UNIVERSIDAD DE LOS ANDES

NATURAL LANGUAGE PROCESSING

ISIS 4221

Tarea 2

Autores:
Isabella MARTÍNEZ
Gustavo MENDEZ
Paula DAZA
Nicolás KLOPSTOCK

Septiembre 18, 2024

Tabla de contenidos

Universidad de los Andes

1 Introducción

Esta tarea aborda la construcción de modelos de lenguaje utilizando N-gramas, específicamente unigramas, bigramas y trigramas, para los datasets 20Newsgroups (20N) y Blog Authorship Corpus (BAC). Los modelos generados utilizan suavizado de Laplace para evitar probabilidades nulas, y se evaluarán a través de la métrica de *perplejidad* sobre conjuntos de prueba separados. La implementación de los modelos también incluye una función para generar texto automáticamente basado en los N-gramas.

2 Construcción de Modelos de Lenguaje

En esta sección, se detalla la construcción de modelos de lenguaje N-grama utilizando dos conjuntos de datos: `20N` y `BAC`. El proceso involucró la consolidación de documentos, tokenización, normalización, y finalmente la creación de los modelos unigramas, bigramas y trigramas utilizando suavizado de Laplace.

2.1 Construcción de Archivos Consolidados

El primer paso fue consolidar todos los documentos en un solo archivo para cada conjunto de datos. Esto facilita el procesamiento en etapas posteriores. Los archivos consolidados generados fueron `consolidated_news.txt` para el conjunto `20N` y `consolidated_bac.txt` para el conjunto `BAC`.

Los archivos pueden consultarse a través de los siguientes enlaces:

- consolidated_news.txt
- consolidated_bac.txt

2.2 Tokenización y Normalización

A continuación, se realizó la tokenización de los textos. Para esto, el notebook implementa la función `tokenize_sentences`, que toma los textos consolidados y los tokeniza por oración, asegurando las siguientes transformaciones:

- Normalización del texto (minúsculas, etc.).
- Reemplazo de números con el token `NUM`.
- Inclusión de etiquetas de inicio y fin de oración (`<s>` y `</s>`).
- Modelado de tokens con frecuencia unitaria como `<UNK>`.

Este proceso permite una representación más uniforme de los datos, lo cual es esencial para la construcción de los modelos de lenguaje. La siguiente función muestra un fragmento condensado de cómo se implementó el preprocesamiento:

```
def preprocess_text(file_path: str) -> List[str]:
    sentences = sent_tokenize(text.lower())
    tokens = ['<s>'] + tokens + ['</s>']
    token_freq = Counter(tokens)
    final_tokens = [token if token_freq[token] > 1 else '<UNK>' for token in tokens]
    return final_tokens
```

Esta función se aplica a los archivos `consolidated_news.txt` y `consolidated_bac.txt`, generando oraciones tokenizadas y preprocesadas que se guardan en los archivos de salida:

- processed_news.txt
- processed_bac.txt

2.3 División del Conjunto de Datos

Posteriormente, se divide el corpus preprocesado en conjuntos de entrenamiento y prueba en una proporción 80-20, mediante la función `generate_training_testing`. La siguiente función muestra un fragmento de cómo se implementó la división:

```
def generate_training_testing(final_sentences: List[str], filename: str) -> None:
    sents = final_sentences.copy()
    random.shuffle(sents)

    split_index = int(0.8 * len(sents))

    training_sentences = sents[:split_index]
    testing_sentences = sents[split_index:]

    with open(f'{filename}_training.txt', 'w', encoding='utf-8') as training_file:
        for sentence in training_sentences:
            training_file.write(' '.join(sentence) + '\n')

    with open(f'{filename}_testing.txt', 'w', encoding='utf-8') as testing_file:
        for sentence in testing_sentences:
            testing_file.write(' '.join(sentence) + '\n')
```

Se aplica esta función para ambos conjuntos de datos. Los archivos generados son los siguientes:

- 20N_training.txt
- 20N_testing.txt
- BAC_training.txt
- BAC_testing.txt

2.4 Construcción de los Modelos N-Gram

Finalmente, se construyeron modelos de unigramas, bigramas y trigramas para ambos conjuntos de datos utilizando suavizado de Laplace. Esto garantiza que se asignen probabilidades no nulas a los N-gramas no vistos en los datos de entrenamiento, en otras palabras, para manejar las combinaciones de palabras raras o no observadas. Se muestra un fragmento de código de cómo se construyeron los modelos N-Gram:

```
def calculate_ngram_probabilities(tokenized_sentences: List[List[str]], n: int)
    -> Dict[Tuple[str, ...], float]:
```

```
    ngram_counts = build_ngram_counts(tokenized_sentences, n)
    n_minus_1_gram_counts = build_ngram_counts(tokenized_sentences, n-1) if n > 1 else None

    vocabulary = set(token for sentence in tokenized_sentences for token in sentence)
    vocab_size = len(vocabulary)

    ngram_probabilities = {}

    for ngram, count in tqdm(ngram_counts.items(), desc=f"Calculando {n}-gramas"):
        if n == 1:
            total_count = sum(ngram_counts.values())
            ngram_probabilities[ngram] = (count + 1) / (total_count + vocab_size)
        else:
            priori = ngram[:-1]
            priori_count = n_minus_1_gram_counts[priori]
            ngram_probabilities[ngram] = (count + 1) / (priori_count + vocab_size)

    return ngram_probabilities
```

Los modelos se crean para unigramas, bigramas y trigramas tanto para el conjunto de datos `20N` como para `BAC`. Cada modelo es guardado en un archivo `.pkl` para su posterior uso. Los archivos generados contienen las probabilidades asociadas a cada n-grama y se pueden acceder desde los siguientes enlaces:

- 20N_unigramas.pkl
- 20N_bigramas.pkl
- 20N_trigramas.pkl
- BAC_unigramas.pkl
- BAC_bigramas.pkl
- BAC_trigramas.pkl

2.5 Justificación de la Eficiencia

El uso de suavizado de Laplace permite que incluso los N-gramas no observados tengan una probabilidad distinta de cero, lo que evita problemas de subestimación en los modelos. Aunque esta estrategia es eficiente para conjuntos de datos de tamaño moderado, en casos de gran escala podrían considerarse enfoques más avanzados, como modelos basados en redes neuronales o *transformers*.

3 Evaluación del Modelo de Lenguaje

En esta sección se evalúan los modelos de lenguaje construidos en las etapas anteriores. Se abordarán los siguientes aspectos:

1. Cálculo de la perplejidad de los modelos.
2. Generación automática de oraciones usando el mejor modelo de lenguaje.

3.1 Cálculo de la Perplejidad

La perplejidad es una métrica comúnmente utilizada en modelos de lenguaje, y mide cuán bien un modelo es capaz de predecir una secuencia de palabras. Un valor más bajo de perplejidad indica que el modelo es más eficiente en la predicción de secuencias de palabras, mientras que un valor alto indica un peor desempeño. La fórmula para calcular la perplejidad de un modelo N-gramas es la siguiente:

$$PP(S) = \left(\prod_{i=1}^{T} P\left(w_i \mid w_{i-n} \dots w_{i-1}\right) \right)^{-1/T}$$

Sin embargo, debido a las bajas probabilidades, se utilizó el espacio logarítmico para evitar problemas de desbordamiento de variables. La fórmula en espacio logarítmico es la siguiente:

$$PP(S) = \exp\left[-\frac{1}{T} \sum_{i=1}^{T} \log\left(P\left(w_i \mid w_{i-n} \dots w_{i-1}\right)\right) \right]$$

A continuación, se muestra la implementación en Python utilizada para calcular la perplejidad, adaptada para operar en espacio logarítmico:

```
def calculate_perplexity_log(model: Dict[Tuple[str, ...], float],
test_sentences: List[List[str]], n: int) -> float:

    total_log_prob = 0
    total_ngrams = 0

    for sentence in test_sentences:
        sentence_log_prob = 0
        n_grams = list(nltk.ngrams(sentence, n))
        for ngram in n_grams:
            prob = model.get(ngram, 1e-10)
            sentence_log_prob += math.log(prob)
        total_log_prob += sentence_log_prob
        total_ngrams += len(n_grams)
    avg_log_prob = total_log_prob / total_ngrams

    return math.exp(-avg_log_prob)
```

3.1.1 Resultados de Perplejidad

En este análisis, hemos calculado la perplejidad para los modelos de unigramas, bigramas y trigramas sobre los datasets `20N` y `BAC`. A continuación, discutimos los resultados obtenidos para cada uno.

