In [1]:

```
import gzip
from collections import defaultdict
import math
import scipy.optimize
import numpy
import string
import random
import dateutil.parser
from sklearn import linear_model
```

In [2]:

```
def parse(f):
    for l in gzip.open(f):
        yield eval(l)
```

In [3]:

```
# Download data from below:
# https://cseweb.ucsd.edu/classes/fa21/cse258-b/files/
dataset = list(parse("trainRecipes.json.gz"))
```

In [4]:

```
len(dataset)
```

Out[4]:

```
200000
```

In [5]:

```
train = dataset[:150000]
valid = dataset[150000:175000]
test = dataset[175000:]
```

In [6]:

```
dataset[1]
```

Out[6]:

```
{'name': 'double delicious cookie bars',
 'minutes': 40,
 'contributor_id': '26865936',
 'submitted': '2007-08-27',
 'steps': 'preheat oven to 350f\tin 13x9-inch baking pan , melt bu
tter in oven\tsprinkle crumbs evenly over butter\tpour milk evenly
over crumbs\ttop with remaining ingredients\tpress down firmly\tba
ke 25-30 minutes or until lightly browned\tcool completely , chill
if desired , and cut into bars',
 'description': 'from "all time favorite recipes". for fun, try su
bstituting butterscotch or white chocolate chips for the semi-swee
t and/or peanut butter chips. make sure you cool it completely or
the bottom will crumble!',
 'ingredients': ['butter',
  'graham cracker crumbs',
  'sweetened condensed milk',
  'semi-sweet chocolate chips',
  'peanut butter chips'],
 'recipe_id': '98015212'}
```

Section 1

Question 1 (a)

In [7]:
```
# feature 1 (a) use the length of the recipe
def feat1a(d):
    length_step = len(d['steps'])
    length_ingredients = len(d['ingredients'])
    return [length_step, length_ingredients]
X = [feat1a(d) for d in dataset]
print(X[0])
```

```
[743, 9]
```

Question 1 (b)

In [8]:
```
for d in dataset:
    t = dateutil.parser.parse(d['submitted'])
    d['submitted'] = t
```

In [9]:
```
dataset[0]['submitted']
```

Out[9]:
```
datetime.datetime(2004, 5, 21, 0, 0)
```

In [10]:
```
min_year = min([d['submitted'].year for d in dataset])
max_year = max([d['submitted'].year for d in dataset])
```

In [11]:
```
min_year, max_year
```

Out[11]:
```
(1999, 2018)
```

In [12]:
```
year_length = max_year - min_year + 1
year_length
```

Out[12]:
```
20
```

In [13]:
```
# feature 1 b
def feat1b(d):
    month = [0]*12
    pd = d['submitted']
    month[pd.month-1] = 1
    year = [0]*year_length
    year[pd.year - min_year] = 1
    return month[:-1] + year[:-1]
```

In [14]:

```
X = [feat1b(d) for d in dataset]
print(X[0])
```

```
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0]
```

Question 1(c)

In [15]:

```
recipeCount = defaultdict(int)
def popular(dataset):
    for d in dataset:
        for recipe in d['ingredients']:
            recipeCount[recipe] += 1
    mostPopular = [(recipeCount[x], x) for x in recipeCount]
    mostPopular.sort()
    mostPopular.reverse()
    return mostPopular
```

In [16]:

```
mostPopular = popular(dataset)
mostPopular1 = mostPopular[:50]
```

In [17]:

```
# feature 1 c
def feat1c(d):
    ingredients = [0]*50
    for pd in d['ingredients']:
        for i in range(50):
            if pd == mostPopular[i][1]:
                ingredients[i] = 1
    return ingredients
X = [feat1c(d) for d in dataset]
print(X[0])
```

```
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0]
```

