APP FOR PLACING ORDERS AND BARGAINING WITH AI AT A RESTAURANT

Shounak Mulay, Rushabh More, Priyanka Wagh
Department of Computer Engineering,
Shivajirao S Jondhale College of Engineering, Dombivli (E),
Mumbai University, Maharashtra, India

Prof. Saroja T.V
Associate Professor, Department of Computer Engineering,
Shivajirao S Jondhale College of Engineering, Dombivli (E),
Mumbai University, Maharashtra, India

ABSTRACT
The proposed system attempts to implement various new technologies to an app for enabling users to place orders and bargain the prices with a machine learning powered bot. Such a system can provide new experience to the customers while increasing business profitability and efficiency with the help of collected data. This system comprises of 4 parts. The Android app at the frontend, and the database, natural language processing platform and the machine learning model to predict price as the backend deployed on the cloud. The app is designed with modular architecture pattern so that changes can be made easily to fit any business model. Since the predictions for bargaining are provided by a machine learning algorithm, the system performance and accuracy is likely to improve with additional data.

Keywords: Machine learning, Prediction, Android, Regression, Cloud.

Cite this Article: Shounak Mulay, Rushabh More, Priyanka Wagh and Prof. Saroja T.V, App for Placing orders and Bargaining with AI at a Restaurant, International Journal of Computer Engineering and Technology, 10(1), 2019, pp. 196-202.

1. INTRODUCTION
Dialogue systems and conversational agents - including chatbots, personal assistants and voice control interfaces - are becoming ubiquitous in modern society. Examples of these include personal assistants on mobile devices, technical support help over telephone lines, as well as online bots selling anything from fashion clothes and cosmetics to legal advice and self-help therapy. Combining these agents with the upcoming technology of machine learning we can achieve some interesting results.
Our system is based on Dialogflow - a Natural Language Processing (NLP) platform that can converse with users. The logic for the NLP platform that enables it to bargain with users is provided by a machine learning model - more specifically a Tensorflow Regression model. The initial dataset used to train the model is generated using observations in the real world. The performance of this model is likely to improve as we incrementally train it on new data collected from the customers using the system. The model is deployed on the cloud (Google Cloud Platform ML Engine).

This system comprising of the NLP platform, the machine learning model and the databases associated with it can be presented to the user in any form like an app or a website. Here we demonstrate an android app based on a Model-View-ViewModel (MVVM) architecture pattern. Implementing MVVM pattern gives us the ability to easily make changes to the app and also customise the app to fit the different need of different businesses.

2. SYSTEM OVERVIEW

Cloud Computing is a major part of the technology industry today. It enables us to use the most innovative technology, accessible from anywhere to help get our application to the users quickly without worrying about the underlying infrastructure and maintenance costs. Our system leverages this power of cloud computing to provide a modular and reactive platform to implement a bargaining chatbot and food ordering system.

Our system comprises of the following components:
1. Android App
2. Online Database
3. Machine Learning Model
4. Dialogflow - NLP Platform

2.1. Android App

The android app is the frontend for the system. Users will interact with the bot and other backend systems through this app. This app follows a reactive design pattern using Model-View-ViewModel (MVVM) architecture[1]. The main players in the MVVM pattern are:

- The View - that informs the ViewModel about the user’s actions.
- The ViewModel - exposes streams of data relevant to the View
- The DataModel - abstracts the data source. The ViewModel works with the DataModel to get and save the data.

The View is the actual user interface in the app. The View observers data streams exposed by the ViewModel and reacts accordingly. On the other hand the Views also notify the ViewModel about different actions. View has a reference to the ViewModel but ViewModel has no information about the View. This is the modularity and separation of concerns provided by MVVM. We can change the view without altering anything in the ViewModel and the app would still work.
App for Placing orders and Bargaining with AI at a Restaurant

The DataModel, also referred to as a Repository, comprises of all the data related operation for the app. The Repository is the single source of truth. Handling the local and remote data sources is the job of a repository. Other parts of the app should not be concerned if the data was fetch locally or from the network. The ViewModel send and receives data from the DataModel which decides how and where to fetch and update the data. Just like the ViewModel the Repository enables us to change the underlying data implementation without altering the rest of the app.

Thus any changes as per the requirements of different businesses can be easily implemented with minimal effort.

