ELL884: Quiz 1

Answer the questions in the space provided. No partial marks will be awarded so write in detail.

Name: _____ Entry Number: _____

Total Marks: 50 **Time: 60 minutes** **Mon, Feb 17, 2025**

Question 1: RegEx

1. Suppose you have to create your own command line argument parser. You want to extract key-value pairs with optional fields. Write a single regex that captures username, action, and success if present, each as a group. Examples are given as, **(2 marks)**

```
user=Alpha action=login success=1
user=Beta action=download
user=Charlie success=0
user=Delta success=0 action=upload
```

```
"user=(\w+)(?:\s+action=(\w+))?(?:\s+success=(\d+))?"
```

2. You want to accept strings like `2.3e4`, `-4.56E+2`, `3.14E-10`, `3e3` but reject invalid forms such as `2.3e`, `.e4`, or `3.14.15e2`. Write regex to capture this. **(1 mark)**

```
"^[+-]?(\d+(\.\d+)?)([eE][+-]?\d+)$"
```

3. Construct a regex pattern that correctly matches all valid English plural nouns while avoiding false positives like *is*, and *has*. Explore all edge cases and justify your choices. **(2 marks)**

```
"\b(?:\w*(?:[^sxyzaeiou])s|(?:[cs]h|[zx])es|[^aeiou]ies|feet|teeth|geese|mice|men|women|
children|oxen|data|phenomena|criteria|indices|analyses|ellipses)\b"
```

- **Excludes common non-noun words:** the, is, has
- **Avoids singular words with misleading endings:** bus, hiss, miss
- **Includes irregular plurals explicitly, covering cases that don't follow standard rules.**

Missing: alumni (from alumnus), cacti (from cactus), fungi (from fungus), nuclei (from nucleus), phenomena (from phenomenon), criteria (from criterion), and automata (from automaton).

Question 2: N-grams

1. Given a vocabulary size V and a corpus of size N, derive the number of parameters required to train an n-gram model. **(2 marks)**

   $V^{N-1} \times (V-1)$

2. Suppose you have two corpora: - **Corpus A** (in-domain, small) - **Corpus B** (out-of-domain, large). You suspect that your test data is more similar to Corpus A, but you do not have enough training data in Corpus A to obtain reliable N-gram counts. Devise a mixture approach to combine bigram estimates from both corpora. How do you determine the mixture weight α using expectation maximization or cross-validation? **(3 marks)**

   Given bigrams $P(w_i|w_{i-1})$ estimated from both corpora, we define a mixture approach, $P_{\text{mix}}(w_i|w_{i-1}) = \alpha P_A(w_i|w_{i-1}) + (1-\alpha)P_B(w_i|w_{i-1})$

   EM Algorithm for α Estimation

   E-Step: Compute Expected Probabilities For each bigram (w_{i-1}, w_i), compute the probability contribution from each corpus:

$$\gamma_i = \frac{\alpha P_A(w_i|w_{i-1})}{\alpha P_A(w_i|w_{i-1}) + (1-\alpha)P_B(w_i|w_{i-1})}$$

   where γ_i represents the probability that the bigram (w_{i-1}, w_i) is drawn from Corpus A.

   M-Step: Update α Update α based on the sum of γ_i:

$$\alpha = \frac{1}{N} \sum_{i=1}^{N} \gamma_i$$

where N is the total number of bigram occurrences in the validation dataset. Repeat until convergence.

Cross-Validation Approach

(a) Partition Corpus A into **training** and **validation** sets.

(b) Train bigram models with different values of α using a **grid search**.

(c) Evaluate **perplexity** on the validation set.

(d) Select the α that minimizes **perplexity**.

3. Assume you have a bigram model over a large vocabulary V. Instead of relying on traditional discrete smoothing (e.g., Laplace or Kneser–Ney), you decide to embed each word into a continuous space $\mathbb{R}^d$ and use a smooth function $f(\mathbf{e}_{w_{n-1}}, \mathbf{e}_{w_n})$ to parametrize $P(w_n \mid w_{n-1})$. Describe how you would guarantee that $P(\cdot \mid w_{n-1})$ forms a valid probability distribution for each w_{n-1} in the vocabulary. In other words, explain how to ensure non-negativity and that the probabilities sum to 1. **(5 marks)**

