Análisis de Interacciones en Contact Centers con Speech Analytics y Tokenización

Alexis Rodrigo Arce Delgadillo

Diseño de Compiladores

Profesor: Sergio Andrés Aranda Zemán

Noviembre 2024

Índice

Resumen

Este trabajo presenta un sistema para el análisis de interacciones en contact centers utilizando técnicas de Speech Analytics y tokenización. El objetivo es procesar el texto derivado de las transcripciones de audio, clasificando las palabras en categorías como saludos, despedidas, identificaciones, palabras prohibidas, palabras positivas y palabras negativas, así como realizar un análisis de sentimientos basado en esas palabras. El sistema utiliza un autómata finito determinista (AFD) para identificar los lexemas y estructurar el flujo de interacción. Además, el AFD permite detectar y verificar el cumplimiento de un protocolo estándar de atención por parte del agente. Finalmente, se presentan y analizan los resultados obtenidos del procesamiento de textos simulados, destacando las conclusiones sobre la efectividad del sistema en la mejora de la calidad de atención al cliente.

1. Introducción

El propósito de este trabajo práctico es implementar un sistema que procese las interacciones comunes en un centro de contacto (contact center) y que, utilizando técnicas de tokenización y análisis léxico, permita evaluar la calidad de la atención brindada por los agentes. El objetivo final es identificar el sentimiento de las conversaciones y verificar si el protocolo de atención al cliente ha sido seguido correctamente.

El sistema desarrollado se centra en la clasificación léxica de las palabras en categorías específicas, como saludos, despedidas, identificaciones y palabras con connotaciones emocionales (positivas y negativas). Este enfoque es esencial en el contexto de un contact center, donde la precisión y la calidad de la comunicación son fundamentales para garantizar la satisfacción del cliente. Para lograr esto, se implementa un Autómata Finito Determinista (AFD) que facilita la identificación y estructuración de lexemas, permitiendo un mapeo efectivo de la interacción entre el cliente y el agente.

El uso del AFD permite procesar las interacciones de manera eficiente y precisa, clasificando los lexemas en tiempo real. Esta capacidad no solo optimiza el flujo de datos, sino que también habilita la verificación del cumplimiento de un protocolo estándar de atención, asegurando que se sigan las mejores prácticas durante la interacción. La estructura del AFD es flexible, lo que permite la incorporación de nuevos términos y categorías a medida que evoluciona el lenguaje utilizado en el servicio al cliente.

Además, el análisis de sentimientos se lleva a cabo mediante una tabla de puntuaciones asignadas a las palabras identificadas. Este análisis permite evaluar la calidad de la comunicación y la satisfacción del cliente de manera cuantitativa, proporcionando una visión integral del estado emocional de la interacción. La capacidad de medir el tono y el sentimiento detrás de las palabras ayuda a identificar patrones de comportamiento, lo que resulta invaluable para la mejora continua de los procesos de atención al cliente. Los resultados obtenidos del procesamiento de textos se presentan al final del análisis, destacando la efectividad del sistema para identificar patrones de comportamiento en el desempeño del servicio al cliente.

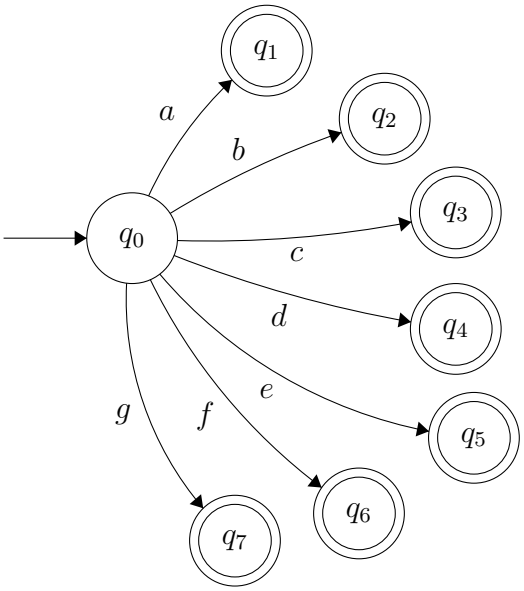
2. Desarrollo

2.1. Descripción del algoritmo

El algoritmo implementado es un sistema de análisis de interacciones en contact centers que utiliza técnicas de tokenización y análisis de sentimientos. A continuación se detallan los componentes y el funcionamiento del algoritmo:

2.1.1. Autómata Finito Determinista (AFD)

El AFD es la estructura central que permite la clasificación de las palabras en diferentes categorías como saludos, identificaciones, despedidas, palabras positivas, negativas y prohibidas. Se inicia con un estado inicial (q0) y tiene varios estados finales que corresponden a las categorías mencionadas. Las transiciones se definen para cada palabra de interés, lo que permite al AFD aceptar o rechazar lexemas basándose en las palabras que se encuentran en el texto analizado.



Símbolo	Representación - Tokens
q_0	INICIO
q_1	SALUDO
q_2	IDENTIFICACION
q_3	DESPEDIDA
q_4	PALABRAS POSITIVAS
q_5	PALABRAS NEGATIVAS
q_6	PALABRAS PROHIBIDAS
q_7	ERRORES LEXICOS
Símbolo	Representación - Patrones
a	Lista saludos
b	Lista identificación
c	Lista despedidas
d	Lista palabras positivas
e	Lista palabras negativas
f	Lista palabras prohibidas
g	otras palabras

Figura 1. Representación de la clase AFD

2.1.2. Tokenización

El texto se procesa para extraer los lexemas, que son palabras individuales. Cada lexema se evalúa utilizando el AFD; si el lexema coincide con una de las transiciones definidas, se acepta y se cuenta en la categoría correspondiente. Si no, se considera un error léxico.

