# ÍNDICE

# INTRODUCCIÓN

Es importante hablar primeramente de los desafíos que implica implementar un procesador natural de lenguaje (NLP) por sus siglas en inglés.

## Dificultades del procesamiento del lenguaje natural

El procesamiento del lenguaje natural (PLN) es un campo de estudio que se ocupa de la interacción entre las computadoras y el lenguaje humano. Aunque avances significativos se han logrado en esta área, todavía existen desafíos importantes en el procesamiento del lenguaje natural en España.

**Ambigüedad:** El lenguaje humano es inherentemente ambiguo y puede tener múltiples interpretaciones. Esto dificulta la tarea de las máquinas para comprender adecuadamente el significado de las palabras y las oraciones.

**Contexto cultural y regional:** El procesamiento del lenguaje natural en español tiene sus propios desafíos debido a las diferencias culturales y regionales. Las variaciones en el uso del lenguaje, la jerga local y las expresiones idiomáticas complican el desarrollo de modelos de PLN que sean precisos y efectivos en el contexto español.

**Variedad de dialectos:** El español se habla en muchos países, y cada uno tiene sus propias variaciones y dialectos. La comprensión y generación de texto en diferentes dialectos puede ser difícil para los sistemas de PLN, ya que necesitan estar entrenados en esos dialectos específicos.

**Datos insuficientes:** Aunque el español es uno de los idiomas más hablados del mundo, la disponibilidad de datos etiquetados y corpus de entrenamiento para el PLN en español es limitada en comparación con otros idiomas como el inglés. La falta de datos suficientes puede afectar el rendimiento y la precisión de los modelos de PLN en español.

# TOKENIZADOR MÍNIMO DE PROCESAMIENTO DE LENGUAJE NATURAL

En el desarrollo de un tokenizador mínimo para procesamiento de lenguaje natural (MNLPTK) destinado a la evaluación de llamadas telefónicas, se intentó utilizar una técnica basada en la teoría de autómatas finitos para identificar patrones en el texto. Este enfoque implicaba convertir expresiones regulares (regex) a un Autómata Finito No Determinista (AFN) utilizando la construcción de Thompson, y luego transformar el AFN resultante en un Autómata Finito Determinista (AFD) mediante el algoritmo de subconjuntos.

Aunque esta metodología es teóricamente sólida y ampliamente estudiada en el ámbito de los compiladores y la teoría de autómatas, su aplicación práctica en el contexto específico de este tokenizador resultó ser **menos que ideal**. A continuación, se exploran las razones por las cuales este enfoque no es la mejor opción para el proyecto en cuestión.

**Complejidad Computacional**

La conversión de una regex a un AFN y luego a un AFD implica varios pasos complejos que pueden resultar en un sistema complicado y difícil de manejar:

- **Regex a AFN:** La construcción de Thompson es un proceso relativamente directo, pero puede generar un gran número de estados y transiciones, especialmente cuando se trata de regex complejas que contienen múltiples operadores y estructuras repetitivas.

- **AFN a AFD:** La conversión de un AFN a un AFD utilizando el algoritmo de subconjuntos puede provocar una explosión exponencial en el número de estados. Cada estado del AFD representa un conjunto de estados del AFN, lo que puede llevar a una cantidad enorme de estados en el AFD resultante, especialmente si el AFN original es grande y complejo.

**Legibilidad y Mantenimiento del Código**

El código resultante de la implementación de estos algoritmos puede ser complicado y difícil de mantener:

La legibilidad del código es crucial para el mantenimiento a largo plazo, especialmente en equipos donde los desarrolladores pueden cambiar con el tiempo. Un sistema basado en autómatas finitos tiende a ser menos legible y más difícil de comprender que uno basado en regex directamente.

Utilizar regex directamente en el código es más simple y fácil de entender. Las expresiones regulares permiten expresar patrones de texto de manera concisa y clara, facilitando su modificación y extensión en el futuro.

**Flexibilidad y Adaptabilidad**

Si se requiere agregar nuevos patrones o modificar los existentes, hacerlo directamente con regex es mucho más sencillo y directo:

Simplemente se actualiza la expresión regular en lugar de tener que reconstruir completamente el AFN y el AFD, lo cual es un proceso costoso y propenso a errores.

