Section 1 (Regression)

In [15]:

```
import gzip
from collections import defaultdict
import math
import scipy.optimize
import numpy as np
import string
import random
from sklearn import linear_model
```

In [136]:

```
import dateutil.parser
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import mean_squared_error, confusion_matrix
from sklearn.linear_model import LinearRegression, LogisticRegression
```

In [4]:

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

In [5]:

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

In [6]:

```
len(dataset)
```

Out[6]:

```
200000
```

In [7]:

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

In [8]:

```
dataset[1]
```

Out[8]:

```
{'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 butter in ov
en\tsprinkle crumbs evenly over butter\tpour milk evenly over crumbs\ttop wi
th remaining ingredients\tpress down firmly\tbake 25-30 minutes or until lig
htly browned\tcool completely , chill if desired , and cut into bars',
 'description': 'from "all time favorite recipes". for fun, try substituting
butterscotch or white chocolate chips for the semi-sweet and/or peanut butte
r 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'}
```

Question 1

In [124]:

```
def feat1a(d):
    features = []
    for datum in d:
        features.append([len(datum['steps']), len(datum['ingredients'])])
    return np.asarray(features, dtype=object)
```

In [125]:

```
def feat1b(d):
    features = []

    t = [dateutil.parser.parse(datum['submitted']) for datum in d]
    t_years = [[day.year] for day in t]
    t_months = [[day.month] for day in t]
    enc = OneHotEncoder(drop='first', handle_unknown='error')
    month_onehot = enc.fit_transform(t_months).toarray()
    year_onehot = enc.fit_transform(t_years).toarray()

    return np.hstack((month_onehot, year_onehot))
```

In [126]:

```
def getTopIngredients(d = dataset):
    ingredientDict = defaultdict(int)

    for datum in d:
        for ingredient in datum['ingredients']:
            ingredientDict[ingredient] += 1

    topIngredientsList = []
    for key in ingredientDict:
        topIngredientsList.append((key, ingredientDict[key]))

    topIngredientsList = sorted(topIngredientsList, key=lambda x: x[1], reverse=True)

    return topIngredientsList
```

In [127]:

```
def feat1c(d):
    features = []

    topIngredientsList = getTopIngredients()[:50]

    ingredientSet = defaultdict(int)
    for i in range(len(topIngredientsList)):
        ingredientSet[topIngredientsList[i][0]] = i

    for datum in d:
        tempVec = [0]*50
        for ingredient in datum['ingredients']:
            if ingredient in ingredientSet:
                tempVec[ingredientSet[ingredient]] = 1
        features.append(tempVec)

    return np.asarray(features, dtype=object)
```

In [128]:

```
def mergeFeatures(features, temp):
    if features.size == 0:
        return temp
    else:
        return np.hstack((features, temp))
```

In [129]:

```
def feat(d, a = True, b = True, c = True):
    # Hint: for Questions 1 and 2, might be useful to set up a function like this
    #       which allows you to "select" which features are included

    features = np.array([])
    if a:
        features = mergeFeatures(features, feat1a(d))
    if b:
        features = mergeFeatures(features, feat1b(d))
    if c:
        features = mergeFeatures(features, feat1c(d))

    return features
```

In [130]:

```
def MSE(y, ypred):
    # Can use library if you prefer
    return mean_squared_error(y, ypred)
```

In [131]:

```
def getLabels(d):
    return np.asarray([datum['minutes'] for datum in d])
```

In [132]:

```
def experiment(mod, a = True, b = True, c = True):
    # Hint: might be useful to write this function which extracts features and
    #       computes the performance of a particular model on those features
    train_feat = feat(train, a, b, c)
    test_feat = feat(test, a, b, c)
    y_train = getLabels(train)
    print('first features: ', train_feat[0])
    mod.fit(train_feat, y_train)
    y_pred_test = mod.predict(test_feat)
    y_test = getLabels(test)
    print('MSE: ', MSE(y_test, y_pred_test))
```

