Assignment 3

Theory

1. **Explain negative sampling. How do we approximate the word2vec training computation using this technique?**

   The idea behind the word2vec models is that the words that appear in the same context (near each other) should have similar word vectors. Therefore, we should consider some notion of similarity in our objective when training the model. This is done using the dot product since when vectors are similar, their dot product is larger.

   The Skip-gram model works in a way that, given an input, it predicts the surrounding or context words. Using this method, we can learn a hidden layer that we'll use to calculate how probable a word is to occur as the context of the input.
   In negative sampling, instead of considering all the words in the vocabulary as positive examples, a small number of negative samples are randomly chosen from the vocabulary. The objective of negative sampling is to train the model to distinguish between positive and negative examples. The negative samples are chosen such that they are unlikely to appear in the context of the positive word, hence they represent unlikely or false contexts.
   The probability of a word being chosen as a negative sample is proportional to its frequency in the corpus, raised to the power of a parameter called the "sampling factor". Words with higher frequency will be more likely to be chosen as negative samples, while words with lower frequency will be less likely to be chosen.

   According to the original description of the Skip-gram model, the objective of this model is to maximise the average log-probability of the context words occurring around the input word over the entire vocabulary:

   $$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log(p(w_{t+j}|w_t))$$

   where T is all the words in the training data and c is the training context window. The size of this window can affect the training accuracy and computational cost. Larger windows lead to higher accuracy as well as higher computational cost. One way to calculate the above probability is to use the softmax function:

   $$p(w_O|w_I) = \frac{exp(v'^T_w v_{w_I})}{\sum_{w=1}^{W} exp(v'^T_w v_{w_I})}$$

   where v_w and v'_w are the vector representations of the word w as the input and output, respectively. Also, W i the number of words in the entire vocabulary.

   However, the main problem comes up when we want to calculate the denominator, which is a normalizing factor that has to be computed over the entire vocabulary. Considering the fact the size of the vocabulary can reach hundreds of thousands or even several million words, the computation becomes intractable. This is where negative sampling comes into play and makes this computation feasible.

So by defining a new objective function, negative sampling aims at maximising the similarity of words in the same context and minimising it when they occur in different contexts.

2. **Explain the concept of semantic similarity and how it is measured using word embeddings. Describe at least two techniques for measuring semantic similarity using word embeddings.**

   Semantic similarity is a concept in natural language processing that measures the degree of relatedness or similarity between two words or phrases based on their meanings. It is a fundamental task in many NLP applications such as information retrieval, text classification, and machine translation.

   Word embeddings are a popular method for representing words as vectors in a high-dimensional space, where words with similar meanings are closer together. There are different techniques for constructing word embeddings, such as Word2Vec and GloVe. Once the embeddings are constructed, semantic similarity can be measured by computing the distance between the word vectors.

   Here are two techniques for measuring semantic similarity using word embeddings:

   1. Cosine similarity: This technique measures the cosine of the angle between two vectors, which ranges from -1 to 1. A cosine similarity of 1 indicates that the two words have the same meaning, while a cosine similarity of -1 indicates that they have opposite meanings. A cosine similarity of 0 means that the two words are orthogonal, i.e., they have no semantic relationship.
   2. Euclidean distance: This technique measures the distance between two vectors in the embedding space. A smaller distance indicates a higher degree of semantic similarity. However, this technique has some limitations, such as the fact that it doesn't take into account the direction of the vectors, which can lead to incorrect results for some pairs of words.

   In practice, there are other more sophisticated techniques for measuring semantic similarity, such as using neural networks or graph-based methods. However, the basic idea is the same: representing words as vectors and measuring their similarity based on their distance or angle in the embedding space.

Implementation

Pre Processing

`Lemmatization` The code uses the WordNetLemmatizer class from the nltk library to lemmatize each word in the input data lines. Lemmatization is the process of reducing each word to its base or dictionary form, so that related words can be treated as a single term in subsequent processing steps.

`Word Frequency` The code tokenizes each line into individual words and counts the frequency of each word in the input data. This step is important to identify frequent and rare words in the data, which can be used to filter out irrelevant or infrequent terms during subsequent processing.

`Filtering Rare Words` The code replaces words with frequency less than a minimum threshold (minFreq) with a special token <UNK>. This is a common way to handle infrequent words, which are unlikely to contribute to the overall meaning of the text, and can instead introduce noise and complexity in the analysis.

The preprocessed data is saved in a file for further use.

Co-occurrence Matrix with SVD

1. The class `SVD` is defined which takes three parameters as input: window_size, cutoff, and data.
2. The data is the input text corpus which is split into sentences, and each sentence is tokenized using the nltk library's word_tokenize function.
3. The vocabulary is created using the `createVocab` function which iterates over the tokenized sentences and counts the frequency of each word in the vocabulary.
4. A co-occurrence matrix is created using the `createCooccurMatrix` function which iterates over the tokenized sentences again, and for each word in the sentence, it calculates the frequency of occurrence of other words within a given window size. The resulting matrix is a sparse matrix, stored in the self.cooccurMatrix attribute of the class.
5. The `createSVD` function is called which uses the sparse matrix's SVD functionality to factorize the co-occurrence matrix into three matrices U, S, and V, where U and V are orthogonal matrices and S is a diagonal matrix containing singular values.
6. The number of dimensions to be retained is determined by a cutoff value passed to the SVD class, which specifies the maximum proportion of the variance to be retained. The first k singular values are retained, where k is the number of singular values required to meet the cutoff condition.
7. The resulting U matrix is normalized and saved to a file "**emeddings.txt**" using the saveEmbeddings function. Each row of the U matrix represents the embedding vector for a corresponding word in the vocabulary.

CBOW with Negative Sampling

1. `Data` class is defined to preprocess the text data and generate the negative samples and subsampling table. It takes in the training data, minimum frequency to consider a word in the vocabulary, and the size of the discarded negative samples. It generates the vocabulary and keeps track of the word count, sentence count, word frequency, and the negative samples.

2. `Word2vecDataLoader` class is defined which is used as the dataloader for the Word2Vec model. It takes in the `Data` object and the window size for the context words. It preprocesses each sentence by subsampling and padding the start and end of the sentence. For each word in the sentence, it creates a tuple of (context words, target word, negative samples) and appends it to `cbowData` list. Finally, the function returns a list of tuples containing the cbow data.

3. The `collate_fn` function is used as the collate function for the DataLoader. It takes in a list of batched cbow data, and returns a tuple of (padded context words tensor, target words tensor, negative samples tensor). It pads the context words tensor to the maximum length of the context words in the batch. It returns the tuple of the same length as the number of tuples in the input batch.
4. The `CBOWModule` class defines the architecture of the CBOWmodel. The constructor initializes two embedding layers: `targetEmbedding` for the target words and `contextEmbedding` for the context words. The forward method takes in the context, target, and negative samples as inputs, where the context and target are both tensor inputs with dimensions `(batch_size, window_size)` and `(batch_size, 1)` respectively, and negatives is a tensor input with dimensions `(batch_size, num_negative_samples)`. The method

computes the loss function for CBOW training, which consists of the negative log likelihood of the positive samples and the negative samples. The output of the forward method is the mean of the loss over the batch.

5. The `embeddingSave` method of the `skipGramModule` class is used to save the learned embeddings to a file. It takes two inputs: `ind2word`, a dictionary that maps index to word, and `output_file`, the file path to which the embeddings are written.

6. The `CBOW` class is used to train a CBOW model. The constructor takes two inputs: `training_data`, the corpus used for training, and `output_file`, the file path to which the learned embeddings are written. The constructor initializes the `Data` and `Word2vecDataLoader` classes for processing the corpus and generating training data. The `Data` class preprocesses the corpus and creates a vocabulary, while the `Word2vecDataLoader` class creates a dataloader for training the CBOW model.

7. The `train` method of the `CBOW` class trains the CBOW model using the dataloader generated by the `Word2vecDataLoader` class. It initializes the optimizer and the scheduler, and then iterates over the epochs and batches to train the model. For each batch, it computes the loss using the Skip-Gram model's forward method, backpropagates the loss to update the model's weights, and adjusts the learning rate using the scheduler.

Layer-By-Layer Architecture :

```
Initialization: The constructor of the class initializes the various attributes of the module, such as the vocabulary size, embedding size, target embedding, and context embedding. It also initializes the weights of the target embedding using uniform distribution and sets the context embedding weights to zero.

Forward Pass: The forward() method of the class takes in three inputs - context, target, and negatives - and computes the loss for a given target word and its context words. It first retrieves the embeddings for the target and context words using the targetEmbedding and contextEmbedding layers respectively.

Target Embedding: The targetEmbedding layer is an instance of the nn.Embedding class in PyTorch, which creates a lookup table that maps each target word to a dense embedding vector of size embedding_size. In this layer, the input target is first embedded using the targetEmbedding layer to get the target embedding vector.

Average Pooling: After retrieving the target embedding, the forward() method takes the average of the context embeddings across the window size dimension (i.e., dim=1) using torch.mean(). This produces a single embedding vector for the context words.

Context Embedding: The contextEmbedding layer is similar to the targetEmbedding layer, but it is used to embed the context words. The forward() method retrieves the context embeddings using the contextEmbedding layer.

Negative Sampling: The forward() method then retrieves the negative embeddings using the contextEmbedding layer for the negative samples. These negative samples are used to compute the negative sampling loss.
```

```
Dot Product: The forward() method computes the dot product of the target and context embeddings using torch.mul() and torch.sum() to obtain a similarity score between the two embeddings.

Sigmoid Activation: The dot product similarity score is then passed through a sigmoid activation function using F.logsigmoid().

Negative Sampling Loss: The forward() method computes the negative sampling loss by taking the dot product of the negative embeddings with the target embedding, passing the result through a sigmoid activation function, and taking the negative log of the result.

Final Loss: The forward() method returns the average of the positive and negative loss scores.

Embedding Save: Finally, the embeddingSave() method is used to save the learned target embeddings to a file. It retrieves the target embeddings from the targetEmbedding layer and writes them to a file in the format expected by the word2vec C implementation.
```

