دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر

گزارش تمرین شماره ۶

درس NLP

بهار ۱۴۰۲

نام و نام خانوادگی
سپهر کریمی آرپناهی

شماره دانشجویی
۸۱۰۱۰۰۴۴۷

پرسش ۱: ربات پاسخگو به پرسش‌های پر تکرار

در این سوال می خواهیم یک ربات پاسخگو به پرسش‌های پر تکرار همراه اول را با Rasa پیاده سازی کنیم. در این تمرین می خواهیم دو رویکرد را پیاده سازی کنیم. در رویکرد اول هر سطر از فایل اکسل داده شده یک intent برای ما تشکیل می دهد. اما در رویکرد دوم، یک کلاس (intent) واحد برای کل پرسش و پاسخ های پر تکرار داریم و هر سطر از فایل اکسل، یک sub-intent یا زیر کلاس ما را تشکیل می دهد.

رویکرد اول : هر جفت پرسش و پاسخ، یک کلاس

در ابتدا نرم‌افزار را نصب کرده و آن را init می کنیم. حال با توجه به آن‌که در رویکرد اول هر سطر از فایل اکسل یک intent ما را تشکیل می‌دهد، ۵۰ intent خواهیم داشت. حال به آماده‌سازی داده ها و انجام تنظیمات Rasa می پردازیم.

- **آماده‌سازی داده‌ها به صورت خودکار**

پرونده nlu.yml: در این قسمت کدی را نوشتیم که در ابتدا دادهای پرونده MCI_Internet_TrainData را خوانده و در یک دیتافریم ذخیره می‌کند. سپس فایل nlu.yml را با دسترسی write باز می‌کند. در ادامه در یک حلقه بر روی سطرهای دیتافریم حرکت کرده و هر هر مرحله با فرمت مناسب ابتدا نام intent را می‌نویسد و سپس example های آن را از سطر ۸ تا ۲۲ دیتافریم در فایل nlu.yml قرار می دهد. فرمت نوشتن در فایل به صورت زیر می باشد.

```
nlu:
- intent: faq_1
  examples: |
    - برای فعال سازی سرویس gprs با استفاده از کد دستوری چیکار کنم؟
    - برای فعالسازی سرویس gprs چه شماره ای باید بگیرم؟
    - برای فعالسازی سرویس gprs چه کدی رو باید بزنم؟
    - چجوری سرویس اینترنت gprs با استفاده از کد دستوری رو فعال کنم؟
```

پرونده domain.yml: در این قسمت کدی را نوشتیم که در ابتدا دادهای پرونده MCI_Internet_TrainData را خوانده و در یک دیتافریم ذخیره می‌کند. سپس فایل domain.yml را با دسترسی write باز می‌کند. در ادامه ابتدا نام کل intent ها را در فایل می‌نویسد. سپس در یک حلقه بر روی سطرهای دیتافریم حرکت کرده و در هر مرحله با فرمت مناسب سپس تمامی response ها را در فایل domain با فرمت مناسب می نویسد. فرمول نوشتن در هر سطر به صورت زیر می‌باشد:

```
responses:
  utter_faq_1:
  - text: "کد دستوری ۱۰۰ ستاره ۱۱۱ مربع را شماره گیری نمایید (GPRS) مشترک گرامی جهت فعالسازی سرویس اینترنت"
```

پرونده rules.yml: در این قسمت کدی را نوشتیم که به ازای کل intent ها قوانین پاسخ به آن را تعریف کند و پیاده سازی کد به این صورت است که، فایل rules.yml را با دسترسی write باز می‌کند. در ادامه در یک حلقه به ازای تعداد intent ها، در هر مرحله با فرمت مناسب ابتدا نام intent را می‌نویسد و سپس پاسخ آن را قرار می دهد که فرمت این فایل به صورت زیر می باشد:

```
- rule: faq 1
  steps:
  - intent: faq_1
  - action: utter_faq_1
```

پرونده nlu_test.yml: در این قسمت کدی را نوشتیم که در ابتدا دادهای پرونده MCI_Internet_TestData را خوانده و در یک دیتافریم ذخیره می‌کند. سپس فایل test_nlu.yml را با دسترسی write باز می‌کند. در ادامه در یک حلقه بر روی سطرهای دیتافریم حرکت کرده و هر هر مرحله با فرمت مناسب ابتدا نام intent را می‌نویسد و سپس example های آن را از سطر ۸ تا ۱۲ دیتافریم در فایل test_nlu.yml قرار می دهد. همچنین فرمت نوشتن در این فایل به صورت زیر می باشد:

```
- intent: faq_1
  examples: |
    - سرویس اینترنتی با استفاده از کد دستوری چجوری فعال میشه؟
    - چجوری سرویس اینترنت با استفاده از کد دستوری فعال میشه؟
    - سرویس اینترنتم نیاز به فعالسازی داره؟ کد دستوری داره؟
    - چجوری سرویس اینترنتم رو با استفاده از کد دستوری فعال کنم؟
    - فعال شدن سرویس اینترنت چجوریه؟
```

همانطور که در صورت سوال گفته شده در این تمرین قرار است عملکرد مدل های زبانی بر پایه Bert را بررسی کنیم. به همین دلیل ابتدا مدل‌های ParsBert و LaBSE را هرکدام با ۱۰۰ دور آموزش می آزماییم و برای مدل بهتر بر روی ۵۰ و ۲۰۰ دور نیز آموزش داده و آن را گزارش می کنیم.

- **مدل اول LaBSE (۱۰۰ دور آموزش):**

- انجام تنظیمات و آموزش ربات

در این قسمت می خواهیم تنظیمات ربات در فایل config.yml را شرح داده و سپس ربات را با آن اموزش دهیم. برای این مدل ابتدا فایل config این مدل را گذاشته و هر قسمت آن را توضیح می دهیم:

config.yml(LaBSE - 100 epoch)

```yaml
language: fa
pipeline:
- name: WhitespaceTokenizer
- name: RegexFeaturizer
- name: LexicalSyntacticFeaturizer
- name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: char_wb
  min_ngram: 1
  max_ngram: 4
- name: LanguageModelFeaturizer
  model_name: bert
  model_weights: rasa/LaBSE
  constrain_similarities: true
- name: DIETClassifier
  epochs: 100
  constrain_similarities: true


policies:
```

کد بالا همانطور که گفته شد، تنظیمات config نرم افزار rasa می باشد. ابتدا در این فایل زبان مورد استفاده ربات را مشخص می کنیم . آن را متناسب با دیتا،برابر با fa قرار می‌دهیم. سپس فایل کانفیگ از دو قسمت کلی تشکیل می شود. Pipeline و policies. قسمت policies را خالی گذاشتیم تا policy های پیش‌فرض را در نظر بگیرد. حال به شرح pipeline می‌پردازیم.

- **WhitespaceTokenizer**: یک توکنایزر معروف است که همانطور که از نام آن پیدا است، ورودی بر اساس whiteSpace توکنایز کرده یعنی اگر نقطه و یا "،" به کلمه ای چسبیده باشد نمی تواند آن را جدا کند.

- **RegexFeaturizer**: ورودی توکنایز شده را به عنوان ورودی گرفته و با استفاده از عبارت‌های منظم، از توکن ها ویژگی استخراج می‌کند. از این featurizer برای استخراج pattern و یا entity از متن استفاده می‌شود.

- **LexicalSyntacticFeaturizer**: این featurizer برای استخراج ویژگی‌های نحوی و معنایی از متن ورودی به کار می‌رود و به مدل کمک می‌کند تا شاختار گرامری و ساختار جمله‌ها را درک کند.

- **CountVectorsFeaturizer**: این featurizer، متن را به عنوان ورودی گرفته و برای هر کلمه بر اساس فرکانس و تکرار آن، وکتورهای عددی می‌سازد و یک representation برای آن تولید می کند.

- **CountVectorsFeaturizer (analyzer: char_wb)**: این featurizer، متن را به عنوان ورودی گرفته ولی متن را در سطر کاراکتر بررسی می‌کند و این کار را با کمک chracter ngram در اینجا n=(1:4) انجام می دهد. این کار به مدل کمک می‌کند تا پترن های در سطح کاراکتر را استخراج کند(معمولا برای زبان های با مورفولوژی بالا پرکاربرد تر است).

- **LanguageModelFeaturizer**: این featurizer از یک مدل زبانی از پیش آموزش دیده(BERT) استفاده کرده و embedding های مبتی بر متن از آن ها تولید می‌کند.

  در این قسمت ما از model_weight های آموزش دیده شده LaBSE استفاده خواهیم کرد که مخفف Language-agnostic BERT Sentence Embedding می‌باشد و یک مدل زبانی چند-زبانه و از پیش آموزش دیده می‌باشد که می‌تواند امبدینگ جمله های با دقت بالایی تولید کند.

- **DIETClassifier**: یک کلاسیفایر می باشد که برای کلاسیفای کردن dialogue intent و استخراج entity ها به کار می رود. در این مرحله این کلاسیفایر را برای ۱۰۰ epoch و با در نظر گرفتن شباهت های میان فیچر ها(constrain_similarities) برای محدود کردن فرآیند آموزش و نگه داشتن similarity در یک بازه محدود استفاده می‌شود.

- ارزیابی ربات

به کمک دستور زیر ربات آموزش داده شده را ارزیابی می‌کنیم و فایل تست test_nlu.yml که در قسمت قبل ساختیم را به عنوان ورودی به آن می‌دهیم.

```
rasa test nlu --nlu test_nlu.yml
```

پس از اجرای کد تست، نتایج در پوشه results ذخیره می‌شوند. که نتایج بدست آمده به شرح زیر می‌باشد:

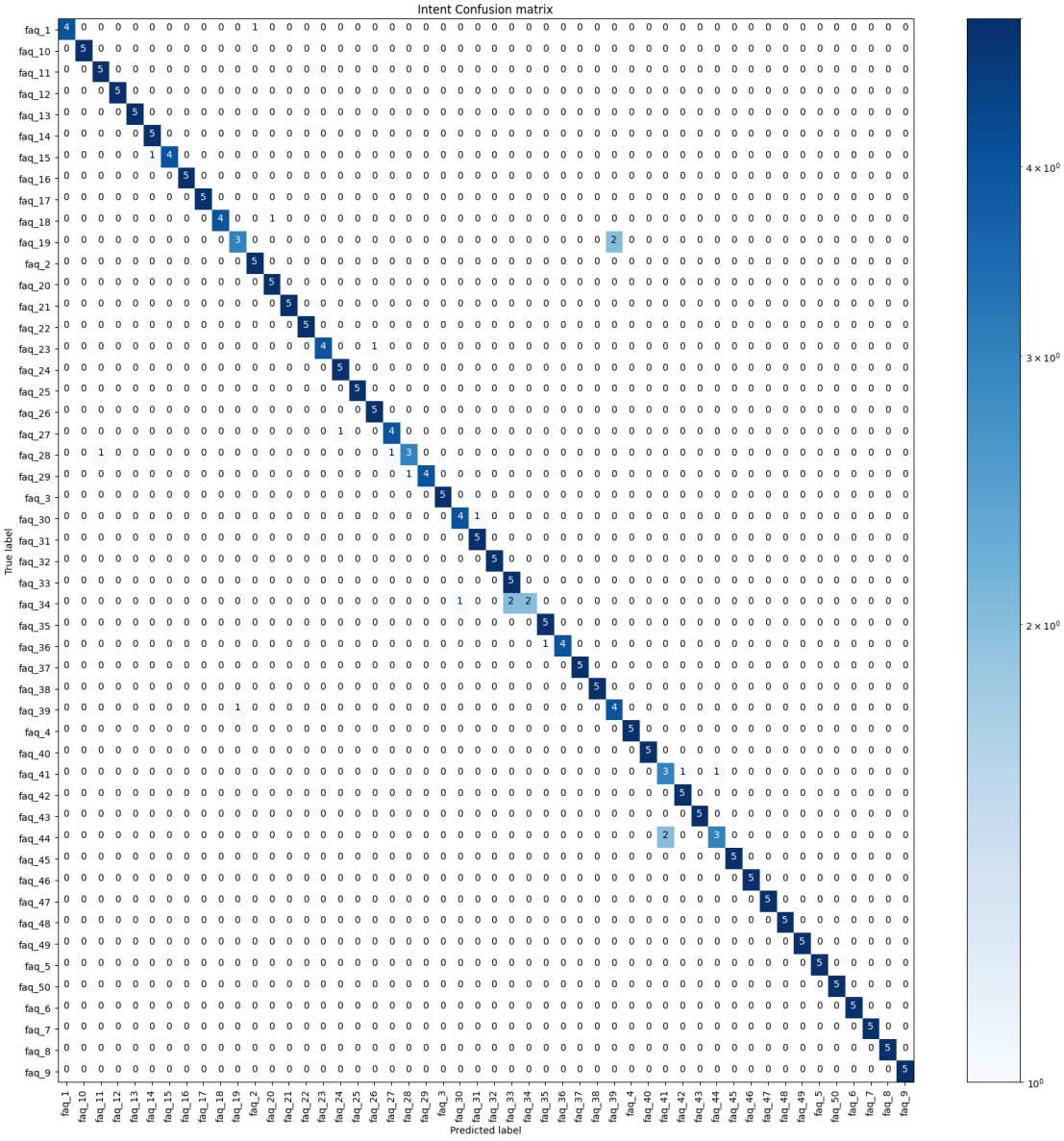
مقادیر نتایج برای هر اندیشه در فایل intent_report_labse_100.json گزارش شده است. در اینجا مقادیر دقت و f1-score را برای کل کلاس ها گزارش می‌کنیم.

