باسمه تعالی

دانشگاه تهران - دانشکده مهندسی برق و کامپیوتر

درس: پردازش زبان طبیعی

گزارش تمرین کامپیوتری دوم

استاد: جناب آقای دکتر هشام فیلی

دستیار استاد: جناب باقرشاهی

دانشجو: مصطفی امیری

شماره دانشجویی: ۸۱۰۱۰۱۰۰۲

اردیبهشت ۱۴۰۲

فهرست مطالب

فهرست تصاویر

۱ سوال اول

۱.۱ آماده سازی داده‌ها

ابتدا فایل داده‌ها را دریافت و در یک دیتافریم قرار دادم:

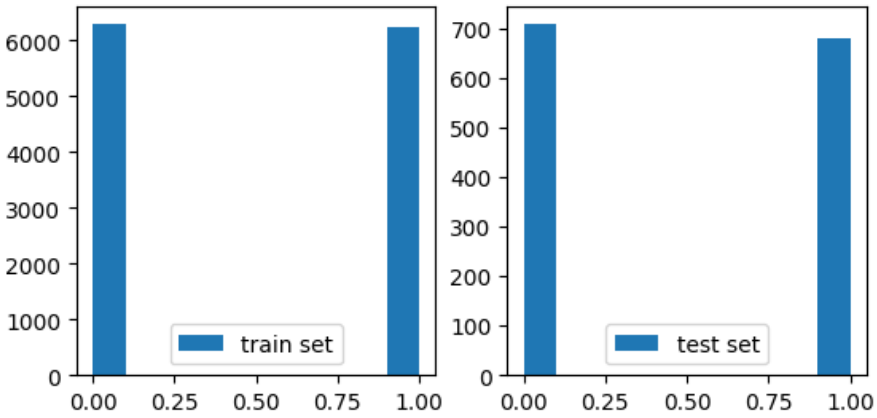
```
snappfood = pd.read_csv(
    "datasets\\Snapfood\\Snappfood - Sentiment Analysis.csv",
    encoding="utf-8",
    on_bad_lines="skip",
    sep="\t")
snappfood = snappfood.drop('Unnamed: 0', axis=1)
snappfood.head()
```

سپس ۲۰ درصد از داده‌ها را انتخاب کردم و ۱۰ درصد از این مقدار را برای تست و بقیه را برای آموزش قرار دادم:

```
train_ds, _= train_test_split(snappfood, test_size=0.8)
train_ds, test_ds = train_test_split(train_ds, test_size=0.1)
```

سپس با استفاده از قطعه کد زیر نمودار هیستوگرام توزیع داده‌های آموزش و تست را برای اطمینان از مشابهت توزیع داده‌ها رسم کردم که نمودار آن به صورت شکل ۱ شد.

```
fig, axes = plt.subplots(ncols=2)
fig.set_figheight(3)
axes[0].hist(train_ds["label_id"], label="train set")
axes[1].hist(test_ds['label_id'], label="test set")
axes[0].legend()
axes[1].legend()
```



شکل ۱: توزیع برچسب داده‌های آموزش و تست

برای پیش پردازش داده‌ها با استفاده از عبارات منظم حروفی که مانند «ی» در کلمه خیییییییییییییییلی چندبار تکرار شده‌اند را حذف و با استفاده از کتابخانه هضم آنها را نرمال کردم و سپس روی داده‌ها این تابع را اعمال کردم. در ادامه نیز نمونه‌هایی از جملات پیش پردازش شده را در خروجی چاپ کردم.

```
def preprocces(text):
  regex = r'(\w)\1{2,}'
  text = re.sub(regex, r"\1", text, 0, re.MULTILINE)
  text = Normalizer().normalize(text)
  return text

# applying preproccess function to data
train_ds["proccessed"] = train_ds['comment'].apply(preprocces)
train_ds["tokens"] = train_ds['proccessed'].apply(WordTokenizer().tokenize)

test_ds["proccessed"] = test_ds['comment'].apply(preprocces)
test_ds["tokens"] = test_ds['proccessed'].apply(WordTokenizer().tokenize)


# printing some example of proccessed data in console
for id in np.random.choice(train_ds.index, 20):
  print(train_ds.loc[id]['comment'])
  print(train_ds.loc[id]['proccessed'])
```

برای نمونه جمله:

- داغ و تازه عاااالی

به جمله زیر تبدیل می‌شود:

- داغ و تازه عالی

tf-idf ۲.۱

سپس برای هر کلمه یک بردار به اندازه تعداد نظرات ایجاد کردم و در درایه متناظر با آن نظر تعداد تکرار آن کلمه در آن نظر را قرار دادم. به این ترتیب بردار $term - frequency(tf)$ برای هر کلمه ساخته شد.

```
vocabs_tf={}
for id, doc in tqdm(enumerate(train_ds['tokens'])):
    for t in doc:
        if t not in stopwords_list():
            if t not in vocabs_tf:
                vocabs_tf[t] = np.array([0]*len(train_ds['proccessed']))
            vocabs_tf[t][id] += 1
```

با استفاده از قطعه کد زیر تعداد تکرار هر کلمه در نظرات مختلف و عبارت دیگر $frequency$ $document$ را محاسبه کردم:

```
vocabs_df ={}
for v in vocabs_tf:
    vocabs_df[v] = 0
for v in tqdm(vocabs_tf):
    vocabs_df[v] = sum([(int)(x>0) for x in vocabs_tf[v]])
```

