner.stream_output():
            # value is the gradient value
            key,value =  mr_job.parse_output_line(line)
            # Update weights
            weights = weights + learning_rate*array(value)
    i = i + 1
    # Write the updated weights to file
    with open('initial_seed.txt', 'w+') as f:
        f.writelines(','.join(str(j) for j in weights))
    # Stop if weights get converged
    if(sum((weights_old-weights)**2)<stop_criteria):
        break

print "Final weights\n"
print weights
```

SK Learn Regression Model

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model


np.random.seed(0)  #  To ensure repeatability of results
shuffle = np.random.permutation(np.arange(np.array(x).shape[0]))
X,Y=x[shuffle],y[shuffle] #shuffle X and Y and get random sample
sample_per = 1
sample_sz = size*sample_per/100
X,Y=X[:sample_sz].reshape((sample_sz,1)),Y[:sample_sz].reshape((sample_sz,1))

#generate weight
w=abs(1/X).ravel()

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(X, Y,sample_weight=w)

# The coefficients
print('Coefficients:', regr.intercept_[0],regr.coef_[0][0])
```

In [ ]:
```

```

In [ ]:
```
y_hat = x * 1.0 - 4
mr_gd_lr = x * 1.00690185 - 3.94910008
sk_lr = X * 1.0006482340101668 - 3.9707791992944776


plt.figure(figsize=(18,6))

plt.subplot(1,3,1)
plt.plot(x, y,'o',mfc='none',alpha=0.1)
plt.plot(x,y_hat,linestyle='dashed',color='green')
plt.title("Original Model")

plt.subplot(1,3,2)
plt.plot(x, y,'o',mfc='none',alpha=0.1)
plt.plot(x,mr_gd_lr,linestyle='dashed',color='green')
plt.title("MR Job")

plt.subplot(1,3,3)
plt.plot(X, Y,'o',mfc='none',alpha=0.4)
plt.plot(X,sk_lr,linestyle='dashed',color='green')
plt.title("SK Learn")
plt.show()

plt.figure(figsize=(10,8))
plt.plot(x,y_hat,linestyle='dashed',color='black',label='Original Model',alpha=
0.5)
plt.plot(x,mr_gd_lr,linestyle='dotted',color='red',label='MR Job')
plt.plot(X,sk_lr,color='blue',label='SK Learn',alpha=0.5)
plt.legend(loc=2)
plt.show()
```

HW6.5.1 (Optional)

Using MRJob and in Python, plot the error surface for the weighted linear regression model using a heatmap and contour plot. Also plot the current model in the original domain space. (Plot them side by side if possible) Plot the path to convergence (during training) for the weighted linear regression model in plot error space and in the original domain space. Make sure to label your plots with iteration numbers, function, model space versus original domain space, etc. Comment on convergence and on the mean squared error using your weighted OLS algorithm on the weighted dataset versus using the weighted OLS algorithm on the uniformly weighted dataset.

In [ ]:

HW6.6 Clean up notebook for GMM via EM

Using the following notebook as a starting point:

http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/0t7985e40fovlkw/EM-GMM-MapReduce%20Design%201.ipynb (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/0t7985e40fovlkw/EM-GMM-MapReduce%20Design%201.ipynb)

Improve this notebook as follows:

-- Add in equations into the notebook (not images of equations)

-- Number the equations

-- Make sure the equation notation matches the code and the code and comments refer to the equations numbers

-- Comment the code

-- Rename/Reorganize the code to make it more readable

-- Rerun the examples similar graphics (or possibly better graphics)

Answer

Please use the link (http://nbviewer.jupyter.org/github/hchandaria/UCB_MIDS_W261/blob/master/hw6/EM-GMM-MapReduce%20Design%201_Modified.ipynb) to see the modified ipython notebook.

HW6.7 Implement Bernoulli Mixture Model via EM

Implement the EM clustering algorithm to determine Bernoulli Mixture Model for discrete data in MRJob.