Accuracy: 0.92

Macro_avg F1-score: 0.9169

Weighted avg F1-score: 0.9169

Micro_avg F1-score: 0.92



شکل ۱ : ماتریس آشفتگی برای مدل LaBSE با ۱۰۰ دور آموزش و رویکرد اول

جدول ۱ : precisionو f۱-score برای مدل LaBSE با ۱۰۰ دور آموزش و رویکرد اول

FAQ	Precision	F1-Score
faq_1	1.0	0.888
faq_26	0.8333	0.9091
faq_30	0.8	0.8000
faq_38	1.0	1.0
faq_11	0.8333	0.9091
faq_44	0.75	0.6667
faq_21	1.0	1.0
faq_41	0.6	0.6
faq_33	0.7143	0.8333
faq_12	1.0	1.0
faq_6	1.0	1.0
faq_35	0.8333	0.9091
faq_24	0.8333	0.9091
faq_40	1.0	1.0
faq_18	1.0	0.8889
faq_20	0.8333	0.9091
faq_14	0.8333	0.9091
faq_17	1.0	1.0
faq_23	1.0	0.8889
faq_36	1.0	0.8889
faq_39	0.6667	0.7273
faq_49	1.0	1.0
faq_5	1.0	1.0
faq_45	1.0	1.0
faq_32	1.0	1.0

FAQ	Precision	F1-Score
faq_8	1.0	1.0
faq_48	1.0	1.0
faq_2	0.8333	0.9091
faq_31	0.8333	0.9091
faq_19	0.75	0.6667
faq_34	1.0	0.5714
faq_7	1.0	1.0
faq_10	1.0	1.0
faq_13	1.0	1.0
faq_47	1.0	1.0
faq_3	1.0	1.0
faq_46	1.0	1.0
faq_27	0.8	0.8000
faq_50	1.0	1.0
faq_29	1.0	0.889
faq_43	1.0	1.0
faq_25	1.0	1.0
faq_16	1.0	1.0
faq_22	1.0	1.0
faq_15	1.0	0.889
faq_37	1.0	1.0
faq_4	1.0	1.0
faq_9	1.0	1.0
faq_42	0.833	0.909
faq_28	0.75	0.667

- ابزار گفت‌وگوی تحت وب

در این مرحله با استفاده از پوشه WebChat قرار داده شده در تمرین، یک رابط تصویری تحت وب برای ربات خود بالا آوردیم. برای استفاده از این رابط کافی است کد زیر را اجرا کنیم:

```
rasa run -m models --enable-api --cors "*" –debug
```

سپس وارد پوشه WebChat شده و index.html را باز می کنیم و نتیجه به صورت زیر می باشد:

شکل ۲ : گفت‌وگو با ربات در رویکرد اول

- **مدل دوم ParsBert :**

- انجام تنظیمات و آموزش ربات

در این قسمت می خواهیم تنظیمات ربات در فایل config.yml را شرح داده و سپس ربات را با آن اموزش دهیم. برای این مدل ابتدا فایل config این مدل را گذاشته و قسمت های جدید آن نسبت به مدل قبل را توضیح می دهیم:

config.yml(ParsBert - 100 epoch)

```
language: fa
pipeline:
- name: WhitespaceTokenizer
- name: RegexFeaturizer
- name: LexicalSyntacticFeaturizer
- name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: char_wb
  min_ngram: 1
  max_ngram: 4
- name: LanguageModelFeaturizer
  model_name: bert
  model_weights: HooshvareLab/bert-base-parsbert-uncased
  constrain_similarities: true
- name: DIETClassifier
  epochs: 100
  constrain_similarities: true


policies:
```

در کد config.yml بالا تغییرات نسبت به مدل قبل مشخص شده است. همانطور که مشخص است، فقط خط کد model_weights عوض شده است و بقیه ساختار pipeline مانند قسمت قبل می‌باشد.

در این مدل به جای LaBSE از مدل ParsBert استفاده کردیم که یک مدل بر پایه Bert می باشد که بر روی دادگان فارسی آموزش دیده است و همینطور یک مدل از پیش آموزش دیده است و ما در اینجا از وزن های آن استفاده خواهیم کرد. که به ما embedding های از پیش آماده برای کلمات خواهد داد.

- **ارزیابی ربات**

به کمک دستور زیر ربات آموزش داده شده را ارزیابی می‌کنیم و فایل تست test_nlu.yml که در قسمت قبل ساختیم را به عنوان ورودی به آن می‌دهیم.

```
rasa test nlu --nlu test_nlu.yml
```

پس از اجرای کد تست، نتایج در پوشه results ذخیره می‌شوند. که نتایج بدست آمده به شرح زیر می‌باشد:

مقادیر نتایج برای هر اندیشه در فایل intent_report_parsbert_100.json گزارش شده است. در اینجا مقادیر دقت و f1-score را برای کل کلاس ها گزارش می‌کنیم.

Accuracy: 0.908

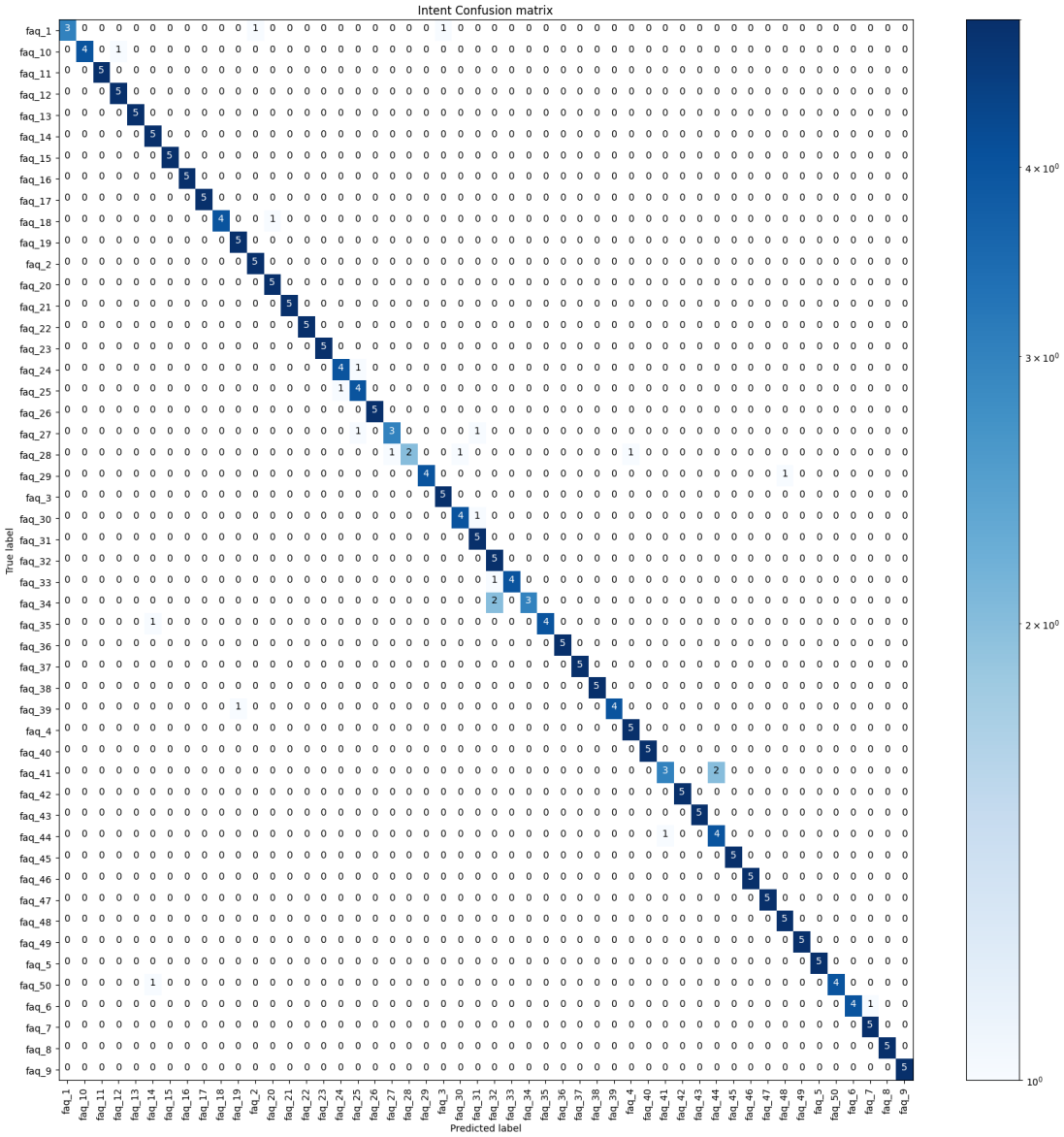
Macro_avg F1-score: 0.905

Weighted avg F1-score: 0.905

Micro_avg F1-score: 0.908

با توجه به توانایی LaBSE در دریافت شباهت‌های معنایی و داشتن contexual understanding از متن، می تواند عملکرد بهتری از مدل parsbert داشته باشد.

همانطور که از نتایج مشخص است مدل LaBSE بهتر از مدل ParsBert عمل کرده است پس آموزش با تعداد 50 و 200 را بر روی مدل LaBSE ادامه می دهیم.



شکل 3 : ماتریس آشفتگی برای مدل ParsBert با ۱۰۰ دور آموزش و رویکرد اول

جدول ۲ : precisionو f۱-score برای مدل ParsBert با ۱۰۰ دور آموزش و رویکرد اول

FAQ	Precision	F1-Score
faq_33	1.0	0.8889
faq_26	1.0	1.0
faq_30	0.8	0.8
faq_46	1.0	1.0
faq_50	1.0	0.8889
faq_44	0.6667	0.7273
faq_10	1.0	0.8889
faq_18	1.0	0.8889
faq_8	1.0	1.0
faq_21	1.0	1.0
faq_25	0.6667	0.7273
faq_16	1.0	1.0
faq_40	1.0	1.0
faq_41	0.75	0.6667
faq_32	0.625	0.7692
faq_38	1.0	1.0
faq_23	1.0	1.0
faq_22	1.0	1.0
faq_45	1.0	1.0
faq_15	1.0	1.0
faq_12	0.8333	0.9091
faq_17	1.0	1.0
faq_36	1.0	1.0
faq_13	1.0	1.0
faq_1	1.0	0.75

FAQ	Precision	F1-Score
faq_29	1.0	0.8889
faq_28	1.0	0.5714
faq_43	1.0	1.0
faq_6	1.0	0.8889
faq_48	0.8333	0.9091
faq_49	1.0	1.0
faq_37	1.0	1.0
faq_19	0.833	0.909
faq_14	0.714	0.833
faq_9	1.000	1.000
faq_7	0.833	0.909
faq_31	0.714	0.833
faq_39	1.000	0.889
faq_34	1.000	0.750
faq_11	1.000	1.000
faq_20	0.833	0.909
faq_5	1.000	1.000
faq_4	0.833	0.909
faq_42	1.000	1.000
faq_35	1.000	0.889
faq_47	1.000	1.000
faq_2	0.833	0.909
faq_27	0.750	0.667
faq_24	0.800	0.800
faq_3	0.833	0.909

- **مدل سوم LaBSE (۵۰ دور آموزش):**

- انجام تنظیمات و آموزش ربات

در این قسمت می خواهیم تنظیمات ربات در فایل config.yml را شرح داده و سپس ربات را با آن اموزش دهیم. برای این مدل ابتدا فایل config این مدل را گذاشته و قسمت های جدید آن نسبت به مدل قبل را توضیح می دهیم:

config.yml(LaBSE - 50 epoch)

```
language: fa
pipeline:
- name: WhitespaceTokenizer
- name: RegexFeaturizer
- name: LexicalSyntacticFeaturizer
- name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: char_wb
  min_ngram: 1
  max_ngram: 4
- name: LanguageModelFeaturizer
  model_name: bert
  model_weights: HooshvareLab/bert-base-parsbert-uncased
  constrain_similarities: true
- name: DIETClassifier
  epochs: 50
  constrain_similarities: true


policies:
```

در کد config.yml بالا تغییرات نسبت به مدل قبل مشخص شده است. همانطور که مشخص است، فقط خط کد epoch برای DIETClassifier عوض شده است و تعداد epoch ها کم شده است و بر روی ۵۰ قرار گرفته است.