مطابق روابط مربوط به محاسبه $tf - idf$ که در فصل ششم کتاب ژورافسکی آمده است بردار مربوط به هر کلمه را ساختم.

$$tf_{t,d} = log_{10}(count(t,d) + 1)$$
$$idf_t = log(\frac{N}{df_t})$$
$$\Rightarrow w_{t,d} = tf_{t,d} \times idf_t$$

```
tfidf = dict()
N = len(train_ds)
```

```
for t in vocabs_tf:
    tfidf[t] = np.log(vocabs_tf[t]+1)/np.log(10) * np.log(N/vocabs_df[t])/np.log
    (10)
```

سپس با استفاده از تابع زیر یک متن را طبق رابطه $d = \frac{w_1+w_2+\cdots+w_k}{k}$ به یک بردار تبدیل می‌کنم.

```
def tfidf_vectorizer(tokens):
    count = 0
    vec = []
    for t in tokens:
        if t in tfidf:
            count+=1
            if len(vec)>0:
                vec += tfidf[t]
            else:
                vec = np.copy(tfidf[t])
    vec = np.array(vec)
    if count != 0:
        vec = vec/count
    else:
        for t in tfidf:
            vec = np.array([0] * len(tfidf[t]))
            break
    return vec
```

سپس تابع فوق را روی داده‌ها اعمال کرده و بعد از تبدیل داده‌ها به بردار یک طبقه‌بند بیز ساده را روی داده‌های آموزش، آموزش داده و روی داده‌های تست نتایج را بدست آوردم.

```
test_ds["vec"] = test_ds["tokens"].apply(tfidf_vectorizer)
train_ds["vec"] = train_ds["tokens"].apply(tfidf_vectorizer)

train_dss = train_ds.dropna()
test_dss = test_ds.dropna()

clf = GaussianNB()
clf.fit(np.array([np.array(x) for x in train_dss["vec"]]), train_dss["label_id"
    ])
prd = clf.predict(np.array([np.array(x) for x in test_dss["vec"]]))

```

```
print(classification_report(test_dss["label_id"], prd))
```

۱.۲.۱ نتایج

نتایج این قسمت روی داده‌های تست به صورت شکل ۲ است.

	precision	recall	f1-score	support
0.0	0.86	0.61	0.71	665
1.0	0.72	0.91	0.80	723
accuracy			0.76	1388
macro avg	0.79	0.76	0.76	1388
weighted avg	0.78	0.76	0.76	1388

شکل ۲: نتایج تحلیل احساسی بر اساس بردار tf-idf

۳.۱ PPMI

به ازاء هر دو کلمه تعداد حضور همزمان آن ها در یک نظر بوسیله قطعه کد زیر شمرده شده است:

```
coaccurance = dict()
for i in vocabs_tf:
    coaccurance[i] = dict()
    for j in vocabs_tf:
        coaccurance[i][j] = 0
for doc in tqdm(train_ds["tokens"]):
    for comb in  combinations(doc, 2):
        if comb[0] in vocabs_tf and comb[1] in vocabs_tf:
            coaccurance[comb[0]][comb[1]] += 1
            coaccurance[comb[1]][comb[0]] += 1
```

سپس تعداد مشاهده هر کلمه به تنهایی نیز محاسبه شده است.

```
for i in tqdm(vocabs_tf):
    count = 0
    for j in vocabs_tf:
        count += coaccurance[i][j]
    coaccurance[i]["total"] = count

total = 0
```

```
coaccurance["total"] = dict()
for j in tqdm(vocabs_tf):
    count = 0
    for i in vocabs_tf:
        count += coaccurance[i][j]
    total += count
    coaccurance["total"][j] = count
coaccurance["total"]["total"] = total
```

مقدار $PPMI$ برای هر دو کلمه نسبت به هم محاسبه و سپس با استفاده از این مقادیر بردار هر کلمه را ساخته‌ام:

```
ppmi = dict()
for i in tqdm(vocabs_tf):
    ppmi[i] = dict()
    for j in vocabs_tf:
        p_i_j = coaccurance[i][j] / coaccurance["total"]["total"]
        p_i = coaccurance[i]["total"] / coaccurance["total"]["total"]
        p_j= coaccurance["total"][j] / coaccurance["total"]["total"]
        ppmi[i][j] = 0
        if p_i != 0 and p_j != 0 and p_i_j !=0:
            ppmi[i][j] = max(np.log(p_i_j)-np.log(p_i)-np.log(p_j), 0)


ppmi_vec = dict()
for i in tqdm(vocabs_tf):
    ppmi_vec[i] = []
    for j in vocabs_tf:
        ppmi_vec[i].append(ppmi[i][j])
```

تابع زیر یک جمله را مشابه آنچه در بخش قبلی گفته شد به یک بردار تبدیل می‌کند:

```
def ppmi_vectorizer(tokens):
    count = 0
    vec = []
    for t in tokens:
        if t in ppmi_vec:
            count+=1
```

```
            if len(vec)>0:
                vec += ppmi_vec[t]
            else:
                vec = np.copy(ppmi_vec[t])
    vec = np.array(vec)
    if count != 0:
        vec = vec/count
    else:
        for t in ppmi_vec:
            vec = np.array([0] * len(ppmi_vec[t]))
            break
    return vec
```

