Inteligencia Artificial 2

Procesamiento de Lenguage Natural Herramientas

Pariente Merida Jessica Pamela
Abasto Martinis Simon Eduardo
Huayllas Pinto Kevin

Facultad de ciencias y tecnologias
Universidad Mayor de San Simon
26 de Noviembre del 2025
Cochabamba-Bolivia

Índice

1. Problema que resolvemos

Muchas empresas reciben mensajes por WhatsApp con intenciones distintas, tales como consultas, reclamos, compras o reseñas. Estos mensajes llegan mezclados en la misma bandeja de entrada, lo que puede generar retrasos en las respuestas, sobrecarga del equipo de atención y riesgo de respuestas tardías o inadecuadas.
El objetivo del proyecto es implementar un sistema que analiza y agrupa automáticamente mensajes por intención (venta, servicio, reclamo), facilitando la asignación al personal adecuado.

- "Hola, ¿tienen stock?" $\rightarrow$ *intención = venta*
- "Llegó dañado" $\rightarrow$ *intención = reclamo*

2. Herramientas utilizadas

2.1. NumPy

Es una biblioteca de Python para el manejo de *arrays* y operaciones numéricas eficientes. Proporciona estructuras de datos y operaciones vectorizadas como sumas, multiplicaciones y álgebra lineal.
Nos permitirá convertir el texto preprocesado en representaciones numéricas estables y facilitar normalizaciones y cálculos de similitud para mejorar la calidad de las características al clasificar las intenciones (venta, servicio, reclamo o información).

2.2. NLTK

Es una biblioteca de Python que permite realizar tokenización (separar palabras y signos de puntuación) y remover *stopwords*, que son palabras comunes que se filtran para quedarnos con tokens relevantes.

- Texto original: "Hola mi nombre es Juan y tengo un reclamo."
- Tokens: [hola, mi, nombre, es, juan, y, tengo, un, reclamo]
- Tokens limpios: [hola, nombre, juan, tengo, reclamo]

2.3. WordNet

Es una base léxica que permite encontrar sinónimos, hipónimos y otras relaciones semánticas entre palabras. Se utilizará para enriquecer las características con sinónimos o clusters semánticos. Por ejemplo, "comprar" y "adquirir" pueden mapear a la misma intención.

- "quiero comprar" $\Rightarrow$ [comprar, adquirir, compré]

2.4. Scikit-learn

Es una biblioteca de Python que proporciona herramientas para tareas de aprendizaje automático. Incluye el vectorizador TF-IDF (*Term Frequency–Inverse Document Frequency*), que transforma documentos de texto en una matriz numérica donde cada columna representa una palabra del vocabulario.
Esto convierte las cadenas resultantes del preprocesamiento en una representación que los modelos pueden utilizar.
El modelo aprende patrones entre estas matrices y sus etiquetas (categorías), para luego predecir nuevas intenciones. Además, permite entrenar un clasificador supervisado que predice automáticamente la intención del mensaje.

3. Proceso de transformación del mensaje

Después del preprocesamiento (minúsculas, tokenización, eliminación de stopwords y puntuación), cada mensaje se convierte en una lista de tokens relevantes.

- Mensaje entrante: “Hola, mi pedido llegó dañado”
- Tokens relevantes: [pedido, llegado, dañado]

El vectorizador TF-IDF transforma cada mensaje en un vector numérico donde:

- **TF**: frecuencia de una palabra dentro del mensaje.
- **IDF**: penaliza palabras frecuentes en todo el corpus (como “hola”) porque aportan poca información discriminativa.

Esto otorga más peso a palabras como “pedido” y “dañado”.
Como resultado, cada mensaje pasa de ser texto a un vector numérico que refleja la importancia relativa de cada término para ese mensaje dentro del conjunto de datos.
Luego, el modelo predice la intención; por ejemplo:

Predicción: RECLAMO (alta probabilidad)

Como acción posterior, el sistema puede enrutar la conversación al equipo de reclamos y marcarla como prioridad alta.

