Relazione su “AlexaBot”

**Introduzione ad AlexaBot :**

1.0 )

Alexabot è un bot basata su NLP dall’inglese Natural Language Processing ossia l’elaborazione del linguaggio naturale , il processo di trattamento automatico delle informazioni scritte in una lingua naturale.

È un ramo dell’intelligenza artificiale applicata.

L’utilizzo del NLP ha reso Alexabot più viva ed interagibile , togliendola dalla monotonia e da frasi dettate precisamente che in caso non fossero corrette ne risulterebbe un errore e quindi nessun feedback di ritorno.

Quindi in questo caso se dovessi chiedere ad Alexa una delle frasi insegnate , Alexa mi ritorna un feedback. (A metà progetto ci sarà un video esempio con Alexa).

Alexa è come una bambina appena nata che deve ancora imparare e capire la natura delle frasi , quindi medianti fasi di Training , le potremmo dare vita anche ad una sua personalità tracciando una funzione (in Machine learning ci sono 2 tipi di apprendimento supervised learning e unsupervised learning , in questo caso utilizziamo il supervised algorithm learning ossia l’analisi dei data provenienti dal Training e produrre una retta in una funzione che preveda i prossimi dati, rendendola effettivamente intelligente), quindi non è solamente un insegnamento di frasi ma anche di poterla rendere più viva e meno robotica, ciò permette di avere un probabile feeling personale tra bot e umano.

Una delle probabili limitazioni di un bot di questo genere , è l’ironia e sarcasmo.

Noi elaboriamo queste interpretazioni come l’ironia e sarcasmo come usuale a seconda del contesto , quindi sappiamo capirne subito l’essenza. Un bot con NPL farà molta fatica a dedurre queste ambiguità risultandone magari con feedbacks diversi da quelli desiderati , per risolvere questi problemi nelle frasi in Training facciamo una analisi semantica applicando in varie Entità alcune parole riferimento sottoscrivendo anche i sinonimi , affinchè il bot possa capirne meglio la frase , e successivamente altre fasi di analisi più complesse.

**Approccio al bot:**

2.0)

Come detto precedentemente AlexaBot è basata su NLP per evitare la monotonia o essere bloccati su determinati frasi da mettere come input.

2.1)

La scelta iniziale è stata se approcciarmi verso iOS o android , e la scelta alla fine è stata iOS per la semplificazione di alcune azioni di sistema , quindi per esempio è più facile accedere alle varie fasi del Life cycle di iOS , alla facilità d’uso di swift e continui aggiornamenti di funzioni che magari su Java ne richiederebbe molteplici righe mentre su Swift viene svolto in modo più istantaneo ed intuitibile senza farti venire mal di testa , per lo sviluppo della GUI con la metal library con oggetti grafici moderni e per ultimo perchè possiedo un iphone per testarlo fisicamente. Oltretutto per sviluppare in iOS bisogna essere forniti di un mac obbligatoriamente , dato che l’IDE può essere SOLO utilizzato dai mac , quindi avendo già a disposizione questi materiali di base , ne ho approfittato di testare le mie capacità da autodidatta e valutare in tempi stretti cosa potevo tirare fuori e cercare di renderlo più presentabile possibile.

2.2)

Definito l’IDE e linguaggio da utilizzare , ho cercato le scelte più optabili e fattibili per creare un bot , dato che quando penso ad un bot mi immagino un robottino che si autogestisca con una sua intelligenza.

Con queste mie pretese ho passato 1 giorno a ricercare ed infine ho trovato dialogFlow che avrebbe fatto al caso mio per poi studiarlo e capirne come si utilizzasse.

DialogFlow è proprietà della google per lo sviluppo di interazioni computer-umano basati puntualmente sul NLP. Viene utilizzata molto professionalmente in ambiti dove viene richiesto un assistente che esegua azioni o risposte a clienti/persone in modo naturale. Come per esempio facebook messenger supporta l’utilizzo di bot per gestire le pagine , al fine di rispondere ai clienti. Esempio : un cliente entra alla pagina del negozio X e chiede a che ora apre il negozio , il bot elabora la frase e risponde con la risposta corretta.

Questo per far capire come attraverso un bot si può gestire una moltitudine di chat e richieste da tanti clienti , senza che una persona fisica debba stare a rispondere , dando un boost ad una pagina/azienda nelle relazioni con i clienti.

2.3)

Prima di iniziare a studiare il funzionamento di dialogFlow , ho visto prima se ci fossero delle librerie già dedicate di dialogFlow per swift , al fine di interfacciare il bot e una app iOS.

Ho trovato la libreria di dialogFlow e si chiama ApiAI , poi mi sono assicurato che le funzioni non erano deprecated e quindi aggiornate ed usabili in Swift 4 (l’ultima versione di swift) , e dopo aver confermato di poterle utilizzare ho iniziato lo studio di dialogFlow per 1 settimana.

