Classifying Emotions of Labeled Spotify Songs for Improved Music Recommendation

Soyeon Kim, Yejin Kim, Yoojin Oh
Ewha Womans University, Dept of Artificial Intelligence

Outline

- Introduction
- Background
- Data Collection and Preprocessing
- Define and Fit models (SVM, KNN)
- Performance Results
- Conclusion

Introduction

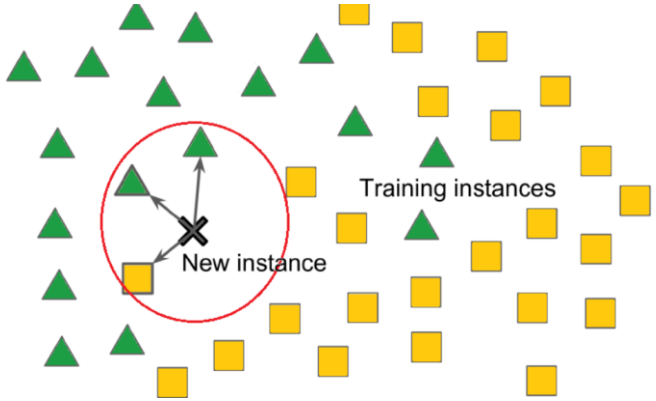
- Usually, music recommend music depending on artist and genre. However, by using data that labeled 4 types of emotion to 278,000 songs, music could be recommended also considering the emotions. This way, user satisfaction will get improved, and will eventually increase loyalty to the service.

- In this project, we will train and evaluate the performance of models on the emotion labeled Spotify Songs dataset in order to contribute in better recommendation afterwards

Background : KNN

◆What is KNN(K-Nearest Neighbor)

- the supervised learning algorithms.
- predict the dependent variable (y-value) of a new observation by using information from K nearby samples



◆ How to choose the appropriate value of K?

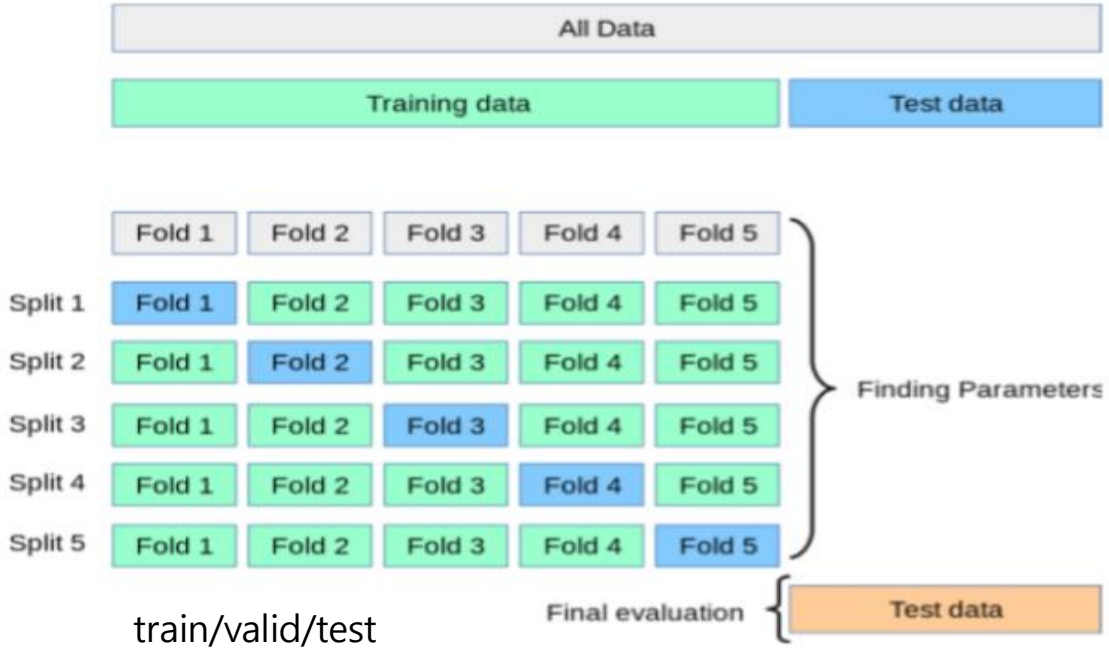
- too large K → misclassify fine boundary regions (underfitting)
- too small K → highly sensitive to outliers (overfitting)

