CS5012 Language & Computation Practical 1 Report

University of St Andrews – School of Computer Science

Student ID: 150014151

10th March, 2020

Table of Contents

1 Introduction ... 3

1.1 Features implemented ... 3

1.2 Usage ... 3

1.3 Tools used ... 3

2 System Design ... 4

2.1 File organisation ... 4

2.2 Data pre-processing ... 4

2.3 Training ... 5

2.4 Testing ... 5

3 Evaluation ... 5

3.1 Basic tagger ... 6

3.2 Smoothing ... 6

3.3 Unknown words ... 7

3.4 Scaling ... 8

3.5 Other languages ... 9

4 Design Considerations Discussion ... 10

4.1 Data structure and underflow ... 10

4.2 Effect of bin size on distribution smoothing ... 10

References ... 12

Appendix A: Universal Tagset ... 13

Appendix B: Evolution of unknown word spelling rules ... 14

Appendix C: Testing Accuracy on Entire Brown Corpus ... 16

1 Introduction

1.1 Features implemented

A POS tagger using the Viberbi algorithm to predict POS tags in sentences is created in Python for this practical. It includes the following implementations:

- Specification:
  - HMM word emission frequency smoothing,
  - Unknown word handling.
- Additional:
  - Extra unknown words rules based on their morphological idiosyncrasies.
  - Command-line parsing interface,
  - HMM training data saving for quicker program execution.

1.2 Usage

Before running the program, create a new virtual environment to install Python libraries such as NLTK and run the following command:

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

To run the POS tagger in Python, move to the "*src*" directory and run the following command.

```
python main.py [-corpus <corpus_name>] [-r] [-d]
```

where:

- -*corpus*: is the name of corpus to use, which can be either "brown" or "floresta". This is an optional argument that defaults to "brown" if nothing is specified.
- -*r:* is a flag that forces the program to recompute the HMM's tag transition and word emission probabilities rather than loading previously computed versions into memory.
- -*d* is a flag that enters *debugging* mode, printing additional statements on the command line.

1.3 Tools used

- Programming language: Python 3.7.
- Python libraries used: NLTK[1].
- Editor: PyCharm[2].
- Version controlling: Git and GitHub (private repository).

[1] NLTK: https://www.nltk.org/api/nltk.html?highlight=witten#nltk.probability.WittenBellProbDist
[2] PyCharm: https://www.jetbrains.com/pycharm/

- Results analysis and charts: Excel.
- Confusion matrix: Scikit-learn[3] and Matplotlib[4].

2 System Design

2.1 File organisation

The project is organised into the following python modules:

- "*main.py*": contains functions critical to the training and testing of the POS tagger (e.g. calculating the HMM's tag transition/word emission probabilities, computing the Viterbi algorithm, etc.).
- "*helpers.py*": contains support functions such as operations on data (e.g. extracting words or tags from the data sets, getting the hapax legomenon from a list, etc.) and printing statements.
- "*config.py*": contains project-wide variables that are set by the command line arguments.

The program runs in three distinct steps: the data pre-processing, the tagger training, and the tagger testing.

2.2 Data pre-processing

The first step consists in splitting command line arguments in order to run different sections of the program (see Section 1.2). This is achieved through Python's "*argparse*" library[5].

Next, sentences from the Brown corpus [1] are split into a training and a testing data set. Start-of-sentence tokens <s> (with a <s> POS) are inserted at the beginning of each sentence in both data sets, as well as end-of-sentences tokens </s> (with a </s> POS), which are appended at the end of each sentence (see code snippet below). The default splits (used for baseline evaluation) are 10,000 sentences in the training set and 500 sentences in the testing set.

```
START_TAG_TUPLE = ("<s>", "<s>")
END_TAG_TUPLE = ("</s>", "</s>")
for sentence in data:
    sentence.insert(0, START_TAG_TUPLE)
    sentence.insert(len(sentence), END_TAG_TUPLE)
```

[3] Scikit-learn: https://scikit-learn.org
[4] Matplotlib: https://matplotlib.org/
[5] Python argparse library: https://docs.python.org/3.7/library/argparse.html

2.3 Training

Using only the training set, the HMM's bigram distribution, in the form of tag transitions and word emissions probabilities, are calculated and smoothed using NLTK's *WittenBellProbDist*[6] function.

