Table of Contents

1 Problem Statement: 2

2 Description of Data 3

2.1 URL: 3

2.2 Description: 3

2.3 File Sizes: 3

2.4 File Sample Sizes: 3

2.5 Format of data file: 3

2.6 Data Schema 4

3 Description of Hardware 4

4 Description of Software 4

5 Steps & Demonstration: 4

5.1 Installation Steps 5

5.1.1 Install Homebrew 5

5.1.2 Install Python 5

5.1.3 Install PIP 5

5.1.4 Install Needed Python Modules 5

5.1.5 Install Apache Spark 5

5.1.6 Starting Jupyter Notebooks 5

5.2 Configuration Steps 5

5.2.1 Set Up Environment Variables 5

5.3 Data Description 6

5.3.1 Relational Database Schema Flow 6

5.3.2 TalkingData Dataframes 7

5.3.2.1 phone\_brand\_device\_model.csv 7

5.3.2.2 gender\_age\_train.csv 8

5.3.2.3 gender\_age\_test.csv 9

5.3.2.4 events.csv 9

5.3.2.5 app\_events.csv 10

5.3.2.6 app\_labels.csv 10

5.3.2.7 label\_categories.csv 11

6 Solution 11

6.1 Overview 11

6.2 Exploratory Data Analysis 12

6.2.1 Getting Needed Python Modules & Setting Up Spark 12

6.2.2 Setting Paths to CSVs 12

6.2.3 Importing CSVs into Spark 13

6.2.4 Getting Basic Shape of app\_events data 13

6.2.5 Most App Installs 13

6.2.6 Most App Installs Plot 14

6.2.7 Most Active Apps Installs Plot 14

6.2.8 Most Active Installed Apps 15

6.2.9 Explore App Categories 16

6.2.10 Explore Gender & Age 16

6.2.11 Mobile App Categories 17

6.3 Data Transformation 21

6.3.1 Combining Datasets 21

6.3.1 Sampling Datasets 22

Data had 207M rows of data. To much to process on my local machine. Sampling 0.01 (1%) of the data. 22

6.3.2 Data Preparation for Modeling 22

6.3.3 Index and encode categorical features for input into modeling functions 24

6.3.4 Apply Transformations For Training Data 26

6.3.5 Casting Datatypes for Saving Features 27

6.3.6 Saving CSV for modeling 28

6.3.7 Prepare Testing Data 28

6.4 Modeling 29

6.4.1 Naive Bayes 29

6.4.2 Decision Tree Classifier 31

7. Summarize: 34

7.1 Lessons Learned: 34

7.2 Pros: 34

7.3 Cons: 35

7.4 Next Steps: 35

8 YouTube URLs 35

# 1 Problem Statement:

TalkingData is China’s largest third-party mobile data platform. Using a SDK that’s integrated with smartphone apps they collect events generated by the smartphone user. This information is used for targeted advertising and mobile analytics.

The objective of this project is to build a model that will predict if the longitude/latitude will be present in the log data will be present in the future given a user’s app usage and smartphone properties.

This could be used to later do behavioral/micro targeting to provide the user with a better advertisement close to their location.

# 2 Description of Data

## 2.1 URL:

<https://www.kaggle.com/c/talkingdata-mobile-user-demographics/data>

## 2.2 Description:

Data below is mobile platform data from a kaggle competition.   It is very similar to data seen on an advertising exchange.

[TalkingData](https://www.talkingdata.com/), China’s largest third-party mobile data platform, understands that everyday choices and behaviors paint a picture of who we are and what we value. Currently, TalkingData is seeking to leverage behavioral data from more than 70% of the 500 million mobile devices active daily in China to help its clients better understand and interact with their audiences. [1]

## 2.3 File Sizes:

* app\_events.csv.zip (211.27 MB)
* App\_labels.csv.zip (4.04 MB)
* Events.csv.zip (62.24 MB)
* Gender\_age\_test.csv.zip (1.05 MB)
* Gender\_age\_train.csv.zip (891.47 KB)
* Label\_categories.csv.zip (7.67 KB)
* Phone\_brand\_device\_model.csv.zip (2.42 MB)

## 2.4 File Sample Sizes:

* app\_events.csv.zip (211.27 MB)
* App\_labels.csv.zip (4.04 MB)
* Events.csv.zip (62.24 MB)
* Gender\_age\_test.csv.zip (1.05 MB)
* Gender\_age\_train.csv.zip (891.47 KB)
* Label\_categories.csv.zip (7.67 KB)
* Phone\_brand\_device\_model.csv.zip (2.42 MB)

## 2.5 Format of data file:

csv format

## 2.6 Data Schema

Below is the data schema from the Kaggle competition website.  See link here: <https://www.kaggle.com/c/talkingdata-mobile-user-demographics/data>

* **gender\_age\_train.csv, gender\_age\_test.csv** - the training and test set

**events.csv, app\_events.csv** - when a user uses TalkingData SDK, the event gets logged in this data. Each event has an event id, location (lat/long), and the event corresponds to a list of apps in app\_events.

* **timestamp:** when the user is using an app with TalkingData SDK
* **app\_labels.csv** - apps and their labels, the label\_id's can be used to join with label\_categories
* **label\_categories.csv** - apps' labels and their categories in text
* **phone\_brand\_device\_model.csv** - device ids, brand, and models
* **phone\_brand:** note that the brands are in Chinese (translation courtesy of user fromandto)
  + 三星 samsung
  + 天语 Ktouch
  + 海信 hisense
  + 联想 lenovo
  + 欧比 obi
  + ...

# 3 Description of Hardware

* MacBook Pro
* Processor: 2.5 GHz Intel Core 7
* Memory: 16 GB 100 MHz DDR3

# 4 Description of Software

* Juptyer Notebooks (<http://jupyter.org/install.html>)
* Spark 2.2.1 (https://spark.apache.org/downloads.html)
* PySpark (https://spark.apache.org/docs/0.9.0/python-programming-guide.html)
* PySpark ML (http://spark.apache.org/docs/2.2.0/api/python/pyspark.ml.html)
* Pandas (https://pandas.pydata.org/)
* Numpy (https://www.scipy.org/scipylib/download.html)
* Matplotlib (https://matplotlib.org/users/installing.html)
* Seaborn (https://seaborn.pydata.org/installing.html)

# 5 Steps & Demonstration:

## 5.1 Installation Steps

### 5.1.1 Install Homebrew

* /usr/bin/ruby -e "$(curl -fsSL <https://raw.githubusercontent.com/Homebrew/install/master/install>)"

### 5.1.2 Install Python

* brew install python

### 5.1.3 Install PIP

* sudo easy\_install pip

### 5.1.4 Install Needed Python Modules

* python -m pip install --user numpy scipy matplotlib ipython jupyter pandas sympy nose seaborn

### 5.1.5 Install Apache Spark

brew update  
brew install scala  
brew install apache-spark

### 5.1.6 Starting Jupyter Notebooks

1. Navigate or “cd” into your project folder
2. Type jupyter notebook into your terminal window

## 5.2 Configuration Steps

### 5.2.1 Set Up Environment Variables

Add following code to your e.g. .bash\_profile

# For a ipython notebook and pyspark integration  
if which pyspark > /dev/null; then  
 export SPARK\_HOME="/usr/local/Cellar/apache-spark/2.1.0/libexec/"  
 export PYTHONPATH=$SPARK\_HOME/python:$SPARK\_HOME/python/build:$PYTHONPATH  
 export PYTHONPATH=$SPARK\_HOME/python/lib/py4j-0.10.4-src.zip:$PYTHONPATH  
Fi

You can check SPARK\_HOME path using following brew command

$ brew info apache-spark  
apache-spark: stable 2.1.0, HEAD  
Engine for large-scale data processing  
https://spark.apache.org/  
/usr/local/Cellar/apache-spark/2.1.0 (1,312 files, 213.9M) \*  
 Built from source on 2017-02-13 at 00:58:12  
From: https://github.com/Homebrew/homebrew-core/blob/master/Formula/apache-spark.rb

## 5.3 Data Description

Data below is mobile platform data from a kaggle competition.  This dataset was chosen for a couple of reasons:

1. Lots of interesting data about a user’s age, gender, geospatial (latitude/longitude), mobile app categories, phone brand, and phone model
2. Data could be used for behavioral targeting
3. Data could be used for micro-targeting
4. Good log data in general for multiple use cases

[TalkingData](https://www.talkingdata.com/), China’s largest third-party mobile data platform, understands that everyday choices and behaviors paint a picture of who we are and what we value. Currently, TalkingData is seeking to leverage behavioral data from more than 70% of the 500 million mobile devices active daily in China to help its clients better understand and interact with their audiences. [1]

