CSE 158, Fall 2019: Homework 3

Name: CUI, HONGJIAN **PID:** U08398995

Tasks (Read prediction):

Question 1:

In [1]:
```
import matplotlib.pyplot as plt
import csv
import numpy as np
import random
from collections import defaultdict
```

In [2]:
```
path = "/home/cui/Projects/PycharmProjects/CSE-158/data/train_Interacti
file = open(path, 'rt')
```

In [3]:
```
header = file.readline()
header = header.strip().split(',')
```

In [4]:
```
dataset = []
```

In [5]:
```
for line in file:
    fields = line.strip().split(',')
    d = dict(zip(header, fields))
    d['rating'] = int(d['rating'])
    dataset.append(d)
```

In [6]:
```
dataset[0]
```

Out[6]:
```
{'userID': 'u79354815', 'bookID': 'b14275065', 'rating': 4}
```

In [7]:
```
X = [[d['userID'], d['bookID'], 1] for d in dataset]
```

In [8]:
```
split = 190000
X_training = X[:split]
X_valid = X[split:]
```

In [9]:
```
X_valid[0]
```

Out[9]:
```
['u35176258', 'b30592470', 1]
```

In [10]:
```
usersPerBook = defaultdict(set)
booksPerUser = defaultdict(set)
bookSets = set()
```

In [11]:
```
for d in dataset:
    user, book = d['userID'], d['bookID']
    usersPerBook[book].add(user)
    booksPerUser[user].add(book)
    bookSets.add(book)
```

In [12]:
```
len(bookSets)
```

Out[12]: 7170

In [13]:
```
len(booksPerUser)
```

Out[13]: 11357

In [14]:
```
valid_user = [d[0] for d in X_valid]
```

In [15]:
```
for user in valid_user:
    booksNotReadSet = bookSets - booksPerUser.get(user)
    book = random.choice(list(booksNotReadSet))
    X_valid.append([user, book, 0])
```

In [16]:
```
len(X_valid)
```

Out[16]: 20000

In [17]:
```
X_valid[0]
```

Out[17]: ['u35176258', 'b30592470', 1]

In [18]:
```
# random.shuffle(X_valid)
```

In [19]:
```
X_valid[0]
```

Out[19]: ['u35176258', 'b30592470', 1]

In [20]:
```
y_valid = [d[2] for d in X_valid]
X_valid = [[d[0], d[1]] for d in X_valid]
```

In [21]:
```
# Baseline model
bookCount = defaultdict(int)
totalRead = 0

for d in dataset:
    bookCount[d['bookID']] += 1
    totalRead += 1

mostPopular = [(bookCount[x], x) for x in bookCount]
mostPopular.sort()
mostPopular.reverse()

return1 = set()
count = 0
for ic, i in mostPopular:
    count += ic
    return1.add(i)
    if count > totalRead / 2: break

prediction = []
for l in X_valid:
    if l[1] in return1:
        prediction.append(1)
    else:
        prediction.append(0)
```

In [22]:
```
correctPredictionValid = np.array(prediction) == np.array(y_valid)
```

In [23]:
```
sum(correctPredictionValid) / len(correctPredictionValid)
```

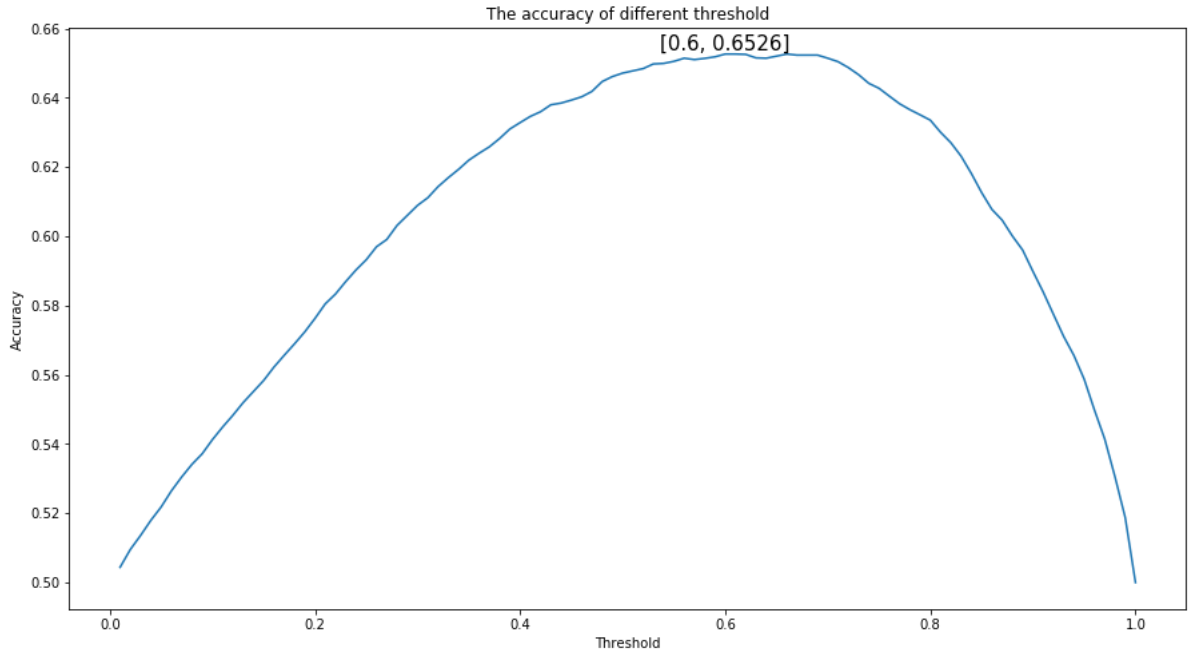
Out[23]:
```
0.6471
```

Question 2:

In [24]:

```
# Baseline model - different threshold
bookCount = defaultdict(int)
totalRead = 0

for d in dataset:
    bookCount[d['bookID']] += 1
    totalRead += 1

mostPopular = [(bookCount[x], x) for x in bookCount]
mostPopular.sort()
mostPopular.reverse()

return1 = set()
count = 0
threshold = 0.01
accuracyList = []

for n in range(100):
    for ic, i in mostPopular:
        count += ic
        return1.add(i)
        if count > totalRead * threshold * (n + 1): break

    prediction = []
    for l in X_valid:
        if l[1] in return1:
            prediction.append(1)
        else:
            prediction.append(0)

    correctPredictionValid = np.array(prediction) == np.array(y_valid)
    accuracy = sum(correctPredictionValid) / len(correctPredictionValid
    accuracyList.append([round(threshold * (n + 1), 2), accuracy])

    return1 = set()
    count = 0
```

In [25]:
```
x_plt = [d[0] for d in accuracyList]
y_plt = [d[1] for d in accuracyList]

plt.figure(figsize=(15,8))

plt.plot(x_plt, y_plt)
y_max = max(y_plt)
x_max = x_plt[y_plt.index(y_max)]
plt.text(x_max, y_max, [x_max, y_max],ha='center', va='bottom', fontsize

plt.title("The accuracy of different threshold")
plt.xlabel("Threshold")
plt.ylabel("Accuracy")
plt.show()
```



Question 3:

In [26]:
```
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    return numer / denom
```

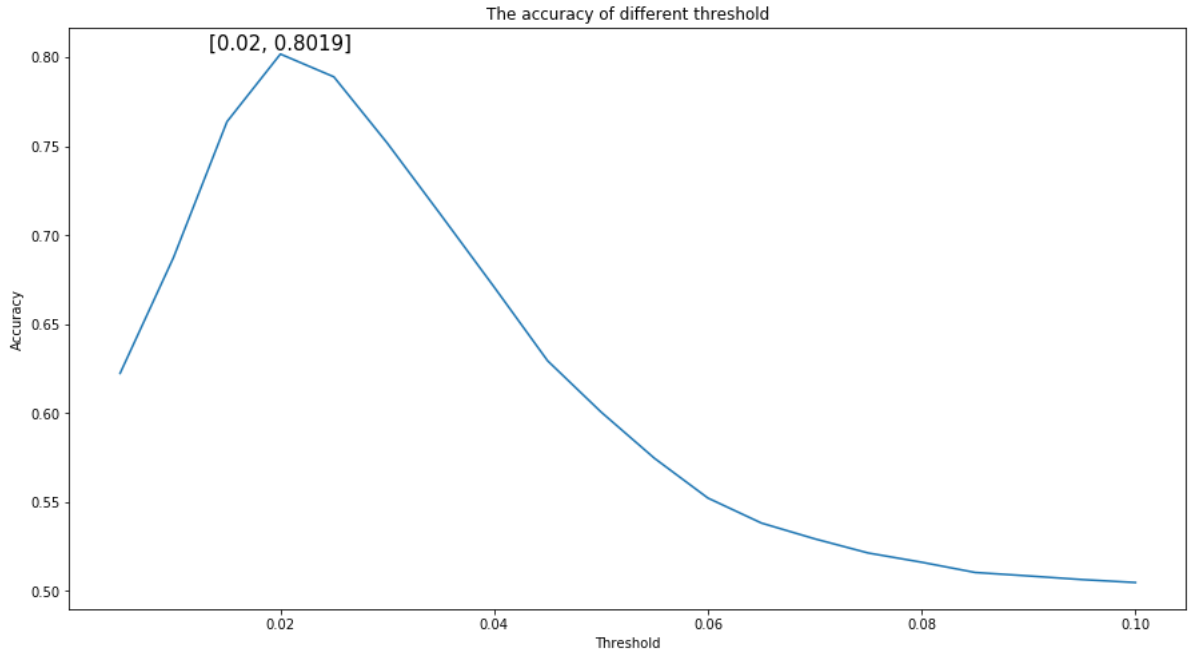
In [27]:
```
def mostSimilar(u, b):
    similarities = []
    books = booksPerUser[u]
    for b2 in books:
        if b2 == b:continue
        sim = Jaccard(usersPerBook[b], usersPerBook[b2])
        similarities.append(sim)
    similarities.sort(reverse=True)
    return similarities[:10]
```

In [28]:
```
mostSimilar(X_valid[0][0], X_valid[0][1])
```

Out[28]:
```
[0.03418803418803419,
 0.01904761904761905,
 0.016129032258064516,
 0.011904761904761904,
 0.00980392156862745,
 0.008695652173913044,
 0.007751937984496124,
 0.006896551724137931,
 0.006802721088435374,
 0.006578947368421052]
```

In [29]:
```
# Baseline model - Jaccard similarity
accuracyList = []
threshold = 0.005

for n in range(0, 20):
    mostSimilarList = []
    prediction = []

    for d in X_valid:
        mostSimilarList = mostSimilar(d[0], d[1])

        if mostSimilarList[0] >= threshold * (n + 1):
            prediction.append(1)
        else:
            prediction.append(0)

    correctPredictionValid = np.array(prediction) == np.array(y_valid)
    accuracy = sum(correctPredictionValid) / len(correctPredictionValid
    accuracyList.append([threshold * (n + 1), accuracy])
```

In [30]:
```
x_plt = [d[0] for d in accuracyList]
y_plt = [d[1] for d in accuracyList]

plt.figure(figsize=(15,8))

plt.plot(x_plt, y_plt)
y_max = max(y_plt)
x_max = x_plt[y_plt.index(y_max)]
plt.text(x_max, y_max, [x_max, y_max],ha='center', va='bottom', fontsize

plt.title("The accuracy of different threshold")
plt.xlabel("Threshold")
plt.ylabel("Accuracy")
plt.show()
```



Question 4:

In [31]:
```
bookCount = defaultdict(int)
totalRead = 0

for d in dataset:
    bookCount[d['bookID']] += 1
    totalRead += 1

mostPopular = [(bookCount[x], x) for x in bookCount]
mostPopular.sort()
mostPopular.reverse()

return1 = set()
count = 0
threshold_popularity = 0.6
threshold_jaccard = 0.02

for ic, i in mostPopular:
    count += ic
    return1.add(i)
    if count > totalRead * threshold_popularity: break

prediction = []
mostSimilarList = []

for l in X_valid:
    mostSimilarList = mostSimilar(l[0], l[1])

    if l[1] in return1 or mostSimilarList[0] >= threshold_jaccard:
        prediction.append(1)
    else:
        prediction.append(0)

correctPredictionValid = np.array(prediction) == np.array(y_valid)
accuracy = sum(correctPredictionValid) / len(correctPredictionValid)
```

In [32]:
```
accuracy
```

Out[32]:
```
0.7493
```

Question 5:

User Name: jameschoe
Display Name: CUI, HONGJIAN

In [33]:

```
bookCount = defaultdict(int)
totalRead = 0

for d in dataset:
    bookCount[d['bookID']] += 1
    totalRead += 1

mostPopular = [(bookCount[x], x) for x in bookCount]
mostPopular.sort()
mostPopular.reverse()

return1 = set()
count = 0
threshold_popularity = 0.6
threshold_jaccard = 0.02

for ic, i in mostPopular:
    count += ic
    return1.add(i)
    if count > totalRead * threshold_popularity: break

mostSimilarList = []
predictions = open("/home/cui/Projects/PycharmProjects/CSE-158/data/pred

for l in open("/home/cui/Projects/PycharmProjects/CSE-158/data/pairs_Rea
    if l.startswith("userID"):
        #header
        predictions.write(l)
        continue
    u,b = l.strip().split('-')
    mostSimilarList = mostSimilar(u, b)
    if b in return1 or mostSimilarList[0] >= threshold_jaccard:
        predictions.write(u + '-' + b + ",1\n")
    else:
        predictions.write(u + '-' + b + ",0\n")

predictions.close()
```

CSE 158, Fall 2019: Homework 3

Name: CUI, HONGJIAN **PID:** U08398995

Tasks (Category prediction):

Question 6:

In [2]:
```
import matplotlib.pyplot as plt
import json
import numpy as np
import random
import string
from collections import defaultdict
from nltk.stem.porter import PorterStemmer
from sklearn import linear_model
```

In [3]:
```
def parseData(fname):
    for l in open(fname):
        yield eval(l)
```

In [4]:
```
path = "/home/cui/Projects/PycharmProjects/CSE-158/data/train_Category.
```

In [5]:
```
dataset = list(parseData(path))
```

In [6]:
```
dataset[0]
```

Out[6]:
```
{'n_votes': 0,
 'review_id': 'r99763621',
 'user_id': 'u17334941',
 'review_text': "Genuinely enthralling. If Collins or Bernard did inve
nt this out of whole cloth, they deserve a medal for imagination. Lets
leave the veracity aside for a moment - always a touchy subject when i
t comes to real life stories of the occult - and talk about the conten
ts. \n The Black Alchemist covers a period of two years in which Colli
ns, a magician, and Bernard, a psychic, undertook a series of psychic
quests that put them in opposition with the titular Black Alchemist. A
s entertainment goes, the combination of harrowing discoveries, ancien
t lore, and going down the pub for a cigarette and a Guinness, trying
to make sense of it all while a hen party screams at each other, is a
winner. It is simultaneously down to earth and out of this world. \n I
t reads fast, both because of the curiousity and because Collins has a
very clear writing style. Sometimes its a little clunky or over repeti
tive and there's a few meetings that get underreported, but I am very
much quibbling here. Mostly important, he captures his own and Bernar
d's sense of wonder, awe and occasionally revulsion enough that I shar
ed them.",
 'rating': 5,
 'genreID': 2,
 'genre': 'fantasy_paranormal'}
```

In [7]:
```
split = 190000
```

In [8]:
```
training_set = dataset[:split]
validation_set = dataset[split:]
```

In [9]:
```
wordCounts = defaultdict(int)
punctuation = set(string.punctuation)
totalWords = 0
stemmer = PorterStemmer()
```

In [10]:
```
for d in training_set:
    text = d['review_text']
    text = text.lower()
    text = [c for c in text if not (c in punctuation)]
    text = ''.join(text)
    words = text.strip().split()
    for word in words:
        w = stemmer.stem(word)
        totalWords += 1
        wordCounts[w] += 1
```

In [11]:
```
len(wordCounts)
```

Out[11]:
```
360391
```

In [12]:
```
popularWords = sorted(wordCounts.items(), key=lambda x:x[1])[-1000:]
```

In [13]:
```
popularWords.reverse()
```

In [14]:
```
[(w[0], round(w[1] / totalWords, 4)) for w in popularWords[:10]]
```

Out[14]:
```
[('the', 0.0489),
 ('and', 0.0296),
 ('a', 0.026),
 ('to', 0.0247),
 ('i', 0.0241),
 ('of', 0.0214),
 ('it', 0.0166),
 ('is', 0.0145),
 ('in', 0.0141),
 ('thi', 0.0128)]
```

In [15]:
```
words = [w[0] for w in popularWords]
```

In [16]:
```
wordId = dict(zip(words, range(len(words))))
wordSet = set(words)
```

Question 7:

In [17]:
```
def feature(datum):
    feat = [0] * len(wordSet)
    text = datum['review_text']
    text = text.lower()
    text = [c for c in text if not (c in punctuation)]
    text = ''.join(text)

    word = text.strip().split()
    for w in word:
        w = stemmer.stem(w)
        if not (w in wordSet): continue
        feat[wordId[w]] += 1
    feat.append(1)

    return feat
```

In [18]:
```
X_train = [feature(d) for d in training_set]
X_validation = [feature(d) for d in validation_set]
y_train = [d['genreID'] for d in training_set]
y_validation = [d['genreID'] for d in validation_set]
```

In [18]:
```
model = linear_model.LogisticRegression(solver='lbfgs', multi_class='au
```

In [19]:
```
model.fit(X_train, y_train)
```

```
/home/cui/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/l
ogistic.py:947: ConvergenceWarning: lbfgs failed to converge. Increase
the number of iterations.
  "of iterations.", ConvergenceWarning)
```

Out[19]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept
=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verb
ose=0,
                   warm_start=False)
```

In [20]:
```
validation_predictions = model.predict(X_validation)
correctPredictionsValidation = validation_predictions == y_validation
```

In [21]:
```
sum(correctPredictionsValidation) / len(correctPredictionsValidation)
```

Out[21]:
```
0.6633
```

Question 8:

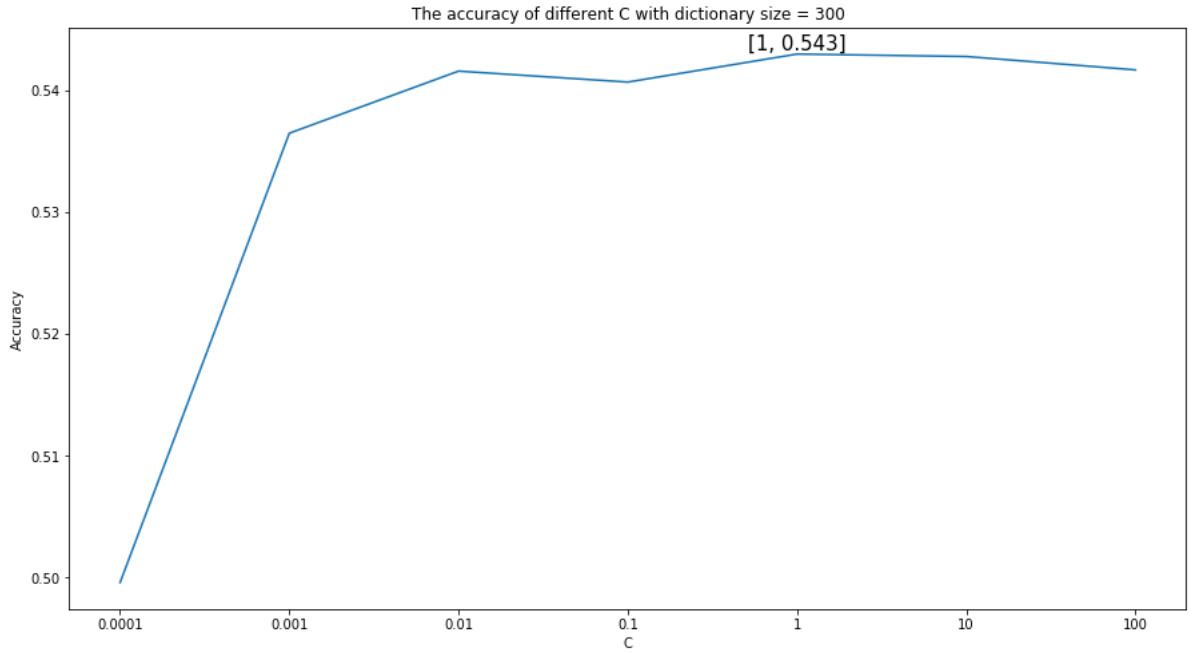
In [22]:
```
# dictionary size from 300 to 2400, c from 0.0001 to 100
dic_size = [300 * (n + 1) for n in range(8)]
C = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
accuracyList = []
```

In [23]:
```
for s in dic_size:
    print("Dictionary size: " + str(s))

    for c in C:
        print("C = " + str(c))

        popularWords = sorted(wordCounts.items(), key=lambda x:x[1])[-s
        popularWords.reverse()
        words = [w[0] for w in popularWords]
        wordId = dict(zip(words, range(len(words))))
        wordSet = set(words)

        X_train = [feature(d) for d in training_set]
        X_validation = [feature(d) for d in validation_set]

        model = linear_model.LogisticRegression(solver='lbfgs', C=c, mu
        model.fit(X_train, y_train)
        validation_predictions = model.predict(X_validation)
        correctPredictionsValidation = validation_predictions == y_vali
        accuracy = sum(correctPredictionsValidation) / len(correctPredi

        accuracyList.append([s, c, accuracy])
```

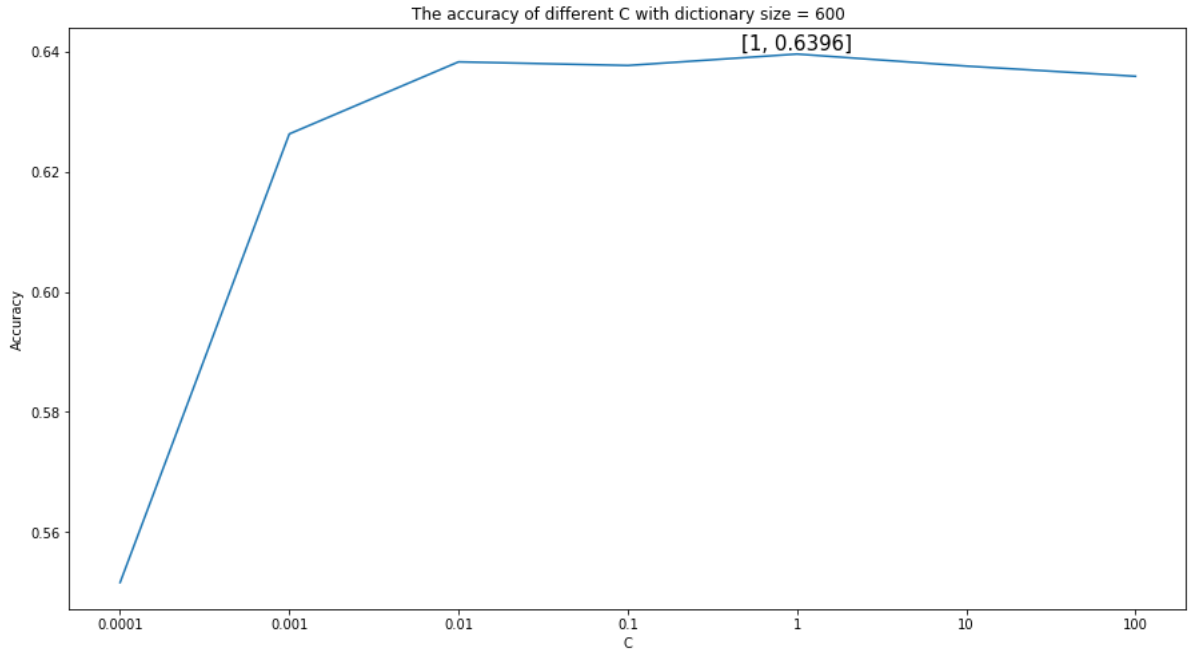
...

In [52]:
```
def draw(acc_list, dic_size):
    plt.figure(figsize=(15,8))

    y_plt = [d[2] for d in acc_list if d[0] == dic_size]

    x_axis = [100 / len(C) * C.index(x) for x in C]

    plt.plot(x_axis, y_plt)

    plt.xticks(x_axis, [str(c) for c in C])
    y_max = max(y_plt)
    x_max = x_axis[y_plt.index(y_max)]
    plt.text(x_max, y_max, [C[y_plt.index(y_max)], y_max],ha='center',

    plt.title("The accuracy of different C with dictionary size = " + s
    plt.xlabel("C")
    plt.ylabel("Accuracy")
    plt.show()
```

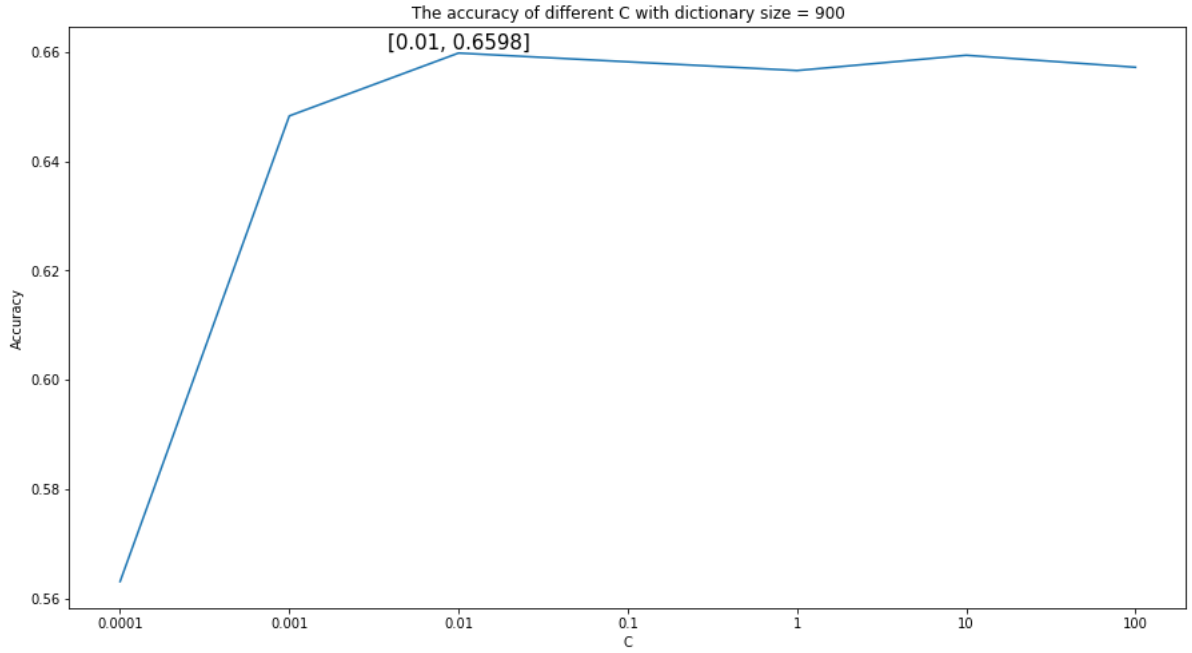
In [53]:
```
draw(accuracyList, 300)
```



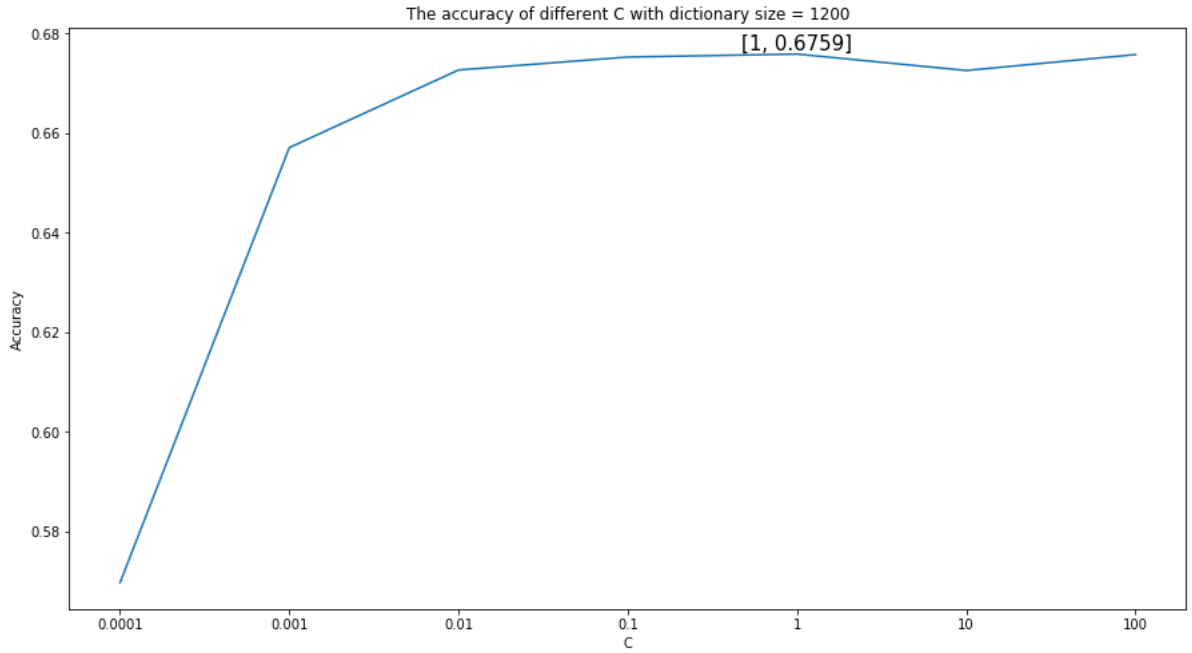
In [54]:
```
draw(accuracyList, 600)
```



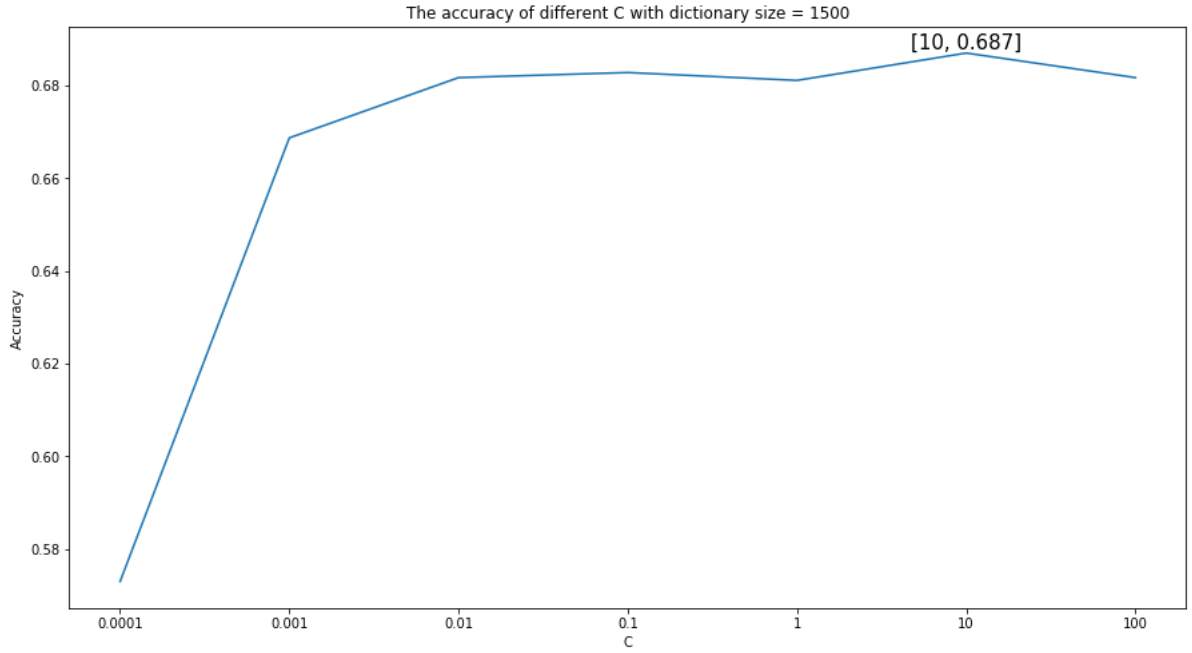
In [55]:
```
draw(accuracyList, 900)
```



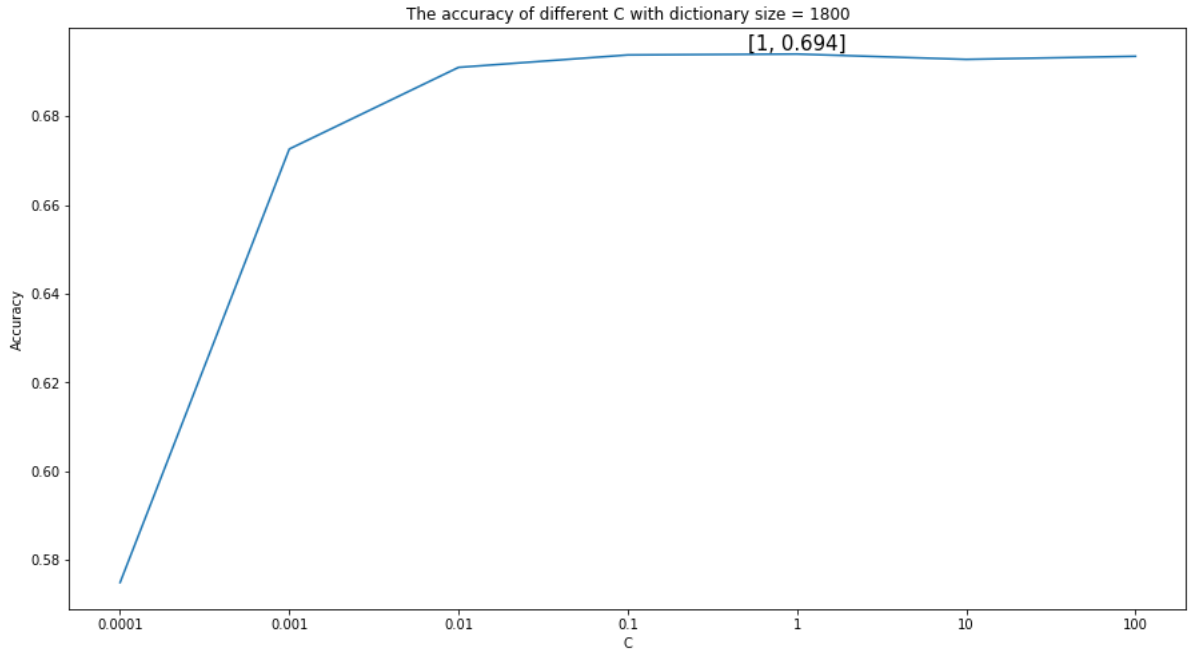
In [56]:
```
draw(accuracyList, 1200)
```



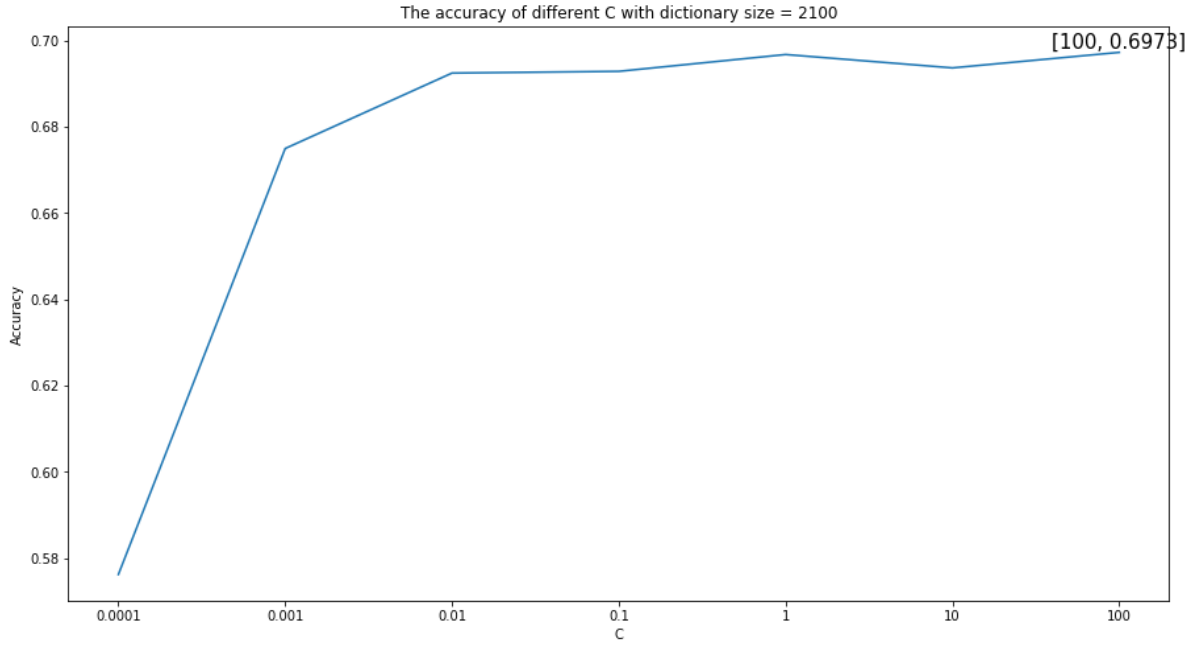
In [57]:
```
draw(accuracyList, 1500)
```



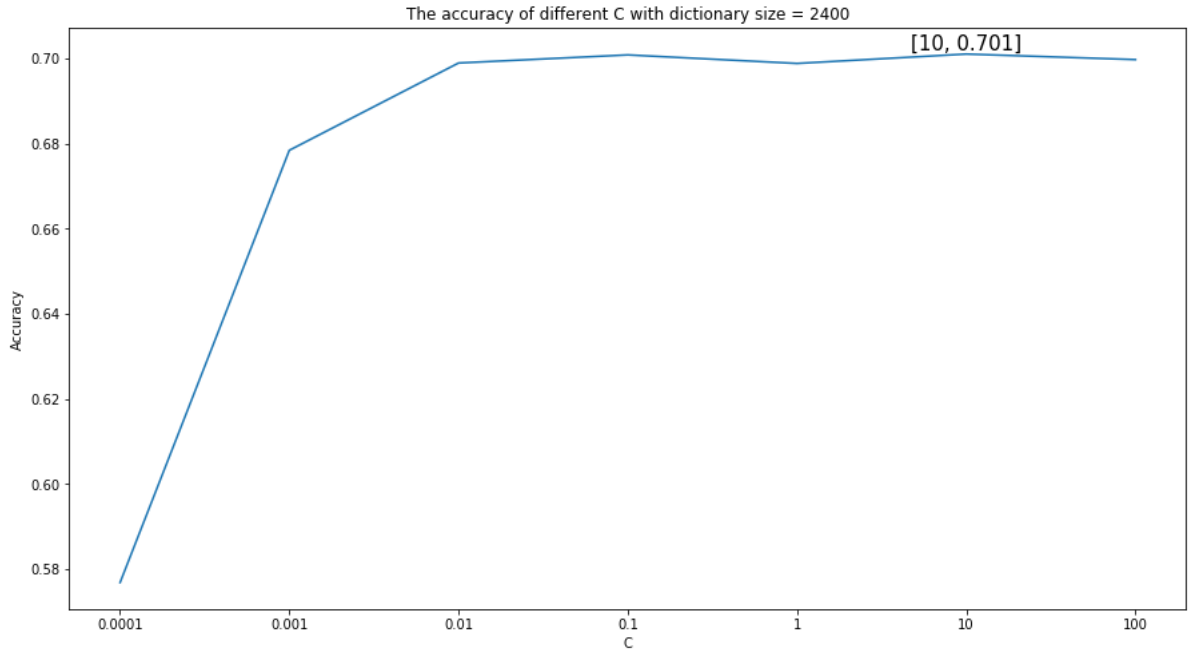
In [58]:
```
draw(accuracyList, 1800)
```



In [59]:
```
draw(accuracyList, 2100)
