Assignment 1

1. Tokenization

Task 1.1: Cleaning and Partitioning

Cleaning Process:

- **Unicode Detritus:** Removed non-printable characters using Python's `isprintable()`.
- **Whitespace Normalization:** Replaced all multi-whitespace (tabs, newlines, multiple spaces) with a single space.
- **Alignment:** Stripped leading and trailing whitespace from every sentence to ensure consistent alignment.

Data Partitioning:
The cleaned corpus was split using a fixed seed (`2023111005`) for reproducibility.

- **Training Set:** 80% (Used for learning n-gram counts and BPE merges).
- **Validation Set:** 10% (Reserved for further hyperparameter tuning, but never used it due to lack of time/compute).
- **Testing Set:** 10% (Used only for Task 2.3 perplexity calculation).

Task 1.2: Tokenizer Implementation & Assumptions

1. **Whitespace Tokenizer:**
   - **Method:** Splits text by whitespace.
   - **Assumptions:** Assumed that all punctuations should be treated as separate tokens.
2. **Regex-based Tokenizer:**
   - **Method:** Used the expression `\w+|[^\w\s]`.
   - **Assumptions:** Assumed a token is either a sequence of alphanumeric characters or a single punctuation mark.
3. **BPE (Byte Pair Encoding):**
   - **Method:** Iteratively merged the most frequent character/subword pairs.
   - **Assumptions:** Trained on a subset of {100000,800000,50000} lines with {500,1000,3000,5000} merges. Assumed an end-of-word marker `</w>` is necessary to prevent subwords from merging across word boundaries incorrectly.

Task 1.3: Tokenization Analysis

English Data Token Behaviour

Whitespace Tokenizer:

1. Sense:
   - Where the words are simply separated only by spaces.
   - ex: "...the hotel looks amazing..."
   - **Tokens:** ['the', 'hotel', 'looks', 'amazing']
2. Non-sense:
   - Apostrophe between the words like "I've" and so on.
   - ex:"...What it's like..."
   - **Tokens:** ["it's"]

Regex Tokenizer:

1. Sense:
   - Apostrophe between the words like "I've" and so on.
   - Shows that "'" carries its own importance
   - Also stuff like comma attached to a word, it seperates comma, so comma now has some specific purpose.
   - ex: "B: Its really important..."
   - T**okens:** ['B', ':', 'Its']
2. Non-sense:
   - Date or number handling.
   - ex:"...8.41 grams of carbohydrates."
   - **Tokens:** ['8', '.', '41']

BPE Tokenizer:

1. Sense:
   - Stuff that has prefix/suffix which in itself is repeated a lot and has some meaning.
   - Like unscrew, enlighten ect.
   - ex: "...discovering the courage..." or "...employment and having..."
   - **Tokens:** ['discover', 'ing'] / ['employ', 'ment']
2. Non-sense:
   - Rare words are split nosencically without its subparts carring meanings
   - ex "AKICIML, so I am wondering..."
   - **Tokens:** ['A', 'K', 'IC', 'IM', 'L'] (Estimated)

Mongolian Data Token Behaviour

Giving examples and doing a deeper dive was difficult, but with a cursory search I found out that, Mongolian is a aglutenative language, so there are lot of suffixes and add-ons, so in that case BPE should excel. which is later also shown by the perplexity scores

---

2. Autocomplete and Language Modelling

Task 2.1: 4-gram Language Model Assumptions

- **Greedy Search:** We assume that predicting the most likely next token iteratively is sufficient for sentence completion.
- **Sentence Padding:** Assumed 3 `<s>` start-of-sentence tokens are required to provide context for the first word in a 4-gram model.
- **MLE Constraint:** For the non-smoothed model, if a 3-word context was never seen in training, the model assumes a probability of zero and terminates generation.

Task 2.2: Smoothing Implementation

- **Witten-Bell:** Implemented as a recursive backoff. It uses the number of unique types following a context to estimate the probability of seeing an "unseen" word.
- **Kneser-Ney:** Implemented using absolute discounting ($D = 0.75$) and continuation counts for unigrams to handle how likely a word is to appear in a new context.

