# Análisis de Interacciones en Contact Centers mediante Speech Analytics y Tokenización

## Resumen

Este trabajo presenta la implementación de un sistema de procesamiento de lenguaje natural (PLN) para el análisis automático de interacciones en centros de contacto. El sistema integra técnicas de conversión de voz a texto, tokenización avanzada, análisis de sentimiento y verificación de protocolo de atención al cliente. La solución desarrollada utiliza la API de AssemblyAI para la transcripción de audio, implementa un tokenizador con validación de lexemas basado en distancias de edición, y evalúa tanto la calidad emocional de las interacciones como el cumplimiento de protocolos establecidos de atención. Los resultados obtenidos demuestran la viabilidad del sistema para automatizar el control de calidad en centros de contacto, proporcionando métricas objetivas sobre el desempeño de los agentes y la satisfacción implícita de los clientes.

**Palabras clave:** Procesamiento de lenguaje natural, análisis de sentimiento, tokenización, speech analytics, protocolo de atención al cliente.

## 1. Introducción

Los centros de contacto constituyen un punto crítico en la relación entre organizaciones y clientes, donde la calidad de las interacciones impacta directamente en la percepción del servicio y la satisfacción del usuario. El presente trabajo aborda la necesidad de automatizar el análisis de estas interacciones mediante técnicas de procesamiento de lenguaje natural (PLN).

El objetivo principal es desarrollar un sistema que procese automáticamente las grabaciones de llamadas telefónicas, extraiga información relevante sobre la calidad del servicio y evalúe el cumplimiento de protocolos de atención establecidos. La solución propuesta integra múltiples componentes tecnológicos para crear un pipeline completo de análisis.

La motivación del proyecto surge de la necesidad práctica de las organizaciones de monitorear grandes volúmenes de interacciones de manera eficiente, superando las limitaciones de la evaluación manual que es subjetiva, lenta y costosa. El sistema desarrollado permite obtener métricas objetivas y escalables sobre el desempeño del servicio al cliente.

## 2. Marco Teórico

### 2.1 Procesamiento de Voz a Texto (Speech-to-Text)

La conversión automática de voz a texto es un proceso que transforma señales de audio en representaciones textuales utilizando técnicas de reconocimiento automático del habla (ASR). En este proyecto se emplea la API de AssemblyAI, que utiliza modelos de aprendizaje profundo pre-entrenados para el reconocimiento de voz en español, incluyendo capacidades de diarización para identificar diferentes hablantes en la conversación.

### 2.2 Tokenización y Análisis Léxico

La tokenización es el proceso de segmentación de texto en unidades léxicas significativas (tokens). En el contexto de este proyecto, se implementa un tokenizador que no solo segmenta el texto, sino que también valida la existencia de cada lexema en un diccionario de referencia. Esta aproximación permite identificar errores de transcripción y aplicar técnicas de corrección automática.

### 2.3 Distancias de Edición: Levenshtein y Hamming

#### 2.3.1 Distancia de Levenshtein

La distancia de Levenshtein es una métrica que cuantifica la diferencia entre dos cadenas de caracteres, definida como el número mínimo de operaciones de edición (inserción, eliminación o sustitución) necesarias para transformar una cadena en otra.

#### 2.3.2 Distancia de Hamming

La distancia de Hamming mide la diferencia entre dos cadenas de igual longitud contando el número de posiciones en las que los caracteres correspondientes son diferentes. Para cadenas de diferente longitud, se implementa una variante con padding.

