Introducing Haiku
Topic Modeling Framework

July 2018

By Ron Oshima

What is Topic Model?

- In ML and NLP, a topic model is a type of statistical model for discovering the abstract "topics" that occur in a collection of documents.

  (from Wikipedia)

- Topic modeling can reveal the latent structure of text data and is useful for knowledge discovery, search relevance ranking, document classification, and so on. One of the major challenges in topic modeling is to deal with large datasets and large numbers of topics in real-world applications.

- A good topic modeling undermine and collect semantically-connected words into the same group

What topic modeling techniques are available?

Python:

- LDA, SVD, and NMF, KMeans
- sklearn, genism
- Mallet, DTM, Tethne

.Net

- Infer.NET (LDA)
- GibbsLDA.NET

AMLS

- LDA using Vowpal Wabbit Lib

Benefits of Haiku's topic modeling framework

- Automated, Systematic and Consistent
- Visualization and Metric Comparison
- Flexible parameter tuning for experimentation and analysis

Non-Matrix Factorization (a.k.a. NMF)

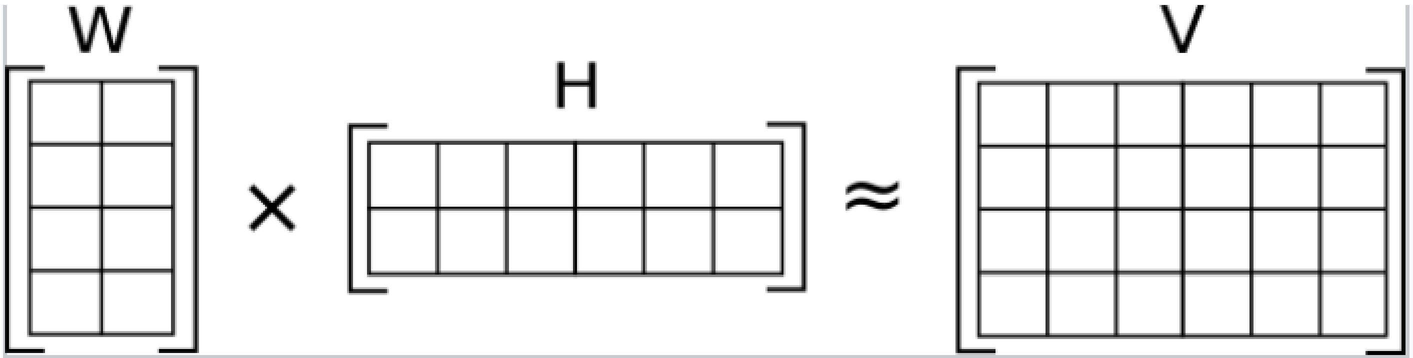


Illustration of approximate non-negative matrix factorization: the matrix $\mathbf{V}$ is represented by the two smaller matrices $\mathbf{W}$ and $\mathbf{H}$, which, when multiplied, approximately reconstruct $\mathbf{V}$.

(from Wikipedia)

Haiku Topic Modeling Framework - NMF example

```
from haiku.TextPreprocess import TextPreprocess
from haiku.topic_modeling import common as tm_common
from haiku.topic_modeling.common import Dataset

# preprocess raw texts using Haiku
dataset = build_dataset(clean_texts_with_stopwords, raw_documents, datadir, dataset_name)

# define parameters for experimentation, if not using built-in NMF function (by default)
# for two APIs: (1) one stop with specified # of topics. (2) optimizing # of topics
from sklearn.decomposition import NMF
dataset.myfunc = lambda: NMF(init = 'nndsvd', n_components = 5, solver = 'mu', beta_loss = 'kullback-leibler')
dataset.myfunc_optimization = lambda k: NMF(init = 'nndsvd', n_components = k, solver = 'mu', beta_loss = 'kullback-leibler')

# run training process, True/False flag defines whether or not to do optimization
best_model = tm_nmf_sklearn.create_model(dataset, True)

# preprocess text, and predict linked topic, as well as the most similar docs from the corpus
mytext = tm_common.preprocess_text(mytext)
tm_nmf_sklearn.predict(mytext, best_model, dataset)
```

NMF

Preparation

```
vectorizer = TfidfVectorizer( analyzer='word',
                              min_df=min_df,              # minimum reqd occurences of a word
                              #stop_words="english",      # remove stop words, optional as it is supposed to be removed already
                              lowercase=True,              # convert all words to lowercase
                              token_pattern='[a-zA-Z0-9]{3,}',  # num chars > 3
                              # max_features=50000,       # max number of uniq words
                              )
vectorized_data = vectorizer.fit_transform(text_data)
```

http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Approach 1

```
x_tfidf = TfidfTransformer(smooth_idf=False).fit_transform(vectorized_data)
xtfidf_norm = normalize(x_tfidf,  norm='l1',  axis=1)          # normalize the TfIdf values to unit length for each row.
model = NMF(n_components = topics_k, init = 'nndsvd');
model.fit(xtfidf_norm)
```

http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html

Approach 2

```
model = NMF(n_components = topics_k, init = 'nndsvd');
model.fit_transform(vectorized_data)
```

