Computer Engineering Department

Natural Language Processing

Assignment 2

Ali Sedaghi
97521378

Table of contents

Theoretical questions

Q1- Cross-Validation[1]

به صورت کلی یک دیتاست را به سه قسمت تقسیم می‌کنیم:

1. داده آموزشی (Train): مدل با دیدن این مجموعه داده آموزش داده می‌شود و ممکن است نسبت به آن Bias داشته باشد و روی آن Over-fit کند.
2. داده ارزیابی (Validation): از این مجموعه برای تنظیم هایپر پارامترهای مدل استفاده می‌شود و در آموزش از آن استفاده نمی‌شود. نتایج روی این مجموعه داده برای بررسی عملکرد مدل بسیار مهم است. هایپر پارامترهای مدل ممکن است نسبت به این داده Over-fit شود.
3. داده تست (Test): هنگامی که بر اساس داده Validation به مدل مناسب دست یافتیم یکبار برای اطمینان نتایج را روی این مجموعه داده بررسی می‌کنیم. مدل نباید نسبت به این مجموعه Bias داشته باشد.

در واقع داده Validation داده آموزش و تست را از هم جدا می‌کند. چند گونه از روش‌های ارزیابی Cross-Validation در ادامه آورده شده است:

Holdout Cross-Validation

در این روش دیتاست به دو قسمت آموزش (Train) و تست (Test) تقسیم می‌شود. معمولا 70 الی 80 درصد داده‌ها برای آموزش و 30 الی 20 درصد داده‌ها برای تست در نظر گرفته می‌شود. این دو زیرمجموعه هیچ اشتراکی با هم ندارند. قبل از جداسازی لازم تا دیتاست را Shuffle کنیم.

مشکل این روش این است که مقدار زیادی از داده که دارای اطلاعات مهمی برای یادگیری مدل است به مدل تزریق نمی‌شود و ممکن است مدل وزن‌های ناقصی را آموزش ببیند. به این روش Train/Test Split نیز می‌گویند. این روش ساده‌ترین و معمول‌ترین گزینه برای ارزیابی مدل است. در مسائلی که حجم دیتاست بزرگ است این روش انتخاب معقولی می‌باشد. در زمینه طبقه‌بندی تصاویر از آن می‌توان استفاده کرد.

[1] https://www.analyticssteps.com/blogs/7-types-cross-validation

Monte Carlo Cross-Validation

در این روش یک زیرمجموعه به صورت رندوم برای آموزش انتخاب می‌شود و باقی دیتا برای ارزیابی استفاده خواهد شد. این فرایند انتخاب زیرمجموعه رندوم برای آموزش چندین مرتبه تکرار می‌شود و زیرمجموعه‌های مختلف و تصادفی برای آموزش و ارزیابی امتحان خواهند شد. تفاوت این روش با روش Holdout در این است که فقط یکبار این جداسازی را انجام نمی‌دهیم و این جداسازی چندین مرتبه تکرار می‌شود. نتیجه آن کاهش Bias است. عیب این روش این است که ممکن است نقاطی همیشه در زیرمجموعه ارزیابی باشند و هیچگاه درون زیرمجموعه آموزش دیده نشوند.

K-fold Cross-Validation[2]

در این روش ابتدا Shuffle می‌کنیم. سپس دیتاست به K زیرمجموعه یا Fold تقسیم می‌شود. مقدار K به عنوان پارامتر این روش بایستی تنظیم شود. این زیرمجموعه‌ها هیچ اشتراکی با یکدیگر ندارند. در این روش به تعداد K مدل در نظر می‌گیریم. سپس مدل i را روی تمامی زیرمجموعه‌ها (Folds) به جز زیرمجموعه i آموزش می‌دهیم. از زیرمجموعه i نیز برای ارزیابی مدل استفاده می‌کنیم. در نهایت نتایج مدل را از میانگین عملکرد روی این زیرمجموعه‌ها به دست می‌آوریم. بعد از به دست آوردن بهترین هایپر پارامترها مدل را روی تمامی دیتاست آموزش می‌دهیم.

