AI_534
MACHINE LEARNING

IA4 Report

CHI-CHIEH WENG
TSU-CHING LIN

Explore word embeddings

Build your own data set of words

29 most similar words:

flight:
'plane', 'flights', 'boarding', 'airline', 'jet', 'flying', 'heading', 'arrival', 'airlines', 'travel', 'shuttle', 'delayed', 'landing', 'route', 'airplane', 'safe', 'booking', 'fly', 'departure', 'waiting', 'landed', 'journey', 'passengers', 'transit', 'delay', 'crew', 'pilot', 'trip', 'taxi'

good:
'great', 'well', 'nice', 'better', 'night', 'bad', 'morning', 'way', 'hope', 'but', 'too', 'really', 'right', 'though', 'there', 'day', 'luck', 'sure', 'it', 'thing', 'pretty', 'think', 'have', 'all', 'yes', 'very', 'again', 'work', 'yeah'

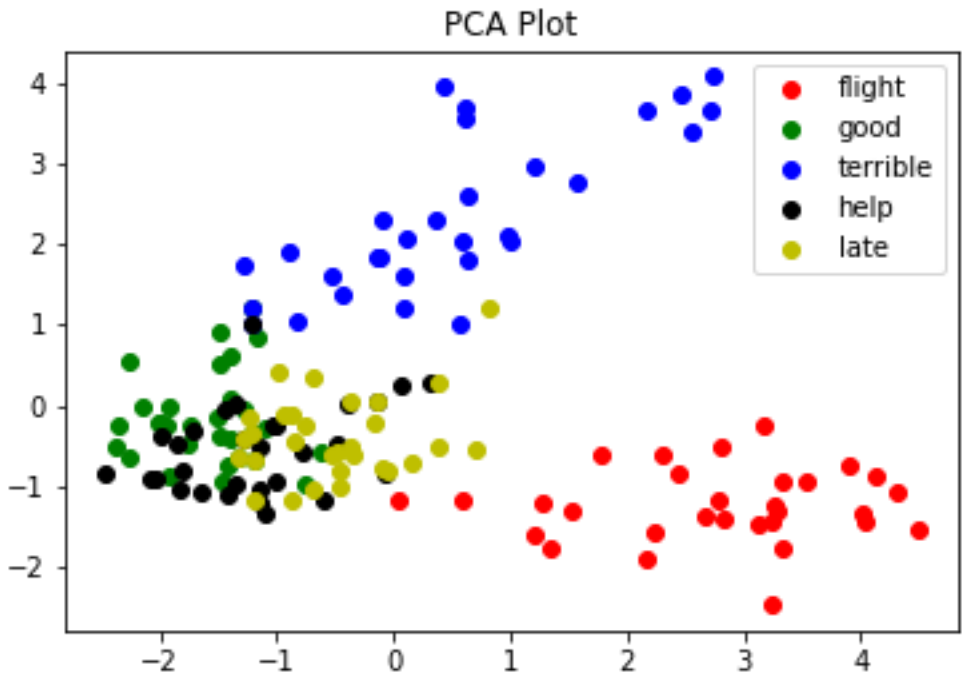
terrible:
'horrible', 'awful', 'bad', 'brutal', 'idea', 'horrendous', 'horrid', 'shitty', 'quite', 'worst', 'similar', 'shame', 'worse', 'crap', 'actual', 'horrific', 'bloody', 'ridiculous', 'such', 'atrocious', 'dreadful', 'sick', 'wtf', 'fucking', 'cruel', 'seriously', 'unreal', 'mess', 'however'

help:
'need', 'helping', 'please', 'pls', 'let', 'us', 'give', 'trying', 'can', 'helps', 'must', 'tell', 'find', 'could', 'plz', 'helped', 'support', 'anyone', 'should', 'save', 'take', 'want', 'bring', 'maybe', 'lets', 'seriously', 'able', 'here', 'needs'

late:
'early', 'earlier', 'usual', 'after', 'again', 'saturday', 'afternoon', 'hour', 'guess', 'missed', 'work', 'hours', 'sunday', 'since', 'night', 'anyway', 'yesterday', 'last', 'maybe', 'yet', 'monday', 'wait', 'either', 'mins', 'wake', 'before', 'thursday', 'hopefully', 'friday'

