به نام خدا

کارگاه هضم NLP

بهاره کاوسی نژاد – 99431217

سوالات تئوری

1. روند شروع به کار یک محصول (Automatic Speech Recognition) ASR و توسعه آن را با توجه به نکات گفته شده سر کارگاه، توضیح دهید.

محصولی مانند حرف، ابتدا با کمک مدل هایی مثل google و mozilla کار خود را آغاز کرد. پس از تبدیل شدن به یک مدل، با اضافه کردن دیتاهایی که متن آنها نیز در دسترس بود مانند پادکست ها و کتاب های صوتی، این مدل بهبود یافت و به دلیل تمرکز روی زبان فارسی حتی از مدل های google و mozilla نیز بهتر شد و اکنون دارای درصد دقت بالاتری است. با این وجود همچنان در برخی از موارد از این مدل ها استفاده می شود.

به طور کلی این روند را می توان اینگونه شرح داد:

1. جمع آوری داده ها: اولین گام جمع آوری مجموعه داده بزرگی از speech recordings است که برای آموزش سیستم ASR استفاده می شود. این مجموعه داده باید متنوع باشد و زبان‌ها، لهجه‌ها و سبک‌های گفتاری مختلف را پوشش دهد تا از robustness سیستم اطمینان حاصل شود.
2. آماده‌سازی داده‌ها: speech dataهای جمع‌آوری‌شده باید از قبل پردازش شوند تا هرگونه نویز ناخواسته حذف شود، سطوح صوتی عادی شود و ضبط‌ها به واحدهای کوچک‌تر مانند جملات یا عبارات تقسیم شوند. Text transcriptهای مربوط به هر بخش صوتی نیز ایجاد می شود.
3. استخراج ویژگی یا feature extraction: برای تبدیل سیگنال گفتار به قالبی مناسب برای تجزیه و تحلیل، از تکنیک های استخراج ویژگی مانند Mel-frequency cepstral coefficients (MFCCs) استفاده می شود. این تکنیک‌ها شکل موج صوتی خام را به دنباله‌ای از بردارهای ویژگی صوتی تبدیل می‌کنند و اطلاعات مربوطه را در مورد سیگنال گفتار می‌گیرند.
4. آموزش مدل آکوستیک: مدل آکوستیک سیستم ASR با استفاده از داده های گفتاری از پیش پردازش شده و transcript های متناظر آنها آموزش داده می شود. این شامل استفاده از الگوریتم‌های یادگیری ماشینی، مانند Hidden Markov Models (HMMs) یا شبکه‌های عصبی عمیق (DNN) برای یادگیری روابط بین ویژگی‌های صوتی و واحدهای آوایی یا واحدهای فرعی مربوطه است.
5. آموزش مدل زبان: علاوه بر مدل آکوستیک، سیستم ASR به یک مدل زبان برای درک زمینه و بهبود دقت تشخیص نیاز دارد. مدل‌های زبانی معمولاً با استفاده از مجموعه‌های متنی بزرگ مانند مقالات خبری، کتاب‌ها یا صفحات وب آموزش داده می‌شوند تا احتمالات توالی کلمات را بیاموزند و توانایی سیستم را برای پیش‌بینی محتمل‌ترین توالی کلمات با توجه به ورودی صوتی بهبود بخشند.

