**토픽 모델링이란?**

- 단어들은 각각의 토픽을 가지고 있고 문장은 단어들을 가지고 있으며, 문서는 그러한 문장들을 가지고 있다.
- 이에 따라, 단어들의 토픽들을 알게 되면, 해당 문서에는 토픽들의 분포가 형성될 것이며, 크게는 해당 문서의 토픽을 알 수 있게 된다.

20 Newgoups 데이터세트로 토픽 모델링하기

In [1]:

```
import nltk; nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/hskimim/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[1]:

```
True
```

In [32]:

```
import re
import numpy as np
import pandas as pd
from pprint import pprint

# Gensim
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel

# spacy for lemmatization
import spacy

# Plotting tools
import pyLDAvis
import pyLDAvis.gensim  # don't skip this
import matplotlib.pyplot as plt
%matplotlib inline

# Enable logging for gensim - optional
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.ERROR)

import warnings
warnings.filterwarnings("ignore",category=DeprecationWarning)
```

stopwords 에는 불필요한 단어, 즉 조사나 관사들을 없애는 툴이다.

In [5]:

```
# NLTK Stop words
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
stop_words.extend(['from', 'subject', 're', 'edu', 'use'])
```

read_json 메소드로 웹에 있는 json 파일을 가지고 온다.

In [6]:

```
# Import Dataset
df = pd.read_json('https://raw.githubusercontent.com/selva86/datasets/master/newsgroups.json')
print(df.target_names.unique())
df.head()
```

```
['rec.autos' 'comp.sys.mac.hardware' 'rec.motorcycles' 'misc.forsale'
 'comp.os.ms-windows.misc' 'alt.atheism' 'comp.graphics'
 'rec.sport.baseball' 'rec.sport.hockey' 'sci.electronics' 'sci.space'
 'talk.politics.misc' 'sci.med' 'talk.politics.mideast'
 'soc.religion.christian' 'comp.windows.x' 'comp.sys.ibm.pc.hardware'
 'talk.politics.guns' 'talk.religion.misc' 'sci.crypt']
```

Out[6]:

| | content | target | target_names |
|---|---|---|---|
| **0** | From: lerxst@wam.umd.edu (where's my thing)\nS... | 7 | rec.autos |
| **1** | From: guykuo@carson.u.washington.edu (Guy Kuo)... | 4 | comp.sys.mac.hardware |
| **10** | From: irwin@cmptrc.lonestar.org (Irwin Arnstei... | 8 | rec.motorcycles |
| **100** | From: tchen@magnus.acs.ohio-state.edu (Tsung-K... | 6 | misc.forsale |
| **1000** | From: dabl2@nlm.nih.gov (Don A.B. Lindbergh)\n... | 2 | comp.os.ms-windows.misc |

In [7]:

```
# Convert to list
data = df.content.values.tolist()

# Remove Emails
data = [re.sub('\S*@\S*\s?', '', sent) for sent in data]

# Remove new line characters
data = [re.sub('\s+', ' ', sent) for sent in data]

# Remove distracting single quotes
data = [re.sub("\'", "", sent) for sent in data]

pprint(data[:1])
```

```
['From: (wheres my thing) Subject: WHAT car is this!? Nntp-Posting-Host: '
 'rac3.wam.umd.edu Organization: University of Maryland, College Park Lines: '
 '15 I was wondering if anyone out there could enlighten me on this car I saw '
 'the other day. It was a 2-door sports car, looked to be from the late 60s/ '
 'early 70s. It was called a Bricklin. The doors were really small. In '
 'addition, the front bumper was separate from the rest of the body. This is '
 'all I know. If anyone can tellme a model name, engine specs, years of '
 'production, where this car is made, history, or whatever info you have on '
 'this funky looking car, please e-mail. Thanks, - IL ---- brought to you by '
 'your neighborhood Lerxst ---- ']
```

- 정규식 표현을 통해서 문장 내에 이메일과 기타 특수 문자들을 없애주었지만, 여전히 난잡해보인다.
- LDA 알고리즘을 사용하기 위해서는, 문장들을 단어들의 묶음으로 변환시켜주는 과정이 필요하다.
- 이러한 과정을 Tokenization 이라고 한다.

## tokenization process

In [8]:

```
def sent_to_words(sentences):
    for sentence in sentences:
        yield(gensim.utils.simple_preprocess(str(sentence), deacc=True))
        # deacc=True removes punctuations
        # 구두점(말끝에 찍는 쉼표나 점들을 의미) 을 없애주는 것이다.

