Dylan Govender – 221040222
COMP703 – Assignment 1 Report – 22/04/2024

[1] HMM-POS-Tagger

[1.1] Data-Preprocessing for an HMM-POS-Tagger:

The data used for this model is saved as the text file, *isiswati_pos_tagging_corpus.txt.* From each line of the file, five components have been extracted. However, the HMM-POS-Tagger requires only two components: 'token', and the 'UPOS'. The corpus, named *tokenized_text*, is generated by saving the 'token' and 'UPOS' in a 2-tuple which is then stored in a sentence array, named *tokenized_sent*, which is subsequently appended to the corpus. To create the sentence array, the full stop token ['.'] must be identified, which serves as an indicator for when the current sentence must be stored into the corpus, and when the sentence array should be reinitialised for storing the next sentence. Below is a sample of the corpus:

```
[[('Ngetulu', 'ADV'), ('kwaloko', 'POSS'), (',', 'PUNC'),
('kuba', 'V'), ('khona', 'CONJ'), ('kuniketela', 'V'),
('kwekwakhiwa', 'POSS'), ('kwemaKomidi', 'POSS'), ('emaWadi', 'N'),
(',', 'PUNC'), ('njengemitimba', 'ADV'), ('yamasipala', 'POSS'),
('lelulekako', 'POSS'), ('kanye', 'ADV'), ('nalabamele', 'ADV'),
('ummango', 'N'), ('kute', 'CONJ'), ('kuhanjiswe', 'V'),
('tidzingo', 'N'), ('netiphakamiso', 'ADV'), ('temmango', 'POSS'),
('kumkhandlu', 'N'), ('.', 'PUNC')]]
Number of sentences: 2682
```

The order for each tuple in a sentence was maintained, since for HMMs, each state depends on a fixed set of previous hidden states, in this case, a tuple will consist of the tag being hidden and the token/word being observed enabling HMMs to capture the local context more effectively. However, to enhance the model's generalisation across different contexts, the sentences in the corpus was shuffled. Cross validation was used to estimate the model's performance and enhance its generalisation, by dividing the corpus, *tokenized_text*, into the training set, *X* and test set, *y*.

[1.2] Training the HMM-POS-Tagger:

Using the HMM model provided by the NLTK library, we can train the model on the training set, *X*, using the provided function, `nltk.HiddenMarkovModelTagger.train(X)`.

[1.3] Evaluating the HMM-POS-Tagger:

The performance of the HMM-POS-Tagger was assessed using metrics such as accuracy, precision, recall, and F1-score, by employing the Scikit-Learn library, on the test set, *y*. The model achieved an accuracy of 76% and a F1-score of 74%. The accuracy measures the proportion of correctly predicted POS-tags out of the total number of predictions, indicating that 76% of the model's predictions was correct. It is calculated as follows:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

Precision evaluates the model's ability in identifying relevant instances and is calculated as follows:

$$Precision = \frac{True\ Positive}{True\ Postive + False\ Negative}$$

In the above calculation, the *True Positive* denotes the number of correct predictions, and the *False Negative* denotes the number of incorrect predictions, where the model failed to identify a particular tag according to the actual tag. Recall measures the precision at the gold standard and reveals whether a model was able to capture relevant instances. F1-score measures the model's performance in identifying and capturing all relevant instances. For this report, accuracy and F1-score offer a more pertinent analysis for comparing both models. F1-score is calculated as follows:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

[1.4] HMM-POS-Tagger - Model Evaluation:

```
Accuracy: 0.76
F1-Score: 0.74
```

[2] CRF-POS-Tagger

[2.1] Data-Preprocessing for the CRF-POS-Tagger:

Unlike HMM, every component of the extracted data can be used for training a CRF-POS-Tagger, albeit requiring more preprocessing. Like HMM, the corpus follows a similar design, but the sentences store 5-tuples instead of 2-tuples. Preprocessing steps for the *morph_analysis* involved removing the square brackets, separating tokens and tags, converting tags to uppercase using regex, and organising each token and its corresponding tag into a 2-tuple. In the *morph_analysis,* if multiple tags are found for a token, then only the last tag of that token was saved. Below is a sample of a sentence in the corpus after preprocessing:

```
[[('Ngetulu', [('nga', 'ADVPRE'), ('tulu', 'ADV')], 'tulu', 'ADV',
'ADV'), ('kwaloko', [('kwa', 'POSSCONC15'), ('loko', 'POS2')], 'loko',
'POSS15', 'POSS')]]
morph_analysis_1 = [('nga', 'ADVPRE'), ('tulu', 'ADV')]
morph_analysis_2 = [('kwa', 'POSSCONC15'), ('loko', 'POS2')]
```