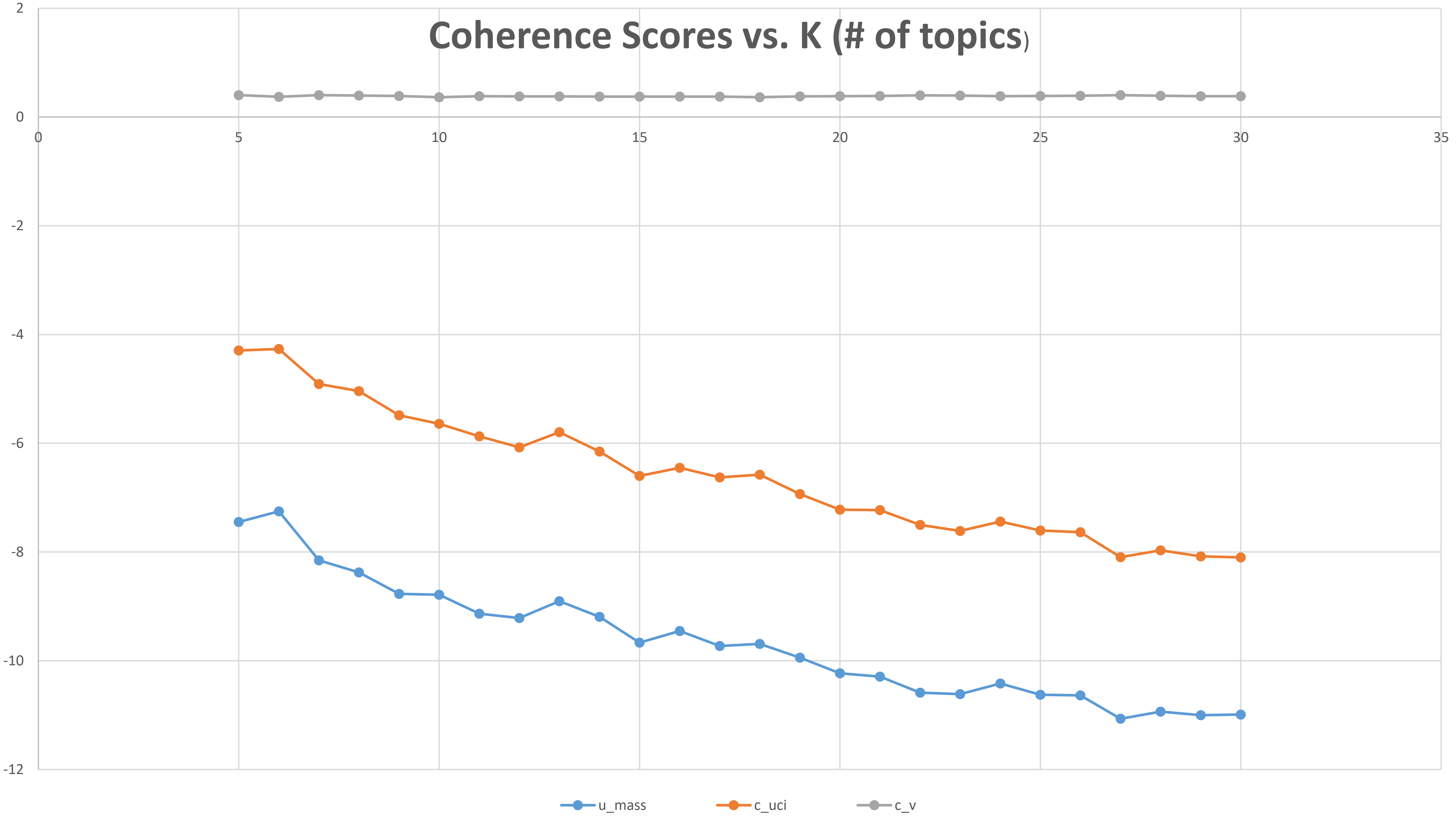
Deciding # of topics?

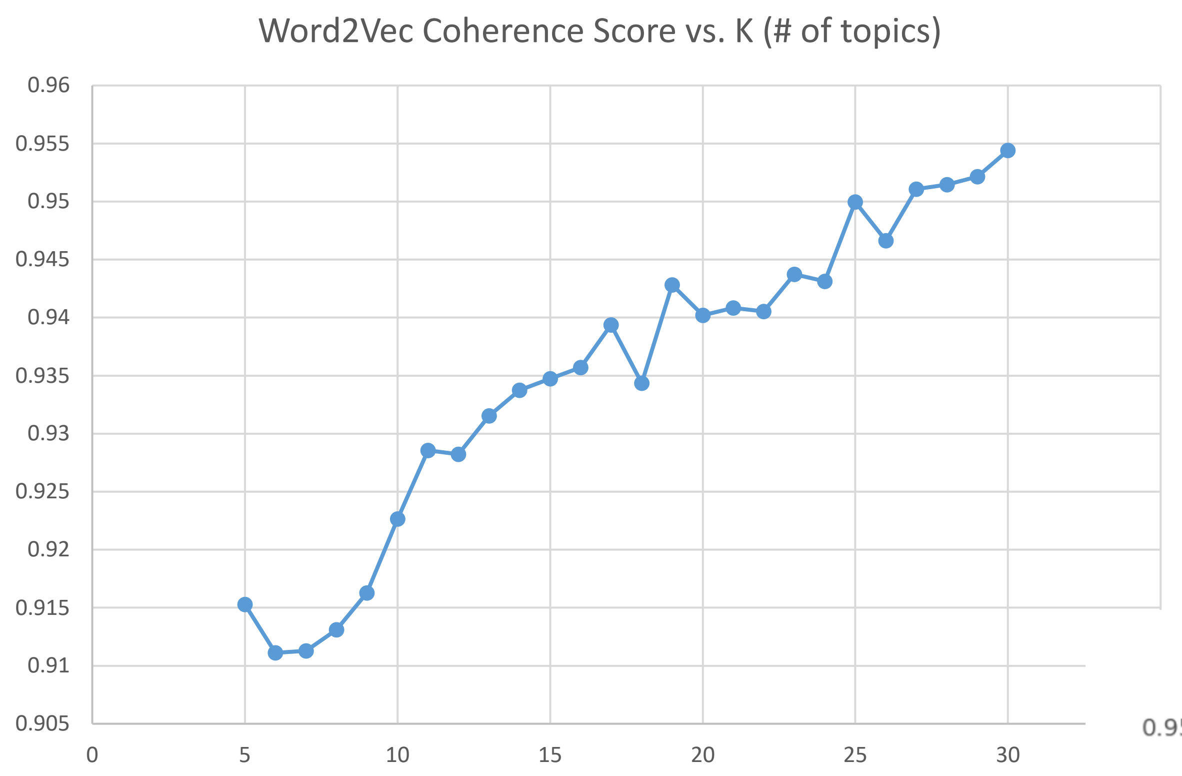
1. Run train-topics with a varying number of topics
2. Analyze topic composition break down.
   - If the majority of the words group to a very narrow number of topics, we need to increase the number of topics.

