Hidden Markov Model

Install Dependencies

to install dependencies:

```
$ pip install -r requirements.txt
```

Data Structure

the data structure must contain tokens and part of speech (PoS), in order each token must be in a single line, in which tokens and PoS must be separated with a space character.

Preprocess

this method is implemented in `HMM` class available in `pltk/HMM/hmm.py` file. this method will convert the data file into a list of tuples, un which each tuple contains the token and its PoS.

> NOTE: each token will be normalized.

> NOTE: The lines having nonstandard structure will be ignored

Train

to train the model we have to populate two matrix:

- $stateTransision_{N \times N}$: $stateTransision[i, j] = P(State_j|State_i)$ WHERE N is the number of states.
- $tokenProbability_{N \times M}$: $tokenProbability[i, j] = P(Token_j|State_i)$ WHERE M is the number of tokens.

> NOTE: To avoid underflow we will use the logarithm value of probabilities

The pseudo code of calculation is described below:

- count the tokens for each state, also number of altered states and record them in appropriate matrix
- find the number of all states(name is N)
- calculate logarithm value of both matrices
- consider the fact that $log(\frac{a}{b})=log(a)-log(b)$ we can avoid deviding small numbers

The algorithm's complication is $O(N \times M)$

Finding The Most Probable State Sequence

consider state sequence as $S_0S_1S_2S_3$ for tokens of $T = t_0t_1t_2t_3$, we have:
$$ P(T) =P(S_{0} |\$)\prod ^{3}_{1} P(S_{i} |S_{i-1}) * P(t_{i} |S_{i}) =e^{log(P(T))} $$

$$ log(P(T)) =P(S_{0} |\$)\sum ^{3}_{1} P(S_{i} |S_{i-1}) +P(t_{i} |S_{i}) $$

as we know $f(x)=e^x$ is a ascending function, which concludes from, if and only if we maximize $log(P(T))$ the $P(T)$ will maximize too.

Too maximize the $log(P(T))$ we can localy focus on each
$P(S_i|S_{i-1})+P(t_i|S_i)$
term, which has N conditions for each token(N is the number of states).

In order too find this value(and also corresponding state sequence), we will form a matrix in which its columns name are tokens and rows name indicates the states.In the next step, for each cell(S_i, t_j) we will sum two arrays index by index,one taken from *stateTransision* matrix's i's row, and the other taken from *tokenProbability* matrix's j's column. At last, we will find and replace the maximum value for whole array with the previous column.

The algorithm's complication is $O(N^2 \times M)$

Using Nltk functions

The *Hidden Markov Mode* is also implemented in `nltk.tag.hmm` module. too train data we can simply use `nltk.tag.hmm.HiddenMarkovModelTagger.train` finction which will return an instance of `nltk.tag.hmm.HiddenMarkovModelTagger` class.

Testing

Persian Data

The persina data available in this assignment has 59162 tokens which is available in `PoS.txt` file. run `runme_persianData.py` to train both hmm models over this data:

```
$ python3 runme_persianData.py
```

In this code 50647 initial tokens are used for training and rest of 13721 tokens used for testing the models accuracy. The output is like this:

```
using nltk.tag.hmm.HiddenMarkovModelTagger

[('اولین', 'ADJ'), ('سیاره', 'N'), ('خارج', 'N'), ('از', 'P'), ('منظومه', 'N'), ('شمسی', 'ADJ'), ('دیده', 'ADJ'), ('شد
[('طی', 'N'), ('سالهای', 'N'), ('اخیر', 'ADJ'), ('ممکن', 'ADJ'), ('است', 'V'), ('تعدادی', 'QUA'), ('سیاره', 'N'), ('رج
accuracy over 13721 tokens: 85.33

using pltk.HMM.HMM (implemented by myself)
the train process started for 50647 size document
100%|██████████████████████████████| 50647/50647 [00:53<00:00, 951.72it/s]

[('اولین', 'ADJ'), ('سیاره', 'N'), ('خارج', 'N'), ('از', 'P'), ('منظومه', 'N'), ('شمسی', 'ADJ'), ('دیده', 'ADJ'), ('شد
[('طی', 'N'), ('سالهای', 'N'), ('اخیر', 'ADJ'), ('ممکن', 'ADJ'), ('است', 'V'), ('تعدادی', 'QUA'), ('سیاره', 'N'), ('رج
100%|██████████████████████████████| 1305/1305 [04:46<00:00,  4.55it/s]
93.40427082574156
```

