**Week 6 Writeup - Model Training, Hyperparameter Tuning, and Model Evaluation**

**Ryan O’Hara, Jack Metzger**

**Model Approach**

For the model approach, we trained the KNN model using the datasets from week 5. KNN is the nonparametric algorithm that classifies the new observations by identifying the most common class among the K closest observations in the training dataset, based on a distance metric. When the link between input features and output labels is not clearly defined and decision boundaries are non-linear, this approach works especially effectively.

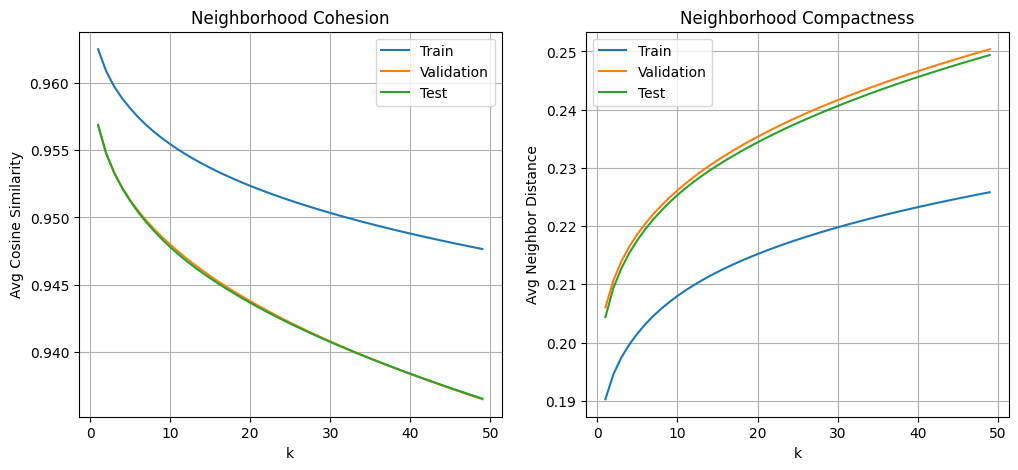
We used the Week 5 dataset for training, validating, and testing. These datasets are suitable for distance-based algorithms, such as KNN, because they include normalized numerical values and one-hot encoded category features.

We need to tune hyperparameters and do feature scaling and encoding for some KNN features. And we have finished the feature selection, so it is not necessary now, and all depends on our special variable, which is called K, and this can be changed by the performance of the model.

**Hyperparameter**

At our current stage in the model building process, the only real hyperparameter we are training is going to be our K value for our K-nearest neighbors model. The problem here was less about tuning K and actually the metric we use to evaluate the tuning. Currently, as we are performing an unsupervised recommendation algorithm, we don't have a real accuracy metric or something that we can tangibly measure our success with. As we are trying to recommend songs that transcend genre, we currently aren't using those as labels. This week we aren't clustering our own songs either, so the only measurement we could evaluate was the distance between the input song and its nearest neighbors.

The two metrics we were able to use to get an idea of how K affects the tightness of our songs in the latent space are average intra-cluster cosine similarity (neighborhood cohesion) and neighborhood density (neighborhood compactness). These metrics are somewhat inverse of each other, where cohesion measures the average similarity between k-nearest songs and compactness measures the distance between k-nearest songs. Obviously, the fewer K the model takes in, the smaller the distance will be between these songs, so the question becomes how small of a K we can be comfortable with without overfitting. Shown below are charts showing these metrics for up to K = 50.



A couple more theoretical questions about our K at this juncture in our model need to be addressed. First, should we even be concerned about overfitting? Normally, overfitting hurts the accuracy of the model as it will be based on the training data, almost copying the characteristics of the training data and those specific trends won’t map to the test data. In our case for this model, because we don’t necessarily have an accuracy measurement, we aren’t hurt in the traditional sense from overfitting. Things where having a low k could impact us are if remixes or remastered songs with very similar lyrics have slipped through the cracks of our data cleaning process, but to my knowledge there are no “repeat” songs in the data. Other than that case, we shouldn’t be impacted by low values of k.