These large probability matrices (which are stored in python dictionaries) are each saved in a Pickle[7] "*.pkl*" file, allowing the whole program to execute faster in future runs by loading the matrices back into memory. A command-line flag can be set to always force both matrices to be re-calculated.

2.4 Testing

The POS tagger is tested on the aforementioned testing data set by comparing the predicted tags with the actual tags. The predicted tags are determined by using the Viterbi algorithm (which makes use of the previously generated HMM's tag transition and word emission probabilities) before back-tracing the generated Viterbi trellis to reconstruct the most likely sequence of POS tags for each sentence in the testing data set.

Finally, the tagger's accuracy is calculated by counting the number of correct predictions for each predicted tag with the actual tags from the Brown corpus.

```
correct_tag_counter = 0
for i in range(0, len(actual_tags)):
    if actual_tags[i] == predicted_tags[i]:
        correct_tag_counter += 1
return (correct_tag_counter / len(actual_tags)) * 100
```

3 Evaluation

The evaluation consists of using the recommended data split of 10,000/500 sentences for the training/testing data sets to evaluate the POS tagger. Although the submitted code only contains the final POS tagger, the evolution of the accuracy was measured by reverting to previous git commits and measuring the accuracy at each milestone (the code used to achieve these accuracies is placed in the appendix). Figure 1 depicts the evolution of the accuracy at each milestone, which is explored in the following subsections.

[6] *WittenBellProbDist*: https://www.nltk.org/api/nltk.html?highlight=witten -nltk.probability.WittenBellProbDist
[7] Pickle: https://docs.python.org/3.7/library/pickle.html

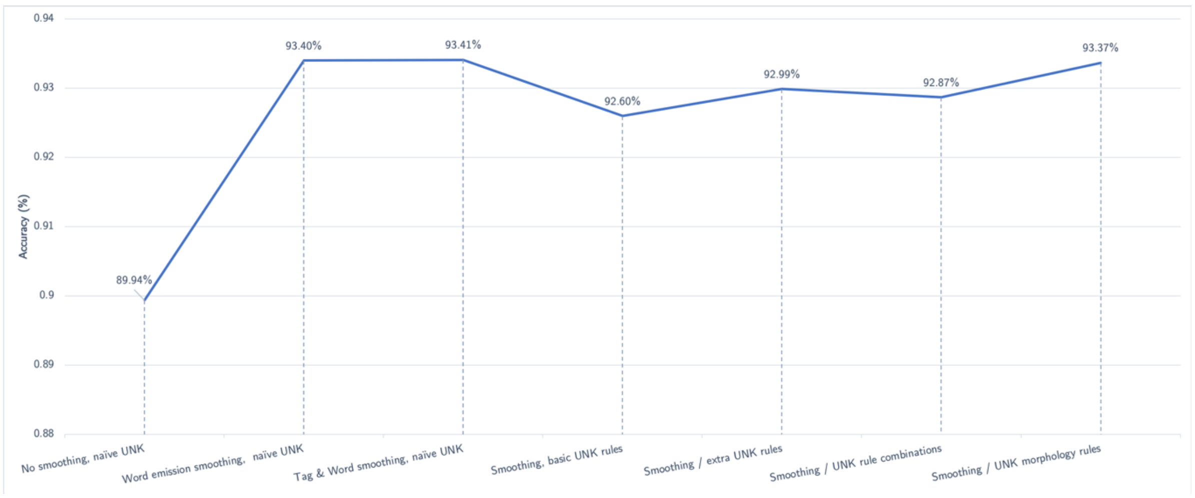


Figure 1: Evolution of the tagger's accuracy.

3.1 Basic tagger

The first tagger consisted in the simplest implementation of the POS tagger mentioned in the "Viterbi algorithm" section of the practical specification, with no smoothing implemented and a naïve unknown word handling.