→ cross-validation

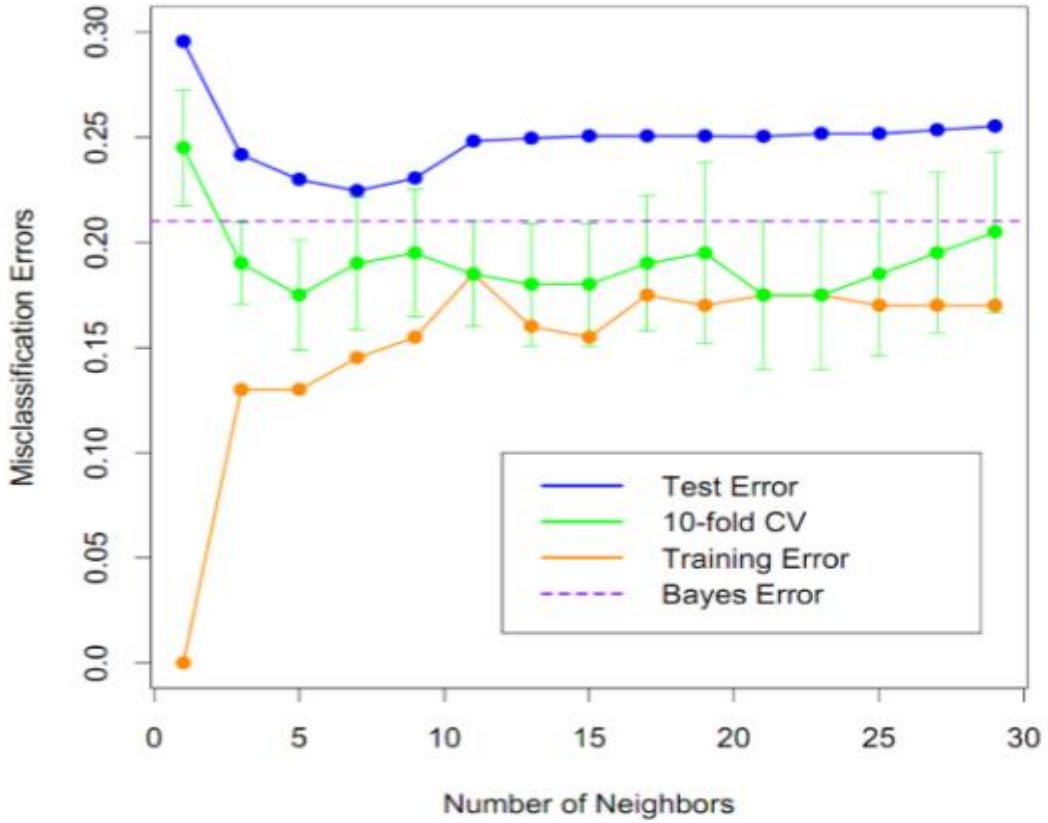
Background : Cross validation

◆ **The problem that cross-validation solves is...**

- overfitting
- sample loss problem



train/valid/test



Background : SVM

◆ What is SVM (Support Vector Machine)

- The supervised machine learning algorithm used for data classification using the Support Vector and Hyperplane
- When input data is given, SVM finds a boundary to classify that data into appropriate datasets.

◆Process of finding the decision boundary in SVM

- the **margin** between classes should be made as wide as possible

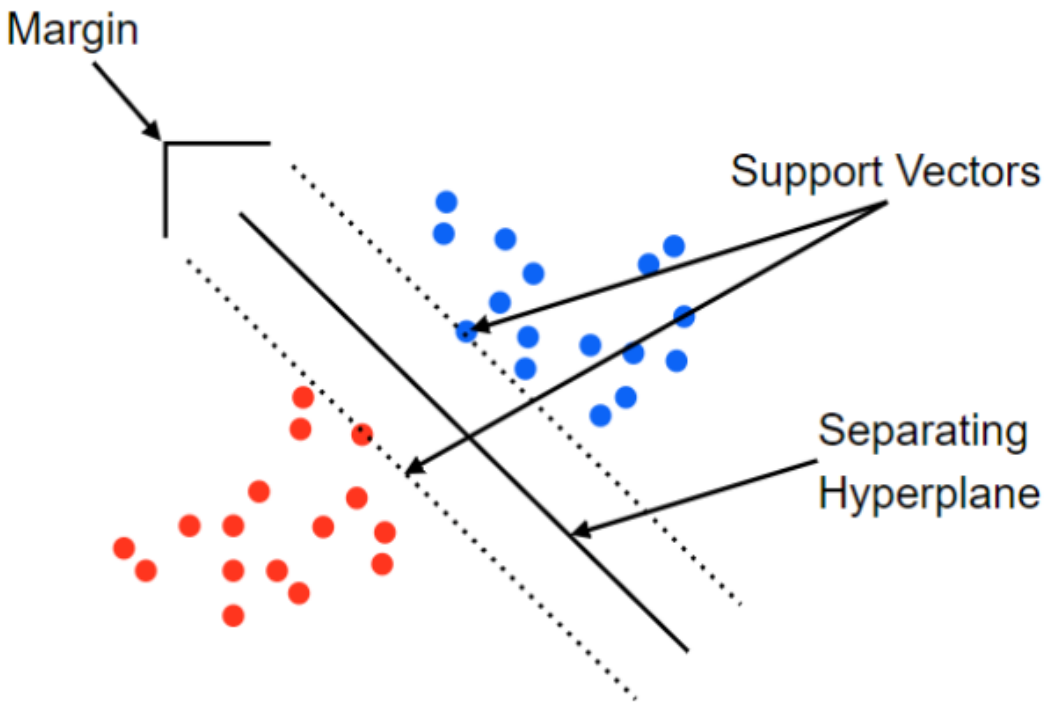
Background : SVM

If) Having a larger empty margin

→ a higher likelihood of successful classification even when new data points are introduced

