

## **Sistemas conversacionales aplicados a la gobernanza Asistencia automatizada al público**

### **Conversational systems applied to governance Automated assistance to the public**

Brandan F., Pandolfi, D. Villagra A.  
fbrandan@uaco.unpa.edu.ar, dpandolfi@uaco.unpa.edu.ar, avillagra@uaco.unpa.edu.ar

Laboratorio de Tecnologías Emergentes (LabTEm)  
Instituto de Tecnología Aplicada (ITA) - Unidad Académica Caleta Olivia  
Universidad Nacional de la Patagonia Austral - Santa Cruz - Argentina

*Recibido: 21/06/2022. Aceptado: 11/12/2022*

### **RESUMEN**

El término de “ciudades inteligentes” es uno de los conceptos más innovadores de la actualidad, un objetivo que muchos países aspiran conseguir e implementar. Un aspecto de estas ciudades inteligentes es la intercomunicación entre varios actores del sector público y privado; la herramienta ideal para realizar esta unión es un chatbot, capaz de intermediar ambos usuarios. Un chatbot es, simplemente, una aplicación de mensajería que es capaz de mantener conversaciones con sus usuarios, bien ofreciéndoles respuestas preestablecidas o generadas inteligentemente entendiendo la intención y los datos del usuario. Este tipo de sistema conversacional utiliza distintas tecnologías de inteligencia artificial y aprendizaje de máquina, y es capaz de captar y analizar volúmenes enormes de datos relevantes de clientes, entender el comportamiento y la intención de la conversación y ofrecer respuestas a las preguntas ajustándolas a diferentes contextos.

En el presente trabajo se ha utilizado una tecnología de código abierto (Rasa), y se ha realizado el estudio de conceptos de procesamiento de lenguaje natural y análisis de modelos de redes neuronales implementados en Rasa, para poder desarrollar el asistente virtual (chatbot). Este chatbot está orientado a la atención al público que es capaz de responder consultas y entregar formularios a los interesados en los cursos y trabajos disponibles de la Oficina de Empleo local en Caleta Olivia utilizando lenguaje natural ameno y sencillo de seguir, también tomando en cuenta posibles reconocimientos erróneos de intenciones y pudiendo aprender de ellos.

**Palabras clave:** Inteligencia Artificial; Procesamiento del Lenguaje Natural; Rasa; Chatbot.

### **ABSTRACT**

The term "smart cities" is one of today's most innovative concepts, a goal that many countries aspire to achieve and implement. One aspect of these smart cities is the intercommunication between various public and private sector actors; the ideal tool to realize this union is a chatbot, capable of intermediating both users.

A chatbot is, quite simply, a messaging application that can hold conversations with its users, either by providing them with pre-established responses or intelligently generated by understanding the user's intent and data. This type of conversational system uses different



artificial intelligence and machine learning technologies and can capture and analyze huge volumes of relevant customer data, understand the behavior and intent of the conversation, and provide answers to questions by adjusting them to different contexts.

In this work, an open-source technology (Rasa) has been used, and the study of natural language processing concepts and neural network model analysis implemented in Rasa has been carried out to develop the virtual assistant (chatbot). This chatbot is oriented to customer service and can answer queries and deliver forms to those interested in the courses and jobs available at the local employment office in Caleta Olivia using natural language that is pleasant and easy to follow, also taking into account possible misrecognition of intentions and being able to learn from them.

**Keywords:** Artificial Intelligence; Natural Language Processing; Rasa; Chatbot.

## INTRODUCCIÓN

El objetivo principal de toda herramienta es facilitar el desempeño de una tarea; se puede ver a la tecnología como un proceso o capacidad de transformar o combinar algo existente para construir algo nuevo. Por lo tanto, se puede decir que la tecnología permite la creación de nuevas herramientas para resolver cuestiones del día a día.

El avance incesante de las nuevas tecnologías llevó al mundo a una nueva revolución del modo de vivir, amplificado por el contexto de pandemia durante los años 2020 y 2021. Este proceso de reestructuración ocurre en todos los campos, pero resulta en especial aparente en el entorno laboral y social.

Según estudios del INDEC del año 2020 en la Argentina, 86 de cada 100 personas utilizan Internet y 88 de cada 100 emplean teléfono celular. En términos comparativos, respecto del mismo periodo del año anterior, se registra un incremento en el uso del Internet (5,6% más) y del teléfono celular (3,6% más). Estamos hablando de un país que cada vez utiliza más las nuevas TIC (Tecnologías de la Información y Comunicación), un entorno donde la eficiencia de comunicación debe ser la mayor posible, la relación entre los recursos empleados y los resultados debe ser la mejor. La disponibilidad y calidad de las comunicaciones se transforma en un punto central a futuro. Con esto empezamos a ver una imagen clara sobre a dónde se dirige el foco de atención del desarrollo del país, se intenta llegar al concepto de ciudad inteligente.

Podemos definir una ciudad inteligente como una ciudad con un gran sistema de elementos arquitectónicos de alta vanguardia íntimamente conectados, que utilizan un gran apoyo sobre el uso y actualización de las nuevas TIC y espera obtener como resultado una mejora de la calidad de vida de sus habitantes. Esta mejora consiste en un fácil acceso a la interacción para el ciudadano con la arquitectura tecnológica de la ciudad; a los elementos institucionales, culturales y tecnológicos que brinda la ciudad.

Según el estudio francés “Claves de éxito de la gestión inteligente de territorios con tecnología digital” (Courtaudon - Markess International, 2012) las iniciativas de gestión inteligente tienen tres puntos focales:

- **Relación de usuarios interactiva y móvil:** Todo el territorio de la ciudad inteligente es un espacio de interacción para el ciudadano consumidor de información (respecto de su vida diaria como servicios públicos, consumos y eventos) y el productor de información a través de redes sociales y servicios móviles (intercambio de opiniones, vinculación de servicios individuales, enriquecimiento de bases de datos de conocimiento compartidas, entre otras).

- **Supervisión optimizada de la ciudad:** La base de este espacio digital es la capacidad de poder comunicar todos los objetos y actores que la componen a la vez, esto gracias a una gran red de telecomunicación y múltiples nodos y sensores que captan información constante. Todo esto para la recolección e intercambio de datos para una eficaz gestión de la ciudad, que analiza de forma precisa todas las situaciones actuales y permite adaptar los recursos a las necesidades, resultando también en un mejor control de los presupuestos.
- **Desarrollo de nuevas formas de cooperación:** El enfoque permite la capacidad de desarrollar proyectos transversales que van más allá de la lógica, del modo de operación de las colectividades y de las competencias que manejan. Esta práctica da lugar a una dinámica de consulta y asociación entre actores públicos (como autoridades intergubernamentales, municipios, agencias de desarrollo) y privados (como servicios de telecomunicaciones, energía, transporte, actores de infraestructuras), donde deberían surgir nuevas formas de cooperación y también nuevas fuentes de eficiencia como la mutualización.

El objetivo de este trabajo es operar sobre estos tres puntos focales según el estudio de Markess International en la ciudad de Caleta Olivia, brindar un espacio de interacción abierto que comunique dos actores de la sociedad, en específico al ciudadano común y la Oficina de Empleo de la localidad. Para cumplir estos puntos se propuso construir un chatbot disponible al público desde la página oficial de la Oficina de Empleo, de forma simple, una aplicación de mensajería capaz de mantener conversaciones con sus usuarios, ofreciendo respuestas preestablecidas o generadas inteligentemente entendiendo la intención y los datos del usuario que lo ayude a realizar operaciones propias de la oficina.

El principal beneficio que brinda este chatbot es la automatización de tareas de recepción como la toma de datos, inscripciones a cursos de capacitación laboral y potencialmente el contacto para la asignación laboral.

El artículo está organizado de la siguiente manera: la Sección 1 presenta una revisión de los conceptos básicos necesarios para empezar el desarrollo, la Sección 2 describe a detalle el problema a resolver por la aplicación, la Sección 3 describe la etapa de investigación para la construcción de la aplicación, luego la Sección 4 muestra el desarrollo concreto del data set para la aplicación, obteniendo una versión prototipo funcional de la aplicación, y se continúa a la Sección 5 donde se realizarán análisis de sus resultados. Finalmente, la Sección 6 presenta conclusiones del proyecto presentado.

## 1. REVISIÓN DE CONCEPTOS

Es de mayor importancia definir concretamente qué es un chatbot o bot conversacional, porque en la actualidad la mayoría de las personas han interactuado con un bot de estas características sabiéndolo o no, y está tan integrado en el día a día que vale la pena ver a detalle a qué nos referimos.

Un chatbot es un programa de computadora capaz de simular y procesar una conversación humana de forma escrita o hablada, permitiendo a sus usuarios interactuar con servicios digitales como si estuvieran tratando con otra persona real. Estos pueden ser tan sencillos como un pequeño programa que responde preguntas con una línea de texto, como un gran y sofisticado asistente digital que aprende y relaciona conceptos a partir de la información que recibe y procesa logrando altos niveles de personalización e “inteligencia”.

Las tres funciones principales de las que está compuesto un chatbot son las siguientes:

- **Agente de dialogo:** Capaz de recibir los mensajes del usuario. El bot recibe una entrada textual u oral, que es analizada para generar una respuesta adecuada.

- **Agente racional:** El bot debe tener acceso a una base de datos con las respuestas adecuadas a las preguntas del usuario. Además, debe ser capaz de almacenar información propia del usuario y del contexto del mensaje, como nombre del usuario, fechas, género, y otros datos similares para dar una respuesta adecuada.
- **Agente personificado:** El lenguaje del bot debe ser lo más natural y real posible, por lo que se le puede dar una personalidad con el objetivo de crear una confianza con el usuario y dar una sensación de charla con otra persona. Comúnmente se les asigna nombres a estos asistentes, por ejemplo, Eliza, Alice, Padi, Alexa, entre otros.

