تمرین پنجم NLP

سوال 1

Generative classifiers و Discriminative classifiers دو رویکرد مهم در یادگیری ماشین هستند به ویژه در تسک classification. درک تفاوت های آن ها می تواند به روشن شدن نحوه عملکردشان و زمان استفاده از هر نوع کمک کند.

Generative classifiers:

Generative classifier ها مانند Naive Bayes بر مدل سازی چگونگی تولید داده ها تمرکز دارند. آن ها توزیع احتمال مشترک P(d,c) را یاد می گیرند، جایی که d نمایانگر داده ها و c نمایانگر برچسب های کلاس است. این به این معنی است که آنها سعی می کنند فرآیند زیرین تولید داده ها برای هر کلاس را درک کنند.

ویژگی های کلیدی:

مدل سازی Joint Probability: آنها P(d|c) (احتمال داده ها با توجه به یک کلاس) و P(c) را تخمین می زنند(the prior probability of the class).

آن ها اغلب از قضیه بیز برای استخراج P(c|d) استفاده می کنند، که احتمال یک کلاس با توجه به داده ها است.

به عنوان مثال Naive Bayes فرض می کند که ویژگی ها با توجه به برچسب کلاس به طور شرطی مستقل هستند که محاسبات را ساده تر می کند.

کاربردها: مناسب برای تولید نمونه های جدید داده و مدیریت موثر داده های گمشده.

مثال:

در سناریوی تشخیص هرزنامه یک مدل Generative مانند Naive Bayes ویژگی هایی مانند فراوانی کلمات در ایمیل ها را تحلیل می کند تا احتمال هرزنامه بودن یا نبودن یک ایمیل را تعیین کند. این مدل احتمال ها را بر اساس داده های مشاهده شده از هر دو نوع ایمیل های اسپم و غیر اسپم محاسبه می کند.

Discriminative classifiers:

Discriminative classifier ها مانند logistic regression با مدل سازی مستقیم احتمال شرطی

P(c|d) را حساب می کنند. آن ها به جای درک چگونگی تولید داده ها بر تمایز بین کلاس ها تمرکز می کنند.

ویژگی های کلیدی:
آن ها یاد می گیرند که برچسب های کلاس را مستقیما از ویژگی های ورودی پیش بینی کنند.
فرآیند آموزش به دنبال حداکثر کردن احتمال شرطی داده های مشاهده شده با توجه به برچسب های کلاس است که اغلب منجر به عملکرد بهتر در classification می شود.
Discriminative classifier ها می توانند ویژگی های مختلف را بدون فرض استقلال بین آن ها ترکیب کنند که این امر باعث ایجاد مرزهای تصمیم گیری دقیق تری می شود.
کاربردها: معمولا در سناریوهایی که دقت بالا در classification ضروری است استفاده می شود.

مثال:
در همان وظیفه تشخیص هرزنامه logistic regression ویژگی هایی مانند حضور کلمات و زمینه آن ها در ایمیل ها را تحلیل می کند تا پیش بینی کند که آیا یک ایمیل هرزنامه است یا خیر. این به طور مستقیم رابطه بین ویژگی های ایمیل و طبقه بندی آن ها به عنوان هرزنامه یا غیرهرزنامه را مدل سازی می کند.

سوال 2

A) جملات:

1. Will mark hunt:
   - **Will**: Noun
   - **mark**: Verb
   - **hunt**: Noun

2. Mark will hunt:
   - **Mark**: Noun
   - **will**: Modal
   - **hunt**: Verb

3. Can Will mark? :
    - **Can**: Modal
    - **Will**: Noun
    - **mark**: Verb

4. Mark can hunt:
    - **Mark**: Noun
    - **can**: Modal
    - **hunt**: Verb

تعداد تکرار هر یک:

Word	Noun	Verb	Modal	Total Count
will	2	0	1	3
mark	2	2	0	4
hunt	1	2	0	3
can	0	0	2	2

احتمال هر یک:

