مقدمه:

در این پروژه می‌خواهیم با استفاده از قانون شبکه‌های بیزین یم مدل ای را آموزش دهیم که با دیدن یک متن از اخبار تشخیص دهد که آن خبر در چه دسته‌ای طبقه بندی می شود. این کار را با استفاده از دیتا ست آموزش انجام می‌دهیم و بعد از ارزیابی آن با همین مدل آموزش دیده یک دیتاست تست را نیز پیش‌بینی می کنیم.

پیش پردازش داده ها:

قبل از شروع برای آموزش دادن مدل با دیتا ست نیاز است تا داده‌های خود را تمییز کنیم تامدل ما بتواند با سرعت و دقت خوبی به پردازش با آن داده‌ها بپردازد.
برای این پروژه داده‌های ما که درون اکسل ای قرار دارد را می‌خواهیم پردازش کنیم این داده‌ها شامل ستون‌های مختلف ای است ولی در اینجا ما از ستون‌های Headline, Short Description برای آموزش مدل استفاده می‌کنیم تا مدل ما به درستی بتواند طبقه آن خبر را که در ستون Category آمده است به درستی تشخیص دهد
این داده‌ها شامل یک سری متن است که آن‌ها شامل کلمات ای است که می‌خواهیم در مدل خود با آن‌ها آموزش دهیم
برای پیش پردازش داده‌ها ابتدا باید آن‌ها را از روی فایل بخوانیم و ستون‌های مربوطه را جدا کنیم برای این کار داریم:

```python
class DataExtractor:
    def __init__(self, data_file_path):
        self.data_frame = pd.read_csv(data_file_path)

    def data_producer(self, type_of_data):
        data = self.data_frame['headline'].fillna('') + ' ' + self.data_frame['short_description'].fillna('')
        if type_of_data == 'train':
            target = self.data_frame["category"]
            return data, target
        else:
            return data
```

کد بالا دیتا را از فایل میخواند و ستون‌های مربوطه را جدا می‌کند به این صورت که داده ی ما شامل ۲ ستون Headline, Description و جواب ما ستون Category خواهد بود.

حال داده‌های ما شامل یک متن است که باید ابتدا آن‌ها را تمییز کنیم برای این کار به ترتیب به صورت زیر عمل می کنیم:

۱) تبدیل همه ی حروف بزرگ به حروف کوچیک:

ابتدا درون متن همه ی حروف را به حروف کوچک تبدیل می کنیم. این کار باعث می‌شود که کلمات ما یکپارچه شود یعنی وقتی در متن به دنبال کلمه‌ای می‌گردیم یا به کلمه‌ای می‌رسیم می‌دانیم که قبلاً در متن در کدام دسته بوده است ولی اگر مثلاً حرف اول آن بزرگ باشد و در جایی دیگر حرف اول کوچک باشد مدل ممکن است تشخیص ندهد که همان کلمه است و به اشتباه دسته را تشخیص دهد. این کار سرعت گشتن در متن را نیز بیشتر می ند چرا که دیگر برای هر کلمه به صورت جدا نیاز نیست چک شود که آیا این کلمه با حرف بزرگ و یا کوچی استفاده شده است یا خیر و متن ما به این صورت یکپارچه تر است:

```
def __text_lower_case(self):
    self.data_frame = self.data_frame.str.lower()
```

۲) حذف اعداد از متن:

سپس می توانییم همه ی اعداد را از متن حذف کنیم چرا که اعداد می‌توانند در هر دسته‌ای باشند و به یادگیری مدل ما کمک نمی‌کند و سرعت گشتن را نیز کم می‌کند پس می‌توانیم با این کار سرعت را بیشتر کرده و در عین حال به متنی یکپارچه تر برسیم

```
def __remove_numbers(self):
    self.data_frame = self.data_frame.str.replace('\d+', '')
```

۳) حذف علایم نگارشی:

مدل ما به دنبال کلمات درون اخبار ها می‌گردد و سعی دارد با کلمات طبقه را تشخیص دهد پس علایم نگارشی کمکی به مدل ما نمی‌کند و حتی ممکن آن‌ها را کلمه تشخیص دهد که باز هم در پیدا کردن طبقه درست دچار مشکل می‌شود

```
def __remove_punctuation(self):
    self.data_frame = self.data_frame.str.replace('[^\w\s]', '')
```

۴) حذف کلمات پر کاربرد و ایست واژه ها:
در هر متن و نوشته‌ای ایست واژه‌های زیادی وجود دارد که این‌ها کلمات اصلی و معنا داری نیستند و همین‌طور به تعداد زیاد تکرار شده‌اند. این واژه‌ها در همه ی طبقات خبری وجود دارند پس حتی ممکن است دقت و سرعت مدل ما را کاهش دهند پس بهتر است با حذف آن‌ها را متنی یکپارچه تر برسیم

```
def __remove_stop_words(self):
    stop_word = stopwords.words('english')
    self.data_frame = self.data_frame.apply(lambda x: [item for item in x.split() if item not in stop_word])
```

۵)تبدیل کلمات به ریشه‌های آن‌ها :
برای اینکه مدل ما تشخیص دهد یک کلمه قبلاً در آن دسته از متن استفاده شده است یا نه نیاز دارد تا کلمات دقیقاً مثلهم باشند و نه مشتقی از یکدیگر برای همین نیاز است تا کلمات را به ریشه ی آن‌ها تغییر دهیم تا مدل بتواند با بالاترین دقت کلمات را تشخیص داده و خبر ها را طبقه بندی کند برای اینکه به ریشه کلمات برسیم ۲ روش وجود دارد:

۱) Stemming:
در این روش کلمات به حالت ساده ی آن‌ها و یا ریشه آن‌ها بدون مشتق تبدیل می شود. این روش کلیه مشتق های یک کلمه مانند ing, s, .. را حذف می‌کند و به کلمه اصلی می رسد. این کار با حذف مشتقات کلمه ها به کلمه های کوچک‌تر و مشترک تر می‌رسد ولی نمی‌تواند مثلاً زمان افعال یا حالت اصلی کلمه را پیدا کند صرفاً آن مشتق از کلمه را حذف می کند.

```
@staticmethod
def __stemming_text(text):
    stemmer = PorterStemmer()
    return [stemmer.stem(w) for w in text]
```