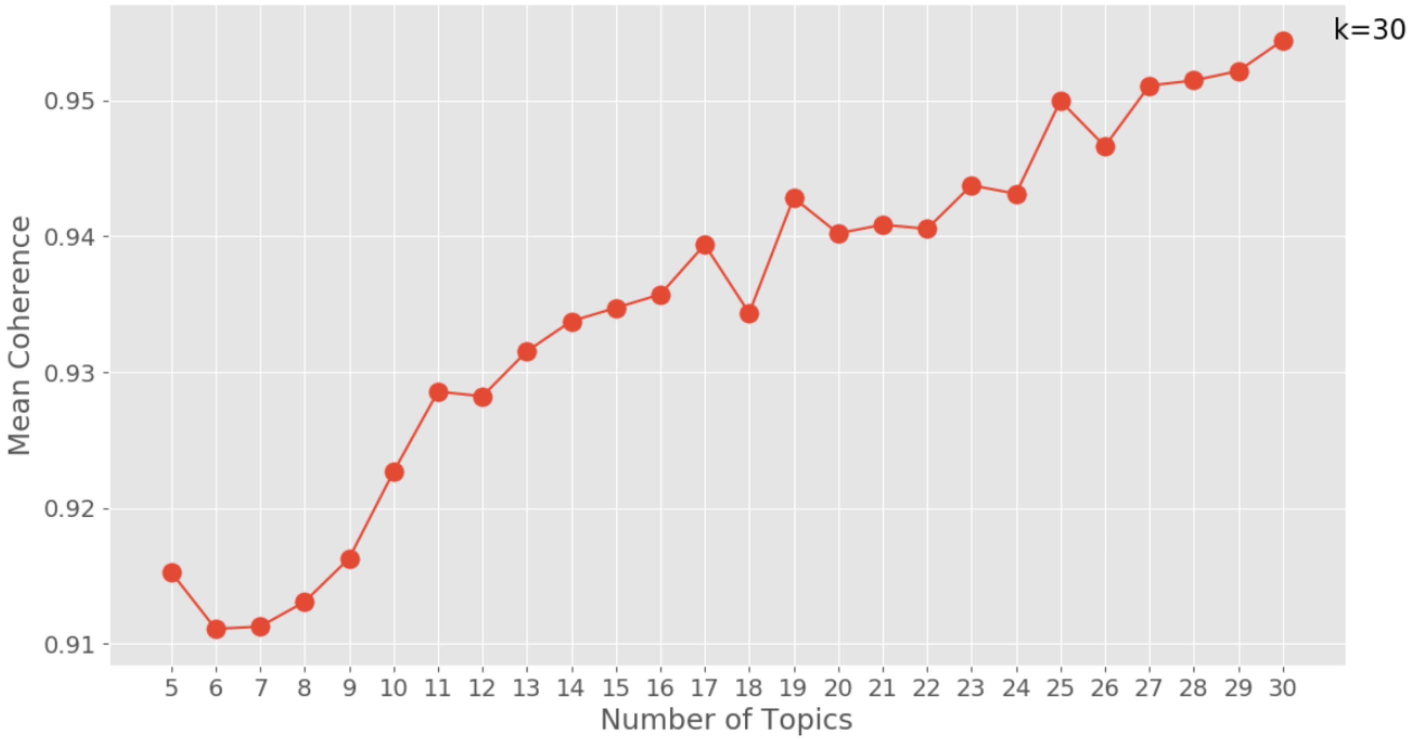
	Topic Num	Num Documents
0	3	6948
1	1	320
2	2	150
3	4	139
4	0	23

   - If related words fall under different topics, the setting is too broad and we need to narrow it down by reducing the number of topics.

Model Optimization by # of topics



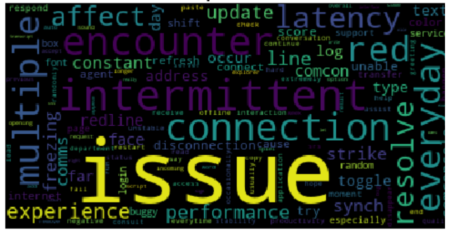

Word2Vec Coherence Score vs. K (# of topics)
0.96
0.955
0.95
0.945
0.94
0.935
0.93
0.925
0.92
0.915
0.91
0.905
0
5
10
15
20
25
30


k=30
Mean Coherence
0.95
0.94
0.93
0.92
0.91
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Topics

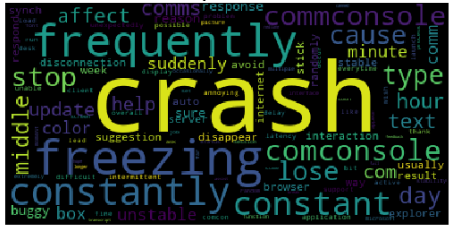
Top words / topic, Word cloud

Topic#	Word	Probability	Topic#	Word	Probability	Topic#	Word	Probability
0	issue	5.902	1	crash	4.194	2	good	3.86
0	intermittent	0.116	1	freezing	0.057	2	far	0.091
0	encounter	0.095	1	frequently	0.049	2	job	0.033
0	connection	0.088	1	constantly	0.047	2	better	0.023
0	red	0.071	1	constant	0.038	2	support	0.02
0	multiple	0.07	1	comconsole	0.029	2	service	0.019
0	everyday	0.068	1	commconsole	0.029	2	thank	0.018
0	latency	0.067	1	stop	0.028	2	person	0.017
0	resolve	0.063	1	lose	0.027	2	look	0.017
0	affect	0.053	1	cause	0.026	2	live	0.017
Topic#	Word	Probability	Topic#	Word	Probability			
3	chat	4.408	4	bug	3.785			
3	close	0.842	4	day	0.056			
3	disconnected	0.647	4	glitch	0.04			
3	accept	0.355	4	affect	0.037			
3	end	0.347	4	session	0.023			
3	window	0.325	4	remove	0.016			
3	transfer	0.317	4	feedback	0.015			
3	agent	0.3	4	intermittent	0.015			
3	come	0.237	4	constant	0.014			
3	case	0.236	4	need	0.014			

