Tested running on flip1 server

IA4 - Group 26

Cheng Zhen
Bharath Padmaraju
Bharghav Srikhakollu

Word embeddings and Sentiment analysis

Explore word embeddings

(a) **Build your own data set of words**

(5 pts) You will begin by building a small set of words to visualize and play with. Specifically, use ’flight’, ’good’, ’terrible’, ’help’ and ’late’ as the initial set of seed words. For each seed word, find 29 most similar words based on the word embedding using Euclidean distance. This will give you a total of 150 words forming five clusters. In the report, please list the 29 most similar words for each seed word.

Answer:

→ **Flight:**

Word	Distance	Word	Distance
Flight	0	Airplane	6.00258
Plane	4.45689	Safe	6.00877
Flights	5.32675	Booking	6.01374
Boarding	5.4882	Fly	6.02409
Airline	5.58355	Departure	6.04052
Jet	5.59065	Waiting	6.04259
Flying	5.59299	Landed	6.09697
Heading	5.6417	Journey	6.12156
Arrival	5.70659	Passengers	6.12345
Airlines	5.85213	Transit	6.12451
Travel	5.89098	Delay	6.13663
Shuttle	5.90023	Crew	6.14757
Delayed	5.91624	Pilot	6.17311
Landing	5.92666	Trip	6.18669
Route	5.96298	Taxi	6.20436

→ **Good:**

Word	Distance	Word	Distance
Good	0	There	4.70148
Great	3.64107	Day	4.70444
Well	3.96383	Luck	4.7177
Nice	4.00469	Sure	4.76147
Better	4.19665	It	4.76949
Night	4.30113	Thing	4.78543

Bad	4.34788	Pretty	4.81795
Morning	4.43596	Think	4.83542
Way	4.52146	Have	4.84457
Hope	4.54451	All	4.85276
But	4.55671	Yes	4.86179
Too	4.57775	Very	4.89081
Really	4.60558	Again	4.9018
Right	4.68877	Work	4.90492
Though	4.69955	Yeah	4.90592

→ Terrible:

Word	Distance	Word	Distance
Terrible	0	Actual	5.73712
Horrible	2.8716	Horrific	5.78009
Awful	4.31316	Bloody	5.78166
Bad	5.24022	Ridiculous	5.80851
Brutal	5.35858	Such	5.82791
Idea	5.45579	Atrocious	5.86491
Horrendous	5.49845	Dreadful	5.87343
Horrid	5.56903	Sick	5.88777
Shitty	5.58304	Wtf	5.89628
Quite	5.59716	Fucking	5.9003
Worst	5.631	Cruel	5.90768
Similar	5.64538	Seriously	5.93984
Shame	5.67322	Unreal	5.94033
Worse	5.71411	Mess	5.97906
Crap	5.73043	However	5.98936

→ Help:

Word	Distance	Word	Distance
Help	0	Plz	5.58522
Need	4.66871	Helped	5.62169
Helping	4.71355	Support	5.6628
Please	4.8564	Anyone	5.68027

Pls	5.26437	Should	5.74155
Let	5.27073	Save	5.74649
Us	5.33329	Take	5.75843
Give	5.35559	Want	5.76772
Trying	5.3827	Bring	5.79499
Can	5.39777	Maybe	5.80615
Helps	5.40568	Lets	5.81187
Must	5.42209	Seriously	5.81942
Tell	5.54717	Able	5.81981
Find	5.55147	Here	5.82526
Could	5.5624	Needs	5.84511