The other side of the coin is do we just want a really large k since we have no real use for the distance measurement. We don’t necessarily care about distance at all, we only care about the actual ranking of said distances. Whether the distance score is 0.1 or 0.5, we only care about what the X closest songs are, so theoretically the K being large and decreasing the neighborhood cohesion scores shouldn’t really matter either. We obviously don’t want to have a k too large, but might it be beneficial for the model to have more information of more songs to base its recommendations off of?

Finally, and most importantly, does K even matter? Since we are, at the core, just taking the songs that exist closest to each other in the latent space, the k value shouldn’t have too much of an impact on the recommendation at all. I tested this by testing the recommendations given on a KNN model trained for values of k = 5, 15, and 50 and while the average accuracy increased as shown in the charts above, the end rankings did not change.

**Model Training**

To train our KNN model for lyric-based song recommendations, we first needed to ensure that the embedded lyrics were properly formatted. The embeddings, which had been generated from song lyrics using a pre-trained FastText model, were initially stored in a format that wasn’t compatible with model training - specifically, they were saved as Python objects within a pandas Series, rather than as uniform NumPy arrays. This caused shape errors when attempting to fit the model. To fix this, we applied np.stack() to convert the list of individual embedding vectors into a 2D NumPy array, ensuring each song’s lyrics were represented as a fixed-length vector suitable for input into the KNN algorithm.

Once the embeddings were cleaned and structured correctly, we split the data into a training and test set. The training set was used to fit the K-Nearest Neighbors model using cosine similarity as the distance metric and starting with n\_neighbors = 5. After fitting the model, we used the test set as our query set - these were the songs for which we wanted to generate recommendations. For each test song, we retrieved the 5 closest neighbors from the training set based on lyrical similarity, effectively identifying songs with similar themes, language, or tone. This allowed us to evaluate how well the model grouped semantically similar songs together.

**Model Evaluation**

We evaluated our model in a more general sense as we are not currently clustering or using any sort of labels in this iteration of our model building process. We did this by gathering the top 5 recommended songs for many different input songs. While we have included some interesting examples below, there are a few general observations that can be made from this model thus far.

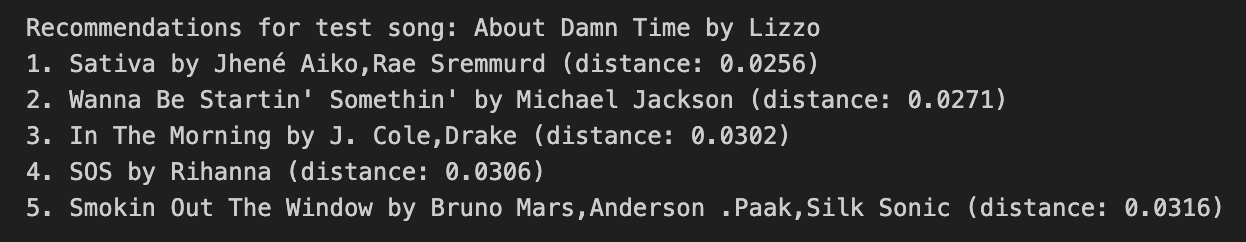
One, while there are certainly recommendations that make a lot of sense, you can tell that there is a reason songs cannot be suggested purely based on the merit of their lyrics. A metal rock song and a rap song can mention many of the same themes and words / phrases, however you would likely never recommend a heavy metal song to a diehard rap enthusiast. While our goal of this model was to transcend genre and use a more sophisticated approach to classify and recommend songs, there is a certain element of genre that needs to be factored in. I believe this will be handled by the other numerical metrics that add more to the model in the future.

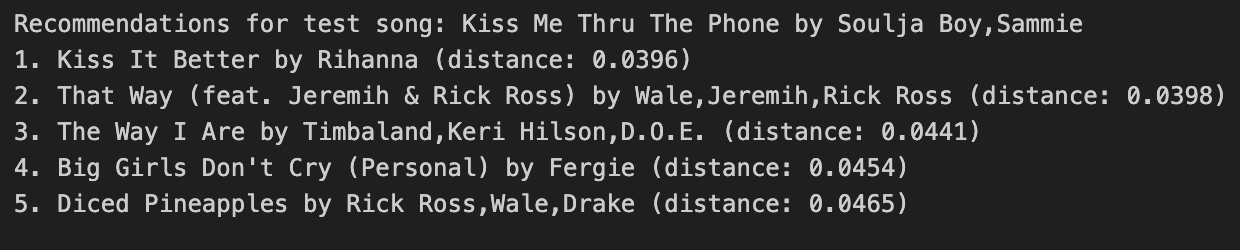
Second, there are many songs that appear more frequently in suggestions than others, and often are more popular songs by more popular artists. Certain Drake and Morgan Wallen songs are frequently included in top 5 recommendation lists. You could make the argument that these songs are popular because they have general themes that are included in many different songs, making them good central candidates that exist in the middle of the latent space. However we are still not convinced that is the case and am curious what makes these songs occur at such frequencies. We suppose it is fair that popular songs should be suggested more often, because people are more likely to like them. This is something to think about going forward, but below are the most recommended songs (interestingly the majority are rap):

