# CODE FOR IBM CLOUD BASED DISTRIBUTION SYSTEM TEAM TRIO

:from IPython.display import Markdown, display, HTMLimport warningswarnings.filterwarnings('ignore')

:

In [ ]

!pip install --index-url https://pypi.python.org/simple -U "pip"!pip install --index-url https://pypi.python.org/simple -U "assistant-dialog-skill-analysis>=1.1.0"

In [3]:

# Standard python librariesimport sys, osimport jsonimport importlibfrom collections import Counter

# External python librariesimport pandas as pdimport numpy as npimport nltknltk.download('stopwords')nltk.download('punkt')import ibm\_watson

# Internal python librariesfrom assistant\_dialog\_skill\_analysis.utils import skills\_utilfrom assistant\_dialog\_skill\_analysis.highlighting import highlighterfrom assistant\_dialog\_skill\_analysis.data\_analysis import summary\_generatorfrom assistant\_dialog\_skill\_analysis.data\_analysis import divergence\_analyzerfrom assistant\_dialog\_skill\_analysis.data\_analysis import similarity\_analyzerfrom assistant\_dialog\_skill\_analysis.term\_analysis import chi2\_analyzerfrom assistant\_dialog\_skill\_analysis.term\_analysis import keyword\_analyzerfrom assistant\_dialog\_skill\_analysis.term\_analysis import entity\_analyzerfrom assistant\_dialog\_skill\_analysis.confidence\_analysis import confidence\_analyzerfrom assistant\_dialog\_skill\_analysis.inferencing import inferencerfrom assistant\_dialog\_skill\_analysis.experimentation import data\_manipulator

[nltk\_data] Downloading package stopwords to

[nltk\_data] /home/dsxuser/nltk\_data...

[nltk\_data] Unzipping corpora/stopwords.zip.

[nltk\_data] Downloading package punkt to /home/dsxuser/nltk\_data...

[nltk\_data] Unzipping tokenizers/punkt.zip.

In [4]:

importlib.reload(skills\_util)

# Change Assistant API version if needed# Find Latest --> https://cloud.ibm.com/docs/services/assistant?topic=assistant-release-notesAPI\_VERSION = '2019-02-28'

# Change URL based on IBM Cloud datacenter you use URL = "https://gateway.watsonplatform.net/assistant/api" # Dallas (Default US South)

#URL = "https://gateway-s.watsonplatform.net/assistant/api" # Dallas Staging#URL = "https://gateway-wdc.watsonplatform.net/assistant/api" # Washington, DC#URL = "https://gateway-fra.watsonplatform.net/assistant/api" # Frankfurt#URL = "https://gateway-syd.watsonplatform.net/assistant/api" # Sydney#URL = "https://gateway-tok.watsonplatform.net/assistant/api" # Tokyo#URL = "https://gateway-lon.watsonplatform.net/assistant/api" # London

# By default we only need the IAM API Key & the Workspace ID

# If you run the notebook regularly you can uncomment the two lines below# & comment out the line after it#iam\_apikey = '###'#workspace\_id = '###'#Prompt user for credentialsiam\_apikey, workspace\_id = skills\_util.input\_credentials()

conversation, workspace = skills\_util.retrieve\_workspace(iam\_apikey=iam\_apikey,

workspace\_id=workspace\_id,

url=URL,

api\_version=API\_VERSION)

#If you do not have IAM based API Keys#but have access to a Username, Password & Workspace ID#You can comment out the two lines above & uncomment the lines below to authenticate#username = 'apikey'#password = '###'#workspace\_id = '###'#conversation, workspace = skills\_util.retrieve\_workspace(username=username,# password=password,# workspace\_id=workspace\_id,# url=URL,# api\_version=API\_VERSION)

Please enter apikey: ········

Please enter workspace-id: ········

In [5]:

# Extract user workspaceworkspace\_data, workspace\_vocabulary = skills\_util.extract\_workspace\_data(workspace)entity\_dict = conversation.list\_entities(workspace\_id).get\_result()entities\_list = [item['entity'] for item in entity\_dict['entities']]# Create workspace data frameworkspace\_pd = pd.DataFrame(workspace\_data)

display(Markdown("### Sample of Utterances & Intents"))display(HTML(workspace\_pd.sample(n = len(workspace\_pd) if len(workspace\_pd)<10 else 10)

.to\_html(index=False)))if entities\_list:

display(Markdown("### Sample of Entities"))

display(HTML(pd.DataFrame({"Entity":entities\_list})

.sample(n = len(entities\_list) if len(entities\_list)<10 else 10)

.to\_html(index=False)))

In [6]:

importlib.reload(summary\_generator)summary\_generator.generate\_summary\_statistics(workspace\_data, entities\_list)

In [7]:

importlib.reload(summary\_generator)class\_imb\_flag = summary\_generator.class\_imbalance\_analysis(workspace\_pd)

In [8]:

importlib.reload(summary\_generator)summary\_generator.scatter\_plot\_intent\_dist(workspace\_pd)

/opt/conda/envs/Python36/lib/python3.6/site-packages/matplotlib/font\_manager.py:1241: UserWarning: findfont: Font family ['normal'] not found. Falling back to DejaVu Sans.

