بسم الله الرحمن الرحیم

گزارش کار تمرین کامپیوتری سوم درس هوش مصنوعی

مهدی فرهنگ - ۸۱۰۱۹۵۴۴۶

ابتدا به توضیح بخش های مختلف code میپردازیم. سپس توضیحاتی را که نیاز است درباره ی عملکرد و تصمیماتی که برای حل سوالات گرفته ایم را خواهیم آورد.

توضیحات بخش منطقی code

Getting_data:

```
10 def getting_data(file_name, first):
11     temp = pd.read_csv(file_name)
12     data = []
13     deleting_words = list(stopwords.words('english'))
14     deleting_words.extend(['.', ',', '..', '...', '?'])
15     ps = PorterStemmer()
16 
17     for i in range(len(temp)):
18         words = nltk.word_tokenize(temp['text'][i])
19         filtered_words = []
20         for word in words:
21             if (ps.stem(word) not in deleting_words) and (ps.stem(word) not in filtered_words):
22                 filtered_words.append(ps.stem(word))
23         data.append([temp[first][i], filtered_words])
24     return data
```

در این تابع، هدف این است که از ورودی اطلاعات را خوانده، normalize کرده، و تمامی داده هایی را که خواندیم، با نوع آن در مواقعی که میخواهیم داده هایمان را train کنیم (spam یا ham بودن) و یا با شماره آن در مواقعی که میخواهیم نوع تعدادی داده را بررسی کنیم، در یک لیست به عنوان خروجی تابع به برنامه بازگردانیم. یعنی در نهایت یک لیست دریافت خواهیم کرد که هر سطر آن دو عضو دارد، عضو اول نوع یا شماره است، عضو دوم لیستی از کلمات جمله ی ایمیل است. که همه normalize شده اند و کلمات تکراری نیز حذف شده اند.

Possibility of word:

```
26 def possibility_of_word(data, word):
27     num_of_spam = 0
28     num_of_ham = 0
29     num_of_word_in_spams = 0
30     num_of_word_in_hams = 0
31     for item in data:
32         if (item[0] == 'spam'):
33             num_of_spam += 1
34             num_of_word_in_spams += 1 if (word in item[1]) else 0
35         else:
36             num_of_ham += 1
37             num_of_word_in_hams += 1 if (word in item[1]) else 0
38     return float((num_of_word_in_spams if num_of_word_in_spams != 0 else 0.000001) / num_of_spam), float((num_of_word_in_h
39 
```

در این تابع، میخواهیم با گرفتن یک کلمه، و البته با گرفتن همه ی داده های تمرین، بررسی کنیم که چه مقدار این کلمه در داده های spam استفاده شده است و چه مقدار در داده های ham. یعنی میخواهیم ببینیم اگر در موقع تشخیص این کلمه را دیدیم، بدانیم این کلمه با چه احتمالی در ایمیل های spam به کار میرود و با چه احتمالی در ایمیل های ham. برای این کار بر روی تمامی ایمیل ها بررسی میکنیم. اگر آن کلمه در ایمیل مربوطه وجود داشت، تعداد وجود این کلمه در این نوع ایمیل را یک عدد اضافه میکنیم. و در نهایت تعداد تکرار کلمه را به تعداد ایمیل های آن نوع (برای هر دو نوع این کار را میکنیم) تقسیم میکنیم. توجه داشته باشیم که اگر این مقدار صفر بود، یعنی کلمه مربوطه در هیچ کدام از انواع داده ها وجود نداشت، مقدار صفر را برنمیگردانیم. زیرا در تابع train که از این تابع استفاده میکند میخواهیم لگاریتم این مقدار را محاسبه کنیم. و برای این کار نباید مقدار صفر باشد. پس یک مقدار اندک را برمیگردانیم.

Train:

```
40  def train(data):
41      spams = {}
42      hams = {}
43      num_of_spam_in_size = [0] * 80
44      num_of_ham_in_size = [0] * 80
45      i = 0
46      for sentence in data:
47          for word in sentence[1]:
48              if word not in spams:
49                  a, b = possibility_of_word(data[0:num_of_training_data], word)
50                  spams[word] = a
51                  hams[word] = b
52          if (i > num_of_training_data):
53              continue
54          if (sentence[0] == 'spam'):
55              num_of_spam_in_size[len(sentence[1])] += 1
56          else:
57              num_of_ham_in_size[len(sentence[1])] += 1
58          i += 1
59      return spams, hams, num_of_spam_in_size, num_of_ham_in_size
60
```

