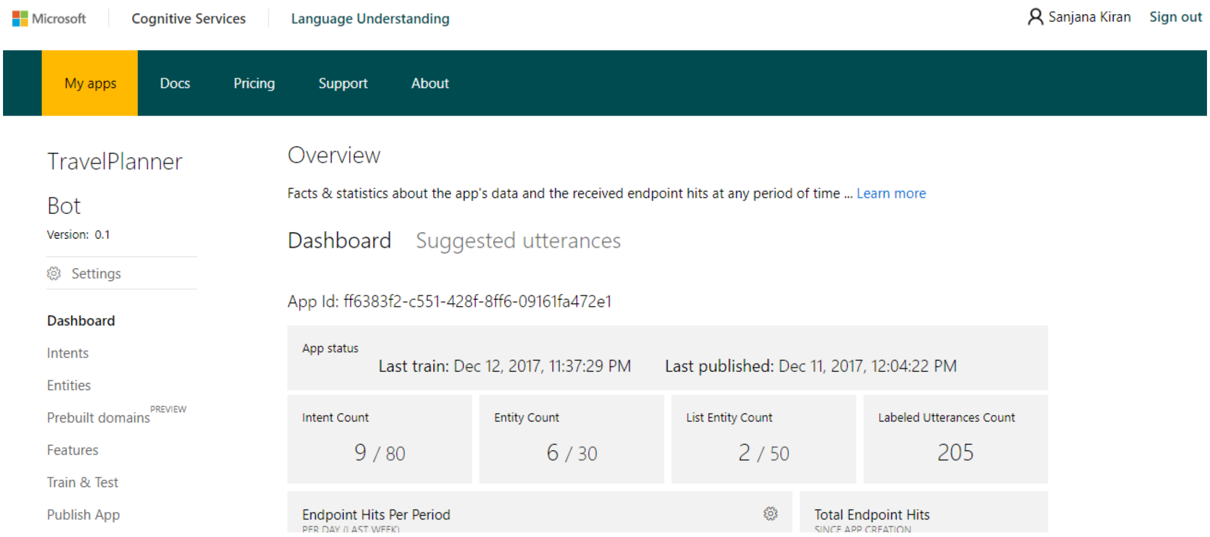
Name-Sanjana Kiran
SUID-310810952

NLP Project Report

Half a chatbot – TravelPlanner Bot
LUIS part-

I have decided to make chatbot which helps users to plan their travel in day to day life and also plan travels ahead.

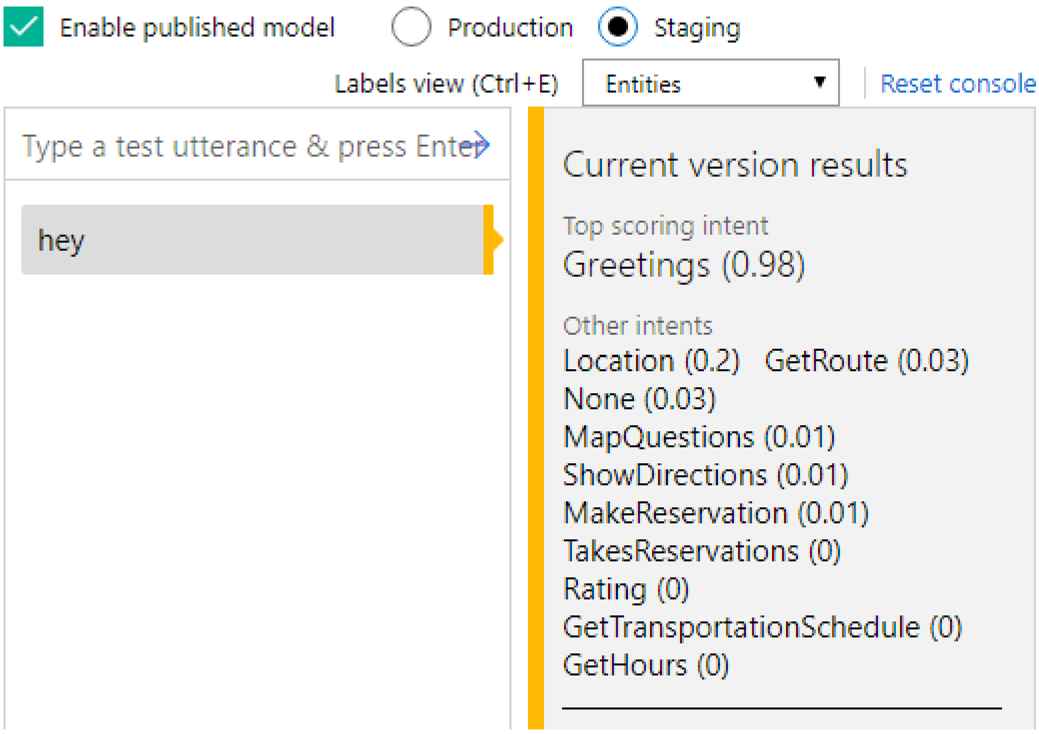
General information about the bot, Number of
Intents 9
Utterances 205
Entities 6 simple entities and 2 list entities
Features 3
URL
https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/ff6383f2-c551-428f-8ff6-09161fa472e1?subscription-key=c6d90ac187dd4d2fa0cd45802a7d14e9&staging=true&spellCheck=true&bing-spell-check-subscription-key={YOUR_BING_KEY_HERE}&verbose=true&timezoneOffset=0&q=



The model is trained and tested. Even published
Trained the model by adding more utterances as possible.

1. Greetings- Considered this as one of the intent so user can have friendly interaction with the bot.
Added 14 utterances. Some of them are hi, hello, thank you, nice talking to you, see you soon etc.