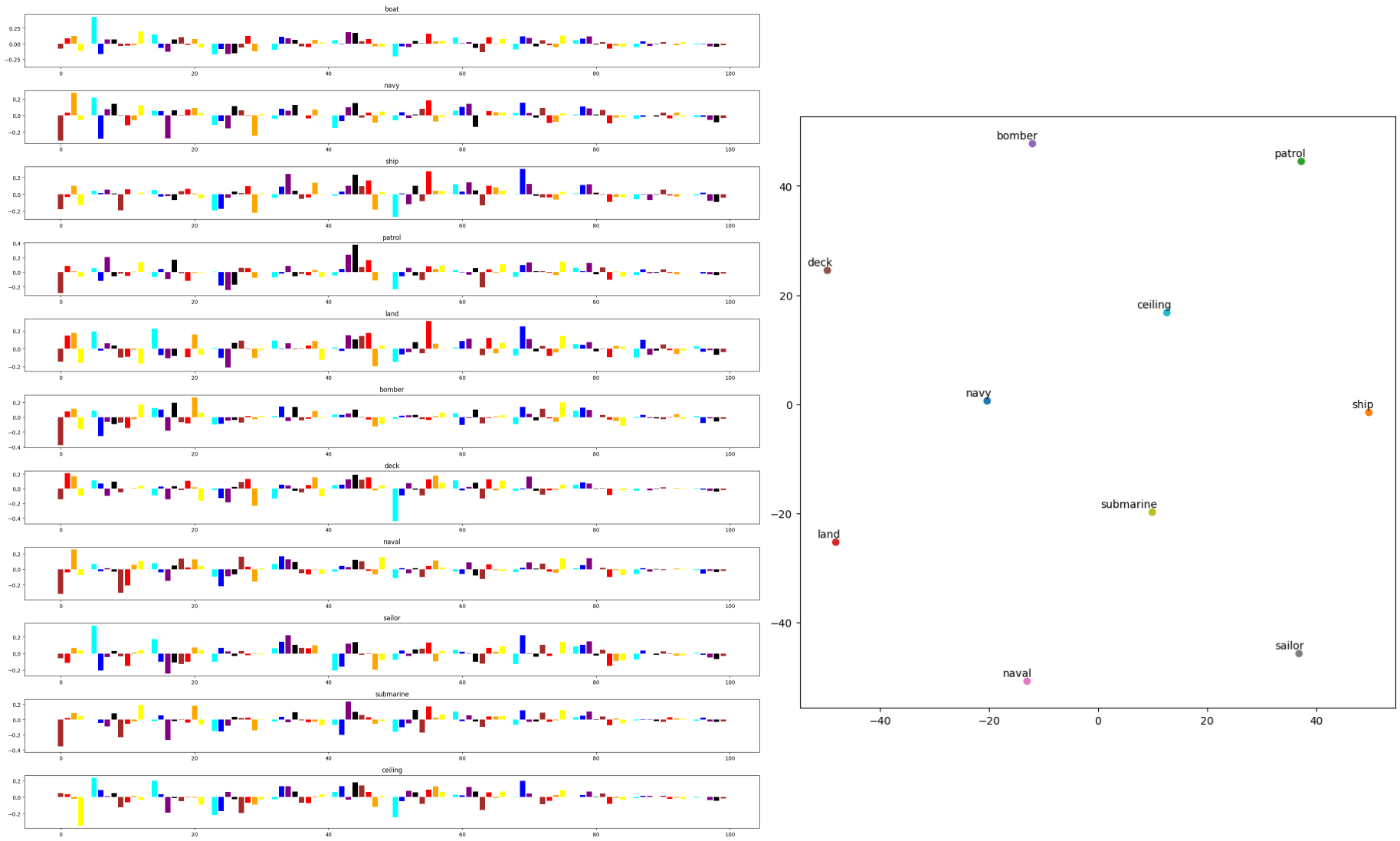
Results

5 Words (Noun, Verb, Pronoun, Adverb, Adjective) are taken as **('boat' , 'review' , 'her' , 'beautifully' , 'smart')**

SVD

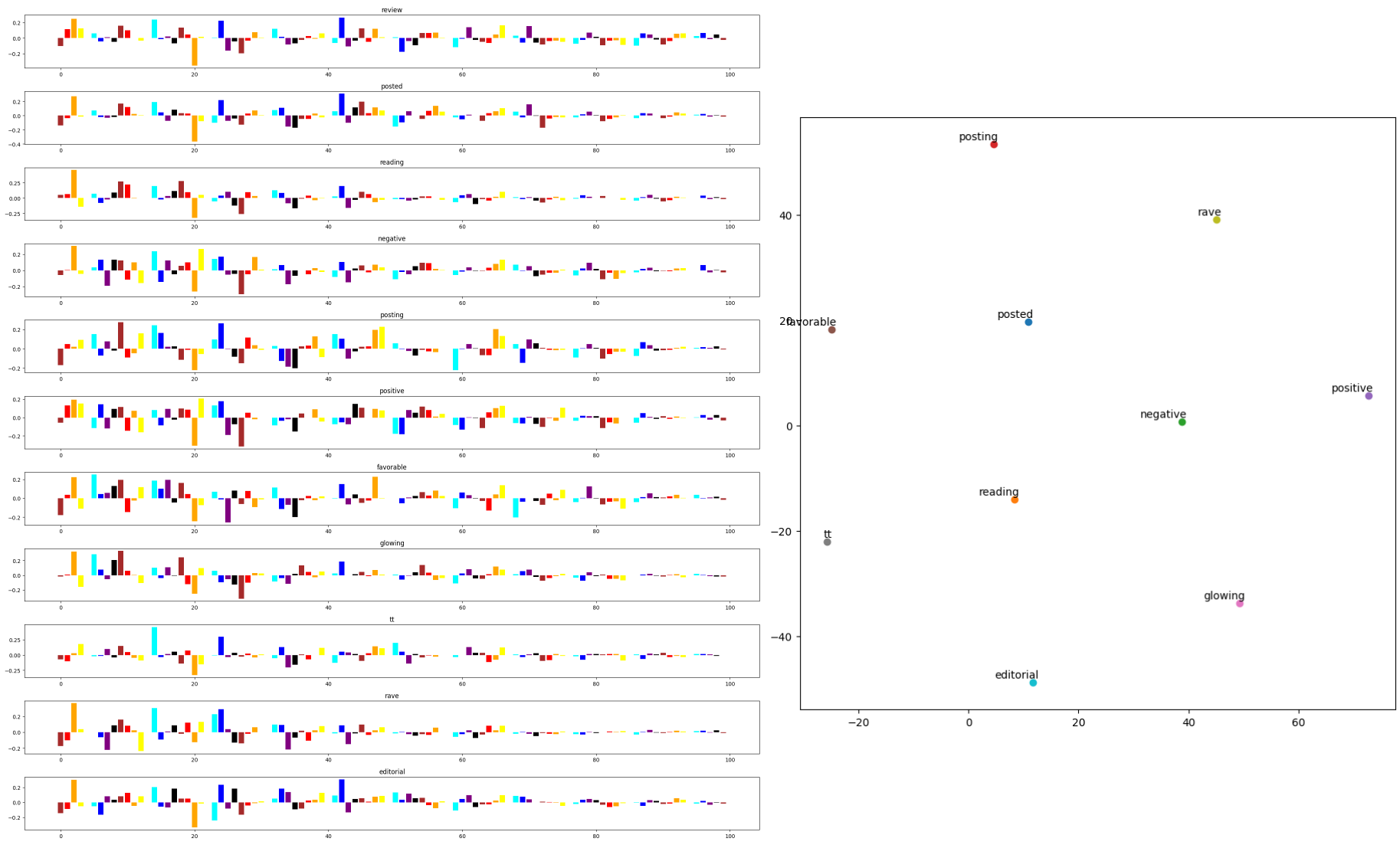
Word: boat

Top 10 similar words: ['navy', 'ship', 'patrol', 'land', 'bomber', 'deck', 'naval', 'sailor', 'submarine', 'ceiling']