Las posibilidades de un chatbot residen en las decisiones del desarrollador y sus necesidades. Podrían resolver múltiples tipos de preguntas o problemas de un cliente, automatizar procesos de forma ilimitada, y al mismo tiempo reducir la necesidad de interacción humana. Se puede obtener una mejora de la satisfacción de los usuarios cuando se les ofrece la posibilidad de utilizar un chat donde pueden ser fácilmente guiados a donde deben dirigirse y resolver sus dudas sin la necesidad de ponerse en contacto con un empleado y esperar la respuesta.

Para poder realizar esto se debe profundizar en el concepto de “Procesamiento del Lenguaje Natural” (NLP de *Natural Language Processing*).

Sus comienzos se dan en 1950 con el artículo publicado por Alan Turing “*Computing Machinery and Intelligence*” (Maquinaria de computación e inteligencia) donde discute la temática de la inteligencia artificial y se introduce el concepto de la Prueba de Turing.

Intentando responder la pregunta “¿Pueden las máquinas pensar?” llega a la conclusión que no es posible a menos que las palabras “máquinas” y “pensar” queden bien definidas. Turing plantea un juego para la máquina a probar: Un sujeto externo debe poder discernir correctamente si está hablando con la máquina o si está hablando con otra persona; si este juez no puede darse cuenta de este hecho de forma consistente entonces la computadora pasa la prueba. El objetivo real de esta prueba no es llegar a la conclusión si una máquina puede pensar o no, sino responder “¿Puede una máquina hacer lo que una entidad pensante puede?” o averiguar si una computadora puede generar una capacidad cognitiva similar a la humana. Toda maquinaria digital es “universal” según Turing, con suficiente memoria y tiempo se puede simular el comportamiento de cualquier otra máquina digital. Por lo que, si cualquier máquina digital es capaz de pasar la prueba entonces toda máquina igual y suficientemente poderosa puede hacerlo.

Luego, entre 1964 y 1966, en el *MIT Artificial Intelligence Laboratory* se desarrolla un sistema simulación de psicoanalista llamado ELIZA (Weizenbaum, 1966) definiendo los principios del procesamiento de lenguaje natural. A ELIZA se la denominó un “*chatter bot*” o “Bot de charla” y era capaz de mantener una ilusión de conversación a través de reconocimiento de patrones y sustitución de palabras. Este bot conversacional era capaz de identificar palabras claves de las entradas del usuario, y respondía con mensajes que guardaba en una base de datos. Al encontrarse un mensaje que no comprende sabe reformularlo a forma de pregunta.

De la propuesta de ELIZA surge PARRY, un chatbot desarrollado por un psiquiatra en la Universidad de Stanford (Colby, 1975) que intenta simular una persona que sufre de paranoia esquizofrénica, implementando un comportamiento basado en creencias de conceptos y conceptualizaciones generales. Un grupo de psiquiatras pusieron a prueba la capacidad de PARRY de emular una conversación con otra persona, quisieron someterlo a la prueba de Turing. Se les mostró un grupo de transcripciones de conversaciones con pacientes reales y uno de conversaciones con PARRY. Se obtuvieron resultados correctos sólo un 48% de las veces, no mejor que adivinar al azar. En varias ocasiones interactuaron ELIZA con PARRY, pero no lograron conversaciones productivas (Cerf, 1972).

Hasta 1980 empiezan a surgir muchos sistemas conversacionales similares a ELIZA, compuestos por una serie compleja de reglas escritas a mano. Uno de estos sistemas de conversación avanzados es SHRDLU (Winograd, 1971), presentado por el MIT con un terminal

gráfico, es capaz de administrar un espacio 3D con distintos objetos geométricos. El usuario puede pedirle a SHRDLU que mueva los objetos, los apile o desapile, hacerle preguntas sobre la posición de alguno de los objetos, y es capaz de volver atrás en el contexto de la conversación para conocer el historial de movimientos realizados.

A finales de los '80 ocurre una revolución en donde surgen los algoritmos de *Machine learning* (Aprendizaje automático), donde primero se modela una base simple de datos y luego utilizando los datos recolectados se empiezan a realizar predicciones. A través de los datos y de la propia experiencia el algoritmo “aprende” y evoluciona, sus predicciones se afinan progresivamente. A partir del desarrollo de la Web 2.0 o Web social (donde el usuario empieza a tomar un rol más activo en la comunicación y es capaz de entregar datos de la misma forma que los recibía en la Web estática) en los años 2000s se habilita la recolección de datos en grandes cantidades, lo que impulsa a que el aprendizaje automático sea mucho más viable. El enfoque se mueve a algoritmos con baja o sin supervisión que toman datos en bruto y saben elegir la información relevante para sus tareas.

Actualmente el enfoque principal de este tipo de proyectos se encuentra en el *Deep learning* (aprendizaje profundo), refiriéndose al uso de múltiples capas en una red. Se crean redes neuronales artificiales (ANN) inspiradas en el procesamiento de información y nodos de comunicación distribuidos de forma similar a los sistemas biológicos. Aunque a diferencia de un cerebro, las redes neuronales tienden a permanecer estáticas y simbólicas, mientras que el cerebro biológico es dinámico y análogo. El uso de esta técnica ayudó a alcanzar grandes logros en el reconocimiento y modelaje de lenguaje.

Una investigación moderna de la percepción de chatbots por personas con trastorno de desarrollo neurológico obtuvo resultados interesantes, en cuanto a cómo acercar información o ayuda a los usuarios utilizando agentes conversacionales que se asemejan al comportamiento de una persona en ciertos aspectos. El agente puesto a prueba llamado “Emoty” es un facilitador emocional y entrenador para personas con déficits cognitivos y emocionales (Catania, Beccalupa y Garzotto, 2020). Al finalizar la serie de pruebas se concluyó que se lo percibió al agente más cercano a una persona cuando se le asignó una voz femenina, utilizaba lenguaje natural e informal, y cuando cometía errores.

Los principales desafíos del NLP enfrentados en este proyecto son la Comprensión del Lenguaje Natural (*Natural Language Understanding*) donde se aplica un preprocesamiento al texto escrito o hablado y que pueda ser interpretado por la máquina fácilmente. Por otro lado, la Generación del Lenguaje Natural (*Natural Language Generation, NLG*) donde la máquina convierte a los datos estructurados en respuestas y los transforma en lenguaje natural para comprensión del usuario.

## 2. DEFINICIÓN DEL PROBLEMA

En una primera instancia se realiza una reunión con el director de la Oficina de Empleo donde se define el alcance que debe tener el proyecto.

Se propone crear un chatbot a integrarse en la página web de la Oficina de Empleo que atienda a los tres siguientes comportamientos, o requerimientos funcionales:

- Recibir como entrada preguntas realizadas en lenguaje natural sobre la disponibilidad de cursos/trabajos, requisitos mínimos, método de inscripciones, y otras preguntas relacionadas.
- Devolver como salida respuestas con información que den solución a las preguntas/peticiones realizadas por el usuario, a través de un diálogo.
- Acercar al usuario los posibles formularios que necesita completar de estar disponibles digitalmente.

Y además deberá cumplir con los siguientes requerimientos no funcionales (características generales y restricciones):

- La aplicación debe estar disponible 24hs los 7 días de la semana.
- La aplicación debe funcionar en cualquier navegador web como Google Chrome, Firefox, IE, Edge, entre otros.
- La aplicación debe funcionar en computadoras usando cualquier sistema operativo como Windows, Linux, IOS, entre otros.
- La aplicación debe ser capaz de procesar hasta 100 operaciones por minuto.
- La aplicación debe responder al usuario en menos de 5 segundos.

### 3. APLICACIÓN PROPUESTA

Para el desarrollo de la aplicación se analizan distintos posibles marcos de trabajo en los que basar el chatbot, se hace especial atención a alternativas de desarrollo gratuitas. Se consideraron las siguientes herramientas, o *frameworks*:

- **BotMan<sup>1</sup> + DialogFlow<sup>2</sup>**: Un marco de trabajo basado en PHP y Laravel, diseñado para la implementación web de chatbots en plataformas como Slack, Telegram, Facebook entre otros. Posee la capacidad de trabajar en conjunto con DialogFlow, la herramienta creada por Google para NLP, en forma de middleware para procesamiento de los mensajes recibidos.
- **Botkit<sup>3</sup>**: Una herramienta para desarrolladores web abierta para la construcción de chatbots, aplicaciones e integraciones a plataformas de mensajería.
- **Rasa<sup>4</sup>**: Con soporte para múltiples lenguajes, comprensión y clasificación de intenciones y captura de contextos del usuario de forma nativa, investigado de forma local y código abierto basado en Python.

Luego de realizar pruebas locales iniciales con cada herramienta se decidió continuar la investigación utilizando Rasa. Las principales razones son las siguientes:

- El costo de implementación: El código abierto resulta beneficioso tanto para el desarrollo, la etapa de pruebas y para la Oficina de Empleo. Rasa está desarrollado bajo licencia Apache Versión 2.0, lo que permite la modificación y distribución del código, además del uso comercial y privado de la herramienta.
- Arquitectura modular: Añadir procedimientos específicos para responder a los requisitos, implementación de modelos propios y extender el funcionamiento del chatbot a futuro resulta muy interesante. Una vez finalizado el desarrollo e implementación se pueden realizar tareas de mantenimiento y extensión con facilidad.
- Posibilidad de aprendizaje: Su estructura transparente permite aprender en profundo el funcionamiento exacto de una implementación de NLP, posibilitando la construcción de sistemas más complejos a futuro.