Name-Sanjana Kiran
SUID-310810952

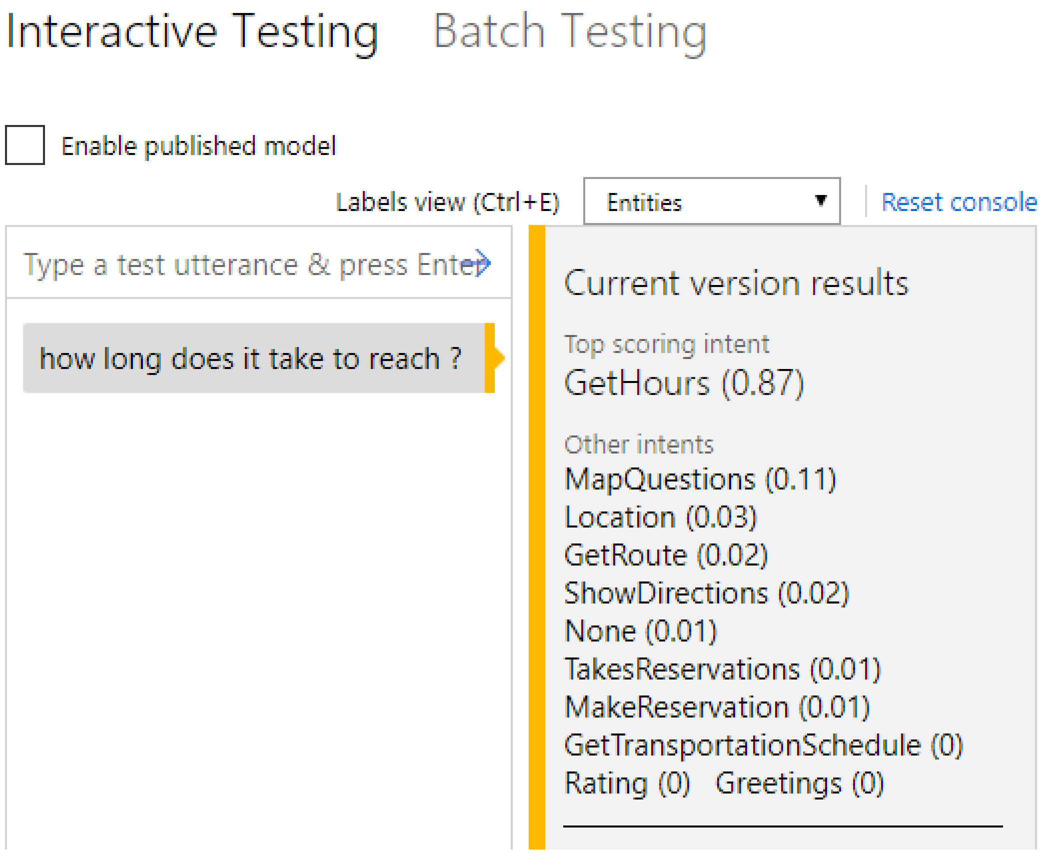


Thus if you give test utterance as “Good Morning”, it classifies it as Greeting and scores it.

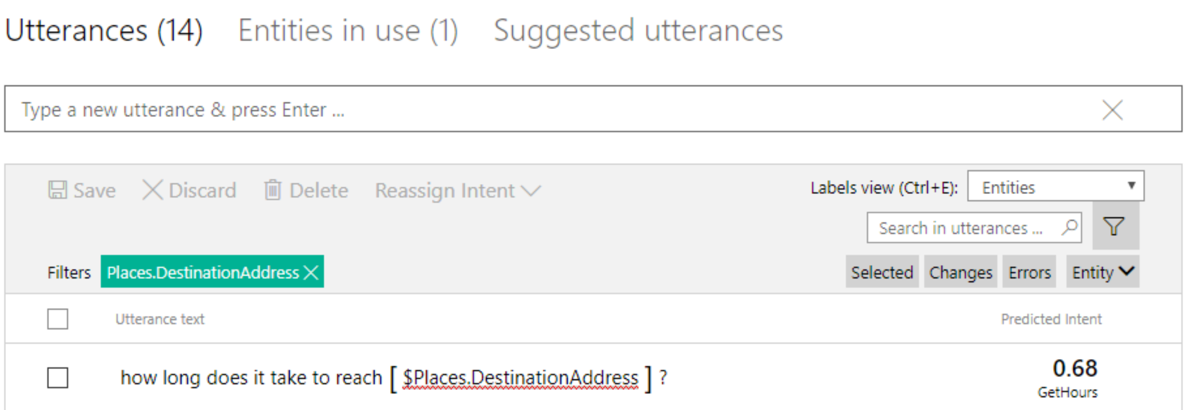
2. GetHours-

Considered this as one the intent so that user can get the hours and plan accordingly.
Added 14 utterances. Some of them are-
How long does it take?
How long does it take to reach NYC?
Tell me the Petco Lynnwood hours
When does the Kirkland Costco close today
Is target open?
How late is taco bell open tonight?

Name-Sanjana Kiran
SUID-310810952



Entity **Place.DestinationAddress** is used under this intent. Example is shown in screenshot. For example - how long does it take to reach office?, here office is considered as that entity.



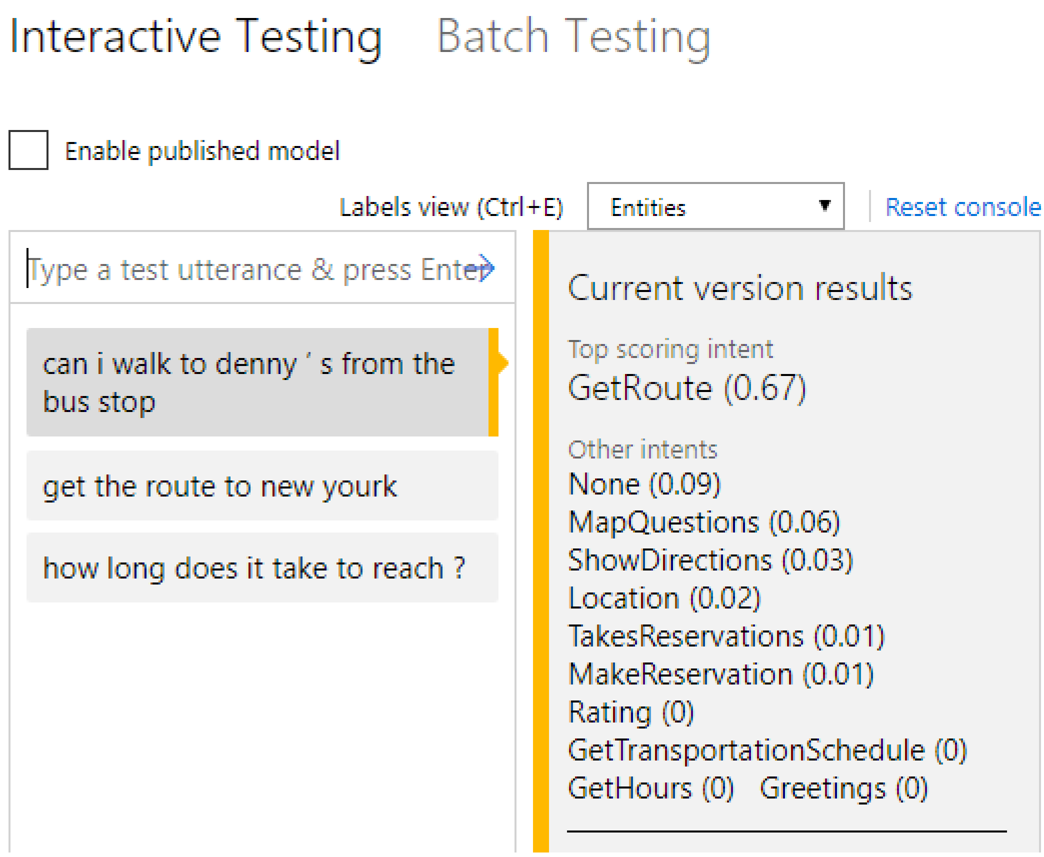
Added the phrase list to increase the accuracy of the model
Hours → timings, open hours, close hours

Thus if you give test utterance like "how many hours does it take to reach school?" it classifies it as GetHours and scores it.

```
{
  "query": " how many hours does it take to reach school",
  "topScoringIntent": {
    "intent": "GetHours",
    "score": 0.9373389
  },
```