↓

Goal : maximizing the margin



Data Collection & Preprocessing

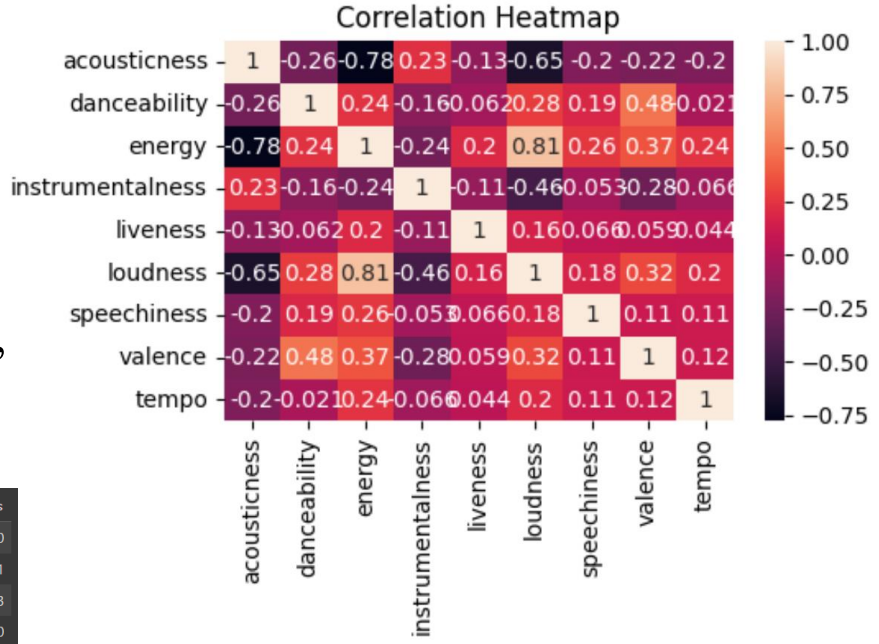
◆ Data Collection

Emotion labeled songs dataset from Kaggle.

label(4) : {'sad': 0, 'happy': 1, 'energetic': 2, 'calm': 3}

feature(9) : ['acousticness', 'danceability', 'energy', 'instrumentalness',
'liveness', 'loudness', 'speechiness', 'valence', 'tempo']

	Unnamed: 0.1	Unnamed: 0	duration (ms)	danceability	energy	loudness	speechiness	acousticness	instrumentalness	liveness	valence	tempo	spec_rate	labels
count	277938.000000	277938.000000	2.779380e+05	277938.000000	277938.000000	277938.000000	277938.000000	277938.000000	277938.000000	277938.000000	277938.000000	277938.000000	2.779380e+05	277938.000000
mean	138968.500000	138968.500000	2.324961e+05	0.552583	0.556866	-10.363654	0.087913	0.386583	0.255044	0.189217	0.449602	119.196002	4.754654e-07	1.179101
std	80233.933896	80233.933896	1.171830e+05	0.188905	0.279681	6.672049	0.112500	0.364504	0.373745	0.163596	0.267471	30.462256	9.190229e-07	1.021033
min	0.000000	0.000000	6.706000e+03	0.000000	0.000000	-60.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000e+00	0.000000
25%	69484.250000	69484.250000	1.720130e+05	0.431000	0.342000	-12.747000	0.035900	0.033800	0.000000	0.096200	0.220000	95.072250	1.531461e-07	0.000000
50%	138968.500000	138968.500000	2.131055e+05	0.571000	0.591000	-8.397000	0.047100	0.262000	0.001090	0.121000	0.434000	119.940000	2.345459e-07	1.000000
75%	208452.750000	208452.750000	2.648660e+05	0.693000	0.792000	-5.842000	0.082200	0.754000	0.645000	0.227000	0.665000	138.869750	4.449937e-07	2.000000
max	277937.000000	277937.000000	3.919895e+06	0.989000	1.000000	4.882000	0.965000	0.996000	1.000000	1.000000	1.000000	244.947000	5.971860e-05	3.000000