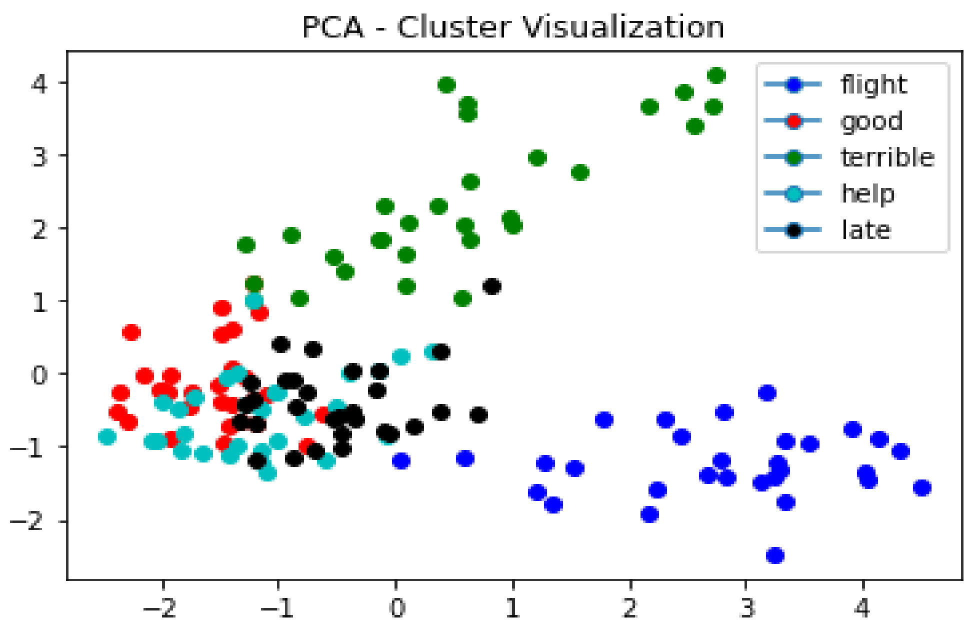
→ Late:

Word	Distance	Word	Distance
Late	0	Night	5.41858
Early	4.00205	Anyway	5.45233
Earlier	5.09402	Yesterday	5.49731
Usual	5.10465	Last	5.49978
After	5.28243	Maybe	5.50156
Again	5.28409	Yet	5.51821
Saturday	5.29169	Monday	5.53958
Afternoon	5.30979	Wait	5.55033
Hour	5.31787	Either	5.55333
Guess	5.33142	Mins	5.56436
Missed	5.36901	Wake	5.59768
Work	5.37432	Before	5.60765
Hours	5.38573	Thursday	5.61733
Sunday	5.40189	Hopefully	5.63757
Since	5.41431	Friday	5.65562

(b) <u>Dimension reduction and visualization:</u>

→ (10 pts) Apply PCA (you can use sklearn.decomposition.pca) to the 150 words and visualize them in 2-d space. In your visualization, you should assign each seed word (and the words similar to that seed word) a distinct color. Do you observe five distinct clusters in your visualization?

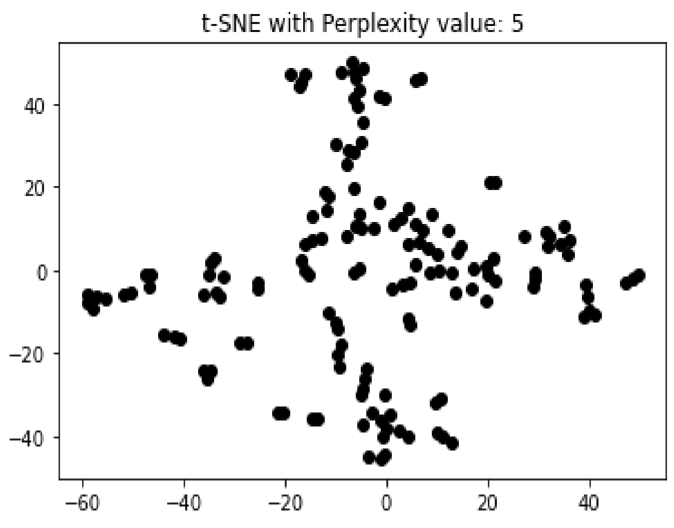
Answer:

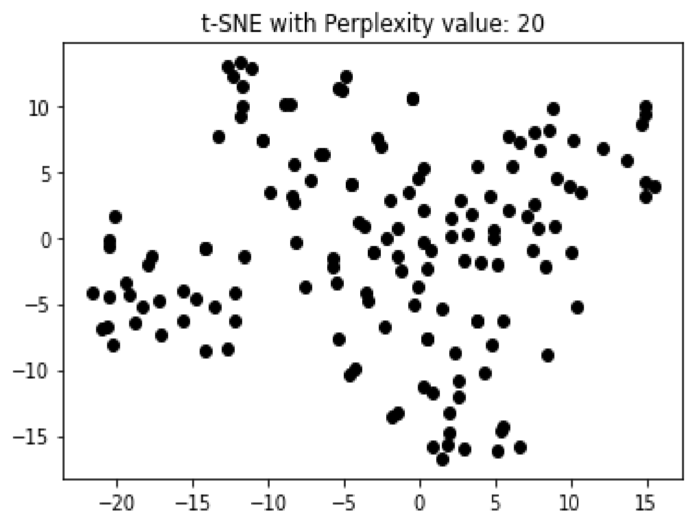


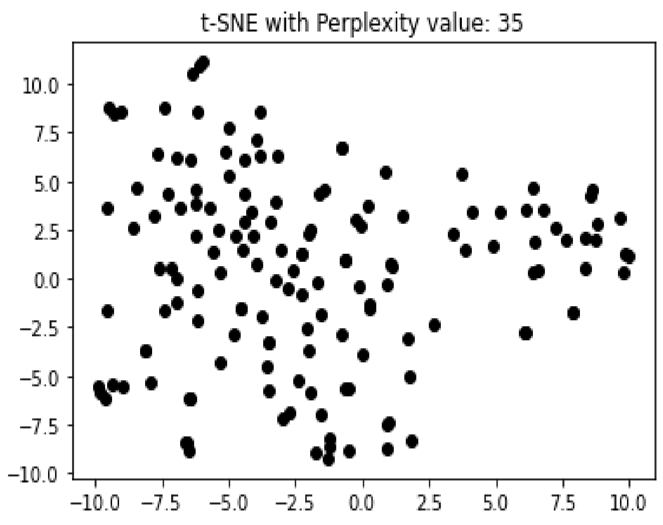
Explanation: The 5 words are mostly in distinct clusters. However, "help", "late" and "good" somewhat overlap. Whereas terrible and flight are clearly the most distinct. It should be noted that the overlap between "help", "late" and "good" only means the three classes are not well separated in 2-D, but not necessarily in other dimensions. There is a chance that if reducing to another dimension through PCA, we can tell the difference from classes at a specific angle (e.g., think about a 3-D plot where we can change the tilting angle to look into).

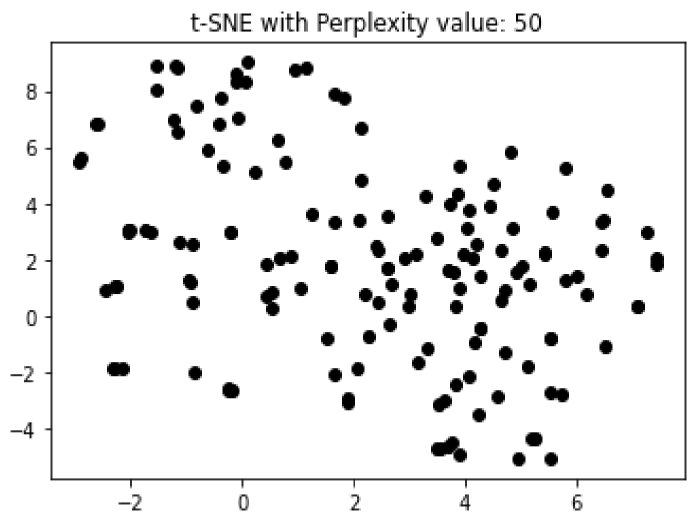
→ (15 pts) Next you will apply t-SNE (you can use sklearn.manifold.TSNE with Euclidean distance) to the 150 words and visualize them in 2-d space using the same color mapping. Note that Perplexity is a critical parameter for t-SNE. It is recommended by the authors of t-SNE that the perplexity value should be between 5 and 50. For this assignment, you will need to explore different perplexity values and observe the resulting impact on the visualization. In your report, you are expected to provide substantially different visualization results by using different perplexity parameters.

Answer: we tested the t-SNE with perplexity values as 5, 20, 35, 50 and the plots, explanation are as below.









<u>Explanation:</u>

The t-SNE results with different perplexity values are shown above. The trend is as expected: when the perplexity is small, we see a better clustering effect from the t-SNE. As suggested in the sklearn official documentation, large perplexity should be used for large data sets. This is because large perplexity considers a more even probability distribution over all points — there are enough points to refer to. In our case, we only have 30 points for each class: we need a small perplexity to focus on the nearest neighbors.

