Department of Mathematics and Statistics

MTH416 Project Report

Music Launch Era Prediction

Anurag Pandey 160140
Prateek Gothwal 160505
Sandarsh Gupta 160685

12th April 2019

Introduction

For this Project, we have used the Million Song Dataset and fitted a multiple linear regression model on this data and used this model to predict the era (year) of release of the songs in the test dataset. For this we have first cleaned and prepared the data, then we have fitted a Linear Regression Model. We then perform variable selection by the methods of Principal Component Analysis. We thus obtain the significant regressors, and check the model using Akaike Information Criterion.

The Dataset

The Dataset we have used is a subset of the Million Song Dataset from Columbia University, New York. This subset consists of 515345 records of songs that were composed during the years 1922 to 2011, both included. Each row in the dataset consists of 91 features. The first feature is the year in which the song was released, and the remaining 90 features are various quantities (float) related to the analysis of song's audio, generated by the Echo Nest. As recommended by database authors, we have used a training set of first 463,715 examples (the values for which we fit the model) and a test set of last 51,630 examples (the values for which we predict the release era using the model). Also it is important to point out that the dataset is highly skewed and fitting a linear regression model will be a challenge. Deep neural networks are the suggested approach for this kind of data. We chose this data to get a hands-on experience of the extent to which linear regression model can produce results. It also helps us realize the limitations of this form of regression as compared to complex neural-networks, the significance of its result.

Approach

Multiple Linear Regression

In statistics, linear regression is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). For more than one explanatory variable, the process is called multiple linear regression. For the initial approach, we fitted the model using all the regressors as shown by the following matrix representation:

$$Y = X\beta \tag{1}$$

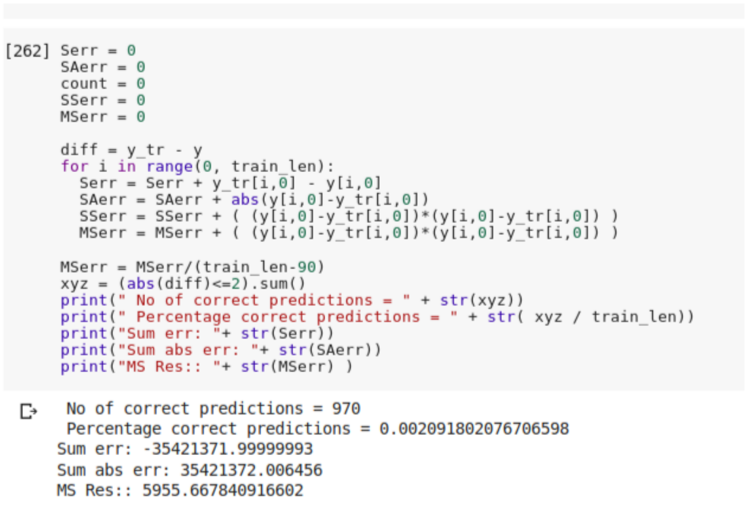
where $\boldsymbol{Y}$ is a vector of dimensions 463,715*1, having year for corresponding data entry , $\boldsymbol{X}$ is a matrix of dimensions 463,715*90, having data entries for years in $\boldsymbol{Y}$, $\boldsymbol{\beta}$ is a vector of dimensions 91*1, having 90 regressors and 1 intercept term.

We obtained $\boldsymbol{\beta}$ using the following equation,

$$\beta = (X^T X)^{-1} X^T Y \tag{2}$$

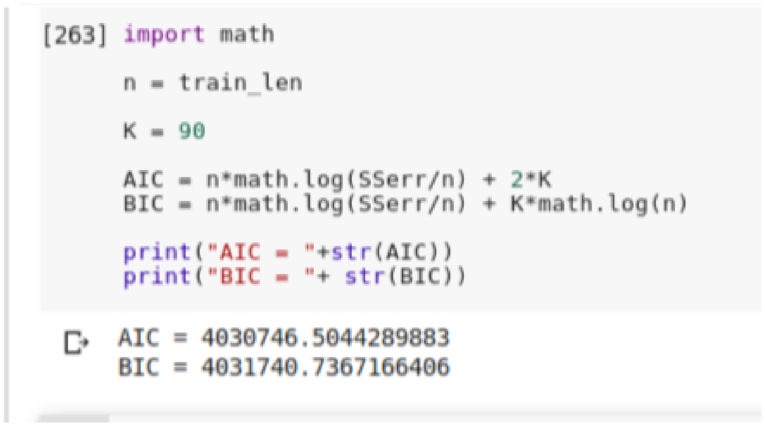
This simple model fitting gave unsatisfactory results. This is evident from the values of MSres amd AIC for the same. The result images are given below. Thus, we quickly switched to Ridge Regression Model.