data_words = list(sent_to_words(data))

print(data_words[:1])
```

```
[['from', 'wheres', 'my', 'thing', 'subject', 'what', 'car', 'is', 'this', 'nntp', 'posting', 'host', 'rac', 'wam', 'umd', 'edu', 'organization', 'university
', 'of', 'maryland', 'college', 'park', 'lines', 'was', 'wondering', 'if', 'anyone', 'out', 'there', 'could', 'enlighten', 'me', 'on', 'this', 'car', 'saw',
'the', 'other', 'day', 'it', 'was', 'door', 'sports', 'car', 'looked', 'to', 'be', 'from', 'the', 'late', 'early', 'it', 'was', 'called', 'bricklin', 'the',
'doors', 'were', 'really', 'small', 'in', 'addition', 'the', 'front', 'bumper', 'was', 'separate', 'from', 'the', 'rest', 'of', 'the', 'body', 'this', 'is',
'all', 'know', 'if', 'anyone', 'can', 'tellme', 'model', 'name', 'engine', 'specs', 'years', 'of', 'production', 'where', 'this', 'car', 'is', 'made', 'histo
ry', 'or', 'whatever', 'info', 'you', 'have', 'on', 'this', 'funky', 'looking', 'car', 'please', 'mail', 'thanks', 'il', 'brought', 'to', 'you', 'by', 'your'
, 'neighborhood', 'lerxst']]
```

### Bigram , Trigram 모델 만들기

- Bigram : 문서에서 함께 자주 등장하는 2개의 단어
- Trigram : 문서에서 함께 자주 등장하는 3개의 단어
- 'front_bumper', 'oil_leak', 'maryland_college_park' etc.
- Phrases : 모델을 빌드한다.
- min_count , threshold : Pharases 의 중요한 두 개의 파라미터
  - min_count (float, optional) : Ignore all words and bigrams with total collected count lower than this value.
  - threshold (float, optional) : Represent a score threshold for forming the phrases (higher means fewer phrases). A phrase of words a followed by b is accepted if the score of the phrase is greater than threshold. Heavily depends on concrete scoring-function, see the scoring parameter.

In [11]:

```
# Build the bigram and trigram models
bigram = gensim.models.Phrases(data_words, min_count=5, threshold=100) # higher threshold fewer phrases.
trigram = gensim.models.Phrases(bigram[data_words], threshold=100)

# Faster way to get a sentence clubbed as a trigram/bigram
bigram_mod = gensim.models.phrases.Phraser(bigram)
trigram_mod = gensim.models.phrases.Phraser(trigram)

# See trigram example
print(trigram_mod[bigram_mod[data_words[0]]])
```

```
/home/hskimim/anaconda3/lib/python3.6/site-packages/gensim/models/phrases.py:598: UserWarning: For a faster implementation, use the gensim.models.phrases.Phr
aser class
  warnings.warn("For a faster implementation, use the gensim.models.phrases.Phraser class")
```

```
['from', 'wheres', 'my', 'thing', 'subject', 'what', 'car', 'is', 'this', 'nntp_posting_host', 'rac_wam_umd_edu', 'organization', 'university', 'of', 'maryla
nd_college_park', 'lines', 'was', 'wondering', 'if', 'anyone', 'out', 'there', 'could', 'enlighten', 'me', 'on', 'this', 'car', 'saw', 'the', 'other', 'day',
 'it', 'was', 'door', 'sports', 'car', 'looked', 'to', 'be', 'from', 'the', 'late', 'early', 'it', 'was', 'called', 'bricklin', 'the', 'doors', 'were', 'real
ly', 'small', 'in', 'addition', 'the', 'front_bumper', 'was', 'separate', 'from', 'the', 'rest', 'of', 'the', 'body', 'this', 'is', 'all', 'know', 'if', 'any
one', 'can', 'tellme', 'model', 'name', 'engine', 'specs', 'years', 'of', 'production', 'where', 'this', 'car', 'is', 'made', 'history', 'or', 'whatever', 'i
nfo', 'you', 'have', 'on', 'this', 'funky', 'looking', 'car', 'please', 'mail', 'thanks', 'il', 'brought', 'to', 'you', 'by', 'your', 'neighborhood', 'lerxst
']
```

In [17]:

```
# Define functions for stopwords, bigrams, trigrams and lemmatization
def remove_stopwords(texts):
    return [[word for word in simple_preprocess(str(doc)) if word not in stop_words] for doc in texts]

def make_bigrams(texts):
    return [bigram_mod[doc] for doc in texts]

def make_trigrams(texts):
    return [trigram_mod[bigram_mod[doc]] for doc in texts]

def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    """https://spacy.io/api/annotation"""
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in allowed_postags])
    return texts_out
```

In [19]:

```
# Remove Stop Words
data_words_nostops = remove_stopwords(data_words)

# Form Bigrams
data_words_bigrams = make_bigrams(data_words_nostops)

# Initialize spacy 'en' model, keeping only tagger component (for efficiency)
# python3 -m spacy download en
nlp = spacy.load('en', disable=['parser', 'ner'])

