Dialogue Management & NLU

NLPP: Chapter 5, 6
AIMA: Chapter 22 AND Jurafsky & Martin
Chapter 24, Available in the Library.
Remember NLPP book??

Conversational Dialogue Systems: Challenges

- **Language understanding**: methods are expensive, data intensive
- **Dialogue manager**: design is currently an art, domain specific, labor intensive
- **Language generation**: canned utterances or template-based, domain specific, labor intensive
- **Active Research**: Methods for quickly customizing systems to new domains, users, applications
Text-Based Dialogue System

- **TEAMS:**
  - NLU: 1 person + crowd sourcing or whole team for corpus/testing?
  - NLG: 1 person
  - Task/DB: ½ person unless significant extension of DB or task model targeting DB
  - DM: 1 to 2 people
Outline

- The Linguistics of Conversation
- Basic Conversational Agents
  - ASR
  - NLU
  - Generation
  - Dialogue Manager
- Dialogue Manager Design
  - Finite State
  - Frame-based
  - Initiative: User, System, Mixed
- Information-State
  - Dialogue-Act Detection
  - Dialogue-Act Generation
- Evaluation
- Utility-based conversational agents
  - MDP, POMDP
Natural Language Understanding (last week)

- Or “NLU”
- Or “Computational semantics”
- There are many ways to represent the meaning of sentences
- For dialogue systems, most common is “Frame and slot semantics”.
An example of a frame

- Show me morning flights from Boston to SF on Tuesday.

SHOW:

FLIGHTS:

ORIGIN:

  CITY: Boston

  DATE: Tuesday

  TIME: morning

DEST:

  CITY: San Francisco

HOW TO GENERATE THIS SEMANTICS?
- Different Methods (see NLTK, chaps 6,7,8)
Semantic Parsing approach

- Context Free Grammar = CFG
- Define a CFG with Semantic nonterminals and word sequence or semantic terminals

CFG in which the LHS of rules is a semantic category:
- LIST -> show me | I want | can I see|...
- DEPARTTIME -> (after | around | before) HOUR | morning | afternoon | evening
- HOUR -> one | two | three…| twelve (am | pm)
- FLIGHTS -> (a) flight| flights
- ORIGIN -> from CITY
- DESTINATION -> to CITY
- CITY -> Boston | San Francisco | Denver | Washington
Thought/Discussion Question

- WHAT ARE THE PROS AND CONS OF THE SEMANTIC PARSING APPROACH?

- Everything written by hand
- Hard to get good coverage?
- Write the same rules repeatedly for many applications?

- HOUR -> one | two | three…| twelve (am | pm)
- CITY -> Boston | San Francisco | Denver | Washington
Other Methods for NLU

- **Syntactic Parsing approach (you can look it up)**
  - Parse it into a syntactic structure using an NL Grammar;
  - NLTK has a chart parser built in
    - `nltk.parse.earleychart`: Data classes and parser implementations for incremental chart parsers, which use dynamic programming to efficiently parse a text.
  - Or go look at Stanford Parser
  - For each rule of the grammar, define a ‘matching rule’ to convert the parse representation into semantics/meaning
NLU Metric: Concept Accuracy

- “Concept accuracy” or “Concept error rate”
- % of semantic concepts that the NLU component returns correctly
- I want to arrive in Austin at 5:00
  - DESTCITY: Boston
  - Time: 5:00
- Concept accuracy = 50%
- Average this across entire dialogue
- “How many of the sentences did the system understand correctly”
Chunking, Text Classification
Tools & Modules in NLTK
Chunking

- **nltk.chunk**: Classes and interfaces for identifying non-overlapping linguistic groups (such as base noun phrases) in unrestricted text. [nltk.chunk.api](https://www.nltk.org/api/nltk.chunk.html)
- **nltk.chunk.regexp**
- **nltk.chunk.named_entity**: Named entity chunker
- **nltk.chunk.util**
What is Chunking?

- Goal: divide a user utterance into a sequence of chunks.
- Noun-phrase chunking:
  \[[l] \text{saw} \ [a \text{ tall man}] \text{ in} \ [\text{the park}]\].
- Verb-phrase chunking:
  The man who \[\text{was in the park}\] \[\text{saw me}\].
- Question answering:
  - What \[\text{Spanish explorer}\] discovered \[\text{the Mississippi River}\]?
### Chunking

- TAG the words in input using a general purpose tagger
  - **nltk.tag**: Classes and interfaces for tagging each token of a sentence with supplementary information, such as its part of speech.

