

1. About the reuters dataset
2. Generating Word Vectors
3. Reducing the Dimensions
4. Evaluating the Vectors

# Contents

---

# Steps followed

---

- Used Reuters.sent() to access all the sentences available and made a npy file for the same.
- Made a dictionary of words and a separate word2onehot function for on-the go development. Total number of unique words were 41473 (only alpha()).
- Made 3 separate functions for the three methods and used softmax in the final layer for getting the loss (negative log likelihood was used)
- Finally,  $W_1 + W_2T$  was used as the weight and was saved as  $W_{final}$
- $np.dot(W_{final} T, x)$  gave us the final embeddings
- Negative samples used were 8.
- They were further tested using intrinsic methods.



**Implementation  
Type**

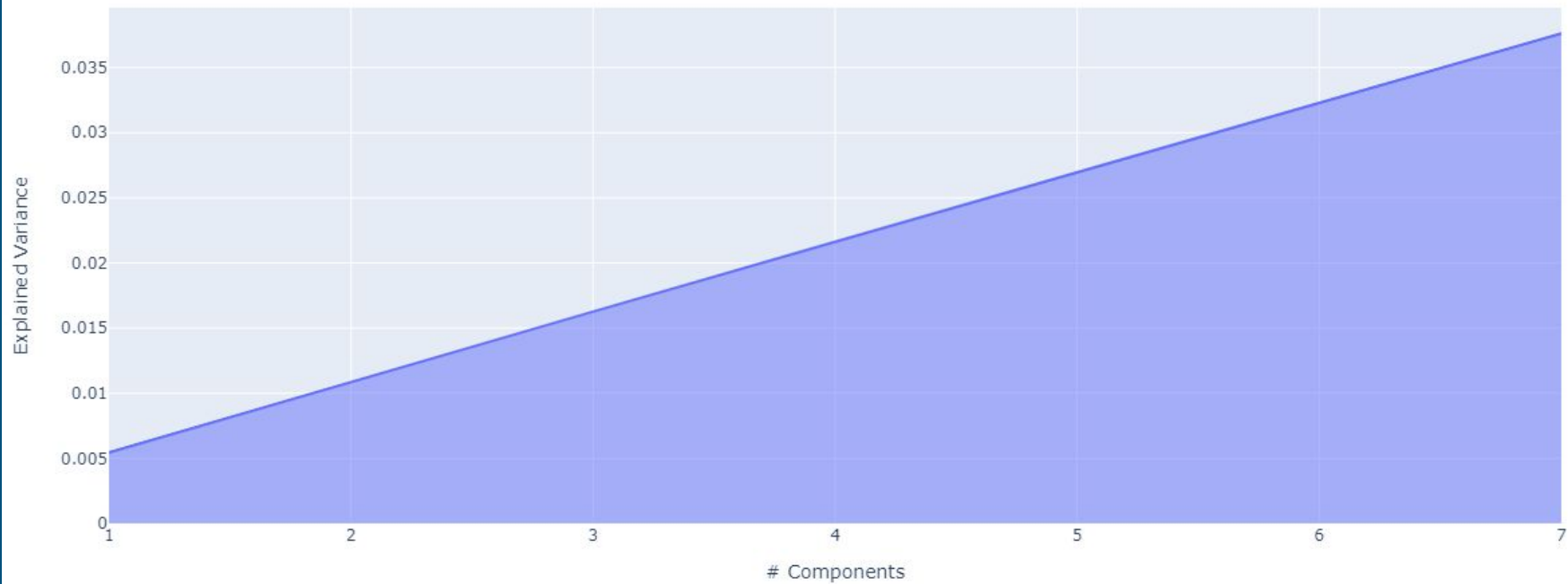
**Vector Size**

**Learning Rate**

**Sub  
sampling**

## Hyperparameter and Model Parameter choice

We experimented with different parameters for implementing Word2vec but kept the window size = 2 as it was the best as per every paper we referred. Also, the number of epochs trained were limited to 5 due to system limitations.



