assignment1_part3

February 4, 2022

[ ]:
```
version = "REPLACE_PACKAGE_VERSION"
```

---

1 Assignment 1 Part 3: Hidden Markov Models (HMMs) (30 pts)

In this assignment, we're going to train a Hidden Markov Model (HMM) that is able to tag words with their part-of-speech (POS) in a sentence.

[ ]:
```
# Configure nltk

import nltk

nltk_data_path = "assets/nltk_data"
if nltk_data_path not in nltk.data.path:
    nltk.data.path.append(nltk_data_path)
```

1.1 Question 1: Load the dataset (20 pts)

Let's first load some POS-tagged data for training and testing our HMM. **Complete the `load_data` function below** that takes two arguments, `data_file`, which specifies the data file to read, and `label`, which indicates whether to include labels in your data.

Under the `assets/conll03` folder, you are given three data files, `eng.train`, `eng.testa` and `eng.testb`, that come from a CoNLL-2003 shared task. The shared task was originally held as a competition for Named Entity Recognition (NER), but the data it provided also includes POS tags that make POS Tagging possible. NER and POS Tagging are very similar tasks and HMMs are capable of handling both of them. After finishing this assignment, you are encouraged to re-purpose your HMM tagger for NER as a self-exercise.

All three data files share the same format. They are concatenation of several documents demarcated by `-DOCSTART- -X- O O`. Sentences are separated by empty lines and each token occupies one line with its POS tag. For example,

```
scientists NNS I-NP O
```

says the token `scientists` has a POS tag of `NNS` (noun, plural). If interested, see here for a reference list of all POS tags. The CoNLL-2003 shared task page also includes some descriptions of the data.

In many NLP tasks, it's commonplace to divide a dataset into training, development and testing sets. Since training and development sets are used for model learning and hyper-parameter tuning, they often include labels (if any). However, labels are often *not* distributed with testing sets to prevent anyone from cheating, especially in competitions. Here we will follow the same convention, by taking `eng.train` as the training set, `eng.testa` as the development set and `eng.testb` as the testing set, and **will only allow the training and development sets to include labels** by specifying the `label` argument:

- **When called with `label=True`**, your function should return a `list`, where each element is a `list` of `tuple` representing each sentence. Each `tuple` contains a token and its POS tag, such as `("scientists", "NNS")`.
- **When called with `label=False`**, your function should also return a `list`, where each element is just a `list` of tokens. POS tags are not included and `tuple` is not needed.

Finally, **in all the three data files, we only consider "substantial" sentences that have at least** 10 **tokens**. You'd need to filter out shorter sentences and should not include them in any of your three subsets of the data.

This function should return a `list`, where each element is either a `list` of `tuple` or a `list` of `str`.

When the argument `label=True`, an example output would be:

```
[[('The', 'DT'), ('European', 'NNP'), ('Commission', 'NNP'), ('said', 'VBD'), ('on', 'IN'), ('
...
[('Africa', 'NNP'), (')', ')'), ('71', 'CD'), ('74', 'CD'), ('75', 'CD'), (',', ','), ('David'
```

When the argument `label=False`, an example output would be:

```
[['SOCCER', '-', 'JAPAN', 'GET', 'LUCKY', 'WIN', ',', 'CHINA', 'IN', 'SURPRISE', 'DEFEAT', '.'
 ...
 ['The', 'lanky', 'former', 'Leeds', 'United', 'defender', 'did', 'not', 'make', 'his', 'Englan
```

[ ]:
```python
def load_data(data_file, label=True):
    """
    Load tokens (and labels, if allowed) from a data_file
    """

    tagged_sents = []

    # YOUR CODE HERE
    raise NotImplementedError()

    return tagged_sents
```

[ ]:
```python
# Let's create the training, development and testing sets using load_data above
# Remember *not* to include labels in the testing set

dataset = {"train": None, "dev": None, "test": None}
```

```
# Fill in the "blank" - replace the "None" above with the correct dataset

# YOUR CODE HERE
raise NotImplementedError()
```

[ ]:
```
# Autograder tests

for subset in dataset:
    assert isinstance(dataset[subset], list), f"Q1: Your {subset} set should be␣
 ↪a list. "

for subset, length in zip(dataset.keys(), [7146, 1783, 1516]):
    assert len(dataset[subset]) == length, f"Q1: Your {subset} set is of an␣
 ↪incorrect length. "

for subset in dataset:
    for index, sent in enumerate(dataset[subset]):
        assert len(sent) >= 10, f"Q1: Sentence at index {index} of your␣
 ↪{subset} set is shorter than required. "
        item_type = tuple if subset in ["train", "dev"] else str
        for item in sent:
            assert isinstance(item, item_type), f"Q1: Your {subset} set should␣
 ↪contain lists of {item_type.__name__}s. "

# Some hidden tests below
```

1.2 Question 2: Train an HMM Tagger (10 pts)

Once we have the training, development and testing sets ready, it's straightforward to train an HMM tagger with the help of the `nltk` library. Specifically, check out the documentation of the `HiddenMarkovModelTagger` class and understand how to use its classmethod `train` to create an HMM tagger. The classmethod `train` also accepts an argument called `test_sequence`, where you can plug in your development data and get a score as an estimate for how well your HMM will perform on unseen data.

This function should return a trained `HiddenMarkovModelTagger` object.

[ ]:
```
from nltk.tag.hmm import HiddenMarkovModelTagger

def train_hmm(train, dev):
    """
    Train an HMM tagger on the training set provided
    """

    hmm_tagger = None

    # YOUR CODE HERE
    raise NotImplementedError()

    return hmm_tagger
```

[ ]:
```
# Train an HMM tagger, which takes a while

hmm_tagger = train_hmm(dataset["train"], dataset["dev"])

# The number it reports is the accuracy on the development set
```

[ ]:
```
# Autograder tests

assert isinstance(hmm_tagger, HiddenMarkovModelTagger), "Q2: Your function␣
 ↪should return a HiddenMarkovModelTagger. "
assert hmm_tagger._states, "Q2: Your HMM's states are empty. The tagger wasn't␣
 ↪trained properly. "

# Some hidden tests
```

Now that we have trained an HMM tagger, let's now apply it to your testing set to see how it does. For example, the line of code below tags the first sentence in the testing set. Feel free to adapt the code to tag more sentences.

[ ]:
```
hmm_tagger.tag(dataset["test"][0])
```