سپس تابع فوق را روی داده‌ها اعمال کرده و بعد از تبدیل داده‌ها به بردار یک طبقه‌بند بیز ساده را روی داده‌های آموزش، آموزش داده و روی داده‌های تست نتایج را بدست آوردم.

```
test_dss["ppmi_vec"] = test_dss["tokens"].apply(ppmi_vectorizer)
train_dss["ppmi_vec"] = train_dss["tokens"].apply(ppmi_vectorizer)

clf = GaussianNB()
clf.fit(np.array([np.array(x) for x in train_dss["ppmi_vec"]]), train_dss["
    label_id"])
prd = clf.predict(np.array([np.array(x) for x in test_dss["ppmi_vec"]]))

print(classification_report(test_dss["label_id"], prd))
```

۱.۳.۱ نتایج

نتایج حاصل روی داده‌های تست به صورت شکل ۳ است.

	precision	recall	f1-score	support
0.0	0.87	0.59	0.70	665
1.0	0.71	0.92	0.80	723
accuracy			0.76	1388
macro avg	0.79	0.75	0.75	1388
weighted avg	0.79	0.76	0.75	1388

شکل ۳: نتایج تحلیل احساسی بر اساس PPMI

نتایج دو روش تفاوت زیادی با هم ندارند و به میزان بسیار کمی روش $tfidf$ نتایج بهتری داشته است.

۲ سوال دوم

۱.۲ پیش‌پردازش

ابتدا با قطعه کد زیر فایل داده شده را خوانده و در متغیر doc ذخیره کردم.ضمنا توضیحات اول فایل هم که از نوشته‌های شکسپیر نیست را حذف کردم.

```
with open("t8.shakespeare.txt", "r") as f:
    doc = ""
    for line in f.readlines():
        doc += line

doc = doc[10520:]
```

در مرحله بعدی توضیحاتی که در بین ستاره یا << >> قرار داشتند را با استفاده از $regex$ حذف کردم.

```
regex = r"\s+<<[\s\d\-\()A-Z_\.,:@;!\"\']+>>\s+"
result = re.sub(regex, r"", doc, 0, re.MULTILINE)
regex = r"\s+\*+[\s\d\-\()A-Za-z_\.,:@!;\"\']+\*+\s+"
result = re.sub(regex, r"", result, 0, re.MULTILINE)
```

سپس فاصله‌های اضافه و اعداد و تکرارهای بیش از ۳ بار یک حرف را حذف کردم.

```
regex = r"(\s)\s+"
result = re.sub(regex, r"\1", result, 0, re.MULTILINE)
regex = r"\d+"
result = re.sub(regex, "", result, 0, re.MULTILINE)
regex = r'(\w)\1{2,}'
result = re.sub(regex, "", result, 0, re.MULTILINE)
```

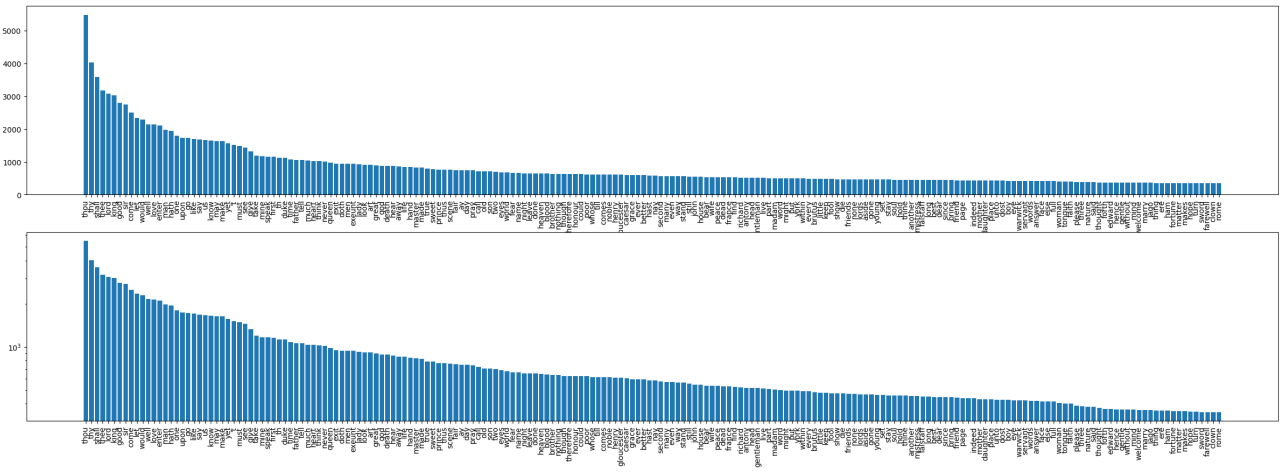
در مرحله بعد جملات و توکن‌های داخل هر جمله را از هم جدا کردم وبرخی توکن‌های بدون استفاده را حذف کردم.

```
sentences = sent_tokenize(result)
sentences = [word_tokenize(s.lower()) for s in sentences]
sentences = [[(re.sub(r"[,!:;\?\.\-\[\]\(\)]","",t)) for t in s if t not in ["'d
    ", "'ll", "'s","!",":", "?", ";", ",", ".","[","]","(", "(","'"]+
    stopwords_list]for s in sentences]
```

در مرحله بعد تعداد تکرار هر توکن را شمرده و بر اساس فراوانی آنها را مرتب کردم:

```
from collections import Counter
freqs = Counter()
for s in sentences:
  freqs.update(s)
freqs_sorted = sorted(freqs.items(),
                      key=lambda p: (p[1], p[0]),
                      reverse=True)
```