**SKG model with 200 vector size and 0.001 learning rate and SGD optimizer while training. [ The epoch were only 5 because of system limitations ]**

STCS\_Rep x Natural La x k Sentiment x k Movie Rev x NLP Assign x Sentiment x Assignme x STCS\_Eval x 8. Analyzi x SKG - Go x

colab.research.google.com/drive/1\_QDPBecRwHE5tRygpLx5j711vwJD23j#scrollTo=gOvxsB2f1aSc

STCS\_Evaluation.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

0.50780275 0.83385042 0.77020529 0.81612729 0.05163622 0.68151408  
0.96316079 0.70072339]

200  
DAMAGE  
[0.40996837 0.2069304 0.95597798 0.4059029 0.71684473 0.67240392  
0.84211547 0.94077734 0.46984744 0.1076451 0.56665903 0.97973174  
0.43406604 0.81785985 0.43724402 0.81032917 0.2297455 0.4109313  
0.73085766 0.63717368 0.76886728 0.95650223 0.37591255 0.89573794  
0.20832078 0.06339759 0.23935857 0.77984625 0.37588849 0.58137912  
0.24202202 0.43848858 0.18044323 0.37321236 0.86374429 0.85594047  
0.93996623 0.40543475 0.98241958 0.4826681 0.87726923 0.40883536  
0.31978658 0.64264954 0.7092702 0.74252918 0.60437025 0.85842987  
0.64992193 0.27046355 0.8583552 0.10096592 0.53734709 0.38688763  
0.61916967 0.54869609 0.46372197 0.8265784 0.40096345 0.61938811  
0.50090334 0.50056295 0.60888135 0.99512296 0.24794827 0.73843014  
0.9300987 0.58450385 0.88072972 0.69191458 0.35126358 0.82055097  
0.23716008 0.92166065 0.91881464 0.62936622 0.51217348 0.29983643  
0.450972 0.71702321 0.28954418 0.20892858 0.31624178 0.57319822  
0.46274834 0.50259051 0.48704535 0.77923202 0.39039048 0.76861166  
0.86150466 0.15345756 0.61559312 0.51322946 0.98900026 0.08275719  
0.711438 0.27301359 0.99502318 0.35440163 0.08848961 0.98302743  
0.47905676 0.48230632 0.52653692 0.94305611 0.49071806 0.03017977  
0.93521083 0.74708224 0.73045161 0.10529282 0.7656874 0.70791419  
0.92261886 0.5091876 0.71180458 0.87095953 0.43059408 0.48702979  
0.1222203 0.18990588 0.59990827 0.29789882 0.18239519 0.5027265  
0.95068805 0.23247187 0.7391227 0.76159769 0.76617354 0.92441826  
0.32262474 0.6786955 0.17178137 0.70924714 0.48929811 0.44175418  
0.64680037 0.47824531 0.78098752 0.56378253 0.84929462 0.33241075  
0.68767068 0.31038059 0.66052447 0.35452749 0.06117003 0.44393811  
0.70826965 0.25734276 0.61806642 0.12140348 0.63727434 0.05823178  
0.04763533 0.51201684 0.29912273 0.34642094 0.64949398 0.9487233  
0.51637285 0.7135187 0.58583289 0.39498699 0.61760792 0.23755686  
0.84830023 0.29247091 0.12061195 0.43121329 0.27204644 0.27358139  
0.64228414 0.44996377 0.9082571 0.77538028 0.33927501 0.70363672  
0.2390847 0.20502493 0.6839743 0.41690059 0.19516493 0.39019889  
0.09834 0.32263477 0.88909352 0.94212907 0.65745159 0.19587051  
0.52575185 0.31069609 0.5551385 0.53534816 0.46486133 0.76767917  
0.88673134 0.82961398]

200  
FROM  
[9.58603703e-01 9.10472188e-01 1.19552113e-01 1.14179978e-01  
9.96230010e-01 3.95923827e-02 8.60081978e-01 4.65288261e-01  
2.88474080e-01 7.32240272e-01 4.71347203e-01 3.65877249e-01

```
[7] X = pca.transform(X)

print(pca.explained_variance_ratio_)


[0.00545335 0.00541809]
```

As we can see the variance is very very less

```
for num, words in enumerate(embeddings):
    embeddings[words] = X[num]

print(embeddings)
```

5304051, -0.56988405]], 'FEAR': array([0.02844994, 0.24416726]), 'DAMAGE': array([0.5346529 , 0.03957628]), 'FROM': array([-0.37487875, -0.33895114]), 'U': array(



After applying PCA, the dimension got reduced to just 2

# Accuracy and Loss for the vectors

1. Cosine Similarity represents the partial accuracy.
  2. The euclidean distance between similar words represent euclidean loss.
  3.  $100 * (1 - \text{loss} + \text{acc}) / 2$  % was the final accuracy.
-

