

مقدمه:

در این پروژه می‌خواهیم با استفاده از قانون شبکه‌های بیزینیم مدل‌ای را آموزش دهیم که با دیدن یک متن از اخبار تشخیص دهد که آن خبر در چه دسته‌ای طبقه بندی می‌شود. این کار را با استفاده از دیتاست آموزش انجام می‌دهیم و بعد از ارزیابی آن با همین مدل آموزش دیده یک دیتاست تست را نیز پیش‌بینی می‌کنیم.

پیش پردازش داده‌ها:

قبل از شروع برای آموزش دادن مدل با دیتاست نیاز است تا داده‌های خود را تمییز کنیم تا مدل ما بتواند با سرعت و دقت خوبی به پردازش با آن داده‌ها بپردازد.

برای این پروژه داده‌های ما که درون اکسل‌ای قرار دارد را می‌خواهیم پردازش کنیم این داده‌ها شامل ستون‌های مختلف‌ای 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 کلمات‌ای 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:

```
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 داده‌ی 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 تمیز کنیم برای این کار به ترتیب به صورت زیر عمل می‌کنیم:

(۱) تبدیل همه ی حروف بزرگ به حروف کوچک:
ابتدا درون متن همه ی حروف را به حروف کوچک تبدیل می‌کنیم. 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 می‌ند 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 می‌توانیم 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 علائم نگارشی:

مدل ما به دنبال کلمات درون اخبار ها می‌گردد و سعی دارد با کلمات طبقه را تشخیص دهد پس علائم 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 خبرها 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) Lemmatization:

این روش همانند روش stemming سعد دارد تا به 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 Lemmatization بیشتر است چرا که در این روش همه کلمات به حالت ساده خود تبدیل می‌شوند و حتی اگر زمان‌های آن‌ها یا مشتقات آن‌ها نیز در متن آمده باشد باز هم یکسان می‌شوند و متن یکپارچه‌تر 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 متن‌ای 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:

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

Likelihood
Class Prior Probability

Posterior Probability
Predictor Prior Probability

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 Likelihood 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 آن 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 می‌شود

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 داده‌ی ارزیابی آن را ارزیابی کرده و میزان خصوصیات شبکه از جمله دقت آن را بررسی کنیم تا بتوانیم به شبکه بهتری دست پیدا کنیم برای این کار ابتدا داده‌ها را از جدول‌ها می‌خوانیم:

```
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 می‌آوریم:

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

برای جدا کردن داده‌های ارزیابی باید ابتدا دسته‌ها 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 همه‌ی 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 همه‌ی 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 می‌کنیم:

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

این معیار نشان می‌دهد که برای هر طبقه تعدادی از آن طبقه که درست پیش‌بینی شده است نسبت 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 طبقه‌ها حساب می‌کنیم

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

معیار بعدی Precision نام دارد که نشان می‌دهد برای هر طبقه چقدر از آن در طبقه‌های دیگر پیش‌بینی شده است. این معیار حاصل تقسیم تعداد ای که مدل به درستی از آن طبقه پیش‌بینی کرده است بر تعداد پیش‌بینی‌ها از آن طبقه است. این معیار نشان می‌دهد چقدر از هر طبقه به اشتباه در طبقه‌های 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

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

معیار آخر دقت کلی شبکه است. این معیار نشان می‌دهد چقدر از داده‌های شبکه به درستی طبقه‌بندی 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 تمیز 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 می‌کنیم:

```
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_texas[class_type] = sum(1 for label in target if label == class_type)
        self.log_class_priors[class_type] = math.log(self.num_texas[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 ربط 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:

$$P(c | X) = P(x_1 | c) \times P(x_2 | c) \times \dots \times P(x_n | c) \times P(c)$$

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 آوریم 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 خروجی 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 آوریم و مدل را ارزیابی کنیم

```
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 از فرمول های معیار ها که در بالا توضیح داده شد این معیار ها را بدست می آوریم و از وضعیت شبکه مطلع می شویم

