A Music Recommender:

LastFM 360k User Recommender

John Bellamy
jbdatasci.com

Agenda:

Part I: A discussion of the problem and an exploration of the steps completed to arrive at final model

Part II: A demonstration of the model.

Part III: Discussion.

Conclusion.

About the Data:

- 360k User Dataset made available by LastFM for non-commercial use.
- 17 million+ ratings on 287k+ Artists.
- USERID is encrypted and we have an incomplete ARTIST_ID in the form of another site's column.

The Problem:

- Use Alternating Least Squares in PySpark to develop a recommender in a distributed setting.
- Test the performance of various models, through parameter optimization and seven-fold validation.
- Demonstrate use of model

Sample Data

Below is the data in raw form. At first blush it seems like one should be able to just do some basic analysis and then make a model. However, the data needs reshaping. Notice the difference between this slide and the next. An algorithm was applied to the user and each letter in the encryption replaced by a number. Then the number was trimmed so that Java stayed happy. An artist ID was also created.

```
In [1]: text_file="hdfs://hadoop-master:9000/users/spark/data/plays.tsv"

In [2]: plays = sc.textFile(text_file)

In [3]: plays.take(5)

Out[3]: ['00000c289a1829a808ac09c00daf10bc3c4e223b\t3bd73256-3905-4f3a-97e2-8b341527f805\tbetty blowtorch\t2137',
         '00000c289a1829a808ac09c00daf10bc3c4e223b\tf2fb0ff0-5679-42ec-a55c-15109ce6e320\tdie Ärzte\t1099',
         '00000c289a1829a808ac09c00daf10bc3c4e223b\tb3ae82c2-e60b-4551-a76d-6620f1b456aa\tmelissa etheridge\t897',
         '00000c289a1829a808ac09c00daf10bc3c4e223b\t3d6bbeb7-f90e-4d10-b440-e153c0d10b53\telvenking\t717',
         '00000c289a1829a808ac09c00daf10bc3c4e223b\tbbd2ffd7-17f4-4506-8572-c1ea58c3f9a8\tjuliette & the licks\t706']

In [4]: # user-mboxsha1 \t musicbrainz-artist-id \t artist-name \t plays
```

Data in Numeric

Below is the basic dataset, but the data is far from ready. Next is descriptive statistics and some visualizations that show opportunity to improve.

```
#Heres our new dataset! It is in the form of index|user_id|artist_name|artist_i
_RDD.take(5)
```

```
[(0, '328911829', 116947, 'camo & krooked', '2137'),
 (13107200, '266493556', 207956, 'grouper', '517'),
 (8912900, '821342251', 47688, 'bob marley', '424'),
 (4718600, '444164354', 9388, 'jimmy eat world', '81'),
 (524300, '717350741', 17624, 'ozzy osbourne', '74')]
```

```
id_lookup = _RDD.map(lambda x : (x[2],x[3]))
```

Desc. Analysis

In [18]:
```
# find the average number of plays
sum_of_ratings = _RDD.map(lambda x: (int(x[4]))).reduce(lambda x,y: x + y)
```

In [19]:
```
average_number_of_listens = sum_of_ratings/float(number_of_entries)
```

In [20]:
```
print('The average rating is: {:.2f}'.format(average_number_of_listens))
```

```
The average rating is: 215.18
```

In [21]:
```
print('The largest number of listens is: {:d}'.format(_RDD.map(lambda x:(int(x[4]))).max()))
```

```
The largest number of listens is: 419157
```