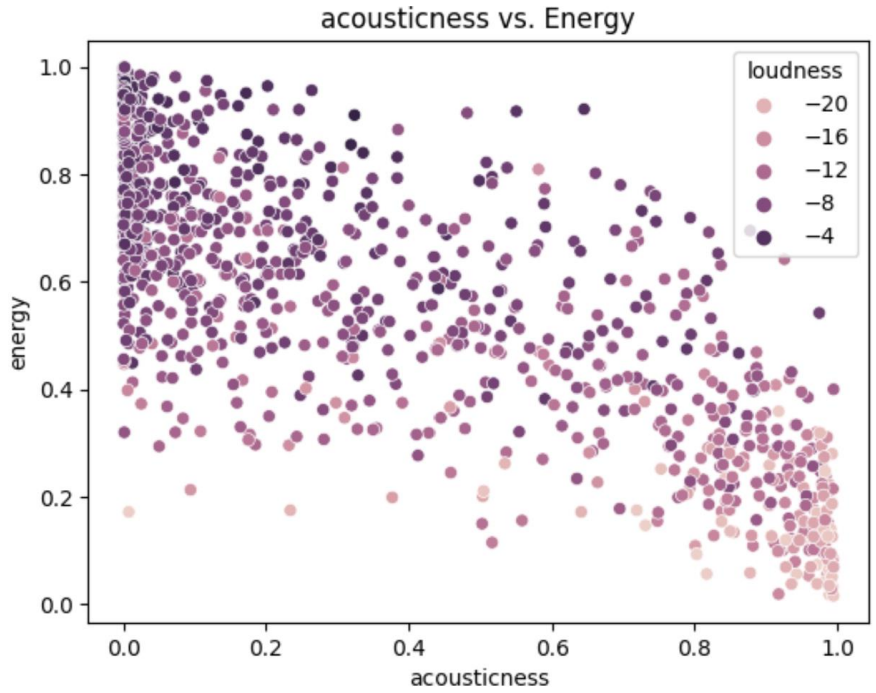


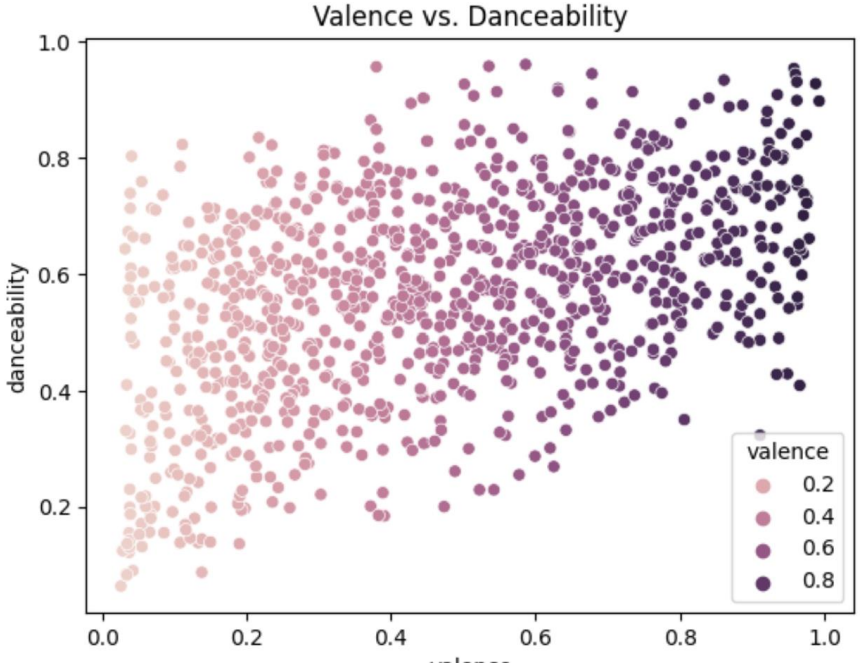
→ Visualize correlation between features as a heatmap

https://www.kaggle.com/datasets/abdullahorzan/moodify-dataset

Data Collection & Preprocessing

◆ Data Collection (Visualization)





→ Visualize the relationship between ‘Acousticness&Energy’ / ‘Valence&Danceability’ using a scatter plot
+ use color to represent additional information based on the 'loudness’ / ‘valence’

Data Collection & Preprocessing

◆ **Features** (total #9)

- Acousticness : measure whether the track is acoustic
- Danceability : describes how suitable a track is for dancing based on a combination of musical elements
- Energy : represents a perceptual measure of intensity and activity
- Instrumentalness : predicts whether a track contains no vocals

  (“Ooh” and “aah” sounds are treated as instrumental in this context)
- Liveness : detects the presence of an audience in the recording
- Loudness : the overall loudness of a track in decibels (dB)
- Speechiness : detects the presence of spoken words in a track
- Valence : describes the musical positiveness conveyed by a track
- Tempo : the overall estimated tempo of a track in beats per minute (BPM)

