به نام آنکه آموخت انسان را آنچه نمی دانست

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

درس پردازش زبان‌های طبیعی

CA4

POS Tagging

behzad.shayegh@ut.ac.ur

بهزاد شایق بروجنی

81 01 96 678

فروردین ۱۳۹۹

فهرست

فهرست **1**

مقدمه **2**

دادگان **2**

بخش اول) RNN **2**

پیاده سازی مدل 2

استفاده از دادگان آموزشی 3

ارزیابی 4

بخش دوم) HMM (دو مدل: ۱. مدل nltk و ۲. مدل دست ساز) **5**

پیاده سازی مدل 5

استفاده از دادگان آموزشی 5

ارزیابی 5

فایل‌های جانبی **8**

مقدمه

با سلام. در این تمرین قصد داریم با استفاده از روش‌های MHH و RNN به برچسب زنی اجزای سخن بپردازیم.

دادگان

دادگان مورد استفاده در این آزمایش، یک داده‌ی Train و یک داده‌ی Test برچسب‌دار با مجموعه‌ی برچسب‌های ۴۲ عضوی به ازای هر یک از مدل‌های MHH و RNN می‌باشد. در دادگان تست مدل RNN سطرهایی خالی وجود داشت که آن‌ها از دادگان حذف شدند و در خروجی، جداول تمیز داده شده‌اند. همچنین در قسمت‌هایی که جداول Header نداشتند، این سطر اضافه شد.

بخش اول) RNN

پیاده سازی مدل

برای پیاده‌سازی این مدل، دو تلاش ناموفق با استفاده از کتابخانه‌های pytorch و keras داشتیم که کد مربوط به آن‌ها نیز ضمیمه شده است. در نهایت، مدل را با استفاده از کتابخانه‌ی keras و بصورت Bidirectional پیاده‌سازی کردیم. اطلاعات مدل نهایی مورد استفاده در ادامه آمده است :

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 104, 128)          2704768
_________________________________________________________________
bidirectional_1 (Bidirection (None, 104, 512)          788480
_________________________________________________________________
time_distributed_1 (TimeDist (None, 104, 43)           22059
_________________________________________________________________
activation_1 (Activation)    (None, 104, 43)           0
=================================================================
Total params: 3,515,307
Trainable params: 3,515,307
Non-trainable params: 0
_________________________________________________________________
```

همانطور که در گزارش بالا مشاهده می‌کنید، برای مدل یک لایه embedding لغات نیز درنظر گرفتیم. اندازه‌ی ۱۰۴ که برای شبکه درنظر گرفته شده است، طول طولانی‌ترین جمله‌ی موجود در دادگان است که در دادگان تست مشاهده شده است.

دیگر اعداد نیز با آزمون و خطا بدست آمده‌اند (البته پس از چند آزمون ناموفق، در نهایت از پارامترهای استفاده شده در وبسایت https://nlpforhackers.io/lstm-pos-tagger-keras/ استفاده شد).

استفاده از دادگان آموزشی

در آموزش شبکه، تفاوت در طول جملات آزاردهنده است. به همین دلیل طول تمام جملات را با استفاده از padding به اندازه طول بزرگ‌ترین جمله موجود افزایش دادیم. همچنین با توجه به اینکه شبکه با اعداد کار می‌کند، به هر لغت یک عدد یکتا نسبت دادیم (عدد padding را برابر صفر درنظر گرفتیم) و از این اعداد استفاده کردیم. همچنین با توجه با اینکه خروجی شبکه، برداری از احتمالات است، برای اینکه شبکه بتواند Loss را (به صورت برداری) محاسبه کند، برچسب‌ها را به صورت بردارهای one-hot به شبکه دادیم. با نرخ ارزیابی ۲۰ درصد، ۱۰ مرتبه‌ی آموزشی را طی کردیم و نتایج به شکل زیر بود :