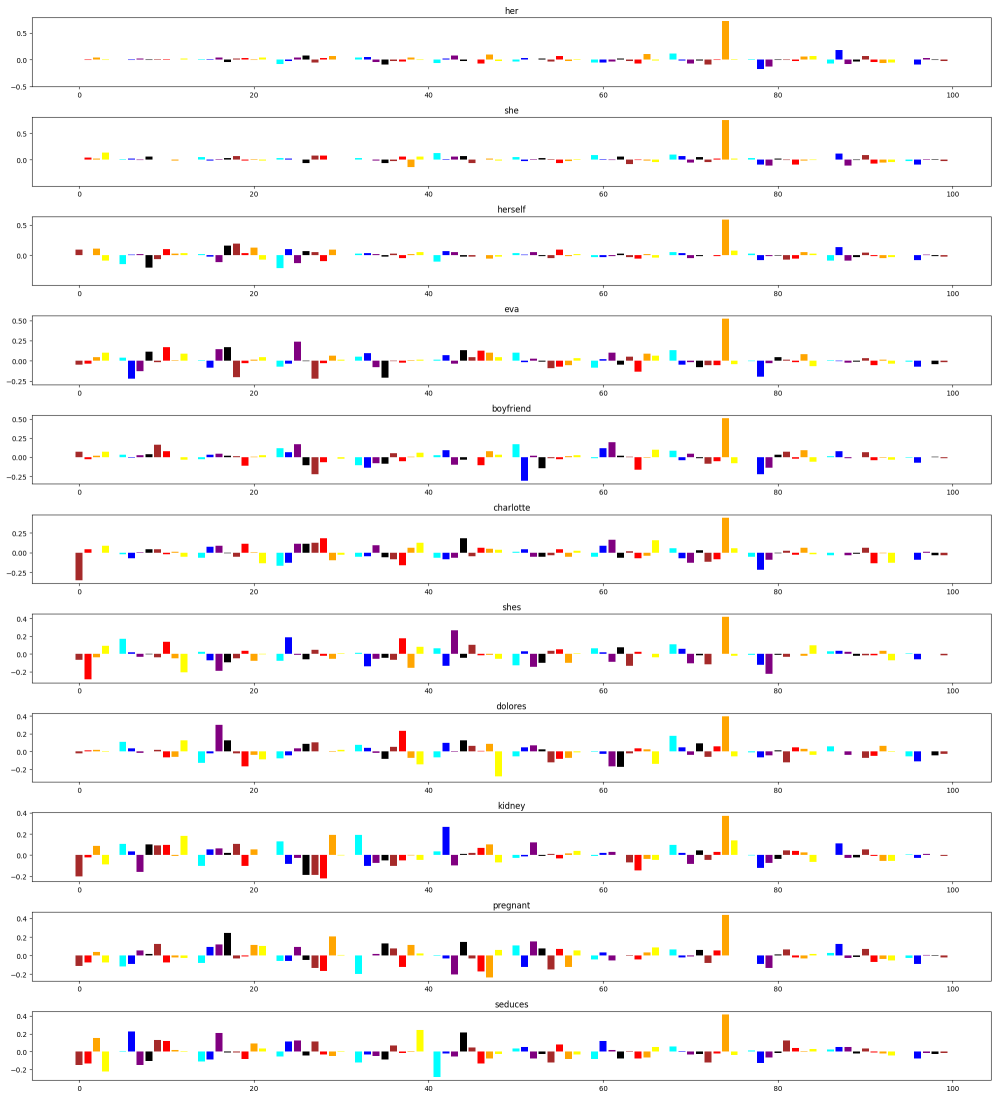
Word: review

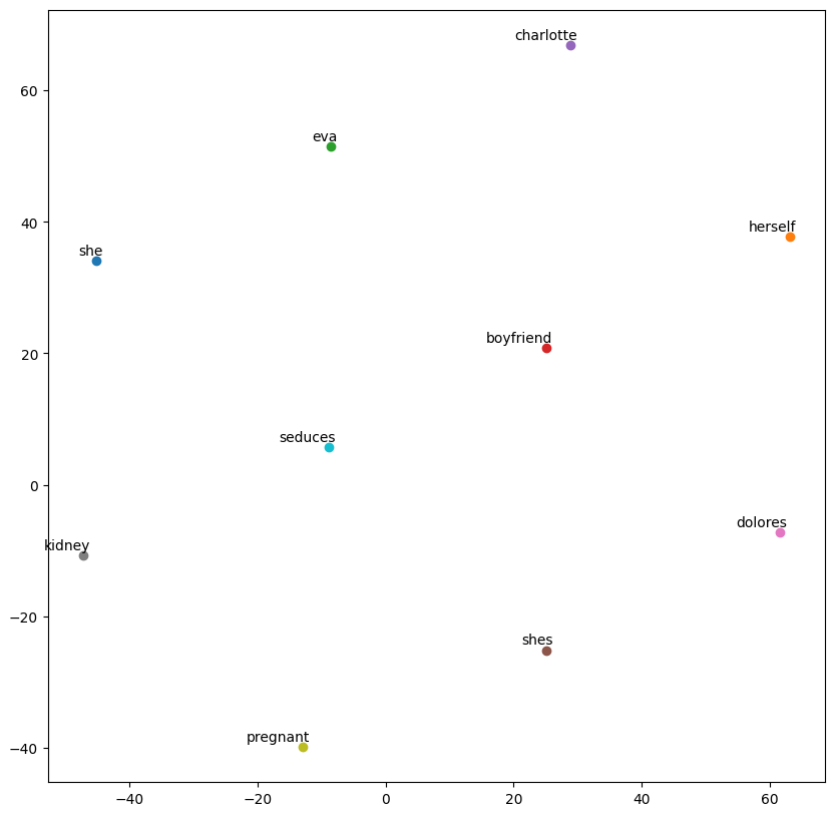
Top 10 similar words: ['posted', 'reading', 'negative', 'posting', 'positive', 'favorable', 'glowing', 'tt', 'rave', 'editorial']



Word: her

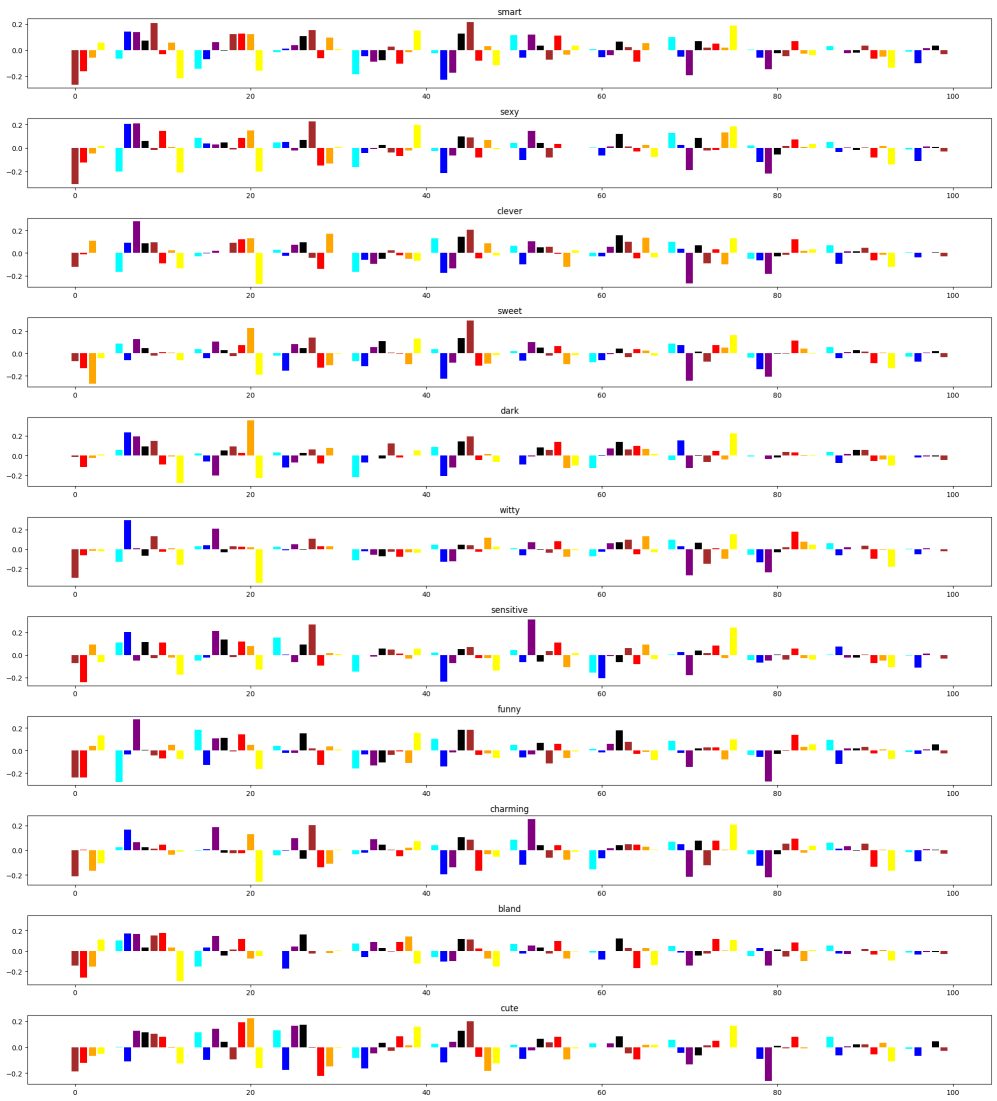
Top 10 similar words: ['she', 'herself', 'eva', 'boyfriend', 'charlotte', 'shes', 'dolores', 'kidney', 'pregnant', 'seduces']

