I. Description of 100-dimensional embedding.

1. Data preparation

   - Downloaded Brown corpus from NLTK and called brown.words() return the text in one long list, then made each word lower case.
   - Removed stopwords(using nltk.corpus.stopwords('english')), punctuations (using string.punctuation) and filtered the words which is not alphanumeric, such as " - -", "i'm" and "he's". (only remaining word.isalnum()).
   - Get a list of 515882 words from corpus after cleaning.

2. Commonly-ocuuring words

   - Used collection package Counter function counting the frequencies of words in words list, and applied most_common() function to get the words with higher count.
   - Select the top 5000 frequen words as vocabulary `V` , the top 1000 frequent words as context words `C`, and saved the words and counts as word-count dictionary.
     `V_counts_dict`: {w: count ...}
     `C_counts_dict`: {c: count ...}

3. Probabilities

   - Computed overall distrubution $Pr(c)$ of context words, and saved as `pr_c_dict` of length of 1000 of context-word-probability dictionary.

     $Pr(c) = \frac{n(c)}{n(total\_c)}$; `# pr_c_dict`: {c: probability ... ...}
   - Defined a window function to get a surrounding window of four words' indicies.

     $Window\ index$ = [index-2, index-1, index+1, index+2]
   - Created a nested dictionary called `CW_counts_dict` with value of $n(w, c)$, which can converted to 5000×1000 matrix.

CW_counts_dict:$\{w : \{c : count\ ...\ ...\}\ ...\ ...\}$

Steps:

For each word in cleaned words if the word in vocabulary list(V), check the surrounding window of four words, then for each surrounding words, if the word in context words list(C), add 1 to the correspoding CW_counts_dict by key as $[w][c]$.

- Devided CW_counts_dict$[w][c]$ values by V_counts_dict$[w]$ value to get $Pr(c|w)$, saved as a nested dictionary called `pr_cw_dict`.

  $Pr(c|w) = \frac{n(w,c)}{n(w)}$;   `# pr_cw_dict`: $\{w : \{c : probability\ ...\ ...\}\ ...\ ...\}$

4. $\phi(w)$

   Represented each vocabulary item by a $|C|$-dimensional vector $\phi(w)$, that each row represents each vocabulary w by 1000 dimensional vector. Define a function to get $\phi(w)$. $\phi(w) = max(0,\ log\frac{Pr(c|w)}{Pr(c)})$

   Steps:

   - Calculated $\frac{Pr(c|w)}{Pr(c)}$, $phi[w][c] = pr\_cw\_dict[w][c]/pr\_c\_dict[c]$
   - If $\frac{Pr(c|w)}{Pr(c)}$ >1, chose $log\frac{Pr(c|w)}{Pr(c)}$, else chose 0, then saved as another nested dictionary(`phi`). `# phi`:{w: {c: $\phi(w)$ ... } ...}
   - Converted the nested dictionary of $\phi(w)$ to a dataframe.

     ```
     phi_df = pd.DataFrame(phi).T
     ```

5. 100 dimensional representation

   Using sklearn.decomposition.PCA to do PCA on $\phi(w)$ transformed 1000-dimension to 100-dimension, saved as a dataframe called `phi_transformed_df`. $\Psi(w) \in R^{100}$

   ```
   pca = PCA(n_components=100)
   phi_transformed = pca.fit_transform(phi_df)
   phi_transformed_d = pd.DataFrame(phi_transformed,index=V)
   ```

II. Nearest Neighbor results.

1. **Cosine_distance**:

   Defined a function of find nearest neighbor. Using sklearn.metrics.pairwise_distances( ) calculated the distance between given vocabulary(w) and each word ($w' \neq w$) in cleaned text, and saved w'-distance pair dictionary, then from the dictionary get the key of min value. NearestNeighbor = min(dict, key=dict.get)

   - Cosine distance: (set matric ='cosein')

     Nearest neighbor: communism – utopian
     Nearest neighbor: autumn – slide
     Nearest neighbor: cigarette – smelled
     Nearest neighbor: pulmonary – artery
     Nearest neighbor: mankind – civil
     Nearest neighbor: africa – asia
     Nearest neighbor: chicago – portland
     Nearest neighbor: revolution – exercise
     Nearest neighbor: september – december
     Nearest neighbor: chemical – milligrams
     Nearest neighbor: detergent – cleaning
     Nearest neighbor: dictionary – text
     Nearest neighbor: storm – autumn
     Nearest neighbor: worship – religion
     Nearest neighbor: school – college
     Nearest neighbor: weather – hot
     Nearest neighbor: tax – income
     Nearest neighbor: art – literature
     Nearest neighbor: data – results
     Nearest neighbor: america – states
     Nearest neighbor: earth – dust
     Nearest neighbor: happy – know
     Nearest neighbor: female – feature
     Nearest neighbor: color – dress
     Nearest neighbor: flight – landing