Frequency estimations are calculated for the tag transition and word emission probabilities, and the Viterbi algorithm with back-tracing is used to calculate the most likely sequence of POS tags each sentence in the test corpus. The accuracy reaches 89.94% with this basic tagger. The accuracy is at its lowest due to the naïve smoothing. Indeed, assigning a pre-defined 1/1000 probability to every unseen word emission means that there is a uniform probability that the word could be any tag, as much a "DET" tag as "<s>" tag, which is naïve way of not handling unknown words. For this reason, smoothing the HMM's tag transition and word emission probabilities is required.

3.2 Smoothing

When smoothing is applied to the HMM's word emission probabilities, the accuracy jumps up to 93.40%, which is due to the fact that probabilities are being assigned to unseen events in the testing set. However, when applying the same relative frequency estimation (Witten-Bell Probability Distribution) to the tag transition probabilities, the accuracy only increases by 0.01%. This likely due to the fact that the universal tagset is used on the tagged Brown corpus, reducing the total number of tags that the POS tagger has to handle to 14 (including <s> and </s>) (see Appendix A for more detail on the universal tagset). Therefore, there is in total 196 possible combinations[8] of tags emissions, which translates to very low chances of having unseen tag transitions occurring in the testing set.

[8] 196 = 14 tags * 14 tags.

Additionally, of these 196 combinations, many are grammatically incorrect, thus further reducing the actual number of counts. For example, many tag transitions are impossible in English, such as an <s> immediately followed by a <.>. However, this particular combination would be possible in other languages such as Spanish, where an upside-down question-mark is used to open sentences that express a question:

- Spanish: *¿Adónde vas, Mark?*
- English: *Where are you going, Mark?*

Therefore, the smoothing will not affect the tag transition probabilities as much as the smoothing applied on the word emissions, as there are potentially 16,256,688 words[9] emissions possible in the Brown corpus alone.

3.3 Unknown words

Initially, the three rules mentioned in the "Unknown words" section of the specification are implemented to handle unknown words. These include replacing words ending in "-ing" with the string "UNK-ing", replacing capitalised words with "UNK-capitalised" and any other unknown word with "UNK" (see Appendix B.1). This led to any word matching a specified rule to be replaced by the appropriate "UNK-_" string, thus giving higher probability mass to the replaced hapax legomena (words that only occur once) rather than keeping low-probability emissions for these rare words.

However, this handling of unknown words proved to be inefficient as it caused the accuracy to drop from 93.41% to 92.6%. This drop is likely caused by the lack of rules, as the majority of words do not fall in the aforementioned rules. Therefore, six additional rules are added to match a greater range of word spellings, such as words ending in "-ed" or with apostrophes followed by an "s", hyphenated words, numbers, decimal numbers and currencies (words starting with "$") (see Appendix B.2). These new rules helped the accuracy increase to 92.99% as less words where being replaced with "UNK" but rather with a more specific string.

An attempt to improve these new rules was to combine them to create new rules. For example, rules related to word spelling were broken down into whether or not they were capitalised (the initial "UNK-ing" string was used to create two new rules: capitalised "Unk-ed" and not capitalised "unk-ed") (see Appendix B.3). However, this caused the accuracy to decrease down to 92.87%, probably due to the proximity of these strings replacing the unseen words.

The ideal solution to fix the problems encountered in the rules merging is to create "UNK" tags that encompass many rules e.g. replacing all words that are spelled like verbs with the tag "UNK-verb". This solution can be implemented by analysing the

[9] 16,256,688 = 1,161,192 words in the Brown corpus * 14 tags.

