Tech Review

*A Comparison of Topic Models and Word Embeddings in **Gensim***

Introduction

The tagline for the tool Gensim is "Topic modeling for humans", and it's exactly this sentiment and bias for ease-of-use that has made the Python library so hugely popular with users. Of course, the expressive power paired with that simplicity and ease-of-use is the other key factor, and I was inspired to write this review, in part, to examine that functionality for myself and make my own judgements. It turns out that Gensim takes a very broad view of "Topic Modeling" as a concept, and that includes what most researchers and practitioners would consider sophisticated **Word Embedding**, which is useful for shallow Natural Language Processing (shallow NLP) tasks, as well as simpler **Vector Space Models**, which obviously have applications to Information Retrieval (IR) outside of reference to any topics. In fact, the functionality offered by Gensim exists somewhere on a spectrum between IR and shallow NLP, drawing from both. Since the topics in the CS 410 Text Information Systems course sit in the same niche space, this analysis will likely benefit any current or future students when they inevitably encounter Gensim in industry, as I did.

While it's easy to motivate *contrasting* these disparate techniques given that CS 410 presents them as constituting different stages in an overall Text Information Pipeline, with IR techniques relevant to large-scale search, and shallow NLP applicable to smaller-scale text mining, only after search has narrowed down the scope, motivation for *comparing* the techniques stems from the Gensim documentation itself[1]:

> *Gensim is designed to process raw, unstructured digital texts (*"plain text"*) using unsupervised machine learning algorithms.*
>
> *The algorithms in Gensim, such as `Word2Vec`, `FastText`, Latent Semantic Indexing (LSI, LSA, `LsiModel`), Latent Dirichlet Allocation (LDA, `LdaModel`) etc, automatically discover the semantic structure of documents by examining statistical co-occurrence patterns within a corpus of training documents. These algorithms are **unsupervised**, which means no human input is necessary – you only need a corpus of plain text documents.*
>
> *Once these statistical patterns are found, any plain text documents (sentence, phrase, word…) can be succinctly expressed in the new, semantic representation and queried for topical similarity against other documents (words, phrases…).*

Based on this domain-agnostic view, we can explore the coverage that Gensim offers for the algorithms defined in the CS 410 course, as well as directions where it is substantially more advanced, that may be an interesting area for students to explore.

[1] https://radimrehurek.com/gensim/intro.html#what-is-gensim

Text Models

To disambiguate the overloaded usage of Topic Models in Gensim documentation, which conflicts with the narrowly defined scope learned in class, we'll use a more generic term for different strategies for modeling the contents of "plain text": Text Model. This way, Topic Model can be used exclusively when referring to extracting actual topics (as an unsupervised learning exercise) or categorizing text into topics (as a supervised learning exercise), but not in search where it is not explicitly needed. Some of the documentation available for Topic Modeling also use the term "concept" to mean "topic", so those are considered synonyms, applicable to the narrower scope.

Vector Space Models

All so-called Vector Space Models have in common the classical Bag-of-Words model representation of text, where the relative ordering of the words is ignored, and only their frequency is counted. Because all Text Models covered rely on modeling user queries as very small documents, no differentiation is made between the model of a document versus that of a query.

TF-IDF Model

Starting at the simplest and earliest, Gensim offers an implementation of the TF-IDF weighted vector model covered in class. This model constructs a document or query vector based on the simple concept of Term Frequency (TF) of the word (term) in the document being modeled, scaled by the Inverse Document Frequency (IDF) of the term in the larger collection of documents (corpus). This gives high weighting to terms to the extent that they appear frequently in the document *disproportionally to how frequently they appear in the larger corpus.*

Unfortunately, we know from our coursework that this simple model has several drawbacks when it comes to IR applications, including incorrectly favoring longer documents, and weighting multiple matches of the same term equivalent to multiple matches of different terms, effecting performance in ranking functions. Therefore, state-of-the-art ranking functions like Okapi BM25 leverage both normalization of IDF by document length, and a diminishing return function on increasing TF. The motivation here is to correct the long document and repeated match bias by applying non-linear functions to both the TF and IDF values, to heuristically tune the system away from bias. In this vein, Gensim offers the ability to supply custom functions to apply to the TF, and to the IDF term and the global Corpus object, called `wlocals` and `wglobal`, respectively, that could be used to model any state-of-the-art ranking function exactly. Moreover, the implementation can be used to directly emulate the Pivoted Length Normalization algorithm exactly.

