L3: Text clustering and topic modelling

Text clustering groups documents in such a way that documents within a group are more 'similar' to other documents in the cluster than to documents not in the cluster. The exact definition of what 'similar' means in this context varies across applications and clustering algorithms.

In this lab you will experiment with both hard and soft clustering techniques. More specifically, in the first part you will be using the k-means algorithm, and in the second part you will be using a topic model based on the Latent Dirichlet Allocation (LDA).

Hard clustering data set

The raw data for the hard clustering part of this lab is a collection of product reviews. We have preprocessed the data by tokenization and lowercasing.

In [131…
```
import pandas as pd
import bz2

with bz2.open('reviews.json.bz2') as source:
    df = pd.read_json(source)
```

When you inspect the data frame, you can see that there are three labelled columns: `category` (the product category), `sentiment` (whether the product review was classified as 'positive' or 'negative' towards the product), and `text` (the space-separated text of the review).

In [143…
```
df['category'].unique()
#df.shape
```

Out[143…
```
array(['music', 'books', 'dvd', 'camera', 'health', 'software'],
      dtype=object)
```

Problem 1: K-means clustering

Your first task is to cluster the product review data using a tf–idf vectorizer and a k-means clusterer.

Start by doing the vectorization. In connection with vectorization, you should also filter out standard English stop words. While you could use spaCy for this task, here it suffices to use the word list implemented in TfidfVectorizer.

In [15]:
```
# TODO: Enter code here to vectorize the data and store it in a variable `re
import spacy
from sklearn.feature_extraction.text import TfidfVectorizer

#nlp = spacy.load("en_core_web_sm",disable=["tagger", "parser"])
#def preprocess(text):
#    doc = nlp(text)
#    return [token.lemma_ for token in doc if token.is_stop == False]


tfidf_vectorizer=TfidfVectorizer(stop_words='english')
reviews = tfidf_vectorizer.fit_transform(df['text'])
```

Test your vectorization by running the following code cell:

In [16]:
```
reviews.shape
```

Out[16]:
```
(11914, 46619)
```

If you used the English stop word list from scikit-learn, then the resulting vocabulary should have 46,619 entries.

Next, cluster the vectorized data. Before doing so, you should read the documentation of the KMeans class, which is scikit-learn's implementation of the k-means algorithm. As you can see, this class has several parameters that you can tweak. For now, the only parameter that you will have to set is the number of clusters. We recommend that you choose $k = 3$.

Tip: Training k-means models will take some time. To speed things up, you can use the `n_init` parameter to control the number of times that the clustering is re-computed with different initial values. The default value for this parameter is 10; here and in the rest of this lab, you may want to set this to a lower value.

In [149…
```
# TODO: Enter code here to cluster the vectorized data
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, n_init=2).fit(reviews)
```

Out[149…
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=2, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

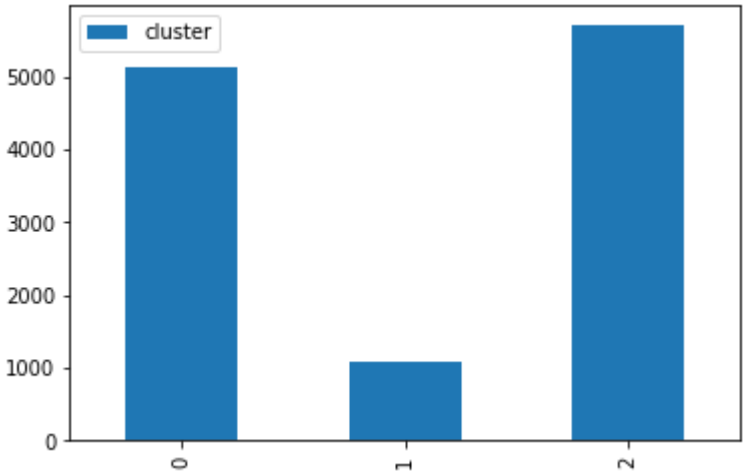
To sanity-check your clustering, create a bar plot with the number of documents per cluster.