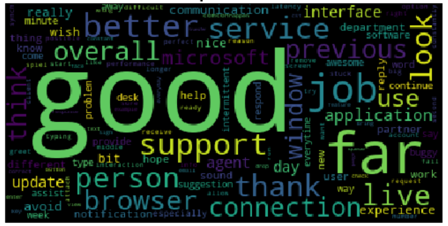
Topic #0



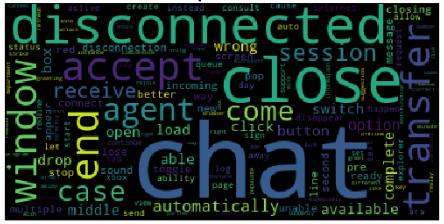
Topic #1



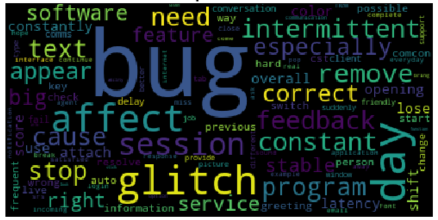
Topic #2



Topic #3



Topic #4

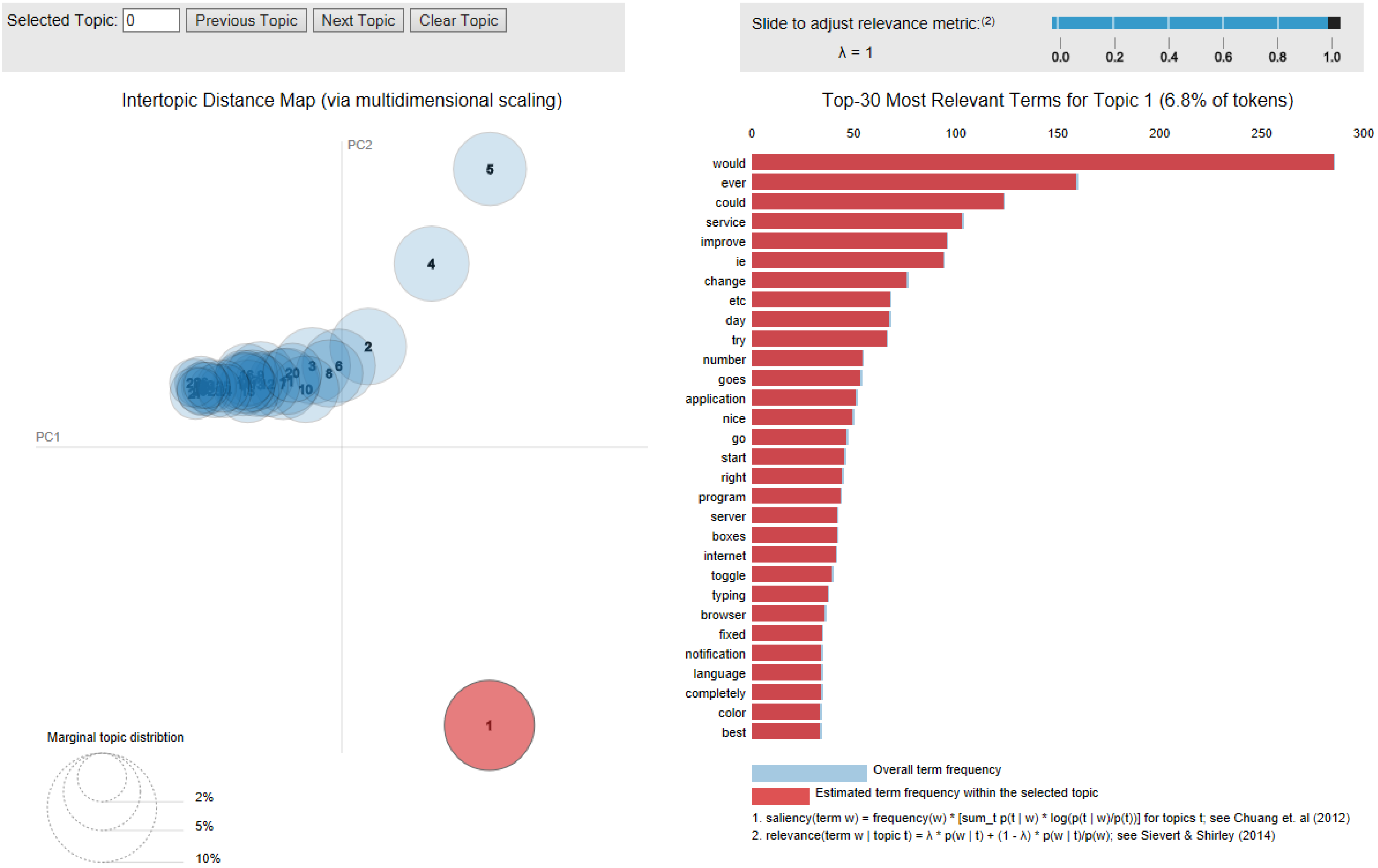


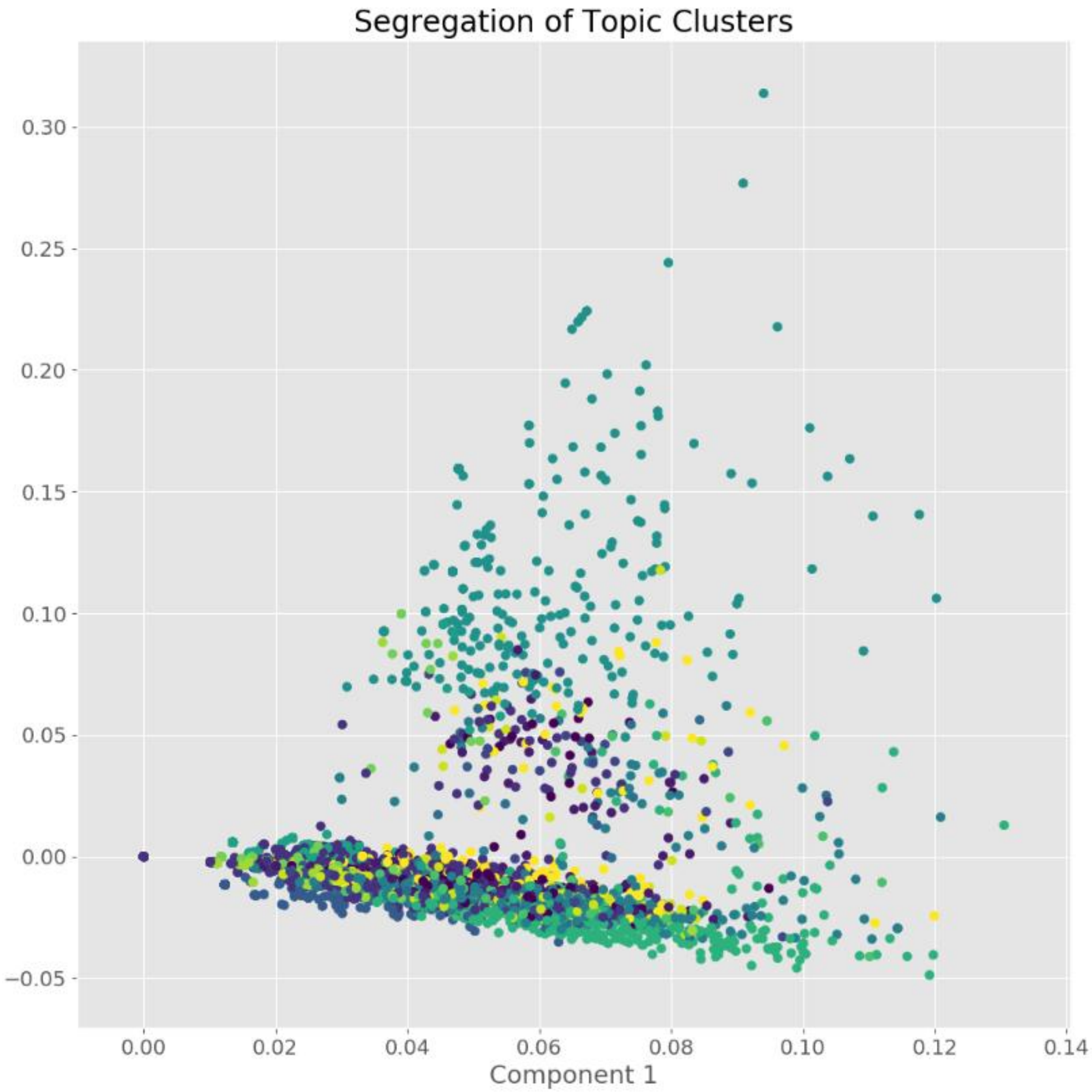
Topic matrix, Topic/Word table

Topic#	Num Documents
0	1002
26	486
3	455
18	407
12	395
5	309
2	300
14	296
10	294
1	282
20	273
4	266
8	247
19	246
7	241
16	233
6	228
24	220
15	194
13	190
22	178
11	173
17	169
9	164
21	142
23	114
25	76

	Topic # 01	Topic # 02	Topic # 03	Topic # 04	Topic # 05	Topic # 06	Topic # 07	Topic # 08	Topic # 09	Topic # 10	Topic # 11	To
0	issue	crash	good	chat	bug	error	tool	asd	freeze	lot	time	ba
1	intermitter	freezing	far	close	day	message	awesome	close	type	box	real	se
2	encounter	frequently	job	disconnect	glitch	respond	use	case	respond	disconnect	multiple	pe
3	connectior	constantly	better	accept	affect	encounter	problem	synch	unable	cause	log	ex
4	red	constant	support	end	session	microsoft	everyday	open	constantly	disconnect	case	ap
5	multiple	comconsol	service	window	remove	log	suck	complete	frequently	face	login	sc
6	everyday	commcons	thank	transfer	feedback	receive	way	comcon	load	problem	load	sc
7	latency	stop	person	agent	intermitter	red	encounter	comconsol	minute	glitch	day	up
8	resolve	lose	look	come	constant	long	browser	load	application	comm	sign	re
9	affect	cause	live	case	need	script	know	account	lose	agent	use	re
10	experience	type	browser	session	stop	com	thank	link	random	affect	red	dr
11	performan	day	overall	receive	cause	line	cause	bring	text	case	stuck	co
12	line	middle	previous	automatica	program	affect	communica	closing	unexpecte	com	restart	pr
13	update	comms	connectior	option	text	try	microsoft	comm	suddenly	come	open	st
14	constant	text	think	able	correct	disconnect	satisfied	window	usually	experience	comconsol	lo
15	synch	hour	use	available	appear	multiple	perfect	create	day	suggestion	line	su
16	strike	affect	window	button	especially	intermitter	helpful	restart	everytime	switch	message	st
17	freezing	update	microsoft	complete	software	change	remove	launch	shift	happen	happen	fr
18	comms	help	application	middle	big	occur	person	tab	randomly	miss	response	da
19	comcon	com	agent	switch	service	stop	live	correctly	comcon	unable	suggestion	lir