Use function to represent feature a, b, c

In [18]:

```
def feat(d, a, b, c):
    X = [1]
    if (a == True):
        X = X + feat1a(d)
    if (b == True):
        X = X + feat1b(d)
    if (c == True):
        X = X + feat1c(d)
    return X
```

In [19]:

```
def experiment(a, b, c, mod):
    X = [feat(d, a, b, c) for d in dataset]
    Y = [d['minutes'] for d in dataset]
    Xtrain = X[:150000]
    Xtest = X[175000:]
    Ytrain = Y[:150000]
    Ytest = Y[175000:]

    mod.fit(Xtrain,Ytrain)
    Ypred = mod.predict(Xtest)
    MSE = sum([(yp - yt)**2 for (yp,yt) in zip(Ypred, Ytest)]) / le

    print("test MSE = " + str(MSE))
```

In [20]:

```
# MSE of feature 1(a)
print("The MSE when using 1(a) feature")
experiment(True, False, False, linear_model.LinearRegression())
```

```
The MSE when using 1(a) feature
test MSE = 6169.549296366476
```

In [21]:

```
# MSE of feature 1(b)
print("The MSE when using 1(b) feature")
experiment(False, True, False, linear_model.LinearRegression())
```

```
The MSE when using 1(b) feature
test MSE = 6396.833687711828
```

In [22]:

```
# MSE of feature 1(c)
print("The MSE when using 1(c) feature")
experiment(False, False, True, linear_model.LinearRegression())
```

```
The MSE when using 1(c) feature
test MSE = 6000.948439855985
```

Question 2

In [23]:

```
# MSE of feature 1(a) and 1 (b)
print("The MSE when using 1(a) and 1 (b) feature")
experiment(True, True, False, linear_model.LinearRegression())
```

```
The MSE when using 1(a) and 1 (b) feature
test MSE = 6157.7540943661925
```

```
In [24]: # MSE of feature 1(a) and 1 (c)
         print("The MSE when using 1(a) and 1 (c) feature")
         experiment(True, False, True, linear_model.LinearRegression())
```

```
The MSE when using 1(a) and 1 (c) feature
test MSE = 5870.115061656083
```

```
In [25]: # MSE of feature 1(b) and 1 (c)
         print("The MSE when using 1(b) and 1 (c) feature")
         experiment(False, True, True, linear_model.LinearRegression())
```

```
The MSE when using 1(b) and 1 (c) feature
test MSE = 5992.663510100711
```

```
In [26]: # MSE of all features
         print("The MSE when using all features")
         experiment(True, True, True, linear_model.LinearRegression())
```

```
The MSE when using all features
test MSE = 5861.253905671346
```

from the above test, we can find that feature extracted from c is the most import

Question 4

(1) For the first implementation, we can simply delete the instances which is unusual in the dataset, such as the instances whose cooking time is over 8 hours. We can set up a range, and the instances whose cooking time is not in the range will be deleted from the dataset. This is convenient and efficient.

(2) For the second implementation, we can transform the variable y, for example, transformation such that $y' = log(y)$ to make it less prone to outliers like cooking time over 8 hours

(3) For the third implementation, instead of regression, we can build this problem as a classification issue. For example, predict whether y is above or below a certain minute to improve the result.

Section 2: Classification

Question 5

In [27]:

```
def feat1c_question2(d):
    ingredients = [0]*50
    for pd in d['ingredients']:
        for i in range(50):
            if pd == 'butter':
                continue
            if pd == mostPopular[i][1]:
                ingredients[i] = 1
    return ingredients
```

In [28]:

```
def y_question2(d):
    for pd in d['ingredients']:
        if pd == 'butter':
            return 1
    return 0
```

In [29]:

```
X_question2 = [feat1c_question2(d) for d in dataset]
Y_question2 = [y_question2(d) for d in dataset]
```

In [30]:

```
X_train = X_question2[:150000]
y_train = Y_question2[:150000]
X_test = X_question2[175000:]
y_test = Y_question2[175000:]
```