Las regex proporcionan una forma flexible y poderosa de definir patrones complejos de manera concisa, permitiendo adaptaciones rápidas y eficientes.

**Extensiones y Mejoras**

Añadir funcionalidades adicionales o mejorar la precisión del análisis es más directo cuando se usan regex:

La flexibilidad de las regex permite expresar patrones complejos de manera concisa, facilitando la extensión del sistema para manejar nuevos tipos de tokens o mejorar la precisión del análisis.

En un sistema basado en AFN y AFD, cada mejora podría requerir cambios significativos en la estructura del autómata, lo que implica un rediseño y una reimplementación compleja.

# ENFOQUE ADOPTADO

Utilizar regex directamente para identificar patrones en el texto es más eficiente y fácil de implementar por las siguientes razones:

- Las bibliotecas modernas de regex en lenguajes de programación como Python están altamente optimizadas y pueden manejar una amplia gama de patrones de manera eficiente.
- Las regex proporcionan una forma declarativa y concisa de definir patrones de texto, lo que simplifica tanto la escritura como el mantenimiento del código.
- Las expresiones regulares se ajustan al programa de estudios de la materia de Diseño de Compiladores, por lo tanto, elegir este enfoque no representaría una opción descabellada para implementar el tokenizador

# EXPLICACIÓN DEL ALGORITMO

## Paso 1: Definición de Funciones Auxiliares

- Se declara la función `generar_variaciones(palabra)`:
  - **Entrada:** Una palabra en español.
  - **Salida:** Una lista de variaciones de la palabra considerando género (masculino, femenino) y número (singular, plural).
  - **Lógica:**

    Si la palabra termina en "o":

        Añadir variaciones terminadas en "a", "os", "as".

    Si la palabra termina en "a":

        Añadir variaciones terminadas en "o", "as", "os".

    Si la palabra termina en "e":

        Añadir la variación terminada en "es".

    Si la palabra termina en "l", "n" o "r":

        Añadir la variación terminada en "es".

    Para otras terminaciones:

        Añadir la variación terminada en "s".

- Se declara la función `generar_listas_variaciones(lista_palabras)`:
  - **Entrada:** Una lista de palabras en español.
  - **Salida:** Una lista de palabras con todas las variaciones de género y número, eliminando duplicados.
  - **Lógica:**

    Para cada palabra en la lista de entrada:

        Generar sus variaciones usando generar_variaciones.

        Añadir las variaciones a una lista.

        Eliminar duplicados usando set.

## Paso 2: Generación de Listas de Palabras

**Generación de listas de palabras positivas, negativas y neutras:**

- **Lógica:**

  Usar generar_listas_variaciones para obtener todas las variaciones de género y número para las listas de palabras originales.

- **Listas Generadas:**

  palabras_positivas

  palabras_negativas

  palabras_neutras

## Paso 3: Definición de Expresiones Regulares

**Crear patrones de expresiones regulares:**

- **Lógica:**

Usar las listas de palabras generadas para crear patrones de expresiones regulares que coincidan con cualquier variación de las palabras en minúsculas y sin acentos. Se utiliza una extensión de las expresiones regulares aprendidas en las clases de Diseño de Compiladores.

- **Tokens Definidos:**
  - saludos_bienvenida
  - despedidas
  - identificación
  - expresiones_positivas
  - expresiones_negativas
  - expresiones_neutras
  - preguntas_ayuda

## Paso 4: Preprocesamiento del Texto

- **Función preprocesar_texto(texto):**
  - **Entrada:** Texto en español.
  - **Salida:** Texto en minúsculas y sin acentos.
  - **Lógica:**

    Convertir todo el texto a minúsculas.

    Reemplazar las letras acentuadas por sus equivalentes sin acento.

## Paso 5: Evaluación del Texto del Funcionario

- **Función evaluar_funcionario(texto):**
  - **Entrada:** Texto del funcionario.
  - **Salida:** Puntuación del funcionario y detalles de los elementos detectados.
  - **Lógica:**

    Preprocesar el texto.

    Inicializar puntuación y detalles.

    Comprobar presencia de saludo, despedida, solicitud de identificación, expresiones positivas y preguntas de ayuda usando expresiones regulares.

    Añadir puntos y detalles según correspondan.