## An example of the technique used:

```
word1 = "provinces"
word2 = "cities"

a = embeddings[word1]
b = embeddings[word2]

dist = np.linalg.norm(a - b)

from numpy import dot
from numpy.linalg import norm

cos_sim = dot(a, b)/(norm(a)*norm(b))
```

```
[91] acc = cos_sim*100
      loss = abs(dist)

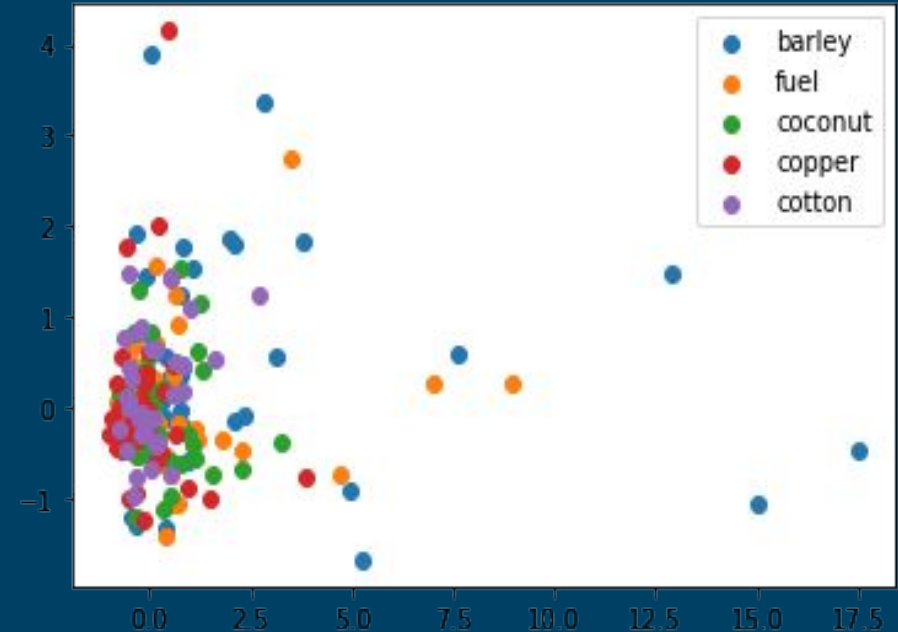
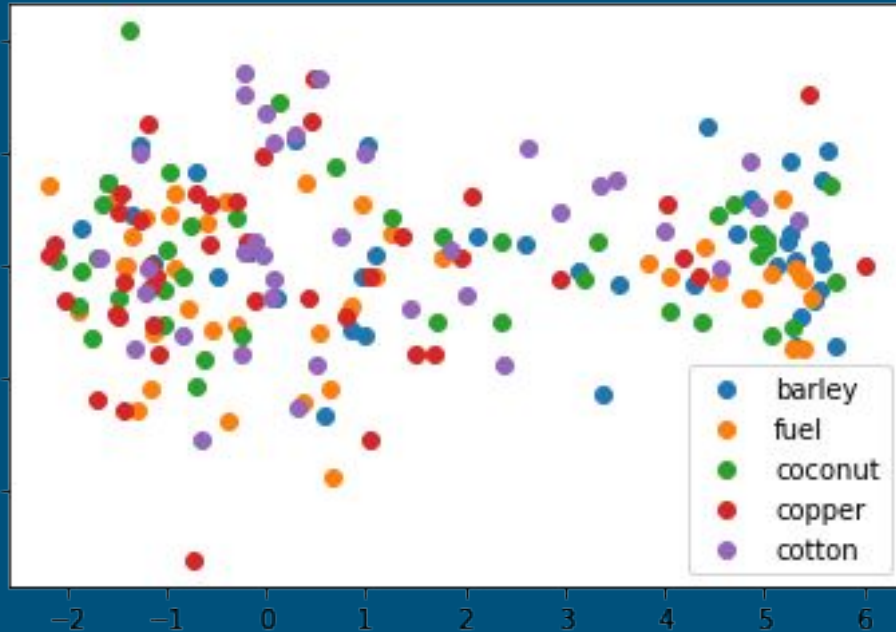
      print("The accuracy is : ", acc , "\nThe Loss is : ", loss)
```

```
The accuracy is : 99.65001633594487
The Loss is : 0.08066750125373907
```