(prop.get\_family(), self.defaultFamily[fontext]))

In [9]:

importlib.reload(summary\_generator)summary\_generator.show\_user\_examples\_per\_intent(workspace\_data)

In [10]:

importlib.reload(chi2\_analyzer)unigram\_intent\_dict, bigram\_intent\_dict = chi2\_analyzer.get\_chi2\_analysis(workspace\_pd)

n [11]:

importlib.reload(keyword\_analyzer)

INTENTS\_TO\_DISPLAY = 30 # Total number of intents for displayMAX\_TERMS\_DISPLAY = 30 # Total number of terms to display

intent\_list = []keyword\_analyzer.seaborn\_heatmap(workspace\_pd, INTENTS\_TO\_DISPLAY, MAX\_TERMS\_DISPLAY, intent\_list)

In [12]:

importlib.reload(keyword\_analyzer)# intent\_list = ['intent1','intent2','intent3'] intent\_list = []

MAX\_TERMS\_DISPLAY = 20 # Total number of terms to display

if intent\_list:

keyword\_analyzer.seaborn\_heatmap(workspace\_pd, INTENTS\_TO\_DISPLAY, MAX\_TERMS\_DISPLAY, intent\_list)

In [13]:

importlib.reload(chi2\_analyzer)ambiguous\_unigram\_df = chi2\_analyzer.get\_confusing\_key\_terms(unigram\_intent\_dict)

In [14]:

importlib.reload(chi2\_analyzer)ambiguous\_bigram\_df = chi2\_analyzer.get\_confusing\_key\_terms(bigram\_intent\_dict)

In [15]:

# Add specific intent or intent pairs for which you would like to see overlapimportlib.reload(chi2\_analyzer)intent1 = 'Goodbye'intent2 = ''chi2\_analyzer.chi2\_overlap\_check(ambiguous\_unigram\_df,ambiguous\_bigram\_df,intent1,intent2)

In [16]:

importlib.reload(similarity\_analyzer)similar\_utterance\_diff\_intent\_pd = similarity\_analyzer.ambiguous\_examples\_analysis(workspace\_pd)

import typesfrom botocore.client import Configimport ibm\_boto3

def \_\_iter\_\_(self): return 0

COS\_AUTH\_ENDPOINT ="https://iam.cloud.ibm.com/identity/token"COS\_ENDPOINT = 'https://s3-api.us-geo.objectstorage.service.networklayer.com'# please fill in the details here:COS\_API\_KEY\_ID = '###' #API key of your COSCOS\_RESOURCE\_CRN = '###' #Resource CRN, i.e. 'crn:v1:bluemix:public...''IBM\_COS\_BUCKET = '###' #Name of your COS bucketIBM\_COS\_FILE\_KEY = '###' #Name of the test data, i.e. 'test\_data\_set.csv'

cos = ibm\_boto3.resource("s3",

ibm\_api\_key\_id=COS\_API\_KEY\_ID,

ibm\_service\_instance\_id=COS\_RESOURCE\_CRN,

ibm\_auth\_endpoint=COS\_AUTH\_ENDPOINT,

config=Config(signature\_version="oauth"),

endpoint\_url=COS\_ENDPOINT)

body = cos.Object(IBM\_COS\_BUCKET, IBM\_COS\_FILE\_KEY).get()['Body']

if not hasattr(body, "\_\_iter\_\_"): body.\_\_iter\_\_ = types.MethodType( \_\_iter\_\_, body )separator = ','test\_df = pd.read\_csv(body, sep=separator, names=["utterance", "intent"])

display(Markdown("### Random Test Sample"))display(HTML(test\_df.sample(n=10).to\_html(index=False)))

18]:

# Maximum of 5 threads for faster inferenceTHREAD\_NUM = 5full\_results = inferencer.inference(conversation,

workspace\_id,

test\_df,

max\_retries=10,

max\_thread=THREAD\_NUM,

verbose=False)

Starting Thread-1

Starting Thread-2

Starting Thread-3

Starting Thread-4

Starting Thread-5

Exiting Thread-1

Exiting Thread-2

Exiting Thread-3

Exiting Thread-5

Exiting Thread-4

--- Total time: 8.201696872711182 seconds ---

In [19]:

importlib.reload(summary\_generator)summary\_generator.generate\_summary\_statistics(test\_df)summary\_generator.show\_user\_examples\_per\_intent(test\_df)

In [21]:

importlib.reload(inferencer)results = full\_results[['correct\_intent', 'top\_confidence','top\_intent','utterance']]accuracy = inferencer.calculate\_accuracy(results)display(Markdown("### Accuracy on Test Data: {} %".format(accuracy)))

In [22]:

importlib.reload(inferencer)wrongs\_df = inferencer.calculate\_mistakes(results)display(Markdown("### Intent Detection Mistakes"))display(Markdown("Number of Test Errors: {}".format(len(wrongs\_df))))

with pd.option\_context('max\_colwidth', 250):

if not wrongs\_df.empty:

display(wrongs\_df)