Topic Models

While the TF-IDF model cannot be applied to Topic Modelling, directly, the basic concepts form the basis of the discussion for more complex implementations (namely, LSA and LDA), so is included for completeness. Moreover, the insight that the "more principled" (read: generative)

Query Likelihood model results in an implementation that is computationally very similar to other “merely heuristic” (read: discriminative) Vector Space Models will be leveraged in our discussion of Latent Semantic Analysis (LSA), to explain the connection to Probabilistic Latent Semantic Analysis (PLSA) learned in class. In what follows, the term **Bag-of-Words vector** will be used to refer to any normalized term frequency representation built from a Bag-of-Words representation of text, of which TF-IDF is an important example, irrespective of the actual math used, which can vary between implementations, depending on whether what is being modeled is a heuristic (discriminative models) or a conditional probability (generative models). This is to stay consistent with the Gensim documentation, which continues to use Bag-of-Words (or BOW), even after a vocabulary dictionary has been augmented to include common N-gram terms; because Bag-of-Terms and Bag-of-N-grams have neither caught on, please keep the general definition in mind.

LSA Model

Latent Semantic Analysis (LSA) is the process of discovering a latent subspace of *concepts* in the Bag-of-Words model vector space, by means of Singular Value Decomposition (SVD) on the so-called Term-Document Matrix, which is simply a collection of all the Bag-of-Words vectors as column vectors. The resulting factorization uses the TSD^T syntax to represent the Term-concept vector matrix (T), the Singular values diagonal matrix (S), and the transpose of the concept-Document vector matrix (D^T), rather than the more familiar USV^T (or $U\Sigma V^T$) used in most treatments of SVD. However, the math is the same, and the result is the ability to map documents (including queries) to the latent concept space, which is a compact representation that has good properties for both query-document similarity comparisons (in search) and for leveraging latent concept space in Topic Modeling (in Text Mining) by way of a one-to-one correspondence between *concept* and *topic*.

In the Gemsim implementation, which goes by the synonym Latent Semantic Indexing (LSI), the USV^T syntax is used for the left singular vectors, singular values, and right singular vectors, respectively, so it helps to be familiar with the original syntax. However, the treatment in the LSI-specific documentation is helpful for grasping what the algorithm is doing, conceptually, so I’ve reused that nomenclature, since it was not explicitly studied in class. To draw a connection to something that *was* studied in class, namely Probabilistic Latent Semantic Analysis, LSA/LSI is the discriminative analog of the generative PLSA algorithm, so has a similar function, but a totally different theoretical justification. In LSA, there is no hidden or latent variable modeled, so a direct SVD implementation results in the topic-term mapping (T or U, depending on naming conventions), individual topic values (S), and the document-topic mapping (D^T or V^T), rather than requiring an iterative implementation like Expectation Maximization (EM). PLSA adds the theoretical grounding for interpreting the mappings as conditional probabilities, and the singular values as probabilities, a correspondence summarized as follows[2]:

[2] https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05

$$P(D,W) = \sum_{Z} \underline{P(Z)}\,\underline{P(D|Z)}\,\underline{P(W|Z)}$$

$$A \approx \underline{U_t}\,\underline{S_t}\,\underline{V_t^T}$$

LDA Model

Latent Dirichlet Allocation (LDA) is a Bayesian extension to PLSA which adds two Dirichlet Prior distributions to model the prior belief about the probabilities the model will be learning, most often used to model a background language model, in order to make to make the topic modeling more discriminative, and to provide implicit smoothing, so the algorithm will generalize well to new documents with new vocabulary (which would otherwise model the probability of unseen terms as zero). The mechanics of the formulation get a bit into probability minutia, but suffice it to say that the model generalizes well thanks to fancy math, and the only difference the implementer needs to understand is the concept of pseudocounts and a switch from the Maximum Likelihood Estimator (MLE) to the Maximum Aposteriori Estimator (MAP) in the EM implementation.

