Text Models for Product Matching

Models Experimented

1. TF-IDF
2. Word2Vec
    a. CBOW
    b. Skipgram
3. Glove
4. N-gram models
    a. Bi-gram
5. FastText
6. Spacy
7. BERT
    a. Feature embeddings extraction
    b. Pre-trained

Dataset (e-comm Products) description

- Dataset is split in 80:20 % train test ratio. Didn’t use stratified sampling because of not having proper target label to split upon
- Selecting 5 text features for training - `title, match_title, description, match_description, Color`

Pre-processing steps:

- Converting text to Lowercase
- Replacing `'&'` char with `'and'` token for keeping the meaning of sentences intact
- Removing HTML Tags
- Substituting `'-'` with `whitespace` token to avoid incorrect merging of words in tokenization step
- Removing special chars
- Removing numericals from text strings
- Performing tokenization with nltk
- Avoided stopwords removal, lemmatization & stemming to preserve overall sentence wise meaning.

Link for code / notebooks:

Testing Objective:

Please refer the following doc for test cases: https://docs.google.com/document/d/1_-iJPI6x_oUHwk_7a0iNz1IEKsKTxAjw8qXh5sqnttk/edit?usp=sharing

Qualitative analysis:

- Matching Product titles
- Word similarity pairs
- Context / Ambiguity analysis
- Alphanumeric words
- Recognising spell mistakes / Out of vocabulary words
- Effect of prefix/suffix
- Change in embeddings of Singular / plural words
- Combined Phrases (n-grams) detection

Quantitative analysis:

- Avg. score of model with test data (1 Lakh product pairs)

Similarity Metrics used:

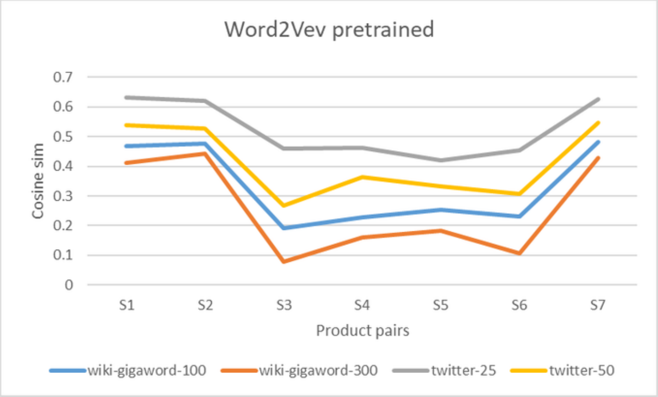
- Since word2vec, glove are word based models and are not suited for sentence embedding as a whole, individual word vectors were combined to represent their sentence embedding in two ways:
  - `Concatenation + padding` with 0's to make embedding dimension equal
  - `Averaging` among all the words in individual sentences
- Spacy, BERT, S-BERT trains sentence embeddings as a whole, and hence they represent full sentences as single multi-dimensional vector
- Cosine similarity was used to measure score of similarity among two text entities

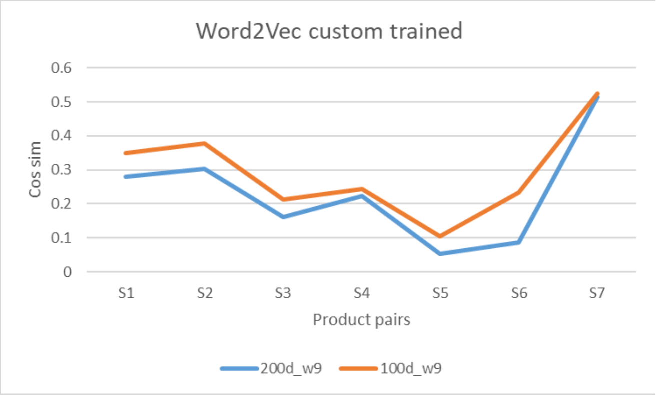
Results

https://docs.google.com/spreadsheets/d/1pgnuFkQd1GPjVndUyOV0iwMRpNwo5OZli4siqEwzjZc/edit?usp=sharing