```
Train on 16000 samples, validate on 4000 samples
Epoch 1/10
16000/16000 [==============================] - 147s 9ms/step - loss: 0.7747 -
accuracy: 0.8196 - val_loss: 0.5339 - val_accuracy: 0.8619
Epoch 2/10
16000/16000 [==============================] - 146s 9ms/step - loss: 0.4034 -
accuracy: 0.8988 - val_loss: 0.2409 - val_accuracy: 0.9369
Epoch 3/10
16000/16000 [==============================] - 144s 9ms/step - loss: 0.1487 -
accuracy: 0.9626 - val_loss: 0.0897 - val_accuracy: 0.9780
Epoch 4/10
16000/16000 [==============================] - 151s 9ms/step - loss: 0.0641 -
accuracy: 0.9841 - val_loss: 0.0570 - val_accuracy: 0.9850
Epoch 5/10
16000/16000 [==============================] - 146s 9ms/step - loss: 0.0439 -
accuracy: 0.9882 - val_loss: 0.0491 - val_accuracy: 0.9859
Epoch 6/10
16000/16000 [==============================] - 147s 9ms/step - loss: 0.0361 -
accuracy: 0.9899 - val_loss: 0.0430 - val_accuracy: 0.9875
Epoch 7/10
16000/16000 [==============================] - 145s 9ms/step - loss: 0.0318 -
accuracy: 0.9908 - val_loss: 0.0411 - val_accuracy: 0.9878
Epoch 8/10
16000/16000 [==============================] - 152s 10ms/step - loss: 0.0290 -
accuracy: 0.9913 - val_loss: 0.0397 - val_accuracy: 0.9880
Epoch 9/10
16000/16000 [==============================] - 145s 9ms/step - loss: 0.0268 -
accuracy: 0.9919 - val_loss: 0.0385 - val_accuracy: 0.9883
Epoch 10/10
16000/16000 [==============================] - 146s 9ms/step - loss: 0.0250 -
accuracy: 0.9924 - val_loss: 0.0383 - val_accuracy: 0.9884
```

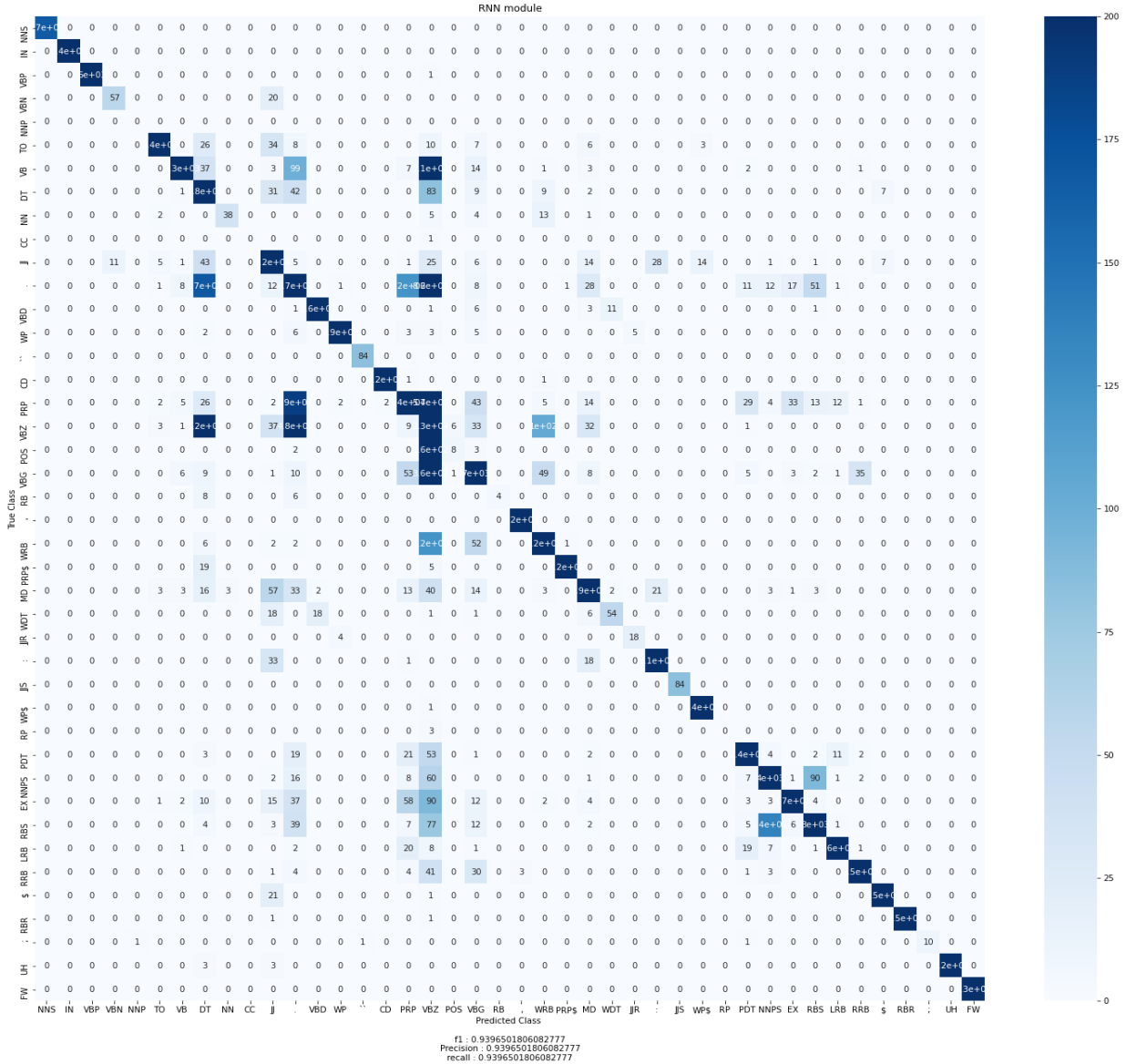
پس Accuracy بر روی دادگان آموزشی به ۹۸.۸۴٪ رسید. توجه کنید که این بازدهی با درنظر گرفتن اعضای padding بوده و خالص نیست.

ارزیابی

با اجرای مدل بر روی دادگان تست و تبدیل احتمالات به برچسب‌های متناظر با گزینه‌ی محتمل‌تر (argmax)، بازدهی Accuracy بر روی دادگان تست به شرح زیر بود :