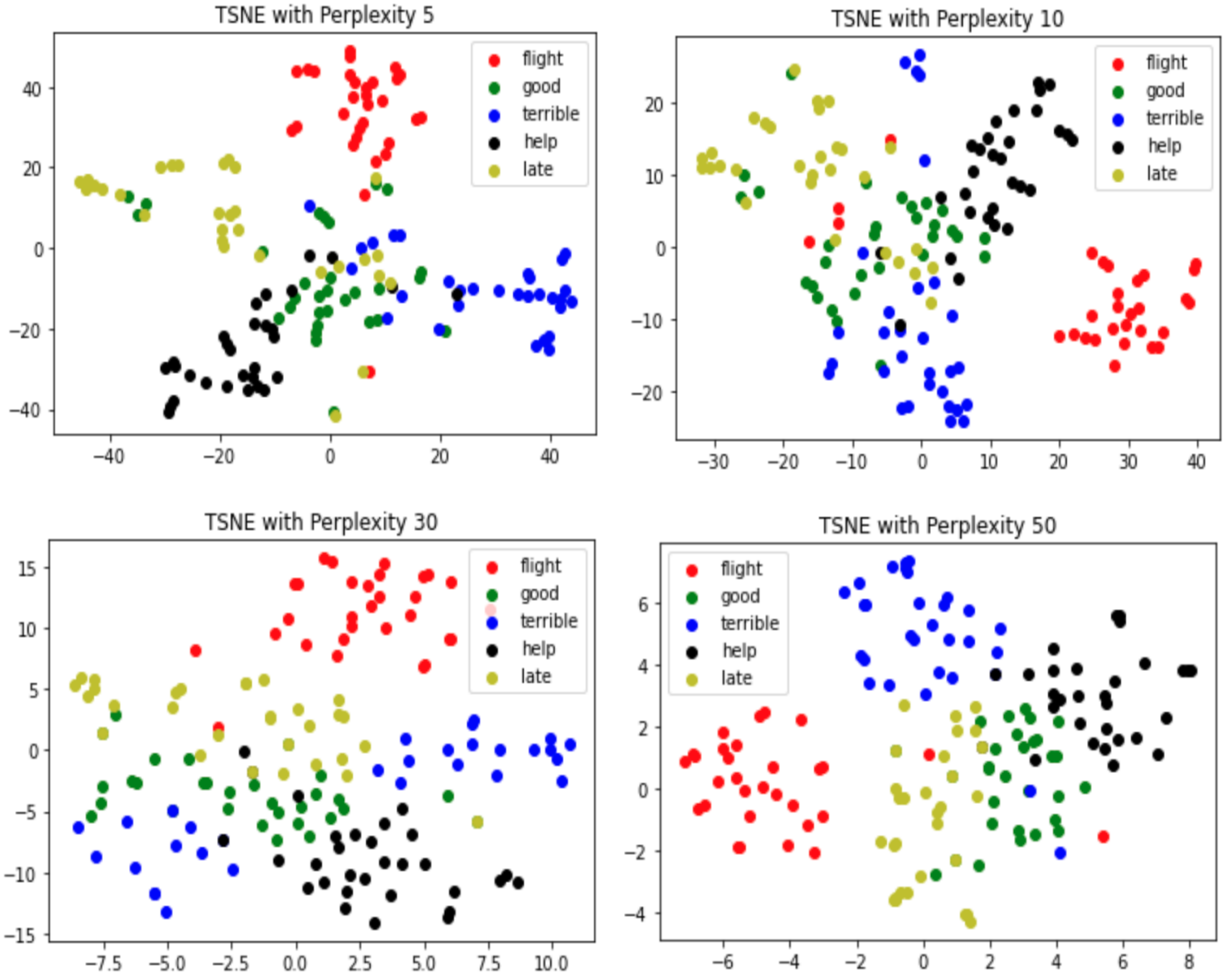
Dimension reduction and visualization

1. PCA plot for 2-d space:



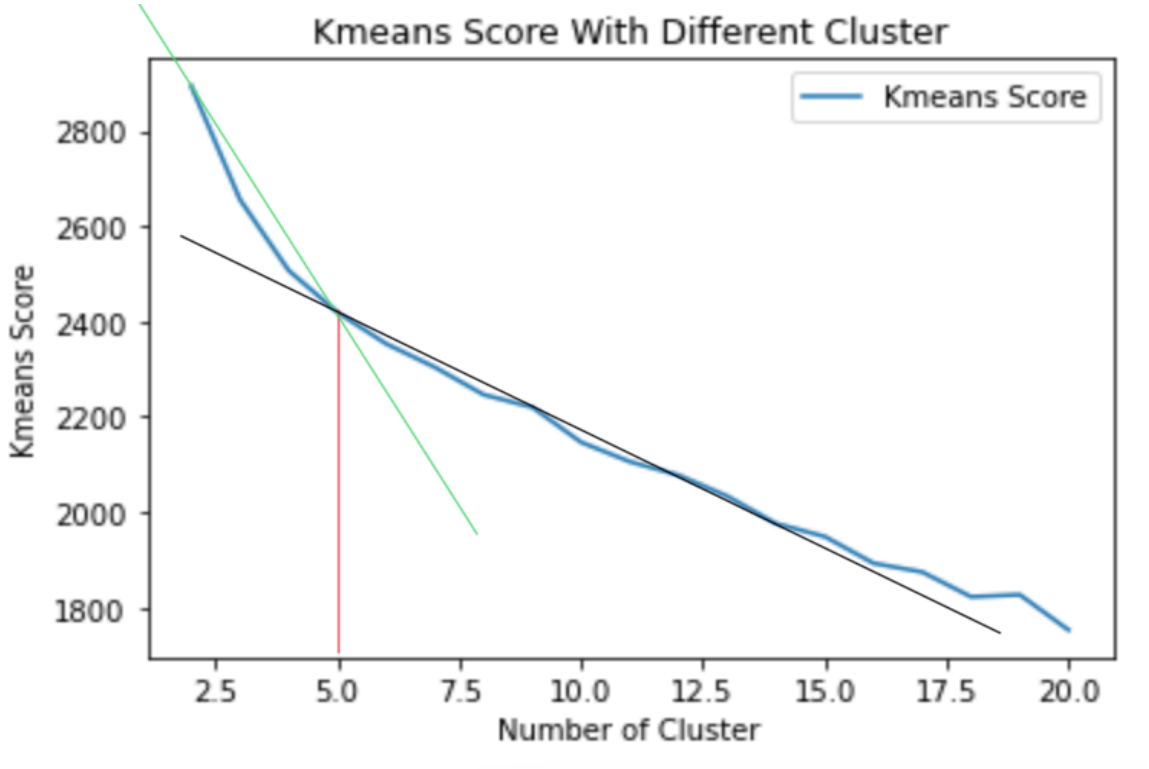
The red and blue clusters can be easily observed, but the yellow, green and black clusters are not well differentiated, so it is not easy to identify them.

2. TSNE with a different number of perplexity:



Although there is no obvious change, we found that when the perplexity becomes larger, the cluster differentiation is better.