- Define a regular expression that matches the sequences of tags in a chunk
  - A simple noun phrase chunk regexp:
    - (Note that `<NN.*>` matches any tag starting with NN)
    - `<DT>? <JJ>* <NN.?>`

- Chunk all matching subsequences:
  - `the/DT little/JJ cat/NN sat/VBD on/IN the/DT mat/NN`
  - `[the/DT little/JJ cat/NN] sat/VBD on/IN [the/DT mat/NN]`

- If matching subsequences overlap, first 1 gets priority
Parsing a Complex Noun Phrase

“head” = central, most critical part of the NP

“stuff that comes before”

“stuff that comes after”
Determiners = DT tag from nltk.tag

- Noun phrases can start with determiners...
- Determiners can be
  - Simple lexical items: the, this, a, an, etc. (e.g., “a car”)
  - Or simple possessives (e.g., “John’s car”)
  - Or complex recursive versions thereof (e.g., John’s sister’s husband’s son’s car)
Premodifiers = JJ tag in nltk.tag

- Come before the head
- Examples:
  - Cardinals, ordinals, etc. (e.g., “three cars”)
  - Adjectives (e.g., “large car”)
- Ordering constraints
  - “three large cars” vs. “?large three cars”
Postmodifiers: Not in the Chunk

- Naturally, come after the head
- Three kinds
  - Prepositional phrases (e.g., “from Seattle”)
  - Non-finite clauses (e.g., “arriving before noon”)
  - Relative clauses (e.g., “that serve breakfast”)
- Similar recursive rules to handle these
  - Nominal → Nominal PP
  - Nominal → Nominal GerundVP
  - Nominal → Nominal RelClause
Chunk (shallow) Parsing

- Chunks are non-overlapping regions of a text
  
  \[ I \] saw \[ a \ tall \ man \] in \[ the \ park \].

- Chunks are non-recursive: cannot contain other chunks

- Chunks are non-exhaustive: Some words not in chunks

- Chunking is easier than parsing
  - Less word-order flexibility within chunks than between chunks
  - More locality: Less context-dependence, Less ambiguity

- Chunks can be used as inputs to dialog act or task classifiers or task-specific routines
Chunking

- TAG the words in input using a general purpose tagger
- **nltk.tag**: Classes and interfaces for tagging each token of a sentence with supplementary information, such as its part of speech.

- Define a regular expression that matches the sequences of tags in a chunk
  A simple noun phrase chunk regexp:
  (Note that `<NN.*>` matches any tag starting with NN)
  `<DT>? <JJ>* <NN.?>

- Chunk all matching subsequences:
  ```
  the/DT little/JJ cat/NN sat/VBD on/IN the/DT mat/NN
  [the/DT little/JJ cat/NN] sat/VBD on/IN [the/DT mat/NN]
  ```

- If matching subsequences overlap, first 1 gets priority
Chunk grammar

- An NP chunk should be formed whenever the chunker finds an optional determiner (DT) followed by any number of adjectives (JJ) and then a noun (NN):
  >>> grammar = "NP: {<DT>?<JJ>*<NN>}"
RegexpParser

- RegexpParser chunker begins with a flat structure in which no tokens are chunked.
- The chunking rules are applied in turn, successively updating the chunk structure.
- Once all of the rules have been invoked, the resulting chunk structure is returned.
>>> sentence = [("the", "DT"), ("little", "JJ"),
  ("yellow", "JJ"),
... ("dog", "NN"), ("barked", "VBD"), ("at", "IN"),
  ("the", "DT"), ("cat", "NN"))]
>>> grammar = "NP: {<DT>?<JJ>*<NN>}" 
>>> cp = nltk.RegexpParser(grammar) 
>>> result = cp.parse(sentence) 
>>> print result
  (S
  (NP the/DT little/JJ yellow/JJ dog/NN)
  barked/VBD
  at/IN
  (NP the/DT cat/NN)
Representation

- IOB: Begin Inside Outside

![IOB representation diagram]

