Representing text - two issues

- Overwhelmingly, all representations of textual features finally take the form of vectors in $\mathbb{R}^n$. This is often true of categorial or other structural entities that are associated with text or words (e.g. constituent parses, PoS tags, semantic tags, representations of meaning either as graphs or as formulae in a logic, etc.)
- Text contains words. So, the first question is: how do we represent sets or sequences of words as a vector given vector representations for the constituent words? The most important feature of text is the words that constitute the text.
- Texts have many more features than just the constituent words. These relational features arise when multiple words come together as in text. For example: parses, enitity-relation associations, semantic tags, dependencies etc. Even word features like PoS tags and senses arise only in the context of text. Isolated words do not have unique PoS tags or senses.
- So, the second question: How to represent and combine these multiple features of text along with the vectors that represent words.

Combining features

There are two basic methods to combine features of text:

- First: compose the feature vectors using mathematical, algebraic operations like sum, average, concatenate etc., and their combinations.
- Second: Learn embeddings for textual entities like - phrase, sentence, paragraph, document - in a vector space in a manner similar to how word vectors were learnt. The input for such learning is a representation of individual features combined in some way - often concatenated.
- In principle one can combine the two methods.

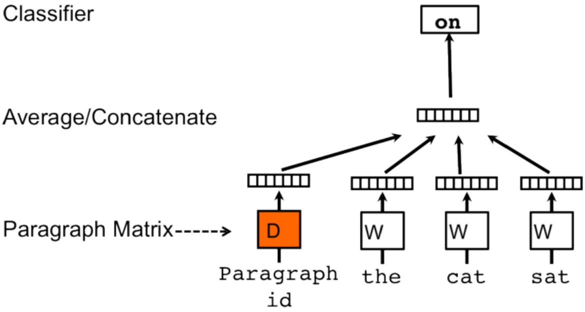
Representing text as words

- Word vectors with fine tuning and linguistic modulation are a widely accepted representation for words in NL tasks.
- Representations for text - that is phrase, sentence, paragraph, document - are not as stable as word representations. Numerous proposals exist. Two main methods:
  - Compose word vectors using mathematical, algebrac operations on the constituent words to get a text representation.
  - Learn text representations in a manner similar to the way word vectors were learnt using neural networks.

Document vectors using algebraic operations

- The simplest representation for a document is an average of all words in the document.
- Another alternative is to calculate a weighted average using tfidf values as weights.

Paragraph vectors - PV-DM[2]



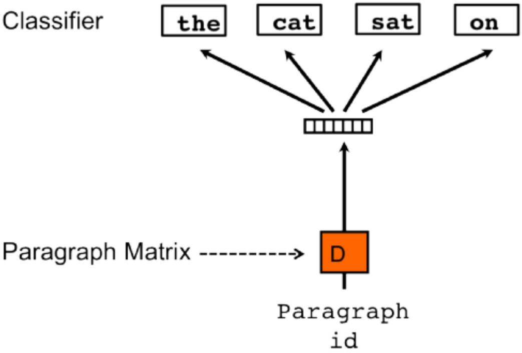
Paragraph vector - distributed memory

For N documents and vocabulary V: $D_{d_D \times N}$ matrix, $W_{d_w \times |V|}$ matrix. d_D, d_w are dimensions of individual document and word vectors respectively (user parameters). Document and word vectors are usually concatenated.

[2] Le, Mikolov, Distributed representations of sentences and documents, ICML, 2014.

- In training predict the subsequent word given the context window words (as in Word2Vec) and the document vector.
- Use backpropagation for training and learn document matrix, word matrix weights and softmax parameters.
- In testing for the test document T, add it as an extra column to D and train the document matrix D weights keeping W and softmax parameters fixed.
- Use the newly trained document vector as a representation for the NL task.

Paragraph vectors - PV-DBoW



Paragraph vector - distributed bag of words

- Input is a one hot document vector. $D_{d_D \times N}$ is the document matrix. Randomly sample a window in the document, then randomly sample a word in the window and predict that word.

- $\mathbf{d_k} = Dd_k$, k is the index of the document in the document corpus, d_k is the one-hot representation of the document, $\mathbf{d_k}$ is the d_D sized document vector for the k^{th} document. $\mathbf{d_k}$ is also the hidden layer output (dummy layer) - no activation function.