Resultados para el Dataset 20N

- **Perplejidad del modelo de unigramas**: 967.68
- **Perplejidad del modelo de bigramas**: 11,460.07
- **Perplejidad del modelo de trigramas**: 1,456,686.47

Estos resultados muestran una tendencia esperada: los modelos de unigramas tienen una perplejidad relativamente baja en comparación con los bigramas y trigramas. Esto ocurre porque los unigramas no dependen de contextos anteriores, simplemente predicen la próxima palabra según su frecuencia. Aunque este enfoque es sencillo, puede ser ineficiente ya que no considera las relaciones entre palabras, lo que aumenta la capacidad de generalización del modelo.

El modelo de bigramas, al incorporar el contexto inmediato de la palabra anterior, tiene una perplejidad significativamente mayor. Este incremento en la perplejidad refleja la mayor complejidad de predecir palabras en función de su contexto local. Sin embargo, como los bigramas solo consideran una palabra de contexto, aún pueden presentar limitaciones en su capacidad de capturar secuencias largas.

El modelo de trigramas presenta la mayor perplejidad, lo que era esperado debido a que intenta predecir una palabra en función de las dos palabras anteriores. Este aumento drástico en la perplejidad sugiere que el modelo de trigramas está sobreajustado, es decir, aprende a capturar patrones muy específicos del conjunto de entrenamiento, lo que lleva a un mal desempeño en los datos de prueba. La gran cantidad de combinaciones posibles de palabras con contextos de dos palabras incrementa la dificultad del modelo para predecir secuencias no vistas en el conjunto de entrenamiento.

Resultados para el Dataset BAC

- **Perplejidad del modelo de unigramas**: 763.64
- **Perplejidad del modelo de bigramas**: 1,879.84
- **Perplejidad del modelo de trigramas**: 216,898.71

Los resultados para el dataset `BAC` muestran un comportamiento similar al observado en el dataset `20N`. El modelo de unigramas tiene una perplejidad relativamente baja, lo que refleja su simplicidad y su enfoque basado en la frecuencia de palabras individuales.

El modelo de bigramas aumenta considerablemente la perplejidad en comparación con el modelo de unigramas, ya que ahora intenta predecir una palabra basada en el contexto de la palabra anterior. Este modelo captura mejor las dependencias locales entre palabras, pero su capacidad se limita a secuencias cortas, lo que explica el incremento en la perplejidad.

Finalmente, el modelo de trigramas tiene una perplejidad mucho mayor, lo que sugiere nuevamente un sobreajuste en los datos de entrenamiento. La combinación de dos palabras anteriores para predecir la siguiente palabra genera una cantidad muy grande de posibles trigramas, lo que dificulta la predicción precisa de secuencias en los datos de prueba.

Interpretación General de los Resultados

En general, la perplejidad aumenta significativamente conforme incrementamos el tamaño del n-grama (de unigramas a trigramas). Esto se debe a que los modelos más complejos requieren más contexto para realizar predicciones, pero también tienen una mayor probabilidad de encontrarse con secuencias que no vieron en el conjunto de entrenamiento, lo que eleva la perplejidad. Este aumento en la perplejidad sugiere que los modelos de trigramas pueden no generalizar bien fuera de los datos de entrenamiento, lo que indica un potencial sobreajuste.

Es importante mencionar que, aunque los modelos de n-gramas más grandes capturan más contexto, también son más susceptibles a la escasez de datos, lo que contribuye a los resultados de alta perplejidad.

3.2 Generación de Texto

La generación de texto con modelos de lenguaje es una tarea desafiante, especialmente con modelos de n-gramas, que están limitados por la cantidad de contexto que pueden utilizar. A continuación, se detallan los resultados de la generación de texto utilizando el modelo de trigramas del dataset `BAC`, ya que mostró un buen desempeño en la generación de secuencias coherentes.

La estrategia utilizada para la generación de texto consiste en tomar una palabra inicial como entrada, y luego generar la secuencia siguiente utilizando el contexto del modelo de trigramas. Si no se encuentra un contexto adecuado o si se alcanza el token de fin de oración (`</s>`), la generación se detiene.

Ejemplos de Generación de Texto

- **Entrada:** `<s>`
  - **Salida:** `<s> s NUM NUM NUM NUM NUM NUM NUM NUM NUM NUM ...`

  En este ejemplo, el modelo genera una secuencia repetitiva de tokens NUM (números), lo que sugiere que está sobreajustado a la estructura numérica común en el dataset. Este tipo de secuencias es común cuando se utilizan tokens de placeholder como NUM.

