Section 4: Spark basics, matplotlib & MLlib

BE BOUNDLESS

Spark & RDD cheat sheet

UNIVERSITY *of* WASHINGTON

SPARK & RDD CHEAT SHEET

Spark & RDD Basics

Apache Spark

It is an open source, Hadoop compatible fast and expressive cluster computing platform

RDD

The core concept in Apache Spark is **RDD** (Resilient Distributed Datasheet) , which is an immutable distributed collection of data which is partitioned across machines in a cluster.

Transformation: It is an operation on an **RDD** such as filter (), map () or union () that yields another **RDD.**

Action: It is an operation that triggers a computation such as count (), first (), take(n) or collect ().

Partition: It is a logical division of data stored on a node in a cluster

Spark Context

It holds a connection with spark cluster management

Driver: The process of running the main () function of an application and creating the SparkContext is managed by driver

Worker: Any node which can run program on the cluster is called worker

Components of Spark

Executors: It consists of multiple tasks; basically it is a JVM process sitting on all nodes. Executors receive the tasks, deserialize it and run it as a task. Executors utilize cache so that so that the tasks can run faster.

Tasks: Jars along with the code is a task

Node: It comprises of a multiple executors

RDD: It is a big data structure which is used to represent data that cannot be stored on a single machine. Hence, the data is distributed, partitioned and split across the computers.

Input: Every **RDD** is made up of some input such as a text file, Hadoop file etc.

Output: An output of functions in Spark can produce **RDD,** it is functional as a function one after other receives an input RDD and outputs an output **RDD.**

Shared Variables on Spark

Broadcast variables: It is a read only variable which will be copied to the worker only once. It is similar to the Distributor cache in MapReduce. We can set, destroy and unpersist these values. It is used to save the copy of data across all the nodes

Example syntax:

```
broadcastVariable = sparkContext.broadcast(500)
broadcastVariable.value
```

Accumulators: The worker can only add using an associative operation, it is usually used in parallel sums and only a driver can read an accumulator value. It is same as counter in MapReduce. Basically, accumulators are variables that can be incremented in a distributed tasks and used for aggregating information

Example syntax:

```
exampleAccumulator = sparkContext.accumulator(1)
exampleAccumulator.add(5)
```

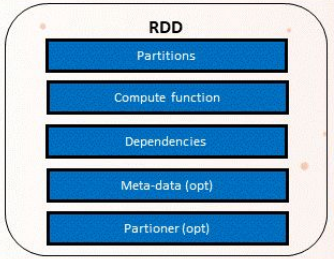
Unified Libraries in Spark

Spark SQL: It is a Spark module which allows working with structured data. The data querying is supported by **SQL** or **HQL**

Spark Streaming: It is used to build scalable application which provides fault tolerant streaming. It also processes in real time using web server logs, Facebook logs etc. in real time.

Mlib(Machine Learning): It is a scalable machine learning library and provides various algorithms for classification, regression, clustering etc.

Graph X: It is an **API** for graphs. This module can efficiently find the shortest path for static graphs.



Function Transformations	Description
map(function)	Returns a new **RDD** by applying function on element
filter(function)	Returns a new dataset formed by selecting those elements of the source on which function returns true
filterByRange(lower, upper)	Returns an **RDD** with elements in the specified range
flatMap(function)	It is similar to the map function but the function returns a sequence instead of a value
reduceByKey(function,[num Tasks])	It is used to aggregate values of a key using a function.
groupByKey([num Tasks])	To convert(K,V) to (K, <iterable V>)
distinct([num Tasks])	This is used to eliminated duplicates from **RDD**
mapPartitions(function)	Runs separately on each partition of **RDD**
mapPartitionsWithIndex(function)	Provides function with an integer value representing the index of the partition
sample(withReplacement, fraction, seed)	Samples a fraction of data using a given random number generating seeds
union()	This returns a new **RDD** containing all elements and arguments from source **RDD**
intersection()	Returns a new **RDD** that contains an intersection of elements in the datasets
Cartesian()	Returns the Cartesian product of all pair of elements
subtract()	New **RDD** created by removing the elements from the source **RDD** in common with arguments
join(RDD,[numTasks])	It joins two elements of the dataset with common arguments. When invoked on (A,B) and (A,C) it creates a new **RDD** (A,(B,C))
cogroup(RDD,[numTasks])	It converts (A,B) to (A, <iterable B>)

Cluster Manager

It is used to allocate resources to each application in a driver program. There are 3 types of cluster managers which are supported by **Apache Spark**

- Standalone
- Mesos
- Yarn

Action Functions	Description
count()	Get the number of data elements in the **RDD**
collect()	Get all the data elements in the **RDD** as an array
reduce(function)	It is used to aggregate data elements into the **RDD** by taking two arguments and returning one
take(n)	It is used to fetch the first n elements of the **RDD**
foreach(function)	It is used to execute function for each data element in the **RDD**
first()	It retrieves the first data element of the **RDD**
saveastextfile(path)	It is used to write the content of **RDD** to a text file or set of text files to the local system
takeordered(n, [ordering])	It will return the first n elements of **RDD** using either the natural order or a custom comparator