این روش بسیار Unbiased و Inclusive است زیرا کل دیتاست هم برای آموزش و هم ارزیابی استفاده شده است. هنگامی که دیتاست حجم بزرگی ندارد و متوسط است استفاده از این روش و روش پایین معقول است.

Stratified K-fold Cross Validation[3]

این روش مشابه روش بالا است. تنها تفاوت آن در نحوه تقسیم کردن دیتاست به K زیرمجموعه است. در روش بالا این تقسیم‌بندی به صورت کاملا Random صورت می‌گیرد و ممکن است توزیع کلاس‌ها در یک زیرمجموعه نامتوازن (Unbalanced) باشد. برای مثال درون زیرمجموعه i ممکن است 90 درصد داده‌ها مربوط به کلاس A باشد و تنها 10 درصد داده‌ها از کلاس B باشد. این روش با بهره‌برداری از Stratified Sampling،

[2] https://machinelearningmastery.com/k-fold-cross-validation/
[3] https://en.wikipedia.org/wiki/Stratified_sampling

زیرمجموعه‌هایی با توزیع متوازن ایجاد می‌کند. این کار باعث می‌شود هر Fold به تنهایی نماینده‌ای از کل دیتاست باشد و عملکرد مدل بهتر شود.

Leave-p-out Cross-Validation

فرض کنیم دیتاست ما شامل N نمونه باشد. پارامتر این روش P است. مدل را روی N - P نمونه آموزش می‌دهیم و روی P نمونه ارزیابی می‌کنیم. این کار را به ازای تمامی ترکیبات ممکن که دارای P نمونه هستند انجام می‌دهیم. در نهایت یک میانگین از نتایج می‌گیریم تا عملکرد کلی مدل حاصل شود. مدلی که میانگین نتایجش از همه بهتر باشد به عنوان بهترین مدل روی N داده آموزش داده می‌شود.

این روش بسیار زمان‌بر است زیرا بایستی تمامی ترکیبات شامل P نمونه امتحان شود اما Bias کمتری دارد. در دیتاست‌های کوچک استفاده از این روش معقول است.

Leave-one-out Cross-Validation

همان روش بالا در حالتی است که P = 1 باشد. در این حالت مدل تقریبا روی تمامی دیتاست (N - 1) نمونه آموزش می‌بیند و تنها یک نمونه برای ارزیابی استفاده می‌شود. این روش بسیار زمان‌بر است و هزینه بالایی دارد. استفاده از آن در جایی که دیتاست بسیار بسیار کوچک است منطقی است.

Rolling Cross-Validation

این روش روی دیتاست‌هایی که بر اساس سری زمانی (Time Series) هستند بسیار موثر است. تقریبا هیچ روشی که تا کنون بررسی کردیم روی این نوع مسائل موثر نیست و روش‌های بالا بیشتر در زمینه دسته‌بندی تصاویر و داده‌های بدون ترتیب زمانی موثر هستند. در این روش یک زیرمجموعه برای آموزش استفاده می‌شود و ادامه آن زیرمجموعه به عنوان داده تست در نظر گرفته می‌شود. در واقع ترتیب دیتاست را بهم نمی‌زنیم یعنی Shuffle نمی‌کنیم زیرا ترتیب زمانی مهم است. این کار را به ازای جاهای مختلف امتحان می‌کنیم. در واقع نقاط مختلف را به عنوان نقطه جداسازی در نظر می‌گیریم.

Q2- Language model

Part A

برای مدل زبانی Bigram کافی است احتمال رخداد کلمه W_i به شرطی که کلمه W_{i-1} پیش از آن بیاید را محاسبه کنیم. علائم منحصر به فرد این دیتا است 4 مورد زیر هستند.

