UNIVERSIDAD PRIVADA FRANZ TAMAYO

FACULTAD DE INGENIERÍA

INGENIERÍA DE SISTEMAS

OBJETIVO DE ANALISIS Y EXTRACCION DE DATOS

ESTUDIANTES: **CARLOS ANDREW LUVI AGUILAR**

MATERIA: **BIG DATA**

LA PAZ – BOLIVIA

Introducción

En la actualidad, los avances tecnológicos en inteligencia artificial, procesamiento digital de señales y aprendizaje automático han abierto nuevas posibilidades en el ámbito de la salud, particularmente en áreas como la logopedia y la fonoaudiología. Estas disciplinas, que se enfocan en el diagnóstico y tratamiento de los trastornos del habla y de la voz, tradicionalmente han dependido de evaluaciones clínicas presenciales y del criterio subjetivo del profesional. Sin embargo, la incorporación de herramientas computacionales permite automatizar y objetivizar gran parte del proceso de evaluación, ofreciendo resultados más precisos, medibles y accesibles.

El presente proyecto tiene como finalidad el desarrollo de un modelo de procesamiento y evaluación de audio que sirva como base para un sistema de logopedia digital. Este sistema está pensado para facilitar el diagnóstico inicial de alteraciones vocales y, además, servir como una herramienta de apoyo durante el tratamiento mediante ejercicios de dictado de voz, retroalimentación visual y sesiones guiadas en video. De esta manera, el paciente puede realizar prácticas desde su hogar, mientras que el profesional puede realizar un seguimiento remoto basado en datos objetivos.

El modelo propuesto está dividido en dos componentes principales. La primera parte se encarga de realizar el procesamiento digital del audio grabado por el usuario, extrayendo variables acústicas fundamentales que describen el comportamiento de la voz. Estos parámetros, tales como la frecuencia fundamental (F0), el jitter, el shimmer y la relación armónico-ruido (HNR), permiten caracterizar cuantitativamente la calidad y estabilidad vocal.

La segunda parte del modelo se centra en la evaluación de dichas variables, aplicando criterios cuantitativos y reglas estadísticas para determinar si la señal vocal obtenida es apta para el tratamiento o si presenta deficiencias que requieren repetir la grabación o atención clínica adicional. Este proceso de validación asegura que los datos utilizados por el sistema sean confiables y adecuados para las tareas de análisis y entrenamiento posterior.

El enfoque planteado no solo busca optimizar el trabajo de los profesionales de la salud, sino también democratizar el acceso a terapias del habla, brindando una plataforma que combina diagnóstico, seguimiento y rehabilitación de manera integrada. Este documento detalla el funcionamiento técnico de ambas fases del modelo, los fundamentos teóricos detrás de las variables acústicas utilizadas, el flujo de procesamiento de datos y los criterios de evaluación aplicados para determinar la calidad del audio.

Primera Parte: Procesamiento y Extracción de Variables de la Voz

El primer componente del modelo está enfocado en el procesamiento digital del audio grabado por el paciente durante los ejercicios de dictado o pronunciación. Para esta tarea, se emplean librerías de Python especializadas en análisis de voz como librosa, parselmouth (interfaz de Praat) y numpy, que permiten extraer parámetros acústicos fundamentales para el análisis clínico del habla.

Entre las principales variables acústicas consideradas se encuentran:

- Frecuencia fundamental (F0): representa la tasa de vibración de las cuerdas vocales y se expresa en Hertz (Hz). Es un indicador esencial del tono de la voz y permite identificar alteraciones como disfonías o irregularidades en la modulación.
- Jitter: mide la variación ciclo a ciclo en la frecuencia fundamental. Niveles elevados de jitter suelen asociarse a inestabilidad vocal y posibles disfunciones laríngeas.
- Shimmer: cuantifica las variaciones en la amplitud de los ciclos de vibración. Un shimmer alto puede indicar problemas de control respiratorio o deficiencias en el cierre glótico.
- HNR (Relación Armónico-Ruido): evalúa la proporción entre los componentes armónicos de la voz y el ruido presente en la señal. Valores bajos reflejan una voz áspera o soplada, mientras que valores altos indican una voz más estable y clara.

El modelo implementado realiza un proceso de filtrado, segmentación y normalización del audio antes del cálculo de las variables, asegurando la eliminación de ruido externo y la homogeneización de los datos. Posteriormente, las características acústicas extraídas se almacenan en una base de datos o archivo estructurado (por ejemplo, formato CSV) para su posterior análisis.

