Introduction

This guide will help you understand and learn the skills you need to build a KNN model to give recommendations. We will go through all the steps of the project from data selection and processing (Cleaning), building the model, model evaluation , and making a simple application to use the model. We will be using python for all of this project, and the libraries used are pandas, numpy, scikit-learn, and pyQt6.

Finding and Preprocessing Data

We found our dataset in Kaggle. You could use other websites or APIs for your dataset depending on what you are looking for, However, based on our experiences finding an already ready dataset is much better than trying to get your data from an API. You still need to clean the data to make sure there are no data points that will mess up our model, but overall the dataset was easier to work with. In our project we placed the code cleaning section of our project in the KNNModel.py file.

Cleaning the data is an important part of the process as well. What we did for cleaning was that we looped through the data and removed the points that had missing columns (audio features), were duplicates, and had unacceptable values for a set feature. Even though the dataset we used was already cleaned by the creator it is always better to do a cleaning yourself as well. Also, it is better to clean all of the columns of the dataset and not the ones that you are planning to use because you will have the case that you decided to use one more column, and you need to go back and change the code for your cleaning.

KNN Model

The first step that you take for your KNN model creation is deciding your hyperparameters, which is the value of K and the method of measurement that you are going to use. K is the number of neighbours and could be any natural number, and method of measurement or the metric could be “Euclidean” or “Cosine”. The Euclidean metric is based on distance on a straight line between two points, and the Cosine metric is based on the angle between the vectors of the two points. In the beginning it does not matter that much what value you start with, you can just use some random hyperparameters like K=10 and metric=”Euclidean”. Later on in the Evaluation process we will optimize the hyperparameter. It is also important to decide the features that you want to use. You have to check your dataset and based on that decide what features you want to work with and implement it to your code.

After that you need to standardize your data. We do this to make sure all the features have the same amount of effect on the data points location, and just because the value of one feature is higher, its being more impactful on the place it lands. To this you make a standard scaler variable and then you use the fit_transform(X), where X is a list of all the values in one of your feature columns or matrix containing all

of your feature columns. This will Standardize the model so that mean is 0 and standard deviation is 1 for all your features. After you do this you can apply weights to some features to make them more impactful than the others. We would recommend making a weights list for all your features and setting all of them to 1 for the start and multiplying them by the weight after you standardize them. This way later in the project you can change the weight of some features without having to rewrite the code. You have to use the StandardScaler module from the sklearn.preprocessing library for this section of code.
We build the model with the following module: "from sklearn.neighbors import NearestNeighbors". The function NearestNeighbors() takes the argument K and the metric used. Then we use the fit() function from our KNN model on the matrix of the feature columns we have to make our model. Here is a snippet of the code that we were just describing in the function fit_scalar_and_knn() in the KNNModel.py file:

```
# Use only the feature columns that actually exist in this CSV
feature_columns = [c for c in FEATURE_COLUMNS if c in df.columns]
X = df[feature_columns].copy()

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply feature weights so tempo / energy / danceability matter more
X_scaled = apply_feature_weights(X_scaled, feature_columns)

knn = NearestNeighbors(n_neighbors=n_neighbors, metric=metric)
knn.fit(X_scaled)

return scaler, knn, X_scaled
```

Model Evaluation

Our evaluation method was 5-fold cross validation that compared the model with different hyperparameters and gave us the best one. Technically it was doing both optimization and evaluation. For example if we made small changes to the model the Evaluation.py script gave us what K and metric works best for the new model, and gave us the model performance on the final test. The final test was done with the 10% of data that was set aside in the beginning of cross validation. Also, we only measured some of our features in our model evaluation and not all of them to save some time. After running this code we would get the best hyperparameters and how good the model is with the said hyperparameters. For our model the best hyperparameters were K=5 and metric = "Euclidean". Here is a picture of what an

output of our evaluation script look like:

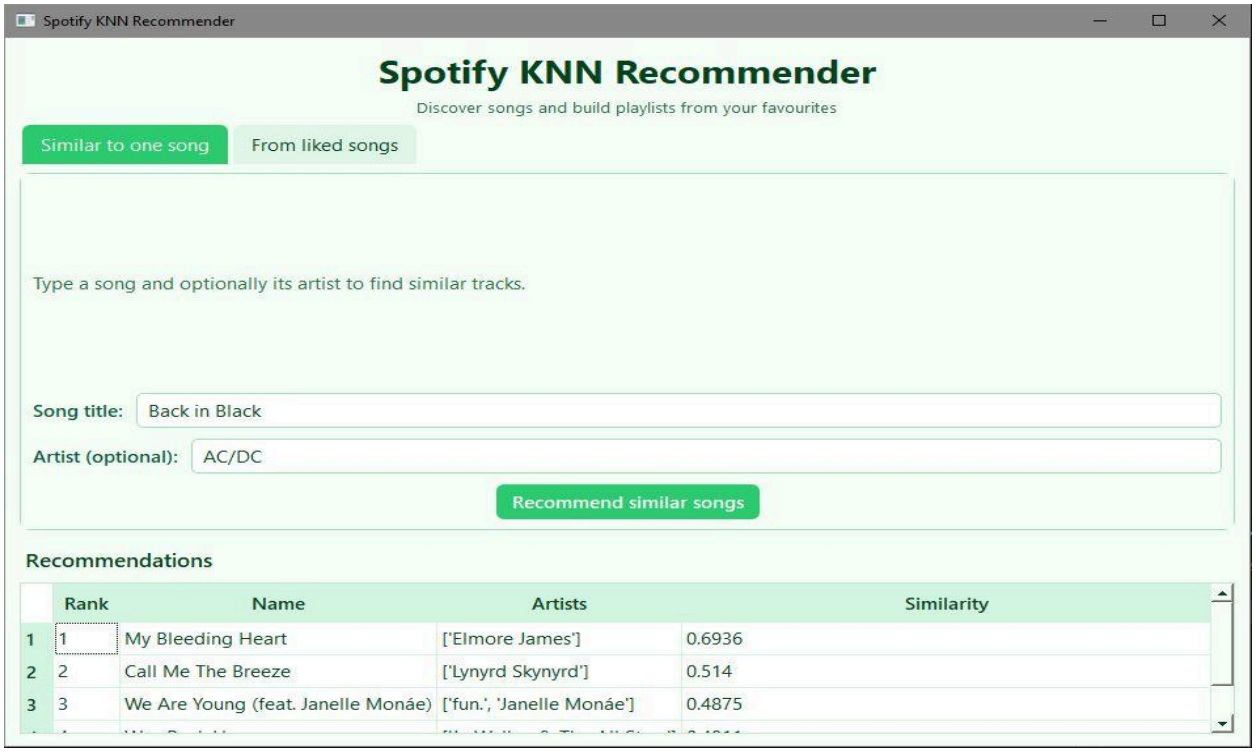
```
=== Cross-validation results (top 10 rows) ===
 k    metric  tempo_bucket_mean  tempo_bucket_std  energy_bucket_mean  energy_bucket_std  loudness_bucket_mean  loudness_bucket_std  combined_mean
 5 euclidean           0.894000          0.008602            0.903000           0.012791              0.700000             0.025243       0.832333
 5    cosine           0.890000          0.015862            0.900200           0.010815              0.695000             0.029346       0.828400
10 euclidean           0.884100          0.009987            0.885500           0.013903              0.690300             0.026459       0.819967
10    cosine           0.877200          0.010361            0.886200           0.013090              0.686500             0.028850       0.816633
15 euclidean           0.876867          0.012042            0.880667           0.014844              0.684933             0.030177       0.814156
20 euclidean           0.870950          0.014357            0.875750           0.013870              0.684200             0.032177       0.810300
15    cosine           0.871667          0.012207            0.878800           0.015440              0.679133             0.031355       0.809867
20    cosine           0.865900          0.012200            0.874850           0.014603              0.679400             0.032271       0.806717

=== Final test scores per label ===
tempo_bucket: 0.902
energy_bucket: 0.910
loudness_bucket: 0.697

Final combined test agreement: 0.836
(venv) cyrussandhu@Cyruss-MacBook-Air-188 music_rec %
```