این تابعی است که در آن برنامه را با کمک داده ها آموزش میدهیم. دو دیکشنری در نظر میگیریم. یکی برای بررسی احتمال وقوع هر کلمه در ایمیل های spam و دیگری برای ایمیل های ham. همچنین دو آرایه به طول ۸۰ نیز در نظر میگیریم. یکی برای بررسی تعداد تکرار ایمیل های با اندازه های مختلف (که index لیست اندازه را مشخص میکند) در ایمیل های spam و دیگری در ایمیل های ham. حال شروع به بررسی در تک تک کلمات در تک تک ایمیل ها میکنیم. اگر کلمه تکراری نبود (در خط ۴۸ لحاظ شده است)، احتمال رخداد کلمه در ایمیل های spam و ham را با استفاده از تابع قبلی محاسبه میکنیم و در دیکشنری تابع را اضافه میکنیم. پس از بررسی تمام کلمات در یک ایمیل خاص، تعداد کلمات در آن را شمرده و به ایمیل های با همان تعداد کلمه در همان نوع ایمیل یکی اضافه میکنیم.

نکته ای که قابل توجه است این است که در اینجا باید دقت کنیم که برای رخ ندادن overfit، باید درصدی از ایمیل ها را train کنیم. فلذا توجه داریم که تعداد مشخصی از داده ها را برای این کار بررسی کنیم.(خط ۴۹ و ۵۲ تا ۵۳)

Guess type of sentence:

حال که برنامه ی خود را آموزش دادیم، میخواهیم بررسی کنیم که یک ایمیل از چه نوعی است.(spam or ham)

```
61 def guess_type_of_sentence(spams, hams, words, num_of_spams, num_of_hams, spam_sizes, ham_sizes):
62     total = num_of_spams + num_of_hams
63     p_all_spams = math.log(num_of_spams / total, 2)
64     p_all_hams = math.log(num_of_hams / total, 2)
65     for word in words:
66         if word in spams:
67             p_all_spams += math.log(spams[word], 2)
68         if word in hams:
69             p_all_hams += math.log(hams[word], 2)
70     temp = len(words)
71     p_all_spams += temp / 3 * math.log(((spam_sizes[temp] if spam_sizes[temp] != 0 else 0.000001) / num_of_spams), 2)
72     p_all_hams += temp / 3 * math.log(((ham_sizes[temp] if ham_sizes[temp] != 0 else 0.000001) / num_of_hams), 2)
73     if (p_all_spams - p_all_hams > math.log(2, 2)):
74         return 'spam'
75     return 'ham'
```

در این تابع، جمله را، و تمامی خروجی های تابع آموزش را به عنوان ورودی به این تابع میدهیم.

اعداد p_all_hams و p_all_spmas اعدادی هستند که احتمال وجود تک تک کلمات جمله، احتمال spam یا ham بودن ایمیل به تنهایی و یا به شرط تعداد کلمات ایمیل و . . . را بررسی میکنند. یعنی این دو عدد، نتیجه در فرمول زیر هستند:

$$\hat{x} = \operatorname{argmax}_x P(x \mid e) \propto \underbrace{P(x)}_{\text{prior}} \underbrace{\prod_{i=1}^{n} P(e_i \mid x)}_{\text{likelihood}}$$

posterior

که مقداری که در ابتدا به آنها میدهیم، prior آنها هستند. و در ادامه به ترتیب likelihood را در آنها ضرب میکنیم.

توجه داریم که از آنجا که اینها اعداد بسیار کوچکی را به ما تحویل میدهند، با لگاریتم آنها کار میکنیم.

حال به ازای تمامی کلمات ایمیل، اگر کلمه در داده های تمرینی ما در مجموعه spam ها و یا ham ها دیده شده بود و احتمال وقوع آن در نوع خاص ایمیل را میدانستیم، نتیجه را در مقدار متغیر های p لحاظ میکنیم. (تا انتهای خط ۶۹)

در اینجا به سراغ نوع دوم تشخیص ایمیل که تعداد کلمات آن ایمیل است آمده و احتمال رخداد آن ها را نیز در متغیر های خود ذخیره میکنیم.

تذکر: میدانیم که تمامی این داده ها در هم ضرب میشوند. و اگر بخواهیم تاثیر طول جمله را به اندازه تاثیر یک کلمه در جمله قرار دهیم شاید منطقی نباشد. پس به مقدار آن توان میدهیم. (که در حالت لگاریتم گرفته شده باید به آن ضریب بدهیم). با سعی و خطا مقدار یک سوم طول جمله برای این توان مقدار مناسبی به نظر آمد.

در نهایت، بر اساس احتمال spam بودن یک ایمیل و یا ham بودن آن، باید تصمیم بگیریم. میتوانیم بگوییم احتمال هر کدام بیشتر بود، ایمیل را از همان نوع در نظر بگیر. اما چون مشکلات تشخیص یک ایمیل ham به عنوان ایمیل spam زیاد است، باید توجه داشته باشیم که نباید صرفا هر کدام بیشتر بود را انتخاب کنیم. با سعی و خطا به نتیجه ای که در خط ۷۳ مشاهده میکنید رسیدیم.