Inter-topic Distance Map generated by LDA gensim

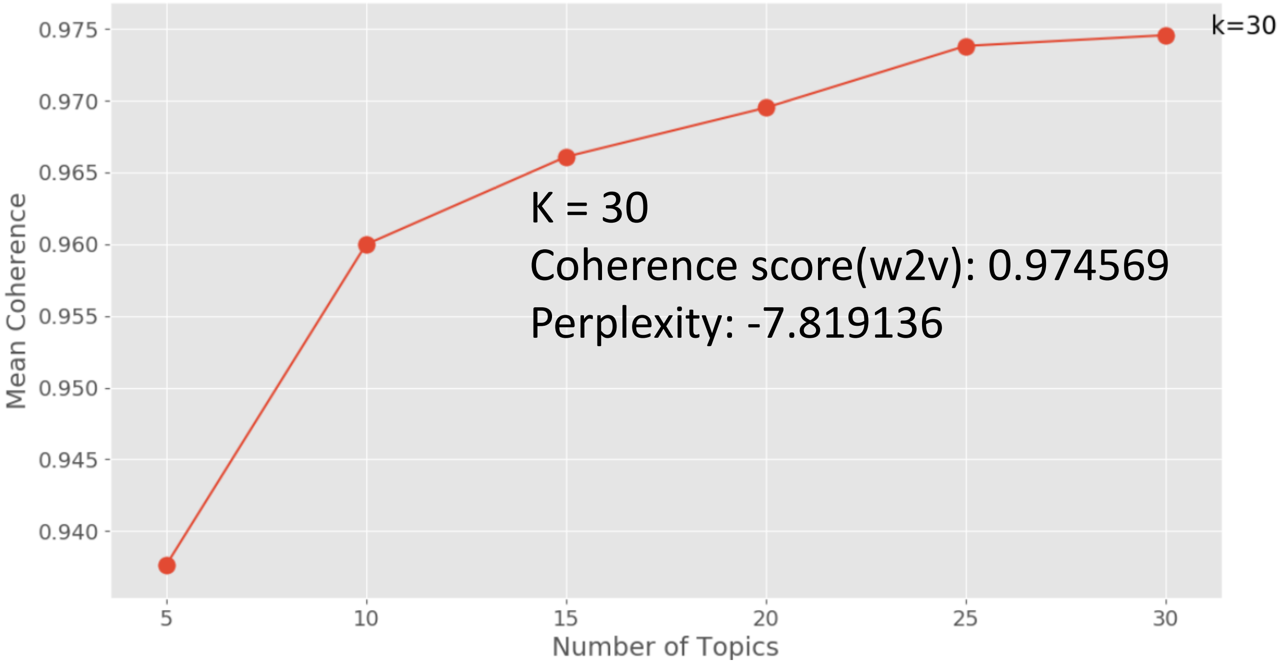




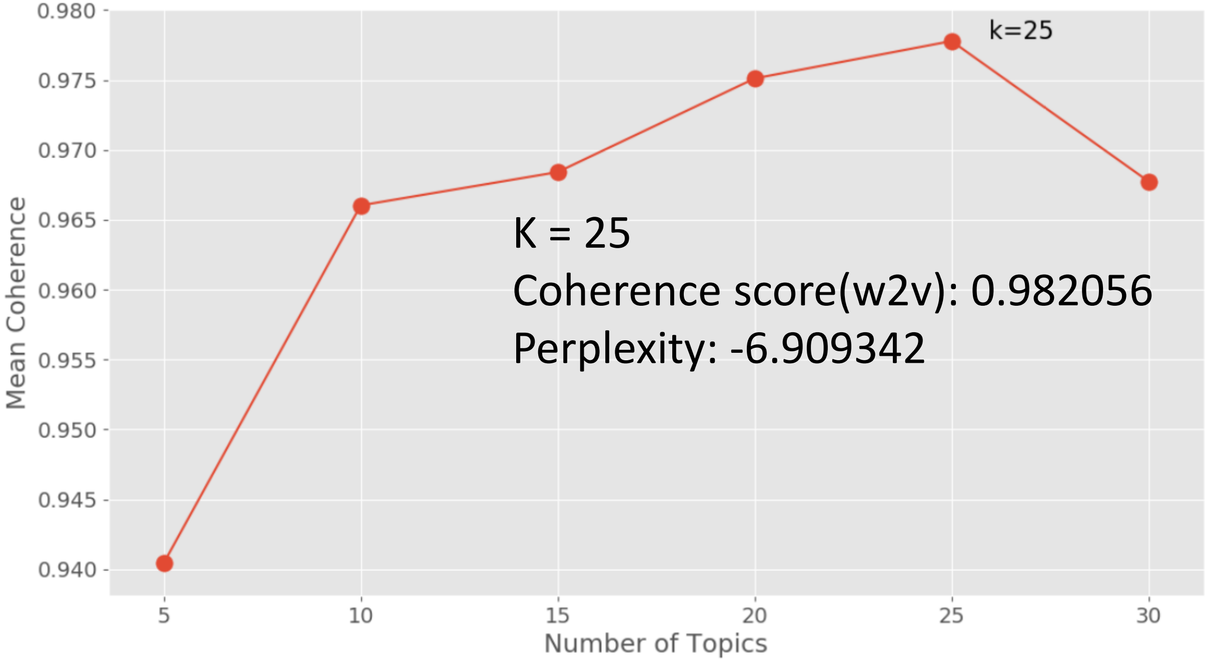
Segregation of Topic Clusters
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
Component 1

Comparison – lemmatizing or not in LDA gensim

Using text prior to lemmatization to create id2word and corpus



Using lemmatized text to create id2word and corpus



K=30

Coherence score(w2v): 0.967741

Perplexity: -6.954868

How to prepare for text?

- Text normalization
  - Clean up – remove emails, new line chars, single quotes, punctuations
  - Remove stop words
  - Lemmatize (only remain Noun, Adj, Adv., and Verb)
- Text vectorization
  - TF, or TFIDF
  - sklearn.feature_extraction.text.CountVectorizer
    - lowercase
    - analyzer=‘word’
    - token_pattern
    - min_df
  - sklearn.feature_extraction.text.TfidfVectorizer
    - stop_words
    - min_df

How to validate results?

Documents -> Topics -> Keywords

- Visualize the topics-keywords
- Find the dominant topic in each sentence
- Find the most representative document for each topic
- Topic distribution across documents

Need to be automated!!!