Esta etapa constituye el núcleo técnico del sistema, pues convierte una señal de voz cruda en un conjunto de valores cuantitativos útiles para el diagnóstico y la evaluación terapéutica.

```python
from flask import Flask, request, jsonify
from flask_cors import CORS
import librosa
import numpy as np
import tempfile
import os
import time
import gc
from scipy.signal import find_peaks
import parselmouth
from parselmouth.praat import call
import warnings
import soundfile as sf
from datetime import datetime

warnings.filterwarnings("ignore")

app = Flask(__name__)
CORS(app)
```

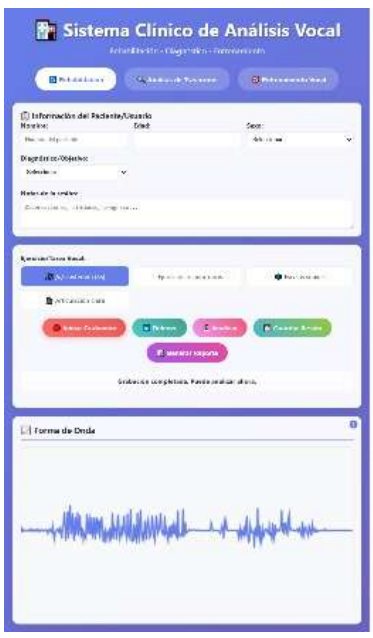
```python
class ClinicalVocalAnalyzer:
    def __init__(self):
        self.sample_rate = 44100
        self.clinical_norms = {
            'pitch': {
                'male': {'min': 85, 'max': 180, 'mean': 120},
                'female': {'min': 165, 'max': 265, 'mean': 215},
                'child': {'min': 250, 'max': 400, 'mean': 300}
            },
            'jitter': {'normal': 1.04, 'pathological': 3.8},
            'shimmer': {'normal': 3.81, 'pathological': 11.0},
            'hnr': {'good': 20, 'acceptable': 15, 'poor': 10}
        }

    def convert_to_serializable(self, obj):
        if isinstance(obj, (np.float32, np.float64)):
            return float(obj)
        elif isinstance(obj, (np.int32, np.int64)):
            return int(obj)
        elif isinstance(obj, np.ndarray):
            return [self.convert_to_serializable(item) for item in obj]
        elif isinstance(obj, dict):
            return {key: self.convert_to_serializable(value) for key, value in obj.items()}
        elif isinstance(obj, list):
            return [self.convert_to_serializable(item) for item in obj]
        else:
            return obj
```

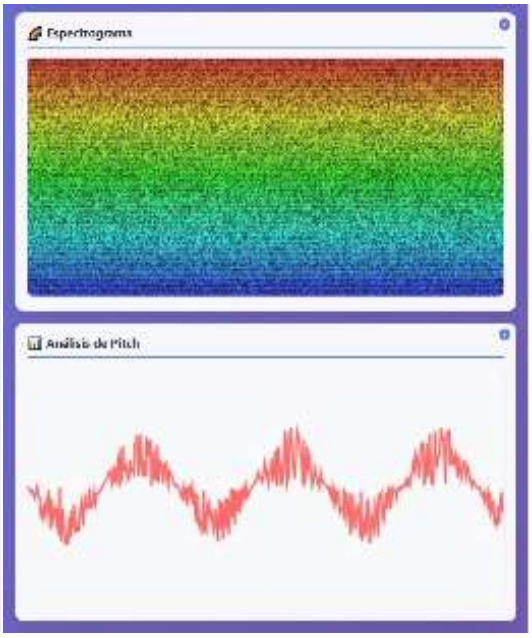
```python
def safe_temp_file_operation(self, y, sr, operation_func):
    temp_file = None
    temp_path = None
    try:
        temp_file = tempfile.NamedTemporaryFile(delete=False, suffix='.wav')
        temp_path = temp_file.name
        temp_file.close()
        sf.write(temp_path, y, sr)
        time.sleep(0.1)
        result = operation_func(temp_path)
        return result
    except Exception:
        return None
    finally:
        if temp_path and os.path.exists(temp_path):
            try:
                gc.collect()
                time.sleep(0.1)
                for attempt in range(3):
                    try:
                        os.chmod(temp_path, 0o777)
                        os.unlink(temp_path)
                        break
                    except PermissionError:
                        if attempt < 2:
                            time.sleep(0.2)
            except:
                pass
```

```python
def calculate_perturbation_measures_corrected(self, y, sr):
    if len(y) < sr * 1.0:
        return {
            'jitter': {'local': 0.0, 'rap': 0.0, 'ppq5': 0.0, 'absolute': 0.0},
            'shimmer': {'local': 0.0, 'local_db': 0.0, 'apq3': 0.0, 'apq5': 0.0}
        }

    y_norm = librosa.util.normalize(y)
    rms_energy = np.sqrt(np.mean(y_norm ** 2))
    if rms_energy < 0.01:
        return {
            'jitter': {'local': 0.0, 'rap': 0.0, 'ppq5': 0.0, 'absolute': 0.0},
            'shimmer': {'local': 0.0, 'local_db': 0.0, 'apq3': 0.0, 'apq5': 0.0}
        }

    f0_values, times, voiced_count = self.calculate_pitch_periods_accurate(y_norm, sr)

    if f0_values is None or len(f0_values) < 10:
        return {
            'jitter': {'local': 0.0, 'rap': 0.0, 'ppq5': 0.0, 'absolute': 0.0},
            'shimmer': {'local': 0.0, 'local_db': 0.0, 'apq3': 0.0, 'apq5': 0.0}
        }

    jitter_local = self.calculate_jitter_accurate(f0_values, 'local')
    jitter_rap = self.calculate_jitter_accurate(f0_values, 'rap')
    jitter_ppq5 = self.calculate_jitter_accurate(f0_values, 'ppq5')
```

```python
def calculate_fundamental_frequency_advanced(self, y, sr):
    def pitch_operation(temp_path):
        try:
            sound = parselmouth.Sound(temp_path)
            pitch = sound.to_pitch(pitch_floor=75.0, pitch_ceiling=600.0)
            pitch_values = pitch.selected_array['frequency']
            pitch_values = pitch_values[pitch_values != 0]

            if len(pitch_values) > 0:
                result = {
                    'mean': float(np.mean(pitch_values)),
                    'std': float(np.std(pitch_values)),
                    'min': float(np.min(pitch_values)),
                    'max': float(np.max(pitch_values)),
                    'median': float(np.median(pitch_values)),
                    'range': float(np.max(pitch_values) - np.min(pitch_values)),
                    'coefficient_variation': float(np.std(pitch_values) / np.mean(pitch_values) * 100) if np.mean(pitch_values) > 0 else 0.0
                }
                return result
            else:
                return {key: 0.0 for key in ['mean', 'std', 'min', 'max', 'median', 'range', 'coefficient_variation']}

        except Exception:
            return {key: 0.0 for key in ['mean', 'std', 'min', 'max', 'median', 'range', 'coefficient_variation']}

    result = self.safe_temp_file_operation(y, sr, pitch_operation)
    return result if result is not None else {key: 0.0 for key in ['mean', 'std', 'min', 'max', 'median', 'range', 'coefficient_variation']}
```

```python
@app.route('/')
def index():
    return """
    <!DOCTYPE html>
    <html>
    <head>
        <title>API Clínica de Análisis Vocal</title>
    </head>
    <body>
        <h1>API Clínica de Análisis Vocal</h1>
        <p>Sistema especializado para rehabilitación vocal, diagnóstico de trastornos y entrenamiento profesional.</p>
        <h3>Endpoints:</h3>
        <ul>
            <li><strong>POST /api/clinical_analyze</strong> - Análisis clínico completo</li>
            <li><strong>GET /api/health</strong> - Estado del sistema</li>
        </ul>
    </body>
    </html>
    """
```





Segunda Parte: Evaluación y Clasificación de las Variables Acústicas

La segunda fase del modelo tiene como propósito evaluar las variables extraídas para determinar si el audio analizado es adecuado para el tratamiento logopédico o si requiere ser repetido debido a deficiencias en la calidad vocal o en la grabación.

Para ello, se implementa un modelo de evaluación basado en umbrales y análisis estadístico, que compara los valores obtenidos con rangos de referencia definidos por la literatura científica o por parámetros adaptativos calibrados según las características individuales del paciente.

Por ejemplo:

- Una frecuencia fundamental fuera del rango normal (entre 85 Hz y 255 Hz, según el sexo y edad del hablante) puede sugerir un tono inadecuado o problemas en la fonación.
- Un jitter superior al 1% o un shimmer mayor al 3% pueden considerarse indicadores de inestabilidad vocal.
- Un HNR menor a 15 dB puede implicar una voz con exceso de ruido o respiración forzada.

El sistema realiza una evaluación automática de estos parámetros, generando un puntaje o calificación que permite clasificar el audio en tres categorías:

1. Aprobado: el audio es claro, estable y útil para continuar el proceso terapéutico.
2. Repetir grabación: la señal presenta irregularidades o ruido excesivo.
3. Posible alteración vocal: se detectan valores anómalos que podrían requerir revisión profesional.

Además, el modelo puede complementarse con técnicas de machine learning (como regresión logística o redes neuronales ligeras) entrenadas con un conjunto de grabaciones previamente evaluadas por expertos. Esto permitiría mejorar la precisión del sistema y adaptarlo a distintas patologías o tipos de voz.

```python
import math
from datetime import datetime

class ClinicalEvaluator:
    """
    Evaluador heurístico/prototipo que recibe los resultados de la parte 1 (analysis_results)
    y decide si el audio es util para tratamiento, su calidad, riesgo y recomendaciones.
    """

    def __init__(self, clinical_norms=None):
        # Normas clinicas (pueden provenir de ClinicalVocalAnalyzer.clinical_norms)
        default_norms = {
            'pitch': {
                'male': {'min': 85, 'max': 180, 'mean': 120},
                'female': {'min': 165, 'max': 265, 'mean': 215},
                'child': {'min': 250, 'max': 400, 'mean': 300}
            },
            'jitter': {'normal': 1.04, 'pathological': 3.8},
            'shimmer': {'normal': 3.81, 'pathological': 11.0},
            'hnr': {'good': 20, 'acceptable': 15, 'poor': 10}
        }
        self.norms = clinical_norms if clinical_norms is not None else default_norms
```

```python
    def compute_quality_score(self, analysis):
        """
        Score de calidad (0-100) basado en jitter, shimmer, hnr, y estabilidad de pitch.
        """
        score = 100.0

        jitter = analysis.get('jitter', 0.0)
        shimmer = analysis.get('shimmer', 0.0)
        hnr = analysis.get('hnr', 0.0)
        pitch_cv = analysis.get('pitch_cv', None)
        audio_duration = analysis.get('session_metadata', {}).get('audio_duration', 0.0)

        # Penalizaciones
        if jitter is not None:
            if jitter > self.norms['jitter']['normal']:
                score -= min((jitter - self.norms['jitter']['normal']) * 15, 35)
        if shimmer is not None:
            if shimmer > self.norms['shimmer']['normal']:
                score -= min((shimmer - self.norms['shimmer']['normal']) * 8, 30)
        if hnr is not None and hnr > 0:
            if hnr < self.norms['hnr']['good']:
                score -= min((self.norms['hnr']['good'] - hnr) * 2, 25)
        if pitch_cv is not None:
            if pitch_cv > 10:
                score -= min((pitch_cv - 10) * 1.8, 20)
        # Duración mínima penaliza
        if audio_duration < 1.0:
            score = 0.0
        elif audio_duration < 2.0:
            score -= 15

        return max(round(score, 2), 0.0)
```

```python
def assess_usability(self, analysis):
    """
    Decide si el audio es 'usable', 'parcialmente_usable' o 'no_usable'
    Basado en quality_score + SNR aproximado si está disponible.
    """
    quality = self.compute_quality_score(analysis)
    verdict = 'no_usable'
    reasons = []

    if quality >= 75:
        verdict = 'usable'
    elif quality >= 45:
        verdict = 'parcialmente_usable'
        reasons.append('Calidad moderada: revisar condiciones de grabación / ruido')
    else:
        verdict = 'no_usable'
        reasons.append('Calidad baja: jitter/shimmer/HNR o duración insuficiente')

    # Revisar indicadores puntuales
    jitter = analysis.get('jitter', 0.0)
    shimmer = analysis.get('shimmer', 0.0)
    hnr = analysis.get('hnr', 0.0)

    if jitter and jitter > self.norms['jitter']['pathological']:
        reasons.append('Jitter muy alto (posible patología)')

    if shimmer and shimmer > self.norms['shimmer']['pathological']:
        reasons.append('Shimmer muy alto (posible patología)')

    if hnr and hnr < self.norms['hnr']['poor']:
        reasons.append('HNR muy bajo (ruido o mala fuente armónica)')

    return {'verdict': verdict, 'quality_score': quality, 'reasons': reasons}
```

```python
def recommend_exercises(self, analysis, patient_info=None):
    """
    Recomendaciones simples basadas en hallazgos.
    """
    recs = []
    jitter = analysis.get('jitter', 0.0)
    shimmer = analysis.get('shimmer', 0.0)
    hnr = analysis.get('hnr', 0.0)
    pitch_mean = analysis.get('pitch', None)

    # Base: si jitter alto -> ejercicios de relajación y control respiratorio
    if jitter > 2.0:
        recs.append({
            'exercise': 'Control respiratorio y fonación sostenida (m: 5-10s)',
            'purpose': 'Reducir variabilidad inter-ciclo (jitter)'
        })
    elif jitter > 1.04:
        recs.append({
            'exercise': 'Ejercicios de resonancia (humming) y fonación suave',
            'purpose': 'Mejorar estabilidad de ciclos'
        })

    # Shimmer: control dinámico de intensidad
    if shimmer > 4.0:
        recs.append({
            'exercise': 'Ejercicios de control de intensidad (sostenido en crescendo/decrescendo)',
            'purpose': 'Reducir variabilidad de amplitud'
        })

    # HNR: si bajo, mejorar apoyo respiratorio y reducir ruido
    if hnr < 15:
        recs.append({
            'exercise': 'Ejercicios de apoyo respiratorio y resonancia (lip trills, straw phonation)',
            'purpose': 'Aumentar proporción armónica/no armónica'
        })
```

