15
Topic Modeling – Summarizing Financial News

In the last chapter, we used the **bag-of-words (BOW)** model to convert unstructured text data into a numerical format. 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)**—or transposed as the term-document matrix—is useful for comparing documents to each other or a query vector for similarity based on their token content and, therefore, finding the proverbial needle in a haystack. It provides informative features to classify documents, such as in our sentiment analysis examples.

However, this document model produces both high-dimensional data and very sparse data, yet 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** to extract hidden themes from documents using **topic modeling**. These themes can produce detailed insights into a large body of documents in an automated way. They are very useful in order to understand the haystack itself and allow us to tag documents based on their affinity with the various topics.

Topic models generate sophisticated, interpretable text features that can be a first step toward extracting trading signals from large collections of documents. They speed up the review of documents, help identify and cluster similar documents, and support predictive modeling.

Applications include the unsupervised discovery of potentially insightful themes in company disclosures or earnings call transcripts, customer reviews, or contracts. Furthermore, the document-topic associations facilitate the labeling by assigning, for example, sentiment metrics or, more directly, subsequent relevant asset returns.

More specifically, after reading this chapter, you'll understand:

- How topic modeling has evolved, what it achieves, and why it matters
- Reducing the dimensionality of the DTM using **latent semantic indexing (LSI)**
- Extracting topics with **probabilistic latent semantic analysis (pLSA)**
- How **latent Dirichlet allocation (LDA)** improves pLSA to become the most popular topic model
- Visualizing and evaluating topic modeling results
- Running LDA using sklearn and Gensim
- How to apply topic modeling to collections of earnings calls and financial news articles

You can find the code samples for this chapter and links to additional resources in the corresponding directory of the GitHub repository. The notebooks include color versions of the images.

Learning latent topics – Goals and approaches

Topic modeling discovers hidden themes that capture semantic information beyond individual words in a body of documents. It aims to address a key challenge for a machine learning algorithm that learns from text data by transcending the lexical level of "what actually 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 practical terms, topic models automatically **summarize large collections of documents** to facilitate organization and management as well as search and recommendations. At the same time, it enables the understanding of documents to the extent that humans can interpret the descriptions of topics.

Topic models also mitigate the **curse of dimensionality** that often plagues the BOW model; representing documents with high-dimensional, sparse vectors can make similarity measures noisy, lead to inaccurate distance measurements, and result in the overfitting of text classification models.

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

These shortcomings of the BOW model prompt the question: how can we learn meaningful topics from data that facilitate a more productive interaction with documentary data?

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 DTM. This approach is similar to the algorithm that we discussed as principal component analysis in *Chapter 13*, *Data-Driven Risk Factors and Asset Allocation with Unsupervised Learning*. While effective, it is difficult to evaluate the results of these models without a benchmark model. In response, probabilistic models have emerged that assume an explicit document generation process and provide algorithms to reverse engineer this process and recover the underlying topics.

The following table highlights key milestones in the model evolution, which we will address in more detail in the following sections:

Model	Year	Description
Latent semantic indexing (LSI)	1988	Captures the semantic document-term relationship by reducing the dimensionality of the word space
Probabilistic latent semantic analysis (pLSA)	1999	Reverse engineers a generative process that assumes words generate a topic and documents as 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 (LSA)**—set out to improve the results of queries that omitted relevant documents containing synonyms of query terms (Dumais et al. 1988). Its goal was to model the relationships between documents and terms so that it could predict that a term should be associated with a document, even though, because of the 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 the **singular value decomposition (SVD)** to find the best lower-rank DTM approximation using k singular values and vectors. In other words, LSI builds on some of the dimensionality reduction techniques we encountered in *Chapter 13*, *Data-Driven Risk Factors and Asset Allocation with Unsupervised Learning*. The authors also experimented with hierarchical clustering but found it too restrictive for this purpose.

In this context, SVD identifies a set of uncorrelated indexing variables or factors that represent each term and document by its vector of factor values. *Figure 15.1* illustrates how SVD decomposes the DTM into three matrices: two matrices that contain orthogonal singular vectors and a diagonal matrix with singular values that serve as scaling factors.

Assuming some correlation in the input DTM, singular values decay in value. Therefore, selecting the T-largest singular values yields a lower-dimensional approximation of the original DTM that loses relatively little information. In the compressed version, the rows or columns that had N items only have $T < N$ entries.