The concept of Dirichlet Prior sampling is technically thorny enough that the Gensim tutorial on LDA saw fit to gloss over the details, and the implementation provides a special `‘auto’` mode for inferring the priors for document-topic distributions and topic-word distributions, `alpha` and `eta`, respectively (some treatments call the second variable β), rather than supplying the priors (or an appropriate sampling method) manually. Nevertheless, the full expressive power of the Dirichlet Prior is available for advanced use cases but falls just short of allowing the equivalent of a PLSA. Specifically, it’s possible to get an implementation of PLSA by specifying uniform Dirichlet priors in LDA[3], but there appears to be no direct way to specify exactly this configuration using Gensim[4].

Word Embeddings

In contrast to Vector Space Models, Word Embeddings do not model the cooccurrence of words, directly, and then analyze that cooccurrence to glean insights. Rather, they are constructed as the byproduct of using a Neural Network[5] to learn one of two flavors of prediction tasks:

1. Given a sliding window of N terms in the corpus, predict the 1 term that’s been omitted, given the other N – 1 (called the Continuous Bag-of-Words, or CBOW, architecture)
2. Given a sliding window of N terms in the corpus, predict the N – 1 missing that have been omitted, given the middle one that’s been left (called the Skip-Gram architecture)

[3] https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation#cite_note-6

[4] https://groups.google.com/g/gensim/c/Dm5uNsyfkno

[5] Well, it’s not *technically* a Neural Network, but works very much like one

For the sake of generality, a small sleight-of-hand was involved in switching from “words” in Word Embeddings to “terms” in the description of the prediction problems. This is because the same algorithm can be used to learn embeddings for either super-word terms (multiword n-grams) or sub-word terms (character n-grams), augmenting the predictive power of the technique. Still, because terms-as-single-words was the first iteration of the growing body of research, we’re left with the phrase Word Embedding for historical reasons, and everyone understands that to mean, more generally, a synonym for Term Embedding.

Word2Vec

As alluded to, the seminal published work[6] in this area assumed that the unit of meaning to embed was the word, and this was a good assumption for a language like English, providing rather striking theoretical results. Specifically, not only did the basic requirement that similar words measure closely get met very well, but an additional unexpected property of the learned embeddings meant that *multiple dimensions of similarity* were learned simultaneously. To illustrate this, the authors present a classic analogy problem from Natural Language Understanding:

Man: King, Woman: _____

In prose, this reads “Man is to King, as Woman is to _____”, where the correct answer for the fluent reader is obviously “Queen”. To model this type of analogy in a vector embedding, it would be insufficient for the vectors for “Man” and “King” to measure the same 1-D distance as the words “Woman” and “Queen”; instead, the N-D (where N is the embedding dimension) must be identical for that relationship to have been learned. The authors assessed that learning with the following algebraic formulation:

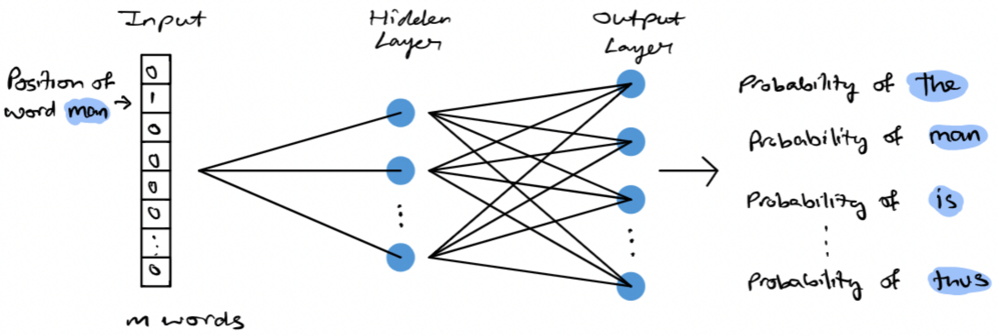
```
vector(“Queen”) =~ vector(”King”) - vector(”Man”) +
vector(”Woman”)
```

Here, the approximate equality was assessed not by a numerical comparison, but by verifying that the resulting computed vector was closer to the target words vector than any *other* vector in the learned embedding corpus, meaning that the system could effectively achieve high accuracy on analogy problems by computing a synthetic vector, then answering with the closest approximation in the corpus.