morphological structures of common English words [2]. Based on Gerald P. Delahunty and James J. Garvey's book "*The English Language: From Sound to Sense*" [2] and an open-source implementation of morphological rules [3], a list of morphological idiosyncrasies commonly used in English are implemented to group words following multiple similar rules under the same "UNK" tag. For examples, words ending in the following substrings "ed", "-ing", "-ate", "-ise", "-ize" and "-ify" are all likely to be verbs and are consequently all replaced by "UNK-verb". The following morphological combinations are used to create four "UNK" tags:

```
NOUN_SUFFIX = ["action", "age", "ance", "cy", "ee", "ence", "er", "hood", "ion", "ism",
"ist", "ity", "ling", ment", "ness", "or", "ry", "scape", "ship", "dom", "ty"]
VERB_SUFFIX = ["ed", "ify", "ise", "ize", "ate", "ing"]
ADJ_SUFFIX = ["ous", "ese", "ful", "i", "ian", "ible", "ic", "ish", "ive", "less", "ly", "able"]
ADV_SUFFIX = ["wise", "wards", "ward"]
```

These lists of rules, combined with the existing ones (see Appendix B.4), help boost the accuracy to 93.37%, which is the highest accuracy of all the different solutions explored for unknown word handling.

A few observations were made on the tuning of parameters when handling unknown words. The first one is the threshold used to consider a word as infrequent. In the final version of the code, only words occurring once are considered as infrequent. However, to experiment, the threshold was increased to two, and the resulting accuracy lowered from 93.37% to 92.54%. Indeed, considering too many words as infrequent means that the number of unseen words in the testing set increases considerably.

The second observation involves the order of the spelling rules to replace words with new "UNK" tags. Indeed, placing rules that encompass too many cases early in the *if* statements causes the more specific ones to never be used. This leads to some "UNK" tags being overused, leading back to the initial problem of not having enough rules. To test this, the order of the rules was inversed, causing the accuracy to drop from 93.37% to 93.35%.

3.4 Scaling

The final tagger with smoothed HMM probabilities and unknown word handling completed (submitted source code) is tested on the entire Brown corpus. An identical

95/5% train/test data split is used on the 57,340 sentences in the corpus, resulting in the following split:

- Train data set: 54,473 sentences
- Test data set: 2,867 sentences

After running for almost 12 minutes, the tagger achieves an accuracy of 94.39%, the highest recorded during the entire practical (see Appendix C). The confusion matrix of the test can be found in Figure 2 below:

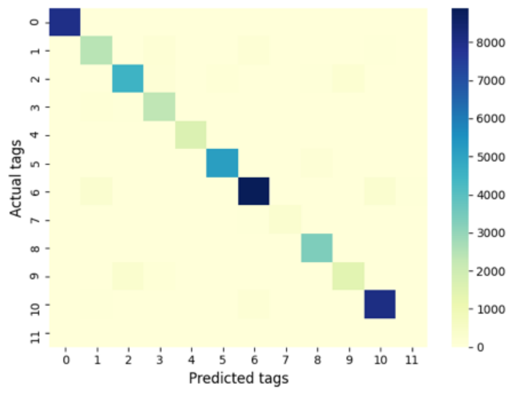


Figure 2: Confusion matrix on entire Brown Corpus.

This reveals how using a larger training set can help increase the accuracy as there will be less hapax legomenon in the training set, and therefore fewer unseen words in the testing set. Because less words will be replaced by "UNK" tags, the tagger will be more accurate in this probability assignment when it selects the most likely POS tag for each word in the test sentences.

3.5 Other languages

Unfortunately, an unsuccessful attempt at testing the tagger with new languages was made. A lot of time was initially wasted on attempting to load French corpora into the program, as NLTK does not provide any.

Ultimately, the Portuguese Floresta Treebank[10] supplied by NLTK was used, but due to the large amounts of tags present in the corpus compared to the universal tagset used with the Brown corpus, the tagger did not scale well, and could not be debugged in time.

[10] Floresta Treebank: https://www.linguateca.pt/Floresta/

4 Design Considerations Discussion

This section covers discussions over design decisions that were made when encountering either general POS tagging-related or Python-related problems.

