Midterm_stub

November 4, 2021

1 Section 1 (Regression)

[7]:
```
import gzip
from collections import defaultdict
import math
import scipy.optimize
import numpy
import string
import random
from sklearn import linear_model
```

[8]:
```
def parse(f):
    for l in gzip.open(f):
        yield eval(l)
```

[9]:
```
# Download data from below:
# https://cseweb.ucsd.edu/classes/fa21/cse258-b/files/
dataset = list(parse("trainRecipes.json.gz"))
```

[10]:
```
len(dataset)
```

[10]:
```
200000
```

[11]:
```
train = dataset[:150000]
valid = dataset[150000:175000]
test = dataset[175000:]
```

[12]:
```
dataset[0]
```

[12]:
```
{'name': 'sexy fried eggs for sunday brunch',
 'minutes': 10,
 'contributor_id': '14298494',
 'submitted': '2004-05-21',
 'steps': 'heat a ridged griddle pan\tlightly brush the tomato slices and bread
with some olive oil\tcook the tomato slices first , for at least 5 minutes\twhen
they are almost ready , toast the bread in the same pan until well bar-
marked\tin the meantime , pour a little olive oil into a small frying pan and
```

```
crack in the egg\tallow it to set for a minute or so and add the garlic and
chilli\tcook for a couple of minutes , spooning the hot oil over the egg until
cooked to your liking\tplace the griddled bread on a plate and quickly spoon the
tomatoes on top\tthrow the chives into the egg pan and splash in the balsamic
vinegar\tseason well , then slide the egg on to the tomatoes and drizzle the pan
juices on top\tserve immediately , with a good cup of tea !',
 'description': 'this is from silvana franco\'s book "family" which i love. i
made these for brunch yesterday and we loved them so much that we had them again
today!',
 'ingredients': ['plum tomato',
  'ciabatta',
  'olive oil',
  'egg',
  'garlic clove',
  'chili',
  'chives',
  'balsamic vinegar',
  'salt and pepper'],
 'recipe_id': '06432987'}
```

1.1 Question 1

[13]:
```
def feat1a(d):
    return [len(d['steps']), len(d['ingredients'])]

print(f"Feature vector for first training sample of 1a: \n{feat1a(dataset[0])}")
```

```
Feature vector for first training sample of 1a:
[743, 9]
```

[14]:
```
maxYear = -math.inf
minYear = math.inf

for elem in dataset:
    year = int(elem['submitted'][:4])
    if year > maxYear: maxYear = year
    if year < minYear: minYear = year

print(f"Year of newest submission: {maxYear}\nYear of oldest submission:␣
 ↪{minYear}")
```

```
Year of newest submission: 2018
Year of oldest submission: 1999
```

[15]:
```
def feat1b(d):
    year = int(d['submitted'][:4])
    numYears = maxYear - minYear
```

```
    yearFeat = [0]*numYears
    if year != maxYear:
        yearFeat[maxYear-year-1] = 1

    month = int(d['submitted'][5:7])
    monthFeat = [0]*11
    monthFeat[month-2] = 1
    return yearFeat + monthFeat

print(f"Feature vector for first training sample of 1b: \n{feat1b(dataset[0])}")
```

```
Feature vector for first training sample of 1b:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0]
```

[16]:
```
ingredientCount = {}
for elem in dataset:
    for ingredient in elem['ingredients']:
        if ingredient in ingredientCount:
            ingredientCount[ingredient] += 1
        else:
            ingredientCount[ingredient] = 1

ingredientCount = dict(sorted(ingredientCount.items(), key=lambda item:␣
 ↪item[1], reverse=True))

topFiftyIngredients = []
for key in ingredientCount.keys():
    if len(topFiftyIngredients) >= 50:
        break
    else:
        topFiftyIngredients.append(key)
```

