14
Topic Modeling

In the last chapter, we converted unstructured text data into a numerical format using the bag-of-words model. This model abstracts from word order and represents documents as word vectors, where each entry represents the relevance of a token to the document.

The resulting **document-term matrix** (**DTM**), (you may also come across the transposed term-document matrix) is useful to compare documents to each other or to a query vector based on their token content, and quickly find a needle in a haystack or classify documents accordingly.

However, this document model is both high-dimensional and very sparse. As a result, it does little to summarize the content or get closer to understanding what it is about. In this chapter, we will use unsupervised machine learning in the form of topic modeling to extract hidden themes from documents. These themes can produce detailed insights into a large body of documents in an automated way. They are very useful to understand the haystack itself and permit the concise tagging of documents because using the degree of association of topics and documents.

Topic models permit the extraction of sophisticated, interpretable text features that can be used in various ways to extract trading signals from large collections of documents. They speed up the review of documents, help identify and cluster similar documents, and can be annotated as a basis for predictive modeling. Applications include the identification of key themes in company disclosures, or earnings call transcripts, customer reviews or contracts, annotated using, for example, sentiment analysis or direct labeling with subsequent asset returns.

More specifically, in this chapter, we will cover these topics:

- What topic modeling achieves, why it matters, and how it has evolved
- How **Latent Semantic Indexing** (**LSI**) reduces the dimensionality of the DTM
- How **probabilistic Latent Semantic Analysis** (**pLSA**) uses a generative model to extract topics

- How **Latent Dirichlet Allocation** (**LDA**) refines pLSA and why it is the most popular topic model
- How to visualize and evaluate topic modeling results
- How to implement LDA using sklearn and gensim
- How to apply topic modeling to collections of earnings calls and Yelp business reviews

The code samples for the following sections are in the directory of the GitHub repository for this chapter, and references are listed in the main README file.

Learning latent topics: goals and approaches

Topic modeling aims to discover hidden topics or themes across documents that capture semantic information beyond individual words. It aims to address a key challenge in building a machine learning algorithm that learns from text data by going beyond the lexical level of what has been written to the semantic level of what was intended. The resulting topics can be used to annotate documents based on their association with various topics.

In other words, topic modeling aims to automatically summarize large collections of documents to facilitate organization and management, as well as search and recommendations. At the same time, it can enable the understanding of documents to the extent that humans can interpret the descriptions of topics.

Topic models aim to address the curse of dimensionality that can plague the bag-of-words model. The document representation based on high-dimensional sparse vectors can make similarity measures noisy, leading to inaccurate distance measurement and overfitting of text classification models.

Moreover, the bag of words model ignores word order and loses context as well as semantic information because it is not able to capture synonymy (several words have the same meaning) and polysemy (one word has several meanings). As a result, document retrieval or similarity search may miss the point when the documents are not indexed by the terms used to search or compare.

These shortcoming prompt this question: how do we model and learn meaning topics that facilitate a more productive interaction with text data?

From linear algebra to hierarchical probabilistic models

Initial attempts by topic models to improve on the vector space model (developed in the mid-1970s) applied linear algebra to reduce the dimensionality of the document-term matrix. This approach is similar to the algorithm we discussed as principal component analysis in `Chapter 12`, *Unsupervised Learning*, on unsupervised learning. While effective, it is difficult to evaluate the results of these models absent a benchmark model.

In response, probabilistic models emerged that assume an explicit document generation process and provide algorithms to reverse engineer this process and recover the underlying topics.

This table highlights key milestones in the model evolution that we will address in more detail in the following sections:

Model	Year	Description
Latent Semantic Indexing (**LSI**)	1988	Reduces the word space dimensionality to capture semantic document-term relationships by
Probabilistic Latent Semantic Analysis (**pLSA**)	1999	Reverse-engineers a process that assumes words generate a topic and documents are a mix of topics
Latent Dirichlet Allocation (**LDA**)	2003	Adds a generative process for documents: a three-level hierarchical Bayesian model

Latent semantic indexing

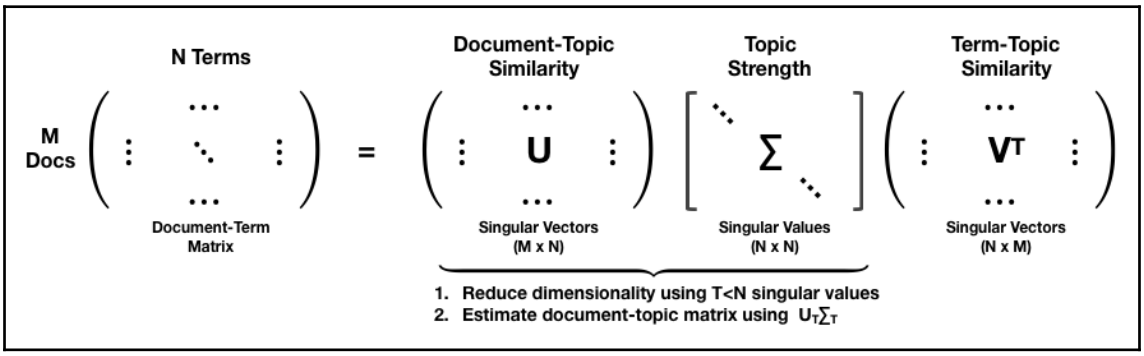
Latent Semantic Indexing (LSI, also called Latent Semantic Analysis) sets out to improve the results of queries that omitted relevant documents containing synonyms of query terms. It aims to model the relationships between documents and terms to be able to predict that a term should be associated with a document, even though, because of variability in word use, no such association was observed.

LSI uses linear algebra to find a given number, *k*, of latent topics by decomposing the DTM. More specifically, it uses **Singular Value Decomposition** (**SVD**) to find the best lower-rank DTM approximation using k singular values and vectors. In other words, LSI is an application of the unsupervised learning techniques of dimensionality reduction we encountered in `Chapter 12`, *Unsupervised Learning* to the text representation that we covered in `Chapter 13`, *Working with Text Data*. The authors experimented with hierarchical clustering but found it too restrictive to explicitly model the document-topic and topic-term relationships, or capture associations of documents or terms with several topics.

In this context, SVD serves the purpose of identifying a set of uncorrelated indexing variables or factors that permit us to represent each term and document by its vector of factor values.

The following figure illustrates how SVD decomposes the DTM into three matrices, two containing orthogonal singular vectors and a diagonal matrix with singular values that serve as scaling factors. Assuming some correlation in the original data, singular values decay in value so that selecting only the largest T singular values produces a lower-dimensional approximation of the original DTM that loses relatively little information. Hence, in the reduced version the rows or columns that had N items only have $T<N$ entries.