dist( Coffee - cocoa ) - dist( City - province ) is the loss

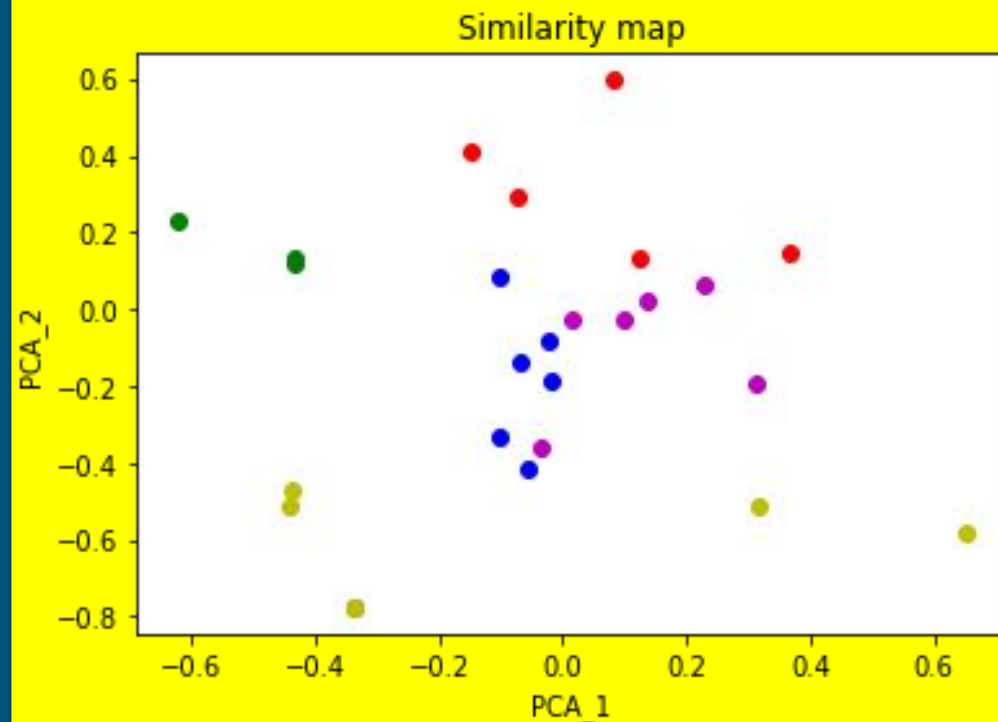
Cosine similarity between city and province is shown.





The dataset contained stop words and other words that hindered with accurate training. The data was not pre-processed properly.

There were words which were present in more than one category, and this effect was not accounted for.



y - hog, g - castor-oil, b - income, p - wheat, r - cocoa

Here, the graph is for the previously shown set parameters.

```
# We choose 5 categories randomly
import random
random.seed(6969)

cats = np.array(reuters.categories())

l = []
for i in range(int(len(reuters.categories()))):
    l.append(i)

cho = random.choices(l,k=5)
print(cho)

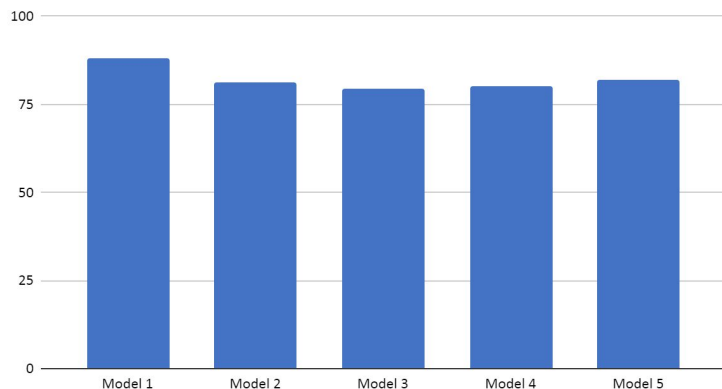
for num,word in enumerate(cho):
    cho[num] = str(cats[word])
del l
print(cho)
```

```
[6, 86, 32, 5, 30]
['cocoa', 'wheat', 'income', 'castor-oil', 'hog']
```

**df.head()** for all the models we tried

Sr.No	Window Size	Choice of Model	Vector Size	Learning Rate	Accuracy
1	2	SKG	300	0.001	87.9
2	2	SKG	200	0.001	81.3
3	2	NSSKG	200	0.001	82
4	2	CBOW	200	0.001	79.5
5	2	CBOW	200	0.5	80.2

Accuracy Scores from df.head()



### General trend

- NSSKG > SKG > CBOW for 300+ vector size otherwise SKG>CBOW>NSSKG.
- Learning Rate inversely proportional to accuracy.
- Best embedding size is 300.