Ho scritto su un foglio delle piccole vignette per rendere l’idea su come volessi gestire queste conversazioni iniziali di testing :

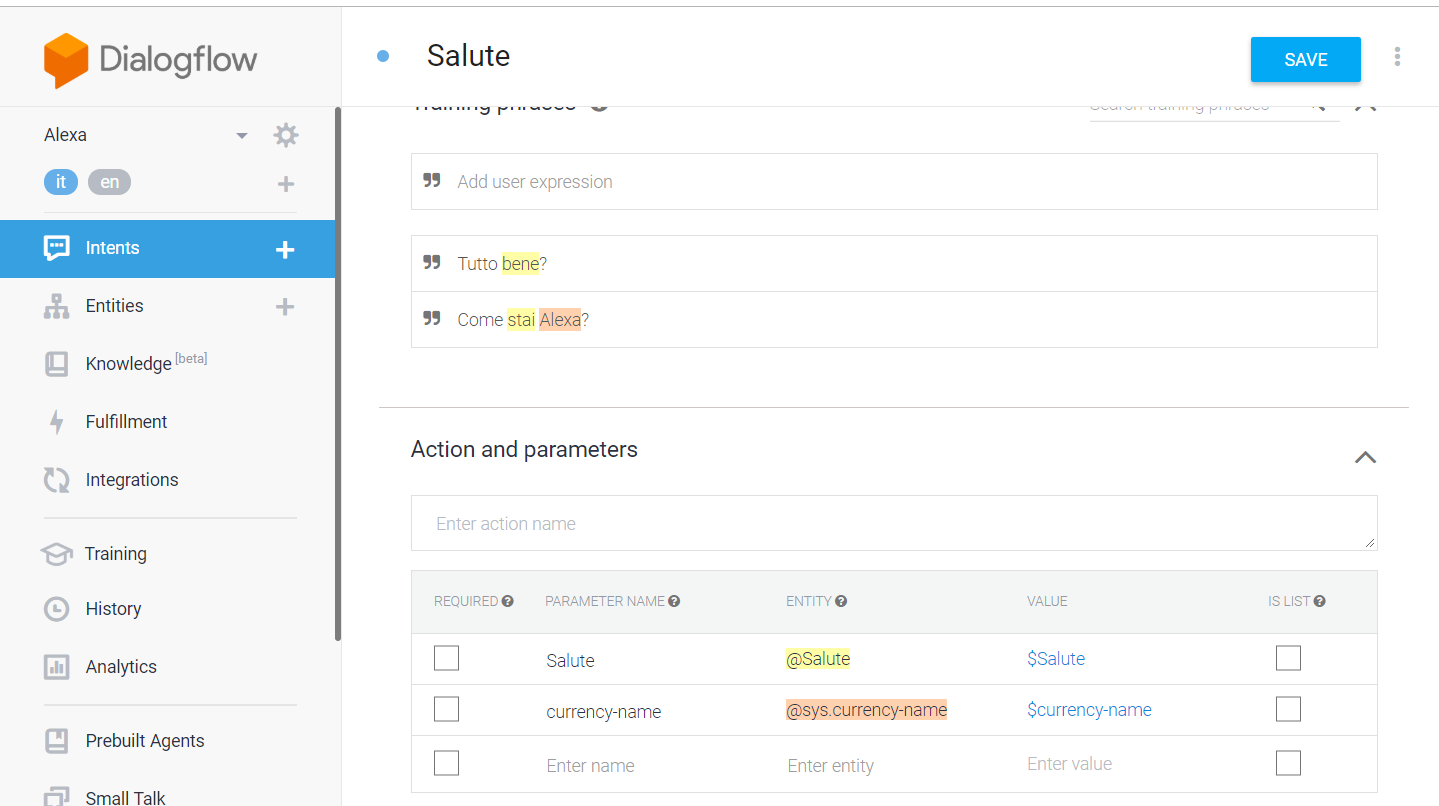


2.4)

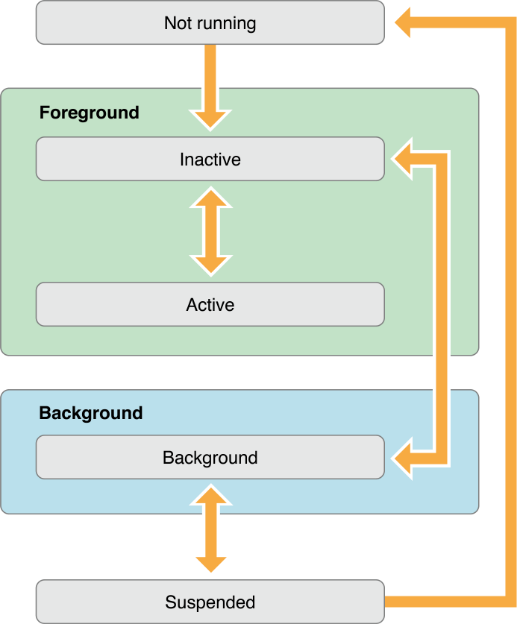
Sessione di training :