1. Ensuring a Valid Probability Distribution

To embed a large vocabulary into a continuous space $\mathbb{R}^d$ and define $P(w_n \mid w_{n-1}) = f\big(\mathbf{e}_{w_{n-1}}, \mathbf{e}_{w_n}\big)$ we must ensure that for each fixed w_{n-1}, $\sum_{w_n \in \mathcal{V}} P(w_n \mid w_{n-1}) = 1 \quad \text{and} \quad P(w_n \mid w_{n-1}) \geq 0$

A common strategy is to define

$$P(w_n \mid w_{n-1}) = \frac{\exp\big(\phi(\mathbf{e}_{w_{n-1}}, \mathbf{e}_{w_n})\big)}{\sum_{w' \in \mathcal{V}} \exp\big(\phi(\mathbf{e}_{w_{n-1}}, \mathbf{e}_{w'})\big)},$$

where $\phi(\cdot, \cdot)$ is a real-valued scoring function (MLP). The use of the **softmax** ensures that probabilities sum to 1 and remain nonnegative.

2. Training via Maximum Likelihood

MLE Objective. Given a corpus of (w_{n-1}, w_n) pairs, the maximum likelihood estimator *(or equivalently, minimizing the cross-entropy loss)* is:

$$\mathcal{L}(\Theta) = - \sum_{(w_{n-1}, w_n) \in \text{data}} \log P(w_n \mid w_{n-1}; \Theta),$$

where Θ includes all parameters: the embeddings $\mathbf{e}_w$ and any transformation weights in $\phi(\cdot, \cdot)$.

Question 3: Minimum Edit Distance

1. You are given two strings S_1 and S_2 of length m and n, respectively. The minimum edit distance between them is defined as the minimum number of insertions, deletions, and substitutions required to transform S_1 into S_2. Modify the standard $O(m \cdot n)$ space DP solution to an $O(\min(m, n))$ space-efficient approach. Give your answer as pseudocode. **(3 marks)**

(a) Use two 1D arrays: `prev` (previous row) and `curr` (current row).

(b) Initialize `prev` as the edit distances from an empty string to S_2.

(c) Iterate over S_1, updating `curr` using `prev`.

(d) Swap `curr` and `prev` after each row computation.

```
def min_edit_distance(s1, s2):
    m, n = len(s1), len(s2)
    if m < n:
        s1, s2 = s2, s1
        m, n = n, m
    prev = list(range(n + 1))
    curr = [0] * (n + 1)
    for i in range(1, m + 1):
        curr[0] = i
        for j in range(1, n + 1):
            if s1[i - 1] == s2[j - 1]:
```

```
                curr[j] = prev[j - 1]
            else:
                curr[j] = 1 + min(prev[j], curr[j - 1], prev[j - 1])
        prev, curr = curr, prev
    return prev[n]
```

2. Instead of comparing two linear strings, now you need to compute the edit distance between a string and a Directed Acyclic Graph (DAG). For instance, you need to find the closest match between a DNA sequence and a reference genome represented as a DAG. The DAG represents multiple possible sequences. Your aim is to find the minimum edit distance from a given string to any path in the DAG. Extend the standard DP approach to handle DAG structures. **(5 marks)**

   Instead of comparing two linear strings, we are given:

   - A query string S of length m.
   - A Directed Acyclic Graph (DAG) $G = (V, E)$, where each node $v \in V$ contains a character.
   - Each path in G represents a valid string sequence.

   The goal is to compute the **minimum edit distance** between S and any valid path in G.

   1. We define $dp[i][v]$, where:
      - i is the position in the query string S.
      - v is a node in the DAG.
      - $dp[i][v]$ represents the minimum edit distance between the substring $S[0:i]$ and any path ending at node v.
   2. We compute $dp[i][v]$ using three possible operations:

$$dp[i][v] = \min \begin{cases} dp[i-1][u] + (S[i-1] \neq \text{char}(v)), & \forall (u,v) \in E \quad \text{(substitution/match)} \\ dp[i-1][v] + 1, & \text{(insertion in } S\text{)} \\ dp[i][u] + 1, & \forall (u,v) \in E \quad \text{(deletion in DAG)} \end{cases}$$

   3. $dp[0][v]$ is initialized as follows:
      - If v is a starting node in the DAG, set $dp[0][v] = 0$.
      - Otherwise, set $dp[0][v] = \infty$.

   We use **topological sorting** to traverse the DAG efficiently.

   (a) Compute a topological order of the DAG nodes.

   (b) Fill the DP table using the recurrence relation.

   (c) Find the minimum edit distance at any ending node in the DAG.