![Trees representation diagram]
Chinking

- Sometimes it is easier to define what we want to exclude from a chunk.
- We can define a **chink** to be a sequence of tokens that is not included in a chunk. In the following example, barked/VBD at/IN is a chink:
  - [ the/DT little/JJ yellow/JJ dog/NN ] barked/VBD at/IN
  - [ the/DT cat/NN ]
More chinking

- Possibilities:
  - If the matching sequence of tokens spans an entire chunk, then the whole chunk is removed;
  - if the sequence of tokens appears in the middle of the chunk, these tokens are removed, leaving two chunks where there was only one before.
  - If the sequence is at the periphery of the chunk, these tokens are removed, and a smaller chunk remains.
Meaning & Structure

- Informal Language & Speech
  - Often want to get meaning without parsing
- Tagging `nltk.tag`
- Chunking & Chinking
- Classification:
  - Façade Example
  - NLTK example
  - [http://www.youtube.com/watch?v=keXW_5-IID0](http://www.youtube.com/watch?v=keXW_5-IID0) has an NLTK classifier demo in the talk as well as other modules
Dialogue Manager

- Controls the architecture and structure of dialogue
  - Takes input from NLU component
  -Maintains some sort of state
  -Interfaces with Task Manager
  -Passes output to NLG modules
- Decides what the system is going to do next
- Key aspect of dialogue system design
- How to design it?
Four architectures for dialogue management

- Finite State
- Frame-based
- Information State
  - Markov Decision Processes
- AI Planning
Finite-State Dialogue Mgmt

- Consider a trivial airline travel system
  - Ask the user for a departure city
  - For a destination city
  - For a time
  - Whether the trip is round-trip or not
Finite State Dialogue Manager

1. What city are you leaving from?
2. Where are you going?
3. What date do you want to leave?
4. Is it a one-way trip?
   - Yes: Do you want to go from <FROM> to <TO> on <DATE>?
   - No: What date do you want to return?

5. If one-way: Book the flight
6. If return: Do you want to go from <FROM> to <TO> on <DATE> returning on <RETURN>?
Finite-state dialogue managers

- System completely controls the conversation with the user.
- It asks the user a series of questions
- Ignoring (or misinterpreting) anything the user says that is not a direct answer to the system’s questions
Dialogue Initiative

- Systems that control conversation like this are **system initiative** or **single initiative**.
- “Initiative”: who has control of conversation
- In normal human-human dialogue, initiative shifts back and forth between participants.
System Initiative

- Systems which completely control the conversation at all times are called system initiative.

- Advantages:
  - Simple to build
  - User always knows what they can say next
  - System always knows what user can say next
    - Known words and Topic: Better performance from NLU
  - Ok for VERY simple tasks (entering a credit card, or login name and password)

- Disadvantage:
  - Too limited
User Initiative

- User directs the system
- Generally, user asks a single question, system answers
- System can’t ask questions back, engage in clarification dialogue, confirmation dialogue
- Used for simple database queries
- User asks question, system gives answer
- Web search is user initiative dialogue.
Problems with System Initiative

- Real dialogue involves give and take!
- In travel planning, users might want to say something that is not the direct answer to the question.
- For example answering more than one question in a sentence:
  - Hi, I’d like to fly from Seattle Tuesday morning
  - I want a flight from Milwaukee to Orlando one way leaving after 5 p.m. on Wednesday.
Single initiative + universals

- We can give users a little more flexibility by adding universal commands
- Universals: commands you can say anywhere
- As if we augmented every state of FSA with these
  - Help
  - Start over
  - Correct
- This describes many implemented systems
- But still doesn’t allow user to say what s/he wants to say
Mixed Initiative

- Conversational initiative can shift between system and user
- Simplest kind of mixed initiative: use the structure of the frame itself to guide dialogue

<table>
<thead>
<tr>
<th>Slot</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORIGIN</td>
<td>What city are you leaving from?</td>
</tr>
<tr>
<td>DEST</td>
<td>Where are you going?</td>
</tr>
<tr>
<td>DEPT DATE</td>
<td>What day would you like to leave?</td>
</tr>
<tr>
<td>DEPT TIME</td>
<td>What time would you like to leave?</td>
</tr>
<tr>
<td>AIRLINE</td>
<td>What is your preferred airline?</td>
</tr>
</tbody>
</table>
Frames are mixed-initiative