Result for Single Dimensional output value



```
[262] Serr = 0
      SAerr = 0
      count = 0
      SSerr = 0
      MSerr = 0

      diff = y_tr - y
      for i in range(0, train_len):
        Serr = Serr + y_tr[i,0] - y[i,0]
        SAerr = SAerr + abs(y[i,0]-y_tr[i,0])
        SSerr = SSerr + ( (y[i,0]-y_tr[i,0])*(y[i,0]-y_tr[i,0]) )
        MSerr = MSerr + ( (y[i,0]-y_tr[i,0])*(y[i,0]-y_tr[i,0]) )

      MSerr = MSerr/(train_len-90)
      xyz = (abs(diff)<=2).sum()
      print(" No of correct predictions = " + str(xyz))
      print(" Percentage correct predictions = " + str( xyz / train_len))
      print("Sum err: "+ str(Serr))
      print("Sum abs err: "+ str(SAerr))
      print("MS Res:: "+ str(MSerr) )
```

```
 No of correct predictions = 970
 Percentage correct predictions = 0.002091802076706598
Sum err: -35421371.99999993
Sum abs err: 35421372.006456
MS Res:: 5955.667840916602
```



```
[263] import math

      n = train_len

      K = 90

      AIC = n*math.log(SSerr/n) + 2*K
      BIC = n*math.log(SSerr/n) + K*math.log(n)

      print("AIC = "+str(AIC))
      print("BIC = "+ str(BIC))
```

```
AIC = 4030746.5044289883
BIC = 4031740.7367166406
```

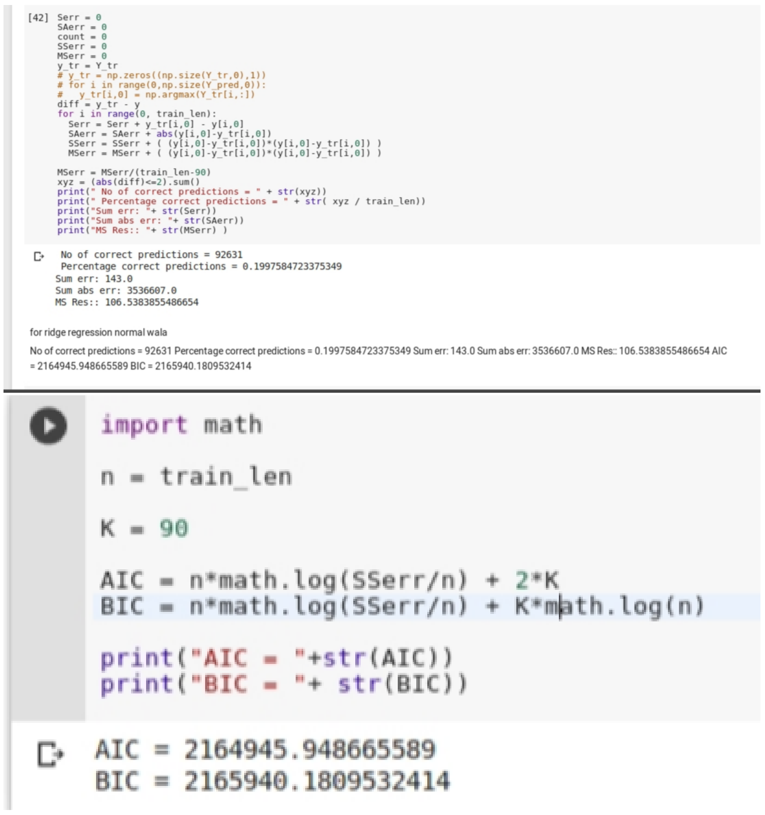
Ridge Regression

The method of least squares on our data gives very poor estimates of the regression coefficients and in sample predictions. To overcome this we use the **Ridge Regression** model. The ridge estimator is found by solving a slightly modified version of the normal equations. Specifically we define the ridge estimator β_R as the solution to