برای پیش بینی کلاس داده ها که هم برای داده های تست و هم برای داده های ارزیابی استفاده کرده ایم 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_texas[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 می آوریم سپس به ازای این کلمات ابتدا اگر در لغت نامه وجود داشت برای هر کلاس تعداد بار تکرار شده آن کلمه را بر تعداد کل تکرار آن متن تقسیم می کنیم و با احتمال آن کلاس جمع می کنیم این همان قاعده بیز است که از آن لگاریتم گرفته ایم . برای اینکه مطمئن شویم که صورت لگاریتم ما ۰ نمی شود از قاعده لاپلاس استفاده کرده ایم به این صورت که صورت لگاریتم را با ۱ جمع کرده ایم و مخرج را با تعداد کل لغات جمع می کنیم بعد از پیدا کردن احتمال برای هر کلاس آن احتمال ای که بزرگ تر باشد نتیجه می دهد که متن داده شده برای آن کلاس پیش بینی می شود

بعد از پیدا کردن معیار های شبکه می توانیم ماتریس آشفتگی مدل خود را بکشیم. ماتریس آشفتگی یک ماتریسی است که نشان می دهد چقدر از پیش بینی های هر کلاس درست و چقدر از آن در کلاس های دیگر بوده است. این ماتریس به ما کمک می کند تا بفهمیم مدل ما در کدام قسمت ها بیشتر اشتباه کرده است و به

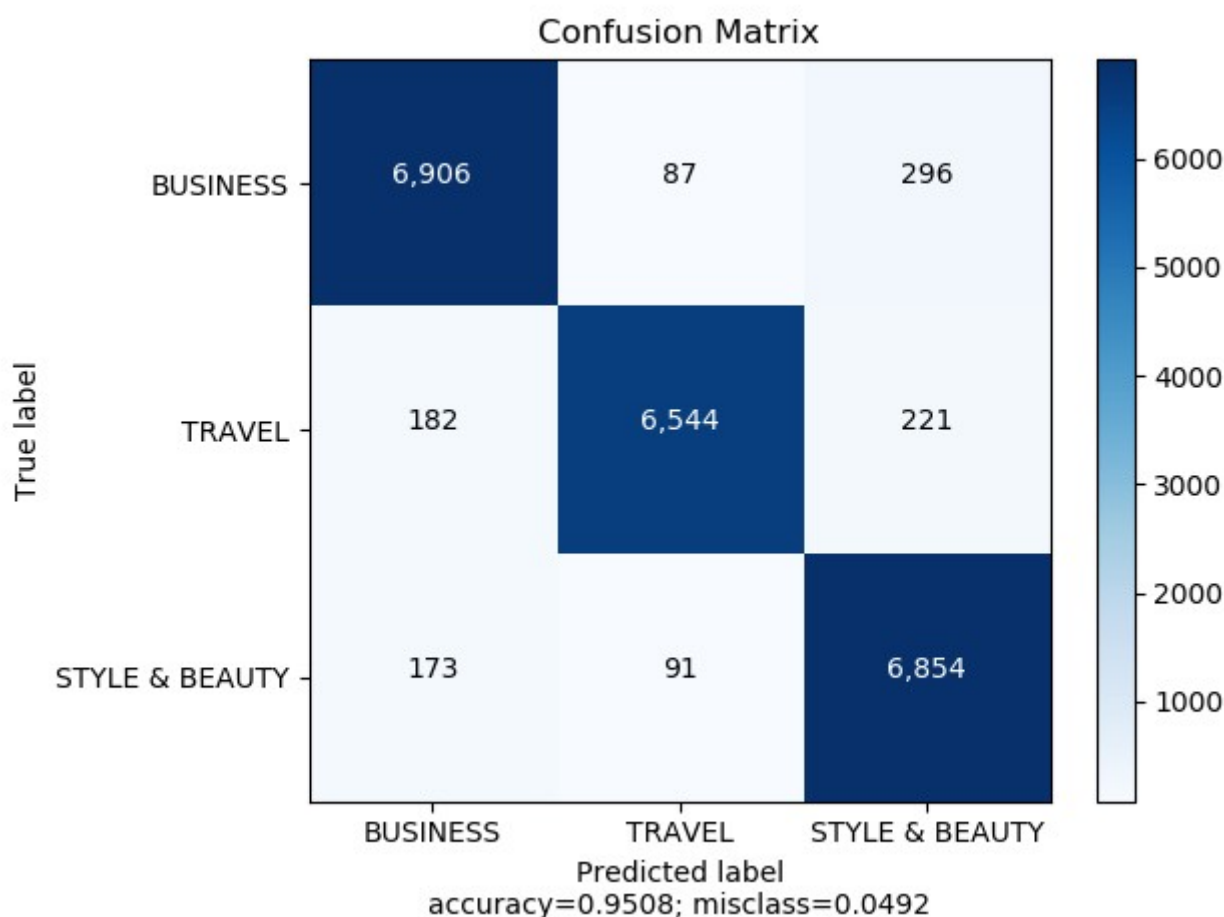
اشتباه کدام کلاس‌ها را جا به جا پیش‌بینی کرده است. در این ماتریس از رنگ هم استفاده می‌شود که در نقاط پررنگ تر به معنی اعداد بزرگ‌تر است سطر و ستون این ماتریس همان اسامی کلاس‌ها هستند که نشان می‌دهد هر کلاس به جای کدام کلاس دیگر گرفته شده است واضح است که قطر این ماتریس پررنگ ترین و با اعداد بزرگ‌تر باید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