## 5.3.1 Relational Database Schema Flow

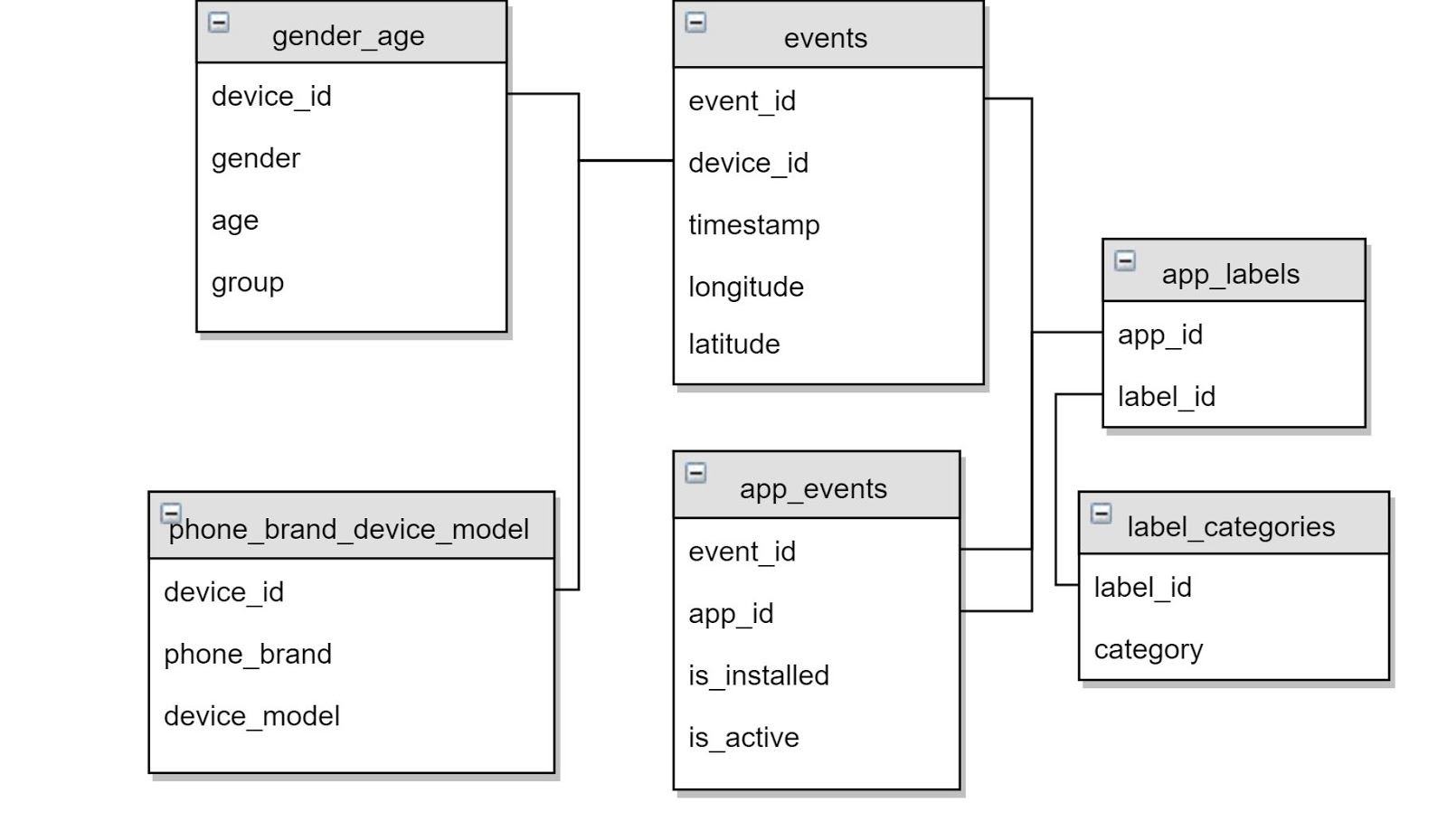
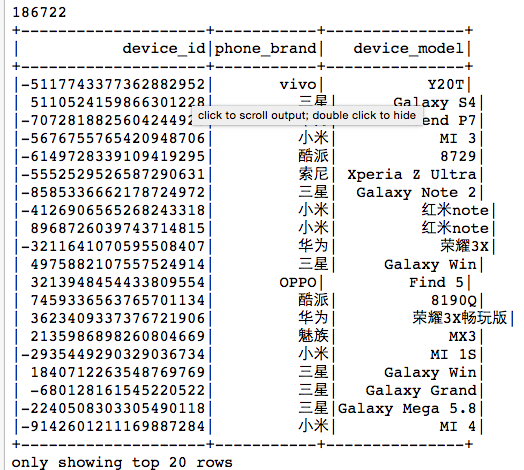


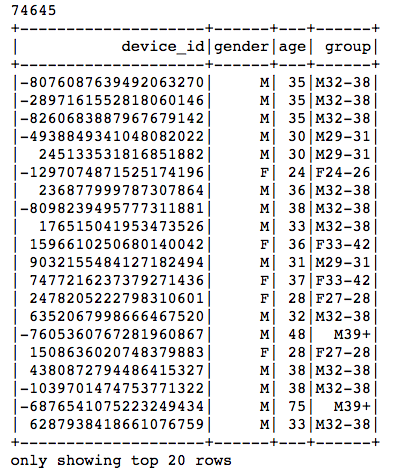
Fig 1 - Relational Databases Flow

## 5.3.2 TalkingData Dataframes

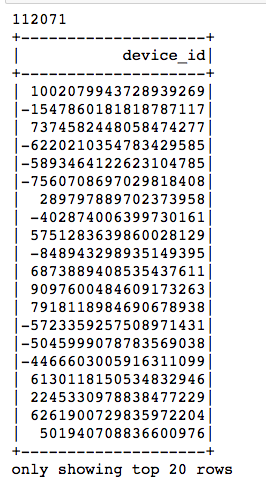
### 5.3.2.1 phone\_brand\_device\_model.csv

****

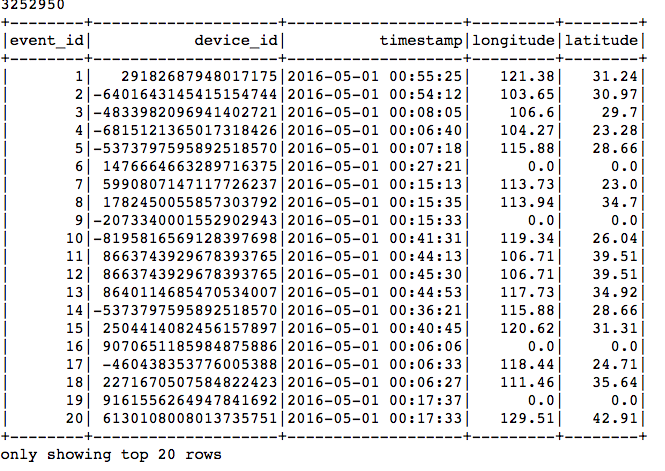
### 5.3.2.2 gender\_age\_train.csv

****

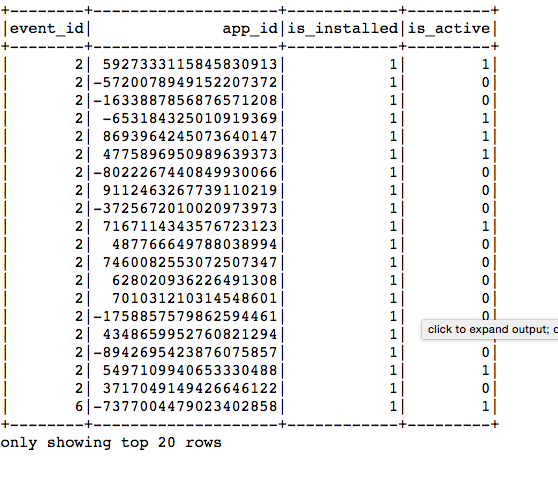
### 5.3.2.3 gender\_age\_test.csv

****

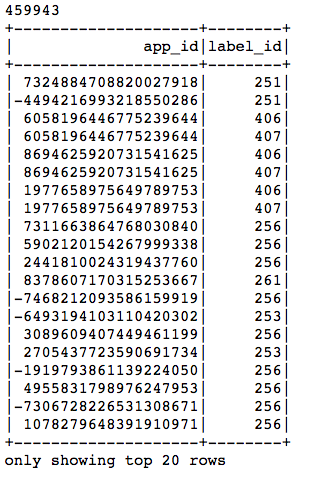
### 5.3.2.4 events.csv

****

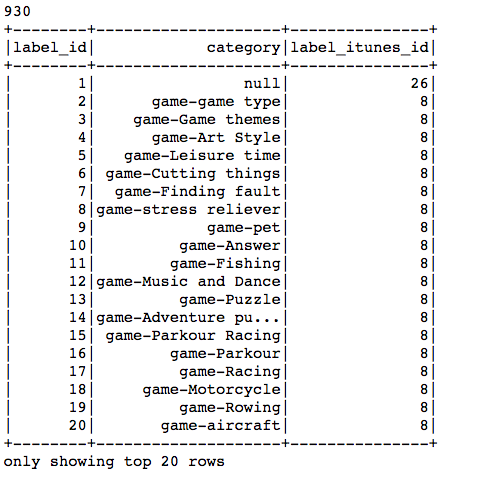
### 5.3.2.5 app\_events.csv

****

### 5.3.2.6 app\_labels.csv

****

### 5.3.2.7 label\_categories.csv

****

# 6 Solution

## 6.1 Overview

My model implemented an Apache Spark 2.2 ML pipeline coded in Python and viewed in Jupyter Notebooks. The Spark jobs were executed on a local machine, but could easily be transferred to an AWS EC2 instance reading data in from Hadoop and S3.