Clustering

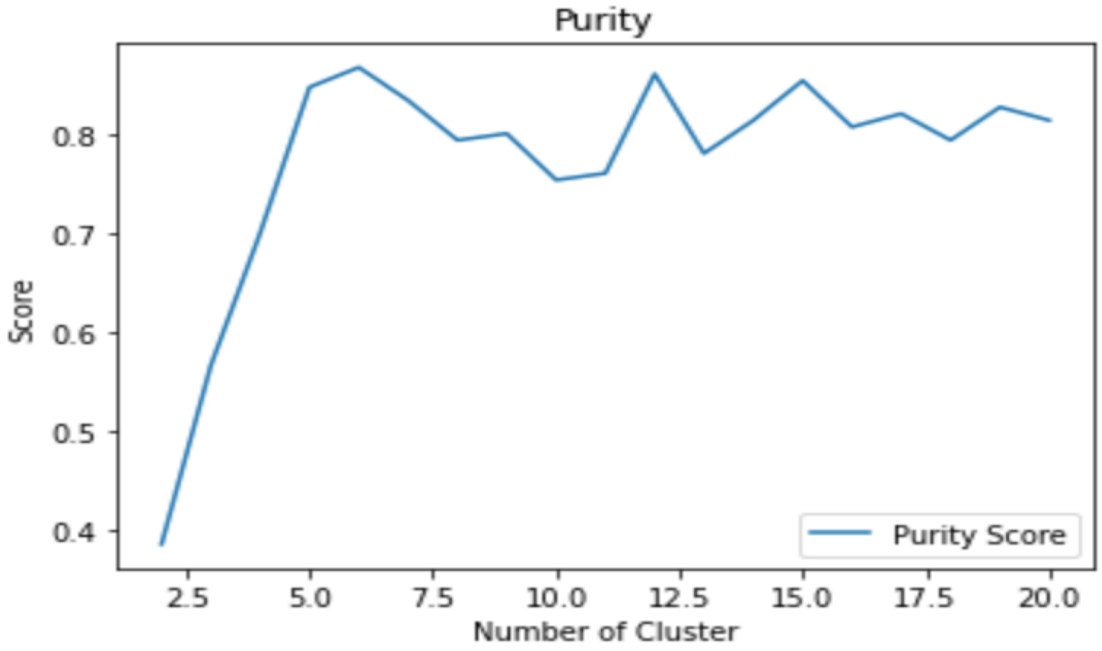


We do find that the kmeans score decreases monotonically when the number of k increases.

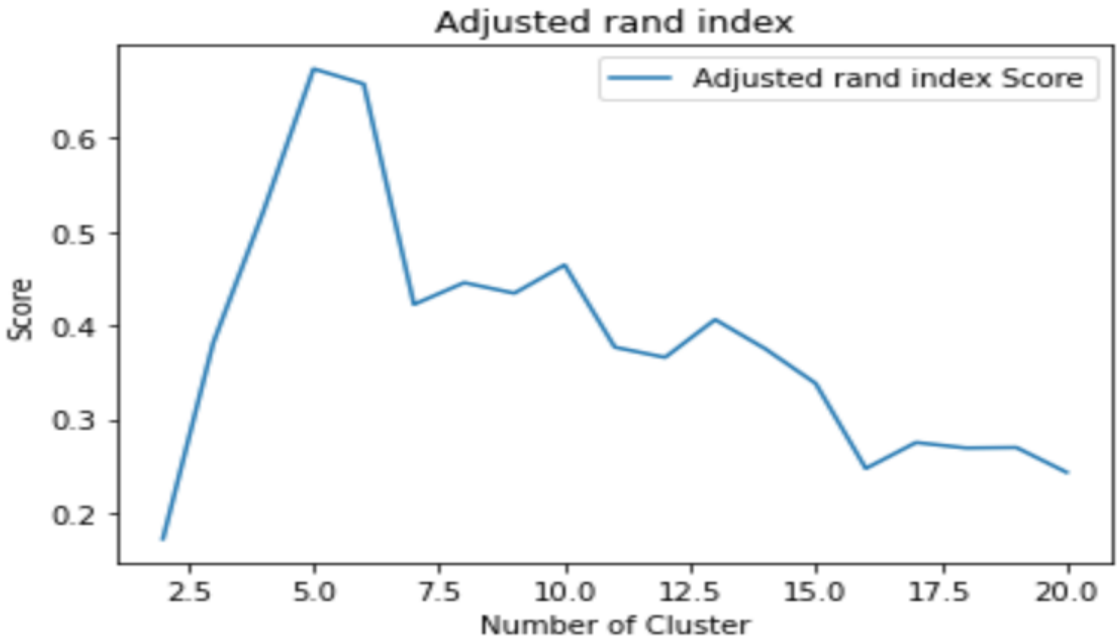
As the plot above, we can clearly observe that the curve decreases sharply when k is from 2 to 5 (green color curve line), but decreases gently when k is greater than 5 (black color curve line). Thus, we can conclude that the best classification is k = 5.