Word	Noun	Verb	Modal	Total Count
will	$\frac{2}{3} = 0.67$	0	$\frac{1}{3} = 0.33$	3
mark	$\frac{2}{4} = 0.50$	$\frac{2}{4} = 0.50$	0	4
hunt	$\frac{1}{3} = 0.33$	$\frac{2}{3} = 0.67$	0	3
can	0	0	$\frac{2}{2} = 1$	2

B)

Sentence 1: <S> → N → V → N → <E>

Sentence 2: <S> → N → M → V → <E>

Sentence 3: <S> → M → N → V → <E>

Sentence 4: <S> → N → M → V → <E>

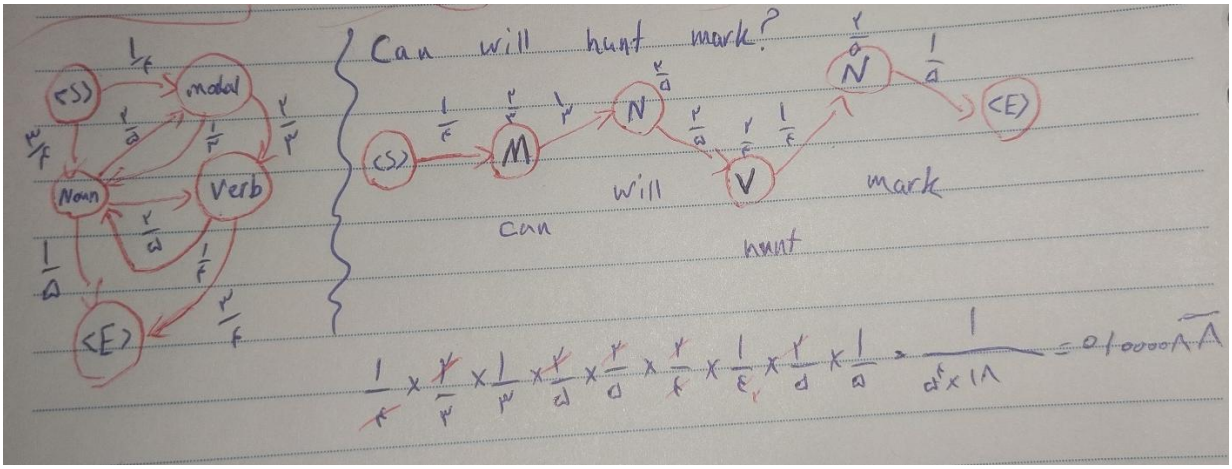
تعداد تکرار هر یک:

	<S>	Noun	Verb	Modal	<E>	Total
<S>	0	3	0	1	0	4
Noun	0	0	2	2	1	5
Verb	0	1	0	0	3	4
Modal	0	1	2	0	0	3
<E>	0	0	0	0	0	0

احتمال هر یک:

	<S>	Noun	Verb	Modal	<E>
<S>	0	$\frac{3}{4} = 0.75$	0	$\frac{1}{4} = 0.25$	0
Noun	0	0	$\frac{2}{5} = 0.40$	$\frac{2}{5} = 0.40$	$\frac{1}{5} = 0.20$
Verb	0	$\frac{1}{4} = 0.25$	0	0	$\frac{3}{4} = 0.75$
Modal	0	$\frac{1}{3} = 0.33$	$\frac{2}{3} = 0.67$	0	0
<E>	0	0	0	0	0

(C



(D

Prompt to LLM:

Example 1:

Text: <S> Will mark hunt <E>

Pos tags: <S> Noun Verb Noun <E>

Example 2:

Text: <S> Mark will hunt <E>

Pos tags: <S> Noun Modal Noun <E>

Example 3:

Text: <S> Can Will mark? <E>

Pos tags: <S> Modal Noun Verb <E>

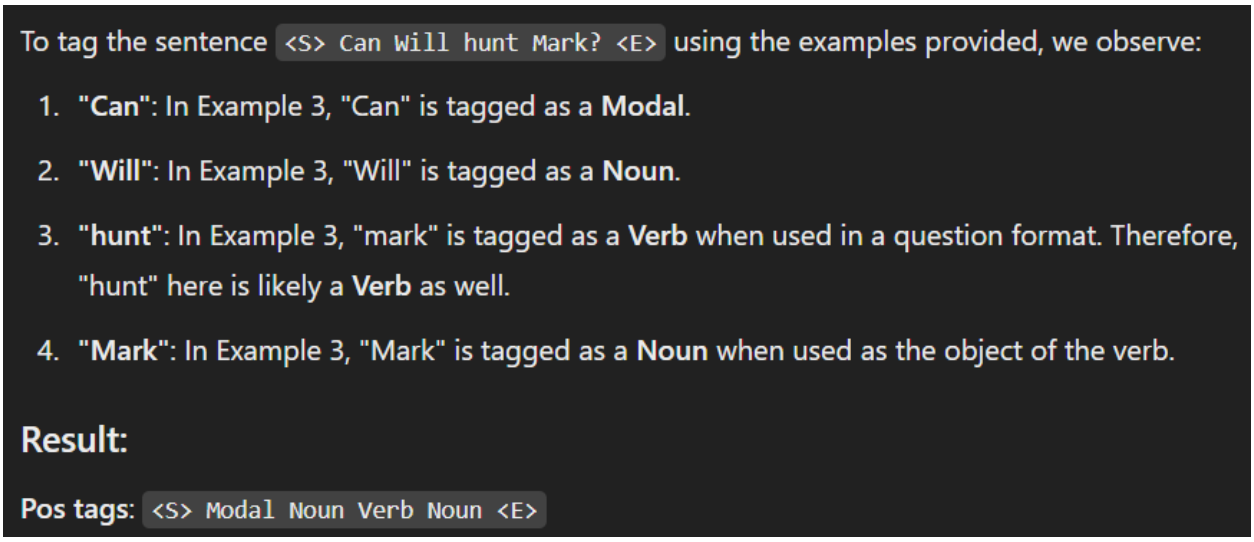
Example 4:

Text: <S> Mark can hunt <E>

Pos tags: <S> Noun Modal Noun <E>

Use this example and tag this sentence:

<S> Can Will hunt Mark? <E>



جواب chatgpt نیز با جوابی که ما با استفاده از دیتاست بدست آوردیم یکی شد.

سوال سوم

جمله:

Astronomers saw stars with ears.

قوانین:

- S → NP VP(1.0)
- P → P NP(1.0)
- VP → V NP(0.7)
- VP → VP PP(0.3)
- P → with(1.0)
- V → saw(1.0)
- NP → NP PP(0.4)
- NP → astronomers(0.1)
- NP → ears(0.18)

- NP → saw(0.04)
- NP → stars(0.18)
- NP → telescopes(0.1)

	1	2	3	4	5
1	$X_{NP} = 0.1$		$X_S = 0.0126$		$X_{S=} 0.015876$
2		$X_{NP} = 0.04$ $X_V = 1.0$	$X_{VP} = 0.126$		$X_{VP} = 0.015876$
3			$X_{NP} = 0.18$		$X_{NP} = 0.01296$
4				$X_P = 0.1$	$X_{PP} = 0.18$
5					$X_{NP} = 0.18$
	astronomers	saw	stars	with	ears

$X_{VP}(2,3) = P(VP \rightarrow V\ NP) \times X_V(2,2) \times X_{NP}(3,3) = 0.7 \times 1.0 \times 0.18 = 0.126$

$X_{PP}(4,5) = P(PP \rightarrow P\ NP) \times X_P(4,4) \times X_{NP}(5,5) = 1.0 \times 1.0 \times 0.18 = 0.18$

$X_S(1,3) = P(S \rightarrow NP\ VP) \times X_{NP}(1,1) \times X_{VP}(2,3) = 1.0 \times 0.1 \times 0.126 = 0.0126$