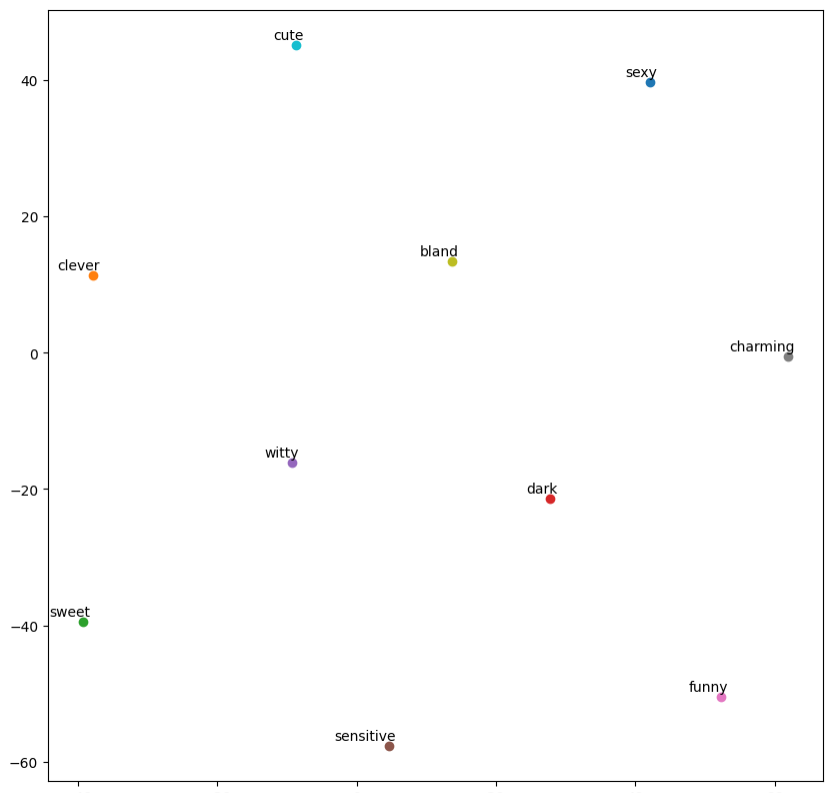




Word: smart

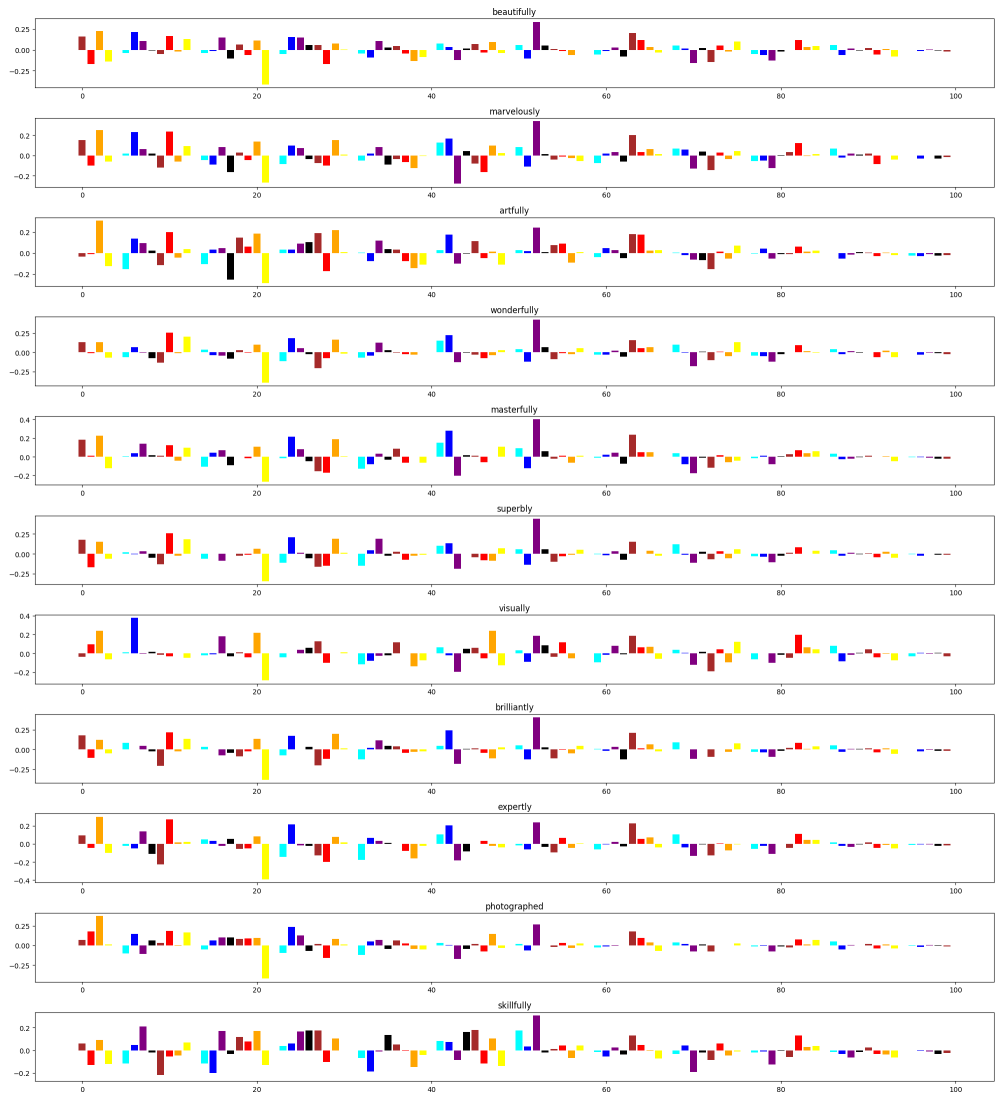
Top 10 similar words: ['sexy', 'clever', 'sweet', 'dark', 'witty', 'sensitive', 'funny', 'charming', 'bland', 'cute']

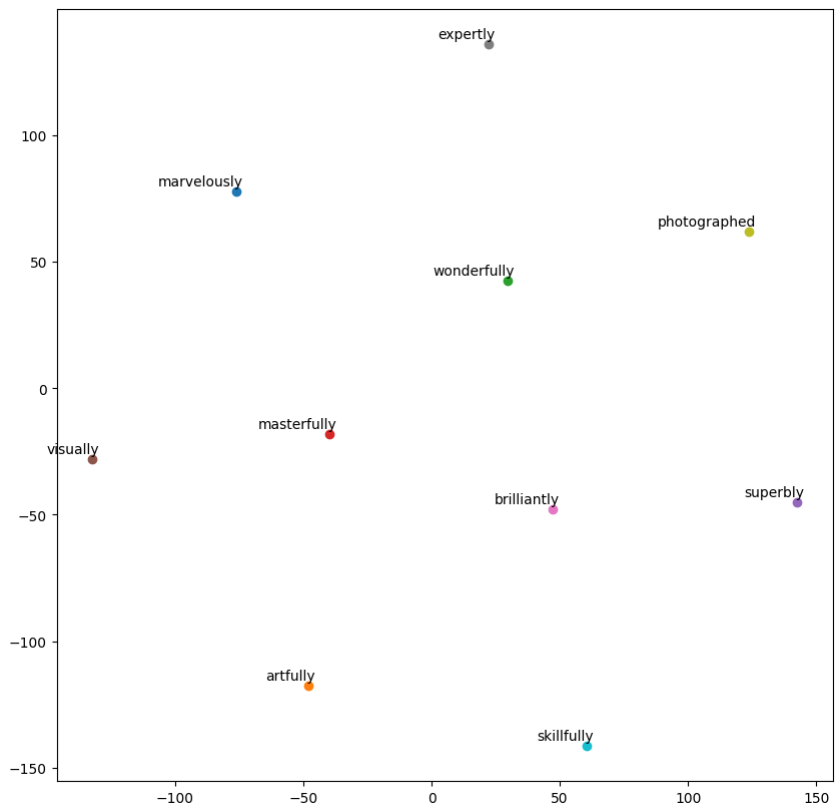




Word: beautifully

Top 10 similar words: ['marvelously', 'artfully', 'wonderfully', 'masterfully', 'superbly', 'visually', 'brilliantly', 'expertly', 'photographed', 'skillfully']