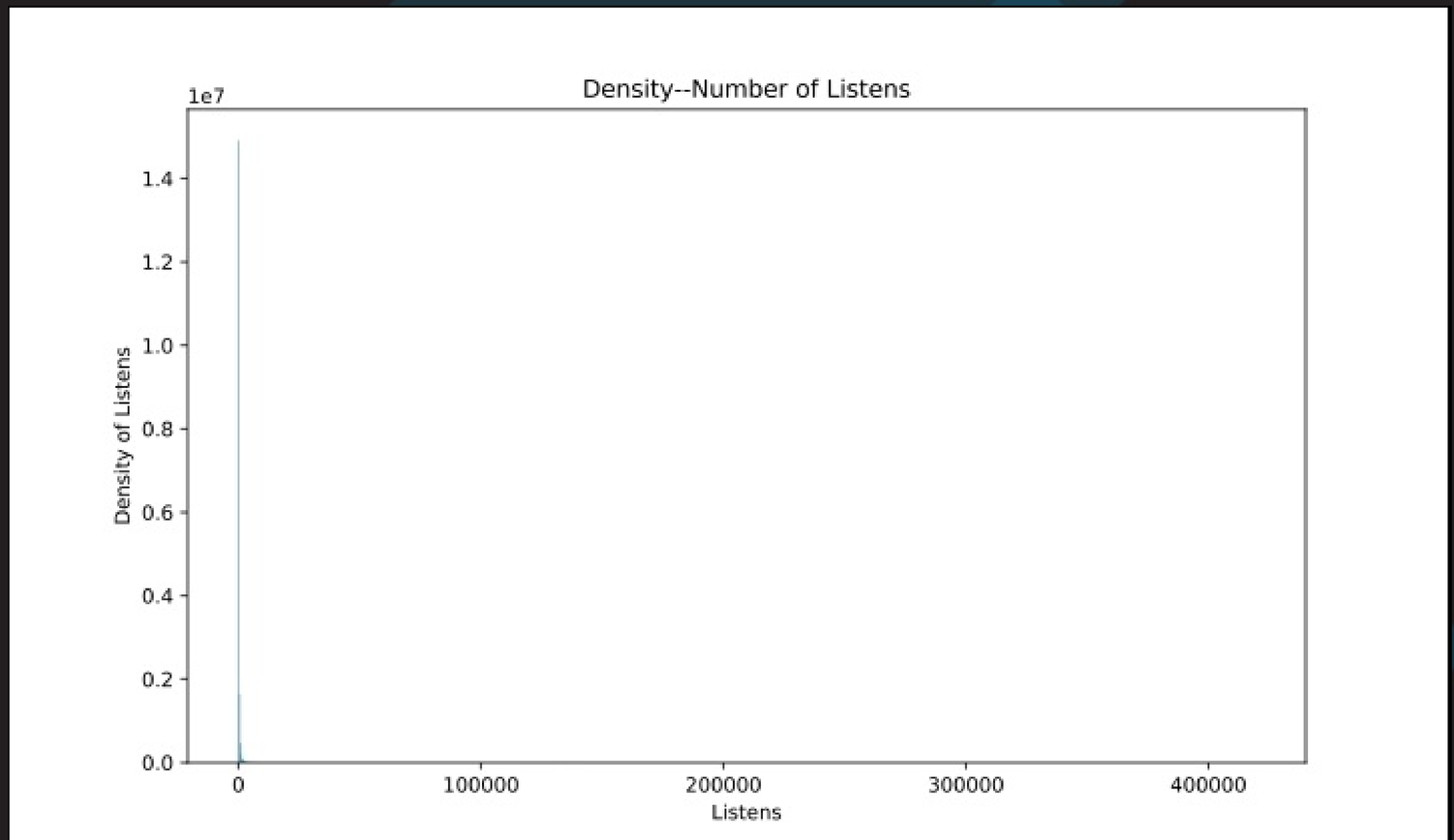
In [22]:
```
print('The smallest number of listens is: {:d}'.format(_RDD.map(lambda x:(int(x[4]))).min()))
```

```
The smallest number of listens is: 0
```

419,157?

- It is clear that there are some outliers. Is it really possible for one user to listen to a song that many times? The average is 215.
- Let's look at a histogram. Hint: it's ugly.

