Music Recommendation System

Big Data and Intelligent Analytics

Group 7

Tanmay Ingle

Bhaskar Akula

Index

Introduction

These systems passively track different sorts of user behavior, such as purchase history, watching habits and browsing activity, in order to model user preferences. Unlike the much more extensively researched explicit feedback, we do not have any direct input from the users regarding their preferences. In particular, we lack substantial evidence on which products consumer dislike. In this work we identify unique properties of implicit feedback datasets.

Tools and their significance in our project

1) ***Apache Spark***: - Recommendation Engine intuitively is solely based on user history and users' behaviors'. Thus it works better with a large amount of data. Also recommendation engines almost need to be run in real time or near real time and thereby making it well suited for big data applications. With speed being a parameter Spark is a better choice than other big data frameworks like conventional Map Reduce.

2) ***Amazon EMR*** - Amazon Elastic Map Reduce (Amazon EMR) is a web service that makes it easy to quickly and cost-effectively process vast amounts of data. Amazon EMR simplifies big data processing, providing a managed Hadoop framework that makes it easy, fast, and cost-effective for you to distribute and process vast amounts of your data across dynamically scalable Amazon EC2 instances.

   The data which we have used for this project contains 20 millions

   Rows and thus processing it on local machine wouldn't have been a great a choice. Solution to this answer is without a doubt cloud computing and Amazon is 10 times bigger than 14 of its next competitors combined. Amazon EMR provides a prebuilt Hadoop framework which makes it very handy to make spark applications on.

3) ***Flask*** – Flask is a micro framework that is mainly used for small applications with simple requirements. Similar applications which provide web services are Django and Pyramid. However both of those come with bootstrapping tools built in. Also a developer with no experience can build python based web applications with ease on Flask. And our project consists of different user based recommendation functions, thus Flask gives it flexibility to bring all these functions and structure it.

4) ***Ipython Notebook*** – Ipython Notebook on EMR was the best tool we came across for understanding and preprocessing of data. With its amalgamation with spark, it is for sure a very user friendly and effective tool for our purpose.

5) ***CherryPy*** – CherryPy slows developers to build web applications in much the same way they would build any other object oriented python program.

Workflow

1) ***Data Acquisition*** :- In collaborative filtering ,we wanted to understand both implicit as well as explicit datasets since both of them work in different way and have different metrics for building models. Music Recommendation Dataset which we used can be found at following link:-

   http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/index.html

   the S3 link for it is as follows:-
   https://s3.amazonaws.com/musicrecommendation/music_6m.txt
   https://s3.amazonaws.com/musicrecommendation/unique_artist.csv
   https://s3.amazonaws.com/musicrecommendation/unique_users.csv
   This data contains around 6 million rows with three different datasets. The main dataset contains userID, artistID, artistname and plays.

   Explicit data can be found as follows: - this data contains approx. 2 million rows. It mainly consist of userID, artistID, Artistname and ratings in two different datasets.

   https://s3.amazonaws.com/musicrecommendation/explicit/music.csv

https://s3.amazonaws.com/musicrecommendation/explicit/ratings.csv

2) ***Storage*** – As we mentioned we have stored all our data in Amazon S3 which will be used in Amazon EMR. We will be transferring the data to hdfs using distcp.

3) ***Data Preprocessing***: - The implicit data was very dirty and had to be cleaned. It contained lots of null values and even the orientation was disturbed. We spent most of time in parsing, preprocessing and cleaning this data. We performed entire preprocessing in IPython Notebook on EMR. Also around 9% of the data consisted of outliers which hampered with our model building and thus was removed. Outliers included example where one user played one artist around 300,000 times which didn't make any sense and had to be removed.

4) Model Building and tuning :-

We used ALS model under Collaborative Filtering in mlllib in spark since it is the only recommendation model available as of now. Once the preprocessing was done, building the model was straightforward. We used ALS.train for explicit data and ALS.trainimplicit for implicit data. The parameters which we tuned were as follows:-

Rank = 15 this is the number of factors which are made during ALS model. These are the hidden features for low rank approximations. After a point it became computationally expensive, thus we stick to rank of 15.

Iterations = 10. These are normal iterations which are used in any other machine learning model.

Regularization parameter = 0.01. This is mainly used to avoid overfitting.