In [42]:
```
# TODO: Enter code here to produce a bar plot of the cluster size
import collections
import matplotlib.pyplot as plt


labels = collections.Counter(kmeans.labels_)

cluster = pd.DataFrame({"cluster":labels})
cluster.plot.bar()
```

Out[42]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc45467f1d0>



Note that sizes may vary considerable between clusters and among different random seeds.

Problem 2: Summarize clusters

Once you have a clustering, you can try to see whether it is meaningful. One useful technique in that context is to generate a **summary** for each cluster by extracting the n highest-weighted terms from the centroid of each cluster. Your next task is to implement this approach.

Hint: You will need to construct an 'inverted vocabulary' that allows you to map from the index of a term back to the original term.

In [128…
```
# TODO: Enter code here to compute the cluster summaries and print them

vocab = tfidf_vectorizer.vocabulary_
vocab2 = {val:key for key,val in vocab.items()}

ind = np.argsort(kmeans.cluster_centers_, axis=1)[:,-10:]

#map ind to vocab
for cluster,index in enumerate(ind):
    print(cluster)
    print( [ vocab2[j] for j in index ])
```

```
0
['good', 'work', 'just', 'used', 'program', 'great', 'does', 'software', 'us
e', 'product']
```

```
1
['great', 'quality', 'battery', 'flash', 'use', 'digital', 'canon', 'picture
s', 'lens', 'camera']
2
['film', 'great', 'quot', 'music', 'just', 'cd', 'like', 'album', 'movie', '
book']
```

Once you have computed the cluster summaries, discuss their quality. Is it clear what the reviews in a given cluster are about? Which clusters are clearest? Which are less clear? Do the cluster summaries contain any unexpected terms? What happens if you re-cluster with, say, $k = 6$?

TODO: Insert your analysis of the clusters here

"music, book , dvd" The summaries of these cluster are not very clear, the quality of the clusters are very low. The cluster 1 and 3 are not clear, the words are mixed with various categories. The clearest cluster is cluster2, because we can guess from this cluster should belongs to the CAMERA category.

So if we re-cluster with k=6, it would be better because the given data has 6 categories

Problem 3: Compare clusterings using the Rand index

In some scenarios, you may have gold-standard class labels available for at least a subset of your documents. In these cases you can compute the **Rand index** of a clustering, and use this measure to compare the quality of different clusterings.

To compute the Rand index, we view a clustering as a binary classifier on (unordered) pairs of documents. The classifier predicts 'positive' if and only if the two documents belong to the same cluster. The (non-normalized) Rand index of the clustering is the accuracy of this classifier relative to a reference in which a document pair belongs to the 'positive' class if and only if the two documents in the pair have the same gold-standard class label.

Compare clusterings with $k \in \{1, 3, 5, 7\}$ clusters. As your evaluation data, use the first 500 documents from the original data set along with their gold-standard categories (from the `category` column). What do you observe? How do you interpret your observations?

In [232…

```
# TODO: Enter code here to compute the Rand indices for the two clusterings

def rand_index(true, pred):

    tp,fn,fp,tn = 0,0,0,0

    for i in range(len(true)-1):
        if true[i]==true[i+1] and pred[i]==pred[i+1]:
            tp += 1

        if true[i]==true[i+1] and pred[i]!=pred[i+1]:
            fn += 1

        if true[i]!=true[i+1] and pred[i]==pred[i+1]:
            fp += 1

        if true[i]!=true[i+1] and pred[i]!=pred[i+1]:
            tn += 1

    rand_ind = (tp+tn)/(tp+fn+fp+tn)

    return(rand_ind)

ks=np.array(([1,3,5,7]))

train = reviews[0:500]
trueLabel = np.array(df[0:500]['category'])


for k in ks:

    kmeans = KMeans(n_clusters=k, n_init=2).fit(train)

    rand_ind = rand_index(true=trueLabel,pred=kmeans.labels_)
    print("k =",k)
    print("rand_index =" ,rand_ind)
```

```
k = 1
rand_index = 0.16032064128256512
k = 3
rand_index = 0.6132264529058116
k = 5
rand_index = 0.7254509018036072
k = 7
rand_index = 0.8056112224448898
```