$$\boldsymbol{X^T Y} = (\boldsymbol{X^T X} + k\boldsymbol{I})\boldsymbol{\beta_R} \tag{3}$$

When implementing this on the given dataset, we observe significant improvement in the in-sample predictions, which is evident from the new MSres value in this case, which is very less as compared to the previous case. We also observe that the AIC value reduces to half of that in the previous case. The code for the same and the results are shown below:

Result of Ridge Regression



```
[42] Serr = 0
     SAerr = 0
     count = 0
     SSerr = 0
     MSerr = 0
     y_tr = Y_tr
     # y_tr = np.zeros((np.size(Y_tr,0),1))
     # for i in range(0,np.size(Y_pred,0)):
     #   y_tr[i,0] = np.argmax(Y_tr[i,:])
     diff = y_tr - y
     for i in range(0, train_len):
       Serr = Serr + y_tr[i,0] - y[i,0]
       SAerr = SAerr + abs(y[i,0]-y_tr[i,0])
       SSerr = SSerr + ( (y[i,0]-y_tr[i,0])*(y[i,0]-y_tr[i,0]) )
       MSerr = MSerr + ( (y[i,0]-y_tr[i,0])*(y[i,0]-y_tr[i,0]) )

     MSerr = MSerr/(train_len-90)
     xyz = (abs(diff)<=2).sum()
     print(" No of correct predictions = " + str(xyz))
     print(" Percentage correct predictions = " + str( xyz / train_len))
     print("Sum err: "+ str(Serr))
     print("Sum abs err: "+ str(SAerr))
     print("MS Res:: "+ str(MSerr) )
```

```
 No of correct predictions = 92631
 Percentage correct predictions = 0.1997584723375349
Sum err: 143.0
Sum abs err: 3536607.0
MS Res:: 106.5383855486654
```

for ridge regression normal wala

No of correct predictions = 92631 Percentage correct predictions = 0.1997584723375349 Sum err: 143.0 Sum abs err: 3536607.0 MS Res:: 106.5383855486654 AIC = 2164945.948665589 BIC = 2165940.1809532414

```
import math

n = train_len

K = 90

AIC = n*math.log(SSerr/n) + 2*K
BIC = n*math.log(SSerr/n) + K*math.log(n)

print("AIC = "+str(AIC))
print("BIC = "+ str(BIC))
```

```
AIC = 2164945.948665589
BIC = 2165940.1809532414
```

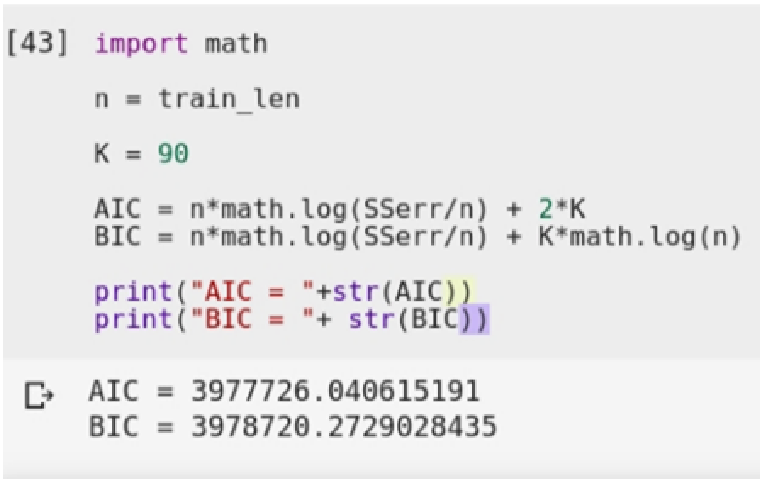
Applying Variable Selection on the obtained model

For further improvement, we applied variable selection method. Principal Component Analysis (PCA) implemented on this does not provide any significant improvement. In fact, the best result was obtained with considering all the 90 regressors in the model.