3. GetRoute-

Considered this as one of the intent so users can get the navigation details.
Added 22 utterances. Some of them are-
Get the route to quilala
Can I walk to denny's from the bus stop
Guide me to the cheapest gas within 5 miles
From lax to Disneyland please
Find directions from work to home



Under this intent used 3 entities

Places.PreferredRoute – labelled in 7 utterances. when the user want the some specific routes like short route, route to avoid traffic, highways, fastest route etc

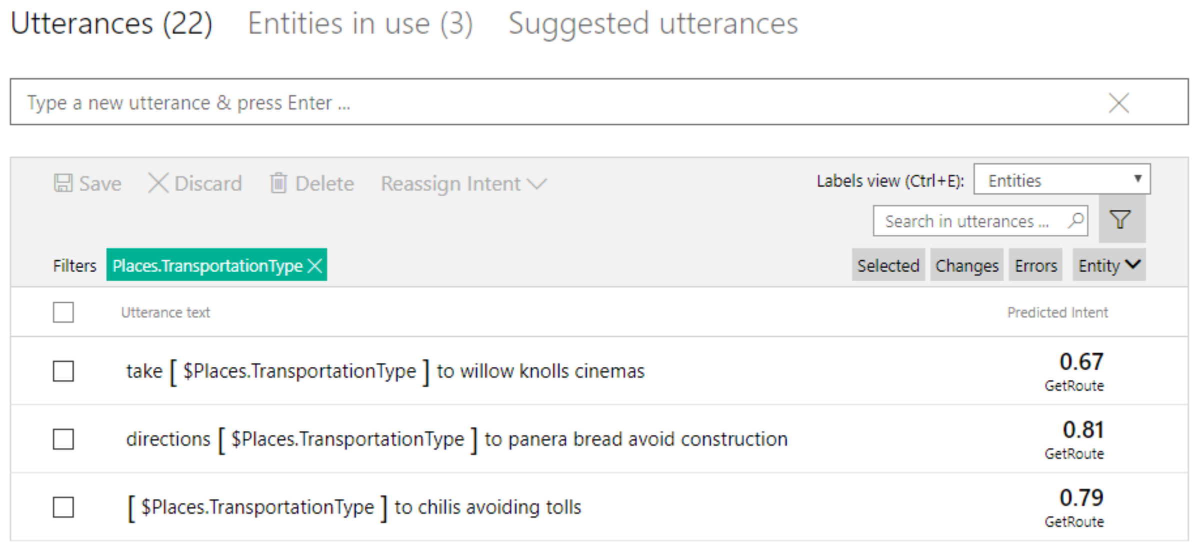
Some of the utterances are like what's shortest route to reach NYC, here the shortest is considered as Places.PreferredRoute

Places.TransportationType- labelled in 3 utterances. transportation types like bus, subway, train, rental cars are considered. Utterances like- I prefer train from port authority to NJ

Name-Sanjana Kiran
SUID-310810952

Places.DestinationAddress- labelled in 2 utterances. The end place the user wants to reach is considered as Places.DestinationAddress entity. Utterance like- how to get home, here home is considered as this entity.

One sample screenshot of the entity, in particular transportation type-



Added phrase list to increase the accuracy route→ way, direction

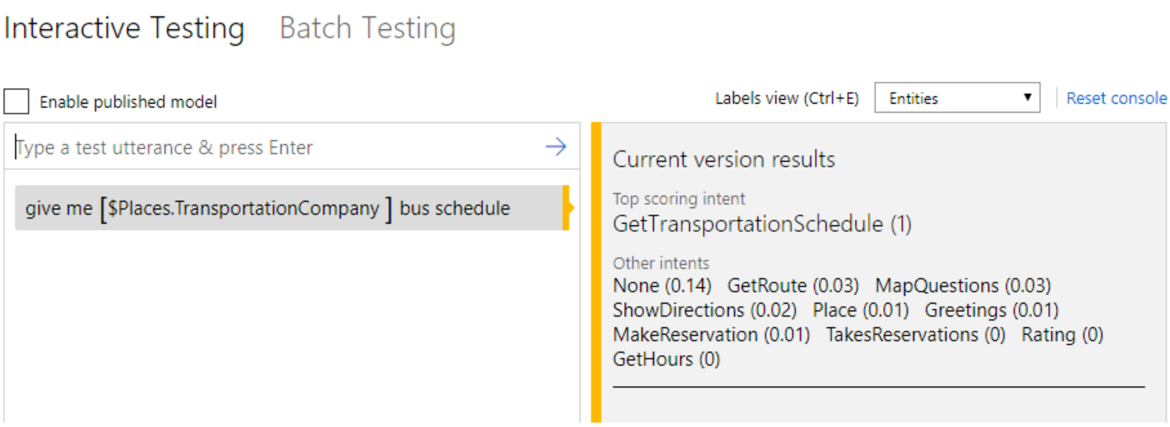
Thus if u give test utterance like “give me less traffic route to office” is classifies as GetRoute intent with score


```
{
  "query": "give me less traffic route to offfice",
  "topScoringIntent": {
    "intent": "GetRoute",
    "score": 0.908317
  },
```

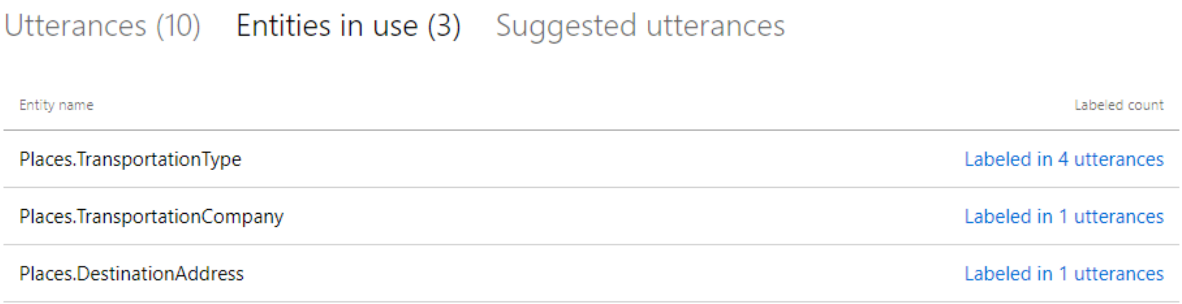

4.GetTransportationSchedule

Considered this intent to help user to get the schedule of flight, bus etc
Added 10 utterances. Some of them are-
Tell me the flight for Thursday
Is it better to take 234 bus to Kirkland
Give me centro bus schedule
What is the train schedule today in Dawsonville
Tell me the bus schedule to Kirkland

Name-Sanjana Kiran
SUID-310810952



Under this intent used 3 entities. They are-



Entity name	Labeled count
Places.TransportationType	Labeled in 4 utterances
Places.TransportationCompany	Labeled in 1 utterances
Places.DestinationAddress	Labeled in 1 utterances

Places.TransportationType- labelled in 3 utterances. transportation types like bus, subway, train, rental cars are considered. Utterances like drumlins Syracuse bus schedule please. Here bus the Places.TransportationType entity

Places.TransportationCompany- companies like Port Authority, Centro etc are considered. utterance like- give me centro bus schedule, here centro is considered as Places.TransportationCompany entity