6. رمزگشایی: در طول فرآیند رمزگشایی، سیستم ASR ویژگی های صوتی یک گفتار را می گیرد و آنها را با مدل های آموزش دیده مطابقت می دهد. این شامل جستجو در میان تمام توالی کلمات ممکن و یافتن محتمل ترین دنباله ای است که با ویژگی های ورودی مطابقت دارد. Beam search یا سایر الگوریتم‌های رمزگشایی برای کاوش کارآمد فضای جستجو و یافتن بهترین دنباله منطبق استفاده می‌شود.
7. ارزیابی و تکرار: سیستم ASR اولیه با استفاده از یک مجموعه داده جداگانه از ضبط گفتار با transcript های شناخته شده ارزیابی می شود. معیارهای ارزیابی شامل Word Error Rate(WER) یا سایر معیارهایی است که دقت رونویسی های سیستم را کمیت می کند. بر اساس نتایج ارزیابی، بهبودهایی در سیستم با اصلاح مدل‌ها، تنظیم پارامترها یا جمع‌آوری داده‌های بیشتر انجام می‌شود. این روند تکراری تا رسیدن به سطح مورد نظر از دقت و عملکرد ادامه می یابد.
8. deployment: پس از آموزش و تنظیم دقیق سیستمASR ، آماده deploy شدن است. می توان آن را در برنامه های مختلف مانند دستیارهای صوتی، خدمات transcription یا تجزیه و تحلیل گفتار مرکز تماس ادغام کرد. سیستم مستقر شده همچنان به یادگیری و بهبود در طول زمان ادامه می دهد زیرا تعاملات و داده های بیشتری با کاربر در دسترس قرار می گیرد.

2. فرض کنید چند مدل (برای مثال harf – whisper – mozilla – google – nemo – ...) برای تبدیل صوت به متن فارسی در دسترس داریم و هرکدام 16 خروجی با ترتیب اولویت احتمال درستی میدهد، برای اینکه از بین این تعداد زیاد خروجی، بتوانیم تعداد 16 خروجی نهایی انتخاب کنیم و خروجی دهیم حداقل دو روش (که در عمل استفاده می شود یا اگر روش جدیدی به ذهنتان میرسد که منطقی است توضیح دهید).

پس از خروجی گرفتن از مدل ها به آنها ضریبی را نسبت می دهیم. به عنوان مثال تعداد تکرار یک کلمه در هر 16 خروجی هر مدل را بدست می آوریم. اگر تعداد تکرار کلمه ای زیاد بود، احتمال وجود آن در متن خروجی بیشتر است، در نتیجه به آن وزن بیشتری اختصاص می دهیم. همچنین اگر کلمه ای جزء خروجی های اول مدل ها بود، وزن آن خیلی بیشتر خواهد بود.

بعد از بدست آمدن وزن کلمات، ماتریسی ساخته می شود که بیانگر تعداد تکرار هر کلمه و اهمیت آن است.

روش دیگر استفاده از Deep Learning برای ترکیب کردن مدل ها می باشد.

همچنین برای ارزیابی مدل های ASR می توان سنجش را بر اساس کلمه و یا بر اساس کاراکتر انجام داد.

همچنین یک Language Model داریم که احتمالات را پیش بینی می کند.

3. مفهوم توابع normalizer، formalizer، lemmatizer، stemmer، chunker، tagger، postagger، embedder، wordembedder و parser را در دنیای پردازش متن توضیح دهید.

- normalizer: normalizer در NLP به تابع یا فرآیندی اشاره دارد که متن را به فرم استاندارد یا نرمال شده تبدیل می کند. معمولاً شامل تبدیل متن با حذف علائم نگارشی، تبدیل به حروف کوچک، حذف

لهجه‌ها، گسترش contractionها و مدیریت سایر قراردادهای خاص زبان است. هدف از نرمال سازی ایجاد یک نمایش ثابت از متن است که تجزیه و تحلیل و پردازش را آسان تر می کند.