- User can answer multiple questions at once.
- System asks questions of user, filling any slots that user specifies.
- When frame is filled, do database query.
- If user answers 3 questions at once, system has to fill slots and not ask these questions again.
- Avoid the strict constraints on order of the finite-state architecture.
Multiple frames

- flights, hotels, rental cars
- Flight legs: Each flight can have multiple legs, which might need to be discussed separately
- Presenting the flights (If there are multiple flights meeting users constraints)
  - It has slots like 1ST_FLIGHT or 2ND_FLIGHT so user can ask “how much is the second one”
- General route information:
  - Which airlines fly from Boston to San Francisco
- Airfare practices:
  - Do I have to stay over Saturday to get a decent airfare?
Multiple Frames

- Need to be able to switch from frame to frame
- Based on what user says.
- Disambiguate which slot of which frame an input is supposed to fill, then switch dialogue control to that frame.
- Main implementation: production rules
  - Different types of inputs cause different productions to fire
  - Each of which can flexibly fill in different frames
  - Can also switch control to different frame
Defining Mixed Initiative

- Mixed Initiative could mean
  - User can arbitrarily take or give up initiative in various ways
    - This is really only possible in very complex plan-based dialogue systems
    - No commercial implementations
    - Important research area
  - Something simpler and quite specific which we will define in the next few slides
True Mixed Initiative

C₁: ...I need to travel in May.
A₁: And, what day in May did you want to travel?
C₂: OK uh I need to be there for a meeting that’s from the 12th to the 15th.
A₂: And you’re flying into what city?
C₃: Seattle.
A₃: And what time would you like to leave Pittsburgh?
C₄: Uh hmm I don’t think there’s many options for non-stop.
A₄: Right. There’s three non-stops today.
C₅: What are they?
A₅: The first one departs PGH at 10:00am arrives Seattle at 12:05 their time. The second flight departs PGH at 5:55pm, arrives Seattle at 8pm. And the last flight departs PGH at 8:15pm arrives Seattle at 10:28pm.
C₆: OK I’ll take the 5ish flight on the night before on the 11th.
C₇: OK.
How mixed initiative is usually defined

- First we need to define two other factors
- Open prompts vs. directive prompts
- Restrictive versus non-restrictive grammar
Open vs. Directive Prompts

- Open prompt
  - System gives user very few constraints
  - User can respond how they please:
    - “How may I help you?” “How may I direct your call?”

- Directive prompt
  - Explicit instructs user how to respond
  - “Say yes if you accept the call; otherwise, say no”
Restrictive vs. Non-restrictive grammars

- Restrictive grammar
  - Language model which strongly constrains the NLU system, based on dialogue state

- Non-restrictive grammar
  - Open language model which is not restricted to a particular dialogue state
## Definition of Mixed Initiative

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Open Prompt</th>
<th>Directive Prompt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restrictive</td>
<td>* Doesn’t make sense*</td>
<td>* System Initiative*</td>
</tr>
<tr>
<td>Non-restrictive</td>
<td>* User Initiative*</td>
<td>* Mixed Initiative*</td>
</tr>
</tbody>
</table>
NJFun

- S: Welcome to NJFun, How May I Help You? (User Initiative)
- S': Welcome to NJFun, Please say an activity name or say list activities to get a list of activities I know about. (System Initiative)
- U: I’d like to find some winetasting in Lambertville this morning.
- S: Did you say you are interested in Lambertville? (Explicit Confirmation)
- U: Yes.
- S: Did you say you want to go in the morning?
- U: Yes
- S: I found a winery in Lambertville that is open in the morning. It is ....
Information-State and Dialogue Acts

- If we want a dialogue system to be more than just form-filling
- Needs to:
  - Decide when the user has asked a question, made a proposal, rejected a suggestion
  - Ground a user’s utterance, ask clarification questions, suggestion plans
- Suggests:
  - Conversational agent needs sophisticated models of interpretation and generation
    - In terms of speech acts and grounding
    - Needs more sophisticated representation of dialogue context than just a list of slots
Information-state architecture