Other Methods tried:

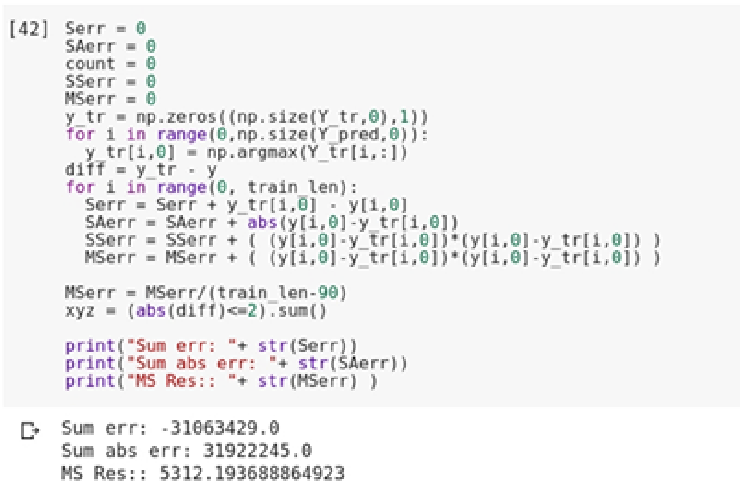
As the year to be predicted is discrete data, one of out of 90 classes (here year), we tried to provide a soft decision boundary for assigning a particular class to an input data. For this, we implemented a Multiple Linear Regression on the data taking the output year as a ninety-dimensional vector. The formula is the same as was used before in previous section. Each elementary vector of this 90 dimensional space is used to represent one of the year in the output range. Thus, the year entries in the data are changed to vectors having entries as 0 and 1. We applied Linear Regression on this new Y matrix. The Y_pred obtained is a ninety dimensional vector score. Each entry (dimension) of the vector represents its closeness to corresponding year. For example, if a vector is one-hot encoding with first dimension as one and rest zero, then the output is 1922. We calculate output 90 dimensional vectors for each test data entry, and measure its closeness with each of the vectors of the 90 years. This gives the closest vector of these 90 vectors and the year for the release of music.
The result obtained were more or less same. So, we proceeded with Ridge Regression on this model. The results for the same are:

Result of Ridge Regression with Multi-Dimensional output value



```
[43] import math

     n = train_len

     K = 90

     AIC = n*math.log(SSerr/n) + 2*K
     BIC = n*math.log(SSerr/n) + K*math.log(n)

     print("AIC = "+str(AIC))
     print("BIC = "+ str(BIC))
```

```
AIC = 3977726.040615191
BIC = 3978720.2729028435
```

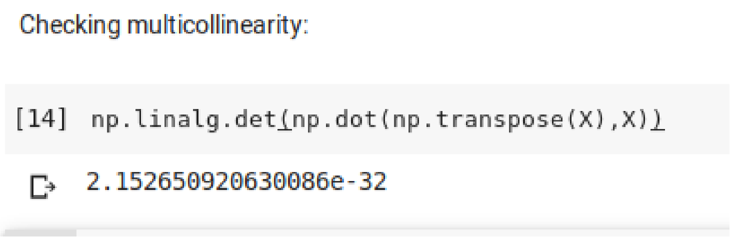


```
[42] Serr = 0
     SAerr = 0
     count = 0
     SSerr = 0
     MSerr = 0
     y_tr = np.zeros((np.size(Y_tr,0),1))
     for i in range(0,np.size(Y_pred,0)):
       y_tr[i,0] = np.argmax(Y_tr[i,:])
     diff = y_tr - y
     for i in range(0, train_len):
       Serr = Serr + y_tr[i,0] - y[i,0]
       SAerr = SAerr + abs(y[i,0]-y_tr[i,0])
       SSerr = SSerr + ( (y[i,0]-y_tr[i,0])*(y[i,0]-y_tr[i,0]) )
       MSerr = MSerr + ( (y[i,0]-y_tr[i,0])*(y[i,0]-y_tr[i,0]) )

     MSerr = MSerr/(train_len-90)
     xyz = (abs(diff)<=2).sum()

     print("Sum err: "+ str(Serr))
     print("Sum abs err: "+ str(SAerr))
     print("MS Res:: "+ str(MSerr) )
```

```
Sum err: -31063429.0
Sum abs err: 31922245.0
MS Res:: 5312.193688864923
```