- ارزیابی ربات

به کمک دستور زیر ربات آموزش داده شده را ارزیابی می‌کنیم و فایل تست test_nlu.yml که در قسمت قبل ساختیم را به عنوان ورودی به آن می‌دهیم.

```
rasa test nlu --nlu test_nlu.yml
```

پس از اجرای کد تست، نتایج در پوشه results ذخیره می‌شوند. که نتایج بدست آمده به شرح زیر می‌باشد:

مقادیر نتایج برای هر اندیشه در فایل intent_report_labse_50.json گزارش شده است. در اینجا مقادیر دقت و f1-score را برای کل کلاس ها گزارش می‌کنیم.

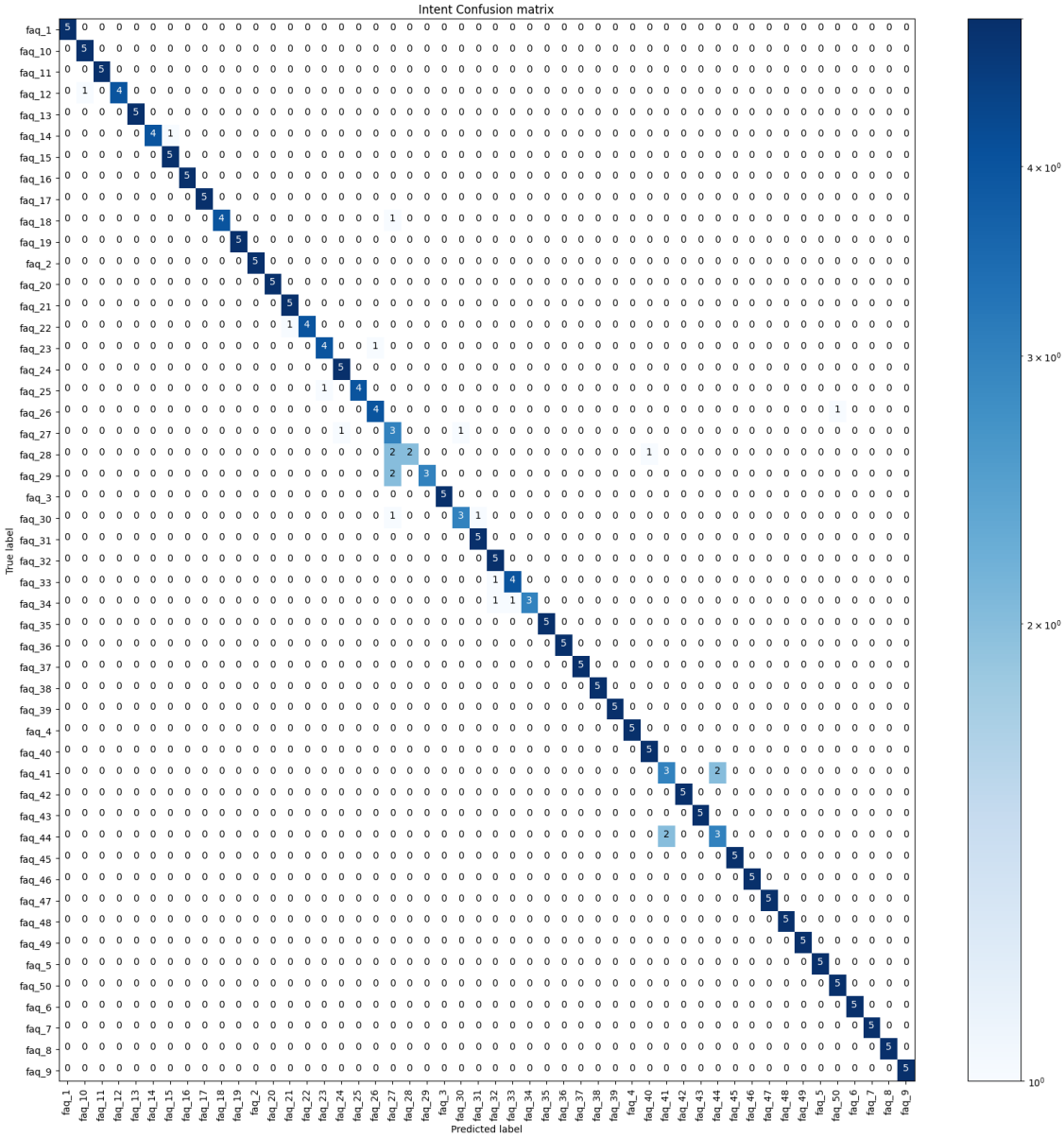
Accuracy: 0.908

Macro_avg F1-score: 0.908

Weighted avg F1-score: 0.908

Micro_avg F1-score: 0.908

همانطور که می‌بینیم دقت و f1 در ۵۰ دور نسبت به ۱۰۰ دور کاهش داشته است. یعنی ۵۰ دور، تعداد دور کافی برای آنکه مدل ما fit شود، نبوده است و عملکرد مدل هنوز به مقدار بهینه نرسیده است.



شکل ۴ : ماتریس آشفتگی برای مدل LaBSE با ۵۰ دور آموزش و رویکرد اول

جدول ۳: precision و f۱-score برای مدل LaBSE با ۵۰ دور آموزش و رویکرد اول

FAQ	Precision	F1-Score
faq_47	1.0	1.0
faq_36	1.0	1.0
faq_4	1.0	1.0
faq_22	1.0	0.8889
faq_12	1.0	0.8889
faq_10	0.8333	0.9091
faq_8	1.0	1.0
faq_44	0.6	0.6
faq_33	0.8	0.8
faq_35	1.0	1.0
faq_38	1.0	1.0
faq_19	1.0	1.0
faq_26	0.8	0.8
faq_15	0.8333	0.9091
faq_31	0.8333	0.9091
faq_25	1.0	0.8889
faq_46	1.0	1.0
faq_40	0.8333	0.9091
faq_13	1.0	1.0
faq_7	1.0	1.0
faq_18	1.0	0.8889
faq_29	1.0	0.75
faq_37	1.0	1.0
faq_39	1.0	1.0

FAQ	Precision	F1-Score
faq_32	0.7143	0.8333
faq_41	0.6	0.6
faq_50	0.8333	0.9091
faq_45	1.0	1.0
faq_43	1.0	1.0
faq_5	1.0	1.0
faq_28	1.0	0.5714
faq_1	1.0	1.0
faq_2	1.0	1.0
faq_27	0.3333	0.4286
faq_34	1.0	0.75
faq_3	1.0	1.0
faq_11	1.0	1.0
faq_23	0.8	0.8889
faq_17	1.0	1.0
faq_48	1.0	1.0
faq_20	1.0	1.0
faq_30	0.6	0.75
faq_24	1.0	1.0
faq_14	1.0	1.0
faq_21	1.0	1.0
faq_42	1.0	1.0
faq_16	1.0	1.0

- **مدل چهارم LaBSE (۲۰۰ دور آموزش):**

- انجام تنظیمات و آموزش ربات

در این قسمت می خواهیم تنظیمات ربات در فایل config.yml را شرح داده و سپس ربات را با آن اموزش دهیم. برای این مدل ابتدا فایل config این مدل را گذاشته و قسمت های جدید آن نسبت به مدل قبل را توضیح می دهیم:

config.yml(LaBSE - 200 epoch)

```
language: fa
pipeline:
- name: WhitespaceTokenizer
- name: RegexFeaturizer
- name: LexicalSyntacticFeaturizer
- name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: char_wb
  min_ngram: 1
  max_ngram: 4
- name: LanguageModelFeaturizer
  model_name: bert
  model_weights: HooshvareLab/bert-base-parsbert-uncased
  constrain_similarities: true
- name: DIETClassifier
  epochs: 200
  constrain_similarities: true


policies:
```

در کد config.yml بالا تغییرات نسبت به مدل قبل مشخص شده است. همانطور که مشخص است، فقط خط کد epoch برای DIETClassifier عوض شده است و تعداد epoch ها افزایش داشته است و بر روی ۲۰۰ قرار گرفته است.

- ارزیابی ربات

به کمک دستور زیر ربات آموزش داده شده را ارزیابی می‌کنیم و فایل تست test_nlu.yml که در قسمت قبل ساختیم را به عنوان ورودی به آن می‌دهیم.

```
rasa test nlu --nlu test_nlu.yml
```

پس از اجرای کد تست، نتایج در پوشه results ذخیره می‌شوند. که نتایج بدست آمده به شرح زیر می‌باشد:

مقادیر نتایج برای هر اندیشه در فایل intent_report_labse_200.json گزارش شده است. در اینجا مقادیر دقت و f1-score را برای کل کلاس ها گزارش می‌کنیم.

Accuracy: 0.912

Macro_avg F1-score: 0.909

Weighted avg F1-score: 0.909

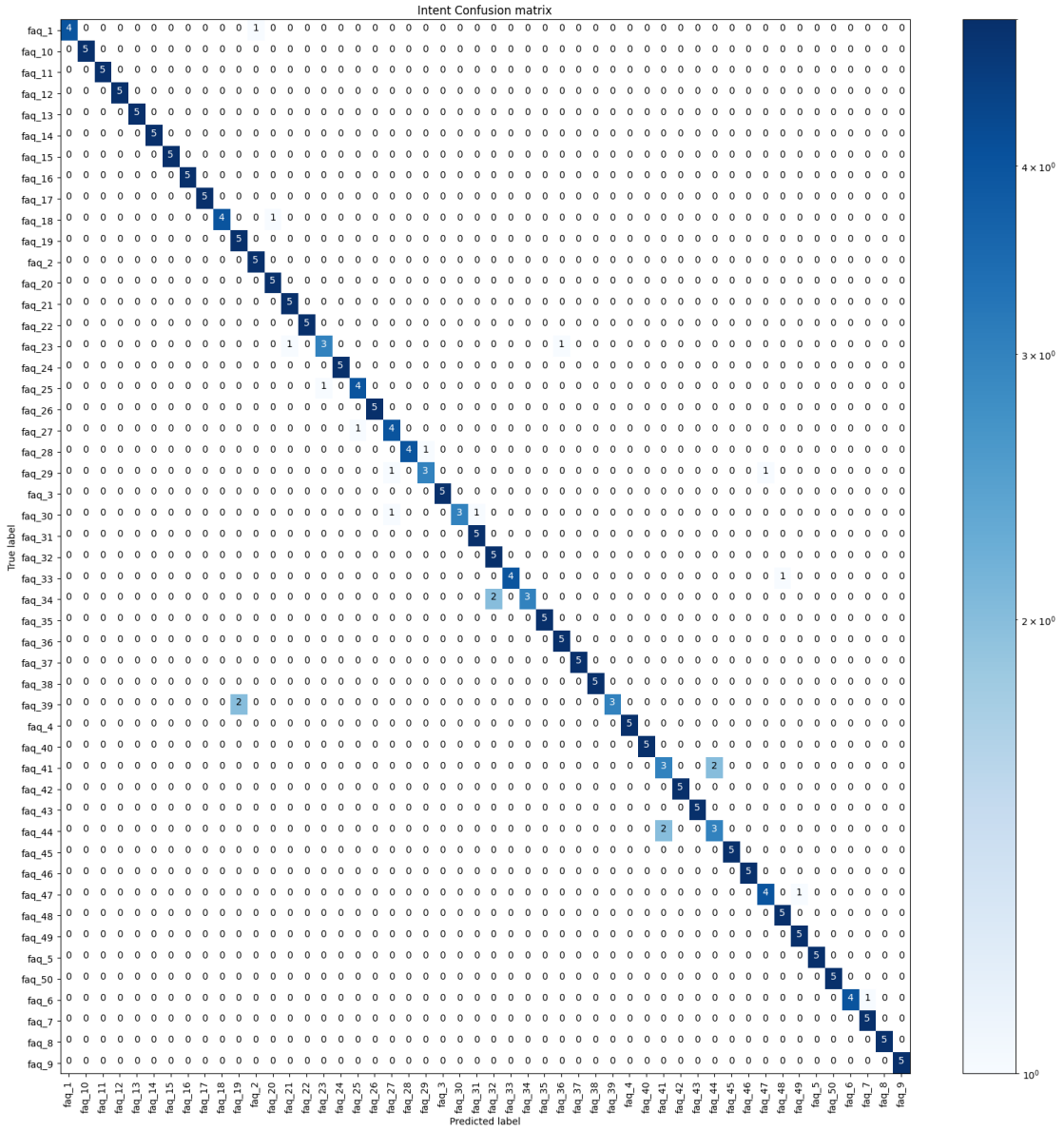
Micro_avg F1-score: 0.912

همانطور که از دقت و f1-score بالا می توان دید، در تکرار ۲۰۰ بار دقت نسبت به ۱۰۰ تکرار کاهش پیدا کرده است. پس نتیجه می‌گیریم که مدل بر روی دیتاست overfit کرده است و در generalize شدن ناموفق بوده است. به همین علت عملکرد مدل بر روی داده تست کاهش پیدا کرده است.