```
Accuracy = 0.9396501806082777
```

همچنین، در این دادگان، معیارهای f1, precision و recall و همچنین confusion matrix به شرح زیر بود این جدول به صورت شفاف‌تر در فایل RNN_CONMATRX.png ضمیمه شده است) :




```
f1 : 0.9396501806082777
Precision : 0.9396501806082777
recall : 0.9396501806082777
```

برچسب‌های حاصل از این پیشبینی نیز در فایل RNN_result.csv (در ستون Predict) ضمیمه شده است.

بخش دوم) HMM (دو مدل: ۱. مدل nltk و ۲. مدل دست ساز)

پیاده سازی مدل

برای پیاده سازی این مدل، یکبار از کتابخانه‌ی `nltk` و ماژول `HiddenMarkovModelTagger` استفاده و بار دیگر، این مدل را از پایه پیاده‌سازی کردیم. در پیاده‌سازی این مدل از الگوریتم viterbi و laplace smoothing استفاده کردیم. برای نگهداری ماتریس‌ها نیز از کتابخانه‌ی pandas استفاده کردیم.

استفاده از دادگان آموزشی

برای آموزش مدل کتابخانه‌ی nltk، نیاز به یک تغییر فرمت در دادگان داشتیم چرا که nltk ورودی را به صورت ngram دریافت می‌کند، پس جملات را به شکل لیستی از جفت‌های لغت و برچسب به مدل ورودی دادیم. از آنجایی که مدل به صورت احتمالاتی عمل می‌کند، نیازی به گام آموزش نبود. بازدهی مدل بر روی دادگان آموزشی به شرح زیر بود :