In [31]:

```
mod = linear_model.LogisticRegression(C=1.0, class_weight='balanced
mod.fit(X_train, y_train)
pred = mod.predict(X_test)
correct = pred == y_test
```

In [32]:

```
TP_ = numpy.logical_and(pred, y_test)
FP_ = numpy.logical_and(pred, numpy.logical_not(y_test))
TN_ = numpy.logical_and(numpy.logical_not(pred), numpy.logical_not(
FN_ = numpy.logical_and(numpy.logical_not(pred), y_test)

TP = sum(TP_)
FP = sum(FP_)
TN = sum(TN_)
FN = sum(FN_)

BER = 1 - 0.5 * (TP / (TP + FN) + TN / (TN + FP))
```

In [33]:

```
BER
```

Out[33]:

```
0.28898437523315856
```

Question 6

In [34]:

```
mostPopular2 = mostPopular[:70]
def feat1c_question2(d):
    ingredients = [0]*70
    for pd in d['ingredients']:
        for i in range(70):
            if pd == 'butter':
                continue
            if pd == mostPopular2[i][1]:
                ingredients[i] = 1
    return ingredients
```

In [35]:

```
def y_question2(d):
    for pd in d['ingredients']:
        if pd == 'butter':
            return 1
    return 0

X_question2 = [feat1c_question2(d) for d in dataset]
Y_question2 = [y_question2(d) for d in dataset]

X_train = X_question2[:150000]
y_train = Y_question2[:150000]
X_test = X_question2[175000:]
y_test = Y_question2[175000:]
X_validation = X_question2[150000:175000]
y_validation = Y_question2[150000:175000]


mod = linear_model.LogisticRegression(C=1.0, class_weight='balanced
mod.fit(X_train, y_train)
pred = mod.predict(X_validation)
correct = pred == y_validation

TP_ = numpy.logical_and(pred, y_validation)
FP_ = numpy.logical_and(pred, numpy.logical_not(y_validation))
TN_ = numpy.logical_and(numpy.logical_not(pred), numpy.logical_not(
FN_ = numpy.logical_and(numpy.logical_not(pred), y_validation)

TP = sum(TP_)
FP = sum(FP_)
TN = sum(TN_)
FN = sum(FN_)

BER = 1 - 0.5 * (TP / (TP + FN) + TN / (TN + FP))
print('When I change the number of ingredients from 50 to 70.')
print('BER of validation set: ' + str(BER))

mod = linear_model.LogisticRegression(C=1.0, class_weight='balanced
mod.fit(X_train, y_train)
pred = mod.predict(X_test)
correct = pred == y_test

TP_ = numpy.logical_and(pred, y_test)
TP_ = numpy.logical_and(pred, y_test)
FP_ = numpy.logical_and(pred, numpy.logical_not(y_test))
TN_ = numpy.logical_and(numpy.logical_not(pred), numpy.logical_not(
FN_ = numpy.logical_and(numpy.logical_not(pred), y_test)

TP = sum(TP_)
FP = sum(FP_)
TN = sum(TN_)
FN = sum(FN_)

BER = 1 - 0.5 * (TP / (TP + FN) + TN / (TN + FP))

print('BER of testing set: ' + str(BER))
```

```
When I change the number of ingredients from 50 to 70.
BER of validation set: 0.27181412337662336
BER of testing set: 0.27340430377454994
```

Section 3

Question 8

In [36]:

```
from collections import defaultdict
usersPerItem = defaultdict(set) # Maps an item to the users who rat
itemsPerUser = defaultdict(set) # Maps a user to the items that the

for d in dataset:
    user,item = d['recipe_id'], d['ingredients']
    for pd in item:
        usersPerItem[user].add(pd)
        itemsPerUser[pd].add(user)
```

In [37]:

```
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    if denom == 0:
        return 0
    return numer / denom
```