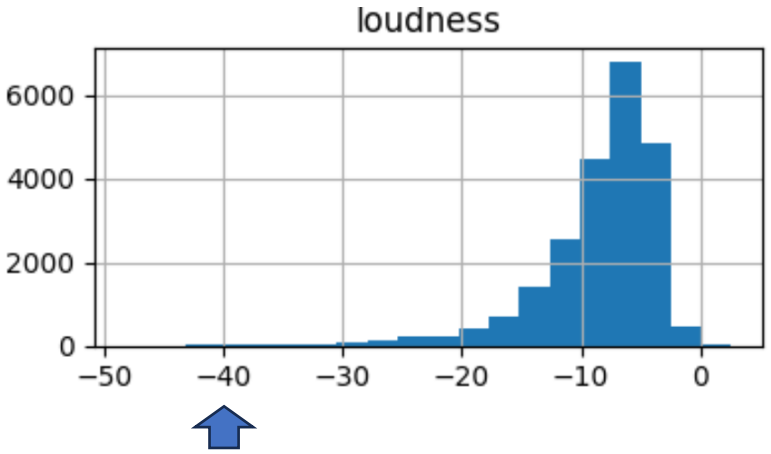
Data Collection & Preprocessing

◆Data Preprocessing

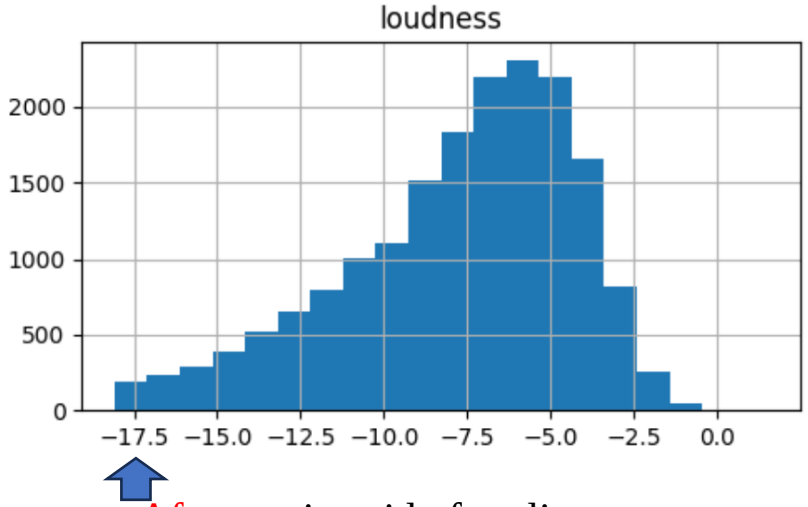
Split Train : Test = 70 : 30 (true amount = 78436, 33616) or 80 : 20

random_state = 42

K-fold Cross-validation = default(5) (little different model by model)



→ Before getting rid of outliers



→ After getting rid of outliers

```
1 df.isnull().sum()
```

```
Unnamed: 0.1        0
Unnamed: 0          0
duration (ms)       0
danceability        0
energy              0
loudness            0
speechiness         0
acousticness        0
instrumentalness    0
liveness            0
valence             0
tempo               0
spec_rate           0
labels              0
uri                 0
dtype: int64
```

→ Checked if there are missing values in each column (=> 0(zero))

Data Collection & Preprocessing

◆ Data Preprocessing – Outlier detection & removal

Number of datasets

- total of original : 236816
- total after getting rid of outliers : 177915

decrease of approximately 24.89%

⇒ IQR(Interquartile Range) = Q3 – Q1

```
temp = ['danceability','liveness', 'loudness', 'speechiness', 'tempo']
```

```
Q1 = df[temp].quantile(q=0.25)
Q3 = df[temp].quantile(q=0.75)
print("Q1 ₩n", Q1)
print("₩nQ3 ₩n", Q3)
```

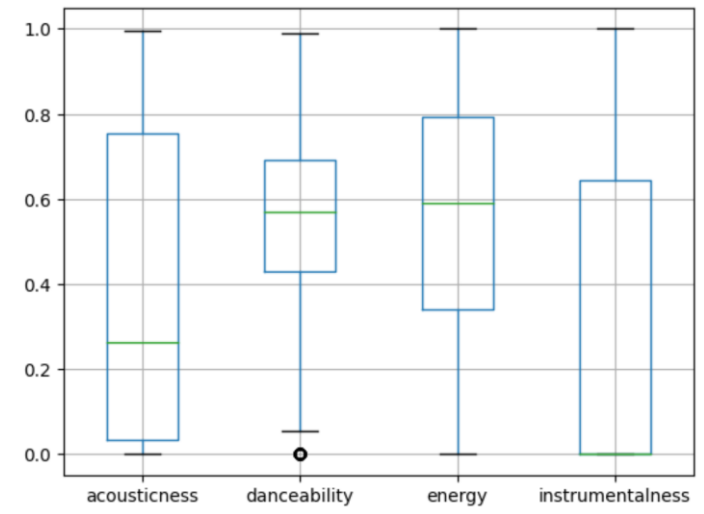
```
Q1
 danceability      0.43100
liveness          0.09620
loudness        -12.74700
speechiness       0.03590
tempo            95.07225
Name: 0.25, dtype: float64

Q3
 danceability      0.69300
liveness          0.22700
loudness         -5.84200
speechiness       0.08220
tempo           138.86975
Name: 0.75, dtype: float64
```