TODO: Insert your discussion of your results here

Topic modelling data set

The data set for the topic modelling part of this lab is the collection of all State of the Union addresses from the years 1975–2000. These speeches come as a single text file with one sentence per line. The following code cell prints the first 5 lines from the data file:

In [138…

```
with open('sotu_1975_2000.txt') as source:
    for i, line in enumerate(source):
        print(line.rstrip())
        if i >= 5:
            break
```

```
mr speaker mr vice president members of the 94th congress and distinguished
guests
twenty six years ago a freshman congressman a young fellow with lots of idea
lism who was out to change the world stood before sam rayburn in the well of
the house and solemnly swore to the same oath that all of you took yesterday
an unforgettable experience and i congratulate you all
two days later that same freshman stood at the back of this great chamber ov
er there someplace as president truman all charged up by his single handed e
lection victory reported as the constitution requires on the state of the un
ion
when the bipartisan applause stopped president truman said i am happy to rep
ort to this 81st congress that the state of the union is good our nation is
better able than ever before to meet the needs of the american people and to
give them their fair chance in the pursuit of happiness it is foremost among
the nations of the world in the search for peace
today that freshman member from michigan stands where mr truman stood and i
must say to you that the state of the union is not good
millions of americans are out of work
```

Take a few minutes to think about what topics you would expect in this data set.

Problem 4: Train a topic model

Your first task on the topic modelling data is to train an LDA model. For this task you will be using spaCy and the gensim topic modelling library.

Start by preprocessing the data using spaCy. Given that the data set for this problem is rather small, you do not have to exclude any components from the standard pipeline. Filter out stop words, non-alphabetic tokens, and tokens less than 3 characters in length. Store the documents as a nested list where the first level of nesting corresponds to the sentences and the second level corresponds to the tokens in each sentence.

In [11]:

```
# TODO: Replace the following lines with your own code for preprocessing the

import spacy

nlp = spacy.load("en_core_web_sm",disable=["tagger", "parser"])

def preprocess(text):

    doc = nlp(text)
    return [token.lemma_ for token in doc if token.is_stop == False and toke



with open('sotu_1975_2000.txt') as source:
    documents = [preprocess(line) for line in source]
```

Test your preprocessing by running the following cell:

In [242…
```
' '.join(documents[42])
```

Out[242… 'reduce oil import million barrel day end year million barrel day end'

You should get the following output:

'reduce oil imports million barrels day end year million barrels day end'

Once you have the list of documents, skim the section Pre-process and vectorize the documents of the gensim documentation to learn how to create the dictionary and the vectorized corpus representation required by gensim. (Note that you cannot use the standard scikit-learn pipeline in this case.) Then, write code to train an LdaModel for $k = 10$ topics, and using default values for all other parameters.

In [13]:
```
# TODO: Enter code here to train an LDA model and store it in a variable `mo


from gensim.models import Phrases
from gensim.corpora import Dictionary
from gensim.models import LdaModel

# Add bigrams and trigrams to docs (only ones that appear 20 times or more).
bigram = Phrases(documents, min_count=20)
for idx in range(len(documents)):
    for token in bigram[documents[idx]]:
        if '_' in token:
            # Token is a bigram, add to document.
            documents[idx].append(token)



# Create a dictionary representation of the documents.
dictionary = Dictionary(documents)

dictionary.filter_extremes(no_below=20, no_above=0.5)

#vectorizced corpus representation
vectorized_corpus = [dictionary.doc2bow(text) for text in documents]


# Train the model on the corpus.
model = LdaModel(vectorized_corpus, num_topics=10)
```