The LSI decomposition of the DTM can be interpreted as shown in *Figure 15.1*:

- The first $\mathrm{M} \times \mathrm{T}$ matrix represents the relationships between documents and topics.
- The diagonal matrix scales the topics by their corpus strength.

- The third matrix models the term-topic relationship.

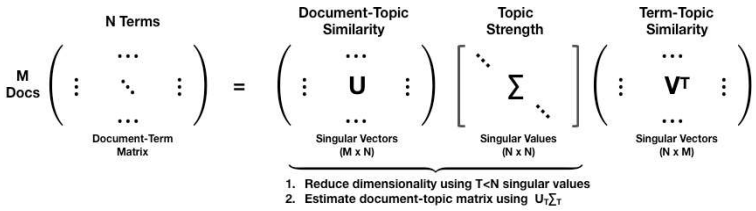

Figure 15.1: LSI and the SVD

The rows of the matrix produced by multiplying the first two matrices $U_T \Sigma_T$ correspond to the locations of the original documents projected into the latent topic space.

How to implement LSI using sklearn

We will illustrate LSI using the BBC articles data that we introduced in the last chapter because they are small enough for quick training and allow us to compare topic assignments with category labels. Refer to the notebook `latent_semantic_indexing` for additional implementation details.

We begin by loading the documents and creating a train and (stratified) test set with 50 articles. Then, we vectorize the data using `TfidfVectorizer` to obtain weighted DTM counts and filter out words that appear in less than 1 percent or more than 25 percent 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)
```

We use scikit-learn's `TruncatedSVD` class, which only computes the *k*-largest singular values, to reduce the dimensionality of the DTM. The deterministic `arpack` algorithm delivers an exact solution, but the default "randomized" implementation is more efficient for large matrices.

We compute five topics to match the five categories, which explain only 5.4 percent of the total DTM variance, so a larger number of topics would be reasonable:

```
svd = TruncatedSVD(n_components=5, n_iter=5, random_state=42)
svd.fit(train_dtm)
svd.explained_variance_ratio_.sum()
0.05382357286057269
```

LSI identifies a new orthogonal basis for the DTM that reduces the rank to the number of desired topics. The `.transform()` method of the trained `svd` object projects the documents into the new topic space. This space

results from reducing the dimensionality of the document vectors and corresponds to the $U_T\Sigma_T$ transformation illustrated earlier in this section:

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

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
```

The topic assignments for this sample align with the average topic weights for each category illustrated in *Figure 15.2* ("Politics" is the right-most bar). 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).

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.

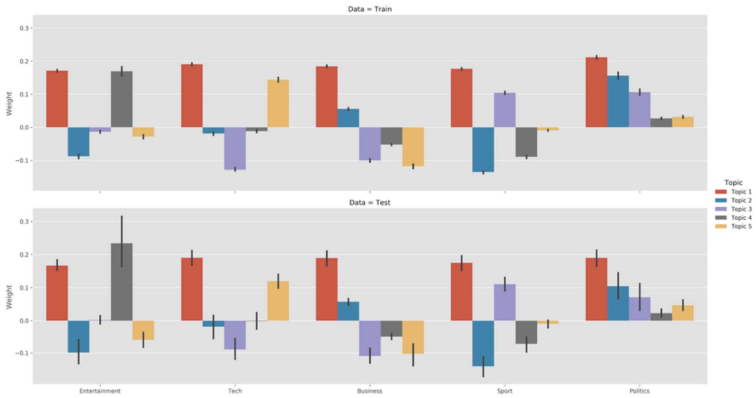

Figure 15.2: LSI topic weights for train and test data

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 clearly differentiated (refer to *Figure 15.3*).

Figure 15.3: Top 10 words per LSI topic

Strengths and limitations

The strengths of LSI include the removal of noise and the mitigation of the curse of dimensionality. It also captures some semantic aspects, like synonymy, and clusters both documents and terms via their topic associations. Furthermore, it does not require knowledge of the document language, and both information retrieval queries and document comparisons are easy to do.

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

Probabilistic latent semantic analysis

Probabilistic latent semantic analysis (**pLSA**) takes a **statistical perspective** on LSI/LSA and creates a generative model to address the lack of theoretical underpinnings of LSA (Hofmann 2001).

pLSA explicitly models the probability word w appearing in document d, as described by the DTM as a mixture of conditionally independent multinomial distributions that involve topics t.

There are both **symmetric and asymmetric formulations** of how word-document co-occurrences come about. The former assumes that both words and documents are generated by the latent topic class. In contrast, the asymmetric model assumes that topics are selected given the document, and words result in a second step given the topic.