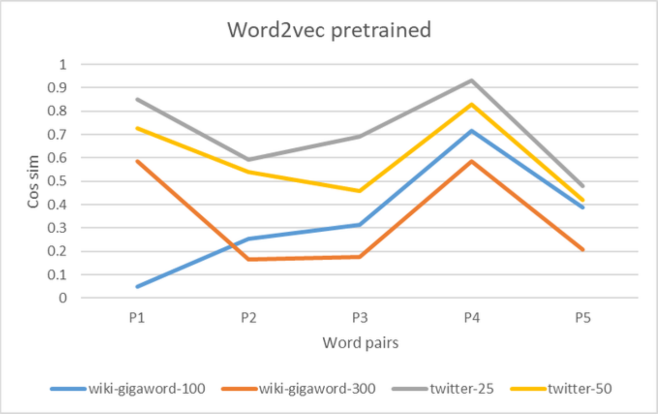
Word2Vec

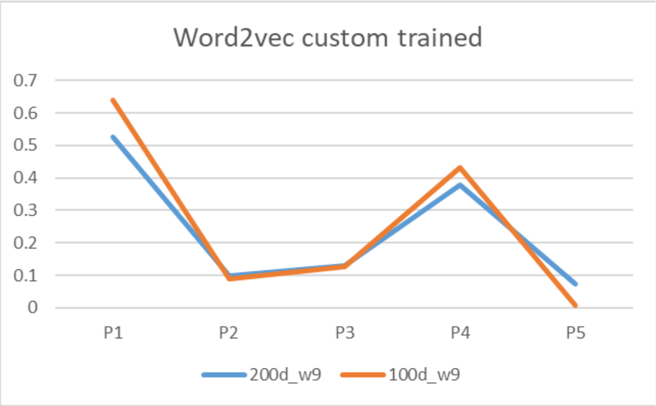
- Has two different architectures: CBOW & Skipgram. By default, CBOW is used in gensim library, and for using Skipgram, set '`sg=1`' while training the model.
- Pre-trained models are hosted in: https://github.com/RaRe-Technologies/gensim-data This has advantage of having basic understanding of language structure as it is trained on large corpus of text
- For custom training, following hyper-parameters are important (in gensim):
  - `window` - context size, it will define the neighbourhood space of each word to consider while training
  - `size` - defines embedding vector dimension
  - `min_count` - used to ignore words appearing in less frequency
  - By experimenting, it was found that optimal config in case of our e-commerce data were (Details of choosing hyperparameters for training: https://docs.google.com/document/d/1MbL_RArBaKY9L1skDnBHQ8PG1242lUlccW96D6WaWQY/edit?usp=sharing):
    - `(window = 9, size = 100, min_count = 1)`
    - setting `min_count=1` has advantage of not missing rare words in small sized dataset, but it can produce poor quality vectors
- Inferences:
- Title pairs from our e-comm dataset, & different cases as per the above defined testing objective were used for analysis. Check Test case doc for details of used test cases https://docs.google.com/document/d/1_-iJPI6x_oUHwk_7a0iNz1IEKsKTxAjw8qXh5sqnttk/edit?usp=sharing
- Below fig shows cosine similarity scores computed for Word2Vec models (on Product title matches)





- Word-similarity tests



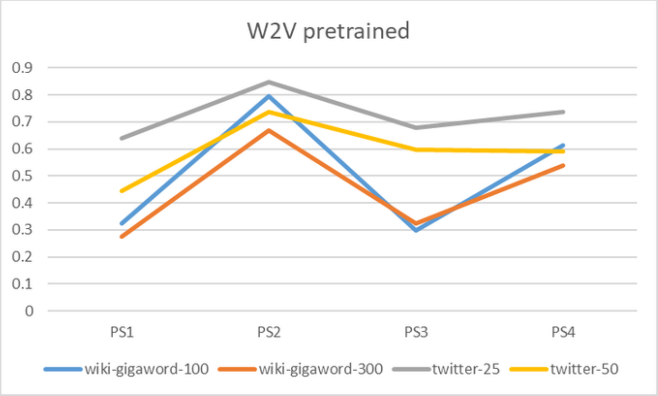


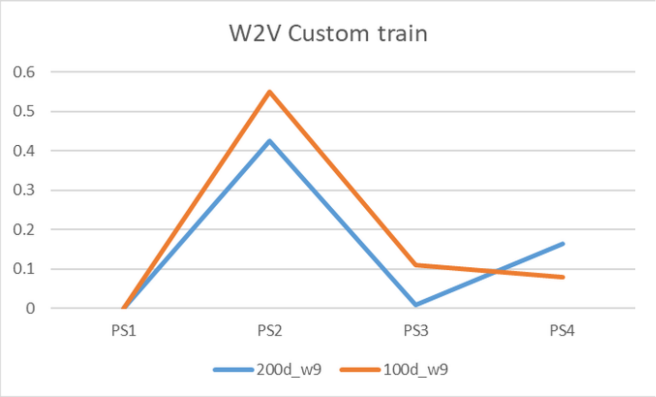
- Alpha-numeric pairs with same meaning

Model	A1	A2
wiki-gigaword-100	0.2849	0.03139
wiki-gigaword-300	0.119	0.0149
twitter-25	0.1089	0.4711
twitter-50	0.06	0.3891

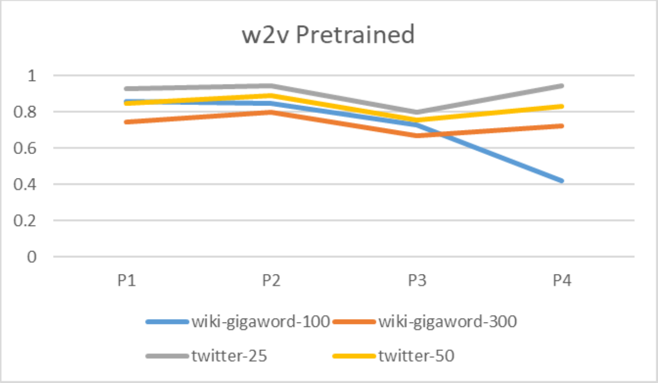
Model	A1	A2
200d_w9	0.0033	0.0135
100d_w9	0.0102	0.0065