[17]:
```
def feat1c(d):
    feat = [0]*50
    ingredients = d['ingredients']
    index = 0
    for elem in topFiftyIngredients:
        if elem in ingredients:
            feat[index] = 1
        index += 1
    return feat

print(f"Feature vector for first training sample of 1c: \n{feat1c(dataset[0])}")
```

```
Feature vector for first training sample of 1c:
```

```
[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, 0]
```

[18]:
```
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
    feature = [1]
    if a: feature += feat1a(d)
    if b: feature += feat1b(d)
    if c: feature += feat1c(d)
    return feature
```

[19]:
```
def MSE(y, ypred):
    # Can use library if you prefer
    differences = [(x-y)**2 for x,y in zip(y,ypred)]
    return sum(differences) / len(differences)
```

[20]:
```
# Splitting dataset
# Dataset not sorted after date, thus splitting it in the trivial way
dataTrain = dataset[:len(dataset)*3//4]
dataValidation = dataset[len(dataset)*3//4:len(dataset)*7//8]
dataTest = dataset[len(dataset)*7//8:]
```

[21]:
```
def train_model(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
    XTrain = [feat(data, a, b, c) for data in dataTrain]
    yTrain = [data['minutes'] for data in dataTrain]
    mod.fit(XTrain, yTrain)
```

[22]:
```
model1a = linear_model.LinearRegression()
train_model(model1a, True, False, False)

X1aTest = [feat(data, True, False, False) for data in dataTest]
y1aTest = [d['minutes'] for d in dataTest]
y1aPred = model1a.predict(X1aTest)

print(f"MSE for 1a: {MSE(y1aTest, y1aPred)}")
```

```
MSE for 1a: 6169.549296366476
```

[23]:
```
model1b = linear_model.LinearRegression()
y1bPred = train_model(model1b, False, True, False)

X1bTest = [feat(data, False, True, False) for data in dataTest]
y1bTest = [d['minutes'] for d in dataTest]
```

```
y1bPred = model1b.predict(X1bTest)

print(f"MSE for 1b: {MSE(y1bTest, y1bPred)}")
```

```
MSE for 1b: 6396.644907898458
```

[24]:
```
model1c = linear_model.LinearRegression()
y1cPred = train_model(model1c, False, False, True)

X1cTest = [feat(data, False, False, True) for data in dataTest]
y1cTest = [d['minutes'] for d in dataTest]
y1cPred = model1c.predict(X1cTest)

print(f"MSE for 1c: {MSE(y1cTest, y1cPred)}")
```

```
MSE for 1c: 6000.948439855985
```

1.2 Question 2

[25]:
```
modelAll = linear_model.LinearRegression()
yAllPred = train_model(modelAll, True, True, True)

XAllTest = [feat(data, True, True, True) for data in dataTest]
yAllTest = [d['minutes'] for d in dataTest]
yAllPred = modelAll.predict(XAllTest)

print(f"MSE for model with all features: {MSE(yAllTest, yAllPred)}")
```

```
MSE for model with all features: 5861.087768668749
```

[26]:
```
model1bc = linear_model.LinearRegression()
y1bcPred = train_model(model1bc, False, True, True)

X1bcTest = [feat(data, False, True, True) for data in dataTest]
y1bcTest = [d['minutes'] for d in dataTest]
y1bcPred = model1bc.predict(X1bcTest)

print(f"MSE for model without 1a: {MSE(y1bcTest, y1bcPred)}")
```

```
MSE for model without 1a: 5992.513432436371
```

[27]:
```
model1ac = linear_model.LinearRegression()
y1acPred = train_model(model1ac, True, False, True)

X1acTest = [feat(data, True, False, True) for data in dataTest]
y1acTest = [d['minutes'] for d in dataTest]
y1acPred = model1ac.predict(X1acTest)
```

```
print(f"MSE for model without 1b: {MSE(y1acTest, y1acPred)}")
```

```
MSE for model without 1b: 5870.115061656081
```

[28]:
```
model1ab = linear_model.LinearRegression()
y1abPred = train_model(model1ab, True, True, False)

X1abTest = [feat(data, True, True, False) for data in dataTest]
y1abTest = [d['minutes'] for d in dataTest]
y1abPred = model1ab.predict(X1abTest)

print(f"MSE for model without 1c: {MSE(y1abTest, y1abPred)}")
```

```
MSE for model without 1c: 6157.511552782132
```

