Multiple Linear Regression Model

Collins Tse Jacky Ke Rebecca Bachtra Christy Yau

Thursday 18th December, 2025

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
import sklearn.linear_model as lm
from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant
from sklearn.preprocessing import StandardScaler

# function for computing 10 -fold CV error
def compute_CV_error(model, X_train, Y_train):
    kf = KFold(n_splits=10)
    validation_errors = []

    for train_idx, valid_idx in kf.split(X_train):
        split_X_train, split_X_valid = X_train.iloc[train_idx], X_train.iloc[valid_idx]
        split_Y_train, split_Y_valid = Y_train.iloc[train_idx], Y_train.iloc[valid_idx]
        model.fit(split_X_train, split_Y_train)
        error = np.sqrt(mean_squared_error(split_Y_valid, model.predict(split_X_valid)))
        validation_errors.append(error)

    return np.mean(validation_errors)
```

0.1 Fitting Linear Regression Model

We filter our data to only include quantitative features that can be used to build a model for predicting a song's popularity. We then split the filtered data into training and test set. We split out 10% of the data for the test set and train our model on the training set. From the training set, we set 10% of the data aside as a validation set for evaluating our model performance. We use this model to predict the popularity score from data on the test set and find the RMSE.

```
# keep only quantitative, non -NA feautres
df = pd.read_csv("data/processed/spotify_clean.csv")
remove_col = ['track_id', 'artists', 'album_name', 'track_name', 'explicit', 'track_genre',
filtered_df = df.drop(remove_col, axis = 1)
filtered_df.head()
```

	popularity	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature	duration_min
0	73	0.676	0.461	1	-6.746	0	0.143	0.0322	0.000001	0.358	0.715	87.917	4	3.844433
1	55	0.420	0.166	1	-17.235	1	0.0763	0.924	0.000006	0.101	0.267	77.489	4	2.493500
2	57	0.438	0.359	0	-9.734	1	0.0557	0.210	0.000000	0.117	0.120	76.332	4	3.513767
3	71	0.266	0.0596	0	-18.515	1	0.0363	0.905	0.000071	0.132	0.143	181.740	3	3.365550
4	82	0.618	0.443	2	-9.681	1	0.0526	0.469	0.000000	0.0829	0.167	119.949	4	3.314217

```
# train/test split
feature_df = filtered_df.drop(['popularity'], axis = 1)
X = feature_df
Y = filtered_df['popularity']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1, random_state = 42)
X_train, X_validation, Y_train, Y_validation = train_test_split(X_train, Y_train, random_sta

# multiple linear regression model
linear_model = lm.LinearRegression()
linear_model.fit(X_train, Y_train)

# test predictions & error
train_rmse = np.sqrt(mean_squared_error(Y_train, linear_model.predict(X_train)))
validation_rmse = np.sqrt(mean_squared_error(Y_validation, linear_model.predict(X_validatio
cv_error = compute_CV_error(linear_model, X_validation, Y_validation)
print("Train RMSE:", train_rmse)
print("Validation RMSE:", validation_rmse)
print("CV Error:", cv_error)
```

```
Train RMSE: 20.28688052621643
Validation RMSE: 20.256848972124786
CV Error: 20.2629301918764
```

0.2 Checking Collinearity (VIF)

Using the variance inflation factor (VIF), we check for collinearity between all features and then perform regularization to reduce overfitting and allow for a more generalized model.

```
# Variance Inflation Factor (VIF)
X = add_constant(feature_df)

# Compute VIF for each column
vif = pd.DataFrame()
vif["feature"] = X.columns
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

print(vif)
vif.to_csv('results/regression_vif.csv')
```

```
feature         VIF
0              const  170.662614
1       danceability    1.565985
2             energy    4.261457
3                key    1.022827
4           loudness    3.269276
5               mode    1.041646
6        speechiness    1.146349
7       acousticness    2.417361
8   instrumentalness    1.470513
9           liveness    1.158525
10           valence    1.600743
11             tempo    1.096353
12    time_signature    1.082554
13      duration_min    1.052097
```

Energy and loudness show moderate collinearity but have a VIF >5, so we decided to keep these features.

1 Regression Model with LASSO Regularization

1.1 Normalization

Before performing regularization, we need to normalize our features so they are on the same scale. Then we regenerate the training and test sets using this new rescaled data and find an optimum regularization hyperparamter using 4-fold CV.

```
ss = StandardScaler()
ss.fit(feature_df)
```

```
features_scaled = pd.DataFrame(ss.transform(feature_df), columns = feature_df.columns)
features_scaled.head()
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature	duration_min
0	0.644260	-0.675972	-1.203286	0.335731	-1.324600	0.490464	-0.875177	-0.535478	0.723669	0.634036	-1.133609	0.226215	0.513495
1	-0.804610	-1.425610	-0.903218	-1.673094	0.754945	-0.098361	1.760797	-0.535465	-0.659507	-0.772028	-1.079854	0.226215	-0.704151
2	-0.702713	-1.307347	-1.468410	-1.436523	0.754945	-0.280213	-1.749638	-0.535485	-1.129173	-1.329510	-0.518271	0.226215	-0.162163
3	-1.676128	-1.824025	-1.478419	-1.418236	0.754945	-0.451480	1.704637	-0.535263	-1.643360	-0.242010	1.981637	-1.979187	-0.240899
4	0.316001	-0.746129	-0.923077	-0.726373	0.754945	-0.307584	0.415912	-0.535486	-0.687914	-1.850700	-0.870037	0.226215	-0.268168

1.2 Fit Model with Ridge

```
# train/test split on normalized data
X2 = features_scaled
X_train2, X_test2, Y_train2, Y_test2 = train_test_split(X2, Y, test_size = 0.10, random_stat
X_train2, X_validation2, Y_train2, Y_validation2 = train_test_split(X_train2, Y_train2, ranc

# find optimal hyperparamters with CV
lambdas = 10**np.linspace( -5, 4, 40)
ridge_cv = RidgeCV(alphas = lambdas, cv=10)
ridge_cv.fit(X_train2, Y_train2)

print("Best hyperparameter:", ridge_cv.alpha_)
```

```
Best hyperparameter: 242.44620170823308
```

```
# fit model
ridge_model = Ridge(alpha = ridge_cv.alpha_)
ridge_model.fit(X_train2, Y_train2)

# prediction error
ridge_train_rmse = np.sqrt(mean_squared_error(Y_train2, ridge_model.predict(X_train2)))
ridge_validation_rmse = np.sqrt(mean_squared_error(Y_validation2, ridge_model.predict(X_vali
ridge_cv_error = compute_CV_error(ridge_model, X_validation2, Y_validation2)
print("Train RMSE:", ridge_train_rmse)
print("Validation RMSE:", ridge_validation_rmse)
print("CV Error:", ridge_cv_error)
```

```
Train RMSE: 20.286895340842147
Validation RMSE: 20.256703572773905
CV Error: 20.262751315493187
```

2 Regression Model with LASSO Regularization

2.1 Fit Model with LASSO

```
# find optimal hyperparameter with CV
lasso_cv = LassoCV(alphas = lambdas, cv=4)
lasso_cv.fit(X_train2, Y_train2)

print("Best hyperparameter:", lasso_cv.alpha_)
```

```
Best hyperparameter: 0.0058780160722749115
```

```
# fit model
lasso_model = Lasso(alpha = lasso_cv.alpha_)
lasso_model.fit(X_train2, Y_train2)

# prediction error
lasso_train_rmse = np.sqrt(mean_squared_error(Y_train2, lasso_model.predict(X_train2)))
lasso_validation_rmse = np.sqrt(mean_squared_error(Y_validation2, lasso_model.predict(X_vali
lasso_cv_error = compute_CV_error(lasso_model, X_validation2, Y_validation2)
print("Train RMSE:", lasso_train_rmse)
print("Validation RMSE:", lasso_validation_rmse)
print("CV Error:", lasso_cv_error)
```

```
Train RMSE: 20.28690280094442
Validation RMSE: 20.256791215739355
CV Error: 20.26282675002194
```

3 Compare Models

```
models = pd.DataFrame({"Model": ['Multiple Linear Reg', '+ Ridge', "+ LASSO"],
                       "Train RMSE": [train_rmse, ridge_train_rmse, lasso_train_rmse],
                       "Validation RMSE": [validation_rmse, ridge_validation_rmse, lasso_val
                       "CV Error": [cv_error, ridge_cv_error, lasso_cv_error]})
models.to_csv('results/mlr_models_comparison.csv')
models
```

	Model	Train RMSE	Validation RMSE	CV Error
0	Multiple Linear Reg	20.286881	20.256849	20.262930
1	+ Ridge	20.286895	20.256704	20.262751
2	+ LASSO	20.286903	20.256791	20.262827

Although the differences are marginal, the multiple linear regression model with ridge regularization has the lowest CV error.

4 Evaluating Model Performance

We evaluate our model's perfomance on unseen data by testing our model on the test set set aside in the beginning.

```
chosen_model_rmse = np.sqrt(mean_squared_error(Y_test2, ridge_model.predict(X_test2)))
chosen_model_cv_error = compute_CV_error(lasso_model, X_test2, Y_test2)
print("Model RMSE:", chosen_model_rmse)
print("Model CV Error:", chosen_model_cv_error)
print("Model Coefficients:", ridge_cv.coef_)
```

```
Model RMSE: 20.030353435003935
Model CV Error: 20.031258034921237
Model Coefficients: [ 1.72511138 -0.55015578 -0.05363698  0.27393151 -0.38793665 -1.3783287
 -0.62505565 -2.96602234  0.11761864 -2.20657216  0.15295424  0.36485984
 -0.43688028]
```

4.1 Interpreting Coefficients

```
model_coeff = pd.DataFrame({'Features': feature_df.columns,
                            'Coeff': ridge_cv.coef_})
model_coeff
```

	Features	Coeff
0	danceability	1.725111
1	energy	-0.550156
2	key	-0.053637
3	loudness	0.273932
4	mode	-0.387937
5	speechiness	-1.378329
6	acousticness	-0.625056
7	instrumentalness	-2.966022
8	liveness	0.117619
9	valence	-2.206572
10	tempo	0.152954
11	time_signature	0.364860
12	duration_min	-0.436880

Danceability, loudness, liveness, and tempo have positive coefficients, meaning that these features increase the predicted popularity of a song. In other words, songs that are easier to dance to, louder, more lively, and faster are predicted to be more popular. The remaining features have negative coefficients, meaning that they decrease the predicted popularity of a song. The feature with the smallest coefficient is 'key', indicating that it has little impact on popularity. The feature with the largest positive and negative coefficient are 'danceability' and ‘instrumentalness’ respectively, indicating that these features have a relatively large influence on popularity. We can interpret the coefficients as follows: A 1-unit increase in *feature* (holding all other features constant) increases the predicted popularity of a song by approximately *feature coefficient*. For example, a 1-unit increase in danceability (holding all other features constant) increases the predicted popularity by approximately 1.73.