۲) Lemataization:
این روش همانند روش stemming سعد دارد تابه ریشه کلمات برسد ولی بر خلاف آن فقط مشتقات را حذف نمی‌کند بلکه با دانشی که از قبل از کلمات دارد ریشه اصلی کلمه در حالت ساده را پیدا می کند. این روش به کلمات ساده‌تری می‌رسد یعنی حالت مشترک بین کلمات بیشتر می‌شود و مدل می‌تواند به دقت بیشتری نیز برسد

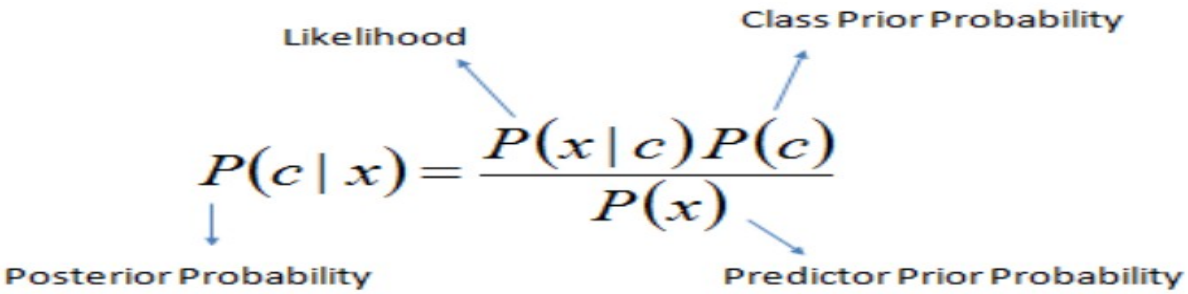
```
def __do_lemmatization(self):
    self.data_frame = self.data_frame.apply(self.__lemmatize_text)
```

تأثیر استفاده از Lemataization بیشتر است چرا که در این روش همه کلمات به حالت ساده خود تبدیل می‌شوند و حتی اگر زمان های آن‌ها یا مشتقات آن‌ها نیز در متن آمده باشد باز هم یکسان می‌شوند و متن یکپارچه تر می‌شود ولی در حالت stem چون ممکن است زمان یا مشتق کلمه فرق داشته باشد ممکن از یک کلمه ۲ مدل داشته باشیم و مدل ما نتواند تشخیص دهد که آن‌ها یکسان هستند و دقت آن کاهش یابد

پردازش متن با قاعده بیزین:

در این قسمت می‌خواهیم با استفاده از متن ای که در قسمت قبل تمیز کردیم استفاده کنیم و با استفاده از قاعده بیزین تشخیص دهیم که هر کدام از اخبار مربوط به کدام دسته قرار دارد

برای این کار ابتدا باید از فرمول شبکه بیزین استفاده کنیم فرمول به صورت زیر است:



در این پروژه P(c) یا Class Prior احتمال وقوع هر کلاس است که با تقسیم تعداد آن کلاس به کل تعداد کلاس‌ها بدست می‌آید

P(x|c) یا Liklihood احتمال وقوع هر لغت در متن به شرط آن کلاس است. یعنی احتمالی که آن لغت در آن کلاس بیاید.

P(x) احتمال وقوع آن لغات در کل است که در این پروژه یک مقدار ثابت برای همه کلاس‌ها است که می‌توان ار را حذف کرد

P(c|x) یا Posterior احتمال کلاس به شرط لغات آن است یعنی اگر آن لغات بیاید به چه احتمالی کلاس ما c می شود

ابتدا برای این پروژه می‌خواهیم ۲ دسته داده را از هم جدا کنیم و سپس در قسمت بعد دسته ۳ را به آن اضافه کنیم و مدل را آموزش دهیم تا ۳ دسته را از هم جدا کند
برای این کار باید داده‌هایی را که از اکسل مربوط به آن خواندیم به ۲ دسته ارزیابی و آموزش تقسیم کنیم تا بتوانیم با داده‌های آموزش شبکه را آموزش دهیم و با داده ی ارزیابی آن را ارزیابی کرده و میزان خصوصیات شبکه از جمله دقت آن را بررسی کینم تا بتوانیم به شبکه بهتری دست پیدا کنیم
برای این کار ابتدا داده‌ها را از جدول ها می خوانیم:

```
train, target = DataExtractor('data.csv').data_producer('train')
test = DataExtractor('test.csv').data_producer('test')
```

و سپس از داده‌های آموزش ۲ دسته داده به دست می اوریم:

```
(x_train, y_train), (x_validation, y_validation) = text_classifier.split_validation_data(train, target)
```

برای جدا کردن داده‌های ارزیابی باید ابتدا دسته ها را جدا کرده و سپس از هر دسته ۸۰ درصد را برای آموزش و ۲۰ درصد را برای ارزیابی استفاده کنیم. اگر بخواهیم از کل داده‌ها ۲۰ درصد را جدا کنیم این ۲۰ درصد ممکن است فقط از ۱ طبقه یا ۲ طبقه باشند و همه ی طبقات را در بر نگیرند که در این صورت ارزیابی ما درست نموده زیرا مثلاً یک طبقه را در نظر نگرفته‌ایم این کار باعث می‌شود ارزیابی ما از همه ی طبقات درست باشد و همین‌طور تعداد مناسبی از هر طبقه را برای ارزیابی داریم تا به بهترین نتایج از وضعیت مدل خود برسیم

```
for i in range(len(self.classes)):
    class_train_data[i], class_validation_data[i], class_train_target[i], class_validation_target[
        i] = train_test_split(train_data[i], train_target[i], test_size=0.2)
```

برای ارزیابی شبکه از ۳ معیار استفاده می کنیم:

$$\text{Recall} = \frac{\textit{Correct Detected Category}}{\textit{All Category}}$$

این معیار نشان می‌دهد که برای هر طبقه تعدادی از آن طبقه که درست پیش‌بینی شده است نسیت به کل تعداد آن طبقه چند است واضح است که اگر این مقدار ۱ باشد یعنی تمام حالت‌هایی