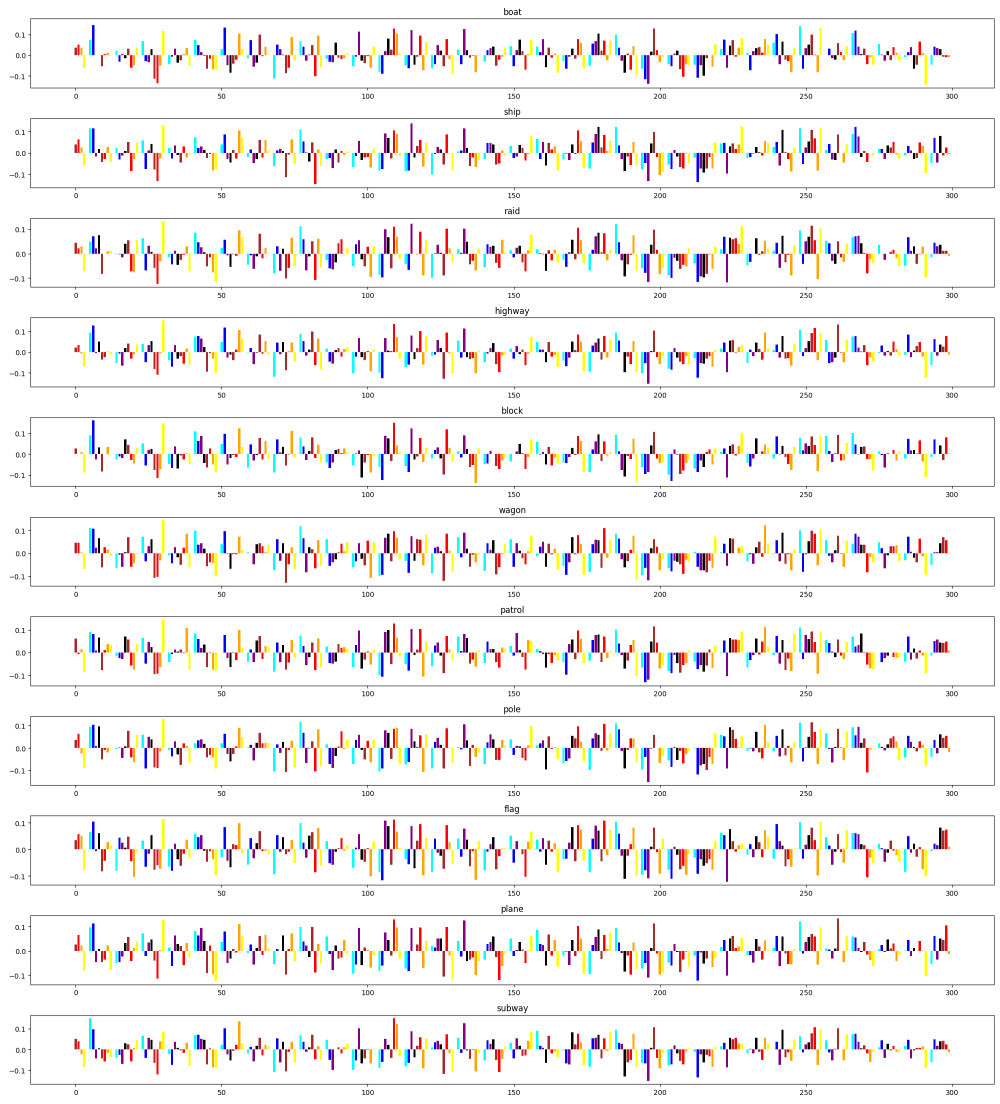


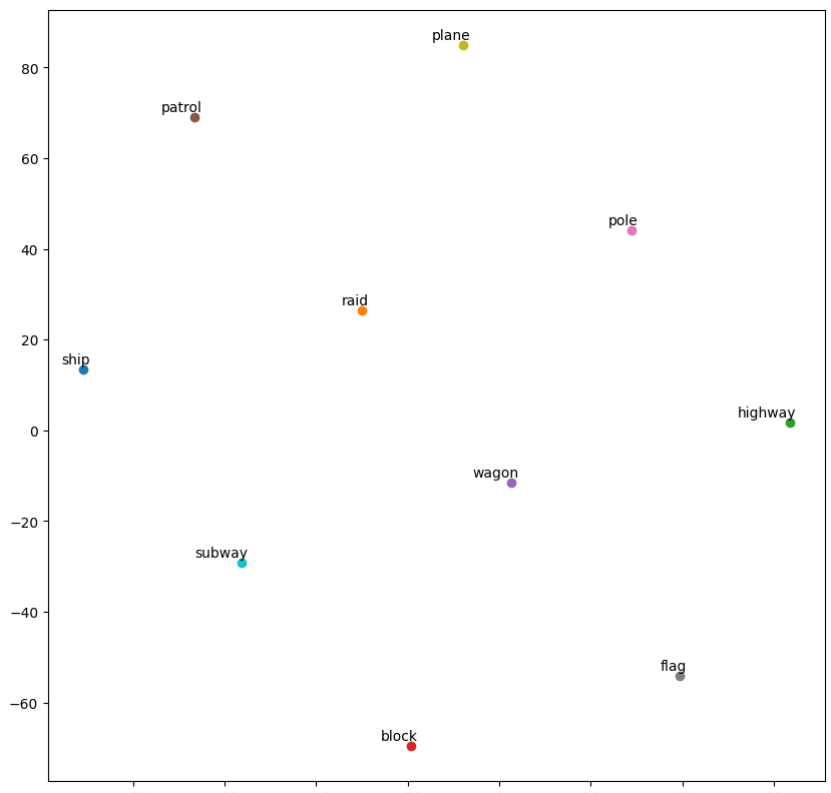


CBOW

Word: boat

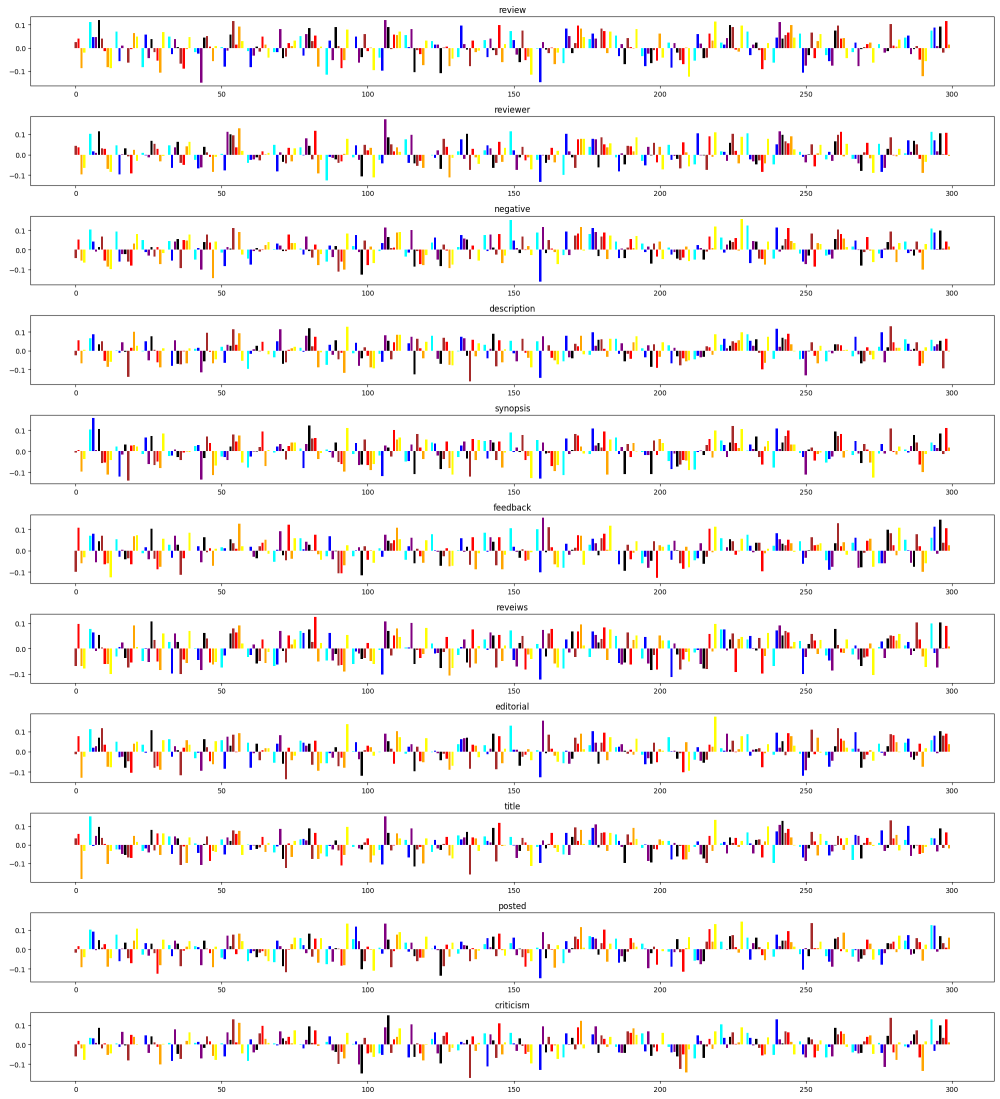
Top 10 similar words: ['ship', 'raid', 'highway', 'block', 'wagon', 'patrol', 'pole', 'flag', 'plane', 'subway']