جدول ۴ : precisionو f۱-score برای مدل LaBSE با ۲۰۰ دور آموزش و رویکرد اول

FAQ	Precision	F1-Score
faq_37	1.0	1.0
faq_2	0.833	0.909
faq_6	1.0	0.889
faq_18	1.0	0.889
faq_28	1.0	0.889
faq_3	1.0	1.0
faq_33	1.0	0.889
faq_8	1.0	1.0
faq_49	0.833	0.909
faq_40	1.0	1.0
faq_5	1.0	1.0
faq_41	0.6	0.6
faq_43	1.0	1.0
faq_10	1.0	1.0
faq_39	1.0	0.75
faq_25	0.8	0.8
faq_9	1.0	1.0
faq_13	1.0	1.0
faq_32	0.714	0.833
faq_19	0.714	0.833
faq_35	1.0	1.0
faq_4	1.0	1.0
faq_15	1.0	1.0
faq_7	0.833	0.909
faq_29	0.75	0.667

FAQ	Precision	F1-Score
faq_48	0.833	0.909
faq_38	1.0	1.0
faq_30	1.0	0.75
faq_17	1.0	1.0
faq_24	1.0	1.0
faq_46	1.0	1.0
faq_27	0.667	0.727
faq_47	0.8	0.8
faq_34	1.0	0.75
faq_16	1.0	1.0
faq_44	0.6	0.6
faq_1	1.0	0.889
faq_45	1.0	1.0
faq_31	0.833	0.909
faq_50	1.0	1.0
faq_36	0.833	0.909
faq_26	1.0	1.0
faq_20	0.833	0.909
faq_22	1.0	1.0
faq_23	0.75	0.667
faq_21	0.833	0.909
faq_12	1.0	1.0
faq_42	1.0	1.0
faq_11	1.0	1.0
faq_14	1.0	1.0



شکل ۵ : ماتریس آشفتگی برای مدل LaBSE با ۲۰۰ دور آموزش و رویکرد اول

○ رویکرد دوم : یک کلاس یا اندیشه واحد به نام پرسش‌وپاسخ پرتکرار

در این مرحله می‌خواهیم رویکرد دوم را پیاده‌سازی کنیم. یعنی در اینجا تمام ۵۰ کلاسintent قسمت قبل هر کدام یک subintent می شوند و جزئی از یک کلاس مادر پرسش و پاسخ پرتکرار می‌باشند. البته در این قسمت دو کلاس greet و goodbye نیز اضافه می‌کنیم. یعنی در این قسمت سه intent و برای intent=faq ۵۰ subintent داریم.

- **آماده‌سازی داده‌ها به صورت خودکار**

پرونده nlu.yml: در این قسمت کدی را نوشتیم که در ابتدا دادهای پرونده MCI_Internet_TrainData را خوانده و در یک دیتافریم ذخیره می‌کند. سپس فایل nlu.yml را با دسترسی write باز می‌کند. در ادامه در یک حلقه بر روی سطرهای دیتافریم حرکت کرده و در هر مرحله با فرمت مناسب ابتدا نام intent در اینجا faq را می‌نویسد و سپس subintent را با / شماره گذاری می‌کند. و سپس example های آن را از سطر ۸ تا ۲۲ دیتافریم در فایل nlu.yml قرار می دهد. فرمت نوشتن در فایل به صورت زیر می باشد.

تفاوت ای مرحله با روش قبل در این می‌باشد که چون می‌خواهیم subintent تعریف کنیم به جای faq_1 از faq/1 استفاده می کنیم.

```
nlu:
- intent: faq/1
  examples: |
    - با استفاده از کد دستوری چیکار کنم؟ gprs برای فعال سازی سرویس
    - چه شماره ای باید بگیرم؟ gprs برای فعالسازی سرویس
    - چه کدی رو باید بزنم؟ gprs برای فعالسازی سرویس
    - با استفاده از کد دستوری رو فعال کنم؟ gprs چجوری سرویس اینترنت
```

همچنین در این مرحله دو کلاس greet و goodbye نیز اضافه می‌کنیم که پیاده‌سازی چند جمله اول هر کلام از intent ها به صورت زیر می‌باشد:

```
- intent: greet
  examples: |
    - سلام
    - سلاام
    - سلام ربات
    - سلاام ربات
    - چطوری
```

```
- intent: goodbye
  examples: |
    - ممنون خداحافظ
    - خدا حافظ
    - بای بای ربات
    - خداحافظ ربات عزیز
```

پرونده domain.yml: در این قسمت کدی را نوشتیم که در ابتدا دادهای پرونده MCI_Internet_TrainData را خوانده و در یک دیتافریم ذخیره می‌کند. سپس فایل domain.yml را با دسترسی write باز می‌کند. در ادامه ابتدا نام کل intent ها در اینجا (faq, greet , goodbye) را در فایل می‌نویسد. سپس در یک حلقه بر روی سطرهای دیتافریم حرکت کرده و در هر مرحله با فرمت مناسب سپس تمامی response ها را در فایل domain با فرمت مناسب می نویسد. فرمول نوشتن در هر سطر به صورت زیر می‌باشد.

تفاوت ای مرحله با روش قبل در این می‌باشد که چون می‌خواهیم subintent تعریف کنیم به جای utter_faq_1 از utter_faq/1 استفاده می کنیم.

```
responses:
  utter_faq/1:
  - text: "کد دستوری ۱۰۰ ستاره ۱۱۱ مربع را شماره گیری نمایید (GPRS) مشترک گرامی جهت فعالسازی سرویس اینترنت "
```

همچنین در این مرحله پاسخ دو کلاس greet و goodbye نیز اضافه می‌کنیم که پیاده‌سازی آن ها به صورت زیر می‌باشد:

```
  utter_greet:
  - text: "سلام خوبی؟ من ربات همراه اولم!"

  utter_goodbye:
  - text: "به امید دیدار مجدد :)"
```

پرونده rules.yml: در این قسمت به ازای هر intent ، قوانین پاسخ به آن را تعریف کند و در این قسمت فقط هر intent را به پاسخش ارتباط می‌دهیم که به صورت زیر نوشته می‌شود.

```
- rule: faq
  steps:
  - intent: faq
  - action: utter_faq

- rule: greet
  steps:
  - intent: greet
  - action: utter_greet

- rule: goodbye
  steps:
  - intent: goodbye
  - action: utter_goodbye
```

پرونده nlu_test.yml: در این قسمت کدی را نوشتیم که در ابتدا دادهای پرونده MCI_Internet_TestData را خوانده و در یک دیتافریم ذخیره می‌کند. سپس فایل test_nlu.yml را با دسترسی write باز می‌کند. در ادامه در یک حلقه بر روی سطرهای دیتافریم حرکت کرده و هر هر مرحله با فرمت مناسب ابتدا نام intent و subintent را می‌نویسد و سپس example های آن را از سطر ۸ تا ۱۲ دیتافریم در فایل test_nlu.yml قرار می دهد. همچنین فرمت نوشتن در این فایل به صورت زیر می باشد:

```
- intent: faq/1
  examples: |
    - سرویس اینترنتی با استفاده از کد دستوری چجوری فعال میشه؟
    - چجوری سرویس اینترنت با استفاده از کد دستوری فعال میشه؟
    - سرویس اینترنتم نیاز به فعالسازی داره؟ کد دستوری داره؟
    - چجوری سرویس اینترنتم رو با استفاده از کد دستوری فعال کنم؟
    - فعال شدن سرویس اینترنت چجوریه؟
```

همانطور که در صورت سوال گفته شده در این تمرین قرار است عملکرد مدل های زبانی بر پایه Bert را بررسی کنیم. به همین دلیل ابتدا مدل‌های ParsBert و LaBSE را هرکدام با ۱۰۰ دور آموزش می آزماییم و برای مدل بهتر بر روی ۵۰ و ۲۰۰ دور نیز آموزش داده و آن را گزارش می کنیم.

- **مدل اول LaBSE (۱۰۰ دور آموزش):**

- انجام تنظیمات و آموزش ربات

در این قسمت می خواهیم تنظیمات ربات در فایل config.yml را شرح داده و سپس ربات را با آن اموزش دهیم. برای این مدل ابتدا فایل config این مدل را گذاشته و هر قسمت آن را توضیح می دهیم:

config.yml(LaBSE - 100 epoch)

```
language: fa
pipeline:
- name: WhitespaceTokenizer
- name: RegexFeaturizer
- name: LexicalSyntacticFeaturizer
- name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: char_wb
  min_ngram: 1
  max_ngram: 4
- name: LanguageModelFeaturizer
  model_name: bert
  model_weights: rasa/LaBSE
  constrain_similarities: true
- name: DIETClassifier
  epochs: 100
  constrain_similarities: true
- name: ResponseSelector
  epochs: 100
  constrain_similarities: true
  retrieval_intent: faq


policies:
```

کد بالا مانند کد اول می‌باشد با این تفاوت که در اینجا responseSelcetor اضافه شده است. اضافه کردن این خط برای آن است که responseSelcetor پس از انجام classification بر روی intent ها، برای انتخاب responce آن intent از میان subintent های تعریف شده و پاسخ آن می‌باشد و به این صورت عمل می‌کند که عمل

training را دوباره و فقط بر روی داده های retrieval_intent مشخص شده انجام می‌دهد. تا مدل بتواند در میان جملات subintentکلاسیفای کند.

- **ارزیابی ربات**

به کمک دستور زیر ربات آموزش داده شده را ارزیابی می‌کنیم و فایل تست test_nlu.yml که در قسمت قبل ساختیم را به عنوان ورودی به آن می‌دهیم.

```
rasa test nlu --nlu test_nlu.yml
```

پس از اجرای کد تست، نتایج در پوشه results ذخیره می‌شوند. که نتایج بدست آمده به شرح زیر می‌باشد:

مقادیر نتایج برای هر اندیشه در فایل 2_intent_report_labse_100.json گزارش شده است. در اینجا مقادیر دقت و f1-score را برای کل کلاس ها گزارش می‌کنیم.

