به نام خدا

گزارش پروژه دوم درس هوش مصنوعی
دسته‌بندی آگهی‌های دیوار با استفاده از بیز

علی خوش‌طینت ۸۱۰۱۹۶۴۶۲

هدف پروژه:

در این پروژه قصد داریم تا با استفاده از قوانین بیز، آگهی‌های دیوار را دسته‌بندی کنیم.

داده‌های لود شده:

در این پروژه ما دو دیتاست را لود میکنیم. یکی مربوط به train و یکی هم برای test. دیتای train شامل چندین ردیف میباشد که هر ردیف شامل تیتر آگهی، متن آگهی و دسته‌بندی آن میباشد. دیتای test نیز همین ساختار را دارد. ما در این پروژه سعی میکنیم ستون categories را برای دیتای test پیش‌بینی کنیم و ببینیم پیش بینی مان تا چه اندازه درست بوده.

فاز اول - پیش پردازش داده:

در ابتدا باید مقداری پیش پردازش روی آگهی ها انجام شود. به این منظور سعی میکنیم برخی کلمات که تاثیری در انتخاب category ندارند را حذف کنیم. این کلمات را stop_word مینامیم. که در کد موجود بوده و شامل علامت‌ها مانند '؟' و کلماتی مانند 'با' 'در' 'و' یا کاراکترهایی مانند '\r' میباشند. همچنین پیش از حذف stop word ها سعی میکنیم که متن را normalize نیز بکنیم. به این معنی که فاصله‌ها را تبدیل به نیم‌فاصله کرده و برخی اصلاحات نگارشی روی آن انجام میدهد.

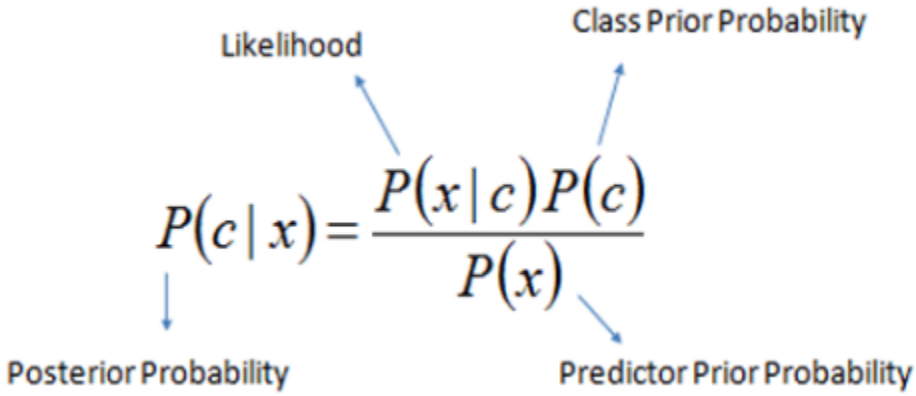
سپس کلمات عنوان و توضیحات هر آگهی استخراج کرده و سعی میکنیم به ازای هر category ، ببینیم چه کلماتی با چه فرکانسی تکرار شده اند.

```
normalizer = Normalizer()
def extract_words(sentence):
    return [w for w in word_tokenize(normalizer.normalize(sentence)) if w not in stop_words]


def add_to_category_dict(category, words):
    for word in words:
        if word in category_words[category]:
            category_words[category][word] += 1
        else:
            category_words[category][word] = 1
    category_words_count[category] += 1


for index, row in train_data.iterrows():
    row_category = row['categories']
    title_words = extract_words(row['title'])
    desc_words = extract_words(row['description'])
    add_to_category_dict(row_category, set(title_words + desc_words))
```

در تصویر بالا ما به ازای هر category، کلمات و تعداد آنها را مشخص کرده‌ایم. همچنین یک دیکشنری نیز برای تعداد کل کلمات یک category مشخص کرده‌ایم که در آینده از همه اینها استفاده میکنیم.

سوال۱. در stemming سعی میکنیم یک کلمه را به ریشه آن تبدیل کنیم. اما در بیشتر مواقع صرفا یک پسوند یا پیشوند حذف میشود. در واقع در اینجا به معنی زیاد توجهی نمیشود و صرفا حروفی را از کلمه حذف میکند. اما در lemmization به معنی نیز توجه میشود و ریشه برگردانده شده درست‌تر میباشد. بنابراین نیازمند دانش زبانی نیز میباشد تا بتواند دیکشنری‌هایش را بسازد و تلاش میکند تا ریشه کلمه را درست برگرداند.

