CS985 - MACHINE LEARNING FOR DATA ANALYTICS

Assignment Report - Regression

<u>**CS985MLDAGroup9**</u>

Members:

HARSHAN RETHINAVELU SELVAKUMAR - 202480548

MANOJ KUMAR DHARMARAJ - 202468855

JHANSI VELURI - 202384615

PRATHUSHA PUNJARLA - 202351330

MAISAM BARKAT ALI DOHTA - 202359049

Problem Statement:

The aim is to develop a machine learning model to predict the popularity of songs based on the various attributes given in the dataset. The train and test datasets are provided. The main objective is to build a regression model that can predict the popularity of songs using the given attributes while minimizing the Root Mean Squared Error (RMSE).

Libraries and modules used:

1. **Numpy** - It is used for numerical calculations. This library is very much convenient in handling complex mathematical operations.
2. **Pandas** - It is used for the analysis of the data. Most commonly used for data preprocessing and it provides data structures like DataFrame.
3. **Matplotlib** - It is used for data visualizations. This library is used for creating histograms, scatterplots, etc.

4. **Sklearn.model_selection** - It consists of various functions for splitting the datasets, model evaluation, etc. 'train_test_split' is a function within this module which is used for splitting a dataset into two subsets.
5. **Sklearn.preprocessing** - It is a module within the 'scikit-learn' library which consists of functions. These functions perform pre-processing data before training the models. It includes various techniques such as Feature Scaling, encoding categorical variables, etc. 'StandardScaler' is a class that is used to transform features to have a mean of 0 and a standard deviation of 1.
6. **Sklearn.linear_model** - It is a module within 'scikit-learn' library which provides techniques for fitting the linear models into the data. Linear Regression is a fundamental regression algorithm provided by this module.
7. **Sklearn.metrics** – It is a module within 'scikit-learn' library which is used for assessing the performance of the model. It provides various evaluation metrics such as Mean Squared Error (MSE), R-Squared score, etc. 'mean_squared_error' is a function that calculates the Mean Squared Error (MSE) between the target values and predicted values. The root of MSE is called as Root Mean Squared Error (RMSE).

Code with Explanation:

Import Libraries:

```
import numpy as np  # numpy library is imported and an alias 'np' is assigned to the library

import pandas as pd  # pandas library is imported and an alias 'pd' is assigned to the library

import os   # os library is imported

    for dirname, _, filenames in os.walk('/kaggle/input'):   # Initialization of a loop that iterates over the directory and the file mentioned
       for filename in filenames:  # Iterates over each and every filename in the directory
          print(os.path.join(dirname, filename))  # Prints the full path of each file
```

Read the Train and Test Datasets:

spotify_tr = pd.read_csv("/kaggle/input/cs9856-spotify-regression-problem-2024/CS98XRegressionTrain.csv") **# Reads 'CS98XRegressionTrain.csv' file from the directory to pandas dataframe named 'spotify_tr'**

spotify_tr **# Displays the dataframe**

spotify_te = pd.read_csv("/kaggle/input/cs9856-spotify-regression-problem-2024/CS98XRegressionTest.csv") **# Reads 'CS98XRegressionTest.csv' file from the directory to pandas dataframe named 'spotify_te'**

spotify_te.head(2) **# Displays the first two rows of the dataframe**

Data Preprocessing:

spotify_tr = spotify_tr.dropna() **# Removes the rows with missing values and assigns the result back to 'spotify_tr'**

spotify_tr.columns **# Displays the list of columns in the dataframe**

columns = ['bpm','nrgy','dnce','dB','live','val','dur','acous','spch','pop'] **# Displays the list of columns to be selected from the dataframe 'spotify_tr'**

spotify_tr = spotify_tr[columns] **# Assigns only the selected columns in the list 'Columns' to the dataframe 'spotify_tr'**

spotify_tr.head() **# Displays the first five rows of the dataframe**

spotify_tr.shape **# Returns a tuple consisting of the number of rows and columns in the dataframe**

import matplotlib.pyplot as plt **# matplotlib.pyplot is imported and given an alias 'plt'**

