



EECS E6893 Big Data Analytics

HW1: Clustering, Classification, and Spark MLlib

Frank Ou Yang, ho2271@columbia.edu

Agenda

- Spark Dataframe
- Spark SQL
- Spark MLlib
- HW1
 - Iterative K-means clustering
 - Logistic Regression

Spark Dataframe

- An *abstraction*, an immutable distributed collection of data like RDD
- Data is organized into named columns, like a table in DB
- Create from RDD, Hive table, or other data sources
- Easy conversion with Pandas Dataframe

Spark Dataframe: read from csv file

```
# read data from csv into Dataframe
```

```
df = spark.read.format("csv").option("header", 'true').load("gs://big_data_ta/data/citibike_stations.csv")
```

```
type(df)
```

```
pyspark.sql.dataframe.DataFrame
```

```
df.show(1)
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|station_id|          name|short_name|  latitude|  longitude|region_id|rental_methods|capacity|eightd_has_key
_dispenser|num_bikes_available|num_bikes_disabled|num_docks_available|num_docks_disabled|is_installed|is_renting|is_r
eturning|eightd_has_available_keys|      last_reported|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|      382|University Pl & E...|  5905.11|40.73492695|-73.99200509|      71|KEY,CREDITCARD|      0|
false|      0|      0|      0|      0|      false|      false|      fa
lse|      false|1970-01-01 00:00:00|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

only showing top 1 row

Spark Dataframe: common operations

```
df.printSchema()
```

```
root
|-- station_id: string (nullable = true)
|-- name: string (nullable = true)
|-- short_name: string (nullable = true)
|-- latitude: string (nullable = true)
|-- longitude: string (nullable = true)
|-- region_id: string (nullable = true)
|-- rental_methods: string (nullable = true)
|-- capacity: string (nullable = true)
|-- eightd_has_key_dispenser: string (nullable = true)
|-- num_bikes_available: string (nullable = true)
|-- num_bikes_disabled: string (nullable = true)
|-- num_docks_available: string (nullable = true)
|-- num_docks_disabled: string (nullable = true)
|-- is_installed: string (nullable = true)
|-- is_renting: string (nullable = true)
|-- is_returning: string (nullable = true)
|-- eightd_has_available_keys: string (nullable = true)
|-- last_reported: string (nullable = true)
```

```
df.count()
```

Spark Dataframe: common operations

```
df.columns
```

```
['station_id',  
 'name',  
 'short_name',  
 'latitude',  
 'longitude',  
 'region_id',  
 'rental_methods',  
 'capacity',  
 'eightd_has_key_dispenser',  
 'num_bikes_available',  
 'num_bikes_disabled',  
 'num_docks_available',  
 'num_docks_disabled',  
 'is_installed',  
 'is_renting',  
 'is_returning',  
 'eightd_has_available_keys',  
 'last_reported']
```

Spark Dataframe: common operations

```
df.describe().show()
```

```
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+
|summary|station_id|name|short_name|latitude|longitude|reg
ion_id|rental_methods|capacity|eightd_has_key_dispenser|num_bikes_available|num_bikes_disabled|num_docks_av
ailable|num_docks_disabled|is_installed|is_renting|is_returning|eightd_has_available_keys|last_reported|
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+
|count|843|843|843|843|843|843|
843|843|843|843|843|843|843|
843|843|843|843|843|843|843|
|mean|2434.425860023725|null|5806.786515151487|40.73212559944772|-73.9749901186049|70.93950177
935943|null|31.419928825622776|null|14.565836298932384|0.5693950177935944|16.1897983
3926453|0.05219454329774614|null|null|null|null|null|
|stddev|1421.1204113008778|null|1175.6743390458948|0.0387451696148341|0.031207687758326202|0.23854913482
063428|null|12.052012437572532|null|11.188256063195926|0.8613434732614029|13.15807566
4204775|0.6499620307701626|null|null|null|null|null|
|min|116|1 Ave & E 110 St|3460.01|40.655399774478312|-73.9077436|
70|KEY,CREDITCARD|0|false|false|false|0|0|
0|0|false|false|false|false|1970-01-01 00:00:00|
|max|83|York St|JC106|40.814394437915816|-74.0836394|
71|KEY,CREDITCARD|79|true|true|9|6|
9|9|true|true|true|true|2019-09-02 00:00:00|
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+
```

Spark Dataframe: common operations