در شکل ۴ نمودار فراوانی ۲۰۰ کلمه پرتکرار به صورت عادی و با مقیاس لگاریتمی رسم کردم



شکل ۴: نمودار فراوانی ۲۰۰ توکن پرتکرار

با در نظر گرفتن ۲۳۰۰۰ توکن فراوانی مابقی توکن‌ها کمتر از ۱ درصد کل خواهد بود.

```
vocab_size = 23000
sum_freq_pruned = sum(f for _, f in freqs_sorted[vocab_size:])
print(sum_freq_pruned/sum(f for _, f in freqs_sorted)*100)
# 0.9895637457690861
```

برای سایر توکن‌ها یک توکن UNK در نظر گرفتم و در ابتدای این دیکشنری اضافه کردم و فراوانی آن را نیز برابر فراوانی توکن‌های حذف شده در نظر گرفتم.

```
freqs_sorted = [("UKN", sum_freq_pruned)] + freqs_sorted[:vocab_size-1]
```

با توجه به توضیحات کتاب ژورافسکی برای انتخاب نمونه‌های منفی به نسبت فراوانی توکن‌ها با توان ۰/۷۵ استفاده شده مطابق رابطه زیر که با قطعه کد زیر این نسب محاسبه و سپس برای تبدیل به اعداد صحیح تمام آنها

را مقیاس کرده و در یک جدول دیگر به تعداد مقدار مقیاس شده از آن توکن درج کردم.

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_{w'} count(w')^\alpha}$$

```
ns_table = {}
sum_freq = 0
for w, freq in freqs_sorted:
    ns_freq = freq **  0.75
    ns_table[w] = ns_freq
    sum_freq += ns_freq

scaler = (1<<24) / sum_freq
ns_table = [(w, freq, int(round(ns_table[w]*scaler))) for w, freq in
   freqs_sorted]

ns_table_expanded = []
for i, (_, _, count) in enumerate(ns_table):
  ns_table_expanded.extend([i] * count)
neg_ix = np.random.randint(len(ns_table_expanded), size=1000)
_ =plt.hist([ns_table_expanded[k] for k in neg_ix], bins=100)
```

سپس یک دیکشنری برای تبدیل توکن‌ها به آیدی عددی ایجاد کردم.

```
voc = { w:i for i, (w, _, _ ) in enumerate(ns_table) }
```

تابع زیر نیز وظیفه تولید داده‌های آموزشی را به عهده دارد. ساختار داده‌های تولیدی به این صورت است که یک پنجره را روی جمله حرکت می‌دهیم و به ازاء هر کلمه، هریک از کلمات مجاور آن و داخل پنجره را به عنوان بافتار و به تعداد ۴ نمونه منفی طبق توزیعی که در جدول تولید شده بدست آمده است انتخاب می‌کنیم و سپس آیدی آن کلمه جزء بافتار و آیدی این ۴ کلمه منفی را در یک آرایه قرار می‌دهم. در نتیجه این تابع بسته‌ای از داده‌ها شامل یک داده هدف و آرایه‌ای که عضو اول آن کلمه مثبت و بقیه کلمات منفی هستند برمی‌گرداند.
ضمنا توجه داریم که برچسب اولین داده ۱ و بقیه صفر است و بنابراین در خروجی این تابع برچسب را برمی‌گردانم.

```
def get_data(batch_size, sentences, window_size, ns_size=5):
  target = []
  context = []

```

```
  for s in sentences:
    ps = np.random.random(size=len(s))
    encoded = [voc.get(t, 0) for t in s]
    for i in range(len(encoded)):
      start = max(0, i-window_size)
      end = min(i+window_size+1, len(encoded))
      for j in range(start, end):
        if j != i:
            target.append(encoded[i])
            context.append(np.append( np.array([encoded[j]]), ns_table_expanded[
   np.random.randint(len(ns_table_expanded), size=ns_size)] ))
        if len(target) == batch_size:
          yield target, context
          target = []
          context = []
  yield target, context
```

۲.۲ آموزش مدل

ابتدا توابع کمکی زیر را برای محاسبه تابع هزینه و گرادیان تعریف کردم:

```
def sigmoid(x):
  return 1/(1+np.exp(-x))

def log_sigmoid(x):
  return np.log(1/(1+np.exp(-x)))
```

۱.۲.۲ تابع هزینه

رابطه مربوط به تابع هزینه به صورت زیر است:

$$L_{CE} = -\left[log\sigma(c_{pos}.w) + \sum_{i=1}^{k} log\sigma(-c_{neg_i}.w)\right]$$

برای پیاده سازی این تابع هزینه با توجه به ساختار داده‌ها و برچسبها ابتدا مقدار $(-1)^{(1-y)}$ را محاسبه کردم. به این ترتیب حاصل برای داده اول ۱ و برای بقیه منفی ۱ خواهد بود.