که آن طبقه تکرار شده است توسط مدل ما به درستی پیش‌بینی شده است. این معیار را برای تمام طبقه ها حساب می‌کنیم

$$\text{Precision} = \frac{\textit{Correct Detected Category}}{\textit{Detected Category (This includes wrong detections)}}$$

معیار بعدی Precision نام دارد که نشان می‌دهد برای هر طبقه چقدر از آن در طبقه های دیگر پیش‌بینی شده است. این معیار حصل تقسیم تعداد ای که مدل به درستی از آن طبقه پیش‌بینی کرده است بر تعداد پیش‌بینی ها از آن طبقه است. این معیار نشان می‌دهد چقدر از هر طبقه به اشتباه در طبقه های دیگر پیش‌بینی شده است واضح است اگر این مقدار ۱ باشد به این معنی است که همه حالت‌های پیش‌بینی شده برای آن طبقه به درستی پیش‌بینی شده بوده است

$$\text{Accuracy} = \frac{\textit{Correct Detected}}{\textit{Total}}$$

معیار آخر دقت کلی شبکه است. این معیار نشان می‌دهد چقدر از داده‌های شبکه به درستی طبقه بندی شده است که می‌تواند دقت شبکه را به ما بدهد

بعد از اینکه داده‌های ارزیابی و آموزش را از هم جدا کردیم ابتدا داده‌ها را با استفاده از کد قبلی که زده بودیم تمییز می‌کنیم تا بتوانیم آن‌ها را به مدل خود دهیم این کار را برای هر ۳ دسته داده ی آموزش ارزیابی و تست باید انجام دهیم زیرا قرار است این داده‌ها را به شبکه دهیم و برای اینکه دقت شبکه بالا رود باید داده‌های آن تمیز و یکپارچه باشند

```
cleaned_x_train = TextProcessing(x_train).clean_text()
cleaned_x_validation = TextProcessing(x_validation).clean_text()
cleaned_x_test = TextProcessing(test).clean_text()
```

بعد از آن داده‌های آموزش را به مدل خود می‌دهیم تا با آن‌ها آموزش ببیند و بتواند برای اخبار ای که آن‌ها را ندیده است تعمیم یابد برای این کار از قانون بیزین استفاده می کنیم:

```
text_classifier.fit(cleaned_x_train, y_train)
```

```
def fit(self, data, target):
    total_number_of_text = len(data)
    for class_type in self.classes:
        self.num_texes[class_type] = sum(1 for label in target if label == class_type)
        self.log_class_priors[class_type] = math.log(self.num_texes[class_type] / total_number_of_text)
        self.word_counts[class_type] = {}

    for x, y in zip(data, target):
        if not self.classes.__contains__(y):
            continue
        class_type = y
        counts = self.__get_word_counts(x)
        for word, count in counts.items():
            if word not in self.vocab:
                self.vocab.add(word)
            if word not in self.word_counts[class_type]:
                self.word_counts[class_type][word] = 0.0
            self.word_counts[class_type][word] += count
```

برای اینکه از قاعده بیز استفاده کنیم فرض کردیم که کلمات به هم رب ندارند یعنی احتمال اینکه یک خبر به ازای کلمات آن طبقه بندی شود برابر می‌شود با احتمال تک تک آن کلمات به شرط دسته ی آن خبر ضرب در احتمال آن دسته مانند زیر:

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

درواقع فرض کرده‌ایم که کلمات به هم ربط ندارند و احتمال آن‌ها به شرط بقیه کلمات مستقل است

حال برای اینکه این احتمال را بدست اوریم چون می‌دانیم این اعداد احتماء خیلی کوچک هستند و با ضرب آن‌ها به اعداد کوچک‌تری می‌رسیم ممکن است عدد انقدر کوچک شود که گرد کردن آن توسط کامپیوتر به ۰ برسد برای همین از دو طرف تساوی لگاریتم می‌گیریم تا بتوانیم این اعداد را بزرگ‌تر کنیم و از حالت ضرب چند عدد کوچک به جمع چند عدد بزرگ‌تر برسیم و البته می‌دانیم نیاز نیست این لگاریتم را برگردانیم زیرا مدل ما هر احتمالی که بزرگ‌تر شد آن طبقه را برای خوروجی در نظر می‌گیرد و چون لگاریتم نیز خطی است عدد با لگاریتم بزرگ‌تر احتمال بیشتری دارد و چون مخرج کسر نیز ثابت است دیگر نیاز

به محاسبه آن نداریم و فقط همان فرمول بالا را محاسبه کرده و بیشترین احتمال را برای هر کلاس در نظر می‌گیریم
حال برای محاسبه آن نیاز داریم تا احتمال هر کلاس را محاسبه کنیم برای این کار باید تعداد خبر های هر کلاس را به کل اخبار تقسیم کنیم در این صورت prior مربوط به کلاس‌های ما بدست می‌آید
سپس به ازای هر ورودی برای هر کلمه آن یک کلمه به لغت‌نامه خود اضافه می‌کنیم و تعداد تکرار آن کلمه را برای همان کلاس خود نگه می‌داریم در این صورت مدل ما می‌داند اگر آن کلمه دوباره تکرار شود برای کدام کلاس احتمال آن بیش تر است و می‌تواند کلاس مورد نظر را به درستی تقسیم‌بندی کند

بعد از آموزش کلاس نیاز داریم با داده‌های ارزیابی معیار های مورد نظر خود را بدست اوریم و مدل را ارزیابی کنیم

```python
def evaluate(self, validation_data, validation_target):
    result = self.predict(validation_data)
    all_category = dict.fromkeys(self.classes, 0)
    detected_category = dict.fromkeys(self.classes, 0)
    correct_detected_category = dict.fromkeys(self.classes, 0)
    total = len(result)
    for i in range(len(validation_target)):
        for class_type in self.classes:
            if class_type == validation_target[i] and validation_target[i] == result[i]:
                correct_detected_category[class_type] += 1
            if class_type == validation_target[i]:
                all_category[class_type] += 1
            if class_type == result[i]:
                detected_category[class_type] += 1

    recall = dict.fromkeys(self.classes, 0.0)
    precision = dict.fromkeys(self.classes, 0.0)

    for class_type in self.classes:
        recall[class_type] = correct_detected_category[class_type] / all_category[class_type]
        precision[class_type] = correct_detected_category[class_type] / detected_category[class_type]
    accuracy = sum(correct_detected_category.values()) / total
    print('Recall is: ', recall)
    print('Precision is: ', precision)
    print('Accuracy is: ', accuracy)
```

