UDACITY

< Return to Classroom

Part of Speech Tagging

REVIEW

HISTORY

Meets Specifications

I love Japan. I have visited Nagoya and Osaka back in the year 2015.

All the Best!!

FURTHER READING

Below are links to some material you might find interesting for more insight on the subject matter
AI in Practice: Identifying Parts of Speech in Python
POS Tagging and Chunking in NLP
POS for Social Media

OVERALL COMMENTS

Congratulations on finishing the project 🎉

You did a great job and should be proud of yourself. After reviewing this submission, I am impressed and satisfied with the effort and understanding put in to make this project a success.

All the requirements have been met successfully 💯 %

I want you to get the best out of this review and hence I have tried to provide you a holistic experience in this review by adding :-

- Few Suggestions which you can try and improve your python skills.
- Appreciation where you did great
- Some learning opportunities to learn POS/HMM beyond coursework

I hope you find these suggestions and informative 😃 👍🏽

Keep doing the great work and all the best for future project.

Kindly rate my work as a project reviewer! I have tried to provide you a detailed feedback basis on the work submitted. Your positive feedback is very helpful and appreciated - thank you!

General Requirements

- Includes `HMM Tagger.ipynb` displaying output for all executed cells
- Includes `HMM Tagger.html`, which is an HTML copy of the notebook showing the output from executing all cells

Good Job on submitting all the correct files

Submitted notebook has made no changes to test case assertions

No modifications have been made to the test cases

Baseline Tagger Implementation

Emission count test case assertions all pass.

- The emission counts dictionary has 12 keys, one for each of the tags in the universal tagset
- "time" is the most common word tagged as a NOUN

Well Done! Emission counts dictionary has 12 keys and "time" is the most common word tagged as NOUN.

An efficient way of implementing pair_counts is as follows

An efficient way of implementing pair_counts is as follows:

```
from collections import Counter, defaultdict
def pair_counts(tags, words):
    # TODO: Finish this function!

    d = defaultdict(lambda: defaultdict(int))

    for tag,word in zip(tags,words):
        d[tag][word]+=1


    return d
    raise NotImplementedError


# Calculate C(t_i, w_i)
#This allows use to strip the data stream into 2 parts a.k.a tags and words
tags = [tag for i, (word, tag) in enumerate(data.training_set.stream())]
words = [word for i, (word, tag) in enumerate(data.training_set.stream())]
#testing
emission_counts = pair_counts(tags, words)
```

PRO TIP to improve python skills
You could use defaultdict to avoid explicit initialization of dictionary keys.

- Read more about Default Dict
- Read more about Counters

You have used return_dict and {} for dictionary declaration, while these are correct, defaultdict is more robust and allows you to avoid errors which might occur due to improper initializations

```
assert len(emission_counts) == 12, \
       "Uh oh. There should be 12 tags in your dictionary."
assert max(emission_counts["NOUN"], key=emission_counts["NOUN"].get) == 'time',
 \
       "Hmmm...'time' is expected to be the most common NOUN."
HTML('<div class="alert alert-block alert-success">Your emission counts look goo
d!</div>')
```

Your emission counts look good!

Baseline MFC tagger passes all test case assertions and produces the expected accuracy using the universal tagset.

- >95.5% accuracy on the training sentences
- 93% accuracy the test sentences

MFC tagger accuracy looks good.
You could use itertools.chain to merge different tuples of words and sequences
Refer this link to learn more about chains

Calculating Tag Counts

All unigram test case assertions pass

Your tag unigrams look good.
Take note of this pythonic way of calculating unigram

```
def unigram_counts(sequences):
  return Counter(chain(*sequences))
tag_unigrams = unigram_counts(data.training_set.Y)
```

All bigram test case assertions pass

All tests are passing
You could take note of following pythonic way of calculating bigrams

```
def bigram_counts(sequences):
  counts = Counter()
  counts.update(chain(*(zip(s[:-1], s[1:]) for s in sequences)))
  return counts
tag_bigrams = bigram_counts(data.training_set.Y)
```

All start and end count test case assertions pass

You could take note of following pythonic way of implementing start end counts

For Starting counts the alternative implementation is :

```
def starting_counts(sequences):
    # TODO: Finish this function!
    d = Counter(sequences)
    return d
    raise NotImplementedError

# TODO: Calculate the count of each tag starting a sequence
tags = [tag for i, (word, tag) in enumerate(data.stream())]
starts_tag = [i[0] for i in data.Y]
tag_starts = starting_counts(starts_tag)
```

For ending counts the alternative implementation is:

```
def ending_counts(sequences):
    d = Counter(sequences)
    return d
    raise NotImplementedError

# TODO: Calculate the count of each tag ending a sequence
ends_tag = [i[len(i)-1] for i in data.Y]
tag_ends = ending_counts(ends_tag)
```

Basic HMM Tagger Implementation

All model topology test case assertions pass

Basic HMM tagger passes all assertion test cases and produces the expected accuracy using the universal tagset.

- >97% accuracy on the training sentences
- >95.5% accuracy the test sentences

Great! Accuracy on both training and testing data sets are above threshold

DOWNLOAD PROJECT

RETURN TO PATH