NLP Assignment -3

1) What is text classification? Steps to build text classification and its applications.

Ans|

Text classification is a natural language processing (NLP) task that involves assigning one or more categories to a given piece of text. These categories are predefined, and the goal is to categorize the text into one or more of these categories. For example, in email spam filtering, the categories are "spam" and "non-spam," and the task is to classify each incoming email into one of these categories. Text classification is widely used across various domains such as social media, e-commerce, healthcare, law, and marketing.

Steps to Build a Text Classification System:

1. **Data Collection:**
   - Collect or create a labeled dataset suitable for the task.
2. **Data Splitting:**
   - Split the dataset into training, validation, and test sets, then decide on evaluation metrics.
3. **Text Transformation:**
   - Transform raw text into feature vectors.
4. **Model Training:**
   - Train a classifier using the feature vectors and corresponding labels from the training set.
5. **Model Evaluation:**
   - Benchmark the model's performance on the test set using the chosen evaluation metrics.
6. **Deployment:**
   - Deploy the model to serve the real-world use case and monitor its performance.

Applications of Text Classification:

1. **Content Classification and Organization:**
   - Used for classifying large amounts of textual data, such as news websites, blogs, product reviews, and tweets.
   - Powers content organization, search engines, and recommendation systems.
   - Examples: Tagging product descriptions on e-commerce sites, routing customer service requests, and organizing emails in Gmail.
2. **Customer Support:**
   - Helps identify actionable tweets or social media posts that brands should respond to.
   - Differentiates between actionable content and noise.
3. **E-commerce:**
   - Sentiment analysis is used to understand customer perceptions based on their comments.

- Aspect-based sentiment analysis is a more refined approach, focusing on different facets of a product or service.

4. **Other Applications:**
   - Language identification, such as automatic language detection in Google Translate.
   - Authorship attribution, which identifies the authors of unknown texts.
   - Triaging posts in online support forums for mental health services.
   - Segregating fake news from real news.

2) With code snippet explain the classification modeling using Naïve Bayes classifier.

Ans|

The Naive Bayes classifier is a probabilistic model that applies Bayes' theorem with the assumption of feature independence. It estimates the probability of each class given the features in the text and chooses the class with the highest probability.

Here's a step-by-step explanation of how to implement text classification using the Naive Bayes classifier in Python with the `scikit-learn` library:

Step 1: Train-Test Split

First, you need to load the dataset and split it into training and testing sets. The `train_test_split` function is used for this purpose, where 75% of the data is used for training and 25% for testing by default.

```
from sklearn.model_selection import train_test_split

# Assume our_data is a pandas DataFrame containing the text and relevance columns

X = our_data.text  # the column containing the textual data

y = our_data.relevance  # the column we are learning to predict

# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

Step 2: Pre-process and Vectorize the Text Data

Before feeding the text data into the Naive Bayes model, it needs to be pre-processed and converted into numerical features. Common pre-processing steps include lowercasing, removing punctuation, digits, and stop words. The `CountVectorizer` is used to convert the pre-processed text into a Bag of Words (BoW) model, where each text is represented as a vector of word counts.

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
# Assume clean is a function defined to perform text pre-processing

vect = CountVectorizer(preprocessor=clean)  # Vectorizer with the custom pre-processing function

# Transform the training and testing data into feature vectors

X_train_dtm = vect.fit_transform(X_train)

X_test_dtm = vect.transform(X_test)

# Check the shape of the resulting feature vectors

print(X_train_dtm.shape, X_test_dtm.shape)
```

Step 3: Train the Naive Bayes Classifier

With the feature vectors ready, you can now train the Naive Bayes model using the training data.

```
from sklearn.naive_bayes import MultinomialNB

from sklearn import metrics

# Instantiate a Multinomial Naive Bayes classifier

nb = MultinomialNB()

# Train the model on the training data

nb.fit(X_train_dtm, y_train)

# Make class predictions for the test data

y_pred_class = nb.predict(X_test_dtm)
```

Step 4: Evaluate the Classifier

The classifier's performance can be evaluated using various metrics, such as accuracy, precision, recall, and the confusion matrix.

```
#Evaluate the accuracy of the model

accuracy = metrics.accuracy_score(y_test, y_pred_class)

print("Accuracy: ", accuracy)

# Display the confusion matrix
```

```
conf_matrix = metrics.confusion_matrix(y_test, y_pred_class)

print("Confusion Matrix:\n", conf_matrix)
```