This reduced decomposition can be interpreted as illustrated next, where the first $M \times T$ matrix represents the relationships between documents and topics, the diagonal matrix scales the topics by their corpus strength, and the third matrix models the term-topic relationship:



The rows of the matrix that results from the product of the first two matrices, $U_T\Sigma_T$, corresponds to the locations of the original documents projected into the latent topic space.

How to implement LSI using sklearn

We will illustrate the application of LSI using the BBC article data that we introduced in the last chapter because it is small enough to permit quick training and allow us to compare topic assignments to category labels. See the `latent_semantic_indexing` notebook for additional implementation details:

1. We begin by loading the documents and creating a train and (stratified) test set with 50 articles.
2. Then, we vectorize the data using `TfidfVectorizer` to obtain weighted DTM counts and filter out words that appear in less than 1% or more than 25% of the documents, as well as generic stopwords, to obtain a vocabulary of around 2,900 words:

```
vectorizer = TfidfVectorizer(max_df=.25, min_df=.01,
stop_words='english',
binary=False)
train_dtm = vectorizer.fit_transform(train_docs.article)
test_dtm = vectorizer.transform(test_docs.article)
```

3. We use `sklearn`'s `TruncatedSVD` class, which only computes the k largest singular values to reduce the dimensionality of the document-term matrix. The deterministic arpack algorithm delivers an exact solution, but the default randomized implementation is more efficient for large matrices.
4. We compute five topics to match the five categories, which explain only 5.4% of the total DTM variance so higher values would be reasonable:

```
svd = TruncatedSVD(n_components=5, n_iter=5, random_state=42)
svd.fit(train_dtm)
svd.explained_variance_ratio_
array([0.00187014, 0.01559661, 0.01389952, 0.01215842, 0.01066485])
```

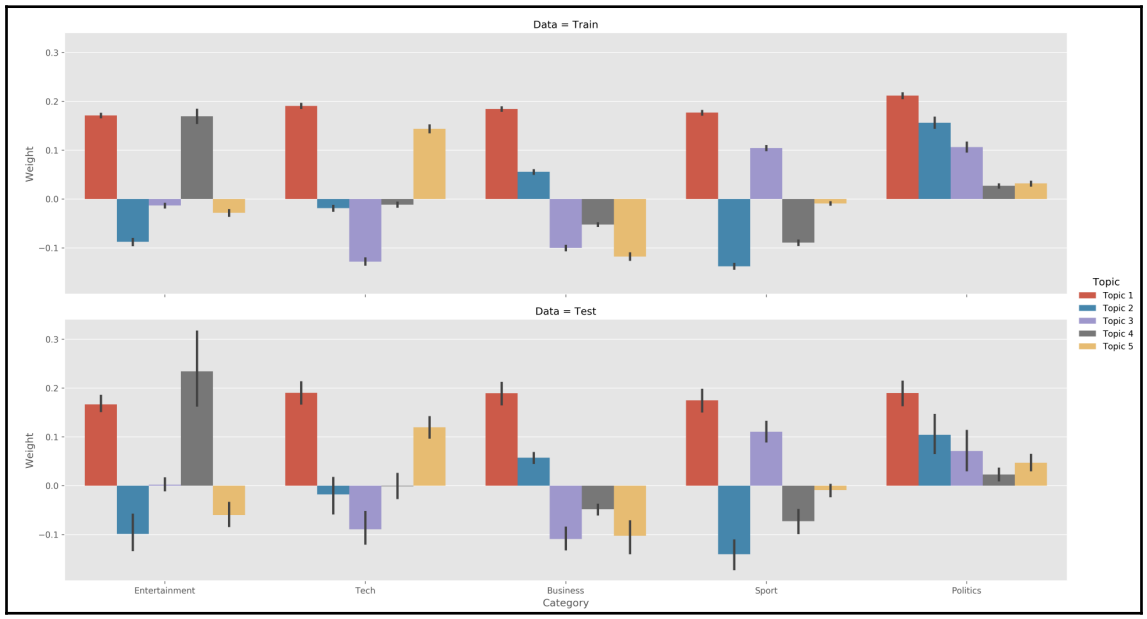
5. LSI identifies a new orthogonal basis for the document-term matrix that reduces the rank to the number of desired topics.
6. The `.transform()` method of the trained `svd` object projects the documents into the new topic space that is the result of reducing the dimensionality of the document vectors and corresponds to the $U_T\Sigma_T$ transformation illustrated before:

```
train_doc_topics = svd.transform(train_dtm)
train_doc_topics.shape
(2175, 5)
```

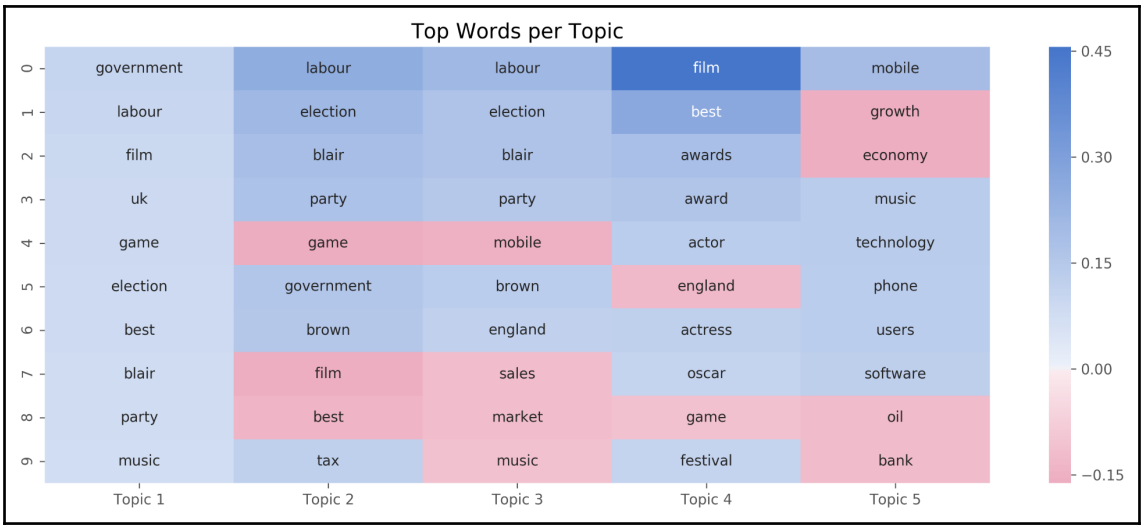
7. We can sample an article to view its location in the topic space. We draw a `Politics` article that is most (positively) associated with topics 1 and 2:

```
i = randint(0, len(train_docs))
train_docs.iloc[i, :2].append(pd.Series(doc_topics[i],
index=topic_labels))
Category Politics
Heading What the election should really be about?
Topic 1 0.33
Topic 2 0.18
Topic 3 0.12
Topic 4 0.02
Topic 5 0.06
```

8. The topic assignments for this sample align with the average topic weights for each category illustrated next (`Politics` is the leftmost). They illustrate how LSI expresses the k topics as directions in a k-dimensional space (the notebook includes a projection of the average topic assignments per category into two-dimensional space).
9. Each category is clearly defined, and the test assignments match with train assignments. However, the weights are both positive and negative, making it more difficult to interpret the topics:



10. We can also display the words that are most closely associated with each topic (in absolute terms). The topics appear to capture some semantic information but are not differentiated:



Pros and cons

The benefits of LSI include the removal of noise and mitigation of the curse of dimensionality, while also capturing some semantics and clustering both documents and terms.