برای این کار ابتدا برای همه ی داده‌ها خروجی آن‌ها را با مدل خود پیش‌بینی می‌کنیم سپس برای هر داده پیش‌بینی شده در هر کلاس تعداد درست پیش‌بینی شده تعداد آن کلاس و تعداد پیش‌بینی شده برای آن کلاس را بدست می‌آوریم

سپس با استفاده تز فرمول های معیار ها که در بالا توضیح داده شد این معیار ها را بدست می‌آوریم و از وضعیت شبکه مطلع می شوریم
برای پیش‌بینی کلاس داده‌ها که هم برای داده‌های تست و هم برای داده‌های ارزیابی استفاده کرده‌ایم به صورت زیر است:

```
def predict(self, tokenized_test):
    result = []
    for x in tokenized_test:
        counts = self.__get_word_counts(x)
        class_type_score = dict.fromkeys(self.classes, 0.0)
        for word, _ in counts.items():
            if word not in self.vocab:
                continue

            for class_type in self.classes:
                log_w_given_class = math.log((self.word_counts[class_type].get(word, 0.0) + 1) / (
                        self.num_texes[class_type] + len(self.vocab)))
                class_type_score[class_type] += log_w_given_class

        for class_type in self.classes:
            class_type_score[class_type] += self.log_class_priors[class_type]

        class_type_score = {k: v for k, v in sorted(class_type_score.items(), key=lambda item: item[1])}
        result.append(list(class_type_score.keys())[-1])
    return result
```

که در آن به ازای هر ورودی ابتدا کلمات و تعداد استفاده آن کلمه را بدست می‌آوریم سپس به ازای این کلمات ابتدا اگر در لغت‌نامه وجود داشت برای هر کلاس تعداد بار تکرار شده آن کلمه را بر تعداد کل تکرار آن متن تقسیم می‌کنیم و با احتمال آن کلاس جمع می‌کنیم این همان قاعده بیز است که از آن لگاریتم گرفته‌ایم .
برای اینکه مطمین شویم که صورت لگاریتم ما ۰ نمی‌شود از قاعده لاپلاس استفاده کرده‌ایم به این صورت که صورت لگاریتم را با ۱ جمع کرده‌ایم و مخرج را با تعداد کل لغات جمع می‌کنیم بعد از پیدا کردن احتمال برای هر کلاس آن احتمال ای که بزرگ‌تر باشد نتیجه می‌دهد که متن داده شده برای آن کلاس پیش‌بینی می‌شود

بعد از پیدا کردن معیار های شبکه می‌توانیم ماتریس آشفتگی مدل خود را بکشیم. ماتریس آشفتگی یک ماتریسی است که نشان می‌دهد چقدر از پیش‌بینی های هر کلاس درست و چقدر از آن در کلاس‌های دیگر بوده است. این ماتریس به ما کمک می‌کند تا بفهمیم مدل ما در کدام قسمت‌ها بیشتر اشتباه کرده است و به

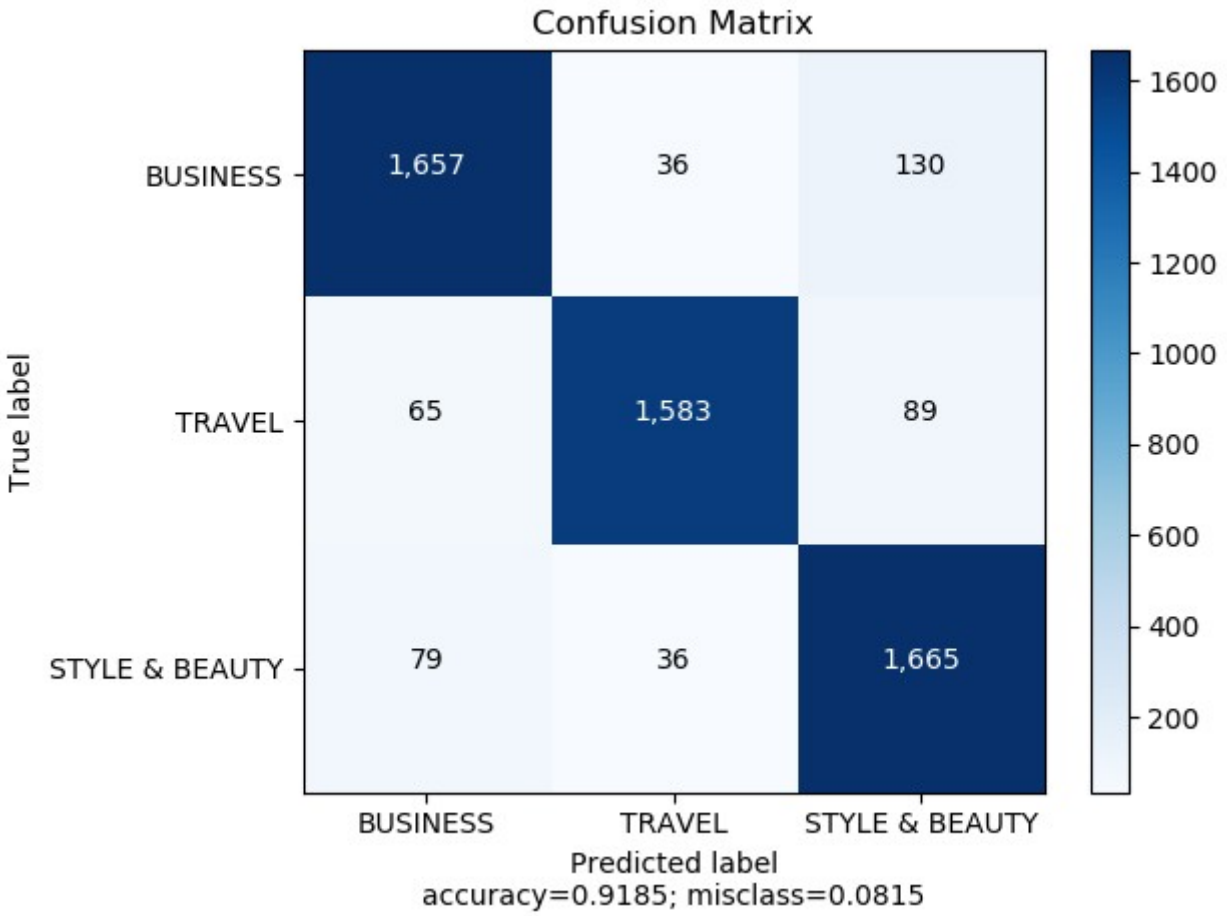
اشتباه کدام کلاس‌ها را جا به جا پیش‌بینی کرده است. در این ماتریس از رنگ هم استفاده می‌شود که در نقاط پررنگ تر به معنی اعداد بزرگ‌تر است سطر و ستون این ماتریس همان اسامی کلاس‌ها هستند که نشان می‌دهد هر کلاس به جای کدام کلاس دیگر گرفته شده است واضح است که قطر این ماتریس پررنگ ترین و با اعداد بزرگ‌تر باید باشد که به معنی درست پیش‌بینی کردن است