# Do lemmatization keeping only noun, adj, vb, adv
data_lemmatized = lemmatization(data_words_bigrams, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV'])

print(data_lemmatized[:1])
```

```
[['where', 's', 'thing', 'car', 'nntp_post', 'host', 'rac_wam', 'umd', 'organization', 'university', 'maryland_college', 'park', 'line', 'wonder', 'anyone',
'could', 'enlighten', 'car', 'see', 'day', 'door', 'sport', 'car', 'look', 'late', 'early', 'call', 'bricklin', 'door', 'really', 'small', 'addition', 'front
_bumper', 'separate', 'rest', 'body', 'know', 'anyone', 'tellme', 'model', 'name', 'engine', 'spec', 'year', 'production', 'car', 'make', 'history', 'whatev'
, 'info', 'funky', 'look', 'car', 'mail', 'thank', 'bring', 'neighborhood', 'lerxst']]
```

- LDA 모델에 들어가야 하는 두 가지 입력변수는 딕셔너리와(id2word) 코퍼스(corpus)이다.
- gensim 은 문서 내에 있는 단어별로 유니크한 아이디를 할당해준다.
- 아래의 각각의 엘리먼트 튜플당 의미하는 것은 [word_id,word_frequency] 이다.

In [20]:

```
# Create Dictionary
id2word = corpora.Dictionary(data_lemmatized)

# Create Corpus
texts = data_lemmatized

# Term Document Frequency
corpus = [id2word.doc2bow(text) for text in texts]

# View
print(corpus[:1])
```

```
[[(0, 1), (1, 2), (2, 1), (3, 1), (4, 1), (5, 1), (6, 5), (7, 1), (8, 1), (9, 2), (10, 1), (11, 1), (12, 1), (13, 1), (14, 1), (15, 1), (16, 1), (17, 1), (18
, 1), (19, 1), (20, 1), (21, 1), (22, 2), (23, 1), (24, 1), (25, 1), (26, 1), (27, 1), (28, 1), (29, 1), (30, 1), (31, 1), (32, 1), (33, 1), (34, 1), (35, 1)
, (36, 1), (37, 1), (38, 1), (39, 1), (40, 1), (41, 1), (42, 1), (43, 1), (44, 1), (45, 1), (46, 1), (47, 1), (48, 1), (49, 1), (50, 1)]]
```

만약 해당 id에 속한 단어를 보고싶으면

In [27]:

```
id2word[0]
```

Out[27]:

```
'addition'
```

위의 표는 컴퓨터가 읽기 쉽게끔 만들어준 것이고, Counter 객체처럼 사람이 읽기 쉽게 만든 것은 아래와 같다.

In [28]:

```
# Human readable format of corpus (term-frequency)
[[(id2word[id], freq) for id, freq in cp] for cp in corpus[:1]]
```

Out[28]:

```
[[('addition', 1),
  ('anyone', 2),
  ('body', 1),
  ('bricklin', 1),
  ('bring', 1),
  ('call', 1),
  ('car', 5),
  ('could', 1),
  ('day', 1),
  ('door', 2),
  ('early', 1),
  ('engine', 1),
  ('enlighten', 1),
  ('front_bumper', 1),
  ('funky', 1),
  ('history', 1),
  ('host', 1),
  ('info', 1),
  ('know', 1),
  ('late', 1),
  ('lerxst', 1),
  ('line', 1),
  ('look', 2),
  ('mail', 1),
  ('make', 1),
  ('maryland_college', 1),
  ('model', 1),
  ('name', 1),
  ('neighborhood', 1),
  ('nntp_post', 1),
  ('organization', 1),
  ('park', 1),
  ('production', 1),
  ('rac_wam', 1),
  ('really', 1),
  ('rest', 1),
  ('s', 1),
  ('see', 1),
  ('separate', 1),
  ('small', 1),
  ('spec', 1),
  ('sport', 1),
  ('tellme', 1),
  ('thank', 1),
  ('thing', 1),
  ('umd', 1),
  ('university', 1),
  ('whatev', 1),
  ('where', 1),
  ('wonder', 1),
  ('year', 1)]]
```

여태까지 해온 것이 LDA 모델 생성에 필요한 것들을 전부 한 것이다. 코퍼스와 딕셔너리를 생성한 것에 더해서, 우리는 몇 개의 토픽을 할당할 것인지에 대한 결정을 해주어야 한다.

- alpha , eta 는 토픽들의 떨어진 정도(sparsity)에 영향을 끼치는 하이퍼 파라미터이다. 도큐먼트에 따르면, 디폴트값은 1.0/num_topics prior 이다.
- chunksize 는 각각의 training chunk 에 사용될 문서의 갯수를 의미한다. 확실하지는 않지만, batch_size 와 유사한 의미를 갖는 것으로 해석된다.
  - IN ADDITION : Text chunking, also referred to as shallow parsing, is a task that follows Part-Of-Speech Tagging and that adds more structure to the sentence. The result is a grouping of the words in “chunks”.

In [29]:

```
# Build LDA model
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                            id2word=id2word,
                                            num_topics=20,
                                            random_state=100,
                                            update_every=1,
                                            chunksize=100,
                                            passes=10,
                                            alpha='auto',
                                            per_word_topics=True)
```

위의 모델을 통해 반환되는 것은 토픽의 수는 20개이고 각각의 키워드(단어들의 집합)와 키워드들 간의 조합이 특정한 토픽의 가중치를 정해주는데 기여하는 것이다.

- lda_model 객체에 print_topics 메소드를 operating 하면, 각각의 키워드들이 토픽에 기여하는 가중치(importance)를 알 수 있다.
- 0부터 19까지 총 20개에 해당하는 토픽이 있는 것을 알 수 있고, 각각의 토픽에 위치해있는 키워드들과 이들 키워드들이 해당 토픽에서 가지는 중요도가 순서대로 나와있다.

In [30]:

```
# Print the Keyword in the 10 topics
pprint(lda_model.print_topics())
doc_lda = lda_model[corpus]
```

```
[(0,
  '0.034*"_" + 0.029*"blue" + 0.021*"tank" + 0.014*"cubs_suck" + 0.012*"eg" + '
  '0.012*"gas" + 0.011*"hi" + 0.007*"henry_spencer" + 0.007*"cigarette" + '
  '0.006*"xlib"'),
 (1,
  '0.020*"value" + 0.020*"bus" + 0.019*"specifically" + 0.015*"function" + '
  '0.015*"associate" + 0.013*"motorcycle" + 0.011*"properly" + 0.010*"code" + '
  '0.008*"confuse" + 0.008*"error"'),
 (2,
  '0.023*"window" + 0.020*"card" + 0.017*"file" + 0.014*"drive" + 0.013*"use" '
  '+ 0.013*"system" + 0.011*"problem" + 0.010*"run" + 0.009*"color" + '
  '0.009*"do"'),
 (3,
  '0.035*"game" + 0.033*"team" + 0.019*"player" + 0.017*"play" + 0.017*"win" + '
  '0.016*"hockey" + 0.013*"season" + 0.011*"contact" + 0.011*"year" + '
  '0.010*"nhl"'),
 (4,
  '0.026*"wire" + 0.017*"circuit" + 0.015*"faq" + 0.015*"connect" + '
  '0.013*"wiring" + 0.013*"voice" + 0.013*"cover" + 0.011*"outlet" + '
  '0.011*"neutral" + 0.010*"conference"'),
 (5,
  '0.030*"government" + 0.024*"gun" + 0.022*"law" + 0.017*"state" + '
  '0.015*"right" + 0.013*"public" + 0.012*"protect" + 0.011*"american" + '
  '0.010*"police" + 0.010*"criminal"'),
 (6,
  '0.074*"max" + 0.018*"cost" + 0.015*"price" + 0.014*"year" + 0.012*"sale" + '
  '0.011*"sell" + 0.010*"obvious" + 0.009*"pay" + 0.007*"canada" + '
  '0.007*"total"'),
 (7,
  '0.047*"line" + 0.042*"organization" + 0.031*"write" + 0.026*"article" + '
  '0.019*"would" + 0.018*"university" + 0.017*"nntp_post" + 0.015*"host" + '
  '0.015*"not" + 0.014*"get"'),
 (8,
  '0.019*"science" + 0.014*"computer_science" + 0.012*"prove" + '
  '0.011*"homeopathy" + 0.010*"review" + 0.010*"development" + 0.010*"object" '
  '+ 0.010*"univ" + 0.010*"text" + 0.010*"gordon_bank"'),
 (9,
  '0.039*"space" + 0.011*"power" + 0.009*"launch" + 0.008*"ground" + '
  '0.008*"switch" + 0.008*"build" + 0.007*"project" + 0.007*"high" + '
  '0.007*"radio" + 0.007*"mon"'),
 (10,
  '0.815*"ax" + 0.003*"nyi" + 0.003*"stl" + 0.003*"buffalo" + 0.002*"pool" + '
  '0.002*"brian_kendig" + 0.002*"sunday" + 0.002*"finland" + 0.002*"espn" + '
  '0.002*"lemieux"'),
 (11,
  '0.040*"armenian" + 0.013*"league" + 0.011*"turk" + 0.011*"turkish" + '
  '0.011*"greek" + 0.011*"baseball" + 0.010*"serdar_argic" + 0.007*"road" + '
  '0.006*"cal" + 0.006*"hug"'),
 (12,
  '0.011*"national" + 0.009*"center" + 0.009*"year" + 0.009*"april" + '
  '0.009*"study" + 0.007*"george" + 0.007*"research" + 0.006*"march" + '
  '0.006*"mission" + 0.006*"student"'),
 (13,
  '0.011*"may" + 0.010*"would" + 0.009*"make" + 0.007*"people" + '
  '0.007*"question" + 0.007*"also" + 0.007*"many" + 0.007*"point" + '
  '0.007*"mean" + 0.006*"must"'),
 (14,
  '0.015*"israel" + 0.012*"israeli" + 0.009*"attack" + 0.009*"kill" + '
  '0.009*"military" + 0.008*"hour" + 0.008*"land" + 0.008*"international" + '
  '0.008*"committee" + 0.007*"soldier"'),
 (15,
  '0.020*"mail" + 0.017*"information" + 0.015*"available" + 0.015*"include" + '
  '0.013*"send" + 0.012*"program" + 0.011*"list" + 0.010*"ca" + 0.010*"also" + '
  '0.010*"internet"'),
 (16,
  '0.029*"key" + 0.022*"chip" + 0.014*"use" + 0.013*"system" + 0.011*"phone" + '
  '0.010*"encryption" + 0.010*"bit" + 0.010*"technology" + 0.009*"wiretap" + '
  '0.009*"device"'),
 (17,
  '0.025*"christian" + 0.023*"god" + 0.012*"man" + 0.011*"life" + '
  '0.011*"religion" + 0.010*"bible" + 0.009*"believe" + 0.008*"law" + '
  '0.008*"belief" + 0.008*"die"'),
 (18,
  '0.051*"not" + 0.032*"do" + 0.022*"say" + 0.021*"go" + 0.019*"would" + '
  '0.018*"be" + 0.015*"think" + 0.014*"know" + 0.014*"s" + 0.013*"people"'),
 (19,
  '0.017*"light" + 0.015*"rise" + 0.015*"paul" + 0.012*"fire" + 0.011*"edge" + '
  '0.010*"building" + 0.009*"water" + 0.009*"teach" + 0.008*"mother" + '
  '0.008*"girl"')]
```

In [31]:

```
# Compute Perplexity
print('\nPerplexity: ', lda_model.log_perplexity(corpus))  # a measure of how good the model is. lower the better.

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model, texts=data_lemmatized, dictionary=id2word, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```
Perplexity:  -8.7540937582

Coherence Score:  0.515758393755
```

pyLDAvis 만큼 jupyter notebook에서 LDA랄 잘 작동하면서 시각화하는 툴도 없다.

In [33]:

```
# Visualize the topics
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(lda_model, corpus, id2word)
vis
```

Out[33]:

- Gensim 의 LDA 알고리즘보다 Mallet 버전이 더 나은 퀄리티를 보여준다.
- https://www.machinelearningplus.com/wp-content/uploads/2018/03/mallet-2.0.8.zip (https://www.machinelearningplus.com/wp-content/uploads/2018/03/mallet-2.0.8.zip) 깔고 해당 경로를 아래에 넣어주면 된다.

In [39]:

```
# Download File: http://mallet.cs.umass.edu/dist/mallet-2.0.8.zip
mallet_path = '/home/hskimim/Documents/mallet-2.0.8/bin/mallet' # update this path
ldamallet = gensim.models.wrappers.LdaMallet(mallet_path, corpus=corpus, num_topics=20, id2word=id2word)
# Show Topics
pprint(ldamallet.show_topics(formatted=False))
```