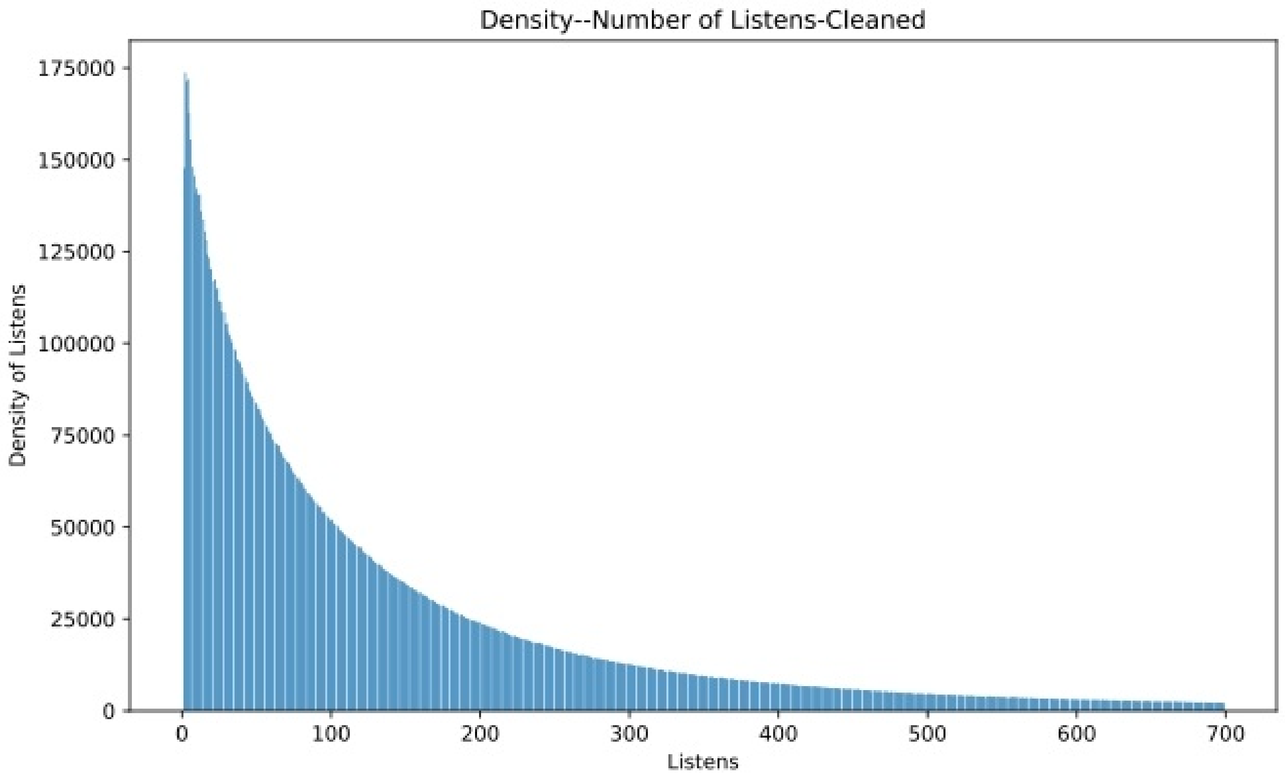
Histogram I


Density--Number of Listens
1e7
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Density of Listens
0
100000
200000
300000
400000
Listens

Cleaning...

Sometimes there isn't a clear way to proceed. This is the "art" part of data science. After a few iterations, 700 was chosen as the maximum number of listens, which is the implicit rating, as discussed in model section. Let's see the new histogram.

Histogram II



Another Opportunity:

Some artists have as many as many as 77.3k listens. Others have only 1. We don't want to eliminate an artist, so the artists that had been listened to (rated) more than 180 were separated into anoter dataset. A random sample selecting 180 of each artist was then taken and those artists added back to the dataset.

```
print('The highest-rated artist had : {:d} listens'.format(rated.map(lambda x: (int(x[1]))).max()))
```

```
The highest-rated artist had : 77348 listens
```