- **Entrada:** `<s> this`
  - **Salida:** `<s> this is the best of all the time`

  En este caso, el modelo genera una oración que es coherente gramaticalmente, aunque el contenido semántico puede parecer un poco genérico. Esto sugiere que el modelo ha aprendido estructuras comunes, como las secuencias que comienzan con `this is`.

- **Entrada:** `how do`
  - **Salida:** `how do you think you re not going to be a good thing`

  Esta oración tiene una estructura válida, aunque el contenido carece de sentido en algunos puntos. Sin embargo, demuestra que el modelo ha aprendido las relaciones entre palabras comunes en preguntas o expresiones como `how do you think`.

- **Entrada:** `i think that`
  - **Salida:** `i think that i m not sure if i can t wait to see the movie`

  Esta es una oración bastante coherente y natural, lo que sugiere que el modelo ha capturado correctamente algunas estructuras comunes de oraciones. La secuencia tiene sentido y fluye bien, mostrando que el modelo de trigramas es capaz de generar oraciones útiles en ciertos contextos.

Resumen sobre la Generación de Texto

El modelo de trigramas genera oraciones que son, en su mayoría, coherentes gramaticalmente. Sin embargo, como se puede observar en algunos ejemplos, el modelo tiende a generar contenido repetitivo o sin sentido cuando se encuentra con estructuras menos comunes. Esto es indicativo de que los modelos de n-gramas, aunque efectivos para capturar dependencias locales, tienen limitaciones para generar contenido novedoso y complejo debido a la falta de comprensión semántica y a su dependencia de patrones vistos en el conjunto de entrenamiento.

En general, el modelo de trigramas del dataset `BAC` mostró un desempeño aceptable en la generación de oraciones, aunque con algunas limitaciones. Es importante considerar el balance entre simplicidad y capacidad de generalización en futuros desarrollos de modelos de lenguaje más avanzados.

Universidad de los Andes

4 Conclusión

En este trabajo, hemos abordado la construcción y evaluación de modelos de lenguaje N-gram utilizando los conjuntos de datos 20 Newsgroups (20N) y Blog Authorship Corpus (BAC). A lo largo de las distintas etapas, implementamos procesos clave como la limpieza y consolidación de los textos, la creación de representaciones N-gram, y la evaluación mediante la métrica de perplejidad.

Primero, se realizó la preparación de los textos mediante técnicas de preprocesamiento como la normalización y tokenización, asegurando la correcta estructuración para los modelos de lenguaje. Luego, se dividieron los datos en conjuntos de entrenamiento y prueba, lo que permitió construir y evaluar modelos de unigramas, bigramas y trigramas.

Los resultados obtenidos en la métrica de perplejidad mostraron una clara tendencia: conforme se incrementa el tamaño del n-grama, la perplejidad del modelo también aumenta. Este fenómeno se debe a la mayor complejidad de los modelos de trigramas y bigramas, que requieren un contexto más extenso para predecir con precisión el siguiente token. Sin embargo, la presencia de datos insuficientes o contextos poco frecuentes en los conjuntos de prueba contribuye al incremento de la perplejidad.

En la generación automática de texto, observamos que los modelos trigramas tienden a generar oraciones más coherentes cuando se les proporciona un buen contexto inicial. No obstante, también es evidente que estos modelos tienden a repetir secuencias de tokens comunes, debido a la naturaleza de los datos de entrenamiento y las limitaciones de los modelos N-gram. A pesar de estas limitaciones, los modelos lograron producir frases razonables que reflejan el estilo del corpus de entrenamiento.

En resumen, el uso de modelos de lenguaje N-gram es efectivo para tareas de procesamiento de lenguaje natural donde el contexto inmediato es relevante, aunque su rendimiento puede verse limitado en escenarios más complejos donde se requieren dependencias a largo plazo. Los resultados demuestran la importancia del balance entre la simplicidad de los modelos y la cantidad de datos disponibles para su entrenamiento. Para futuras investigaciones, sería interesante explorar modelos más avanzados, como las redes neuronales, que pueden superar estas limitaciones y mejorar los resultados en tareas de generación de texto y comprensión de lenguaje natural.