For training a CRF-POS-Tagger, relevant features must be extracted from the data and stored into the dictionary, named *features*. The *morph_analysis* component was crucial in this step, as it provides the token's root word, prefixes, and suffixes. The root word assists the CRF model to identify the general syntactic category of a word, while prefixes and suffixes can influence the grammatical function of the word. Data leakage into the features was avoided to ensure the accuracy of the model. If we added the POS-tag for each root, suffix, and prefix into the features, then the model uses this to its advantage and provides an accuracy of 99.9%, constituting cheating. Therefore, it's crucial to exclude POS-tags from the features, allowing the model to learn to identify and capture relevant information autonomously. Below is a sample of the training set, *X*, after storing the extracted features of the data:

```
[{'token': 'Ngetulu', 'token_length': 7, 'lemma': 'tulu',
'previous_token': '', 'next_token': 'kwaloko', 'lower_cased_token':
'ngetulu', 'is_start': True, 'is_end': False, 'is_capitalized': True,
'is_upper_cased': False, 'is_lower_cased': False, 'is_punctuation':
False, 'is_numeric': False, 'has_hyphen': False, 'has_capital_inside':
False, 'root': 'tulu', 'prefix_1': 'nga', 'prefix_2': '', 'prefix_3':
'', 'suffix_1': '', 'suffix_2': '', 'suffix_3': ''}]
```

After extracting the relevant features from the data, it is split into the training set, *X*, and the test set, *y*. Cross-validation was used to ensure that the model's performance can be evaluated and its ability to generalise effectively on unseen data can be assessed.

[2.2] Training the CRF-POS-Tagger:
The CRF model for POS-tagging was constructed using the crf_suite library, and the model was trained using the function: `crf.fit(X_train, y_train)`. This function fitted the CRF model to the training data, where *X_train* represented the input features and *y_train* represented the corresponding POS-tags.

[2.3] Evaluating the CRF-POS-Tagger:
The functionality from the crf_suite library was utilised to calculate the accuracy and F1-score of the model, by predicting on the test set, *X_test*, to generate the *y_pred* test set, which was then compared to the actual test set, *y_test*. The model achieved an accuracy of 94%, indicating that 94% of its predictions were correct. Additionally, an F1-score of 84% indicates a strong balance between precision and recall which suggests that the model effectively identified and captured a significant portion of all relevant instances.

[2.4] CRF-POS-Tagger Model Evaluation

```
Accuracy: 0.94
F1-Score: 0.83
```

[3] Comparison between the HMM-POS-Tagger and CRF-POS-Tagger:

[3.1] Model Structure

HMM-POS-Tagger: HMM is a generative model. HMMs model the joint distribution of the words and POS-tags. In the model, the likelihood of transitioning from one POS-tag to another is governed by the transition probabilities, and the likelihood of observing a particular word given its associated POS-tag is governed by the emission probabilities. HMMs are computationally less expensive.

CRF-POS-Tagger: CRF is a discriminative model. CRFs model the conditional probability of a sequence of POS-tags given a sequence of words. CRFs do not make the strong independence assumption about observations that HMMs do. Instead, CRFs capture dependencies between adjacent labels more flexibly. CRFs are computationally more expensive.

[3.2] Model Performance

HMM-POS-Tagger: In HMM, each state depends on a fixed set of previous hidden states, as mentioned previously, which enabled the HMM to capture the local context. This model may struggle to capture complex dependencies between words in a sequence. They are limited in capturing long-range dependencies and might only perform well if there are strong dependencies between adjacent POS-tags. This model achieved an accuracy of 76% and a F1-score of 74%. The HMM's performance is weaker than the CRF's performance.

CRF-POS-Tagger: In CRF, non-consecutive states (shown in the extracted features) can be considered as a feature function which enabled this model to capture the global and local context. CRF-POS-Tagger captured more complex dependencies between adjacent POS-tags, making them substantially more accurate than HMMs, especially where context is used to determine a POS-tag. The CRF-POS-Tagger outperforms the HMM-POS-Tagger with an accuracy of 94% and a F1-score of 84%.

[4] Conclusion
Based on the evaluation and comparison between each model, it can be concluded that the CRF-POS-Tagger is better than the HMM-POS-Tagger. CRFs are better than HMMs.