In [133]:

```
model = LinearRegression(fit_intercept=True)
print('Model 1A: ')
experiment(model, b = False, c = False)
print('\nModel 1B: ')
experiment(model, a = False, c = False)
print('\nModel 1C: ')
experiment(model, b = False, a = False)
```

```
Model 1A:
first features:  [743 9]
MSE:  6169.549296366498

Model 1B:
first features:  [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. 0.]
MSE:  6396.833687711815

Model 1C:
first features:  [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]
MSE:  6000.948439855976
```

Question 2

In [135]:

```
model = LinearRegression(fit_intercept=True)
print('Ablation Model 1A: ')
experiment(model, a = False)
print('\nAblation Model 1B: ')
experiment(model, b = False)
print('\nAblation Model 1C: ')
experiment(model, c = False)
print('\nAblation Model Full: ')
experiment(model)
```

```
Ablation Model 1A:
first features:  [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 0.0 0.
0 1.0 0.0 0.0
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 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 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
MSE:  5992.663510100702

Ablation Model 1B:
first features:  [743 9 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]
MSE:  5870.115061656059

Ablation Model 1C:
first features:  [743 9 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
0.0 0.0 1.0 0.0
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0]
MSE:  6157.754094366206

Ablation Model Full:
first features:  [743 9 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
0.0 0.0 1.0 0.0
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 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]
MSE:  5861.2539056713895
```

In [103]:

```
print("""
By removing feature C, we get the largest increase in MSE, thus we
can argue that feature C is an important feature
""")
```

```
By removing feature C, we get the largest increase in MSE, thus we
can argue that feature C is an important feature
```

Question 4

In [89]:

```
print("""
MSE is a poor choice of loss in datasets that have significant outliers
because the predictor may be dominated by a few very long cooking times
in the 8+ hour range. By using MSE we try to force the model to also be
a good predictor for these outliers that are not representative of the data

1) Transforming the output variable such that y_new = log(y)
2) Simply delete outlying instances from dataset by defining some range
    [y_min, y_max] and discarding all instances (x, y) outside of that range
""")
```

```
MSE is a poor choice of loss in datasets that have significant outliers
because the predictor may be dominated by a few very long cooking times
in the 8+ hour range. By using MSE we try to force the model to also be
a good predictor for these outliers that are not representative of the data

1) Transforming the output variable such that y_new = log(y)
2) Simply delete outlying instances from dataset by defining some range
    [y_min, y_max] and discarding all instances (x, y) outside of that range
```

Section 2 (Classification)

Question 5

In [194]:

```
def BER(predictions, y):
    # Implement following this logic or otherwise
    TP = sum([(p and l) for (p,l) in zip(predictions, y)])
    FP = sum([(p and not l) for (p,l) in zip(predictions, y)])
    TN = sum([(not p and not l) for (p,l) in zip(predictions, y)])
    FN = sum([(not p and l) for (p,l) in zip(predictions, y)])

    #TN, FP, FN, TP = confusion_matrix(y, predictions).ravel()

    FPR = FP/(FP + TN)
    FNR = FN/(FN + TP)


    return (FPR + FNR) * 0.5
```

In [195]:

```
def getTopIngredientsSansButter(d = dataset):
    ingredientDict = defaultdict(int)

    for datum in d:
        for ingredient in datum['ingredients']:
            ingredientDict[ingredient] += 1

    topIngredientsList = []
    for key in ingredientDict:
        if key == 'butter':
            continue
        topIngredientsList.append((key, ingredientDict[key]))

    topIngredientsList = sorted(topIngredientsList, key=lambda x: x[1], reverse=True)

    return topIngredientsList
```

In [196]:

```
def feat2(d, dict_size, mostPopularInd):
    features = []
    for datum in d:
        fIng = [0] * dict_size
        for i in datum['ingredients']:
            if i == 'butter':
                continue
            if i in mostPopularInd:
                fIng[mostPopularInd[i]] = 1
        features.append(fIng)
    return np.asarray(features, dtype=object)
```

In [197]:

```
def getButterLabel(d):
    y = []
    for datum in d:
        foundButter = False
        for ingredient in datum['ingredients']:
            if ingredient == 'butter':
                foundButter = True
                break;
        if foundButter:
            y.append(1)
        else:
            y.append(0)
    return np.asarray(y)
```