While the details of the Neural-Network-like architecture used to achieve this great performance are well out-of-scope for this treatment (as is the fact that removing a non-linear activation function makes their implementation *not technically a Neural Network*), the general summary is that the basic architecture consists of only two layers: one hidden, and one output (i.e.: it is not “deep”, having only one hidden layer). This makes it very simple by modern standards, and further lack of non-linear activation functions also make it extremely computationally efficient to train. The output of the process is the hidden layer weights for the

[6] https://arxiv.org/pdf/1301.3781.pdf

model, where the (somewhat contrived) prediction tasks are merely a computationally efficient mechanism for arriving at those weights. The following diagram has a good graphical representation of the Skip-Gram architecture, for clarity[7]:



The Gensim implementation of Word2vec embeddings, models.word2vec, provides both the Hierarchical Softmax activation function from the original paper, as well as Negative Sampling from a follow-on paper from the same research team[8]. Again, the basis for these strategies in Information Theory cannot be covered in any depth, except to say that they are both critical optimizations improving the computational tractability of a large language model with an immense vocabulary[9], which is the space in which Skip-Gram really shines. Additionally, specific usage examples in Gensim linked documentation cover the extension of Word2Vec from words to multiword n-grams, a generalization discussed above.

fastText

FastText, also stylized fastText, is an extension of the Word2Vec strategy originally developed by researchers at Facebook AI Research (FAIR)[10]. The primary insight of this extension was that the morphological structure of words could be inferred efficiently by training on sub-word parts; for computational simplicity, their implementation uses character n-grams. This resulted in improved word embedding for so-called Out-Of-Vocabulary (OOV) words, which would normally be an insurmountable problem for Word2Vec, working off a fixed (if gigantic) vocabulary from the training corpus. To approximate OOV words, a word vector is built by summing up vectors of component character n-grams, and the results are good approximately, subject to the constraint that at least one constituent part of the word was learned by the model.

According to the fastText tutorial in Gensim, the attention to sub-word components makes fastText superior to Word2Vec on syntactic tasks, like recognizing the similarity between a correctly-spelled word and a common misspelling (e.g.: *accommodation* vs. *accomodation*) or

[7] https://towardsdatascience.com/word2vec-explained-49c52b4ccb71
[8] https://arxiv.org/abs/1310.4546
[9] https://towardsdatascience.com/hierarchical-softmax-and-negative-sampling-short-notes-worth-telling-2672010dbe08
[10] https://techcrunch.com/2016/08/18/facebooks-artificial-intelligence-research-lab-releases-open-source-fasttext-on-github/

recognizing the similarity of singular and plural forms of the same word (e.g.: *night* vs. *nights*) which work, again, even if one of the words is OOV. This was especially important to improving this category of tasks on so-called *morphologically rich* languages like German and Turkish, where the linked tutorial gives an excellent explanation of the phenomenon:

> *[Sub-word morphological structure] is especially significant for morphologically rich languages (German, Turkish) in which a single word can have a large number of morphological forms, each of which might occur rarely, thus making it hard to train good word embeddings.*

At the same time, given a large enough corpus, fastText can perform comparably to Word2Vec on semantic problems, for example, the analogy problem[11]:

```
Query triplet (A - B + C)? psx sony nintendo
gamecube 0.803352
nintendogs 0.792646
playstation 0.77344
sega 0.772165
gameboy 0.767959
arcade 0.754774
playstationjapan 0.753473
gba 0.752909
dreamcast 0.74907
famicom 0.745298
```

Summary

As shown, Gensim treats disparate strategies for modeling “plain text” documents uniformly by repurposing the Topic Model concept and generalizing it to cover many flavors of Text Models useful across both Information Retrieval and Text Mining. This generalization is conceptually useful but suffers a Public Relations problem by virtue of the name colliding with a well-known concept from Text Mining. Instead, I propose taking the “Semantic” from LSA and “Embedding” from Word Embeddings and referring to the general concept by a new umbrella term, **Semantic Embedding**, which I think evokes the same understanding that the Gensim documentation is meant to convey. *Semantic*, in that it’s meant to model the underlying *meaning* of text in a way that approximates how human’s use it, with all the nuance of synonym and polysemy that entails. *Embedding*, because it’s a generalization over latent subspaces (in Vector Space Models) and learned network weights (in Word Embedding) to communicate that some amount of dense and useful *information* has been gleaned from a large volume of otherwise sparse *data*. Those representations are equally applicable to search in IR and clustering in Text Mining, so long as the distance measure has conceptual meaning to the application.

[11] https://fasttext.cc/docs/en/unsupervised-tutorial.html