Introduction

The PPMI (Positive Point wise Mutual Information) vector model is a type of distributional semantic model used in natural language processing and computational linguistics. It is designed to capture the meaning of words based on their co-occurrence with other words in a large corpus of text.

PPMI is a statistical measure that reflects the strength of association between two words, based on the frequency of their co-occurrence in a corpus, relative to their individual frequency of occurrence. It is defined as the logarithm of the ratio of the observed frequency of co-occurrence to the expected frequency of co-occurrence if the two words were independent.

The PPMI vector model represents words as high-dimensional vectors, where each dimension corresponds to a context word that co-occurs with the target word in the corpus. The value of each dimension is the PPMI score between the target word and the context word. The resulting vectors capture the semantic similarity between words based on their co-occurrence patterns in the corpus.

One of the advantages of the PPMI vector model is its ability to capture both syntactic and semantic information, which makes it useful for a wide range of natural language processing tasks, such as word similarity, semantic role labeling, and sentiment analysis.

The PPMI (Positive Pointwise Mutual Information) model is used in natural language processing and computational linguistics for several reasons:

Capturing Semantic Similarity: The PPMI model aims to capture the semantic similarity between words based on their co-occurrence patterns in a corpus. By measuring the strength of association between words, it provides a way to represent the meaning of words in a high-dimensional vector space. This representation enables various downstream tasks such as word similarity, clustering, and semantic analysis.

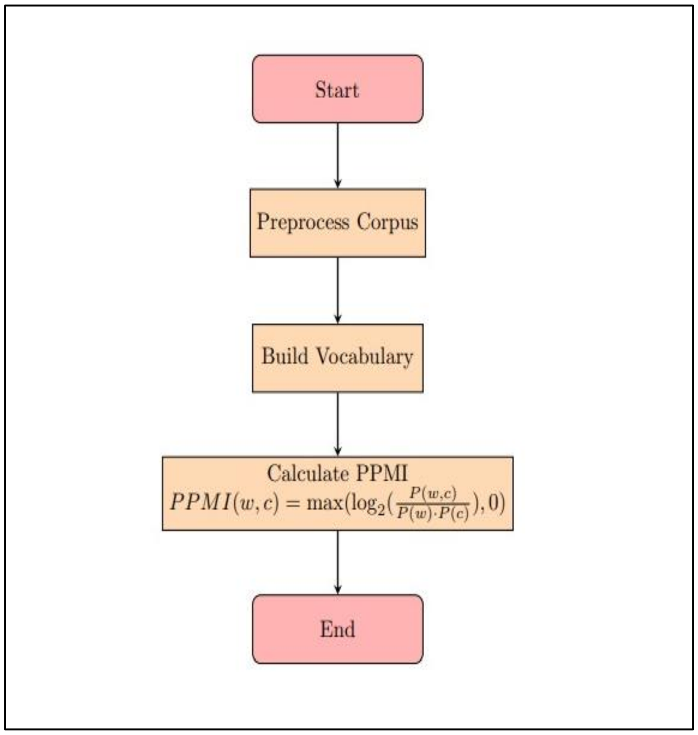
Contextual Information: The PPMI model considers the context in which words appear, taking into account the surrounding words and their frequencies. By incorporating contextual information, it can capture both syntactic and semantic relationships between words. This makes it valuable for tasks that rely on understanding the meaning of words in different contexts.

Dimensionality Reduction: The PPMI model reduces the high-dimensional co-occurrence matrix into lower-dimensional vectors that capture the essential semantic information. This dimensionality reduction helps in managing the computational complexity of working with large vocabularies and enables efficient processing and analysis of word representations.

Mitigating Data Sparsity: Language data is often sparse, meaning that not all word pairs co-occur frequently. The PPMI model addresses this issue by discounting infrequent co-occurrences, which helps in focusing on the more informative and meaningful associations between words. This way, it can handle the sparsity problem and provide more robust word representations.