2.1.3. Análisis de Sentimiento

Se utiliza una tabla de sentimientos que asigna puntajes a palabras positivas y negativas. Al procesar los lexemas, se acumula un puntaje total que determina si el

sentimiento general de la interacción es positivo, negativo o neutral.

2.1.4. Verificación del Protocolo de Atención

Se verifica si se cumplen ciertas condiciones de atención al cliente, como la presencia de saludos y despedidas, y se detectan palabras que no son apropiadas.

2.1.5. Estructura de Datos

El AFD tiene un diccionario para almacenar las transiciones entre estados y otro para los identificadores de lexemas. Un contador permite llevar el registro de cuántas veces se han reconocido las palabras de cada categoría. La tabla de sentimientos utiliza una lista enlazada donde es posible agregar o eliminar palabras y ponderaciones mediante el menú del sistema.

2.1.6. Salida de Resultados

Los resultados del análisis, que incluyen el sentimiento general, puntajes, palabras clasificadas, y la verificación del protocolo, se guardan en un archivo de texto y en un archivo JSON que representa el estado del AFD.

2.2. Decisiones Tomadas

2.2.1. Selección de un AFD

Se decidió implementar un Autómata Finito Determinista (AFD) como estructura principal para la clasificación de palabras debido a su capacidad para manejar eficientemente un conjunto finito de estados y transiciones. Esta elección permite un análisis rápido de las entradas de texto y una categorización clara de los lexemas.

2.2.2. Definición de Estados y Transiciones

Se definieron múltiples estados finales en el AFD, cada uno asociado a distintas categorías de palabras: saludos, identificaciones, despedidas, palabras prohibidas, palabras positivas y palabras negativas. Esta organización estructurada permite un análisis claro y eficiente de las interacciones con los clientes. Las palabras correspondientes a cada categoría se cargan previamente desde archivos de texto, listos para su uso en el análisis.

Las transiciones del autómata se implementaron para reconocer un conjunto específico de palabras, seleccionadas cuidadosamente para representar las categorías relevantes dentro del contexto de atención al cliente y la verificación del protocolo que debe seguir el agente en diversas situaciones.

2.2.3. Implementación de la Tokenización

Se optó por utilizar expresiones regulares para extraer lexemas del texto de entrada. Este enfoque permite una identificación precisa de las palabras individuales y su procesamiento en el AFD. Se incorporó un manejo de errores léxicos mediante la creación de un estado **q_ERROR_LX** para capturar palabras no reconocidas. Esto

mejora la robustez del algoritmo al permitir una respuesta adecuada ante entradas inesperadas.

2.2.4. Verificación del Protocolo

Se implementó un mecanismo para verificar el cumplimiento del protocolo de atención al cliente, asegurando que las interacciones contengan elementos clave como saludos y despedidas. Esto ayuda a evaluar la calidad del servicio proporcionado por el agente. Esta decisión se tomó con el objetivo de garantizar que el análisis no solo sea técnico, sino también relevante para las prácticas de atención al cliente.

2.2.5. Salida y Visualización de Resultados

Se decidió que los resultados del análisis se guardarían tanto en un archivo de texto como en un archivo JSON. Esto permite no solo una fácil revisión por parte de los usuarios, sino también la posibilidad de almacenar el estado del AFD para futuros análisis. La estructura JSON se eligió para facilitar la visualización y el análisis posterior del comportamiento del AFD y el funcionamiento de las transiciones.

2.3. Alcance del Proyecto

El alcance del proyecto se puede definir a través de los siguientes puntos clave:

2.3.1. Clasificación de Lexemas

El sistema identifica y clasifica lexemas en diferentes categorías, permitiendo una mejor comprensión del contexto de la interacción. Esto incluye clasificaciones específicas para saludos, identificaciones, despedidas, palabras prohibidas y palabras que expresan sentimientos.

2.3.2. Análisis de Sentimientos

Se realiza un análisis de sentimientos sobre las interacciones, evaluando las palabras utilizadas para determinar si la conversación fue positiva, negativa o neutral. Esto se realiza mediante la tabla de sentimientos donde cada palabra posee una ponderación pre-cargada desde un archivo de texto.

2.3.3. Verificación de Protocolos de Atención

El sistema verifica si se cumplen los protocolos de atención al cliente, asegurando que las interacciones incluyan los elementos necesarios para una atención efectiva. Esto incluye la detección de saludos y despedidas apropiadas por parte del agente.

2.3.4. Manejo de palabras no clasificadas

La implementación del estado q_ERROR_LX permite manejar palabras no reconocidas o errores léxicos, mejorando la robustez del sistema ante entradas inesperadas.

2.3.5. Generación de Informes

Los resultados del análisis se generan en archivos de texto donde se imprimen los reportes, una tabla de lexemas y un JSON con las transiciones de cada palabra analizada, lo que permite una fácil revisión.

2.4. Estrategias Aplicadas

Las estrategias aplicadas en el desarrollo del sistema incluyen:

2.4.1. Desarrollo Modular

El sistema se construyó de manera modular, separando las diferentes funcionalidades en clases y métodos. Esto permite un mantenimiento más sencillo y la posibilidad de añadir nuevas características en el futuro.

2.4.2. Uso de AFD para Eficiencia

La elección de un AFD como base del sistema permite una evaluación rápida y eficiente de los lexemas, facilitando el procesamiento de grandes volúmenes de texto de interacciones.