In [212]:

```
def experiment(reg = 1, dict_size = 50, report_train=False, report_valid=False, report_test
    # Hint: run an experiment with a particular regularization strength, and a particular d
    # extract features...
    topIngredientsList = getTopIngredientsSansButter()[:dict_size]

    ingredientSet = defaultdict(int)
    for i in range(len(topIngredientsList)):
        ingredientSet[topIngredientsList[i][0]] = i

    feat_train = feat2(train, dict_size, ingredientSet)
    y_train = getButterLabel(train)

    mod = linear_model.LogisticRegression(C=reg, class_weight='balanced', solver = 'lbfgs')
    mod.fit(feat_train, y_train)

    feat_valid = feat2(valid, dict_size, ingredientSet)
    y_valid = getButterLabel(valid)
    y_valid_pred = mod.predict(feat_valid)

    if report_train:
        y_train_pred = mod.predict(feat_train)
        print('Train BER: ', BER(y_train_pred, y_train))

    if report_valid:
        print('Validation BER: ', BER(y_valid_pred, y_valid))

    if report_test:
        feat_test = feat2(test, dict_size, ingredientSet)
        y_test = getButterLabel(test)
        y_test_pred = mod.predict(feat_test)

        print('Test BER: ', BER(y_test_pred, y_test))

    return mod, BER(y_valid_pred, y_valid)
```

In [205]:

```
experiment(report_test=True)
```

```
BER:  0.28930328282968243
```

Out[205]:

```
(LogisticRegression(C=1, class_weight='balanced'), 0.28951737136608635)
```

Question 6

In [217]:

```
def pipeline():
    BER = 1
    for reg in [0.01, 1, 100]:
        for dsize in [50, 100, 200]:
            # Example values, can pick any others...
            print('reg: {}, dsize: {}'.format(reg, dsize))
            _, temp_ber = experiment(reg=reg, dict_size=dsize, report_train=True, report_va

            if BER > temp_ber:
                c_opt = reg
                d_opt = dsize
            print()

    print('Optimum model: ')
    experiment(reg=c_opt, dict_size=d_opt, report_test=True)

    return c_opt, d_opt
```

In [218]:

```
pipeline()
```

```
reg: 0.01, dsize: 50
Train BER:  0.2901895036037755
Validation BER:  0.29033267701492

reg: 0.01, dsize: 100
Train BER:  0.26404572820163397
Validation BER:  0.26473200940198605

reg: 0.01, dsize: 200
Train BER:  0.24644410101153835
Validation BER:  0.2461171573321106

reg: 1, dsize: 50
Train BER:  0.2898832003476495
Validation BER:  0.28951737136608635

reg: 1, dsize: 100
Train BER:  0.2621956833656805
Validation BER:  0.2644673467541458

reg: 1, dsize: 200
Train BER:  0.24338886530751214
Validation BER:  0.2453328273275703

reg: 100, dsize: 50
Train BER:  0.289919044614668
Validation BER:  0.28951737136608635

reg: 100, dsize: 100
Train BER:  0.26220723251465694
Validation BER:  0.26441473732653636

reg: 100, dsize: 200
Train BER:  0.24332036088407077
Validation BER:  0.2451614076906133

Optimum model:
Test BER:  0.24564315945337192
```

Out[218]:

```
(100, 200)
```

Section 3 (Recommender Systems)

Question 8

In [247]:

```
# Utility data structures
ingsPerItem = defaultdict(set)
itemsPerIng = defaultdict(set)
recipeNameMapping = defaultdict(str)
```

In [251]:

```
for d in dataset:
    r = d['recipe_id']
    recipeNameMapping[d['recipe_id']] = d['name']
    for i in d['ingredients']:
        ingsPerItem[r].add(i)
        itemsPerIng[i].add(r)
```

In [221]:

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

In [229]:

```
def mostSimilarRecipes(recipe, N):
    similarities = []
    ingredients = ingsPerItem[recipe]
    for i2 in ingsPerItem:
        if i2 == recipe: continue
        sim = Jaccard(ingredients, ingsPerItem[i2])
        similarities.append((sim,i2))
    similarities.sort(key=lambda x: x[0], reverse=True)

    return similarities[:N]
```