As a unit test use the dataset in the following slides:

https://www.dropbox.com/s/maoj9jidxj1xf5l/MIDS-Live-Lecture-06-EM-Bernouilli-MM-Systems-Test.pdf?dl=0 (https://www.dropbox.com/s/maoj9jidxj1xf5l/MIDS-Live-Lecture-06-EM-Bernouilli-MM-Systems-Test.pdf?dl=0)

Cross-check that you get the same cluster assignments and cluster Bernouilli models as presented in the slides after 25 iterations. Dont forget the smoothing.

As a full test: use the same dataset from HW 4.5, the Tweet Dataset. Using this data, you will implement a 1000-dimensional EM-based Bernoulli Mixture Model algorithm in MrJob on the users by their 1000-dimensional word stripes/vectors using K = 4. Use the same smoothing as in the unit test.

Repeat this experiment using your KMeans MRJob implementation fron HW4. Report the rand index score using the class code as ground truth label for both algorithms and comment on your findings.

Here is some more information on the Tweet Dataset.

Here you will use a different dataset consisting of word-frequency distributions for 1,000 Twitter users. These Twitter users use language in very different ways, and were classified by hand according to the criteria:

0: Human, where only basic human-human communication is observed.

1: Cyborg, where language is primarily borrowed from other sources (e.g., jobs listings, classifieds postings, advertisements, etc...).

2: Robot, where language is formulaically derived from unrelated sources (e.g., weather/seismology, police/fire event logs, etc...).

3: Spammer, where language is replicated to high multiplicity (e.g., celebrity obsessions, personal promotion, etc...)

Check out the preprints of recent research, which spawned this dataset:

http://arxiv.org/abs/1505.04342 (http://arxiv.org/abs/1505.04342) http://arxiv.org/abs/1508.01843 (http://arxiv.org/abs/1508.01843)

The main data lie in the accompanying file:

topUsers_Apr-Jul_2014_1000-words.txt

and are of the form:

USERID,CODE,TOTAL,WORD1_COUNT,WORD2_COUNT,... . .

where

USERID = unique user identifier CODE = 0/1/2/3 class code TOTAL = sum of the word counts

Using this data, you will implement a 1000-dimensional K-means algorithm in MrJob on the users by their 1000-dimensional word stripes/vectors using several centroid initializations and values of K.

Bernoulli Mixture Model

In a Bernoulli Mixture Model a document is a vector of Booleans indicating the presence of a term. In this model, we generate a document by first picking a cluster k with prob- ability αk and then generating the terms of the document according to the parameters qmk.

The mixture model is :

$P(d|w_k;\theta) = \sum_{k=1}^{K} \alpha_k (\prod_{t_m \in d} q_{mk})(\prod_{t_m \notin d}(1 - q_{mk}))$ ⇒ equation 1

Where,

K = number of clusters

The maximization step recomputes the conditional parameters q_{mk} and the priors α_k as follows:

$q_{mk} = \frac{\sum_{n=1}^{N}(r_{nk})I(t_m \in d_n)}{\sum_{n=1}^{N} r_{nk}}$ ⇒ equation 2

$\alpha_k = \frac{\sum_{n=1}^{N} r_{nk}}{N}$ ⇒ equation 3

Please note that IR example has smoothing ϵ added in both equation 2 and 3 for but we have done that in the E step.

The expectation step computes the soft assignment of documents to clusters given the current parameters q_{mk} and α_k

Then, the probability of a document given its class is simply the product of the probability of the attribute values over all word attributes:

$P(d_i|w_k;\theta) = \prod_{t=1}^{|V|}$

Bit, is either 0 or 1, indicating whether word t_m occurs at least once in the document.