4. Normalización: Stemming vs. Lematización

La normalización es un paso crítico para reducir la dimensionalidad del vocabulario unificando variantes de una misma palabra. Se evaluaron dos enfoques principales:

- **Lematización:** Consiste en reducir una palabra a su raíz léxica o lema (forma de diccionario) mediante un análisis morfológico completo. Aunque es lingüísticamente precisa, es computacionalmente costosa y requiere bases de datos léxicas extensas.
- **Stemming (Derivación):** Es un proceso heurístico que recorta los afijos (sufijos y prefijos) de las palabras para obtener su raíz o *stem*. Aunque puede generar raíces que no son palabras válidas, es un proceso rápido y eficiente.

4.0.1. Justificación e Implementación del Stemming

Para este proyecto decidimos utilizar **Stemming**. La decisión se basó en la eficiencia computacional y la simplicidad de implementación. Dado que el objetivo es clasificar intenciones en venta de autos y repuestos, la precisión semántica estricta de la lematización no era necesaria; bastaba con agrupar variantes morfológicas (ej. “comprar”, “comprado”) bajo un mismo token para que el modelo detectara patrones.

Implementación: Se desarrolló un algoritmo de stemming basado en reglas personalizadas (*rule-based approach*) en Python, sin depender de librerías externas para este paso específico. La función, denominada `obtener_raiz_casera`, aplica una serie de recortes de sufijos comunes en el idioma español.
El algoritmo verifica la terminación de cada token y elimina los siguientes sufijos para obtener la raíz:

- Verbos en infinitivo: *-ar, -er, -ir*.
- Plurales y terminaciones comunes: *-as, -es, -os, -s*.

```python
def obtener_raiz_casera(palabra):
    if palabra.endswith("ar"):
        return palabra[:-2]

    if palabra.endswith("er"):
        return palabra[:-2]

    if palabra.endswith("ir"):
        return palabra[:-2]

    if palabra.endswith("as"):
        return palabra[:-2]

    if palabra.endswith("es"):
        return palabra[:-2]

    if palabra.endswith("os"):
        return palabra[:-2]

    if palabra.endswith("s"):
        return palabra[:-1]


    return palabra
```

Listing 1: Ejemplo de función para tokenizar

Por ejemplo, al procesar la entrada “niñas”, el algoritmo identifica el sufijo *-as* y devuelve la raíz “niñ”, reduciendo la variabilidad del vocabulario de entrada.

5. Representación Vectorial: TF-IDF vs. Count Vectorizer

Para que el modelo de Machine Learning pueda procesar el texto, es necesario convertir los tokens en vectores numéricos. Se compararon dos técnicas:

- **Count Vectorizer:** Representa el texto basándose en la frecuencia absoluta de cada palabra. Su principal desventaja es que tiende a dar demasiado peso a palabras muy frecuentes que pueden no aportar significado distintivo (ruido).
- **TF-IDF (Term Frequency - Inverse Document Frequency):** Pondera las palabras equilibrando su frecuencia en el documento actual con su rareza en todo el corpus. Penaliza las palabras genéricas y resalta las específicas.

5.0.1. Justificación e Implementación de TF-IDF

Decidimos implementar **TF-IDF** porque es superior discriminando la relevancia de los términos en tareas de clasificación. En nuestro corpus, palabras específicas como “aceite”, “sedan” o “precio” son determinantes para distinguir entre las categorías *VENTA* y *SERVICIO*. El Count Vectorizer daría el mismo peso a una palabra común que a estas palabras clave, mientras que TF-IDF asigna un valor mayor a los términos que realmente definen la intención.