In [256]:

```
recommendedRecipes = mostSimilarRecipes('06432987', 5)
```

In [257]:

```
recommendedRecipes
```

Out[257]:

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

In [258]:

```
for item in recommendedRecipes:
    print(recipeNameMapping[item[1]])
```

```
chez panisse zucchini fritters
red wine steak and mushrooms
warm mushroom and romaine salad
family favorite salad dressing
fresh tomato garden pasta
```

Question 9

In [230]:

```
def mostSimilarIngredients(ingredient, N):
    similarities = []
    recipes = itemsPerIng[ingredient]
    for ing in itemsPerIng:
        if ing == ingredient: continue
        sim = Jaccard(recipes, itemsPerIng[ing])
        similarities.append((sim, ing))
    similarities.sort(key=lambda x: x[0], reverse=True)

    return similarities[:N]
```

In [232]:

```
mostSimilarIngredients('butter', 5)
```

Out[232]:

```
[(0.22315311514274808, 'salt'),
 (0.2056685424969639, 'flour'),
 (0.19100394157199166, 'eggs'),
 (0.17882420717656095, 'sugar'),
 (0.17040052045973944, 'milk')]
```

Question 10

In [285]:

```
print("""
Remove all the ingredients that occur in more than
'coverage %' recipes. These are highly common ingredients and
thus do not hold the 'essence' of a recipe.

Then, define a similarity threshold till which we'll find similar
ingredients of each ingredient in the list provided. Then we
make an expanded list of all of these ingredients and find the
recipes that have ingredients most similar to these ingredients.
""")
```

```
Remove all the ingredients that occur in more than
'coverage %' recipes. These are highly common ingredients and
thus do not hold the 'essence' of a recipe.

Then, define a similarity threshold till which we'll find similar
ingredients of each ingredient in the list provided. Then we
make an expanded list of all of these ingredients and find the
recipes that have ingredients most similar to these ingredients.
```

In [262]:

```
topIngredients = getTopIngredients(dataset)
```

In [275]:

```
coverage = 0.1
recipeCount = len(dataset)
```

In [276]:

```
newTopIngredients = []
for datum in topIngredients:
    if (datum[1] > coverage * recipeCount):
        continue
    newTopIngredients.append(datum)
```

In [278]:

```
validIngredients = set([item[0] for item in newTopIngredients])
```

In [247]:

```
# Utility data structures
ingsPerItem = defaultdict(set)
itemsPerIng = defaultdict(set)
recipeNameMapping = defaultdict(str)
```

In [279]:

```
for d in dataset:
    r = d['recipe_id']
    recipeNameMapping[d['recipe_id']] = d['name']
    for i in d['ingredients']:
        if i in validIngredients:
            ingsPerItem[r].add(i)
            itemsPerIng[i].add(r)
```

In [280]:

```
def mostEssence_ialRecipes(ingredients, similarity_threshold, N): #Get it? I'll show myself
    similar_ingredients = []
    for ingredient in ingredients:
        temp = mostSimilarIngredients(ingredient, similarity_threshold)
        similar_ingredients.extend([item[1] for item in temp])
        similar_ingredients.append(ingredient)

    similar_ingredients = set(similar_ingredients)
    similarities = []
    for i2 in ingsPerItem:
        sim = Jaccard(similar_ingredients, ingsPerItem[i2])
        similarities.append((sim,i2))
    similarities.sort(key=lambda x: x[0], reverse=True)

    return similarities[:N]
```

In [281]:

```
recommendedRecipes = mostEssence_ialRecipes({'cinnamon', 'cherries', 'butterscotch', 'vodka
                                             similarity_threshold=2,
                                             N=5)
```

In [282]:

```
recommendedRecipes
```

Out[282]:

```
[(0.23076923076923078, '25952034'),
 (0.23076923076923078, '15152859'),
 (0.21428571428571427, '34731021'),
 (0.21428571428571427, '32115995'),
 (0.21428571428571427, '02927915')]
```

In [283]:

```
for item in recommendedRecipes:
    print(recipeNameMapping[item[1]])
```

```
atlas
sea breezes
sex on the moon
nola blue glacier martini
rangoon ruby cocktail
```