

CS 491/691 Project 1

Natural Language Processing

Please submit on zip file to Webcampus in a folder named Project1. You are provided with a `test_script.py` which includes a `load_data` function for the N-Gram portion of the project. I will use the `test_script.py` to evaluate your implementations. Make sure you can run `test_script.py` without errors. Assume you are running `test_script.py` in Project1 with `automata.py`, `edit_distance.py` and `ngrams.py`.

Automata (30 points)

Please implement the following functions in a file titled `automata.py`.

```
def create_automata(option):
```

This function should take an option (1, 2, or 3) and return an automaton (however you decide to store it, e.g. in a list, in a dictionary, as a graph of some kind). The automata returned must be a DFA. The options should result in automata that recognize the following languages:

1. baa^+
2. $(abba)^+$ or $(baab)^+$
3. zero-ninety nine (Remember from the quiz? Assume each transition can recognize/produce only one word)

```
def generate_language(automata):
```

This function should take an automaton as input (however you decide to store it, e.g. in a list, in a dictionary, as a graph of some kind) and produce a word/utterance in the language (aka a string). This will involve some random choices being made, so make sure to use the `random` package. Assume your input automata is a DFA, so there are no epsilon transitions.

```
def recognize_language(automata, utterance):
```

This function should take an automaton as input (however you decide to store it, e.g. in a list, in a dictionary, as a graph of some kind) as well as an utterance (string). It should return an accept or reject decision. Return 0 for reject and 1 for accept.

Minimum Edit Distance (30 points)

Please implement the following functions in a file titled `edit_distance.py`.

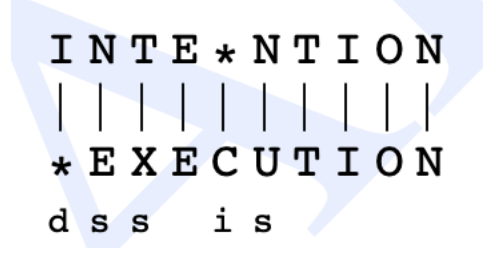
```
def calc_min_edit_dist(source, target):
```

This function should use dynamic programming to calculate the edit distance between a source word and a target word. That is, we are trying to transform the source word into the target

word. The output should be the edit distance, assuming insertion costs 1, deletion costs 1 and substitution costs 2.

```
def align(source, target):
```

This function should use dynamic programming to align a source word and target word. The output of this function should be an alignment between the source and target words. So you should return three lists of characters. The first looks like INTENTION below, with insertions indicated by a *, the second looks like EXECUTION below with deletions indicated by a *, and the third is a list with the operation being performed, like [d s s i s x x x x]. Use x where no operation is performed (that is, a substitution with the same letter).



```
I N T E * N T I O N
| | | | | | | | |
* E X E C U T I O N
d s s i s
```

N-Grams (40 points)

Please implement the following functions in a file titled ngrams.py.

```
def train_ngram(train_data, n):
```

This function should train an n-gram model using the provided training data (list of sentences/utterances). Your function should be able to handle any n from 1 to 3. Your function should return your n-gram model (n-gram probabilities however you want to store them, e.g. a list, a dictionary, etc.)

```
def generate_language(ngram_model, max_words):
```

This function should take a n-gram model as input (however you chose to store it) and a maximum number of words to generate. The function should return a sentence/utterance generated from the n-gram model. This will require sampling from a probability distribution. You are welcome to do this however you like (random.choices provides a solution to this I believe).

```
def calculate_probability(utterance, ngram_model):
```

This function should take an utterance (string) and an n-gram model and output the probability of the utterance given the ngram model.