to test both models initially we try to tag two sentences:

token	nltk output	pltk output(implemented by myself)
'اولین'	'ADJ'	'ADJ'
'سیاره'	'N'	'N'
'خارج'	'N'	'N'
'از'	'P'	'P'
'منظومه'	'N'	'N'
'شمسی'	'ADJ'	'ADJ'
'دیده'	'ADJ'	'ADJ'
'شد'	'V'	'V'
'.'	'DELM'	'DELM'
'طی'	'N'	'N'
'سالهای'	'N'	'N'
'اخیر'	'ADJ'	'ADJ'
'ممکن'	'ADJ'	'ADJ'
'است'	'V'	'V'
'تعدادی'	'QUA'	'QUA'
'سیاره'	'N'	'N'
'خارج'	'N'	'N'
'از'	'P'	'P'
'منظومه'	'N'	'N'
'شمسی'	'ADJ'	'ADJ'
'دیده'	'ADJ'	'ADJ'
'باشند'	'V'	'V'
'.'	'DELM'	'DELM'

The accuracies are:

nltk accuracy	pltk accuracy
85.33	93.40

English Data

The english data used in this assignment has 100676 tokens which is available from nltk module

```
import nltk
nltk.download('treebank')
all_data = treebank.tagged_sents()
print(sum(len(i) for i in all_data)) # 100676
```

run `runme_englishData.py` to train both hmm models over this data:

```
$ python3 runme_englishData.py
```

In this code 50647 initial tokens are used for training and rest of 13721 tokens used for testing the models accuracy. The output is like this:

```
using nltk.tag.hmm.HiddenMarkovModelTagger

[('Today', 'NN'), ('is', 'VBZ'), ('a', 'DT'), ('good', 'JJ'), ('day', 'NN'), ('.', '.')]
[('Joe', 'NNP'), ('met', 'VBD'), ('Joanne', '-NONE-'), ('in', 'IN'), ('Delhi', 'NNS'), ('.', '.')]
[('Chicago', 'NNP'), ('is', 'VBZ'), ('the', 'DT'), ('birthplace', 'NN'), ('of', 'IN'), ('Ginny', 'DT')]
accuracy over 20039 tokens: 90.11

using pltk.HMM.HMM (implemented by myself)
[('Today', 'NN'), ('is', 'VBZ'), ('a', 'DT'), ('good', 'JJ'), ('day', 'NN'), ('.', '.')]
[('Joe', 'NNP'), ('met', 'VBD'), ('Joanne', 'NNP'), ('in', 'IN'), ('Delhi', 'NNP'), ('.', '.')]
[('Chicago', 'NNP'), ('is', 'VBZ'), ('the', 'DT'), ('birthplace', 'NNP'), ('of', 'IN'), ('Ginny', 'NNP')]
accuracy over 20039 tokens: 87.8586755826139
```

to test both models initially we try to tag two sentences:

token	nltk output	pltk output(implemented by myself)
'Today'	'NN'	'NN'
'is'	'VBZ'	'VBZ'
'a'	'DT'	'DT'
'good'	'JJ'	'JJ'
'day'	'NN'	'NN'
'.'	'.'	'.'
'Joe'	'NNP'	'NNP'
'met'	'VBD'	'VBD'
'Joanne'	'-NONE-'	'NNP'
'in'	'IN'	'IN'
'Delhi'	'NNS'	'NNP'
'.'	'.'	'.'
'Chicago'	'NNP'	'NNP'
'is'	'VBZ'	'VBZ'
'the'	'DT'	'DT'
'birthplace'	'NN'	'NNP'
'of'	'IN'	'IN'
'Ginny'	'DT'	'NNP'

The accuracies are:

nltk accuracy	pltk accuracy
90.11	87.85