4.1 Data structure and underflow

Initially, the training and testing data sets were merged into a single list of words, thus losing their sentence structure. This caused underflow issues in the Viterbi algorithm function as the entire testing set was now one large sentence. Due to the division of decreasingly small terms, the program eventually underflowed when reaching values in the zone of 1-300. Indeed, Python cannot store floats that are smaller than 1e-308 [4], thus underflowing and causing 0 probabilities that led to further errors down the execution flow. Two countermeasures were implemented to protect against underflow:

- Modifying existing data structures and functions to maintain the sentence structure of the datasets. This often led to embedded loops, the outer one looping across all sentences and the inner one looping through all words in the sentence.

- Discarding sentences that are over a pre-defined maximum sentence length. This maximum value is set to 150 words and ensures that large sentences cannot cause underflow.

4.2 Effect of bin size on distribution smoothing

Regarding the number of bins to use when smoothing the HMM's tag transition and word emission probabilities, different values were tested (ranging from 100 to 1,000,000,000) and their accuracies measures compared.

Coincidentally, the number of bins for the tag transitions and word emissions matches the number of digits in the number of possible combinations for each (recall that there are 196 possible combinations of tag transitions and 16,256,688 possible combinations of word emissions): 100 bins are used to smooth the tag transitions probabilities and 10,000,000

It is important to test different bin sizes, as using the wrong number can greatly affect the accuracy. Indeed, if number of bins is too high (the range of each bin becomes very small), events are split into more outcomes, therefore considerably decreasing their values and making the distribution too dense, causing spikes in the distribution (see Figure 3, right). Whereas if the number of bins is too small, then their range is too large and the distribution is no longer representative of the underlying data (see Figure 3, left).

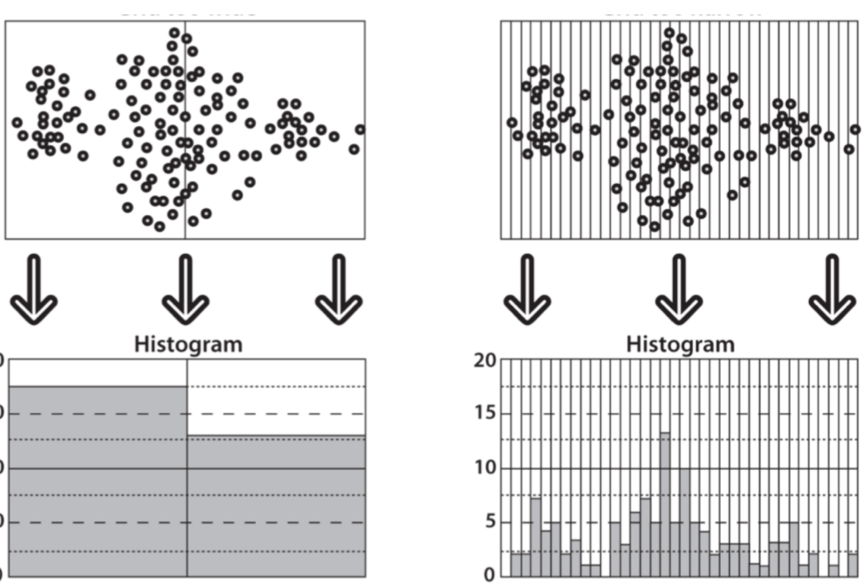


Figure 3: Image depicting the effect of the number of bins when smoothing distributions [5].

References

[1] W. N. Francis and H. Kucera, "Brown Corpus Manual," *Department of Linguistics, Brown University*, 1979. [Online]. Available: http://korpus.uib.no/icame/brown/bcm.html. [Accessed: 09-Mar-2020].

[2] G. P. Delahunty and J. J. Garvey, *The English Language: From Sound to Sense*. 2010.

[3] M. Tosik, "viterbi-pos-tagger/hmm.py at master · melanietosik/viterbi-pos-tagger," 2018. [Online]. Available: https://github.com/melanietosik/viterbi-pos-tagger/blob/master/scripts/hmm.py. [Accessed: 10-Mar-2020].

[4] "sys — System-specific parameters and functions — Python 3.7.7rc1 documentation," *Python Software Foundation*, 2020. [Online]. Available: https://docs.python.org/3.7/library/sys.html#sys.float_info. [Accessed: 09-Mar-2020].