```
df.describe('capacity').show()
```

summary	capacity
count	843
mean	31.419928825622776
stddev	12.052012437572532
min	0
max	79

```
df.select('station_id').distinct().count()
```

843

Spark Dataframe: conversion with Pandas

```
# conversion with Pandas
```

```
import pandas as pd
```

```
pandaDf = df.toPandas()
```

```
pandaDf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 843 entries, 0 to 842
```

```
Data columns (total 18 columns):
```

```
station_id      843 non-null object
```

```
name            843 non-null object
```

```
short_name      843 non-null object
```

```
latitude        843 non-null object
```

```
longitude       843 non-null object
```

```
region_id      843 non-null object
```

```
rental_methods  843 non-null object
```

```
capacity        843 non-null object
```

```
eightd_has_key_dispenser 843 non-null object
```

```
num_bikes_available 843 non-null object
```

```
num_bikes_disabled 843 non-null object
```

```
num_docks_available 843 non-null object
```

```
num_docks_disabled 843 non-null object
```

```
is_installed     843 non-null object
```

```
is_renting       843 non-null object
```

```
is_returning     843 non-null object
```

```
eightd_has_available_keys 843 non-null object
```

```
last_reported    843 non-null object
```

```
dtypes: object(18)
```

```
memory usage: 118.6+ KB
```

Work with Spark SQL

```
# Play with Spark SQL  
# Register the DataFrame as a SQL temporary view  
df.createOrReplaceTempView("citibike")
```

```
sqlDF = spark.sql("""  
    SELECT COUNT (DISTINCT station_id)  
    FROM citibike  
    """)  
sqlDF.show()
```

```
+-----+  
|count(DISTINCT station_id)|  
+-----+  
|                        843|  
+-----+
```

```
# get data out of df  
sqlDF.select("count(DISTINCT station_id)").collect()[0]["count(DISTINCT station_id)"]
```

843

Spark MLlib

- Spark's scalable machine learning library
- Tools:
 - ML Algorithms: classification, regression, clustering, and collaborative filtering
 - Featurization: feature extraction, transformation, dimensionality reduction, and selection
 - Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
 - Persistence: saving and load algorithms, models, and Pipelines
 - Utilities: linear algebra, statistics, data handling, etc.

Example: K-means clustering with Spark MLlib

```
from pyspark.mllib.clustering import KMeans
```

```
clusters = KMeans.train(data, 10, maxIterations=20, initializationMode="random")
```

```
# cluster centers  
len(clusters.centers)
```

10

HW1

HW1

- Document clustering with K-means
 - “Implement” iterative K-means clustering in Spark
 - L1, L2 distance functions
 - Different initialization strategies
 - Plot the cluster assignment result with T-SNE dimensionality reduction
- Binary classification with Spark MLlib
 - Preprocess df with ML Pipeline
 - Logistic regression

Iterative K-means

- In each iteration, k centroids are initialized, each point in the space is assigned to the nearest centroid, and the centroids are re-computed
- Pseudo code:

Algorithm 1 Iterative k -Means Algorithm

```
1: procedure ITERATIVE  $k$ -MEANS
2:   Select  $k$  points as initial centroids of the  $k$  clusters.
3:   for iterations := 1 to MAX_ITER do
4:     for each point  $p$  in the dataset do
5:       Assign point  $p$  to the cluster with the closest centroid
6:     end for
7:     for each cluster  $c$  do
8:       Recompute the centroid of  $c$  as the mean of all the data points assigned to  $c$ 
9:     end for
10:  end for
11: end procedure
```

Iterative K-means in Spark

Hint:

Spark operations you might need:
map, reduceByKey, collect, keys

Algorithm 1 Iterative k -Means Algorithm