```
[(11,
  [('armenian', 0.020657466848641091),
   ('people', 0.010075991170870612),
   ('turkish', 0.0079698057254545141),
   ('world', 0.0068914387774014728),
   ('history', 0.0065712985896982257),
   ('turk', 0.0064533522047549247),
   ('greek', 0.0060489645992350335),
   ('turkey', 0.005711974927968458),
   ('war', 0.0056782759608418002),
   ('government', 0.0053581357731385532)]),
 (8,
  [('work', 0.012249956306900333),
   ('power', 0.011614420311730406),
   ('line', 0.010788223518009501),
   ('point', 0.0084367403358807727),
   ('problem', 0.0081507491380543064),
   ('good', 0.0076582087417976135),
   ('high', 0.0075311015427636284),
   ('ground', 0.0073722175439711463),
   ('time', 0.0067525699486804683),
   ('find', 0.0067049047490427239)]),
 (7,
  [('god', 0.021306849989923191),
   ('christian', 0.016873054616297557),
   ('people', 0.008834001388360168),
   ('religion', 0.0080950354927558955),
   ('bible', 0.007893499339409275),
   ('church', 0.0071209440849138994),
   ('word', 0.0068186398548939695),
   ('faith', 0.0064827462659829364),
   ('atheist', 0.0059229236177978815),
   ('life', 0.0058557449000156752)]),
 (9,
  [('write', 0.016513761467889909),
   ('people', 0.011320445609436436),
   ('israel', 0.010714285714285714),
   ('article', 0.010370249017038008),
   ('state', 0.0099442988204456097),
   ('israeli', 0.0089941022280471822),
   ('arab', 0.0083060288335517693),
   ('organization', 0.0075688073394495417),
   ('jew', 0.0074868938401048491),
   ('line', 0.0072411533420707729)]),
 (5,
  [('people', 0.013245513324067919),
   ('write', 0.012363284790621791),
   ('make', 0.012266602211613995),
   ('article', 0.010332950631458095),
   ('post', 0.0093056982295002724),
   ('exist', 0.0091969303281165017),
   ('thing', 0.0090156504924768874),
   ('question', 0.0085443229198138865),
   ('reason', 0.0083388724394223221),
   ('argument', 0.0077708622877515256)]),
 (16,
  [('drive', 0.023383709967717051),
   ('card', 0.017569922578064737),
   ('system', 0.016441447875896353),
   ('problem', 0.015884352769762591),
   ('window', 0.013027454789589464),
   ('scsi', 0.012641773562266092),
   ('driver', 0.012256092334942719),
   ('mac', 0.011084764163071737),
   ('bit', 0.010341970688226723),
   ('work', 0.010184841299317201)]),
 (12,
  [('_', 0.052802948855782524),
   ('organization', 0.030164337275380124),
   ('line', 0.018706803870373215),
   ('ca', 0.01474427891260943),
   ('newsreader_tin', 0.0070956842266932879),
   ('air', 0.0059898633082475809),
   ('cx', 0.0052833666103517122),
   ('md', 0.0041775456919060051),
   ('ms', 0.0039932422054983875),
   ('ed', 0.0036553524804177544)]),
 (6,
  [('car', 0.027885998803731066),
   ('article', 0.013918750707253593),
   ('write', 0.013837921725213793),
   ('bike', 0.011332223281979987),
   ('organization', 0.010329943904686464),
   ('line', 0.0099257989944874642),
   ('good', 0.0096994778447760232),
   ('drive', 0.0076464217009650981),
   ('engine', 0.0064824843595919753),
   ('dod', 0.0060136762637611344)]),
 (0,
  [('game', 0.021562927496580026),
   ('team', 0.018621751025991791),
   ('play', 0.014825581395348838),
   ('hockey', 0.0095417236662106702),
   ('player', 0.0092510259917920664),
   ('win', 0.0081395348837209301),
   ('year', 0.0075752393980848152),
   ('goal', 0.0070622435020519835),
   ('line', 0.0067373461012311901),
   ('season', 0.0066176470588235293)]),
 (1,
  [('line', 0.04087163320612798),
   ('organization', 0.035285658979038771),
   ('sale', 0.016997584941835814),
   ('mail', 0.016868536032299097),
   ('price', 0.01616798480909979),
   ('sell', 0.01436130007558579),
   ('good', 0.01091385063510499),
   ('buy', 0.010821672842578766),
   ('university', 0.010305477204431908),
   ('interested', 0.0099367660343270096)])]
```

In [40]:

```
# Compute Coherence Score
coherence_model_ldamallet = CoherenceModel(model=ldamallet, texts=data_lemmatized, dictionary=id2word, coherence='c_v')
coherence_ldamallet = coherence_model_ldamallet.get_coherence()
print('\nCoherence Score: ', coherence_ldamallet)
```

```
Coherence Score:  0.629787393251
```

51에서 62로 Coherence Score 가 올라갔다.

**LDA 에서 최적의 토픽 갯수 찾기**

- 많은 LDA 모델을 토픽 갯수(k)를 다르게 해서, 많이 시행해본 후, Coherance value 가 가장 높은 것을 선택한다.
- 높은 Coherance Value 를 가지는 k를 선택하는 것은 유의미하고, 세부적인 토픽을 할당할 수 있게끔 한다.
- 여러개의 토픽에 키워드가 많이 중첩되면, 이것은 k를 너무 높게 할당했다는 신호가 될 수 있다.
- compute_coherence_values라는 메소드는 다수의 LDA 모델에 대한 Coherance value 를 알려준다.

In [41]:

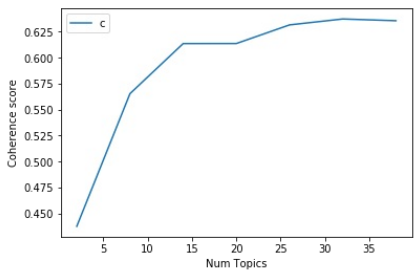
```
def compute_coherence_values(dictionary, corpus, texts, limit, start=2, step=3):
    """
    Compute c_v coherence for various number of topics

    Parameters:
    ----------
    dictionary : Gensim dictionary
    corpus : Gensim corpus
    texts : List of input texts
    limit : Max num of topics