Places.DestinationAddress- utterance like- Subway options from space needle to NYC, here NYC is considered as Paces.DestinationAddress entity

Thus if you give test utterance like “what is the Port Authority bus schedule for today?”, it classifies as GetTransportationSchedule intent with score

```
{
  "query": "what is the port authority bus schedule for today?",
  "topScoringIntent": {
    "intent": "GetTransportationSchedule",
    "score": 0.9728889
  },
```

5. MakeReservation-

Considered this as intent as planner has to have reservations

Added 8 utterances, some of them are-

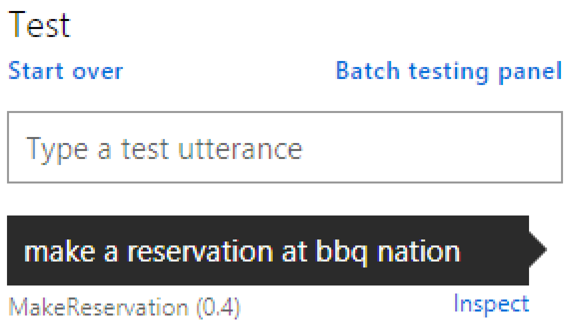
Make a reservation at papa john’s for 2

Reservations at Hilton paris please

Set appointment at Manchester

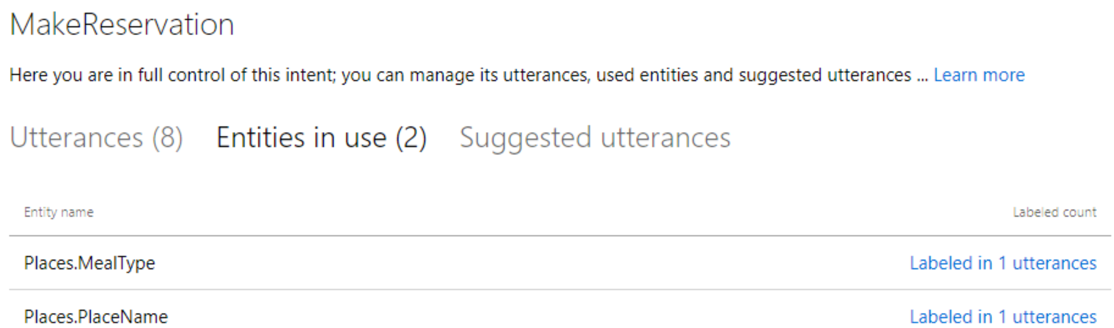
Make reservation of room with sea view

Name-Sanjana Kiran
SUID-310810952



Note: Homepage of LUIS was updated so display of the bot has changed

Under this intent used 2 entities they are-
Places.Mealtype- utterances like book chicken wings at kfc at 8 pm. Here chicken wings is considered as Place.MealType.
PlaceName- utterance like Set appointment at Manchester. Here Manchester is considered as Placename entity



Added a phrase list to increase the accuracy
Reservation→ book, reserve, order

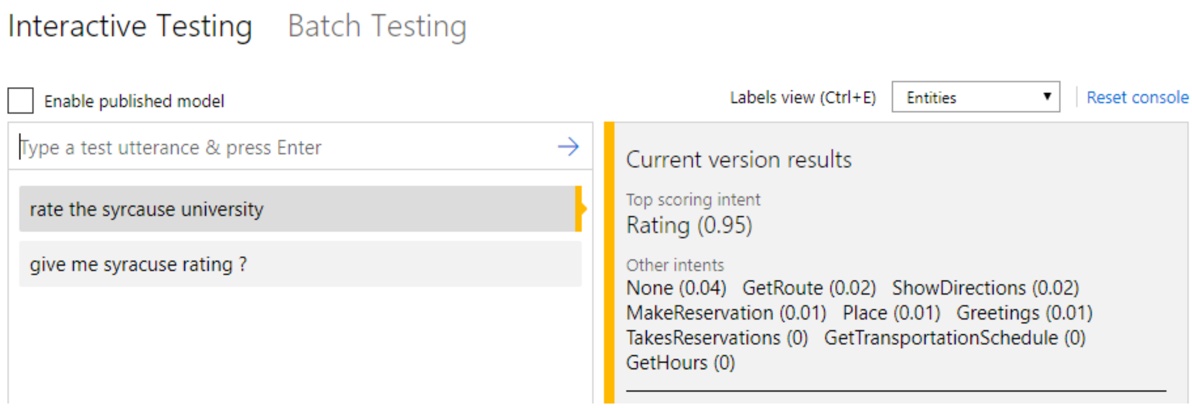
Thus if you give test utterance like "do I have reservation for 2", its classified as MakeReservation with high score


```
{
  "query": "do i have reservation for 2",
  "topScoringIntent": {
    "intent": "MakeReservation",
    "score": 0.2960027
  },
  "intents": [
    {
      "intent": "MakeReservation",
      "score": 0.2960027
    },
```

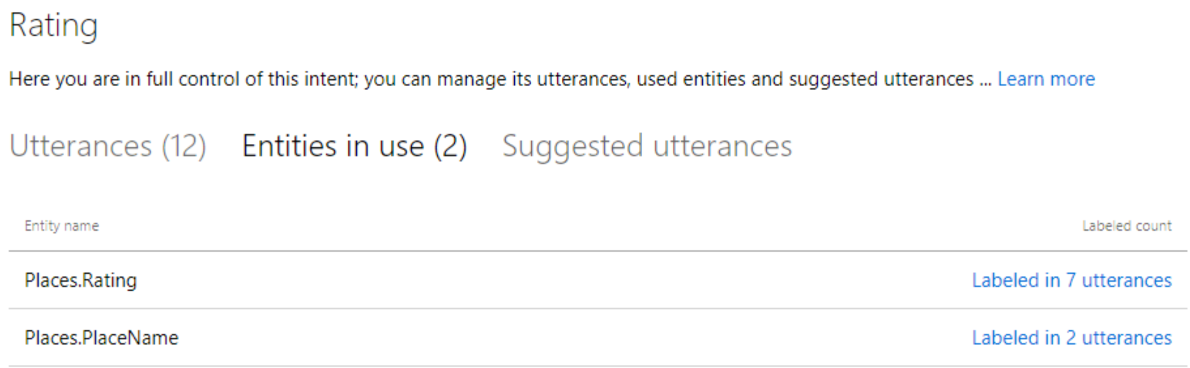

6. Ratings-
Considered this as an entity because as a user wants to check the ratings of the place or restaurant before checking it out to have good experience.
Added 12 utterances under this. Some of them are-
Rate tgif's Friday
Rate Walmart on yelp.com
What's the new restaurant's rating?

Name-Sanjana Kiran
SUID-310810952

Rate old time buffet
Post applebees rating 5 on yelp



Under this intent 2 entities are used and they are-



Place.Rating
Generally the range is defined to rate the place or restaurants. I have considered 5 stars as the range. Example of an utterance with this entity is-
That was good. Rate the restaurant with 3 stars. Here 3 stars is considered as Place.Rating entity
Its just the old users feedbacks sharing to new users.