```
Train accuracy over 87151 tokens: 97.07
```

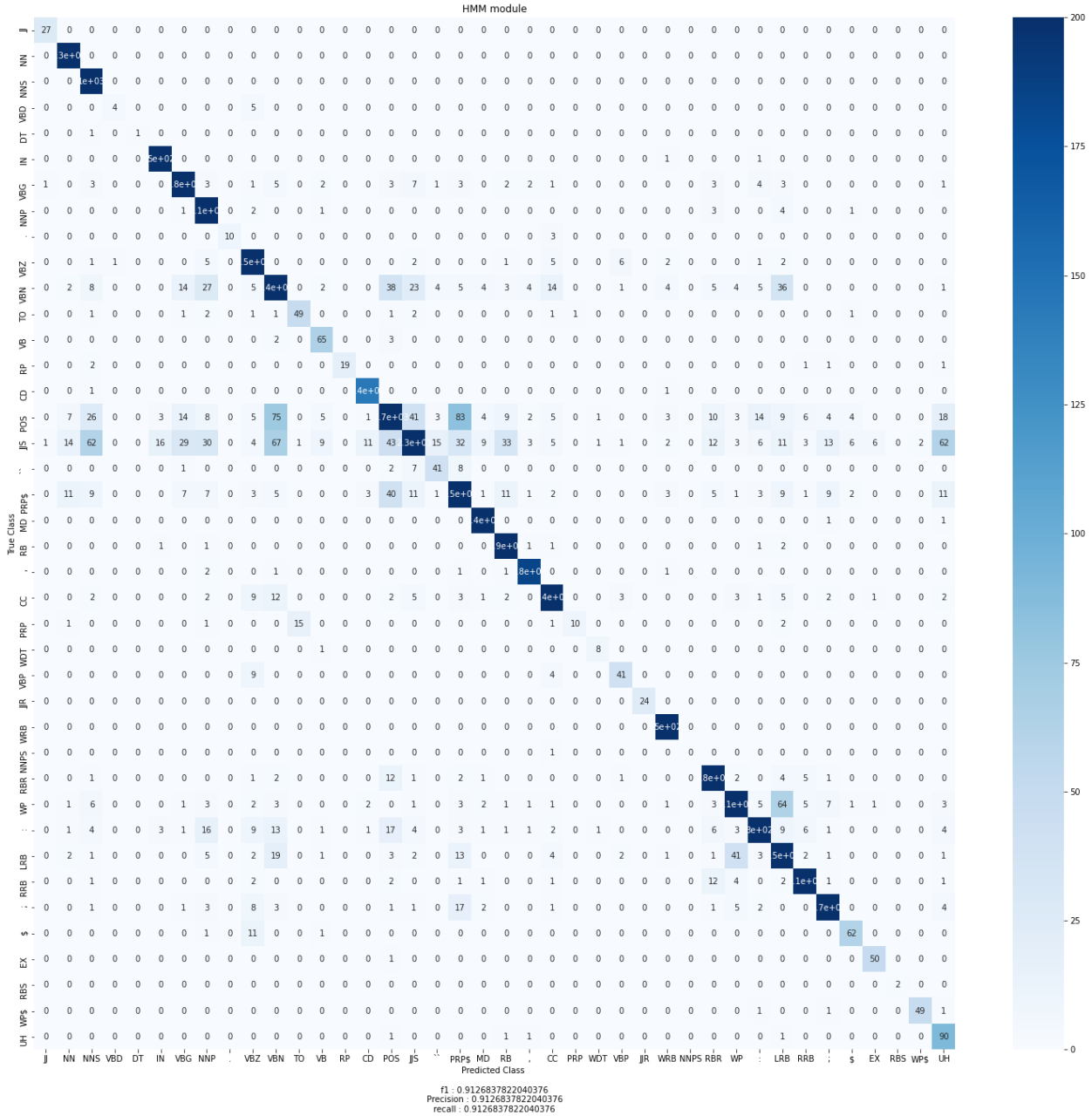
مدلی که خودمان پیاده‌سازی کردیم اما در فرمت پانداس عمل می‌کند و نیازی به تغییر دادگان نداشت.

ارزیابی

با اجرای مدل کتابخانه‌ی nltk بر روی دادگان تست، بازدهی Accuracy بر روی دادگان تست به شرح زیر بود :