$r_{nk} = \frac{\alpha_k(\prod_{t_m \in d} q_{mk})(\prod_{t_m \notin d}(1-q_{mk}))}{\sum_{k=1}^{K} \alpha_k(\prod_{t_m \in d} q_{mk})(\prod_{t_m \notin d}(1-q_{mk}))}$ ⇒ equation 4

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
%%writefile BMMtest.txt
hot chocolate cocoa beans
cocoa ghana africa
beans harvest ghana
cocoa butter
butter truffles
sweet chocolate
sweet sugar
sugar cane brazil
sweet sugar beet
sweet cake icing
cake black forest
```

BMM Initialization for IR example

In [ ]:
```
%%writefile mr_BMMEmInitialize.py
from __future__ import division
from mrjob.job import MRJob

from numpy import mat, zeros, shape, random, array, zeros_like, dot, linalg
from random import sample
import json, re
from math import pi, sqrt, exp, pow

WORD_RE = re.compile(r"[\w']+")

class MrBMM_EmInit(MRJob):
#     DEFAULT_PROTOCOL = 'json'
    vocab={}
    qmk ={}

    def __init__(self, *args, **kwargs):
        super(MrBMM_EmInit, self).__init__(*args, **kwargs)

        self.numMappers = 1     #number of mappers
        self.count = 0



    def configure_options(self):
        super(MrBMM_EmInit, self).configure_options()
        self.add_passthrough_option(
            '--k', dest='k', default=2, type='int',
            help='k: number of densities in mixture')
        self.add_passthrough_option(
            '--pathName', dest='pathName', default="", type='str',
            help='pathName: pathname where intermediateResults_BMM.txt is store
d')

    def mapper_init(self):
        for i in range (0,self.options.k):
            self.vocab[i]={}
        #build vocab first
        with open('BMMtest.txt') as f:
            for line in f:
                words = re.findall(WORD_RE,line)
                for word in words:
                    for i in range(self.options.k):
                        if word in self.vocab[i]:
                            continue
                        self.vocab[i][word]=0

    def mapper(self, key, xjIn):
        #something simple to grab random starting point
        #collect the first 2k
        self.count += 1
        if self.count == 6:
            yield (0,xjIn)
        if self.count ==7:
            yield(1,xjIn)

    def reducer(self, key, xjIn):
        for xj in xjIn:
            words = re.findall(WORD_RE,xj)
            for word in words:
                if word not in self.vocab[key]:
                    continue
                self.vocab[key][word]=1
                yield key, word

        jDebug = json.dumps([self.vocab])
        debugPath = self.options.pathName + 'debug_BMM.txt'
        fileOut = open(debugPath,'w')
        fileOut.write(jDebug)
```

In [ ]:
```
! python mr_BMMEmInitialize.py --pathName '/Users/hetal/programming/W261/UCB_MI
DS_W261/hw6/' --file 'BMMtest.txt' -q BMMtest.txt
```

In [ ]:
```
%%writefile mr_BMixEmIterate.py
from __future__ import division
from mrjob.job import MRJob

from math import sqrt, exp, pow,pi
from numpy import zeros, shape, random, array, zeros_like, dot, linalg, log,exp
import numpy as np
import json, re
import collections
from decimal import *

WORD_RE = re.compile(r"[\w']+")

# @terms = terms present in the incoming document
# @alpha = prior probability for cluster k
# @qmk_local = vocab probability for cluster k
# return partially calculated  rnk (numerator)
def bernoulli(terms, alpha,qmk_local,n,k,epsilon):
    lprob = 0.0 #log probability
    for key,value in qmk_local.iteritems():
        if key not in terms: #if we have never seen the term in the document
            continue
        lprob += log(terms[key]*(value + epsilon) + ((1-terms[key] )*(1-value +
 epsilon)))
    lprob =  Decimal(log(alpha)+lprob).exp()
    return lprob



class MrBMixEm(MRJob):
    DEFAULT_PROTOCOL = 'json'

    def __init__(self, *args, **kwargs):
        super(MrBMixEm, self).__init__(*args, **kwargs)

        fullPath = self.options.pathName + 'intermediateResults_BMM.txt'
        fileIn = open(fullPath)
        inputJson = fileIn.read()
        fileIn.close()
        inputList = json.loads(inputJson)
        temp = inputList[0]
        self.alpha = array(temp)           #prior class probabilities
        temp = inputList[1] #entire vocab