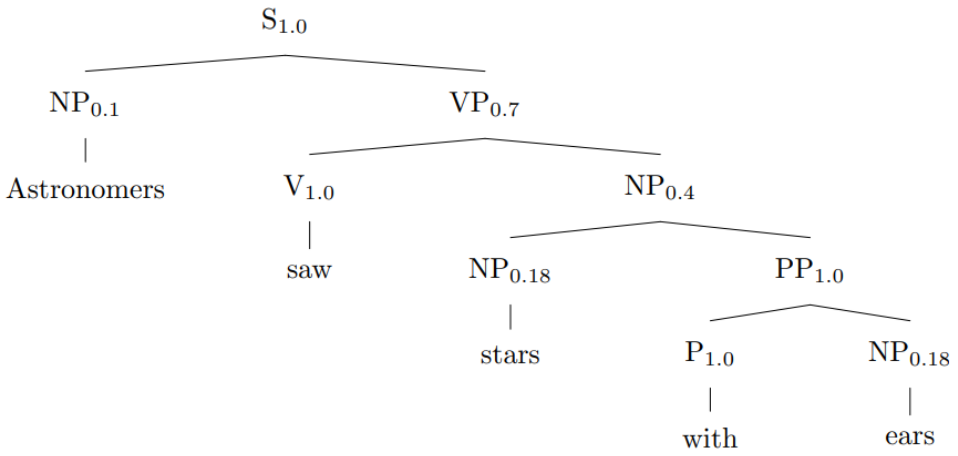
$X_{NP}(3,5) = P(NP \rightarrow NP\ PP) \times X_{NP}(3,3) \times X_{PP}(4,5) = 0.4 \times 0.18 \times 0.18 = 0.1296$

$X_{VP}(2,5) = (P(VP \rightarrow V\ NP) \times X_V(2,2) \times X_{NP}(3,5)) + (P(VP \rightarrow VP\ PP) \times X_{VP}(2,3) \times X_{PP}(4,5)) =$

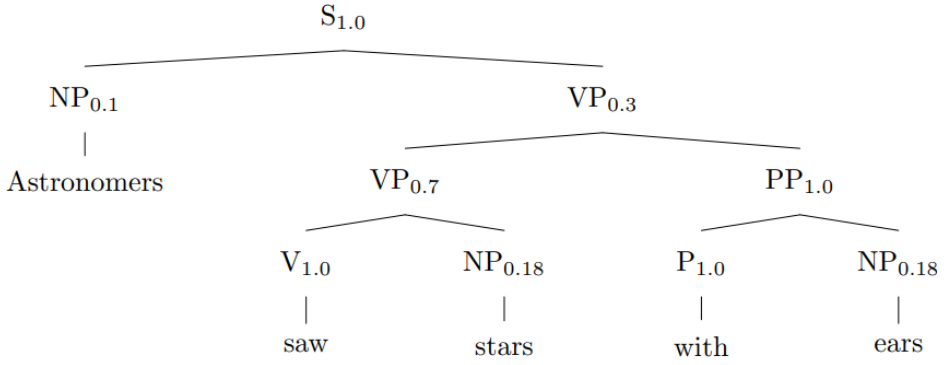
$(0.7 \times 1.0 \times 0.01296) + (0.3 * 0.126 * 0.18) = 0.009072 + 0.006804 = 015876$

جدول نهایی:

$X_{S=} 0.015876$				
	$X_{VP} = 0.015876$			
$X_S = 0.0126$		$X_{NP} = 0.01296$		
	$X_{VP} = 0.126$		$X_{PP} = 0.18$	
$X_{NP} = 0.1$	$X_{NP} = 0.04$ $X_V = 1.0$	$X_{NP} = 0.18$	$X_P = 0.1$	$X_{NP} = 0.18$
astronomers	saw	stars	with	ears



$P(t_1) = 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \times 0.18 \times 1.0 \times 1.0 \times 0.18 = 0.0009072$



$P(t_2) = 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \times 0.18 \times 1.0 \times 1.0 \times 0.18 = 0.0006804$

$P(t_1) + P(t_2) = 0.0009072 + 0.0006804 = 0.0015876$

سوال چهارم

```
train_data = emotion_data["train"].to_pandas()
validation_data = emotion_data["validation"].to_pandas()
test_data = emotion_data["test"].to_pandas()

combined_data = pd.concat([train_data, validation_data])

filtered_data = combined_data[combined_data["label"].isin([0, 1])]
test_data = test_data[test_data["label"].isin([0, 1])]
```