در این پروژه برای داده‌های ارزیابی می‌توانیم این ماتریس را بکشیم به این صورت که ابتدا برای همه ی داده‌های آن کلاس‌ها را پیش‌بینی می‌کنیم و در ماتریس با مقادیر واقعی آن مقایسه می‌کنیم داریم:

```
text_classifier.confusion(y_train, cleaned_x_train)
```

```
def confusion(self, train_target, train_data):
    train_prediction = self.predict(train_data)
    cm = confusion_matrix(y_true=train_target, y_pred=train_prediction)

    names = []
    for i in range(43):
        names.append(i)
    self.plot_confusion_matrix(cm=cm,
                               normalize=False,
                               target_names=self.classes,
                               title="Confusion Matrix")
```

و در این صورت ماتریس به شکل زیر می شود:



که میبینیم بیشترین مقادیر روی قطر است که بع معنی پیش‌بینی درست و برای مثال روی سطر اول ستون سوم می‌بینیم که به تعداد ۲۹۶ بار مدل به اشتباه برای کلاس ۱ پیش‌بینی کلاس ۳ کرده است که اشتباه است و همین‌طور برای بقیه اعداد همین برقرار است

در آخر می‌توانیم با پیش‌بینی خروجی تست مقادیر کلاس‌های آن را با مدل خود پیش‌بینی کنیم و این مقادیر را درون یک اکسل بریزیم:

```
result = text_classifier.predict(cleaned_x_test)
pd.DataFrame(result, columns=['category']).to_csv('output.csv', index=True, index_label='index')
```

فاز اول:

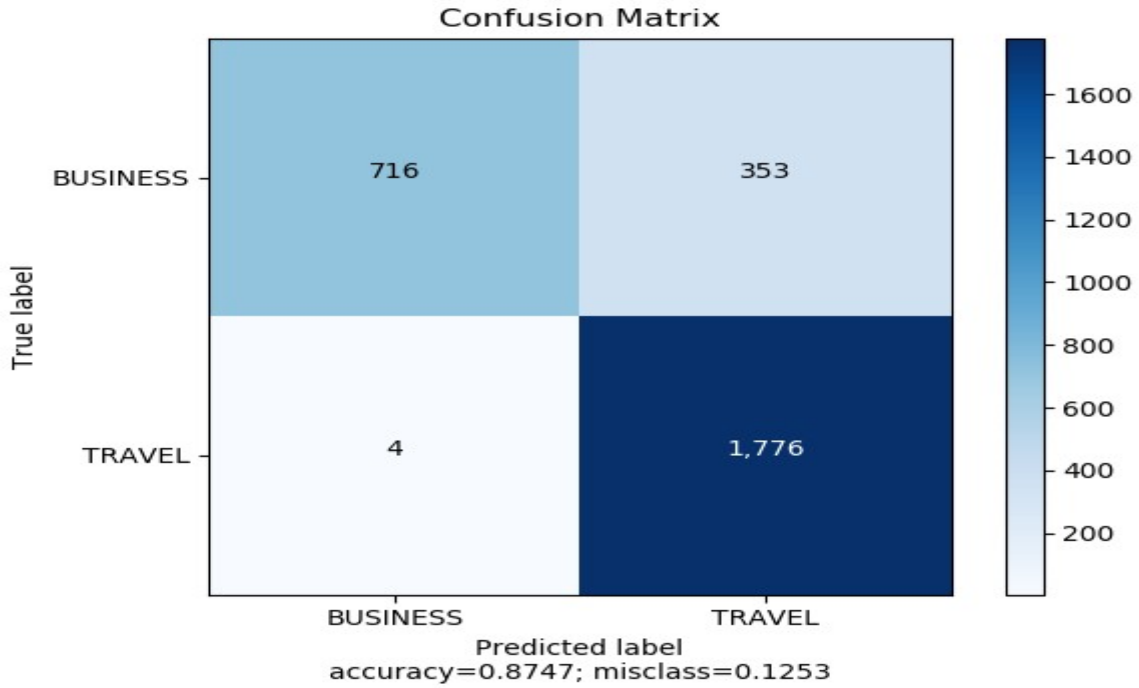
در این پروژه کلاس‌ها را در ۲ فاز جدا می‌کنیم در فاز اول فق ۲ کلاس از ۳ کلاس را جدا کرده آموزش و ارزیابی می‌کنیم این کار ها را همان‌طور که در بالا آورده شده است انجام می دهیم:
برای این فاز ۲ کلاس را در نظر می گیریم:

```
# self.classes = ['BUSINESS', 'TRAVEL', 'STYLE & BEAUTY']
self.classes = ['BUSINESS', 'TRAVEL']
```

و بعد از اجرای کد داریم:
خروجی ها به شکل زیر می شوند:

Phase 1	Travel	Business
Recall	0.99	0.69
Precision	0.84	0.99
Accuracy	0.88	_____

و ماتریس آشفتگی به صورت:



می‌بینیم که در ابتدا مقادیر Recall, Precision مخصوصاَ برای کلاس Business خیلی فرق دارند دلیل این تفاوت این است که داده‌های این کلاس نسبت به داده‌های کلاس دیگر خیلی کمتر هستند و مدل