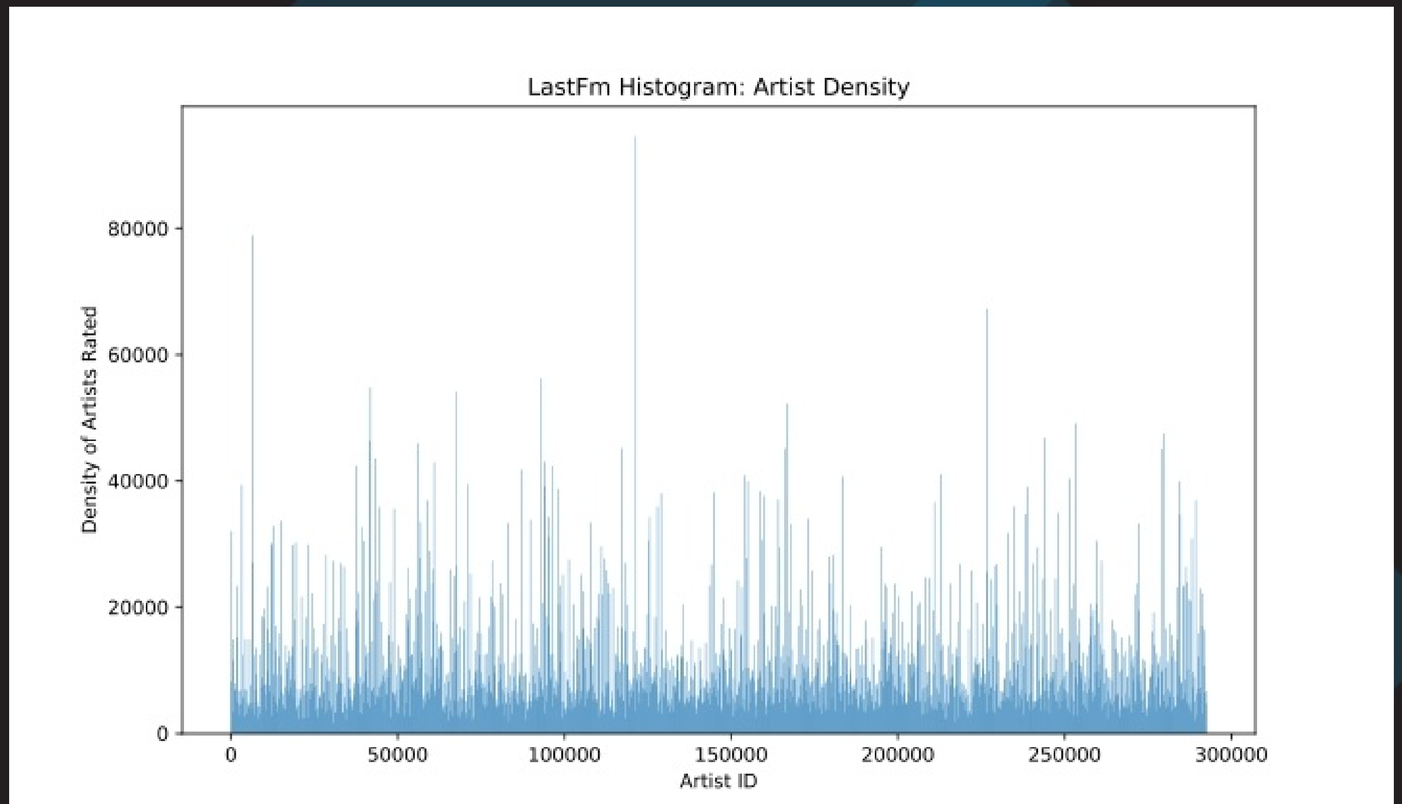
```
print('The average-rated artist had : {:f} listens'.format(rated.map(lambda x: (int(x[1]))).mean()))
```

```
The average-rated artist had : 60.971792 listens
```