2.2. Online Database

The functionality of the reactive app can be maximised with a real-time database. A realtime database is a database which uses real-time processing to handle workloads, i.e any changes to the database are reflected in its implementation, in our case the android app, within milliseconds of the change.

In our system we are using Firebase Firestore - a No-SQL real-time database. Data in firestore is stored in the form of collections and documents. Collections contain multiple documents and documents themselves can contain fields and sub-collections. Fields are key value pairs that store data.

This database contains all of the restaurant and user data. Apart from regular operations and report generations, this data is valuable asset for the improvement of the machine learning model. This data can be ingested to train the model at regular intervals using Firebase Cloud Functions.

2.3. Machine Learning Model

Machine learning (ML) is a category of algorithm that allows software application to become more accurate in predicting outcomes without being explicitly programmed.

Our system uses a basic Tensorflow Regression model to predict the price of a drink given other parameters such as drink type, current cost, offered cost, etc. that the bot will use to bargain with the user. The initial dataset used to train the model was generated on real world transaction observations and some assumptions. It consists of 80,000 rows of data. Each row represents a drink transaction and a label i.e the value to be predicted by the model given other parameters.

We used the Keras API to build our model.

The summary of the model is as follows:

| Layer Type | Output Shape | Param # |
|------------|--------------|---------|
| Inputs (Dense) | (None, 28) | 700 |
| dense_0 (Dense) | (None, 64) | 1152 |
| outputs (Dense) | (None, 1) | 65 |

A machine learning model is made up of multiple layers. Our model has 3 layers - one input layer with 20 nodes, one output layer with one node as this is a regression model and one hidden layer with 64 nodes. All 3 layers are Dense layers. A dense layer is just a regular layer of neurons in a neural network. Each neuron receives input from all the neurons in the previous layer, thus densely
connected. The layer has a weight matrix $W$, a bias vector $b$, and the activations of previous layer $a$.

$$Y = \sum (weight * input) + bias$$

To train a machine learning model we need Activation Functions for each layer, an Optimizer for the model, and a Loss Function that the model will minimize while training.

Activation Functions are really important for the Neural Network to learn and make sense of something really complicated. These functions decide whether a node is activated or not by checking its $Y$ value. Their main purpose is to convert an input signal of a node to an output signal. That output signal now is used as an input in the next layer in the stack. We have used the ReLu\(^\text{[2]}\) activation function for our model. ReLu is the most widely used activation function. ReLu stands for Rectified Linear Unit. It is defined as:

$$A(x) = \max(0,x)$$

It gives an output $x$ if $x$ is positive and 0 otherwise.

The main advantage of ReLu function over other activation functions is that it does not activate all neurons at the same time. Only a few neurons are activated making the network sparse and thus efficient and easy for computation.

Loss Function is a method of evaluating how well a algorithm models a dataset. If the prediction of the model is off by a large amount the loss function will output a higher number. Machine Learning Model learns by minimizing this loss amount. The loss function used for our model is Mean Absolute Error (MAE)\(^\text{[3]}\). Mathematical formulation for MAE is:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

MAE is measured as the average of sum of absolute differences between predictions and actual observations. MAE needs more complicated tools such as linear programming to compute the gradients plus it is more robust to outliers since it does not make use of square like Mean Squared Error.

Optimizer tie together the loss function and model parameters by updating the model in response to the output of the loss function. In simple terms, optimizer shapes and molds the model into its most accurate possible form by changing the weights. The loss function is the guide for the optimizer telling it whether it’s moving in the right or wrong direction. Optimizers adjust the weights in steps of the Learning Rate provided to it. Learning Rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect to the loss gradient. In our model we use the Adam Optimizer\(^\text{[4]}\). Adam is an adaptive
learning rate optimization algorithm that’s been designed specifically for training deep neural networks. It computes individual learning rates for different parameters. It can be looked at as a combination of **RMSprop** and **Stochastic Gradient Descent**.

With these functions and the dataset we trained our model for 150 epochs. The loss for the prediction was minimized to around 6 ₹ of the labeled value. The prediction error can be further reduced by using a more complex model or using more real world data. The following figure shows the train and validation error for the model with epochs on x axis and MAE on the y axis.