حال بخش های تصمیم گیری code ما به انتها رسیده. باید مقدار را تست کرده و اگر از نتیجه راضی بودیم، ایمیل ها را بررسی کنیم که آیا spam هستند یا ham.

توضیحات توابع test و check:

Test:

```
77  def test(data, spams, hams, spam_sizes, ham_sizes):
78      num_of_spam = 0
79      num_of_ham = 0
80      for item in data:
81          if (item[0] == 'spam'):
82              num_of_spam += 1
83          else:
84              num_of_ham += 1
85
86      result = []
87      for i in range(num_of_training_data, len(data)):
88          result.append(guess_type_of_sentence(spams, hams, data[i][1], num_of_spam, num_of_ham, spam_sizes, ham_sizes))
89
90      total = len(result)
91      true_spam = 0
92      true_ham = 0
93      false_spam = 0
94      false_ham = 0
95
96      for i in range(total):
97          if (data[num_of_training_data + i][0] == 'spam'):
98              if (result[i] == 'spam'):
99                  true_spam += 1
100             else:
101                 false_spam += 1
102         else:
103             if (result[i] == 'ham'):
104                 true_ham += 1
105             else:
106                 false_ham += 1
107     print('number of test data = ', total)
108     print('Recall = ', true_spam / (true_spam + false_spam))
109     print('Precision = ', true_spam / (true_spam + false_ham))
110     print('Accuracy = ', (true_spam + true_ham) / total)
111     # print(true_spam, true_ham, false_spam, false_ham)
```

این تابع، به ازای تمامی ۴۲۰۰ تا از ورودی ها مقدار train را میگیرد (از تابع train) و به ازای ۸۷۲ ایمیل بعدی مقدار را تست میکند. این کار برای جلوگیری از رخداد overfit و برای بررسی دقت تابع انجام میشود.

به ازای تمامی ورودی ها، بررسی میکنیم که آیا spam ها را درست تشخیص داده ایم یا نه و یا ham ها را درست تشخیص داده ایم یا نه.

در نهایت، مقادیر Recall, Precision, Accuracy را بر اساس spam های درست تشخیص داده شده، ham های درست تشخیص داده شده، spam های اشتباه تشخیص داده شده و ham های اشتباه تشخیص داده شده به دست می آوریم و برروی خروجی چاپ میکنیم.

Check:

```
113 def check(input_file_name, output_file_name, spams, hams, spam_sizes, ham_sizes):
114     evaluate_data = getting_data(input_file_name, 'id')
115     result = [0] * len(evaluate_data)
116     for i in range(len(evaluate_data)):
117         result[i] = guess_type_of_sentence(spams, hams, evaluate_data[i][1], sum(spam_sizes), sum(ham_sizes), spam_sizes,
118     # know that in the test, number of spams and sum of spam_sizes are diffrent.
119     # that is why we didn't calculate sum of spam_sizes in the function itself
120
121     f = open(output_file_name, "w")
122     f.write("id,type\n")
123     for i in range(len(result)):
124         f.write("%d," % (i + 1))
125         f.write((result[i]) + '\n')
126     f.close()
```

این تابع نیز از ورودی یک سری ایمیل که نوع آن را نمیدانیم میگیرد، تمامی آن ها را تشخیص میدهد و در یک فایل خروجی را چاپ میکند. روند انجام آن واضح است و نیاز به توضیح خاصی ندارد.

توضیح overfit:

این مشکل در زمانی پیش می آید که در زمان آموزش برنامه، برنامه بر اساس یک خصلت مشترک که در داده های یکسان و از یک نوع (در مثال ما spam یا ham بودن ایمیل) وجود دارد تصمیم میگیرد که آیا داده ما این ویژگی را دارد یا نه. در حالی که تصمیم گیری بر اساس خصلت مربوطه اشتباه است. برای مثال در کار ما، اگر برنامه بر اساس فونت و یا اندازه نوشته ها تصمیم بگیرد که ایمیل از چه نوعی است، اشتباه است.

برای این که این مشکل رخ ندهد، داده های تمرین را به دو بخش تقسیم میکنیم. بر اساس اکثریت آنها، برنامه را آموزش میدهیم و با بقیه ی داده ها، تست میکنیم که آیا درست تشخیص میدهیم یا نه. اگر نتیجه تشخیص ما مقدار خوبی نبود، یعنی که در تشخیص و آموزش داده ها overfit رخ داده است و باید آن را حل کنیم.

اجرای برنامه و نتایج به دست آمده بر روی code ما:

حال برنامه را اجرا میکنیم.