    Ajustar la puntuación a una escala del 1 al 5.

## Paso 6: Evaluación del Texto del Cliente

- **Función evaluar_cliente(texto):**
  - **Entrada:** Texto del cliente.
  - **Salida:** Puntuación del cliente, palabras buenas, malas y neutras encontradas.
  - **Lógica:**

    Preprocesar el texto.

    Inicializar puntuación en 5.

    Encontrar y contar palabras positivas, negativas y neutras usando expresiones regulares.

    Ajustar la puntuación sumando palabras buenas y restando palabras malas.

    Ajustar la puntuación a una escala del 1 al 5.

## Paso 7: Calificación de la Experiencia

- Función calificar_experiencia(puntuacion):
  - **Entrada:** Puntuación (entero).
  - **Salida:** Calificación (positiva, neutra, negativa).
  - **Lógica:**

    Si la puntuación es mayor a 3: "positiva".

    Si la puntuación es igual a 3: "neutra".

    Si la puntuación es menor a 3: "negativa".

## Paso 8: Preguntar al Usuario sobre Palabras Neutras

- Función preguntar_incluir_palabra_neutra(palabra):
  - **Entrada:** Una palabra neutra.
  - **Salida:** Booleano indicando si se debe incluir la palabra como negativa.
  - **Lógica:**

    Preguntar al usuario si desea incluir la palabra como negativa.

## Paso 9: Evaluación Principal

- Función `evaluar_llamada(archivo_funcionario, archivo_cliente)`:
  - **Entrada:** Archivos de texto del funcionario y del cliente.
  - **Salida:** Puntuaciones y calificaciones para el funcionario y el cliente, e impresiones detalladas.
  - **Lógica:**

    Leer los archivos de texto.

    Evaluar el texto del funcionario.

    Evaluar el texto del cliente.

    Imprimir los resultados.

  Preguntar al usuario sobre palabras neutras y recalcular la puntuación del cliente si se añaden palabras neutras como negativas.

# EJEMPLO DE APLICACIÓN

cliente.txt

```
Hola, buenos días. Sí, mi documento es 12345678.

Gracias por la ayuda, pero estoy un poco cansado de tener que llamar tantas veces. La última vez fue un desastre y no solucionaron nada.

Espero que esta vez sí puedan resolverlo, porque estoy harto de esta situación.

Hasta luego.
```

funcionario.txt

```
Buen día, mi nombre es Juan y estoy aquí para ayudarle. ¿Podría proporcionarme su documento o cédula, por favor?

Gracias por esperar. Hemos registrado su solicitud y en breve le daremos una respuesta. ¿Hay algo más en lo que pueda asistirle?

Hasta luego, que tenga un buen día.
```

El funcionario obtiene puntos según los siguientes criterios (o tokens), empezando desde una puntuación de "0":

1. **Saludo de bienvenida detectado**: +1 punto
2. **Despedida detectada**: +1 punto
3. **Solicitud de identificación detectada**: +1 punto
4. **Uso de expresiones positivas detectado**: +1 punto

En el texto del funcionario proporcionado, se detectan todas estas características:

- **Saludo de bienvenida**: "Buen día"
- **Despedida**: "Hasta luego"
- **Solicitud de identificación:** "documento o cédula"
- **Expresiones positivas:** "gracias"

Por lo tanto, la puntuación total es 4/5.

## Evaluación del Cliente

El cliente obtiene puntos en función de las palabras positivas y negativas:

1. Se empieza con una puntuación base de 5.

2. Por cada palabra positiva encontrada, se suma 1 punto.
3. Por cada palabra negativa encontrada, se resta 1 punto.

En el texto del cliente proporcionado:

- **Palabras positivas**: “gracias”, “resolver” (2 palabras positivas)
- **Palabras negativas**: “cansado”, “desastre”, “harto” (3 palabras negativas)

La puntuación se calcula así:

- Puntuación base: 5
- Sumar 2 puntos por las palabras positivas: 5 + 2 = 7
- Restar 3 puntos por las palabras negativas: 7 – 3 = 4