```
Accuracy = 0.9126837822040376
```

همچنین، در این دادگان، معیارهای f1, precision و recall و همچنین `confusion matrix` به شرح زیر بود این جدول به صورت شفاف‌تر در فایل HMM_CONMATRX.png ضمیمه شده است) :



f1 : 0.9126837822040376

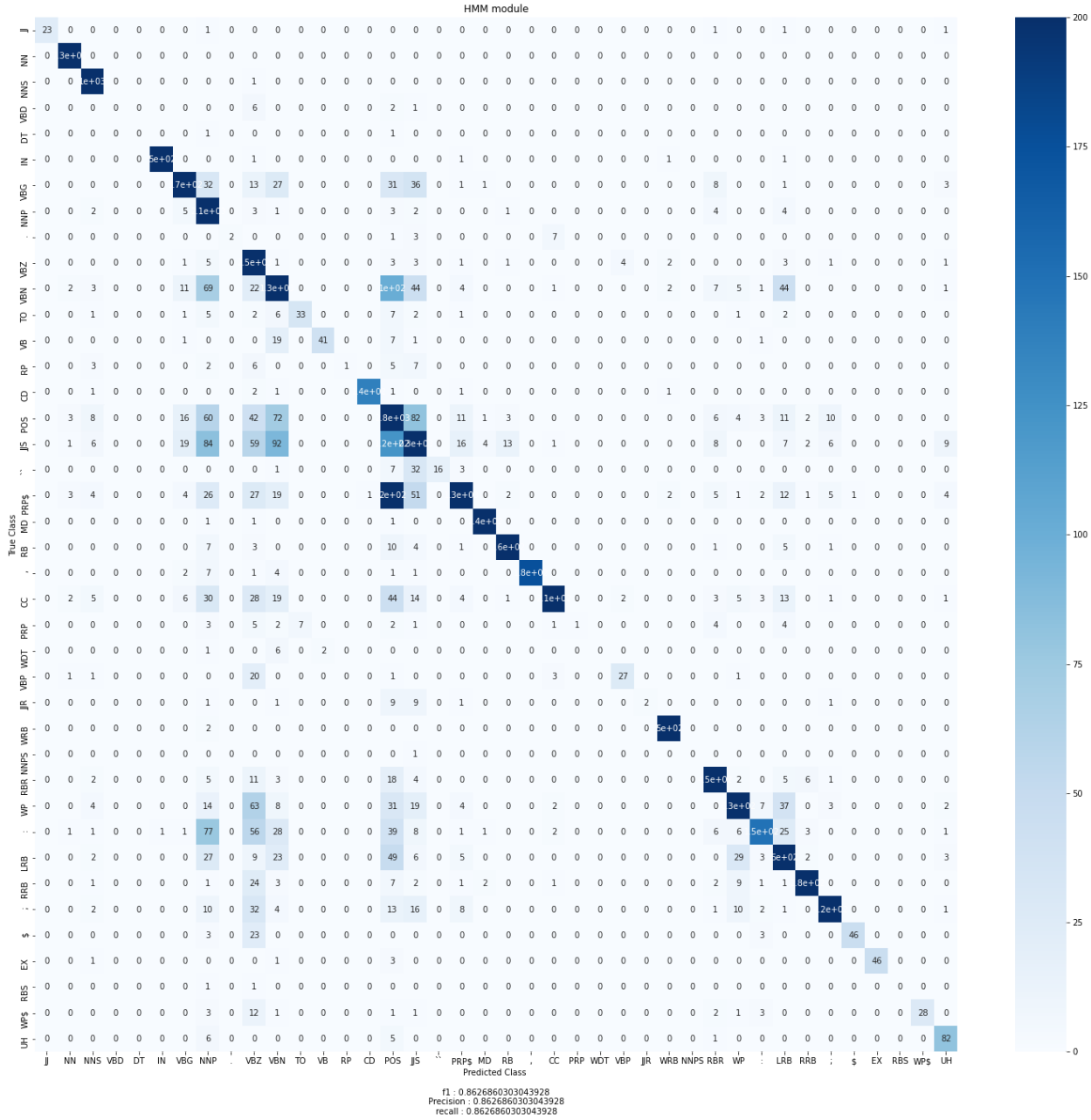
Precision : 0.9126837822040376

recall : 0.9126837822040376

همانطور که مشاهده می‌کنید، با وجود بازدهی پایین‌تر نسبت به RNN، پراکندگی ماتریس به‌هم‌ریختگی در این مدل کم‌تر است! برچسب‌های حاصل از این پیشبینی نیز در فایل HMM_result.csv (در ستون predict) ضمیمه شده است.

بازدهی Accuracy برای مدلی که خودمان پیاده سازی کردیم اما به شرح زیر بود :

همچنین، در این دادگان، معیارهای f1, precision و recall و همچنین `confusion matrix` به شرح زیر بود این جدول به صورت شفاف‌تر در فایل HMM2_CONMATRX.png ضمیمه شده است) :