Para iniciar el desarrollo del chatbot se debe preparar la configuración del modelo, definir los componentes y políticas que se usarán para hacer predicciones basadas en las entradas del usuario. Dentro del archivo *config.yml* se encuentran los apartados “*pipeline*” y “*policias*”.

El pipeline es el que define qué componentes van a analizar las entradas del usuario y clasificar sus partes en intenciones y entidades. Las intenciones son la forma en la que se deben clasificar

<sup>1</sup> Página oficial BotMan - <https://botman.io/>

<sup>2</sup> Página oficial DialogFlow - <https://cloud.google.com/dialogflow>

<sup>3</sup> Página GitHub Botkit - <https://github.com/howdyai/botkit>

<sup>4</sup> Página oficial Rasa: <https://rasa.com/>

Página GitHub Rasa: <https://github.com/RasaHQ/rasa>

los mensajes del usuario, y las entidades son nombres, datos personales del usuario como un correo o número de teléfono para potencialmente almacenar.

Hay dos tipos de pipelines, según su forma de extraer las características de cada mensaje para entrenar la predicción de intenciones y la extracción de entidades, estos son:

- **Incrustaciones pre-entrenadas** (*Pre-trained embeddings*): Se cargan modelos ya pre-entrenados que se utilizan para representar palabras como vectores numéricos. Estas incrustaciones pueden entender aspectos semánticos y sintácticos similares entre palabras representadas por vectores similares. Un ejemplo de una librería de este tipo es spaCy que dispone de una buena incrustación pre-entrenada en español, por lo que una implementación usando esta librería requiere un menor tiempo de entrenamiento en el idioma. Si se tiene una baja cantidad de datos iniciales, como puede ser al comienzo del desarrollo, se pueden obtener muy buenos resultados en poco tiempo. La única desventaja de este tipo de incrustaciones es el nulo control de palabras específicas o siglas del ámbito que se va a utilizar; sólo son entrenados sobre conjuntos de datos públicos como Wikipedia.
- **Incrustaciones supervisadas** (*Supervised embeddings*): En esta metodología se crean las incrustaciones desde cero sobre el conjunto de entrenamiento disponible. Ese pipelining permite la comprensión de mensajes con múltiples intenciones, haciendo muy versátil para contestar mensajes inesperados. A coste de generar el modelo desde cero se necesitan muchos datos de entrenamiento, pero se adapta con facilidad al camino del dominio propio. Puede aprender cualquier idioma, no está limitado al pre-entrenamiento que pueda llegar a tener.

Para el desarrollo de la aplicación se decidió utilizar la incrustación supervisada para tener un mayor control sobre los caminos lógicos que podrá utilizar el bot.

Los componentes a utilizar en el chatbot según *config.yml* se muestran a continuación en la Figura 1.

```
language: es
pipeline:
  - name: WhitespaceTokenizer
  - name: RegexFeaturizer
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
  - name: CountVectorsFeaturizer
  analyzer: "char_wb"
  min_ngram: 1
  max_ngram: 4
  - name: DIETClassifier
  epochs: 100
  - name: EntitySynonymMapper
  - name: ResponseSelector
  epochs: 100
```

Figura 1 - Extracción de *pipelines* del archivo *config.yml*

Las partes que intervienen en el proceso de procesamiento de texto son las siguientes:

- *WhitespaceTokenizer*: Tokenizador. La tokenización consiste en dividir la cadena de caracteres recibida y separarlas en pequeñas partes llamadas “*tokens*”, que se utilizarán para analizar las intenciones. Cualquier carácter que no corresponda a *a-z,A-Z,0-9\_#@&* será sustituido en un espacio en blanco, y luego se separa la oración en palabras rodeadas de espacios blancos.

Se utiliza la función `extract_tokens` para obtener todos los tokens de un texto usando expresiones regulares. Primero utiliza la función `.lower()` para pasar el texto entero a minúscula, y luego usa la función `re.sub()` para llenar los espacios en blanco donde no cumple con la expresión regular. Finalmente, la función `.split()` divide los tokens.

A continuación, se muestra un ejemplo de esta tokenización en la Figura 2.

```
text = 'Esto es un ejemplo utilizando WhitespaceTokenizer.'
tokens = extract_tokens(text)
print(tokens)
>>> ['esto', 'es', 'un', 'ejemplo', 'utilizando', 'WhitespaceTokenizer']
```

Figura 2 - Ejemplo en consola de tokenización

- **RegexFeaturizer:** Expresiones regulares. Este componente crea una lista de expresiones regulares definidas en el conjunto de entrenamiento. Requiere que el texto haya sido previamente tokenizado. Para cada expresión regular y cada token, se indica con un 1 si el token coincide con la expresión regular o con 0 si no coincide. Las características esparcidas (*sparse features*) permiten un entrenamiento incremental, se añadirán patrones a los datos de entrenamiento en un futuro. A continuación, se muestra un ejemplo de la creación de una lista en la Figura 3.

```
patterns = ["[a-zA-Z]{5}", "[0-9]{4}-[0-9]{2}-[0-9]{2}"]
tokens = [adios, 2022-02-03, produce, 1998-08-08]
vec = np.zeros([len(token), len(patterns)])
RegexFeat(patterns, tokens, vec)
>>> [[1.0, 0.0], [0.0, 1.0], [0.0, 0.0], [0.0, 1.0]]
```

Figura 3 - Ejemplo de creación de lista en consola

- **LexicalSyntacticFeaturizer:** Clasificador de intención. Requiere un texto previamente tokenizado. Este componente crea características para la extracción de entidades. Se mueve utilizando un protocolo de ventana deslizante sobre cada token del mensaje del usuario y crea características de acuerdo con la configuración. Por ejemplo, algunas características son los primeros y últimos caracteres del token. Las características creadas por este componente también son dispersas.
- **CountVectorsFeaturizer:** Clasificador de intención. Requiere un texto previamente tokenizado. Se utilizan dos instancias de *CountVectorsFeaturizer* bajo esta configuración. La primera extrae características basándose en las palabras. La segunda basándose en caracteres *ngram*. Crea un modelo de “bolsa de palabras” con la función *CountVectorizer()*. Las características creadas por este componente también son dispersas.
- **DIETClassifier:** Clasificador de intención. DIET significa Transformador de Entidad de Doble Intención (*Dual Intent Entity Transformer*). Su modelo sirve para la extracción de entidades y para la clasificación de intenciones, está basado en transformadores. Para la extracción de entidades se utilizan capas CRF Campos Aleatorios Condicionales (*Conditional Random Fields*).
- **EntitySynonymMapper:** Sinónimos. Este componente se encarga de modificar las entidades extraídas del componente anterior por los sinónimos definidos en *nlu.md*, así los valores de las entidades detectadas siempre se le asignan los mismos valores. Esto simplifica las acciones futuras de NLG, de la generación de las respuestas. A continuación, en la Figura 4, se muestra un ejemplo de uso de sinónimos.

```

synonyms = {"valores mensuales": "mensual", "cifras mensuales": "mensual"}

EntitySynonym(dic1, synonyms)
>>> [{"text": "Necesito los valores mensuales",
      "intent": "grafica_mensual",
      "entities": [{"entity": "tipo_grafica",
                    "start": 14,
                    "end": 30,
                    "extractor": "DIETClassifier",
                    "value": "mensual"}]}]

EntitySynonym(dic2, synonyms)
>>> [{"text": "Entregar cifras mensuales",
      "intent": "grafica_mensual",
      "entities": [{"entity": "tipo_grafica",
                    "start": 10,
                    "end": 25,
                    "extractor": "DIETClassifier",
                    "value": "mensual"}]}]

//Los diccionarios dic1 y dic2 tienen la misma estructura de salida, aunque las
//frases de entrada son distintas.

```

Figura 4 - Ejemplo de uso de sinónimos en consola

Los componentes utilizados en la generación de las respuestas también se encuentran en el archivo *config.yml* en el apartado “*policies*”. La elección de la respuesta más adecuada se efectúa utilizando los componentes indicados en este apartado.

Los componentes utilizados en la construcción de este bot son los descritos a continuación en la Figura 5.

```

policies:
- name: MemoizationPolicy
  max_history: 3
- name: RulePolicy
- name: TEDPolicy
  max_history: 5
  epochs: 100

```

Figura 5 - Extracción de *policies* del archivo *config.yml*

Se prosigue a describir el funcionamiento de cada política:

- *MemoizationPolicy*: Este componente recuerda las historias con las que se entrenó el modelo. Se establece el parámetro *max\_history* que controla cuánto del diálogo toma en cuenta para decidir qué opción tomar. Confirma si la conversación actual coincide con una de las historias de *stories.yml*, si coincide intenta predecir la próxima acción con una confianza de 1.0. Si no existe una coincidencia la política predice “None” con una confianza de 0.0. Como la aplicación a desarrollar no tiene múltiples desvíos en sus historias un valor medio como 3 le da suficiente espacio para tomar buenas decisiones sin retrasar el procesamiento.

- **RulePolicy:** Esta política maneja las partes de la conversación que siguen un comportamiento fijo. Hace las predicciones basándose en reglas del modelo de entrenamiento.
- **TEDPolicy:** La política de Transformación de Dialogo Incrustado (*Transformer Embedding Dialogue*) es una arquitectura de multitarea para la predicción de próximas acciones y de reconocimiento de entidades. Consiste en varios codificadores transformadores que comparten ambas tareas. Una secuencia de etiquetas de entidades se predice a través de un Campo Condicional Aleatorio sobre la salida del transformador del mensaje del usuario. Para la próxima predicción la salida del CRF y las etiquetas son incrustadas en un solo espacio vectorial, que se utiliza para eliminar muestras negativas y minimiza similitudes equivocadas. El parámetro *epochs* indica la cantidad de veces que el algoritmo revisará la información de entrenamiento. Una baja cantidad de *epochs* hará más rápido el entrenamiento del modelo.