```
1: procedure ITERATIVE  $k$ -MEANS
2:   Select  $k$  points as initial centroids of the  $k$  clusters.
3:   for iterations := 1 to MAX_ITER do
4:     for each point  $p$  in the dataset do
5:       Assign point  $p$  to the cluster with the closest centroid
6:     end for
7:     for each cluster  $c$  do
8:       Recompute the centroid of  $c$  as the mean of all the data points assigned to  $c$ 
9:     end for
10:  end for
11: end procedure
```

```
# iterative k-means
for _ in range(MAX_ITER):
    # Transform each point to a combo of point, closest centroid, count
    # point -> (closest_centroid, (point, 1))

    # Re-compute cluster center

    # For each cluster center (key), aggregate its value by summing up points and count
    # Average the points for each centroid: divide sum of points by count
```


Document clustering

More From Medium

More from Wisecrack



Colonel Sanders Wants to Be Your Daddy



Amanda Sche... Wisecrack
Sep 13 · 5 min read ★



132



More from Wisecrack



Platforms Own You, Now What?



Wisecrack in Wisecrack
Sep 10 · 6 min read ★



590



More from Wisecrack



Millennials Are Shaking Up Wall Street, But Is It Helping Them?



Thomas Ambr... Wisecrack
Sep 12 · 9 min read ★



14



Two reasons why Antoine Griezmann has doubts over moving to Man United

independent.co.uk · 1h · Miguel Delaney an...



Azuz11

added to Soccer Player

Write a comment...



MORE STORIES:

Manchester United:
Jose Mourinho £85
Million Transfer
Target 'Agrees Per...

NEWSWEEK.COM

Mar
secu
with
Grie

INTE



COVER STORIES

Plot the result with t-SNE

```
from sklearn.manifold import TSNE
```

```
# RDD -> np array  
data_np = np.array(data.collect())
```

```
data_np.shape
```

```
(4601, 58)
```

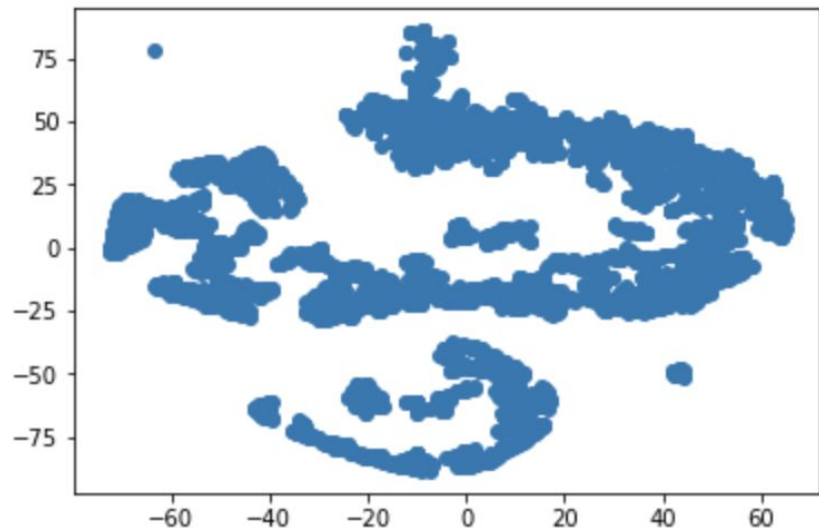
```
data_embedded = TSNE(n_components=2).fit_transform(data_np)
```