However, the results of LSI are difficult to interpret because topics are word vectors with both positive and negative entries. There is also no underlying model that would permit the evaluation of fit and provide guidance when selecting the number of dimensions or topics.

Probabilistic latent semantic analysis

Probabilistic Latent Semantic Analysis (pLSA) takes a statistical perspective on LSA and creates a generative model to address the lack of theoretical underpinnings of LSA.

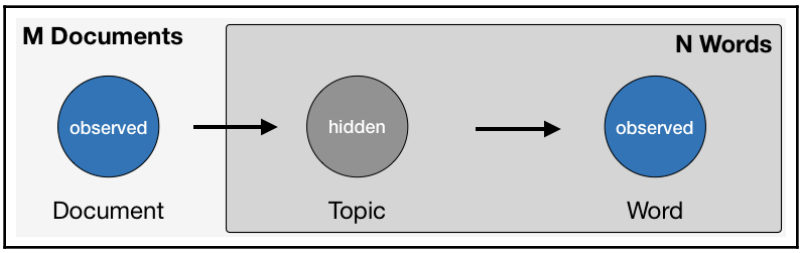
pLSA explicitly models the probability each co-occurrence of documents d and words w described by the DTM as a mixture of conditionally independent multinomial distributions that involve topics t.

The symmetric formulation of this generative process of word-document co-occurrences assumes both words and documents are generated by the latent topic class, whereas the asymmetric model assumes the topics are selected given the document, and words result from a second step given the topic:

$$P(w,d) = \underbrace{\sum_t P(d \mid t)P(w \mid t)}_{\text{symmetric}} = \underbrace{P(d)\sum_t P(t \mid d)P(w \mid t)}_{\text{asymmetric}}$$

The number of topics is a hyperparameter chosen before training and is not learned from the data.

Probabilistic models often use the following plate notation to express dependencies. The following figure encodes the relationships just describe for the asymmetric model. Each rectangle represents multiple items, such as **M Documents** for the outer and **N Words** for each document for the inner block. We only observe the documents and their content, and the model infers the hidden or latent topic distribution:



The benefit of using a probability model is that we can now compare models by evaluating the probability they assign to new documents given the parameters learned during training.

How to implement pLSA using sklearn

pLSA is equivalent to non-negative matrix factorization using a Kullback-Leibler Divergence objective (see references on GitHub `https://github.com/PacktPublishing/Hands-On-Machine-Learning-for-Algorithmic-Trading`). Hence, we can use the `sklearn.decomposition.NM` class to implement this model, following the LSA example.

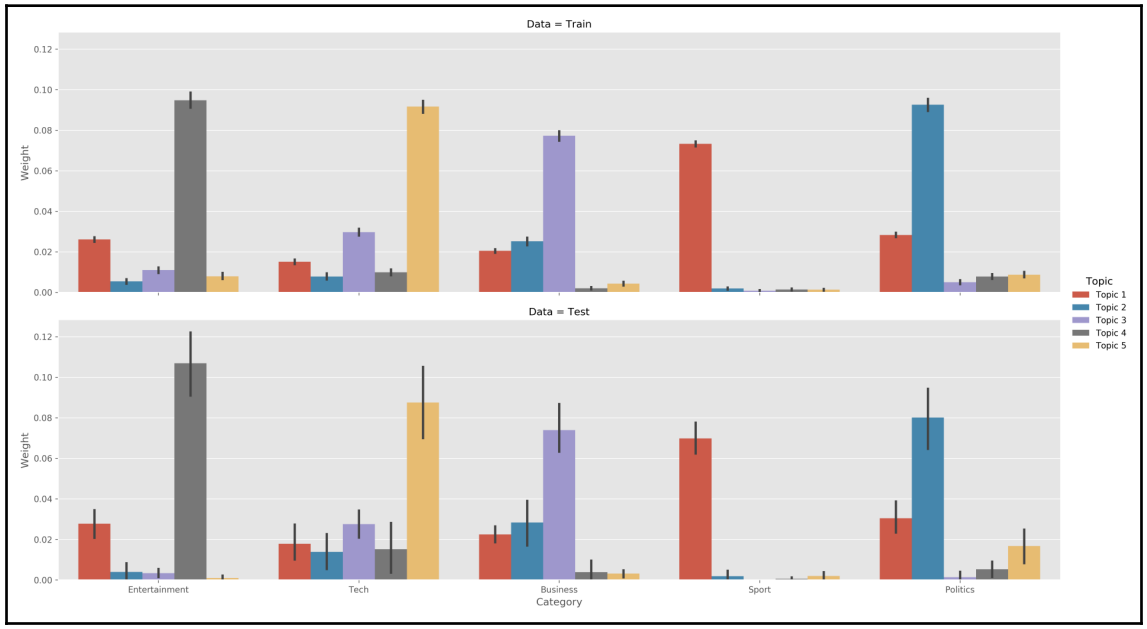
Using the same train-test split of the DTM produced by the `TfidfVectorizer`, we fit pLSA as follows:

```
nmf = NMF(n_components=n_components,
random_state=42,
solver='mu',
beta_loss='kullback-leibler',
max_iter=1000)
nmf.fit(train_dtm)
```