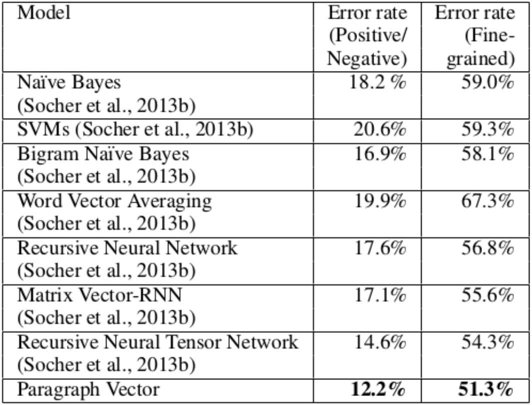
- $\mathbf{z} = U\mathbf{d_k} + \mathbf{b}$, $\mathbf{z}$ is the $|V|$ sized output vector, $U_{|V| \times d_D}$ output weight matrix, $\mathbf{b}$ is the bias. The final output is calculated via a softmax, so j^{th} component of $\hat{\mathbf{z}}$ is: $\hat{\mathbf{z}}_j = \frac{e^{\mathbf{z}_j}}{\sum_{i=1}^{|V|} e^{\mathbf{z}_i}}$

- Let $\mathbf{t}$ be the expected or true output (a one-hot vector of size $|V|$ then the loss is (cross entropy loss):

$$
\begin{aligned}
e(\mathbf{t}, \hat{\mathbf{z}}) &= \sum_{i=1}^{|V|} \mathbf{t}_i \log(\hat{\mathbf{z}}_i) \quad \mathbf{t}_i \text{ is 0 for all except chosen output word in } V \\
&= \log(\hat{\mathbf{z}}_j) \quad \text{assuming } j \text{ is index of chosen output word}
\end{aligned}
$$

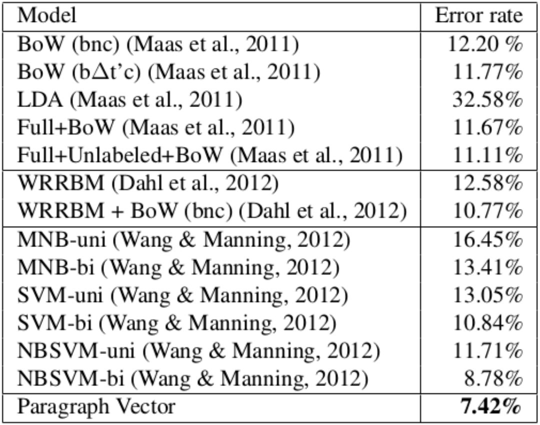
Performance of paragraph vectors

- Data sets:
  - Stanford sentiment data set. 8544 sentences training, 1101 validation, 2210 testing. Sentiment levels in range $[0, 1]$. Labels for phrases and for sentence.
  - IMDB data set. 100000 reviews, 25000 training, 25000 testing, 50000 unlabelled. Only binary labels positive/negative.
- Document vector was average of PV-DM, PV-DBoW. Paragraph and word vector dimension 400. Optimal window size 8 (for PV-DM). Binary prediction for phrases and sentence. Classifier was logistic regression.



Model	Error rate (Positive/ Negative)	Error rate (Fine-grained)
Naïve Bayes (Socher et al., 2013b)	18.2 %	59.0%
SVMs (Socher et al., 2013b)	20.6%	59.3%
Bigram Naïve Bayes (Socher et al., 2013b)	16.9%	58.1%
Word Vector Averaging (Socher et al., 2013b)	19.9%	67.3%
Recursive Neural Network (Socher et al., 2013b)	17.6%	56.8%
Matrix Vector-RNN (Socher et al., 2013b)	17.1%	55.6%
Recursive Neural Tensor Network (Socher et al., 2013b)	14.6%	54.3%
Paragraph Vector	**12.2%**	**51.3%**