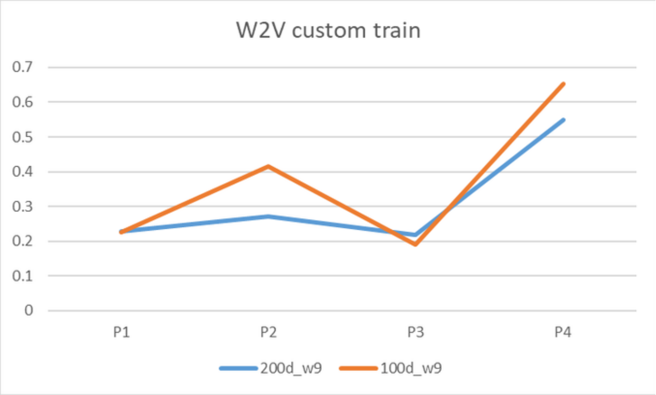
- Effect of prefix/suffix





- Singular / Plural case





- Context Ambiguity case

Model	C1	C2

Model	C1	C2

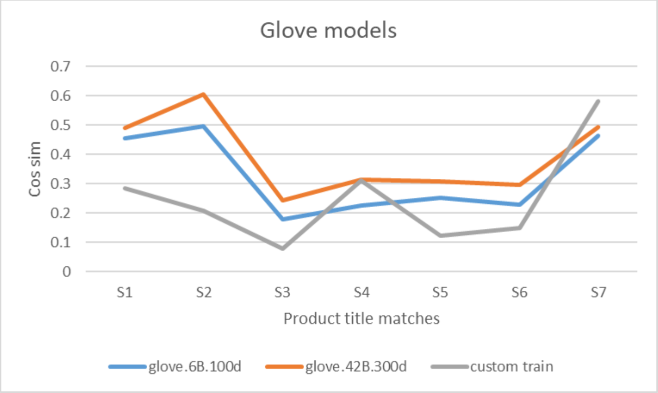
wiki-gigaword-100	0.8624	0.9551
wiki-gigaword-300	0.8232	0.8105
twitter-25	0.9315	0.9734
twitter-50	0.9072	0.8794

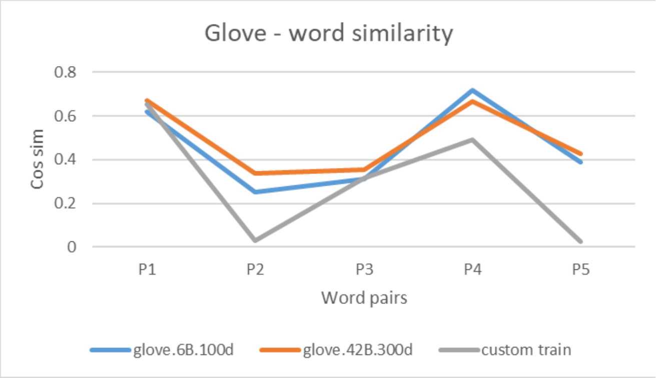
200d_w9	0.7984	0.9825
100d_w9	0.8781	0.9313

Glove

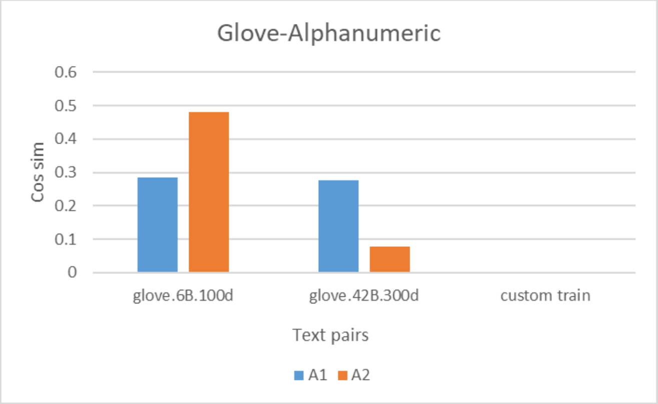
- For pre-trained models, used similar gensim data as of W2Vec, but Glove models differ in how they calculate embeddings. Glove does it by calculating co-occurrence matrix first
- In case of custom training, the pre-trained model was loaded as initial weights and retrained with our product dataset

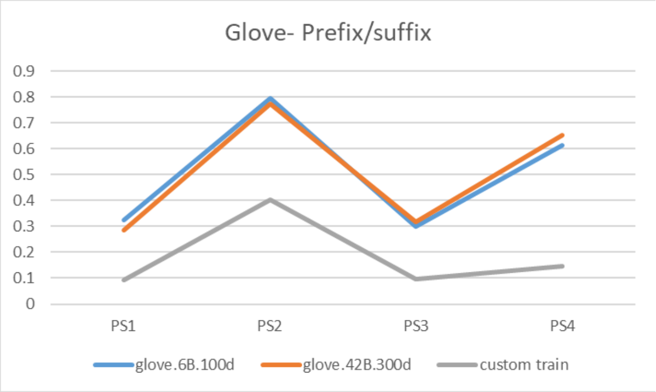
Product Title matches Word-similarity scores



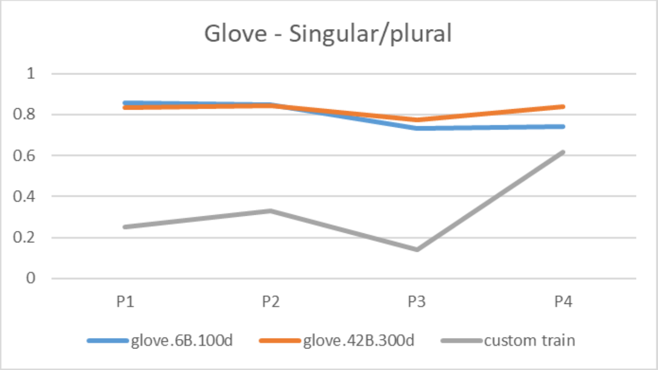


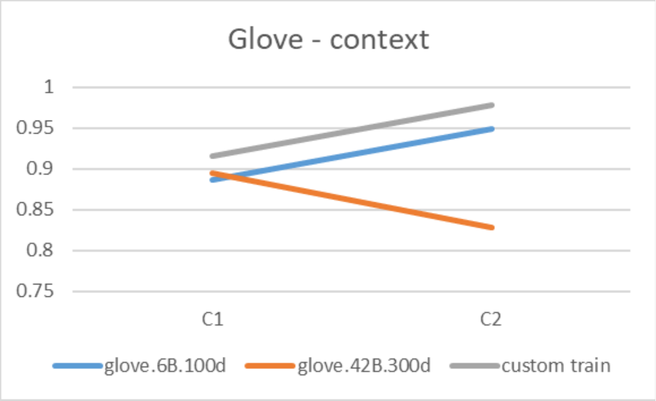
Alpha-numeric pairs Effect of prefix/suffix