RDD Persistence Method Functions	Description
MEMORY_ONLY (default level)	It stores the **RDD** in an available cluster memory as deserialized Java object
MEMORY_AND_DISK	This will store **RDD** as a deserialized Java object. If the **RDD** does not fit in the cluster memory it stores the partitions on the disk and reads them
MEMORY_ONLY_SER	This stores **RDD** as a serialized Java object, this is more CPU intensive
MEMORY_ONLY_DISK_SER	This option is same as above but stores in a disk when the memory is not sufficient
DISC_ONLY	This option stores **RDD** only on the disk
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	This is same as the other levels except that the partitions are replicated on 2 slave nodes

Persistence Method Function	Description
cache()	It is used to avoid unnecessary recomputation . This is same as persist(MEMORY_ONLY)
persist([Storage Level])	Persisting the **RDD** with the default storage level
unpersist()	Marking the **RDD** as non persistent and removing the block from memory and disk
checkpoint()	It saves a file inside the checkpoint directory and all the reference of its parent **RDD** will be removed

FURTHERMORE:
Spark Certification Training Course

UNIVERSITY of WASHINGTON

matplotlib cheat sheet

UNIVERSITY *of* WASHINGTON

matplotlib

Cheat sheet — Version 3.5.0

Quick start

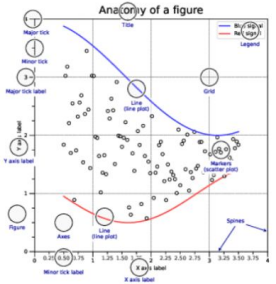
```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt


X = np.linspace(0, 2*np.pi, 100)
Y = np.cos(X)

fig, ax = plt.subplots()
ax.plot(X, Y, color='green')

fig.savefig("figure.pdf")
fig.show()
```

Anatomy of a figure



Subplots layout

```
subplot[s](rows,cols,…)
fig, axs = plt.subplots(3, 3)

G = gridspec(rows,cols,…)
ax = G[0,:]

ax.inset_axes(extent)

d=make_axes_locatable(ax)
ax = d.new_horizontal('10%')
```

Getting help

- matplotlib.org
- github.com/matplotlib/matplotlib/issues
- discourse.matplotlib.org
- stackoverflow.com/questions/tagged/matplotlib
- gitter.im/matplotlib
- twitter.com/matplotlib
- Matplotlib users mailing list

Basic plots

```
plot([X],Y,[fmt],…)
X, Y, fmt, color, marker, linestyle

scatter(X,Y,…)
X, Y, [s]izes, [c]olors, marker, cmap

bar[h](x,height,…)
x, height, width, bottom, align, color

imshow(Z,…)
Z, cmap, interpolation, extent, origin

contour[f]([X],[Y],Z,…)
X, Y, Z, levels, colors, extent, origin

pcolormesh([X],[Y],Z,…)
X, Y, Z, vmin, vmax, cmap

quiver([X],[Y],U,V,…)
X, Y, U, V, C, units, angles

pie(X,…)
Z, explode, labels, colors, radius

text(x,y,text,…)
x, y, text, va, ha, size, weight, transform

fill[_between][x](…)
X, Y1, Y2, color, where
```

Advanced plots

```
step(X,Y,[fmt],…)
X, Y, fmt, color, marker, where

boxplot(X,…)
X, notch, sym, bootstrap, widths

errorbar(X,Y,xerr,yerr,…)
X, Y, xerr, yerr, fmt

hist(X, bins, …)
X, bins, range, density, weights

violinplot(D,…)
D, positions, widths, vert

barbs([X],[Y], U, V, …)
X, Y, U, V, C, length, pivot, sizes

eventplot(positions,…)
positions, orientation, lineoffsets

hexbin(X,Y,C,…)
X, Y, C, gridsize, bins
```

Scales

```
ax.set_[xy]scale(scale,…)
linear  any values
log     values > 0
symlog  any values
logit   0 < values < 1
```

Projections

```
subplot(…,projection=p)
p='polar'
p='3d'
p=Orthographic()
from cartopy.crs import Cartographic
```

Lines

```
linestyle or ls
"-"  ":"  "--"  "-."  (0,(0.01,2))

capstyle or dash_capstyle
"butt"  "round"  "projecting"
```

Markers

```
'.' 'o' 's' 'p' 'X' '*' 'p' 'D' '<' '>' '^' 'v'
'1' '2' '3' '4' '+' 'x' '|' '_' 4 5 6 7
markevery
10   [0, -1]   (25, 5)   [0, 25, -1]
```