Paragraph vector performance on Stanford sentiment data set



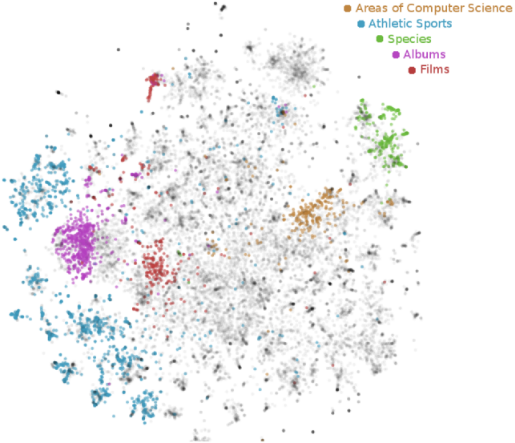
Model	Error rate
BoW (bnc) (Maas et al., 2011)	12.20 %
BoW (bΔt'c) (Maas et al., 2011)	11.77%
LDA (Maas et al., 2011)	32.58%
Full+BoW (Maas et al., 2011)	11.67%
Full+Unlabeled+BoW (Maas et al., 2011)	11.11%
WRRBM (Dahl et al., 2012)	12.58%
WRRBM + BoW (bnc) (Dahl et al., 2012)	10.77%
MNB-uni (Wang & Manning, 2012)	16.45%
MNB-bi (Wang & Manning, 2012)	13.41%
SVM-uni (Wang & Manning, 2012)	13.05%
SVM-bi (Wang & Manning, 2012)	10.84%
NBSVM-uni (Wang & Manning, 2012)	11.71%
NBSVM-bi (Wang & Manning, 2012)	8.78%
Paragraph Vector	**7.42%**

Paragraph vector performance on IMDB data set

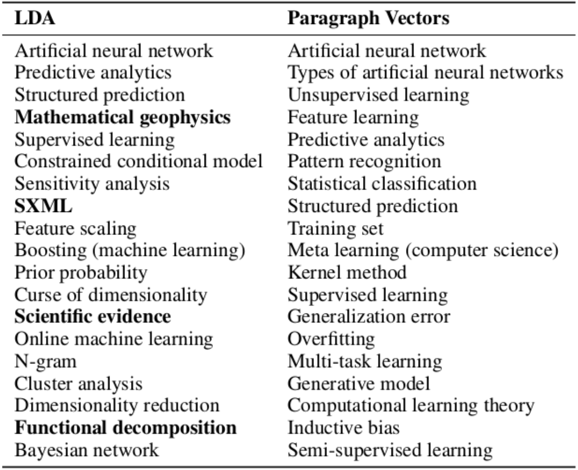
Performance on other tasks[3]

- Data sets: arXiv repository (886000 papers), Wikipedia corpus(4490000 articles, 915715 words).
- Word and document vectors trained simultaneously - PV-DBoW.

[3] Document embedding with paragraph vectors, arXiv:1507.07998, 29 Jul. 2015.

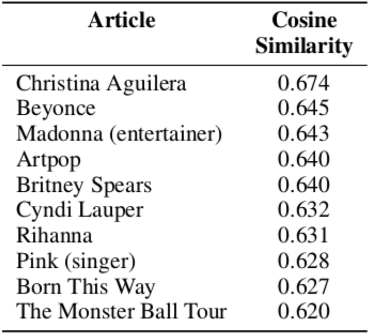


Paragraph vector: Wikipedia t-SNE visualization



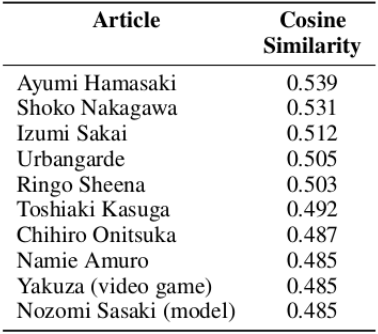
LDA	Paragraph Vectors
Artificial neural network	Artificial neural network
Predictive analytics	Types of artificial neural networks
Structured prediction	Unsupervised learning
Mathematical geophysics	Feature learning
Supervised learning	Predictive analytics
Constrained conditional model	Pattern recognition
Sensitivity analysis	Statistical classification
SXML	Structured prediction
Feature scaling	Training set
Boosting (machine learning)	Meta learning (computer science)
Prior probability	Kernel method
Curse of dimensionality	Supervised learning
Scientific evidence	Generalization error
Online machine learning	Overfitting
N-gram	Multi-task learning
Cluster analysis	Generative model
Dimensionality reduction	Computational learning theory
Functional decomposition	Inductive bias
Bayesian network	Semi-supervised learning

