TASK 2

NAME: YEO ZHENG XU ISAAC

In [1]:
```
from pyspark.mllib.regression import LabeledPoint
import numpy as np
from pyspark.sql import Row
from pyspark.sql import functions as sql_functions
from pyspark.sql.types import *
import matplotlib.pyplot as plt
import matplotlib.cm as cm
```

Read & Load in DataFile

In [2]:
```
musicsongsdata = sqlContext.read.load('/FileStore/tables/YearPredictionMSD.tx
t', 'text')
numbr_points = musicsongsdata.count()
```

In [3]:
```
att_descr = "90 attributes, 12 = timbre average, 78 = timbre covariance. \nThe
first value is the year (target), ranging from 1922 to 2011. \nFeatures extrac
ted from the 'timbre' features from The Echo Nest API. \nWe take the average a
nd covariance over all 'segments', each segment being described by a 12-dimens
ional timbre vector."

df = musicsongsdata.rdd.map(lambda row: str(row['value']).split(",")).map(lamb
da row: LabeledPoint(row[0], [float(x) for x in row[1:]])).toDF(["Features",
"Year"])
```

In [4]:
```
print (att_descr)
print ('\nNumber of data points: ', numbr_points, "\n")
```

```
90 attributes, 12 = timbre average, 78 = timbre covariance.
The first value is the year (target), ranging from 1922 to 2011.
Features extracted from the 'timbre' features from The Echo Nest API.
We take the average and covariance over all 'segments', each segment being describe
d by a 12-dimensional timbre vector.

Number of data points:  515345
```

In [5]:
```
musicsongsdata.take(1)
```

```
Out[5]: [Row(value='2001,49.94357,21.47114,73.07750,8.74861,-17.40628,-13.09905,-2
5.01202,-12.23257,7.83089,-2.46783,3.32136,-2.31521,10.20556,611.10913,951.08960,69
8.11428,408.98485,383.70912,326.51512,238.11327,251.42414,187.17351,100.42652,179.1
9498,-8.41558,-317.87038,95.86266,48.10259,-95.66303,-18.06215,1.96984,34.42438,11.
72670,1.36790,7.79444,-0.36994,-133.67852,-83.26165,-37.29765,73.04667,-37.36684,-
3.13853,-24.21531,-13.23066,15.93809,-18.60478,82.15479,240.57980,-10.29407,31.5843
1,-25.38187,-3.90772,13.29258,41.55060,-7.26272,-21.00863,105.50848,64.29856,26.084
81,-44.59110,-8.30657,7.93706,-10.73660,-95.44766,-82.03307,-35.59194,4.69525,70.95
626,28.09139,6.02015,-37.13767,-41.12450,-8.40816,7.19877,-8.60176,-5.90857,-12.324
37,14.68734,-54.32125,40.14786,13.01620,-54.40548,58.99367,15.37344,1.11144,-23.087
93,68.40795,-1.82223,-27.46348,2.26327')]
```

prepare data before testing

In [6]:
```
year_data = df.select("Year").groupBy("Year").count()
year_data.show()
```

```
+------+-----+
  Year|count|
+------+-----+
1988.0| 5611|
1976.0| 2179|
1951.0|   74|
1940.0|   52|
1928.0|   52|
1979.0| 3108|
1953.0|  133|
1987.0| 5122|
1959.0|  592|
1934.0|   29|
1978.0| 2926|
1968.0| 1867|
2010.0| 9396|
1936.0|   25|
1967.0| 1718|
1964.0|  945|
1993.0|10525|
2001.0|21590|
1965.0| 1120|
1954.0|  123|
+------+-----+
only showing top 20 rows
```

Add data into dataframe

In [7]:
```
year_data = year_data.toPandas()
year_data
```

	Year	count
0	1988.0	5611
1	1976.0	2179
2	1951.0	74
3	1940.0	52
4	1928.0	52
5	1979.0	3108
6	1953.0	133
7	1987.0	5122
8	1959.0	592
9	1934.0	29
10	1978.0	2926
11	1968.0	1867
12	2010.0	9396
13	1936.0	25
14	1967.0	1718
15	1964.0	945
16	1993.0	10525
17	2001.0	21590
18	1965.0	1120
19	1954.0	123
20	1984.0	3368
21	1973.0	2596
22	1980.0	3101
23	1966.0	1377
24	1997.0	15182
25	1992.0	9543
26	1990.0	7256
27	1995.0	13257
28	2009.0	31038
29	1938.0	19
...	...	...
59	2008.0	34760
60	1952.0	77
61	1999.0	18238
62	1981.0	3162

	Year	count
63	1975.0	2482
64	2011.0	1
65	1931.0	35
66	1989.0	6670
67	2002.0	23451
68	1926.0	19
69	1958.0	583
70	1962.0	605
71	1942.0	24
72	2006.0	37534
73	1961.0	571
74	2004.0	29607
75	1941.0	32
76	1955.0	275
77	1947.0	57
78	1972.0	2288
79	1991.0	8647
80	1935.0	24
81	1983.0	3386
82	1977.0	2502
83	1974.0	2184
84	1924.0	5
85	1946.0	29
86	2005.0	34952
87	2003.0	27382
88	1971.0	2131