In [14]:
```
# to train LDA model.

from gensim.models import LdaModel

# Make a index to word dictionary.
temp = dictionary[0]  # This is only to "load" the dictionary.
id2word = dictionary.id2token



model = LdaModel(
    corpus=vectorized_corpus,
    id2word=id2word,
    num_topics=10

)
```

Once you have a trained model, run the following cell to print the topics:

In [268…
```
model.print_topics()
```

Out[268…
```
[(0,
  '0.291*"long_term" + 0.122*"tax_credit" + 0.085*"tax_cut" + 0.078*"interes
t_rate" + 0.075*"white_house" + 0.074*"economic_growth" + 0.063*"american_pe
ople" + 0.061*"social_security" + 0.012*"health_care" + 0.012*"federal_gover
nment_federal_government"'),
 (1,
  '0.186*"fellow_americans" + 0.186*"health_care" + 0.126*"health_insurance"
+ 0.071*"welfare_reform" + 0.061*"economic_growth" + 0.043*"god_bless" + 0.0
42*"balance_budget" + 0.035*"balance_budget_balance_budget" + 0.034*"young_p
eople" + 0.030*"economic_growth_economic_growth"'),
 (2,
  '0.202*"vice_president" + 0.160*"low_income" + 0.142*"congress_pass" + 0.1
05*"private_sector" + 0.058*"past_year" + 0.052*"unite_state" + 0.051*"year_
ago" + 0.045*"persian_gulf" + 0.026*"year_ago_year_ago" + 0.023*"tax_credi
t"'),
 (3,
  '0.493*"american_people" + 0.058*"foreign_policy" + 0.057*"middle_east" +
0.048*"state_union" + 0.027*"balance_budget_balance_budget_balance_budget" +
0.021*"middle_east_middle_east" + 0.018*"million_americans" + 0.017*"soviet_
union" + 0.016*"fellow_americans" + 0.016*"human_right"'),
 (4,
  '0.234*"past_year" + 0.206*"tax_credit" + 0.125*"million_americans" + 0.12
5*"meet_challenge" + 0.056*"ask_congress" + 0.046*"state_union" + 0.022*"bal
ance_budget_balance_budget" + 0.018*"welfare_reform" + 0.016*"health_care" +
0.013*"economic_growth"'),
 (5,
  '0.168*"young_people" + 0.088*"state_union" + 0.071*"social_security_socia
l_security" + 0.062*"health_care_health_care" + 0.059*"health_care" + 0.055
*"state_union_state_union" + 0.039*"million_americans" + 0.038*"health_care_
health_care_health_care_health_care_health_care_health_care_health_care_heal
th_care" + 0.023*"white_house" + 0.021*"health_care_health_care_health_care_
health_care_health_care_health_care_health_care_health_care_heal
th_care_health_care_health_care_health_care_health_care_health_care_health_c
are"'),
 (6,
  '0.303*"unite_state" + 0.240*"balance_budget" + 0.044*"tax_cut" + 0.043*"m
an_woman" + 0.039*"past_year" + 0.030*"american_people" + 0.023*"nuclear_wea
```

```
pon" + 0.022*"human_right" + 0.017*"unite_state_unite_state" + 0.016*"econom
ic_growth"'),
 (7,
  '0.224*"social_security" + 0.125*"member_congress" + 0.114*"social_securit
y_social_security" + 0.101*"private_sector" + 0.075*"federal_government" +
0.052*"common_sense" + 0.046*"vice_president" + 0.036*"take_office" + 0.031
*"fellow_americans" + 0.027*"social_security_social_security_social_security
_social_security_social_security_social_security_social_security_social_secu
rity"'),
 (8,
  '0.392*"year_ago" + 0.123*"million_americans" + 0.069*"fellow_citizen" +
0.061*"soviet_union" + 0.046*"state_local" + 0.023*"interest_rate" + 0.023*"
unite_state" + 0.023*"tax_cut_tax_cut" + 0.018*"past_year" + 0.017*"state_un
ion"'),
 (9,
  '0.304*"ask_congress" + 0.262*"cold_war" + 0.080*"federal_government" + 0.
053*"american_people" + 0.043*"federal_government_federal_government" + 0.03
1*"past_year" + 0.031*"nuclear_weapon" + 0.031*"take_office" + 0.017*"year_a
go" + 0.015*"state_union"')]
```

Inspect the topics. Can you 'label' each topic with a short description of what it is about? Do the topics match your expectations? Summarize your discussion in a short text.