Accuracy: 0.94

Macro_avg F1-score: 0.936

Weighted avg F1-score: 0.936

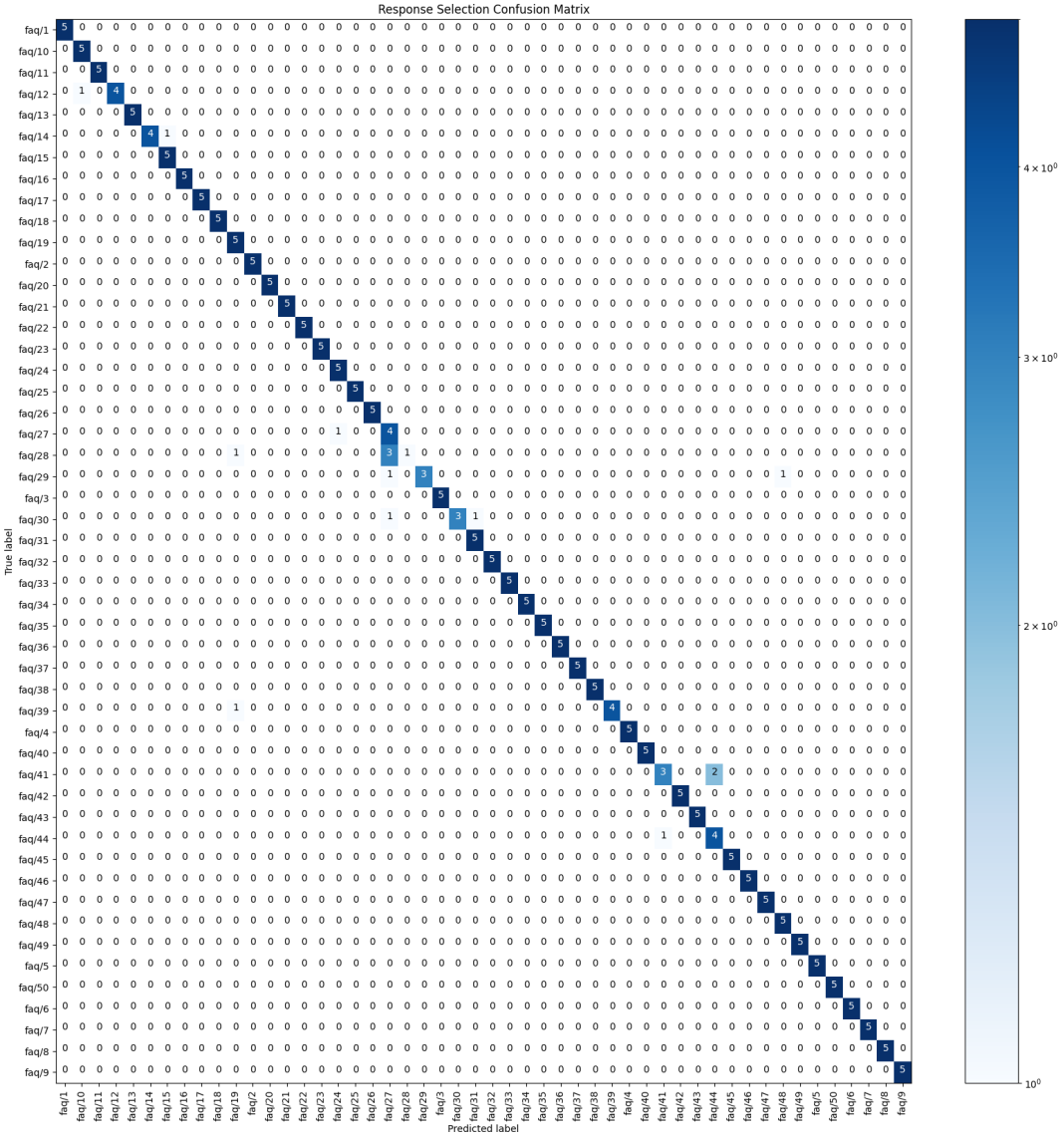
Micro_avg F1-score: 0.94

همانطور که می‌بینیم در رویکرد اول نسبت به رویکرد دوم عملکرد بهتری خواهیم داشت. یکی از دلایل می‌تواند این باشد که ما ابتدا یک faq با ۵۰ تا زیر کلاس ساختیم و با یکی کردن کل سوال ها در یک کلاس faq، برای مدل فهم faq ها واضح‌تر می شود همچنین مدل می‌تواند تفاوت میان sub-intent ها را بهتر یاد بگیرد. پس نتیجه می‌گیریم استفاده از sub-intent دقت مدل را بالاتر می‌برد.

جدول ۵: precisionو f۱-score برای مدل LaBSE با ۱۰۰ دور آموزش و رویکرد دوم

FAQ	Precision	F1-Score
faq/9	1.0	1.0
faq/17	1.0	1.0
faq/39	1.0	0.8889
faq/31	0.8333	0.9091
faq/11	1.0	1.0
faq/23	1.0	1.0
faq/37	1.0	1.0
faq/30	1.0	0.75
faq/8	1.0	1.0
faq/21	1.0	1.0
faq/13	1.0	1.0
faq/47	1.0	1.0
faq/3	1.0	1.0
faq/35	1.0	1.0
faq/10	0.8333	0.9091
faq/7	1.0	1.0
faq/18	1.0	1.0
faq/4	1.0	1.0
faq/32	1.0	1.0
faq/46	1.0	1.0
faq/25	1.0	1.0
faq/50	1.0	1.0
faq/48	0.8333	0.9091
faq/15	0.8333	0.9091
faq/6	1.0	1.0

FAQ	Precision	F1-Score
faq/28	1.0	0.3333
faq/5	1.0	1.0
faq/12	1.0	0.8889
faq/26	1.0	1.0
faq/36	1.0	1.0
faq/1	1.0	1.0
faq/42	1.0	1.0
faq/43	1.0	1.0
faq/44	0.6667	0.7273
faq/34	1.0	1.0
faq/19	0.7143	0.8333
faq/27	0.444	0.571
faq/2	1.000	1.000
faq/45	1.000	1.000
faq/40	1.000	1.000
faq/41	0.750	0.667
faq/49	1.000	1.000
faq/16	1.000	1.000
faq/24	0.833	0.909
faq/29	1.000	0.750
faq/20	1.000	1.000
faq/38	1.000	1.000
faq/14	1.000	0.889
faq/33	1.000	1.000
faq/22	1.000	1.000



شکل ۶ : ماتریس آشفتگی برای مدل LaBSE با ۱۰۰ دور آموزش و رویکرد دوم

- ابزار گفت‌وگوی تحت وب

در این مرحله با استفاده از پوشه WebChat قرار داده شده در تمرین، یک رابط تصویری تحت وب برای ربات خود بالا آوردیم. برای استفاده از این رابط کافی است کد زیر را اجرا کنیم:

```
rasa run -m models --enable-api --cors "*" –debug
```

سپس وارد پوشه WebChat شده و index.html را باز می کنیم و نتیجه به صورت زیر می باشد:

شکل ۷ : گفت‌وگو با ربات در رویکرد دوم

- **مدل دوم ParsBert :**

- انجام تنظیمات و آموزش ربات

در این قسمت می خواهیم تنظیمات ربات در فایل config.yml را شرح داده و سپس ربات را با آن اموزش دهیم. برای این مدل ابتدا فایل config این مدل را گذاشته و قسمت های جدید آن نسبت به مدل قبل را توضیح می دهیم:

config.yml(ParsBert - 100 epoch)

```
language: fa
pipeline:
- name: WhitespaceTokenizer
- name: RegexFeaturizer
- name: LexicalSyntacticFeaturizer
- name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: char_wb
  min_ngram: 1
  max_ngram: 4
- name: LanguageModelFeaturizer
  model_name: bert
  model_weights: HooshvareLab/bert-base-parsbert-uncased
  constrain_similarities: true
- name: DIETClassifier
  epochs: 100
  constrain_similarities: true
- name: ResponseSelector
  epochs: 100
  constrain_similarities: true
  retrieval_intent: faq


policies:
```

در کد config.yml بالا تغییرات نسبت به مدل قبل مشخص شده است. همانطور که مشخص است، فقط خط کد model_weights عوض شده است و بقیه ساختار pipeline مانند قسمت قبل می‌باشد.

در این مدل به جای LaBSE از مدل ParsBert استفاده کردیم که یک مدل بر پایه Bert می باشد که بر روی دادگان فارسی آموزش دیده است و همینطور یک مدل از پیش آموزش دیده است و ما در اینجا از وزن های آن استفاده خواهیم کرد. که به ما embedding های از پیش آماده برای کلمات خواهد داد.

- **ارزیابی ربات**

به کمک دستور زیر ربات آموزش داده شده را ارزیابی می‌کنیم و فایل تست test_nlu.yml که در قسمت قبل ساختیم را به عنوان ورودی به آن می‌دهیم.

```
rasa test nlu --nlu test_nlu.yml
```

پس از اجرای کد تست، نتایج در پوشه results ذخیره می‌شوند. که نتایج بدست آمده به شرح زیر می‌باشد:

مقادیر نتایج برای هر اندیشه در فایل 2_intent_report_parsbert_100.json گزارش شده است. در اینجا مقادیر دقت و f1-score را برای کل کلاس ها گزارش می‌کنیم.

Accuracy: 0.904

Macro_avg F1-score: 0.898

Weighted avg F1-score: 0.898

Micro_avg F1-score: 0.904

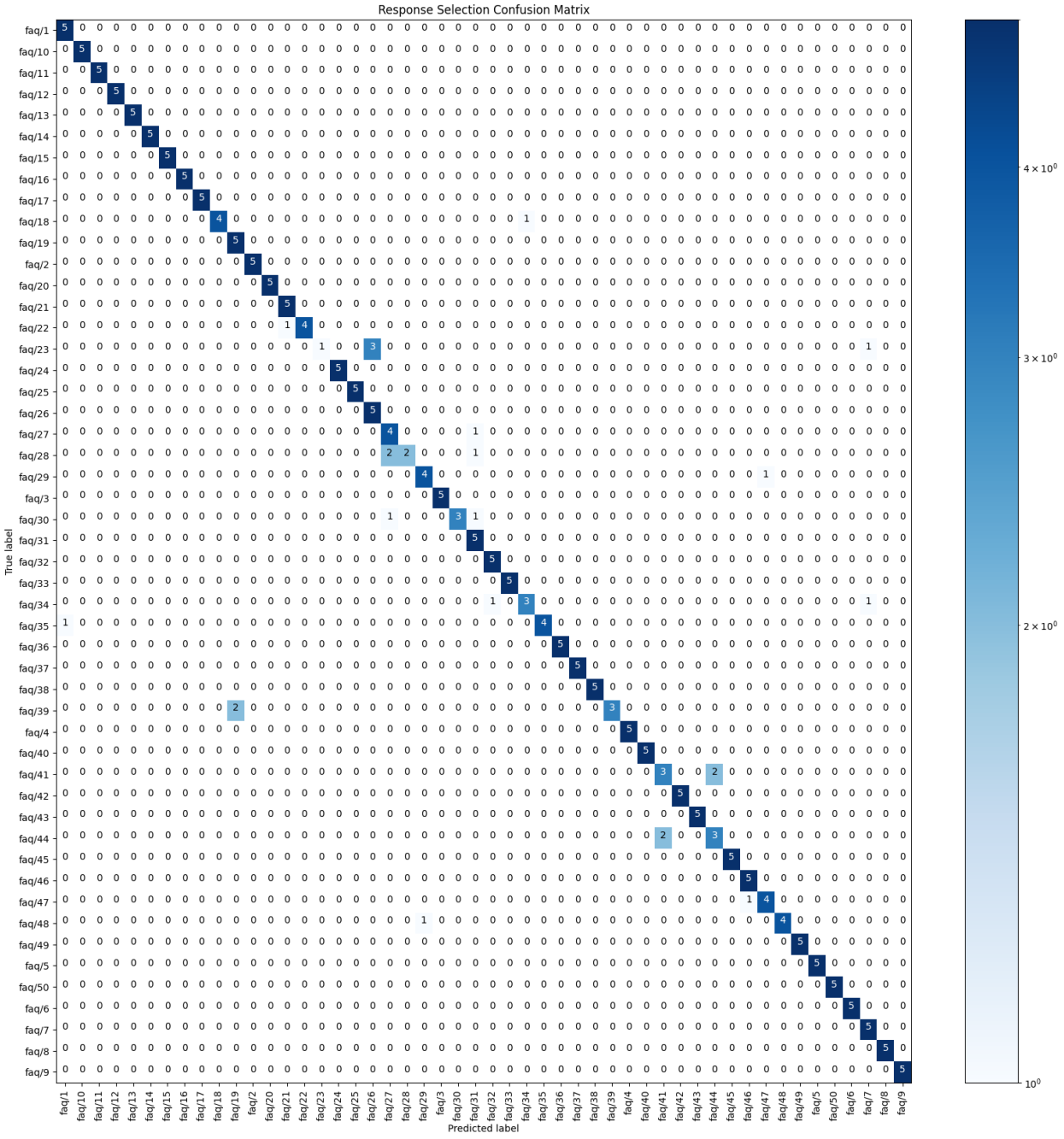
با توجه به توانایی LaBSE در دریافت شباهت‌های معنایی و داشتن contexual understanding از متن، می تواند عملکرد بهتری از مدل parsbert داشته باشد.

همانطور که از نتایج مشخص است مدل LaBSE بهتر از مدل ParsBert عمل کرده است پس آموزش با تعداد 50 و 200 را بر روی مدل LaBSE ادامه می دهیم.