2.4.3. Implementación de Tablas de Sentimientos

Se diseñó una tabla de sentimientos que asigna puntajes específicos a palabras según su connotación, lo cual permite un análisis más detallado y preciso de las emociones expresadas en las interacciones. Estos datos, pre-cargados desde un archivo de texto para garantizar su persistencia, incluyen la flexibilidad de agregar o eliminar palabras según se requiera, facilitando así la personalización y actualización continua del análisis de sentimiento.

2.4.4. Recopilación de Datos y Mejora Continua

Se incorporó un enfoque que permite la adición de nuevas palabras a las categorías existentes, lo que permite que el sistema evolucione a medida que se identifican nuevas necesidades esto mediante el uso de documentos de texto donde las palabras son guardadas.

2.4.5. Validación de Resultados

Se implementaron mecanismos para validar los resultados del sistema, asegurando que los análisis sean precisos y útiles para evaluar la calidad del servicio de atención al cliente.

2.5. Código Fuente

En esta sección, se presenta el código fuente del sistema desarrollado para el análisis de interacciones en contact centers. El código está organizado en módulos, clases y métodos que permiten la tokenización de texto, el análisis de sentimientos y la verificación de protocolos de atención.

El sistema está estructurado de la siguiente manera:

```
+-- data
|   +-- despedidas.txt
|   +-- identificaciones.txt
|   +-- palabras_prohibidas.txt
|   +-- palabras_y_puntajes.txt
|   +-- saludos.txt
|
+-- doc
|   +-- ensayo.pdf
|
+-- models
|   +-- afd.py
|   +-- analisis.py
|   +-- file_utils.py
|   +-- menu.py
|   +-- sentimientos.py
|   +-- __init__.py
|
+-- output
|   +-- afd.json
|   +-- reporte.txt
|   +-- tabla_lexemas.txt
|
+-- test
|    +-- prueba-texto1-negativo.txt
|    +-- prueba-texto2-positivo.txt
|    +-- prueba-texto3-positivo.txt
|    +-- prueba-texto4-neutral.txt
|    +-- prueba.txt
|
|- tokenizador.py
|- README.md
```

2.5.1. Clase AFD

```
import re
import json
import os

class AFD:
    def __init__(self, estado_inicial, estados_finales):
        self.estado_inicial = estado_inicial
        self.estados_finales = estados_finales
        self.transiciones = {estado_inicial: {}}
        self.lexema_ids = {}
        self.next_id = 1
        self.transiciones_usadas = {}

    def agregar_transicion(self,estado_origen,lexema,estado_destino):
        if estado_origen not in self.transiciones:
            self.transiciones[estado_origen] = {}
        if lexema in self.transiciones[estado_origen]:
            raise Exception("Transicion duplicada")
        self.transiciones[estado_origen][lexema] = estado_destino

        # No incrementar el contador aquí
        if lexema not in self.lexema_ids:
            self.lexema_ids[lexema] = self.next_id
            self.next_id += 1
        return self.lexema_ids[lexema]


    def evaluar_lexema(self, lexema):
        estado_actual = self.estado_inicial
        if lexema in self.transiciones[estado_actual]:
            estado_actual = self.transiciones[estado_actual][lexema]
        else:
            return estado_actual, False

        return estado_actual, True
    def vaciar(self):
        self.transiciones = {self.estado_inicial: {}}
        self.lexema_ids.clear()
        self.next_id = 1
        self.transiciones_usadas.clear()
    def get_ids(self):
        return self.lexema_ids

    def guardar_en_json(self, filename):
        data = {
```

```
            'estado_inicial': self.estado_inicial,
            'estados_finales': self.estados_finales,
            'transiciones': {self.estado_inicial:{}}
        }
        for estado, transiciones in self.transiciones_usadas.items():
            for transicion in transiciones:
                data['transiciones'][self.estado_inicial][transicion] = estado

        with open(os.path.join("output",filename), 'w', encoding='utf-8') as f:
            json.dump(data, f, ensure_ascii=False, indent=4)

    def generar_tabla_lexemas(self):
        tabla = []
        for estado, lexemas in self.transiciones_usadas.items():
            for lexema in lexemas:
                patrones = [lexema
                    for lexema in self.transiciones[self.estado_inicial]
                    if self.transiciones[self.estado_inicial][lexema] == estado]
                tabla.append({
                    'Lexema': lexema,
                    'Token': estado,
                    'Patron': patrones
                })
        return tabla
```

Explicación de la Clase AFD

Inicialización: La clase `AFD` se inicializa definiendo el estado inicial, los estados finales y las transiciones que definen el flujo del autómata.
Métodos:

- **agregar_transicion:** Permite agregar transiciones entre estados mediante lexemas, asegurando que no haya duplicados en las transiciones.
- **evaluar_lexema:** Evalúa si un lexema puede ser procesado desde el estado inicial hacia uno de los estados finales, indicando si se encontró una transición válida.
- **vaciar:** Reinicia el AFD, eliminando todas las transiciones y reiniciando los IDs de lexemas.
- **guardar_en_json:** Guarda el estado y las transiciones del autómata en un archivo JSON.
- **generar_tabla_lexemas:** Genera una tabla organizada de lexemas, mostrando su token y patrón asociado, para facilitar el análisis de la clasificación de palabras.