- Information state
- Dialogue act interpreter
- Dialogue act generator
- Set of update rules
  - Update dialogue state as acts are interpreted
  - Generate dialogue acts
- Control structure to select which update rules to apply
Information-state

![Diagram of Natural Language Understanding, Dialogue Act Interpreter, Information State, Natural Language Generation, and Dialogue Act Generator]
Dialogue acts

- Also called “conversational moves”
- An act with (internal) structure related specifically to its dialogue function
- Incorporates ideas of grounding
- Incorporates other dialogue and conversational functions that Austin and Searle didn’t seem interested in
Verbmobil task

- Two-party scheduling dialogues
- Speakers were asked to plan a meeting at some future date
- Data used to design conversational agents which would help with this task
- (cross-language, translating, scheduling assistant)
<table>
<thead>
<tr>
<th>Act Type</th>
<th>Natural Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>THANK</td>
<td>thanks</td>
</tr>
<tr>
<td>GREET</td>
<td>Hello Dan</td>
</tr>
<tr>
<td>INTRODUCE</td>
<td>It’s me again</td>
</tr>
<tr>
<td>BYE</td>
<td>Alright, bye</td>
</tr>
<tr>
<td>REQUEST-COMMENT</td>
<td>How does that look?</td>
</tr>
<tr>
<td>SUGGEST</td>
<td>June 13th through 17th</td>
</tr>
<tr>
<td>REJECT</td>
<td>No, Friday I’m booked all day</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>Saturday sounds fine</td>
</tr>
<tr>
<td>REQUEST-SUGGEST</td>
<td>What is a good day of the week for you?</td>
</tr>
<tr>
<td>INIT</td>
<td>I wanted to make an appointment with you</td>
</tr>
<tr>
<td>GIVE_REASON</td>
<td>Because I have meetings all afternoon</td>
</tr>
<tr>
<td>FEEDBACK</td>
<td>Okay</td>
</tr>
<tr>
<td>DELIBERATE</td>
<td>Let me check my calendar here</td>
</tr>
<tr>
<td>CONFIRM</td>
<td>Okay, that would be wonderful</td>
</tr>
<tr>
<td>CLARIFY</td>
<td>Okay, do you mean Tuesday the 23rd?</td>
</tr>
</tbody>
</table>
Automatic Interpretation of Dialogue Acts

- How do we automatically identify dialogue acts?
- Given an utterance:
  - Decide whether it is a QUESTION, STATEMENT, SUGGEST, or ACK
- Recognizing illocutionary force will be crucial to building a dialogue agent
- Perhaps we can just look at the form of the utterance to decide?
Can we just use the surface syntactic form?

- YES-NO-Q’s have auxiliary-before-subject syntax:
  - Will breakfast be served on USAir 1557?
- STATEMENTS have declarative syntax:
  - I don’t care about lunch
- COMMAND’s have imperative syntax:
  - Show me flights from Milwaukee to Orlando on Thursday night
<table>
<thead>
<tr>
<th>Surface form</th>
<th>Locutionary Force</th>
<th>Illocutionary Force</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can I have the rest of your sandwich?</td>
<td>Question</td>
<td>Request</td>
</tr>
<tr>
<td>I want the rest of your sandwich</td>
<td>Declarative</td>
<td>Request</td>
</tr>
<tr>
<td>Give me your sandwich!</td>
<td>Imperative</td>
<td>Request</td>
</tr>
</tbody>
</table>
Dialogue Act ambiguity

- Can you give me a list of the flights from Atlanta to Boston?
  - This looks like an INFO-REQUEST.
  - If so, the answer is:
    - YES.
  - But really it’s a DIRECTIVE or REQUEST, a polite form of:
    - Please give me a list of the flights…
- What looks like a QUESTION can be a REQUEST
Dialogue Act ambiguity

- Similarly, what looks like a STATEMENT can be a QUESTION:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>OPEN-OPTION</td>
<td>I was wanting to make some arrangements for a trip that I’m going to be taking uh to LA uh beginning of the week after next</td>
</tr>
<tr>
<td>A</td>
<td>HOLD</td>
<td>OK uh let me pull up your profile and I’ll be right with you here. [pause]</td>
</tr>
<tr>
<td>A</td>
<td>CHECK</td>
<td>And you said you wanted to travel next week?</td>
</tr>
<tr>
<td>U</td>
<td>ACCEPT</td>
<td>Uh yes.</td>
</tr>
</tbody>
</table>
Indirect speech acts

