Assignment 2

Austin Chau

HW: POS Tagging

John saw the saw and decided to take it to the table.
NNP VBD DT NN CC VBD TO VB PRP IN DT NN

Intro: Data

- Part of the Penn Treebank POS data set
    - Collection of sentences pretagged with POS
- Read the README file
    - `./train_hmm.py ptb.2-21.tgs ptb.2-21.txt > my.hmm`
- ptb.2-21.tgs → all the POS tags
- ptb.2-21.txt → all the sentences
- train_hmm.py → zip the two files together

Intro: Train_hmm

- Either perl or python is fine → they do the same thing
- Zip the POS and sentences together
- Generates my.hmm
  - List of emissions and transitions
    - Emissions → token and tag pair, and the probability assigned to it
      - float(emissions[tag][token]) / emissionsTotal[tag]
    - Transitions → previous token and current token pair, and the probability assigned to it
      - transitions[prevtag][tag]) / transitionsTotal[prevtag]

Task 1

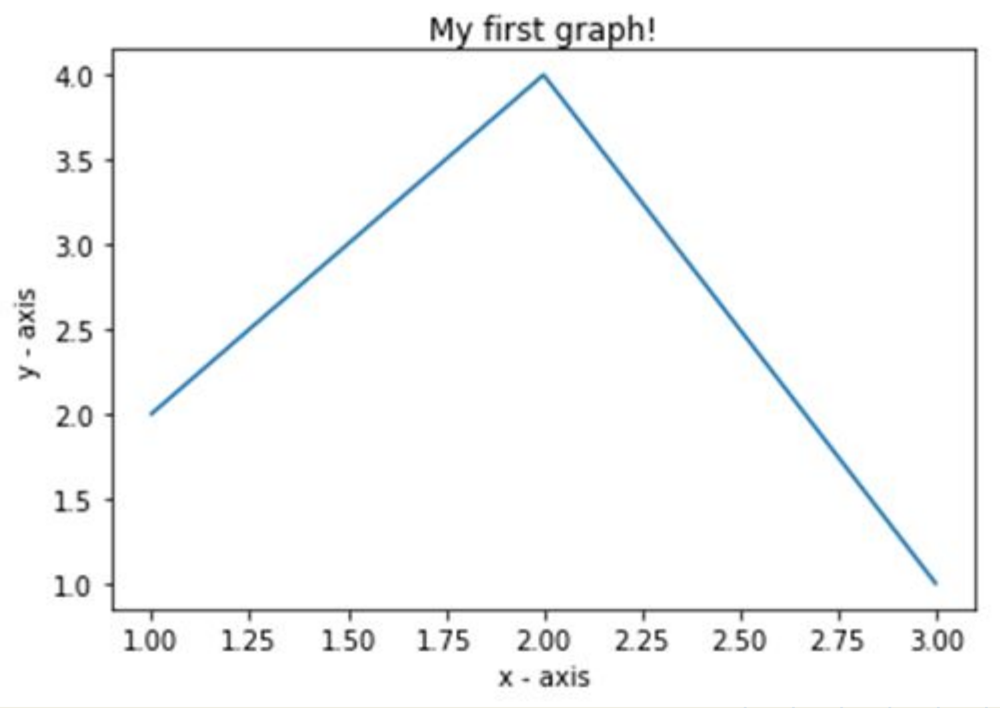
- Train the model on subsets of the training data of different sizes
  - Divide the dataset into several subsets (based on index or randomly)
  - Get different performance by increasing the number of subsets
  - Plot the performance

- Resizing can be done in train_hmm.py, where you don't use all the pairs taken from the tagFile and tokenFile

