```
import os
import nltk
from gensim.models import CoherenceModel
from nltk.collocations import *
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords


# Gensim
import gensim
import gensim.corpora as corpora
from nltk.corpus import wordnet as wn
from nltk.stem.wordnet import WordNetLemmatizer

# Stop words
stop_words = stopwords.words('english')
numbers = [str(i) for i in range(0, 1000)]
stop_words.extend(['.', '-', '--', ':', '_', 'â\200\224', ',', '&', '(', ')', '[', ']', '{'
, '}', '=', '+', '/', ';', '"\"',
                   "\\", '``', '`', '\'', "\'\'", "â\200\231", 'would', 'could', 'must', 'h
ath', 'thou', 'locke', 'emeric'])
stop_words.extend(numbers)


def get_lemma(word):
    lemma = wn.morphy(word)
    if lemma is None:
        return word
    else:
        return lemma


def get_lemma2(word):
    return WordNetLemmatizer().lemmatize(word)


def tokens(text):
    """tokens of a text with stopwords"""
    tokens_1 = word_tokenize(text)
    tokens_2 = [i.lower() for i in tokens_1 if i != 'I']
    # To delete uderscore inside the string
    tokens_3 = []
    for j in tokens_2:
        if j[0] == '_' and j[-1] == '_':
            j = j[1:-1]
        elif j[0] == '_':
            j = j[1:]
        elif j[-1] == '_':
            j = j[:-1]
        tokens_3.append(j)
    tokens_4 = [i for i in tokens_3 if i not in stop_words and len(i) > 3]
    tokens_final = [get_lemma(token) for token in tokens_4]
    return tokens_final

########################" GENSIM MODEL ############################

def make_data_model(file_path):
    text_data = []
    for file in os.listdir(file_path):
        with open(file_path + file, 'r') as f:
            lines = f.readlines()
            string = ''
            for i in lines:
                string += i
            t = tokens(string)
            text_data.append(t)
    return text_data
```

```
def make_lda_model(token_list, num_topics, passes=10):
    """
    Function to generate LDA models
    Args:
        token_list: a list of lists, one list for each docs
        num_topics: number of topics
        passes: passes for the LDA model
    Returns:
        dictionary
        corpus: document-term matrix, a list of vectors equal to the number of docs.
        lda_model
    """
    dictionary = corpora.Dictionary(token_list)
    dictionary.filter_extremes(no_below=15, no_above=0.8)
    dictionary.save('dictionary.gensim')

    # doc2bow() converts the dictionary into a bag of words.
    corpus = [dictionary.doc2bow(text) for text in token_list]

    lda_model = gensim.models.ldamodel.LdaModel(corpus, num_topics=num_topics,
                                                id2word=dictionary, passes=passes, per_word
_topics=True)
    lda_model.save('model' + str(num_topics) + '.gensim')  # to save te model
    return lda_model, corpus, dictionary


def model_coherence(token_list, model, dictionary):
    """Compute the Coherence complexity of the lda model"""
    coherence_model_lda = CoherenceModel(model=model, texts=token_list, dictionary=dictiona
ry, coherence='c_v')
    coherence_lda = coherence_model_lda.get_coherence()
    return coherence_lda


def model_perplexity(lda_model, corpus):
    """Compute perplexity of the lda model"""
    # a measure of how good the model is. lower the better.
    return lda_model.log_perplexity(corpus)


def lda_model_topics(lda_model):
    """Return topics identified from the lda model"""
    topics = lda_model.print_topics(num_words=10)
    return topics


def load_lda_model(dict_path, model, token_list):
    """Load Lda model"""
    dictionary = corpora.Dictionary.load(dict_path)
    corpus = [dictionary.doc2bow(text) for text in token_list]
    lda_model = gensim.models.ldamodel.LdaModel.load(model)
    return lda_model, corpus, dictionary


def topic_doc(doc, lda_model, dictionary):
    """find the topic of a text"""
    doc_l = open(doc, 'r')
    lines = doc_l.readlines()
    string = ''
    for i in lines:
        string += i
    current_doc = tokens(string)
    other_corpus = [dictionary.doc2bow(current_doc)]
    unseen_doc = other_corpus[0]
    vector = list(lda_model[unseen_doc])[0]
    best_topic = max(vector, key=lambda item: item[1])[0]
    print(best_topic)
```

```
# path to the texts
path_text = '/home/macaire/Bureau/nlpproject/processed_with_script/'
# path to the dictionary saved
path_model = '/home/macaire/Bureau/nlpproject/topic_modeling/'

if __name__ == "__main__":
    text_data = make_data_model(path_text)
    lda, corpus, dictionary = load_lda_model(path_model+'dictionary.gensim', path_model+'mo
del17.gensim', text_data)
    print('Topics : ', lda_model_topics(lda))
    for filename in os.listdir(path_text):
        topic_doc(path_text+filename, lda, dictionary)
```