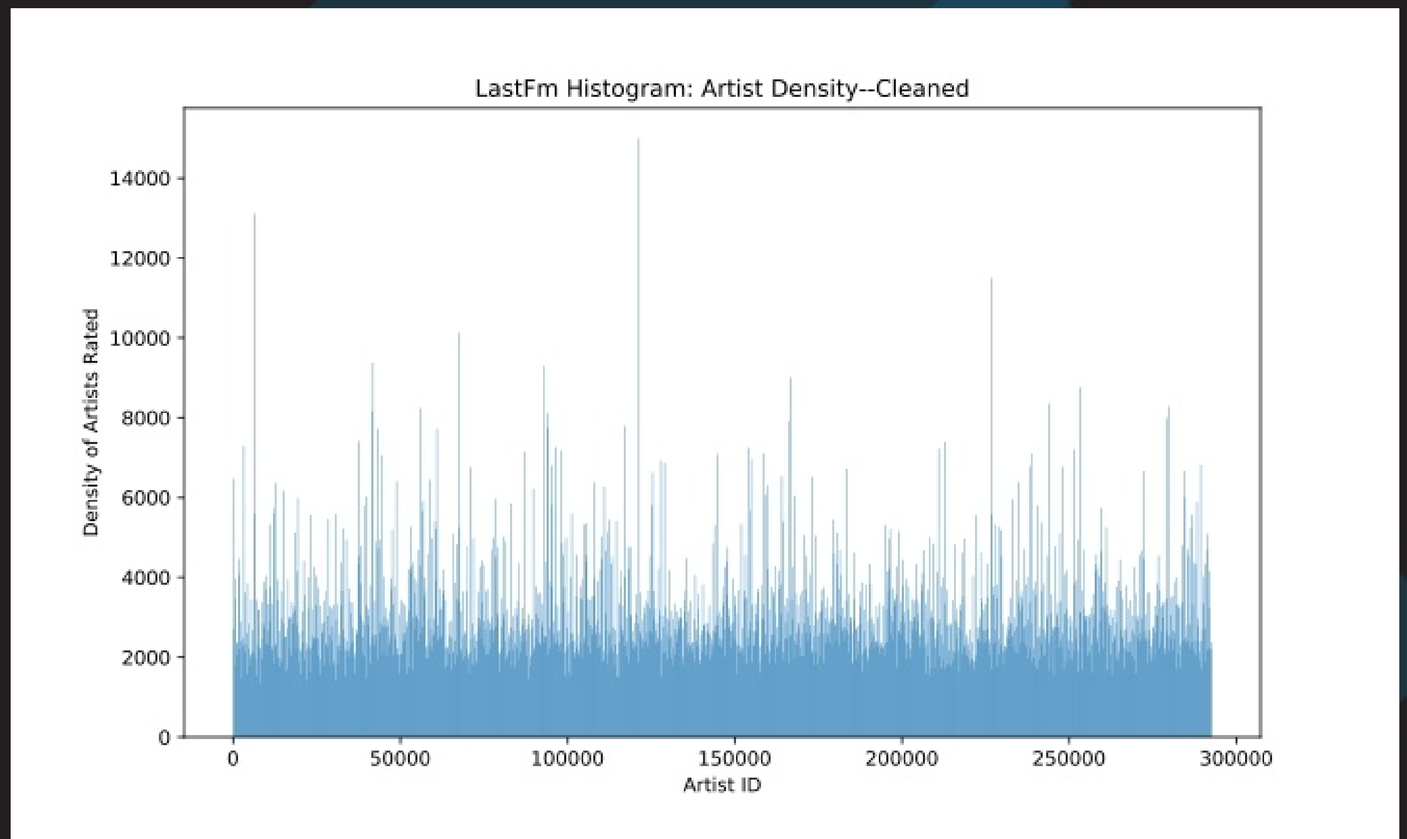
```
print('The lowest-rated artist had : {:d} listens'.format(rated.map(lambda x: (int(x[1]))).min()))
```

```
The lowest-rated artist had : 1 listens
```

Histogram III



Histogram IV



Give & Take

We took out a good portion of ratings, since they were the same artists. It seems that the dataset was dominated by 10840 artists, or a little less than 5%. This 5% made up about 26% of the whole dataset. There are a little over 100 artists who had over 50k ratings.