Singular / Plural case Context Ambiguity case

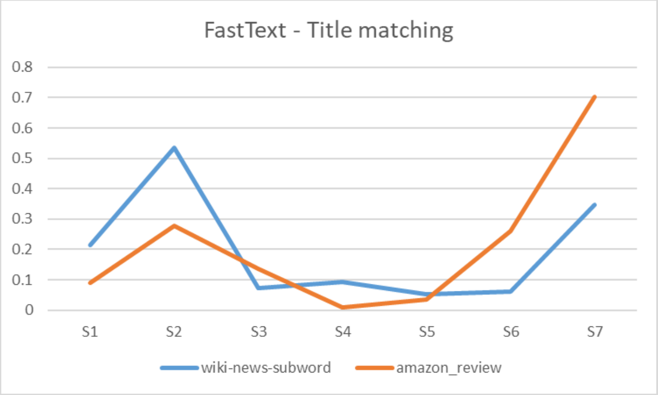


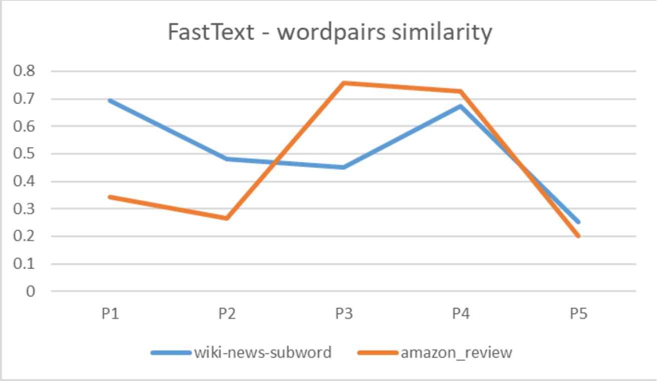


FastText

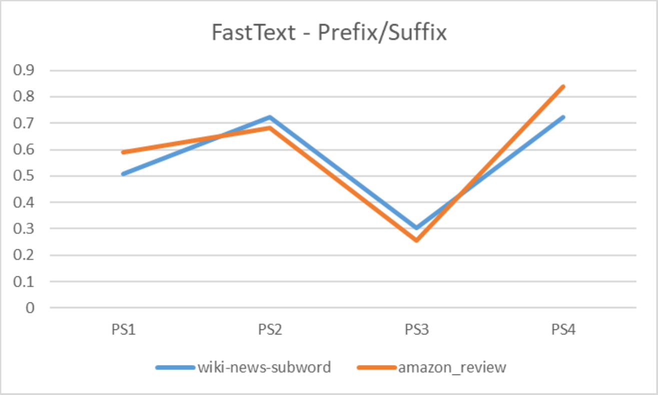
- FastText uses subword tokens (character n-grams) to calculate embeddings, thus making it possible to represent mis-spelled or out-of-vocabulary words in some case.
- FastText operates at a *character* level but Word2Vec operates at a *word* level
- Inferences: FasText was able to represent some phrases such as ‘new_york’, 'water_proof’ etc

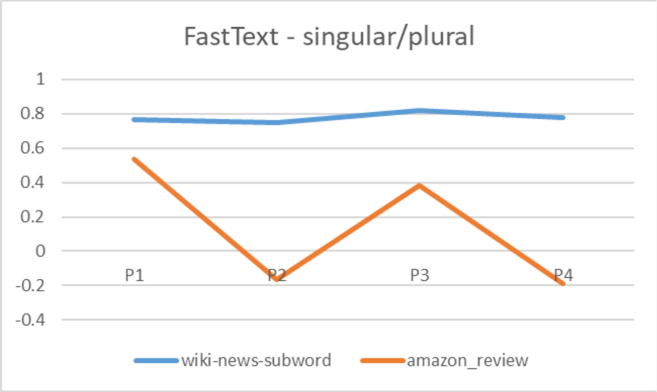
Product Title matches Word-pair similarity



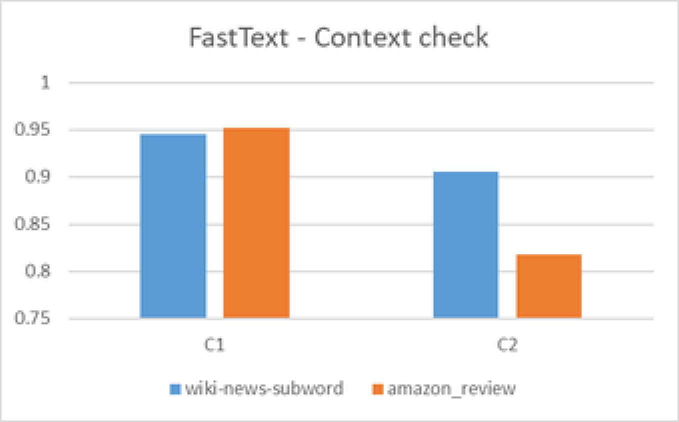


Effect of prefix/suffix Singular / Plural case





Context Ambiguity case



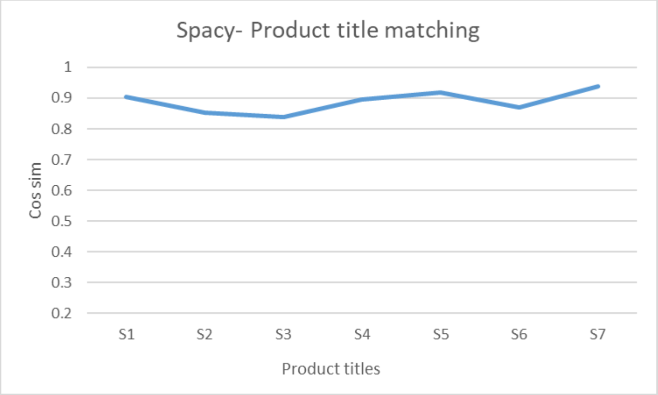
Bi-gram (using Word2vec)