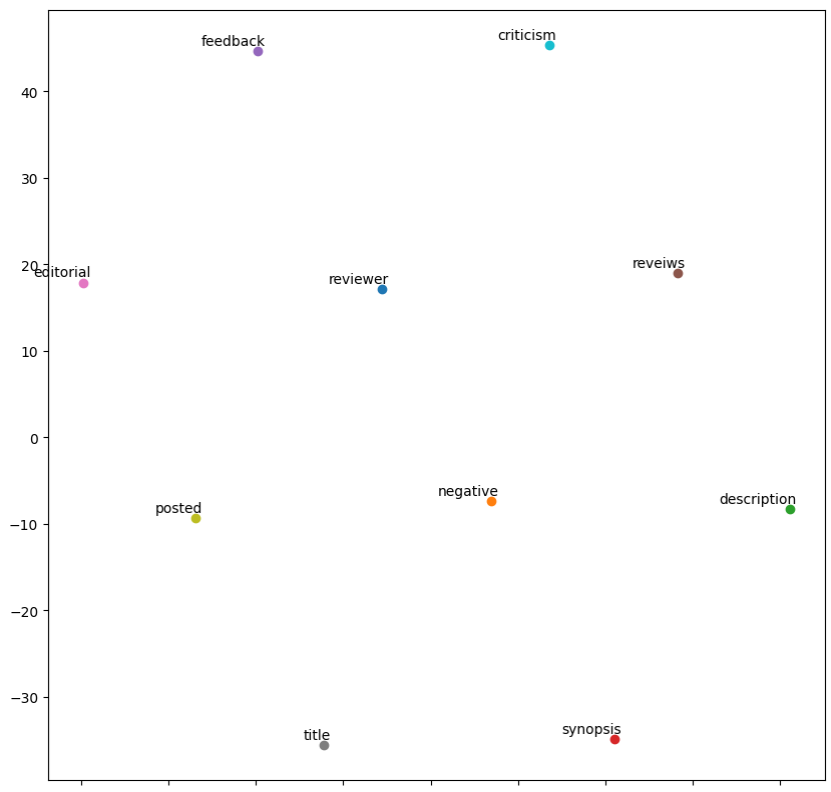




Word: review

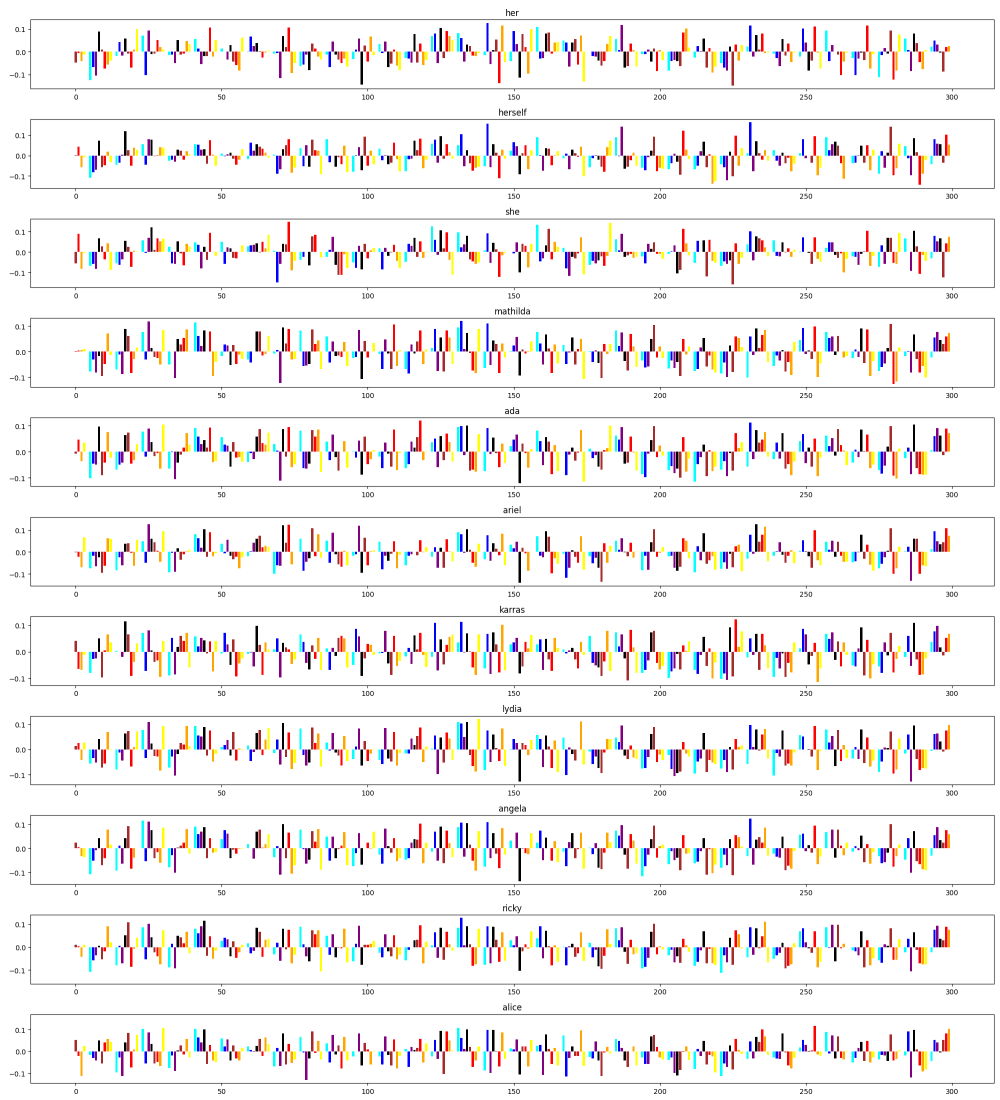
Top 10 similar words: ['reviewer', 'negative', 'description', 'synopsis', 'feedback', 'reveiws', 'editorial', 'title', 'posted', 'criticism']

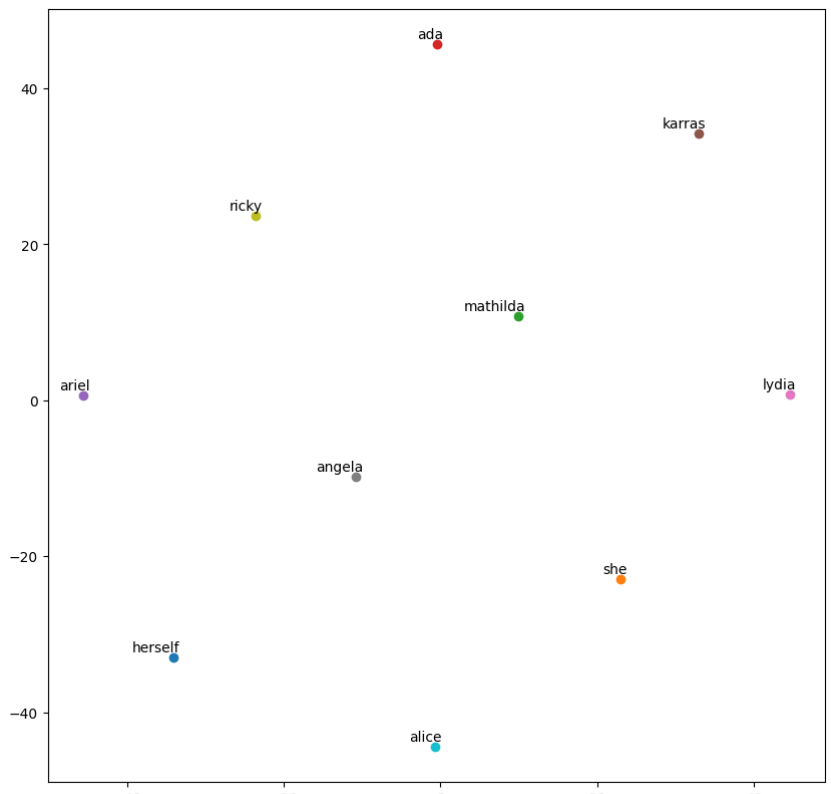




Word: her

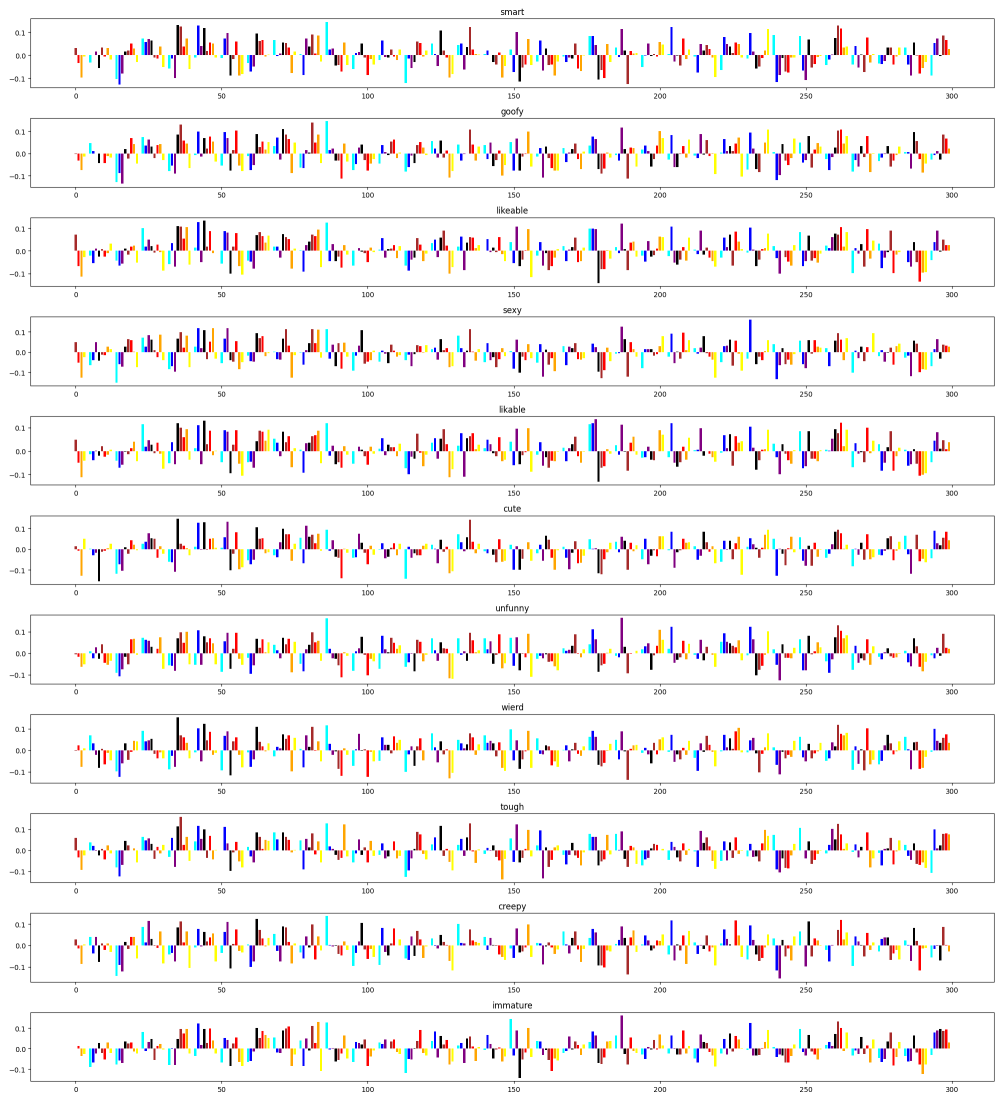
Top 10 similar words: ['herself', 'she', 'mathilda', 'ada', 'ariel', 'karras', 'lydia', 'angela', 'ricky', 'alice']