```
data_embedded.shape
```

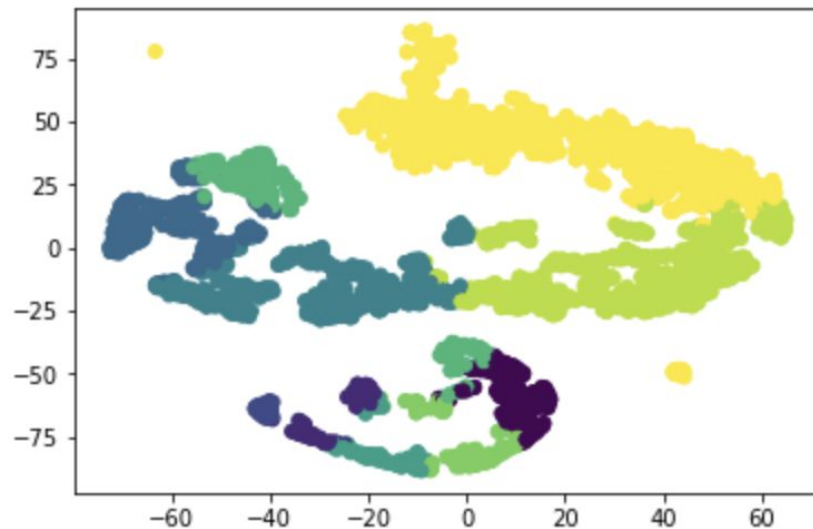
```
(4601, 2)
```

```
vis_x = data_embedded[:, 0]  
vis_y = data_embedded[:, 1]  
plt.scatter(vis_x, vis_y, cmap=plt.cm.get_cmap("jet", 10))  
plt.show()
```

Plot the result with t-SNE

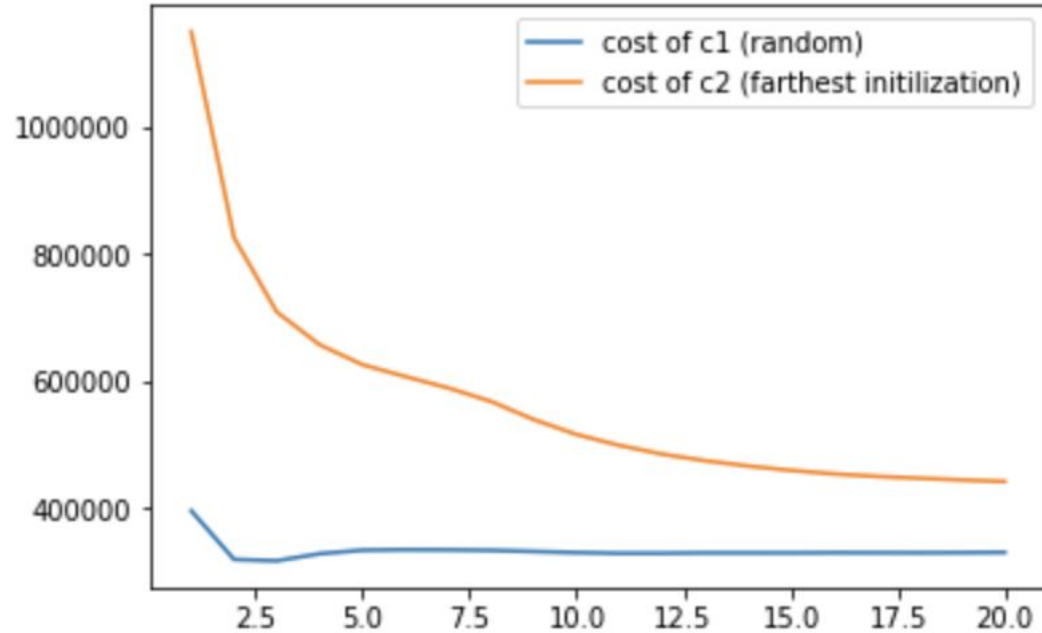


Before clustering



After clustering

Plot the cost of each iteration



Binary classification with Spark MLlib

- Adult dataset from UCI Machine Learning Repository
- Given information of a person, predict if the person could earn > 50k per year
- Workflow
 - Data loading: load data into Dataframe
 - Data preprocessing: Convert the categorical variables into numeric variables with ML Pipelines and Feature Transformers
 - Modelling: Conduct classification with Logistic Regression model
 - Evaluation

References

- <https://spark.apache.org/docs/latest/sql-getting-started.html>
- <https://www.analyticsvidhya.com/blog/2016/10/spark-dataframe-and-operations/>
- <https://spark.apache.org/docs/latest/ml-guide.html>