%matplotlib inline **# Checks whether the plots are shown inline within the cells**

spotify_tr.hist(bins=50,figsize=(8,10)) **# Generates histograms for numerical attributes in the dataframe. 'bins' defines the intervals to be used and 'figsize' defines the size of the histogram**

plt.show() **# Displays the generated histograms**

x = spotify_tr.drop('pop',axis=1) **# Creates a new dataframe 'x' where the column 'pop' is dropped and all other columns of the dataframe 'spotify.tr' are stored**

y = spotify_tr['pop'] **# Creates a new dataframe 'y' where the column 'pop' is stored**

x **# Displays the dataframe 'x'**

y **# Displays the dataframe 'y'**

Splitting of data

from sklearn.model_selection import train_test_split **# Imports train_test_split function of 'model_selection' module from the 'scikit-learn' library**

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,random_state = 40) **# 'train_test_split' function splits 'x' and 'y' into training and testing sets where 'x' is the the feature and 'y' is the target variable. test_size = 0.2 means that 20% of data will be used for testing and the remaining 80% data will be used for training. random_state will produce the same results everytime when running the code**

print(x_train.shape) **# Displays a tuple consisting of the number of rows and columns in the training set**

print(x_test.shape) **# Displays a tuple consisting of the number of rows and columns in the testing set**

Feature Scaling:

from sklearn.preprocessing import StandardScaler **# Imports StandardScaler from the preprocessing module of scikit-learn library**

scaler = StandardScaler() **# Creates an instance of StandardScaler class**

scaler.fit(x_train) **# Calculates the mean and standard deviation of each feature in the training data 'x_train' after fitting the scaler to the fitting data**

x_train_scaled = scaler.transform(x_train**)** **# Transforms the training data 'x_train' using the fitted scaler**

x_test_scaled = scaler.transform(x_test) **# Transforms the testing data 'x_test' using the fitted scaler**

 # Fit the data once and transform

x_test_scaled = pd.DataFrame(x_test_scaled, columns=['bpm','nrgy','dnce','dB','live','val','dur','acous','spch']) **# Converts the scaled test data into pandas dataframe and columns are assigned to it**

x_test_scaled

x_train_scaled = pd.DataFrame(x_train_scaled, columns=['bpm','nrgy','dnce','dB','live','val','dur','acous','spch']) **# Converts the scaled training data into pandas dataframe and columns are assigned to it**

x_train_scaled **# Displays the dataframe**

Model Training:

#Linear Regression

from sklearn.linear_model import LinearRegression **# Imports LinearRegression from the linear_model module of scikit-learn library**

lin_model = LinearRegression() **# Creates an instance of LinearRegression class** lin_model.fit(x_train_scaled,y_train) **# Fits the Linear Regression model to the 'x_train_scaled' and 'y_train' where 'x_train_scaled' is the scaled training data and 'y_train' is the target variable**

lin_model.intercept_ **# Gives the y-intercept of the regression line**

y_preds = lin_model.predict(x_test_scaled) **# The trained linear model 'lin_model' is applied to the scaled test data 'x_test_scaled' to make predictions for the target variable**

y_preds **# Displays the predicted values of the target variable**

from sklearn.metrics import mean_squared_error **# Imports mean_squared_error from the metrics module of scikit-learn library**

lin_rmse = mean_squared_error(y_test, y_preds, squared=False) **# Calculates RMSE between 'y_test' and 'y_preds'. (RMSE = √MSE)**

lin_rmse **# Displays the RMSE Value**

len(y_preds) **# Displays the length of 'y_preds'**

#Random Forest Regression

from sklearn.ensemble import RandomForestRegressor **# Imports RandomForestRegressor from ensemble module of scikit-learn library**

rfr_model = RandomForestRegressor(random_state=1) **# Creates an instance of the RandomForestRegressor class**

rfr_model.fit(x_train_scaled, y_train) **# Trains the Random forest regression model 'rfr_model' using scaled training data 'x_train_scaled' and target values 'y_train'**

rfr_pred = rfr_model.predict(x_test_scaled) **# Make predictions on the scaled test data 'x_test_scaled' using trained Random forest regression model 'rfr_model'**

rfr_pred

rfr_mse = mean_squared_error(y_test,rfr_pred) **# Calculates the mean squared error between 'y_test' and 'rfr_pred'**

rfr_rmse = np.sqrt(rfr_mse) **# RMSE = √MSE**

rfr_rmse **# Displays the RMSE value**

spotify_test_scaled = scaler.transform(spotify_te[['bpm','nrgy','dnce','dB','live','val','dur','acous','spch']]) **# Transforms the test data using the fitted scaler**

spotify_test_scaled = pd.DataFrame(spotify_test_scaled, columns=['bpm','nrgy','dnce','dB','live','val','dur','acous','spch']) **# Converts the scaled test data to pandas dataframe and column names are assigned to it**