Evaluate topic modeling result

- Model evaluation
  - A model with higher log-likelihood, higher coherence score, and lower perplexity is considered to be good.
    - Log likelihood
    - Perplexity
    - Coherence score
      - u_mass
      - c_uci
      - c_v
      - w2v

- Comparison among models
  - Practical measure:
    - Topic prediction
    - u_mass, c_uci
    - c_v
    - w2v coherence score

      - Take average of similarity numbers (cosine) between every two words in each topic (Ti), and then take average Ti from all topics.

Best practice (1)

- Text preprocessing
  - Remove punctuations, special code text
  - Remove stopwords, or not
  - Lemmatizing instead of stemming
  - Noun/Adj/Adv/Verb, or Noun-only approach

- Text normalization
  - TFIDF, rather than TF

- Increase # of training iterations
  - LDA in sklearn (“max_iter”)
  - LDA in genism (“pass”)

Best practice (2)

- Performance awareness - Time complexity
  - Polynomial in NMF
  - Proportional to n_samples * iterations in LDA
    - ‘learning_method=online’ for large dataset

- Optimize model different # of topics
  - LDA in sklearn using GridSearchCV
  - other params, e.g. “learning_decay”
  - Add extra topics to collect topic noises

- Human interpretation aid
  - List topic/keywords, topic matrix
  - Visualize by word cloud, topic distribution, topic clustering

- Use metrics for quantitative comparison
  - higher log-likelihood, higher coherence score (“cv”, “u_mass”), and lower perplexity
  - Finding topics with high semantic coherence (use word embedding)

Tools & Functions

functions/tools	basic tools			
	NMF (sklearn)	LDA (sklearn)	LDA mallet (gensim)	LDA (gensim)
Key input params		learning_decay		
Model optimization	YES	GridSearchCV	YES	YES
List topics/top words	YES	YES	YES	YES
Topic matrix	YES	YES	x	YES
Word cloud	YES	YES	YES	YES
Visualize topic distribution	x	YES	x	YES
Coherence (w2v model)	YES	?	YES	YES
Coherence (c_s, u_mass)	YES	?	YES	YES
Perplexity	?	YES	?	YES
Log Likelyhood	?	YES	?	?
Predict topic(s)	YES	YES	YES	YES
Retrieve similar documents	YES	YES	YES	YES

Thoughts to improve topic modeling quality

- Use predefined topic list
- Use Noun-only approach
- Group NMF to scalability solution
- Use original text without stop words removal
- Dynamic Topic Modeling (a.k.a. DTM)
- Train model -> Clean up -> Re-Train model -> Clean up -> ……

Factors and Tips

- Corpora of texts have some unique characteristics
  - Text length
- Choice of a method depends on
  - The definition of "topics“ (high co-occurance, semantic-similarity)
  - The purpose of topic finding (representation of docs, summarization, outlier detection and etc.)
- It's usually a good idea to start with KMeans or NMF, and to quickly get a better understanding of the structures of texts, including but not limited to,
  - sparseness of words in topics
  - sparseness of topics in documents
  - number of topics
  - number of words in each topic
  - what does co-occurrance imply in your data
- LDA is a transformation from bag-of-words counts into a topic space of lower dimensionality. LDA is flexible for different types of tasks. But its parameter tuning should be based on a good understanding of the data. So if you want to try LDA, keep at least another model such as KMeans or NMF as a baseline.
- SVD is mostly useful to capture the variances in the texts. For example, if your data is semi-structured, e.g., forms of a template, screenshots, html tables, SVD might be useful in analyzing them when used together with regular expressions.

LDA and SVD

- LDA is one of the most mentioned due to:
  - its good performances on many different types of texts
  - its intuitive interpretation as a "generative" process.
  - Intuitively, LDA finds topics as a group of words that have high co-occurrences among different documents. On the other side, documents from the similar mixture of topics should also be similar, such that they can be described by these topics in a "compact" way. So ideally the similarity in the latent topic space would imply the the similarity in both the observed word space as well as the document space.
- The LDA algorithms has two main parameters controlling
  - how sparse the topics are in terms of the distribution of keywords in each topic
  - how sparse the documents are in terms of the distribution of topics in each document
- There is an upper limit of the # of topics generated by SVD due to its computation algorithm - using other vectorization method, such as tf/idf for n-grams or word embedding may help.
- SVD may have problems if you have texts that are mostly similar to each other, but their slight differences actually determine their topics.

NMF and KMeans

- NMF seems to work very well with short texts out-of-box.
  - NMF can be mostly seen as a LDA of which the parameters have been fixed to enforce a sparse solution. So it may not be as flexible as LDA if you want to find multiple topics in single documents, e.g., from long articles.
- NMF is usually cheaper in computation compared to LDA.
  - The main cons of NMF is its gradual inconsistency of results when keep increasing number of topics.when you set the number of topics to be too high than the reality in texts, NMF might generate some rubbish out of nowhere. LDA is more robust to a big variety of different topic numbers.
- KMeans: cheap and powerful
  - Clustering method such as KMeans can group documents based on their vector representations (or even directly based on their distance matrices). However it is not usually seen as a topic-finding method because it is hard to explain its results as groups of keywords. However, when used together with tf/tfidf, the centers of the clusters can be interpreted as a probability over words in the same way as in LDA and NMF.

- Text Preprocessing
    - Removing extra characters (quotes, punctuations, …) is a must??
    - Stop-word removal often has a major impact
    - TF-IDF often leads to more useful topics than raw term frequencies
    - Stemming, or Lemmatization ??
    - Only take NOUN, or ADJ, ADV?
- Initialization
    - Random initialization of both NMF and LDA can lead to unstable results, particularly for larger datasets
- Scalability
    - NMF typically more scalable than LDA, but running times can increase considerably as number of topics K increases
    - In "parameter selection" process, there can be several candidates of "good" K for many cases. The choice of coherence measure can produce different results.
- Interpretation
    - Topic models reflect the structure of the data available. Best uses carefully an exploratory tool to aid human interpretation. – increase human interpretability
- Interoperability

How?

