import nltk  
nltk.download('punkt')  
nltk.download('wordnet')  
from nltk.stem import WordNetLemmatizer  
lemmatizer = WordNetLemmatizer()  
import json  
import pickle  
  
import numpy as np  
from keras.models import Sequential  
from keras.layers import Dense, Activation, Dropout  
from keras.optimizers import SGD  
import random  
  
words=[]  
classes = []  
documents = []  
ignore\_words = ['?', '!']  
data\_file = open('intents.json').read()  
intents = json.loads(data\_file)  
  
  
for intent in intents['intents']:  
 for pattern in intent['patterns']:  
  
 # take each word and tokenize it  
 w = nltk.word\_tokenize(pattern)  
 words.extend(w)  
 # adding documents  
 documents.append((w, intent['tag']))  
  
 # adding classes to our class list  
 if intent['tag'] not in classes:  
 classes.append(intent['tag'])  
  
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore\_words]  
words = sorted(list(set(words)))  
  
classes = sorted(list(set(classes)))  
  
print (len(documents), "documents")  
  
print (len(classes), "classes", classes)  
  
print (len(words), "unique lemmatized words", words)  
  
  
pickle.dump(words,open('words.pkl','wb'))  
pickle.dump(classes,open('classes.pkl','wb'))  
  
# initializing training data  
training = []  
output\_empty = [0] \* len(classes)  
for doc in documents:  
 # initializing bag of words  
 bag = []  
 # list of tokenized words for the pattern  
 pattern\_words = doc[0]  
 # lemmatize each word - create base word, in attempt to represent related words  
 pattern\_words = [lemmatizer.lemmatize(word.lower()) for word in pattern\_words]  
 # create our bag of words array with 1, if word match found in current pattern  
 for w in words:  
 bag.append(1) if w in pattern\_words else bag.append(0)  
  
 # output is a '0' for each tag and '1' for current tag (for each pattern)  
 output\_row = list(output\_empty)  
 output\_row[classes.index(doc[1])] = 1  
  
 training.append([bag, output\_row])  
# shuffle our features and turn into np.array  
random.shuffle(training)  
training = np.array(training)  
# create train and test lists. X - patterns, Y - intents  
train\_x = list(training[:,0])  
train\_y = list(training[:,1])  
print("Training data created")  
  
  
# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd output layer contains number of neurons  
# equal to number of intents to predict output intent with softmax  
model = Sequential()  
model.add(Dense(128, input\_shape=(len(train\_x[0]),), activation='relu'))  
model.add(Dropout(0.5))  
model.add(Dense(64, activation='relu'))  
model.add(Dropout(0.5))  
model.add(Dense(len(train\_y[0]), activation='softmax'))  
  
# Compile model. Stochastic gradient descent with Nesterov accelerated gradient gives good results for this model  
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)  
model.compile(loss='categorical\_crossentropy', optimizer=sgd, metrics=['accuracy'])  
  
#fitting and saving the model  
hist = model.fit(np.array(train\_x), np.array(train\_y), epochs=200, batch\_size=5, verbose=1)  
model.save('chatbot\_model.h5', hist)  
  
print("model created")