Por lo que la puntuación final es de 4

# CASOS DE PRUEBA

**Prueba 1**

cliente.txt

```
Hola, buenos días.

Si, tengo un problema ahora.

Mi documento es 12345678.

Gracias por la ayuda con mi problema anterior. Pero estoy muy frustrado con el servicio general. La última vez fue un desastre total y no solucionaron nada. Espero que esta vez se pueda resolver. Estoy cansado de tener que llamar tantas veces. No, Hasta luego.
```

funcionario.txt

```
Buen día, mi nombre es María y estoy aquí para ayudarle.

¿Podría proporcionarme su documento o cédula, por favor?

Gracias por esperar. Hemos registrado su solicitud y en breve le daremos una respuesta.

Le agradecemos su paciencia. ¿Hay algo más en lo que pueda asistirle? Hasta luego, que tenga un buen día.
```

**Prueba 2**

Cliente.txt

```
Buenas tardes. Estoy muy molesto con el servicio que he recibido. Ha sido extremadamente lento

y frustrante. Cada vez que llamo, tengo que explicar mi situación desde el principio y nada

parece mejorar. Espero que puedan resolverlo esta vez. Gracias.
```

Funcionario.txt

```
Buenas tardes, mi nombre es Pedro y estoy aquí para asistirle. ¿Podría proporcionarme su documento?

Le agradezco por su paciencia. Hemos registrado su solicitud y en breve recibirá una respuesta.

¿Hay algo más en lo que pueda asistirle hoy? Hasta luego.
```

**Prueba 3**

Cliente.txt

```
Hola, buenas noches. Sí, mi cédula es 98765432. Estoy muy satisfecho con el servicio que he

recibido hasta ahora. Todo ha sido excelente y muy eficiente. Muchas gracias por la ayuda

constante y el buen trato recibido. Hasta luego.
```

Funcionario.txt

```
Buenas noches, soy Carla y estoy aquí para ayudarle. ¿Puede darme su cédula, por favor? Gracias

por esperar. Hemos registrado su solicitud y le daremos una respuesta pronto. Agradecemos su

paciencia y colaboración. Que tenga una buena noche. Hasta luego.
```

**Prueba 4**

Cliente.txt

```
Hola. Mi documento es 45678901. No estoy contento con el servicio en absoluto. Ha sido un

problema constante desde que empecé a usarlo. Cada vez que intento obtener una solución,

me encuentro con más inconvenientes. Espero que esta vez puedan resolverlo de manera efectiva.

Adiós.
```

Funcionario.txt

```
Buen día, soy Juan y estoy aquí para ayudarle. ¿Podría darme su documento? Gracias por su

paciencia. Hemos registrado su solicitud y estamos trabajando en ello. ¿Hay algo más en lo que

pueda ayudarle hoy? Hasta luego.
```

**Prueba 5**

Cliente.txt

Buenas tardes. Mi documento es 23456789. Estoy bastante decepcionado con el servicio. Ha habido

muchos errores y parece que nunca se resuelven. He tenido que llamar múltiples veces sin éxito.

Espero que esta vez puedan arreglarlo. Hasta luego.

Funcionario.txt

Buenas tardes, soy Laura y estoy aquí para asistirle. ¿Puede darme su documento, por favor?

Gracias por esperar. Hemos registrado su solicitud y pronto recibirá una respuesta. Agradecemos

su paciencia y comprensión. Que tenga un buen día. Hasta luego.

# CARACTERÍSTICAS NO IMPLEMENTADAS Y DECISIONES TOMADAS

Algo no contemplado y que me hubiese gustado incluir es la detección de palabras como “cancelar” o similares. Esto podría indicar que el cliente desea cancelar el servicio, lo cual debería automáticamente abortar el análisis. Un cliente que se comunica para cancelar el servicio representa una pérdida para la empresa, por lo que este caso particular requiere un tratamiento especial.

Es importante también considerar el contexto, ya que existen problemas como la polisemia, donde una palabra puede tener varios significados. Por ejemplo, la palabra "banco" puede referirse a una institución financiera o a un asiento. Decidí eliminar los acentos para simplificar el análisis, aunque esto podría generar confusiones. Por ejemplo:

> *"Es usted muy cortés conmigo"*
>
> *"Los cortes que hubieron en la última semana afectaron los servicios y me gustaría que puedas ayudarme"*

En la segunda oración, la palabra “cortes” se confundiría con la palabra “cortés”, afectando la puntuación del análisis. Para futuras implementaciones, es necesario contemplar el contexto de la oración.