- formalizer: formalizer تابعی است که در NLP برای تبدیل متن غیررسمی یا محاوره‌ای به فرمی رسمی یا استاندارد استفاده می‌شود. ممکن است شامل جایگزینی کلمات عامیانه با معادل های رسمی آنها، تصحیح اشتباهات املایی و بهبود ساختارهای دستوری برای رعایت قوانین زبان رسمی باشد. هدف formalizer افزایش خوانایی، اطمینان از وضوح و حفظ ثبات در متن است.
- lemmatizer: lemmatizer تابعی است که کلمات را به شکل پایه یا متعارف خود کاهش می دهد که به آن لم می گویند. برای استخراج لم یک کلمه، بافت و بخشی از گفتار آن را در نظر می گیرد. برای مثال، لم «running»، «run» و لم «better»، «good» خواهد بود. Lemmatization به کاهش inflected fromهای کلمات به شکل ریشه مشترک آنها کمک می کند و به کارهایی مانند تجزیه و تحلیل متن، بازیابی اطلاعات و یادگیری ماشین کمک می کند.
- Stemmer: یک stemmer تابعی است که کلمات را به شکل پایه یا ریشه خود کاهش می دهد که به عنوان stem شناخته می شود. بر خلاف lemmatizerها، stemmerها بافت یا بخشی از گفتار یک کلمه را در نظر نمی گیرند. آنها قواعد ساده زبانی را برای حذف پیشوندها یا پسوندها از کلمات اعمال می کنند، که اغلب منجر به stem هایی می شود که کلمات واقعی فرهنگ لغت نیستند. به عنوان مثال، stem "running"، "run" و stem "better" "bet" خواهد بود. Stemming معمولاً برای normalize کردن متن برای کارهایی مانند indexing، جستجو و بازیابی اطلاعات استفاده می شود.
- Chunker: یک chunker تابعی است که کلمات را بر اساس ساختار نحوی آنها در واحدهای معنی دار به نام chunk گروه بندی می کند. Chunk ها می توانند از ترکیب های مختلفی از کلمات مانند noun phrases، verb phrases یا prepositional phrases تشکیل شوند. Chunking معمولاً برای استخراج اطلاعات خاص از متن استفاده می شود، مانند named entities یا key phrases.
- tagger: یک tagger، تابعی است که تگ های دستوری یا Part Of Speech (POS) را به هر کلمه در یک جمله اختصاص می دهد. تگ های POS نشان دهنده نقش و دسته بندی یک کلمه در یک جمله هستند، مانند اسم، فعل، صفت یا قید. برچسب‌گذاری یک مرحله ضروری در بسیاری از وظایف NLP از جمله تجزیه، تحلیل احساسات و استخراج اطلاعات است.
- postagger: postagger که مخفف عبارت part-of-speech tagger است، نوع خاصی از برچسب گذاری است که بر اختصاص تگ های POS به کلمات در یک جمله تمرکز دارد. ساختار نحوی یک جمله را تجزیه و تحلیل می کند و هر کلمه را با برچسب POS مربوطه آن برچسب گذاری می کند.
- embedder: یک تابع یا الگوریتمی است که کلمات یا جملات را به نمایش های dense و numerical تبدیل می کند که به آن word embeddings یا sentence embeddings می گویند. Enbeddingها

اطلاعات معنایی و متنی کلمات یا جملات را دریافت می‌کنند و ماشین‌ها را قادر می‌سازند تا زبان طبیعی را درک و پردازش کنند. آنها به طور گسترده در وظایف مختلف NLP مانند تجزیه و تحلیل احساسات، ترجمه ماشینی و طبقه بندی اسناد استفاده می شوند.

- wordembedder: wordembedder نوع خاصی از embedder است که word embeddingها را تولید می کند. بر اساس روابط معنایی بین کلمات، کلمات جداگانه را به بردارهای dense و numerical در فضایی با ابعاد بالا نگاشت می کند. این تعبیه‌ها را می‌توان برای اندازه‌گیری شباهت کلمات، انجام word analogies یا به عنوان ویژگی‌های ورودی برای وظایف NLP که downstream هستند، استفاده کرد.
- parser: parser تابع یا الگوریتمی است که ساختار دستوری یک جمله را تجزیه و تحلیل می کند و به هر کلمه برچسب های نحوی مانند فاعل، مفعول یا فعل اختصاص می دهد. هدف آن درک روابط بین کلمات و چگونگی تشکیل یک جمله معنادار است. Parsing برای کارهایی مانند درک جمله، پاسخ به سؤال و تجزیه و تحلیل نحوی بسیار مهم است.