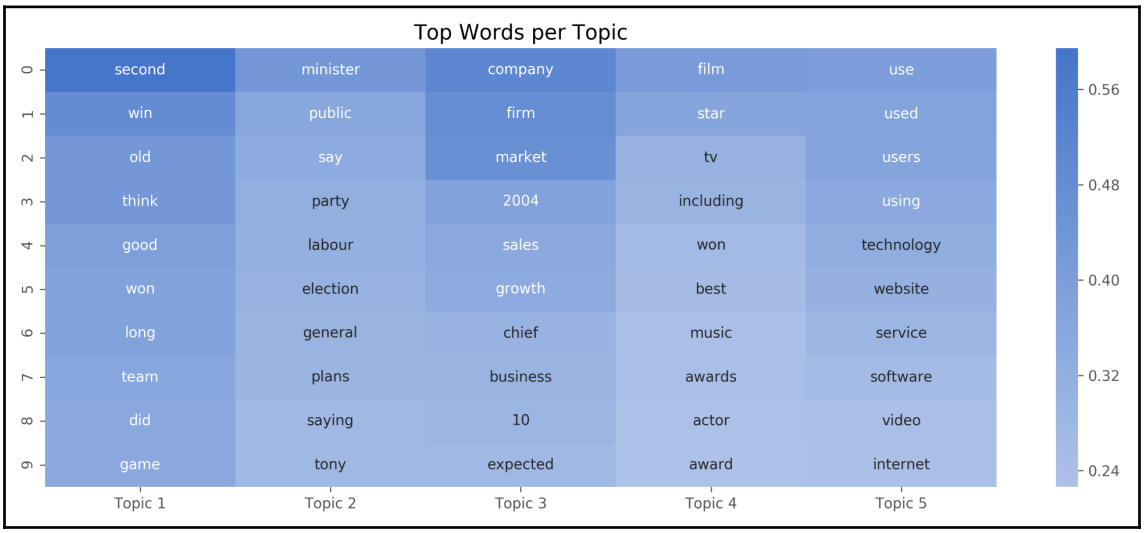
We get a measure of the reconstruction error, which is a substitute for the explained variance measure from before:

```
nmf.reconstruction_err_
316.2609400385988
```

Due to its probabilistic nature, pLSA produces only positive topic weights that result in more straightforward topic-category relationships for the test and training sets:



We can also see that the word lists that describe each topic begin to make more sense; for example, the **Entertainment** category is most directly associated with **Topic 4**, which includes the words **film**, **start**, and so on:



Latent Dirichlet allocation

Latent Dirichlet allocation (**LDA**) extends pLSA by adding a generative process for topics.

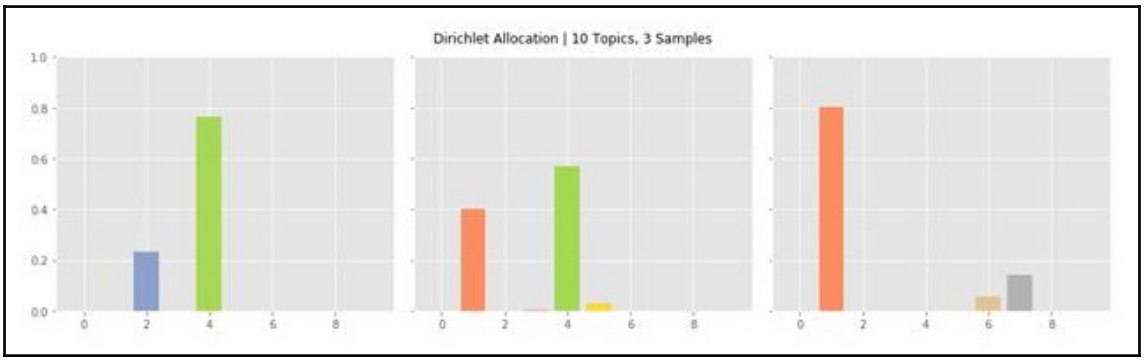
It is the most popular topic model because it tends to produce meaningful topics that humans can relate to, can assign topics to new documents, and is extensible. Variants of LDA models can include metadata such as authors, or image data, or learn hierarchical topics.

How LDA works

LDA is a hierarchical Bayesian model that assumes topics are probability distributions over words, and documents are distributions over topics. More specifically, the model assumes that topics follow a sparse Dirichlet distribution, which implies that documents cover only a small set of topics, and topics use only a small set of words frequently.

The Dirichlet distribution

The Dirichlet distribution produces probability vectors that can be used with discrete distributions. That is, it randomly generates a given number of values that are positive and sum to one as expected for probabilities. It has a parameter of positive, real value that controls the concentration of the probabilities. Values closer to zero mean that only a few values will be positive and receive most probability mass. The following screenshot illustrates three draws of size 10 for $\alpha = 0.1$ (the `dirichlet_distribution` notebook contains a simulation so you can experiment with different parameter values):



Dirichlet allocation

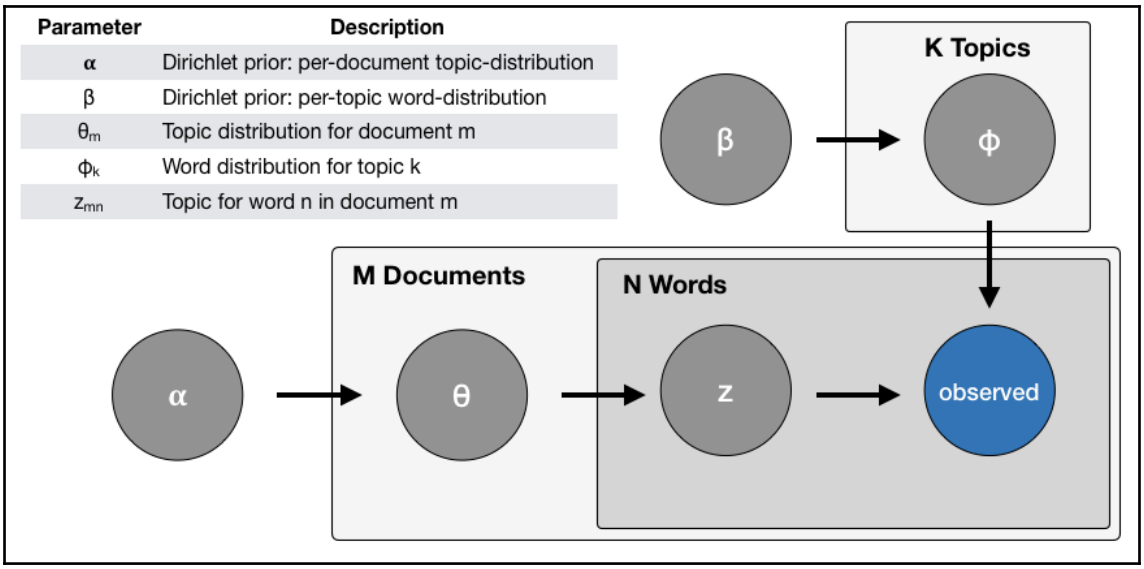
The generative model

The Dirichlet distribution figures prominently in the LDA topic model, which assumes the following generative process when an author adds an article to a body of documents:

1. Randomly mix a small subset of shared topics *K* according to the topic probabilities
2. For each word, select one of the topics according to the document-topic probabilities
3. Select a word from the topic's word list according to the topic-word probabilities

As a result, the article content depends on the weights of each topic and on the terms that make up each topic. The Dirichlet distribution governs the selection of topics for documents and words for topics and encodes the idea that a document only covers a few topics, while each topic uses only a small number of words frequently.

