تمرین دوم

یاسمن لطف‌اللهی - ۹۷۵۲۱۴۸۶

۱. Cross validation

از راهکارهای جلوگیری از overfit شدن مدل، استفاده از Cross validation است. cross validation انواع مختلف دارد اما نحوه کار آن ها به طور کلی به نحو زیر است:

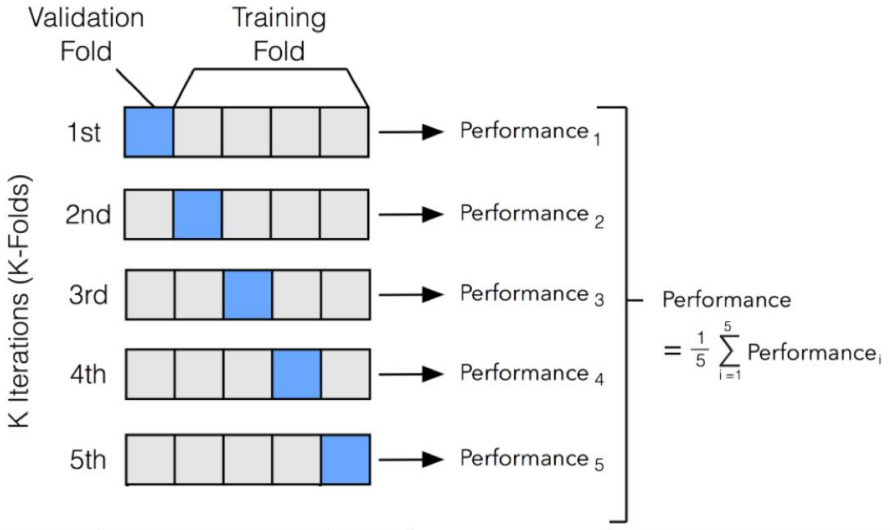
- تقسیم داده به چند زیرمجموعه
- جدا کردن بخشی از داده به عنوان hold out در مرحله و آموزش روی بقیه داده
- ارزیابی روی داده جدا کرده

این ارزیابی به صورت کلی به دو دسته Non-Exhaustive و Exhaustive تقسیم می شود.

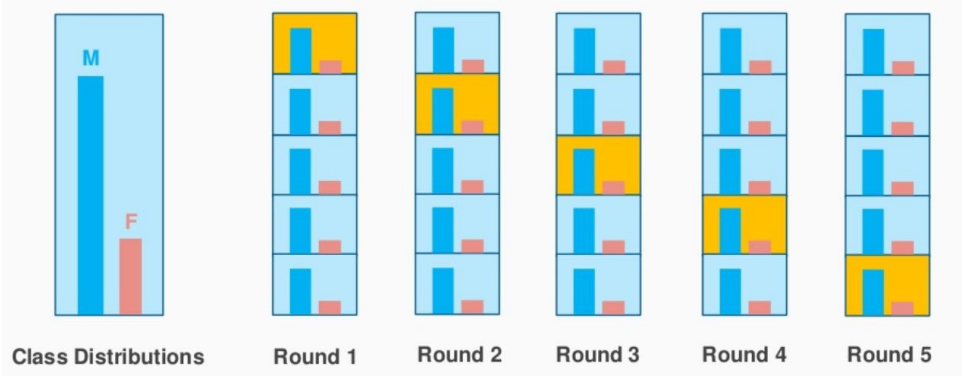
- Exhaustive: داده را به تمام روش های ممکن تقسیم می کند. از نمونه این دسته: Leave one out Cross-Validation (LOOCV).
  - **Leave one out Cross-Validation (LOOCV)**: آموزش به تعداد داده iterate می شود و در هر iteration یک نمونه از داده بیرون کشیده می شود.
  - **کاربرد**: این روش هزینه محاسباتی بالایی دارد و برای همین برای دیتاست های کوچک به کار می رود و برای حالتی که دیتاست بزرگی داریم مناسب نیست.

- Non- Exhaustive: تمام حالت های ممکن بررسی نمی شود، برای همین پیچیدگی محاسباتی کمتری دارند و برای دیتاست های بزرگ مناسب هستند. روش هایی که در این دسته قرار دارند: k-Fold Cross-Validation، Stratified K-Fold Cross Validation.

- **k-Fold Cross-Validation**: داده به هم ریخته می شود و به k دسته مساوی تقسیم می شود. در هر iteration یکی از این دسته ها به عنوان holdout خارج می شود و آموزش روی بقیه دسته ها انجام می شود.