Aplicando todas estas configuraciones de pipeline y políticas se encuentran redes neuronales trabajando en multitarea para la clasificación de intenciones, la extracción de entidades y por otra parte la generación de las respuestas. La estructura encargada de la clasificación y extracción basado en transformadores se llama DIET, mientras que la estructura recurrente encargada de la NLG se llama LSTM.

Inmediatamente, se describe el funcionamiento de ambas estructuras de redes neuronales:

- **DIET:**

Su nombre indica Transformador Dual de Intenciones y Entidades (*Dual Intent and Entity Transformer*), es una estructura muy flexible y modificable, compuesta de distintos componentes. En la versión 1.8 de Rasa DIET reemplazó a BERT por su gran eficiencia y modularidad. BERT es una red neuronal recurrente que requiere una gran cantidad de poder de cálculo. DIET obtiene una precisión y rendimiento similar a modelos pre-entrenados de gran escala y se entrena hasta seis veces más rápido. Utiliza un modelo compacto “*plug and play*” o “conectar y funcionar”, además de utilizar un modelo secuencial que toma en cuenta el orden de las palabras ingresadas.

En la Figura 3 se puede observar la entrada del usuario pasando por el modelo pre-entrenado BERT y entregando un resultado procesado a DIET, que separa la intención y las entidades involucradas. En el ejemplo de la Figura 6 se ingresa “Yo quiero jugar ping pong”, y DIET separa la intención “Jugar juego” y la entidad “Juego = ‘ping pong’”.

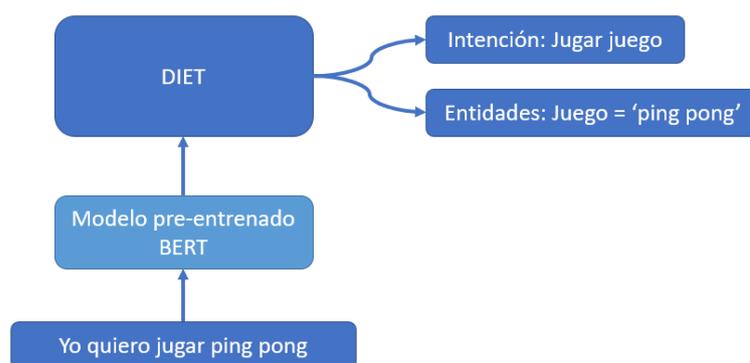


Figura 6 - Ilustración a alto nivel de DIET (Adaptado de Rasa “*Introducing DIET*”)

- **LSTM:**

Las redes neuronales recurrentes (*Recurrent Neural Network, RNN*) permiten las entradas de longitud variable y conexiones entre las neuronas mismas, posibilitando la

intercomunicación y cooperación entre procesos. El resultado de una capa puede ser útil para una capa siguiente.

Hay que tener en cuenta que una conversación no sólo importa el mensaje actual, sino también el anterior. Las redes neuronales utilizan el concepto de recurrencia para generar una salida, o activación. La red utiliza la entrada actual y la activación generada por la iteración anterior, y de esta forma se relacionan ambas partes del diálogo.

Para definir un instante de tiempo  $t$ , un número entero que define la posición del elemento dentro de la secuencia, primero definimos  $x_t$  e  $y_t$  como entrada y predicción de la red recurrente en el instante de tiempo  $t$ . La salida  $a_t$  la definimos como se ilustra en la ecuación (1):

$$a_t = f(W_{aa} \cdot a_{t-1} + W_{ax} \cdot x_t + b_a) \quad (1)$$

Donde  $f$  es la función de activación y  $W_{aa}$ ,  $W_{ax}$  y  $b_a$  son los coeficientes de la red.

En cada instante de tiempo  $t$  la red tiene dos entradas. La entrada actual  $x_t$  y la activación anterior  $a_{t-1}$ , que viene de la memoria de la red. Esto se puede observar en la Figura 7.

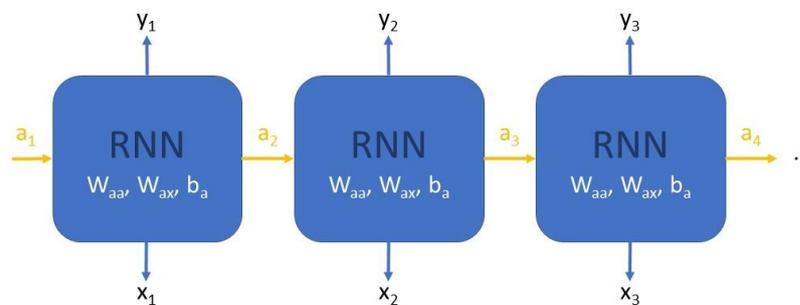


Figura 7 - Notación desplegada para redes neuronales recurrentes  
(Adaptado de Graves et al., 2013)

Se presenta un problema al momento de conectar dos piezas de información a mucha distancia; en teoría es posible que se realice la conexión entre ambos puntos, pero mientras mayor sea la brecha entre datos mayor es la dificultad que tendrá la red de conectar ambos conceptos. No es suficiente tener solo la información de los pasos anteriores, se necesita un mayor contexto. Para resolver este problema surgen las redes LSTM (*Long Short Term Memory*) propuestas por Hochreiter y Schmidhuber (1997). Estas son un tipo especial de RNN, su comportamiento principal es recordar información por largos periodos de tiempo.

Las LSTM también tienen una estructura en cadena como las RNN normales, pero en lugar de utilizar una sola capa de red neuronal, utilizan cuatro que interactúan de una forma especial. Su estructura interna está descrita en la Figura 8.

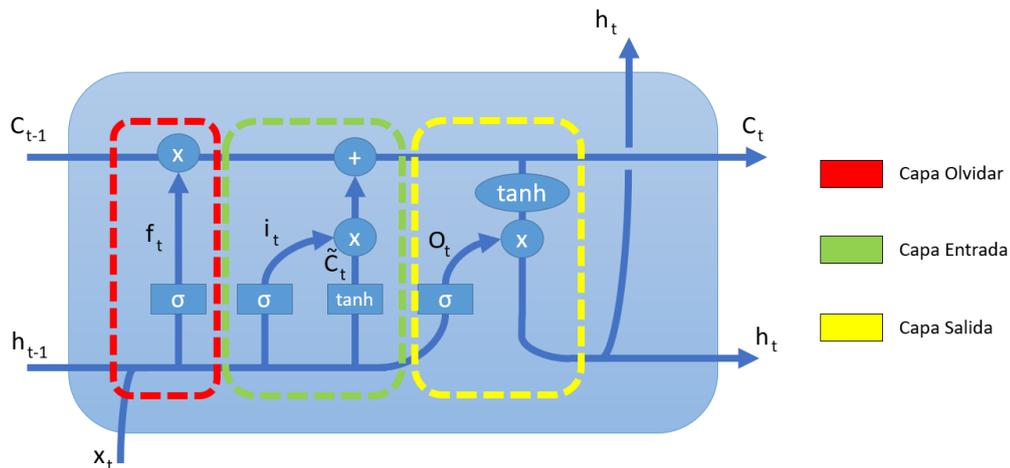


Figura 8 - Estructura interna celda LSTM (Adaptado de Olah, 2015)

En la parte superior de la celda se encuentra la celda de estado que puede añadir y quitar datos sobre la memoria de la red. La primera puerta sigmoideal (en color rojo) es la capa de olvidar, que decide qué dato se va a desechar de la celda  $C_{t-1}$ . Sus valores de entrada son el estado oculto anterior  $h_{t-1}$  y la entrada actual  $x_t$ . De esto se obtiene un valor entre 0 y 1 por cada número en la celda de estado. Una salida 1 indica que la información debe mantenerse, mientras que una salida 0 indica que debe eliminarse. Los coeficientes  $W$  y  $b$  son los pesos y sesgos, o parámetros entrenados de la neurona. La salida de esta capa se representa como se muestra en la ecuación (2):

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (2)$$

La segunda parte consiste en decidir qué nueva información se añadirá a la celda de estado, y a su vez se divide en dos capas distintas. La primera capa sigmoideal (en color azul) de entrada indica los valores nuevos a agregar. La segunda capa tangente hiperbólica crea los nuevos valores de los datos a ser agregados a la celda de estado. Esta salida se define como  $\check{C}_t$ . La información final para incluir es la combinación de las dos salidas ilustradas en las siguientes ecuaciones (3)(4):

$$i_t = \sigma(W_i \cdot (h_{t-1}, x_t) + b_i) \quad (3)$$

$$\check{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (4)$$

Luego se multiplica el antiguo estado de la celda  $C_{t-1}$  por  $f_t$ , y a ese resultado se le suma  $i_t \cdot \check{C}_t$ , para restar lo que se debe quitar y agregar lo que se debe añadir. La celda resultante sigue la fórmula de la ecuación (5):

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \check{C}_t \quad (5)$$

El último paso de la celda LSTM es la capa sigmoïdal de salida (en color amarillo), que decide qué datos se van a devolver. El estado actual pasa por una capa tangente hiperbólica y es multiplicada por la salida de la capa sigmoïdal, y sólo devuelve la información requerida. Esto se ilustra en las ecuaciones siguientes (6)(7):

$$o_t = \sigma(W_o \cdot [h_{t-1} \cdot x_t] + b_o) \quad (6)$$

$$h_t = o_t \cdot \tanh(C_t) \quad (7)$$

Estas redes LSTM se entrenan de la misma forma que las redes *feedforward*, como las redes DIET, por ejemplo. Utilizan un algoritmo de *backpropagation* para la actualización de pesos de valor.