Metrics used: - We used RMSE as our metric for both the datasets. We divided the dataset into training and validation dataset and tuned the models with different rank, reg_param to get least RMSE.

For Explicit dataset – we compared the predicted and actual ratings given by users which were in the range of 1-10. We achieved an RMSE of 0.9.

Interpretation: Since mean of the range was 5, 0.9 is roughly 20% of mean so are model pretty decent. It means that every time the model will be off by around 1.0 rating.

For Implicit dataset: - we compared the actual and predicted number of plays of every artist by the user. We got an RMSE of 90. The mean of output variable was 240 thus it wasn't a good model. This is understood since with implicit data RMSE is not the perfect metric to be used. The better metric would have been Mean Average precision. We infer that Python Api of spark isn't that developed to calculate MAP as the metric. We tried to code it however weren't able to succeed in it. Thus we used RMSE.

We used the above models to get two kind of outputs for the recommendation:-

1) Get top k recommended artists given userID and K
2) Get predicted ratings given userID and artistID. This will give a scale as to how will a user like an artist.

5) ***Deployment of model:*** - We built RESTful API on top of apache spark recommendation system. We used Flask framework for the same. We also deployed this API to web server provided CherryPy framework. This way it can handle multiple web applications thus providing music recommendation tailored to the user hence achieving scalability.

Deployment Steps are as follows:-

1) Keep all the datasets in S3
2) Launch an Amazon EMR cluster
3) Put flask_application.py and music_reco_engine into /home/Hadoop/.version/spark1.3.1 (.py files are included with this report)
4) Put main_server.py in /home/hadoop/IPythonNB(which is os.getcwd())
5) Download CherryPy in /home/Hadoop/IPythonNB using
   wget https://pypi.python.org/packages/source/C/CherryPy/CherryPy-3.8.0.tar.gz#md5=542b96b2cd825e8120e8cd822bc18f4b
6) Then tar –xvf Cherrypy

7) Put datasets into hdfs (/user/hadoop/datasets) from S3 using distcp command.

hadoop fs -mkdir /user

hadoop fs -mkdir /user/hadoop

hadoop distcp s3n://musicrecommendation/datasets /user/hadoop

8) Then install flask and cherrypy-paste by following command
   sudo pip install flask
   sudo pip install cherrypy paste

9) Then go to spark directory and use following command
   bin/spark-submit --total-executor-cores 14 --executor-memory 6g /home/hadoop/IPythonNB/server.py
   total executor cores and memory are optional.

10) You can see that Engine has started running at local host 5142. Just create a tunnel to localhost:5142 to use the recommendation engine

11) Then there are three functions which are written to use this recommendation engine. As mentioned above in model building.

a) To get top k recommendations use

http://localhost:5432/1/ratings/top/10 (this basically will give top 10 recommendations for user with userID 1) (template is as follows: - http://<server IP>:5432/<userID>/ratings/top/<k>

[["forthcoming fire", 4.441913190836468, 34], ["rita wright", 4.435091673414826, 36], ["bullet for my valentine", 4.4279767312668534, 70754], ["gina g", 4.3988662396076688, 83], ["aimee mann supertramp jon brion", 4.3802433423061284, 36], ["goldie lookin chain", 4.3762247988192975, 92], ["taylor deupree", 4.3618760024615773, 58], ["dashboard confessional", 4.349131037908851, 55613], ["marques houston", 4.3329218273457002, 6396], ["stormwarrior", 4.3322896654755141, 46077]]

b) To get predicted ratings for user and an artist requested by a user.

http://localhost:5432/1/ratings/200

(This basically will give a requested rating for user with userID and artistID

http://<server IP>:5432/<userID>/ratings/<artistID>)

Future Scope

1) Use it with Spark Streaming for real time recommendations as opposed to near real time.
2) Use Mean Average Precision as metric in python for better results with implicit datasets.

Challenges faced:-

1) Preprocessing of this data was a very big challenge
2) Integrating flask and cherryPy on local machine and further on EMR was very challenging for us.
3) Trying to write a python code for getting Mean Average Precision metric was pretty tough.

References:-

http://www.cherrypy.org/

http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/index.html

https://www.codementor.io/spark/tutorial/building-a-recommender-with-apache-spark-python-example-app-part1

http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html

ISBN 978-1-78328-851-9 – Machine learning with spark