Plot

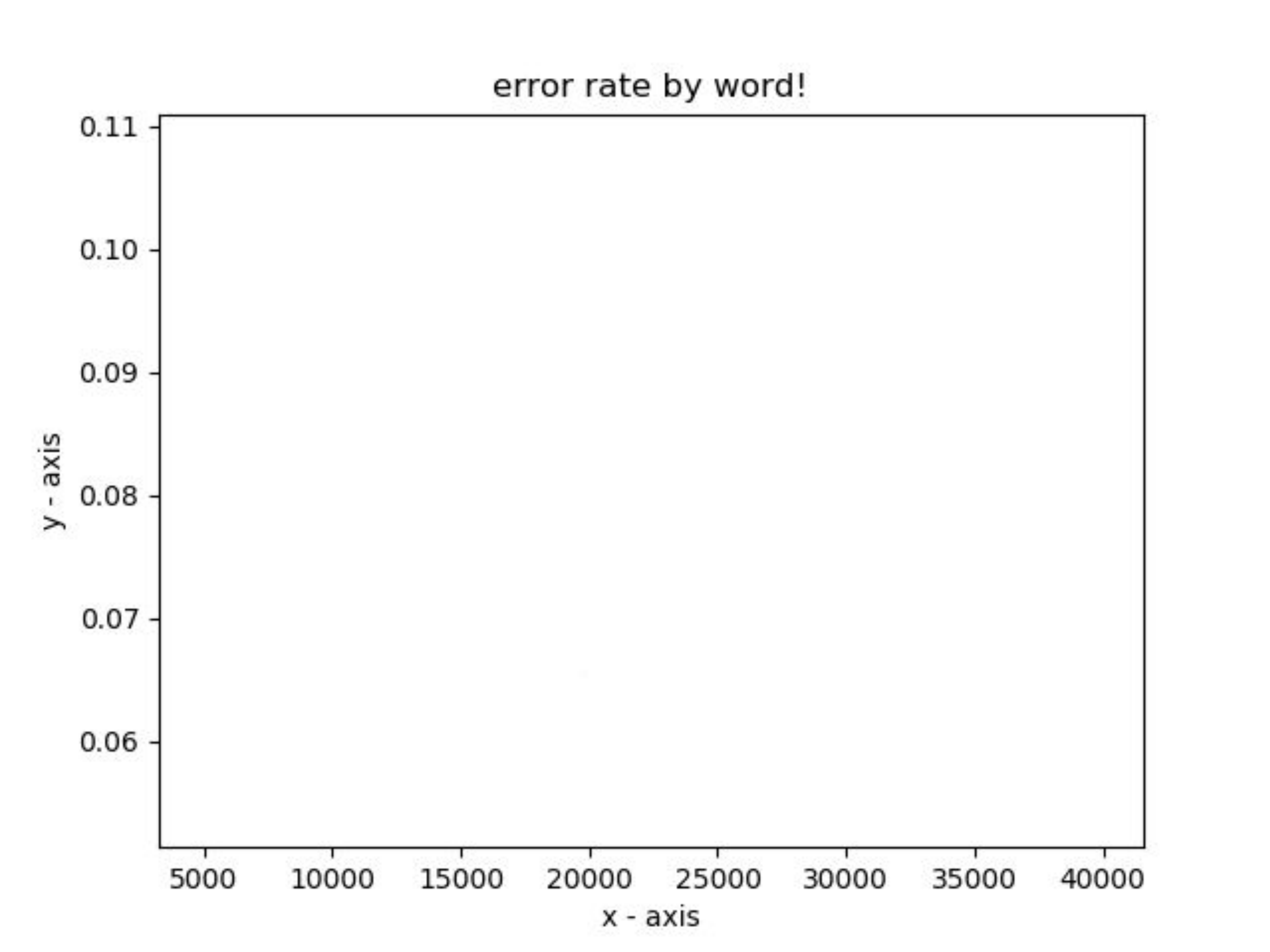
- pip install matplotlib

```python
# importing the required module
import matplotlib.pyplot as plt

# x axis values
x = [1,2,3]
# corresponding y axis values
y = [2,4,1]

# plotting the points
plt.plot(x, y)

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')

# giving a title to my graph
plt.title('My first graph!')

# function to show the plot
plt.show()
```



Plot

- Learning curve



Task 1

- Generate a learning curve
- Give your thoughts about getting more POS-tagged data and and how it would affect your system?

Task 2

- Come up with a way to improve the model

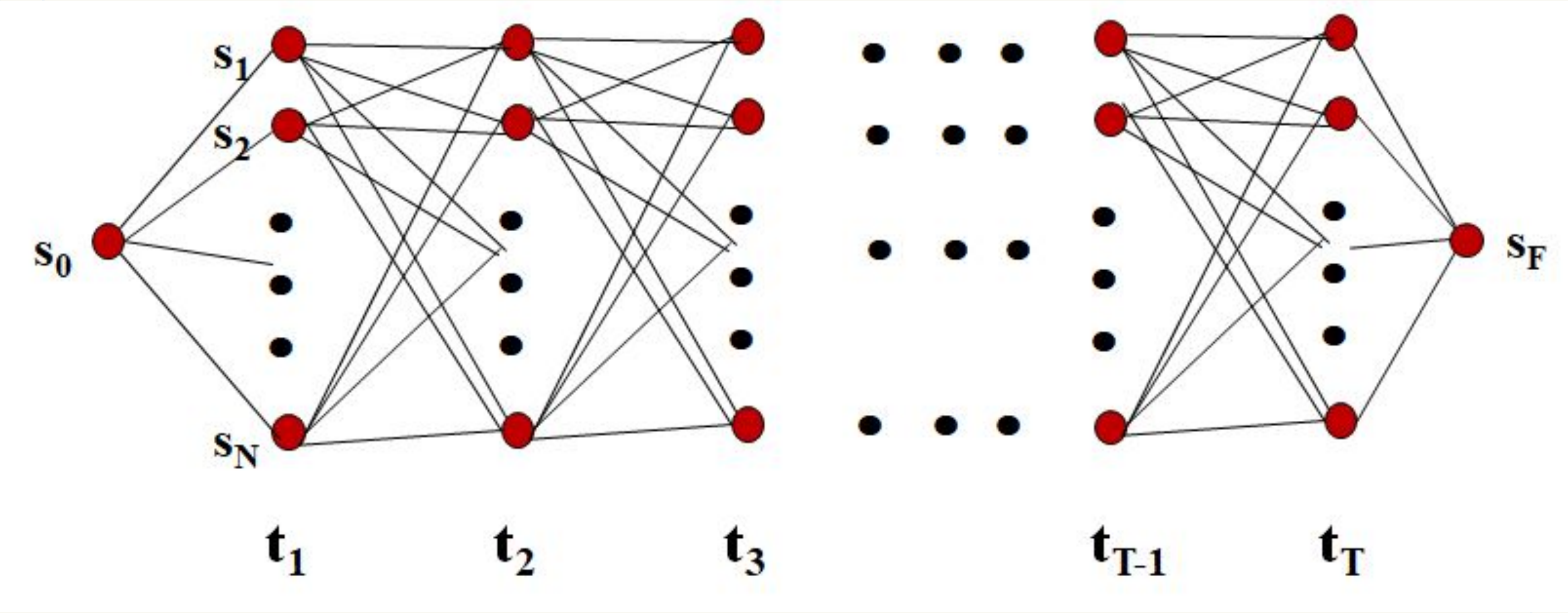
  - Build a replacement for the train_hmm.{py,pl} script (using HMM)