Las funciones básicas que debe cumplir el chatbot son las siguientes:

- Comprender todo tipo de saludos, despedidas y mensajes de gracias; y poder responderlos de forma adecuada.
- Comprender si se le está solicitando información o un formulario de cierto curso específico, o un listado de los cursos disponibles.
- Posibles errores gramaticales de parte del usuario deben ser considerados.

Para el desarrollo de este chatbot se utilizó la versión de Rasa 2.8.23 y el sistema operativo Windows. El lenguaje de programación de Rasa es Python, que durante este proyecto se utilizó la versión 3.8.10, y para la escritura de código se utilizó el programa Sublime Text 3.

Para empezar el desarrollo de Rasa se realiza una instalación limpia, se crean los archivos básicos y algunos datos de entrenamiento.

Dentro de la carpeta *models* se encuentran todos los modelos entrenados. El cuerpo de entrenamiento se compone de los archivos *nlu.md*, *stories.md* y *domain.yml*.

Los archivos *endpoints.yml* y *credentials.yml* se utilizan para conectar el bot a diferentes canales de comunicación.

En el archivo *actions.py* se implementan todas las acciones personalizadas.

A continuación, se describen los archivos básicos que componen Rasa:

- *nlu.md*: Donde están declaradas las oraciones de entrenamiento. Se indican las posibles entidades y se les asigna una intención principal. Luego el bot las analiza como referencia para aprender a identificar partes similares en otras oraciones del usuario. Tiene además dos apartados opcionales, el de sinónimos que asocia distintas entidades a un mismo valor, es utilizado por *EntitySynonymMapper*. Y las expresiones regulares para una extracción de entidades más rápida, es utilizado por *RegexFeaturizer*.
- *domain.yml*: Contiene todo el dominio del bot, sus respuestas, slots, entidades, intenciones y un listado de las acciones personalizadas que se crearon.

Los slots son la memoria del bot, almacenan información sobre entidades de forma clave:valor, estos se utilizan mucho en las acciones personalizadas para que influyan en la estructura del texto. Los tipos de slots son los siguientes: texto, booleano, categórico, flotante, lista y sin característica.

Las acciones personalizadas son funciones que ejecutan código, pueden utilizar los slots para crear una respuesta más dinámica al mensaje del usuario.

El archivo tiene una pequeña sección de configuración donde se puede especificar después de cuánto tiempo de inactividad se cierra la sesión de conversación actual y se abre una nueva, con el parámetro *session\_expiration\_time* en minutos. Y el parámetro *carry\_over\_slots\_to\_new\_session* toma un valor booleano que indica si los slots actuales serán recordados en futuras sesiones o no.

- *stories.md*: Se almacenan simulaciones de conversaciones entre el bot y un posible usuario llamadas historias, estas estructuras se utilizan para entrenar el modelo de

gestión de dialogo. Se realiza la conversión de intención del mensaje del usuario y las respuestas según la acción del bot.

Un ejemplo sencillo de historia almacenada se puede ver en la Figura 9.

<pre> stories: - story: Saludar y subscribir   steps:   - intent: saludo   - action: pronunciar_saludo   - intent: subscribir   - action: formulario_subscripción   - active_loop: formulario_subscripción </pre>
<p>Una conversación que sigue esta historia sería la siguiente:</p> <p>Usuario: Hola, buenos días.  Bot: Hola, bienvenido.  Usuario: Quiero subscribirme a las noticias.  Bot: Aquí está el formulario de subscripción: (Formulario)</p>

Figura 9 - Ejemplo de historia almacenada

- *actions.py*: Como fueron descritas en la descripción de *domain.yml*, las acciones personalizadas son respuestas del bot con un código personificado. Puede incluir sólo texto específico como respuesta, extraer entidades de una respuesta y devolverlas al usuario, extraer datos de una base de datos externa, extraer datos y devolver un resultado. En este archivo está definido el código de cada acción.
- *credentials.yml* y *endpoints.yml*: El primer archivo, credenciales, sirve para brindar soporte a través de páginas como Facebook, Slack, Telegram. Toda la configuración se encuentra comentada y se debe habilitar sólo los módulos que se utilizarán. El archivo *endpoint* lista los extremos finales del canal de comunicación con una API que interactúe con el sistema. Se utilizan para obtener datos o solicitudes nuevas.

Para entrenar el modelo se ejecuta el comando “*rasa train*”, y los resultados se almacenan en la carpeta *./models*.

Si ya existen modelos en la carpeta sólo se entrenarán las partes modificadas. Los archivos de entrenamiento se almacenan con el nombre *<EstampaDeTiempo>.tar.gz*. Utilizando la bandera *-fixed-model-name* se le puede especificar un nombre diferente.

Además, como nueva característica de Rasa 2.2, se puede utilizar la bandera *-finetune* para un desarrollo enfocado en charlas. Se toman conversaciones anteriores con usuarios y los datos de entrenamiento provistos para reducir el tiempo de entrenamiento de un modelo nuevo.

#### 4. DESARROLLO DEL DATA SET PARA LA APLICACIÓN

Adentrando en la construcción específica del data set del área de Oficina de Empleo de gobernanza local, se describirán los datos entregados al chatbot para aprender las normas del negocio y posibles estructuras de conversación. Este modelo sirve como prototipo y prueba de utilización de Rasa sobre el cual luego se pueda expandir el proyecto. Además de estos datos de entrenamiento también se entregará al chatbot una recolección de artículos provenientes de Wikipedia a fin de familiarizarse con el lenguaje español y conocer su sintaxis.

Al bot se le entrega una serie de archivos describiendo cómo debe afrontar una conversación. Se describen las posibles intenciones que puede tener un usuario al expresarse en los datos NLU, las potenciales respuestas que el bot puede entregar, unas historias confeccionadas para entrenarlo en rutas de conversación, los slots de memoria o datos que puede requerir del usuario y los formularios que recolecta, y por último las reglas a seguir, conversaciones que siempre debe seguir sin importar conversaciones anteriores.

La especificación completa del modelo se puede encontrar en Anexo A del documento.

## 5. ANÁLISIS DEL MODELO DE LA APLICACIÓN

Luego de desarrollar todos los archivos necesarios para el entrenamiento, se prosigue a realizar pruebas a este modelo obtenido para la aplicación. Como parte del diseño experimental se evalúa su capacidad de seguir una conversación esperada de un usuario y su nivel de confianza para detectar las intenciones de cada mensaje de entrada, y se lo compara con un modelo sin entrenamiento específico de las reglas del negocio.

A continuación, en la Figura 10, se muestra un ejemplo de conversación con el chatbot junto a las intenciones y acciones identificadas, siguiendo la historia “saludo\_progresar\_inscripcion”.

**Usuario:** Hola, buenas tardes

intent: saludo

action: utter\_saludo

**Chatbot:** ¡Hey! ¿Cómo puedo serte de ayuda hoy?

action: utter\_oferta\_actual

**Chatbot:** Las inscripciones para el plan Proemplear y Progresar continúan en vigencia. Además, contamos con cursos y talleres a disponibilidad.

Actualmente tenemos [Cantidad de cursos] disponibles, y son los siguientes:

[Listado de cursos disponibles]

**Usuario:** ¿Puedo obtener más información sobre progresar?

intent: progresar

action: utter\_progresar

**Chatbot:** Es un programa integral de Becas Educativas que acompaña al alumno en todos los niveles de formación durante su trayectoria académica, a través de un incentivo económico y un importante estímulo personal, que le permitirá poder avanzar en sus estudios hasta completarlos. El objetivo del programa es fortalecer el ingreso, la permanencia y el egreso de los estudiantes, unificando todas las becas educativas del gobierno nacional.

¿Desea anotarse al programa Progresar?

**Usuario:** ¿Cuáles son los requisitos para anotarme?

intent: requerimientos

action: utter\_progresar\_requisitos

**Chatbot:** Sus requisitos son los siguientes:

Ser argentino/a nativo/a; naturalizado/a o extranjero/a, con residencia legal de cinco (5) años en el país y contar con DNI.

Ser alumna/o regular de una institución educativa.

Tener entre diecisiete (17) y veinticuatro (24) años de edad cumplidos.

Estudiantes avanzadas/os en la carrera, hasta treinta (30) años cumplidos.

Estudiantes de enfermería sin límite de edad.

Se extiende hasta 35 años de edad a las personas con hijas/os a cargo menores de 18 años pertenecientes a hogares monoparentales.

Quedan exceptuadas/os del límite de edad las personas trans, pertenecientes a pueblos indígenas, con discapacidad o refugiadas.

Los ingresos de la/el joven y los de su grupo familiar no deberán ser superiores a tres (3) Salarios Mínimos, Vitales y Móviles (SMVM), a excepción de que los y las jóvenes solicitantes sean titulares de una pensión no contributiva por invalidez otorgada en el marco del artículo 9° de la Ley N° 13.478.

¿Desea anotarse al programa Progresar?