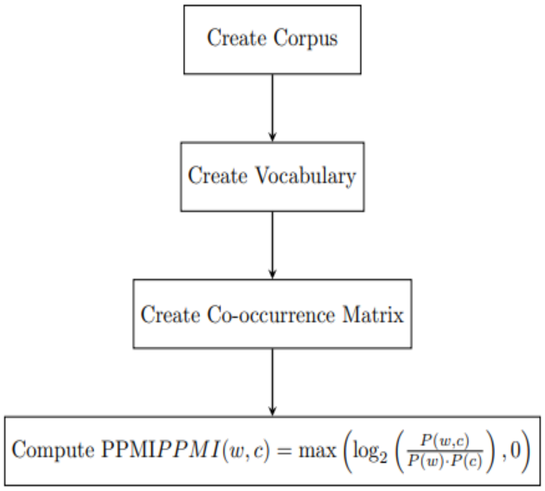
Performance on NLP Tasks: The PPMI model has demonstrated good performance on various natural language processing tasks. It has been used successfully in tasks such as word similarity, word analogy, sentiment analysis, text classification, and machine translation.

Flow Diagram:



Start
Preprocess Corpus
Build Vocabulary
Calculate PPMI
$PPMI(w,c) = \max(\log_2(\frac{P(w,c)}{P(w)\cdot P(c)}), 0)$
End

Block diagram:



Create Corpus
Create Vocabulary
Create Co-occurrence Matrix
Compute PPMI$PPMI(w,c) = \max\left(\log_2\left(\frac{P(w,c)}{P(w)\cdot P(c)}\right), 0\right)$

Algorithm

1. First, the necessary imports are made, including numpy for numerical computations and CountVectorizer from scikit-learn for text processing.

2. A corpus of sentences is defined. This corpus represents the collection of text used to compute the co-occurrence matrix and PPMI matrix.

3. A vocabulary set is created by iterating over each sentence in the corpus, splitting it into individual words, and adding them to the set. This vocabulary set represents all unique words in the corpus.

4. An instance of CountVectorizer is initialized with the defined vocabulary and a tokenizer function that splits each sentence into words.

5. The fit_transform method is called on the vectorizer object, passing the corpus. This step converts the corpus into a co-occurrence matrix, where each row represents a word from the vocabulary, and each column represents a context word from the vocabulary. The values in the matrix represent the count of co-occurrences between the word and the context word.

6. The co-occurrence matrix is converted to a dense array using the toarray method.

7. The compute_ppmi function is defined to compute the PPMI matrix given a co-occurrence matrix.

8. Inside the compute_ppmi function, the sum of all elements in the co-occurrence matrix is computed, as well as the sums of elements in each row and column.

9. A matrix of zeros, ppmi_matrix, is initialized to store the PPMI values.

10. Two nested loops iterate over each element in the co-occurrence matrix, and for each element, the observed probability p_pair of the word pair is computed.

11. The expected probabilities p_i and p_j of the individual words assuming independence are computed based on the sums over rows and columns.

12. The PPMI value is calculated using the formula ppmi = max(0, np.log2(p_pair / p_ind)), where p_ind represents the expected probability of the word pair assuming independence.

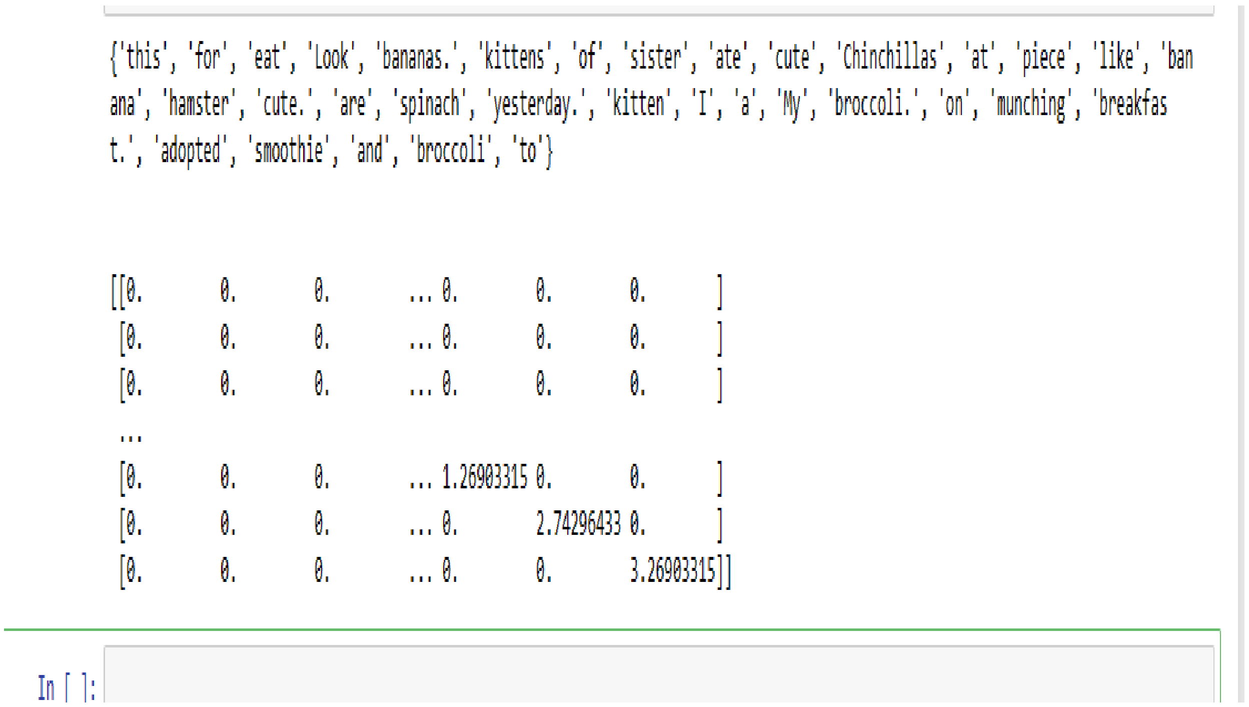
13. The PPMI value is stored in the ppmi_matrix.

