Data Description

frames.json : A dataset fetched from Microsoft Research that contains 1369 dialogues and 19,986 conversations between a user and a simulated virtual assistant. This dataset works up on a new type of virtual assistant which responds to a person planning a vacation and needs to book a hotel and a flight.

Data Resource Link - <https://www.microsoft.com/en-us/research/project/frames-dataset/>

**What is frames.json?**

Frames is precisely meant to encourage research towards conversational agents which can support decision-making in complex settings, in this case – booking a vacation including flights and a hotel. More than just searching a database, we believe the next generation of conversational agents will need to help users explore a database, compare items, and reach a decision.

The dialogues in Frames were collected in a Wizard-of-Oz fashion. Two humans talked to each other via a chat interface. One was playing the role of the user and the other one was playing the role of the conversational agent. We call the latter a wizard as a reference to the Wizard of Oz, the man behind the curtain. The wizards had access to a database of 250+ packages, each composed of a hotel and round-trip flights. We gave users a few constraints for each dialogue and we asked them to find the best deal. This resulted in complex dialogues where a user would often consider different options, compare packages, and progressively build the description of her ideal trip.

**Dataset Format**

Each dialogue has five main fields: user\_id, wizard\_id, id, userSurveyRating and turns.

These are some of the important fields in the Frames dataset –

**Global Properties**

| Key Name | Example |
| --- | --- |
| id | Refers to a unique identification for the dialogue. |
| user\_id | Refers to a unique identifier for the user taking part in the dialogue. |
| wizard\_id | Refers to a unique identifier for the wizard taking part in the dialogue. |

**Labels**

| Key Name | Example |
| --- | --- |
| userSurveyRating | A value that represents the user’s satisfaction with the Wizard’s service, ranging from 1 – complete dissatisfaction to 5 – complete satisfaction. |
| wizardSurveyTaskSuccessful | A boolean which is true if the wizard thinks at the end of the dialogue that the user’s goal was achieved. |

**Turns**

| Key Name | Example |
| --- | --- |
| author | The author of the message in a dialogue. i.e. “user” or “wizard”. |
| text | The sentence that the author uttered. It is the exact text that the author of a turn said. E.g. “text”: “Consider it done. Have a great trip!”. |
| labels | JSON object which has three keys: active\_frame, acts, and acts\_without\_refs. The active\_frame is the id of the currently active frame. The acts are the dialogue acts for the current utterance. Each act has a name and arguments args. The name is the name of the dialogue act, for instance, offer, or inform. The args contain the slot types (key) and slot values (val), for instance budget=$2000. Slot values are optional. An act contains a ref tag whenever a user or wizard refers to a past frame. The acts\_without\_refs are similar to the acts except that they do not have these ref tags. We define the frame tracking task as the task that takes as input the acts\_without\_refs and outputs the acts. |
| timestamp | Unix timestamp denoting the time at which the current turn occurred. |
| frames | List of frames up to the current turn. Each frame has the following keys: frame\_id, frame\_parent\_id, requests, binary\_questions, compare\_requests, and info. |
| db | It can only occur during a wizard’s turn. It is a list of search queries made by the wizard with the associated list of search results.  E.g. “db”: {“search”: [{“ORIGIN\_CITY”: “Montreal”}], “result”: []} |

**Frames**

| Key Name | Example |
| --- | --- |
| frame\_id | Id of the frame. |
| frame\_parent\_id | Id of the parent frame. |
| requests, binary\_questions, compare\_requests | Requests are questions related to one frame, for instance “what is the price of this package?”. Compare\_requests concern several frames. For example, the user might ask to compare different packages: “What is the guest rating of these two hotels?”. Binary\_questions are questions with both a slot type and a slot value. These are special cases of requests and compare\_requests, for instance “are both hotels 3.5 stars?”. |
| info | The info contains all the constraints set by the user or the wizard in the frame. These constraints are expressed as slot types which have a value. Note that each slot can have multiple values, which accumulate as long as the frame does not change. For example, the price can be both “1000 USD” and “cheapest”. There are two additional fields to keep track of specific aspects of the dialogue:  **REJECTED** a boolean value expressing if the user negated or affirmed an offer made by the wizard.  **MOREINFO** a boolean value expressing whether the user wants to know more about the frame in question |

Review of Algorithms Used

Naïve Bayes classifier : We used Naïve Bayesian classifier as an algorithm to classify the final conversations of each user interaction with the virtual assistant to appreciation and non-appreciation categories. The accuracy for this algorithm results in a very high success percentage which is concluded in the later part of this project.

**What is** Naïve Bayes classifier**?**

A Naive Bayes Classifier is a supervised machine-learning algorithm that uses the Bayes’ Theorem, which assumes that features are statistically independent. The theorem relies on the *naive* assumption that input variables are independent of each other, i.e. there is no way to know anything about other variables when given an additional variable. Regardless of this assumption, it has proven itself to be a classifier with good results.

# **How does Naive Bayes Classifier works?**

Given a vector **x** of features, Naive Bayes calculates the probability that the vector belongs to each class.

P( Ck | x1, x2, … xn )

Thanks to Bayes’ Theorem, we know that it is equal to:

P( Ck | **x**) = (P(**x** | Ck) \*P(Ck) ) / P(**x**)

We know P(Ck) because of the class distribution in our data.

P(**x** | Ck) is equivalent to its joint probability P(Ck ,x1 , x2, …, xn). By the chain rule in probabilities we can expand it to:

P(Ck ,x1 , x2, …, xn) = P(x1 | x2, …, xn, Ck) \* P(x2, …, xn, Ck)

= P(x1 | x2, …, xn, Ck) \* P(x2, | x3,…, xn, Ck) \* P(x3,…, xn, Ck)

= ….

= P(x1 | x2, …, xn, Ck) \* P(x2, | x3,…, xn, Ck) \* P(xn-1,| xn, Ck)\* P(xn | Ck)\* P(Ck)

We now make a strong assumption on the conditional probabilities we just calculated. We assume that they are conditionally independent. In other words, knowing that { xi+1,…, xn } occurred doesn't affect the probability of xi occurring. Put more formally:

P(xi, | xi+1,…, xn, Ck) = P(xi, | Ck).

This means that: P(Ck ,x1 , x2, …, xn) = 

To calculate each of the conditional class probabilities—P(xi | Ck )—we use a likelihood function to model the probability distribution. One of the most common is the Gaussian or normal distribution.

We need not calculate P(**x**) since it’s constant for each input. But you can calculate it in terms of the conditional class probabilities as illustrated below:

P(**x**) =

To make a prediction, we choose the class that had the highest score.

# **Summary**

Advantages of Naive Bayes Classifiers

* Simple model
* Fast
* Scalable
* Requires little data

Disadvantages of Naive Bayes Classifiers

* Assumes feature independence
* Must choose the likelihood function

Analysis and Results