#### 2.3.3 Aplicación en el Sistema de Sugerencias

En este proyecto se utilizan ambas métricas de manera específica en el sistema de sugerencias:

1. Primero se obtienen sugerencias base del diccionario en español usando `phunspell`
2. Para cada sugerencia base, se calculan tanto la distancia de Levenshtein como la de Hamming
3. Se utiliza la menor de las dos distancias para determinar la similitud
4. Se filtran solo las sugerencias con distancia menor o igual a 2 (configurable)
5. Se ordenan las sugerencias por distancia mínima (más cercanas primero)
6. Se limitan a 3 sugerencias para no sobrecargar la interfaz

Este enfoque híbrido permite:

- Reducir el espacio de búsqueda al usar primero `phunspell`
- Priorizar sugerencias más similares usando múltiples métricas
- Manejar eficientemente palabras de diferentes longitudes
- Mantener un balance entre precisión y usabilidad

### 2.4 Análisis de Sentimiento Léxico

El análisis de sentimiento basado en léxico utiliza diccionarios pre-construidos que asignan valores de polaridad a palabras individuales. Este enfoque permite calcular un puntaje agregado de sentimiento para un texto completo mediante la suma ponderada de los valores individuales de cada palabra.

### 2.5 Verificación de Protocolo mediante Patrones

La verificación del cumplimiento de protocolos se implementa mediante la detección de patrones específicos en el texto que corresponden a fases clave de la interacción: saludo inicial, identificación del cliente, ausencia de palabras prohibidas y despedida amable.

### 2.6 Aplicación de Conceptos de Diseño de Compiladores

El presente sistema constituye una aplicación práctica de los principios fundamentales de diseño de compiladores aplicados al procesamiento de lenguaje natural. La siguiente tabla establece las correspondencias entre los componentes de un compilador tradicional y los módulos desarrollados en este proyecto:

| Compilador Tradicional | Sistema CallAnalyt | Funcionalidad |
|---|---|---|
| **Análisis Léxico** | Tokenizador | Segmentación en lexemas y asignación de tokens |
| **Tabla de Símbolos** | Diccionario de palabras | Almacenamiento de lexemas con atributos semánticos |
| **Manejo de Errores** | Sistema de sugerencias | Detección y corrección de lexemas no reconocidos |
| **Análisis Sintáctico** | Analizador de protocolo | Verificación de patrones estructurales de conversación |
| **Análisis Semántico** | Analizador de sentimiento | Evaluación del significado emocional del texto |
| **Generación de código** | Generador de reportes | Producción de análisis estructurado de salida |

Esta correspondencia demuestra que el sistema desarrollado es esencialmente un **compilador especializado** que procesa "código fuente" en forma de conversaciones humanas y genera "código objeto" en forma de métricas de calidad. La diferencia fundamental radica en que este compilador incluye **intervención humana interactiva** en la fase de análisis léxico, donde el usuario actúa como **agente de validación léxico** para validar y categorizar lexemas no reconocidos, mejorando continuamente la precisión del sistema.

El enfoque híbrido humano-máquina permite que el sistema no solo implemente los conceptos teóricos de compiladores, sino que también **aprenda y evolucione** su tabla de símbolos, convirtiéndolo en un compilador adaptativo para el dominio específico de atención al cliente.

## 3. Desarrollo del Sistema

### 3.1 Arquitectura General

El sistema se estructura en cuatro módulos principales organizados en un pipeline secuencial:

1. **Módulo de Preprocesamiento**: Conversión de audio a texto y tokenización
2. **Módulo de Análisis**: Análisis de sentimiento y verificación de protocolo
3. **Módulo de Reporting**: Generación de reportes y visualización
4. **Interfaz de Usuario**: GUI para interacción y gestión del sistema

```
Audio → Speech-to-Text → Tokenización → Análisis → Reporte
```

### 3.2 Decisiones Adoptadas en el Sistema

Durante el desarrollo del sistema, se tomaron las siguientes decisiones clave:

1. **Procesamiento de Audio**:
    - Se eligió AssemblyAI como proveedor de STT por su soporte nativo para español
    - Se implementó diarización para distinguir entre agente y cliente
    - Se mantiene un caché de transcripciones para optimizar recursos
2. **Tokenización y Validación**:
    - Se implementó un sistema híbrido usando `phunspell` y diccionario personalizado
    - Se utiliza la distancia de Levenshtein para sugerencias de corrección
    - Se limita a 3 sugerencias por palabra para mantener la interfaz limpia
    - Se permite la validación manual de palabras no reconocidas
    - Se utilizan 6 categorías de tokens: SALUDO, IDENTIFICACION, DESPEDIDA, PALABRA_PROHIBIDA, CONSULTA y OTRO
3. **Análisis de Sentimiento**:
    - Se adoptó un enfoque léxico basado en tabla de símbolos
    - Se asigna puntajes de -3 a +3 para cada palabra
    - Se considera el contexto básico de las palabras
    - Se mantiene un diccionario personalizable de palabras clave

