Assignment NO.4 Solutions

NLP | Fall 1401 | Dr.Minayi

Student name : **Amin Fathi**

Student id : **400722102**

Problem 1

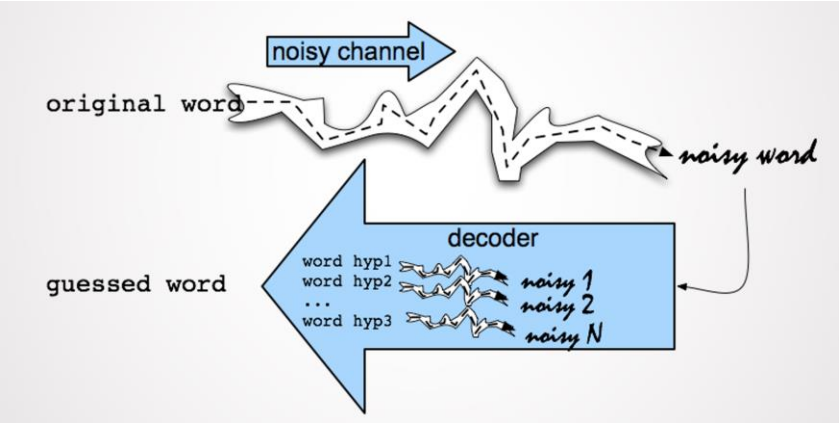
a)

Errorها را می‌توان به دو دسته‌ی کلی تقسیم کرد. Non-word Erro و Real-word Errors

Non-word Erro به این معناست که کلمه اشتباه نوشته شده اصلاً در دیکشنری وجود ندارد و کلمه به صورت کلی نادرست می‌باشد. مثلاً graffe که منظور giraffe بوده است. کلمه‌ی graffe در دیکشنری وجود ندارد. در Real-word Errors کلمه‌ی اشتباه در دیکشنری وجود دارد و به خودش دارای معنا است ولی مقصود کلمه‌ی دیگری بوده است. Real-word Errorsها خود به دو دسته‌ی Typographical errors و cognitive errors تقسیم می‌شود. در Typographical errors، خطا به صورت غیرعمد به وجود آمده است. مثلاً هنگام تایپ کردن سریع جای دو کاراکتر اشتباه زده شده است. به عنوان مثال، three و there. در خطای cognitive errors خطا به دلیل شناخت اشتباه انسان رخ می‌دهد و مثلا کلمات هم‌آوا اشتباه گرفته می‌شوند. به عنوان مثال، two و too.

b)

مدل خطای احتمالی را برای زبان‌ها و زمینه‌های مختلف آموزش دیده باشد و بتواند پارامترهای خود را تنظیم کند، مدل noisy channelنامیده می‌شود. اگرnoisy channel ، درست مدل شود، حدس صحیحی می‌تواند در مورد کلمه مورد نظر بزند. به عبارتی می‌توان گفت که noisy channel یک روش برای اصلاح غلط املایی و یا همان spelling errors می‌باشد. این روش به شکل زیر درنظر گرفته می‌شود.



این روش اینگونه بیان می‌کند که با هر اشتباهی کلمه‌ی ما noisy می‌شود و به noisy word تبدیل می‌شود. به عبارتی دیگر می‌توان این مسیر را مانند یک encder درنظر گرفته که کلمه صحیح را به کلمه نویزی تبدیل می‌کند. در این روش مسیر نویزی معکوس شده و به کلمه‌ی صحیح می‌رسیم. به عبارتی احتمال را محاسبه کرده و کلمه با بیشترین احتمال را کلمه صحیح درنظر می‌گیریم. به عبارت ریاضی از رابطه زیر برای پیدا کردن محتمل‌ترین کلمه استفاده می‌کنیم. همان‌طور که مشاهده می‌شود، از قاعده‌ی بیز نیز استفاده شده است.

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}} P(w \mid x)$$

$$= \underset{w \in V}{\operatorname{argmax}} \frac{P(x \mid w) P(w)}{P(x)}$$

$$= \underset{w \in V}{\operatorname{argmax}} P(x \mid w) P(w)$$

c)

کد به پیوست ارسال شده است .

