In [1]:
```
###### Understanding intents and entities
```

In [2]:
```
bot_template = "BOT : {0}"
user_template = "USER : {0}"

# Define a function that responds to a user's message: respond
def respond(message):
    # Concatenate the user's message to the end of a standard bot respone
    bot_message = "I can hear you! You said: " + message
    # Return the result
    return bot_message
```

In [3]:
```
# Define a function that sends a message to the bot: send_message
def send_message(message):
    # Print user_template including the user_message
    print(user_template.format(message))
    # Get the bot's response to the message
    response = respond(message)
    # Print the bot template including the bot's response.
    print(bot_template.format(response))

# Send a message to the bot
send_message("hello")
```

In [16]:
```
import re
keywords = {
            'greet': ['hello', 'hi', 'hey'],
            'thankyou': ['thank', 'thx'],
            'goodbye': ['bye', 'farewell']
           }
# Define a dictionary of patterns
patterns = {}

# Iterate over the keywords dictionary
for intent,words in keywords.items():
    # Create regular expressions and compile them into pattern objects
    patterns[intent] = re.compile("|".join(words))

# Print the patterns
print(patterns)
```

```
{'greet': re.compile('hello|hi|hey'), 'thankyou': re.compile('thank|thx'), 'goodbye': re.compile('bye|farewell')}
```

In [17]:
```
responses = {'greet': 'Hello you! :)',
             'thankyou': 'you are very welcome',
             'default': 'default message',
             'goodbye': 'goodbye for now'
            }

# Define a function to find the intent of a message
def match_intent(message):
    matched_intent = None
    for intent, pattern in patterns.items():
        # Check if the pattern occurs in the message
        if pattern.search(message):
            matched_intent = intent
    return matched_intent

# Define a respond function
def respond(message):
    # Call the match_intent function
    intent = match_intent(message)
    # Fall back to the default response
    key = "default"
    if intent in responses:
        key = intent
    return responses[key]

# Send messages
send_message("hello!")
send_message("bye byeee")
send_message("thanks very much!")
```

```
USER : hello!
BOT : Hello you! :)
USER : bye byeee
BOT : goodbye for now
USER : thanks very much!
BOT : you are very welcome
```

find_name中的关键词好像没给出?

In [6]:

```python
# Define find_name()
def find_name(message):
    name = None
    # Create a pattern for checking if the keywords occur
    name_keyword = re.compile("name|call") #假设 keyword 为 name 和 call
    # Create a pattern for finding capitalized words
    name_pattern = re.compile("[A-Z]{1}[a-z]*")
    if name_keyword.search(message):
        # Get the matching words in the string
        name_words = name_pattern.findall(message)
        if len(name_words) > 0:
            # Return the name if the keywords are present
            name = ' '.join(name_words)
    return name

# Define respond()
def respond(message):
    # Find the name
    name = find_name(message)
    if name is None:
        return "Hi there!"
    else:
        return "Hello, {0}!".format(name)

# Send messages
send_message("my name is David Copperfield")
send_message("call me Ishmael")
send_message("people call me Cassandra")
```

```
USER : my name is David Copperfield
BOT : Hello, David Copperfield!
USER : call me Ishmael
BOT : Hello, Ishmael!
USER : people call me Cassandra
BOT : Hello, Cassandra!
```

In [7]:

```python
###### Word vectors
```

In [2]:

```python
import spacy
import numpy as np
sentences = [' i want to fly from boston at 838 am and arrive in denver at 1110 in the morning',
 ' what flights are available from pittsburgh to baltimore on thursday morning',
 ' what is the arrival time in san francisco for the 755 am flight leaving washington',
 ' cheapest airfare from tacoma to orlando',
 ' round trip fares from pittsburgh to philadelphia under 1000 dollars',
 ' i need a flight tomorrow from columbus to minneapolis',
 ' what kind of aircraft is used on a flight from cleveland to dallas',
 ' show me the flights from pittsburgh to los angeles on thursday',
 ' all flights from boston to washington',
 ' what kind of ground transportation is available in denver',
 ' show me the flights from dallas to san francisco',
 ' show me the flights from san diego to newark by way of houston',
 ' what is the cheapest flight from boston to bwi',
 ' all flights to baltimore after 6 pm',
 ' show me the first class fares from boston to denver',
 ' show me the ground transportation in denver',
 ' all flights from denver to pittsburgh leaving after 6 pm and before 7 pm',
 ' i need information on flights for tuesday leaving baltimore for dallas dallas to boston and boston to baltimore',
 ' please give me the flights from boston to pittsburgh on thursday of next week',
 ' i would like to fly from denver to pittsburgh on united airlines',
 ' show me the flights from san diego to newark',
 ' please list all first class flights on united from denver to baltimore',
 ' what kinds of planes are used by american airlines',
 " i'd like to have some information on a ticket from denver to pittsburgh and atlanta",
 " i'd like to book a flight from atlanta to denver",
 ' which airline serves denver pittsburgh and atlanta',
 " show me all flights from boston to pittsburgh on wednesday of next week which leave boston after 2 o'clock pm",
 ' atlanta ground transportation', ' i also need service from dallas to boston arriving by noon',
 ' show me the cheapest round trip fare from baltimore to dallas']
```