Question 4: Word Representations

1. Given word embeddings $\mathbf{w}_1$, $\mathbf{w}_2$, ..., $\mathbf{w}_N$ in a high-dimensional space, define a cosine similarity metric and prove that it satisfies the triangle inequality. **(3 marks)**

   Given word embeddings $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_N$ in $\mathbb{R}^d$, the cosine similarity between two vectors $\mathbf{w}_i$ and $\mathbf{w}_j$ is:

$$\cos(\theta) = \frac{\mathbf{w}_i \cdot \mathbf{w}_j}{\|\mathbf{w}_i\| \|\mathbf{w}_j\|}$$

   where θ is the angle between $\mathbf{w}_i$ and $\mathbf{w}_j$. The cosine distance is: $d(\mathbf{w}_i, \mathbf{w}_j) = 1 - \cos(\theta)$.

   We need to show: $d(\mathbf{w}_i, \mathbf{w}_k) \leq d(\mathbf{w}_i, \mathbf{w}_j) + d(\mathbf{w}_j, \mathbf{w}_k)$,

   which expands to: $1 - \cos(\theta_{ik}) \leq (1 - \cos(\theta_{ij})) + (1 - \cos(\theta_{jk}))$,

   simplifying to: $\cos(\theta_{ik}) \geq \cos(\theta_{ij}) + \cos(\theta_{jk}) - 1$.

   From the law of cosines in inner product space,

$$\cos(\theta_{ik}) = \frac{\mathbf{w}_i \cdot \mathbf{w}_k}{\|\mathbf{w}_i\| \|\mathbf{w}_k\|}.$$

Applying the triangle inequality for dot products, $\mathbf{w}_i \cdot \mathbf{w}_k \geq \mathbf{w}_i \cdot \mathbf{w}_j + \mathbf{w}_j \cdot \mathbf{w}_k - \|\mathbf{w}_j\|^2$.

Dividing by $\|\mathbf{w}_i\|\|\mathbf{w}_k\|$, we get, $\cos(\theta_{ik}) \geq \cos(\theta_{ij}) + \cos(\theta_{jk}) - 1$.

Thus, $d(\mathbf{w}_i, \mathbf{w}_j)$ satisfies the triangle inequality, proving it is a valid metric.

2. In word2vec negative sampling, frequent words are sampled with probability proportional to $P(w)^{\frac{3}{4}}$ rather than $P(w)$. Prove that this sampling reduces the variance of gradient estimates. **(5 marks)**

The loss function for a positive (correct) word pair (w_t, w_c) is:

$$\mathcal{L}_+ = -\log \sigma(v_{w_c}^T v_{w_t})$$

For a **negative sample** w_{neg}, the loss function is:

$$\mathcal{L}_- = -\sum_{w_{\text{neg}}} \log \sigma(-v_{w_{\text{neg}}}^T v_{w_t})$$

where Negative samples w_{neg} are drawn from the distribution $P_{\text{neg}}(w)$.

The total loss is: $\mathcal{L} = \mathcal{L}_+ + \mathcal{L}_-$

The gradient update for a negative sample w_{neg} is:

$$\frac{\partial \mathcal{L}_-}{\partial v_{w_t}} = \sum_{w_{\text{neg}}} \sigma(-v_{w_{\text{neg}}}^T v_{w_t}) v_{w_{\text{neg}}}$$

The variance of this estimate depends on the **expected squared norm** of sampled embeddings:

$$\mathbb{E}[\|v_{w_{\text{neg}}}\|^2] = \sum_w P_{\text{neg}}(w)\|v_w\|^2$$

If we use **word frequency** $P(w)$, frequent words dominate the sampling, leading to

- Over-representation of common words (e.g., "the", "of", "and").
- High variance in gradient updates, since rare words are rarely sampled.

Instead, using $P(w)^{3/4}$ **smooths the distribution**, balancing frequent and rare words.

Why Does $P(w)^{3/4}$ Reduce Variance? We analyze the variance using **importance sampling**:

$$\text{Var}(\hat{g}) = \mathbb{E}[\|g\|^2] - (\mathbb{E}[g])^2$$

If $P_{\text{neg}}(w) = P(w)$, then: $\text{Var}(g) \propto \sum_w P(w)\|v_w\|^2 - \left(\sum_w P(w)\|v_w\|\right)^2$

If $P_{\text{neg}}(w) = P(w)^{\frac{3}{4}}$, then: $\text{Var}(g) \propto \sum_w P(w)^{\frac{3}{4}}\|v_w\|^2 - \left(\sum_w P(w)^{\frac{3}{4}}\|v_w\|\right)^2$

By Jensen's inequality, using $P(w)^{\frac{3}{4}}$ reduces the variance because it smooths the sampling probabilities, making them less extreme. This results in more balanced sampling and prevents frequent words from dominating updates.