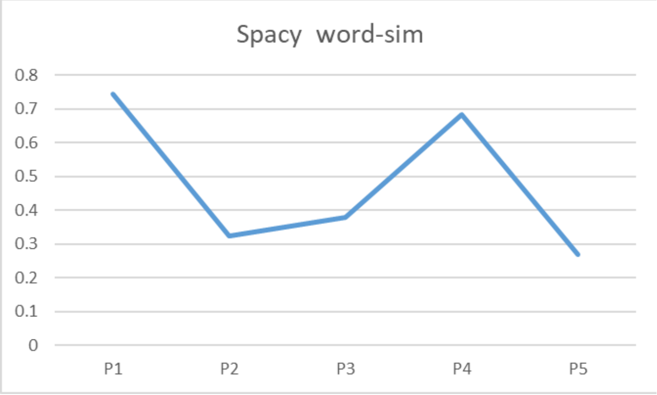
- This model is especially useful in detecting phrases such as ‘wireless charger', 'water proof’ etc.
- This was trained with our product data using bi-gram transformer feature of gensim and then trained the model similar to Word2Vec
- Performance was not better than any previous models shown above, only advantage being it can identify common phrases and provide proper embeddings to them instead of treating them as separate words

Spacy Models

- Spacy models can be found here to load as an API or import as a module: https://spacy.io/models/en
- For this we are not using concatenating word vectors as done in previous models above, because spacy calculates unique single embedding for a sentence as whole, & also provides a similarity checking feature
- Advantage of this is - it provides many other features such as POS tagging, NER, dependency parsing other than just Text similarity
- It has a transformer based model too (RoBERTa base) called '`en_core_web_trf`'
- Inferences - great performance in sentence similarity matching, but has a slight disadvantage (compared to S-BERT model in next section) while working with ambiguous words (eg: Apple fruit v/s Apple PC)

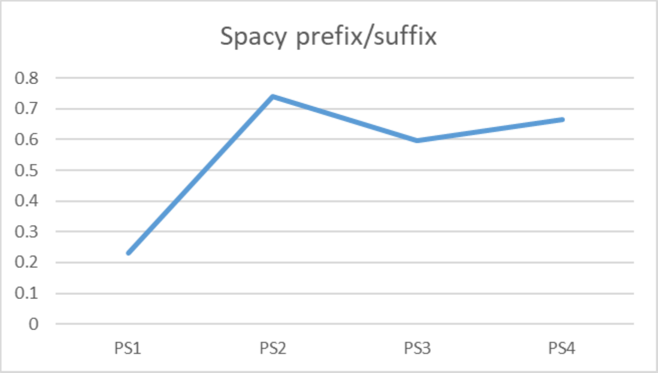
Product Title matches Word-pair similarity

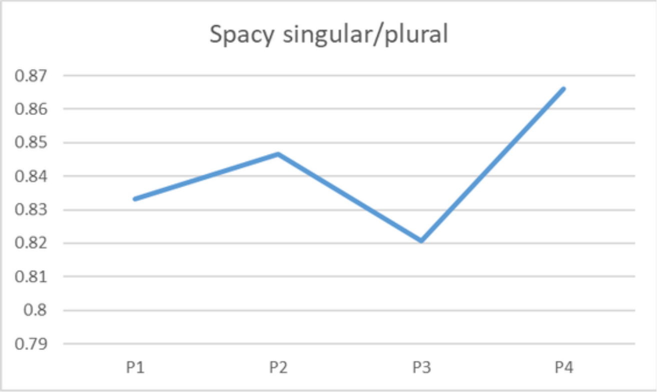




Note: In the word-similarity charts, some pairs are deliberately selected with opposite meaning words (P2, P5) to check model's performance. Check test doc for details https://docs.google.com/document/d/1_-iJPI6x_oUHwk_7a0iNz1IEKsKTxAjw8qXh5sqnttk/edit?usp=sharing

Effect of prefix/suffix Singular / Plural case





Alpha-numeric pairs with same meaning Context / Ambiguity check

Model	3 ½ Pillow, 3.5 Pillow	2M size shirt, shirt
en_core_web_md	0.7066	0.6268

Model	C1	C2
en_core_web_md	0.9428	0.8949

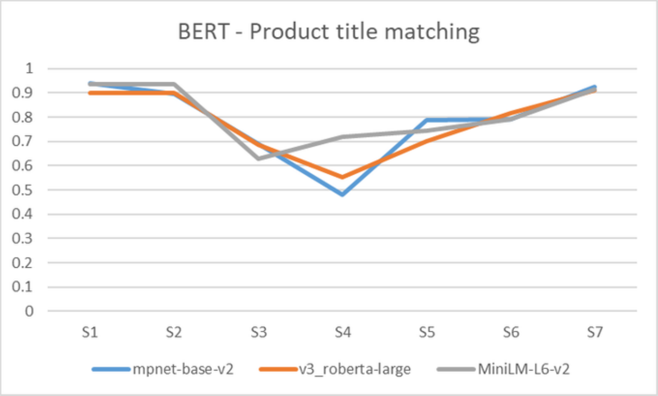
- C1 = (I bought a Apple PC on sale, 'I bought a Apple fruit on sale)
- C2 = (river bank holds surprises for people, Federal bank holds surprise for people)
- Cosine sim score for C1, C2 shouldn't be too high, else it indicates the model didn't capture the context properly