  - Write your own Viterbi algorithm in Python (recommended)

Implement a trigram HMM

- Learn from train_hmm.py
- Keep the observations for each state
- Add two initial states before the first token
- Update previous two states in training

Viterbi for trigram



Viterbi for trigram

Input: a sentence $x_1 \ldots x_n$, parameters $q(s|u, v)$ and $e(x|s)$.
Definitions: Define $\mathcal{K}$ to be the set of possible tags. Define $\mathcal{K}_{-1} = \mathcal{K}_0 = \{*\}$, and $\mathcal{K}_k = \mathcal{K}$ for $k = 1 \ldots n$.
Initialization: Set $\pi(0, *, *) = 1$.
Algorithm:

- For $k = 1 \ldots n$,
  - For $u \in \mathcal{K}_{k-1}$, $v \in \mathcal{K}_k$,

$$\pi(k, u, v) = \max_{w \in \mathcal{K}_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$$

$$bp(k, u, v) = \arg\max_{w \in \mathcal{K}_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$$

- Set $(y_{n-1}, y_n) = \arg\max_{u \in \mathcal{K}_{n-1}, v \in \mathcal{K}_n} (\pi(n, u, v) \times q(\text{STOP}|u, v))$
- For $k = (n-2) \ldots 1$,

$$y_k = bp(k+2, y_{k+1}, y_{k+2})$$

- **Return** the tag sequence $y_1 \ldots y_n$

Implement a trigram HMM

- Use log
  - Initialization
  - Observations out of vocabulary
  - Overflow

- Use dictionaries in Python

Smooth the probability estimates

- To solve data sparsity $P(t_i|t_{i-1}t_{i-2})$
  - Replace every tag by its first letter only, the tags would still be meaningful, only more coarse
  - Estimate the probability by combining more robust but weaker estimators

$$P(t_i|t_{i-1}t_{i-2}) = \lambda_3 \hat{P}(t_i|t_{i-1}t_{i-2}) + \lambda_2 \hat{P}(t_i|t_{i-1}) + \lambda_1 \hat{P}(t_i)$$

How to choose λ?

function DELETED-INTERPOLATION(*corpus*) **returns** $\lambda_1, \lambda_2, \lambda_3$

$\lambda_1 \leftarrow 0$
$\lambda_2 \leftarrow 0$
$\lambda_3 \leftarrow 0$
foreach trigram t_1, t_2, t_3 with $C(t_1, t_2, t_3) > 0$
  depending on the maximum of the following three values
    case $\frac{C(t_1,t_2,t_3)-1}{C(t_1,t_2)-1}$: increment λ_3 by $C(t_1, t_2, t_3)$
    case $\frac{C(t_2,t_3)-1}{C(t_2)-1}$: increment λ_2 by $C(t_1, t_2, t_3)$
    case $\frac{C(t_3)-1}{N-1}$: increment λ_1 by $C(t_1, t_2, t_3)$
  end
end
normalize $\lambda_1, \lambda_2, \lambda_3$
return $\lambda_1, \lambda_2, \lambda_3$

Task 2

- Describe your approach clearly
- Provide the performance of your model on ptb.22.*
- Run your tagger on ptb.23.txt (the test data) and turn in the code and output
- Do not search ptb23.tag

Task 3: Different Languages

- Run the baseline model and your model from Task 2 on Japanese and Bulgarian
- Report and explain the results
  - Why are the baseline model and your model performing better/worse on Japanese and/or Bulgarian?

Bonus Task:

- Current:
  - We have only been reporting #errors compared to the test data in terms of words and sentence.
- Are there better metrics you can do to evaluate the difference between your model and the baseline?
  - What model can you use instead of just tallying up mismatch with the test set?

Bonus task

- In your answer
  - Your modified script
  - Your results in the PDF
  - Your explanation:
    - Why you get this results
    - Why your metric is better