Implementación: Se implementó el cálculo de TF-IDF desde cero para comprender la lógica matemática subyacente antes de utilizar el clasificador. El proceso consta de dos componentes:
1. **TF (Term Frequency):** Se calculó como el conteo bruto de apariciones de un término t en un documento d:

```
def tf(mensaje, vocabulario):
    palabras_mensaje = mensaje.split()
    vector_tf = []

    for palabra in vocabulario:
        conteo = palabras_mensaje.count(palabra)
        vector_tf.append(conteo)

    return vector_tf
```

Listing 2: Ejemplo de función para tokenizar

2. **IDF (Inverse Document Frequency):** Se implementó utilizando suavizado (*smoothing*) para evitar divisiones por cero, siguiendo la fórmula estándar:

$$IDF(t) = \log\left(\frac{N+1}{df(t)+1}\right) + 1 \tag{1}$$

Donde N es el número total de documentos en el corpus de entrenamiento y $df(t)$ es la cantidad de documentos que contienen el término t.

```
   def idf(lista_solo_mesajes,vocabulario):
    idf_diccionario  = {}


    for palabra in vocabulario:
        df = 0
        for mensaje in lista_solo_mesajes:
            palbras_del_nesaje = mensaje.split()

            if palabra in palbras_del_nesaje:
                df+=1

        idf_diccionario[palabra] = math.log((N + 1) / (df + 1)) + 1

    return idf_diccionario
```

Listing 3: Ejemplo de función para tokenizar

Finalmente, la matriz de características se construyó multiplicando ambos valores ($TF \times IDF$) para cada palabra del vocabulario en cada mensaje, generando vectores.

```
def TFIDF(lista_solo_mensajes, vocabulario, idf_entrenado):
    matriz_tfidf = []
```

```
3
4
5       for mensaje in lista_solo_mensajes:
6           vector_tf = tf(mensaje, vocabulario)
7
8           vector_tfidf = []
9           for i, valor_tf in enumerate(vector_tf):
10              palabra_actual = vocabulario[i]
11              valor_idf = idf_entrenado.get(palabra_actual, 0)
12
13              vector_tfidf.append(valor_tf * valor_idf)
14
15          matriz_tfidf.append(vector_tfidf)
16      return matriz_tfidf
```

Listing 4: Ejemplo de función para tokenizar

La forma de usar una librería para simplificar, se puede usar la siguiente librería:

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2
3
4
5 vectorizador  = TfidfVectorizer()
6
7 x = vectorizador.fit_transform(lista_solo_mensajes)
```

Listing 5: Ejemplo de función para tokenizar

6. Post procesamiento y Clasificación

Esta sección detalla las técnicas de post-procesamiento y clasificación utilizadas para el asistente virtual. Se exploran dos vertientes principales: un enfoque geométrico a través de espacios vectoriales y un enfoque probabilístico mediante modelos estocásticos.

6.1. Métodos Geométricos en Espacios Vectoriales

Para aplicar métricas de distancia, primero se transforman los mensajes de texto a vectores numéricos utilizando técnicas como TF-IDF (Term Frequency - Inverse Document Frequency). Una vez en el espacio vectorial R^n, se utilizaron las siguientes métricas.

6.1.1. Distancia Euclidiana

Es la métrica más intuitiva, representando la longitud del segmento de línea recta que conecta dos puntos (vectores) en el espacio. Dado un vector de consulta q y un vector de entrenamiento d:

$$d_{eucl}(q, d) = \sqrt{\sum_{i=1}^{n}(q_i - d_i)^2} \tag{2}$$

Se clasifica el mensaje seleccionando la categoría del vector d que minimice esta distancia. Su principal desventaja en texto es la sensibilidad a la magnitud; documentos largos pueden parecer distantes de documentos cortos aunque compartan el mismo tema.

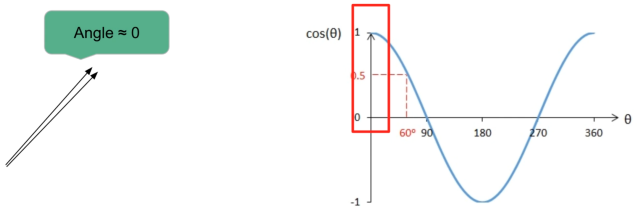

Figura 2: Coseno Cercano

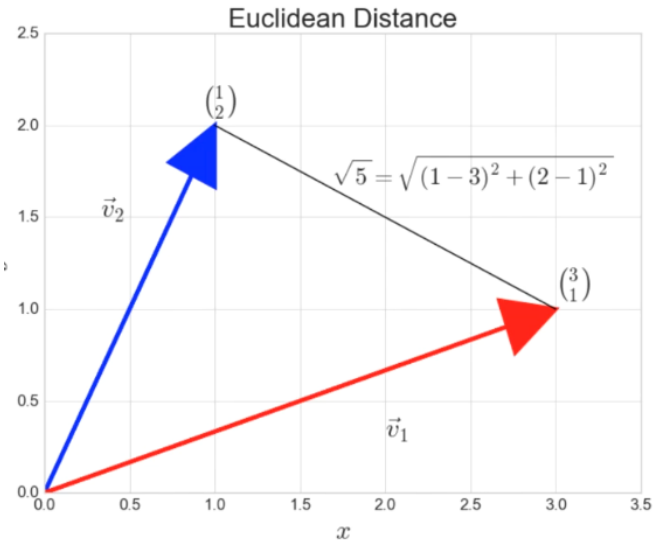

Figura 1: Distancia Euclidiana

6.1.2. Similitud del Coseno

Esta métrica evalúa la orientación de los vectores en lugar de su magnitud, midiendo el coseno del ángulo θ entre ellos. Es robusta ante variaciones en la longitud del texto.

$$\text{Similitud}(q, d) = \cos(\theta) = \frac{q \cdot d}{\|q\|\|d\|} = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{i=1}^{n} q_i^2}\sqrt{\sum_{i=1}^{n} d_i^2}} \tag{3}$$

El valor oscila entre -1 y 1 (o 0 y 1 en espacios de frecuencia positiva), donde 1 indica identidad semántica en términos vectoriales.

6.1.3. Distancia del Coseno

Para mantener la coherencia con algoritmos de minimización de error, se convierte la similitud en una distancia:

$$d_{cos}(q, d) = 1 - \text{Similitud}(q, d) \tag{4}$$

Aquí, un valor cercano a 0 implica máxima similitud.

6.2. Modelos de Markov

Los Modelos de Markov son herramientas estocásticas utilizadas para modelar sistemas que cambian pseudo-aleatoriamente a través de una secuencia de estados. En el contexto del Procesamiento de Lenguaje Natural (NLP), estos modelos tratan el lenguaje no como un conjunto desordenado de palabras (”bag of words”), sino como una secuencia temporal donde el orden es fundamental para la coherencia semántica y sintáctica.
Un modelo de Markov se puede visualizar como un grafo dirigido donde los nodos representan los estados (palabras o tokens) y las aristas representan la probabilidad de transición entre ellos.

6.2.1. La Propiedad de Markov

La base fundamental de estos modelos es la **Propiedad de Markov** (o suposición de memoria corta). Esta propiedad establece que el futuro es independiente del pasado dado el presente.
Matemáticamente, para una secuencia de palabras $w_1, w_2, \ldots, w_t$, la probabilidad condicional de la palabra actual depende únicamente de la palabra inmediatamente anterior (en un modelo de primer orden), y no de toda la historia del mensaje:

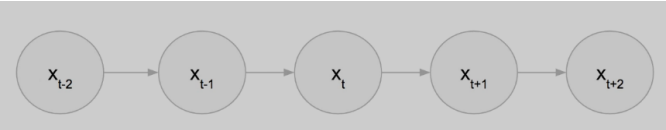