        self.qmk = {int(k):v for k,v in temp.items()}

        self.vocab = []
        self.new_qmk = {}
        for i in range(self.options.k):
            self.new_qmk[i]={}
        for key,value in self.qmk.iteritems():
            for term,val in value.iteritems():
                self.new_qmk[key][term]=0
                if(val != 0):
                    self.vocab.append(term)

        self.vocab =  set(self.vocab)

        self.new_alphas = zeros_like(self.alpha)        #partial weighted sum o
f weights

        self.numMappers = 1             #number of mappers
        self.count = 0                  #passes through mapper


    def configure_options(self):
        super(MrBMixEm, self).configure_options()

        self.add_passthrough_option(
```

In [ ]:
```
!python mr_BMixEmIterate.py --file intermediateResults_BMM.txt --pathName '/Use
rs/hetal/programming/W261/UCB_MIDS_W261/hw6/' -q BMMtest.txt
```

In [ ]:
```
%load_ext autoreload
%autoreload 2

from mr_BMMEmInitialize import MrBMM_EmInit
from mr_BMixEmIterate import MrBMixEm
import json
from math import sqrt

#function to calculate distance
def dist(x,y):
    #euclidean distance between two lists
    sum = 0.0
    for i in range(len(x)):
        temp = x[i] - y[i]
        sum += temp * temp
    return sqrt(sum)

#first run the initializer to get starting centroids
filePath = 'BMMtest.txt'
mrJob = MrBMM_EmInit(args=[filePath,'--file',filePath,'--pathName','/Users/heta
l/programming/W261/UCB_MIDS_W261/hw6/'])
with mrJob.make_runner() as runner:
    runner.run()

#pull out the centroid values to compare with values after one iteration
emPath = "intermediateResults_BMM.txt"
fileIn = open(emPath)
paramJson = fileIn.read()
fileIn.close()

k = 2

delta = 10
iter_num = 0
#Begin iteration on change in centroids
while iter_num <= 24:
    print "Iteration" + str(iter_num)
    iter_num = iter_num + 1
    #parse old centroid values
    oldParam = json.loads(paramJson)
    #run one iteration
    oldMeans = oldParam[1]
    old_qmk_dict = {int(k):v for k,v in oldMeans.items()}

    mrJob2 = MrBMixEm(args=[filePath,'--file',emPath,'--pathName','/Users/hetal
/programming/W261/UCB_MIDS_W261/hw6/'])
    with mrJob2.make_runner() as runner:
        runner.run()

    #compare new centroids to old ones
    fileIn = open(emPath)
    paramJson = fileIn.read()
    fileIn.close()
    newParam = json.loads(paramJson)

    k_means = len(newParam[1])
    newMeans = newParam[1]
    new_qmk_dict = {int(k):v for k,v in newMeans.items()}

    #convert dictionary to array so we can compute distance
    new_qmk = old_kmk =  [[]*k for x in xrange(k)]
    for i in range(k):
        for key , value in new_qmk_dict[i].iteritems() :
            new_qmk[i].append(float(value))
        for key , value in old_qmk_dict[i].iteritems() :
            old_qmk[i].append(float(value))

    delta = 0.0
    for i in range(k_means):
```

In [ ]:

Initalization Code for Tweet data

In [ ]:
```
%%writefile mr_BMMEmInitialize.py
from __future__ import division
from mrjob.job import MRJob

import numpy as np
from random import sample
import json, re
from math import pi, sqrt, exp, pow