**Usuario:** Si, quiero anotarme.  
 intent: anotarse  
 action: utter\_anotarse

**Chatbot:** ¡Genial! Para inscribirte necesito que me escribas tu correo electrónico para enviarte el formulario de inscripción.  
 action: email\_form  
 active\_loop: email\_form

**Usuario:** Mi email es fbrandan@uaco.unpa.edu.ar.  
 intent: email  
 active\_loop: null  
 action: utter\_despedida

**Chatbot:** Gracias por charlar conmigo, espero haber ayudado con sus consultas.

Figura 10 - Ejemplo de conversación con el chatbot

En la Figura 11 subsiguiente se encuentra la captura de pantalla que podría ver el usuario realizando esta interacción.

```

C:\WINDOWS\system32\cmd.exe - rasa shell
Bot loaded. Type a message and press enter (use '/stop' to exit):
Your input -> Hola, buenas tardes
¡Hey! ¿Cómo puedo serte de ayuda hoy?
Las inscripciones para el plan Proemplear y Progresar continúan en vigencia. A
demás contamos con cursos y talleres a disponibilidad.
Actualmente tenemos [NCursos] disponibles, y son los siguientes:
[ListadoCursosActuales]
Your input -> ¿Puedo obtener más información sobre progresar?
Es un programa integral de Becas Educativas que acompaña al alumno en todos lo
s niveles de formación durante su trayectoria académica, a través de un incent
ivo económico y un importante estímulo personal, que le permitirá poder avanza
r en sus estudios hasta completarlos. El objetivo del programa es fortalecer e
l ingreso, la permanencia y el egreso de los estudiantes, unificando todas las
becas educativas del gobierno nacional.
¿Desea anotarse al programa Progresar?
Your input -> ¿Cuáles son los requisitos para anotarme?
Sus requisitos son los siguientes:
  Ser argentino/a nativo/a; naturalizada/o o extranjera/o, con residencia leg
al de cinco (5) años en el país y contar con DNI.
  Ser alumna/o regular de una institución educativa.
  Tener entre diecisiete (17) y veinticuatro (24) años de edad cumplidos.
  Estudiantes avanzadas/os en la carrera, hasta treinta (30) años cumplidos.
  Estudiantes de enfermería sin límite de edad.
  Se extiende hasta 35 años de edad a las personas con hijas/os a cargo menor
es de 18 años pertenecientes a hogares monoparentales.
  Quedan exceptuadas/os del límite de edad las personas trans, pertenecientes
a pueblos indígenas, con discapacidad o refugiadas.
  Los ingresos de la/el joven y los de su grupo familiar no deberán ser super
iores a tres (3) Salarios Mínimos, Vitales y Móviles (SMVM), a excepción de qu
e los y las jóvenes solicitantes sean titulares de una pensión no contributiva
por invalidez otorgada en el marco del artículo 9° de la Ley N° 13.478.
¿Desea anotarse al programa Progresar?
Your input -> Si, quiero anotarme
¡Perfecto! Sólomente necesito tu correo electrónico para enviarte el formuari
o de inscripción.
Your input -> Mi email es fbrandan@uaco.unpa.edu.ar
Gracias por charlar conmigo, espero haber ayudado con sus consultas.
  
```

Figura 11 - Ejemplo de conversación con el chatbot

Se realizó un análisis de los niveles de confianza del bot, leyendo distintas oraciones como posibles entradas para las estructuras analizadas inicialmente como objeto de estudio. La estructura pre-entrenamiento basada en la estructura básica provista por Rasa + DialogFlow en español, y la estructura supervisada desarrollada para la aplicación basada en artículos científicos de Wikipedia y conocimiento de las reglas del negocio.

El objetivo de este análisis es conocer la eficacia de cada modelo en entender la intención del usuario en cada una de sus entradas ante la serie de pruebas. Para esto se utilizaron herramientas disponibles por el *framework* que permiten ver la precisión de entendimiento que percibe el bot ante cada entrada. Durante el análisis se usaron entradas complejas o poco comunes que la aplicación podría y debería entender para ser de mayor utilidad.

Para la organización de los datos recolectados se disponen las entradas analizadas junto a la intención real que planeaba el usuario, a continuación, se muestra la intención identificada por la aplicación que no siempre coincide con la real, luego el valor que obtuvo cada estructura para evaluar la entrada como esa intención. Podemos ver estos valores como un porcentaje de seguridad con el que la aplicación realiza la clasificación. Los resultados del análisis de confianza se encuentran en la Tabla 1.

Entrada	Intención real	Intención identificada	Estructura pre-entrenamiento	Estructura supervisada
Hollaa	saludo		0.93 (93%)	<b>1.00 (100%)</b>
Muchisisimas gracias!!	despedida		0.79 (79%)	<b>0.99 (99%)</b>
Q ncesito?	requerimientos		0.39 (39%)	<b>0.99 (99%)</b>
Ncsito info progresar	progresar		0.85 (85%)	<b>0.99 (99%)</b>
asta la prxma v	despedida		0.55 (55%)	<b>0.98 (98%)</b>
como estás?	saludo		0.90 (90%)	<b>0.93 (93%)</b>
pudo cargar mi curriculum?	carga_cv		0.80 (80%)	<b>0.99 (99%)</b>
que es prgresr	progresar		0.72 (72%)	<b>0.82 (82%)</b>
ayuda	contacto	cursos_disponibles	0.61 (61%)	<b>0.69 (69%)</b>
como funciona?	contacto		0.98 (98%)	<b>0.99 (99%)</b>
que crso stan dnado?	cursos_disponibles	contacto	0.59 (59%)	<b>0.60 (60%)</b>
me pdes dar cn algien	contacto		<b>0.51 (51%)</b>	<b>0.51 (51%)</b>
cauh asta lugo	despedida		0.63 (63%)	<b>0.86 (86%)</b>
jskdjv	-	saludo	0.70 (70%)	<b>0.90 (90%)</b>
???????	-	contacto	0.62 (62%)	<b>0.71 (71%)</b>
Promedio de Confianza	-		0.70 (70%)	<b>0.86 (86%)</b>

Tabla 1 - Resultados de análisis de confianza

Se puede observar el beneficio en general de utilizar una estructura personalizada que conoce las reglas de negocio al reconocer con mayor seguridad las necesidades y peticiones del usuario, además de conocer con seguridad palabras clave como “progresar” o sus derivadas. En general la estructura supervisada obtuvo mejores resultados, obteniendo un promedio de confianza de un 86% sobre el 70% que obtuvo la otra estructura analizada, pudiendo ser a causa del idioma en el que se trabaja, español. La estructura de pre-entrenamiento está principalmente configurada para el idioma inglés.

También observamos la necesidad de agregar la identificación de nuevas intenciones no previstas en el primer análisis. Se puede incluir una intención del usuario pidiendo ayuda sobre el funcionamiento básico del bot, ante casos de nunca haber operado un sistema similar. También se identifica la necesidad de agregar una intención “*fallback*” capaz de solicitar al usuario la reformulación de la entrada si la confianza es muy inferior al promedio, como se puede ver en las dos últimas entradas de prueba.

Como último paso de puesta a prueba del modelo, se prosigue a realizar correcciones al chatbot. Esto se puede hacer con entrenamiento interactivo manual, indicando por comandos cuál fue la intención que debía extraer de la entrada en caso la identificación sea errónea. A partir de esta indicación se agregan los mensajes pertinentes a las intenciones o se agregan nuevas historias para detectarlas correctamente la próxima vez.

En la Figura 12 se puede observar parte de este proceso de corrección. Se le indica al chatbot a reconocer correctamente un mensaje mal clasificado inicialmente.

Se le ingresa una entrada y la aplicación notifica qué intención detectó, al no ser correcta la clasificación “contacto” se le indica de forma manual que se trata de un mensaje con intención “cursos\_disponibles”. Luego la aplicación consulta si hay alguna entidad en el mensaje que no se detectó. Para finalizar muestra el historial completo de la conversación paso a paso, ahora clasificando correctamente la entrada inicial.

```
? Your input -> que crso stan dnado?
? Your NLU model classified 'que crso stan dnado?' with intent 'contacto' and there are no entities, is this correct? No
? What intent is it? 0.00 cursos_disponibles
? Please mark the entities using [value](type) notation que crso stan dnado?
-----
Chat History
# Bot You
1 action_listen
2 hola buenas
intent: saludo 1.00
3 utter_saludo
¡Hey! ¿Cómo puedo serte de ayuda hoy?
utter_oferta_actual
Las inscripciones para el plan Proemplear y Progresar continúan en vigencia. Además contamos con cursos y talleres a disponibilidad.
Actualmente tenemos [NCursos] disponibles, y son los siguientes:
[ListadoCursosActuales]
action_listen
4 que crso stan
dnado?
intent: cursos_disponibles 1.00
```

Figura 12 - Corrección manual de entrada mal clasificada

Al terminar la sesión de entrenamiento manual se crea un nuevo modelo que implementa lo aprendido a su estructura. Este modo de aprendizaje lleva más tiempo y esfuerzo, pero es más preciso al momento de detectar posibles mal entendidos.

## 6. CONCLUSIONES

El presente trabajo se ha centrado en el desarrollo de un chatbot orientado a la atención de público para la Oficina de Empleo de la ciudad de Caleta Olivia. Para llevar a cabo esta tarea ha sido fundamental iniciarse en el campo del procesamiento del lenguaje natural, entendiendo todas las técnicas necesarias para que una máquina interprete el lenguaje humano.

Se ha comenzado con una revisión de conceptos y de herramientas para la construcción del chatbot. Se ha presentado el desarrollo teórico necesario para comprender las etapas para procesar el lenguaje natural y los modelos de inteligencia artificial implicados en la creación del asistente.

Es importante confirmar la decisión de utilizar la plataforma de desarrollo Rasa, por su entrenamiento más adecuado con las estructuras supervisadas utilizadas, la posibilidad de expansión que ofrece al proyecto, su disponibilidad y escalabilidad.

Se generó un dataset específico que permite la identificación de 10 posibles intenciones, siendo nuestro chatbot capaz de atender estas solicitudes con un alto nivel de confianza (86%) no obstante, se nota la necesidad de continuar mejorando y ampliando el dataset.