Detecting outliers using boxplots →

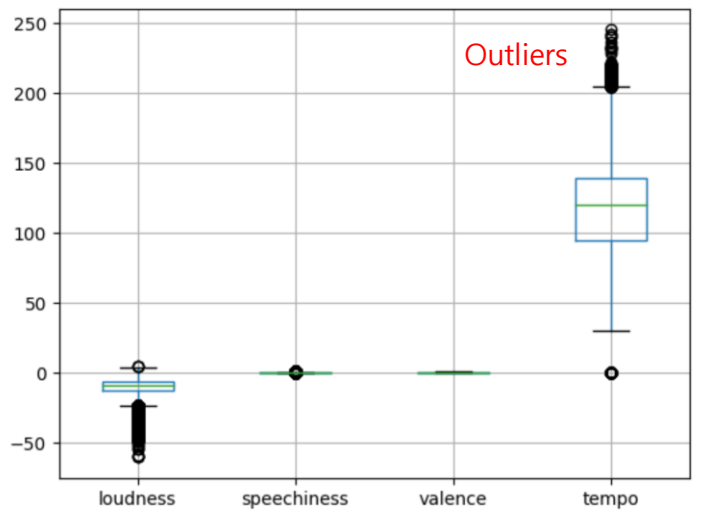
```
1 df.boxplot(column= features[:4])
```

<Axes: >




```
1 df.boxplot(column= features[5:])
```

<Axes: >



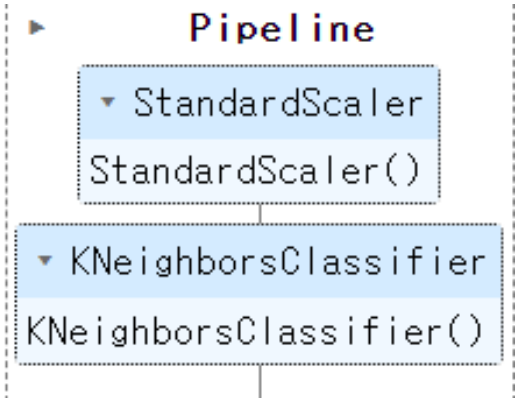
Define & Fit models (1) : KNN

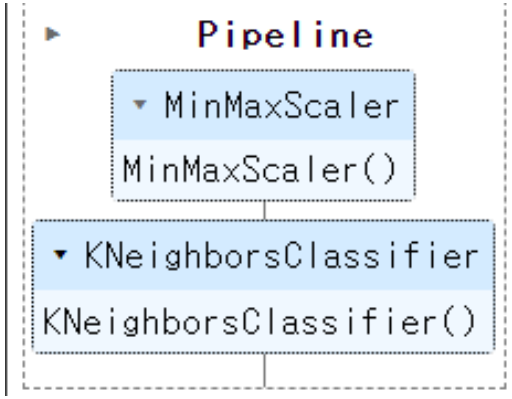
Defined Pipeline in order to apply cross validation and Normalization for each step.

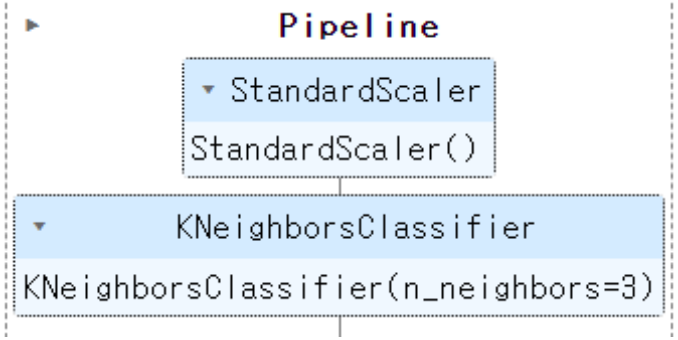
```
# KNN
model = Pipeline([("scaling", StandardScaler()), ("modeling", KNeighborsClassifier(n_neighbors=5))]).fit(X_train, y_train)

cv_scores = cross_val_score(estimator= model, X = X_train, y = y_train, scoring= "accuracy")
```

5-fold cross validation







KNN Model #1
Test_size = 0.3
StandardScaler()
N_neighbors = 5

KNN Model #2
Test_size = 0.2
StandardScaler()
N_neighbors = 5

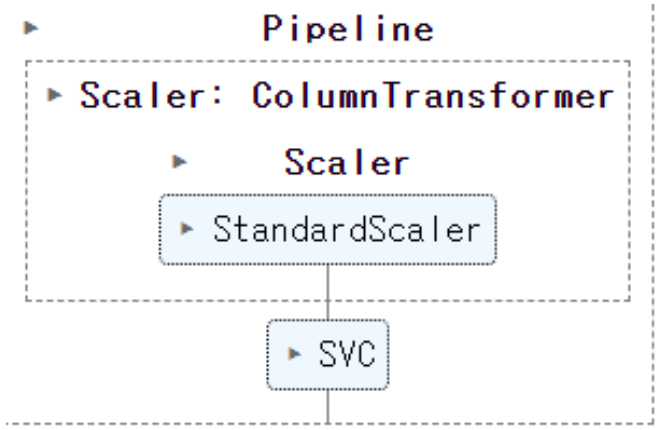
KNN Model #3
Test_size = 0.3
MinMaxScaler()
N_neighbors = 5