- Utterances which use a surface statement to ask a question
- Utterances which use a surface question to issue a request
DA interpretation as statistical classification

- Lots of clues in each sentence that can tell us which DA it is:
  - Words and Collocations:
    - *Please* or *would you*: good cue for REQUEST
    - *Are you*: good cue for INFO-REQUEST
  - Conversational Structure
    - *Yeah* following a proposal is probably AGREEMENT; *yeah* following an INFORM probably a BACKCHANNEL
Classification tasks

Assign the correct **class label** for a given input/object.

In basic classification tasks, each input is considered in isolation from all other inputs, and the set of labels is defined in advance.

**Relevant Examples:**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Object</th>
<th>Label’s categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tagging</td>
<td>Word</td>
<td>POS</td>
</tr>
<tr>
<td>Dialog Acts</td>
<td>Utterance</td>
<td>Greeting, Ask, Clarify</td>
</tr>
<tr>
<td>Task/Topic</td>
<td>Utterance</td>
<td>Directions, PhoneNum, Review</td>
</tr>
</tbody>
</table>

Adapted from: Foundations of Statistical NLP (Manning et al)
Understanding Meaning via Classification

- Two ways:
  - Define patterns by hand that map to classes
  - Learn automatically from data how to assign utterances to classes
Two phases of the Façade NLU (by hand)

NLU: Surface text to discourse acts

Surface text
“You two look so happy in this wedding picture”

Discourse acts (~25)
- Agree
- Disagree
- Praise
- Refer to
- ...

Conversation management: Discourse acts to reactions

Context: Affinity Game
- Proposer
- Proposer
- Priority map

Context: Global
- Proposer
- Proposer
- Priority map

Selector
Our goal is to decide for each sentence what dialogue act it is

This is a **classification task** (we are making a 1-of-N classification decision for each sentence)

With **N** classes (= number of dialog acts).

Two models corresponding to the kinds of cues from the input sentence.

- **Conversational Structure:** Probability of one dialogue act following another
  \[ P(\text{Answer}|\text{Question}) \]

- **Words and Syntax:** Probability of a sequence of words given a dialogue act:
  \[ P(\text{“do you”}|\text{Question}) \]

Will talk about NLTK classify modules after Evaluation
Evaluation & Metrics
Dialogue System Evaluation

- **Key point.**
- Whenever we design a new algorithm or build a new application, need to evaluate it
- Two kinds of evaluation
  - **Extrinsic:** embedded in some external task
  - **Intrinsic:** some sort of more local evaluation.

- How to evaluate a dialogue system?
- What constitutes success or failure for a dialogue system?
Dialogue System Evaluation

- It turns out we’ll need an evaluation metric for two reasons
  - 1) the normal reason: we need a metric to help us compare different implementations
    - can’t improve it if we don’t know where it fails
    - Can’t decide between two algorithms without a goodness metric
  - 2) a new reason: we will need a metric for “how good a dialogue went” as an input to reinforcement learning:
    - automatically improve our conversational agent performance via learning
Task Success

- % of subtasks completed
- Correctness of each questions/answer/error msg
- Correctness of total solution
  - Attribute-Value matrix (AVM)
  - Kappa coefficient
- Users’ perception of whether task was completed
Efficiency Cost


- Total elapsed time in seconds or turns
- Number of queries
- Turn correction ratio: number of system or user turns used solely to correct errors, divided by total number of turns
Quality Cost

- # of times system failed to understand anything
- # of times system had to provide help
- Inappropriateness (verbose, ambiguous) of system’s questions, answers, error messages
NLU Metric: Concept Accuracy

- “Concept accuracy” or “Concept error rate”
- % of semantic concepts that the NLU component returns correctly
- I want to arrive in Austin at 5:00
  - DESTCITY: Boston
  - Time: 5:00
- Concept accuracy = 50%
- Average this across entire dialogue
- “How many of the sentences did the system understand correctly”
User Satisfaction: Sum of Many Measures