Paragraph vector vs LDA: nearest neighbour for *machine learning*



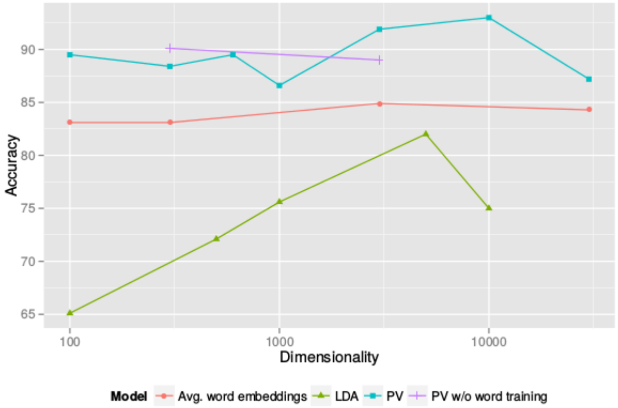
Article	Cosine Similarity
Christina Aguilera	0.674
Beyonce	0.645
Madonna (entertainer)	0.643
Artpop	0.640
Britney Spears	0.640
Cyndi Lauper	0.632
Rihanna	0.631
Pink (singer)	0.628
Born This Way	0.627
The Monster Ball Tour	0.620

Paragraph vector: nearest neighbour to ’Lady Gaga’

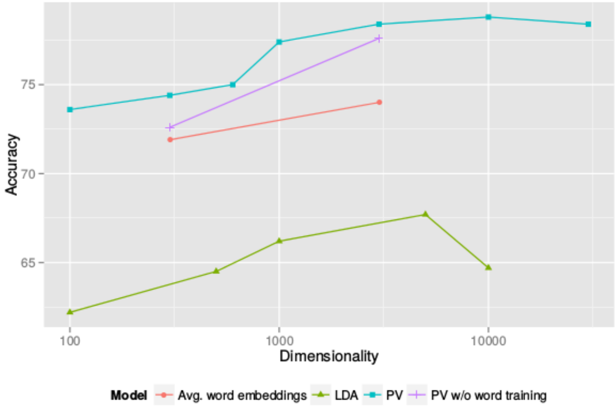


Article	Cosine Similarity
Ayumi Hamasaki	0.539
Shoko Nakagawa	0.531
Izumi Sakai	0.512
Urbangarde	0.505
Ringo Sheena	0.503
Toshiaki Kasuga	0.492
Chihiro Onitsuka	0.487
Namie Amuro	0.485
Yakuza (video game)	0.485
Nozomi Sasaki (model)	0.485

Paragraph vector: nearest neighbour to pv(‘Lady Gaga’P-wv(‘American’)+wv(‘Japanese’)



Handmade triplet articles data set (172 triplets)



Wikipedia triplets data set (19876 triplets)

Problem specific embeddings

- In E-commerce applications products are labelled using a hierarchical catalogue tree.
  Example 1: *Home decor*→*Lights and lamps*→*Bulbs*→*LED bulbs*
  Example 2: *Home furnishings*→*Bed*→*Pillows and pillow covers*→ *Pillow covers*
- A large number of sellers sell on such platforms. They normally give a short product description but do not provide catalogue based tags.
- The problem was: given a textual product description label it by the most appropriate path in the catalogue tree.
- This can be used by the platform search engine to improve both speed and quality of search results.

Graded Weighted Bag of Words Vector[4]

- Each document is represented in a D dimensional space where $D = K * d + K$, K is the number of semantic clusters in the corpus.
- Each document is also concatenated with the *icf (inverse cluster frequency)* - this is calculated using idf values of words in the document.
- Semantic clusters are created from the documents in the corpus using an appropriate clustering algorithm.

[4] Vivek Gupta et al. Product classification in E-commerce using distributional semantics, CoLING, 2016.

```
Data: Documents $D_n$, n = 1 . . . N
Result: Document vectors $gwBo\vec{W}V_{D_n}$, n = 1 . . . N
Train SGNS model to obtain word vector representation $(wv_n)$ using all document $D_n$, $n = 1..N$;
Calculate idf values for all words: $idf(w_j)$, $j = 1..|V|$ ; /* $|V|$ is vocabulary size */
Use K-means algorithm for clustering all words in $V$ using their word-vectors into K clusters;
for $i \in (1..N)$ do
    Initialize cluster vector $c\vec{v}_k = \vec{0}$, $k = 1..K$;
    Initialize cluster frequency $icf_k = 0$, $k = 1..K$;
    while not at end of document $D_i$ do
        read current word $w_j$ and obtain wordvec $w\vec{v}_j$;
        obtain cluster index $k = idx(w\vec{v}_j)$ for wordvec $w\vec{v}_j$;
        update cluster vector $c\vec{v}_k += w\vec{v}_j$;
        update cluster frequency $icf_k += idf(w_j)$;
    end
    obtain $gwBo\vec{W}V_{D_i} = \bigoplus_{k=1}^{K} c\vec{v}_k \oplus icf_k$ ;          /* $\oplus$ is concatenation */
end
```