We are left with a dataset of only 4673605. But we will get a better model, without the added bias of the artists with an extremely high number of ratings.

About the ALS Alg.

Alternating Least Squares:

- Works well with parallelization, due to the (m + y) + (n + y), the matrix A = XY, and X = mxn and y is the transpose or m. Basically the cost is "+," so to speak, not "*". It is "*" only when needed.
- The goal is the assumption that the complete ratings matrix R is approximately equal to transpose(X)*Y, where R is the ratings matrix.
- It is called alternating, because the transpose(x) side and the "Y" or predictor side are optimized alternatingly; in otherwords, once Y is optimized, for errors, X is optimized. When both sides are optimized, the algorithm ends, or converges.

Hyper-parameters

In the past, I have noticed that the explicit recommender works fairly well for things like "number of listen," or "number of sales," if the predictor's interval is somewhat compact.

Here, we see two models, one using PySpark's ALS implicit and explicit options. The explicit model seemed to converge very quickly, and had an RMSE of .12. Hyper-parameter-finding techniques did little good for either model, and for the implicit model found hyper-parameters that resulted in an overfit model.

Implicit Model

```
Optimal hyperparameters
alpha           40.000000
min_error             inf
model_number    82.000000
n_factors       80.000000
n_iter          10.000000
second_best      0.000000
train_mse        0.138185
dtype: float64
```

Implicit Model Test

```
model = ALS.trainImplicit(rdd_training, rank=rank, iterations=n_iter, alpha=alpha, lambda_=lambda_, 
predictions = model.predictAll(predict_test).map(lambda r: ((r[0], r[1]), r[2]))
rates_and_preds = rdd_test.map(lambda r: ((int(r[0]), int(r[1])), float(r[2]))).join(predictions)
error = math.sqrt(rates_and_preds.map(lambda r: (r[1][0] - r[1][1])**2).mean())

print('The error for the test data is {:.2f}'.format(error))
```

```
The error for the test data is 0.25
```

After a little more tuning:

```
predict_test = rdd_test.map(lambda x: (x[0], x[1]))

predictions = complete_model.predictAll(predict_test).map(lambda r: ((r[0], r[1
rates_and_preds = rdd_test.map(lambda r: ((int(r[0]), int(r[1])), float(r[2])))
error = math.sqrt(rates_and_preds.map(lambda r: (r[1][0] - r[1][1])**2).mean())

print('For testing data the RMSE is {:.2f}'.format(error))
```

```
For testing data the RMSE is 0.17
```

Explicit Model

```
New optimal hyperparameters
lam                                                          0.01
min_error                                                     inf
model        <pyspark.mllib.recommendation.MatrixFactorizat...
n_factors                                                      80
n_iter                                                         20
test_mse                                                  0.12245
dtype: object
```

Explicit Test

```
# Now for the tst dataset
model = ALS.train(rdd_training, rank=5, iterations=5, seed=5)
predictions = model.predictAll(predict_test).map(lambda r: ((r[0], r[1]), r[2]))
rates_and_preds = rdd_test.map(lambda r: ((int(r[0]), int(r[1])), float(r[2])))
error = math.sqrt(rates_and_preds.map(lambda r: (r[1][0] - r[1][1])**2).mean())

print('The error for the test data is {:.2f}'.format(error))
```

```
The error for the test data is 0.12
```