Step 5: Addressing Performance Issues

If the classifier's performance is not satisfactory, you might consider reducing the feature vector size, addressing class imbalance, or trying different algorithms. For example, to reduce feature sparsity, you can limit the number of features used in the `CountVectorizer`.

```
# Reducing the number of features

vect = CountVectorizer(preprocessor=clean, max_features=5000)

# Re-transform the training and testing data

X_train_dtm = vect.fit_transform(X_train)

X_test_dtm = vect.transform(X_test)

# Retrain the Naive Bayes classifier

nb.fit(X_train_dtm, y_train)

# Re-evaluate the model

y_pred_class = nb.predict(X_test_dtm)

print("Accuracy: ", metrics.accuracy_score(y_test, y_pred_class))
```

3) With code snippet explain the classification modeling using Logistic Regression.

Ans|

Logistic Regression is a discriminative classifier that aims to learn the probability distribution over all classes, unlike Naive Bayes, which is a generative classifier. Logistic Regression learns the weights for individual features based on how important they are in making a classification decision. The goal is to learn a linear separator between classes in the training data to maximize the probability of correct classification.

Below is the code snippet that demonstrates how to use Logistic Regression for text classification:

```
from sklearn.linear_model import LogisticRegression
```

```
# Instantiate a Logistic Regression classifier with balanced class weights

logreg = LogisticRegression(class_weight="balanced")

# Train the model using the training data

logreg.fit(X_train_dtm, y_train)

# Make class predictions for the test data

y_pred_class = logreg.predict(X_test_dtm)

# Print the accuracy of the classifier

print("Accuracy: ", metrics.accuracy_score(y_test, y_pred_class))
```

Explanation:

- class_weight="balanced": This parameter boosts the weights for classes in inverse proportion to the number of samples for that class, which helps in handling class imbalance.

- Accuracy: The classifier in this example achieves an accuracy of 73.7%.

This approach is useful for cases where class imbalance might affect the performance of the classifier. However, in this specific dataset, Logistic Regression seems to perform worse than Naive Bayes.

4) With code snippet explain the classification modeling using SVM.

Ans|

Support Vector Machine (SVM) is a discriminative classifier like logistic regression, but it aims to find an optimal hyperplane that separates classes in a higher-dimensional space with the maximum possible margin. SVMs are capable of learning non-linear separations between classes and are particularly effective in high-dimensional spaces.

Below is the code snippet that demonstrates how to use SVM for text classification:

```
from sklearn.svm import LinearSVC

from sklearn.feature_extraction.text import CountVectorizer

from sklearn import metrics
```

Step 1: Vectorize the text data with a maximum of 1,000 features

```
vect = CountVectorizer(preprocessor=clean, max_features=1000)
```

X_train_dtm = vect.fit_transform(X_train) # Combine vectorization and transformation

X_test_dtm = vect.transform(X_test)

Step 2: Instantiate the SVM classifier with balanced class weights

classifier = LinearSVC(class_weight='balanced')

Step 3: Train the SVM classifier on the training data

classifier.fit(X_train_dtm, y_train)

Step 4: Make predictions on the test data

y_pred_class = classifier.predict(X_test_dtm)

Step 5: Print the accuracy of the classifier

print("Accuracy: ", metrics.accuracy_score(y_test, y_pred_class))

Explanation:

- max_features=1000: The vectorizer restricts the number of features to 1,000 to manage the training time, given that SVMs can be computationally intensive.

- class_weight='balanced': This option adjusts the weights inversely proportional to class frequencies in the input data, helping to handle any class imbalance.

The SVM classifier in this example has been shown to perform better in certain categories compared to logistic regression, especially in separating classes with a maximum margin.

5) With code snippet explain the classification modeling using CNN.

Ans|

To define, train, and evaluate a Convolutional Neural Network (CNN) model for text classification using Keras, follow these steps:

1. Define the CNN Architecture:

- Create a `Sequential` model.

- Add an embedding layer.

- Add several `Conv1D` layers for convolution operations.