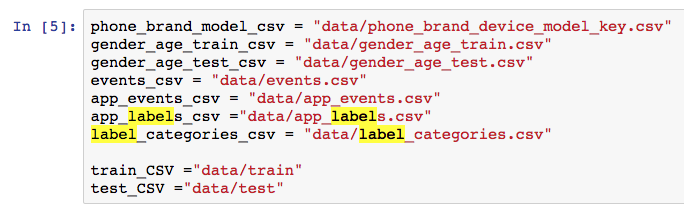
## 6.2 Exploratory Data Analysis

In Jupyter Notebooks we will explore the data

### 6.2.1 Getting Needed Python Modules & Setting Up Spark

### 

### 6.2.2 Setting Paths to CSVs



### 6.2.3 Importing CSVs into Spark

Note: I am only going to show one import, but it will be the same process to get all CSVs

|  |
| --- |
| app\_events = spark.read.csv(app\_events\_csv, header=True, mode="DROPMALFORMED", inferSchema='true', encoding="utf-8").persist()  app\_events.show() |

### 6.2.4 Getting Basic Shape of app\_events data

### 

### 6.2.5 Most App Installs

|  |
| --- |
| app\_events\_installed = app\_events.filter((col("is\_installed") == 1)) \  .groupby('app\_id') \  .agg({"is\_installed":"sum"})\  .rdd.sortBy(lambda x: x[1], ascending=False)\  .toDF()\  .toPandas()\  .set\_index("app\_id")  app\_events\_installed |

### 6.2.6 Most App Installs Plot

### 

### 6.2.7 Most Active Apps Installs Plot

|  |
| --- |
| app\_events\_active = app\_events.filter((col("is\_active") == 1)) \  .groupby('app\_id') \  .agg({"is\_active":"sum"})\  .rdd.sortBy(lambda x: x[1], ascending=False)\  .toDF()\  .toPandas()\  .set\_index("app\_id")  app\_events\_active  mpl.rcParams['figure.figsize'] = (15.0, 6.0)  app\_events\_active[0:50].plot(kind = 'bar', color = 'red') |

### 

### 6.2.8 Most Active Installed Apps

|  |
| --- |
| apps\_installed\_active = app\_events.filter((col("is\_installed") == 1) & (col("is\_active") == 1))\  .groupby('app\_id') \  .agg({"is\_installed":"sum"})\  .rdd.sortBy(lambda x: x[1], ascending=False)\  .toDF()\  .toPandas()\  .set\_index("app\_id")  apps\_installed\_active  mpl.rcParams['figure.figsize'] = (15.0, 6.0)  apps\_installed\_active[0:50].plot(kind = 'bar', color = 'purple') |

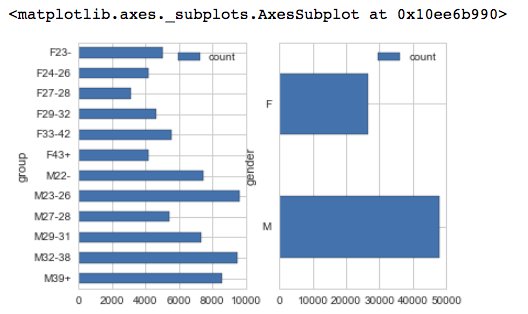
### 

### 6.2.9 Explore App Categories



### 6.2.10 Explore Gender & Age

|  |
| --- |
| gender\_age\_train = spark.read.csv(gender\_age\_train\_csv, header=True, mode="DROPMALFORMED", inferSchema='true', encoding="utf-8").persist()  print gender\_age\_train.count()  gender\_age\_train.show()  fig, axs = plt.subplots(1,2)  gender\_age\_train.groupby("group").count().rdd.sortBy(lambda x: x[0], ascending=False).toDF().toPandas().set\_index(["group"]).plot(ax=axs[0], kind='barh')  gender\_age\_train.groupby("gender").count().rdd.sortBy(lambda x: x[0], ascending=False).toDF().toPandas().set\_index(["gender"]).plot(ax=axs[1], kind='barh') |



### 6.2.11 Mobile App Categories

* First there are 930 different label categories for apps. This is way to many features to look at for this type of prediction.
* Looking at iTunes they roll their categories up to 25 different categories of apps.
* Added an unknown category because TalkingData has a bunch of categories that don't map to anything, or are simply marked as "unknown"
* Going to pain stakingly convert 930 categories to 26 categories manually.
  + New label will be label\_itunes\_id

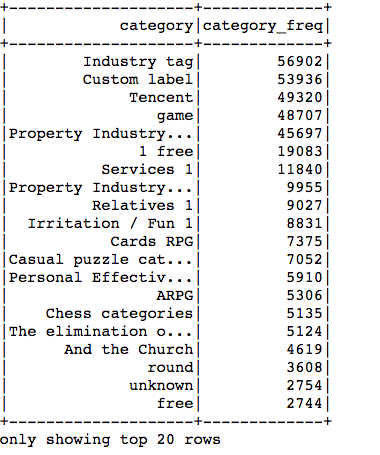
## 

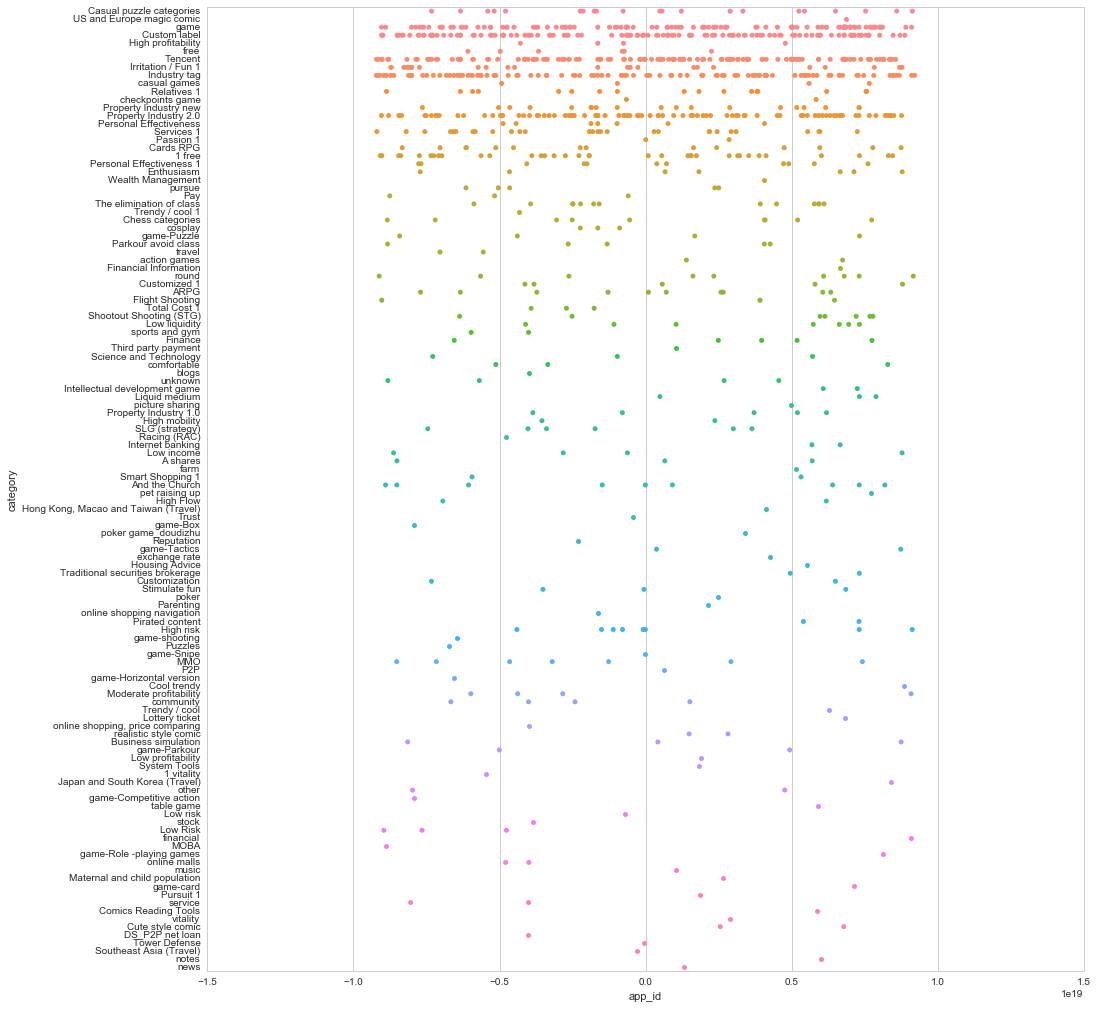
#### 6.2.11.1 Converted all 930 app categories into 25