[5] A. K. Gary Bradski, *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, 2016.

Appendix A: Universal Tagset

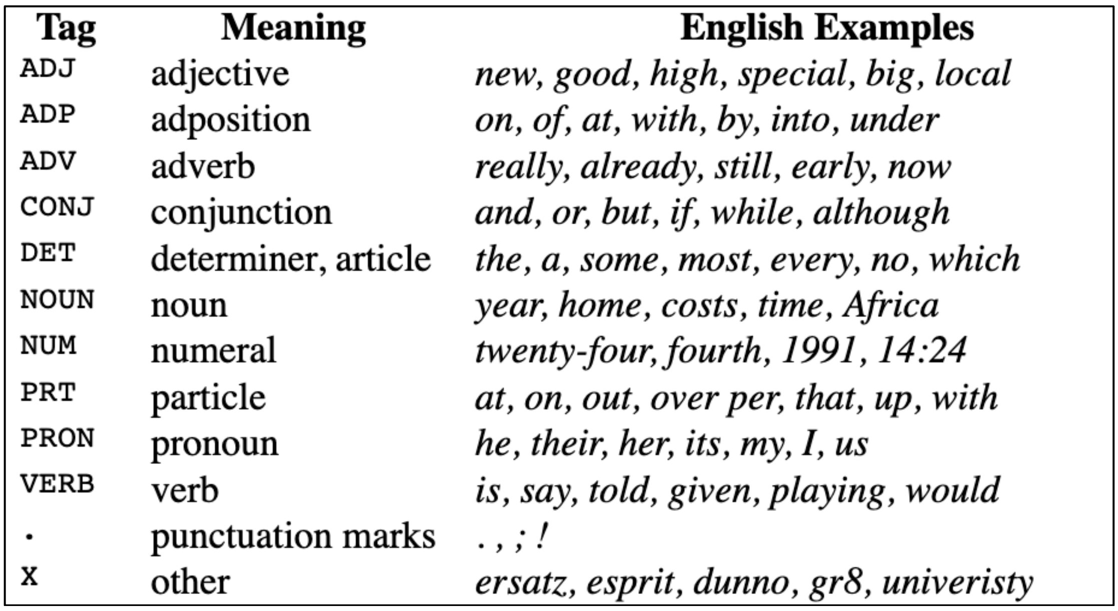
The Brown corpus is used with the "universal tagset" (see code snippet below) to reduce the number of POS tags from 283[11] to 12 (excluding <s> and </s>).

```python
import nltk
from nltk.corpus import brown

nltk.download('brown')
nltk.download('universal_tagset')

tagged_sentences = brown.tagged_sents(tagset='universal')
```

The universal POS tags and their meaning can be found below:



Tag	Meaning	English Examples
ADJ	adjective	*new, good, high, special, big, local*
ADP	adposition	*on, of, at, with, by, into, under*
ADV	adverb	*really, already, still, early, now*
CONJ	conjunction	*and, or, but, if, while, although*
DET	determiner, article	*the, a, some, most, every, no, which*
NOUN	noun	*year, home, costs, time, Africa*
NUM	numeral	*twenty-four, fourth, 1991, 14:24*
PRT	particle	*at, on, out, over per, that, up, with*
PRON	pronoun	*he, their, her, its, my, I, us*
VERB	verb	*is, say, told, given, playing, would*
.	punctuation marks	*. , ; !*
X	other	*ersatz, esprit, dunno, gr8, univeristy*

[11] 283 POS tags includes all merging combinations of tags that can be merged, which is checked by retrieving all the unique values from the list of all tags in the data. POS tags: http://korpus.uib.no/icame/manuals/BROWN/INDEX.HTM#bc6

Appendix B: Evolution of unknown word spelling rules

Appendix B.1 Basic UNK rules

```
def unknown_words_rules(word: str) -> str:
    if word.endswith('ing'):
        return"UNK-ing"
    elif word.istitle():
        return "UNK-capitalised"
    return "UNK"
```