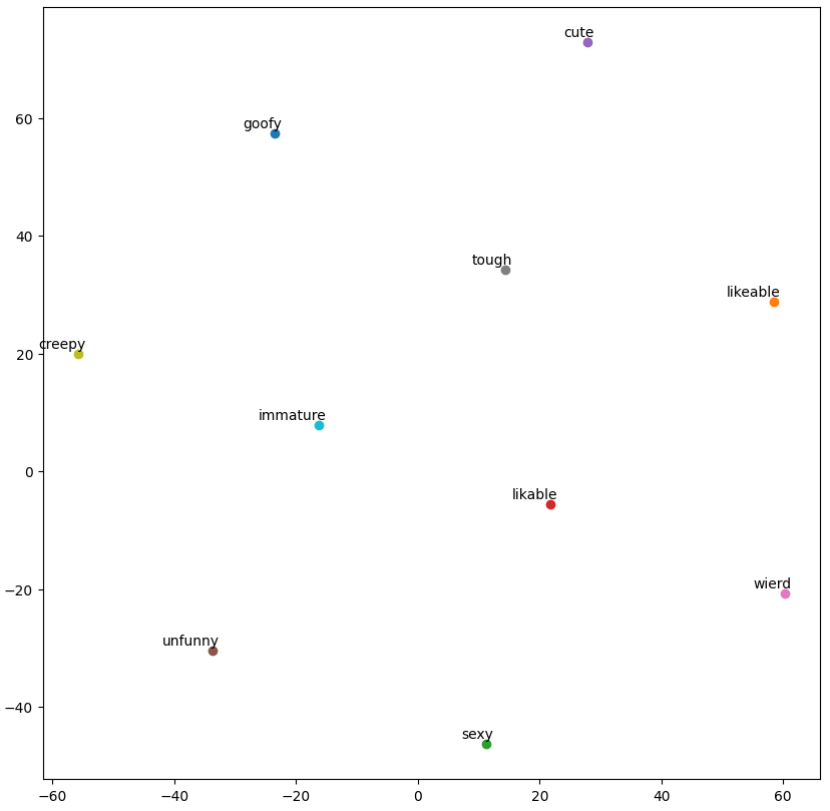




Word: smart

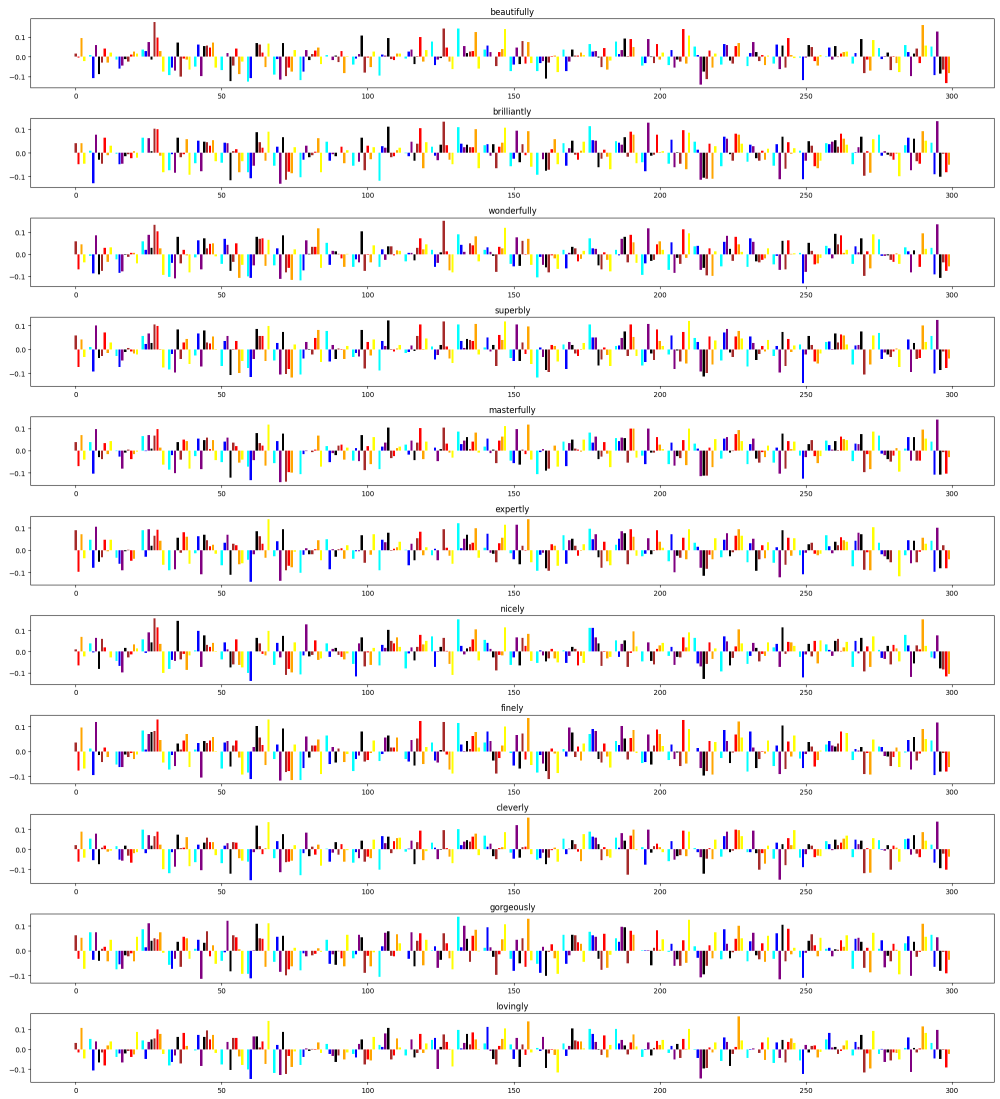
Top 10 similar words: ['goofy', 'likeable', 'sexy', 'likable', 'cute', 'unfunny', 'wierd', 'tough', 'creepy', 'immature']