سوال ۲.



احتمال اینکه پیشامد c رخ دهد به شرطی که x رخ داده باشد را Posterior میگوییم. در مساله ما به این معنی میباشد که چقدر احتمال دارد یک آگهی متعلق به کتگوری c باشد.

برای Likelihood میتوانیم بگوییم احتمال اینکه یشامد x رخ دهد با فرض اینکه c رخ داده باشد. که در مساله ما به معنی احتمال رخداد آگهی (مجموع تیتر و متن) در یک کتگوری c میباشد. که ما برای هر ۶ کتگوری این احتمال را حساب میکنیم. برای محاسبه Likelihood باید به ازای تک تک کلمات، احتمال کتگوری c را حساب کنیم. یعنی حساب کنیم این کلمه با چه احتمالی متعلق به c است. که میشود:

$$P(x1|c) * P(x2|c) * P(x3|c) .... * p(xn|c)$$

که هر کدام از xi ها نشان‌دهنده یک کلمه است. برای محاسبه احتمال کلمه در کتگوری نیز سهم آن کلمه (تعداد تکرار در کتگوری) به کل کلمات آن کتگوری را محاسبه میکنیم.

برای Prior میتوان گفت که احتمال رخداد کتگوری است. با توجه به اینکه میدانیم در دیتای train همه کتگوری ها به تعداد یکسانی آگهی دارند و احتمال P(c) به ازای هر category برابر ۱/۶ است، پس میتوانیم از آن صرفنظر کنیم.

احتمال xi که برای به روزرسانی prior استفاده میشود را Evidence میگویند. در مساله ما به معنی احتمال رخداد کلمه در کتگوری میباشد.

حل مساله با بیز:

برای حل این مساله ما نیاز به متغیرهایی داشتیم که در بخش ارزیابی مورداستفاده قرار میگرفت:

```python
from dataclasses import dataclass


@dataclass
class PredicateData:
    total: int
    correct_detected: int
    all_detected_class: Dict[str, int]
    total_class: Dict[str, int]
    correct_detected_class: Dict[str, int]
```

اولی به معنی همه ردیف‌های دیتاست است. دومی به معنی همه تخمین‌های درست، سومی به معنای همه ردیف‌های هر کتگوری و آخری به معنی درست تخمین زده‌های یک کتگوری میباشد. در اینجا سعی شده نامگذاری مانند صورت پروژه باشد.
سپس سعی میکنیم فرمول زیر را پیاده‌سازی کنیم:

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

```
test_data = pd.read_csv('Data/divar_test.csv')


def predict_category(smoothing = False):
    predicate_data = PredicateData(
        0,
        0,
        make_dict_with_cats(),
        make_dict_with_cats(),
        make_dict_with_cats()
    )


    for index, row in test_data.iterrows():
        all_text_words = extract_words(row['title']) + extract_words(row['description'])
        max_cat = '' if smoothing == True else categories[random.randint(0, 5)]
        max_prob = 0
```

```
            max_cat = category
    if max_prob == 0:
        max_cat = categories[random.randint(0,5)]
    predicate_data.total += 1
    predicate_data.all_detected_class[max_cat] += 1
    predicate_data.total_class[row['categories']] += 1
    if max_cat == row['categories']:
        predicate_data.correct_detected += 1
        predicate_data.correct_detected_class[max_cat] += 1
return predicate_data
```

```
P_c_X = {}
for category in categories:
    P_c_X[category] = 1
    for word in all_text_words:
        P_xi_c = 0
        if word not in category_words[category]:
            category_words[category][word] = 0
        if smoothing:
            P_xi_c = (category_words[category][word] + 1) / \
                (category_words_count[category] + len(category_words[category]) + 1)
        else:
            P_xi_c = category_words[category][word]/category_words_count[category]
        P_c_X[category] *= P_xi_c

    P_c_X[category] *= 1/6
    if P_c_X[category] > max_prob:
        max_prob = P_c_X[category]
        max_cat = category
if max_prob == 0:
    max_cat = categories[random.randint(0,5)]
```

Biagram

سوال ۳.

مثال ۱ - این ماشین هزار بار کارشناسی شده

مثال ۲ - این قفسه بار سنگینی دارد.