1.2.1 Reasoning on result

By comparing the MSEs when excluding one of the features at a time, it looks like the most important feature was the one implemented in task 1c, since the exclution of this lead to the most significant increase in the MSE. The exclution of the features from task 1a also led to a quite significant increase in the MSE, which makes it the second most important feature. The exclution of the features from task 1b only lead to a slight increase in the MSE. However, with the amount of data we have it seems like the best predictor is the one when all features are included.

1.3 Question 4

The problem with this is that, since the error is squared, hence Mean SQUARE Error, the prediction errors on the outliers with long cooking time will be further amplified by the square, and thus be dominant on the MSE. To optimize the MSE, the predictor will thus focus more on the few recipes with long cooking times rather than the big amount of recipes with short cook times, whoch is an obvious drawback.

There are several ways to design a better predictor in such a dataset. The simplest way is to just remove the outliers from the dataset. This however does not give us any practical means for prediction of outliers. Another way is to tranform the output variable to a variable which downscales the varibles with greater magnitude more than the other, such as the logarithmic value. To find this appropriate transformation may be hard, and may require careful tuning. A third way may be to reframe the problem as a classification problem, e.g. predict whether the cooking time is above 20 minutes. This does however give os a very coarse perdiction. A fourth and last way may be to use an objective that are not as sensitive to outliers as the MSE. One oportunity is to use MAE.

2 Section 2 (Classification)

2.1 Question 5

[29]:
```
topIngredients = []
for key in ingredientCount.keys():
    if key!="butter":
        topIngredients.append(key)
```

[30]:
```
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)])

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

    BER = 0.5*(FPR + FNR)
    return BER
```

[31]:
```
def feat2(d, N, mostPopularInd):
    feat = [0]*N
    ingredients = d['ingredients']
    index = 0
    for elem in mostPopularInd[:N]:
        if elem in ingredients:
            feat[index] = 1
        index += 1
    return feat
```

[33]:
```
def containButter(ingredients):
    if 'butter' in ingredients:
        return 1
    return 0

def experiment(reg = 1, dict_size = 50):
    # Hint: run an experiment with a particular regularization strength, and a␣
 ↪particular one-hot encoding size
    # extract features...
    XTrain = [feat2(d, dict_size, topIngredients) for d in dataTrain]
    yTrain = [containButter(d['ingredients']) for d in dataTrain]

    XValid = [feat2(d, dict_size, topIngredients) for d in dataValidation]
    yValid = [containButter(d['ingredients']) for d in dataValidation]
```

```
    # (etc.)
    mod = linear_model.LogisticRegression(C=reg, class_weight='balanced',␣
↪solver = 'lbfgs')
    mod.fit(XTrain, yTrain)

    XTest = [feat2(d, dict_size, topIngredients) for d in dataTest]
    yTest = [containButter(d['ingredients']) for d in dataTest]

    yPred = mod.predict(XTest)

    sum = 0
    for y_, yPred_ in zip(yTest, yPred):
        if y_ == yPred_:
            sum += 1

    accuracy = sum/len(yPred)
    print("-----------TEST PERFORMANCE-----------")
    print(f"BER: {BER(yPred, yTest)} \nAccuracy: {accuracy}")


experiment()
```