Was the system easy to understand? (TTS Performance)
Did the system understand what you said? (ASR Performance)
Was it easy to find the message/plane/train you wanted? (Task Ease)
Was the pace of interaction with the system appropriate? (Interaction Pace)
Did you know what you could say at each point of the dialog? (User Expertise)
How often was the system sluggish and slow to reply to you? (System Response)
Did the system work the way you expected it to in this conversation? (Expected Behavior)
Do you think you’d use the system regularly in the future? (Future Use)
Some particular success metric

- Could we use it to help drive learning?
- Recent work uses various metrics to learn an optimal **policy** or **strategy** for how the conversational agent should behave.
Utility Agent
(see Walker et al 1998 in reading)
Utility

- A utility function
  - maps a state or state sequence
  - onto a real number
  - describing the goodness of that state
  - i.e. the resulting “happiness” of the agent

- Principle of Maximum Expected Utility:
  - A rational agent should choose an action that maximizes the agent’s expected utility
Modeling a dialogue system as a probabilistic agent (decision theoretic planning)

- A conversational agent can be characterized by:
  - The current knowledge of the system
    - A set of states $S$ the agent can be in
  - A set of actions $A$ the agent can take
  - A goal $G$, which implies
    - A success metric that tells us how well the agent achieved its goal
    - A way of using this metric to create a strategy or policy $\pi$ for what action to take in any particular state.
What do we mean by actions $A$ and policies $\pi$?

- Kinds of decisions a conversational agent needs to make:
  - When should I ground/confirm/reject/ask for clarification on what the user just said?
  - When should I ask a directive prompt, when an open prompt?
  - When should I use user, system, or mixed initiative?
Learning & Dialogue Management

- Could we learn what the right action is
  - Ask an open question or a direct one?
  - Present one option or several?

- By learning a policy which,
  - given various information about the current state,
  - dynamically chooses the action which maximizes dialogue success
Text Classification
Tools & Modules in NLTK
Training a Classifier from Data

- Define classes
  - E.g. tasks, utterance types, dialog acts
- Label example utterances
- Define and extract features
- Train the classifier
- Evaluate its performance; see if you can get better features or more data to improve performance
Classification Tools

- **nltk.classify**: Classes and interfaces for labeling tokens with category labels (or class labels). **nltk.classify.api**: Interfaces for labeling tokens with category labels (or class labels).

- **nltk.classify.weka**: Classifiers that make use of the external 'Weka' package.

- **nltk.classify.naivebayes**: A classifier based on the Naive Bayes algorithm.

- Etc etc
Supervised classification

A classifier is called **supervised** if it is built based on training corpora containing the correct label for each input.
Define a feature extractor: a feature for each word, indicating whether the document contains that word.

```python
def document_features(document):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains(%s)' % word] = (word in document_words)
    return features

print(document_features(movie_reviews.words('pos/cv957_8737.txt')))
```
Now that we've defined our feature extractor, we can use it to train a classifier.

```python
featuresets = [(document_features(d), c) for (d, c) in documents]
train_set, test_set = featuresets[100:], featuresets[:100]
classifier = nltk.NaiveBayesClassifier.train(train_set)

>>> print nltk.classify.accuracy(classifier, test_set)
0.81
```
We can examine the classifier to determine which features it found most effective for distinguishing the review’s sentiment.

```
>>> classifier.show_most_informative_features(5)
Most Informative Features
    contains(outstanding) = True             pos : neg   =   11.1 : 1.0
    contains(seagal) = True                   neg : pos   =   7.7 : 1.0
    contains(wonderfully) = True              pos : neg   =   6.8 : 1.0
    contains(damon) = True                     pos : neg   =   5.9 : 1.0
    contains(wasted) = True                    neg : pos   =   5.8 : 1.0
```

Apparently in this corpus, a review that mentions "Seagal" is almost 8 times more likely to be negative than positive, while a review that mentions "Damon" is about 6 times more likely to be positive.
Classification

- Define classes
- Label text (Collect lots of utterances for your ChatBot)
- Extract Features
- Choose a classifier
  - The Naive Bayes Classifier
  - Decision Tree
  - Boostexter
- Train it (and test it)
- Use it to classify new examples
Categories (Labels, Classes)