The plate notation for the LDA model summarizes these relationships:



Reverse-engineering the process

The generative process is fictional but turns out to be useful because it permits the recovery of the various distributions. The LDA algorithm reverse-engineers the work of the imaginary author and arrives at a summary of the document-topic-word relationships that concisely describes the following:

- The percentage contribution of each topic to a document
- The probabilistic association of each word with a topic

LDA solves the Bayesian inference problem of recovering the distributions from the body of documents and the words they contain by reverse-engineering the assumed content generation process. The original paper uses **variational Bayes** (**VB**) to approximate the posterior distribution. Alternatives include Gibbs sampling and expectation propagation. Later, we will illustrate implementations using the sklearn and gensim libraries.

How to evaluate LDA topics

Unsupervised topic models do not provide a guarantee that the result will be meaningful or interpretable, and there is no objective metric to assess the result as in supervised learning. Human topic evaluation is considered the gold standard but is potentially expensive and not readily available at scale.

Two options to evaluate results more objectively include perplexity, which evaluates the model on unseen documents, and topic coherence metrics, which aim to evaluate the semantic quality of the uncovered patterns.

Perplexity

Perplexity, when applied to LDA, measures how well the topic-word probability distribution recovered by the model predicts a sample, for example, unseen text documents. It is based on the entropy $H(p)$ of this distribution p and computed with respect to the set of tokens w:

$$2^{H(p)} = 2^{-\sum_w p(w) \log_2 p(w)}$$

Measures closer to zero imply the distribution is better at predicting the sample.

Topic coherence

Topic coherence measures the semantic consistency of the topic model results, that is, whether humans would perceive the words and their probabilities associated with topics as meaningful.

To this end, it scores each topic by measuring the degree of semantic similarity between the words most relevant to the topic. More specifically, coherence measures are based on the probability of observing the set of words W that define a topic together.

We use two measures of coherence that have been designed for LDA and shown to align with human judgment of topic quality, namely the UMass and the UCI measures.

The UCI metric defines a word pair's score to be the sum of the **Pointwise Mutual Information** (**PMI**) between two distinct pairs of (top) topic words w_i, $w_j \in w$ and a smoothing factor ε:

$$\text{coherence}_{\text{UCI}} = \sum_{(w_i, w_j) \in W} \log \frac{p(w_i, w_j) + \epsilon}{p(w_i)p(w_j)}$$

The probabilities are computed from word co-occurrence frequencies in a sliding window over an external corpus such as Wikipedia, so that this metric can be thought of as an external comparison to a semantic ground truth.

In contrast, the UMass metric uses the co-occurrences in a number of documents *D* from the training corpus to compute a coherence score:

$$\text{coherence}_{\text{UMass}} = \sum_{(w_i, w_j) \in W} \log \frac{D(w_i, w_j) + \epsilon}{D(w_j)}$$

Rather than a comparison to an extrinsic ground truth, this measure reflects intrinsic coherence. Both measures have been evaluated to align well with human judgment. In both cases, values closer to zero imply that a topic is more coherent.

How to implement LDA using sklearn

Using the BBC data as before, we use `sklearn.decomposition.LatentDirichletAllocation` to train an LDA model with five topics (see the sklearn documentation for detail on parameters, and the notebook `lda_with_sklearn` for implementation details):

```
lda = LatentDirichletAllocation(n_components=5,
                                n_jobs=-1,
                                max_iter=500,
                                learning_method='batch',
                                evaluate_every=5,
                                verbose=1,
                                random_state=42)
ldat.fit(train_dtm)
LatentDirichletAllocation(batch_size=128, doc_topic_prior=None,
             evaluate_every=5, learning_decay=0.7, learning_method='batch',
             learning_offset=10.0, max_doc_update_iter=100, max_iter=500,
```

```
                mean_change_tol=0.001, n_components=5, n_jobs=-1,
                n_topics=None, perp_tol=0.1, random_state=42,
                topic_word_prior=None, total_samples=1000000.0, verbose=1)
```

The model tracks the in-sample perplexity during training and stops iterating once this measure stops improving. We can persist and load the result as usual with sklearn objects:

```
joblib.dump(lda, model_path / 'lda.pkl')
lda = joblib.load(model_path / 'lda.pkl')
```

How to visualize LDA results using pyLDAvis

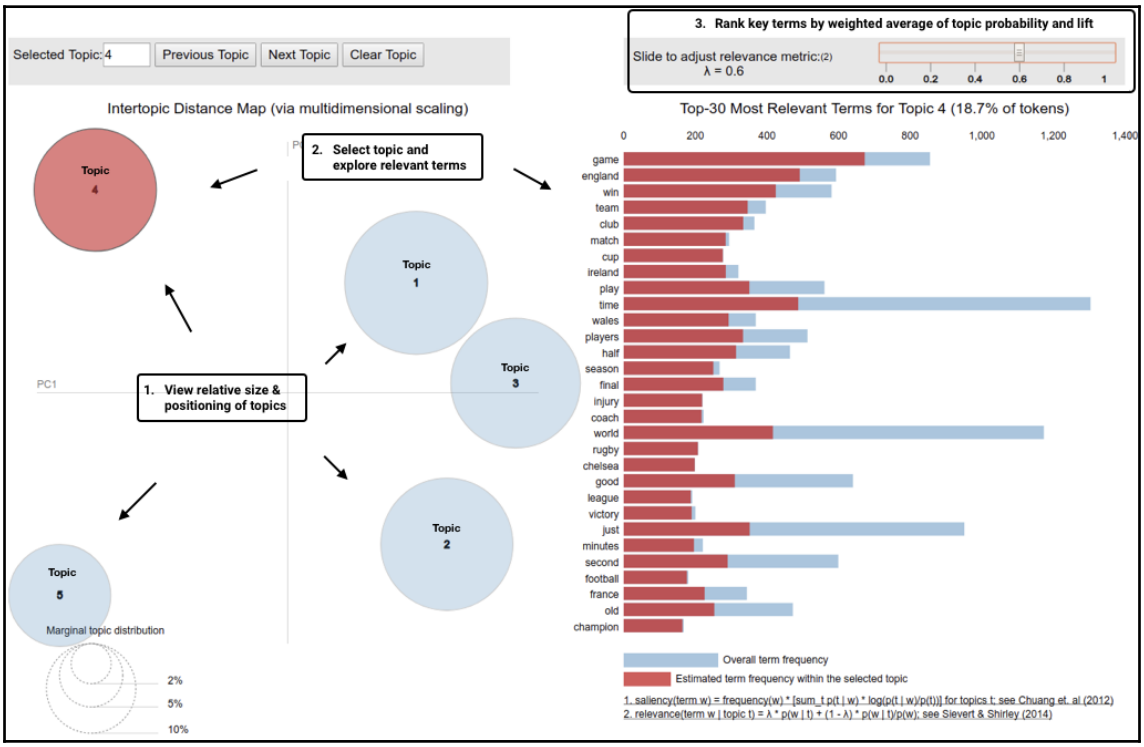
Topic visualization facilitates the evaluation of topic quality using human judgment. pyLDAvis is a Python port of LDAvis, developed in R and `D3.js`. We will introduce the key concepts; each LDA implementation notebook contains examples.

