Yong Liang 23941603
hw 3
Due: 3/03/2016

1. (25 points) Spark Warm-Up Exercises Starting with the code provided in local_run_me.py that uses the local data, answer the following warm-up questions.

a. (5 pts) How many data cases are contained in train_data.txt?

num of datarow: 5000

b. (5 pts) What is the average year of the songs contained in train_data.txt?

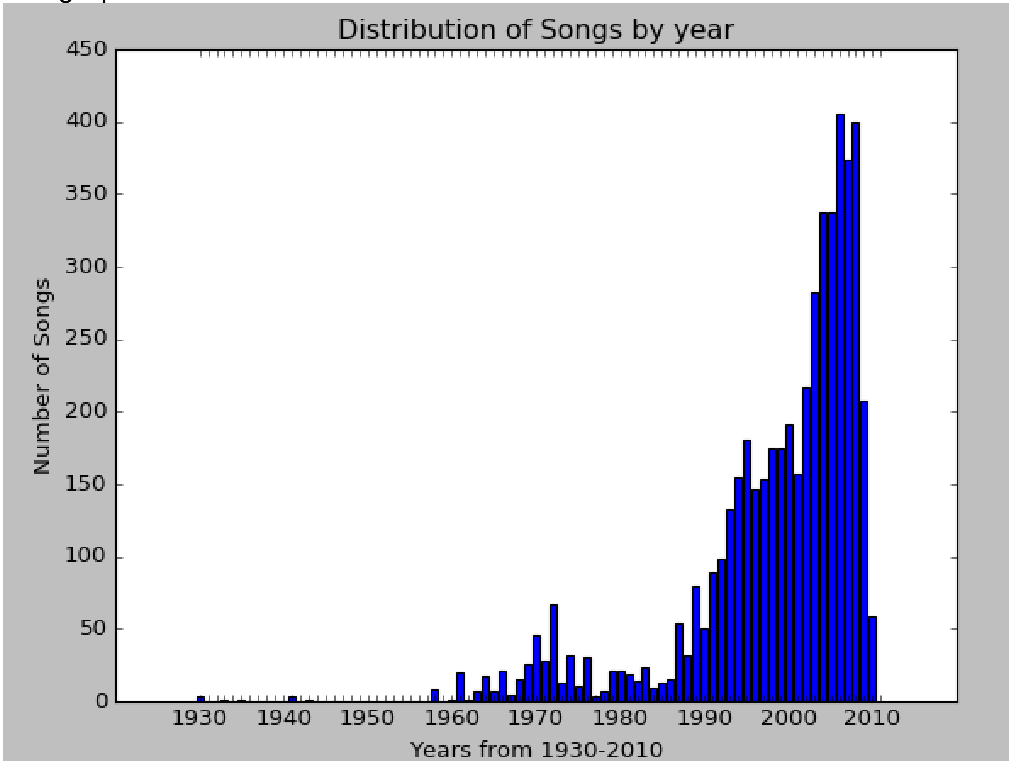
avg year: 1998.5696

c. (5 pts) What is the sample standard deviation of the years contained in train_data.txt?