- Add `MaxPooling1D` layers for pooling operations.

- Use `GlobalMaxPooling1D` to summarize features.

- Add `Dense` layers for fully connected operations.

- Compile the model with a loss function, optimizer, and evaluation metrics.

2. Train and Evaluate the Model:

- Fit the model on training data and validate using validation data.

- Evaluate the model on test data.

```
from keras.models import Sequential

from keras.layers import Conv1D, MaxPooling1D, GlobalMaxPooling1D, Dense, Embedding

print('Define a 1D CNN model.')

cnnmodel = Sequential()

cnnmodel.add(embedding_layer)  # Add pre-trained or trainable embedding layer

cnnmodel.add(Conv1D(128, 5, activation='relu'))

cnnmodel.add(MaxPooling1D(5))

cnnmodel.add(Conv1D(128, 5, activation='relu'))

cnnmodel.add(MaxPooling1D(5))

cnnmodel.add(Conv1D(128, 5, activation='relu'))

cnnmodel.add(GlobalMaxPooling1D())

cnnmodel.add(Dense(128, activation='relu'))

cnnmodel.add(Dense(len(labels_index), activation='softmax'))

cnnmodel.compile(loss='categorical_crossentropy',

 optimizer='rmsprop',

 metrics=['acc'])


cnnmodel.fit(x_train, y_train,

 batch_size=128,
```

```
 epochs=1, validation_data=(x_val, y_val))

score, acc = cnnmodel.evaluate(test_data, test_labels)

print('Test accuracy with CNN:', acc)
```

Alternative with Trainable Embedding Layer:

If you want to train the embedding layer from scratch rather than using pre-trained embeddings:

```
print("Defining and training a CNN model, training embedding layer on the fly instead of using pre-trained embeddings")

cnnmodel = Sequential()

cnnmodel.add(Embedding(MAX_NUM_WORDS, 128))  # Define new embedding layer

# Add convolutional and pooling layers as before

cnnmodel.fit(x_train, y_train,

 batch_size=128,

 epochs=1, validation_data=(x_val, y_val))

score, acc = cnnmodel.evaluate(test_data, test_labels)

print('Test accuracy with CNN:', acc)
```

6) With code snippet explain the classification modeling using LSTM.

Ans|

To define, train, and evaluate an LSTM model for text classification using Keras, follow these steps:

Train and Evaluate the Model:

- Fit the model on training data and validate using validation data.
- Evaluate the model on test data.

```
from keras.models import Sequential

from keras.layers import LSTM, Dense, Embedding

print("Defining and training an LSTM model, training embedding layer on the fly")
```

```
rnnmodel = Sequential()
rnnmodel.add(Embedding(MAX_NUM_WORDS, 128))  # Add embedding layer
rnnmodel.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))  # Add LSTM layer
rnnmodel.add(Dense(2, activation='sigmoid'))  # Add dense layer with sigmoid activation
rnnmodel.compile(loss='binary_crossentropy',
 optimizer='adam',
 metrics=['accuracy'])

print('Training the RNN')
rnnmodel.fit(x_train, y_train,
 batch_size=32,
 epochs=1,
 validation_data=(x_val, y_val))
score, acc = rnnmodel.evaluate(test_data, test_labels,
 batch_size=32)
print('Test accuracy with RNN:', acc)
```

In this example, the model uses:

- An `Embedding` layer to transform words into vectors.

- An `LSTM` layer to capture sequential dependencies in the text.

- A `Dense` layer with a `sigmoid` activation function to output class probabilities.

7) Explain the case study on Corporate Ticketing using block diagram.

Ans|

Case Study: Corporate Ticketing

Pipeline for Building a Classifier with No Training Data

1. **Start with No Labeled Data**

- **Initial Options**:
  - **Public APIs/Libraries**: Use APIs like Google's, which classify content into numerous categories. Map these categories to relevant ones for your system.
  - **Public Datasets**: Use datasets such as 20 Newsgroups to train a basic classifier. For instance, classify all topics into one category and medical-related topics into another.
  - **Weak Supervision**: Create a small labeled dataset using rules based on keywords (e.g., fever, headache) to identify medical tickets.