Places.PlaceName-here utterances with places rating is considered. Example, Please rate NYC, here NYC is entity.

Added a phrase list to increase the accuracy
Rating→ review, feedback, insights

Thus to check the rating of place before visiting it can be done, give test utterance as "what is the rating of Syracuse university"

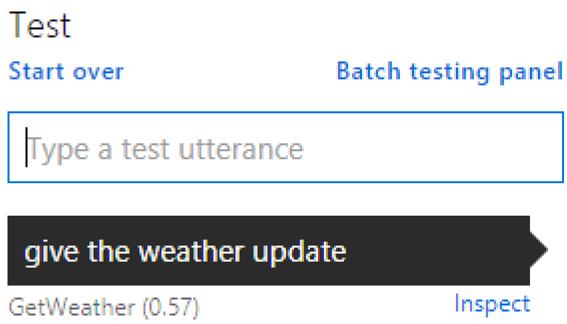

```
{
  "query": "what is rating of syracuse university",
  "topScoringIntent": {
    "intent": "Rating",
    "score": 0.20919463
  },
```

Name-Sanjana Kiran
SUID-310810952

7. GetWeather-

Considered this as one of the intent as its important to check weather before planning anything. For instance, in places like Syracuse its so mandatory to check it daily (with user experience adding this as intent.)

Added 10 utterances. Some of them are-
How is the climate here?
Texas is so sunny
Is it going to rain/snow today
Give me weather update
Rainy day



Note: Homepage of LUIS was updated so display of the bot has changed

Added phrase list to increase the accuracy of the model
Weather→ climate, temperature

Thus if you give test utterance as “what is the climate in Florida?” its considered as GetWeather with high score

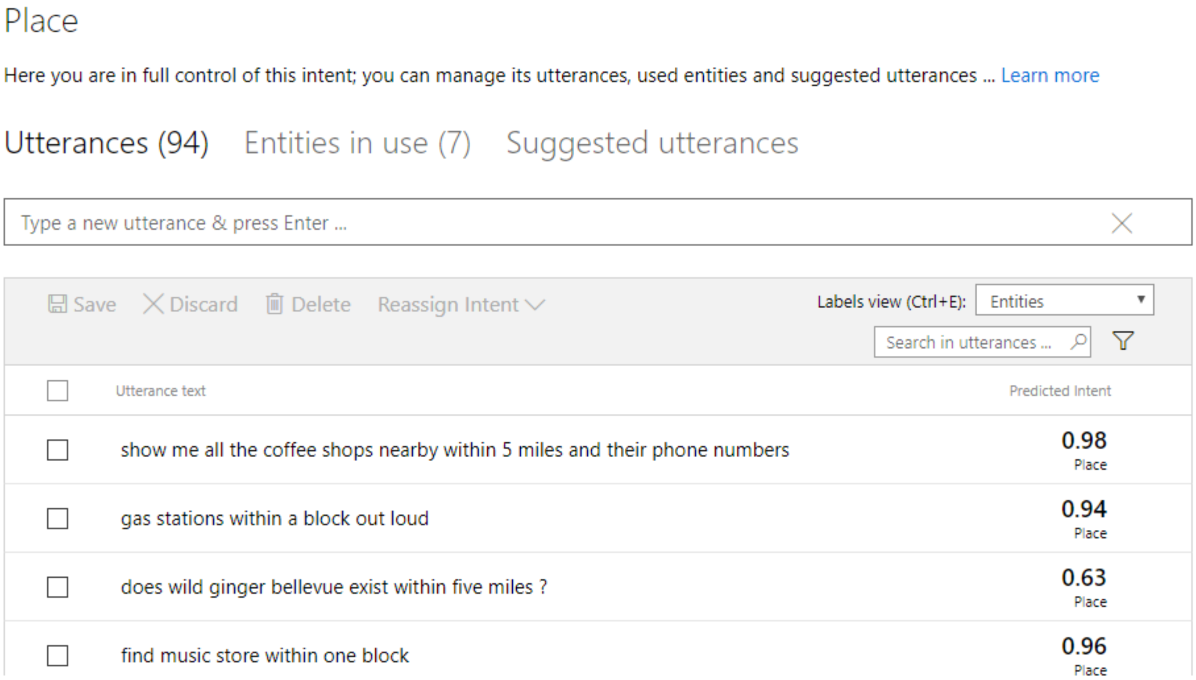

```
{
  "query": "how the climate in  florida?",
  "topScoringIntent": {
    "intent": "GetWeather",
    "score": 0.782410145
  },
  "intents": [
    {
      "intent": "GetWeather",
      "score": 0.782410145
    },
    {
      "intent": "Place",
      "score": 0.0193653554
    },
```

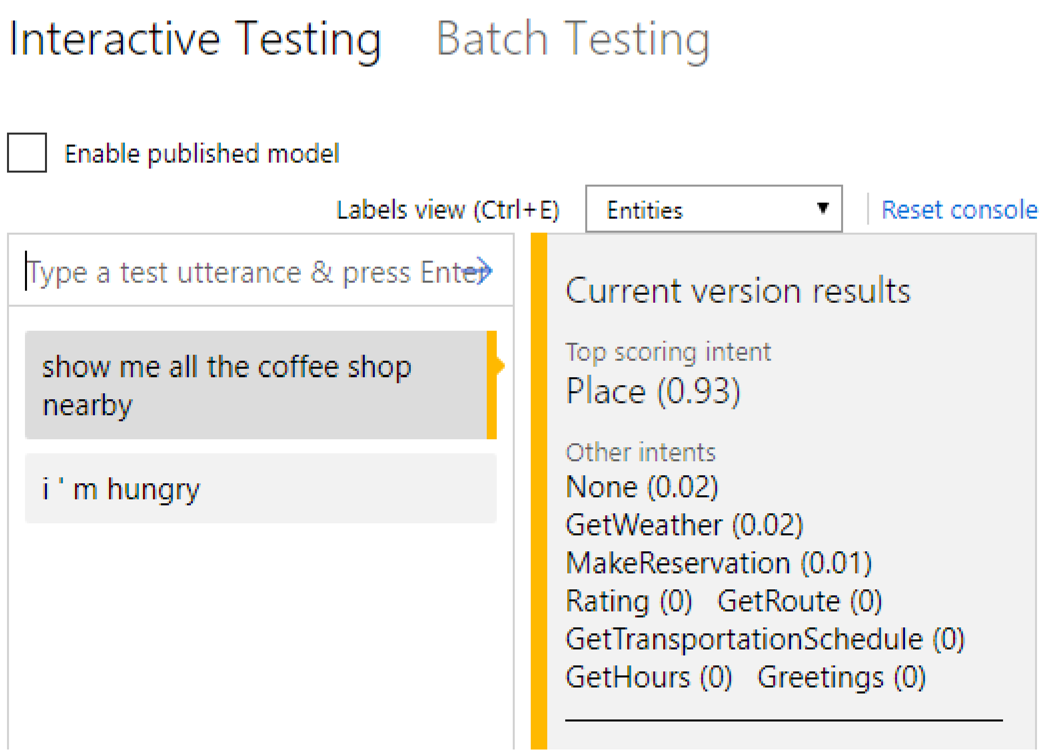

8. Place-

Considered this as one of the important intent. Considered places, location of restaurants, location of specific buildings under this.
Added 94 utterances. Tried to add as many utterances I could
Some of them are-