2.2 Question 6

[35]:
```
def containButter(ingredients):
    if 'butter' in ingredients:
        return 1
    return 0

def experiment(reg = 1, dict_size = 50):
    # Hint: run an experiment with a particular regularization strength, and a␣
↪particular one-hot encoding size
    # extract features...
    XTrain = [feat2(d, dict_size, topIngredients) for d in dataTrain]
    yTrain = [containButter(d['ingredients']) for d in dataTrain]

    XValid = [feat2(d, dict_size, topIngredients) for d in dataValidation]
    yValid = [containButter(d['ingredients']) for d in dataValidation]


    # (etc.)
    mod = linear_model.LogisticRegression(C=reg, class_weight='balanced',␣
↪solver = 'lbfgs')
    mod.fit(XTrain, yTrain)
```

```
    XTest = [feat2(d, dict_size, topIngredients) for d in dataTest]
    yTest = [containButter(d['ingredients']) for d in dataTest]

    yTrainPred = mod.predict(XTrain)
    yValidPred = mod.predict(XValid)


    print("-----------TRAIN PERFORMANCE-----------")
    print(f"BER: {BER(yTrainPred, yTrain)}")
    print("-----------VALIDATION PERFORMANCE-----------")
    print(f"BER: {BER(yValidPred, yValid)}")



def pipeline():
    for C in [0.01, 1, 100]:
        for dsize in [50, 100, 500]:
            print(f"Result with regularization constant = {C}, and amount of␣
 ↪ingredients N = {dsize}")
            experiment(C, dsize)

pipeline()
```

```
Result with regularization constant = 0.01, and amount of ingredients N = 50
-----------TRAIN PERFORMANCE-----------
BER: 0.2901895036037755
-----------VALIDATION PERFORMANCE-----------
BER: 0.29033267701492
Result with regularization constant = 0.01, and amount of ingredients N = 100
-----------TRAIN PERFORMANCE-----------
BER: 0.26404572820163397
-----------VALIDATION PERFORMANCE-----------
BER: 0.26473200940198605
Result with regularization constant = 0.01, and amount of ingredients N = 500
-----------TRAIN PERFORMANCE-----------
BER: 0.22851785954772086
-----------VALIDATION PERFORMANCE-----------
BER: 0.2306062614911914
Result with regularization constant = 1, and amount of ingredients N = 50
-----------TRAIN PERFORMANCE-----------
BER: 0.2898832003476495
-----------VALIDATION PERFORMANCE-----------
BER: 0.28951737136608635
Result with regularization constant = 1, and amount of ingredients N = 100
-----------TRAIN PERFORMANCE-----------
BER: 0.2621956833656805
-----------VALIDATION PERFORMANCE-----------
```

```
BER: 0.2644673467541458
Result with regularization constant = 1, and amount of ingredients N = 500
----------TRAIN PERFORMANCE----------
BER: 0.2233752436848266
----------VALIDATION PERFORMANCE----------
BER: 0.22508648596557473
Result with regularization constant = 100, and amount of ingredients N = 50
----------TRAIN PERFORMANCE----------
BER: 0.289919044614668
----------VALIDATION PERFORMANCE----------
BER: 0.28951737136608635
Result with regularization constant = 100, and amount of ingredients N = 100
----------TRAIN PERFORMANCE----------
BER: 0.26220723251465694
----------VALIDATION PERFORMANCE----------
BER: 0.26441473732653636
Result with regularization constant = 100, and amount of ingredients N = 500
----------TRAIN PERFORMANCE----------
BER: 0.2233087265518917
----------VALIDATION PERFORMANCE----------
BER: 0.2252307228942743
```

[36]:
```python
def experiment(reg = 1, dict_size = 50):
    # Hint: run an experiment with a particular regularization strength, and a␣
 ↪particular one-hot encoding size
    # extract features...
    XTrain = [feat2(d, dict_size, topIngredients) for d in dataTrain]
    yTrain = [containButter(d['ingredients']) for d in dataTrain]

    XValid = [feat2(d, dict_size, topIngredients) for d in dataValidation]
    yValid = [containButter(d['ingredients']) for d in dataValidation]



    # (etc.)
    mod = linear_model.LogisticRegression(C=reg, class_weight='balanced',␣
 ↪solver = 'lbfgs')
    mod.fit(XTrain, yTrain)

    XTest = [feat2(d, dict_size, topIngredients) for d in dataTest]
    yTest = [containButter(d['ingredients']) for d in dataTest]

    yPred = mod.predict(XTest)

    sum = 0
    for y_, yPred_ in zip(yTest, yPred):
        if y_ == yPred_:
            sum += 1
```

```
    accuracy = sum/len(yPred)
    print("-----------TEST PERFORMANCE-----------")
    print(f"BER: {BER(yPred, yTest)} \nAccuracy: {accuracy}")

print(f"Test performance for selected model(Regularization constant = {1}, N = {500}):")
experiment(1, 500)
```

```
Test performance for selected model(Regularization constant = 1, N = 500):
-----------TEST PERFORMANCE-----------
BER: 0.22551312999705395
Accuracy: 0.77828
```