Multicollinearity

In statistics, multicollinearity is a phenomenon in which one predictor variable in a multiple regression model can be linearly predicted from the others with a substantial degree of accuracy. In this situation, the coefficient estimates of the multiple regression may change erratically in response to small changes in the model or the data.



Checking multicollinearity:

```
[14] np.linalg.det(np.dot(np.transpose(X),X))
```

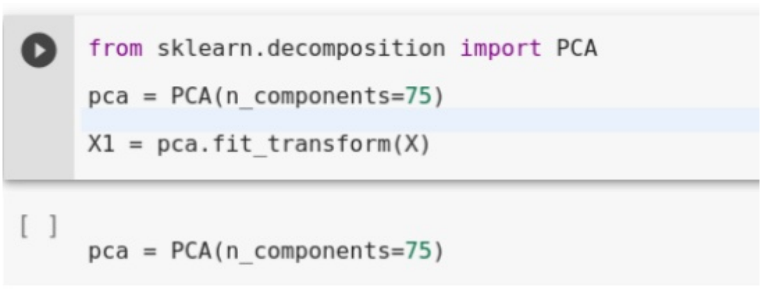
```
2.152650920630086e-32
```

We observed that the value of determinant for current model is very close to zero. This indicates presence of multicollinearity in our model and thus, for removing it from our model, we have used Principle Component Analysis (PCA) for Variable Selection.

PCA

PCA is a dimension reduction tool used to reduce a large set of correlated predictor variables to a smaller, less correlated set that still contains most of the information in the larger set. The first principal component contains as much of the variability in the data as possible, and the principal components following the first, account for remaining variability as much as they possibly can.

Following is the implementation of PCA we have used,



```
from sklearn.decomposition import PCA
pca = PCA(n_components=75)
X1 = pca.fit_transform(X)
```

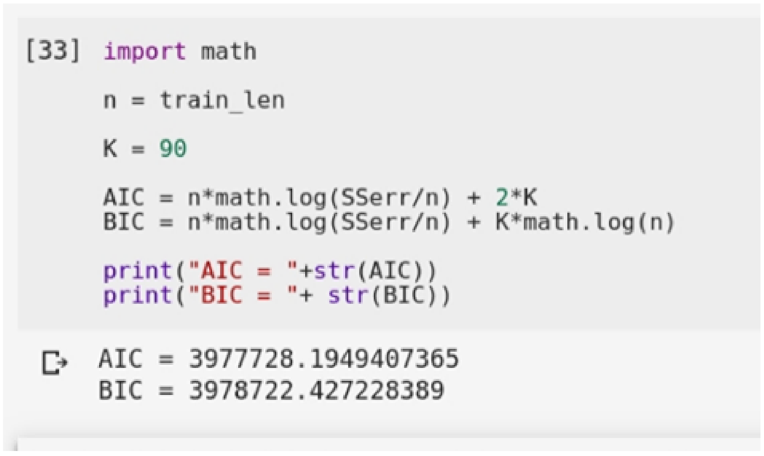
```
[ ]
pca = PCA(n_components=75)
```

We used AIC and BIC criterion for selecting our model. The details for the various models tried and their AIC and BIC values are listed below :