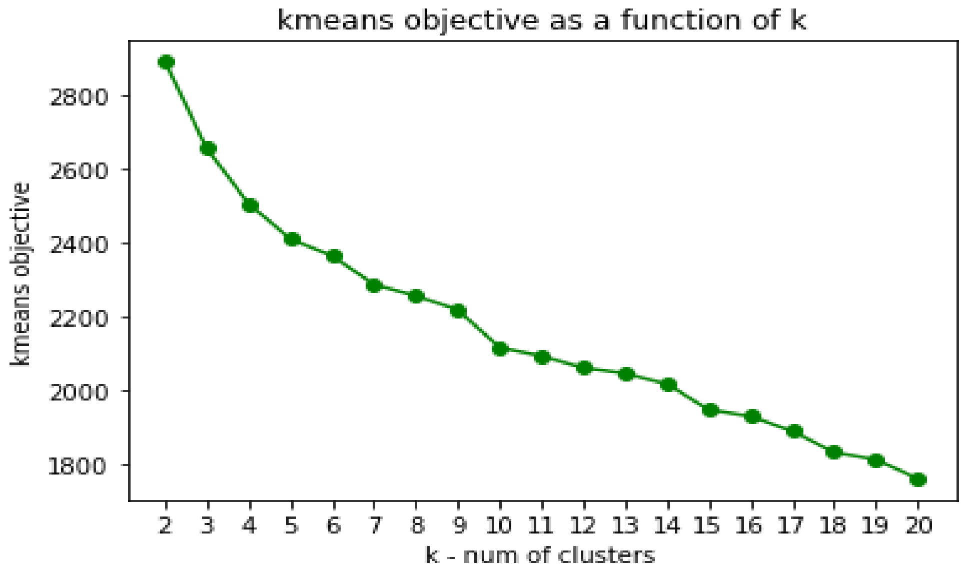
(c) <u>Clustering:</u>

→ (20 pts) For this part you will apply k-means clustering (you can use sklearn.cluster.kmeans, you can keep most default parameters except for n cluster) to your words using different k values ranging from 2 to 20. For each k value, record the resulting kmeans objective (or inertia as in sklearn), which measures:

$$\sum_{i=1}^{k} \sum_{x \in C_i} |\mathbf{x} - \mu_i|^2$$

Plot the kmeans objective as a function of k. Do you observe monotonically decreasing objective value as we increase k? Do you see any evidence from this curve that suggests k = 5? Provide an explanation for your observations.

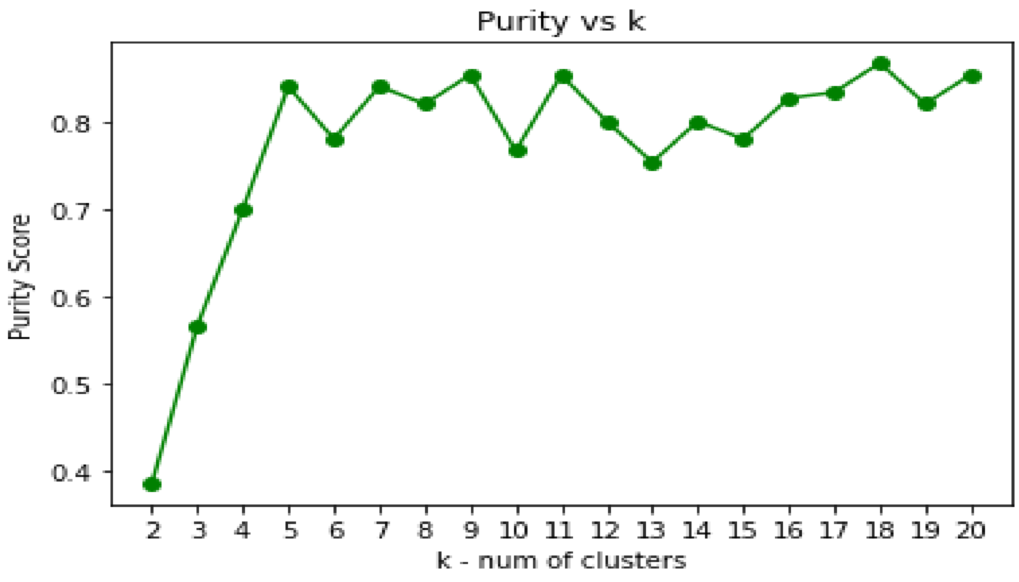
Answer: For kmeans clustering - along with "n_clusters", passed the "random_state" as "10". Verified with TA and took some random_state just to make sure we can reproduce the same results when we run the same code multiple times.