3 Section 3 (Recommender Systems)

3.1 Question 8

[37]:
```
# Utility data structures
ingsPerItem = defaultdict(set)
itemsPerIng = defaultdict(set)
```

[38]:
```
for d in dataset:
    r = d['recipe_id']
    for i in d['ingredients']:
        ingsPerItem[r].add(i)
        itemsPerIng[i].add(r)
```

[39]:
```
def Jaccard(s1, s2):
    numerator = len(s1.intersection(s2))
    denominator = len(s1.union(s2))
    if(denominator == 0):
        return 0
    return numerator/denominator
```

[40]:
```
def mostSimilar8(item, N):
    similarities = []
    ingredients = ingsPerItem[item]
    for i in ingsPerItem:
        if (i==item): continue
        similarity = Jaccard(ingredients, ingsPerItem[i])
        similarities.append((similarity, i))
    similarities.sort(reverse=True)
    return similarities[:N]

recipe_id = dataset[0]['recipe_id']
fiveBest = mostSimilar8(recipe_id, 5)
```

```
print('\t Recipe ID \t Similarity')
for index, elem in enumerate(fiveBest):
    print(f"{index+1} \t {elem[1]} \t {elem[0]}")
```

```
         Recipe ID       Similarity
1         68523854        0.4166666666666667
2         12679596        0.38461538461538464
3         79675099        0.36363636363636365
4         56301588        0.36363636363636365
5         87359281        0.35714285714285715
```

[41]:
```
def mostSimilar9(ing, N):
    similarities = []
    items = itemsPerIng[ing]
    for i in itemsPerIng:
        if (i==ing): continue
        similarity = Jaccard(items, itemsPerIng[i])
        similarities.append((similarity, i))
    similarities.sort(reverse=True)
    return similarities[:N]

ing = 'butter'
fiveBest = mostSimilar9(ing, 5)

print('\t Ingredient \t Similarity')
for index, elem in enumerate(fiveBest):
    print(f"{index+1} \t {elem[1]} \t\t {elem[0]}")
```

```
         Ingredient      Similarity
1         salt            0.22315311514274808
2         flour           0.2056685424969639
3         eggs            0.19100394157199166
4         sugar           0.17882420717656095
5         milk            0.17040052045973944
```

3.2 Question 10

There are obviously tons of different ways of solving this task. While the trivial solution obviously might be a good solution since you get the most similar recipe to the excact ingredients the user had in mind, it may as the task says be too inflexible.

For me, the most critical part when I chose what recipe I want to cook is the time it takes. My approach is therefor to start by predicting the time it will take to make the meal based on the ingredients, and nattow down the possible recommendations to recipes taking +-25% of that time. After that, I will find the most similar ingredients to the ingredients in the original set, try change them one at a time, and find the new most similar recipes with the one different ingredient. After that I will sort all the found recipes by the similarity, and recommend the N with highest similarity.