2. **Deploy Initial Model**
   - **Production**: Implement the baseline model derived from public APIs, datasets, or weak supervision.
3. **Collect Feedback**
   - **Explicit Feedback**: Direct feedback from users (e.g., medical counsel or hospital) on the relevance of tickets.
   - **Implicit Feedback**: Data from ticket response times and rates that indicate model performance.
4. **Refine Model**
   - **Active Learning**: Use feedback to select instances that need labeling and refine the model based on this data.
5. **Iterate and Improve**
   - **Ongoing Data Collection**: Continue to collect more labeled data and refine the model.
   - **Build Sophisticated Models**: As more data is collected, develop more advanced models.

Block Diagram

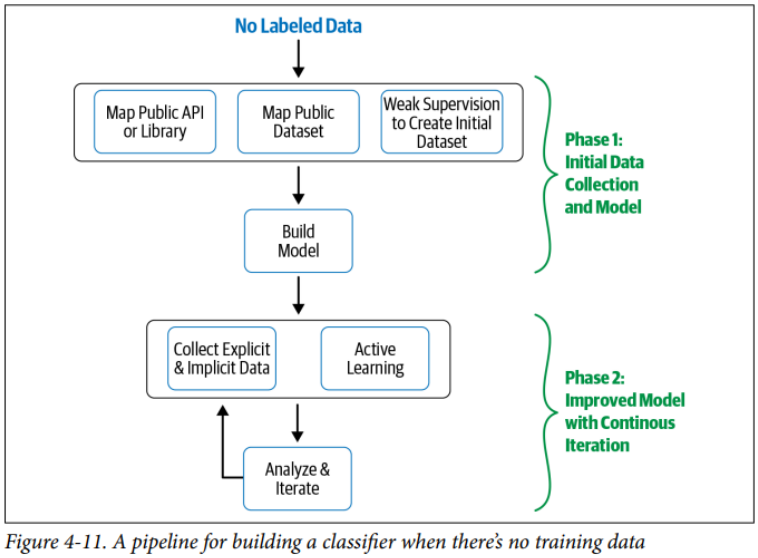


Figure 4-11. A pipeline for building a classifier when there's no training data

8) What are chat bots? What are it's benefits and application.

Ans|

Chatbots are interactive systems that allow users to interact in natural language, typically via text but also using speech interfaces. They enable users to perform various actions and get responses within messaging platforms or other interfaces. The evolution of chatbots has led to conversational interfaces that use voice commands instead of screens or mice.

Benefits of Chatbots

-Natural Interaction: Users can interact using natural language, making the experience more intuitive and user-friendly.

-Convenience: Chatbots provide a conversational interface that simplifies interactions without requiring traditional input devices.

- Accessibility: They are available on various platforms, allowing users to interact seamlessly from their preferred messaging apps.

Applications of Chatbots

1. Shopping and E-commerce:

- Place or modify orders
- Handle payments
- Recommend items and provide a conversational recommendation system

2. News and Content Discovery:

- Help users find relevant articles and content based on conversational queries

3. Customer Service:

- Lodge complaints
- Answer frequently asked questions (FAQs)
- Navigate queries through predefined conversational flows

4. Medical:

- Provide information based on symptoms
- Elicit useful health information from patients, especially older patients

5. Legal:

- Serve FAQs

- Ask follow-up questions to provide more specific legal information.

9) Explain in detail the taxonomy of chatbot.

Ans|

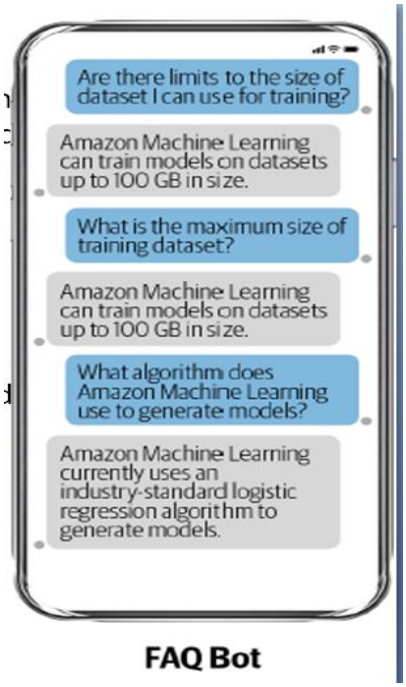
Chatbots can be classified based on how they interact with users and their intended functionality. This classification affects their design and use in various domains. The taxonomy includes the following types:

1. Exact Answer or FAQ Bot

- **Characteristics:** This type of chatbot provides fixed responses based on predefined answers. It retrieves a specific, correct response according to the user's query.