```
f1 : 0.8626860303043928
Precision : 0.8626860303043928
recall : 0.8626860303043928
```

همانطور که مشاهده می‌شود، عملکرد این مدل از مدل آماده‌ی کتابخانه‌ی nltk ضعیف‌تر است که احتمالا به دلیل روش ضعیف smoothing ما (laplace smoothing) می‌باشد. توجه کنید که این پیاده سازی از صفر تا صد بدون الگو برداری و صرفا از روی اطلاعات تئوری بوده است و عملکرد نسبتا ضعیف آن می‌تواند به دلیل عدم بهینه‌سازی باشد.

فایل‌های جانبی

به همراه این گزارش، یک پوشه به نام Codes ارائه می‌شود که حاوی فایل‌های زیر است :

RNN_CONMATRX.png : تصویر ماتریس به‌هم‌ریختگی مدل RNN.

HMM_CONMATRX.png : تصویر ماتریس به‌هم‌ریختگی مدل HMM.

HMM2_CONMATRX.png : تصویر ماتریس به‌هم‌ریختگی مدل HMM.

RNN.ipynb : فایل ژوپیتر کد پروژه که شامل بخش RNN می‌باشد.

HMM.ipynb : فایل ژوپیتر کد پروژه که شامل بخش HMM با پیاده سازی nltk می‌باشد.

HMM2.ipynb : فایل ژوپیتر کد پروژه که شامل بخش HMM با پیاده سازی خودمان می‌باشد.

RNN.html : نسخه قابل رویت فایل RNN.ipynb.

HMM.html : نسخه قابل رویت فایل HMM.ipynb.

HMM2.html : نسخه قابل رویت فایل HMM2.ipynb.

RNN_result.csv : خروجی بخش RNN.

HMM_result.csv : خروجی بخش HMM.

HMM2_result.csv : خروجی بخش HMM2.

فایل‌های تلاش‌های ناموفق: در پوشه‌ی Codes، پوشه‌ای به نام failed حاوی کدهای ناموفق وجود دارد :

NLP_CA_4_RNN_Part1_Pytorch_Failed.ipynb: که تلاش ناموفق پیاده سازی RNN با استفاده از کتابخانه‌ی pytorch است.

NLP_CA_4_RNN_Part2_Keras_failed.ipynb: که تلاش ناموفق پیاده سازی RNN با استفاده از کتابخانه‌ی keras (تلاش اول) است.

NLP-CA#4-RNN-Part0.ipynb: پردازش دادگان برای استفاده در NLP_CA_4_RNN_Part1_Pytorch_Failed.ipynb.