std : 10.6645347906

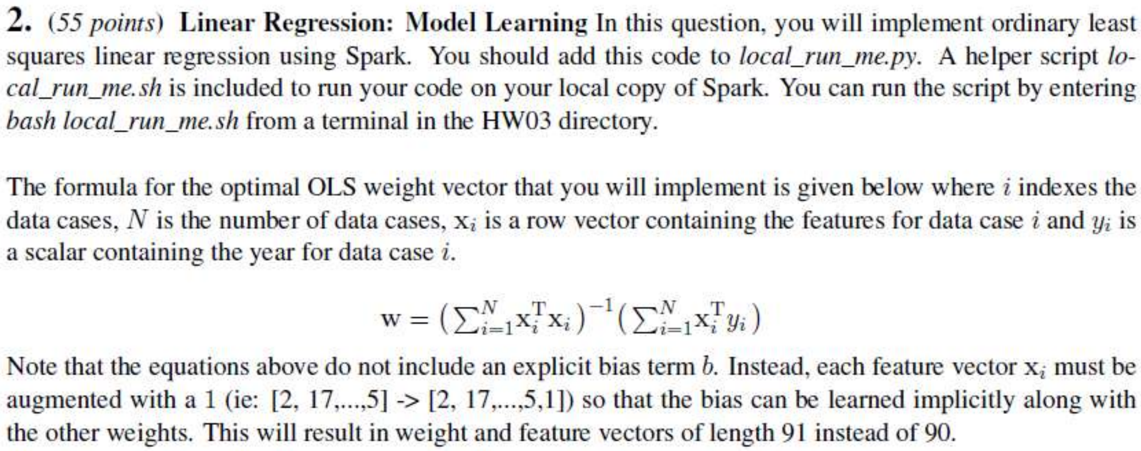
d. (5 pts) What is the distribution of years contained in train_data.txt? Provide your answer as a bar chart with one bar per year showing the number of songs for that year.

The graph looks like bimodal distribution around 1970 and 2005



e. (5 pts)What is the inner product (dot product) between the first and second columns of features values?

feat.1 dot feat.2: 1429983.30713



2. *(55 points)* **Linear Regression: Model Learning** In this question, you will implement ordinary least squares linear regression using Spark. You should add this code to *local_run_me.py*. A helper script *local_run_me.sh* is included to run your code on your local copy of Spark. You can run the script by entering *bash local_run_me.sh* from a terminal in the HW03 directory.

The formula for the optimal OLS weight vector that you will implement is given below where i indexes the data cases, N is the number of data cases, x_i is a row vector containing the features for data case i and y_i is a scalar containing the year for data case i.

$$\mathrm{w} = \left(\sum_{i=1}^{N} \mathrm{x}_i^{\mathrm{T}} \mathrm{x}_i\right)^{-1} \left(\sum_{i=1}^{N} \mathrm{x}_i^{\mathrm{T}} y_i\right)$$

Note that the equations above do not include an explicit bias term b. Instead, each feature vector x_i must be augmented with a 1 (ie: [2, 17,...,5] -> [2, 17,...,5,1]) so that the bias can be learned implicitly along with the other weights. This will result in weight and feature vectors of length 91 instead of 90.

a. (10 pts) Write code for the computation PN i=1xTi yi using Spark transformations and actions. Note that this produces a length 91 vector as a result. Run this computation over the local data and include the first four elements of this vector in your report.

[4.37434207e+08 2.68183018e+07 9.23771928e+07 2.03370028e+07]

b. (15 pts) Write code that computes PN i=1xTi xi using Spark transformations and actions. Note that this computation is an outer product and the result in our case will be a 91 _ 91 matrix. Run this computation over the local data and include the upper-left 4 _ 4 block of the matrix in your report as the answer the this question. (Hint: You can make use of Numpy functions within Spark map operations.)

```
9747914.64573 1429983.30713 2278545.30941 429219.001377
1429983.30713 12125731.7677 660413.890919 290676.5843
2278545.30941 660413.890919 6322846.35116 522586.117166
429219.001377 290676.5843 522586.117166 1277273.25805
```

c. (10 pts) Use Numpy to combine the output of the two previous computations to obtain the linear regression weight vector w according to the formula shown above. Run this computation over the local data and include the first four elements of the resulting length 91 weight vector in your report.

array([0.8487824 , -0.06103316, -0.02729652, -0.01136234])

d. (10 pts)Given a weight vector w, you can make predictions using the equation ^yi = wT xi. Write Spark code to map the feature values in the RDD to predicted song year values using the weight vector learned in the previous question. Using your code, make predictions using the features contained in the local copy of test_data.txt. Include in your report the predicted years for the first four test data cases.