جدول ۶: precisionو f۱-score برای مدل ParsBert با ۱۰۰ دور آموزش و رویکرد دوم

FAQ	Precision	F1-Score
faq/17	1.0	1.0
faq/16	1.0	1.0
faq/36	1.0	1.0
faq/10	1.0	1.0
faq/15	1.0	1.0
faq/5	1.0	1.0
faq/35	1.0	0.888
faq/12	1.0	1.0
faq/32	0.833	0.909
faq/39	1.0	0.75
faq/45	1.0	1.0
faq/41	0.6	0.6
faq/1	0.833	0.909
faq/24	1.0	1.0
faq/48	1.0	0.888
faq/7	0.714	0.833
faq/2	1.0	1.0
faq/20	1.0	1.0
faq/8	1.0	1.0
faq/19	0.714	0.833
faq/28	1.0	0.571
faq/31	0.625	0.769
faq/25	1.0	1.0
faq/49	1.0	1.0
faq/43	1.0	1.0

FAQ	Precision	F1-Score
faq/38	1.0	1.0
faq/14	1.0	1.0
faq/34	0.75	0.667
faq/18	1.0	0.889
faq/3	1.0	1.0
faq/42	1.0	1.0
faq/26	0.625	0.769
faq/37	1.0	1.0
faq/4	1.0	1.0
faq/13	1.0	1.0
faq/29	0.8	0.8
faq/11	1.0	1.0
faq/9	1.0	1.0
faq/21	0.833	0.909
faq/6	1.0	1.0
faq/44	0.6	0.6
faq/30	1.0	0.75
faq/23	1.0	0.333
faq/40	1.0	1.0
faq/27	0.571	0.667
faq/22	1.0	0.889
faq/33	1.0	1.0
faq/46	0.833	0.909
faq/50	1.0	1.0
faq/47	0.8	0.8



شکل ۸ : ماتریس آشفتگی برای مدل ParsBert با ۱۰۰ دور آموزش و رویکرد دوم

- **مدل سوم LaBSE (۵۰ دور آموزش):**

- انجام تنظیمات و آموزش ربات

در این قسمت می خواهیم تنظیمات ربات در فایل config.yml را شرح داده و سپس ربات را با آن اموزش دهیم. برای این مدل ابتدا فایل config این مدل را گذاشته و قسمت های جدید آن نسبت به مدل قبل را توضیح می دهیم:

config.yml(LaBSE - 50 epoch)

```
language: fa
pipeline:
- name: WhitespaceTokenizer
- name: RegexFeaturizer
- name: LexicalSyntacticFeaturizer
- name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: char_wb
  min_ngram: 1
  max_ngram: 4
- name: LanguageModelFeaturizer
  model_name: bert
  model_weights: HooshvareLab/bert-base-parsbert-uncased
  constrain_similarities: true
- name: DIETClassifier
  epochs: 50
  constrain_similarities: true


policies:
```

در کد config.yml بالا تغییرات نسبت به مدل قبل مشخص شده است. همانطور که مشخص است، فقط خط کد epoch برای DIETClassifier عوض شده است و تعداد epoch ها کم شده است و بر روی ۵۰ قرار گرفته است.

- ارزیابی ربات

به کمک دستور زیر ربات آموزش داده شده را ارزیابی می‌کنیم و فایل تست test_nlu.yml که در قسمت قبل ساختیم را به عنوان ورودی به آن می‌دهیم.

```
rasa test nlu --nlu test_nlu.yml
```

پس از اجرای کد تست، نتایج در پوشه results ذخیره می‌شوند. که نتایج بدست آمده به شرح زیر می‌باشد:

مقادیر نتایج برای هر اندیشه در فایل 2_intent_report_labse_50.json گزارش شده است. در اینجا مقادیر دقت و f1-score را برای کل کلاس ها گزارش می‌کنیم.

Accuracy: 0.932

Macro_avg F1-score: 0.928

Weighted avg F1-score: 0.928

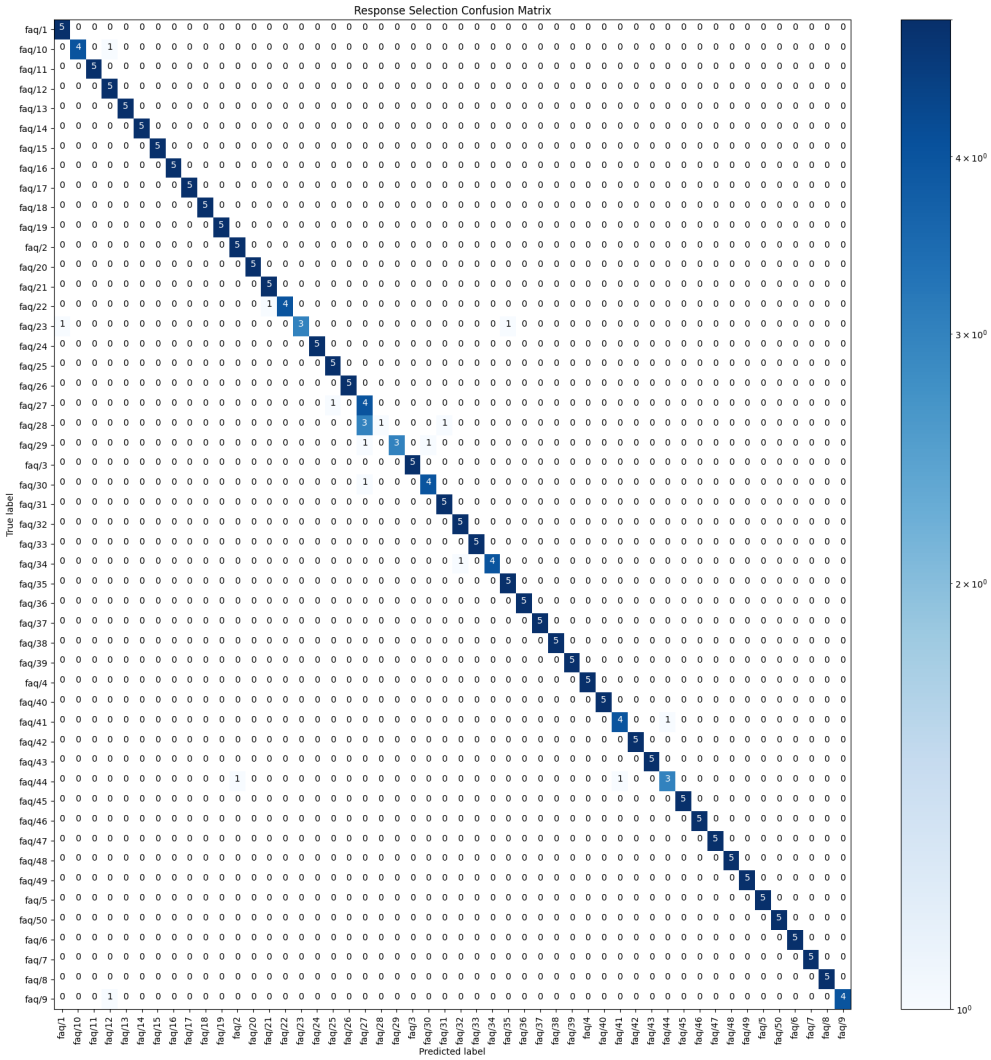
Micro_avg F1-score: 0.932

همانطور که می‌بینیم دقت و f1 در ۵۰ دور نسبت به ۱۰۰ دور کاهش داشته است. یعنی ۵۰ دور، تعداد دور کافی برای آنکه مدل ما fit شود نبوده است و عملکرد مدل هنوز به مقدار بهینه نرسیده است.

جدول ۷: precision و f۱-score برای مدل LaBSE با ۵۰ دور آموزش و رویکرد دوم

FAQ	Precision	F1-Score
faq/19	1.0	1.0
faq/44	0.75	0.6667
faq/22	1.0	0.8889
faq/11	1.0	1.0
faq/2	0.8333	0.9091
faq/41	0.8	0.8
faq/39	1.0	1.0
faq/33	1.0	1.0
faq/46	1.0	1.0
faq/47	1.0	1.0
faq/1	0.8333	0.9091
faq/20	1.0	1.0
faq/15	1.0	1.0
faq/29	1.0	0.75
faq/36	1.0	1.0
faq/24	1.0	1.0
faq/10	1.0	0.8889
faq/17	1.0	1.0
faq/42	1.0	1.0
faq/4	1.0	1.0
faq/18	1.0	1.0
faq/30	0.8	0.8
faq/3	1.0	1.0
faq/12	0.7143	0.8333
faq/28	1.0	0.3333

FAQ	Precision	F1-Score
faq/8	1.0	1.0
faq/43	1.0	1.0
faq/32	0.8333	0.9091
faq/50	1.0	1.0
faq/40	1.0	1.0
faq/7	1.0	1.0
faq/6	1.0	1.0
faq/14	1.0	1.0
faq/13	1.0	1.0
faq/21	0.833	0.909
faq/48	1.0	1.0
faq/38	1.0	1.0
faq/23	1.0	0.75
faq/25	0.833	0.909
faq/16	1.0	1.0
faq/26	1.0	1.0
faq/35	0.833	0.909
faq/45	1.0	1.0
faq/34	1.0	0.889
faq/9	1.0	0.889
faq/37	1.0	1.0
faq/49	1.0	1.0
faq/5	1.0	1.0
faq/31	0.833	0.909
faq/27	0.444	0.571



شکل ۹ : ماتریس آشفتگی برای مدل LaBSE با ۵۰ دور آموزش و رویکرد دوم

- **مدل چهارم LaBSE (۲۰۰ دور آموزش):**

- انجام تنظیمات و آموزش ربات

در این قسمت می خواهیم تنظیمات ربات در فایل config.yml را شرح داده و سپس ربات را با آن اموزش دهیم. برای این مدل ابتدا فایل config این مدل را گذاشته و قسمت های جدید آن نسبت به مدل قبل را توضیح می دهیم:

config.yml(LaBSE - 200 epoch)

```
language: fa
pipeline:
- name: WhitespaceTokenizer
- name: RegexFeaturizer
- name: LexicalSyntacticFeaturizer
- name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: char_wb
  min_ngram: 1
  max_ngram: 4
- name: LanguageModelFeaturizer
  model_name: bert
  model_weights: HooshvareLab/bert-base-parsbert-uncased
  constrain_similarities: true
- name: DIETClassifier
  epochs: 200
  constrain_similarities: true


policies:
```

در کد config.yml بالا تغییرات نسبت به مدل قبل مشخص شده است. همانطور که مشخص است، فقط خط کد epoch برای DIETClassifier عوض شده است و تعداد epoch ها افزایش داشته است و بر روی ۲۰۰ قرار گرفته است.

- ارزیابی ربات

به کمک دستور زیر ربات آموزش داده شده را ارزیابی می‌کنیم و فایل تست test_nlu.yml که در قسمت قبل ساختیم را به عنوان ورودی به آن می‌دهیم.

```
rasa test nlu --nlu test_nlu.yml
```

پس از اجرای کد تست، نتایج در پوشه results ذخیره می‌شوند. که نتایج بدست آمده به شرح زیر می‌باشد:

مقادیر نتایج برای هر اندیشه در فایل 2_intent_report_labse_200.json گزارش شده است. در اینجا مقادیر دقت و f1-score را برای کل کلاس ها گزارش می‌کنیم.

Accuracy: 0.936

Macro_avg F1-score: 0.934

Weighted avg F1-score: 0.934

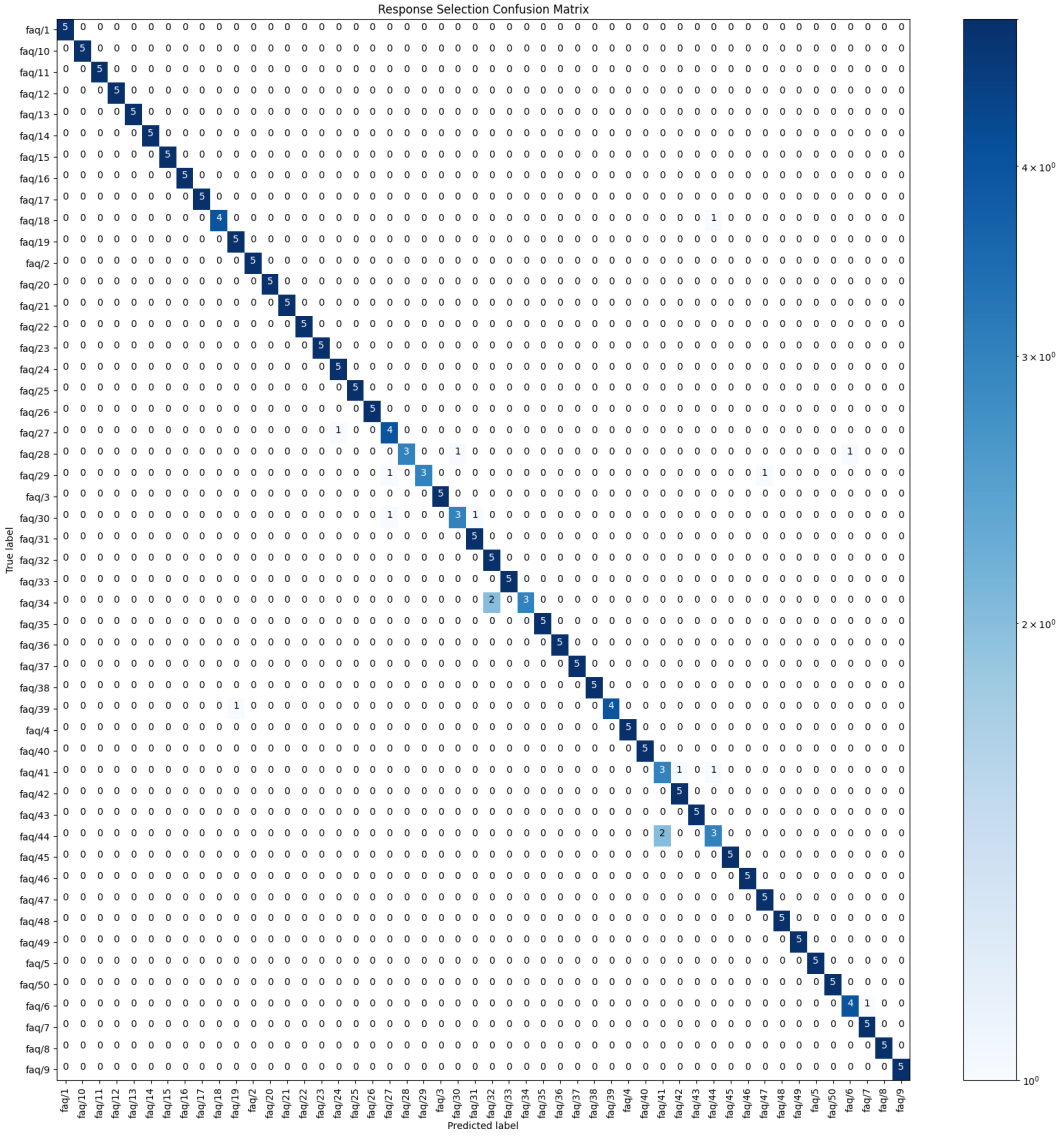
Micro_avg F1-score: 0.936

همانطور که از دقت و f1-score بالا می توان دید، در تکرار ۲۰۰ بار دقت نسبت به ۱۰۰ تکرار کاهش پیدا کرده است. پس نتیجه می‌گیریم که مدل بر روی دیتاست overfit کرده است و در generalize شدن ناموفق بوده است. به همین علت عملکرد مدل بر روی داده تست کاهش پیدا کرده است.