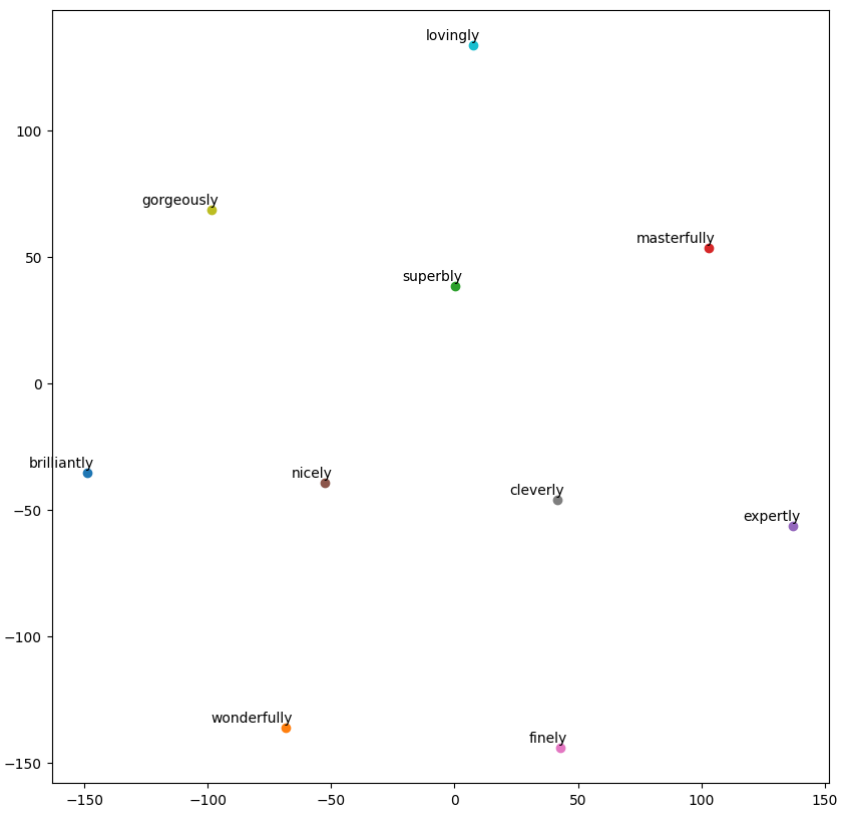




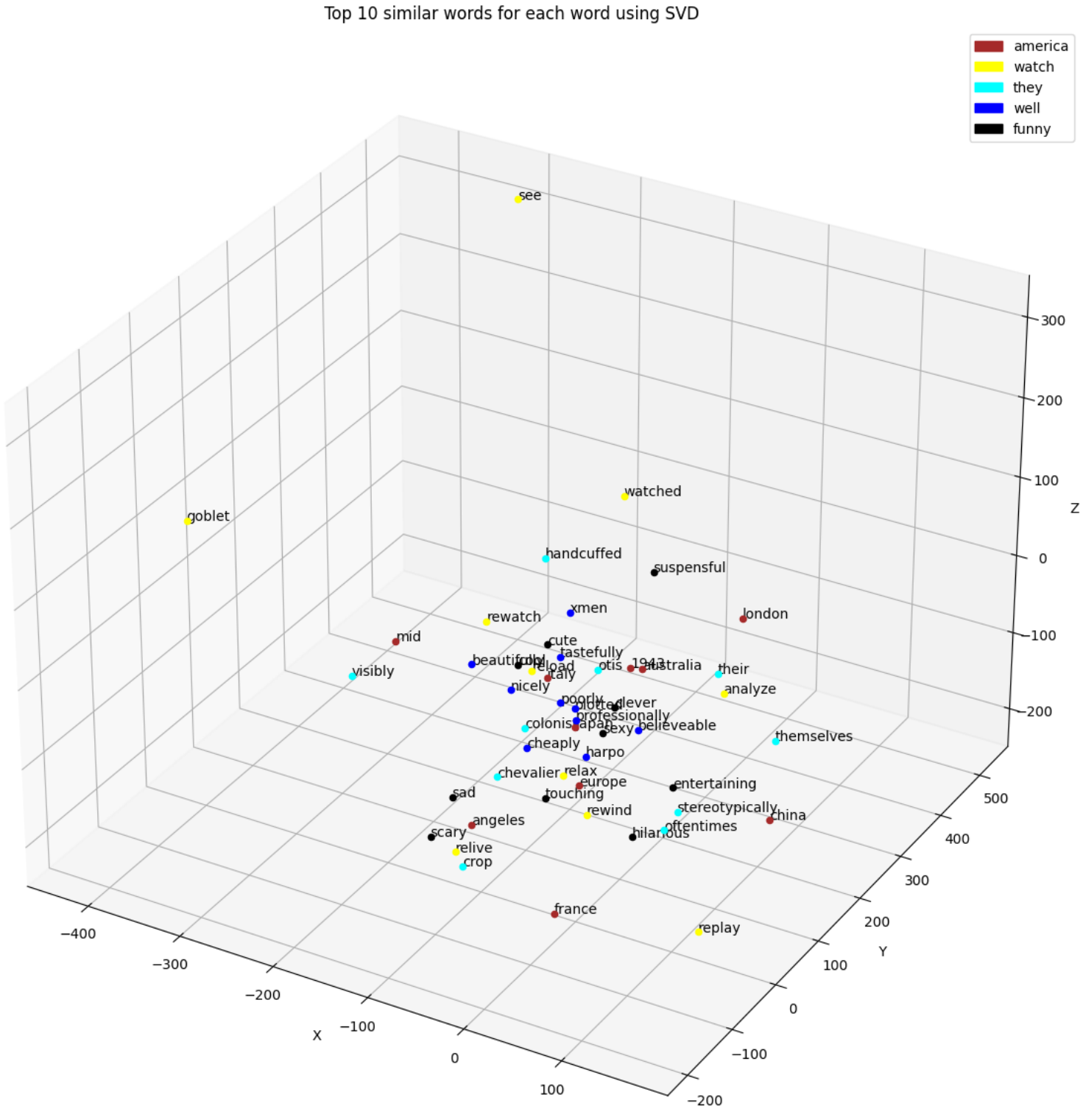
Word: beautifully

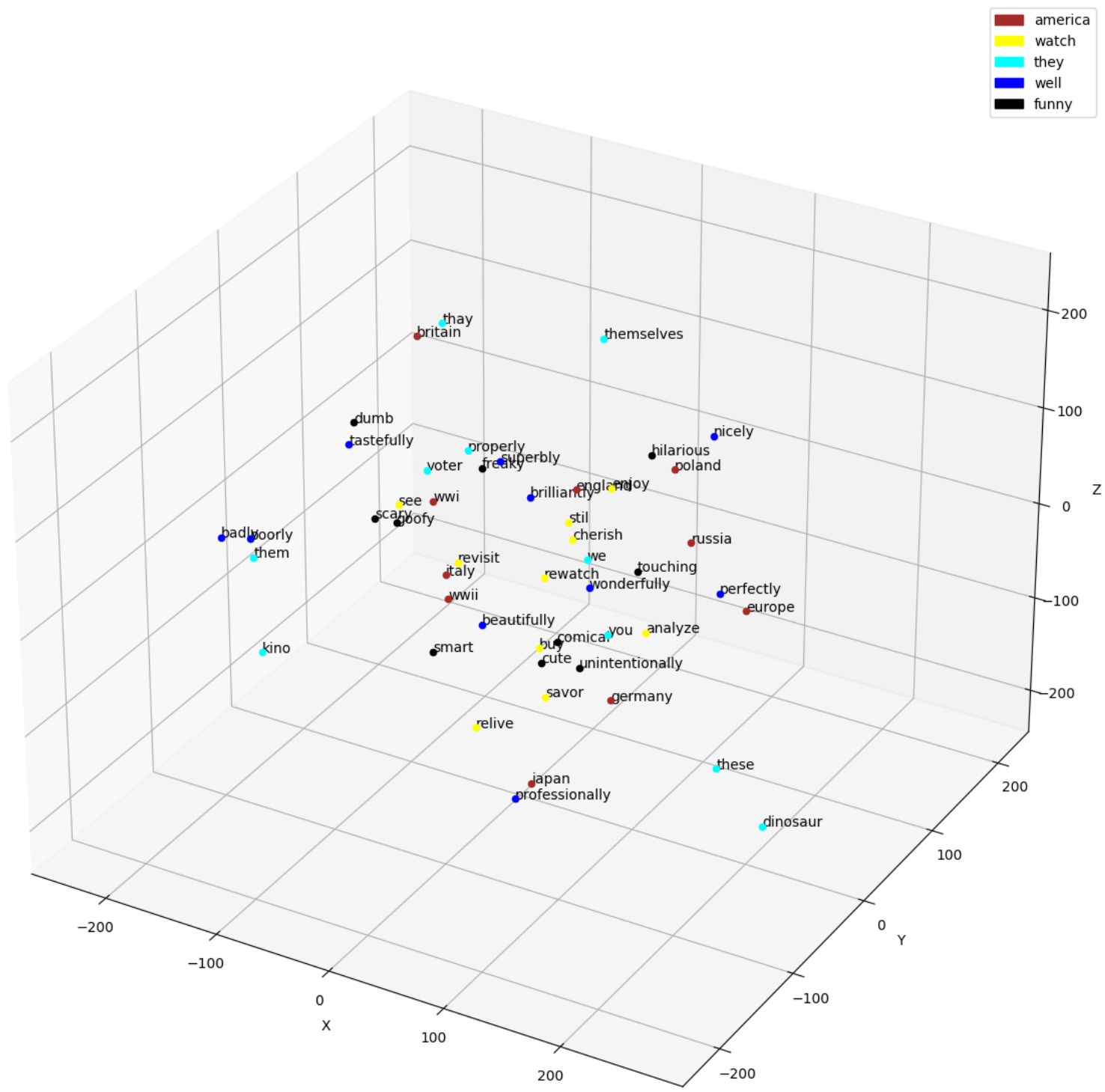
Top 10 similar words: ['brilliantly', 'wonderfully', 'superbly', 'masterfully', 'expertly', 'nicely', 'finely', 'cleverly', 'gorgeously', 'lovingly']





For words (america, watch, they, well, funny)





Word: titanic

Top 10 similar words: ['holocaust', '1928', 'surrounding', 'voyage', 'sinking', 'bombing', 'apollo', 'bulk', 'matrix', 'crux']

Word: titanic

Top 10 similar words: ['lotr', '1960', 'robocop', '74', 'talkie', 'iceberg', 'ninety', 'braveheart', 'phantom', '2000s']

Glove Embedding for word "**titanic**" :
[ship, liner, voyage, ocean, disaster, tragedy, iceberg, vessel, cruise, maritime]

The word **Titanic** is a proper noun, so the results don't appear very satisfying in both embedding but the pretrained Glove embedding is very nice because it is trained on very large corpus like Common Crawl, Wikipedia, Twitter etc.

For the above 5 words, results seem good for both embedding, we can get idea by seeing vectors bar graph , that they look very similar for same word. Few more words in both embedding.

Word	SVD	CBOW
funny	'scary', 'entertaining', 'sad', 'hilarious', 'sexy'	'hilarious', 'cute', 'scary', 'freaky', 'comical'
big	'huge', 'rabid', 'hardcore', 'fan', 'lifelong'	'huge', 'hit', 'smash', 'hockey', 'bang'
lovely	'beautiful', 'delight', 'gorgeous', 'charming', 'delightful'	'gorgeous', 'beautiful', 'classy', 'adorable', 'delightful'
hero	'protagonist', 'vampire', 'gentleman', 'blooded', 'dinosaur'	'villain', 'protagonist', 'psychopath', 'villian', 'vampire'
police	'thief', 'murderer', 'terrorist', 'sheriff', 'miller'	'fbi', 'terrorist', 'cia', 'federal', 'bank'
director	'producer', 'screenwriter', 'gilliam', 'editor', 'terry'	'writer', 'producer', 'ridley', 'screenwriter', 'filmmaker'
robert	'michael', 'john', 'james', 'hoffman', 'downey'	'raul', 'redford', 'englund', 'julia', 'richard'
america	'china', 'europe', 'angeles', 'japan', 'mid'	'japan', 'europe', 'wwii', 'russia', 'wwi'
war	'civil', 'ii', 'ww', 'vietnam', 'post'	'vietnam', 'civil', 'viet', 'wwii', 'submarine'
acting	'directing', 'scenery', 'writing', 'direction', 'music'	'casting', 'directing', 'script', 'timing', 'direction'
support	'protect', 'save', 'source', 'defend', 'homeland'	'guidance', 'resource', 'staff', 'undertake', 'dispense'
watch	'rewatch', 'see', 'watched', 'goblet', 'reload'	'rewatch', 'revisit', 'cherish', 'see', 'analyze'
doing	'exactly', 'happened', 'happening', 'constitutes', 'rebelling'	'do', 'quits', 'quitting', 'done', 'bragging'
basically	'essentially', 'hub', 'stupid', 'supposedly', 'rory'	'essentially', 'basicly', 'afterall', 'terrible', 'horrible'
recently	'previously', 'twice', 'yesterday', 'several', 'videotape'	'previously', 'dec', 'nov', '2011', 'oct'
amazingly	'incredibly', 'intensely', 'intelligent', 'unusually', 'incredible'	'exceptionally', 'unbelievably', 'incredibly', 'extraordinarily', 'stunningly'
well	'professionally', 'nicely', 'poorly', 'tastefully', 'harpo'	'poorly', 'nicely', 'tastefully', 'beautifully', 'badly'
thankfully	'obviously', 'unfortunately', 'terrify', 'goer', 'tomb'	'corrected', 'sadly', 'captioned', 'censored', 'shortened'

Word	SVD	CBOW
them	'merge', 'trapping', 'individuality', 'together', 'disparate'	'eachother', 'themselves', 'ourselves', 'whining', 'yourselves'
mine	'hulot', 'my', 'andromeda', 'compilation', 'lot'	'ours', 'my', 'hers', 'xmas', 'moviegoing'
they	'colonist', 'themselves', 'their', 'oftentimes', 'handcuffed'	'we', 'themselves', 'thay', 'them', 'voter'

For most of the words, CBOW seems to word better than SVD.