در این پروژه برای داده‌های آموزش 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 پیش بینی درست و برای مثال روی سطر اول ستون سوم می بینیم که به تعداد ۲۹۶ بار مدل به اشتباه برای کلاس ۱ پیش بینی کلاس ۳ کرده است که اشتباه است و همین طور برای بقیه اعداد همین برقرار است

در آخر می توانیم با پیش بینی خروجی تست مقادیر کلاس های آن را با مدل خود پیش بینی 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:

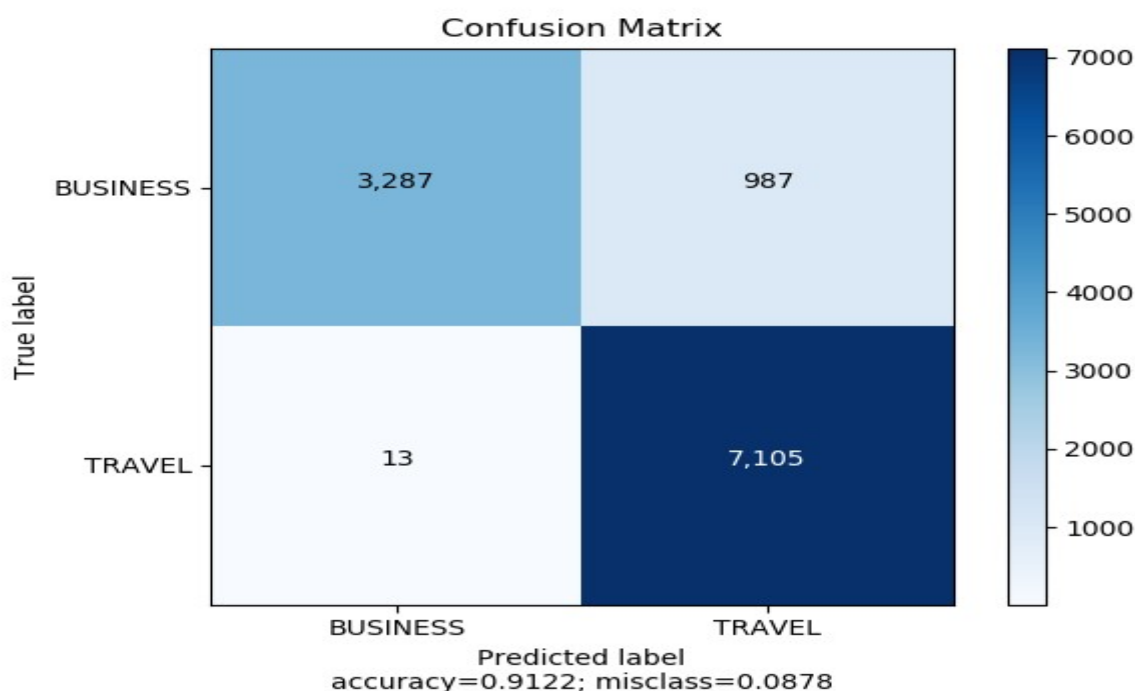
در این پروژه کلاس‌ها را در ۲ فاز جدا می‌کنیم در فاز اول فقط ۲ کلاس از ۳ کلاس را جدا کرده آموزش و ارزیابی می‌کنیم این کار ها را همان‌طور که در بالا آورده شده است انجام می‌دهیم: برای این فاز ۲ کلاس را در نظر می‌گیریم:

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

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

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 مخصوصاً 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. می‌توانیم 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 می‌توانیم 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_target):
    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))

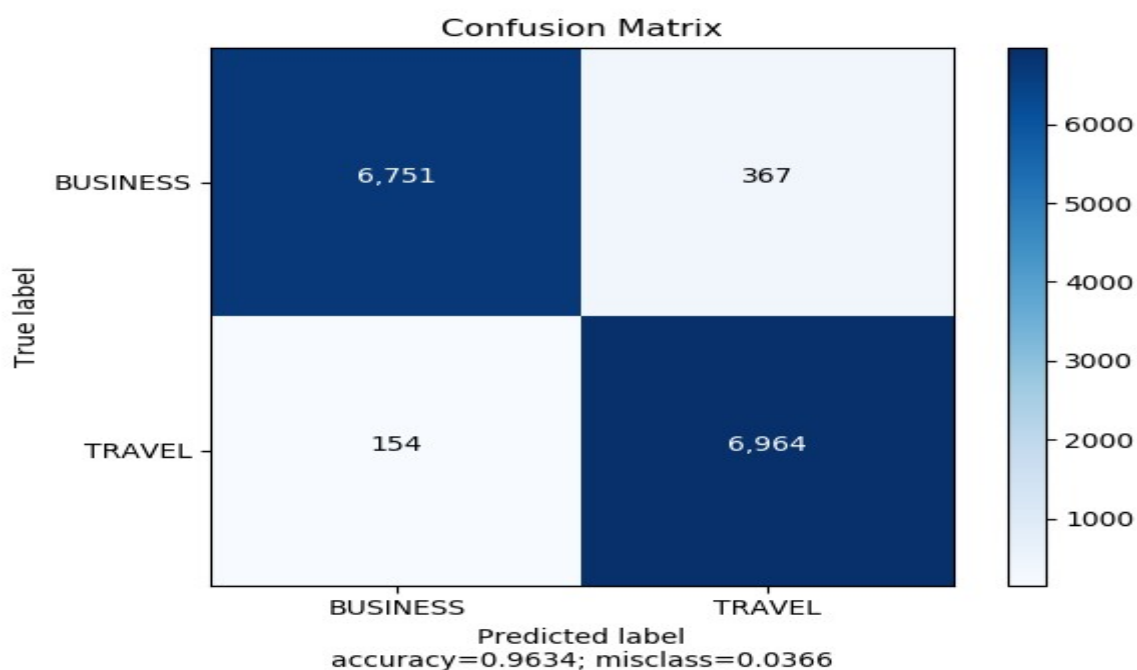
        repeat_element = len(max(class_validation_data, key=len)) - len(class_validation_data[i])
        class_validation_data[i].extend(random.sample(class_validation_data[0], repeat_element))

        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	



همان طور که می بینیم اختلاف اعداد Recall, Precision کم شده و دقت مدل ما بالاتر رفته است و در ماتریس آشفتگی نیز دیده می شود که مدل توانسته با دقت بیشتری کلاس ها را جدا کند و خطای کمتری داشته است این مشکلات با برابر کردن اعضای کلاس ها بر طرف شده است