## 

#### 6.2.11.2 Looking at Talking Data Mobile App Categories

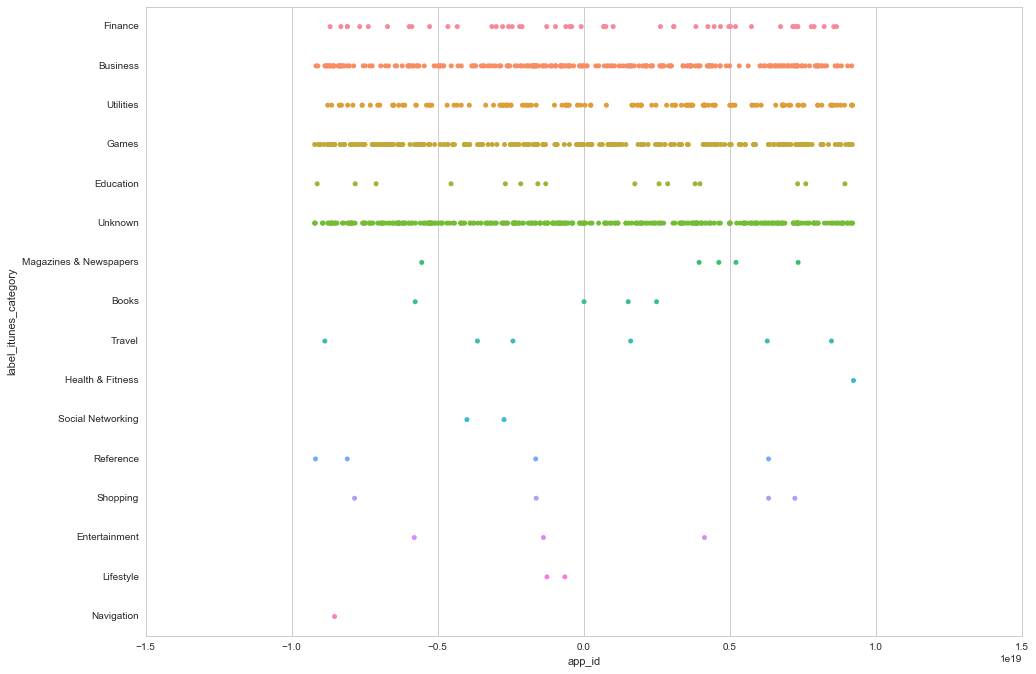
|  |
| --- |
| app\_cat\_freq = app\_labels.groupby("category") \  .agg(count(lit(1)) \  .alias("category\_freq")) \  .sort(col("category\_freq").desc())  app\_cat\_freq.show(20)  # sns.countplot(x="label\_itunes\_category", data=label\_categories, palette="Greens\_d");  app\_labels\_pandas = app\_labels.toPandas()  app\_labels\_pandas = app\_labels\_pandas.sample(1000, replace=True)  a4\_dims = (15.7, 17.27)  # df = mylib.load\_data()  fig, ax = pyplot.subplots(figsize=a4\_dims)  sns.stripplot(x="app\_id", y="category", data=app\_labels\_pandas); |





#### 6.2.11.3 Looking at New Categories

|  |
| --- |
| app\_cat\_itunes\_freq = app\_labels.groupby("label\_itunes\_category") \  .agg(count(lit(1)) \  .alias("category\_freq")) \  .sort(col("category\_freq").desc())  app\_cat\_itunes\_freq.show(100)  app\_labels\_pandas = app\_labels.toPandas()  app\_labels\_pandas = app\_labels\_pandas.sample(1000, replace=True)  a4\_dims = (15.7, 11.27)  fig, ax = pyplot.subplots(figsize=a4\_dims)  sns.stripplot(x="app\_id", y="label\_itunes\_category", data=app\_labels\_pandas); |



## 6.3 Data Transformation

### 6.3.1 Combining Datasets

I combined all datasets so I could view the data from a holistic point of view

|  |
| --- |
| combined\_data\_df = phone\_brand\_model.join(gender\_age\_train, "device\_id", "left\_outer") \  .join(events, "device\_id", "left\_outer") \  .join(app\_events, "event\_id", "left\_outer") \  .join(app\_labels, "app\_id", "left\_outer") \  .join(label\_categories, "label\_id", "left\_outer") \  .persist()  combined\_data\_df.show() |

## 

### 6.3.1 Sampling Datasets

### Data had 207M rows of data. To much to process on my local machine. Sampling 0.01 (1%) of the data.

|  |
| --- |
| combined\_data\_cleaned = combined\_data\_df.sample(False, 0.01, seed=0).persist()  print combined\_data\_cleaned.count()  combined\_data\_cleaned.show() |

### 6.3.2 Data Preparation for Modeling

Created a base set of features that include:

* Event counts
* Day of week counts
* Weekend counts
* Weekday counts
* Time of day counts
* Latitude seen counts
* Longitude seen counts
* Latitude and latitude seen counts

|  |
| --- |
| events\_features = combined\_data\_df.select( \  col("event\_id"),  col("device\_id"),  col("app\_id"),  when(col("latitude") != 0.0, 1).otherwise(lit(0)).alias("lat\_count"),  when((col("longitude") != 0.0), 1).otherwise(lit(0)).alias("lng\_count"),  when((col("latitude") != 0.0) & (col("longitude") != 0.0), 1).otherwise(lit(0)).alias("lat\_lng\_count"),  hour(col("timestamp")).alias("hour"),  when(date\_format("timestamp", 'E') == 'Mon', lit(1)).otherwise(lit(0)).alias("mon\_count"),  when(date\_format("timestamp", 'E') == 'Tue', 1).otherwise(0).alias("tue\_count"),  when(date\_format("timestamp", 'E') == 'Wed', 1).otherwise(0).alias("wed\_count"),  when(date\_format("timestamp", 'E') == 'Thu', 1).otherwise(0).alias("thu\_count"),  when(date\_format("timestamp", 'E') == 'Fri', 1).otherwise(0).alias("fri\_count"),  when(date\_format("timestamp", 'E') == 'Sat', 1).otherwise(0).alias("sat\_count"),  when(date\_format("timestamp", 'E') == 'Sun', 1).otherwise(0).alias("sun\_count"),  when((date\_format("timestamp", 'E') == 'Sat') | (date\_format("timestamp", 'E') == 'Sun'), 1).otherwise(0).alias("weekend\_count"),  when((date\_format("timestamp", 'E') != 'Sat') & (date\_format("timestamp", 'E') != 'Sun'), 1).otherwise(0).alias("weekday\_count"),  when(date\_format(col("timestamp"),"a") == "AM",1).otherwise(0).alias("am\_count"),  when(date\_format(col("timestamp"),"a") == "PM",1).otherwise(0).alias("pm\_count"),  when(date\_format(col("timestamp"),"HH") == 0,1).otherwise(0).alias("h0\_count"),  when(date\_format(col("timestamp"),"HH") == 1,1).otherwise(0).alias("h1\_count"),  when(date\_format(col("timestamp"),"HH") == 2,1).otherwise(0).alias("h2\_count"),  when(date\_format(col("timestamp"),"HH") == 3,1).otherwise(0).alias("h3\_count"),  when(date\_format(col("timestamp"),"HH") == 4,1).otherwise(0).alias("h4\_count"),  when(date\_format(col("timestamp"),"HH") == 5,1).otherwise(0).alias("h5\_count"),  when(date\_format(col("timestamp"),"HH") == 6,1).otherwise(0).alias("h6\_count"),  when(date\_format(col("timestamp"),"HH") == 7,1).otherwise(0).alias("h7\_count"),  when(date\_format(col("timestamp"),"HH") == 8,1).otherwise(0).alias("h8\_count"),  when(date\_format(col("timestamp"),"HH") == 9,1).otherwise(0).alias("h9\_count"),  when(date\_format(col("timestamp"),"HH") == 10,1).otherwise(0).alias("h10\_count"),  when(date\_format(col("timestamp"),"HH") == 11,1).otherwise(0).alias("h11\_count"),  when(date\_format(col("timestamp"),"HH") == 12,1).otherwise(0).alias("h12\_count"),  when(date\_format(col("timestamp"),"HH") == 13,1).otherwise(0).alias("h13\_count"),  when(date\_format(col("timestamp"),"HH") == 14,1).otherwise(0).alias("h14\_count"),  when(date\_format(col("timestamp"),"HH") == 15,1).otherwise(0).alias("h15\_count"),  when(date\_format(col("timestamp"),"HH") == 16,1).otherwise(0).alias("h16\_count"),  when(date\_format(col("timestamp"),"HH") == 17,1).otherwise(0).alias("h17\_count"),  when(date\_format(col("timestamp"),"HH") == 18,1).otherwise(0).alias("h18\_count"),  when(date\_format(col("timestamp"),"HH") == 19,1).otherwise(0).alias("h19\_count"),  when(date\_format(col("timestamp"),"HH") == 20,1).otherwise(0).alias("h20\_count"),  when(date\_format(col("timestamp"),"HH") == 21,1).otherwise(0).alias("h21\_count"),  when(date\_format(col("timestamp"),"HH") == 22,1).otherwise(0).alias("h22\_count"),  when(date\_format(col("timestamp"),"HH") == 23,1).otherwise(0).alias("h23\_count")  ) \  .groupby("device\_id") \  .agg(  approx\_count\_distinct("event\_id").alias("events\_per\_device\_count"),  approx\_count\_distinct("app\_id").alias("apps\_per\_device\_count"),  (approx\_count\_distinct("app\_id") / approx\_count\_distinct("event\_id")).alias("apps\_per\_event\_avg"),  func.sum("lat\_count").alias("lat\_count"),  func.sum("lng\_count").alias("lng\_count"),  func.sum("lat\_lng\_count").alias("lat\_lng\_count"),  min("hour").alias("min\_hour"),  max("hour").alias("max\_hour"),  func.sum("mon\_count").alias("mon\_count"),  func.sum("tue\_count").alias("tue\_count"),  func.sum("wed\_count").alias("wed\_count"),  func.sum("thu\_count").alias("thu\_count"),  func.sum("fri\_count").alias("fri\_count"),  func.sum("sat\_count").alias("sat\_count"),  func.sum("sun\_count").alias("sun\_count"),  func.sum("weekend\_count").alias("weekend\_count"),  func.sum("weekday\_count").alias("weekday\_count"),  func.sum("am\_count").alias("am\_count"),  func.sum("pm\_count").alias("pm\_count"),  func.sum("h0\_count").alias("h0\_count"),  func.sum("h1\_count").alias("h1\_count"),  func.sum("h2\_count").alias("h2\_count"),  func.sum("h3\_count").alias("h3\_count"),  func.sum("h4\_count").alias("h4\_count"),  func.sum("h5\_count").alias("h5\_count"),  func.sum("h6\_count").alias("h6\_count"),  func.sum("h7\_count").alias("h7\_count"),  func.sum("h8\_count").alias("h8\_count"),  func.sum("h9\_count").alias("h9\_count"),  func.sum("h10\_count").alias("h10\_count"),  func.sum("h11\_count").alias("h11\_count"),  func.sum("h12\_count").alias("h12\_count"),  func.sum("h13\_count").alias("h13\_count"),  func.sum("h14\_count").alias("h14\_count"),  func.sum("h15\_count").alias("h15\_count"),  func.sum("h16\_count").alias("h16\_count"),  func.sum("h17\_count").alias("h17\_count"),  func.sum("h18\_count").alias("h18\_count"),  func.sum("h19\_count").alias("h19\_count"),  func.sum("h20\_count").alias("h20\_count"),  func.sum("h21\_count").alias("h21\_count"),  func.sum("h22\_count").alias("h22\_count"),  func.sum("h23\_count").alias("h23\_count")  ) \  .na.fill(0) \  .persist()  events\_features.show() |