Otro punto a mejorar sería subdividir las despedidas, ya que una despedida como “adios” o “hasta luego” debería tener una menor puntuación que una frase como “que tenga un buen día”. En esta última, se debería considerar el tiempo subjuntivo del verbo tener, utilizado para expresar deseos.

No se contemplan superlativos como “estuve contentísimo”, los cuales deberían incrementar o disminuir el puntaje en más de un punto, ya que denotan extremo enfado o extrema alegría. Tampoco se contemplan expresiones idiomáticas, que son expresiones fijas cuyo significado no puede deducirse de las palabras que la componen. Por ejemplo, "tirar la toalla" significa rendirse, y "estar en las nubes" significa estar distraído.

No hemos contemplado casos en los que una negación o doble negación puede alterar el significado de la frase. Por ejemplo, "No estoy insatisfecho" podría interpretarse como una satisfacción moderada, pero el algoritmo actual podría detectar "insatisfecho" como negativo.

El análisis no contempla el sarcasmo o la ironía, donde una expresión positiva puede ser usada con un sentido negativo. Por ejemplo, "¡Qué maravilloso servicio!" puede ser irónico si se dice en un contexto de insatisfacción.

Los conectores como "pero", "aunque", "sin embargo" pueden cambiar el sentimiento de una oración. Por ejemplo, "El servicio fue rápido, pero no eficiente" contiene una conjunción que cambia el sentimiento de positivo a negativo.

El análisis actual es muy lineal y no tiene en cuenta el contexto general de la conversación. Una serie de comentarios positivos seguidos de un comentario extremadamente negativo pueden cambiar la percepción global de la interacción.

La presencia de signos de exclamación puede intensificar el sentimiento expresado. Por ejemplo, "¡Estoy muy feliz con el servicio!" tiene un tono más positivo que "Estoy feliz con el servicio".

# Conclusión

El análisis de texto automático presenta varios desafíos que van más allá de la simple detección de palabras positivas o negativas. Aspectos como el contexto, la negación, la ironía, las entidades nombradas, y las construcciones gramaticales complejas deben ser considerados para obtener un análisis de sentimiento más preciso y significativo. Implementar soluciones que puedan manejar estos casos de manera efectiva requerirá técnicas más avanzadas de procesamiento del lenguaje natural, como el análisis de contexto y el uso de modelos de aprendizaje profundo entrenados con grandes corpus de datos.

# CODIGO FUENTE DEL TOKENIZADOR

El tokenizador también se encuentra disponible en el repositorio GitHub:

https://github.com/Marceeaax/Tokenizador

El tokenizador se encuentra dentro de un archivo llamado **`tokenizador.py`**

En el directorio raíz también se encuentran los archivos **`cliente.txt`** y **`funcionario.txt`**, que son los archivos de prueba para el tokenizador. Estos pueden ser editados a elección.

Para ejecutar el código solo es necesario tener instalado **Python** y ejecutar en una terminal