ابتدا تابع hamming طبق زیر نوشته می‌شود. در این تابع فرمول محاسبه‌ی فاصله همینگ پیاده‌سازی شده و دو رشته با طول برابر را دریافت می‌کند. کاراکترهای دو رشته را تک به تک با یکدیگر مقایسه کرده و هر کجا کاراکترها یکسان نباشد، شمارنده یک واحد افزایش پیدا می‌کند و در آخر نیز تعداد کاراکترهای نابرابر را برمی‌گرداند.

```
def hamming_distance(str1, str2):
    if len(str1) != len(str2):
        raise Exception('Inalid inputs, inputs should have same size')
    distance = 0
    for character1, character2 in zip(str1, str2):
        if character1 != character2:
            distance +=1
    return distance
```

نتیجه‌ی اجرای تابع hamming برای دو رشته کاراکتر با طول یکسان در زیر نشان داده شده است.

```
str1 = 'GGGCCGTTGGT'
str2 = 'GGACCGTTGAC'
print(hamming_distance(str1, str2))
```

```
3
```

قسمت زیر عملکرد خواسته شده در صورت سوال را اعمال می‌کند. این قسمت به این صورت عمل می‌کند که یک رشته‌ی dna، max_mismatch و pattern را به عنوان دریافت کرده و مکان‌هایی که در آن pattern دارای فاصله‌ی همینگ حداکثر به اندازه‌ی مقدار mismatch می‌باشد را برمی‌گرداند.

```
def pattern_matching(pattern, dna_string, max_mismatch):
    positions = list()
    for i in range(len(dna_string)-len(pattern)+1):
        distance = hamming_distance(pattern, dna_string[i:i+len(pattern)])
        if distance <= max_mismatch:
            positions.append(i)
    return positions
```

```
with open('prob1_c2.txt') as f:
    pattern , dna_string, max_mismatch = f.readlines()
pattern = pattern.strip()
dna_string = dna_string.strip()
max_mismatch = int(max_mismatch.strip())
print(pattern_matching(pattern, dna_string, max_mismatch))
```

```
[5, 8, 11, 26, 31, 60, 76, 79, 82, 124, 144, 152, 167, 176, 179, 191, 193, 205, 219, 247, 250, 255, 258, 265, 272, 275, 296, 298, 299, 301, 30
```

در قسمت سوم ابتدا رشته‌ای که طولش مطابق با عدد داده شده در فایل prob1_c3 بود و بیشترین تکرار را داشت پیدا می‌کند. از این رشته استفاده می‌شود تا رشته یاpattern ای انتخاب شود که دارای فاصله‌ی همینگ به اندازه‌ی mismatch گفته شده در فایل است. سپس الگویی با فاصله‌ی همینگ به اندازه‌ی پرتکرارترین رشته است می‌یابد. در انتها نیز خروجی نهایی چاپ می‌شود. خروجی نیز در انتها نشان داده شده است.

```
def occurrence_of_pattern(pattern, dna_string, max_mismatch):
    count = 0
    for i in range(len(dna_string)- len(pattern)+1):
        distance = hamming_distance(pattern, dna_string[i:i+len(pattern)])
        if distance <= max_mismatch:
            count +=1
    return count
```

```
def most_frequent_pattern(dna_string, pattern_length, max_mismatch):
    nucleotides = ['A','C','G','T']
    patterns = [''.join(nucleotide) for nucleotide in product(nucleotides, repeat=pattern_length)]
    frequencies = dict()
    for pattern in patterns:
        frequencies[pattern] = occurrence_of_pattern(pattern, dna_string, max_mismatch)
    return max(frequencies, key= frequencies.get)
```

```
with open('prob1_c3.txt') as f:
    dna_string, _ = f.readlines()
dna_string = dna_string.strip()
pattern_length, max_mismatch = _.strip().split(' ')
pattern_length = int(pattern_length)
max_mismatch = int(max_mismatch)
print(most_frequent_pattern(dna_string, pattern_length, max_mismatch))
```

```
ATTTA
```

Problem 2

a)

مدل Discriminative مرز تصمیم‌گیری بین کلاس‌ها را مدل می‌کند Generative .توزیع واقعی کلاس‌ها را مدل می‌کند یا به عبارتی توزیع joint بین داده و برچسب‌ها را یاد می‌گیرد. به عبارتی این مدل توزیع احتمال ورودی و برچسب را با استفاده از قانون بیز یاد می‌گیرد. از طرفی Discriminative احتمال شرطی را یاد می‌گیرد. همان‌طور که بیان شد مرز تصمیم بین نمونه‌ها را مدل می‌کنند.

b)

مدل‌های Discriminative داده‌های بیشتری برای آموزش نسبت به Generative نیاز دارند؛ زیرا مدل‌های Generative بیشتر biased هستند. چرا که فرضیات قوی‌تری را درنظر می‌گیرند که همان فرض استقلال شرطی می‌باشد. به‌طور کلی Generative مدل‌ها می‌توانند با missing dataها کار کنند ولی Discriminativeها نمی‌توانند و دلیل این است که Generativeها می‌توانند posterior را با marginalize کردن روی متغییرهای دیده نشده تخمین بزنند ولی در مورد مدل‌های Discriminative تمامی ویژگی‌های هر نمونه نیاز است.