2.5.2. Clase TokenizadorAFD

```
class TokenizadorAFD:
    def __init__(self):
        self.afd = self.construir_afd()
        self.lista_salida = []
        self.posicion = 0
    def cargar_palabras_y_puntajes(self, archivo):
        palabras_positivas = []
        palabras_negativas = []
        with open(os.path.join("data", archivo), "r", encoding="utf-8") as f:
            for linea in f:
                palabra, puntaje = linea.strip().split(",")
                puntaje = int(puntaje)
                if puntaje > 0:
                    palabras_positivas.append(palabra)
                else:
                    palabras_negativas.append(palabra)
            f.close()
        return palabras_positivas, palabras_negativas
    def cargar_palabras(self,archivo):
        lexemas = []
        with open(os.path.join("data",archivo),"r",encoding="utf-8") as f:
            for linea in f:
                lexemas.append(linea.strip())
        return lexemas

    def construir_afd(self):
        estado_inicial = "q0"
        estado_finales = {
            'SALUDO': 'q_SALUDO',
            'IDENTIFICACION': 'q_IDENTIFICACION',
            'DESPEDIDA': 'q_DESPEDIDA',
            'ERROR_LX': 'q_ERROR_LX',
            'POSITIVO': 'q_POSITIVO',
            'NEGATIVO': 'q_NEGATIVO',
            'PALABRA_PROHIBIDA': 'q_PALABRA_PROHIBIDA'

        }
        afd = AFD(estado_inicial, estado_finales)

        saludos = self.cargar_palabras("saludos.txt")
        identificaciones = self.cargar_palabras("identificaciones.txt")
        despedidas = self.cargar_palabras("despedidas.txt")
        palabras_prohibidas = self.cargar_palabras("palabras_prohibidas.txt")
        palabras_positivas, palabras_negativas =
                    self.cargar_palabras_y_puntajes("palabras_y_puntajes.txt")
```

```
    for saludo in saludos:
        afd.agregar_transicion(estado_inicial, saludo, "q_SALUDO")

    for identificacion in identificaciones:
        afd.agregar_transicion(estado_inicial,
                               identificacion, "q_IDENTIFICACION")

    for despedida in despedidas:
        afd.agregar_transicion(estado_inicial, despedida,
                               "q_DESPEDIDA")

    for palabra_positiva in palabras_positivas:
        afd.agregar_transicion(estado_inicial, palabra_positiva,
                               "q_POSITIVO")

    for palabra_negativa in palabras_negativas:
        afd.agregar_transicion(estado_inicial, palabra_negativa,
                               "q_NEGATIVO")

    for palabra_prohibida in palabras_prohibidas:
        afd.agregar_transicion(estado_inicial, palabra_prohibida,
                               "q_PALABRA_PROHIBIDA")

    return afd


def tokenizador(self, texto):
    lexemas = re.findall(r'\b\w+:|\b\w+', texto.lower())

    for lexema in lexemas:
        estado, es_aceptado = self.afd.evaluar_lexema(lexema)

        if es_aceptado:
            print(f"Lexema aceptado: [{lexema}] se detecto como {estado}")
            self.posicion += 1

            if estado not in self.afd.transiciones_usadas:
                self.afd.transiciones_usadas[estado] = []
            self.afd.transiciones_usadas[estado].append(lexema)

        else:
            print(f"Lexema no aceptado: [{lexema}]")
            self.agregar_error_lexico(lexema)
            self.posicion += 1
```

```
                error_estado = "q_ERROR_LX"
                if error_estado not in self.afd.transiciones_usadas:
                    self.afd.transiciones_usadas[error_estado] = []
                self.afd.transiciones_usadas[error_estado].append(lexema)
        return lexemas


    def agregar_error_lexico(self, error_lx):
        self.afd.agregar_transicion("q0", error_lx, "q_ERROR_LX")
```

Explicación de la Clase TokenizadorAFD

Inicialización: La clase `TokenizadorAFD` se inicializa creando el autómata finito determinista (AFD) y configurando la lista de salida y la posición actual para el procesamiento de tokens.

Métodos:

- **cargar_palabras_y_puntajes:** Carga palabras junto con sus puntajes desde un archivo, organizándolas en listas de palabras positivas y negativas para facilitar su uso en el AFD.
- **cargar_palabras:** Carga un conjunto de lexemas desde un archivo de texto, permitiendo agrupar palabras relacionadas, como saludos o identificaciones, en listas para ser utilizadas en el autómata.
- **construir_afd:** Define el AFD inicializando el estado de partida y los estados finales. Además, agrega las transiciones de los estados, asociando palabras como saludos, identificaciones, despedidas, palabras prohibidas y términos con connotaciones positivas o negativas a sus respectivos estados finales.
- **tokenizador:** Toma el texto de entrada y lo divide en lexemas utilizando expresiones regulares. Luego, para cada lexema, verifica si existe una transición definida en el AFD. Si se encuentra una transición, el lexema es clasificado en el estado correspondiente; si no, se trata como un error léxico y se registra en el estado de error. Este proceso también actualiza la posición del token para un análisis secuencial.
- **agregar_error_lexico:** Registra un lexema no reconocido en el estado de error del AFD, permitiendo una mejor identificación y manejo de palabras no válidas en el contexto de análisis léxico.
- **guardar_en_json:** Este método, aunque no se muestra en el código, típicamente permite guardar la configuración actual del AFD en un archivo JSON, lo cual facilita la observación de estados y transiciones entre ejecuciones.