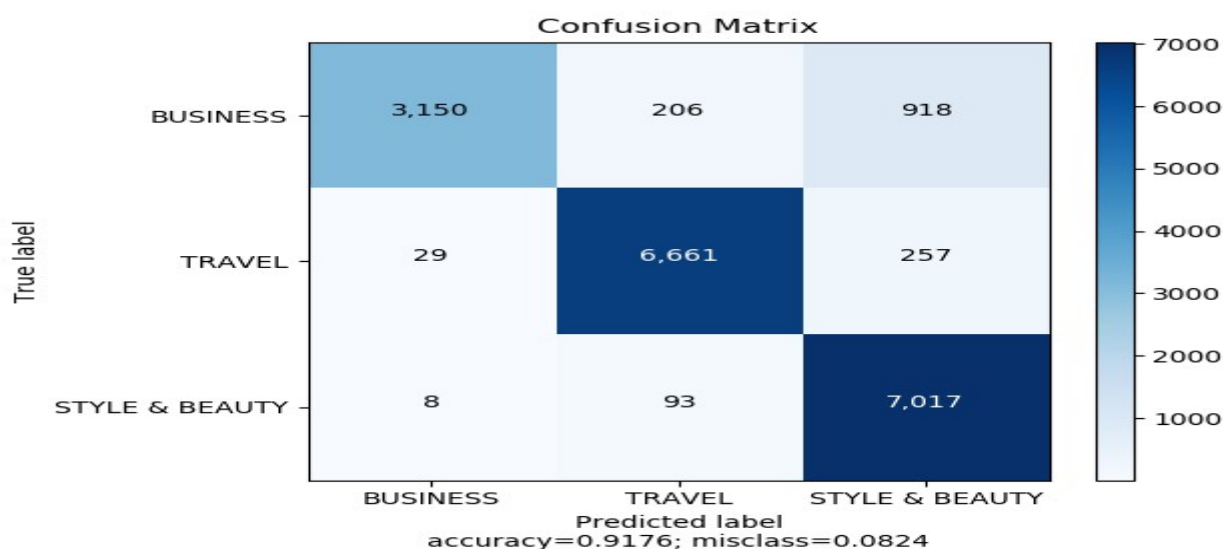
فاز دوم:

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	_____

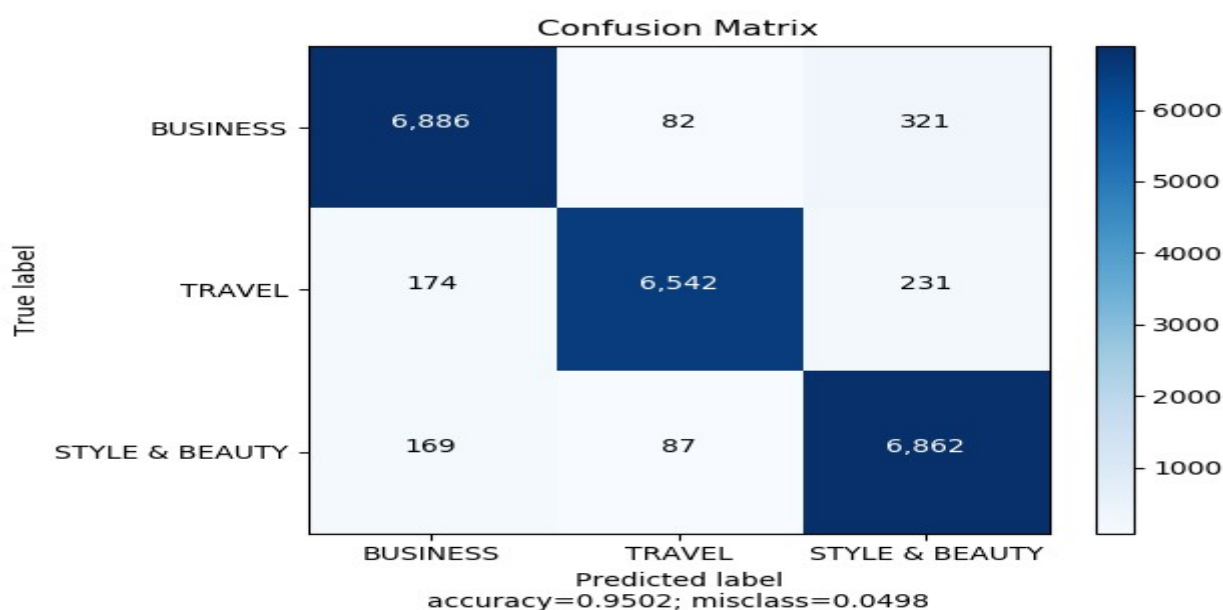


همان‌طور که می‌بینیم اختلاف بین اعداد زیاد است و شبکه برای کلاس Business اشتباه زیاد داشته است و لی دو کلاس دیگر کمتر است

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 داده ایم