<S> </S> A B

رابطه محاسبه مدل زبانی Bigram به صورت زیر است:

$$P(W_i \mid W_{i-1}) = \frac{Count\ (W_{i-1}W_i)}{Count\ (W_{i-1})}$$

با توجه به این که چهار Type داریم بایستی 16 رابطه را محاسبه کنیم.

$P(< S > \mid < S >) = \frac{0}{3} = 0$

$P(< S > \mid < /S >) = \frac{0}{3} = 0$

$P(< S > \mid A) = \frac{0}{10} = 0$

$P(< S > \mid B) = \frac{0}{11} = 0$

$P(< /S > \mid < S >) = \frac{0}{3} = 0$

$P(< /S > \mid < /S >) = \frac{0}{3} = 0$

$P(< /S > \mid A) = \frac{2}{10} = 0.2$

$P(< /S > \mid B) = \frac{1}{11}$

$P(A \mid < S >) = \frac{0}{3} = 0$

$P(A \mid < /S >) = \frac{0}{3} = 0$

$P(A \mid A) = \frac{2}{10} = 0.2$

$P(A \mid B) = \frac{8}{11}$

$P(B \mid < S >) = \frac{3}{3} = 1.0$

$P(B \mid < /S >) = \frac{0}{3} = 0$

$P(B \mid A) = \frac{6}{10} = 0.6$

$P(B \mid B) = \frac{2}{11}$

Part B

در حالت بدون Add-1 Smoothing در قسمت الف محاسبه شد و داشتیم:

$$P(A \mid B) = \frac{8}{11} = 0.72$$
$$P(< S > \mid B) = \frac{0}{11} = 0$$
$$P(A \mid < /S >) = \frac{0}{3} = 0$$

در حالت با Add-1 Smoothing از رابطه زیر استفاده می‌کنیم:

$$P(W_i \mid W_{i-1}) = \frac{Count\,(W_{i-1}W_i) + 1}{Count\,(W_{i-1}) + |V|}$$

مقدار |V| برابر تعداد Typeهای پیکره است که مقدار آن برابر 4 است.

$$P(A \mid B) = \frac{8+1}{11+4} = \frac{9}{15} = 0.6$$
$$P(< S > \mid B) = \frac{0+1}{11+4} = \frac{1}{15} = 0.06$$
$$P(A \mid < /S >) = \frac{0+1}{3+4} = \frac{1}{7} = 0.14$$

Q3- Bayes rule

مقدار Predicted برابر عدد پیش‌بینی شده توسط سیستم است. مقدار Real نیز مقدار واقعی ورودی است.

$$P(P = 7 \mid R = 7) = 0.5$$
$$P(P = 8 \mid R = 7) = 0.5$$
$$P(P = 7 \mid R = 8) = 0.3$$
$$P(P = 8 \mid R = 8) = 0.7$$

برای هر مقدار P و T که P = T و به جز 7 و 8 داریم:

$$P(P \mid R) = 1.0$$

برای هر مقدار P و T که P ≠ T و به جز 7 و 8 داریم:

$$P(P \mid R) = 0.0$$

با توجه به اینکه احتمال رخداد همه‌ی اعداد یکسان است داریم:

$$P(A) = 0.1$$

رابطه قانون بیز به صورت زیر است:

$$P(P \cap R) = P(P \mid R) \times P(R)$$

احتمال تشخیص صحیح عدد 7 به صورت زیر محاسبه می‌شود:

$$P(P = 7 \cap R = 7) = P(P = 7 \mid R = 7) \times P(A = 7) = 0.5 \times 0.1 = 0.05 = 5\%$$

احتمال تشخیص صحیح عدد 8 به صورت زیر محاسبه می‌شود:

$$P(P = 8 \cap R = 8) = P(P = 8 \mid R = 8) \times P(A = 8) = 0.7 \times 0.1 = 0.07 = 7\%$$