WORD_RE = re.compile(r"[\w']+")

class MrBMM_EmInit(MRJob):
#     DEFAULT_PROTOCOL = 'json

    def __init__(self, *args, **kwargs):
        super(MrBMM_EmInit, self).__init__(*args, **kwargs)

        self.numMappers = 1     #number of mappers
        self.count = 0


    def configure_options(self):
        super(MrBMM_EmInit, self).configure_options()
        self.add_passthrough_option(
            '--k', dest='k', default=4, type='int',
            help='k: number of densities in mixture')
        self.add_passthrough_option(
            '--pathName', dest='pathName', default="", type='str',
            help='pathName: pathname where intermediateResults_BMM.txt is store
d')

    def mapper(self, key, xjIn):
        if self.count <= 2*self.options.k:
            self.count += 1
            yield (1,xjIn)

    def reducer(self, key, xjIn):
        cent=[]
        for xj in xjIn:
            terms = xj.strip().split(',')[3:]
            cent.append(terms) #append the data points
        index = sample(range(len(cent)), self.options.k) #based on the number o
f clusters, select those many points randomly

        qmk =np.zeros([self.options.k,len(cent[0])])
        qmk_i = 0
        for i in index:
            for j in range(len(cent[i])):
                if (cent[i][j] == '0' ):
                    continue
                qmk[qmk_i][j]=1
            qmk_i +=1

        #also need a starting guess at the phi's - prior probabilities
        #initialize them all with the same number - 1/k - equally probably for
each cluster

        alpha = np.zeros(self.options.k,dtype=float)

        for i in range(self.options.k):
            alpha[i] = 1.0/float(self.options.k)

        #form output object
        outputList = [alpha.tolist(),qmk.tolist()]

        jsonOut  = json.dumps(outputList)

        #write new parameters to file
        fullPath = self.options.pathName + 'intermediateResults_BMM_tweet.txt'
```

In [1]:
```
! python mr_BMMEmInitialize.py --pathName '/Users/hetal/programming/W261/UCB_MI
DS_W261/hw6/' -q topUsers_Apr-Jul_2014_1000-words.txt
```

In [7]:
```
%%writefile mr_BMixEmIterate.py
from __future__ import division
from mrjob.job import MRJob

from math import sqrt, exp, pow,pi
from numpy import zeros, shape, random, array, zeros_like, dot, linalg, log,exp
import numpy as np
import json, re
import collections
from decimal import *

WORD_RE = re.compile(r"[\w']+")

# @terms = terms present in the incoming document
# @alpha = prior probability for cluster k
# @qmk_local = vocab probability for cluster k
#@epsilon = smoothing
# return partially calculated  rnk (numerator)
def bernoulli(terms, alpha,qmk_local,epsilon):
    lprob = 0.0 #log probability
    for i in range(len(qmk_local)):
        if np.isnan(terms[i]) : #if we have never seen the term in the document

            continue
        lprob += log(terms[i]*(qmk_local[i] + epsilon) + ((1-terms[i] )*(1-qmk_
local[i] + epsilon)))
    lprob =  Decimal(log(alpha)+lprob).exp()
    return lprob



class MrBMixEm(MRJob):
    DEFAULT_PROTOCOL = 'json'

    def __init__(self, *args, **kwargs):
        super(MrBMixEm, self).__init__(*args, **kwargs)

        fullPath = self.options.pathName + 'intermediateResults_BMM_tweet.txt'
        fileIn = open(fullPath)
        inputJson = fileIn.read()
        fileIn.close()
        inputList = json.loads(inputJson)
        temp = inputList[0]
        self.alpha = array(temp)           #prior class probabilities
        self.qmk = inputList[1] #entire vocab
        self.vocab = [] # for the first round determing which indexes have data
        self.new_qmk = zeros_like(self.qmk)
        for i in range(len(self.qmk)) :
            for j in range(len(self.qmk[i])):
                if(self.qmk[i][j] != 0):
                    self.vocab.append(j)

        self.vocab =  set(self.vocab)

        self.new_alphas = zeros_like(self.alpha)        #partial weighted sum o
f weights

#         self.numMappers = 1             #number of mappers
        self.count = 0                  #passes through mapper



    def configure_options(self):
        super(MrBMixEm, self).configure_options()

        self.add_passthrough_option(
            '--k', dest='k', default=4, type='int',
            help='k: number of densities in mixture')
        self.add_passthrough_option(
            '--pathName', dest='pathName', default="", type='str',
```