سپس این بردار را تغییر شکل داده و به اندازه ابعاد امبدینگ مدنظر مقادیر را برای ضرب کردن در بردار مربوط به

توکن‌ها تکرار کردم. بنابراین برای یک بسته از برچسب‌های داده مراحل انجام کار به صورت روابط زیر خواهد بود:

$$y = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ & & \cdots & & \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \Rightarrow$$

$$(-1)^{(1-y)} = \begin{pmatrix} 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 & -1 \\ & & \cdots & & \\ 1 & -1 & -1 & -1 & -1 \end{pmatrix}$$

$$\begin{pmatrix} \begin{pmatrix} \begin{pmatrix} 1 \end{pmatrix} & \begin{pmatrix} -1 \end{pmatrix} & \begin{pmatrix} -1 \end{pmatrix} & \begin{pmatrix} -1 \end{pmatrix} \end{pmatrix} \\ \begin{pmatrix} \begin{pmatrix} 1 \end{pmatrix} & \begin{pmatrix} -1 \end{pmatrix} & \begin{pmatrix} -1 \end{pmatrix} & \begin{pmatrix} -1 \end{pmatrix} \end{pmatrix} \\ \cdots \\ \begin{pmatrix} \begin{pmatrix} 1 \end{pmatrix} & \begin{pmatrix} -1 \end{pmatrix} & \begin{pmatrix} -1 \end{pmatrix} & \begin{pmatrix} -1 \end{pmatrix} \end{pmatrix} \end{pmatrix}$$

$$\begin{pmatrix} \begin{pmatrix} \begin{pmatrix} 1 & 1 & \dots & 1 \end{pmatrix} & \begin{pmatrix} -1 & -1 & \dots & -1 \end{pmatrix} & \begin{pmatrix} -1 & -1 & \dots & -1 \end{pmatrix} & \begin{pmatrix} -1 & -1 & \dots & -1 \end{pmatrix} \end{pmatrix} \\ \begin{pmatrix} \begin{pmatrix} 1 & 1 & \dots & 1 \end{pmatrix} & \begin{pmatrix} -1 & -1 & \dots & -1 \end{pmatrix} & \begin{pmatrix} -1 & -1 & \dots & -1 \end{pmatrix} & \begin{pmatrix} -1 & -1 & \dots & -1 \end{pmatrix} \end{pmatrix} \\ \cdots \\ \begin{pmatrix} \begin{pmatrix} 1 & 1 & \dots & 1 \end{pmatrix} & \begin{pmatrix} -1 & -1 & \dots & -1 \end{pmatrix} & \begin{pmatrix} -1 & -1 & \dots & -1 \end{pmatrix} & \begin{pmatrix} -1 & -1 & \dots & -1 \end{pmatrix} \end{pmatrix} \end{pmatrix}$$

این کار با استفاده از کد زیر انجام می‌شود:

```
np.power(-1, 1-y).reshape(batch_size,1+neg_sample, 1).repeat(embed_size,axis=-1)
```

در مرحله بعد با ضرب این ماتریس در امبدینگ مربوط به $[c_{pos}, c_{neg_1}, \dots, c_{neg_k}]$ و ضرب اینها در بردار مربوط به امبدینگ کلمه هدف مقادیر مورد نظر در تابع هزینه تولید و با اعمال تابع $log(sigmoid)$ مقدار هزینه بدست می‌آید:

```
def loss(t, c, y, batch_size):
  n_c = np.power(-1, 1-y).reshape(batch_size,1+neg_sample, 1).repeat(embed_size,
    axis=-1)*c.reshape(batch_size,(1+neg_sample),embed_size)
```

```
  return -np.sum(log_sigmoid(np.sum(t.repeat(1+neg_sample, axis=0).reshape(
    batch_size,(1+neg_sample),embed_size)*n_c,axis=-1)),axis=-1)
```

۲.۲.۲ ایجاد مدل و محاسبه گرادیان

با استفاده از کد زیر مقادیر اولیه و ماتریس‌ها مدل مقدار دهی اولیه می‌شود.

```
epochs = 10
batch_size = 128
embed_size = 100
neg_sample = 4
lr = 0.001
T = np.random.random(size=(len(ns_table), embed_size))
C = np.random.random(size=(len(ns_table), embed_size))
```

روابط مربوط به محاسبه گرادین به صورت زیر است:

$$\frac{\partial L_{CE}}{\partial c_{pos}} = (\sigma(c_{pos}.w) - 1))w$$
$$\frac{\partial L_{CE}}{\partial c_{neg}} = \sigma(c_{neg}.w)w$$
$$\Rightarrow \frac{\partial L_{CE}}{\partial c} = (\sigma(c.w) - y)w$$
$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos}.w) - 1]c_{pos} + \sum_{i=1}^{k}[\sigma(c_{neg_i}.w)]c_{neg_i}$$
$$\Rightarrow \frac{\partial L_{CE}}{\partial w} = \sum[\sigma(c.w) - y]c$$

این مقادیر طبق روابط زیر محاسبه و سپس وزنها بر اساس این گرادیان‌ها به روز می‌شوند:

```
c_diff = (sigmoid(np.sum(T[t].repeat(1+neg_sample, axis=0).reshape(
    epoch_batch_size,(1+neg_sample),embed_size)*r_c,axis=-1)) - y).reshape(
    epoch_batch_size,1+neg_sample,1).repeat(embed_size,axis=-1)*T[t].repeat(1+
    neg_sample,axis=0).reshape(epoch_batch_size,1+neg_sample,embed_size)

t_diff = np.sum((sigmoid(np.sum(T[t].repeat(1+neg_sample, axis=0).reshape(
    epoch_batch_size,(1+neg_sample),embed_size)*r_c,axis=-1)) - y).reshape(
    epoch_batch_size,1+neg_sample,1).repeat(embed_size,axis=-1)*r_c, axis=1)
```