[42]:
```
topIngredients = []
for key in ingredientCount.keys():
    if key!="butter":
        topIngredients.append(key)
```

[43]:
```
# Start by predicting how long time the cooking will take based on the␣
↪ingredient list
def feat(ingredients):
    featTopIngredients = [0]*500
    index = 0
    for elem in topIngredients[500]:
        if elem in ingredients:
            feat[index] = 1
        index += 1
    return [1] + [len(ingredients)] + featTopIngredients
```

[44]:
```
def predictTime(ingredients):
    # Training model to predict time based on ingredients
    X = [feat(d['ingredients']) for d in dataset]
    y = [d['minutes'] for d in dataset]

    model = linear_model.LinearRegression()
    model.fit(X, y)

    return model.predict(ingredients)
```

[58]:
```
def nRecomendations(ingredientSet, N):
    ingredientList = list(ingredientSet)
    predictedTime = predictTime([feat(ingredientList)])
    maxTime = predictedTime*1.25
    minTime = predictedTime*0.75

    print(f"Predicted time: {predictedTime}")

    possibleRecipes = []
    for d in dataset:
        if d['minutes'] < maxTime and d['minutes'] > minTime:
            possibleRecipes.append(d)

    ingsPerItem = defaultdict(set)
    for d in possibleRecipes:
        r = d['recipe_id']
        for i in d['ingredients']:
            ingsPerItem[r].add(i)

    mostSimilarIng = []
    for elem in ingredientSet:
```

```
        mostSimilarIng += mostSimilar9(elem, 1)

    bestRecommendations = []
    bestRecommendations += mostSimilarByIng(ingredientSet, ingsPerItem,  N)
    for i in range(len(ingredientSet)):
        similarIngredientList = ingredientList
        similarIngredientList[i] = mostSimilarIng[i]
        for elem in mostSimilarByIng(set(similarIngredientList), ingsPerItem,␣
↪N):
            if elem not in bestRecommendations:
                bestRecommendations.append(elem)

    bestRecommendations.sort(reverse=True)
    for i in range(N):
        for d in dataset:
            if d['recipe_id'] == bestRecommendations[i][1]:
                print(f"Recommendation #{i}: \n{d}")
    return bestRecommendations[:N]
```

[59]:
```
def mostSimilarByIng(ingredients, ingsPerItem, N):
    similarities = []
    for i in ingsPerItem:
        similarity = Jaccard(ingredients, ingsPerItem[i])
        similarities.append((similarity, i))
    similarities.sort(reverse=True)
    return similarities[:N]
```

[60]:
```
ingredients = {'cinnamon', 'cherries', 'butterscotch', 'vodka'}
nRecomendations(ingredients, 3)
```

```
Predicted time: [41.74373487]
Recommendation #0:
{'name': 'morello cherry sauce', 'minutes': 40, 'contributor_id': '22505516',
'submitted': '2009-09-17', 'steps': 'dissolve sugar in the water over a low heat
, add the cherries , cinnamon and lemon zest and bring to the boil , reduce heat
, and simmer until very syrupy\tremove cherry mix from the heat when syrupy and
thick enough\tadd lemon juice and brandy\tallow mix to cool , and keep until
ready to serve\tserve over your favourite icecream', 'description': 'found this
recipe from the cook and the chef. this sauce is great served over vanilla or a
rich chocolate ice cream. i think this would go well over pancakes also. i have
added brandy to the original recipe, so you can leave brandy out if prefered.',
'ingredients': ['water', 'sugar', 'cherries', 'lemon, zest of', 'cinnamon',
'lemon juice', 'brandy'], 'recipe_id': '93220713'}
Recommendation #1:
{'name': 'sour cherry pie', 'minutes': 45, 'contributor_id': '43223120',
'submitted': '2001-07-25', 'steps': 'mix cherries with sugar , flour , salt ,
cinnamon and tapioca\tmight look kind of funny , but that is ok\tpour into
```

```
unbaked pie crust\tdot with butter\tbake at 375 for 35 minutes', 'description':
'', 'ingredients': ['cherries', 'sugar', 'flour', 'salt', 'cinnamon', 'minute
tapioca', 'butter'], 'recipe_id': '15446036'}
Recommendation #2:
{'name': 'summertime cherry peach cobbler', 'minutes': 50, 'contributor_id':
'62509453', 'submitted': '2009-07-10', 'steps': 'put oven rack in middle
position and preheat oven to 425\tbutter a 13x9x2-inch glass or ceramic baking
dish\twhisk together cornstarch , cinnamon , sugar , and peaches in a large bowl
and toss to combine\tadd the more delicate cherries last and give a gentle
stir\ttransfer fruit mixture into the baking dish and bake for ten to fifteen
minutes until bubbling\twhile fruit bakes , whisk the flour , baking powder ,
and salt together\tadd buttermilk and melted butter mix just until a dough
forms\tdrop batter onto fruit and sprinkle with remaining tablespoon of
sugar\tbake until top is golden brown , about 20 - 25 minutes\tserve warm with
vanilla ice cream , if desired', 'description': "two of summer's tastiest fruits
combine into this sweet, crowd-pleasing cobbler. recipe adapted from the central
market in fort worth, tx.", 'ingredients': ['cornstarch', 'sugar', 'cherries',
'peaches', 'all-purpose flour', 'baking powder', 'unsalted butter',
'buttermilk', 'cinnamon'], 'recipe_id': '00722531'}
```