Recommendations

Even though the Implicit model will be properly abondoned, the model was used to get a prediction based on a user's recommendation, for comparison. The same user and recommendation was entered for each model.

```
[('56','3018',.0001),
 ('56','153162',.0001), #Not an actual artist
 ('56','288963',.0001),# Alain Dzukam. Another movie
  ('56','14368',.0001), #David guetta. This is a movie
  ('56','204066',.1), #Rihanna
  ('56','174344',.0001),# Kaiser Chiefs
  ('56','95545',.0001), #Iron maiden
  ('56','58275',.0001),# Dire straits
 ('56','43548',.18),#Regina Spektor
 ('56','68390',.19), #Artst is 'edith piaf'
 ('56','281461',.20), #Artist is 'maria callas'
 ('56','201482',.17),# Atist is 'bryn terfel'
 ('56','243041',.17), # Anna netrebko
 ('56','244054',.17), # The residents
 ('56','256484',.18), #The Legendary Pink Dots
 ('56','206805',.17), #bjork
 ('56','127801',.0001), #Madonna
 ('56','96194',.16), #The killers
 ('56','216449',.0001), #Taylor Swift
 ('56','113418',.0001), #Miley Cyrus
 ('56','253951',.0001), # recommended: Madonna feat. Justin TimberlaK
 ('56','279301',.0001) # recommended: t-pain
```

Recommendations Implicit

```
print('Your top ten recommended artists:')
for x in top_items:
    artist = str(x[1])
    print(set(artist_name_lookup_table.lookup(artist)))
```

```
Your top ten recommended artists:
{'modest mouse'}
{'eric clapton'}
{'andrew bird'}
{'die Ärzte'}
{'blink-182'}
{'yann tiersen'}
{'bob dylan'}
{'nightwish'}
{'u2'}
{'katie melua'}
{'band of horses'}
{'the future sound of london'}
{'the prodigy'}
{'tiësto'}
{'finntroll'}
{'george michael'}
{'bright eyes'}
{'extremoduro'}
{'herbie hancock'}
{'stars'}
```

Recommendations Explicit

```
print('Your top ten recommended artists:')
for x in top_items:
    artist = str(x[1])
    print(set(artist_name_lookup_table.lookup(artist)))
```

```
Your top ten recommended artists:
{'bippp (v.a.)'}
{'heinz holliger - zehetmair - larcher - holliger - etc.'}
{'dj philip'}
{'active coma'}
{'alberto cortéz y facundo cabral'}
{'jens rachut'}
{'marco antonios solis'}
{'bellhouse'}
{'mike shiflet'}
{'digital mystikz & loefah'}
{'andy vores'}
{'andrew davis'}
{'electric eel-shock'}
{'alexander schatten'}
{'lech jankowski'}
{'manu chao !'}
{'c.a.r.n.e.'}
{'□□'}
{'die dødelsäcke'}
{'amor-te'}
```

Let it go!

The implicit model acutally didn't do too badly, but the explicit model seemed to pick up on my "tastes" in music. However, the explicit model is not usable either. For once it *is* the data's fault.

We have some recommendations for artists that are only on LastFM. Even more troublesome, I got several recommendations for what were supposed to be artist names, but in fact were movies or even clips for movies. These should not be in this dataset.

However, this dataset is great to practice building a recommender, because it is fairly big and it is dirty. Perhaps the model could be decent if one could take all the non-music non-LastFm-exclusive content out.

Discussion

ALS is not necessarily the easiest algorithm to implement, because we need the input triple to be numeric. In the case of this dataset, we only had one numeric column. In fact, that's why this dataset and ALS was chosen. As a data scientist, you should be able to "recode" a column to fit your needs.

Furthermore, rarely will one find the best model, the best hyper-parameters, or the right permutation of the dataset straightaway. It takes work, a procedural attitude towards model development, and an "intuition" of what is working and what can be improved.

An RMSE of .12 and a model which seems to be fit quite well, is an accomplishment for what appears to be a noisy dataset.

Conclusion

In this project, 17+ million artist ratings were loaded into a Hadoop cluster. Using PySpark and a Jupyter Notebook, the encrypted userid was recoded into a number that is less that the maximum Java int. An artistid was then created for each artist.

Looking at descriptive statistics, outliers for number of listens were removed, and regularization for artist listens was performed. The final output was then modelled.

Loaded in the third and last notebook, the "clean" dataset was then broken into the seven-fold datasets. An implicit and explicit model was compared, with a graph used to help find the ideal hyper-parameters. The best parameters and the best model were then selected.

Finally, the completed model was demonstrated.