pyLDAvis displays the global relationships between topics while also facilitating their semantic evaluation by inspecting the terms most closely associated with each topic and, inversely, the topics associated with each term. It also addresses the challenge that terms that are frequent in a corpus tend to dominate the multinomial distribution over words that define a topic. LDAVis introduces the relevance *r* of the term *w* to topic *t*, to produce a flexible ranking of key terms using a weight parameter *0<=ƛ<=1*.

With ϕ_{wt} as the model's probability estimate of observing the term w for topic *t*, and as the marginal probability of w in the corpus:

$$r(w, k|\,\lambda) = \lambda log(\phi_{kw}) + (1 - \lambda) log \frac{\phi kw}{p_w}$$

The first term measures the degree of association of term t with topic w, and the second term measures the lift or saliency, that is, how much more likely the term is for the topic than in the corpus.



Topic 14

The tool allows the user to interactively change λ to adjust the relevance, which updates the ranking of terms. User studies have found that $\lambda=0.6$ produces the most plausible results.

How to implement LDA using gensim

`gensim` is a specialized NLP library with a fast LDA implementation and many additional features. We will also use it in the next chapter on word vectors (see the `latent_dirichlet_allocation_gensim` notebook for details).

It facilitates the conversion of DTM produced by sklearn into gensim data structures as follows:

```
train_corpus = Sparse2Corpus(train_dtm, documents_columns=False)
test_corpus = Sparse2Corpus(test_dtm, documents_columns=False)
id2word = pd.Series(vectorizer.get_feature_names()).to_dict()
```

Gensim LDA algorithm includes numerous settings, which are as follows:

```
LdaModel(corpus=None,
         num_topics=100,
         id2word=None,
         distributed=False,
         chunksize=2000, # No of doc per training chunk.
         passes=1,       # No of passes through corpus during training
         update_every=1, # No of docs to be iterated through per update
         alpha='symmetric',
         eta=None,       # a-priori belief on word probability
         decay=0.5,    # % of lambda forgotten when new doc is examined
         offset=1.0,   # controls slow down of first few iterations.
         eval_every=10,   # how often estimate log perplexity (costly)
         iterations=50,         # Max. of iterations through the corpus
         gamma_threshold=0.001, # Min. change in gamma to continue
         minimum_probability=0.01, # Filter topics with lower
                                     probability
         random_state=None,
         ns_conf=None,
         minimum_phi_value=0.01, # lower bound on term probabilities
         per_word_topics=False,  #  Compute most word-topic
                                    probabilities
         callbacks=None,
         dtype=<class 'numpy.float32'>)
```

Gensim also provides an `LdaMulticore` model for parallel training that may speed up training using Python's multiprocessing features for parallel computation.

Model training just requires instantiating the `LdaModel` object as follows:

```
lda = LdaModel(corpus=train_corpus,
num_topics=5,
id2word=id2word)
```

Topic coherence measures whether the words in a topic tend to co-occur together. It adds up a score for each distinct pair of top-ranked words. The score is the log of the probability that a document containing at least one instance of the higher-ranked word also contains at least one instance of the lower-ranked word.

Large negative values indicate words that don't co-occur often; values closer to zero indicate that words tend to co-occur more often. `gensim` permits topic coherence evaluation that produces the topic coherence and shows the most important words per topic:

```
coherence = lda_gensim.top_topics(corpus=train_corpus,  coherence='u_mass')
```

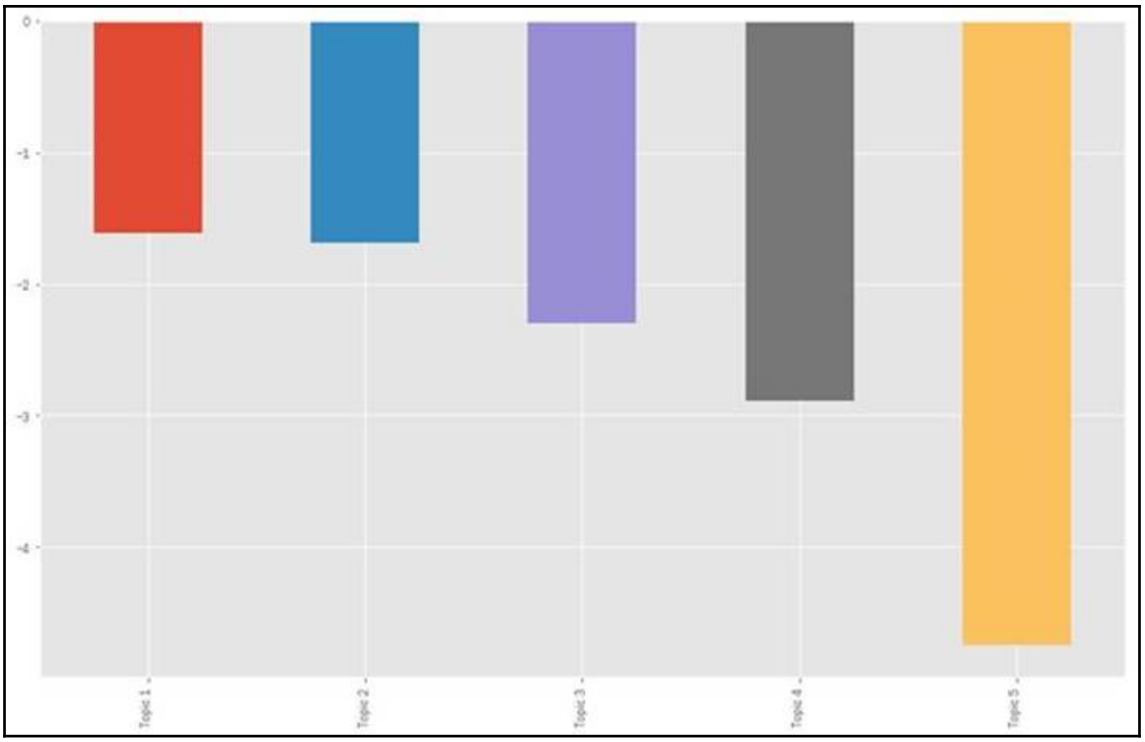
We can display the results as follows:

```
topic_coherence = []
topic_words = pd.DataFrame()
for t in range(len(coherence)):
    label = topic_labels[t]
    topic_coherence.append(coherence[t][1])
    df = pd.DataFrame(coherence[t][0], columns=[(label, 'prob'), (label,
'term')])
    df[(label, 'prob')] = df[(label, 'prob')].apply(lambda x:
'{:.2%}'.format(x))
    topic_words = pd.concat([topic_words, df], axis=1)
topic_words.columns = pd.MultiIndex.from_tuples(topic_words.columns)
pd.set_option('expand_frame_repr', False)
print(topic_words.head())
pd.Series(topic_coherence, index=topic_labels).plot.bar();
```