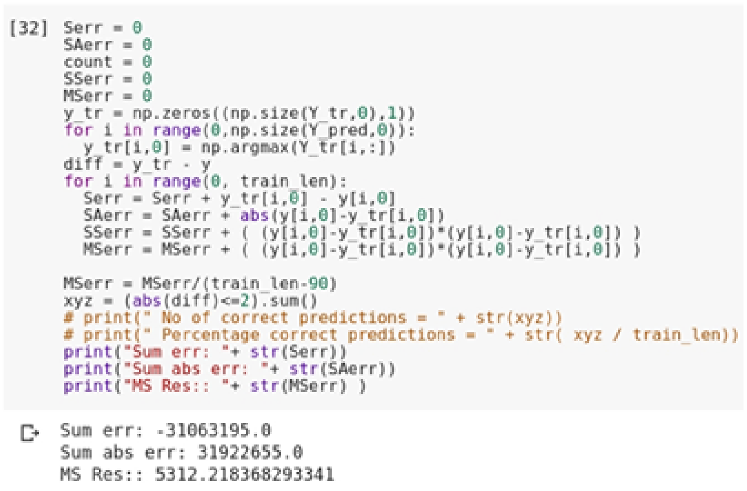
n	lambda	AIC	BIC
60	0.50	3977760.072592438	3978754.304880091
55	0.50	3977760.471157472	3978754.703445124
85	0.50	3977760.465509410	3978754.697797063
80	0.50	3977747.457653259	3978741.689940911
75	0.50	3977747.112735172	3978741.345022824
75	0.25	3977735.986541676	3978730.218829328
75	0.15	3977728.194940736	3978722.427228389
70	0.15	3977729.193586414	3978723.425874066
70	0.25	977736.4628864222	3978730.695174074

For best result, we choose the model with the minimum value of AIC and BIC criterion i.e with number of regressors = 75 and $\lambda = 0.15$. The specific results in this case is given below:

Results after applying PCA



```
[33] import math

     n = train_len

     K = 90

     AIC = n*math.log(SSerr/n) + 2*K
     BIC = n*math.log(SSerr/n) + K*math.log(n)

     print("AIC = "+str(AIC))
     print("BIC = "+ str(BIC))

     AIC = 3977728.1949407365
     BIC = 3978722.427228389
```



```
[32] Serr = 0
     SAerr = 0
     count = 0
     SSerr = 0
     MSerr = 0
     y_tr = np.zeros((np.size(Y_tr,0),1))
     for i in range(0,np.size(Y_pred,0)):
       y_tr[i,0] = np.argmax(Y_tr[i,:])
     diff = y_tr - y
     for i in range(0, train_len):
       Serr = Serr + y_tr[i,0] - y[i,0]
       SAerr = SAerr + abs(y[i,0]-y_tr[i,0])
       SSerr = SSerr + ( (y[i,0]-y_tr[i,0])*(y[i,0]-y_tr[i,0]) )
       MSerr = MSerr + ( (y[i,0]-y_tr[i,0])*(y[i,0]-y_tr[i,0]) )

     MSerr = MSerr/(train_len-90)
     xyz = (abs(diff)<=2).sum()
     # print(" No of correct predictions = " + str(xyz))
     # print(" Percentage correct predictions = " + str( xyz / train_len))
     print("Sum err: "+ str(Serr))
     print("Sum abs err: "+ str(SAerr))
     print("MS Res:: "+ str(MSerr) )

     Sum err: -31063195.0
     Sum abs err: 31922655.0
     MS Res:: 5312.218368293341
```

Result

The results of our work can be summarized in the following points :

- Of various methods tried one should use the single dimensional ridge regression model for prediction purposes.
- It has an accuracy of around 19 % with an error margin of 2 years, which is quite reasonable to assume, considering that the type of music available in market doesn't change in such small time interval.
- For better accuracy one can try Deep Learning Methods with some modifications, as the data is highly skewed and not good to work with Linear Regression Algorithms.

Acknowledgement

We would like to express our gratitude to **Dr. Sharmistha Mitra**, our academic and project mentor, for providing us with the guidance and patience needed throughout this project. Without her teaching, this project would not be possible. We would like to thank her for giving us this opportunity to explore the real life application of regression models and hence, giving us an idea about their statistical viewpoint in day to day life.

References

- Wikipedia - Multicollinearity
- Wikipedia - Ridge Regression
- Wikipedia - Linear Regression
- Introduction to Linear Regression Analysis, 5th edition by Douglas C. Montgomery and Elizabeth A. Peck.