### 6.3.3 Index and encode categorical features for input into modeling functions

|  |
| --- |
| from pyspark.ml import Pipeline  from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorIndexer  sI1 = StringIndexer(inputCol="device\_id", outputCol="device\_index").setHandleInvalid("keep")  en1 = OneHotEncoder(dropLast=False, inputCol="device\_index", outputCol="device\_vec")  sI2 = StringIndexer(inputCol="events\_per\_device\_count", outputCol="events\_per\_device\_count\_index").setHandleInvalid("keep")  en2 = OneHotEncoder(dropLast=False, inputCol="events\_per\_device\_count\_index", outputCol="events\_per\_device\_count\_vec")  sI3 = StringIndexer(inputCol="apps\_per\_device\_count", outputCol="apps\_per\_device\_count\_index").setHandleInvalid("keep")  en3 = OneHotEncoder(dropLast=False, inputCol="apps\_per\_device\_count\_index", outputCol="apps\_per\_device\_count\_vec")  sI4 = StringIndexer(inputCol="apps\_per\_event\_avg", outputCol="apps\_per\_event\_avg\_index").setHandleInvalid("keep")  en4 = OneHotEncoder(dropLast=False, inputCol="apps\_per\_event\_avg\_index", outputCol="apps\_per\_event\_avg\_vec")  sI5 = StringIndexer(inputCol="lat\_count", outputCol="lat\_count\_index").setHandleInvalid("keep")  en5 = OneHotEncoder(dropLast=False, inputCol="lat\_count\_index", outputCol="lat\_count\_vec")  sI6 = StringIndexer(inputCol="lng\_count", outputCol="lng\_count\_index").setHandleInvalid("keep")  en6 = OneHotEncoder(dropLast=False, inputCol="lng\_count\_index", outputCol="lng\_count\_vec")  sI7 = StringIndexer(inputCol="lat\_lng\_count", outputCol="lat\_lng\_count\_index").setHandleInvalid("keep")  en7 = OneHotEncoder(dropLast=False, inputCol="lat\_lng\_count\_index", outputCol="lat\_lng\_count\_vec")  sI8 = StringIndexer(inputCol="geo\_found", outputCol="geo\_found\_index").setHandleInvalid("keep")  en8 = OneHotEncoder(dropLast=False, inputCol="geo\_found\_index", outputCol="geo\_found\_vec")  sI9 = StringIndexer(inputCol="min\_hour", outputCol="min\_hour\_index").setHandleInvalid("keep")  en9 = OneHotEncoder(dropLast=False, inputCol="min\_hour\_index", outputCol="min\_hour\_vec")  sI10 = StringIndexer(inputCol="max\_hour", outputCol="max\_hour\_index").setHandleInvalid("keep")  en10 = OneHotEncoder(dropLast=False, inputCol="max\_hour\_index", outputCol="max\_hour\_vec")  sI11 = StringIndexer(inputCol="mon\_count", outputCol="mon\_count\_index").setHandleInvalid("keep")  en11 = OneHotEncoder(dropLast=False, inputCol="mon\_count\_index", outputCol="mon\_count\_vec")  sI12 = StringIndexer(inputCol="tue\_count", outputCol="tue\_count\_index").setHandleInvalid("keep")  en12 = OneHotEncoder(dropLast=False, inputCol="tue\_count\_index", outputCol="tue\_count\_vec")  sI13 = StringIndexer(inputCol="wed\_count", outputCol="wed\_count\_index").setHandleInvalid("keep")  en13 = OneHotEncoder(dropLast=False, inputCol="wed\_count\_index", outputCol="wed\_count\_vec")  sI14 = StringIndexer(inputCol="thu\_count", outputCol="thu\_count\_index").setHandleInvalid("keep")  en14 = OneHotEncoder(dropLast=False, inputCol="thu\_count\_index", outputCol="thu\_count\_vec")  sI15 = StringIndexer(inputCol="fri\_count", outputCol="fri\_count\_index").setHandleInvalid("keep")  en15 = OneHotEncoder(dropLast=False, inputCol="fri\_count\_index", outputCol="fri\_count\_vec")  sI16 = StringIndexer(inputCol="sat\_count", outputCol="sat\_count\_index").setHandleInvalid("keep")  en16 = OneHotEncoder(dropLast=False, inputCol="sat\_count\_index", outputCol="sat\_count\_vec")  sI17 = StringIndexer(inputCol="sun\_count", outputCol="sun\_count\_index").setHandleInvalid("keep")  en17 = OneHotEncoder(dropLast=False, inputCol="sun\_count\_index", outputCol="sun\_count\_vec")  sI18 = StringIndexer(inputCol="weekend\_count", outputCol="weekend\_count\_index").setHandleInvalid("keep")  en18 = OneHotEncoder(dropLast=False, inputCol="weekend\_count\_index", outputCol="weekend\_count\_vec")  sI19 = StringIndexer(inputCol="weekday\_count", outputCol="weekday\_count\_index").setHandleInvalid("keep")  en19 = OneHotEncoder(dropLast=False, inputCol="weekday\_count\_index", outputCol="weekday\_count\_vec")  sI20 = StringIndexer(inputCol="am\_count", outputCol="am\_count\_index").setHandleInvalid("keep")  en20 = OneHotEncoder(dropLast=False, inputCol="am\_count\_index", outputCol="am\_count\_vec")  sI21 = StringIndexer(inputCol="pm\_count", outputCol="pm\_count\_index").setHandleInvalid("keep")  en21 = OneHotEncoder(dropLast=False, inputCol="pm\_count\_index", outputCol="pm\_count\_vec")  sI22 = StringIndexer(inputCol="h0\_count", outputCol="h0\_count\_index").setHandleInvalid("keep")  en22 = OneHotEncoder(dropLast=False, inputCol="h0\_count\_index", outputCol="h0\_count\_vec")  sI23 = StringIndexer(inputCol="h1\_count", outputCol="h1\_count\_index").setHandleInvalid("keep")  en23 = OneHotEncoder(dropLast=False, inputCol="h1\_count\_index", outputCol="h1\_count\_vec")  sI24 = StringIndexer(inputCol="h2\_count", outputCol="h2\_count\_index").setHandleInvalid("keep")  en24 = OneHotEncoder(dropLast=False, inputCol="h2\_count\_index", outputCol="h2\_count\_vec")  sI25 = StringIndexer(inputCol="h3\_count", outputCol="h3\_count\_index").setHandleInvalid("keep")  en25 = OneHotEncoder(dropLast=False, inputCol="h3\_count\_index", outputCol="h3\_count\_vec")  sI26 = StringIndexer(inputCol="h4\_count", outputCol="h4\_count\_index").setHandleInvalid("keep")  en26 = OneHotEncoder(dropLast=False, inputCol="h4\_count\_index", outputCol="h4\_count\_vec")  sI27 = StringIndexer(inputCol="h5\_count", outputCol="h5\_count\_index").setHandleInvalid("keep")  en27 = OneHotEncoder(dropLast=False, inputCol="h5\_count\_index", outputCol="h5\_count\_vec")  sI28 = StringIndexer(inputCol="h6\_count", outputCol="h6\_count\_index").setHandleInvalid("keep")  en28 = OneHotEncoder(dropLast=False, inputCol="h6\_count\_index", outputCol="h6\_count\_vec")  sI29 = StringIndexer(inputCol="h7\_count", outputCol="h7\_count\_index").setHandleInvalid("keep")  en29 = OneHotEncoder(dropLast=False, inputCol="h7\_count\_index", outputCol="h7\_count\_vec")  sI30 = StringIndexer(inputCol="h8\_count", outputCol="h8\_count\_index").setHandleInvalid("keep")  en30 = OneHotEncoder(dropLast=False, inputCol="h8\_count\_index", outputCol="h8\_count\_vec")  sI31 = StringIndexer(inputCol="h9\_count", outputCol="h9\_count\_index").setHandleInvalid("keep")  en31 = OneHotEncoder(dropLast=False, inputCol="h9\_count\_index", outputCol="h9\_count\_vec")  sI32 = StringIndexer(inputCol="h10\_count", outputCol="h10\_count\_index").setHandleInvalid("keep")  en32 = OneHotEncoder(dropLast=False, inputCol="h10\_count\_index", outputCol="h10\_count\_vec")  sI33 = StringIndexer(inputCol="h11\_count", outputCol="h11\_count\_index").setHandleInvalid("keep")  en33 = OneHotEncoder(dropLast=False, inputCol="h11\_count\_index", outputCol="h11\_count\_vec")  sI34 = StringIndexer(inputCol="h12\_count", outputCol="h12\_count\_index").setHandleInvalid("keep")  en34 = OneHotEncoder(dropLast=False, inputCol="h12\_count\_index", outputCol="h12\_count\_vec")  sI35 = StringIndexer(inputCol="h13\_count", outputCol="h13\_count\_index").setHandleInvalid("keep")  en35 = OneHotEncoder(dropLast=False, inputCol="h13\_count\_index", outputCol="h13\_count\_vec")  sI36 = StringIndexer(inputCol="h14\_count", outputCol="h14\_count\_index").setHandleInvalid("keep")  en36 = OneHotEncoder(dropLast=False, inputCol="h14\_count\_index", outputCol="h14\_count\_vec")  sI37 = StringIndexer(inputCol="h15\_count", outputCol="h15\_count\_index").setHandleInvalid("keep")  en37 = OneHotEncoder(dropLast=False, inputCol="h15\_count\_index", outputCol="h15\_count\_vec")  sI38 = StringIndexer(inputCol="h16\_count", outputCol="h16\_count\_index").setHandleInvalid("keep")  en38 = OneHotEncoder(dropLast=False, inputCol="h16\_count\_index", outputCol="h16\_count\_vec")  sI39 = StringIndexer(inputCol="h17\_count", outputCol="h17\_count\_index").setHandleInvalid("keep")  en39 = OneHotEncoder(dropLast=False, inputCol="h17\_count\_index", outputCol="h17\_count\_vec")  sI40 = StringIndexer(inputCol="h18\_count", outputCol="h18\_count\_index").setHandleInvalid("keep")  en40 = OneHotEncoder(dropLast=False, inputCol="h18\_count\_index", outputCol="h18\_count\_vec")  sI41 = StringIndexer(inputCol="h19\_count", outputCol="h19\_count\_index").setHandleInvalid("keep")  en41 = OneHotEncoder(dropLast=False, inputCol="h19\_count\_index", outputCol="h19\_count\_vec")  sI42 = StringIndexer(inputCol="h20\_count", outputCol="h20\_count\_index").setHandleInvalid("keep")  en42 = OneHotEncoder(dropLast=False, inputCol="h20\_count\_index", outputCol="h20\_count\_vec")  sI43 = StringIndexer(inputCol="h21\_count", outputCol="h21\_count\_index").setHandleInvalid("keep")  en43 = OneHotEncoder(dropLast=False, inputCol="h21\_count\_index", outputCol="h21\_count\_vec")  sI44 = StringIndexer(inputCol="h22\_count", outputCol="h22\_count\_index").setHandleInvalid("keep")  en44 = OneHotEncoder(dropLast=False, inputCol="h22\_count\_index", outputCol="h22\_count\_vec")  sI45 = StringIndexer(inputCol="h23\_count", outputCol="h23\_count\_index").setHandleInvalid("keep")  en45 = OneHotEncoder(dropLast=False, inputCol="h23\_count\_index", outputCol="h23\_count\_vec") |