Con la utilización de la librería spaCy se espera lograr un mejor entrenamiento, un buen nivel de confianza y mayor eficiencia de respuesta ajustando la semántica de Rasa al idioma español. Como trabajo futuro se pretende extender el modelo de entrenamiento prototipo con el objetivo de abarcar con mayor certeza todos los procesos de la Oficina de Empleo de Caleta Olivia, lo que requiere una investigación más detallada de la entrada y salida de datos de mesa de entrada. Además, se planea integrar el proyecto al sistema en actual funcionamiento del área.

También se pretende agregar nuevas intenciones genéricas que puedan surgir de un usuario en cuanto al funcionamiento del chatbot. Esto incluirá una respuesta por defecto en caso de que se comprenda una intención con baja confianza (<60% por ejemplo), al estilo “¿Podría reformular su mensaje?”, o “No comprendí eso, ¿me lo podrías repetir?”. La inclusión de un mensaje explicando el funcionamiento básico del chatbot también es una opción interesante de agregar, haciendo la aplicación más accesible para personas que nunca utilizaron un servicio similar. Se tiene como objetivo generar un modelo más robusto en general, capaz de comprender mensajes con errores de ortografía o de sintaxis por igual.

Finalmente, se plantea explorar la posibilidad de ampliar la accesibilidad de la aplicación incluyendo una extensión capaz de hacer reconocimiento de voz, ya que Rasa puede utilizarse en ambos extremos de la comunicación, recibiendo audio del usuario y traduciendo la entrada a texto para comprensión del bot; o comunicar las respuestas del bot de forma auditiva para un posible usuario que lo prefiera o necesite.

## AGRADECIMIENTOS

Los autores agradecen al equipo de trabajo del Laboratorio de Tecnologías Emergentes (LabTEm) del Instituto de Tecnología Aplicada (ITA) y a la Universidad Nacional de la Patagonia Austral (UNPA), por la colaboración y por todo el financiamiento realizado en el desarrollo de este trabajo.

## REFERENCIAS

BUNK, T. VARSHNEYA, D. VLASOV, V. NICHOL, A. (11 de Mayo de 2020). DIET: lightweight language understanding for dialogue systems.  
<https://arxiv.org/pdf/2004.09936.pdf>



- CATANIA, F., BECCALUVA, E., & GARZOTTO, F. (2020). The conversational agent "emoty" perceived by people with neurodevelopmental disorders: Is it a human or a machine? *Chatbot Research and Design*. [https://doi.org/10.1007/978-3-030-39540-7\\_5](https://doi.org/10.1007/978-3-030-39540-7_5)
- CERF, V. (1973). PARRY Encounters the DOCTOR. *DataTracker*. Retrieved from <https://datatracker.ietf.org/doc/html/rfc439>. <https://doi.org/10.17487/rfc0439>
- COLBY, K. M. (1975). Artificial Paranoia. A Computer Simulation of Paranoid Processes. <https://doi.org/10.1016/c2013-0-02631-x>
- COURTAUDON, A. (25 de septiembre de 2012). La gestion intelligente avec le numérique : une nouvelle dynamique pour les villes et territoires français. *Blog Administration Numérique - MARKESS International*. <https://archive.wikiwix.com/cache/index2.php?url=http%3A%2F%2Fblog.administrati onnumerique.markess.com%2F2012%2F09%2F1a-gestion-intelligente-avec-le-numerique-une-nouvelle-dynamique-pour-les-villes-et-territoires-francais%2F#federation=archive.wikiwix.com>
- GRAVES, A., MOHAMED, A.-RAHMAN, & HINTON, G. (2013). Speech recognition with deep recurrent neural networks. 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. <https://doi.org/10.1109/icassp.2013.6638947>
- HERNÁNDEZ, L. (2019). Chatbot Orientado a la Atención del Cliente en la Plataforma de Consulmed de Venezuela, C.A. Universidad católica Andrés Bello.
- HOCHREITER, S., & SCHMIDHUBER, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735-1780. <https://doi.org/10.1162/neco.1997.9.8.1735>
- INSTITUTO NACIONAL DE ESTADÍSTICA Y CENSOS (INDEC). (2020). Acceso y uso de tecnologías de la información y la comunicación. EPH Cuarto trimestre de 2020. *Informes Técnicos Ciencia y Tecnología*, 5(1), 1-16. Recuperado de [https://www.indec.gob.ar/uploads/informesdeprensa/mautic\\_05\\_213B13B3593A.pdf](https://www.indec.gob.ar/uploads/informesdeprensa/mautic_05_213B13B3593A.pdf)
- JIAO, A. (2020). An intelligent chatbot system based on entity extraction using Rasa NLU and Neural Network. *Journal of Physics: Conference Series*, 1487(1). <https://doi.org/10.1088/1742-6596/1487/1/012014>
- MANTHA, M. (2021, August 19). Introducing diet: State-of-the-art architecture that outperforms fine-tuning Bert and is 6x faster to train. *Rasa*. Retrieved from <https://rasa.com/blog/introducing-dual-intent-and-entity-transformer-diet-state-of-the-art-performance-on-a-lightweight-architecture/>
- OLAH, C. (27 de Agosto de 2015). Understanding Lstm Networks. <http://colah.github.io/posts/2015-08-Understanding-LSTMs>.
- ORACLE. (2022). What is a chatbot?. *Oracle Cloud Infrastructure*. <https://www.oracle.com/chatbots/what-is-a-chatbot/>
- SCHMITT, X., KUBLER, S., ROBERT, J., PAPADAKIS, M., & LETRAON, Y. (2019). A replicable comparison study of NER software: STANFORDNLP, NLTK, opennlp, spacy, gate. 2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS). <https://doi.org/10.1109/snams.2019.8931850>
- THOMPSON, H. J. (2021). Clasificación de Preguntas en Idioma Español en Sistemas de Búsqueda de Respuestas. Universidad Nacional de San Luis.
- TURING, A. M. (1950). Computing machinery and intelligence, *Mind*, 49, (433-460). <https://doi.org/10.1093/mind/LIX.236.433>
- VARSÁVKY, O. (1974). Capítulo 1. Introducción, En R. Ladenheim, H. L. González, *Estilos tecnológicos* (pp. 28-39). Buenos Aires, Argentina. Biblioteca Nacional Mariano Moreno.
- WEIZENBAUM, J. (1966). ELIZA-a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1). <https://doi.org/10.1145/365153.365168>

WINOGRAD, T. (1971). Procedures as a representation for data in a computer program for understanding natural language. Massachusetts Inst of Tech Cambridge Project Mac.

## ANEXO A

En este Anexo se detalla la especificación completa del modelo construido con una breve explicación de la función de cada archivo para el entrenamiento del chatbot.

En el archivo “*nlu.yml*” de la Figura Anexo 1 se encuentran los datos NLU que describen las posibles intenciones que puede tener un usuario al expresarse. Se le entregan al bot algunas oraciones de ejemplo para asociar a cada intención. Al recibir una entrada, el bot busca entre estas palabras coincidentes para calcular la confianza de detección de intención.

<p>nlu:</p> <ul style="list-style-type: none"> <li>- intent: saludo</li> </ul> <p>examples:  </p> <ul style="list-style-type: none"> <li>- Hola, ¿cómo estás?</li> <li>- Hola</li> <li>- Buen día</li> <li>- Buenas tardes</li> </ul> <ul style="list-style-type: none"> <li>- intent: proemplear</li> </ul> <p>examples:  </p> <ul style="list-style-type: none"> <li>- ¿Qué es proemplear?</li> <li>- ¿Podes contarme qué es proemplear?</li> <li>- Proemplear?</li> </ul> <ul style="list-style-type: none"> <li>- intent: progresar</li> </ul> <p>examples:  </p> <ul style="list-style-type: none"> <li>- ¿Qué es progresar?</li> <li>- Me podés dar más información sobre Progresar?</li> <li>- Progresar?</li> <li>- Cómo es Progresar?</li> </ul> <ul style="list-style-type: none"> <li>- intent: requerimientos</li> </ul> <p>examples:  </p> <ul style="list-style-type: none"> <li>- ¿Cuáles son los requerimientos?</li> <li>- ¿Qué necesito para esto?</li> <li>- ¿Se necesita algo?</li> <li>- ¿Qué requisitos hay?</li> <li>- Tengo que cumplir algo?</li> <li>- ¿Qué requisitos necesito para anotarme?</li> </ul> <ul style="list-style-type: none"> <li>- intent: contacto</li> </ul> <p>examples:  </p> <ul style="list-style-type: none"> <li>- ¿Con quién tengo que hablar?</li> <li>- ¿Tengo que hablar con alguien?</li> <li>- ¿Puedo hablar en persona?</li> </ul>	<ul style="list-style-type: none"> <li>- intent: anotarse</li> </ul> <p>examples:  </p> <ul style="list-style-type: none"> <li>- Sí, quiero anotarme</li> <li>- Anotame</li> <li>- Quiero entrar</li> <li>- Quiero anotarme</li> </ul> <ul style="list-style-type: none"> <li>- intent: email</li> </ul> <p>examples:  </p> <ul style="list-style-type: none"> <li>- Mi correo es ejemplo@ejemplo.com</li> <li>- Mi email es algo@ejemplo.com</li> <li>- ejemplo@ejemplo.com</li> </ul> <ul style="list-style-type: none"> <li>- intent: cursos_disponibles</li> </ul> <p>examples:  </p> <ul style="list-style-type: none"> <li>- ¿Qué cursos están disponibles?</li> <li>- A qué me puedo anotar?</li> <li>- información sobre cursos</li> <li>- Qué talleres hay?</li> <li>- Hay algún taller disponible?</li> </ul> <ul style="list-style-type: none"> <li>- intent: carga_cv</li> </ul> <p>examples:  </p> <ul style="list-style-type: none"> <li>- ¿Cómo puedo cargar mi CV?</li> <li>- Quiero cargar mi CV</li> <li>- Carga cv</li> <li>- Quiero cargar mi curriculum</li> <li>- Puedo cargar mi curriculum vitae?</li> <li>- Curriculum Vitae</li> </ul> <ul style="list-style-type: none"> <li>- intent: despedida</li> </ul> <p>examples:  </p> <ul style="list-style-type: none"> <li>- Adiós, muchas gracias</li> <li>- Gracias</li> <li>- Chau</li> </ul>
---	--