Name-Sanjana Kiran
SUID-310810952

Give me the list of best family restaurants in nearby location
Show me play areas
How's maimi?
I would love to visit California
Find nearby gas within 15 min
Locate takeout and delivery menu nearby





Under this intent used 4 entities
They are-

Name-Sanjana Kiran
SUID-310810952

Place

Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... Learn more

Utterances (94) Entities in use (4) Suggested utterances

Entity name	Labeled count
Places.MealType	Labeled in 9 utterances
Places.DestinationAddress	Labeled in 2 utterances
Places.PlaceName	Labeled in 2 utterances
Places.TransportationType	Labeled in 1 utterances

Enities in use in this intent
Places.MealType- considered indian, american, Chinese, Mexican, ittalian as meal types.
9 utterances under this intent has this entity some of them are-
Where's good Chinese place? Here Chinese is the entity
Fine me some place to eat Chinese tonight! Here Chinese is the entity
Locate best Italian pizza place to dine. here Italian pizza is entity

Places.DestinationAddress- 2 utterences under this intent has this entity. Example-
how's maimi? Here Miami is the entity

Places.PlaceName- 2 utterances under this intent has this entity. Example-
locate NYC please. Here NYC is the entity

Places.TransportationType- 1 utterances under this intent has this entity.
Example- auto parts shops within a 10 min driving car time please. Here driving car is the entity

This if you give test utterance as "show me nearby highly rated restaurants", its classifies as place with high score and ratings as next

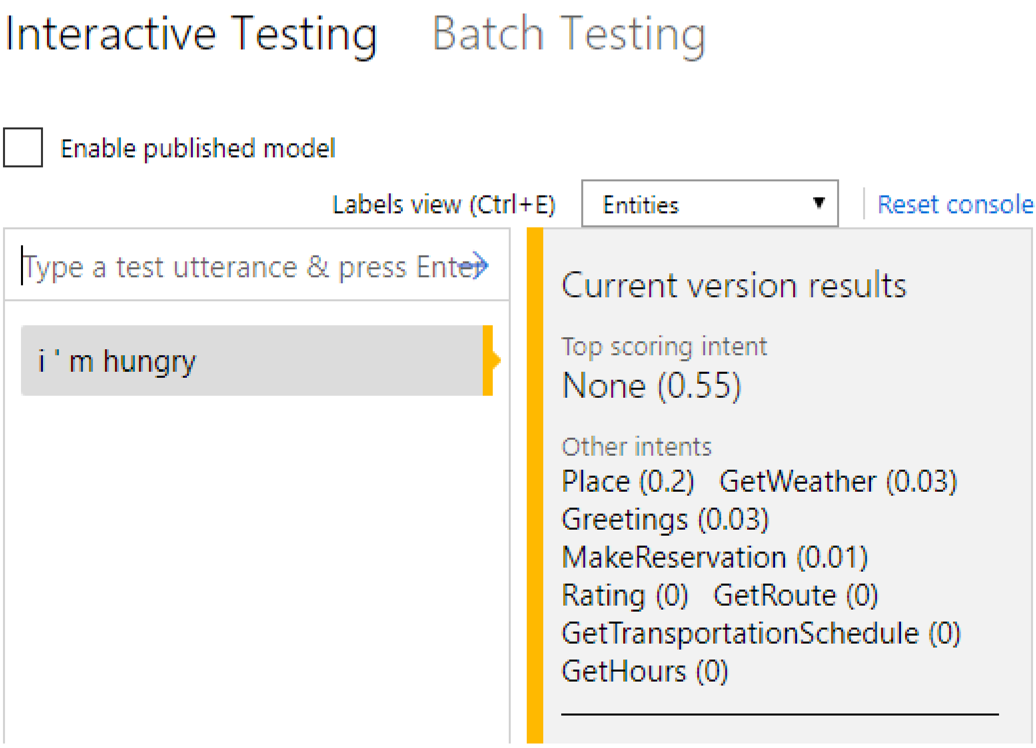

```
{
  "query": "show me the nearby highly rated restaurants",
  "topScoringIntent": {
    "intent": "Place",
    "score": 0.969350159
  },
  "intents": [
    {
      "intent": "Place",
      "score": 0.969350159
    },
    {
      "intent": "Rating",
      "score": 0.0112599125
    },
```


9. None
Some irrelevant questions asked by users are considered none intent
Added 19 utterances under this intent. Some of them are-

Name-Sanjana Kiran
SUID-310810952

Cancel I’m hungry.
I want to get Pepcid coupons
I want to get a Gatorade
I want to catch bus route 221
Etc etc etc

Spell check is enabled, so sentences with wrong spelling are also considered as none intent



Features considered-
Features help you improve your models' (intents and entities) detection and prediction accuracy in utterances. It’s for advanced models.

Name-Sanjana Kiran
SUID-310810952

Create new phrase list

Search phrase lists

Name	Value	Mode	Status
weather	climate,humidity,temperature	Interchangeable	Enabled
reservation	reserved,booked,book,reserve,order	Interchangeable	Enabled
rate	rating,review,feedback,reviews,rated,rati ...	Interchangeable	Enabled
status	approved,accepted,authorized,approves,app ...	Interchangeable	Enabled
route	left,right,this way,wrong direction,go st ...	Interchangeable	Enabled
cuisine	indian,american,european,mexican,canadian ...	Interchangeable	Enabled

Conclusion of TravelPlanner Bot-

Trained the model to classify the query asked by the user to one of the 8 intents and MISC ones are classified as 'none' intent.

You can test the model here by adding your question at the end of URL
https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/ff6383f2-c551-428f-8ff6-09161fa472e1?subscription-key=c6d90ac187dd4d2fa0cd45802a7d14e9&staging=true&spellCheck=true&bing-spell-check-subscription-key={YOUR_BING_KEY_HERE}&verbose=true&timezoneOffset=0&q=

Simple Sentiment Classification

Chosen the Movie Reviews Corpus which categorizes each review as positive or negative.
Sentiment classification on sentences in movie_reviews

First one(baseline)

Defined a feature extractor for documents, so the classifier will know which aspects of the data it should pay attention to.

For document topic identification, defined a feature for each word, indicating whether the document contains that word. To limit the number of features that the classifier needs to process, we begin by constructing a list of the 2000 most frequent words in the overall corpus. Then defined a feature extractor that simply checks whether each of these words is present in a given document.