- **Functionality:** It is linked to a set of responses and does not depend on previous interactions. For each query, the bot selects an appropriate answer from its pre-existing set of responses.

- **Example:**



FAQ Bot

2. Flow-Based Bot

- **Characteristics:** These chatbots are more complex than FAQ bots as they handle conversations with varying responses based on user input over time.

- **Functionality:** The bot follows a predefined flow of conversation, asking specific questions to collect information and achieve a goal. It tracks user inputs throughout the interaction to provide relevant responses.

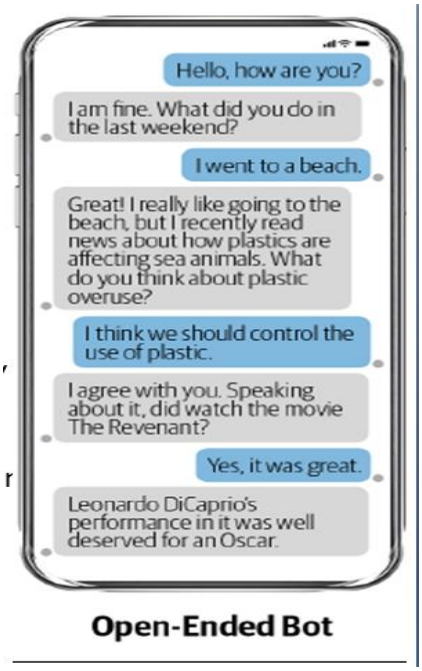
- **Example:**

Flow-Based Bot

3. Open-Ended Bot

- Open-ended bots are intended mainly for entertainment, where the bot is supposed to converse with the user about various topics.
- The bot doesn't have to maintain specific directions or flows of the conversation.
- This bot carries out a conversation without any pre-existing template or fixed question-answer pairs.
- It transitions fluently from one topic to another to maintain the interesting conversation.

Example:



10) Discuss and explain the types of chat bot based of interaction with the user and write the comparison between them.

Ans|

Chatbots can be classified based on their interaction with users into two main types: **goal-oriented** and **chitchat bots**. Here's a detailed comparison of these types:

1. Goal-Oriented Chatbots

Definition: Goal-oriented chatbots are designed to achieve specific objectives or assist with particular tasks. They guide the conversation toward fulfilling a predefined goal.

Features:

- Structured Interaction: Conversations are structured and revolve around specific tasks or objectives.

- Dialog Manager: Uses a dialog manager to control and guide the conversation flow based on predefined intents and entities.

- Natural Language Understanding (NLU): Analyzes user input to identify intents and extract entities.

- Task Execution: Aims to complete specific actions, such as booking a ticket, ordering food, or providing information.

- Response Generation: Generates responses based on the intent and context, often using template-based or data-driven approaches.

Applications:

- Customer Service: Handling inquiries and processing requests.

- E-Commerce: Assisting with product orders and transactions.

- Scheduling: Managing appointments and reminders.

Example: A chatbot designed to help users order a pizza, where it understands commands like "order pizza" and processes specific details such as size and toppings.

2. Chitchat Bots

Definition: Chitchat bots engage in open-domain, unstructured conversations without specific goals. They aim to provide natural and engaging interactions.

Features:

- **Unstructured Interaction:** Conversations can be free-form and cover a wide range of topics.

- Natural Language Generation (NLG): Focuses on generating coherent, on-topic, and contextually appropriate responses.

- Emotional Engagement: Designed to handle various topics and maintain a conversational flow that feels natural and engaging.

- Personalization: May involve understanding user sentiment and context to create a more personalized interaction.

Applications:

- Companionship: Addressing loneliness and providing emotional support.

- Entertainment: Engaging users in casual conversations or games.

- Customer Interaction: Offering a more engaging and human-like interaction experience.

Example: A chatbot designed to engage users in casual conversations about various topics like hobbies or current events, similar to a virtual friend or conversational partner.