برداشت دیگر از احتمالات خواسته شده به این صورت است که احتمال ورودی عدد X به شرط آنکه عدد X پیش‌بینی شده باشد. مقدار آن به صورت زیر محاسبه می‌شود:

$$P(R = 7 \mid P = 7) = \frac{P(A=7) \times P(P=7 \mid R=7)}{\sum P(A=i) \times P(P=7 \mid R=i)} = \frac{0.05}{0.05 + 0.03} = \frac{5}{8}$$

$$P(R = 8 \mid P = 8) = \frac{P(A=8) \times P(P=8 \mid R=8)}{\sum P(A=i) \times P(P=7 \mid R=i)} = \frac{0.07}{0.05 + 0.07} = \frac{7}{12}$$

Practical questions

Q1- Naive bayes

Cleaning data

دو تابع کمکی برای تبدیل لیست شامل ایندکس به کلمات و بالعکس تعریف شده است.

2.0. Index to word and vice versa

```
[3]  1 word_index = imdb.get_word_index()
     2 index_word = {i: word for word, i in word_index.items()}

[4]  1 def mapper(sources, map_dict):
     2     destinations = []
     3     for source in sources:
     4         destination = [map_dict.get(element) for element in source]
     5         destinations.append([d for d in destination if d is not None])
     6     return np.array(destinations, dtype=object)

[5]  1 x_train_words = mapper(x_train, index_word)
     2 x_train_indexes = mapper(x_train_words, word_index)
     3
     4 x_test_words = mapper(x_test, index_word)
     5 x_test_indexes = mapper(x_test_words, word_index)

[6]  1 print(x_train[0])
     2 print(x_train_words[0])
     3 print(x_train_indexes[0])

[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100,
['the', 'as', 'you', 'with', 'out', 'themselves', 'powerful', 'lets', 'loves', 'their', 'becomes
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100,
```

در ادامه با استفاده از Stop Words درون NLTK این کلمات از دیتاست حذف شده است. همچنین تمامی کلماتی که طولی کمتر از دو دارند نیز از درون دیتاست حذف شده اند. این حذف کردن باعث شده است مدل فضای کمتری اشغال کند و احتمالات مربوط به N-gram سریع‌تر محاسبه شود. در حالتی که این حذف انجام نشود دقت مدل کمی بالاتر خواهد بود (حدود 3 الی 5 درصد)

```
[7]  1 nltk.download('stopwords')
     2 stop_words = set(nltk.corpus.stopwords.words('english'))

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
[8]  1 def cleaner(sentences, stop_words):
     2     new_sentences = []
     3     for sentence in sentences:
     4         new_sentence = [word for word in sentence if word not in stop_words and len(word) > 2]
     5         new_sentences.append(new_sentence)
     6     return new_sentences
```

```
[9]  1 x_train_words_cleaned = cleaner(x_train_words, stop_words)
     2 x_train_indexes_cleaned = mapper(x_train_words_cleaned, word_index)
     3
     4 x_test_words_cleaned = cleaner(x_test_words, stop_words)
     5 x_test_indexes_cleaned = mapper(x_test_words_cleaned, word_index)
```

```
[10] 1 print(x_train_words[0])
     2 print(x_train[0])
     3 print(x_train_indexes_cleaned[0])
     4 print(x_train_words_cleaned[0])

['the', 'as', 'you', 'with', 'out', 'themselves', 'powerful', 'lets', 'loves', 'their', 'becomes', 'reaching'
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112,
[973, 1622, 1385, 458, 4468, 3941, 173, 256, 838, 112, 670, 22665, 480, 284, 150, 172, 112, 167, 21631, 336,
['powerful', 'lets', 'loves', 'becomes', 'reaching', 'journalist', 'lot', 'anyone', 'atmosphere', 'never', 'r
```

Tokenization

این دیتاست در حالت عادی توکنایز شده است و نیازی به این کار نیست. هر نمونه این دیتاست لیستی از ایندکس‌ها است و هر ایندکس بیانگر یک کلمه است.