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

Name-Sanjana Kiran
SUID-310810952

return features

- used Naïve Bayes Classifier
- training set is approximately 90% of data
- accuracy of the classifier is 73.7

```
# training using naive Baysian classifier, training set is 90% of data
train_set, test_set = featuresets[1000:], featuresets[:1000]
classifier = nltk.NaiveBayesClassifier.train(train_set)

# evaluate the accuracy of the classifier
nltk.classify.accuracy(classifier, test_set)
```

```
0.737
```

- The evaluation measures showing performance of classifier

```
# evaluation measures showing performance of classifier
from nltk.metrics import *
reflist = []
testlist = []
for (features, label) in test_set:
    reflist.append(label)
    testlist.append(classifier.classify(features))

# Confusion matrix gives true positives, false negatives, false positives, and true negatives
#   where we interpret pos as "yes" and neg as "no"
cm = ConfusionMatrix(reflist, testlist)
print(cm)

# define a set of item identifiers that are gold labels and a set of item identifiers that are predicted labels
# this uses index numbers for the labels
refpos = set([i for i,label in enumerate(reflist) if label == 'pos'])
refneg = set([i for i,label in enumerate(reflist) if label == 'neg'])
testpos = set([i for i,label in enumerate(testlist) if label == 'pos'])
testneg = set([i for i,label in enumerate(testlist) if label == 'neg'])

# compute precision, recall and F-measure for each label

def printmeasures(label, refset, testset):
    print(label, 'precision:', precision(refset, testset))
    print(label, 'recall:', recall(refset, testset))
    print(label, 'F-measure:', f_measure(refset, testset))

printmeasures('pos', refpos, testpos)
printmeasures('neg', refneg, testneg)
```

- **Output confusion matrix and precision, recall and F-measure**

Name-Sanjana Kiran
SUID-310810952

```
    |   n   p |
    |   e   o |
    |   g   s |
----+---------+
neg |<372>123 |
pos | 140<365>|
----+---------+
(row = reference; col = test)

pos precision: 0.7479508196721312
pos recall: 0.7227722772277227
pos F-measure: 0.7351460221550856
neg precision: 0.7265625
neg recall: 0.7515151515151515
neg F-measure: 0.7388282025819265
```

Second one

Negation features

One strategy with negation words is to negate the word following the negation word, while other strategies negate all words up to the next punctuation or use syntax to find the scope of the negation.

Here, go through the document words in order adding the word features, but if the word follows a negation words, change the feature to negated word.

Start the feature set with all 2000 word features and 2000 Not word features set to false. If a negation occurs, add the following words as a Not word feature (if it's in the top 1000 feature words), and otherwise add it as a regular feature word.

Negation words considered

```
# this list of negation words includes some "approximate negators"
negationwords = ['no', 'not', 'never', 'none', 'nowhere', 'nothing', 'noone', 'rather',
           'hardly', 'scarcely', 'rarely', 'seldom', 'neither', 'nor']

def NOT_features(document, word_features, negationwords):
  features = {}
  for word in word_features:
    features['contains({})'.format(word)] = False
    features['contains(NOT{})'.format(word)] = False
  # go through document words in order
  for i in range(0, len(document)):
    word = document[i]
    if ((i + 1) < len(document)) and ((word in negationwords) or (word.endswith("n't"))):
      i += 1
      features['contains(NOT{})'.format(document[i])] = (document[i] in word_features)
    else:
      features['contains({})'.format(word)] = (word in word_features)
  return features
```

- used Naïve Bayes Classifier

Name-Sanjana Kiran
SUID-310810952

- training set is approximately 90% of data
- accuracy of the classifier is 77.2

```
train_set, test_set = NOT_featuresets[1000:], NOT_featuresets[:1000]
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, test_set)
```

```
0.777
```

```
# evaluation measures showing performance of classifier
from nltk.metrics import *
reflist = []
testlist = []
for (features, label) in test_set:
    reflist.append(label)
    testlist.append(classifier.classify(features))

# Confusion matrix gives true positives, false negatives, false positives, and true negatives
#   where we interpret pos as "yes" and neg as "no"
cm = ConfusionMatrix(reflist, testlist)
print(cm)

# define a set of item identifiers that are gold labels and a set of item identifiers that are predicted labels
# this uses index numbers for the labels
refpos = set([i for i,label in enumerate(reflist) if label == 'pos'])
refneg = set([i for i,label in enumerate(reflist) if label == 'neg'])
testpos = set([i for i,label in enumerate(testlist) if label == 'pos'])
testneg = set([i for i,label in enumerate(testlist) if label == 'neg'])

# compute precision, recall and F-measure for each label

def printmeasures(label, refset, testset):
    print(label, 'precision:', precision(refset, testset))
    print(label, 'recall:', recall(refset, testset))
    print(label, 'F-measure:', f_measure(refset, testset))

printmeasures('pos', refpos, testpos)
printmeasures('neg', refneg, testneg)
```

- **output confusion matrix and precision, recall and F-measure are**

```
    |   n   p |
:k to scroll output; double click to hide
    |   g   s |
----+---------+
neg |<399>117 |
pos | 111<373>|
----+---------+
(row = reference; col = test)

pos precision: 0.7612244897959184
pos recall: 0.7706611570247934
pos F-measure: 0.7659137577002053
neg precision: 0.7823529411764706
neg recall: 0.7732558139534884
neg F-measure: 0.7777777777777779
```

Third one

One more source of features often used in sentiment sentence-level classifications is bigram features.

We'll start by importing the collocations package and creating a short cut variable name for the bigram association measures.

Name-Sanjana Kiran
SUID-310810952

```
from nltk.collocations import *
bigram_measures = nltk.collocations.BigramAssocMeasures()
```

We create a bigram collocation finder using the original movie review words, since the bigram finder must have the words in order.

```
finder = BigramCollocationFinder.from_words(all_words_list)
```

The chi-squared measure to get bigrams that are informative features.

```
bigram_features = finder.nbest(bigram_measures.chi_sq, 500)
```

Feature extraction function that has all the word features as before, but also has bigram features

```
def bigram_document_features(document, word_features, bigram_features):
  document_words = set(document)
  document_bigrams = nltk.bigrams(document)
  features = {}
  for word in word_features:
    features['contains({})'.format(word)] = (word in document_words)
  for bigram in bigram_features:
    features['bigram({} {})'.format(bigram[0], bigram[1])] = (bigram in document_bigrams)
  return features
```

- used Naïve Bayes Classifier
- training set is approximately 90% of data
- accuracy of the classifier is 72.2

```
# train a classifier and report accuracy
train_set, test_set = bigram_featuresets[1000:], bigram_featuresets[:1000]
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, test_set)
```

```
0.738
```

Name-Sanjana Kiran
SUID-310810952

```
# evaluation measures showing performance of classifier
from nltk.metrics import *
reflist = []
testlist = []
for (features, label) in test_set:
    reflist.append(label)
    testlist.append(classifier.classify(features))

# Confusion matrix gives true positives, false negatives, false positives, and true negatives
#   where we interpret pos as "yes" and neg as "no"
cm = ConfusionMatrix(reflist, testlist)
print(cm)

# define a set of item identifiers that are gold labels and a set of item identifiers that are predicted labels
# this uses index numbers for the labels
refpos = set([i for i,label in enumerate(reflist) if label == 'pos'])
refneg = set([i for i,label in enumerate(reflist) if label == 'neg'])
testpos = set([i for i,label in enumerate(testlist) if label == 'pos'])
testneg = set([i for i,label in enumerate(testlist) if label == 'neg'])

# compute precision, recall and F-measure for each label

def printmeasures(label, refset, testset):
    print(label, 'precision:', precision(refset, testset))
    print(label, 'recall:', recall(refset, testset))
    print(label, 'F-measure:', f_measure(refset, testset))

printmeasures('pos', refpos, testpos)
printmeasures('neg', refneg, testneg)
```

output confusion matrix and precision, recall and F-measure are

```
    |  n   p |
lick to expand output; double click to hide output
    |  g   s |
----+---------+
neg |<371>124 |
pos | 138<367>|
----+---------+
(row = reference; col = test)

pos precision: 0.7474541751527495
pos recall: 0.7267326732673267
pos F-measure: 0.7369477911646586
neg precision: 0.7288801571709234
neg recall: 0.7494949494949495
neg F-measure: 0.7390438247011953
```

Fourth one

POS Features

There are some classification tasks where part-of-speech tag features can have an effect. This is more likely for shorter units of classification, such as sentence level classification or shorter social media such as tweets.