- **Stratified K-Fold Cross Validation**: در روش قبلی، داده به هم ریخته می شد و تقسیم می شد. اما ممکن است دسته بندی ها متعادل نباشند و عملکرد مدل را تحت تاثیر قرار دهد. در این روش، برای حل این مشکل، در این داده جوری دسته بندی می شود که هر دسته نشان دهنده مناسبی از کل دیتاست باشد و دسته ها متعادل باشند. آموزش به شکل قبل است.



۲. مدل زبانی

الف) آموزش

- تعداد هر توکن:

<s>	3
A	10
B	11
</s>	3

- تعداد bigram ها:

	<s>	A	B	</s>
<s>	0	0	3	0
A	0	2	6	2
B	0	8	2	1
</s>	0	0	0	0

- نتیجه بدون add-1 smoothing:

	<s>	A	B	</s>
<s>	0	0	3/3=1	0
A	0	2/10=0.2	6/10=0.6	2/10=0.2
B	0	8/11=0.72	2/11=0.18	1/11=0.09
</s>	0	0	0	0

ب)

بدون Add-1 smoothing

- P(A|B) = 8/11 = 0.72
- P(B|<s>) = 3/3 = 1
- P(</s>|A) = 2/10 = 0.2

با add-1 smoothing:

- P(A|B) = 8+1/11+4 = 9/15 = 0.6
- P(B|<s>) = 3+1/3+4 = 4/7 = 0.5714
- P(</s>|A) = 2+1/10+4 = 3/14 = 0.2142

۳. قانون بیز

- P(y=7|x=7) = 0.5
- P(y=8|x=7) = 0.5
- P(y=7|x=8) = 0.3
- P(y=8|x=8) = 0.7
- P(x=7|y=7) = ?
- P(x=8|y=8) = ?

با قانون بیز داریم:

$$\mathrm{P}(\mathrm{x}=7|\mathrm{y}=7) = \frac{P(y=7|x=7)P(x=7)}{P(y=7)}$$

$$\mathrm{P}(\mathrm{x}=8|\mathrm{y}=8) = \frac{P(y=8|x=8)P(x=8)}{P(y=8)}$$

با توجه به صورت سوال، احتمال هر ورودی برابر است، درنتیجه P(x=7) = 0.1. چون P(x=0) با P(x=1) و پیشامدهای مشابه از هم مستقل اند، داریم:

$$\begin{aligned} P(y=7) = {} & P(x=0).P(y=7|x=0) + P(x=1).P(y=7|x=1) + P(x=2).P(y=7|x=2) + P(x \\ & =3).P(y=7|x=3) + P(x=4).P(y=7|x=4) + P(x=5).P(y=7|x=5) + P(x \\ & =6).P(y=7|x=6) + P(x=7).P(y=7|x=7) + P(x=8).P(y=7|x=8) + P(x \\ & =9).P(y=7|x=9) \\ & = 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 0.5 \\ & + 0.1 \times 0.3 + 0.1 \times 0 = 0.05 + 0.03 = 0.08 \end{aligned}$$

همچنین:

$$\begin{aligned} P(y=8) = {} & P(x=0).P(y=8|x=0) + P(x=1).P(y=8|x=1) + P(x=2).P(y=8|x=2) + P(x \\ & =3).P(y=8|x=3) + P(x=4).P(y=8|x=4) + P(x=5).P(y=8|x=5) + P(x \\ & =6).P(y=8|x=6) + P(x=7).P(y=8|x=7) + P(x=8).P(y=8|x=8) + P(x \\ & =9).P(y=8|x=9) \\ & = 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 0 + 0.1 \times 0.5 \\ & + 0.1 \times 0.7 + 0.1 \times 0 = 0.05 + 0.07 = 0.12 \end{aligned}$$

درنتیجه:

$$\mathrm{P}(\mathrm{x}=7|\mathrm{y}=7) = \frac{P(y=7|x=7)P(x=7)}{P(y=7)} = \frac{0.5 \times 0.1}{0.08} = 0.625$$

$$\mathrm{P}(\mathrm{x}=8|\mathrm{y}=8) = \frac{P(y=8|x=8)P(x=8)}{P(y=8)} = \frac{0.7 \times 0.1}{0.12} = 0.583$$

۴. دسته بند بیز

پیاده سازی

برای لود دیتاست از keras استفاده شده. با داده تعداد کلمات، خود کتابخانه توکن های unk را جایگزین کلماتی که فرکانس کمتری دارند، می کند.

```
from keras.datasets import imdb
INDEX_FROM = 3
(x_train, y_train), (x_test, y_test) = imdb.load_data(index_from=INDEX_FROM, num_words=5000)

word_index = imdb.get_word_index()
# Reverse the word index to obtain a dict mapping indices to words
word_index = {k:(v+INDEX_FROM) for k,v in word_index.items()}
word_index["<PAD>"] = 0
word_index["<START>"] = 1
word_index["<UNK>"] = 2
word_index["<UNUSED>"] = 3
inverted_word_index = dict((i, word) for (word, i) in word_index.items())

# Decode the first sequence in the dataset
print("Example of dataset: ")
print("Text: ", " ".join(inverted_word_index[i] for i in x_train[0]))
print("Sentiment: ", y_train.astype(bool)[0])
```