Stemming

همانطور که در کلاس درس و اسلایدها گفته شد، Stemming در مدل‌های زبانی مبتنی بر احتمالات باعث خراب شدن نتایج می‌شود. در واقع یافتن ریشه کلمات اطلاعات بسیار مهمی را از بین می‌برد و باعث می‌شود در N-gram موارد زیادی از <UNK> ایجاد شود.

N-gram models

از ابزار NLTK برای تولید N-gram استفاده شده است. سپس فرکانس هر N-gram محاسبه شده است و با استفاده از آن مقادیر احتمالی ایجاد شده است. مقدار احتمالی هر N-gram به صورت زیر محاسبه شده است:

value = (freq + 1) / (denominator + vocab_size)

در این رابطه Denominator بسته به اینکه N چه مقداری است محاسبه می‌شود. برای مثال در محاسبه Trigram زیر به صورت زیر عمل می‌کنیم:

Trigram: (w1w2w3)

Value = (Count (w1w2w3) + 1) / (Count (w1w2) + |V|)

این مقادیر به ازای هر کلاس جداگانه حساب می‌شود. مشاهده می‌شود که Add-1 Smoothing نیز اعمال شده است. مقدار |V| برای کلاس منفی برابر 61337 و برای کلاس مثبت 63648 می‌باشد.

```python
def get_vocab_size(x, y):
    x_dict = {0: [], 1: []}
    for i, sample in enumerate(x):
        class_idx = y[i]
        x_dict[class_idx] += sample

    p_vocab_size = len(collections.Counter(x_dict[1]))
    n_vocab_size = len(collections.Counter(x_dict[0]))

    return n_vocab_size, p_vocab_size
```

```python
def build_ngrams(x, y, n):
    ngrams_dict = {0: [], 1: []}

    for i, sample in enumerate(x):
        class_idx = y[i]
        sample_ngrams = ngrams(sample, n)
        ngrams_dict[class_idx].extend(sample_ngrams)

    p_ngrams_freq = collections.Counter(ngrams_dict[1])
    n_ngrams_freq = collections.Counter(ngrams_dict[0])

    return n_ngrams_freq, p_ngrams_freq
```

```python
def get_model(freqs, vocab_size, before_freqs=None):
    denominator = sum(freqs.values())
    model = dict()
    for gram, freq in freqs.items():
        if before_freqs != None:
            denominator = before_freqs[gram[:-1]]
        value = (freq + 1) / (denominator + vocab_size)
        model[gram] = value
    return model
```

Naive Bayes classifier

پس از به دست آوردن مدل‌های احتمالاتی زبانی از روی داده آموزشی، یک دسته‌بند مبتنی بر بیز ایجاد می‌کنیم تا با ورودی گرفتن داده تست و ساختن N-gram های آن و بررسی احتمال هر N-gram پیش‌بینی صحیح را برای هر کلاس انجام دهد. با توجه به این که درون داده تست ممکن است N-gram هایی ایجاد شود که درون داده آموزشی نیست، از مهموم <UNK> استفاده کرده ایم تا 0 وارد محاسبات نشود. همچنین برای اینکه مقادیر احتمالاتی به صفر میل پیدا نکنند، به جای ضرب کردن ابتدا لگاریتم گرفته ایم سپس با هم جمع کرده ایم.

```python
def nb_predict(x_test, n, model, neg_class_prob, unk_factor, vocab_size):
    preds = []

    for sample in x_test:
        neg_prob = np.log(neg_class_prob)
        pos_prob = np.log(1 - neg_class_prob)

        ngrams_list = ngrams(sample, n)
        for gram in ngrams_list:
            gram_neg_prob = model[0].get(gram)
            gram_pos_prob = model[1].get(gram)

            # <UNK>
            if gram_neg_prob == None:
                gram_neg_prob = 1 / (unk_factor[0] + vocab_size[0])
            if gram_pos_prob == None:
                gram_pos_prob = 1 / (unk_factor[1] + vocab_size[1])

            gram_neg_prob = np.log(gram_neg_prob)
            gram_pos_prob = np.log(gram_pos_prob)

            neg_prob += gram_neg_prob
            pos_prob += gram_pos_prob

        if neg_prob > pos_prob:
            preds.append(0)
        else:
            preds.append(1)
    return np.array(preds)
```