c)

از رابطه‌ی زیر استفاده می‌شود.

$$P(c \mid d, \lambda) = \frac{\exp \sum_i \lambda_i f_i(c, d)}{\sum_{c'} \exp \sum_i \lambda_i f_i(c', d)}$$

برای shiraz :

$$(Location|Shiraz) = \frac{e^0}{e^0 + e^0 + e^0} = 0.333$$

$$P(Person|Shiraz) = \frac{e^0}{e^0 + e^0 + e^0} = 0.333$$

$$P(verb|Shiraz) = \frac{e^0}{e^0 + e^0 + e^0} = 0.333$$

برای Sue:

$$(Location|Sue) = \frac{e^0}{e^0 + e^{0.4} + e^0} = 0.286$$

$$P(Person|Sue) = \frac{e^{0.4}}{e^0 + e^{0.4} + e^0} = 0.427$$

$$P(Verb|Sue) = \frac{e^0}{e^0 + e^{0.4} + e^0} = 0.286$$

برای Pasargad:

$$(Location|Pasargad) = \frac{e^0}{e^0 + e^0 + e^0} = 0.333$$

$$P(Person|Pasargad) = \frac{e^0}{e^0 + e^0 + e^0} = 0.333$$

$$P(Verb|Pasargad) = \frac{e^0}{e^0 + e^0 + e^0} = 0.333$$

Problem 3

a)

با توجه به naïve bayes برای به دست آوردن کلاس یک موجودیت، تعداد زیادی اعداد احتمالاتی در هم ضرب می‌شوند. مقدار این اعداد بین صفر و یک است. در نتیجه عدد حاصل از ضرب این اعداد، عددی بسیار کوچک می‌شود که در قالب اعداد اعشاری استاندارد قابل ذخیره نیست و نتیجه صفر می‌شود. از این رو برای حل این مشکل از اعداد احتمالاتی لگاریتمی استفاده می‌شود که دلیل آن خاصیت لگاریتم در بزرگ کردن اعداد بین صفر و یک است. به عبارتی می‌توان اینگونه نیز بیان کرد که ضرب اعداد بین صفر و یک درهم دیگر بسیار کوچک شده و باعث می‌شود که نتیجه underflow شناخته شده و یک راه جلوگیری از این موضوع استفاده از لگاریتم است.

Macro:

$$precision = \frac{\frac{95}{111} + \frac{62}{73}}{2} = 0.85$$

$$recall = \frac{\frac{95}{107} + \frac{62}{69}}{2} = 0.89$$

$$F1 = \frac{2}{\frac{1}{p} + \frac{1}{r}} = \frac{2}{\frac{1}{0.85} + \frac{1}{0.89}} = 0.86$$

Micro:

$$precision = \frac{95 + 62}{95 + 62 + 16 + 11} = \frac{157}{184} = 0.85$$

$$recall = \frac{95 + 62}{95 + 62 + 12 + 7} = \frac{157}{176} = 0.89$$

$$F1 = \frac{2}{\frac{1}{p} + \frac{1}{r}} = \frac{2}{\frac{1}{0.85} + \frac{1}{0.89}} = 0.86$$

b)

کد به پیوست ارسال شده است.

c)

کد به پیوست ارسال شده است.

ابتدا importهای لازم انجام می‌شود.

```
# importing the libraries
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
import re
import numpy as np
import string
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('omw-1.4')
nltk.download('wordnet')
import string
from nltk.stem import WordNetLemmatizer
import collections
from collections import Counter
from sklearn.model_selection import train_test_split as tts
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

سپس تابع preprocess تعریف شده است که آنچه در صورت سوال از ما خواسته شده را در این تابع پیاده‌سازی کرده‌ایم. این تابع کلمات را از هم جدا کرده و به صورت یک لیست برمی‌گرداند.

```
######## Your Code Here ########
def preprocessing(data):
    lemmatizer = WordNetLemmatizer()

    # Lowercase
    data = data.lower()

    # Removing HTML tags
    data = re.sub(r"<.*?>", " ", data)

    # Removing hyperlinks
    data = re.sub(r"https?://\S+", "", data)

    # Removing numbers
    data = re.sub(r"\b[0-9]+\b\s*", "", data)

    # Removing punctuations
    data = re.sub(f"[{re.escape(string.punctuation)}]", "", data)

    # Spliting sentence to words
    words = data.split()

    # Applying word lemmatization
    words = [lemmatizer.lemmatize(word) for word in words]

    # Removing stopwords
    words = [word for word in words if word not in stopwords.words('english') and '_' not in word]

    return words
```

سپس ۲۰٪ داده‌ها را به مجموعه آزمایشی و ۸۰٪ را به مجموعه آموزشی تخصیص می‌دهیم.