```
Overwriting mr_BMixEmIterate.py
```

In [8]:
```
! python mr_BMixEmIterate.py \
--pathName '/Users/hetal/programming/W261/UCB_MIDS_W261/hw6/' \
--file 'intermediateResults_BMM_tweet.txt' \
--file 'cluster_assignments'
-q \
topUsers_Apr-Jul_2014_1000-words.txt
```

Driver code

In [14]:
```
%load_ext autoreload
%autoreload 2

from mr_BMMEmInitialize import MrBMM_EmInit
from mr_BMixEmIterate import MrBMixEm
import json
from math import sqrt

#function to calculate distance
def dist(x,y):
    #euclidean distance between two lists
    sum = 0.0
    for i in range(len(x)):
        temp = x[i] - y[i]
        sum += temp * temp
    return sqrt(sum)

#first run the initializer to get starting centroids
filePath = 'topUsers_Apr-Jul_2014_1000-words.txt'
mrJob = MrBMM_EmInit(args=[filePath,'--pathName','/Users/hetal/programming/W261
/UCB_MIDS_W261/hw6/'])
with mrJob.make_runner() as runner:
    runner.run()

#pull out the centroid values to compare with values after one iteration
emPath = "intermediateResults_BMM_tweet.txt"
fileIn = open(emPath)
paramJson = fileIn.read()
fileIn.close()

k = 4

delta = 10
iter_num = 0
#Begin iteration on change in centroids
while delta > 0.0001:
    print "Iteration" + str(iter_num)
    iter_num = iter_num + 1
    #parse old centroid values
    oldParam = json.loads(paramJson)
    old_alpha = oldParam[0]
    #run one iteration
    old_qmk = oldParam[1]

    mrJob2 = MrBMixEm(args=[filePath,'--file',emPath,'--pathName','/Users/hetal
/programming/W261/UCB_MIDS_W261/hw6/'])
    with mrJob2.make_runner() as runner:
        runner.run()

    #compare new centroids to old ones
    fileIn = open(emPath)
    paramJson = fileIn.read()
    fileIn.close()
    newParam = json.loads(paramJson)

    clusters = len(newParam[1])
    new_alpha = newParam[0]
    new_qmk = newParam[1]


    delta = 0.0
#     for i in range(clusters):
    delta += dist(new_alpha,old_alpha)

    print oldParam[0]

print "Iteration" + str(iter_num)
print new_alpha
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
Iteration0
[0.25, 0.25, 0.25, 0.25]
Iteration1
[0.9618331222137054, 0.0010000333288894814, 0.016333322263695507, 0.0208335221
9370948]
Iteration2
[0.9615336485318104, 0.0010000154383694182, 0.014444728269157705, 0.0230216077
6066261]
Iteration3
[0.9611248910967705, 0.0010000081403497355, 0.013815197965741958, 0.0240599027
9713767]
Iteration4
[0.9604182116896928, 0.0010000070328964738, 0.013581702152178535, 0.0250000791
25232454]
Iteration5
[0.9608553614659487, 0.0010000069630589875, 0.013127448861346081, 0.0250171827
0964633]
Iteration6
[0.9600259575269915, 0.0010000069577990945, 0.012972623165717814, 0.0260014123
49491552]
Iteration7
[0.9590337669102873, 0.0010000006913985314, 0.012969124155125557, 0.02699710202
0601665]
Iteration8
[0.9589596198326018, 0.0010000006966143332, 0.012966938515359978, 0.02707343468
5895046]
Iteration9
[0.958031090845865, 0.0010000069767846659, 0.012966417687920177, 0.02800248448
943001]
Iteration10
[0.9577436116205024, 0.0010000070253156719, 0.012964851738706750, 0.02829152961
5475042]
Iteration11
[0.957035251197139, 0.0010000070362882636, 0.012964112077887124, 0.02900062968
868552]
Iteration12
[0.9570015229315424, 0.0010000070484900104, 0.012962857566814714, 0.0290356124
5315311]
```

In [ ]:

HW6.8 (Optional) 1 Million songs

Predict the year of the song. Ask Jimi

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: