CSE 258 - Homework4.

January 8, 2020

[1]:
```
import numpy
import urllib
import scipy.optimize
import random
from collections import defaultdict # Dictionaries with default values
import nltk
import string
from nltk.stem.porter import *
from sklearn import linear_model
import ast
```

0.0.1 Data Processing

[2]:
```
def parseDataFromFile(fname):
  for l in open(fname):
    yield ast.literal_eval(l)
```

read the first 10,000 reviews from the corpus

[3]:
```
print("Reading data...")
data = list(parseDataFromFile("train_Category.json"))[:10000]
print("done")
```

```
Reading data...
done
```

[4]:
```
data[1]
```

[4]:
```
{'n_votes': 0,
 'review_id': 'r24440074',
 'user_id': 'u08070901',
 'review_text': 'Pretty decent. The ending seemed a little rush but a good
ending to the first trilogy in this series. The fact that most of the time it is
a military fantasy makes it interesting. Also all of the descriptions of food
just make me hungry.',
 'rating': 5,
 'genreID': 2,
 'genre': 'fantasy_paranormal'}
```

[5]:
```
corpus = [d['review_text'] for d in data]
```

[6]:
```
corpus[1]
```

[6]:
```
'Pretty decent. The ending seemed a little rush but a good ending to the first
trilogy in this series. The fact that most of the time it is a military fantasy
makes it interesting. Also all of the descriptions of food just make me hungry.'
```

read the reviews without capitalization or punctuation

[7]:
```
wordCount = defaultdict(int)
punctuation = set(string.punctuation) # give the all sets of punctuation.
for corp in corpus:
    r = ''.join([c for c in corp.lower() if not c in punctuation])
    for w in r.split():
        wordCount[w] += 1

print(len(wordCount))
```

```
73286
```

[9]:
```
counts_uni = [(wordCount[w], w) for w in wordCount]
counts_uni.sort()
counts_uni.reverse()
```

[10]:
```
counts_uni[:5]
```

[10]:
```
[(73431, 'the'), (44301, 'and'), (39577, 'a'), (36821, 'to'), (36581, 'i')]
```

[11]:
```
# corpus: each review with capitalization and punctuation
# texts: each review without capitalization and punctuation

# ---- unigram ----
# wordCount: repeat times of unigram(unsorted)
# counts_uni: sorted unigram repeat times
# df: frequency of every word, regradless of repeat in one review
# uni_1k: top 1k unigrams
# uni_Id: top 1k bigrams with index

# ---- bigram ----
# bigrams_count: repeat times of bigram(unsorted)
# counts_bi: sorted bigram repeat times
# words: top 1k bigrams
# wordId: top 1k bigrams with index

# ---- uni&bi ----
# bigrams_count(update): repeat times of unigram&bigram(unsorted)
# countAll: repeat times of uni&bi(sorted)
# words_all: top 1k unigram&bigram
# wordId_all: top 1k unigram&bigram with index
```

```
# ---- function ----
# puncFilter: eliminate the punctutation and capitalization
# bigrams: each one of it is a bigram
# feature: bigram-count-feat
# feature_combine: unigram&bigram-count-feat
# feature_idf: unigram-idf-feat
```

1 Task1

How many unique bigrams are there amongst the reviews? List the 5 most-frequently-occurring bigrams along with their number of occurrences in the corpus

[12]:
```
# string.punctuation: give the sets of all punctuation.
punctuation = set(string.punctuation)

def puncFilter(data, remove): # remove the punctuation or not
    if remove == True:
        return ''.join([c for c in data.lower() if not c in punctuation]).
 ↪split()
    else:
        return ' '.join(re.findall(r"\w+|[^\w\s]", data.lower())).split()
```

[13]:
```
def bigrams(text, remove):
    return nltk.bigrams(puncFilter(text, remove))
```

[14]:
```
test = bigrams(corpus[0], True)
lst = list(test)
lst[:5]
```

[14]:
```
[('genuinely', 'enthralling'),
 ('enthralling', 'if'),
 ('if', 'collins'),
 ('collins', 'or'),
 ('or', 'bernard')]
```

[15]:
```
bigrams_count = defaultdict(int)
for text in corpus:
    r = list(bigrams(text, True))
    for w in r:
        bigrams_count[w] += 1
```

[16]:
```
counts_bi = [(bigrams_count[w], w) for w in bigrams_count]
counts_bi.sort()
counts_bi.reverse()
```

[17]:
```
len(counts_bi)
```

[17]:
```
521502
```

[18]:
```
counts_bi[:5]
```

[18]:
```
[(7927, ('of', 'the')),
 (5850, ('this', 'book')),
 (5627, ('in', 'the')),
 (3189, ('and', 'the')),
 (3183, ('is', 'a'))]
```

2 Task2

The code provided performs least squares using the 1000 most common unigrams. Adapt it to use the 1000 most common bigrams and report the MSE obtained using the new predictor (use bigrams only, i.e., not unigrams+bigrams) (1 mark). Note that the code performs regularized regression with a regularization parameter of 1.0. The prediction target should be the rating field in each review.

[19]:
```
words = [x[1] for x in counts_bi[:1000]]
words[:5]
```

[19]:
```
[('of', 'the'), ('this', 'book'), ('in', 'the'), ('and', 'the'), ('is', 'a')]
```

[20]:
```
r = list(bigrams(corpus[0], True))
r[:5]
```

[20]:
```
[('genuinely', 'enthralling'),
 ('enthralling', 'if'),
 ('if', 'collins'),
 ('collins', 'or'),
 ('or', 'bernard')]
```

[21]:
```
len(words[0])
```

[21]:
```
2
```

[22]:
```
wordId = dict(zip(words, range(len(words)))) # from 0 to 1000
```

[23]:
```
def feature(datum):
    feat = [0]*len(words) # 1000
    r = list(bigrams(datum, True))
    for w in r:
        if w in words:
            feat[wordId[w]] += 1
    feat.append(1) #offset
    return feat
```

[24]:
```
r = feature(corpus[0])
len(corpus), len(data)
```

[24]:
```
(10000, 10000)
```

[25]:
```
X = [feature(c) for c in corpus]
y = [d['rating'] for d in data]
```

[26]:
```
clf = linear_model.Ridge(1.0, fit_intercept=False) # MSE + 1.0 l2
clf.fit(X, y)
theta = clf.coef_
predictions = clf.predict(X)
```

[27]:
```
predictions[:5], y[:5]
```

[27]:
```
(array([3.76533043, 3.45717033, 3.50221229, 3.72832576, 3.56647851]),
 [5, 5, 4, 5, 5])
```

[28]:
```
def getMSE(predictions, y):
    return numpy.mean((y - predictions)**2)
```

[29]:
```
print(getMSE(predictions, y))
```

```
1.0178804824879226
```

3 Task 3

Repeat the above experiment using unigrams and bigrams, still considering the 1000 most common. That is, your model will still use 1000 features (plus an offset), but those 1000 features will be some combination of unigrams and bigrams. Report the MSE obtained using the new predictor

[30]:
```
bigrams_count.update(wordCount)
```

[31]:
```
len(wordCount), len(bigrams_count)
```

[31]:
```
(73286, 594788)
```

[32]:
```
countAll = [(bigrams_count[w], w) for w in bigrams_count]
```

[33]:
```
countAll.sort(key=lambda t: t[0])
countAll.reverse()
```

[34]:
```
countAll[15:20]
```

[34]:
```
[(11131, 'with'),
 (9638, 'her'),
 (9138, 'as'),
 (7927, ('of', 'the')),
 (7207, 'on')]
```

[35]:
```
words_all = [x[1] for x in countAll[:1000]]
```

[36]:
```
words_all[15:20]
```

[36]:
```
['with', 'her', 'as', ('of', 'the'), 'on']
```

[37]:
```
wordId_all = dict(zip(words_all, range(len(words_all))))
```

[38]:
```
def feature_combine(datum):
    feat = [0]*len(words) # 1000
```

```
    r = list(bigrams(datum, True))
    for w in r:
        if w in words_all:
            feat[wordId_all[w]] += 1

    r = list(puncFilter(datum, True))
    for w in r:
        if w in words_all:
            feat[wordId_all[w]] += 1

    feat.append(1) #offset
    return feat
```

[39]:
```
X = [feature_combine(c) for c in corpus]
y = [d['rating'] for d in data]
```

[40]:
```
X[0][:10]
```

[40]:
```
[8, 9, 14, 3, 2, 9, 2, 2, 4, 2]
```

[41]:
```
clf = linear_model.Ridge(1.0, fit_intercept=False) # MSE + 1.0 l2
clf.fit(X, y)
theta = clf.coef_
predictions = clf.predict(X)
```

[42]:
```
print(getMSE(predictions, y))
```

```
0.9683729530414934
```

4 Task 4

What is the inverse document frequency of the words stories, magician, psychic, writing, and wonder? What are their tf-idf scores in the first review (using log base 10, following the first definition of tf-idf given in the slides)

[43]:
```
from math import log
```

[44]:
```
# frequency of every word, regradless of repeat in one review

df = defaultdict(int)
for c in corpus:
    r = list(puncFilter(c, True))  # based on unigram
    words = set(r)
    for w in words:
        df[w] += 1
```

[45]:
```
# Inverse document frequency

def findIdf(word):
```

```
    f = df[word]
    if f == 0:
        return log(len(corpus), 10)
    return log(len(corpus) / float(f), 10)
```

[46]:
```
findIdf("stories")
```

[46]:
```
1.1174754620451195
```

[47]:
```
# number of times the word appears in text[i]

def findTf(word, text):
    words = text
    c = 0
    for w in words:
        if w == word:
            c += 1
    return c
```

[48]:
```
texts = [puncFilter(c, True) for c in corpus]
```

[49]:
```
findTf("a", texts[0]), findIdf("a"), findIdf("magnus"), log(10000/763, 10)
```

[49]:
```
(14, 0.09156860103399361, 3.221848749616356, 1.1174754620451195)
```

[50]:
```
def tfidf(word, text):
    return findTf(word, text) * findIdf(word)
```

[51]:
```
tfidf("magnus", texts[10])
```

[51]:
```
16.10924374808178
```

[52]:
```
words_5 = ['stories', 'magician', 'psychic', 'writing', 'wonder']
```

[53]:
```
for w in words_5:
    print('"%s \t idf: %f \t tf-idf:%f"' % (w, findIdf(w), tfidf(w, texts[0])))
```

```
"stories         idf: 1.117475   tf-idf:1.117475"
"magician        idf: 2.657577   tf-idf:2.657577"
"psychic         idf: 2.602060   tf-idf:5.204120"
"writing         idf: 0.997834   tf-idf:0.997834"
"wonder          idf: 1.767004   tf-idf:1.767004"
```

5 Task 5

Adapt your unigram model to use the tfidf scores of words, rather than a bag-of-words representation. That is, rather than your features containing the word counts for the 1000 most common unigrams, it should contain tfidf scores for the 1000 most common unigrams. Report the MSE of this new model.

[54]:
```
uni_1k = [x[1] for x in counts_uni[:1000]]
```

[55]:
```
uni_1k[11:13]
```

[55]:
```
['was', 'book']
```

[56]:
```
uni_Id = dict(zip(uni_1k, range(len(uni_1k))))
```

[57]:
```
def feature_idf(datum):
    feat = [0]*len(uni_1k) # 1000

    r = list(puncFilter(datum, True))
    for w in r:
        if w in uni_1k:
            feat[uni_Id[w]] = tfidf(w, r)
    feat.append(1) #offset
    return feat
```

[58]:
```
X = [feature_idf(c) for c in corpus]
y = [d['rating'] for d in data]
```

[59]:
```
len(X[100])
```

[59]:
```
1001
```

[60]:
```
clf = linear_model.Ridge(1.0, fit_intercept=False) # MSE + 1.0 l2
clf.fit(X, y)
theta = clf.coef_
predictions = clf.predict(X)
```

[61]:
```
print(getMSE(predictions, y))
```

```
0.9660150616760588
```

6 Task 6

Which other review has the highest cosine similarity compared to the first review (provide the review id, or the text of the review)

[62]:
```
from sklearn.metrics.pairwise import cosine_similarity
```

[63]:
```
cos_sim = []
for i in range(1, len(data)):
    d = data[i]
    similarity = cosine_similarity(X[0:1], X[i:i+1])[0,0]
    cos_sim.append((similarity, d['review_id']))
cos_sim.sort()
cos_sim.reverse()
```

[64]:
```
print("cosine similarity:" , cos_sim[0][0])
print("review id:" , cos_sim[0][1])
```

```
cosine similarity: 0.34862531225799814
review id: r81495268
```

7 Task 7

Implement a validation pipeline for this same data, by randomly shuffling the data, using 10,000 reviews for training, another 10,000 for validation, and another 10,000 for testing.1 Consider regularization parameters in the range {0.01, 0.1, 1, 10, 100}, and report MSEs on the test set for the model that performs best on the validation set. Using this pipeline, compare the following alternatives in terms of their performance:
Unigrams vs. bigrams
Removing punctuation vs. preserving it. The model that preserves punctuation should treat punctuation characters as separate words, e.g. Amazing! would become [amazing, !]
tfidf scores vs. word counts
In total you should compare 2 ⊏ 2 ⊏ 2 = 8 models, and produce a table comparing their performance

[65]:
```
from random import shuffle
```

[66]:
```
data_All = list(parseDataFromFile("train_Category.json"))
```

[67]:
```
shuffle(data_All)
```

[68]:
```
train = data_All[:10000]
validation = data_All[10000:20000]
test = data_All[20000:30000]
```

[70]:
```
train_x = [d['review_text'] for d in train]
train_y = [d['rating'] for d in train]
validation_x = [d['review_text'] for d in validation]
validation_y = [d['rating'] for d in validation]
test_x = [d['review_text'] for d in test]
test_y = [d['rating'] for d in test]
```

[71]:
```
uni_cnt =  defaultdict(int)
bi_cnt =defaultdict(int)
uni_cnt_ = defaultdict(int)
bi_cnt_ = defaultdict(int)

for text in train_x:
    r = list(puncFilter(text, True))
    for w in r:
        uni_cnt[w] += 1

    r = list(bigrams(text, True))
    for w in r:
        bi_cnt[w] += 1

    r = list(puncFilter(text, False))
    for w in r:
        uni_cnt_[w] += 1
```

```
    r = list(bigrams(text, False))
    for w in r:
        bi_cnt_[w] += 1
```

[72]:
```
cnt_uni = [(uni_cnt[w], w) for w in uni_cnt]
cnt_bi = [(bi_cnt[w], w) for w in bi_cnt]
cnt_uni_ = [(uni_cnt_[w], w) for w in uni_cnt_]
cnt_bi_ = [(bi_cnt_[w], w) for w in bi_cnt_]
cnt_uni.sort()
cnt_uni.reverse()
cnt_bi.sort()
cnt_bi.reverse()
cnt_uni_.sort()
cnt_uni_.reverse()
cnt_bi_.sort()
cnt_bi_.reverse()
```

[73]:
```
uni = [x[1] for x in cnt_uni[:1000]]
bi = [x[1] for x in cnt_bi[:1000]]
uni_ = [x[1] for x in cnt_uni_[:1000]]
bi_ = [x[1] for x in cnt_bi_[:1000]]
```

[74]:
```
id_uni = dict(zip(uni, range(len(uni))))
id_bi = dict(zip(bi, range(len(bi))))
id_uni_ = dict(zip(uni_, range(len(uni_))))
id_bi_ = dict(zip(bi_, range(len(bi_))))
```

[75]:
```
def feature_count(datum, dataset, uniBi, uniBiId, remove):
    feat = [0]*len(dataset) # 1000
    r = list(uniBi(datum, remove))
    for w in r:
        if w in dataset:
            feat[uniBiId[w]] += 1
    feat.append(1) #offset
    return feat
```

[76]:
```
def feature_tfidf(datum, dataset, uniBi, uniBiId, remove):
    feat = [0]*len(dataset) # 1000
    r = list(uniBi(datum, remove))
    for w in r:
        if w in dataset:
            feat[uniBiId[w]] = tfidf(w, datum)
    feat.append(1) #offset
    return feat
```

[77]:
```
lamdas = [0.01, 0.1, 1, 10, 100]
```

[78]:
```
def valid(X, y, lamdas):
    min_mse = 100
```

```
    best_lam = None
    for lam in lamdas:
        clf = linear_model.Ridge(lam, fit_intercept=False) # MSE + 1.0 l2
        clf.fit(X, y)
        mse = getMSE(clf.predict(X), y)

        if mse < min_mse:
            min_mse = mse
            best_lam = lam

    return best_lam
```

[79]:
```
def pipeline_count(dataset, uniBi, uniBiId, remove):
    X = [feature_count(c, dataset, uniBi, uniBiId, remove) for c in train_x]
    y = train_y
    valid_X = [feature_count(c, dataset, uniBi, uniBiId, remove) for c in␣
 ↪validation_x]
    lam = valid(valid_X, validation_y, lamdas)
    clf = linear_model.Ridge(lam, fit_intercept=False) # MSE + 1.0 l2
    clf.fit(X, y)
    test_X = [feature_count(c, dataset, uniBi, uniBiId, remove) for c in test_x]
    predictions = clf.predict(test_X)
    mse = getMSE(predictions, test_y)
    return lam, mse
```

[80]:
```
def pipeline_tfidf(dataset, uniBi, uniBiId, remove):
    X = [feature_tfidf(c, dataset, uniBi, uniBiId, remove) for c in train_x]
    y = train_y
    valid_X = [feature_tfidf(c, dataset, uniBi, uniBiId, remove) for c in␣
 ↪validation_x]
    lam = valid(valid_X, validation_y, lamdas)
    clf = linear_model.Ridge(lam, fit_intercept=False) # MSE + 1.0 l2
    clf.fit(X, y)
    test_X = [feature_count(c, dataset, uniBi, uniBiId, remove) for c in test_x]
    predictions = clf.predict(test_X)
    mse = getMSE(predictions, test_y)
    return lam, mse
```

[81]:
```
# Uni, removing punctuation, word count
lamd2, mse2 = pipeline_count(uni, puncFilter, id_uni, True)
```

```
(0.01, 1.1909286683314602)
```

[82]:
```
# Uni, removing punctuation, tfidf
lamd2, mse2 = pipeline_tfidf(uni, puncFilter, id_uni, True)
lamd2, mse2
```

```
(0.01, 1.3219871724186665)
```

[83]:
```
# Bi, removing punctuation, word count
lamd3, mse3 = pipeline_count(bi, bigrams, id_bi, True)
lamd3, mse3
```

```
(0.01, 1.2300564220457386)
```

[84]:
```
# Bi, removing punctuation, tfidf
lamd4, mse4 = pipeline_tfidf(bi, bigrams, id_bi, True)
lamd4, mse4
```

```
(0.01, 1.3294717385220551)
```

[85]:
```
# Uni, preserving punctuation, word count
lamd5, mse5 = pipeline_count(uni_, puncFilter, id_uni_, False)
lamd5, mse5
```

```
(0.01, 1.1898371657126983)
```

[86]:
```
# Uni, preserving punctuation, tfidf
lamd6, mse6 = pipeline_tfidf(uni_, puncFilter, id_uni_, False)
lamd6, mse6
```

```
(0.01, 1.3104545969416221)
```

[87]:
```
# Bi, removing punctuation, word count
lamd7, mse7 = pipeline_count(bi_, bigrams, id_bi_, False)
lamd7, mse7
```

```
(0.01, 1.2420305825444256)
```

[88]:
```
# Bi, removing punctuation, tfidf
lamd8, mse8 = pipeline_tfidf(bi_, bigrams, id_bi_, False)
lamd8, mse8
```

```
(0.01, 1.3294717385220551)
```

[90]:
```
title = ['Uni, removing punctuation, count',
         'Uni, removing punctuation, tfidf',
         'Bi, removing punctuation, count',
         'Bi, removing punctuation, tfidf',
         'Uni, preserving punctuation, count',
         'Uni, preserving punctuation, tfidf',
         'Bi, removing punctuation, count',
         'Bi, removing punctuation, tfidf']
```

```
lambd = [lamd1, lamd2, lamd3, lamd4, lamd5, lamd6, lamd7, lamd8]
msee = [mse1, mse2, mse3, mse4, mse5, mse6, mse7, mse8]
for i in range(0,8):
    print('"%s \t lamda: %.2f \t MSE: %f"' % (title[i], lambd[i], msee[i]))
```

```
"Uni, removing punctuation, count        lamda: 0.01     MSE: 1.190929"
"Uni, removing punctuation, tfidf        lamda: 0.01     MSE: 1.321987"
"Bi, removing punctuation, count         lamda: 0.01     MSE: 1.230056"
"Bi, removing punctuation, tfidf         lamda: 0.01     MSE: 1.329472"
"Uni, preserving punctuation, count      lamda: 0.01     MSE: 1.189837"
"Uni, preserving punctuation, tfidf      lamda: 0.01     MSE: 1.310455"
"Bi, removing punctuation, count         lamda: 0.01     MSE: 1.242031"
"Bi, removing punctuation, tfidf         lamda: 0.01     MSE: 1.329472"
```

[ ]:
```
# "Unigram, removing punctuation, count the frequency"
# has the lowest MSE: 1.190929 when lambda is 0.01
```