[60]:
```
[(0.2222222222222222, '93220713'),
 (0.2222222222222222, '15446036'),
 (0.18181818181818182, '00722531')]
```

[61]:
```
ingredients2 = {'chili', 'chicken', 'garlic', 'onion'}
nRecomendations(ingredients2, 5)
```

```
Predicted time: [41.74373487]
Recommendation #0:
{'name': 'roast potatoes gauci style', 'minutes': 50, 'contributor_id':
'45866524', 'submitted': '2006-03-23', 'steps': "cut the potato into 1 inch
cubes & place in a plastic bag & put into the microwave for 2 mins\tput in the
onion , garlic , chili , salt , pepper & wholegrain mustard along with the olive
oil\tshake it all up until potato is well coated\tempty it onto an oven tray &
put in the oven at about 200degc for about 40 minutes\ttest with a fork when you
think they're done", 'description': "threw this together a couple of weeks ago
and it's great! wholegrain mustard makes this a flavoursome side dish.",
'ingredients': ['potatoes', 'onion', 'garlic', 'chili', 'coarse grain mustard',
'salt & pepper', 'olive oil'], 'recipe_id': '34107311'}
Recommendation #1:
{'name': 'baked mexican rice vegetarian', 'minutes': 50, 'contributor_id':
'01030700', 'submitted': '2012-07-26', 'steps': 'place the oil in a saucepan and
cook the onion until translucent\tadd the garlic , cumin and chilli and fry over
a medium heat until fragrant\tadd the rice , stock and paste , stir to
combine\twhen almost to the boil , transfer to a 2 litre capacity oven proof
dish with a lid\tcover and bake at 180oc for 35-40 minutes\tremove from the oven
and let sit for 5 minutes before serving', 'description': "a tomato based
mexican rice. can be made the day before and re-heated (easy for entertaining).
```

```
this resembles our favourite mexican restaurant's rice dish that i love. if you
double, add extra cooking time.", 'ingredients': ['oil', 'onion', 'garlic',
'ground cumin', 'chili powder', 'white rice', 'chicken', 'tomato paste'],
'recipe_id': '38703750'}
Recommendation #2:
{'name': 'south african curry', 'minutes': 50, 'contributor_id': '38120744',
'submitted': '2009-10-04', 'steps': "saute onion in a little oil til soft\tadd
curry powder and turmeric\tsaute for 30 sec\tdon't let burn !\tadd chopped
tomatoes and cook down to a paste\tadd garlic and ginger\tadd chicken and
potatoes\tsalt to taste and let simmer until chicken and potatoes are done",
'description': 'got this from a friend of mine in south africa. i usually use
chicken breasts cut into 1 in. pieces as it cuts down on cooking time.',
'ingredients': ['onion', 'curry powder', 'turmeric', 'tomatoes', 'chicken',
'garlic', 'ginger', 'potato', 'salt'], 'recipe_id': '91198504'}
Recommendation #3:
{'name': 's more chicken', 'minutes': 35, 'contributor_id': '66262951',
'submitted': '2004-05-25', 'steps': 'brown meaty chicken pieces in butter with
sliced onion and minced garlic\tplace browned chicken and onions in pressure
cooker\tdump remaining ingredients over chicken\theat on high until pressure
cooker top starts to rock , then reduce heat to medium-high\tcook for 15
minutes\tserve over rice with sauce', 'description': 'this chicken recipe is
easy and delicious! yum! yum! we love it so much i make it often! i believe this
may be indonesian, since my boyfriend got the recipe from his dutch-indonesian
mom.', 'ingredients': ['chicken', 'butter', 'onion', 'garlic', 'water', 'soy
sauce', 'salt', 'pepper', 'nutmeg'], 'recipe_id': '46790934'}
Recommendation #4:
{'name': 'frito chili pie', 'minutes': 45, 'contributor_id': '57791277',
'submitted': '2005-04-17', 'steps': 'preheat oven to 350 degrees\tspray a square
casserole dish with nonstick cooking spray\tline the bottom of casserole dish
with tamales\ttop tamales with a thin layer of corn chips\tspread 1 can of chili
on top of corn chips\ttop chili with 1 cup cheese and 1 / 2 cup of chopped
onion\tadd another layer of corn chips\tcover with remaining chili\ttop chili
with remaining cheese and onions\tbake 30 minutes', 'description': "a great
quick meal that was a favorite in my family when i was growing up and that my
kids love too! recipe has been adjusted to 2 cans of chili as suggested by
reviewers. i normally use one and don't use the entire bag of fritos. you can
adjust the chili to frito ratio to your taste. more chili is always good.",
'ingredients': ['fritos corn chips', 'chili', 'tamales', 'cheddar cheese',
'onion'], 'recipe_id': '96290950'}
```