4. ویدیوی Turing را به مدل حرف دادیم و این خروجی را به ما داد. فایل زیرنویس منتشر شده از سمت خود سازنده ویدیو را که در این لینک قابل دانلود است، با آن تطابق دهید و پس از نرمالسازی، CER (Character Error Rate) و WER (Word Error Rate) خروجی مدل را با استفاده از کتابخانه هضم و لینک 1 و لینک 2 حساب کنید.

کتابخانه hazm را با استفاده از دستور زیر نصب کرده و فایل ها را باز کرده و نمایش می دهیم:

```
!pip install hazm
# Using the with statement (recommended)
with open("HarfOutp", "r") as file_object:
    Harf_Output = file_object.read()
Harf_Output
```

```
with open("Original.txt", "r") as file_object:
    Original_Output = file_object.read()
Original_Output
```

نرمالسازی را با استفاده از تابع Normalizer موجود در کتابخانه hazm انجام می دهیم و نتایج را نمایش می دهیم:

```
from hazm import Normalizer
normalizer = Normalizer()
Harf_Output_Normalized = normalizer.normalize(Harf_Output)
Harf_Output_Normalized
```

```
normalizer = Normalizer()
Original_Output_Normalized = normalizer.normalize(Original_Output)
Original_Output_Normalized
```

مقادیر WER و CER را با استفاده از کتابخانه evaluate محاسبه می کنیم:

نصب کتابخانه های لازم:

```
!pip install evaluate
!pip install jiwer --quiet
```

پیش از دادن رشته ها به محاسبه گر، طول دو رشته ورودی را یکسان می کنیم:

```
from evaluate import load
from hazm import word_tokenize
wer = load("wer")
# Ensure that the number of predictions and references are equal
# Here we are taking the minimum length to ensure both lists are of the same
length
# Split the strings into lists of words
Harf_Output_Normalized_words = word_tokenize(Harf_Output_Normalized)
Original_Output_Normalized_words = word_tokenize(Original_Output_Normalized)
wer_score =
wer.compute(predictions=Harf_Output_Normalized_words[:min(len(Harf_Output_Normali
zed_words), len(Original_Output_Normalized_words))],
                        references=Original_Output_Normalized_words[:min(len(Harf
_Output_Normalized_words), len(Original_Output_Normalized_words))])
```

مقدار wer:

```
0.9933010792705626
```

برای cer نیز داریم:

```
from evaluate import load
from hazm import word_tokenize
cer = load("cer")
# Ensure that the number of predictions and references are equal
# Here we are taking the minimum length to ensure both lists are of the same
length
# Split the strings into lists of words
Harf_Output_Normalized_words = word_tokenize(Harf_Output_Normalized)
Original_Output_Normalized_words = word_tokenize(Original_Output_Normalized)
cer_score =
cer.compute(predictions=Harf_Output_Normalized_words[:min(len(Harf_Output_Normali
zed_words), len(Original_Output_Normalized_words))],
```

```
                        references=Original_Output_Normalized_words[:min(len(Harf
_Output_Normalized_words), len(Original_Output_Normalized_words))])
```

مقدار cer:

```
1.128
```

5. تعداد فعل ها و قیدهای موجود در دوفایل مورد بحث در سوال قبل را با استفاده از کتابخانه هضم محاسبه کنید.