In [25]:

importlib.reload(confidence\_analyzer)

# Calculate intent with most test examplesfor label in list(test\_df['intent'].value\_counts().index):

if label != skills\_util.OFFTOPIC\_LABEL:

MOST\_FREQUENT\_INTENT = label

break

# Specify intents of interest for analysis INTENT\_LIST = [MOST\_FREQUENT\_INTENT]

analysis\_df\_list = confidence\_analyzer.analysis(results, INTENT\_LIST)

Threshold Analysis for Intent: General\_Connect\_to\_Agent

Out of Domain examples fewer than 5 thus no False Acceptance Rate (FAR) calculated

3.3 Highlight term importance

This intent can be ground-truth or an incorrectly predicted intent. It provides term level insights about which terms the classifier thought were important in relation to that specific intent.

Even if the system predicts an intent correctly, the terms which the intent classifier thought were important may not be as expected by human insight. Human insight might suggest that the intent classifier is focusing on the wrong terms.

The score of each term in the following highlighted images can be viewed as importance factor of that term for that specific intent. The larger the score, the more important the term.

You can get the highlighted images for either wrongly-predicted utterances or utterances where the classifier returned a low confidence.

he closest agent?" # input exampleintent = "General\_Connect\_to\_Agent" # input an intent in your workspace which you are interested in.

inference\_results = inferencer.inference(conversation=conversation,

workspace\_id=workspace\_id,

test\_data=pd.DataFrame({'utterance':[utterance],

'intent':[intent]}),

max\_retries = 10,

max\_thread = 1,

verbose = False)

highlighter.get\_highlights\_in\_batch\_multi\_thread(conversation,

workspace\_id,

inference\_results,

None,

1,

1)

Identified 1 problematic utterances

Starting Thread-1

Starting Thread-2

Starting Thread-3

Starting Thread-4

Starting Thread-5

Exiting Thread-4

Exiting Thread-1

Exiting Thread-3

Exiting Thread-5

Exiting Thread-2

--- Total time: 0.8515188694000244 seconds ---

In [27]:

importlib.reload(highlighter)

# The output folder for generated images# Note modify this if you want the generated images to be stored in a different directory

highlighting\_output\_folder = './highlighting\_images/'if not os.path.exists(highlighting\_output\_folder):

os.mkdir(highlighting\_output\_folder)

# The threshold the prediction needs to achieve below which # it will be considered as `out of domain` or `offtopic` utterances. threshold = 0.2

# Maximum number of test set examples whose highlighting analysis will be conductedK=25highlighter.get\_highlights\_in\_batch\_multi\_thread(conversation,

workspace\_id,full\_results,

highlighting\_output\_folder,

threshold,

K)

Identified 9 problematic utterances

Starting Thread-1

Starting Thread-2

Starting Thread-3

Starting Thread-4

Starting Thread-5

Exiting Thread-4

Exiting Thread-5

Exiting Thread-1

Exiting Thread-3

Exiting Thread-2

--- Total time: 8.803994178771973 seconds ---

In [28]:

importlib.reload(confidence\_analyzer)correct\_thresh, wrong\_thresh = 0.3, 0.7correct\_with\_low\_conf\_list, incorrect\_with\_high\_conf\_list = confidence\_analyzer.abnormal\_conf(

full\_results, correct\_thresh, wrong\_thresh)

In [29]:

if len(correct\_with\_low\_conf\_list) > 0:

display(Markdown("#### Examples correctedly predicted with low confidence"))

with pd.option\_context('max\_colwidth', 250):

display(HTML(correct\_with\_low\_conf\_list.to\_html(index=False)))

In [30]:

if len(incorrect\_with\_high\_conf\_list) > 0:

display(Markdown("#### Examples incorrectedly predicted with high confidence"))

with pd.option\_context('max\_colwidth', 250):

display(HTML(incorrect\_with\_high\_conf\_list.to\_html(index=False)))

Examples incorrectedly predicted with high confidence

In [31]:

importlib.reload(entity\_analyzer)importlib.reload(inferencer)if entities\_list:

THREAD\_NUM = 5# we allow a maximum of 5 threads for faster inference

train\_full\_results = inferencer.inference(conversation,

workspace\_id,

workspace\_pd,

max\_retries=10,

max\_thread=THREAD\_NUM,

verbose=False)

entity\_label\_correlation\_df = entity\_analyzer.entity\_label\_correlation\_analysis(

train\_full\_results, entities\_list)

with pd.option\_context('display.max\_colwidth', 200):

entity\_label\_correlation\_df.index = np.arange(1, len(entity\_label\_correlation\_df) + 1)

display(entity\_label\_correlation\_df)else:

display(Markdown("### Target workspace has no entities."))

Starting Thread-1Starting Thread-2

Starting Thread-3

Starting Thread-4

Starting Thread-5

Exiting Thread-2

Exiting Thread-4

Exiting Thread-5

Exiting Thread-3

Exiting Thread-1

--- Total time: 26.995890140533447 seconds ---