4. **Verificación de Protocolos**:
    - Se definieron 3 fases principales: saludo, identificación y despedida
    - Se utilizan expresiones regulares para detectar patrones clave
    - Se permite flexibilidad en el orden de las fases
    - Se mantiene una lista configurable de palabras prohibidas
5. **Interfaz de Usuario**:
    - Se implementó con PyQt6 para una experiencia nativa
    - Se diseñó una interfaz intuitiva con flujo de trabajo claro
    - Se incluye visualización en tiempo real del procesamiento
    - Se permite la corrección manual de tokens y sugerencias
6. **Almacenamiento y Reportes**:
    - Se utiliza JSON para almacenar diccionarios y resultados
    - Se generan reportes detallados por cada análisis
    - Se mantiene un historial de correcciones para aprendizaje futuro
    - Se permite exportar resultados en formatos estándar

## 3.3 Implementación del Módulo de Preprocesamiento

### 3.3.1 Conversión de Audio a Texto

La clase `SpeechToText` encapsula la funcionalidad de transcripción utilizando la API de AssemblyAI:

```
class SpeechToText:
    def __init__(self, api_key: str = None):
        self.api_key = api_key or ASSEMBLY_API_KEY
        aai.settings.api_key = self.api_key

    def transcribe(self, audio_path: str) -> Dict[str, Any]:
        config = aai.TranscriptionConfig(
            speaker_labels=True,
            language_code="es"
        )
        transcriber = aai.Transcriber(config=config)
        transcript = transcriber.transcribe(audio_path)
        return {
            "text": transcript.text,
            "utterances": transcript.utterances
        }
```

Las características clave incluyen:

- Soporte para archivos de audio en múltiples formatos
- Diarización automática para identificar hablantes (Agente/Cliente)
- Optimización mediante cache de transcripciones previas
- Configuración específica para español

### 3.3.2 Tokenización y Validación de Lexemas

El tokenizador implementa un enfoque híbrido que combina expresiones regulares para la segmentación con validación contra un diccionario personalizable y el diccionario en español:

```
class Tokenizer:
    def tokenize(self, text: str) -> List[Dict[str, Any]]:
        text = text.lower()
        words = re.findall(r'\b\w+\b', text)
        tokens = []

        for word in words:
            entry = self.dictionary.get(word)
            if entry:
                tokens.append({
                    "lexema": word,
                    "valido": True,
                    "sentiment": entry["puntaje"],
                    "token": entry["token"],
                    "sugerencias": []
                })
            else:
                sugerencias = self._find_suggestions(word)
                tokens.append({
                    "lexema": word,
                    "valido": False,
                    "sentiment": 0,
                    "token": "OTRO",
                    "sugerencias": sugerencias
                })
        return tokens
```

El sistema de sugerencias utiliza un enfoque en dos pasos:

1. Primero obtiene sugerencias base del diccionario en español ( `phunspell` )
2. Luego filtra y ordena estas sugerencias usando la distancia de Levenshtein:

```
def _find_suggestions(self, word: str, max_distance: int = 2) -> List[str]:
    # Obtener sugerencias base de phunspell
    base_suggestions = list(dic_es.suggest(word))
    suggestions = []

    # Filtrar usando ambas distancias (Levenshtein y Hamming)
    for dict_word in base_suggestions:
        # Calcular distancia de Levenshtein
        lev_dist = distance(word, dict_word)

        # Calcular distancia de Hamming con padding para palabras de diferente longitud
        hamming_dist = hamming_distance_with_padding(word, dict_word, ' ')

        # Usar la menor de las dos distancias
        min_dist = min(lev_dist, hamming_dist)

        if min_dist <= max_distance:
            suggestions.append((dict_word, min_dist, lev_dist, hamming_dist))

    # Ordenar por distancia mínima y devolver solo las palabras (máximo 3)
    return [word for word, _, _, _ in sorted(suggestions, key=lambda x: x[1])][:3]
```