در این بخش ابتدا داده های train و val و test را جدا می کنیم و به DataFrame تبدیل می کنیم و سپس داده های train و val را باهم ادغام می کنیم. در نهایت داده های با برچسب 0 و 1 را فیلتر می کنیم.

```
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words("english"))
def preprocess_text(text):
    text = text.lower()
    text = text.translate(str.maketrans("", "", string.punctuation))
    tokens = word_tokenize(text)
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    return " ".join(tokens)

filtered_data["text"] = filtered_data["text"].apply(preprocess_text)
test_data["text"] = test_data["text"].apply(preprocess_text)
vocab = list(set(word for sentence in filtered_data["text"] for word in sentence.split()))
word_to_idx = {word: idx for idx, word in enumerate(vocab)}

def text_to_vector(sentence):
    vector = np.zeros(len(vocab))
    for word in sentence.split():
        if word in word_to_idx:
            vector[word_to_idx[word]] += 1
    return vector

X = np.array([text_to_vector(sentence) for sentence in filtered_data["text"]])
y = filtered_data["label"].values

X_test = np.array([text_to_vector(sentence) for sentence in test_data["text"]])
y_test = test_data["label"].values
```

در ادامه نوبت به پیش پردازش متن میرسد که ابتدا تمام حروف متن به حروف کوچک تبدیل می شود. در ادامه تمامی نشانه های نگارشی (مثل نقطه، ویرگول) از متن حذف می شود.سپس متن را به لیستی از کلمات (توکن ها) تبدیل می کنیم و در ادامه ریشه یابی هر کلمه (lemmatization) با استفاده از WordNetLemmatizer، فقط برای کلماتی که در لیست stop_words نیستند انجام می شود.

در ادامه تابع مربوطه را روی دیتاهای train و test اعمال می کنیم. سپس یک لیست از تمامی کلمات یکتای موجود در متن های پردازش شده دیتافریم filtered_data ایجاد میکند. سپس هر جمله شکسته می شود و کلماتش استخراج می شوند و تمامی کلمات به یک مجموعه (set) تبدیل می شوند (برای حذف تکراری ها). و در نهایت به یک لیست تبدیل می شود.

در ادامه یک دیکشنری ای که هر کلمه در واژگان (vocab) را به یک اندیس (شماره یکتا) نگاشت می دهد می سازیم.

تابع text_to_vector تابعی برای تبدیل یک جمله به یک بردار است که اول یک بردار صفر به طول واژگان (vocab) برای هر جمله می سازد. سپس برای هر کلمه در جمله، حلقه اجرا می شود و بررسی می کند که آیا کلمه در واژگان وجود دارد یا خیر اگر کلمه وجود داشته باشد، مقدار عنصر مربوط به آن کلمه در بردار یک واحد افزایش می یابد.

در ادامه همه جملات داده های آموزشی و تستی به یک بردار تبدیل می شود و سپس به یک آرایه numpy تبدیل می شود.

```
# The Sigmoid Function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Loss Function (Binary Cross-Entropy)
def compute_loss(y_true, y_pred):
    return - (y_true * np.log(y_pred + 1e-15) + (1 - y_true) * np.log(1 - y_pred + 1e-15))
```

در ادامه برای شروع آموزش ابتدا تابع سیگموید را پیاده سازی می کنیم و برای محاسبه loss از Binary Cross-Entropy استفاده می کنیم.

```python
# Stochastic Gradient Descent Algorithm for Logistic Regression
def train_logistic_regression_sgd(X, y, learning_rate=0.01, bias=0, epochs=100):
    n_samples, n_features = X.shape
    np.random.seed(0)
    theta = np.random.randn(n_features)  # Random initialize parameters

    for epoch in range(epochs):
        for i in range(n_samples):
            # Select one sample at a time
            x_i = X[i]
            y_i = y[i]

            # Compute prediction (sigmoid)
            y_pred = sigmoid(np.dot(theta, x_i)+bias)

            # Compute loss
            loss = compute_loss(y_i, y_pred)

            # Compute gradient
            gradient_w = (y_pred - y_i) * x_i
            gradient_b = (y_pred - y_i)

            # Update parameters
            theta -= learning_rate * gradient_w
            bias -= learning_rate * gradient_b

        # Print loss at each epoch for monitoring
        if epoch % 10 == 0:
            y_preds_epoch = sigmoid(np.dot(X, theta))
            epoch_loss = np.mean(compute_loss(y, y_preds_epoch))
            print(f"Epoch {epoch}, Loss: {epoch_loss:.3}")

    return theta , bias
```

در اینجا یک تابع برای logistic_regression با استفاده از sgd می سازیم که طبق فرمول قرار داده شده کار میکند.