Figura 3: Visualización de la Propiedad de Markov: El estado futuro depende solo del estado actual.

$$P(w_t|w_{t-1}, w_{t-2}, \ldots, w_1) \approx P(w_t|w_{t-1}) \tag{5}$$

Esta simplificación reduce drásticamente la complejidad computacional. Sin esta propiedad, para un vocabulario V y una secuencia de longitud N, tendríamos que calcular V^N parámetros, lo cual es computacionalmente inviable. Gracias a la propiedad de Markov, el problema se reduce a calcular V^2 parámetros.

6.2.2. Componentes del Modelo: π y A

Un Modelo de Markov se define formalmente mediante dos componentes principales que deben ser aprendidos durante la fase de entrenamiento:

1. **Vector de Probabilidades Iniciales (π):** Representa la probabilidad de que una oración comience con una palabra específica w_i.

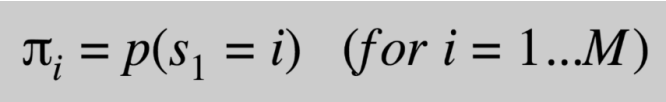

Figura 4: Definición del vector inicial pi

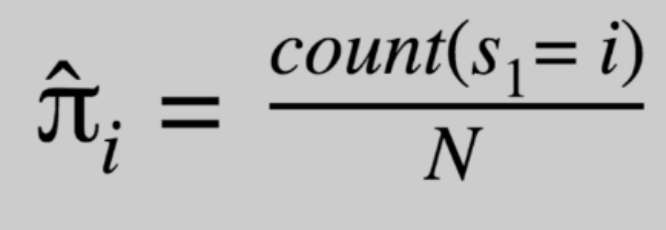

Figura 5: Demostración de como obtener pi

2. **Matriz de Transición (A):** Una matriz donde cada elemento a_{ij} representa la probabilidad de moverse del estado i al estado j (de la palabra w_i a w_j).

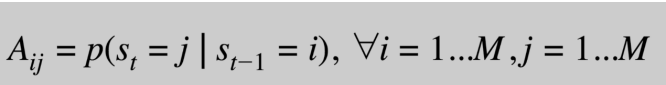

Figura 6: Definicion de la matriz A

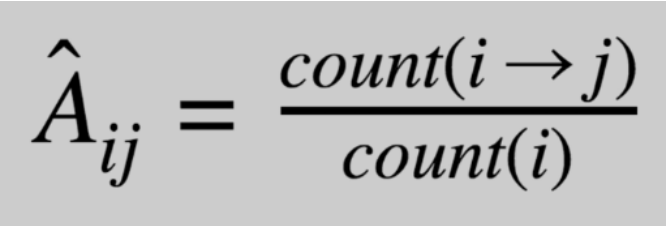

Figura 7: Demostración de como obtener A

Obtención de los parámetros (Entrenamiento) Para obtener estos componentes, utilizamos el método de Estimación por Máxima Verosimilitud (MLE) contando las frecuencias en el corpus de entrenamiento:
Para el vector inicial π:

$$\pi_i = \frac{C(start \to w_i)}{\sum_j C(start \to w_j)} \tag{6}$$

Para la matriz de transición A, la probabilidad de transición de la palabra w_i a la palabra w_j es:

$$a_{ij} = P(w_j|w_i) = \frac{C(w_i, w_j)}{C(w_i)} \tag{7}$$

Donde $C(w_i, w_j)$ es el conteo de veces que el bigrama (w_i, w_j) aparece en el corpus, y $C(w_i)$ es la frecuencia total de la palabra w_i.

6.2.3. Clasificación de Texto con Modelos de Markov Bayesianos

Para utilizar estos modelos en la clasificación de intenciones o categorías, empleamos un enfoque generativo supervisado. A diferencia de un modelo oculto de Markov (HMM) tradicional no supervisado, aquí conocemos las etiquetas de las oraciones.
El proceso detallado para clasificar un nuevo mensaje $M = \{w_1, w_2, \ldots, w_n\}$ es el siguiente:

1. Entrenamiento por Categoría En lugar de crear un único modelo global, entrenamos una cadena de Markov independiente $\lambda_k = (A^{(k)}, \pi^{(k)})$ para cada categoría posible C_k (ej. C_1 = "Ventas", C_2 = "Soporte").

- El modelo λ_{ventas} aprenderá que "precio"suele seguir a "qué".
- El modelo $\lambda_{soporte}$ aprenderá que "funciona"suele seguir a "no".

2. Cálculo de la Verosimilitud (Likelihood) Para clasificar el mensaje M, calculamos la probabilidad de que este mensaje haya sido generado por cada uno de los modelos específicos. Usando la regla de la cadena con la propiedad de Markov:

$$P(M|C_k) = \pi^{(k)}_{w_1} \times \prod_{t=2}^{n} a^{(k)}_{w_{t-1}w_t} \tag{8}$$

Nota técnica: Es fundamental aplicar **suavizado (Smoothing)** (como Laplace Smoothing) durante el cálculo de A y π. Si el mensaje contiene una transición $w_i \to w_j$ que nunca se vio en la categoría C_k, la probabilidad sería 0, anulando toda la ecuación. El suavizado asigna una probabilidad mínima a transiciones no vistas.