[1998.8538107469153, 1999.7755196299984, 1998.7605265328787, 1997.6113422908302]
Rounded:
[1999, 2000, 1999, 1998]

e. e. (10 pts) Write Spark code to compute the MAE of the predictions obtained using the weight vector w. Recall that the MAE is defined as: 1 N XN i=1 jyi □ ^yij Apply your script to the true and predicted target values computed in the previous question. Include the MAE you find in your report. (Hint: It may be helpful to modify the solution to the previous question slightly so that it returns an RDD where the first column corresponds to the true year and the second corresponds to the predicted year.)

MAE: 8.6462831952345098

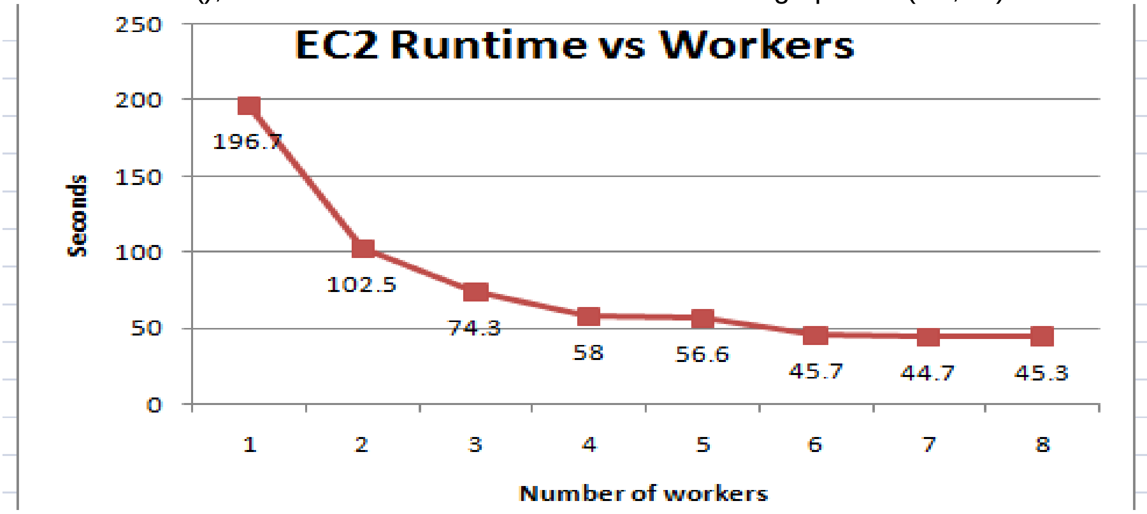
3. (20 points) Linear Regression: Learning at Scale In this question, you will test your code at scale using Amazon EC2. Separate instructions have been provided for setting up an Amazon AWS account, redeeming Amazon credits, and starting a Spark cluster on EC2.

a. (10 pts)The procedure described above will run your training code over all 463,715 training examples and test it on all 51,630 test instances. Report the MAE your code obtains on the full test set as your answer to this question.

MAE score 6.80049646324

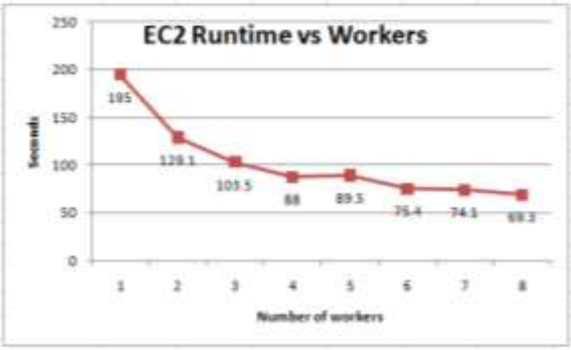
b. (10 pts)The procedure described above will run your code using between 1 and 8 spark workers and will output the total time needed to train and test the model using your code for each number of workers. The time when running with a single worker should not be more than 5 minutes. When the run completes, the code will output the amount of time used for each number of workers as a list of lists in the format [number of workers, run time in seconds]. Produce a line plot showing the run time of your code versus the number of workers and include it in your report as your answer to this question.