Metrics

مطابق روابط درون اسلایدهای درس ابتدا مقادیر TP و TN و FN و FP محاسبه شده است. سپس متریک‌های Accuracy و Precision و Recall و F1 محاسبه شده است.

```python
def calculate_metrics(y_r, y_p):
    tp = 0
    tn = 0
    fn = 0
    fp = 0

    for i in range(y_test.shape[0]):
        if y_r[i] == 1:
            if y_p[i] == 1:
                tp += 1
            elif y_p[i] == 0:
                fn += 1
        elif y_r[i] == 0:
            if y_p[i] == 0:
                tn += 1
            elif y_p[i] == 1:
                fp += 1

    a = (tp + tn) / (tp + tn + fn + fp)
    p = tp / (tp + fp)
    r = tp / (tp + fn)
    f1_score = (2 * p * r) / (p + r)

    return {
        'accuracy': a,
        'precision': p,
        'recall': r,
        'f1_score': f1_score,
    }
```

Results

جدول نتایج برای 3 حالت Unigram و Bigram و Trigram در زیر آورده شده است:

	Accuracy	Precision	Recall	F1-Score
Unigram	81.3	86.1	74.6	79.9
Bigram	80.4	89.1	69.3	77.9
Trigram	62.7	85.0	31.0	45.4

نکته جالب این است که اگر دیتاست را از Stop Words ها پاک نکنیم نتایج بسیار متفاوت خواهد بود:

	Accuracy	Precision	Recall	F1-Score
Unigram	80.4	85.3	73.3	78.9
Bigram	85.7	92.1	78.1	84.5
Trigram	84.8	93.8	74.6	83.1

در واقع با تمیز کردن دیتاست موارد بسیاری از <UNK> به خصوص در حالت Trigram ایجاد می‌شود که باعث می‌شود دسته‌بند احتمالات درستی را در نظر نگیرد.

در ادامه 3 جمله ابتدایی درون تست ست در 3 حالت Unigram و Bigram و Trigram مورد بررسی قرار می‌گیرد. این جملات به صورت زیر هستند:

1. the wonder own as by is sequence i i jars roses to of hollywood br of down shouting getting boring of ever it sadly sadly sadly i i was then does don't close faint after …

	Unigram	Bigram	Trigram
Predicted	0	0	0
Real	0	0	0

به نظر می‌رسد این جمله برای مدل‌ها دشوار نبوده، وجود کلماتی مانند Boring و Sadly باعث شده نظر قاطعانه منفی باشد.

2. the as you world's is quite br mankind most that quest are chase to being quickly of little it time hell to plot br of something long put are of every place this consequence council …

	Unigram	**Bigram**	**Trigram**
Predicted	1	1	1
Real	1	1	0

در حالت Trigram مدل اشتباه کرده، دلیل آن وجود Trigram هایی است که درون داده آموزش وجود نداشته اند و مدل با احتمال مصنوعی <UNK> با آن‌ها رفتار کرده است.

3. the plot near ears recent halliburton cosmopolitan of him flicks frank br by excellent sans br of past loyalty near really all grief family four victim's to movie that obvious family …

	Unigram	**Bigram**	**Trigram**
Predicted	1	1	1
Real	1	1	0

وجود کلماتی مانند Excellent باعث شده در حالت Unigram و Bigram نظر مثبت ثبت شود. اما اثر این کلمه در حالت Trigram با توجه وجود نداشتن جایگشت‌های مختلف خنثی شده است.

نکته: نتایج کامل‌تر و جزییات درون نوت‌بوک موجود است.