In [3]:
```
# Load the spacy model: nlp, en_core_web_md
nlp = spacy.load("en_core_web_md")

# Calculate the length of sentences
n_sentences = len(sentences)

# Calculate the dimensionality of nlp
embedding_dim = nlp.vocab.vectors_length

# Initialize the array with zeros: X
X = np.zeros((n_sentences, embedding_dim))

# Iterate over the sentences
for idx, sentence in enumerate(sentences):
    # Pass each each sentence to the nlp object to create a document
    doc = nlp(sentence)
    # Save the document's .vector attribute to the corresponding row in X
    X[idx, :] = doc.vector
print(X)
```

```
[[ 0.07225148  0.23085858 -0.05721947 ...  0.04366079 -0.11307659
   0.21412031]
 [ 0.11721275  0.04901224 -0.14363982 ... -0.14155565 -0.06796242
   0.18658884]
 [ 0.13610677  0.17966814 -0.05222185 ...  0.04048143 -0.16161631
   0.12550484]
 ...
 [ 0.16177949  0.0670125   0.088523   ... -0.0717375  -0.066895
   0.0520265 ]
 [ 0.11427727  0.11960033 -0.03754008 ... -0.07084992 -0.142048
   0.19235454]
 [-0.00103583  0.01718292 -0.04143599 ... -0.06478167 -0.04346119
   0.14688456]]
```

In [10]:
```
#### Intent classification with sklearn
```

In [11]:
```
import pandas as pd
X_train = pd.read_csv('SVM/X_train.csv')
X_test = pd.read_csv('SVM/X_test.csv')
y_train = pd.read_csv('SVM/y_train.csv')['label']
y_test = pd.read_csv('SVM/y_test.csv')['label']
```

In [12]:
```
# Import SVC
from sklearn.svm import SVC

# Create a support vector classifier
clf = SVC(gamma="auto")

# Fit the classifier using the training data
clf.fit(X_train,y_train)

# Predict the labels of the test set
y_pred = clf.predict(X_test)

# Count the number of correct predictions
n_correct = 0
for i in range(len(y_test)):
    if y_pred[i] == y_test[i]:
        n_correct += 1

print("Predicted {0} correctly out of {1} test examples.".format(n_correct, len(y_test)))
print("Accuracy rate: ",n_correct/len(y_test))
```

```
Predicted 162 correctly out of 201 test examples.
Accuracy rate:  0.8059701492537313
```

In [ ]:
```
#### Entity extraction
```

In [4]:
```
include_entities = ['DATE', 'ORG', 'PERSON']
def extract_entities(message):
    ents = dict.fromkeys(include_entities)
    doc = nlp(message)
    for ent in doc.ents:
        if ent.label_ in include_entities:
            ents[ent.label_] = ent
    return ents
print(extract_entities('friends called Mary who have worked at Google since 2010'))
print(extract_entities('people who graduated from MIT in 1999'))
```

```
{'DATE': 2010, 'ORG': Google, 'PERSON': Mary}
{'DATE': 1999, 'ORG': MIT, 'PERSON': None}
```

In [5]:
```
## Assigning roles using spaCy's parser
def entity_type(word):
    _type = None
    if word.text in colors:
        _type = "color"
    elif word.text in items:
        _type = "item"
    return _type

colors = ['black', 'red', 'blue']
items = ['shoes', 'handback', 'jacket', 'jeans']
```