### 6.3.4 Apply Transformations For Training Data

|  |
| --- |
| new\_train = Pipeline(stages=[  sI1, en1,  sI2, en2,  sI3, en3,  sI4, en4,  sI5, en5,  sI6, en6,  sI7, en7,  sI8, en8,  sI9, en9,  sI10, en10,  sI11, en11,  sI12, en12,  sI13, en13,  sI14, en14,  sI15, en15,  sI16, en16,  sI17, en17,  sI18, en18,  sI19, en19,  sI20, en20,  sI21, en21,  sI22, en22,  sI23, en23,  sI24, en24,  sI25, en25,  sI26, en26,  sI27, en27,  sI28, en28,  sI29, en29,  sI30, en30,  sI31, en31,  sI32, en32,  sI33, en33,  sI34, en34,  sI35, en35,  sI36, en36,  sI37, en37,  sI38, en38,  sI39, en39,  sI40, en40,  sI41, en41,  sI42, en42,  sI43, en43,  sI44, en44,  sI45, en45  ]).fit(events\_features\_new).transform(events\_features\_new);  new\_train.show() |

### 6.3.5 Casting Datatypes for Saving Features

Spark at this point did not like the vector types I created and could not write the CSV so here I am casting the types so it can be saved for modeling later.

|  |
| --- |
| # https://stackoverflow.com/questions/42641296/casting-multiple-columns-astype  new\_train\_features = new\_train.select(  new\_train["device\_id"].cast(StringType()).alias("device\_id"),  new\_train["events\_per\_device\_count"].cast(DoubleType()).alias("events\_per\_device\_count"),  new\_train["apps\_per\_device\_count"].cast(DoubleType()).alias("apps\_per\_device\_count"),  new\_train["apps\_per\_event\_avg"].cast(DoubleType()).alias("apps\_per\_event\_avg"),  new\_train["lat\_count"].cast(DoubleType()).alias("lat\_count"),  new\_train["lng\_count"].cast(DoubleType()).alias("lng\_count"),  new\_train["lat\_lng\_count"].cast(DoubleType()).alias("lat\_lng\_count"),  new\_train["min\_hour"].cast(DoubleType()).alias("min\_hour"),  new\_train["max\_hour"].cast(DoubleType()).alias("max\_hour"),  new\_train["mon\_count"].cast(DoubleType()).alias("mon\_count"),  new\_train["tue\_count"].cast(DoubleType()).alias("tue\_count"),  new\_train["wed\_count"].cast(DoubleType()).alias("wed\_count"),  new\_train["thu\_count"].cast(DoubleType()).alias("thu\_count"),  new\_train["fri\_count"].cast(DoubleType()).alias("fri\_count"),  new\_train["sat\_count"].cast(DoubleType()).alias("sat\_count"),  new\_train["sun\_count"].cast(DoubleType()).alias("sun\_count"),  new\_train["weekend\_count"].cast(DoubleType()).alias("weekend\_count"),  new\_train["weekday\_count"].cast(DoubleType()).alias("weekday\_count"),  new\_train["am\_count"].cast(DoubleType()).alias("am\_count"),  new\_train["pm\_count"].cast(DoubleType()).alias("pm\_count"),  new\_train["h0\_count"].cast(DoubleType()).alias("h0\_count"),  new\_train["h1\_count"].cast(DoubleType()).alias("h1\_count"),  new\_train["h2\_count"].cast(DoubleType()).alias("h2\_count"),  new\_train["h3\_count"].cast(DoubleType()).alias("h3\_count"),  new\_train["h4\_count"].cast(DoubleType()).alias("h4\_count"),  new\_train["h5\_count"].cast(DoubleType()).alias("h5\_count"),  new\_train["h6\_count"].cast(DoubleType()).alias("h6\_count"),  new\_train["h7\_count"].cast(DoubleType()).alias("h7\_count"),  new\_train["h8\_count"].cast(DoubleType()).alias("h8\_count"),  new\_train["h9\_count"].cast(DoubleType()).alias("h9\_count"),  new\_train["h10\_count"].cast(DoubleType()).alias("h10\_count"),  new\_train["h11\_count"].cast(DoubleType()).alias("h11\_count"),  new\_train["h12\_count"].cast(DoubleType()).alias("h12\_count"),  new\_train["h13\_count"].cast(DoubleType()).alias("h13\_count"),  new\_train["h14\_count"].cast(DoubleType()).alias("h14\_count"),  new\_train["h15\_count"].cast(DoubleType()).alias("h15\_count"),  new\_train["h16\_count"].cast(DoubleType()).alias("h16\_count"),  new\_train["h17\_count"].cast(DoubleType()).alias("h17\_count"),  new\_train["h18\_count"].cast(DoubleType()).alias("h18\_count"),  new\_train["h19\_count"].cast(DoubleType()).alias("h19\_count"),  new\_train["h20\_count"].cast(DoubleType()).alias("h20\_count"),  new\_train["h21\_count"].cast(DoubleType()).alias("h21\_count"),  new\_train["h22\_count"].cast(DoubleType()).alias("h22\_count"),  new\_train["h23\_count"].cast(DoubleType()).alias("h23\_count"),  new\_train["geo\_found"].cast(DoubleType()).alias("geo\_found"),  new\_train["label"].cast(DoubleType()).alias("label"),  new\_train["device\_index"].cast(DoubleType()).alias("device\_index"),  new\_train["events\_per\_device\_count\_index"].cast(DoubleType()).alias("events\_per\_device\_count\_index"),  new\_train["apps\_per\_device\_count\_index"].cast(DoubleType()).alias("apps\_per\_device\_count\_index"),  new\_train["apps\_per\_event\_avg\_index"].cast(DoubleType()).alias("apps\_per\_event\_avg\_index"),  new\_train["lat\_count\_index"].cast(DoubleType()).alias("lat\_count\_index"),  new\_train["lng\_count\_index"].cast(DoubleType()).alias("lng\_count\_index"),  new\_train["lat\_lng\_count\_index"].cast(DoubleType()).alias("lat\_lng\_count\_index"),  new\_train["geo\_found\_index"].cast(DoubleType()).alias("geo\_found\_index"),  new\_train["min\_hour\_index"].cast(DoubleType()).alias("min\_hour\_index"),  new\_train["max\_hour\_index"].cast(DoubleType()).alias("max\_hour\_index"),  new\_train["mon\_count\_index"].cast(DoubleType()).alias("mon\_count\_index"),  new\_train["tue\_count\_index"].cast(DoubleType()).alias("tue\_count\_index"),  