Explanation: We observe that the k-means object trends downward as the number of clusters increases. This makes sense because we end up with more centroids, decreasing the mean distance. Proving by contradiction: 1) assume there exists a k such that increasing from k to k+1 increases the k-mean objective. 2) To add a cluster from k to k+1, the most conservative attempt is to simply keep every clustering the same but randomly choose one point P to form a new cluster. Not necessarily the optimal clustering but this will ensure that the k-mean objective decreases by the distance from P to its original centroid (because P is now its own centroid). 3) Therefore, the assumption does not hold — the k-mean objective goes down as we increase the value of k.

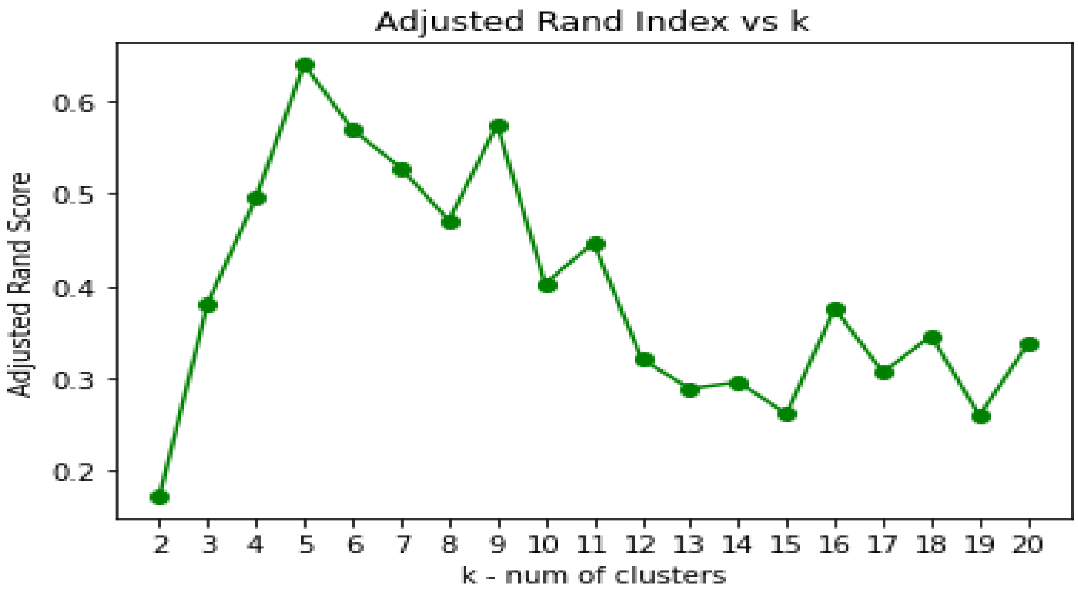
→ Plot each metric you get as a function of k. Do you k = 5 give the best score for different metrics? Provide an explanation for your observation. Which of these three metrics are appropriate to use if we are evaluating two different clustering algorithms that automatically search for the number of clusters in the data (that is, one algorithm might find five clusters in the data while the other might find ten)?