نتوانسته به درستی این کلاس را آموزش ببیند و کلاس دیگر تأثیر بیشتری داشته است برای همین است که تعداد زیادی از داده‌ها برای کلاس Business به اشتباه تشخیص داده شده است که این اشتباه در ماتریس آشفتگی نیز دیده می‌شود و این مقادیر برای Travel خیلی بهتر است زیرا تعداد داده‌های آن زیاد بوده و شبکه بدرستی این کلاس را آموزش دیده و می‌تواند بیشتر داده‌ها را درست تشخیص دهد

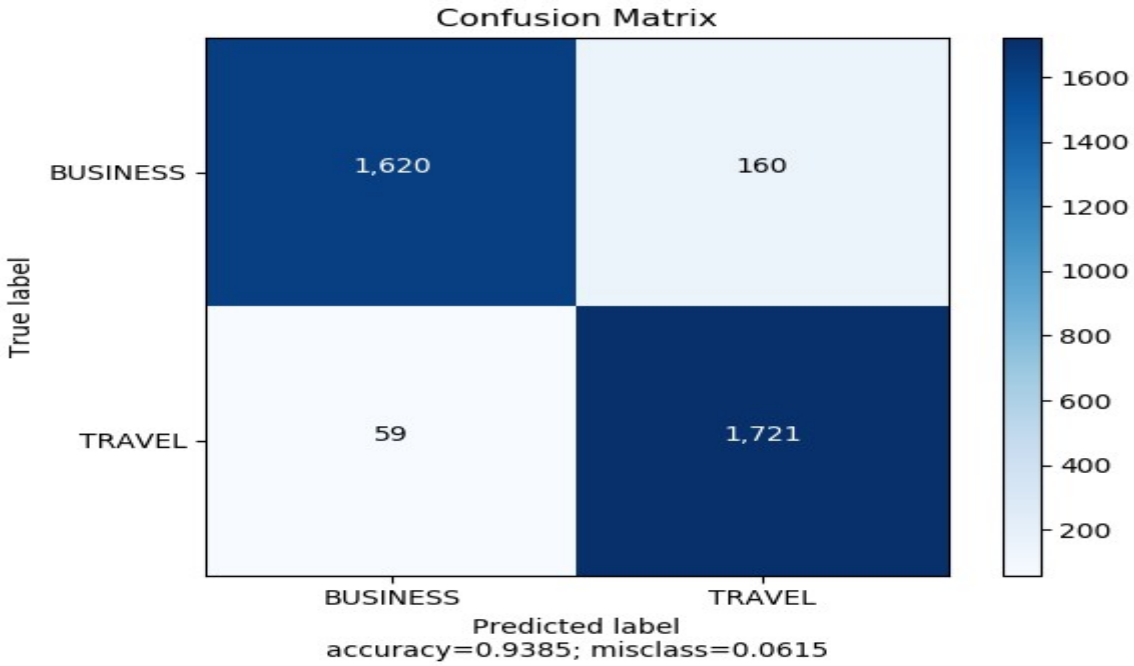
برای حل این مشکل نیاز به OverSampling داریم. Over sampling روش‌های مختلفی دارد و برای آن است که بتوانیم مشکل اختلاف در تعداد داده را حل کنیم. می توانییم داده‌ها را over sample کنیم تا تعداد آن‌ها بیشتر برابر شود و مدل بتواند به درستی همه ی کلاس‌ها را آموزش ببیند. این کار چند روش دارد که ساده‌ترین آن این است که از همان داده‌هایی که داریم انقدر تکرار کنیم تا تعداد کل داده‌های آن برابر با تعداد داده‌های کلاس‌های دیگر شود با این روش می توانییم تمام داده‌های کلاس‌ها را برابر کنیم تا مدل با دقت مناسب بتواند همه ی کلاس‌ها را از هم جدا کند
برای اینکار به تعداد اختلاف بیشترین کلاس با هر کلاس عضو رندوم از کلاس را برمی‌داریم و به همان کلاس اضافه می‌کنیم تا تعداد اعضای کلاس‌ها برابر شود

```
(class_train_data, class_train_target), (
    class_validation_data, class_validation_target) = self.over_sample_data(class_train_data,
                                                                            class_validation_data,
                                                                            class_train_target,
                                                                            class_validation_target)
```

```
def over_sample_data(self, class_train_data, class_validation_data, class_train_target, class_validation_tar
    for i in range(len(self.classes)):
        repeat_element = len(max(class_train_data, key=len)) - len(class_train_data[i])
        class_train_data[i].extend(random.sample(class_train_data[0], repeat_element))

        repeat_element = len(max(class_train_target, key=len)) - len(class_train_target[i])
        class_train_target[i].extend(random.sample(class_train_target[0], repeat_element))

        repeat_element = len(max(class_validation_data, key=len)) - len(class_validation_data[i])
        class_validation_data[i].extend(random.sample(class_validation_data[0], repeat_element))

        repeat_element = len(max(class_validation_target, key=len)) - len(class_validation_target[i])
        class_validation_target[i].extend(random.sample(class_validation_target[0], repeat_element))

    return (class_train_data, class_train_target), (class_validation_data, class_validation_target)
```

بعد از استفاده از over sampling برای فاز اول داریم:

Phase 1	Travel	Business
Recall	0.97	0.9
Precision	0.90	0.96
Accuracy	0.93	_____



همان‌طور که می‌بینیم اختلاف اعداد Recall, Precision کم شده و دقت مدل ما بالاتر رفته است
و در ماتریس آشفتگی نیز دیده می‌شود که مدل توانسته با دقت بیشتری کلاس‌ها را جدا کند و خطای کمتری داشته است