new\_train["wed\_count\_index"].cast(DoubleType()).alias("wed\_count\_index"),  new\_train["thu\_count\_index"].cast(DoubleType()).alias("thu\_count\_index"),  new\_train["fri\_count\_index"].cast(DoubleType()).alias("fri\_count\_index"),  new\_train["sat\_count\_index"].cast(DoubleType()).alias("sat\_count\_index"),  new\_train["sun\_count\_index"].cast(DoubleType()).alias("sun\_count\_index"),  new\_train["weekend\_count\_index"].cast(DoubleType()).alias("weekend\_count\_index"),  new\_train["weekday\_count\_index"].cast(DoubleType()).alias("weekday\_count\_index"),  new\_train["am\_count\_index"].cast(DoubleType()).alias("am\_count\_index"),  new\_train["pm\_count\_index"].cast(DoubleType()).alias("pm\_count\_index"),  new\_train["h0\_count\_index"].cast(DoubleType()).alias("h0\_count\_index"),  new\_train["h1\_count\_index"].cast(DoubleType()).alias("h1\_count\_index"),  new\_train["h2\_count\_index"].cast(DoubleType()).alias("h2\_count\_index"),  new\_train["h3\_count\_index"].cast(DoubleType()).alias("h3\_count\_index"),  new\_train["h4\_count\_index"].cast(DoubleType()).alias("h4\_count\_index"),  new\_train["h5\_count\_index"].cast(DoubleType()).alias("h5\_count\_index"),  new\_train["h6\_count\_index"].cast(DoubleType()).alias("h6\_count\_index"),  new\_train["h7\_count\_index"].cast(DoubleType()).alias("h7\_count\_index"),  new\_train["h8\_count\_index"].cast(DoubleType()).alias("h8\_count\_index"),  new\_train["h9\_count\_index"].cast(DoubleType()).alias("h9\_count\_index"),  new\_train["h10\_count\_index"].cast(DoubleType()).alias("h10\_count\_index"),  new\_train["h11\_count\_index"].cast(DoubleType()).alias("h11\_count\_index"),  new\_train["h12\_count\_index"].cast(DoubleType()).alias("h12\_count\_index"),  new\_train["h13\_count\_index"].cast(DoubleType()).alias("h13\_count\_index"),  new\_train["h14\_count\_index"].cast(DoubleType()).alias("h14\_count\_index"),  new\_train["h15\_count\_index"].cast(DoubleType()).alias("h15\_count\_index"),  new\_train["h16\_count\_index"].cast(DoubleType()).alias("h16\_count\_index"),  new\_train["h17\_count\_index"].cast(DoubleType()).alias("h17\_count\_index"),  new\_train["h18\_count\_index"].cast(DoubleType()).alias("h18\_count\_index"),  new\_train["h19\_count\_index"].cast(DoubleType()).alias("h19\_count\_index"),  new\_train["h20\_count\_index"].cast(DoubleType()).alias("h20\_count\_index"),  new\_train["h21\_count\_index"].cast(DoubleType()).alias("h21\_count\_index"),  new\_train["h22\_count\_index"].cast(DoubleType()).alias("h22\_count\_index"),  new\_train["h23\_count\_index"].cast(DoubleType()).alias("h23\_count\_index")  ) |

### 6.3.6 Saving CSV for modeling

|  |
| --- |
| test\_events\_features\_new = test\_events\_features\_new.select('\*', (test\_events\_features\_new.geo\_found).alias('label')).persist() |

### 6.3.7 Prepare Testing Data

Repeat steps above to save data for testing data.

## 6.4 Modeling

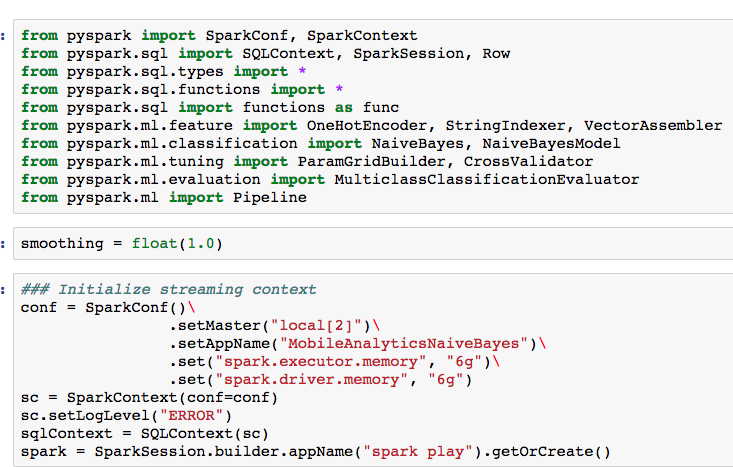
Start out with a Naïve Bayes Classifier (<https://en.wikipedia.org/wiki/Naive_Bayes_classifier>) because it is a good model for probabilistic classification.

I also provide an example of Decision Tree Classifier. The best results in my testing were when I set parameters to max\_bins=32,50, max\_depth=10, and max\_iterations=20, and entrophy=gini.

Random Forrest Classifier or LogisticRegressionWithLBFGS would probably yield more accurate results. Also, I noticed most people in the Kaggle competitions using XGBoost (<https://github.com/dmlc/xgboost>).

### 6.4.1 Naive Bayes

#### 6.4.1.1 Set up Spark and add Python Modules



#### 6.4.1.2 Import Data

|  |
| --- |
| train = spark.read.csv("data/features/train/part-00000-bd729f1f-8191-464e-bae5-e18e9c4973fe-c000.csv", header=True, mode="DROPMALFORMED", inferSchema='true', encoding="utf-8").persist()  print train.count()  train.show()  test = spark.read.csv("data/features/test/part-00000-4a55161f-e77a-4886-b86e-d8eeced88ff7-c000.csv", header=True, mode="DROPMALFORMED", inferSchema='true', encoding="utf-8").persist()  print test.count()  test.show() |

#### 6.4.1.3 Vector Assembler

|  |
| --- |
| assembler = VectorAssembler(  inputCols=[  "device\_index",  "events\_per\_device\_count\_index",  "apps\_per\_device\_count\_index",  "apps\_per\_event\_avg\_index",  "lat\_count\_index",  "lng\_count\_index",  "lat\_lng\_count\_index",  "min\_hour\_index",  "max\_hour\_index",  "mon\_count\_index",  "tue\_count\_index",  "wed\_count\_index",  "thu\_count\_index",  "fri\_count\_index",  "sat\_count\_index",  "sun\_count\_index",  "weekend\_count\_index",  "weekday\_count\_index",  "am\_count\_index",  "pm\_count\_index",  "h0\_count\_index",  "h1\_count\_index",  "h2\_count\_index",  "h3\_count\_index",  "h4\_count\_index",  "h5\_count\_index",  "h6\_count\_index",  "h7\_count\_index",  "h8\_count\_index",  "h9\_count\_index",  "h10\_count\_index",  "h11\_count\_index",  "h12\_count\_index",  "h13\_count\_index",  "h14\_count\_index",  "h15\_count\_index",  "h16\_count\_index",  "h17\_count\_index",  "h18\_count\_index",  "h19\_count\_index",  "h20\_count\_index",  "h21\_count\_index",  "h22\_count\_index",  "h23\_count\_index"  ],  outputCol="features"  ) |