در مثال بالا شاید بتوان گفت که با biagram نیز میتوان تشخیص داد که بار متعلق به کدام دسته است. مثلا اگر قبل از بار عدد یا چند بیان شده بود به معنی دفعه بوده و اگر نشده بود معنی دیگری دارد. اگر مثال سخت تر شود شاید نیاز باشد از ngram استفاده کرد.

Additive Smoothing

سوال ۴. یکی از مشکلاتی که در بخش حل مساله با بیز داشتیم این بود که به ازای یک آگهی ممکن بود برخی کلمات وجود داشته باشند که در دیتای train، اصلا وجود نداشته و متعلق به هیچ گروهی نبوده اند. در این صورت احتمال آن آگهی و کلمه صفر شده و در نتیجه با توجه به ضرب زیر، احتمال کتگوری به ازای آگهی صفر میشود.

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

حتی ممکن است کلمه متعلق به هیچ کتگوری نبوده باشد که در این صورت برای همه کتگوری ها صفر میشود.
مشکل دیگر این است که ممکن است یک کلمه در TRAIN، فقط در یک کتگوری آورده شده باشد. که در این صورت در test ، با قطعیت تشخیص میدهیم که کتعلق به همان کتگوری است. چراکه احتمال اینکه کلمه متعلق به یک کتگوری باشد میشود سهم (تکرار) کلمه در کتگوری تقسیم بر همه تکرارهای کلمات آن کتگوری. و وقتی که در هیچ کتگوری دیگری نیامده به این معنی است برای بقیه صفر میشود. ولی میدانیم که ممکن است اینگونه نباشد و کلمه متعلق به کتگوری دیگری نیز باشد.

سوال ۵.

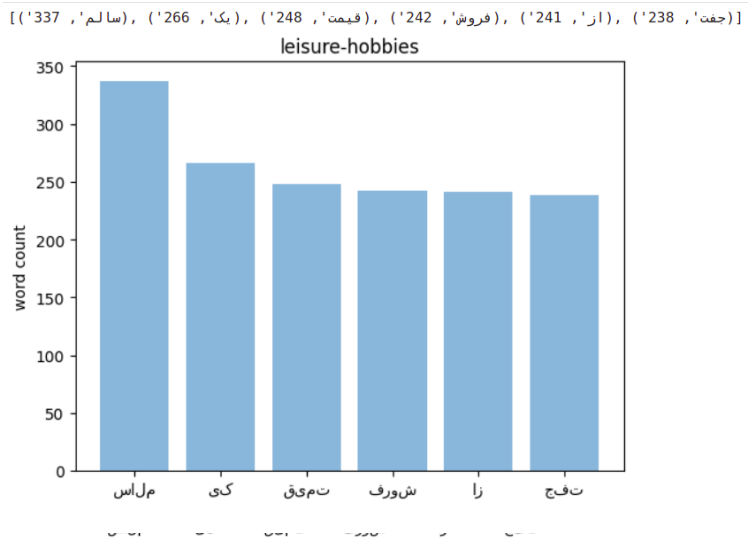
در روش Additive Smoothing به این صورت عمل میکنیم که به جای اینکه بگوییم احتمال کلمه به ازای کتگوری برابر است با تکرار کلمه در کتگوری تقسیم بر کل تکرارها میگوییم:

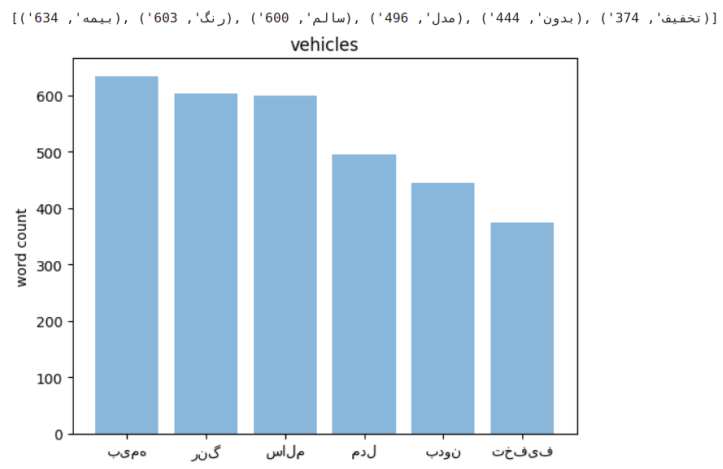
$$P(w|c) = (count(w,c) + 1) / (count(c) + |v| + 1)$$