ابتدا theta با وزن های تصادفی مقداردهی می شوند. سپس یک حلقه روی دوره ها (epochs) و یک حلقه روی نمونه ها می زنیم و برای هر نمونه ماتریس تتا را در نمونه ضرب می کنیم و با استفاده از تابع سیگموید y را پیش بینی می کنیم. در ادامه loss را محاسبه می کنیم. در ادامه گرادیان ها را محاسبه می کنیم و در نهایت پارامترها را به‌روزرسانی می کنیم.

در هر 10 ایپاک نیز loss چاپ می شود.

```
learning_rate = 0.01
epochs = 1000
bias = 0
theta , bias = train_logistic_regression_sgd(X, y, learning_rate,bias, epochs)
```

در اینجا تابع را با پارامتر هایی که مشخص کردیم صدا می زنیم که بخشی از نتیجه آن به شرح زیر است:

```
Epoch 810, Loss: 0.00697
Epoch 820, Loss: 0.00689
Epoch 830, Loss: 0.00682
Epoch 840, Loss: 0.00675
Epoch 850, Loss: 0.00668
Epoch 860, Loss: 0.00661
Epoch 870, Loss: 0.00655
Epoch 880, Loss: 0.00648
Epoch 890, Loss: 0.00642
Epoch 900, Loss: 0.00635
Epoch 910, Loss: 0.00629
Epoch 920, Loss: 0.00623
Epoch 930, Loss: 0.00617
Epoch 940, Loss: 0.00612
Epoch 950, Loss: 0.00606
Epoch 960, Loss: 0.006
Epoch 970, Loss: 0.00595
Epoch 980, Loss: 0.0059
Epoch 990, Loss: 0.00584
```

پس از 1000 ایپاک به loss 0.00584 میرسیم.

```
y_preds = [sigmoid(np.dot(theta, x) + bias) >= 0.5 for x in X_test]
accuracy = np.mean(y_preds == y_test)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.9780564263322884
```

دقت مدل نیز در کد بالا قابل مشاهده است.

```python
y_preds = np.array([1 if sigmoid(np.dot(theta, x) + bias) >= 0.5 else 0 for x in X_test])

# Initialize counts
tp = 0  # True Positive
tn = 0  # True Negative
fp = 0  # False Positive
fn = 0  # False Negative

for true, pred in zip(y_test, y_preds):
    if true == 1 and pred == 1:
        tp += 1
    elif true == 0 and pred == 0:
        tn += 1
    elif true == 0 and pred == 1:
        fp += 1
    elif true == 1 and pred == 0:
        fn += 1
precision = tp / (tp + fp) if (tp + fp) > 0 else 0
recall = tp / (tp + fn) if (tp + fn) > 0 else 0
f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0

# Display the results
print("Confusion Matrix:")
print(np.array([[tn, fp], [fn, tp]]))
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F-Measure: {f1:.2f}")
```

در نهایت مقادیر ارزیابی را با استفاده از فرمول هایشان به صورت دستی محاسبه می کنیم که به شرح زیر است:

```
Confusion Matrix:
[[561  20]
 [  8 687]]
Precision: 0.97
Recall: 0.99
F-Measure: 0.98
```

و در نهایت مقادیر ارزیابی با استفاده از کتابخانه های مربوطه:

```python
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix

precision = precision_score(y_test, y_preds)
recall = recall_score(y_test, y_preds)
f1_score = f1_score(y_test, y_preds)
confusion_matrix = confusion_matrix(y_test, y_preds)

print("Confusion Matrix:")
print(confusion_matrix)
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1_score:.2f}")
```

```
Confusion Matrix:
[[561  20]
 [  8 687]]
Precision: 0.97
Recall: 0.99
F1 Score: 0.98
```

محمد حقیقت - 403722042