ابتدا داده را به متن تبدیل می کنیم و علائم نگارشی را حذف می کنیم:

```
train_text = []

# remove punctuation
for x in x_train:
  text = " ".join(inverted_word_index[i] for i in x[1:])
  text = text.translate(str.maketrans('', '', string.punctuation))
  train_text.append(text)
```

سپس با word tokenize توکن سازی می کنیم. با Snowball stemmer، stem هارا استخراج می کنیم و سپس کلمات stop words را حذف می کنیم.

```
train_tokens = []
for text in train_text:
  tokens = word_tokenize(text)
  # remove stop words
  stemmed_tokens=[(stemmer.stem(i)).lower() for i in tokens]
  filtered_tokens = [token for token in stemmed_tokens if token not in english_stopwords]
  train_tokens.append(filtered_tokens)
```

هنگام ساخت نمونه از کلاس classifier آموزش انجام می شود و احتمالات مورد نیاز محاسبه می شوند.

```
class classifier:
  def __init__(self, train_tokens, y, classes_n=2):
    self.classes_n = classes_n
    self.train_tokens = train_tokens
    self.y = y
    # self.p_ci[0] = neg
    # self.p_ci[1] = pos

    # creating vocab
    self.vocab, self.vocab_pos, self.vocab_neg = self.build_vocab()
    self.vocab["unk"]=0
    self.vocab_pos["unk"]=0
    self.vocab_neg["unk"]=0

    # number of words in each class
    self.ns = np.zeros((2,1))
    self.ns[1] = sum(self.vocab_pos.values())
    self.ns[0] = sum(self.vocab_neg.values())

    # index to word and word to index arrays
    self.itow = list(self.vocab.keys())
    self.wtoi = {w:idx for idx,w in enumerate(self.itow)}

    # calculating probabilities
    self.p_ci = np.zeros((classes_n,1))
    self.p_wk_cj = np.zeros((len(self.vocab), classes_n))

    self.find_p_ci()
    self.find_p_wk_cj()
```

ابتدا یک vocabulary کلی و یک vocabulary برای دسته منفی و یکی برای دسته مثبت ساخته می شود که تعداد تکرار کلمات را ذخیره می کند.

```
def build_vocab(self):
  vocab = dict()
  vocab_pos = dict()
  vocab_neg = dict()

  for i in range(len(self.train_tokens)):
    tokens = self.train_tokens[i]
    for token in tokens:
      # add token to general vocab
      n = vocab.get(token, 0)
      vocab[token] = n + 1
      # add token to its class
      if (self.y[i] == 0):
        n = vocab_neg.get(token, 0)
        vocab_neg[token] = n + 1
      else:
        n = vocab_pos.get(token, 0)
        vocab_pos[token] = n + 1

  return vocab, vocab_pos, vocab_neg
```

به کمک آن ها، احتمال تعلق داشتن هر کلمه به هر کلاس محاسبه می شود:

```
def find_p_wk_cj(self):
  words = self.itow

  # negative, j = 0
  for k in range(len(words)):
    w = words[k]
    self.p_wk_cj[k, 0] = ((self.vocab_neg.get(w, 0)+1) / (self.ns[0] + len(self.itow)+1))

  # positive, j = 1
  for k in range(len(words)):
    w = words[k]
    self.p_wk_cj[k, 1] = ((self.vocab_pos.get(w, 0)+1) / (self.ns[1] + len(self.itow)+1))
```

برای پیش بینی، طبق فرمول لگاریتم احتمال هر توکن جمع می شود. برای جلوگیری از underflow از فضای log استفاده شده.

```
def predict(self, test_tokens):
  ps = []
  pred = []
  for test in test_tokens:
    p = np.ones((self.classes_n, 1))
    # add probability of class
    p[0] = self.p_ci[0]
    p[1] = self.p_ci[1]
    for token in test:
      for c in range(self.classes_n):
        if token in self.wtoi:
          token_idx = self.wtoi[token]
          # add (instead of multiply because of log) probabiliy of word belonging to class c
          p[c] += (np.log(self.p_wk_cj[token_idx, c]))
        else:
          # add 1/dominator if token doesnt exist in vocab (no occurence in train set)
          p[c] += (np.log(1/(self.ns[c] + len(self.itow)+1)))

    ps.append(p)
    pred.append(np.argmax(p))
  return np.array(ps), np.array(pred)
```