**`python tokenizador.py`**

```
import re

# Función para generar variaciones de una palabra en número y género
def generar_variaciones(palabra):
    variaciones = [palabra]
    # Si la palabra termina en 'o', generar variaciones en 'a', 'os' y
'as'
    if palabra.endswith('o'):
        variaciones.append(palabra[:-1] + 'a')
        variaciones.append(palabra + 's')
        variaciones.append(palabra[:-1] + 'as')
    # Si la palabra termina en 'a', generar variaciones en 'o', 'as' y
'os'
    elif palabra.endswith('a'):
        variaciones.append(palabra[:-1] + 'o')
        variaciones.append(palabra + 's')
        variaciones.append(palabra[:-1] + 'os')
    # Si la palabra termina en 'e', generar variación en 'es'
    elif palabra.endswith('e'):
        variaciones.append(palabra + 's')
    # Si la palabra termina en 'l', 'n' o 'r', generar variación en 'es'
    elif palabra.endswith('l') or palabra.endswith('n') or
palabra.endswith('r'):
        variaciones.append(palabra + 'es')
    # Para otros casos, generar variación en 's'
    else:
        variaciones.append(palabra + 's')
    return variaciones

# Función para generar listas de variaciones a partir de una lista de
palabras
def generar_listas_variaciones(lista_palabras):
    lista_variaciones = []
    # Para cada palabra en la lista, generar sus variaciones y añadirlas
a la lista de variaciones
```

```
    for palabra in lista_palabras:
        lista_variaciones.extend(generar_variaciones(palabra))
    return list(set(lista_variaciones))  # Eliminar duplicados

# Listas de palabras positivas, negativas y neutras (sin acentos y en
minúsculas)
# Las palabras son pasadas a la función generar_listas_variaciones para
incluir sus variaciones en número y género
palabras_positivas = generar_listas_variaciones([
    'gracias', 'por favor', 'ayuda', 'resolver', 'excelente', 'bueno',
'bien', 'fantastico', 'perfecto',
    'genial', 'asombroso', 'maravilloso', 'esplendido', 'magnifico',
'grandioso', 'estupendo', 'increible',
    'extraordinario', 'positivo', 'satisfactorio', 'brillante',
'fenomenal', 'impresionante', 'eficaz', 'eficiente',
    'rapido', 'puntual', 'comodo', 'seguro', 'fiable', 'amable',
'cortes', 'educado', 'respetuoso', 'simpatico',
    'agradable', 'cordial', 'dispuesto', 'atento', 'considerado',
'profesional', 'diligente', 'responsable',
    'competente', 'excepcional', 'impecable', 'meticuloso', 'cuidadoso',
'detallista', 'minucioso', 'preciso',
    'justo', 'equilibrado', 'coherente', 'logico', 'razonable',
'sensato', 'inteligente', 'listo', 'ingenioso',
    'habil', 'talentoso', 'virtuoso', 'artistico', 'creativo',
'innovador', 'original', 'visionario', 'perceptivo',
    'perspicaz', 'sagaz', 'agudo', 'brillante', 'lucido', 'sabio',
'experto', 'maestro', 'veterano', 'guia',
    'lider', 'jefe', 'mentor', 'entrenador', 'tutor', 'profesor',
'docente', 'facilitador', 'orador', 'disertante',
    'conferenciante', 'comunicador', 'informador', 'notificador',
'anunciador', 'publicista', 'promotor', 'difusor',
    'divulgador', 'recomendador', 'consultor', 'asesor', 'consejero',
'guia', 'psicologo', 'terapeuta', 'orientador'
])

palabras_negativas = generar_listas_variaciones([
    'mal', 'desastre', 'harto', 'cansado', 'cancelar', 'malo',
'deficiente', 'terrible',
    'pesimo', 'inaceptable', 'horrible', 'decepcionante', 'incompetente',
'frustrante', 'insatisfactorio',
    'lento', 'tardado', 'dificil', 'imposible', 'queja', 'reclamo',
'denuncia', 'fallo', 'error',
    'defecto', 'defectuoso', 'incidente', 'inconveniente', 'irresuelto',
'insuficiente', 'fallido',
    'engorroso', 'problematico', 'lamentable', 'aburrido', 'molesto',
'enojado', 'furioso', 'enfurecido',
    'indignado', 'agrio', 'amargo', 'desagradable', 'desesperante',
'triste', 'deprimente', 'abrumador',
```

```
    'deprimido', 'abatido', 'agobiado', 'perturbador', 'desmoralizador',
'desalentador', 'intolerable',
    'insoportable', 'irritante', 'irrespetuoso', 'arrogante',
'prepotente', 'soberbio', 'altanero',