```



In [60]:
```
draw(accuracyList, 2400)
```



Images as shown above, we can find that when the dictionary size = 2400 and C = 10, the accuracy is the highest.

In [19]:
```
popularWords = sorted(wordCounts.items(), key=lambda x:x[1])[-2400:]
popularWords.reverse()
words = [w[0] for w in popularWords]
wordId = dict(zip(words, range(len(words))))
wordSet = set(words)

X_train = [feature(d) for d in training_set]
X_validation = [feature(d) for d in validation_set]

model = linear_model.LogisticRegression(solver='lbfgs', C=10, multi_cla
model.fit(X_train, y_train)
```

```
/home/cui/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/l
ogistic.py:947: ConvergenceWarning: lbfgs failed to converge. Increase
the number of iterations.
  "of iterations.", ConvergenceWarning)
```

Out[19]:
```
LogisticRegression(C=10, class_weight=None, dual=False, fit_intercept=
True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verb
ose=0,
                   warm_start=False)
```

In [21]:
```
dataset_test = list(parseData("/home/cui/Projects/PycharmProjects/CSE-1
```

In [56]:
```
predictions = open("/home/cui/Projects/PycharmProjects/CSE-158/data/pre
predictions.write("userID-reviewID,prediction\n")

for l in dataset_test:
    cat = model.predict([feature(l)])
    predictions.write(l['user_id'] + '-' + l['review_id'] + "," + str(ca

predictions.close()
```