2.5.3. Análisis de Sentimientos y Verificación de Protocolo

```
import re
def analizar_sentimiento(tokens, tabla_sentimientos, tokenizador):
    puntaje_total = 0
    palabras_positivas = []
    palabras_negativas = []

    palabras_rudas_detectadas = False

    for token in tokens:
        estado, es_aceptado = tokenizador.afd.evaluar_lexema(token)
        if es_aceptado and estado == 'q_PALABRA_PROHIBIDA':
            palabras_rudas_detectadas = True
            break

    if palabras_rudas_detectadas:
        return {
            "sentimiento": "Negativo",
            "puntaje_total": -1,
            "palabras_positivas": [],
            "palabras_negativas": []
        }

    for token in tokens:
        puntaje = tabla_sentimientos.obtener_puntaje(token)
        puntaje_total += puntaje

        if puntaje > 0:
            palabras_positivas.append(token)
        elif puntaje < 0:
            palabras_negativas.append(token)

    sentimiento = "Positivo" if puntaje_total > 0
        else "Negativo" if puntaje_total < 0 else "Neutral"

    return {
        "sentimiento": sentimiento,
        "puntaje_total": puntaje_total,
        "palabras_positivas": palabras_positivas,
        "palabras_negativas": palabras_negativas
    }

def verificar_protocolo(lexemas, tokenizador):
    resultados = {
        "saludo": False,
        "identificacion": False,
```

```
        "despedida": False,
        "palabras_prohibidas": False
    }

    analizando_agente = False
    mensaje_agente = []

    for lexema in lexemas:
        # Detectar inicio de mensaje del agente
        if "agente:" in lexema:
            analizando_agente = True
            mensaje_agente.append(lexema.split("agente:", 1)[1].strip())
        elif "cliente:" in lexema and analizando_agente:
            analizando_agente = False
            procesar_mensaje_agente(mensaje_agente, tokenizador, resultados)
            mensaje_agente = []  # Reiniciar el mensaje

        elif analizando_agente:
            mensaje_agente.append(lexema)

    if mensaje_agente:
        procesar_mensaje_agente(mensaje_agente, tokenizador, resultados)

    return resultados

def procesar_mensaje_agente(mensaje_agente, tokenizador, resultados):
    mensaje = ' '.join(mensaje_agente)
    tokens = re.findall(r'\b\w+\b', mensaje.lower())

    for token in tokens:
        estado,es_aceptado = tokenizador.afd.evaluar_lexema(token)

        if es_aceptado:
            if estado == 'q_SALUDO':
                resultados["saludo"] = True
            elif estado == 'q_IDENTIFICACION':
                resultados["identificacion"] = True
            elif estado == 'q_DESPEDIDA':
                resultados["despedida"] = True
            elif estado == 'q_PALABRA_PROHIBIDA':
                resultados["palabras_prohibidas"] = True
```

Funciones de Análisis de Sentimiento y Verificación de Protocolo

Función `analizar_sentimiento`
La función `analizar_sentimiento` clasifica el sentimiento de un conjunto de lexemas (palabras) y detecta la presencia de palabras prohibidas.

Entradas:

- `lexemas`: Lista de palabras de la conversación.
- `tabla_sentimientos`: Objeto para obtener el puntaje asociado a cada palabra.
- `tokenizador`: Instancia de `TokenizadorAFD` que permite identificar el estado asociado a cada lexema.

Proceso:

1. Se revisa cada lexema para verificar si pertenece al estado `q_PALABRA_PROHIBIDA`, indicando la detección de una palabra prohibida y terminando el análisis de inmediato con un sentimiento "Negativo".
2. Si no hay palabras prohibidas, se calcula el `puntaje_total` sumando los puntajes asociados a cada lexema: las palabras con puntaje positivo se agregan a `palabras_positivas` y las de puntaje negativo a `palabras_negativas`.
3. Según el `puntaje_total`, el sentimiento resultante es:
    - "Positivo" si el puntaje es mayor a 0.
    - "Negativo" si el puntaje es menor a 0.
    - "Neutral" si el puntaje es igual a 0.

Salida: Un diccionario con el sentimiento, puntaje total y listas de palabras positivas y negativas.

Función `verificar_protocolo`
Esta función verifica si el agente sigue el protocolo mediante la detección de un saludo, identificación, despedida y ausencia de palabras prohibidas.

Entradas:

- `lexemas`: Lista de palabras extraídas de la interacción.
- `tokenizador`: Instancia de `TokenizadorAFD` que permite identificar el estado de cada lexema.

Proceso:

1. Se separan los mensajes del agente y del cliente en el texto.
2. Para cada segmento del agente, se llama a la función `procesar_mensaje_agente`.

Función `procesar_mensaje_agente`
Esta función procesa cada mensaje del agente y clasifica cada lexema en los estados de protocolo esperados.

Entradas:

- `mensaje_agente`: Lista de lexemas de los mensajes del agente.
- `tokenizador`: Instancia del `TokenizadorAFD`.
- `resultados`: Diccionario que almacena el estado del protocolo (saludo, identificación, despedida y palabras prohibidas).

Proceso:

1. Combina los mensajes del agente en una cadena y la tokeniza.
2. Revisa cada lexema para determinar si pertenece a alguno de los estados de protocolo (`q_SALUDO`, `q_IDENTIFICACION`, `q_DESPEDIDA`, `q_PALABRA_PROHIBIDA`).
3. Actualiza el diccionario `resultados` según el estado detectado.

Salida: El diccionario `resultados` actualizado.