La prima sessione di training è stata su domande iniziale come per esempio Come ti chiami , come stai , etc

Come precedentemente ho menzionato , il bot capisce mediante Entità , parlando in modo pratico invece è sottolineare una parola e convertirla come Entità riconoscibile al bot. Esempio : nella frase “Come stai?” , prendo la parola “stai” e lo catalogo come entità nominata Salute , ciò permette al bot di capire quale è la parola importante da prendere in considerazione e successivamente implementare risposte che abbiano riferimenti / sinonimi a quella Entità. Ovviamente al bot dobbiamo definire anche tutti i sininomi che vogliamo usare al fine di rendere il bot più intelligente e intuitiva possibile.



Dopo aver fatto due training per confermare che le frasi semplici diano le risposte corrette , ho iniziato la parte di integrazione su swift che è stato molto semplice , dove praticamente ho sfruttato una funzione di sistema che si situa nel before launching , quindi ancora prima che l’utente veda la prima interfaccia dell’app , ha già caricato le informazioni richieste , tra cui la configurazione via access token tra dialogFlow e swift , tutto questo grazie alle librerie ApiAI. Successivamente una interfaccia visiva per ora semplice , quindi un testo dove visualizziamo le risposte del bot e un editText per scrivere e mandare le stringhe al bot.



2.5)

Integrazione di RESTful API :

L’app a questo punto si potrebbe dire terminata , svolge il suo compito di fare una chiaccherata semplice senza alcun riferimento a dati reali , ma potrebbe ovviamente se dovessimo integrare link esterni da cui trarre informazioni. Sarebbe stato fin troppo limitante un bot con una intelligenza circa infinitesima , che può solo rispondere a domande semplici , volevo trarne il meglio.

Allora ho cercato come avrei potuto chiedere il meteo o informazioni inerenti al mondo , dargli tutte le fonti possibili per soddisfare le esigente dell’utente.

Ho visto che era possibile integrare degli webhook al bot , ossia dei callback HTTP/HTTPS che possiamo definire noi , quindi mediante dei triggers possiamo poi svolgere azioni.

A questo punto ho cercato chi poteva offrire servizi dando API Keys collegato ad un database di informazioni del meteo , da cui avrei potuto trarne dati.

Come mediatore tra il bot e il webhook ho utilizzato node.js per la gestione del lato server , e dato che i normali domain non supportano i simil-framework come node.js , dobbiamo utilizzare dei cloud platform , ossia dei servizi offerti attraverso internet con configurazioni personalizzabili. In questo caso ho preso heroku come primo cloud platform dove ho redetto il mio webhook che avrei utilizzato per trarre i dati.

2.5.1)

Sviluppo nel cloud platform heroku :

Heroku è un cloud platform molto semplice che da a disposizione storage per svilupparci , il mio unico interesse nell’utilizzare Heroku è solo la gestione dei webhook utilizzando node.js che è attivamente supportato. Per utilizzare Heroku serve la CLI , non essendoci un file manager online , i pacchetti vengono scaricati e caricati attraverso scripts bash , quindi ho installato i moduli richiesti per utilizzare node e poi i moduli per utilizzare heroku. Esempio di comando script di heroku è heroku git:clone –a alexabotai , dove stiamo clonando la git repository situata nel cloud platform nel path attuale , oppure per caricare nel cloud file modificati diamo heroku git push master , dove facciamo un deploy con le modifiche.

2.5.2)

Sviluppo nel google cloud functions :

Heroku è stato una scelta ottimale per approcciarmi ai cloud platforms , ma non da altre features che mi avrebbe permesso di gestire meglio i trigger in http o https o generalmente i routes , allora mi sono spostato sulla piattaforma di google cloud , che ha a disposizioni tutti i strumenti e documentazioni ben fatte al fine di rendermi lo sviluppo dell’app meno stressante , tenendo conto che mi mancavano pochissimi giorni.

Il forte di google cloud è che ha disposizione di moduli fatti da loro creando un ecosistema con funzioni dedicati e fatti da google. Quindi non avresti dovuto cercare accorgimenti da terze parti per rendere al meglio efficente il tuo webhook , quando c’è google che dispone tutti i strumenti per lavorare nel suo cloud platform.

Oltretutto la gestione della CLI è molto semplice e hai anche a disposizione il file manager , cosa che ha ridotto i tempi di sviluppo drasticamente.

3)

Conclusione : ecco il video

<https://drive.google.com/file/d/1bsTRVyssfVnU_bWpoViHaDZ46jNOxs9I/view?usp=sharing>

Il link per vedere il video presentazione