بنابراین فرآیند آموزش به صورت زیر خواهد بود:

```
for i in range(epochs):
  print("epoch {}:".format(i+1), end=" ")
  epoch_loss = 0
  bT = np.copy(T)
  bC = np.copy(C)
  for t, c in tqdm(get_data(batch_size, sentences, 5, neg_sample)):
    epoch_batch_size = len(t)
    y= np.array([[1]+[0]*neg_sample]*epoch_batch_size)
    #if epoch_batch_size == batch_size:
    epoch_loss = (epoch_loss * i + np.mean(loss(T[t], C[c], y, epoch_batch_size)
   ))/(i+1)
    r_c = C[c].reshape(epoch_batch_size,(1+neg_sample),embed_size)
    c_diff = (sigmoid(np.sum(T[t].repeat(1+neg_sample, axis=0).reshape(
   epoch_batch_size,(1+neg_sample),embed_size)*r_c,axis=-1)) - y).reshape(
   epoch_batch_size,1+neg_sample,1).repeat(embed_size,axis=-1)*T[t].repeat(1+
   neg_sample,axis=0).reshape(epoch_batch_size,1+neg_sample,embed_size)
    t_diff = np.sum((sigmoid(np.sum(T[t].repeat(1+neg_sample, axis=0).reshape(
   epoch_batch_size,(1+neg_sample),embed_size)*r_c,axis=-1)) - y).reshape(
   epoch_batch_size,1+neg_sample,1).repeat(embed_size,axis=-1)*r_c, axis=1)
    C[c] -= lr*c_diff.reshape(epoch_batch_size,(1+neg_sample),embed_size)
    T[t] -= lr*t_diff
  print("loss: {:.3f}".format(epoch_loss))
  if bLoss < epoch_loss:
    print("early stop")
    T = bT
    C = bC
    break
  bLoss = epoch_loss
```

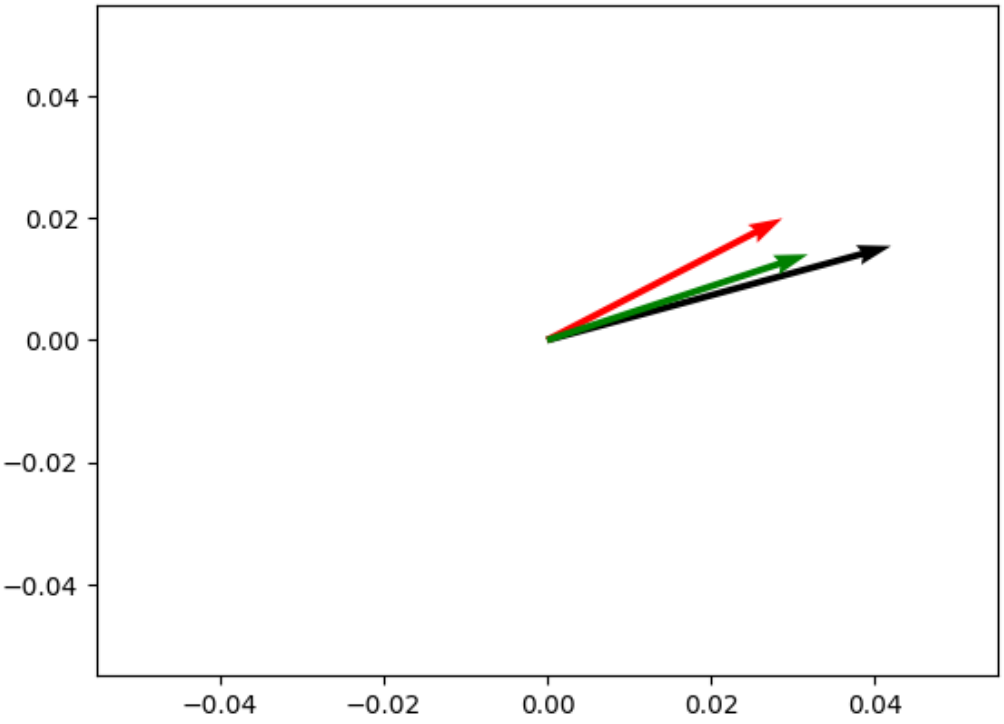
۳.۲ نتایج

در مرحله بعد دو ماتریس را با هم جمع کرده و بردارهای گفته شده در سوال را رسم کردم.

```
V = T+C
from sklearn.decomposition import PCA
pca = PCA(n_components=2, random_state=12)
n_V = pca.fit_transform(V)
```

```
km = n_V[[voc.get("king", 0)]] - n_V[[voc.get("man", 0)]]
qw = n_V[[voc.get("queen", 0)]] - n_V[[voc.get("woman", 0)]]
bs = n_V[[voc.get("brother", 0)]] - n_V[[voc.get("sister", 0)]]
data = np.array([km[0] , qw[0], bs[0]])
origin = np.array([[0, 0, 0], [0, 0, 0]])
plt.quiver(*origin, data[:, 0], data[:, 1], color=['black', 'red', 'green'],
    scale=1.5)
```

بردار سیاه تفاضل $king - man$ و بردار قرمز بردار $queen - woman$ و بردار سبز بردار $brother - sister$ را نشان می‌دهد.



شکل ۵: نمایش بردارهای تفاضل کلمات

علاوه بر ضمیمه کردن کدها، تمام موارد در کولب هم قابل دسترس است.