- If we sample from $P_{\text{neg}}(w) = P(w)$, rare words contribute disproportionately to variance.
- If we **smooth** the distribution with $P(w)^{3/4}$, the variance is minimized.

Question 5: HMMs

1. Traditional HMMs assume discrete observation distributions $P(o_t|q_t) = B(q_t, o_t)$, where $B(q_t, o_t)$ represents the emission probability of observation o_t from state q_t. However, in many real-world applications (e.g., speech recognition), observations are continuous values, which limits the application of HMMs. Therefore, modify the standard HMM to support continuous emissions using Gaussian Mixture Models. The Gaussian probability density function is given as,

$$\mathcal{N}(o_t|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(o_t - \mu)^T \Sigma^{-1}(o_t - \mu)\right),$$

where d is the observation dimensionality, μ is the mean vector, Σ is the covariance matrix. Given that $c_{q_t}^{(i)}$ is the mixing weight for the i-th Gaussian, satisfying $\sum_i c_{q_t}^{(i)} = 1$, compute the formulae of new emission probability and

derive new formulations for the forward and Viterbi algorithms, with initialization, recursion, and termination conditions clearly stated. **(10 marks)**

Write complete and exact answers with correct notations, no marks will be provided even if your answer is "closely accurate".

Instead of using a discrete probability table $B(q, o)$, we assume that each state q_t generates observations from a mixture of Gaussians:

$$P(o_t|q_t) = \sum_{m=1}^{M} c_{q_t}^{(m)} \mathcal{N}(o_t|\mu_{q_t}^{(m)}, \Sigma_{q_t}^{(m)})$$

where,

- M is the number of Gaussian components.
- $c_{q_t}^{(m)}$ is the mixing weight for the m-th Gaussian in state q_t, satisfying $\sum_m c_{q_t}^{(m)} = 1$.
- $\mathcal{N}(o_t|\mu, \Sigma)$ is the Gaussian probability density function (PDF):

$$\mathcal{N}(o_t|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(o_t - \mu)^T \Sigma^{-1}(o_t - \mu)\right)$$

  where,

  - d is the observation dimensionality.
  - μ is the mean vector.
  - Σ is the covariance matrix.

Thus, the new emission probability in GMM-HMMs is:

$P(o_t|q_t) = \sum_{i=1}^{M} c_{q_t}^{(i)} \frac{1}{(2\pi)^{d/2}|\Sigma_{q_t}^{(i)}|^{1/2}} \exp\left(-\frac{1}{2}(o_t - \mu_{q_t}^{(i)})^T {\Sigma_{q_t}^{(i)}}^{-1}(o_t - \mu_{q_t}^{(i)})\right)$

The **Forward algorithm** computes the probability of observing the sequence $O = (o_1, o_2, ..., o_T)$ given the HMM parameters.

Initialization: For each state q_j:

$$\alpha_1(j) = \pi_j P(o_1|q_j),$$

where:

- π_j is the **initial probability** of state q_j.
- $P(o_1|q_j)$ is the **GMM emission probability**.

Recursion: For $t = 2$ to T:

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} P(o_t|q_j),$$

where:

- a_{ij} is the transition probability from state q_i to q_j.
- $P(o_t|q_j)$ is computed using the **GMM emission probability**.

Termination: The probability of the observation sequence is:

$$P(O|\lambda) = \sum_{j=1}^{N} \alpha_T(j).$$

The **Viterbi algorithm** finds the most probable sequence of hidden states given the observations.

Initialization: For each state q_j:

$$\delta_1(j) = \pi_j P(o_1|q_j),$$
$$\psi_1(j) = 0.$$

Recursion: For $t = 2$ to T:

$$\delta_t(j) = \max_i \left(\delta_{t-1}(i) a_{ij}\right) P(o_t|q_j),$$
$$\psi_t(j) = \arg\max_i \left(\delta_{t-1}(i) a_{ij}\right).$$

Termination:

$$P^* = \max_j \delta_T(j), \quad q_T^* = \arg\max_j \delta_T(j).$$

Backtracking: For $t = T-1, T-2, ..., 1$:

$$q_t^* = \psi_{t+1}(q_{t+1}^*).$$