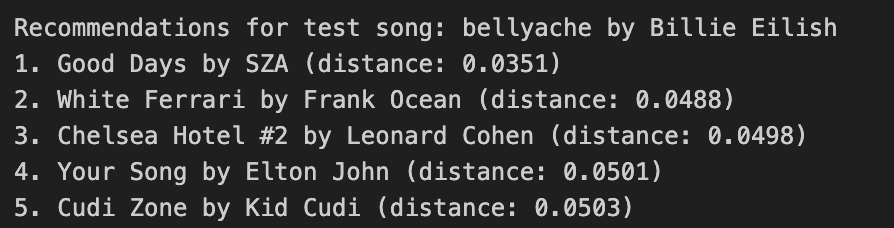


Lastly, rap songs seemed to generate the best recommendations in that it most often recommended other rap songs, often by similar / the same artist. This occurred far less frequently in other genres and there are a couple reasons we think this could be the case. First, the type of language used in rap songs typically lean further into vulgarity than other genres, so this repeated vulgar language can cluster rap songs pretty close to one another. Second, rap songs average more words per song than any other genre, so it is possible that the higher count of similar words further groups these songs together.

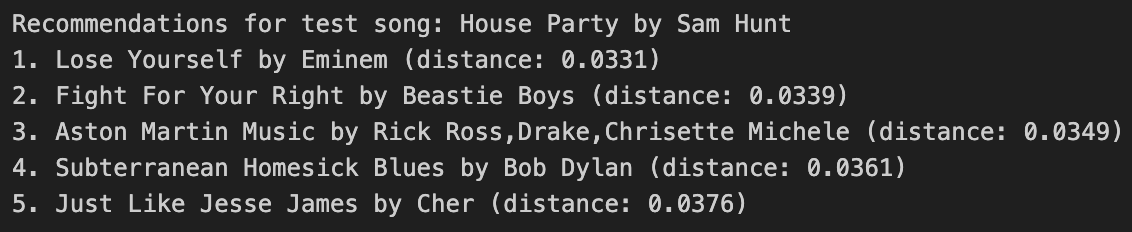
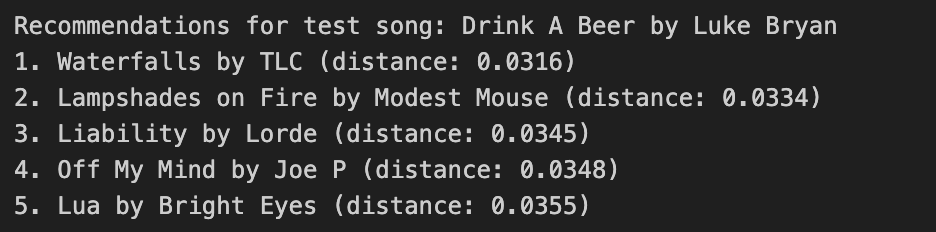
Below are some examples of recommendations for songs across all genres and popularity. These first two show pop songs be matched with other pop songs from similar artists:



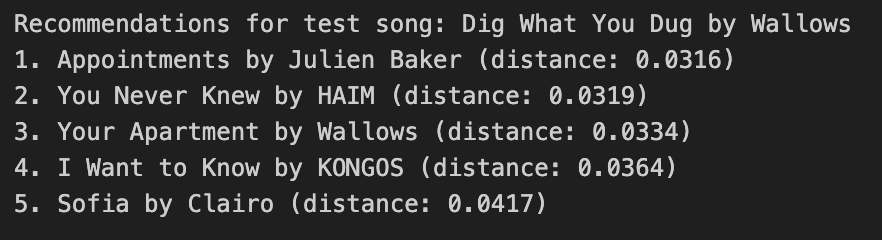
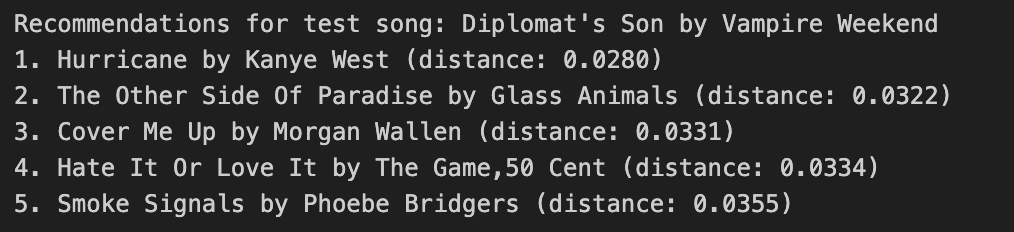




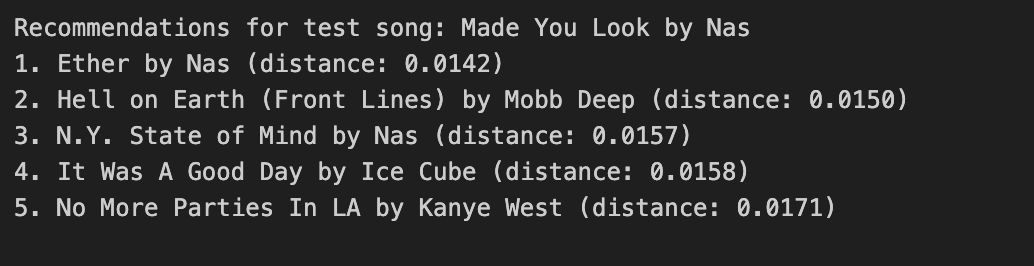
Then there is an example of a country song that gets paired very closely with rap and other miscellaneous genres. The model seems to struggle with country music currently.

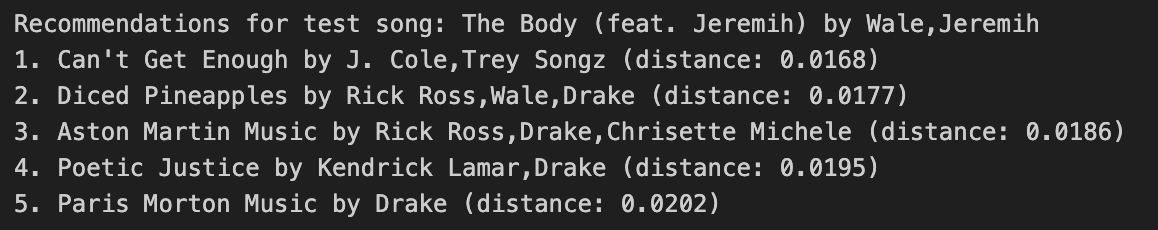


Other indie songs seem to recommend songs properly:

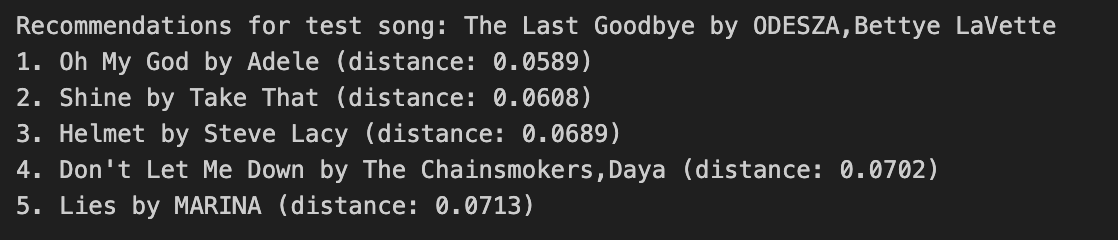
  


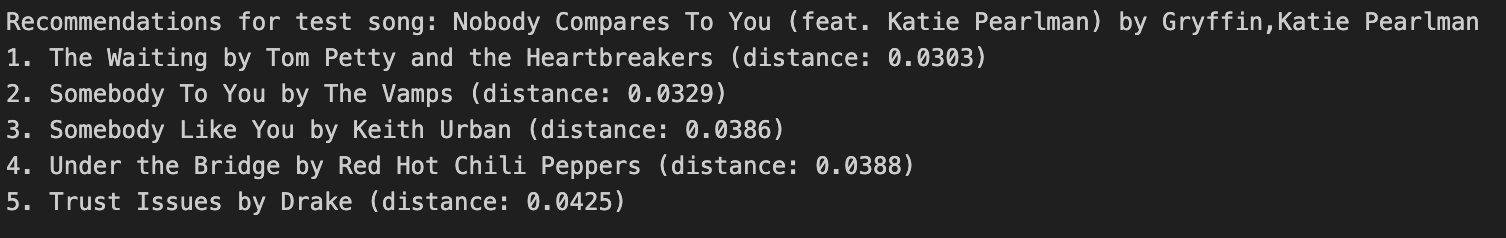
As previously mentioned, rap songs are getting really solid recommendations currently:



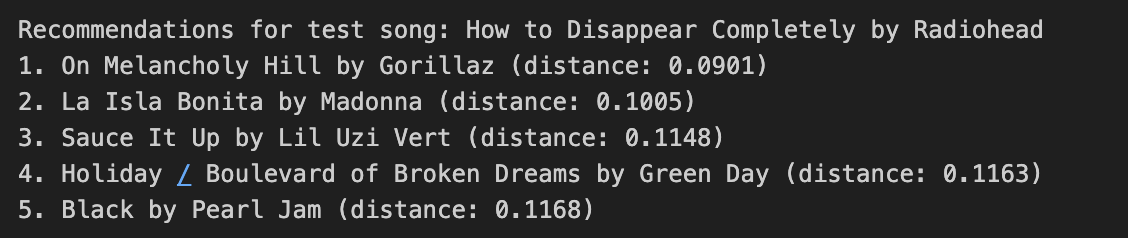
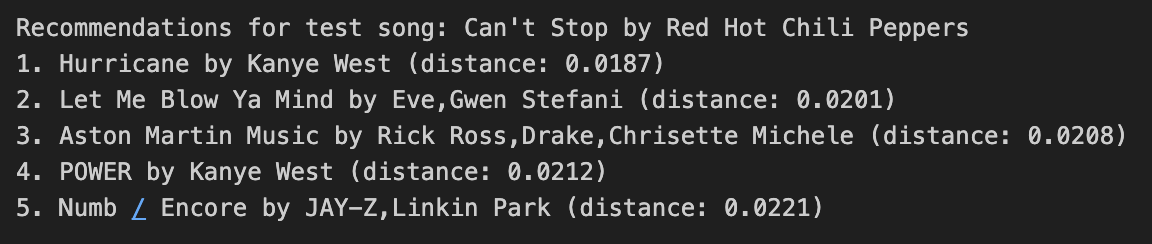
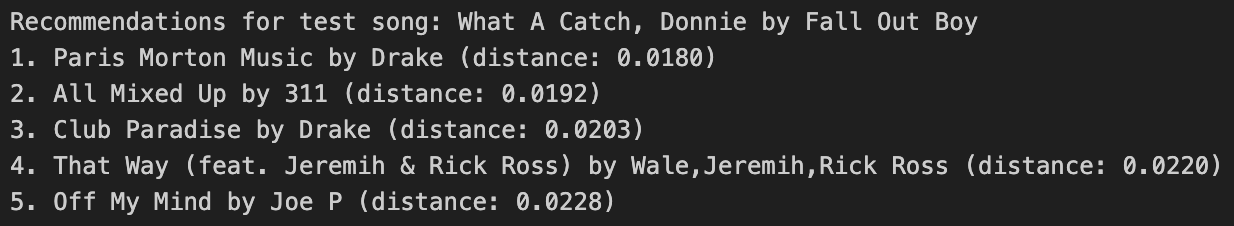


It also makes sense that EDM gets poor recommendations as that genre is most dependent on the actual production of the music and less on lyrics. They also typically have far fewer lyrics:





Rock and alternative often give rap songs interestingly, but still can give out decent recommendations (as shown by Radiohead):



Any advice on accuracy measurements and next steps for models would be appreciated - we will continue to try to book time to meet and chat about our questions around accuracy and model development!