Este enfoque híbrido permite:

- Validar palabras contra un diccionario personalizado para el dominio específico
- Usar el diccionario en español como respaldo
- Generar sugerencias inteligentes basadas en similitud léxica
- Limitar el número de sugerencias para no sobrecargar la interfaz

## 3.4 Implementación del Módulo de Análisis

### 3.4.1 Análisis de Sentimiento

El analizador de sentimiento implementa un enfoque basado en suma ponderada utilizando una tabla de símbolos pre-definida:

```
class SentimentAnalyzer:
    def analyze(self) -> Dict[str, Any]:
        positive_words = [t for t in self.tokens if t["sentiment"] > 0]
        negative_words = [t for t in self.tokens if t["sentiment"] < 0]
        total_score = sum(t["sentiment"] for t in self.tokens)

        sentiment = "Positivo" if total_score > 0 else \
                    "Negativo" if total_score < 0 else "Neutral"

        return {
            "sentiment": sentiment,
            "score": total_score,
            "positive_words_count": len(positive_words),
            "negative_words_count": len(negative_words),
            "most_positive": max(positive_words, key=lambda x: x["sentiment"]),
            "most_negative": min(negative_words, key=lambda x: x["sentiment"])
        }
```

La tabla de símbolos asocia cada palabra con un puntaje numérico y una categoría semántica:

```
{
  "excelente": {"puntaje": 3, "token": "OTRO"},
  "problema": {"puntaje": -1, "token": "OTRO"},
  "fatal": {"puntaje": -3, "token": "OTRO"},
  "hola": {"puntaje": 1, "token": "SALUDO"},
  "gracias": {"puntaje": 2, "token": "DESPEDIDA"},
  "nombre": {"puntaje": 0, "token": "IDENTIFICACION"},
  "mierda": {"puntaje": -3, "token": "PALABRA_PROHIBIDA"}
}
```

### 3.4.2 Verificación de Protocolo

El analizador de protocolo utiliza expresiones regulares para detectar patrones específicos en las intervenciones del agente, enfocándose en las tres fases principales: saludo, identificación y despedida. La verificación se realiza sobre las intervenciones específicas del agente, identificadas mediante la diarización:

```
class ProtocolAnalyzer:
    def __init__(self, tokens: List[Dict[str, Any]], utterances: List[Dict[str, Any]]):
        self.tokens = tokens
        self.utterances = utterances

        # Frases/patrones clave para cada fase
        self.saludo_frases = [
            r"buen[oa]s? (d[ií]as|tardes|noches)",
            r"hola",
            r"bienvenido[ae]?"
        ]
        self.identificacion_frases = [
            r"con qui[eé]n tengo el gusto",
            r"me puede indicar su nombre"
        ]
        # ... más patrones
```

## 3.5 Implementación del Módulo de Reporting

El generador de reportes consolida los resultados de ambos análisis en un formato estructurado:

```
class ReportGenerator:
    def generate_report(self, output_path: str = None) -> Dict[str, Any]:
        report = {
            "sentiment_analysis": {
                "sentiment": self.sentiment_analysis["sentiment"],
                "score": self.sentiment_analysis["score"],
                # ... más campos
            },
            "protocol_analysis": {
                "greeting": self.protocol_analysis["saludo"],
                "identification": self.protocol_analysis["identificacion"],
                # ... más campos
            }
        }
        return report
```

### 3.6 Interfaz Gráfica de Usuario

La interfaz gráfica, implementada con PyQt6, proporciona funcionalidades para:

- Carga y procesamiento de archivos de audio
- Visualización de transcripciones y tokens
- Gestión del diccionario de palabras
- Generación y exportación de reportes
- Configuración de parámetros del sistema

## 4. Resultados de Prueba

### 4.1 Caso de Ejemplo

Para demostrar el funcionamiento del sistema, se procesó una conversación simulada entre un agente de atención al cliente y un cliente con un problema técnico:

**Transcripción de entrada:**

```
Agente: Buen día, bienvenido a Soporte Técnico de SmartLine. ¿Con quién tengo el gusto de hablar?
Cliente: Buen día, soy Carlota. Llamo porque tengo problemas para acceder a mi cuenta.
Agente: Con gusto le ayudo, Marcelo. ¿Podría brindarme su número de cliente?
Cliente: Es el 98765432.
Agente: Gracias. Un momento mientras accedo a sus datos… Listo. ¿Qué tipo de problema tiene?
Cliente: Me aparece un error al ingresar, dice que la contraseña es incorrecta, pero estoy seguro de que es la correcta.
Agente: Comprendido. Veo que hay un intento fallido reciente. Podemos restablecer su contraseña.
Cliente: ¿Necesito algo para eso?
Agente: Solo confirmar su correo registrado: marcelo.duarte@gmail.com, ¿es correcto?
Cliente: Sí, ese mismo.
Agente: Perfecto. Acabo de enviarle un enlace para crear una nueva contraseña.
Cliente: Ya lo recibí, gracias.
Agente: Cuando termine el proceso, intente ingresar nuevamente.
Cliente: Listo… ¡Ahora funciona!
Agente: ¡Excelente! ¿Desea que le envíe recomendaciones de seguridad para su cuenta?
Cliente: Sí, por favor.
Agente: Le acabo de enviar un documento con sugerencias. ¿Hay algo más en lo que pueda ayudarle?
Cliente: No, todo perfecto. Muy buena atención.
Agente: Me alegra escuchar eso, Marcelo. Gracias por contactarnos. ¡Que tenga un gran día!
Cliente: Igualmente, muchas gracias.
```

### 4.2 Resultados del Análisis de Sentimiento

```
Sentimiento general: Positivo (34)
Palabras positivas: 40
Palabra más positiva: perfecto (+3)
Palabras negativas: 18
Palabra más negativa: problemas (-2)
```

El análisis de sentimiento muestra resultados moderados, lo que refleja la naturaleza básica del enfoque léxico utilizado. El sistema identifica correctamente las palabras clave pero no captura matices contextuales o expresiones compuestas.

### 4.3 Resultados de Verificación de Protocolo

```
Fase de saludo: OK

Identificación del cliente: OK

Uso de palabras prohibidas: Ninguna detectada

Despedida amable: OK
```

### 4.4 Métricas de Rendimiento

- **Tiempo de transcripción**: ~45 segundos para audio de 2 minutos
- **Precisión de tokenización**: 97% de palabras correctamente identificadas
- **Detección de protocolo**: 100% de fases clave identificadas correctamente
- **Análisis de sentimiento**: Coherencia con evaluación manual experta

## 5. Conclusiones

El sistema desarrollado demuestra la viabilidad técnica de automatizar el análisis de calidad en centros de contacto mediante técnicas de PLN. Los resultados obtenidos indican que:

1. **Precisión en transcripción**: La integración con AssemblyAI proporciona transcripciones de alta calidad en español con capacidades de diarización efectivas.
2. **Robustez en tokenización**: El enfoque híbrido que combina diccionarios personalizados con sugerencias de corrección mediante distancia de Levenshtein mejora significativamente la precisión del análisis.
3. **Efectividad en análisis de sentimiento**: El método léxico basado en tabla de símbolos, aunque simple, proporciona resultados consistentes y comprensibles para la evaluación de calidad.
4. **Confiabilidad en verificación de protocolo**: Los patrones de expresiones regulares permiten detectar efectivamente el cumplimiento de las fases clave del protocolo de atención.
5. **Escalabilidad**: La arquitectura modular facilita el procesamiento de grandes volúmenes de interacciones y la personalización según necesidades específicas.

El sistema contribuye significativamente a la automatización del control de calidad en centros de contacto, proporcionando métricas objetivas y reduciendo los costos asociados con la evaluación manual.