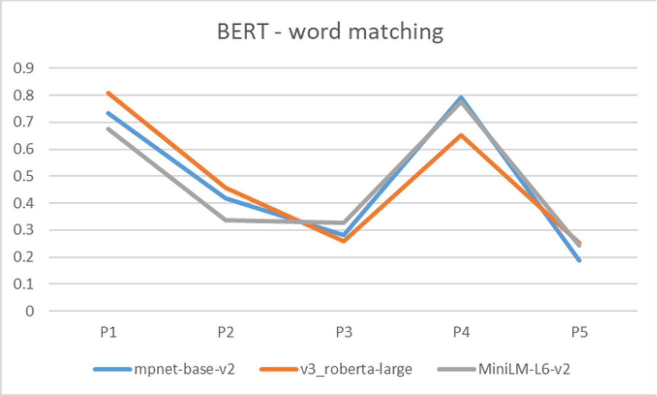
BERT / Sentence-BERT Model

- This is a variation of BERT model specifically designed for computing dense vector representations for **sentences**, **paragraphs** (and even images) https://www.sbert.net/docs/pretrained_models.html
- With BERT, we have an option of extracting embedding layers of Encoder layers in its Architecture, as per the reference http://jalammar.github.io/illustrated-bert/ , the sum of last 4 layers provides best performance.
- Here we test S-BERT's model with base as BERT (`all-mpnet-base-v2`) & RoBERTa (`v3-roberta-large`)
- Inferences -
    - scored slightly less than spacy model in sentence similarity, but excelled at other tasks such as interpreting alpha-numeric phrases, distinguishing different forms of words like singular/plural, prefix/suffix.

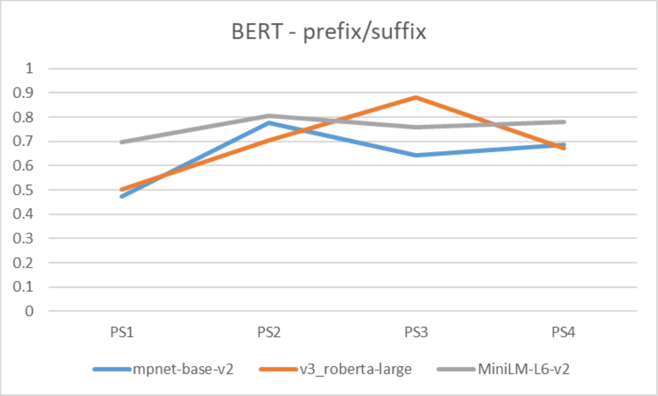
- It was also able to identify combined phrases such as '`water proof`', '`new york`' etc.
- It can even work with mis-spelled words and can provide approximate meaningful embeddings.
- It's best in properly identifying context (distinguished sentences like - Bought an Apple fruit, bought an Apple PC)

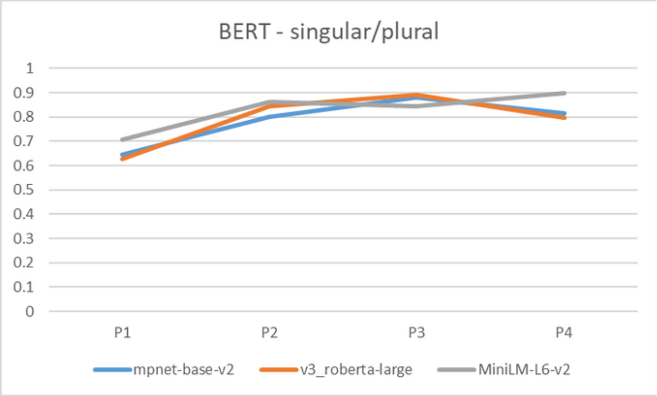
Product Title matches Word-pair similarity



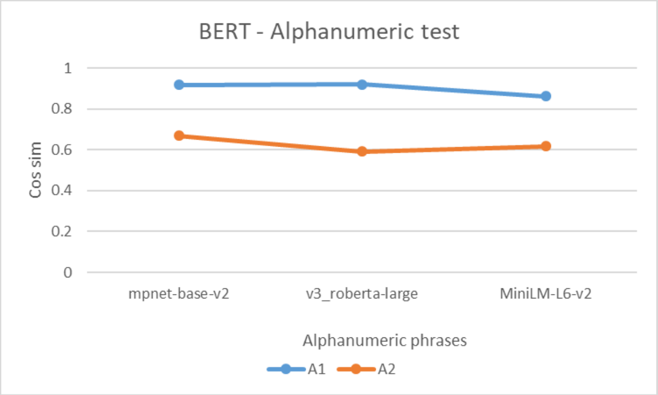


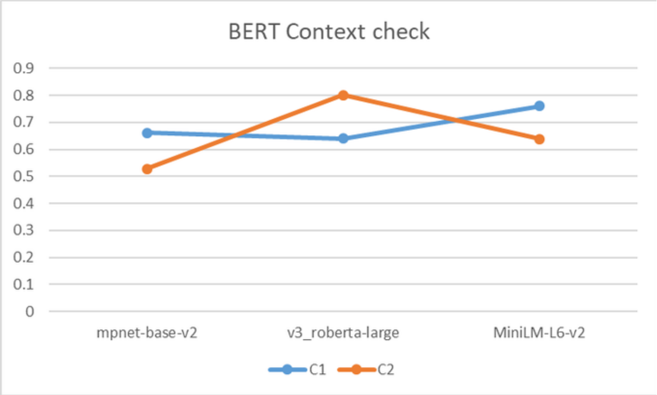
Effect of prefix/suffix Singular / Plural case