1. LDA in sklearn
    - LatentDirichletAllocation
        - n_topics, max_iter, batch_size
        - learning_method, ……
    - To find the best model:
        - sklearn.model_selection.GridSearchCV
            - lda – from LatentDirichletAllocation()
            - param_grid= {‘n_components’: […], ‘learning_decay’: […]}

2. LDA in gensim
    - gensim.models.ldamodel.LdaModel
        - corpus, id2word,
        - num_topics, passes, chunk_size
        - decay, alpha, ……

3. LDA Mallet model

- gensim.models.wrappers.LdaMallet
  - corpus, num_topics, id2word

```
# Create Dictionary
id2word = genism.corpora.Dictionary(texts)

# Create Corpus
corpus = [id2word.doc2bow(text) for text in texts]
```

- Mallet
  - a Java-based package for statistical NLP, document classification, clustering, topic modeling, information extraction, and other ML apps.

  Tutorial of how to use Mallet topic modeling tool:
  https://programminghistorian.org/en/lessons/topic-modeling-and-mallet

  - uses Gibbs sampling

4. NMF in sklearn

- sklearn.decomposition.NMF
  - A matrix (from TfidfVectorizer and fit_transform)
  - Init, n_components, beta_loss

  http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html

5. SVD in sklearn

- sklearn.decomposition.TruncatedSVD

TfidfVectorizer:

- Convert a collection of raw documents to a matrix of TF-IDF features. Equivalent to **CountVectorizer** followed by **TfidfTransformer**.

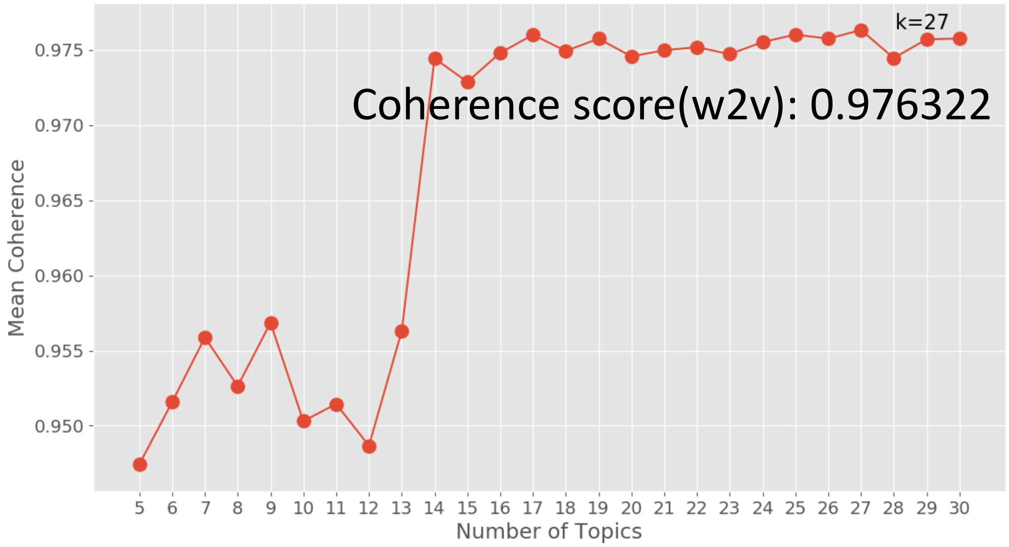
Fitting LDA models with tf features

http://scikit-learn.org/stable/auto_examples/applications/plot_topics_extraction_with_nmf_lda.html#sphx-glr-auto-examples-applications-plot-topics-extraction-with-nmf-lda-py

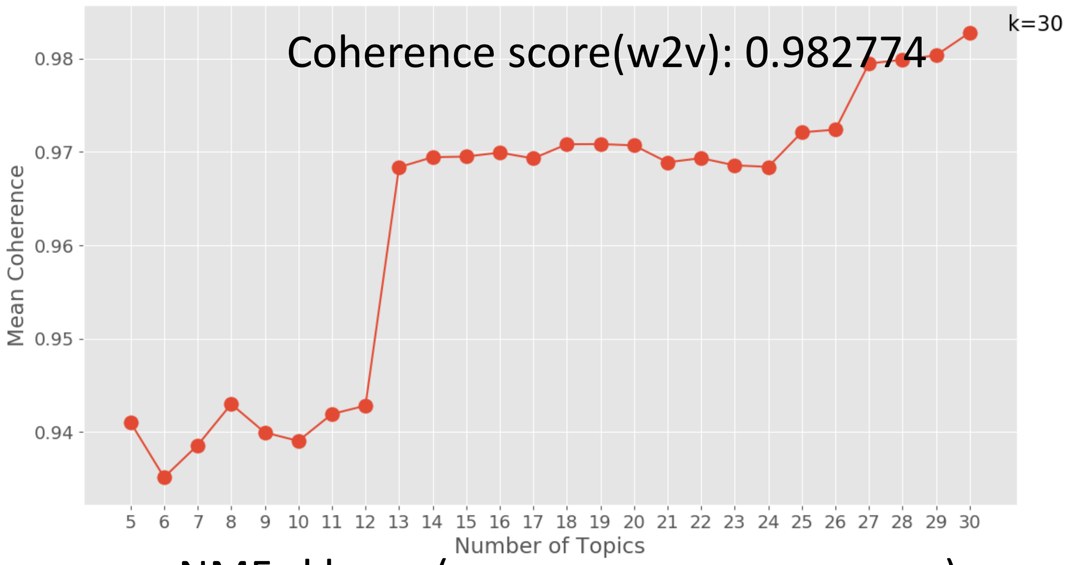
T**fidfTransformer**

- Transform a count matrix to a normalized tf or tf-idf representation

Optimizing # of topics



NMF sklearn (TfidfVectorizer)



NMF sklearn (TfidfVectorizer+TfidfTransformer)

LDA gensim (TfidfVectorizer)