این مشکلات با برابر کردن اعضای کلاس‌ها بر طرف شده است

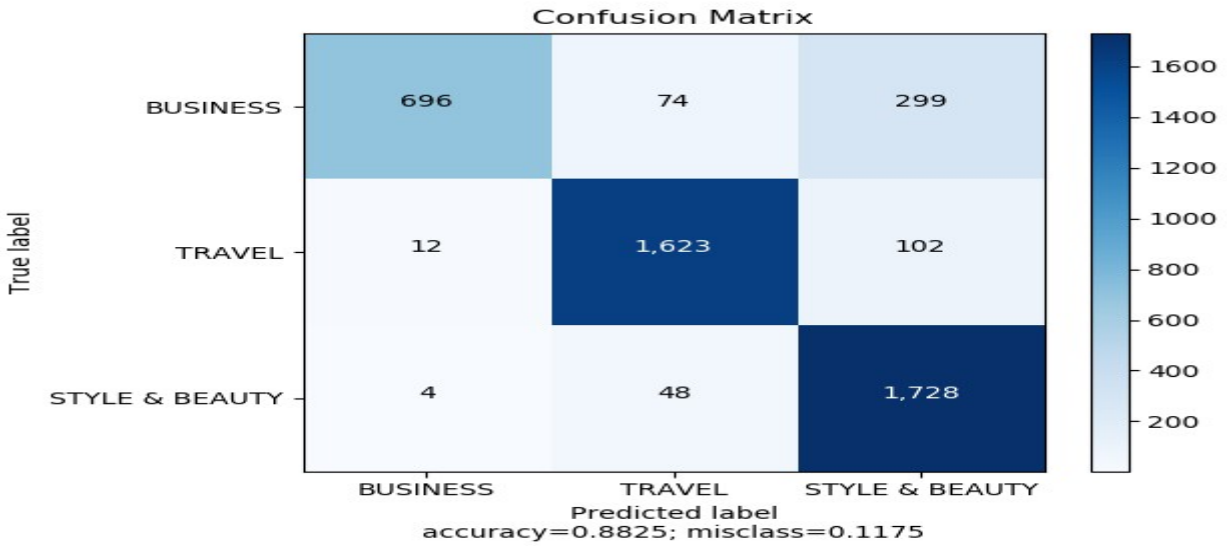
فاز دوم:

در این فاز کلاس ۳ را هم به دو کلاس قبلی اضافه می‌کنیم و داریم:

```
self.classes = ['BUSINESS', 'TRAVEL', 'STYLE & BEAUTY']
```

بعد از اجرای کد داریم:

Phase 2	Travel	Business	Style & Beauty
Recall	0.97	0.65	0.91
Precision	0.79	0.98	0.94
Accuracy	_____	0.87	_____

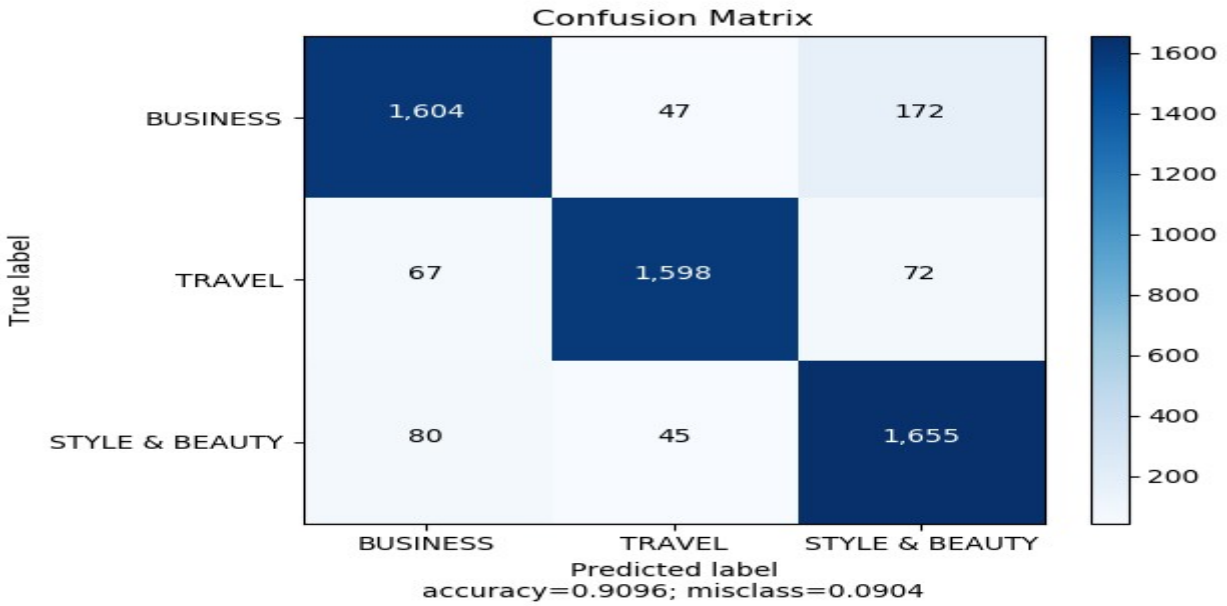


همان‌طور که میبینیم اختلاف بین اعداد زیاد است و شبکه برای کلاس Business اشتباه زیاد داشته است و لی دو کلاس دیگر کمتر است

این همان دلیل فاز قبل است که تعداد اعضا برابر نیست و کلاس Business اعضای کمتری دارد به همین دلیل مدل نتوانسته به خوبی برای این کلاس آموزش ببیند درحالی که تأثیر کلاس‌های دیگر بیشتر است

برای درست کردن آن مانند فاز قبل over sampling انجام می‌دهیم تا تعداد داده‌ها در کلاس‌ها برابر شود
بعد از اجرای کد داریم:

Phase 2	Travel	Business	Style & Beauty
Recall	0.95	0.89	0.90
Precision	0.88	0.92	0.94
Accuracy	_____	0.91	_____



همان‌طور که دیده می‌شود مدل به دقت بالا تری رسیده است و برای کلاس‌های مختلف توانسته به خوبی آموزش ببیند و اختلاف اعداد آن کمتر شده است

این اختلاف را با over sampling انجام داده‌ایم تا تعداد داده‌ها در کلاس‌های مختلف برابر شود و این کار را با استفاده از اضافه کردن تعدادی داده از همان کلاس که به صورت رندوم برداشته شده است انجام داده‌ایم

سوالات:

۱)

برای اینکه آن‌ها را مقایسه کنیم همان‌طور که در اول هم گفته شد چون lemmatization می‌تواند کلمات را یکپارچه تر کند چون ریشه اصلی کلمه در حالت ساده را پیدا می‌کند جواب بهتری می‌دهد