This shows the following top words for each topic:

Topic 1		**Topic 2**		**Topic 3**		**Topic 4**		**Topic 5**	
Probability	Term	Probability	Term	Probability	Term	Probability	Term	Probability	Term
0.55%	online	0.90%	best	1.04%	mobile	0.64%	market	0.94%	labour
0.51%	site	0.87%	game	0.98%	phone	0.53%	growth	0.72%	blair
0.46%	game	0.62%	play	0.51%	music	0.52%	sales	0.72%	brown
0.45%	net	0.61%	won	0.48%	film	0.49%	economy	0.65%	election
0.44%	used	0.56%	win	0.48%	use	0.45%	prices	0.57%	united

And the corresponding coherence scores, which highlight the decay of topic quality (at least in part due to the relatively small dataset):



Decay of topic quality

Topic modeling for earnings calls

In `Chapter 3`, *Alternative Data for Finance*, we learned how to scrape earnings call data from the SeekingAlpha site. In this section, we will illustrate topic modeling using this source. I'm using a sample of some 500 earnings call transcripts from the second half of 2018. For a practical application, a larger dataset would be highly desirable. The `earnings_calls` directory contains several files, with examples mentioned later.

See the `lda_earnings_calls` notebook for details on loading, exploring, and preprocessing the data, as well as training and evaluating individual models, and the `run_experiments.py` file for the experiments described here.

Data preprocessing

The transcripts consist of individual statements by a company representative, an operator, and usually a question and answer session with analysts. We will treat each of these statements as separate documents, ignoring operator statements, to obtain 22,766 items with mean and median word counts of 144 and 64, respectively:

```
documents = []
for transcript in earnings_path.iterdir():
    content = pd.read_csv(transcript / 'content.csv')
    documents.extend(content.loc[(content.speaker!='Operator') &
(content.content.str.len() > 5), 'content'].tolist())
len(documents)
22766
```

We use `spaCy` to preprocess these documents as illustrated in `Chapter 13`, *Working with Text Data* (see the notebook) and store the cleaned and lemmatized text as a new text file.

Data exploration reveals domain-specific stopwords such as year and quarter that we remove in a second step, where we also filter out statements with fewer than ten words so that some 16,150 remain.

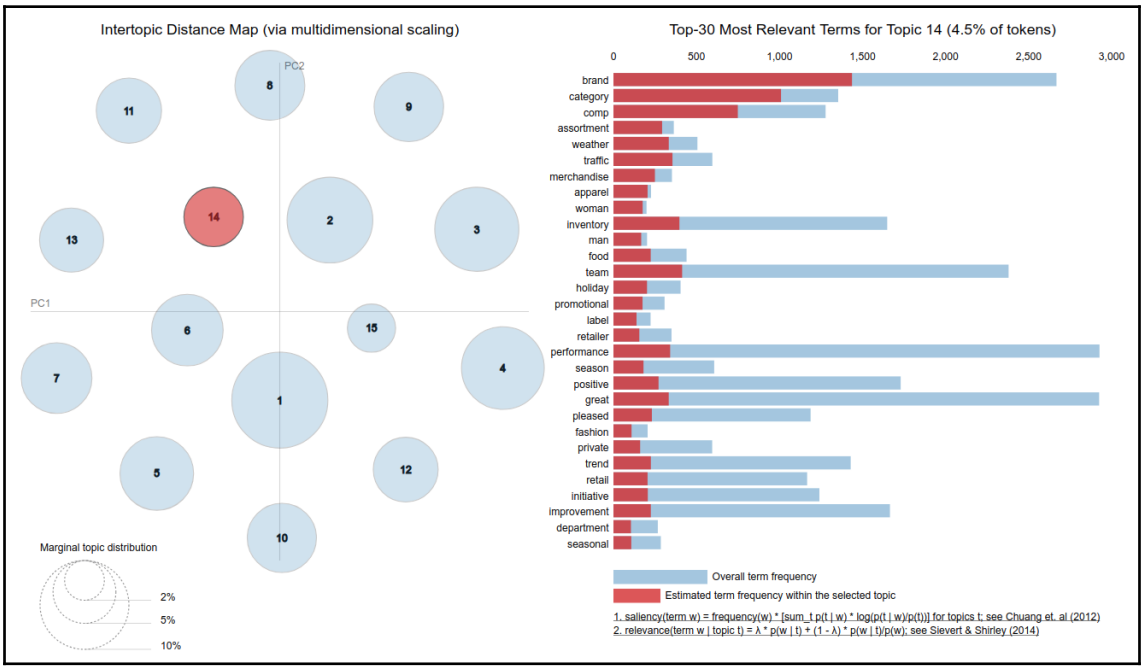
Model training and evaluation

For illustration, we will create a document-term matrix containing terms appearing in between 0.5% and 50% of documents for around 1,560 features. Training a 15-topic model using 25 passes over the corpus takes a bit over two minutes on a four-core i7.