2.5.4. Clase Manejo de Tabla de Sentimientos

```
    class NodoSentimiento:
    def __init__(self, palabra, puntaje):
        self.palabra = palabra
        self.puntaje = puntaje
        self.siguiente = None

class TablaSentimientos:
    def __init__(self):
        self.cabeza = None

    def agregar_palabra(self, palabra, puntaje):
        nuevo_nodo = NodoSentimiento(palabra, puntaje)
        if self.cabeza is None:
            self.cabeza = nuevo_nodo
        else:
            actual = self.cabeza
            while actual.siguiente:
                actual = actual.siguiente
            actual.siguiente = nuevo_nodo

    def obtener_puntaje(self, palabra):
        actual = self.cabeza
        while actual:
            if actual.palabra == palabra:
```

```
            return actual.puntaje
        actual = actual.siguiente
    return 0  # Retornar 0 si la palabra no se encuentra

def eliminar_palabra(self, palabra):
    actual = self.cabeza
    anterior = None
    while actual:
        if actual.palabra == palabra:
            if anterior is None:  # Eliminar la cabeza
                self.cabeza = actual.siguiente
            else:
                anterior.siguiente = actual.siguiente
            return  # Palabra eliminada
        anterior = actual
        actual = actual.siguiente
    print(f"La palabra '{palabra}' no se encontró
    en la tabla de sentimientos.")
```

Estructura de la Tabla de Sentimientos

La estructura de la `TablaSentimientos` y la clase `NodoSentimiento` permiten almacenar y gestionar palabras asociadas a un puntaje de sentimiento en una lista enlazada. Esta estructura se utiliza para calcular el puntaje de sentimiento de un conjunto de palabras.

Clase `NodoSentimiento`

La clase `NodoSentimiento` representa un nodo en la lista enlazada y contiene los atributos:

- `palabra`: La palabra asociada al sentimiento.
- `puntaje`: El puntaje de sentimiento de la palabra, que puede ser positivo, negativo o cero.
- `siguiente`: Referencia al siguiente nodo en la lista enlazada.

Clase `TablaSentimientos`

La clase `TablaSentimientos` contiene métodos para gestionar la lista de palabras con sus puntajes. Sus métodos principales son:

- `agregar_palabra(palabra, puntaje)`: Este método agrega un nuevo nodo al final de la lista enlazada con la palabra y puntaje especificados. Si la lista está vacía, el nodo se convierte en la cabeza.

- `obtener_puntaje(palabra)`: Busca en la lista la palabra proporcionada y devuelve el puntaje asociado. En caso de no encontrar la palabra, retorna 0. Este método recorre la lista desde la cabeza hasta el final.
- `eliminar_palabra(palabra)`: Elimina el nodo que contiene la palabra especificada. Para ello:
  - Si el nodo a eliminar es la cabeza, la cabeza se actualiza para que apunte al siguiente nodo.
  - Si la palabra se encuentra en otro nodo, se enlaza el nodo anterior al siguiente nodo del actual.

  Si la palabra no se encuentra, se muestra un mensaje indicando que no existe en la tabla.

Uso de la Tabla de Sentimientos

Esta estructura permite almacenar y recuperar palabras con sus puntajes de manera eficiente, facilitando el análisis de sentimientos en otras funciones del programa mediante métodos de inserción, búsqueda y eliminación en una lista enlazada.

2.5.5. Gestión de Sentimientos

```
import os
from models.sentimientos import TablaSentimientos

def eliminar_archivo(nombre_archivo):
    try:
        os.remove(os.path.join("output",nombre_archivo))
    except FileNotFoundError:
        pass
    except Exception as e:
        pass
def cargar_palabras_y_puntajes(archivo):
    tabla_sentimientos = TablaSentimientos()
    with open(os.path.join("data", archivo), "r", encoding="utf-8") as f:
        for linea in f:
            palabra, puntaje = linea.strip().split(",")
            tabla_sentimientos.agregar_palabra(palabra, int(puntaje))
        f.close()
    return tabla_sentimientos

def agregar_palabra_sentimiento(tabla_sentimientos, afd):
    palabra = input("Ingrese la nueva palabra: ").strip()
    puntaje = int(input("Ingrese el puntaje de la palabra
    (positivo o negativo): "))
    eliminar_archivo("tabla_lexemas.txt")
    eliminar_archivo("afd.json")
```

```
        tabla_sentimientos.agregar_palabra(palabra, puntaje)
        afd.agregar_transicion(afd.estado_inicial, palabra,
        f"q_{puntaje > 0 and 'POSITIVO' or 'NEGATIVO'}")
        with open(os.path.join("data",
        "palabras_y_puntajes.txt"), "a", encoding="utf-8") as f:
            f.write(f"{palabra},{puntaje}\n")

        print(f"Palabra '{palabra}' con puntaje {puntaje} añadida con éxito.")

    def eliminar_palabra_sentimiento(tabla_sentimientos):
        palabra = input("Ingrese la palabra a eliminar: ").strip()
        tabla_sentimientos.eliminar_palabra(palabra)
        eliminar_archivo("tabla_lexemas.txt")
        eliminar_archivo("afd.json")
        with open(os.path.join("data",
        "palabras_y_puntajes.txt"), "r", encoding="utf-8") as f:
            lineas = f.readlines()

        with open(os.path.join("data"
        "palabras_y_puntajes.txt"), "w", encoding="utf-8") as f:
            for linea in lineas:
                if not linea.startswith(f"{palabra},"):
                    f.write(linea)

        print(f"Palabra '{palabra}' eliminada con éxito.")
```

Módulo de Gestión de Sentimientos

Este módulo permite la carga, adición y eliminación de palabras con sus puntajes de sentimiento en una `TablaSentimientos`, así como la actualización de un Autómata Finito Determinista (AFD) para clasificar palabras. También se realiza la persistencia de datos en archivos.