برای اینکه امتحان کنیم پروژه را در فاز ۳ در دوحالت یک بار با stem و یک بار با lemmatization امتحان می‌کنیم و داریم:

با Lemmatization:

```
def clean_text(self):
    self.__text_lower_case()
    self.__remove_numbers()
    self.__remove_punctuation()
    self.__remove_stop_words()
    # self.__do_stemming()
    self.__do_lemmatization()
    return self.data_frame
```

```
Recall is:  {'BUSINESS': 0.8963247394404827, 'TRAVEL': 0.9544943820224719, 'STYLE & BEAUTY': 0.9159470351180196}
Precision is:  {'BUSINESS': 0.9396204715353651, 'TRAVEL': 0.8771295818275684, 'STYLE & BEAUTY': 0.9561298076923077}
Accuracy is:  0.9220973782771535
```

و با Stem:

```
def clean_text(self):
    self.__text_lower_case()
    self.__remove_numbers()
    self.__remove_punctuation()
    self.__remove_stop_words()
    self.__do_stemming()
    # self.__do_lemmatization()
    return self.data_frame
```

```
Recall is:  {'BUSINESS': 0.8826110806363138, 'TRAVEL': 0.9308988764044944, 'STYLE & BEAUTY': 0.9119170984455959}
Precision is:  {'BUSINESS': 0.9194285714285715, 'TRAVEL': 0.8684486373165619, 'STYLE & BEAUTY': 0.9417360285374554}
Accuracy is:  0.9082397003745318
```

همان‌طور که میبینیم دقت در حالت Lemmatization بیشتر است

۲)

tf_idf یک شاخص برای پردازش متن است که می‌تواند اهمیت هر کلمه را در جمله تعیین کند. در الگوریتم Bag Of Word برای هر کلمه تکرار در نظر گرفته می‌شود یعنی مثلاً اگر ۲ کلمه example و exaple را داشته باشیم در BoW با هر دو یکسان بر خورد می‌کند چرا که کلمه های متفاوتی هستند و تکرار یکسانی دارند ولی در استفاده از این شاخص اهمیت یک کلمه با کلمه دیگر فرق دارد و میزان استفاده از آن کلمه در نظر گرفته می‌شود پس در این شاخص چون کلمه example بیشتر از exaple استفاده شده است تأثیر کلمه exaple کم می‌شود در این حالت ما ۲ شاخص tf و idf را باید محاسبه کنیم که شاخص tf را برای همه ی متن های ورودی و شاخص idf z را برای همه ی کلمات به شکل زیر می توانییم حساب کنیم:

> *IDF =Log[(# Number of documents) / (Number of documents containing the word)] and*
>
> *TF = (Number of repetitions of word in a document) / (# of words in a document)*

که نشان می‌دهد هر کلمه در جمله چقدر تأثیر دارد چرا که مثلاً در BoW کلمه the به تعداد زیاد تکرار شده است ولی معنی جمله به خاطر آن نیست ولی در این شاخص این کلمه تأثیر کمتری دارد چرا که ضرب دو شاخص tf, idf آن کم می‌شود
برای این پروژه می توانستیم با این شاخص ابتدا بعد از تمیز کردن داده برای تمام لغات ای که در لغت‌نامه داریم tf را برای همه متن ها و سپس idf لغات را حساب کنیم و با این شاخص تصمیم بگیریم کدام کلمات مهم‌تر هستند و تأثیر آن کلمات را مثلاً در تکرار آن‌ها بیشتر کنیم تا مدل بتواند بهتر کلاس‌ها را جدا کند چرا که برخی کلمات مهم‌تر در هر کلاس بدست می‌آید و تأثیر آن‌ها بیشتر می‌شود و در این صورت دقت مدل ما بیشتر می‌شود

۳)

Precision : تعداد داده هایی که به درستی توسط مدل در یک دسته تشخیص داده شده اند نسبت به کل داده هایی که مدل ما به درستی یا اشتباه در یک دسته منظور کرده است.
بنابراین در این معیار تنها به دانسته های مدل اکتفا شد است و به دانسته های خود ما از محیط وابسته نمی باشد.
لذا ممکن است در یک محیط با وجود داشتن precision زیاد recall مقدار کمی داشته باشد و در واقع این نشان میدهد که نمیتوان تنها با اکتفا به precision که اطلاعات محیط واقعی را به طور مطلوب در محاسبات وارد نکرده است، اطلاعات دقیقی از مناسب بودن مدل ارایه کرد.

به طور مثال: در مدل تشخیص بیماری کرونا ممکن است precision مقدار بالایی داشته باشد ولی recall مقدار کمی، در نتیجه مدل ما اگر کسی را مبتلا تشخیص دهد به احتمال بالا درست تشخیص داده است اما در کل احتمال آن که یک نفر کرونا داشته باشد و مدل آن را تشخیص دهد پایین است(recall مقدار کمی دارد.)

۴)

مدل ما برای پیدا کردن طبقه بندی یک متن احتمال وجود تک تک کلمات را به شرط آن کلاس حساب کی ند و در هم ضرب می‌کند جال اگز یک کلمه فقط در یک کلاس آمده باشد نتیجه می‌شود احتمال آن به شرط بقیه ی کلاس‌ها صفر است پس ضرب آن در بقیه احتمال ها نیز صفر می‌شود پس نتیجه می‌دهد مدل هر وقت برای داده‌های غیر آموزش آن کلمه را ببیند حتماً کلاس را همان کلاس کلمه می‌گیرد چرا که احتمال آن به شرط کلاس‌های دیگر صفر است
برای حل این مشکل از این احتمال لگاریتم گرفتیم که ضرب را به جمع تبدیل کرد که در این صورت ادن ۰ را برای کلاس‌های دیگر احتمال کل را صفر نمی‌کرد ولی وقتی این کلمه در کلاس‌های دیگر نباشد صورت لگاریتم صفر می‌شود که برای این کار از

قاعده لاپلاس استفاده کرده‌ایم که صورت را با ۱ و مخرج را با تعداد کل لغات جمع می‌کنیم و این اثر صفر از بین می رود