با استفاده از مدل pos_tagger نقش هر کلمه در متن را مشخص می کنیم و نتایج را نمایش می دهیم:

برای نتایج حرف داریم:

```
from hazm import POSTagger
tagger = POSTagger(model='pos_tagger.model')
Harf_Tags = tagger.tag(word_tokenize(Harf_Output_Normalized))
Harf_Tags
```

سپس در یک حلقه تعداد فعل ها و قید ها را شمرده و چاپ می کنیم:

```
Harf_verb_count = 0
Harf_adverb_count = 0
for token, tag in Harf_Tags:
        if tag.startswith('VERB'):  # Verbs start with 'VB'
            Harf_verb_count += 1
        elif tag.startswith('ADV'):  # Adverbs start with 'RB'
            Harf_adverb_count += 1
print("Number of verbs in Harf: ", Harf_verb_count)
print("Number of adverbs in Harf: ", Harf_adverb_count)
```

نتیجه به صورت زیر خواهد بود:

```
Number of verbs in Harf:  294
Number of adverbs in Harf:  81
```

برای متن اصلی نیز داریم:

```
Number of verbs in Original:  290
Number of adverbs in Original:  77
```

6. ریشه فعلی که در هر فایل بیشترین تکرار را داشته است با استفاده از کتابخانه هضم بیابید.

تابع find_most_frequent_verb را به منظور پیدا کردن فعل با بیشترین تکرار تعریف می کنیم:

```
from collections import Counter
def find_most_frequent_verb(text_with_tags):
    # Tokenization
    # tokens_with_tags = [tuple(word_tag.split(',')) for word_tag in
text_with_tags]
    tokens_with_tags = text_with_tags
    # Filtering and Counting Verbs
```

```
    verbs = [token for token, tag in tokens_with_tags if tag.startswith('VERB')]
    verb_counts = Counter(verbs)

    # Finding the most frequent verb
    most_frequent_verb = verb_counts.most_common(1)[0][0]

    return most_frequent_verb
```

برای حرف داریم:

```
most_frequent_verb_harf = find_most_frequent_verb(Harf_Tags)
print("Most frequent verb in Harf:", most_frequent_verb_harf)
```

نتیجه:

```
Most frequent verb in Harf: بود
```

ریشه را با استفاده از تابع Lemmatizer پیدا می کنیم:

```
from hazm import Lemmatizer
lemmatizer_harf = Lemmatizer()
lemmatizer_harf.lemmatize (most_frequent_verb_harf)
```

نتیجه:

```
'بود#ست'
```

و برای متن اصلی:

```
Most frequent verb in Original: بود
```

نتیجه ریشه:

```
'بود#ست'
```

7. با استفاده از هضم، کدی بنویسید که لیستی از لیست های حاوی 5 کلمه فارسی بگیرد و کلمه ی بی ربط نسبت به بقیه لیست درهر گروه را برگرداند.

این کار را با استفاده از مدل word_embedding و تابع doesnt_match انجام می دهیم:

```
from hazm import WordEmbedding
word_embedding = WordEmbedding (model_type = 'fasttext', model_path =
'word2vec.bin' )
```

ورودی اول:

```
word_embedding.doesnt_match ([ 'بدرود' , 'پنجره' , 'خداحافظ' , 'درود' , 'سلام' ])
```

نتیجه:

```
'پنجره'
```

ورودی دوم:

```
word_embedding.doesnt_match ([  'موش' , 'گربه' , 'مداد' , 'حیوان' , 'سگ' ])
```

نتیجه:

'مداد'

ورودی سوم:

```
word_embedding.doesnt_match ([ 'کاهو' , 'پیاز' , 'بازی' , 'گوجه' , 'خیار' ])
```

نتیجه:

'بازی'

منابع:

- https://www.youtube.com/watch?v=3H_E8WghLkQ
- https://drive.google.com/file/d/1cxUi6L1EHfLl8lvSpChe-JVOPm9BpLNG/view
- https://drive.google.com/file/d/1L1Ytomz5A5CpHJqJCFStt_OMwpSZ6xgh/view
- https://github.com/roshan-research/hazm
- https://huggingface.co/spaces/evaluate-metric/wer
- https://www.roshan-ai.ir/hazm/
- https://www.roshan-ai.ir/harf/