- Labeling data
- 2 problems:
  - Decide the possible classes (which ones, how many)
    - Domain and application dependent
    - Trade-off between accuracy and coverage
- Label text
  - Difficult, time consuming, inconsistency between annotators
Features

- >>> text = "Seven-time Formula One champion Michael Schumacher took on the Shanghai circuit Saturday in qualifying for the first Chinese Grand Prix."
- >>> label = "sport"
- >>> labeled_text = LabeledText(text, label)

- Here the classification takes as input the whole string
- What’s the problem with that?
- What are the features that could be useful for this example?
Feature terminology

- **Feature**: An aspect of the input that is relevant to the task
- **Feature value**: the realization of the feature in the text

Some typical features

- **Words**: Kerry, Schumacher, China…
- **Frequency of word**: Kerry(10), Schumacher(1)…
- **Chunks, parse trees, POS**
- **Are there dates? Yes/no**
- **Capitalization** (is word capitalized?)
- **Are there PERSONS? Yes/no**
- **Are there ORGANIZATIONS? Yes/no**
- **WordNet**:
  - Holonyms (China is part of Asia),
  - Synonyms (China, People's Republic of China)
Feature Types

- **Boolean (or Binary) Features**
- Features that generate boolean (binary) values.
- Boolean features are the simplest and the most common type of feature.

\[ f_1(\text{text}) = \begin{cases} 1 & \text{if text contain “Grace”} \\ 0 & \text{otherwise} \end{cases} \]

\[ f_2(\text{text}) = \begin{cases} 1 & \text{if text contain PERSON} \\ 0 & \text{otherwise} \end{cases} \]
Feature Types

- **Integer Features**
- Features that generate integer values.
- Integer features can be used to give classifiers access to more precise information about the text.

- $F_1(\text{input}) = \text{Number of times input contains "Grace"}$
- $F_2(\text{input}) = \text{Number of times input contains PERSON}$
Feature selection

- Selecting relevant features and deciding how to encode them has enormous impact on the learning method's ability to extract a good model.
- How do we choose the “right” features?
- Trial-and-error using intuitions about what information is relevant to the problem.
- Not enough features or features not general: can’t find general discriminating patterns (low accuracy).
- Too many features: learns idiosyncrasies of your training data that don't generalize (overfits).
Training/Development/Test

- To refining the feature set, we perform **error analysis**.
- **The development set**, containing the corpus data for creating the model. This development set is then subdivided into the **training set** and the **dev-test set**.
- The training set is used to train the model, and the dev-test set is used to perform error analysis.
  - Look at errors, change features or model
- The test set serves in our final evaluation of the system.
Training

- Adaptation of the classifier to the data
- Usually the classifier is defined by a set of parameters
- Training is the procedure for finding a “good” set of parameters
- Goodness is determined by an optimization criterion such as misclassification rate
Training size

- The more the better! (usually)
  - Make sure that test set contains instances for all classes
- Results for text classification (from Shen & Yang)

Figure 1: Test error vs training size on the newsgroups rec.sport.baseball and rec.sport.hockey
Confusion Matrix: Error Analysis

- A table that shows, for each class, which ones your algorithm got right and which wrong

Gold standard

Algorithm's guess
Can use this to merge or get rid of classes

Table 4: Confusion matrix for single line identification using 2000 lines of training data and bigram statistics. Rows, which sum to 100, are the correct language. Columns are the language identified.
The Test Set

- The test set typically has the same format as the training set.
- A distinct set of “unseen” utterances set aside and labelled.
- Trade-off between the amount of data available for testing and development and the amount available for training.
- Both sets need to have representative examples.
- If you use classification in your project, your evaluation can include the subtask of classification accuracy, over time, as you refine it and make it better.
Accuracy

- The simplest metric: accuracy, measures the percentage of inputs in the test set that the classifier correctly labeled.
  - For example, a spam classifier that predicts correctly spam 60 times in a test set containing 80 emails would have an accuracy of \( \frac{60}{80} = 75\% \).
- Important to take into consideration the frequencies of the individual class labels
  - If only 1/100 is spam, an accuracy of 90% is bad
  - If ½ is spam, accuracy of 90% is good
- Important: compare with fair baselines

Python code example:

```python
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>>> print 'Accuracy: %.2f' % nltk.classify.accuracy(classifier, test_set)
0.75
```