KNN Model #4
Test_size = 0.3
StandardScaler()
N_neighbors = 3

Define & Fit models (2) : SVM

```
tf2_lr = ColumnTransformer([('Scaler', StandardScaler(), slice(0,9))]) ##
svm = SVC(kernel='linear')
pipeline = Pipeline([('Scaler',tf2_lr),('Support Vector Machine', svm)])

pipeline.fit(X_train, y_train)
```

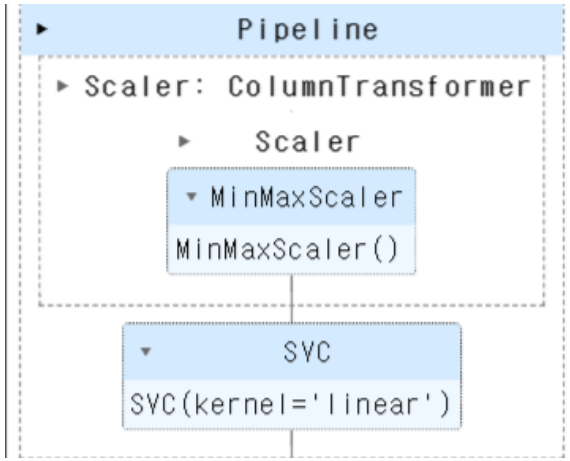


SVM Model #1
Test_size = 0.3
StandardScaler()
Kernel = 'linear'

```
tf2_lr = ColumnTransformer([('Scaler', MinMaxScaler(), slice(0, 9))])
svm = SVC(kernel='linear')
pipeline = Pipeline([('Scaler', tf2_lr), ('Support Vector Machine', svm)])

cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

# Perform cross-validation and get predicted values for each fold
y_pred_cv = cross_val_predict(pipeline, X_train, y_train, cv=cv)
```



SVM Model #2
Test_size = 0.3
MinMaxScaler()
Kernel = 'linear'
3-fold cross-validation

Performance Results (1) : KNN

```
print(f"Mean accuracy cv: {np.mean(cv_scores)}")

model.score(X_test, y_test)
```

Mean Accuracy cv :

0.835

class	precision	recall	f1-score	support
0	0.86	0.83	0.84	21127
1	0.81	0.85	0.83	23221
2	0.81	0.77	0.79	10586
3	0.88	0.90	0.89	7445
accuracy			0.83	62379
macro avg	0.84	0.84	0.84	62379
weighted avg	0.83	0.83	0.83	62379

KNN Model #1

0.814

precision	recall	f1-score	support
0.79	0.77	0.78	595
0.76	0.80	0.78	797
0.88	0.87	0.88	947
0.71	0.65	0.68	69
		0.81	2408
0.79	0.77	0.78	2408
0.82	0.81	0.81	2408

KNN Model #2

0.843

Overall high f1-score

precision	recall	f1-score	support
0.87	0.84	0.85	21127
0.81	0.86	0.83	23221
0.82	0.78	0.80	10586
0.90	0.91	0.90	7445
		0.84	62379
0.85	0.85	0.85	62379
0.84	0.84	0.84	62379

KNN Model #3

0.825

precision	recall	f1-score	support
0.84	0.83	0.84	21127
0.81	0.83	0.82	23221
0.79	0.77	0.78	10586
0.88	0.90	0.89	7445
		0.83	62379
0.83	0.83	0.83	62379
0.83	0.83	0.83	62379

KNN Model #4

Performance results (2) : SVM

SVM Model #1

Accuracy: 0.833

	precision	recall	f1-score	support
0	0.85	0.83	0.84	21127
1	0.80	0.83	0.82	23221
2	0.82	0.77	0.79	10586
3	0.90	0.92	0.91	7445
accuracy			0.83	62379
macro avg	0.84	0.84	0.84	62379
weighted avg	0.83	0.83	0.83	62379

Output Ex :

```
y_pred_cv, len(y_pred_cv)
```

```
(array([1, 3, 3, ..., 1, 0, 1]), 145550)
```

SVM Model #2

Accuracy: 0.834
F1-score (macro): 0.84
F1-score (weighted) : 0.83

```
[19] from sklearn.metrics import accuracy_score, f1_score
     f1_score_cv = f1_score(y_train, y_pred_cv, average='weighted')


     print("Cross-Validated F1-Score:", f1_score_cv)
```