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

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

The **plate notation** in *Figure 15.4* describes the statistical dependencies in a probabilistic model. More specifically, it encodes the relationship just described for the asymmetric model. Each rectangle represents multiple

items: the outer block stands for *M* documents, while the inner shaded rectangle symbolizes *N* words for each document. We only observe the documents and their content; the model infers the hidden or latent topic distribution:

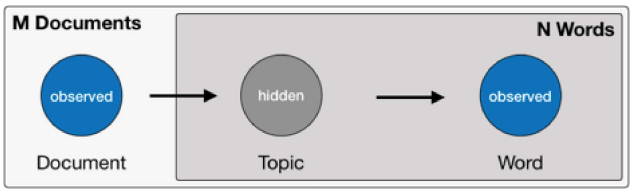

Figure 15.4: The statistical dependencies modeled by pLSA in plate notation

Let's now take a look at how we can implement this model in practice.

How to implement pLSA using sklearn

pLSA is equivalent to **non-negative matrix factorization** (**NMF**) using a Kullback-Leibler divergence objective (view the references on GitHub). Therefore, we can use the `sklearn.decomposition.NMF` class to implement this model following the LSI example.

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

```
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 that is a substitute for the explained variance measure from earlier:

```
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, as shown in *Figure 15.5*:

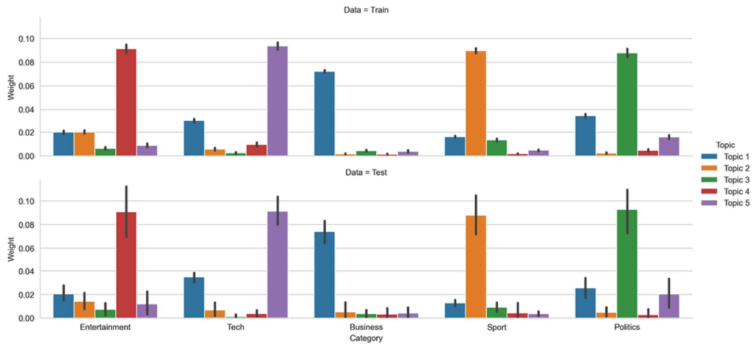

Figure 15.5: pLSA weights by topic for train and test data

We also note 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," "star," and so forth, as you can see in *Figure 15.6*:

Figure 15.6: Top words per topic for pLSA

Strengths and limitations

The benefit of using a probability model is that we can now compare the performance of different models by evaluating the probability they assign to new documents given the parameters learned during training. It also means that the results have a clear probabilistic interpretation. In addition, pLSA captures more semantic information, including polysemy.

On the other hand, pLSA increases the computational complexity compared to LSI, and the algorithm may only yield a local as opposed to a global maximum. Finally, it does not yield a generative model for new documents because it takes them as given.

Latent Dirichlet allocation

Latent Dirichlet allocation (LDA) extends pLSA by adding a generative process for topics (Blei, Ng, and Jordan 2003). 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, like 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 reflect only a small set of topics, and topics use only a limited number of terms frequently.

The Dirichlet distribution

The Dirichlet distribution produces probability vectors that can be used as a discrete probability distribution. That is, it randomly generates a given number of values that are positive and sum to one. It has a parameter α 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 of the probability mass. *Figure 15.7* illustrates three draws of size 10 for $\alpha = 0.1$:

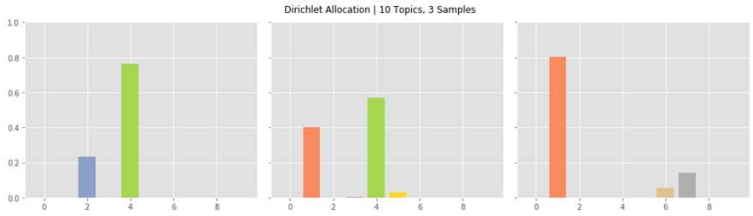

Figure 15.7: Three draws from the Dirichlet distribution

The notebook `dirichlet_distribution` contains a simulation that lets you experiment with different parameter values.