![Graph showing train and validation error](image)

The plot of prediction error and count for 16000 test samples shows that the model has almost no error for around 13000 samples of the total 16000.

![Graph showing prediction error and count](image)

This model is deployed on the Google Cloud Platform ML Engine. ML Engine deploys the model on the cloud to serve prediction and provides a web endpoint to integrate ML into our application. In our system the ML Engine gets called from the Dialogflow backend logic whenever it requires predictions. Using ML Engine also gives us the ability to deploy multiple variants of the model and serve different models to different users or switch default models instantly to change the prediction behaviour of the bot.

### 2.4. Dialogflow - NLP Platform

Dialogflow is a Natural Language Processing Platform that can identity user input and respond to it. Dialogflow works by mapping user input to Intents and giving the corresponding response. Intents are representation of one dialog turn within the conversation. An intent is triggered when the user input matches the train phrases provided to that intent. This mechanism works well for basic conversations. But to place orders and bargain with users, we need to perform some calculations. This is done via a Fulfillment. Fulfillment is code that’s deployed as a Webhook that lets the dialogflow agent call business logic on an intent-by-intent basis. We have implemented the webhook code on Firebase Cloud Functions. Whenever the dialogflow agent requires to perform a complex task in response to user
request, it triggers the webhook which calculates the appropriate parameters and sends a response. Webhook is the place where the parameters from the dialogflow agent are extracted and formatted for prediction from the ML Engine. It then calls the ML Engine for prediction and passes the response of the model back to the dialogflow agent.

Dialogflow agent has a response time limit for the webhook of 5000 milliseconds. In some instances this time is not enough for the cloud function to finish processing and submit a response, for example updating the database before responding to the dialogflow agent. As a solution to this we have implemented a Google Cloud Pub/Sub topic to update the database. Whenever the agent has to place an order it calls the webhook cloud function. The function calculates the response and publishes a message with the data to be updated in the database to the Pub/Sub topic. Then it sends a response back to the agent. This message published to the topic triggers another cloud function that reads the contents of the message and update the Firestore database with the appropriate data. Thus the response from the bot and the database entry, both are immediately visible to the user of the app.

These all systems work together to provide the user with the experience of bargaining with the chat bot. The user sends an order request to the bot from the android app. This request is matched by the dialogflow agent to an intent and calls the webhook for business logic processing. The machine learning model is called from the webhook and the prediction from the model is passed to the dialogflow agent that in turn sends a natural language response with the received parameters to the android app. This process keeps on repeating until an agreement is reached or the user cancels the ongoing conversation. If an agreement is reached then the agent calls the webhook to update the database with the order details. The webhook pushes a message to pub/sub topic and responds to the agent, which in turn sends a response to the app confirming the order. The message in the pub/sub topic triggers another cloud function which updates the firestore database. This update is reflected in the app within milliseconds due to the real-time nature of the database. Once the app receives the update the repository exposes this data to the viewmodel which instantaneously updates the view of the app.

The following diagram shows the flow of the entire system:

![Diagram showing the flow of the system](image)

3. CONCLUSION

The proposed system works by letting users bargain for their drinks prices and places orders. This system provides a new way of placing orders in a restaurant that is convenient and interesting for the users while providing efficiency and important business data for analytics to the business owners.

The current system being deployed far away for the location of use raises some challenges with latency management. The current latency is represented below:
There is room for further optimizations to get improved results.

4. FUTURE SCOPE

There is a great deal of scope for the improvement of this system. The most major improvement to the system would be to improve upon the machine learning model. More noise free data and a better algorithm is likely to improve the accuracy and performance of the prediction system a great deal.

Another area of improvement could be the latency of the system. This aspect will automatically improve if the system is continuously processing requests. To reduce the latency further we could optimise the cloud functions to take less time to complete processing, also we could deploy all the cloud services to a datacenter location closest to us.

In terms of adding functionality, the easiest addition to the system could be dynamic pricing. The app instantly reacts to the changes in the database. We could implement another cloud function that would update the menu prices based on the current demand at regular intervals.

REFERENCES

[1] https://medium.com/upday-devs/android-architecture-patterns-part-3-model-view-viewmodel-e7eeee76b73b
[2] https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning
[3] https://en.wikipedia.org/wiki/Mean_absolute_error
[4] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/