89 rows × 2 columns

average year

In [9]:

```
avg_year = year_data["Year"]
average_year = sum(avg_year)/len(avg_year)

print("Average year: ", average_year)
```

```
Average year:  1966.9887640449438
```

prepare data for training and testing

In [10]:
```
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(predictionCol = 'prediction')

weights = [.8, .1, .1]
seed = 42
parsed_train_data_df, parsed_val_data_df, parsed_test_data_df = df.randomSplit
(weights, seed= seed)

parsed_train_data_df.cache()
parsed_val_data_df.cache()
parsed_test_data_df.cache()
n_train = parsed_train_data_df.count()
n_val = parsed_val_data_df.count()
n_test = parsed_test_data_df.count()

print ('Training dataset size: {0}'.format(n_train))
print ('Validation dataset size: {0}'.format(n_val))
print ('Testing dataset size: {0}'.format(n_test))
```

```
Training dataset size: 412604
Validation dataset size: 51220
Testing dataset size: 51521
```

Baseline Regression Model

In [11]:
```
preds_and_labels_test = parsed_test_data_df.rdd.map(lambda row: (float(1967),
float(row['Year'])))
preds_and_labels_test_df = sqlContext.createDataFrame(preds_and_labels_test, [
"prediction", "label"])
rmse_test_base = evaluator.evaluate(preds_and_labels_test_df)

print ('Baseline Model RMSE = {0:.3f}'.format(rmse_test_base))
```

```
Baseline Model RMSE = 33.216
```

Linear regression model

In [12]:
```
from pyspark.ml.regression import LinearRegression
from pyspark.ml.linalg import Vectors, VectorUDT
from pyspark.sql.functions import udf
```

In [13]:
```
# Linear regression model parameter values
num_iters = 500  # iterations
reg = 1e-1  # regParam
alpha = .2
use_intercept = True  # intercept

parsed_train_data_df = parsed_train_data_df.rdd.map(lambda row: (Vectors.dense
(row["Features"]), float(row['Year'])))
parsed_train_data_df = sqlContext.createDataFrame(parsed_train_data_df,["featu
res","label"])
parsed_train_data_df
lin_reg = LinearRegression(maxIter = num_iters, regParam = reg, elasticNetPara
m = alpha, fitIntercept = use_intercept, labelCol = 'label', featuresCol = 'fe
atures')

first_model = lin_reg.fit(parsed_train_data_df)
```

In [14]:
```
coeffs_LR1 = first_model.coefficients
intercept_LR1 = first_model.intercept
print (coeffs_LR1, intercept_LR1)
```

```
[0.8347217130845156,-0.05190599145213811,-0.04168276068338578,-0.003693849860723591
3,-0.010676590141455236,-0.2033236563392337,0.0,-0.08277333044626019,-0.05829932380
4725314,0.01572249772368674,-0.12697748915585005,-0.007405290217998796,0.0425753715
8251787,0.0003437618917719309,-0.00036459934760200405,0.0005395708543848927,0.00046
051872564733534,0.001190216034788362,0.0018663702878860752,0.0021872015887201726,0.
0005128886692581681,0.0,0.007382789167725535,0.0023336370948281826,-0.0027614073300
38428,2.635020903561012e-06,0.001450525479621095,0.00041657686149853694,0.000396040
99193300704,0.0,-0.0011464663545873 72,-0.0009722868779609 16,-0.004574281577854 3,0.0
0173960723813341 23,0.0010954161862237 26,-0.0046973397559325475,-0.00015294551878303
67,0.0007257669233730452,0.0013914899817465908,-0.0016325733146836721,-0.0023397667
968733903,-0.0005481037273406006,-0.0012143070528128287,-0.0012905608295990143,-0.0
026280100620368267,0.005088837976661 2655,0.00047790112763 6254,-0.0017651116480884 27
5,0.0,0.0015861264705711753,0.0002792305648505766,-0.0012010019250170267,0.00124009
54182563964,0.0,0.0,4.479901966925109e-05,-0.0017924627036103 59,0.0018663986382184 5
6,-0.0013909959979994907,0.0,-0.0024407596982693016,-0.0018359873711215964,-0.00620
6821566157933,0.0010367800369333 73,-0.0016813917836626413,0.00047143220909953124,0.
0,-0.00031238850397667 79,-0.004031351266301818,-0.0036279634175283 34,-0.00099743800
11918134,8.786089655478002e-05,0.0008226595966438574,0.0037902228269242 97,0.0032358
621112919834,0.012066899424304247,4.64381941543498e-05,-0.004257756372207 57,0.0,-7.
176733244424648e-05,0.0,-0.00033017056435866464,0.0015010172978703294,0.00109974852
69308013,0.025956049836993674,-0.00017676608927660 21,0.0009605829563652412,-0.02690
474124046 8186,-0.0011078252600576308,-0.0007826079873739886] 1953.347068077764
```

In [15]:
```
parsed_val_data_df = parsed_val_data_df.rdd.map(lambda row: (Vectors.dense(row
["Features"]), float(row['Year'])))
parsed_val_data_df = sqlContext.createDataFrame(parsed_val_data_df,["features"
,"label"])

val_pred_df = first_model.transform(parsed_val_data_df)
rmse_val_LR1 = evaluator.evaluate(val_pred_df)

print ('Validation RMSE:LR1 = ',  rmse_val_LR1)
```

```
Validation RMSE:LR1 =  9.546136810087917
```

RMSE derived from implementing Linear regression is around 9.xx

Results were greatly improved by implementing Linear regression from the Baseline Regression model which was around 33.216