    'despectivo', 'desdeñoso', 'cinico', 'sarcasmo', 'insulto',
'humillante', 'ofensivo', 'degradante',
    'hostil', 'antipatico', 'desalmado', 'cruel', 'brutal', 'sadico',
'malvado', 'vil', 'corrupto', 'deshonesto',
    'mentiroso', 'falso', 'enganador', 'tramposo', 'ladrones', 'timador',
'estafador', 'defraudador',
    'delincuente', 'criminal', 'inmoral', 'perverso', 'pecador',
'malevolo', 'demoniaco', 'siniestro',
    'oscuro', 'tenebroso', 'maldito', 'diabolico'
])

palabras_neutras = generar_listas_variaciones([
    'problema', 'inconveniente', 'dificultad', 'complicacion', 'retraso',
'espera', 'demora', 'consulta',
    'solicitud', 'peticion', 'pregunta', 'duda', 'incertidumbre',
'confusion', 'indecision', 'cuestion',
    'tramite', 'procedimiento', 'proceso', 'operacion', 'situacion',
'caso', 'circunstancia', 'condicion'
])

# Crear patrones de expresiones regulares a partir de las listas (sin
acentos y en minúsculas)
# Estos patrones permiten buscar las palabras en los textos
saludos_bienvenida = re.compile(r'\bbuen(?:os?)?
(?:dias?|tardes?|noches?)\b', re.IGNORECASE)
despedidas = re.compile(r'\b(?:hasta luego|adios|que tenga un buen
dia|que tenga un lindo resto de jornada|que le vaya bien|que tenga una
buena tarde|que tenga una buena noche|nos vemos|hasta la
proxima|cuidese|buenas noches)\b', re.IGNORECASE)
identificacion = re.compile(r'\b(?:documento|cedula)\b', re.IGNORECASE)
expresiones_positivas = re.compile(r'\b(?:' +
'|'.join(palabras_positivas) + r')\b', re.IGNORECASE)
expresiones_negativas = re.compile(r'\b(?:' +
'|'.join(palabras_negativas) + r')\b', re.IGNORECASE)
expresiones_neutras = re.compile(r'\b(?:' + '|'.join(palabras_neutras) +
r')\b', re.IGNORECASE)
preguntas_ayuda = re.compile(r'\b(?:algo mas|otra
(?:cosa|pregunta|inquietud|duda|consulta|ayuda)|que mas|alguna otra
(?:pregunta|inquietud|consulta|cosa|duda|ayuda|necesidad|problema|asunto|
detalle))\b', re.IGNORECASE)

# Función para preprocesar el texto
def preprocesar_texto(texto):
    # Convertir a minúsculas
    texto = texto.lower()
```

```
    # Eliminar acentos
    acentos = {
        'á': 'a', 'é': 'e', 'í': 'i', 'ó': 'o', 'ú': 'u',
        'à': 'a', 'è': 'e', 'ì': 'i', 'ò': 'o', 'ù': 'u',
        'ä': 'a', 'ë': 'e', 'ï': 'i', 'ö': 'o', 'ü': 'u',
        'â': 'a', 'ê': 'e', 'î': 'i', 'ô': 'o', 'û': 'u'
    }
    for acento, letra in acentos.items():
        texto = texto.replace(acento, letra)
    return texto

# Función para procesar el texto del funcionario
def evaluar_funcionario(texto):
    # Preprocesar el texto para convertirlo a minúsculas y eliminar
acentos
    texto = preprocesar_texto(texto)
    puntuacion = 0
    detalles = []

    # Buscar saludos de bienvenida
    if saludos_bienvenida.search(texto):
        puntuacion += 1
        detalles.append("Saludo de bienvenida detectado")
    else:
        detalles.append("Falta saludo de bienvenida")

    # Buscar despedidas
    if despedidas.search(texto):
        puntuacion += 1
        detalles.append("Despedida detectada")
    else:
        detalles.append("Falta despedida")

    # Buscar solicitudes de identificación
    if identificacion.search(texto):
        puntuacion += 1
        detalles.append("Solicitud de identificacion detectada")
    else:
        detalles.append("Falta solicitud de identificacion")

    # Buscar expresiones positivas
    if expresiones_positivas.search(texto):
        puntuacion += 1
        detalles.append("Uso de expresiones positivas detectado")
    else:
        detalles.append("Falta uso de expresiones positivas")

    # Buscar preguntas de ayuda adicional
    if preguntas_ayuda.search(texto):
```

```
        puntuacion += 1
        detalles.append("Pregunta de ayuda adicional detectada")
    else:
        detalles.append("Falta pregunta de ayuda adicional")

    # Ajustar la puntuacion a una escala del 1 al 5
    puntuacion = max(1, min(puntuacion, 5))

    return puntuacion, detalles

# Función para procesar el texto del cliente
def evaluar_cliente(texto):
    # Preprocesar el texto para convertirlo a minúsculas y eliminar