Comparison

Feature	Goal-Oriented Chatbots	Chitchat Bots
Purpose	Achieve specific tasks or goals	Engage in open-domain, casual conversation
Interaction Type	Structured and task-focused	Unstructured and free-form
Dialog Management	Uses a dialog manager to guide conversation	Focuses on generating natural, engaging responses
Natural Language Processing	Focuses on intent and entity extraction	Focuses on generating coherent and relevant responses
Applications	Customer service, e-commerce, scheduling	Companionship, entertainment, casual engagement

11) Explain in detail the pipeline for building dialogue systems and components of dialog system.

Ans|

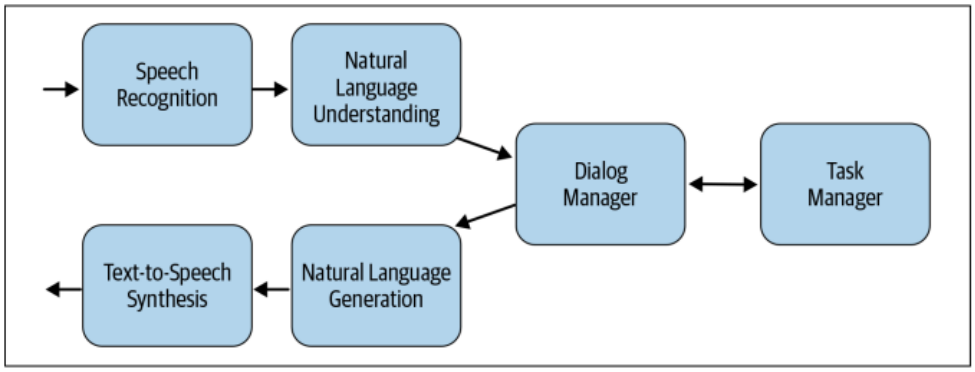


Figure 6-3. Pipeline for a dialog system

Pipeline for Building Dialog Systems

1. Speech Recognition

- Function: Converts human speech into text.

- Details: Transcribes spoken language into written text. Advanced speech-to-text models are used for high accuracy in industrial applications.

2. Natural Language Understanding (NLU)

- Function: Analyzes and understands the transcribed text.

- Details: Handles tasks such as sentiment detection, named entity extraction, and coreference resolution to extract useful information from the text.

3. Dialog and Task Manager

- Function: Manages and guides the conversation flow based on the extracted information.

- Details: Organizes information from NLU and uses rules or complex mechanisms like reinforcement learning to manage goal-oriented conversations and achieve specific objectives.

4. Natural Language Generation (NLG)

- Function: Generates human-readable responses based on the dialog manager's strategy.

- Details: Produces text responses which can be either template-based or generated using models.

5. Speech Synthesis

- Function: Converts the generated text response back into spoken language.

- Details: Transforms the text response into speech, allowing voice-based systems to communicate with the user. This step is essential for voice-based dialog systems and ensures that the responses are delivered audibly to the user.

Components of a Dialog System

1. Dialog Act or Intent

- Definition: The purpose behind a user command.

- Examples: In a pizza ordering scenario, intents might be "orderPizza" or "getStockQuote."

2. Slot or Entity

- Definition: Specific information related to the intent.

- Examples: For a pizza order, entities might include "medium" (size) and "extra cheese" (topping).

3. Dialog State or Context

- Definition: The current status of the conversation, including dialog acts and state-value pairs.

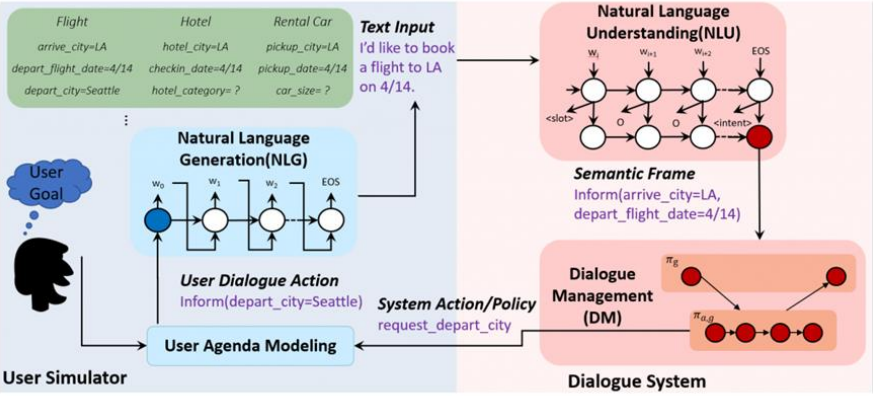
- Details: Captures the ongoing conversation state and historical information from previous exchanges.