ابتدا آنرا آموزش میدهیم.

```
131  t1 = time()
132  data = getting_data('train_test.csv', 'type')
133  # if we were testing our train model, the line below should be commented
134  # num_of_training_data = len(data)
135  spams, hams, spam_sizes, ham_sizes = train(data)
136  t2 = time()
137  print('training time = ', t2 - t1)
```

کار خاص انجام نشده است و نیاز به توضیح ندارد. صرفا داده ها را از ورودی خواندیم و برنامه را آموزش دادیم. مدت زمان این مرحله معمولا در حدود ۱۰ ثانیه انجام میشود.

حال داده ها را تست میکنیم. میخواهیم ببینیم که آیا به مقدار خوبی این کار را انجام میدهد و یا خیر. و این که آیا overfit رخ داده است و یا خیر

```
139  # for testing our training model.
140
141  t1 = time()
142  test(data, spams, hams, spam_sizes, ham_sizes)
143  t2 = time()
144  print('testing time = ', t2 - t1)
```

حال برنامه را اجرا میکنیم و خروجی به این صورت خواهد بود:

```
training time =  10.905410051345825
number of test data =  872
Recall =  0.9292035398230089
Precision =  0.9375
```

Accuracy = 0.9827981651376146

testing time = 0.0064885616302490234

و این نتایج برای ما قابل قبول است. پس میتوانیم از این برنامه استفاده کنیم.

حال برنامه را با داده های خواسته شده تست میکنیم.

```
146 t1 = time()
147 check('evaluate.csv', 'out.csv', spams, hams, spam_sizes, ham_sizes)
148 t2 = time()
149 print('checking time = ', t2 - t1)
150
```

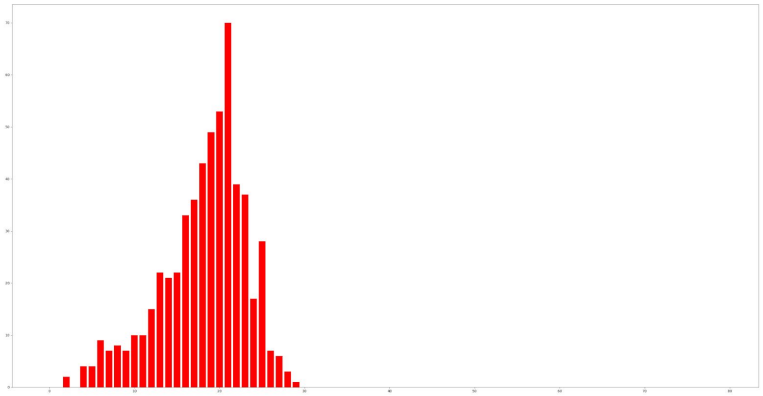
نتایج به صورت زیر خواهد بود:

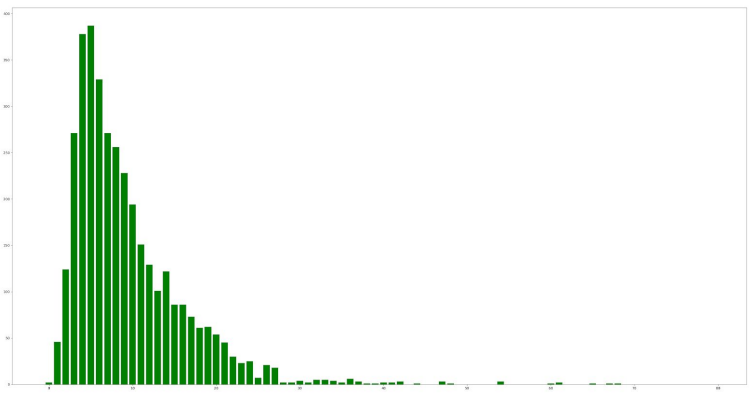
training time = 11.964723110198975

checking time = 0.3329341411590576

خروجی برنامه نیز در فایل out.csv نوشته شده است که به همراه گزارش آپلود شده است.

برای این که ثابت کنیم که استفاده از خصیصه ای مثل طول پیام گزینه ی مناسبی است، از نمودار های زیر استفاده میکنیم:





نمودار سبز رنگ، تعداد تکرار ایمیل های با طول های مشخص در ایمیل های ham و نمودار قرمز رنگ مربوط به ایمیل های spam است. این تفاوت در نمودار ها نشان میدهد که استفاده از ویژگی طول نمودار کار درستی است.

کد های مربوط به این نمودار ها در تابع train اضافه شده اند. به صورت زیر

```
    x = [0] * 80
    for i in range(80):
        x[i] = i
    plt.bar(x, num_of_spam_in_size, color = 'red')
    plt.show()
    plt.bar(x, num_of_ham_in_size, color = 'green')
    plt.show()
```