3. Regla de Decisión Bayesiana Finalmente, aplicamos el Teorema de Bayes para encontrar la clase más probable. Multiplicamos la verosimilitud obtenida en el paso anterior por la probabilidad a priori de la clase $P(C_k)$ (frecuencia general de la categoría):

$$\hat{C} = \underset{k}{\operatorname{argmax}} \left(P(C_k) \cdot P(M|C_k) \right) \tag{9}$$

En la práctica, para evitar el desbordamiento numérico por multiplicar probabilidades muy pequeñas (underflow), se maximiza la suma de los logaritmos:

$$\hat{C} = \underset{k}{\operatorname{argmax}} \left(\log P(C_k) + \log \pi^{(k)}_{w_1} + \sum_{t=2}^{n} \log a^{(k)}_{w_{t-1}w_t} \right) \tag{10}$$

La categoría que maximice este valor será la etiqueta asignada al mensaje.

6.3. Naive Bayes Multinomial

Finalmente, se implementó el clasificador Naive Bayes. Este modelo es probabilístico pero, a diferencia de Markov, ignora el orden de las palabras (asunción de independencia total).

6.3.1. Fundamento

Se basa en el Teorema de Bayes aplicado a características independientes:

$$P(C_k|x) \propto P(C_k) \prod_{i=1}^{n} P(x_i|C_k) \tag{11}$$

Donde x_i son las palabras del mensaje.

6.3.2. Implementación Robusta

Para la implementación práctica se utilizaron dos técnicas críticas:

1. **Suavizado de Laplace:** Se añade 1 a todos los conteos de frecuencia para evitar que palabras desconocidas ($P(w|C) = 0$) anulen la probabilidad total de la oración.

2. **Log-Probabilidades:** Se trabaja en el espacio logarítmico ($\log(a \cdot b) = \log a + \log b$) para transformar multiplicaciones de números muy pequeños en sumas, evitando el desbordamiento inferior aritmético (numerical underflow).

6.4. Conclusión

S

7. Elección del Modelo Final y Conclusión

Tras evaluar experimentalmente los enfoques geométricos (Distancia Euclidiana, Similitud del Coseno) y los enfoques probabilísticos (Cadenas de Markov, Naive Bayes), se ha determinado que el algoritmo **Naive Bayes Multinomial** será el método seleccionado para la implementación final del asistente virtual.
Esta decisión se fundamenta en tres pilares técnicos:

1. **Eficiencia Computacional y Escalabilidad:** A diferencia de los métodos de distancia (Lazy Learning), que requieren comparar cada nueva consulta contra la totalidad de la base de datos histórica (complejidad lineal $O(N)$), Naive Bayes utiliza un modelo pre-entrenado de probabilidades. Esto permite una clasificación en tiempo constante $O(1)$ independiente del volumen de datos almacenados, garantizando respuestas en tiempo real.

2. **Robustez ante la Variabilidad del Lenguaje:** Mientras que los modelos de Markov son estrictos respecto al orden secuencial, Naive Bayes (bajo el enfoque *Bag of Words*) demuestra mayor flexibilidad. Permite identificar correctamente la intención del usuario basándose en la presencia de palabras clave fuertes (ej. ”precio”, ”garantía”), sin verse afectado negativamente si el usuario altera el orden sintáctico de la oración.

3. **Manejo de Esparcidad (Sparsity):** Gracias a la implementación del Suavizado de Laplace, el modelo maneja eficazmente el problema de las palabras desconocidas, evitando que términos fuera del vocabulario de entrenamiento colapsen el sistema, una ventaja crítica sobre los métodos vectoriales rígidos.

En conclusión, Naive Bayes ofrece el equilibrio óptimo entre precisión en la clasificación de intenciones cortas y bajo costo computacional, cumpliendo con los requisitos de latencia y robustez necesarios para un entorno de producción.