با این کار دیگر هیچ گاه صورت کسر صفر نمیشود و در نتیجه هیچ گاه این احتمال صفر نخواهد شد. در مخرج نیز علاوه بر کل کلمات c، تعداد واژه‌های موجود در دایره لغات را نیز جمع میزنیم و بعلاوه یک میکنیم. که این یک برای اضافه شدن کلمه unknown جدید است. بنابراین اگر یک کلمه در یک کتگوری موجود نباشد احتمال اینکه این آگهی متعلق به این کتگوری نباشد صفر نمیشود.
ما در تابع predict_category به این صورت smoothing را دخیل کرده‌ایم:

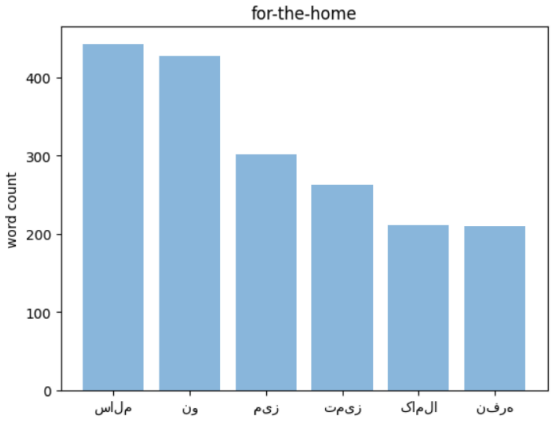
```
for category in categories:
    P_c_X[category] = 1
    for word in all_text_words:
        P_xi_c = 0
        if word not in category_words[category]:
            category_words[category][word] = 0
        if smoothing:
            P_xi_c = (category_words[category][word] + 1) / \
                (category_words_count[category] + len(category_words[category]) + 1)
        else:
            P_xi_c = category_words[category][word]/category_words_count[category]
        P_c_X[category] *= P_xi_c
```

سوال ۶.

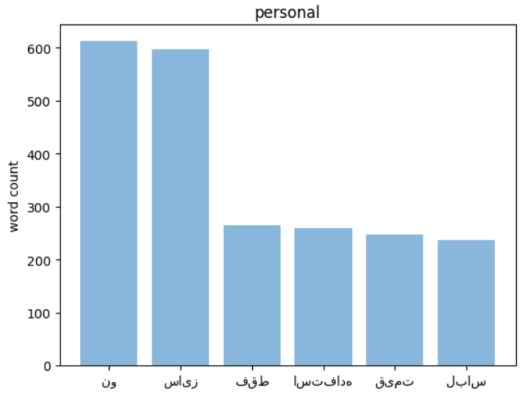




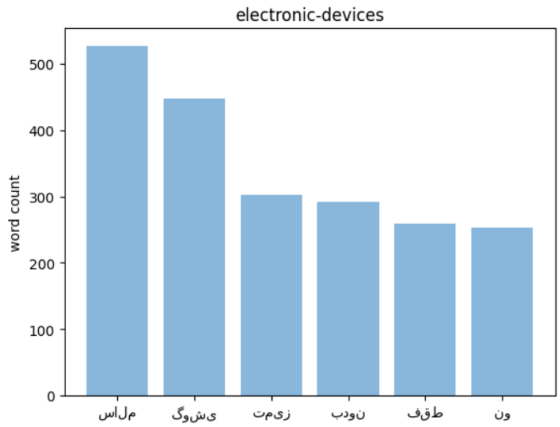
[('442', 'سالم'), ('427', 'نو'), ('302', 'میز'), ('263', 'تمیز'), ('211', 'کاملا'), ('210', 'نفره')]

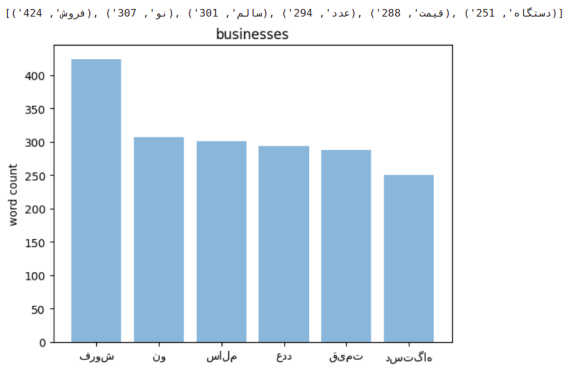


[('613', 'نو'), ('597', 'سایز'), ('265', 'فقط'), ('260', 'استفاده'), ('248', 'قیمت'), ('237', 'لباس')]



[('527', 'سالم'), ('448', 'گوشی'), ('302', 'تمیز'), ('292', 'بدون'), ('259', 'فقط'), ('253', 'نو')]





ارزیابی

سوال ۷.