With rdd.cache(), and correct rdd matrix transformation using np.outer(arr,arr)

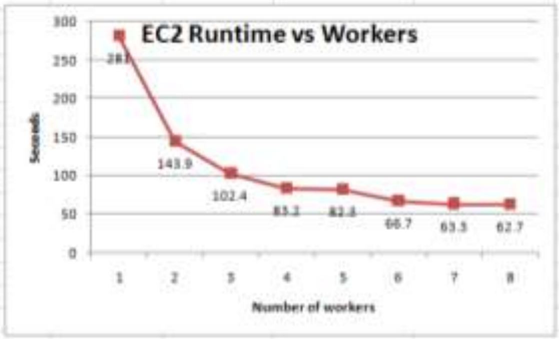


c. (10 pts)Bonus: Investigate different implementations of the linear regression learning code to see if you can find a solution with better parallel scalability than your initial solution. If you find a significantly better implementation, describe what you changed and provide a line plot showing the run time of your alternate implementation versus the number of workers. Submit both your initial and updated code files.

Wrong way: XT * X by first .collect() on X

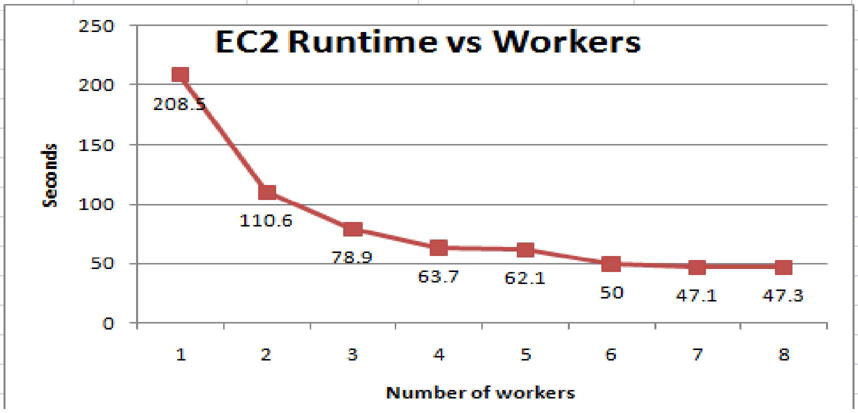


Slow: Correct way, but use no .cache():



Speedup: Cal XT* X and XT*Y in one step:

```
        full_matrix             = rdd_train_1.map(lambda arr: np.outer(arr,arr[1:]))
        xfeatures_matrix        = full_matrix.map(lambda arr: arr[1:]).reduce(lambda a,b:a+b)
        x_dot_y_sum             = full_matrix.map(lambda arr: arr[0]).reduce(lambda a,b:a+b)…
```



Based on my analysis,

- Caching speeds up process by 30% at each worker scale
- Running one big matrix doesn’t seem to be any different, maybe even slower. In theory, giant job is faster than running multiple small jobs due to communication delay.

d. (10 pts) Bonus: Investigate modifications to the basic OLS algorithm to see if you can find a solution with better accuracy on the full data set. Things to try are feature normalization, feature selection, output normalization, regularization, etc. If you find a significant enhancement to the model, provide the resulting MAE, describe what you changed, and assess the impact on parallel scalability. Submit both your initial and updated code files.

Feature Normalization, take the Euclidean norm. Then follow the same matrix multiplication. Apply the same norm vector to the test_features, then run predict.

Backward feature selection, by select columns of features. I didn’t implement it.

Output normalization push out of range data into range, seem to perform well.

```
        for index, x in enumerate(predict):
                if x <1930:
                        predict[index] = 1930
                if x >2010:
                        predict[index] = 2010
```

Local:

Old MAE:	8.6462831952345098	took 3 seconds
+Norm feat:	8.64628319538	about 10 seconds slower in time
+Output norm:	8.56793856959	no difference in time

Full dataset:

MAE score	6.80049646324	took 3 minutes on single worker
+Output norm:	6.76462036478	no difference in time
+Norm feat	6.76462036581	took 12 mins [[1, 730.19767212867737]]