همچنین با شمارش تعداد هر کلاس، $P(c_i)$ محاسبه می شود:

```
def find_p_ci(self):
  pos_n = np.count_nonzero(self.y)
  neg_n = len(self.y) - pos_n
  self.p_ci[1] = pos_n / len(self.y)
  self.p_ci[0] = 1 - self.p_ci[1]
```

نتایج هر مدل در شکل زیر قابل مشاهده است:

model	recall	precision	f1score	accuracy
unigram	0.81504	0.851555	0.832897	0.83648
bigram	0.8516	0.866363	0.858918	0.86012
trigram	0.79064	0.812479	0.801411	0.80408

مشاهده می شود که مدل bigram بهترین نتیجه را دارد. مدل unigram چون ترتیب را اصلا درنظر نمی گیرد، قادر نیست به خوبی عمل کند. مدل bigram توانسته با کمی در نظر گرفتن ترتیب بهتر شود. اما مدل trigram نتوانسته از bigram بهتر عمل کند. احتمال به این دلیل که دیتاست برای یادگیری تمام trigram ها به اندازه کافی بزرگ نبوده و تمام trigram های تست در داده آموزش دیده نشدند. در کل می توان نتیجه گرفت که ترتیب در نتیجه مدل تاثیرگذار است.

متن زیر، متن بعد از پیش پردازش یکی از نمونه هایی است که مدل unigram اشتباه، منفی پیش بینی کرده.

im absolut unk movi isnt sold love movi unk disney unk demand theyd eventu sell id buy copi everybodi know everyth everybodi movi good job havent figur whi disney hasnt put movi dvd vhs rental unk least havent seen ani copi wick good movi seen kid new generat dont get see think least put back channel movi doesnt deserv cheap unk deserv real thing im movi dvd

شکل زیر مقایسه تعداد برخی از توکن های این متن در نمونه های مثبت و منفی را نشان می دهد.

```
token dont, pos count: 3300, neg count: 5145
token get, pos count: 6459, neg count: 7638
token see, pos count: 7480, neg count: 6615
token think, pos count: 4290, neg count: 4619
token least, pos count: 1102, neg count: 2011
token put, pos count: 1529, neg count: 1599
token back, pos count: 2540, neg count: 2419
token channel, pos count: 187, neg count: 330
token movi, pos count: 22636, neg count: 28999
token doesnt, pos count: 1932, neg count: 2600
token deserv, pos count: 611, neg count: 558
token cheap, pos count: 201, neg count: 689
```

برخی کلمات مانند movie هستند که sentiment ای ندارند و نباید تاثیری در نتیجه بگذارند. اما مشاهده می شود که در این دیتاست، تفاوت زیادی بین تعداد آن در نمونه های مثبت و منفی است، و این تفاوت در محاسبه احتمالات نیز تاثیر دارد. برای جلوگیری از این کار، باید هنگام پیش پردازش این گونه کلمات را حذف کنیم.

دلیل دیگری که مدل این متن را منفی پیش بینی کرده، می تواند این باشد که کاربر از افعال نفی بسیاری مانند doesn't استفاده کرده که در نمونه های منفی تعداد بسیار بیشتری داشته. برخی کلمات به تنهایی sentiment ای ندارند اما با همراهی با کلمات دیگر می توانند مفهومی داشته باشند که این مدل قادر به درک آن نیست.

همین نمونه توسط مدل bigram نیز اشتباه پیش بینی شده:

```
token movi unk, pos count: 2174, neg count: 2852
token movi good, pos count: 170, neg count: 149
token movi dvd, pos count: 28, neg count: 20
token vhs rental, pos count: 2, neg count: 1
token love movi, pos count: 348, neg count: 84
token im absolut, pos count: 3, neg count: 1
```

مشاهده می شود که movie unk تفاوت تعداد زیادی دارد. توکن های unk در داده تست می توانند هم مثبت باشند و هم منفی و این می تواند در تشخیص مدل تداخل ایجاد کند.

همین نمونه توسط مدل trigram نیز اشتباه پیش بینی شده:

```
token dont get see, pos count: 10, neg count: 4
token movi good job, pos count: 3, neg count: 1
token absolut unk movi, pos count: 2, neg count: 8
token unk movi isnt, pos count: 8, neg count: 12
token love movi unk, pos count: 17, neg count: 6
token unk disney unk, pos count: 18, neg count: 10
```

تصویر بالا تمام trigram های این متن است که در داده آموزش دیده شده. واضح است که دیده نشدن تمام trigram ها به دلیل کوچک بودن دیتاست تاثیر زیادی روی عملکرد آن دارد.