- Euclidean distance: (set `matric ='euclidean'`)

  Nearest neighbor: communism – utopian
  Nearest neighbor: autumn – slide
  Nearest neighbor: cigarette – smelled
  Nearest neighbor: pulmonary – artery
  Nearest neighbor: mankind – fatal
  Nearest neighbor: africa – asia
  Nearest neighbor: chicago – portland
  Nearest neighbor: revolution – twentieth
  Nearest neighbor: september – december
  Nearest neighbor: chemical – milligrams
  Nearest neighbor: detergent – fabrics
  Nearest neighbor: dictionary – text
  Nearest neighbor: storm – autumn
  Nearest neighbor: worship – voluntary
  Nearest neighbor: school – college
  Nearest neighbor: weather – cheap
  Nearest neighbor: tax – income
  Nearest neighbor: art – english
  Nearest neighbor: data – include
  Nearest neighbor: america – peoples
  Nearest neighbor: earth – dust
  Nearest neighbor: happy – wondered
  Nearest neighbor: female – nude
  Nearest neighbor: color – pure
  Nearest neighbor: flight – landing

As we can see from the results, the nearest neighbor words are have high similarity, such as 'school' – 'college', 'september' – 'december', 'tax' – 'income', and 'chicago – portland', and there's no much difference between the two distance measurments.

III. Clustering

Tried to use two different algorithsm, **KMeans** and **Agglomerative** clustering, and using **Silhouette Coefficient** to compared different model that if the clusters dense and well seperated.

- KMeans: sklearn.cluster.KMeans Separate samples in n groups of equal variance, minimizing within-cluster sum-of-squares, using euclidean distance.
  `k_means = KMeans(n_clusters=100,init='k-means++')`
  `silhouette_score`= -0.017
- Hierarchical clustering: sklearn.cluster.AgglomerativeClustering Set affinity as 'cosine distance' and 'euclidean distance' with four different linkage. It supports Ward, single, average, and complete linkage strategies.
  - **ward** minimizes the variance of the clusters being merged. Only for euclidean distance.
  - **average** uses the average of the distances of each observation of the two sets.
  - **complete** or maximum linkage uses the maximum distances between all observations of the two sets.
  - **single** uses the minimum of the distances between all observations of the two sets.

Fitted different clustering model on 100-dimensional transformed $\phi(w)$ data, and checked each model silhouette coefficient, then checked each cluster word count with high silhouette_score. After compareration, I found that even though some of the Agglomerative clustering model like `(affinity='euclidean',linkage='complete')` have high silhouette score, most of the clusers have little words, which caused by overfitting.

Since the AgglomerativeClustering(affinity='euclidean',linkage='ward') is same as the KMeans algorithm, and KMeans silhouette_score is higher than AgglomerativeClustering model, I selected KMeans algorithm and euclidean distance for the clustering. And the clusters seems coherent.

A few of the meaningful clusters:

- **group_12** : ['school', 'children', 'members', 'college', 'university', 'students', 'class', 'schools', 'student', 'activities', 'negro', 'interested', 'professional', 'parents', 'active', 'jewish', 'teachers', 'teaching']

- **group_55** : ['cost', 'total', 'rate', 'tax', 'increase', 'costs', 'pay', 'amount', 'higher', 'increased', 'due', '1961', 'sales', 'average', 'lower', 'additional', 'fiscal', 'income', 'rates', 'annual', 'operating', 'gain', 'estimated', 'wages']
- **group_64**: ['white', 'black', 'red', 'dark', 'brown', 'hair', 'blue', 'color', 'beautiful', 'arms', 'blood', 'green', 'sun', 'heavy', 'deep', 'mouth', 'teeth', 'thin', 'bright', 'rose', 'neck', 'watched', 'gray', 'walls', 'dress', 'thick', 'wore', 'nose', 'stared', 'sky', 'pale', 'tall', 'yellow', 'tiny', 'pink', 'skin', 'golden', 'shade', 'beard', 'blonde']
- **group_80**: ['war', 'south', 'west', 'north', 'peace', 'america', 'east', 'europe', 'british', 'germany', 'france', 'berlin', 'russia', 'china', 'britain', 'africa', 'asia', 'atlantic', 'eastern']
- **group_93** : ['day', 'last', 'year', 'later', 'next', 'days', 'early', 'week', 'morning', 'months', 'return', 'late', 'hours', 'weeks', 'nearly', 'summer', 'month', 'spring', 'spent']
- **group_99** : ['years', 'three', 'four', 'five', 'ago', 'six', 'minutes', 'miles', 'hundred', 'ten', 'seven', 'eight', 'dollars', 'thousand', 'nine', 'twenty', 'fifty', 'thirty', 'fifteen', 'twelve', 'eleven']