    Returns:
    -------
    model_list : List of LDA topic models
    coherence_values : Coherence values corresponding to the LDA model with respective number of topics
    """
    coherence_values = []
    model_list = []
    for num_topics in range(start, limit, step):
        model = gensim.models.wrappers.LdaMallet(mallet_path, corpus=corpus, num_topics=num_topics, id2word=id2word)
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=texts, dictionary=dictionary, coherence='c_v')
        coherence_values.append(coherencemodel.get_coherence())

    return model_list, coherence_values
```

In [42]:

```
# Can take a long time to run.
model_list, coherence_values = compute_coherence_values(dictionary=id2word, corpus=corpus, texts=data_lemmatized, start=2, limit=40, step=6)
```

In [43]:

```
# Show graph
limit=40; start=2; step=6;
x = range(start, limit, step)
plt.plot(x, coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc='best')
plt.show()
```




In [44]:

```
# Print the coherence scores
for m, cv in zip(x, coherence_values):
    print("Num Topics =", m, " has Coherence Value of", round(cv, 4))
```

```
Num Topics = 2  has Coherence Value of 0.4377
Num Topics = 8  has Coherence Value of 0.5654
Num Topics = 14  has Coherence Value of 0.6136
Num Topics = 20  has Coherence Value of 0.6137
Num Topics = 26  has Coherence Value of 0.6316
Num Topics = 32  has Coherence Value of 0.6373
Num Topics = 38  has Coherence Value of 0.6356
```

In [47]:

```
optimal_model= gensim.models.wrappers.LdaMallet(mallet_path, corpus=corpus, num_topics=32, id2word=id2word)
```

**해당 문장에서 지배적인 토픽을 찾기**

- 토픽 모델링의 주요 활용점은 해당 문서의 토픽이 무엇이냐에 관한 것이다.
- 이를 알아내기 위해서는 해당 문서에서 가장 기여를 많이 한, 즉 중요도가 가장 높은 토픽의 넘버를 찾아야 한다.
- format_topics_sentences() 메소드는 보여지는 테이블로 훌륭하게 정보를 병합해준다.

In [48]:

```python
def format_topics_sentences(ldamodel=lda_model, corpus=corpus, texts=data):
    # Init output
    sent_topics_df = pd.DataFrame()

    # Get main topic in each document
    for i, row in enumerate(ldamodel[corpus]):
        row = sorted(row, key=lambda x: (x[1]), reverse=True)
        # Get the Dominant topic, Perc Contribution and Keywords for each document
        for j, (topic_num, prop_topic) in enumerate(row):
            if j == 0:  # => dominant topic
                wp = ldamodel.show_topic(topic_num)
                topic_keywords = ", ".join([word for word, prop in wp])
                sent_topics_df = sent_topics_df.append(pd.Series([int(topic_num), round(prop_topic,4), topic_keywords]), ignore_index=True)
            else:
                break
    sent_topics_df.columns = ['Dominant_Topic', 'Perc_Contribution', 'Topic_Keywords']

    # Add original text to the end of the output
    contents = pd.Series(texts)
    sent_topics_df = pd.concat([sent_topics_df, contents], axis=1)
    return(sent_topics_df)


df_topic_sents_keywords = format_topics_sentences(ldamodel=optimal_model, corpus=corpus, texts=data)

# Format
df_dominant_topic = df_topic_sents_keywords.reset_index()
df_dominant_topic.columns = ['Document_No', 'Dominant_Topic', 'Topic_Perc_Contrib', 'Keywords', 'Text']

# Show
df_dominant_topic.head(10)
```

Out[48]:

| | Document_No | Dominant_Topic | Topic_Perc_Contrib | Keywords | Text |
|---|---|---|---|---|---|
| **0** | 0 | 13.0 | 0.1650 | car, bike, engine, dod, road, drive, speed, ri... | From: (wheres my thing) Subject: WHAT car is t... |
| **1** | 1 | 22.0 | 0.1328 | drive, scsi, system, disk, problem, speed, mem... | From: (Guy Kuo) Subject: SI Clock Poll - Final... |
| **2** | 2 | 13.0 | 0.3219 | car, bike, engine, dod, road, drive, speed, ri... | From: (Irwin Arnstein) Subject: Re: Recommenda... |
| **3** | 3 | 19.0 | 0.2672 | window, server, application, run, set, display... | From: (Tsung-Kun Chen) Subject: ** Software fo... |
| **4** | 4 | 15.0 | 0.2223 | card, window, mac, driver, monitor, apple, pro... | From: (Don A.B. Lindbergh) Subject: Diamond SS... |
| **5** | 5 | 13.0 | 0.3812 | car, bike, engine, dod, road, drive, speed, ri... | From: (Robert Loper) Subject: Re: SHO and SC N... |
| **6** | 6 | 30.0 | 0.0949 | price, sale, sell, organization, line, buy, of... | From: (Kim Richard Man) Subject: SyQuest 44M c... |
| **7** | 7 | 30.0 | 0.2020 | price, sale, sell, organization, line, buy, of... | From: (Kirtley Wilson) Subject: Mirosoft Offic... |
| **8** | 8 | 18.0 | 0.2575 | write, fire, article, people, start, news, day... | Subject: Re: Dont more innocents die without t... |
| **9** | 9 | 10.0 | 0.2490 | question, exist, claim, argument, reason, evid... | From: (Jon Livesey) Subject: Re: Genocide is C... |

**각각의 토픽을 대표하는 문서찾기**

- 가끔 토픽 키워드(단어)는 단지 토픽들을 구성하는 것에 그치지 않는 경우가 있다.
- 토픽을 이해하는 것을 넘어서, 토픽을 형성하는데 가장 많은 기여를 한 문서를 찾아낼 수도 있다.

In [49]:

```python
# Group top 5 sentences under each topic
sent_topics_sorteddf_mallet = pd.DataFrame()

sent_topics_outdf_grpd = df_topic_sents_keywords.groupby('Dominant_Topic')

for i, grp in sent_topics_outdf_grpd:
    sent_topics_sorteddf_mallet = pd.concat([sent_topics_sorteddf_mallet,
                                             grp.sort_values(['Perc_Contribution'], ascending=[0]).head(1)],
                                            axis=0)

# Reset Index
sent_topics_sorteddf_mallet.reset_index(drop=True, inplace=True)

# Format
sent_topics_sorteddf_mallet.columns = ['Topic_Num', "Topic_Perc_Contrib", "Keywords", "Text"]

# Show
sent_topics_sorteddf_mallet.head()
```

Out[49]:

| | Topic_Num | Topic_Perc_Contrib | Keywords | Text |
|---|---|---|---|---|
| **0** | 0.0 | 0.8453 | armenian, people, turkish, turk, turkey, greek... | From: (Serdar Argic) Subject: To be exact, 2.5... |
| **1** | 1.0 | 0.8592 | organization, line, good, drug, water, cover, ... | From: (Jeff Mason) Subject: Marvel, DC, Valian... |
| **2** | 2.0 | 0.8197 | president, make, work, money, year, government... | From: (Clinton/Gore 92) Subject: CLINTON: Back... |
| **3** | 3.0 | 0.9179 | government, encryption, technology, key, syste... | From: (Clipper Chip Announcement) Subject: tex... |
| **4** | 4.0 | 0.7373 | year, game, run, good, hit, win, team, player,... | From: Subject: ALL-TIME PEAK PLAYERS Organizat... |

**문서를 넘어선 토픽 분배**

- 마지막으로 우리는 해당 정보에서 어떤 것들이 가장 많이 거론되었는지를 토픽의 크기(volume)과 분포(distribution)로 이해할 수 있게 된다.

In [52]:

```
# Number of Documents for Each Topic
topic_counts = df_topic_sents_keywords['Dominant_Topic'].value_counts()

# Percentage of Documents for Each Topic
topic_contribution = round(topic_counts/topic_counts.sum(), 4)

# Topic Number and Keywords
topic_num_keywords = df_topic_sents_keywords[['Dominant_Topic', 'Topic_Keywords']]

# Concatenate Column wise
df_dominant_topics = pd.concat([topic_num_keywords, topic_counts, topic_contribution], axis=1)

# Change Column names
df_dominant_topics.columns = ['Dominant_Topic', 'Topic_Keywords', 'Num_Documents', 'Perc_Documents']

# Show
df_dominant_topics.iloc[:10]
```

Out[52]:

| | Dominant_Topic | Topic_Keywords | Num_Documents | Perc_Documents |
|---|---|---|---|---|
| **0** | 13.0 | car, bike, engine, dod, road, drive, speed, ri... | 163.0 | 0.0144 |
| **1** | 22.0 | drive, scsi, system, disk, problem, speed, mem... | 202.0 | 0.0179 |
| **2** | 13.0 | car, bike, engine, dod, road, drive, speed, ri... | 231.0 | 0.0204 |
| **3** | 19.0 | window, server, application, run, set, display... | 370.0 | 0.0327 |
| **4** | 15.0 | card, window, mac, driver, monitor, apple, pro... | 542.0 | 0.0479 |
| **5** | 13.0 | car, bike, engine, dod, road, drive, speed, ri... | 179.0 | 0.0158 |
| **6** | 30.0 | price, sale, sell, organization, line, buy, of... | 241.0 | 0.0213 |
| **7** | 30.0 | price, sale, sell, organization, line, buy, of... | 88.0 | 0.0078 |
| **8** | 18.0 | write, fire, article, people, start, news, day... | 451.0 | 0.0399 |
| **9** | 10.0 | question, exist, claim, argument, reason, evid... | 466.0 | 0.0412 |