```
######## Your Code Here ########
train_data, test_data = tts(data , test_size=0.2)
```

در تابع زیر مجموعه داده آموزشی و کلاس‌های آن دریافت شده و p(word|class) را برای تمام کلمات مجموعه آموزشی، تعداد تکرار هر کلمه و تعداد توکن یکتا در این مجموعه داده را برمی‌گرداند. همان‌طور که مشاهده می‌شود از naïve bayse استفاده شده است.

```
def naive_bayes_training(data, classes):
    probability = dict()
    count = dict()

    for c in classes:
        # find out probability of each class
        probability[c] = dict()
        probability[c]['count'] = len(data[data['sentiment']==c])
        probability[c]['probability'] = probability[c]['count']/len(data)

        # prepare data for counting occurrence of each word
        reviews = data[data['sentiment'] == c]['review'].to_list()
        tokens = list()
        for review in reviews:
            tokens.extend(review)

        # find out number of occurrence of each word in a class
        count[c]= dict()
        count[c]['token_count'] =collections.Counter(tokens)
        count[c]['len'] = len(tokens)

        # find out number of unique tokens in our dataset
    v = len(set(list(count['positive']['token_count'].keys())+ list(count['negative']['token_count'].keys())))

        # using naive bayes with add-1 smoothing rule for calculating the probability P(word|c)
    for class_name in classes:
        for word, c in count[class_name]['token_count'].items():
            probability[class_name][word] = (c+1)/ (v + count[class_name]['len'])

    return probability, count, v
```

تابع زیر برای گرفتن خروجی برای نمونه‌های آزمایش نوشته شده است.

```
######## Your Code Here ########
def naive_bayes_testing(model,count, v, X_test, classes):
    y_pred = list()
    for sample in X_test:
        p = dict()
        for class_name in classes:
            p[class_name] = np.log(model[class_name]['probability'])
            for word in sample:
                if word in model[class_name]:
                    p[class_name] += np.log(model[class_name][word])
                else:
                    p[class_name] += np.log(1 / (v + count[class_name]['len']))

        y_pred.append(max(p, key=p.get))
    return y_pred
```

```
y_pred = naive_bayes_testing(model, count, v, test_data.review, classes)
y_true = test_data.sentiment
```

در آخر برای به دست آوردن معیارهای ارزیابی خواسته شده، ابتدا TP، TN، FP و FN را به دست آورده و معیارهای خواسته شده را محاسبه می‌کنیم.

```
######## Your Code Here ########
TP = sum(real == pred and real == 'positive' for real, pred in zip(y_true,y_pred))
TN = sum(real == pred and real == 'negative' for real, pred in zip(y_true,y_pred))
FP = sum(real != pred and pred == 'positive' for real, pred in zip(y_true,y_pred))
FN = sum(real != pred and pred == 'negative' for real, pred in zip(y_true,y_pred))
```

Precision

```
[12] ######## Your Code Here ########
     p = {'positive' : TP/(TP+FN), 'negative': TN/(TN+FP)}
     print(p)
```

```
{'positive': 0.8389780154486037, 'negative': 0.8881034134518279}
```

Recall

```
[13] ######## Your Code Here ########
     r = {'positive' : TP/(TP+FP), 'negative': TN/(TN+FN)}
     print(r)
```

```
{'positive': 0.8843423799582464, 'negative': 0.8439539347408829}
```

F-measure

```
[14] ######## Your Code Here ########
     f= dict()
     f['positive'] = 2*p['positive']*r['positive']/ (p['positive'] + r['positive'])
     f['negative'] = 2*p['negative']*r['negative']/ (p['negative'] + r['negative'])
     print(f)
```

```
{'positive': 0.8610631161703425, 'negative': 0.8654659974411966}
```

Confustion matrix

```
######## Your Code Here ########
import matplotlib.pyplot as plt
confusion_matrix = [[TP,FN],[FP,TN]]
fig, ax = plt.subplots(figsize=(7.5,7.5))
ax.matshow(confusion_matrix)
labels = ['positive', 'negative']
for i in range(2):
    for j in range(2):
        ax.text(x=j, y=i,s=confusion_matrix[i][j], va='center', ha='center', size='xx-large')

ax.set_xticklabels(['']+labels)
ax.set_yticklabels(['']+labels)
plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```