<ul style="list-style-type: none"> <li>- ¿Cuándo está abierto?</li> <li>- ¿Cuándo abre la oficina?</li> <li>- ¿Dónde está la oficina?</li> <li>- Dónde?</li> <li>- Con quién me puedo contactar?</li> <li>- ¿Cuál es el número de teléfono?</li> <li>- Tienen teléfono?</li> <li>- Contacto?</li> </ul>	<ul style="list-style-type: none"> <li>- Hasta luego</li> </ul>
---	---

Figura Anexo 1 – Archivo “*nlu.yml*”

En el archivo “*domain.yml*” de la Figura Anexo 2 están descritas las potenciales respuestas que el bot puede entregar. En el caso de múltiples posibles respuestas, el bot elegirá al azar una salida para responder.

<p>responses:</p> <p>utter_saludo:</p> <ul style="list-style-type: none"> <li>- text:  </li> <li style="padding-left: 20px;">¡Hola! ¿Cómo puedo ayudarte?</li> <li>- text:  </li> <li style="padding-left: 20px;">¡Hey! ¿Cómo puedo serte de ayuda hoy?</li> </ul> <p>utter_oferta_actual:</p> <ul style="list-style-type: none"> <li>- text:  </li> <li style="padding-left: 20px;">Las inscripciones para el plan Proemplear y Progresar continúan en vigencia. Además, contamos con cursos y talleres a disponibilidad.</li> <li style="padding-left: 20px;">Actualmente tenemos [NCursos] disponibles, y son los siguientes: [ListadoCursosActuales]</li> </ul> <p>utter_oferta_requisitos:</p> <ul style="list-style-type: none"> <li>- text:  </li> <li style="padding-left: 20px;">Para los cursos necesitamos los siguientes requisitos:</li> <li style="padding-left: 40px;">Ser mayor de 18 años, o contar con autorización de tutor/a.</li> <li style="padding-left: 40px;">Completar el formulario de inscripción [formulario_inscripcion].</li> <li style="padding-left: 40px;">Haber leído y aceptado el protocolo de Covid [protocolo_COVID].</li> <li style="padding-left: 20px;">¿Desea inscribirse a algún curso?</li> </ul> <p>utter_proemplear:</p> <ul style="list-style-type: none"> <li>- text:  </li> <li style="padding-left: 20px;">Proemplear es un programa del Ministerio de Trabajo, Empleo y Seguridad Social de la Nación. Tiene como objetivo aumentar las oportunidades de inserción laboral de trabajadores/as desocupados/as. Ofrece a las empresas incentivos económicos para la contratación y formación de trabajadores.</li> <li style="padding-left: 20px;">¿Desea anotarse al programa Proemplear?</li> </ul> <p>utter_proemplear_requisitos:</p> <ul style="list-style-type: none"> <li>- text:  </li> <li style="padding-left: 20px;">Sus requisitos son los siguientes:</li> <li style="padding-left: 40px;">Ser un/a joven de entre 18 y 24 años.</li> <li style="padding-left: 40px;">Participar de PROG.R.ES.AR, el Programa Jóvenes con Más y Mejor Trabajo, trabaja con Seguro de Capacitación y Empleo o ser desocupado del Seguro por Desempleo.</li> </ul>
---

¿Desea anotarse al programa Proemplear?

utter\_progresar:

- text: |

Es un programa integral de Becas Educativas que acompaña al alumno en todos los niveles de formación durante su trayectoria académica, a través de un incentivo económico y un importante estímulo personal, que le permitirá poder avanzar en sus estudios hasta completarlos. El objetivo del programa es fortalecer el ingreso, la permanencia y el egreso de los estudiantes, unificando todas las becas educativas del gobierno nacional.

¿Desea anotarse al programa Progresar?

utter\_progresar\_requisitos:

- text: |

Sus requisitos son los siguientes:

Ser argentino/a nativo/a; naturalizado/a o extranjero/a, con residencia legal de cinco (5) años en el país y contar con DNI.

Ser alumna/o regular de una institución educativa.

Tener entre diecisiete (17) y veinticuatro (24) años de edad cumplidos.

Estudiantes avanzadas/os en la carrera, hasta treinta (30) años cumplidos.

Estudiantes de enfermería sin límite de edad.

Se extiende hasta 35 años de edad a las personas con hijas/os a cargo menores de 18 años pertenecientes a hogares monoparentales.

Quedan exceptuadas/os del límite de edad las personas trans, pertenecientes a pueblos indígenas, con discapacidad o refugiadas.

Los ingresos de la/el joven y los de su grupo familiar no deberán ser superiores a tres (3) Salarios Mínimos, Vitales y Móviles (SMVM), a excepción de que los y las jóvenes solicitantes sean titulares de una pensión no contributiva por invalidez otorgada en el marco del artículo 9° de la Ley N° 13.478.

¿Desea anotarse al programa Progresar?

utter\_contacto:

- text: |

Contáctate con la Oficina de Empleo al 485-0996 de lunes a viernes, de 08:00 a 14:00hs.

En la dirección B° 3 de Febrero-Antártida Argentina y Gdor. Gregores.

utter\_anotarse:

- text: |

¡Genial! Para inscribirte necesito que me escribas tu correo electrónico para enviarte el formulario de inscripción.

- text: |

¡Perfecto! Solamente necesito tu correo electrónico para enviarte el formulario de inscripción.

utter\_carga\_CV:

- text: |

Para cargar tu Curriculum Vitae tienes que enviarlo a esta dirección de correo electrónico ([intermediacionlaboralmco@gmail.com](mailto:intermediacionlaboralmco@gmail.com)) junto con una foto con fondo blanco de 4cmx4cm.

utter\_despedida:

- text: | Gracias por charlar conmigo, espero haber ayudado con sus consultas.

- text: | Hasta la próxima, gracias por hablar conmigo.

Figura Anexo 2 - Archivo “domain.yml”



En el archivo “*stories.yml*” de la Figura Anexo 3 se describen “historias” de entrenamiento o rutas de conversación que puede seguir un usuario y el bot debe saber dirigir.

<pre> stories: - story: saludo_oferta_inscripcion   steps:   - intent: saludo   - action: utter_saludo   - action: utter_oferta_actual   - intent: requerimientos   - action: utter_oferta_requisitos   - intent: anotarse   - action: utter_anotarse   - action: email_form   - active_loop: email_form  - story: saludo_proemplear_inscripcion   steps:   - intent: saludo   - action: utter_saludo   - action: utter_oferta_actual   - intent: proemplear   - action: utter_proemplear   - intent: requisitos   - action: utter_proemplear_requisitos   - intent: anotarse   - action: utter_anotarse   - action: email_form   - active_loop: email_form  - story: saludo_progresar_inscripcion   steps:   - intent: saludo   - action: utter_saludo   - action: utter_oferta_actual   - intent: progresar   - action: utter_progresar   - intent: requisitos   - action: utter_progresar_requisitos   - intent: anotarse   - action: utter_anotarse   - action: email_form   - active_loop: email_form </pre>	<pre> - story: saludo_cursos_inscripcion   steps:   - intent: saludo   - action: utter_saludo   - action: utter_oferta_actual   - intent: cursos_disponibles   - action: utter_oferta_actual   - intent: requisitos   - action: utter_oferta_requisitos   - intent: anotarse   - action: utter_anotarse   - action: email_form   - active_loop: email_form  - story: saludo_contacto   steps:   - intent: saludo   - action: utter_saludo   - action: utter_oferta_actual   - intent: contacto   - action: utter_contacto   - action: utter_despedida  - story: saludo_carga_CV   steps:   - intent: saludo   - action: utter_saludo   - action: utter_oferta_actual   - intent: carga_cv   - action: utter_carga_CV   - action: utter_despedida </pre>
---	---

Figura Anexo 3 - Archivo “*stories.yml*”

En el archivo “*forms.yml*” de la Figura Anexo 4 se encuentran los slots de memoria, o datos que puede requerir el bot del usuario y los formularios que los recolectan. Estos datos pueden ser tratados durante la conversación para personalizar la experiencia del usuario o para utilizar en algún proceso.

```

slots:
  email:
    type: text
    mappings:
      - type: from_text
    conditions:
      - active_loop: email_form
        requested_slot: email
forms:
  email_form:
    required_slots:
      - email

```

Figura Anexo 4 - Archivo “forms.yml”

Por último, en el archivo “rules.yml” de la Figura Anexo 5 se describen las reglas o conversaciones estrictas que debe seguir el bot, sin importar en qué parte de una conversación se encuentre actualmente.

```

rules:
  - rule: activar formulario email
    steps:
      - intent: anotarse
      - action: email_form
      - active_loop: email_form

  - rule: entregar formulario email
    condition:
      - active_loop: email_form
    steps:
      - action: email_form
      - active_loop: null
      - action: utter_despedida

```

Figura Anexo 5 - Archivo “rules.yml”