جدول ۸: precisionو f۱-score برای مدل LaBSE با ۲۰۰ دور آموزش و رویکرد دوم

FAQ	Precision	F1-Score
faq/18	1.0	0.8889
faq/14	1.0	1.0
faq/13	1.0	1.0
faq/24	0.8333	0.9091
faq/6	0.8	0.8
faq/25	1.0	1.0
faq/36	1.0	1.0
faq/48	1.0	1.0
faq/42	0.8333	0.9091
faq/50	1.0	1.0
faq/33	1.0	1.0
faq/8	1.0	1.0
faq/45	1.0	1.0
faq/12	1.0	1.0
faq/22	1.0	1.0
faq/29	1.0	0.75
faq/32	0.7143	0.8333
faq/20	1.0	1.0
faq/46	1.0	1.0
faq/34	1.0	0.75
faq/43	1.0	1.0
faq/3	1.0	1.0
faq/38	1.0	1.0
faq/15	1.0	1.0
faq/41	0.6	0.6

FAQ	Precision	F1-Score
faq/44	0.6	0.6
faq/40	1.0	1.0
faq/17	1.0	1.0
faq/27	0.6667	0.7273
faq/9	1.0	1.0
faq/47	0.8333	0.9091
faq/28	1.0	0.75
faq/31	0.833	0.909
faq/26	1.000	1.000
faq/39	1.000	0.889
faq/30	0.750	0.667
faq/11	1.000	1.000
faq/7	0.833	0.909
faq/4	1.000	1.000
faq/5	1.000	1.000
faq/10	1.000	1.000
faq/37	1.000	1.000
faq/2	1.000	1.000
faq/49	1.000	1.000
faq/19	0.833	0.909
faq/35	1.000	1.000
faq/23	1.000	1.000
faq/1	1.000	1.000
faq/16	1.000	1.000
faq/21	1.000	1.000



شکل ۱۰ : ماتریس آشفتگی برای مدل LaBSE با ۲۰۰ دور آموزش و رویکرد دوم

تحلیل نتایج هرکدام از رویکردها:

با بررسی نتایج دو مدل متوجه شدیم که **مدل دوم(استفاده از sub-intent) عملکرد بهتری دارد**. به طور کلی استفاده از sub-intent برای پرسش‌وپاسخ های پرکابرد روشی موثر می‌باشد زیرا باعث می‌شود که مدل زبانی تفاوت‌های هرکدام از سوالات از یکدیگر را بهتر شناسایی کرده و پترن‌های بیشتری را برای هرکدام از faq ها بدست بیاورد.

همینطور به طور کلی دیدیم که در **هر دو رویکرد مدل با ۱۰۰ دور بهترین دقت را داشت** که دلیل آن overfit شدن در ۲۰۰ دور و کمی underfit شدن در ۵۰ دور می‌باشد.
همچنین در هر دو رویکرد LaBSE عملکرد بهتری داشت. پس مدل با رویکرد دوم با تعداد دور ۱۰۰ و استفاده از مدل LaBSE بهترین نتیجه‌گیری را داشت.

پرسش ۲: استخراج مقادیر ارزش‌ها

در این پرسش می خواهیم با استخراج مقادیر ارزش‌ها در متن صحبت با کاربر یک ربات را با نرم افزار rasa پیاده سازی کنیم. در این سوال باید ۲ نوع ارزش از متن استخراج کنیم. ارزش اول که برای بدست آورن آن از عبارت های منظم استفاده می‌کنیم و در Rasa از ابزار regex entity extractor برای بدست آورن آن استفاده خواهیم کرد و ارزش دوم که نمی‌توانیم آن را با عبارت های منظم بیان کنیم و برای آن باید از CRF entity extractor استفاده کرد.

در این سوال بنده با شماره دانشجویی ۸۱۰۱۰۰۴۴۷ سناریو اختصاص داده‌شده به من با توجه به فرمول داده شده، 4=(447%6+1) سناریو ۴ می‌باشد

سناریو چهارم : استخراج مقادیر ارزش‌های برند تلویزیون و ابعاد آن

در این ربات ما می‌خواهیم مقادیر ارزش های برند تلویزیون و ابعاد آن را با هنگامی که ربات در حال صحبت با کابر است استخراج کنیم.

۱. **آماده سازی دادگان و آموزش مدل**

در این قسمت ما می‌خواهیم ربات خرید تلویزیون را طاحی کنیم. برای پیاده‌سازی این سناریو ربات ما باید ۴ اندیشه متفاوت را تشخیص دهد.

پرونده rules.yml:

اندیشه اول : ask_tv

این اندیشه برای مطرح کردن درخواست به ربات می‌باشد و در آن کاربر مکالمه را با ربات شروع کرده و درخواست خرید تلویزیون را مطرح خواهد کرد. نمونه‌های پیاده‌سازی شده این اندیشه به شکل زیر می‌باشد:

```
nlu:
- intent: ask_tv
  examples: |
    - میخوام تلویزیون بخرم
    - میخواهم تیلویزیون بخرم
    - تلویزیون میخوام بخرم
    - تلویزیون میخوام
    - یک تلویزیون میخوام
    - میخوام تلویزیون بخرم
    - لطفا یه تلویزیون به من بدید
    - سلام یک تلویزیون میخواستم
    - سلام تلویزیون دارید؟
```

```
    - می‌خواهم یک تلویزیون بخرم .
    - به دنبال خرید تلویزیونی هستم.
    - قصد دارم یک تلویزیون بخرم.
    - می‌خواهم تلویزیونی با کیفیت بالا بخرم.
    - می‌خواهم تلویزیونی با صفحه‌ی بزرگ بخرم.
    - لطفاً گزینه‌های تلویزیون را به من نشان دهید.
    - من مایل به خرید یک تلویزیون هوشمند هستم.
```

اندیشه دوم : ask_tv_detail

این اندیشه برای ارائه اطلاعات توسط کاربر و جزئیات خرید تلویزیون استفاده می‌شود و برای آن است که entity ها توسط کاربر وارد شود. در این اندیشه ما دو نوع entity وارد کردیم که entity اول که اینچ تلویزیون است را می‌توان با عبارت های منظم استخراج کرد که توسط regex entity extractor بدست می‌اید و الگوی regex ان به شرح زیر است:

```
- regex: tv_inch
  examples: |
    - \\b\d+(?:.\d+)?(?=\s*(?:اینچ|inch)\b)
```

اما entity دوم ساختار پیچیده تری دارد که توسط CRF entity extractor بدست می‌آید. حال برای تعریف کردن اسلات ها در ورودی به صورت زیر عمل می‌کنیم و هرکدام از اسلات هایی که می‌خواهیم توسط entity extractor ها بدست آیند در کروشه گذاشته و در پرانتز نام entity را می‌نویسیم. که نمونه‌های تولید شده برای این اندیشه به شرح زیر خواهند بود:

```
- intent: ask_tv_detail
  examples: |
    - [samsung](company_name) [55](tv_inch) inch میخوام
    - [LG](company_name) [22](tv_inch) inch میخواستم
    - [سونی](company_name) [69](tv_inch) inch باشه
    - [samsung](company_name) [45](tv_inch) اینچی میخوام
    - [سامسونگ](company_name) [55](tv_inch) اینچی میخواستم
    - [x-vision](company_name) [27](tv_inch) بدید لطفا
    - [ایکس ویژن](company_name) [88](tv_inch) اینچ نیاز دارم
    - [تی سی ال](company_name) [75](tv_inch) اینچ نیاز دارم.
    - [TCL](company_name) [65](tv_inch) اینچی نیاز دارم.
    - [سامسونگ](company_name) [55](tv_inch) اینچی میخواستم بخرم
    - [میخوام تلویزیون آلجی](company_name) [55](tv_inch) اینچی بخرم
    - [میخوام تلویزیون سامسونگ](company_name) [49](tv_inch) اینچی بخرم
    - میخوام تلویزیون [panasonic](company_name) [39](tv_inch) اینچ بخرم
```

اندیشه سوم : affirm

این اندیشه برای تایید کاربر استفاده می‌شود می‌شود. نمونه های این اندیشه به شرح زیر می‌باشند:

```
- intent: affirm
  examples: |
    - اره
    - عاره
    - عاااره
    - عااره
    - بله
    - درسته
    - همینه
    - درست است
    - همین است
    - ارههههه
    - بلههه
    - ارهه
    - اره
    - تایید میکنم
    - تایید می‌کنم
    - بله لطفا
    - بلههه
```

اندیشه چهارم : deny

این اندیشه برای رد درخواست توسط کابر استفاده می‌شود و نمونه های آن به شرح زیر می‌باشند:

```
- intent: deny
  examples: |
    - خیر
    - نه
    - غلطه
    - غلط است
    - درست نیست
    - این نیست
    - اشتباهه
    - اشتباه است
    - اشتباه هست
    - نههه
    - خییر
    - نهه
```

```
    - نخواستیم
    - نه این رو نمیخواستم
    - اشتباه فهمیدی
```

پرونده rules.yml:

این پرونده مانند سوال قبل تایین می‌کند هرکدام از intent ها با چه action ای پاسخ داده شوند. به این سوال ما ۴ اندیشه تعریف کردیم که برای هرکدام از آنها یک پاسخ داشتیم و در rules.yml ارتباط بین آن ها را تعیین می‌کنیم. که پیاده‌سازی آن به شکل زیر می‌باشد:

```yaml
rules:

- rule: ask tv
  steps:
  - intent: ask_tv
  - action: utter_ask_tv

- rule: ask tv detail
  steps:
  - intent: ask_tv_detail
  - action: utter_ask_tv_detail

- rule: affirm
  steps:
  - intent: affirm
  - action: utter_affirm

- rule: deny
  steps:
  - intent: deny
  - action: utter_deny
```

پرونده domain.yml:

در این پرونده همانند سوال قبل اول intent های ربات را می‌نویسیم. سپس entity های ربات را می‌نویسیم. در ادامه Slot ها را تعریف می‌کنیم و هرکدام از Slot ها را با entity های گفته شده نگاشت می کنیم. به عنوان مثال slot=company_name به entity=company_name نگاشت می شود.

```yaml
slots:
```

```
  company_name:
    type: text
    mappings:
      - type: from_entity
        entity: company_name

  tv_inch:
    type: text
    mappings:
      - type: from_entity
        entity: tv_inch
```

در ادامه response های هرکدام از intent ها را برای ربات تعریف می‌کنیم که پیاده سازی یکی از utter ها به شکل زیر می‌باشد:

```
  utter_ask_tv_detail:
  - text: "آیا میخواهید تلویزیون {company_name} {tv_inch} اینچی را به سبد خرید شما اضافه کنم؟ "
```

پرونده stories.yml:

این پرونده ما ارتباط داستانی ربات را تعریف خواهیم کرد. در این سوال ما دو مسیر داستانی خواهیم داشت. مسیر اول که کاربر در آن affirm کند و تلویزیون به سبد خرید اضافه شود و مسیر دوم که کاربر در آن deny کند و ربات از اول شروع کند. دو مسیر گفته شده در این پرونده نوشته شده‌اند

```
stories:

- story: happy path
  steps:
  - intent: ask_tv
  - action: utter_ask_tv
  - intent: ask_tv_detail
  - action: utter_ask_tv_detail
  - intent: affirm
  - action: utter_affirm

- story: sad path
  steps:
  - intent: ask_tv
  - action: utter_ask_tv
```

```
  - intent: ask_tv_detail
  - action: utter_ask_tv_detail
  - intent: deny
  - action: utter_deny
```

۲. پیاده‌سازی و تحلیل نتایج

♦ توضیحات config

در این سوال config ما به صورت زیر خواهد بود: (قسمت‌های سفید کل مانند مدل اول سوال اول می‌باشد.)

```
language: fa
pipeline:
- name: WhitespaceTokenizer
- name: RegexFeaturizer
- name: LexicalSyntacticFeaturizer
- name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: char_wb
  min_ngram: 1
  max_ngram: 4
- name: LanguageModelFeaturizer
  model_name: bert
  model_weights: rasa/LaBSE
  constrain_similarities: true
- name: RegexEntityExtractor
- name: "CRFEntityExtractor"
- name: DIETClassifier
  epochs: 100
  constrain_similarities: true
- name: FallbackClassifier
  threshold: 0.3
  ambiguity_threshold: 0.1
```

- **RegexEntityExtractor:**

یک entity extractor می‌باشد که با استفاده از عبارت‌های منظم entity ها را از متن استخراج می‌کند که می‌تواند برای شناسایی pattern ها در متن و entity هایی که ساختار آن ها را می‌دانیم کاربرد داشته باشد. همچنین از آنجایی که عبارت use_regexes

به صورت default برابر با true می باشد دیگر آن را اضافه نکردیم. در این سوال برای استخراج اینچ که آن را به صورت الگو regex ساختیم از این extractor استفاده می‌کنیم.