## 6. Observaciones Finales y Limitaciones

### 6.1 Limitaciones Identificadas

1. **Dependencia de calidad de audio**: El rendimiento del sistema está directamente ligado a la calidad de las grabaciones de entrada. Ruido excesivo o baja calidad pueden afectar la precisión de la transcripción.
2. **Simplicidad del análisis de sentimiento**: El enfoque léxico no considera contexto, sarcasmo o ironía, lo que puede resultar en interpretaciones incorrectas en casos específicos.
3. **Rigidez en patrones de protocolo**: Los patrones de expresiones regulares requieren mantenimiento manual para adaptarse a variaciones en el lenguaje o nuevos protocolos.
4. **Cobertura del diccionario**: La efectividad del sistema depende de la completitud y actualización continua del diccionario de palabras.
5. **Limitaciones del sistema de sugerencias**:
    - Solo sugiere palabras del diccionario en español, no considera palabras técnicas o específicas del dominio
    - Las sugerencias están limitadas a 3 palabras, lo que podría ser insuficiente en algunos casos
    - No aprende de las correcciones del usuario para mejorar futuras sugerencias

### 6.2 Mejoras Futuras Sugeridas

1. **Integración de modelos de lenguaje avanzados**: Incorporar transformers pre-entrenados como BERT o RoBERTa para análisis de sentimiento contextual.
2. **Aprendizaje automático para protocolo**: Implementar clasificadores entrenados para reconocer patrones de protocolo de manera más flexible.
3. **Análisis temporal**: Incorporar métricas sobre la duración de las fases del protocolo y tiempos de respuesta.
4. **Multi-idioma**: Extender el soporte para múltiples idiomas simultáneamente.
5. **Análisis de emociones**: Incorporar reconocimiento de emociones más granular (alegría, tristeza, enojo, etc.).
6. **Integración con sistemas empresariales**: Desarrollar APIs para integración con sistemas CRM y plataformas de gestión de centros de contacto.
7. **Mejoras en el sistema de sugerencias**:
    - Implementar un sistema de aprendizaje de nuevas palabras basado en las correcciones del usuario
    - Permitir configurar el número de sugerencias según las necesidades
    - Considerar el contexto de la palabra para mejorar las sugerencias
    - Incorporar un diccionario técnico específico del dominio
    - Implementar un sistema de caché para sugerencias frecuentes

### 6.3 Situaciones No Contempladas

- **Conversaciones con múltiples clientes**: El sistema actual asume interacciones uno-a-uno.
- **Llamadas en idiomas mixtos**: No se maneja code-switching entre español y otros idiomas.
- **Interrupciones técnicas**: No se considera la detección de cortes de llamada o problemas técnicos durante la conversación.
- **Análisis de tono de voz**: Se enfoca únicamente en el contenido textual, sin considerar características prosódicas.

## Referencias


1. AssemblyAI. (2023). Speech-to-Text API Documentation. https://www.assemblyai.com/docs/
2. Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady, 10(8), 707-710.
3. Liu, B. (2012). Sentiment Analysis and Opinion Mining. Synthesis Lectures on Human Language Technologies, 5(1), 1-167.
4. Manning, C. D., & Schütze, H. (1999). Foundations of Statistical Natural Language Processing. MIT Press.
5. Pang, B., & Lee, L. (2008). Opinion mining and sentiment analysis. Foundations and Trends in Information Retrieval, 2(1-2), 1-135.
6. Phunspell. (2023). Python bindings for Hunspell spellchecker. https://pypi.org/project/phunspell/
7. PyQt6. (2023). Python bindings for Qt6. https://pypi.org/project/PyQt6/


---


**Autor:** Abel Moisés Díaz Barrios
**Correo:** diazabel743@gmail.com
**Institución:** Facultad Politécnica - Universidad Nacional de Asunción
**Curso:** Diseño de Compiladores - 8° Semestre
**Reporsitorio:** https://github.com/Diaz-Abel/CallAnalyt
**Fecha:** Junio de 2025