spotify_test_scaled

Choosing the best Model

y_preds_csv = rfr_model.predict(spotify_test_scaled) **# The trained Random Forest Regressor model is applied to 'spotify_test_scaled' to make predictions**

y_preds_csv **# Displays the predicted values**

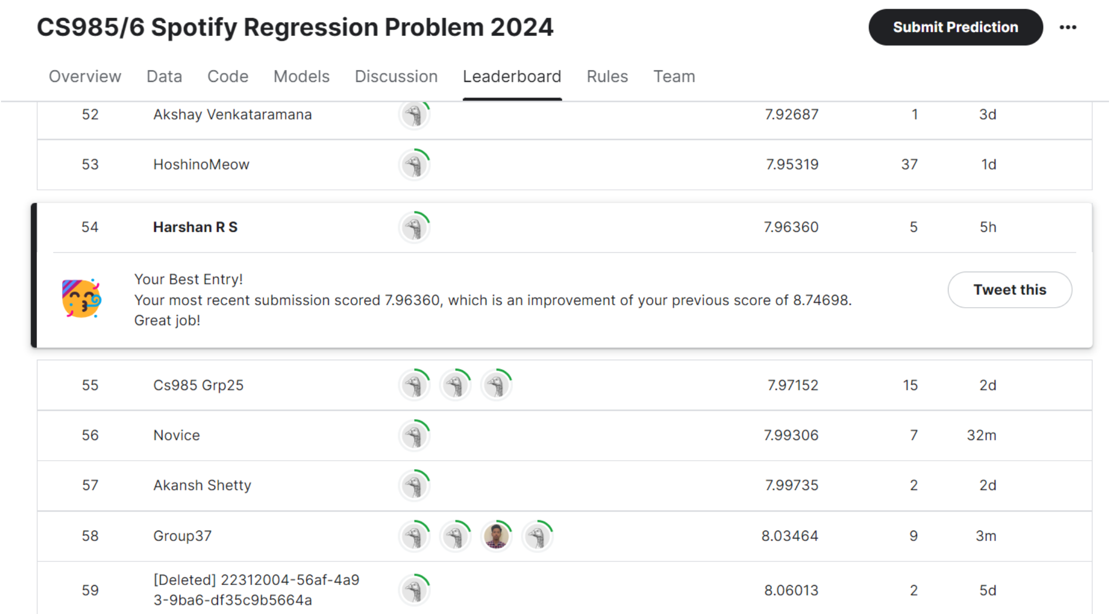
submission = pd.DataFrame({'Id': spotify_te.iloc[:,0], 'pop': y_preds_csv }) **# Creates a dataframe named 'submission' consisting of two columns. The first column is the 'Id' which contains the Ids from test data and it is taken from the first column of 'spotify_te'. The second column consists of the predicted popularity values 'y_preds_csv'**

submission.to_csv('submission.csv', index=False) **# Saves the 'submission' dataframe to CSV file named 'submission.csv'**

Comparison of Model Performance (What worked and what didn't):

In this study, we evaluate the performance of linear regression and random forest regression models for regression. We calculated the Root Mean Squared Error (RMSE) values for both models. The Calculations reveal that the RMSE of the Random Forest Regression model is consistently lower than that of the linear regression model which indicates superior performance. Overall, the lower RMSE of Random Forest Regression compared to Linear Regression suggests that Random Forest Regression is better able to capture the underlying patterns in the data and make more accurate predictions on the test data.

Kaggle Performance:


CS985/6 Spotify Regression Problem 2024

Submit Prediction

Overview Data Code Models Discussion Leaderboard Rules Team

52	Akshay Venkataramana	7.92687	1	3d
53	HoshinoMeow	7.95319	37	1d
54	**Harshan R S**	7.96360	5	5h
55	Cs985 Grp25	7.97152	15	2d
56	Novice	7.99306	7	32m
57	Akansh Shetty	7.99735	2	2d
58	Group37	8.03464	9	3m
59	[Deleted] 22312004-56af-4a93-9ba6-df35c9b5664a	8.06013	2	5d

Your Best Entry!
Your most recent submission scored 7.96360, which is an improvement of your previous score of 8.74698.
Great job!

Tweet this

Conclusion:

In conclusion, we are getting an RMSE value of Random Forest Regression model is 10.814271032045824 which is comparatively lesser than that of Linear Regression Model. Hence, the final predictions are made with the Random Forest regression model.