Aplicación en el Sistema de Logopedia

El sistema de logopedia desarrollado a partir de este modelo busca integrar de manera armónica los procesos de evaluación, diagnóstico y tratamiento del habla dentro de una plataforma digital interactiva. En ella, el paciente puede acceder a una serie de ejercicios diseñados por especialistas, los cuales se adaptan progresivamente al nivel de dificultad y a las necesidades individuales de cada usuario.

El modelo de análisis de voz se ejecuta de forma automática cada vez que el paciente completa un ejercicio de dictado o pronunciación. El sistema graba el audio, realiza el procesamiento y obtiene los parámetros acústicos que permiten evaluar la calidad y estabilidad de la voz. Posteriormente, la evaluación generada por la segunda parte del modelo determina si el resultado es válido y proporciona retroalimentación inmediata al usuario.

Esta retroalimentación se puede mostrar de manera visual a través de gráficos del tono, la estabilidad y el nivel de ruido en la voz. También puede incluir recomendaciones personalizadas como: “habla más pausado”, “mejora tu respiración”, o “controla la intensidad de la voz”. De esta forma, el usuario recibe una guía clara y dinámica sobre su desempeño.

Asimismo, el sistema contempla una base de datos centralizada donde se almacenan los resultados de cada sesión, permitiendo al profesional de logopedia realizar un seguimiento longitudinal del progreso del paciente. Con esta información, el terapeuta puede ajustar los ejercicios, reforzar áreas problemáticas y evaluar de forma objetiva la evolución del tratamiento.

En una etapa más avanzada, el sistema podría incorporar videos tutoriales o modelos de pronunciación visual que muestren los movimientos de los labios, la lengua y el flujo de aire, facilitando la imitación y el aprendizaje por observación. De igual forma, la integración con tecnologías móviles o web permitiría al paciente continuar con sus terapias desde cualquier lugar, garantizando accesibilidad y continuidad del tratamiento.

En suma, esta aplicación tecnológica convierte el modelo de análisis y evaluación de voz en una herramienta completa para la rehabilitación del habla, combinando la precisión científica del procesamiento acústico con la flexibilidad y dinamismo de las plataformas digitales interactivas.

Conclusiones

El desarrollo del modelo propuesto representa un avance significativo en la aplicación de la inteligencia artificial y el procesamiento digital de señales al campo de la logopedia. La combinación de análisis acústico detallado y evaluación automática de la calidad vocal ofrece una herramienta robusta, precisa y accesible tanto para terapeutas como para pacientes.

El sistema propuesto tiene el potencial de optimizar los procesos de diagnóstico, seguimiento y tratamiento de trastornos del habla, reduciendo la dependencia de evaluaciones subjetivas y aumentando la objetividad de los resultados clínicos. Asimismo, su implementación en un entorno interactivo con ejercicios de voz y retroalimentación audiovisual promueve una mayor motivación y adherencia al tratamiento.

En futuras versiones del modelo se prevé integrar técnicas de aprendizaje profundo para la detección automática de patologías vocales y la personalización de los umbrales de evaluación, garantizando así una herramienta adaptativa, eficiente y escalable en el ámbito de la rehabilitación del habla.