14. Finally, the ppmi_matrix is printed.

Python Code:

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer

# Define a corpus
corpus = [
    "I like to eat broccoli and bananas.",
    "I ate a banana and spinach smoothie for breakfast.",
    "Chinchillas and kittens are cute.",
    "My sister adopted a kitten yesterday.",
    "Look at this cute hamster munching on a piece of broccoli."
]

# Define a vocabulary
vocabulary = set([word for sentence in corpus for word in sentence.split()])
print(vocabulary, "\n\n")

# Define a CountVectorizer to convert the corpus to a co-occurrence matrix
vectorizer = CountVectorizer(vocabulary=vocabulary, tokenizer=lambda x: x.split())
X = vectorizer.fit_transform(corpus)

co_occurrence_matrix = (X.T * X).toarray()

# Define a function to compute the PPMI matrix
def compute_ppmi(co_occurrence_matrix):
    # Compute the sum of all elements in the matrix
    sum_over_matrix = np.sum(co_occurrence_matrix)

    # Compute the sum of elements in each row and column
    sum_over_rows = np.sum(co_occurrence_matrix, axis=1)
    sum_over_columns = np.sum(co_occurrence_matrix, axis=0)

    # Compute the total number of rows and columns
    num_rows, num_columns = co_occurrence_matrix.shape

    # Initialize a matrix to store the PPMI values
    ppmi_matrix = np.zeros((num_rows, num_columns))

    # Loop over all elements in the matrix and compute the PPMI value
    for i in range(num_rows):
        for j in range(num_columns):
            # Compute the observed probability of the word pair
            p_pair = co_occurrence_matrix[i, j] / sum_over_matrix

            # Compute the expected probability of the word pair assuming independence
            p_i = sum_over_rows[i] / sum_over_matrix
            p_j = sum_over_columns[j] / sum_over_matrix
            p_ind = p_i * p_j

            # Compute the PPMI value, handling zero values and division by zero
            if p_pair == 0 or p_ind == 0:
                ppmi = 0
            else:
                ppmi = max(0, np.log2(p_pair / p_ind))

            ppmi_matrix[i, j] = ppmi
    return ppmi_matrix

# Compute the PPMI matrix
with np.errstate(divide='ignore', invalid='ignore'):
    ppmi_matrix = compute_ppmi(co_occurrence_matrix)

# Print the PPMI matrix
print(ppmi_matrix)
```

Output:



```
{'this', 'for', 'eat', 'Look', 'bananas.', 'kittens', 'of', 'sister', 'ate', 'cute', 'Chinchillas', 'at', 'piece', 'like', 'ban
ana', 'hamster', 'cute.', 'are', 'spinach', 'yesterday.', 'kitten', 'I', 'a', 'My', 'broccoli.', 'on', 'munching', 'breakfas
t.', 'adopted', 'smoothie', 'and', 'broccoli', 'to'}



[[0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 ...
 [0.         0.         0.         ... 1.26903315 0.         0.        ]
 [0.         0.         0.         ... 0.         2.74296433 0.        ]
 [0.         0.         0.         ... 0.         0.         3.26903315]]
```

```
In [ ]:
```