The common way to use POS tagging information is to include counts of various types of word tags. This feature function counts nouns, verbs, adjectives and adverbs for features.

```
def POS_features(document, word_features):
   document_words = set(document)
   tagged_words = nltk.pos_tag(document)
   features = {}
   for word in word_features:
      features['contains({})'.format(word)] = (word in document_words)
```

Name-Sanjana Kiran
SUID-310810952

```
numNoun = 0
numVerb = 0
numAdj = 0
numAdverb = 0
for (word, tag) in tagged_words:
   if tag.startswith('N'): numNoun += 1
   if tag.startswith('V'): numVerb += 1
   if tag.startswith('J'): numAdj += 1
   if tag.startswith('R'): numAdverb += 1
features['nouns'] = numNoun
features['verbs'] = numVerb
features['adjectives'] = numAdj
features['adverbs'] = numAdverb
return features
```

- used Naïve Bayes Classifier
- training set is approximately 90% of data
- accuracy of the classifier is 72.8

```
# train and test the classifier
train_set, test_set = POS_featuresets[1000:], POS_featuresets[:1000]
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, test_set)
```

```
0.725
```

```
# evaluation measures showing performance of classifier
from nltk.metrics import *
reflist = []
testlist = []
for (features, label) in test_set:
    reflist.append(label)
    testlist.append(classifier.classify(features))

# Confusion matrix gives true positives, false negatives, false positives, and true negatives
#   where we interpret pos as "yes" and neg as "no"
cm = ConfusionMatrix(reflist, testlist)
print(cm)

# define a set of item identifiers that are gold labels and a set of item identifiers that are predicted labels
# this uses index numbers for the labels
refpos = set([i for i,label in enumerate(reflist) if label == 'pos'])
refneg = set([i for i,label in enumerate(reflist) if label == 'neg'])
testpos = set([i for i,label in enumerate(testlist) if label == 'pos'])
testneg = set([i for i,label in enumerate(testlist) if label == 'neg'])

# compute precision, recall and F-measure for each label

def printmeasures(label, refset, testset):
    print(label, 'precision:', precision(refset, testset))
    print(label, 'recall:', recall(refset, testset))
    print(label, 'F-measure:', f_measure(refset, testset))

printmeasures('pos', refpos, testpos)
printmeasures('neg', refneg, testneg)
```

The output confusion matrix and precision, recall and F-measure

```
    |    n    p |
click to expand output; double click to hide output
    |    g    s |
----+-----------+
neg |<365>130 |
pos | 145<360>|
----+-----------+
(row = reference; col = test)

pos precision: 0.7346938775510204
pos recall: 0.7128712871287128
pos F-measure: 0.7236180904522613
neg precision: 0.7156862745098039
neg recall: 0.7373737373737373
neg F-measure: 0.7263681592039801
```

Fifth one

Combining Bigrams and POS features

Feature extraction function gives bigram features and counts nouns, verbs, adjectives and adverbs for features.

```
def Combined_document_features(document, word_features, bigram_features):
  document_words = set(document)
  document_bigrams = nltk.bigrams(document)
  tagged_words = nltk.pos_tag(document)
  features = {}
  numNoun = 0
  numVerb = 0
  numAdj = 0
  numAdverb = 0
  for word in word_features:
    features['contains({})'.format(word)] = (word in document_words)
  for bigram in bigram_features:
    features['bigram({} {})'.format(bigram[0], bigram[1])] = (bigram in document_bigrams)
  for (word, tag) in tagged_words:
    if tag.startswith('N'): numNoun += 1
    if tag.startswith('V'): numVerb += 1
    if tag.startswith('J'): numAdj += 1
    if tag.startswith('R'): numAdverb += 1
  features['nouns'] = numNoun
  features['verbs'] = numVerb
  features['adjectives'] = numAdj
  features['adverbs'] = numAdverb
  return features
```

- used Naïve Bayes Classifier
- training set is approximately 90% of data

Name-Sanjana Kiran
SUID-310810952

- accuracy of the classifier is 72.8

```
# train and test the classifier
train_set, test_set = comb_featuresets[1000:], comb_featuresets[:1000]
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, test_set)
```

```
0.726
```

```
# evaluation measures showing performance of classifier
from nltk.metrics import *
reflist = []
testlist = []
for (features, label) in test_set:
    reflist.append(label)
    testlist.append(classifier.classify(features))

# Confusion matrix gives true positives, false negatives, false positives, and true negatives
#   where we interpret pos as "yes" and neg as "no"
cm = ConfusionMatrix(reflist, testlist)
print(cm)

# define a set of item identifiers that are gold labels and a set of item identifiers that are predicted labels
# this uses index numbers for the labels
refpos = set([i for i,label in enumerate(reflist) if label == 'pos'])
refneg = set([i for i,label in enumerate(reflist) if label == 'neg'])
testpos = set([i for i,label in enumerate(testlist) if label == 'pos'])
testneg = set([i for i,label in enumerate(testlist) if label == 'neg'])

# compute precision, recall and F-measure for each label

def printmeasures(label, refset, testset):
    print(label, 'precision:', precision(refset, testset))
    print(label, 'recall:', recall(refset, testset))
    print(label, 'F-measure:', f_measure(refset, testset))

printmeasures('pos', refpos, testpos)
printmeasures('neg', refneg, testneg)
```

The output confusion matrix and precision, recall and F-measure

```
    |  n   p |
click to expand output; double click to hide output
    |  g   s |
----+---------+
neg |<366>129 |
pos | 145<360>|
----+---------+
(row = reference; col = test)

pos precision: 0.7361963190184049
pos recall: 0.7128712871287128
pos F-measure: 0.7243460764587525
neg precision: 0.7162426614481409
neg recall: 0.7393939393939394
neg F-measure: 0.7276341948310139
```

Conclusion of classification experiments-

1. Baseline and 4 other experiments are performed
2. Accuracy of the baseline and 4 other experiments are calculated

Experiment	**Accuracy**
Baseline	73.7
Negation Features	77.7

Bigram Features	73.8
POS Features	72.5
Combined Features	72.6

3. To evaluate the performance of the classifier Precision, Recall and F-measure are calculated
4. The Confusion Matrix is printed for each experiment