We imported the modules KFold and train_test_split from the sklearn.model_selection library to the cross validations for us. We also used buckets to categorize our features. For example, the loudness feature has three buckets: very quiet, quiet, normal. If we guess 5 neighbors and for loudness feature 3 of them are similar then that 60% agreement score. Here is a code snippet of making buckets for loudness:

```python
def make_loudness_bucket_labels(df: pd.DataFrame) -> pd.Series | None:
    """
    Bucket loudness (typically negative dB) into a few coarse levels.
    """
    if "loudness" not in df.columns:
        return None
    loud = df["loudness"].astype(float)
    # Very rough buckets, you can tweak:
    bins = [-80, -20, -10, 5]
    labels = ["very_quiet", "quiet", "normal"]
    return pd.cut(loud, bins=bins, labels=labels, include_lowest=True)
```

Interface Application

We decided to go with a desktop application that runs on local computers which is the GUI.py. First we wanted to go with web application using github pages, but after some research we found out that might some time to be able to use the python codes on web application, so for simplicity we decided to make application with python, so we can just import the model from the KNNModel.py. We used many modules from PyQt6 in the process of making our interface, and tried to follow the light green and white theme in our application. Our final product is an application that is easy to use and can showcase all capabilities of our KNN model. Here is a picture of our interface:



One important decision we made for our interface was that we made a testApp.py which was a simple console application only used for early tests. This application also showcased all capabilities of our model and was helpful when we were trying to test the model with real people. We would recommend to other people pursuing a similar project to just make a console application in the beginning for the testing as making a full-fledged web app or desktop is a lengthy process. You can leave the creation of your interface as the last task, so you are not rushing it and the task is not slowing you down.

```
=== Spotify KNN Recommender ===
1) Find songs similar to one track
2) Get playlist from your liked songs
3) Quit
Choose an option: 1
Song title: Back In Black
Artist (optional): AC/DC

Recommendations:

 rank                              name                         artists  similarity
    1                 My Bleeding Heart              ['Elmore James']      0.6936
    2                Call Me The Breeze            ['Lynyrd Skynyrd']      0.5140
    3 We Are Young (feat. Janelle Monáe)     ['fun.', 'Janelle Monáe']      0.4875
    4                      Way Back Home ['Jr. Walker & The All Stars']      0.4811
```

Troubleshooting Tips

The first tip I would give is if you are on the team use VCS like github, so that it is easier to revert back after making mistakes. Also, VCS makes working in collaboration much easier. For our first milestone we were not using github, and the communication through discord and constantly sending codes to each other was not a pleasant experience. I would say the first step of you project should be making a github repository and adding everyone to it.

Another tip would be to make a fast and easy to follow pipeline so that when you need to change something later on in a project, you do not have to go through redoing too much work. For example, we decided to change the weights of some of our features very late into the project, but because of how well made our KNNModel.py was it just took changing a few lines. Furthermore, the Evaluation.py also was very helpful to find the best hyperparameters for the new model. Hence, this major change in later part of the project only took us a couple of minutes, and if our pipeline was not as well made as it is right now, this process could have easily taken hours. We would recommend to other groups to make sure all of your codes are flexible for future changes and adjustments.

Summary

This type of project takes a lot of coordination and time, and it is important that they are approached with a competent plan. From our experiences, having a big outline for the project and some milestones to make sure the team is on the right track is very important. If you fall behind on one milestone, it might affect your whole progress significantly as other milestones might not be respected either. It is important to have a group leader that makes sure all the needed deliveries are met. Also, it is also important for all group members to have a proper understanding of the AI model that you are trying to implement. If people are on different pages then it might be hard for the group to make decisions and move forward.

To conclude this document is an easy to follow guide that our group made based on experiences on this project. Even though some project might have different prompt, they will likely follow all the procedures in here as these are not limited to our project, and all of these could be applied to other AI projects as well.