acentos
    texto = preprocesar_texto(texto)
    puntuacion = 5  # Empezamos con una puntuacion base de 5
    palabras_buenas = []
    palabras_malas = []
    palabras_neutras_encontradas = []

    # Buscar y almacenar palabras positivas
    buenas = expresiones_positivas.findall(texto)
    palabras_buenas.extend(buenas)

    # Buscar y almacenar palabras negativas
    malas = expresiones_negativas.findall(texto)
    palabras_malas.extend(malas)

    # Buscar y almacenar palabras neutras
    neutras = expresiones_neutras.findall(texto)
    palabras_neutras_encontradas.extend(neutras)

    # Ajustar la puntuacion en base a las palabras encontradas
    puntuacion += len(buenas)
    puntuacion -= len(malas)

    # Ajustar la puntuacion a una escala del 1 al 5
    puntuacion = max(1, min(puntuacion, 5))

    return puntuacion, palabras_buenas, palabras_malas,
palabras_neutras_encontradas

# Función para calificar la experiencia en base a la puntuación
def calificar_experiencia(puntuacion):
    if puntuacion > 3:
        return "positiva"
    elif puntuacion == 3:
        return "neutra"
    else:
```

```
        return "negativa"

# Función para preguntar al usuario si desea incluir una palabra neutra
como negativa
def preguntar_incluir_palabra_neutra(palabra):
    respuesta = input(f"¿Desea incluir la palabra '{palabra}' como
negativa? (si/no): ").strip().lower()
    return respuesta == 'si'

# Función principal para evaluar ambos archivos
def evaluar_llamada(archivo_funcionario, archivo_cliente):
    with open(archivo_funcionario, 'r', encoding='utf-8') as
f_funcionario:
        texto_funcionario = f_funcionario.read()

    with open(archivo_cliente, 'r', encoding='utf-8') as f_cliente:
        texto_cliente = f_cliente.read()

    print("Evaluando el texto del funcionario...")
    puntuacion_funcionario, detalles_funcionario =
evaluar_funcionario(texto_funcionario)
    calificacion_funcionario =
calificar_experiencia(puntuacion_funcionario)
    print(f"Puntuacion del funcionario: {puntuacion_funcionario}
(Experiencia {calificacion_funcionario})")
    for detalle in detalles_funcionario:
        print(f" - {detalle}")

    print("\nEvaluando el texto del cliente...")
    puntuacion_cliente, palabras_buenas, palabras_malas, palabras_neutras
= evaluar_cliente(texto_cliente)
    calificacion_cliente = calificar_experiencia(puntuacion_cliente)
    print(f"Puntuacion del cliente: {puntuacion_cliente} (Experiencia
{calificacion_cliente})")
    print(f"Palabras buenas detectadas: {', '.join(palabras_buenas) if
palabras_buenas else 'Ninguna'}")
    print(f"Palabras malas detectadas: {', '.join(palabras_malas) if
palabras_malas else 'Ninguna'}")
    print(f"Palabras neutras detectadas: {', '.join(palabras_neutras) if
palabras_neutras else 'Ninguna'}")

    # Inicializar la longitud original de palabras negativas
    longitud_original_negativas = len(palabras_negativas)

    # Preguntar al usuario si desea incluir palabras neutras como
negativas
    for palabra in palabras_neutras:
        if preguntar_incluir_palabra_neutra(palabra):
            palabras_negativas.append(palabra)
```

```
    # Recalcular la puntuacion del cliente si se añadieron palabras
neutras a negativas
    if len(palabras_negativas) > longitud_original_negativas:
        # Actualizar expresiones_negativas
        global expresiones_negativas
        expresiones_negativas = re.compile(r'\b(?:' +
'|'.join(palabras_negativas) + r')\b', re.IGNORECASE)
        puntuacion_cliente, palabras_buenas, palabras_malas, _ =
evaluar_cliente(texto_cliente)
        calificacion_cliente = calificar_experiencia(puntuacion_cliente)
        print(f"\nNueva Puntuacion del cliente: {puntuacion_cliente}
(Experiencia {calificacion_cliente})")
        print(f"Palabras malas detectadas (incluyendo las nuevas): {',
'.join(palabras_malas) if palabras_malas else 'Ninguna'}")

# Ejemplo de uso
archivo_funcionario = 'funcionario.txt'
archivo_cliente = 'cliente.txt'

evaluar_llamada(archivo_funcionario, archivo_cliente)
```