Different metrics:

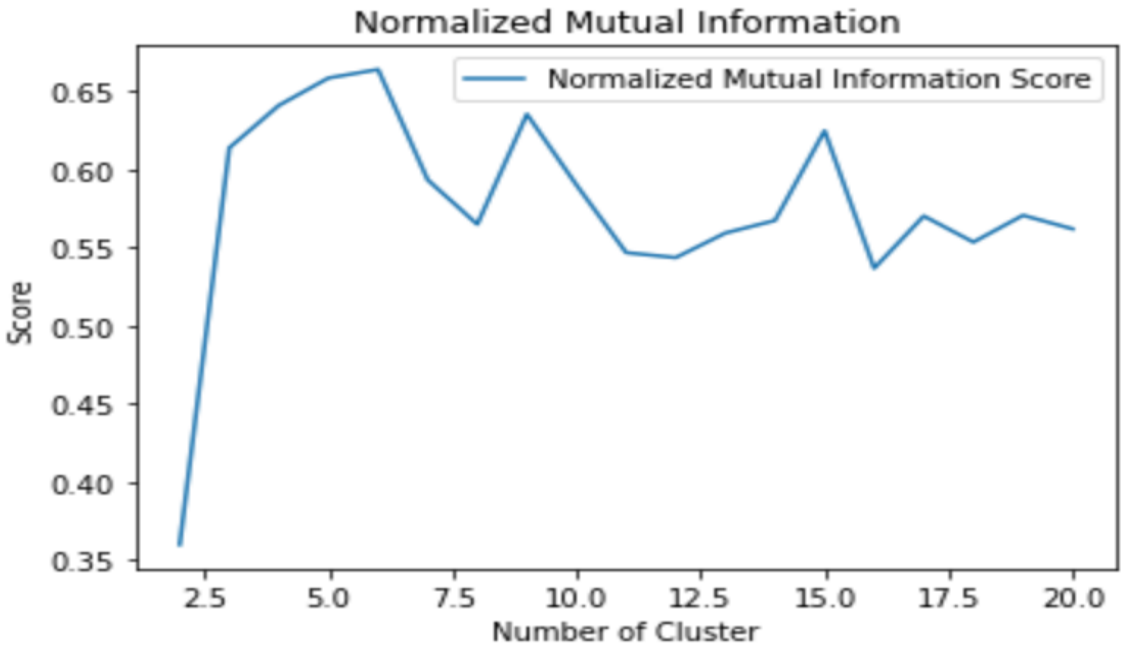
Purity:



Adjusted rand index:



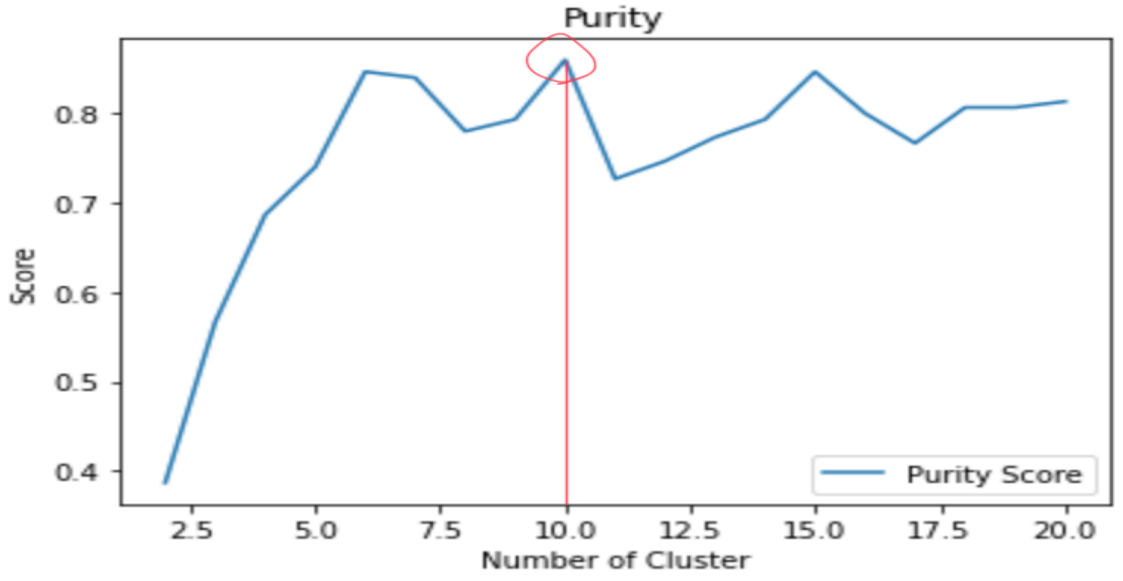
Normalized Mutual Information:



Observation and explain:

Purity:

For the purity method, we find that the scores when k = 11 & 15 are very close to those when k = 5. This is quite hard for us to use this method to find the k. In addition, we also found that other k scores were higher than k=5, such as the red circle in the below plot.



The reason why the purity method has many close high scores is that the results vary each time when we do the classification with Kmeans classifier (local optimum). This will cause the sum of TP and TN numbers will be close on different k. For example, if the result of the class is ‘a’, it’s correct, so the TP is 1 and TN is 0. The next time the result of the class is ‘b’, the TP becomes 0 and the TN becomes 1. These two conditions would cause the result TP+TN to be the same. And we know (TP+TN)/(TP+TN+FP+FN) is the equation of the purity method. So this may be the main reason, in our experiment, different k gives a close high score to the k=5, even though the result is higher.

Adjusted rand index:

For the adjusted rand index method, the result given k=5 has the highest score. And no other number of k has a close score.

As the description on the scikit-learn website, “The Rand Index computes a similarity measure between two clusterings by considering all pairs of samples and counting pairs that are assigned in the same or different clusters in the predicted and true clusterings.” We can know if the number of the clusters is different from the true label then it will have more penalty. This is the reason k=5 gives the highest score (the number of classes is equal to the number of the class of the label).

Reference:
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html

Normalized Mutual Information:

For the Normalized Mutual Information method, the result given k=5 has the highest score, but there are two other k given the high score (k=9 and k=15).

As the description on the scikit-learn website, “Normalized Mutual Information (NMI) is a normalization of the Mutual Information (MI) score to scale the results between 0 (no mutual information) and 1 (perfect correlation). In this function, mutual information is normalized by some generalized mean of H(labels_true) and H(labels_pred)), defined

by the average_method.” Because it is normalized by some generalized mean of H(labels_true) and H(labels_pred)) it may be why k=9 and k=15 give a higher score than other numbers of k except for k=5.

Reference:
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html

Evaluating two different clustering algorithms:

If we do not know the ground truth of the classification, the best approach may be the normalized mutual information. Since sklearn.metrics.normalized_mutual_info_score has the "label_pred" method, which is useful for measuring the consistency of two independent label assignment strategies on the same dataset when the true situation is not known. On the other hand, if we know the ground truth of the classification, the best approach may be the Adjusted rand index. Since this method imposes a relatively large penalty for misclassification, it is good to determine which algorithm is more appropriate.

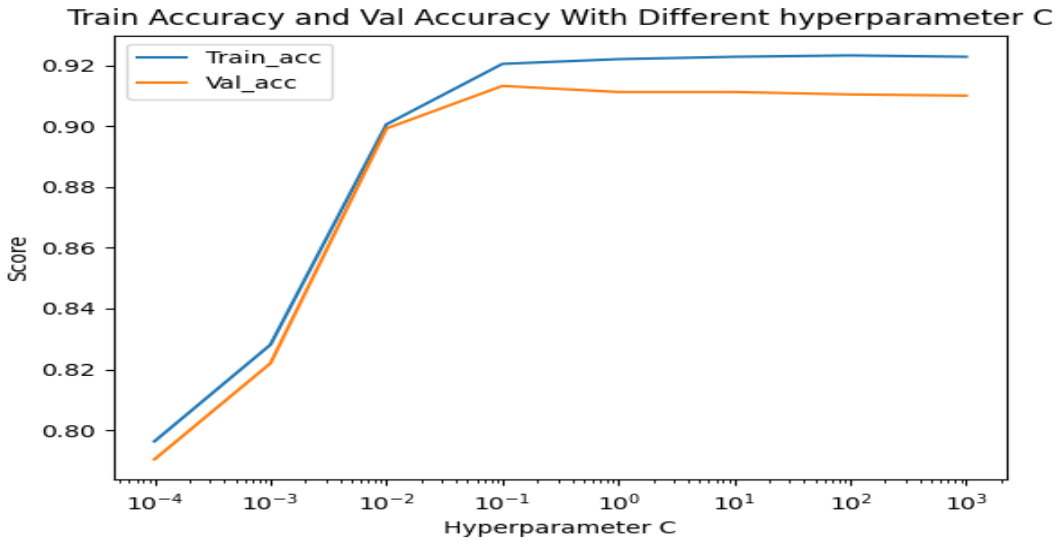
Using word embeddings to improve classification