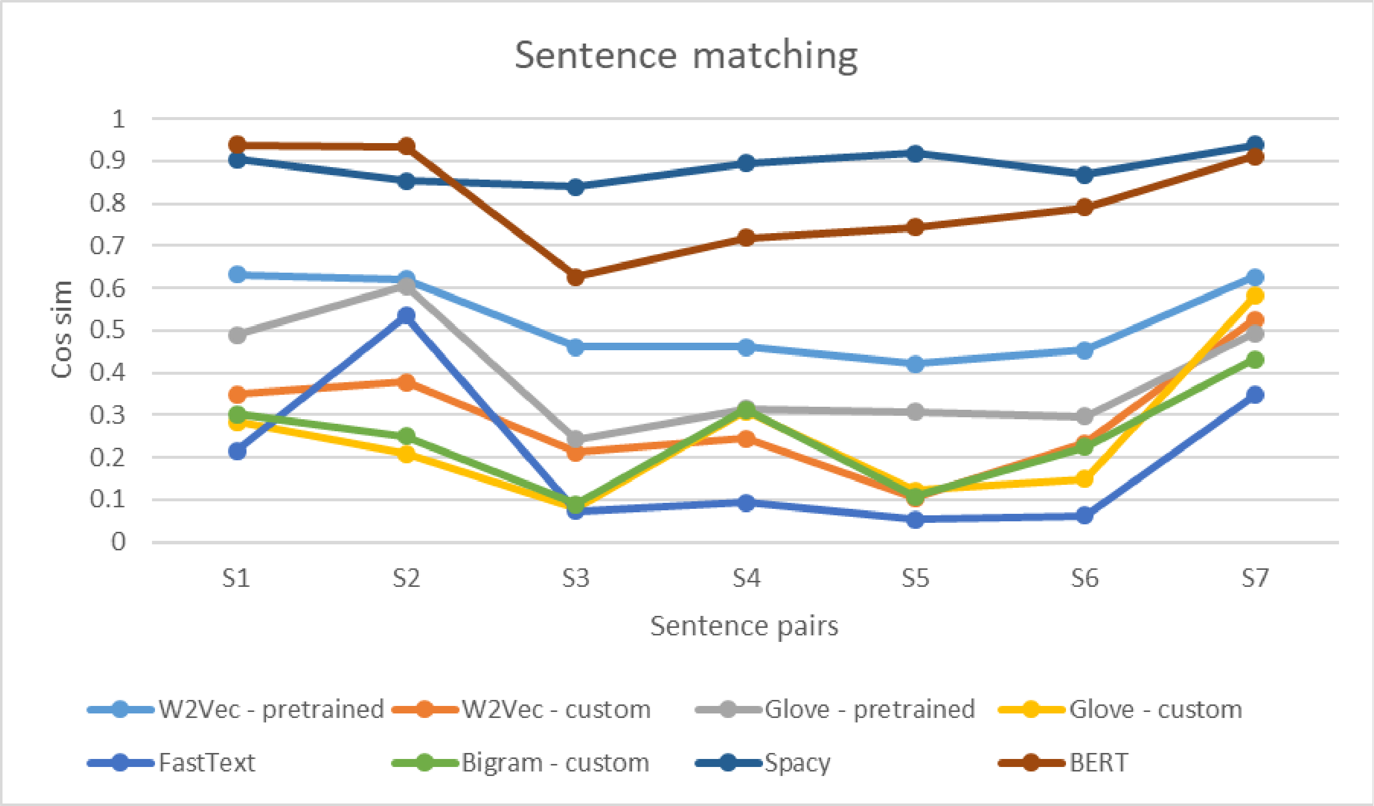
Alpha-numeric pairs with same meaning Context / Ambiguity check





Analysis wrt all models

- Below is the chart for all the models (optimal selected in each category) tested for sentence matching taking product titles as test cases



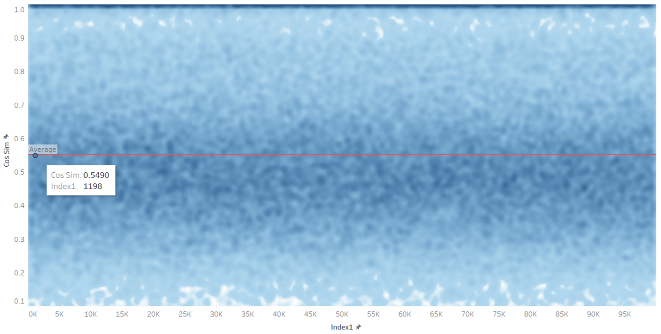
Performance of models on different test objectives

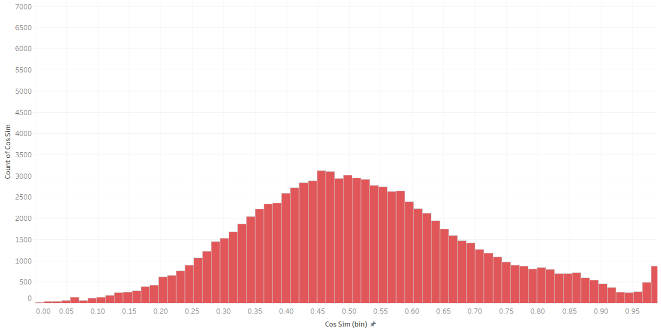
Models	Sentence matching	Word matching	Alpha-numeric words	identifying Prefix / suffix	singular/plural	Context / ambiguity check	Spell-mistakes	Combined phrases
W2Vec pretrained	BAD	GOOD	BAD	GOOD	GOOD	BAD	DOESNT WORK	DOESNT WORK
W2Vec custom trained	BAD	GOOD	BAD	BAD	OKISH	BAD	DOESNT WORK	DOESNT WORK
Glove pretrained	BAD	OKISH	BAD	OKISH	GOOD	BAD	DOESNT WORK	DOESNT WORK
Glove custom trained	BAD	OKISH	BAD	BAD	BAD	BAD	DOESNT WORK	DOESNT WORK
Bigram model (custom train)	BAD	OKISH	BAD	BAD	OKISH	BAD	DOESNT WORK	GOOD
FastText	BAD	GOOD	BAD	GOOD	GOOD	BAD	DOESNT WORK	GOOD
Spacy model	BEST GOOD	GOOD BEST	GOOD	GOOD	GOOD BEST	BAD	DOESNT WORK	GOOD
S-BERT	GOOD	GOOD BEST	GOOD BEST	GOOD BEST	GOOD BEST	GOOD BEST	GOOD BEST	GOOD BEST