TODO: Insert your discussion of the topics here

The genearl topic is the speech to the new congress member, focus on the topics of economic, dipomatic, health care, social seurity and social welfare. From the classification above, we can guess the topics are: 1. economics, 2. health care budget 3.income and pension 4.diplomatic policy 5.welfare 6.health care 7.social security 8.social security 9.tax 10.cold war

Problem 5: Monitor a topic model for convergence

When learning an LDA model, it is important to make sure that the training algorithm has converged to a stable posterior distribution. One way to do so is to plot, after each training epochs (or 'pass', in gensim parlance) the log likelihood of the training data under the posterior. Your last task in this lab is to create such a plot and, based on this, to suggest an appropriate number of epochs.

To collect information about the posterior likelihood after each pass, we need to enable the logging facilities of gensim. Once this is done, gensim will add various diagnostics to a log file `gensim.log` .

In [15]:
```
import logging

logging.basicConfig(filename='gensim.log', format='%(asctime)s:%(levelname)s
```

The following function will parse the generated logfile and return the list of log likelihoods.

In [16]:
```
import re

def parse_logfile():
    matcher = re.compile('(-*\d+\.\d+) per-word .* (\d+\.\d+) perplexity')
    likelihoods = []
    with open('gensim.log') as source:
        for line in source:
            match = matcher.search(line)
            if match:
                likelihoods.append(float(match.group(1)))
    return likelihoods
```

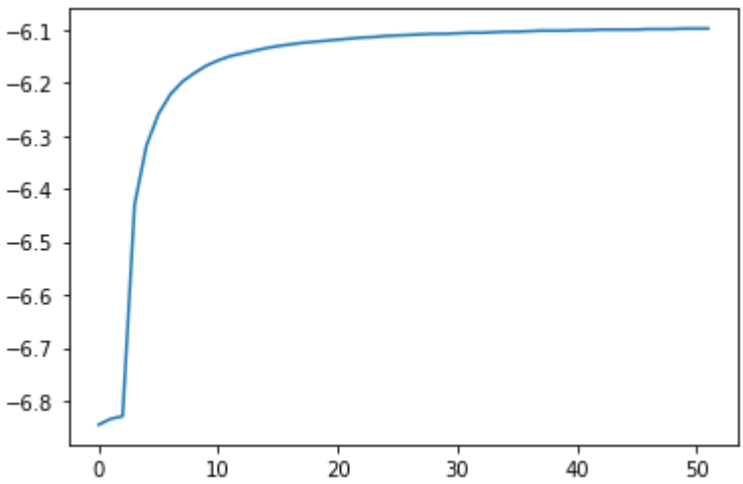
Your task now is to re-train your LDA model for 50 passes, retrieve the list of log likelihoods, and create a plot from this data.

In [7]:
```
parse_logfile()
```

Out[7]: []

In [18]:
```
# TODO: Enter code here to generate the convergence plot
from gensim.models import LdaModel
passes = 50
model = LdaModel(
    corpus=vectorized_corpus,
    id2word=id2word,
    num_topics=10,
    passes=passes
)
```

In [26]:
```
import matplotlib.pyplot as plt
log_llh = parse_logfile()
plt.plot(range(len(log_llh)), log_llh)
plt.show()
```



In [ ]:
```
```

How do you interpret your plot? Based on the plot, what would be a reasonable choice for the number of passes? Retrain your LDA model with that number and re-inspect the topics it finds. Do you consider the new topics to be 'better' than the ones that you got from the 1-pass model in Problem 5?

TODO: Insert your discussion of these questions here

Please read the section 'General information' on the 'Labs' page of the course website before submitting this notebook!