How can you improve the bag-of-words representation for classification using the word embeddings?

We used the weighted average of the embedding of the words to be our improved method. First, We split the sentences from the data and removed some unnecessary words or notations. For example, “the”, “I”, "me", "," and "@". Second, we use TfidfVectorizer to capture the word frequencies and feature names of the processed data. Third, we use the provided embedding function to obtain a 200-dimensional vector for each feature name. Since we want to use a weighted average to represent a tweet, we multiply a word vector (200 dimensional) of each word in a tweet by the number of word frequencies (weight) and divide by the total number of word frequencies (weight) in a tweet, then we can get the average weight for each tweet. Finally, we use the weight average feature in the linear SVM classifier and get the result. Finally, we apply the weighted average of each tweet to the linear SVM classifier and obtain the results.

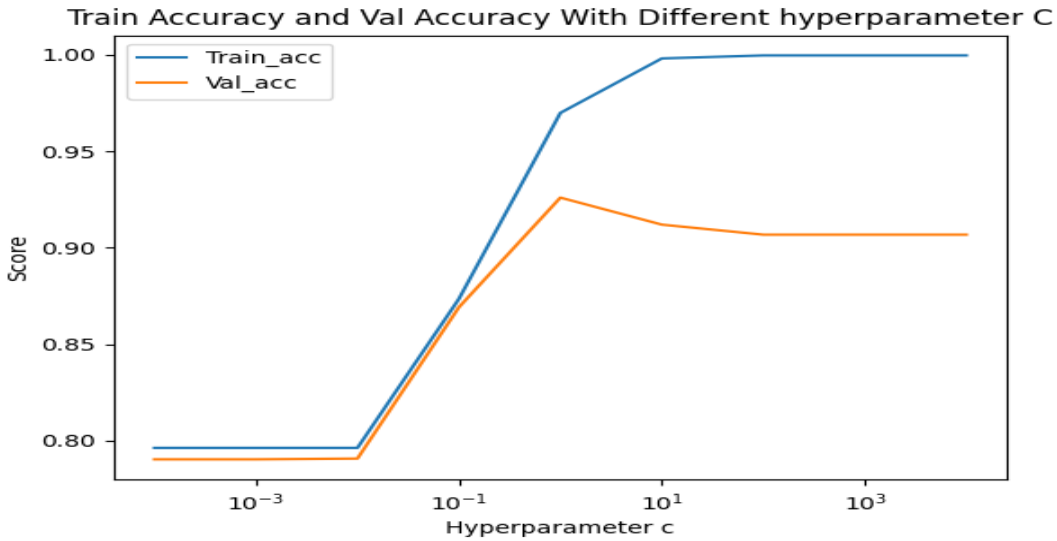
Result:

The training accuracy and the validation accuracy with the average weight method (linear SVM):



We got the highest validation score of **91.32%** (when the c = 0.1).

The train accuracy and the validation accuracy with the origin method (linear SVM):



We got the highest validation score of **92.6%** (when the c = 1).

We got the validation score of **86.78%** (when the c = 0.1).

For the same c (c = 0.1), the new method improves the performance by **5.23**% and does not overfit as much as the original method.