Graphs for scores of product title matches in test dataset

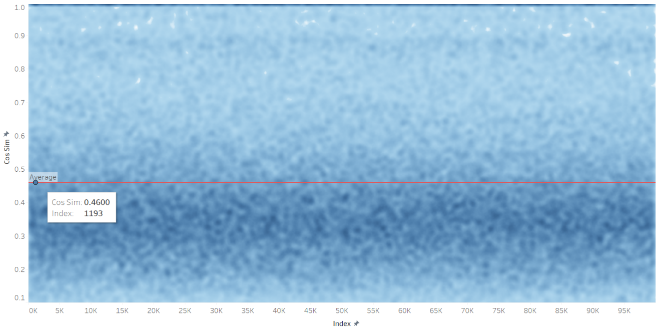
- Density graph shows the region where the cosine sim score of model lies in majority cases, Colormaped according to number of scores in the region
- Average line indicates avg. cosine similarity scores of that particular model
- Histogram plot shows count of cos sim score in specific ranges
- Tested on `1 Lakh` pairs of product titles
- ‘title’ attribute is checked with ground truth '`match_title`' for similarity score
- Inference - Spacy model (`en_core_web_md`) performs best in sentence matching with `Avg of 0.883` cos sim

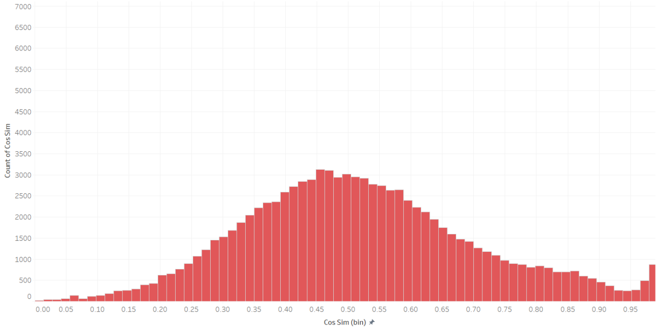
Word2vec (Optimal one selected based on previous results)



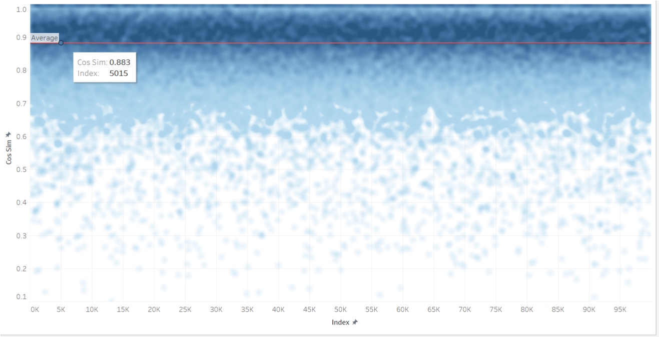


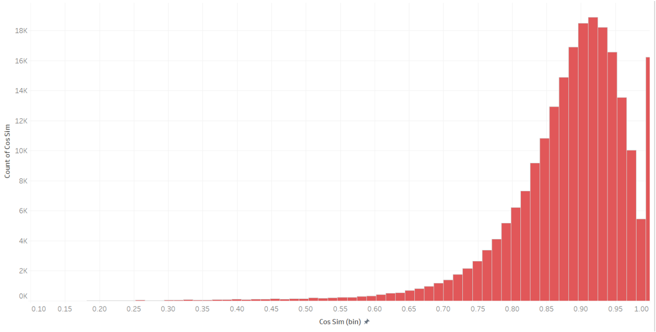
Glove Model



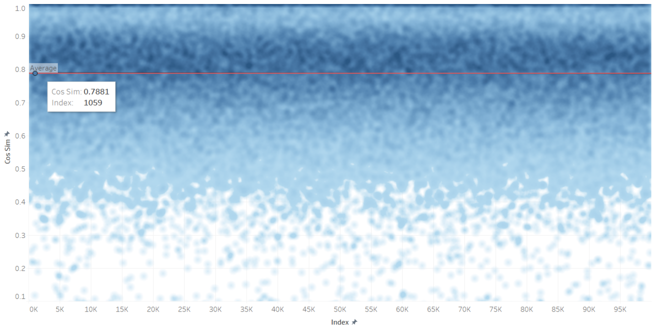


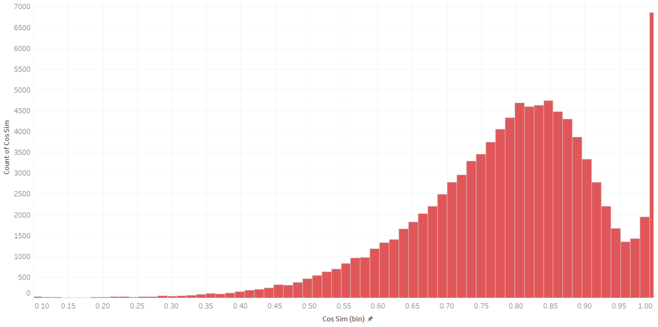
Spacy Model (`en_core_web_md`)





S-BERT Model (`all-mpnet-base-v2`)





Summary

Follow-up

Using custom trained NER for e-comm tags to narrow down text searching space