# tokenize the pattern - split words into array

sentence\_words = nltk.word\_tokenize(sentence)

# stem each word - create short form for word

sentence\_words = [lemmatizer.lemmatize(word.lower()) for word in sentence\_words]

return sentence\_words

# return bag of words array: 0 or 1 for each word in the bag that exists in the sentence

def bow(sentence, words, show\_details=True):

# tokenize the pattern

sentence\_words = clean\_up\_sentence(sentence)

# bag of words - matrix of N words, vocabulary matrix

bag = [0]\*len(words)

for s in sentence\_words:

for i,w in enumerate(words):

if w == s:

# assign 1 if current word is in the vocabulary position

bag[i] = 1

if show\_details:

print ("found in bag: %s" % w)

return(np.array(bag))

def predict\_class(sentence, model):

# filter out predictions below a threshold

p = bow(sentence, words,show\_details=False)

res = model.predict(np.array([p]))[0]

ERROR\_THRESHOLD = 0.25

results = [[i,r] for i,r in enumerate(res) if r>ERROR\_THRESHOLD]

# sort by strength of probability

results.sort(key=lambda x: x[1], reverse=True)

return\_list = []

for r in results:

return\_list.append({"intent": classes[r[0]], "probability": str(r[1])})

return return\_list