Funciones Principales

`eliminar_archivo(nombre_archivo)`

Elimina un archivo en la carpeta `output`, ignorando errores si el archivo no existe o si ocurre otra excepción.

```
try:
    os.remove(os.path.join("output", nombre_archivo))
except FileNotFoundError:
    pass
except Exception as e:
    pass
```

cargar_palabras_y_puntajes(archivo)

Carga palabras y puntajes desde un archivo en la carpeta `data`, creando una instancia de `TablaSentimientos`. El archivo debe contener líneas en el formato `palabra,puntaje`. La función retorna la tabla de sentimientos.

agregar_palabra_sentimiento(tabla_sentimientos, afd)

Esta función permite agregar una nueva palabra con su puntaje de sentimiento a la `TablaSentimientos` y al AFD. Realiza los siguientes pasos:

- Elimina los archivos `tabla_lexemas.txt` y `afd.json` si existen, para evitar conflictos con la actualización.
- Añade la palabra a la tabla de sentimientos con el puntaje especificado.
- Añade una transición al AFD en su estado inicial, con un nuevo estado basado en si el puntaje es positivo o negativo.
- Guarda la palabra y su puntaje en el archivo `palabras_y_puntajes.txt`.

Al finalizar, muestra un mensaje confirmando la adición exitosa de la palabra.

eliminar_palabra_sentimiento(tabla_sentimientos)

Permite eliminar una palabra de la `TablaSentimientos` y del archivo de persistencia:

- Elimina los archivos `tabla_lexemas.txt` y `afd.json` para forzar una actualización.
- Busca y elimina la palabra en el archivo `palabras_y_puntajes.txt`, escribiendo las líneas que no corresponden a la palabra eliminada.

Finalmente, muestra un mensaje confirmando la eliminación exitosa de la palabra.

2.5.6. Función principal

```
import os
from models.afd import TokenizadorAFD
from models.file_utils
import agregar_palabra_sentimiento, eliminar_archivo,
    cargar_palabras_y_puntajes,
eliminar_palabra_sentimiento
from models.menu import mostrar_menu
from models.analisis import analizar_sentimiento, verificar_protocolo


def main():
    tokenizador = TokenizadorAFD()
```

```
tabla_sentimientos =
        cargar_palabras_y_puntajes("palabras_y_puntajes.txt")

while True:
    opcion = mostrar_menu()

    if opcion == '1':
        tokenizador.afd.vaciar()
        tokenizador = TokenizadorAFD()
        tabla_sentimientos =
         cargar_palabras_y_puntajes("palabras_y_puntajes.txt")
        eliminar_archivo("tabla_lexemas.txt")
        eliminar_archivo("afd.json")
        eliminar_archivo("reporte.txt")
        nombre = input("Ingresa el nombre del archivo de entrada
        (con extensión .txt): ")
        archivo = f"test/{nombre}"
        if os.path.exists(archivo):
            with open(archivo, "r", encoding="utf-8") as archivo_texto:
                texto = archivo_texto.read().replace('\n', ' ')
                lexemas = tokenizador.tokenizador(texto)
                tabla_lexemas = tokenizador.afd.generar_tabla_lexemas()
                with open(os.path.join("output","tabla_lexemas.txt"),
                 "w", encoding="utf-8") as archivo_salida:
                    archivo_salida.write(f"{'Lexema':<20}
                    {'Token':<15} {'Patron'}\n")
                    archivo_salida.write("=" * 50 + "\n")
                    for entrada in tabla_lexemas:
                        archivo_salida.write(f"{entrada['Lexema']:<20}
                        {entrada['Token']:<15} {entrada['Patron']}\n")
                print("-"*64)
                resultado_sentimiento = analizar_sentimiento
                (lexemas, tabla_sentimientos, tokenizador)
                print("Detección de sentimientos: ")
                print(f"Sentimiento general:
                {resultado_sentimiento['sentimiento']}")
                print(f"Puntaje total:
                {resultado_sentimiento['puntaje_total']}")
                print(f"Palabras positivas:
                {resultado_sentimiento['palabras_positivas']}")
                print(f"Palabras negativas:
                {resultado_sentimiento['palabras_negativas']}")

                resultado_protocolo = verificar_protocolo
                (lexemas,tokenizador)
                print(f"Verificación del protocolo de
```

```
Atención (Agente): ")
print(f" - Saludo: {'OK'
if resultado_protocolo['saludo'] else 'No detectado'}")
print(f" - Identificación: {'OK' if
resultado_protocolo['identificacion']
else 'No detectado'}")
print(f" - Despedida: {'OK'
if resultado_protocolo['despedida'] else
'No detectado'}")
print(f" - Palabras rudas:
{'Detectadas'
if resultado_protocolo['palabras_prohibidas']
else 'Ninguna detectada'}")
print("-"*64)

output_filename = "reporte.txt"
with open(os.path.join("output",output_filename),
"w", encoding="utf-8") as archivo_salida:
    archivo_salida.write(f"Sentimiento general:
        {resultado_sentimiento['sentimiento']}\n")
    archivo_salida.write(f"Puntaje total:
        {resultado_sentimiento['puntaje_total']}\n")
    archivo_salida.write(f"Palabras positivas:
        {resultado_sentimiento['palabras_positivas']}\n")
    archivo_salida.write(f"Palabras negativas:
        {resultado_sentimiento['palabras_negativas']}\n")
    archivo_salida.write(f"Verificación del
        protocolo de Atención (Agente):\n")
    archivo_salida.write(f" - Saludo:
        {'OK' if resultado_protocolo['saludo']
        else 'No detectado'}\n")
    archivo_salida.write(f" - Identificación:
        {'OK' if resultado_protocolo['identificacion']
        else 'No detectado'}\n")
    archivo_salida.write(f" - Despedida:
        {'OK' if resultado_protocolo['despedida']
        else 'No detectado'}\n")
    archivo_salida.write(f" - Palabras rudas:
        {'Detectadas'
        if resultado_protocolo['palabras_prohibidas']
        else 'Ninguna detectada'}\n")

print(f"[*] Archivo de salida guardado en:
            {output_filename}")
afd_json = "afd.json"
tokenizador.afd.guardar_en_json(afd_json)
```

```
                    print(f"[*] AFD guardado en: {afd_json}")
                    print("[*] Tabla de lexemas guardada en:
                            tabla_lexemas.txt")
            else:
                print(f"El archivo {archivo} no existe.")

        elif opcion == '2':
            tokenizador.afd.vaciar()
            agregar_palabra_sentimiento(tabla_sentimientos, tokenizador.afd)
            tokenizador = TokenizadorAFD()
            tabla_sentimientos = cargar_palabras_y_puntajes
            ("palabras_y_puntajes.txt")

        elif opcion == '3':
            tokenizador.afd.vaciar()
            eliminar_palabra_sentimiento(tabla_sentimientos)
            tokenizador = TokenizadorAFD()
            tabla_sentimientos = cargar_palabras_y_puntajes
            ("palabras_y_puntajes.txt")

        elif opcion == '4':
            print("Saliendo del programa.")
            break

        else:
            print("Opción no válida. Intente de nuevo.")

if __name__ == "__main__":
    main()
```