The generative model

The LDA topic model assumes the following generative process when an author adds an article to a body of documents:

1. Randomly mix a small subset of topics with proportions defined by the Dirichlet probabilities.
2. For each word in the text, 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 weight of each topic and the terms that make up each topic. The Dirichlet distribution governs the selection of topics for documents and words for topics. It 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 in *Figure 15.8* summarizes these relationships and highlights the key model parameters:

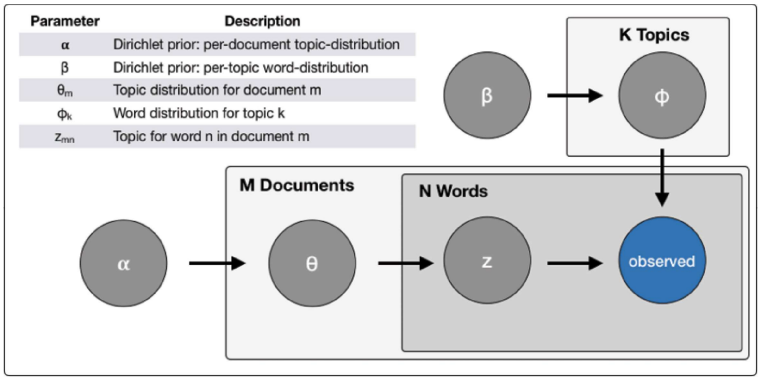

Figure 15.8: The statistical dependencies of the LDA model in plate notation

Reverse engineering the process

The generative process is clearly 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 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 by Blei et al. (2003) uses **variational Bayes (VB)** to approximate the posterior distribution. Alternatives include Gibbs sampling and expectation propagation. We will illustrate, shortly, the implementations by the sklearn and Gensim libraries.

How to evaluate LDA topics

Unsupervised topic models do not guarantee that the result will be meaningful or interpretable, and there is no objective metric to assess the quality of the result as in supervised learning. Human topic evaluation is considered the gold standard, but it 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 of unseen text documents. It is based on the entropy $H(p)$ of this distribution p and is computed with respect to the set of tokens w:

$$2^{h(p)} = 2^{-\Sigma_w p(w) \log_2 p(w)}$$

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 defines a topic together.

There are two measures of coherence that have been designed for LDA and are shown to align with human judgments of topic quality, namely the UMass and the UCI metrics.

The UCI metric (Stevens et al. 2012) 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 like Wikipedia so that this metric can be thought of as an external comparison to semantic ground truth.

In contrast, the UMass metric (Mimno et al. 2011) 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 comparing the model result to extrinsic ground truth, this measure reflects intrinsic coherence. Both measures have been evaluated to align well with human judgment (Röder, Both, and Hinneburg 2015). In both cases, values closer to zero imply that a topic is more coherent.

How to implement LDA using sklearn

We will use the BBC data as before and train an LDA model using sklearn's `decomposition.LatentDirichletAllocation` class with five topics (refer to the sklearn documentation for details on the parameters and the notebook `lda_with_sklearn` for implementation details):

```
lda_opt = 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_opt.pkl')
lda_opt = joblib.load(model_path / 'lda_opt.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` (Sievert and Shirley 2014). We will introduce the key concepts; each LDA application notebook contains examples.

pyLDAvis displays the global relationships among topics while also facilitating their semantic evaluation by inspecting the terms most closely associated with each individual 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 distribution over words that define a topic.

To this end, LDAVis introduces the **relevance** r of term w to topic t. The relevance produces a flexible ranking of terms by topic, by computing a weighted average of two metrics:

- The degree of association of topic t with term w, expressed as the conditional probability $p(w \mid t)$
- The saliency, or lift, which measures how the frequency of term w for the topic t, $p(w \mid t)$, compares to its overall frequency across all documents, $p(w)$

More specifically, we can compute the relevance r for a term w and a topic t given a user-defined weight $0 \leq \lambda \leq$, like the following:

$$r(w, t \mid \lambda) = \lambda \log\big(p(w|t)\big) + (1-\lambda) \log \frac{p(w|t)}{p(w)}$$

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

How to implement LDA using Gensim

Gensim is a specialized **natural language processing** (**NLP**) library with a fast LDA implementation and many additional features. We will also use it in the next chapter on word vectors (refer to the notebook `lda_with_gensim` for details and the installation directory for related instructions).

We convert the DTM produced by sklearn's `CountVectorizer` or `TfIdfVectorizer` 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's LDA algorithm includes numerous settings:

```
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 `LdaModel`, as follows:

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

Gensim evaluates topic coherence, as introduced in the previous section, 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())
```

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	T
0.55%	on-line	0.90%	best	1.04%	mo-bile	0.64%	market	0.94%	l
0.51%	site	0.87%	game	0.98%	phone	0.53%	growth	0.72%	h
0.46%	game	0.62%	play	0.51%	music	0.52%	sales	0.72%	h
0.45%	net	0.61%	won	0.48%	film	0.49%	econ-omy	0.65%	e t
0.44%	used	0.56%	win	0.48%	use	0.45%	prices	0.57%	u

The left panel of *Figure 15.9* displays the topic coherence scores, which highlight the decay of topic quality (at least, in part, due to the relatively small dataset):

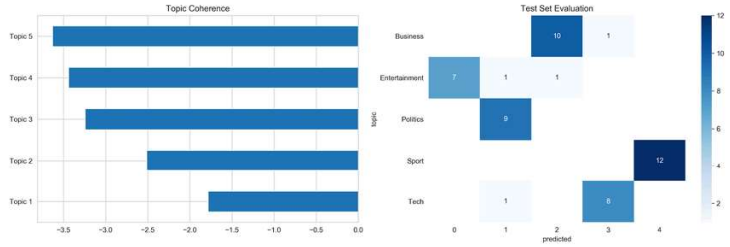

Figure 15.9: Topic coherence and test set assignments

The right panel displays the evaluation of our test set of 50 articles with our trained model. The model makes four mistakes for an accuracy of 92 percent.

Modeling topics discussed in earnings calls

In *Chapter 3*, *Alternative Data for Finance – Categories and Use Cases*, 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 700 earnings call transcripts between 2018 and 2019. This is a fairly small dataset; for a practical application, we would need a larger dataset.

The directory `earnings_calls` contains several files with the code examples used in this section. Refer to the notebook `lda_earnings_calls` 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 next.

Data preprocessing

The transcripts consist of individual statements by company representatives, an operator, and a Q&A session with analysts. We will treat each of these statements as separate documents, ignoring operator statements, to obtain 32,047 items with mean and median word counts of 137 and 62, 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),
len(documents)
32047
```

We use spaCy to preprocess these documents, as illustrated in *Chapter 13*, *Data-Driven Risk Factors and Asset Allocation with Unsupervised Learning*, (refer to the notebook), and store the cleaned and lemmatized text as a new text file.

Exploration of the most common tokens, as shown in *Figure 15.10*, reveals domain-specific stopwords like "year" and "quarter" that we remove in a second step, where we also filter out statements with fewer than 10 words so that some 22,582 remain.

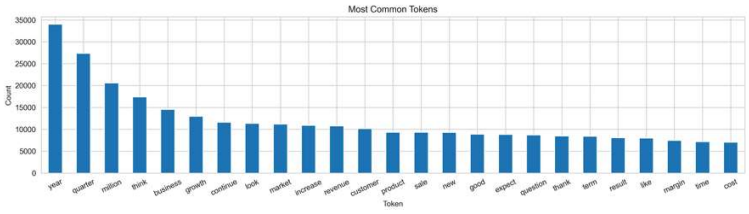

Figure 15.10: Most common earnings call tokens

Model training and evaluation

For illustration, we create a DTM containing terms appearing in between 0.5 and 25 percent of documents that results in 1,529 features. Now we proceed to train a 15-topic model using 25 passes over the corpus. This takes a bit over two minutes on a 4-core i7.

The top 10 words per topic, as shown in *Figure 15.11*, identify several distinct themes that range from obvious financial information to clinical trials (Topic 5), China and tariff issues (Topic 9), and technology issues (Topic 11).

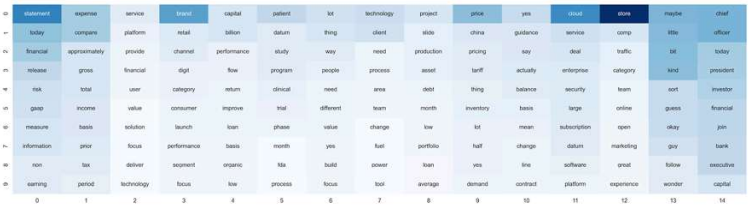

Figure 15.11: Most important words for earnings call topics

Using pyLDAvis' relevance metric with a 0.6 weighting of unconditional frequency relative to lift, topic definitions become more intuitive, as illustrated in *Figure 15.12* for Topic 7 about China and the trade wars:

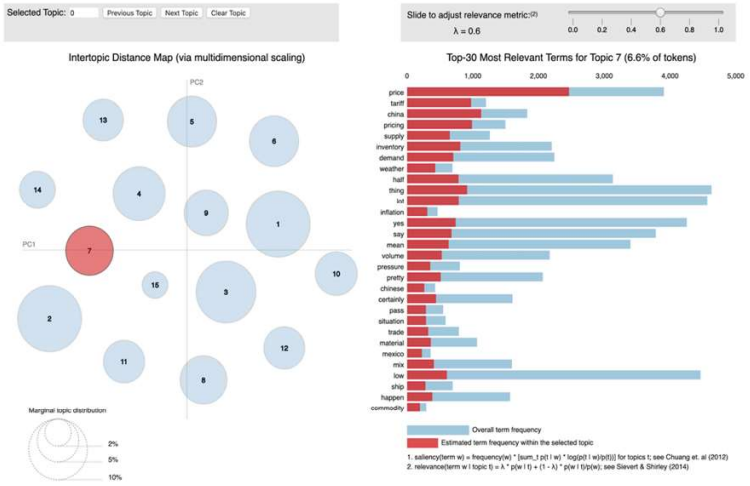

Figure 15.12: pyLDAVis' interactive topic explorer

The notebook also illustrates how you can 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 run 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 percent of documents, respectively, using alternatively binary and absolute counts. We then train LDA models with 3 to 50 topics, using 1 and 25 passes over the corpus.

The chart in *Figure 15.13* 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.

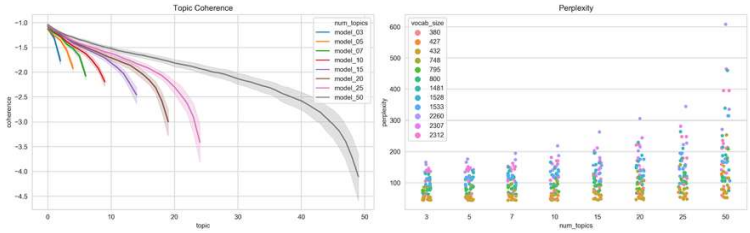

Figure 15.13: Impact of LDA hyperparameter settings on topic quality

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 with financial news

The notebook `lda_financial_news` contains an example of LDA applied to a subset of over 306,000 financial news articles from the first five months of 2018. The datasets have been posted on Kaggle, and the articles have been sourced from CNBC, Reuters, the Wall Street Journal, and more. The notebook contains download instructions.

We select the most relevant 120,000 articles based on their section titles with a total of 54 million tokens for an average word count of 429 words per article. To prepare the data for the LDA model, we rely on spaCy to remove numbers and punctuation and lemmatize the results.

Figure 15.14 highlights the remaining most frequent tokens and the article length distribution with a median length of 231 tokens; the 90th percentile is 642 words.

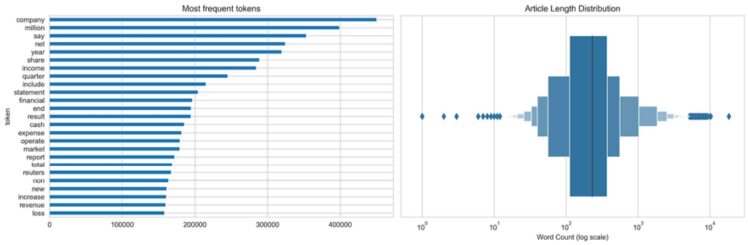

Figure 15.14: Corpus statistics for financial news data

In *Figure 15.15*, we show results for one model using a vocabulary of 3,570 tokens based on `min_df` =0.005 and `max_df` =0.1, with a single pass to avoid the length training time for 15 topics. We can use the `top_topics` attribute of the trained `LdaModel` to obtain the most likely words for each topic (refer to the notebook for more details).

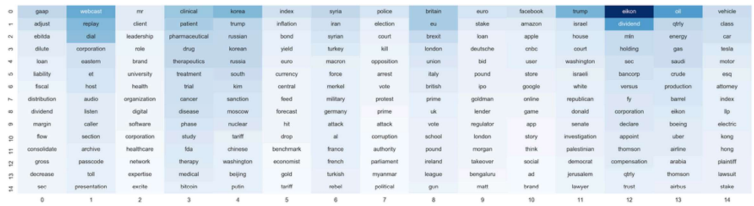

Figure 15.15: Top 15 words for financial news topics

The topics outline several issues relevant to the time period, including Brexit (Topic 8), North Korea (Topic 4), and Tesla (Topic 14).

Gensim provides a `LdaMultiCore` implementation that allows for parallel training using Python's multiprocessing module and improves performance by 50 percent 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 indexing that 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 learned 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.