#### 6.4.1.3 Spark ML Naïve Bayes Modeling

|  |
| --- |
| nb = NaiveBayes(smoothing=1.0, modelType="multinomial")\  .setFeaturesCol("features")\  .setLabelCol("label")\  .setPredictionCol("prediction")\  .setProbabilityCol("probability")\  .setRawPredictionCol("confidence") |

#### 6.4.1.4 Pipeline

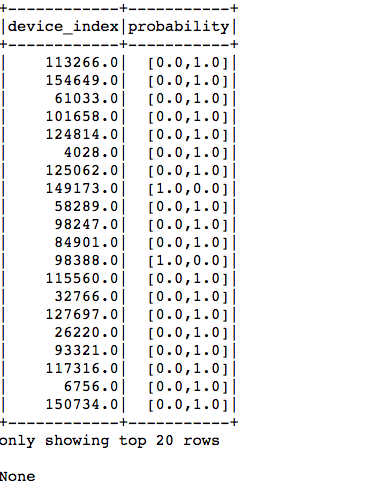
|  |
| --- |
| pipeline = Pipeline(stages=[assembler, nb]) |

#### 6.4.1.5 ParamGridBuilder & CrossValidator

|  |
| --- |
| # Iterate and choose the best fit model  cross\_validator = CrossValidator(estimator=pipeline,  estimatorParamMaps=params,  evaluator=MulticlassClassificationEvaluator(),  numFolds=5) # use 3+ folds in practice    cross\_validator\_model = cross\_validator.fit(new\_train) |

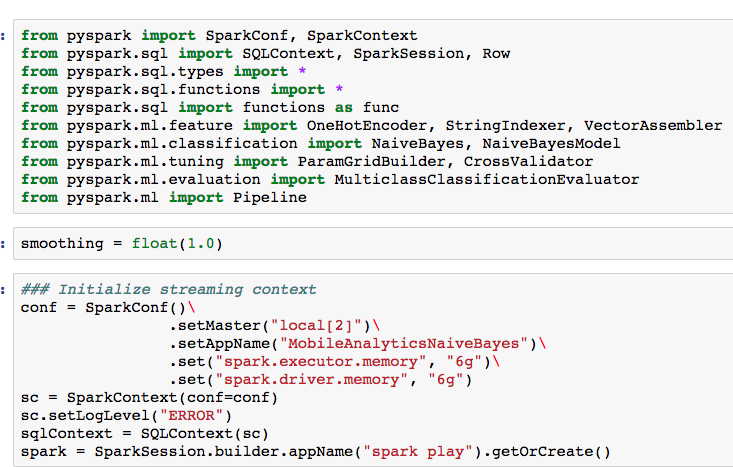
#### 6.4.1.5 Output

|  |
| --- |
| output = cross\_validator\_model.transform(new\_test)\  .select("device\_index","probability")  print output.show()  output.repartition(1).write.option("header", True).csv("data/output/nb") |



### 6.4.2 Decision Tree Classifier

#### 6.4.2.1 Set up Spark and add Python Modules



#### 6.4.2.2 Import Data

|  |
| --- |
| train = spark.read.csv("data/features/train/part-00000-bd729f1f-8191-464e-bae5-e18e9c4973fe-c000.csv", header=True, mode="DROPMALFORMED", inferSchema='true', encoding="utf-8").persist()  print train.count()  train.show()  test = spark.read.csv("data/features/test/part-00000-4a55161f-e77a-4886-b86e-d8eeced88ff7-c000.csv", header=True, mode="DROPMALFORMED", inferSchema='true', encoding="utf-8").persist()  print test.count()  test.show() |

#### 6.4.2.3 Vector Assembler

|  |
| --- |
| assembler = VectorAssembler(  inputCols=[  "device\_index",  "events\_per\_device\_count\_index",  "apps\_per\_device\_count\_index",  "apps\_per\_event\_avg\_index",  "lat\_count\_index",  "lng\_count\_index",  "lat\_lng\_count\_index",  "min\_hour\_index",  "max\_hour\_index",  "mon\_count\_index",  "tue\_count\_index",  "wed\_count\_index",  "thu\_count\_index",  "fri\_count\_index",  "sat\_count\_index",  "sun\_count\_index",  "weekend\_count\_index",  "weekday\_count\_index",  "am\_count\_index",  "pm\_count\_index",  "h0\_count\_index",  "h1\_count\_index",  "h2\_count\_index",  "h3\_count\_index",  "h4\_count\_index",  "h5\_count\_index",  "h6\_count\_index",  "h7\_count\_index",  "h8\_count\_index",  "h9\_count\_index",  "h10\_count\_index",  "h11\_count\_index",  "h12\_count\_index",  "h13\_count\_index",  "h14\_count\_index",  "h15\_count\_index",  "h16\_count\_index",  "h17\_count\_index",  "h18\_count\_index",  "h19\_count\_index",  "h20\_count\_index",  "h21\_count\_index",  "h22\_count\_index",  "h23\_count\_index"  ],  outputCol="features"  ) |

#### 6.4.1.3 Spark ML Decision Tree Classifier

|  |
| --- |
| dt = DecisionTreeClassifier()\  .setFeaturesCol("features")\  .setLabelCol("label")\  .setPredictionCol("prediction")\  .setProbabilityCol("probability")\  .setRawPredictionCol("confidence") |

#### 6.4.1.4 Pipeline

|  |
| --- |
| pipeline = Pipeline(stages=[assembler, dt]) |

#### 6.4.1.5 ParamGridBuilder & CrossValidator

|  |
| --- |
| # Iterate and choose the best fit model  params = ParamGridBuilder()\  .addGrid(dt.maxBins, [max\_bins\_1, max\_bins\_2])\  .addGrid(dt.maxDepth, [max\_depth])\  .addGrid(dt.impurity, ["entropy", "gini"]) \  .build()  # Iterate and choose the best fit model  cross\_validator = CrossValidator(estimator=pipeline,  estimatorParamMaps=params,  evaluator=MulticlassClassificationEvaluator(),  numFolds=5) # use 3+ folds in practice    cross\_validator\_model = cross\_validator.fit(train)  cross\_validator\_model = cross\_validator.fit(new\_train) |

#### 6.4.1.5 Output

|  |
| --- |
| output = cross\_validator\_model.transform(test)\  .select("device\_index","probability") |

### 

# 7. Summarize:

## 7.1 Lessons Learned:

Previous to this final I hadn’t really had to do any exploratory data analysis on an unknown dataset.   The final project has taught me to allow multiple days to explore, and get to really know a dataset before going into trying to create a model.  Next time I am testing out features in a model I will start out small.  Started out with way too many features and it was time consuming.

## 7.2 Pros

I finally got to deep dive into doing a prediction on my own.   I learned new things about Spark, Pandas, and Machine Learning.   Also, this helped me start to think about how to assemble data in a format that I could start to extract features for my models

## 7.3 Cons

This project has been super time consuming.   Leaving it up to the student to do anything within the topic ultimately took up a huge chunk of time.   Another big con was choosing to run Spark locally.   Doing my modeling using something like Zepplin on AWS would have really sped things up.

## 7.4 Next Steps

1. Exploratory Data Analysis
   1. Convert mobile app plots to Histograms <http://blog.madhukaraphatak.com/statistical-data-exploration-spark-part-2/>
   2. Look into Geospatial Data
2. Try different models for better results
   1. XG BOOST Classifier
      1. [http://xgboost.readthedocs.io/en/latest/model.html#objective-function-training-loss-regularization](http://xgboost.readthedocs.io/en/latest/model.html" \l "objective-function-training-loss-regularization" \t "_blank)
      2. <https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/>
      3. [https://www.kaggle.com/kopylovanton/fraud-detection-xgboost-lightgbm](https://www.kaggle.com/kopylovanton/fraud-detection-xgboost-lightgbm" \t "_blank)
   2. Random Forrest Classifier
   3. LogisticRegressionWithLBFGS

# 8 YouTube URLs

## 2 Minute

## [https://youtu.be/ajY8mxXueoU](https://youtu.be/ajY8mxXueoU" \t "_blank)

## 15 Minute

## https://youtu.be/bdbHq6n-a00

# 9 References

1. <https://www.kaggle.com/c/talkingdata-mobile-user-demographics#description>
2. https://gist.github.com/ololobus/4c221a0891775eaa86b0
3. <https://spark.apache.org/docs/1.1.0/mllib-guide.html>
4. <https://github.com/Azure/Azure-MachineLearning-DataScience/tree/master/Misc/Spark/pySpark>
5. <https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/spark-data-exploration-modeling>
6. <http://blog.madhukaraphatak.com/statistical-data-exploration-spark-part-2/>
7. [http://xgboost.readthedocs.io/en/latest/model.html#objective-function-training-loss-regularization](http://xgboost.readthedocs.io/en/latest/model.html" \l "objective-function-training-loss-regularization" \t "_blank)
8. [https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/](https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/" \t "_blank)
9. https://www.kaggle.com/kopylovanton/fraud-detection-xgboost-lightgbm