[61]:
```
[(0.375, '34107311'),
 (0.3333333333333333, '38703750'),
 (0.3, '91198504'),
 (0.3, '46790934'),
 (0.2857142857142857, '96290950')]
```

[62]:
```
ingredients3 = {'tomato', 'turkey', 'rice'}
nRecomendations(ingredients2, 2)
```

```
Predicted time: [41.74373487]
Recommendation #0:
{'name': 'roast potatoes gauci style', 'minutes': 50, 'contributor_id':
'45866524', 'submitted': '2006-03-23', 'steps': "cut the potato into 1 inch
cubes & place in a plastic bag & put into the microwave for 2 mins\tput in the
onion , garlic , chili , salt , pepper & wholegrain mustard along with the olive
oil\tshake it all up until potato is well coated\tempty it onto an oven tray &
put in the oven at about 200degc for about 40 minutes\ttest with a fork when you
think they're done", 'description': "threw this together a couple of weeks ago
and it's great! wholegrain mustard makes this a flavoursome side dish.",
'ingredients': ['potatoes', 'onion', 'garlic', 'chili', 'coarse grain mustard',
'salt & pepper', 'olive oil'], 'recipe_id': '34107311'}
Recommendation #1:
{'name': 'baked mexican rice vegetarian', 'minutes': 50, 'contributor_id':
'01030700', 'submitted': '2012-07-26', 'steps': 'place the oil in a saucepan and
cook the onion until translucent\tadd the garlic , cumin and chilli and fry over
a medium heat until fragrant\tadd the rice , stock and paste , stir to
combine\twhen almost to the boil , transfer to a 2 litre capacity oven proof
dish with a lid\tcover and bake at 180oc for 35-40 minutes\tremove from the oven
and let sit for 5 minutes before serving', 'description': "a tomato based
mexican rice. can be made the day before and re-heated (easy for entertaining).
this resembles our favourite mexican restaurant's rice dish that i love. if you
double, add extra cooking time.", 'ingredients': ['oil', 'onion', 'garlic',
'ground cumin', 'chili powder', 'white rice', 'chicken', 'tomato paste'],
'recipe_id': '38703750'}
```

[62]:
```
[(0.375, '34107311'), (0.3333333333333333, '38703750')]
```