۳ سوال سوم

۱.۳ آماده سازی بردار کلمات

ابتدا فایل وزنهای $glove$ را دریافت و آنها را برای هر کلمه با کد زیر به یک بردار تبدیل کردم. ضمنا در انتهای تمام بردارها مقدار ۱ را نیز برای محاسبه بایاس اضافه کردم.

```
embed_size = 300
glove = defaultdict()
with open("glove.6B."+(str)(embed_size)+"d.txt", "r") as f:
  for line in f.readlines():
    l = line.split()
    glove[l[0]] = np.array([(float)(x) for x in l[1:]]+[1]) # 1 is for bias
```

۲.۳ مدل

ابتدا توابع کمکی زیر را برای محاسبه تابع هزینه و صحت عملکرد مدل تعریف کردم.

```
def softmax(x):
  return np.exp(x)/(np.sum(np.exp(x), axis=-1).repeat(x.shape[-1], axis=-1).
    reshape(x.shape))

def loss(y_true, y_pred):
  return np.mean(np.sum(-y_true*np.log(y_pred),axis=-1))

def accuracy(y_true, y_pred):
  return sum([(int)(p) for p in np.argmax(y_pred, axis=-1) == np.argmax(y_true,
    axis=-1)])/len(y_true)
```

تابع هزینه مورد استفاده تابع هزینه $categorical \quad cross \quad entropy$ است. رابطه مربوط به این تابع هزینه به صورت زیر است:

$$L_{CE} = -\sum y_i log(\hat{y}_i)$$

با توجه به اینکه یک بعد به بردار ویژگیها برای بایاس اضافه کردیم، روابط ریاضی مربوط به مدل به صورت زیر خواهد بود:

$$\mathbf{x} = (x_1, x_2, \ldots, x_{emd}, 1)$$
$$\mathbf{z} = \mathbf{x} \times \mathbf{w}^{\mathbf{T}}$$
$$\hat{\mathbf{y}} = softmax(\mathbf{z})$$

برای محاسبه گرادیان تابع هزینه نسبت به وزنهای $\mathbf{w}$ به صورت زیر عمل می‌کنیم:

$$\frac{\partial L_{CE}}{\partial \mathbf{w}} = \frac{\partial L_{CE}}{\partial \hat{\mathbf{y}}} \times \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}} \times \frac{\partial \mathbf{z}}{\mathbf{w}}$$

با محاسبه این روابط و ساده کردن آنها به رابطه زیر می‌رسیم:

$$\frac{\partial L_{CE}}{\partial \mathbf{w}} = (\hat{\mathbf{y}} - \mathbf{y}) \times \mathbf{x}$$

با توجه به این روابط مدل به صورت زیر تعریف شده است:

```
class logisticmodel():
  def __init__(self, n_features, n_categories):
    self.W = np.random.random((n_categories, n_features+1))

  def forward(self, X):
    self.X = X
    self.pred = softmax((X@self.W.T))
    return self.pred
  def backpropagate(self, y_true, lr):
    self.W = self.W - lr * (self.pred - y_true).T@self.X
  def train(self, X, y, X_val, y_val, batch_size, epochs, lr):
    history = {"train_loss": [], "val_loss": [], "train_acc": [], "val_acc":[]}
    for i in range(epochs):
      for b in range((int)(len(X)/batch_size)):
        pred = self.forward(X[b*batch_size:min((b+1)*batch_size, len(X))])
        self.backpropagate(y[b*batch_size:min((b+1)*batch_size, len(X))], lr)
      if X_val:
        pred = self.forward(X_val)
        history["val_loss"].append(loss(y_val, pred))
```

```
        history["val_acc"].append(accuracy(y_val, pred))
      pred = self.forward(X)
      history["train_loss"].append(loss(y, pred))
      history["train_acc"].append(accuracy(y, pred))

      if (i+1)%100 == 0 or i == 0:
        if X_val:
          print("epoch {:4} loss is: {:.3f} - accuracy is: {:.3f} - val_loss is: {:.3f} - val_acc is: {:.3f}".format(i+1, history["train_loss"][-1], history["train_acc"][-1]*100, history["val_loss"][-1], history["val_acc"][-1]*100))
        else:
          print("epoch {:4} loss is: {:.3f} - accuracy is: {:.3f}".format(i+1, history["train_loss"][-1], history["train_acc"][-1]*100))
    return history
  def test(self, X, y):
    pred = self.forward(X)
    print("============================================================")
    print("                      Test Report                           ")
    print("============================================================")
    print(classification_report(np.argmax(y_test, axis=-1), np.argmax(pred, axis=-1)))
```

۳.۳ داده‌ها

با استفاده از قطعه کد زیر داده‌ها دریافت به سه قسمت آموزش، اعتبارسنجی و تست تقسیم شدند. ضمنا برچسب ها نیز در این مرحله به *one hot* تبدیل شدند.

```
data= pd.read_csv("all-data.csv",  on_bad_lines="skip", encoding="ISO-8859-1", header=None)


X_train, X_test, y_train, y_test = train_test_split(data[1], data[0], test_size=0.1)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.1)

onehot = OneHotEncoder()
```

```
onehot.fit(np.array(y_train).reshape(-1,1))
y_train = onehot.transform(np.array(y_train).reshape(-1,1)).toarray()
y_val = onehot.transform(np.array(y_val).reshape(-1,1)).toarray()
y_test = onehot.transform(np.array(y_test).reshape(-1,1)).toarray()
```