In [38]:

```
def mostSimilar(i, N):
    similarities = []
    users = usersPerItem[i]
    for i2 in usersPerItem:
        if i2 == i: continue
        sim = Jaccard(users, usersPerItem[i2])
        similarities.append((sim,i2))
    similarities.sort(key = lambda x:(-x[0],x[1]))
    return similarities[:N]
```

In [39]:
```
query = dataset[0]['recipe_id']
ms = mostSimilar(query, 5)
print('five most similar recipes which are most similar to butter')
ms
```

```
five most similar recipes which are most similar to butter
```

Out[39]:
```
[(0.4166666666666667, '68523854'),
 (0.38461538461538464, '12679596'),
 (0.36363636363636365, '56301588'),
 (0.36363636363636365, '79675099'),
 (0.35714285714285715, '87359281')]
```

Question 9

In [40]:
```
def mostSimilarUser(i, N):
    similarities = []
    items = itemsPerUser[i]
    for i2 in itemsPerUser:
        if i2 == i: continue
        sim = Jaccard(items, itemsPerUser[i2])
        #sim = Pearson(i, i2) # Could use alternate similarity metr
        similarities.append((sim,i2))
    similarities.sort(key = lambda x:(-x[0],x[1]))
    return similarities[:N]
```

In [41]:
```
query = 'butter'
ms = mostSimilarUser(query, 5)
print('five ingredients which are most similar to butter')
print(ms)
```

```
five ingredients which are most similar to butter
[(0.22315311514274808, 'salt'), (0.2056685424969639, 'flour'), (0.
19100394157199166, 'eggs'), (0.17882420717656095, 'sugar'), (0.170
40052045973944, 'milk')]
```

Question 10

In [42]:
```
from collections import defaultdict
usersPerItem = defaultdict(set) # Maps an item to the users who rat
itemsPerUser = defaultdict(set) # Maps a user to the items that the

for d in dataset:
    user,item = d['recipe_id'], d['ingredients']
    for pd in item:
        usersPerItem[user].add(pd)
        itemsPerUser[pd].add(user)
```

Firstly, I used Jaccard similarity to find similar ingredient of the items in query and put them in the recipe as well. Subsequently, I use Jaccard similarity to find the most 10 similar recipes to the recipe I made and this achieves the flexibility needed in this task

In [43]:
```
def similarQuestion10(query):
    for pd in query:
        usersPerItem['1'].add(pd)
        query2 = mostSimilarUser(pd, 1)
        for pd2 in query2:
            usersPerItem['1'].add(pd2[1])
    ms = mostSimilar('1', 10)
    returnval = []
    for pd3 in ms:
        for d in dataset:
            if d['recipe_id'] == pd3[1]:
                returnval.append((d['recipe_id'], d['ingredients'])
    return returnval
```

In [44]:
```
query = ['vodka', 'sugar']
ms = similarQuestion10(query)
ms
```

Out[44]:
```
[('22558882', ['vodka', 'cranberry juice']),
 ('34964059', ['sugar', 'vodka']),
 ('93617905', ['watermelon', 'vodka', 'lemon juice', 'sugar', 'sal
t']),
 ('04595917',
  ['red cabbage', 'apple', 'cranberry juice', 'vinegar', 'salt', '
sugar']),
 ('52087715',
  ['beet', 'cornstarch', 'salt', 'cider vinegar', 'vodka', 'sugar'
]),
 ('65687449',
  ['vodka',
   'grand marnier',
   'amaretto',
   'cranberry juice',
   'orange slice',
   'sugar']),
 ('76576175',
  ['vodka',
   'pineapple juice',
   'cranberry juice',
   'ginger ale',
   'sugar',
   'ice']),
 ('00656946', ['vodka', 'lime', 'cranberry juice']),
 ('01257924', ['sugar', 'hazelnut-flavored liqueur', 'vodka']),
 ('03250654', ['vodka', 'peach schnapps', 'cranberry juice'])]
```