Task 2.3: Perplexity Results

Results calculated on $10,000$ test sentences:

Mongolian Dataset

Tokenizer	No Smoothing	Witten-Bell	Kneser-Ney
Whitespace	49,377,498.29	475.47	399.68
Regex	49,377,498.29	475.47	399.68
BPE	22,384.27	21.78	18.50

English Dataset

Tokenizer	Smoothing	Witten-Bell	Kneser-Ney
Whitespace	154,031,639.96	430.53	314.47
Regex	154,031,639.96	430.53	314.47

Tokenizer	Smoothing	Witten-Bell	Kneser-Ney
BPE	596,398.92	46.44	36.45

Task 2.4: Observations

What worked (Successes)

1. **Common Phrases (Kneser-Ney):** Basically, if a 4-gram shows up a ton of times in the training data, the model just memorizes it. For common prompts, the answer is usually exactly what you'd expect because it's seen that sequence so often.
2. **Handling Unseen Prompts (Witten-Bell):** When it hit a rare prompt it hadn't seen in training, Witten-Bell was smart enough to "back off" to lower-grams (like unigrams). Instead of just crashing, it finished the sentence with common words like "the" or ".". (Though this did lead to a weird bug where it sometimes just repeated "The. The. The." over and over.)
3. **Subword Completion (BPE):** BPE is actually great for completing partial words. If it hits a rare word, it uses its subword merges to "guess" a likely ending. The Whitespace tokenizer is pretty useless at this since it only sees whole words.

What didn't work (Failures / Non-sense)

1. **Infinite Loops (All Models):** Every model I built sometimes got stuck in a loop, like "of the of the of the of the". This usually happens when "the" is the most likely word to follow "of the"—it creates a cycle that the greedy search just can't escape from.
2. **Early Termination (No Smoothing):** This was a total fail for the non-smoothed model. On unseen prompts like "Punditry was quick", it just stops immediately. Since that specific 3-word context had a count of 0 in the training set, the model had no idea what to do.
3. **Nonsense Completions (BPE):** Sometimes BPE goes a bit off the rails and generates strings like a . a .). This "non-sense" happens when the model is confused and backs off to super frequent punctuation tokens that don't actually fit the sentence.

Which smoothing was better?

Kneser-Ney definitely beat Witten-Bell in my tests. Because it uses continuation counts, it doesn't over-predict common words that only show up in a few specific places. It makes the autocompletion feel a lot more "natural" and way less repetitive.

Assumptions:

Whitespace

Separate words by whitespace, but treat punctuation as independent tokens.

- **Regex Pattern** `([^\w\s])` : Matches any character that is **not** a word character (`\w` : a-z, A-Z, 0-9, _) and **not** whitespace (`\s`).
- **Substitution** `r' \1 '` : Surrounds each punctuation mark with spaces (e.g., "Hello, world!" → "Hello , world !").
- **Final split()**: Splits on whitespace to get tokens.

Example:

```
Input:  "Don't go to U.S.A!"
Output: ['Don', "'", 't', 'go', 'to', 'U', '.', 'S', '.', 'A', '!']
```

Assumption Made: Punctuation provides no semantic value beyond word boundaries, so it should always be separated for whitespace.

Regex

Design Assumption: Match word sequences OR individual punctuation/special characters.

- **Regex Pattern** `\w+|[^\w\s]` : Match either:
  - `\w+` : One or more word characters (greedy matching of consecutive letters/digits/underscores)
  - `|` : OR
  - `[^\w\s]` : A single punctuation/special character
- **findall()**: Returns all non-overlapping matches.

Example:

```
Input:  "Don't go to U.S.A!"
Output: ['Don', "'", 't', 'go', 'to', 'U', '.', 'S', '.', 'A', '!']
```

PS: I have tried saving the models after training into .pkl files and also the mergings of BPE in json, those will be up in the following link, I might add a few bigger models and stuff later too!

PPS: More Validation examples, and pictures of Perplexity tables will be put up to. (after training of larger models)

PPPS: .pkl files and BPE.jsons are present at Google Drive