The top 10 words per topic identify several distinct themes that range from obvious financial information to clinical trials (topic 4) and supply chain issues (12):

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	statement	expense	cloud	basis	patient	channel	focus	brand	want	project	lot	yes	price	maybe	bit
1	financial	total	technology	adjust	study	brand	acquisition	category	right	capital	thing	kind	production	kind	little
2	release	period	service	guidance	program	launch	investment	comp	price	investment	right	little	demand	okay	china
3	risk	income	solution	ebitda	clinical	experience	improve	team	thing	asset	mean	bit	volume	guess	loan
4	gaap	loss	platform	tax	trial	marketing	deliver	inventory	need	debt	actually	half	low	guy	service
5	officer	approximately	datum	low	phase	online	strategy	traffic	contract	portfolio	yes	pretty	capacity	sort	bank
6	chief	non	large	billion	datum	platform	invest	performance	lot	value	people	guidance	fleet	want	credit
7	conference	month	team	earning	development	digital	value	weather	say	balance	way	say	order	just	mention
8	measure	gaap	industry	gross	fda	consumer	progress	great	sure	return	different	thing	vessel	follow	tier
9	information	decrease	provide	approximately	cancer	user	performance	assortment	great	flow	obviously	low	supply	wonder	card

Using pyLDAvis' relevance metric with a 0.6 weighting of unconditional frequency relative to lift, topic definitions become more intuitive, as illustrated for topic 14 about sales performance:



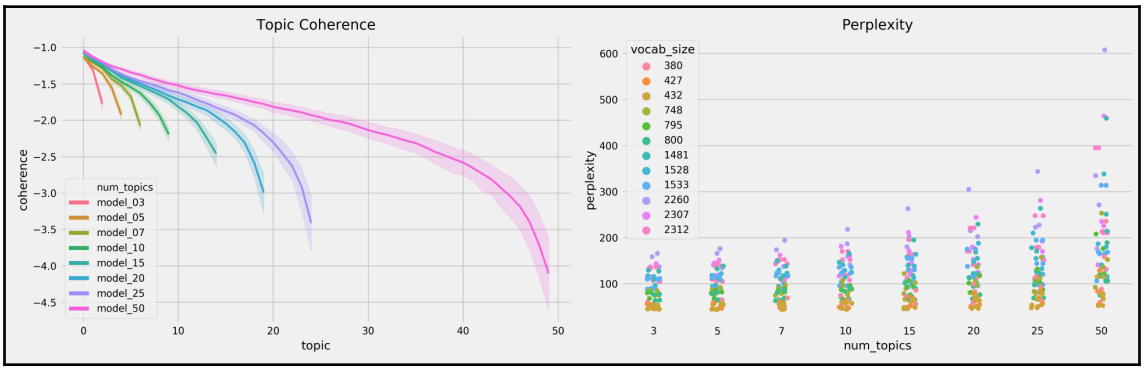
Sales performance for Topic 14

The notebook also illustrates how to look up documents by their topic association. In this case, an analyst can review relevant statements for nuances, use sentiment analysis to further process the topic-specific text data, or assign labels derived from market prices.

Running experiments

To illustrate the impact of different parameter settings, we ran a few hundred experiments for different DTM constraints and model parameters. More specifically, we let the `min_df` and `max_df` parameters range from 50-500 words and 10% to 100% of documents, respectively using alternatively binary and absolute counts. We then trained LDA models with 3 to 50 topics, using 1 and 25 passes over the corpus.

The following chart illustrates the results in terms of topic coherence (higher is better), and perplexity (lower is better). Coherence drops after 25-30 topics and perplexity similarly increases:



The notebook includes regression results that quantify the relationships between parameters and outcomes. We generally get better results using absolute counts and a smaller vocabulary.

Topic modeling for Yelp business reviews

The `lda_yelp_reviews` notebook contains an example of LDA applied to six million business review on Yelp. Reviews are more uniform in length than the statements extracted from the earnings call transcripts. After cleaning as before, the 10th and 90th percentiles range from 14 to 90 tokens.

We show results for one model using a vocabulary of 3,800 tokens based on *min_df=0.1%* and *max_df=25%* with a single pass to avoid a lengthy training time for 20 topics. We can use the `pyldavis topic_info` attribute to compute relevance values for *lambda=0.6* that produce the following word list (see the notebook for details):

	Topic 01	Topic 02	Topic 03	Topic 04	Topic 05	Topic 06	Topic 07	Topic 08	Topic 09	Topic 10	Topic 11	Topic 12	Topic 13	Topic 14	Topic 15	Topic 16	Topic 17	Topic 18	Topic 19	Topic 20
0	say	burger	price	bar	order	rice	car	room	coffee	thank	pizza	store	line	kid	office	location	hair	review	tea	de
1	tell	cheese	sushi	beer	table	taco	call	hotel	breakfast	recommend	friendly	buy	wait	year	massage	dog	nail	yelp	drink	la
2	know	fry	pretty	wine	server	chicken	fix	stay	cream	highly	staff	shop	hour	fun	patient	parking	cut	read	coupon	et
3	bad	sandwich	quality	drink	waitress	soup	phone	vegas	chocolate	work	love	find	long	class	doctor	park	salon	write	shake	le
4	go	salad	buffet	menu	wait	spicy	company	pool	ice	professional	amazing	item	early	play	appointment	street	color	birthday	water	pour
5	ask	chicken	restaurant	atmosphere	minute	shrimp	tell	floor	cake	job	super	sell	minute	game	care	downtown	wash	star	smoothie	un
6	customer	sauce	decent	night	ask	noodle	charge	casino	egg	experience	definitely	product	late	old	staff	area	job	mom	boba	con
7	want	bread	eat	patio	take	roll	credit	strip	brunch	wedding	awesome	purchase	open	son	question	south	stylist	sister	milk	pas
8	not	meat	average	dinner	seat	thai	day	club	donut	guy	favorite	sale	ticket	child	feel	drive	haircut	mother	ayce	que
9	rude	potato	high	cocktail	waiter	dish	repair	bathroom	cookie	amazing	nice	tire	movie	school	pain	north	paint	daughter	green	du

Gensim provides a `LdaMultiCore` implementation that allows for parallel training using Python's multiprocessing module and improves performance by 50% when using four workers. More workers do not further reduce training time though, due to I/O bottlenecks.

Summary

In this chapter, we explored the use of topic modeling to gain insights into the content of a large collection of documents. We covered Latent Semantic Analysis, which uses dimensionality reduction of the DTM to project documents into a latent topic space. While effective in addressing the curse of dimensionality caused by high-dimensional word vectors, it does not capture much semantic information. Probabilistic models make explicit assumptions about the interplay of documents, topics, and words that allow algorithms to reverse engineer the document generation process and evaluate the model fit on new documents. We saw that LDA is capable of extracting plausible topics that allow us to gain a high-level understanding of large amounts of text in an automated way, while also identifying relevant documents in a targeted way.

In the next chapter, we will learn how to train neural networks that embed individual words in a high-dimensional vector space that captures important semantic information and allows us to use the resulting word vectors as high-quality text features.