با استفاده از قطعه کد زیر و مشابه کاری که در سوال اول انجام شد، هر کدام از داده‌ها به یک بردار تبدیل شد.

```
def sentence2vec(sentence):
  vec = []
  stop = nltk.corpus.stopwords.words('english')
  for w in word_tokenize(sentence.lower()):
    v = glove.get(w, glove.get('unk'))
    vec.append(v)
  vec = np.array(vec)
  return np.mean(vec, axis=0)

X_train = [sentence2vec(x) for x in X_train]
X_val = [sentence2vec(x) for x in X_val]
X_test = [sentence2vec(x) for x in X_test]
```

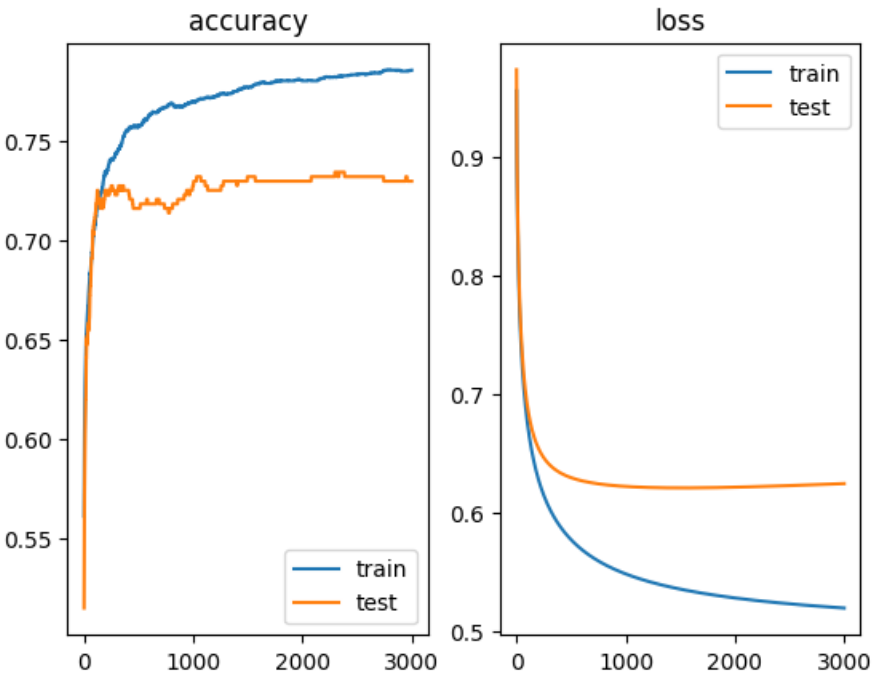
۴.۳ آموزش و تست

با استفاده از قطعه کد زیر مدل آموزش داده شد که نمودار تابع هزینه و صحت مدل نیز در شکل ۶ نمایش داده شده است.

```
batch_size = 64
categories = 3
epochs = 3000
lr = 5e-4

model = logisticmodel(embed_size, categories)
history = model.train(X_train, y_train, X_val, y_val, batch_size, epochs, lr)
fig, ax = plt.subplots(ncols=2)
ax[0].plot(history["train_acc"], label="train")
ax[0].plot(history["val_acc"], label="test")
ax[0].legend()
ax[0].set_title("accuracy")
ax[1].plot(history["train_loss"], label="train")
```

```
ax[1].plot(history["val_loss"], label="test")
ax[1].legend()
ax[1].set_title("loss")
plt.show()
model.test(X_test, y_test)
```



شکل ۶: نمودار تغییر تابع هزینه و صحت عملکرد مدل در حین فرایند آموزش

نهایتا نتایج روی داده‌های تست به صورت شکل ۷ شد.

```
                    Test Report
              precision    recall  f1-score   support

           0       0.64      0.52      0.57        62
           1       0.74      0.82      0.78       283
           2       0.58      0.51      0.54       140

    accuracy                           0.69       485
   macro avg       0.65      0.61      0.63       485
weighted avg       0.68      0.69      0.68       485
```

شکل ۷: نتایج مدل logistic regression

همانطور که نتایج هم پیداست عملکرد روی داده‌هایی که نمونه بیشتر دارند بهتر و روی داده‌های با نمونه‌های کمتر ضعیف‌تر است.

۵.۳ مقایسه با مدل بیز

با استفاده از قطعه کد زیر همین کار با استفاده از مدل بیز هم انجام شد و نتایج در شکل ۸ نشان داده شده است.

```
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(X_train, [np.argmax(y) for y in y_train])
print(classification_report(np.argmax(y_test, axis=-1), clf.predict(X_test)))
```

	precision	recall	f1-score	support
0	0.29	0.47	0.36	62
1	0.70	0.84	0.76	283
2	0.43	0.14	0.21	140
accuracy			0.59	485
macro avg	0.47	0.48	0.44	485
weighted avg	0.57	0.59	0.55	485

شکل ۸: نتایج مدل naive bayes

همانطور که مشخص است نتایج حاصل از مدل بیز ضعیف‌تر است که به نظر می‌رسد با توجه به غیر متوازن بودن داده‌ها و فرض استقلال ویژگی‌ها و ابعاد بالای بردار ویژگی این عملکرد ضعیف‌تر منطقی باشد.

علاوه بر ضمیمه کردن کدها، تمام موارد در کولب هم قابل دسترس است.

کلیه موارد مربوط به این تمرین در گیت هاب نیز در دسترس است.