Appendix B.2 Extra UNK rules

```
import re
def unknown_words_rules(word: str) -> str:
    if word.endswith('ing'):
        return"UNK-ing"
    elif word.endswith('ed'):
        return "UNK-ed"
    elif word.endswith("'s"):
        return"UNK-apostrophe-s"
    elif word.startswith('$'):
        return "UNK-currency"
    elif word.istitle():
        return "UNK-capitalised"
    elif word.isdigit():
        return "UNK-number"
    elif re.compile(r'\d+(?:[,.]\d*)?').match(word):
        return "UNK-decimal-number"
    elif '-' in word:
        return "UNK-hyphen"
    return "UNK"
```

Appendix B.3 UNK rules combinations

```
def unknown_words_rules(word: str) -> str:
    if word.startswith('$'):
        return "UNK-currency"
    elif word.isdigit():
        return "UNK-number"
    elif re.compile(r'\d+(?:[,.]\d*)?').match(word):
        return "UNK-decimal-number"
    elif word.istitle():
        if word.endswith('ing'):
            return "Unk-ing"
        elif word.endswith('ed'):
            return "Unk-ed"
        elif word.endswith('ing'):
            return "Unk-ing"
        elif word.endswith("'s"):
            return "Unk-apostrophe-s"
        elif '-' in word:
            return "Unk-hyphen"
    elif not word.istitle():
        if word.endswith('ing'):
            return "unk-ing"
        elif word.endswith('ed'):
            return "unk-ed"
        elif word.endswith('ing'):
            return "unk-ing"
        elif word.endswith("'s"):
            return "unk-apostrophe-s"
        elif '-' in word:
            return "unk-hyphen"
    return "UNK"
```

Appendix B.4 Morphological UNK rules

```
NOUN_SUFFIX = ["action", "age", "ance", "cy", "ee", "ence", "er", "hood", "ion", "ism", "ist", "ity", "ling",
               "ment", "ness", "or", "ry", "scape", "ship", "dom", "ty"]
VERB_SUFFIX = ["ate", "ify", "ise", "ize", "ed", "ing"]
ADJ_SUFFIX = ["able", "ese", "ful", "i", "ian", "ible", "ic", "ish", "ive", "less", "ly", "ous"]
ADV_SUFFIX = ["ward", "wards", "wise"]

def unknown_words_rules(word: str) -> str:
    if any(char.isdigit() for char in word):
        if word.startswith('$'):
            return "UNK-currency"
        elif re.compile(r'\d+(?:[,.]\d*)?').match(word):
            return "UNK-decimal-number"
        return "UNK-number"
    elif any(char in set(string.punctuation) for char in word):
        return "UNK-punctuation"
    elif any(char.isupper() for char in word):
        return "UNK-uppercase"
    elif any(word.endswith(suffix) for suffix in NOUN_SUFFIX):
        return "UNK-noun"
    elif any(word.endswith(suffix) for suffix in VERB_SUFFIX):
        return "UNK-verb"
    elif any(word.endswith(suffix) for suffix in ADJ_SUFFIX):
        return "UNK-adj"
    elif any(word.endswith(suffix) for suffix in ADV_SUFFIX):
        return "UNK-adv"
    elif word.istitle():
        return "UNK-capitalised"
    elif word.endswith("'s"):
        return"UNK-apostrophe-s"
    elif '-' in word:
        return "UNK-hyphenated"
    return "UNK"
```

Appendix C: Testing Accuracy on Entire Brown Corpus

File - brown recalculate HMM

```
/Users/ajaamour/Environments/CS5012-P1/bin/python /Users/ajaamour/Projects/CS5012-P1/src/main.py -r
Corpus used: Brown Corpus (universal tagset)
File 'data_objects/transition_probabilities.pkl' regenerated and saved at data_objects/transition_probabilities.pkl.
File 'data_objects/emission_probabilities.pkl' regenerated and saved at data_objects/emission_probabilities.pkl.

POS Tagging accuracy on test dataset: 94.39%
--- Runtime: 718.13 seconds ---

Process finished with exit code 0

```

Page 1 of 1