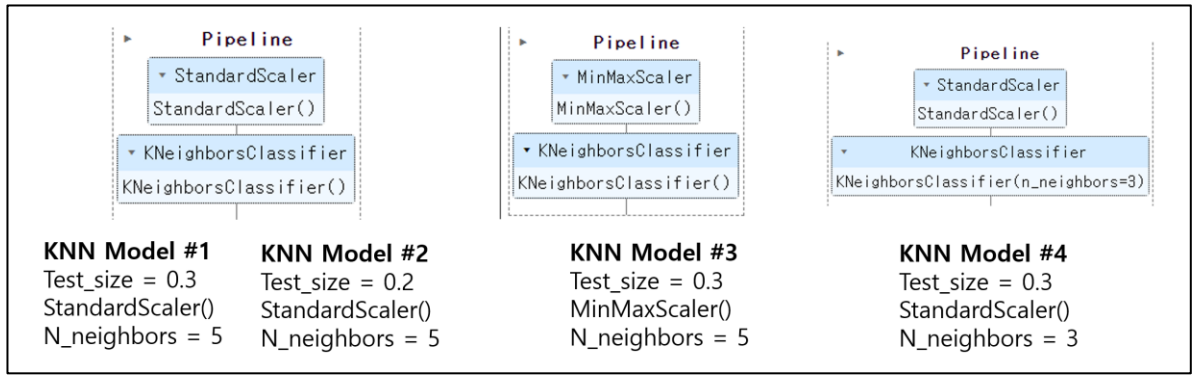
```
     Cross-Validated F1-Score: 0.8341504931969845
```

```
[21] macro_score = f1_score(y_train, y_pred_cv, average='macro')
     print('macro =', macro_score)
```

```
     macro = 0.8417062164867213
```

Performance Results

Among the #**4 KNN models**, the Model #3 had the best quality. (acc = 84%, f1-score 0.83-0.90). It shows that scaling with MinMaxScaler and looking 5 neighbors around the sample is the best situation in this case to get the best output.
Furthermore, in the **SVM models**, those two models gave similar performance, acc as 83%, f1-score as 0.84.

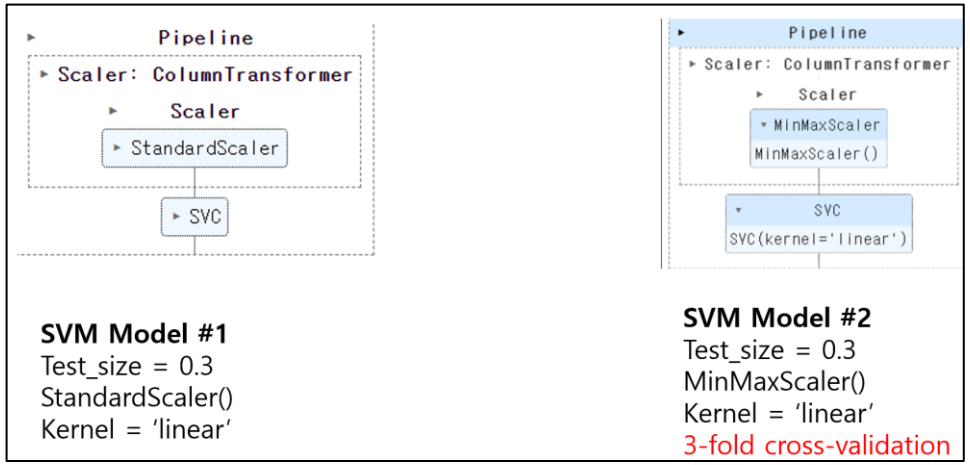


KNN Model #1
Test_size = 0.3
StandardScaler()
N_neighbors = 5

KNN Model #2
Test_size = 0.2
StandardScaler()
N_neighbors = 5

KNN Model #3
Test_size = 0.3
MinMaxScaler()
N_neighbors = 5

KNN Model #4
Test_size = 0.3
StandardScaler()
N_neighbors = 3



SVM Model #1
Test_size = 0.3
StandardScaler()
Kernel = 'linear'

SVM Model #2
Test_size = 0.3
MinMaxScaler()
Kernel = 'linear'
3-fold cross-validation

Discussion & Conclusion

Our task was related to supervised-learning, and the goal was classification of songs into four emotions. In the process, we firstly done some data preprocessing (or EDA) to look on and preprocess data. And we used two types of models, KNN and SVM. KNN was an easy and useful model as it takes short time for training but great performance. Also, in spite of long time fitting in SVM, it made meaningful output.

To find the best model, we needed multiple times of change in hyperparameters. Specifically, we've changed "k"-fold, scaling method, etc. In order to increase the quality of model, cross-validation(such as StratifiedKFold) was also implemented.

We might have better models in the future research if we spend more time on hyperparameter tuning and data preprocessing. (e.g. k of k-fold, scaling, etc)

Thank you

Soyeon Kim, Yejin Kim, Yoojin Oh
Ewha Womans University, Dept of Artificial Intelligence