معیار precision به معنی تخمین‌های درست به کل تخمین میباشد. در حالیکه recall تعدد کل تخمین‌های درست به کل داده‌های آن کلاس میباشد. در رابطه با precision ممکن است تعداد بسیاری از داده‌هایی که بررسی کردیم متعلق به کلاس خاصی هستند واقعا متعلق به کلاس دیگری باشند. البته در مساله ما تفاوت چشمگیری بین این دو وجود ندارد ولی همیشه precision به تنهایی قابل اعتماد نیست.

سوال ۸.

با توجه به اینکه هر کدام از precision و recall به تنهایی شاید کافی نباشد، معیار F1 سعی میکند این دو را با یکدیگر ترکیب کند. که برای این کار از میانگین هارمونیک استفاده میکند. که این روش از وقوع حالت‌های extreme جلوگیری میکند. در واقع اگر دو تا معیار خیلی اختلاف داشته باشند عدد نهایی بیشتر تغییر میکند. مثلا یکی ۱ و دیگری ۰ باشد، میانگین ۰.۵ میشود در حالیکه F1 برابر ۰ میشود.

سوال ۹.

Macro: در میانگین macro، مجموع F1 های کتگوری‌های مختلف را گرفته و به تعداد کل کتگوری ها تقسیم میکنیم. در واقع داریم از F1 های کتگوری‌ها میانگین میگیریم.

Micro: در میانگین micro، تعداد کل تشخیص های درست را به تعداد کل تشخیص های دست و نادرست تقسیم میکنیم.

Weighted: برای محاسبه Weighted، باید F1 را نیز به صورت weightedاب کنیم. به این صورت که به ازای هر کتگوری، f1_score را در تعداد کل داده‌های مربوط به آن کتگوری نیز ضرب میکنم. نهایتا مجموع هر ۶ تا weighted_f1 را حساب کرده و تقسیم بر تعداد کل داده ها میکنیم.

سوال ۱۰.

الف) بدون استفاده از Smoothing:

```
evaluation(predict_data)
```

```
Category         Precision       Recall          F1 Score
leisure-hobbies 0.3716475095785441       0.3233333333333333      0.34581105169340465
vehicles        0.41114982578397213      0.3933333333333333      0.40204429301533223
for-the-home    0.3987138263665595       0.4133333333333333      0.4058919803600655
personal        0.40804597701149425      0.47333333333333333     0.43827160493827166
electronic-devices      0.3298611111111111       0.31666666666666665     0.32312925170068024
businesses      0.3770491803278688       0.38333333333333336     0.38016528925619836
================================================================
Result of the Naive Bayes without without additive smoothing
accuracy: 0.3838888888888889
macro_avg_f1: 0.3825522451606587
micro_avg_f1: 0.3838888888888889
weighted_avg_f1: 0.3825522451606588
```

ب) با استفاده از Smoothing:

```
evaluation(predict_data)
```

```
Category         Precision       Recall          F1 Score
leisure-hobbies 0.9148148148148149       0.8233333333333334      0.8666666666666667
vehicles        0.8914473684210527       0.9033333333333333      0.8973509933774835
for-the-home    0.827922077922078        0.85    0.8388157894736843
personal        0.8539682539682539       0.8966666666666666      0.8747967479674796
electronic-devices      0.878125         0.9366666666666666      0.9064516129032257
businesses      0.8021201413427562       0.7566666666666667      0.7787307032590051
Result of the Naive Bayes without without additive smoothing
accuracy: 0.8611111111111112
macro_avg_f1: 0.8604687522745907
micro_avg_f1: 0.8611111111111112
weighted_avg_f1: 0.860468752274591
```

سوال ۱۱. پس از استفاده از Smoothing نتایج تغییرات چشمگیری داشته اند. چراکه برای کلمات جدید احتمال فرضی در نظر گرفته شده ولی قابل مقایسه با بقیه کتگوری‌ها میباشد.