- **CRFEntityExtractor**

یک entity extractor برای استخراج موجودیت های سخت تری که توسط regexEntityExtractor در متن شناسایی نشدند کاربرد دارد. همانطور که اشاره شد، CRF برای استخراج موجودیت‌های پیچیده تر استفاده می شود و با استفاده از markov chain، لیبل‌های موجودیت‌ها را تعیین می‌کند.

- **FallbackClassifier**

از این کلاسیفایر استفاده کردیم تا مدل زمانی که نسبت به پاسخ نهایی مطمئن نیست، اظهار نظری نکند. همچنین ambihuity_threshold را برابر ۰.۱ وthreshold بر برابر با ۰.۳ قرار دادیم.

بقیه کانفیگ، مانند کانفیگ مدل اول می‌باشد که توضیحات آن در همان قسمت آمده است. همچنین در این قسمت نیز مانند قسمت قبل از policy های پیشفرض استفاده خواهیم کرد.

♦ استخراج ارزش‌ها با استفاده از entity extractor ها:

در سناریو چهارم می خواهیم مقادیر ارزش برند تلویزیون و سایز(اینچ) تلویزیون را انتخاب کنیم. برای بیان اینچ تلویزیون می توانیم یک عبارت منظم بنویسیم و سایز تلویزیون را با آن بیان کنیم. به همین علت از regex entity extractor برای استخراج اینچ تلویزیون در در فایل nly.yml برای آن الگوی regex تعریف کردیم استفاده می‌کنیم. اما برای استخراج ارزش company_name با توجه به آنکه ساختار منظمی ندارد نمی‌توانیم عبارت منظم تعریف کنیم همچنین نام‌های شرکت های تلویزیون سازی می‌تواند بسیار گسترده باشد، پس برای company_name از crf entity extractor استفاده می‌کنیم.

♦ ابزار گفت‌وگوی تحت وب

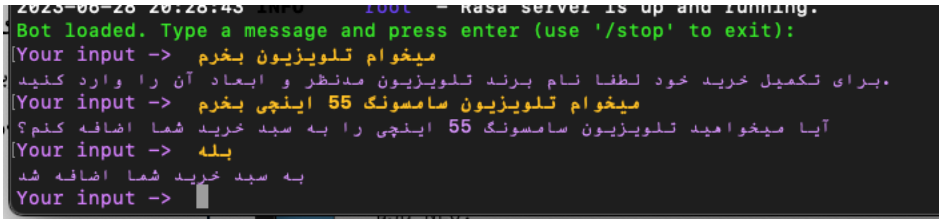
در این مرحله با استفاده از پوشه WebChat قرار داده شده در تمرین، یک رابط تصویری تحت وب برای ربات خود بالا آوردیم. برای استفاده از این رابط کافی است کد زیر را اجرا کنیم:

```
rasa run -m models --enable-api --cors "*" –debug
```

سپس وارد پوشه WebChat شده و index.html را باز می کنیم و نتیجه به صورت زیر می باشد.

شکل ۱۱ : گفتگو با ربات خرید تلویزیون در محیط تحت وب

همچنین به کمک دستور rasa shell:



شکل ۱۲ : گفتگو با ربات خرید تلویزیون در shell

♦ ارزیابی ربات

در این قسمت می‌خ.اهیم با استفاده از روش k-fold مدل را ارزیابی کنیم. سپس نتایج مربوط به اسخراج ارزش‌ها و همچنین طبقه بندی intent ها را گزارش کنیم.

برای ارزیابی ربات با روش k-fold از دستور زیر استفاده می کنیم:

```
rasa test nlu --nlu data/nlu --cross-validation --folds 3
```

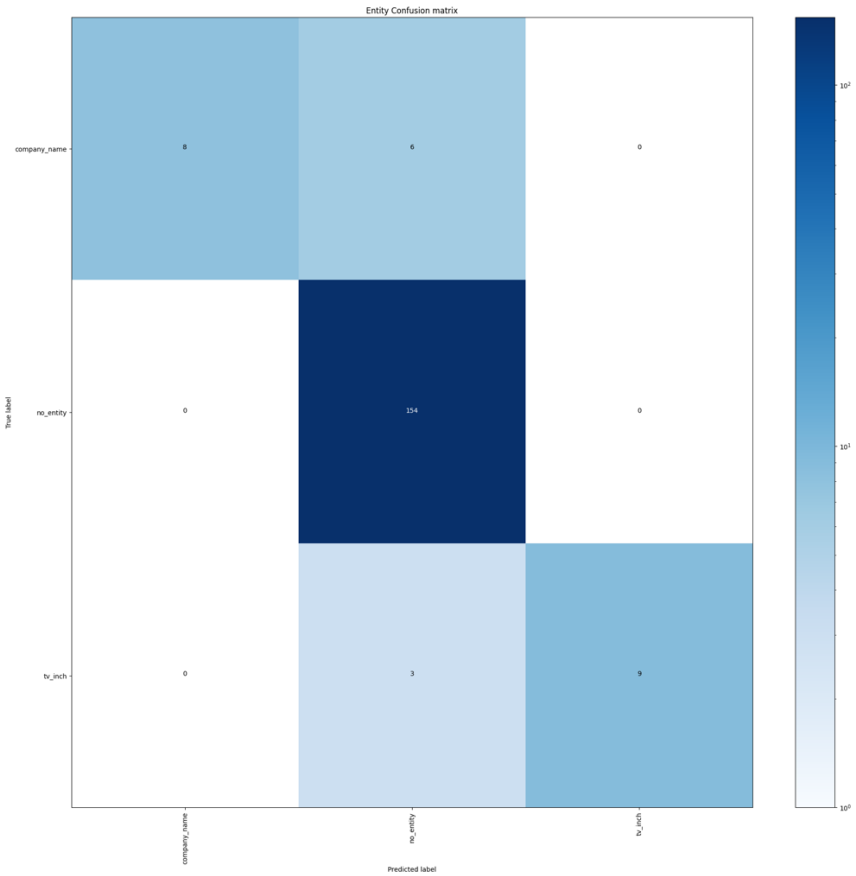
نتایج بدست آمده به شرح زیر می‌باشد:

نتایج استخراج ارزش‌ها:

در اینجا مقادیر دقت و f1-score را برای کل ارزش‌ها گزارش می‌کنیم.

Accuracy: 0.951

F1-score: 0.776



شکل ۱۳ : ماتریش اشفتگی برای استخراج ارزش‌ها

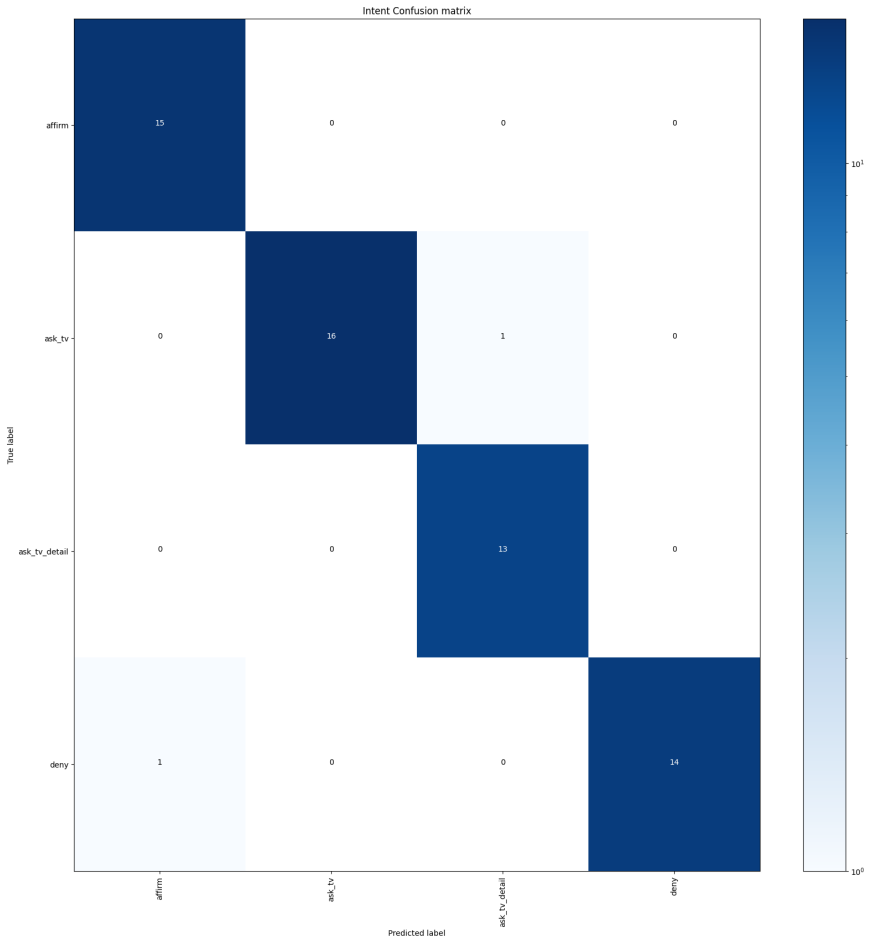
نتایج طبقه‌بندی intent ها :

در اینجا مقادیر دقت و f1-score را برای کل intent ها گزارش می‌کنیم.

Accuracy: 0.982

F1-score: 0.982

Intent	Precision	F1-Score
deny	1.00	0.9655
affirm	0.9375	0.9677
ask_tv_detail	0.9286	0.9630
ask_tv	1.000	0.9697



شکل ۱۵ : ماتریش اشفتگی برای استخراج intent ها

♦ مقایسه entity extractor ها

همانطور که در قسمت قبل گفته شد، در استخراج مقادیر ارزش‌ها، همگامی که با ارزش‌هایی سروکار داریم که با عبارات منظم قابل بیان کردن هستند و یا می‌توانیم الگوی regex برای آن‌ها بنویسیم و یا مقادیر خاصی می‌گیرند، می‌توانیم از regex entity extractor استفاده کنیم. بنابراین همگامی که الگو های استخراج مقادیر ارزش‌ها ثابت و یا ساده باشد

اما عملکرد CRF entity extractor به این صورت است که از الگوریتم hidden markov برای شناسایی و استخراج مقادیر ارزش استفاده می‌کند که به وسیله آن می‌تواند مقادیر ارزش‌ها را شناسایی و استخراج کند. CRF قادر به تشخیص الگوهای پیچیده‌تر و متخیرتر نسبت به regex entity می‌باشد. همینطور با در شرایط وجود ویژگی‌های متنی متفاوت، می‌تواند مقادیر ارزش‌ها را با دقت بالا با توجه به متن استخراج کند.

به عنوان مثال اگر سوال خودمان را در نظر بگیریم، همانطور که از ماتریس آشفتگی معلوم است، شناسایی سایز تلویزیون بسیار دقت بالاتری دارد تا شرکت سازنده تلویزیون. دلیل آن هم واضح است زیرا فرآیند استخراج entity توسط crf بسیار سخت تر می‌باشد و به همین دلیل دقت آن در مقایسه با entity که برای آن عبارت منظم طراحی کردیم پایین تر می‌باشد.