Código Principal del Análisis de Sentimientos y Protocolo de Atención

Este código implementa el flujo principal de un sistema de análisis de sentimientos y verificación de protocolo de atención utilizando un AFD (Autómata Finito Determinista) y una tabla de sentimientos.

Módulos Importados

- `os`: Proporciona funciones para interactuar con el sistema de archivos.
- `models.afd`: Define el AFD y su tokenizador.
- `models.file_utils`: Contiene utilidades para agregar y eliminar palabras de la tabla de sentimientos y manejar archivos.

- `models.menu`: Contiene el menú de opciones.
- `models.analisis`: Incluye funciones para analizar sentimientos y verificar protocolos.

Función `main()`

La función `main()` controla el flujo del programa a través de un bucle que ofrece las siguientes opciones:

- **Opción 1:** Procesar un archivo de texto de entrada.
  - Limpia el AFD y carga la tabla de sentimientos desde `palabras_y_puntajes.txt`.
  - Elimina archivos temporales como `tabla_lexemas.txt`, `afd.json` y `reporte.txt`.
  - Lee el archivo de texto especificado, tokeniza el contenido usando el AFD, y guarda la tabla de lexemas generada en `tabla_lexemas.txt`.
  - Realiza el análisis de sentimientos e imprime los resultados en consola.
  - Verifica el protocolo de atención y muestra en consola si el agente cumple con los requisitos (saludo, identificación, despedida y la ausencia de palabras prohibidas).
  - Guarda el reporte de resultados en `reporte.txt` y las transiciones usadas del AFD en un archivo JSON.
- **Opción 2:** Agregar una nueva palabra a la tabla de sentimientos y al AFD.
  - Limpia el AFD, llama a `agregar_palabra_sentimiento`, recarga el AFD y la tabla de sentimientos.
- **Opción 3:** Eliminar una palabra de la tabla de sentimientos y el archivo de persistencia.
  - Limpia el AFD, llama a `eliminar_palabra_sentimiento`, y recarga el AFD y la tabla de sentimientos.
- **Opción 4:** Salir del programa.

La función `main()` permite interactuar con el sistema de análisis, actualizando el AFD y la tabla de sentimientos de acuerdo con las modificaciones realizadas.

Notas Importantes

El código maneja los resultados de análisis de sentimientos y verificación de protocolo tanto en la consola como en archivos de texto para su revisión.

2.6. Resultados

El siguiente es el resultado del análisis de un ejemplo de conversación entre un agente y un cliente, en el cual se detecta un sentimiento general positivo. Los tokens asociados a palabras positivas superan en número y en puntaje a las palabras negativas, lo que sugiere una interacción satisfactoria. Además, se verifica que el agente ha cumplido con el protocolo de atención al cliente, incluyendo saludo, identificación, y despedida, sin utilizar palabras prohibidas.

```
Sentimiento general: Positivo
Puntaje total: 12
Palabras positivas: ['bien', 'bien', 'genial', 'bien', 'bien',
'perfecto', 'paciencia', 'excelente']
Palabras negativas: ['problema', 'problema']
Verificación del protocolo de Atención (Agente):
    - Saludo: OK
    - Identificación: OK
    - Despedida: OK
    - Palabras rudas: Ninguna detectada
```

3. Conclusión

Este proyecto implementa un sistema integral de análisis de sentimientos y verificación de protocolos de atención mediante el uso de un Autómata Finito Determinista (AFD) y una tabla de sentimientos. La arquitectura del sistema permite la identificación y clasificación de palabras con carga emocional, así como la detección de palabras inapropiadas en las interacciones. Además, el sistema verifica el cumplimiento de los elementos fundamentales del protocolo de atención (saludo, identificación y despedida), proporcionando una evaluación detallada de las respuestas del agente.

La flexibilidad del sistema para agregar o eliminar palabras de la tabla de sentimientos y actualizar el AFD en tiempo real lo hace adaptable a nuevas necesidades o criterios específicos de análisis. Esta capacidad de modificación y persistencia de datos asegura que el sistema se pueda ajustar continuamente para mejorar la precisión de las evaluaciones y adaptarse a diferentes contextos de uso.

En conjunto, este sistema representa una herramienta robusta y adaptable para el análisis automatizado de interacciones en centros de contacto, con potencial de ser extendido o integrado en aplicaciones más amplias de procesamiento de lenguaje natural (PLN) o de mejora de calidad en la atención al cliente.