12) Explain in detail the end-to-end approach.

Ans|

The end-to-end approach in building dialog systems utilizes sequence-to-sequence (seq2seq) models to handle the entire dialog generation process. Here's a detailed explanation based on the provided information:

1. Seq2Seq Models



- **Function:** Seq2seq models are designed to take a sequence of words as input and generate another sequence of words as output. In the context of dialog systems, the input is the user's utterance (a sequence of words), and the output is the bot's response (another sequence of words).

- **Characteristics:** These models are end-to-end trainable, meaning they can be trained as a single unified model without the need for multiple modules. They typically use Long Short-Term Memory (LSTM) networks, but recent advancements have also incorporated transformers, which are state-of-the-art for seq2seq tasks.

- **Tokenization:** To process the input, seq2seq models use tokenization to create word tokens and sequences. This is crucial for capturing the order of tokens in the sequence, which helps in understanding the meaning of the input and generating appropriate responses.

2. Deep Reinforcement Learning for Dialogue Generation

- **Challenge:** Traditional seq2seq models may produce generic or repetitive responses, such as "I don't know," which do not contribute to engaging or goal-oriented conversations.

- **Solution:** Deep reinforcement learning can enhance dialog systems by generating diverse and contextually appropriate responses. This approach involves training the model to optimize responses based on rewards or penalties, which are determined by how well the responses achieve the conversation's goal.

- **Mechanism:** The model performs actions (generates responses) and learns from feedback based on the quality of the conversation. Reinforcement learning helps in exploring and exploiting various responses to improve the overall dialog quality, as shown in comparisons where reinforcement learning-based models generate more diverse responses compared to traditional seq2seq models.

3. Human-in-the-Loop

- **Concept:** Human-in-the-loop involves incorporating human feedback into the training process of dialog systems. This feedback can help correct or guide the model when it encounters queries beyond its scope or generates incorrect responses.

- **Application:** In reinforcement learning setups, human "teachers" provide additional signals or partial rewards to the model. This feedback improves the model's learning process and response quality, making the system more practical and reliable for real-world applications.

- **Advantages:** Combining end-to-end models with human-in-the-loop techniques creates a hybrid system that leverages the efficiency of automated training while ensuring higher accuracy and relevance of responses through human oversight.

13) Explain how Deep Reinforcement Learning techniques are used for Dialogue generation.

Ans|

Deep Reinforcement Learning techniques are employed in dialogue generation to address the limitations of traditional seq2seq models, which often produce generic responses such as "I don't know." Here's how these techniques are used:

1. Challenge with Seq2Seq Models:

- Seq2seq models might generate repetitive or unhelpful responses because they lack the capability to evaluate the quality of the conversation in a way that ensures a good user experience.

2. Role of Reinforcement Learning:

- Reinforcement learning helps improve dialogue generation by evaluating the effectiveness of responses based on a reward system. This involves learning how to generate the best possible responses to achieve specific goals in the conversation.

- Each response generated by the machine is considered an action in reinforcement learning. The model learns to generate responses that maximize a reward defined by how well the response helps achieve the final goal of the conversation.

3. Optimization Through Rewards:

- The model uses rewards to guide its learning process. These rewards are based on the quality and appropriateness of the responses in achieving the conversation's objective. For goal-oriented dialogs, the reward is linked to how effectively the conversation meets the defined goal. In chitchat scenarios, the reward is related to how engaging or interesting the conversation is.

- By exploring and exploiting various response strategies, the reinforcement learning model can produce more diverse and contextually appropriate responses, moving beyond generic outputs.

4. Outcome:

- Reinforcement learning-based models tend to generate more varied and relevant responses compared to traditional seq2seq models. This improvement is due to the model’s ability to evaluate responses based on futuristic rewards and adapt its behavior to better meet the goals of the conversation.