Prof. Björn Gambäck
TA. Tollef Jørgensen

Lab 2 – Tagging

Deadline: 23:59, February 16, 2022

Guidelines

Submit either as a single Jupyter Notebook (with proper formatting!) or a zipped folder with your name (first_last_lab2.zip/ipynb) containing:

1. Source files (Python as .py or Notebooks .ipynb) with the following format: Lab[**Lab#**]_[**Exercise#**].[**py/ipynb**] (e.g., lab2_1a.py)
2. A concise report formatted as a pdf. Feel free to use Word/Pages, or any other means to create pdf files. A template for LaTeX is provided on blackboard if preferred. The report **must contain outputs from your code** along with answers to respective tasks.

For this lab, there is a strict focus on code quality. Don't repeat yourself :-)

Exercises

1 – Ambiguity

Use the *Brown Corpus* and...

(a) Print the 5 most frequent tags in the corpus

(b) How many words are ambiguous, in the sense that they appear with more than two tags?

(c) Print the percentage of ambiguous words in the corpus[1]

(d) Find the top 5 words (longer than 4 characters) with the highest number of distinct tags. Select one of them and print out a sentence with the word in its different forms.

(e) Discuss and think about how you would attack the problem of resolving ambiguous words for a predictive smartphone keyboard

2 – Training a tagger

Explore the performance of a tagger using the *Brown Corpus* and *NPS Chat Corpus* as data sources, with different ratios of train/test data. Use the following ratios:

- Brown 90%/NPS 10%
- Brown 50%/NPS 50%
- NPS 90%/Brown 10%
- NPS 50%/Brown 50%

Create the taggers listed below and comment your results.

(a) Create a *DefaultTagger* using the most common tag in each corpus as the default tag.

(b) Create a combined tagger with the RegEx tagger (see Ch. 5, sec. 4.2) with an initial backoff using the most common default tag. Then, use n-gram taggers as backoff taggers (e.g., *UnigramTagger*, *BigramTagger*, *TrigramTagger*). The ordering is up to you, but justify your choice. Calculate the accuracy of each of the four train/test permutations.

[1] Make sure to not count words more than once...

(c) Select a dataset split of your choice and print a table containing the precision, recall and f-measure for the top 5 most common tags (look up *truncate* in the documentation) and sort each score by count. Do this for all your chosen variations of backoffs (e.g., DefaultTagger, UnigramTagger and BigramTagger).

(d) Using the *Brown Corpus*, create a baseline tagger (e.g. Unigram) with a lookup model (see Ch. 5, sec. 4.3). The model should handle the most 200 common words and store the tags. Evaluate the accuracy on the above permutations of train/test data.

(e) With an arbitrary text from another corpus (or an article you scraped in Lab 1), use the tagger you just created and print a few tagged sentences.

(f) Experiment with different ratios and using only one dataset with a train/test split. Explain your findings.

3 – Tagging with probabilities

Hidden Makrov Models (HMMs) can be used to solve Part-of-Speech (POS) tagging. Use HMMs to calculate probabilities for words and tags, using the appended code.

(a) Implement the missing pieces of the function *task3a()* found in the appended code. Also found on the next page for reference.

(b) Print the probablity of...[2]

- a verb (VB) being "run"
- a preposition (PP) being followed by a verb

A template is found in the code under *task3b()*

(c) Print the 10 most common words for each of the tags *NN, VB, JJ*

(d) Print the probability of the tag sequence *PP VB VB DT NN* for the sentence "I can code some code"

[2]Hint: probdist and cond.freqdist implement a *.prob* method

```python
import nltk
from nltk.corpus import brown
# see https://nltk.readthedocs.io/en/latest/api/nltk.html
# define distinguishable start/end tuples of tag/word
# used to mark sentences
START = ("START", "START")
END = ("END", "END")

def get_tags(corpus):
    tags_words = []
    for sent in corpus.tagged_sents():
        tags_words.append(START)
        # shorten tags to 2 characters each for simplicity
        tags_words.extend([(tag[:2], word) for (word, tag) in sent])
        tags_words.append(END)

    return tags_words

def probDist(corpus, probability_distribution, tag_observation_fn):
    tag_words = get_tags(corpus)
    tags = [tag for (tag, _) in tag_words]
    # conditional frequency distribution over tag/word
    cfd_tagwords = nltk.ConditionalFreqDist(tag_words)
    cpd_tagwords = nltk.ConditionalProbDist(cfd_tagwords, probability_distribution)
    # conditional frequency distribution of observations:
    cfd_tags = nltk.ConditionalFreqDist(tag_observation_fn(tags))
    cpd_tags = nltk.ConditionalProbDist(cfd_tags, probability_distribution)

    return cpd_tagwords, cpd_tags

def task3a():
    corpus = brown
    # maximum likelihood estimate to create a probability distribution
    probability_distribution = None  # IMPLEMENT
    # a function to create tag observations. Hint: can we observe anything with
    unigrams?
    tag_observation_fn = None  # IMPLEMENT
    return probDist(corpus, probability_distribution, tag_observation_fn)

def prettify(prob):
    return "{}%".format(round(prob * 100, 4))

def task3b():
    tagwords, tags = task3a()
    prob_verb_is_run = None  # IMPLEMENT
    prob_v_follows_p = None  # IMPLEMENT
    print("Prob. of a Verb(VB) being 'run' is", prettify(prob_verb_is_run))
    print("Prob. of a Preposition(PP) being followed by a Verb(VB) is",
    prettify(prob_v_follows_p))
```