- Purity



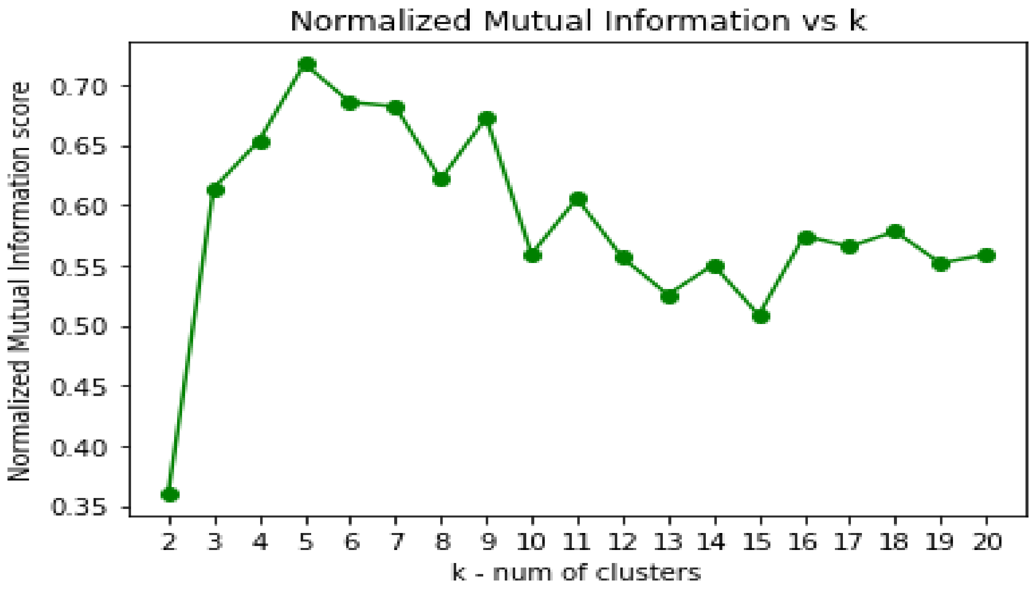
Explanation: At k=5 we do get a max score, as it stabilizes. However, this is just local. At 9, 11 and 18 clusters we see a similar or higher purity score. This pattern makes sense because in general we expect the score to increase as the number of clusters increases as well.

- Adjusted rand index



Explanation: Yes, with the adjusted rand index, the highest score is at 5 clusters. Rand index measures the similarity between clusters. So it makes sense that with a higher number of clusters, the similarity decreases as the number of dimensions increases.

- Normalized mutual information



Explanation: Yes, for normalized mutual information, the highest score is observed at 5 clusters. NMI measures the entropy of our models, and entropy decreases as the uncertainty decreases, giving us a better score. So it makes sense that at 5 clusters the entropy is at its lowest.

Question: Which of these three metrics are appropriate to use if we are evaluating two different clustering algorithms that automatically search for the number of clusters in the data (that is, one algorithm might find five clusters in the data while the other might find ten)?
Answer: Purity is appropriate from the perspective of correctness. As the k increases, purity goes up. We can define a good k such that the purity does not go up much further as k keeps increasing beyond the value. This suggests that more clusters does not help with improving accuracy. Similarly for the normalized mutual information, we can define a good k from which we don't get benefit (in terms of mutual information) by increasing the number of clusters.

Using word embeddings to improve classification (40 pts)

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

Answer:
We have tried the ensemble/bagging methods for improving the classification.

Method	Training Accuracy	Validation Accuracy
Tf-idf Vectorizer + Lightgbm	0.9588	0.9136
Tf-idf Vectorizer + Xgboost	0.2038	0.9124
Tf-idf Vectorizer + Randomforest	0.9997	0.8337

Lightgbm:
Motivation:
Tried lightbgm algorithm as it is known for high-performance gradient boosting. This approach splits the tree leaf wise. When it grows on the same leaf, the algorithm is expected to reduce more loss and can result in better accuracy. Also, it is one of the fastest algorithms. This gave us the motivation to try lightbgm. We observed that the algorithm ran very fast and the validation accuracy was 0.9136. We reached this accuracy without parameter tuning and we understood that with tuning of the parameters like num_leaves can also increase the accuracy as the splitting is taken leaf-wise, but there can be a possibility of overfitting.

Xgboost:
Motivation:
Xgboost is an extreme gradient boosting method. We tried this to compare with the lightgbm and ended up achieving similar validation accuracy similar to the lightgbm, but we have a very low training accuracy. The ~20% of training accuracy is apparently incorrect given the decent testing

accuracy of about 92%. We tried to troubleshoot this by changing another built-in function for counting accuracy from Accuracy_score to model_selection.cross_val_score.

Randomforest:

Motivation:

Random forest has a strong ability of reducing variance from the uncorrelated decision trees. And it is usually considered not vulnerable to overfitting compared to similar decision-tree methods because of the low variance. However, we report a very high training accuracy around 100% with a relatively low testing accuracy (only at 83%): this suggests a big overfitting. We attribute this to two potential reasons: 1) the training data has a small size, 2) the decision tree method is prone to overfitting inherently (random forest is a collection of decision trees). That’s why we also see overfitting from other decision-tree methods we reported.