In [6]:
```
## Assigning roles using spaCy's parser

# Create the document
doc = nlp("let's see that jacket in red and some blue jeans")

# Iterate over parents in parse tree until an item entity is found
def find_parent_item(word):
    # Iterate over the word's ancestors
    for parent in word.ancestors:
        # Check for an "item" entity
        if entity_type(parent) == "item":
            return parent.text
    return None

# For all color entities, find their parent item
def assign_colors(doc):
    # Iterate over the document
    for word in doc:
        # Check for "color" entities
        if entity_type(word) == "color":
            # Find the parent
            item =  find_parent_item(word)
            print("item: {0} has color : {1}".format(item, word))

# Assign the colors
assign_colors(doc)
```

```
item: jacket has color : red
item: jeans has color : blue
```

In [13]:
```
#### Robust NLU with Rasa
```

In [14]:
```
## Rasa NLU
# Import necessary modules
from rasa.nlu.training_data import load_data
from rasa.nlu.config import RasaNLUModelConfig
from rasa.nlu.model import Trainer
from rasa.nlu import config

```

未知的配置文件和训练文件

In [15]:
```
# Create a trainer
trainer = Trainer(config.load(""))

# Load the training data
training_data = load_data("")

# Create an interpreter by training the model
interpreter = trainer.train(training_data)

# Try it out
print(interpreter.parse("I'm looking for a Mexican restaurant in the North of town"))
```

```
WARNING:tensorflow:
The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-sunset.md (https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-sunset.md)
  * https://github.com/tensorflow/addons (https://github.com/tensorflow/addons)
  * https://github.com/tensorflow/io (https://github.com/tensorflow/io) (for I/O related ops)
If you depend on functionality not listed there, please file an issue.

WARNING:tensorflow:From C:\Users\84353\Anaconda3\envs\chat_box\lib\site-packages\tensor2tensor\utils\adafactor.py:27: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From C:\Users\84353\Anaconda3\envs\chat_box\lib\site-packages\tensor2tensor\utils\multistep_optimizer.py:32: The name tf.train.AdamOptimizer is deprecated. Please use tf.compat.v1.train.AdamOptimizer instead.

WARNING:tensorflow:From C:\Users\84353\Anaconda3\envs\chat_box\lib\site-packages\tensor2tensor\models\research\glow_init_hook.py:25: The name tf.train.SessionRunHook is deprecated. Please use tf.estimator.SessionRunHook instead.

WARNING:tensorflow:From C:\Users\84353\Anaconda3\envs\chat_box\lib\site-packages\tensor2tensor\models\research\neural_stack.py:51: The name tf.nn.rnn_cell.RNNCell is deprecated. Please use tf.compat.v1.nn.rnn_cell.RNNCell instead.

WARNING:tensorflow:From C:\Users\84353\Anaconda3\envs\chat_box\lib\site-packages\tensor2tensor\utils\trainer_lib.py:111: The name tf.OptimizerOptions is deprecated. Please use tf.compat.v1.OptimizerOptions instead.


WARNING:tensorflow:From C:\Users\84353\Anaconda3\envs\chat_box\lib\site-packages\tensor2tensor\utils\trainer_lib.py:111: The name tf.OptimizerOptions is deprecated. Please use tf.compat.v1.OptimizerOptions instead.


WARNING:tensorflow:From C:\Users\84353\Anaconda3\envs\chat_box\lib\site-packages\tensorflow_gan\python\estimator\tpu_gan_estimator.py:42: The name tf.estimator.tpu.TPUEstimator is deprecated. Please use tf.compat.v1.estimator.tpu.TPUEstimator instead.


WARNING:tensorflow:From C:\Users\84353\Anaconda3\envs\chat_box\lib\site-packages\tensorflow_gan\python\estimator\tpu_gan_estimator.py:42: The name tf.estimator.tpu.TPUEstimator is deprecated. Please use tf.compat.v1.estimator.tpu.TPUEstimator instead.


---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-4d69d1aab88f> in <module>
      3
      4 # Load the training data
----> 5 training_data = load_data("")
      6
      7 # Create an interpreter by training the model

~\Anaconda3\envs\chat_box\lib\site-packages\rasa\nlu\training_data\loading.py in load_data(resource_name, language)
     62
     63     if not os.path.exists(resource_name):
---> 64         raise ValueError(f"File '{resource_name}' does not exist.")
     65
     66     files = io_utils.list_files(resource_name)

ValueError: File '' does not exist.
```

In [ ]:
```
## Data-efficient entity recognition
```

In [ ]:
```
print(interpreter.parse("show me Chinese food in the centre of town"))
print(interpreter.parse("I want an Indian restaurant in the west"))
print(interpreter.parse("are there any good pizza places in the center?"))
```