Colors

```
'Cn'  'x'  'name'  (R,G,B[,A])  '#RRGGBB[AA]'  'x.y'
```

Colormaps

```
plt.get_cmap(name)
```

Uniform: viridis, magma, plasma

Sequential: Greys, YlOrBr, Wistia

Diverging: Spectral, coolwarm, RdGy

Qualitative: tab10, tab20

Cyclic: twilight

Tick locators

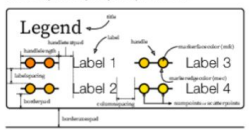
```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_locator(locator)
ticker.NullLocator()
ticker.MultipleLocator(0.5)
ticker.FixedLocator([0, 1, 5])
ticker.LinearLocator(numticks=3)
ticker.IndexLocator(base=0.5, offset=0.25)
ticker.AutoLocator()
ticker.MaxNLocator(n=4)
ticker.LogLocator(base=10, numticks=15)
```

Tick formatters

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_formatter(formatter)
ticker.NullFormatter()
ticker.FixedFormatter(['zero', 'one', 'two', …])
ticker.FuncFormatter(lambda x, pos: "[%.2f]" % x)
ticker.FormatStrFormatter('>%d<')
ticker.ScalarFormatter()
ticker.StrMethodFormatter('{x}')
ticker.PercentFormatter(xmax=5)
```

Ornaments

```
ax.legend(…)
handles, labels, loc, title, frameon
```




```
ax.colorbar(…)
mappable, ax, cax, orientation
```

```
ax.annotate(…)
text, xy, xytext, xycoords, textcoords, arrowprops
```

Event handling

```
fig, ax = plt.subplots()
def on_click(event):
  print(event)
fig.canvas.mpl_connect(
  'button_press_event', on_click)
```

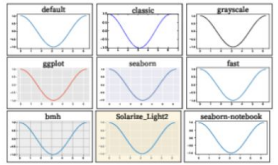
Animation

```
import matplotlib.animation as mpla

T = np.linspace(0, 2*np.pi, 100)
S = np.sin(T)
line, = plt.plot(T, S)
def animate(i):
    line.set_ydata(np.sin(T+i/50))
anim = mpla.FuncAnimation(
    plt.gcf(), animate, interval=5)
plt.show()
```

Styles

```
plt.style.use(style)
```



Quick reminder

```
ax.grid()
ax.set_[xy]lim(vmin, vmax)
ax.set_[xy]label(label)
ax.set_[xy]ticks(ticks, [labels])
ax.set_[xy]ticklabels(labels)
ax.set_title(title)
ax.tick_params(width=10, …)
ax.set_axis_[on|off]()

fig.suptitle(title)
fig.tight_layout()
plt.gcf(), plt.gca()
mpl.rc('axes', linewidth=1, …)
[fig|ax].patch.set_alpha(0)
text=r'$\frac{-e^{i\pi}}{2^n}$'
```

Keyboard shortcuts

ctrl + s Save	ctrl + w Close plot
r Reset view	f Fullscreen 0/1
f View forward	b View back
p Pan view	o Zoom to rect
x X pan/zoom	y Y pan/zoom
g Minor grid 0/1	G Major grid 0/1
l X axis log/linear	L Y axis log/linear

Ten simple rules

1. Know Your Audience
2. Identify Your Message
3. Adapt the Figure
4. Captions Are Not Optional
5. Do Not Trust the Defaults
6. Use Color Effectively
7. Do Not Mislead the Reader
8. Avoid "Chartjunk"
9. Message Trumps Beauty
10. Get the Right Tool

IGTON

Matplotlib : Demo

https://www.tutorialspoint.com/amazon_web_services/amazon_web_services_s3.htm

Choosing the right chart

Relationship	Scatter plot, Marginal Histogram, Scatter plot using Seaborn, Pair Plot in Seaborn, Heat Map
Data over Time	Line Chart, Area Chart, Stack Area Chart, Area Chart Unstacked
Ranking	Vertical Bar Chart, Horizontal Bar Chart, Multi-set Bar Chart, Stack Bar Chart, Lollipop Chart
Distribution	Histogram, Density Curve with Histogram, Density Plot, Box Plot, Strip Plot, Violin Plot, Population Pyramid
Comparison	Bubble Chart, Bullet Chart, Pie Chart, Net Pie Chart, Donut Chart, TreeMap, Diverging Bar, Choropleth Map, Bubble Map

MLlib : Demo

https://www.tutorialspoint.com/amazon_web_services/amazon_web_services_s3.htm

MACHINE LEARNING LIBRARY

CHEAT SHEET

MLlib Basics

MLlib

It is an Apache Spark machine learning library which is scalable; it consists of popular algorithms and utilities

MLlib contains two packages

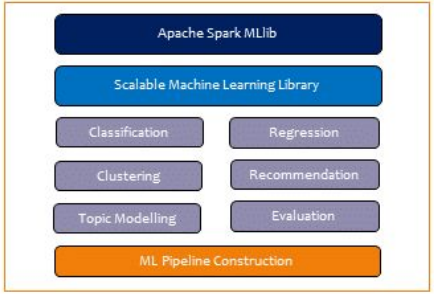
- Spark.mllib
- Spark.ml

To add the **MLlib** the following library is imported:

- In Scala: import org.apache.spark.mllib.linalg.{Vector, Vectors}
- In Java: import org.apache.spark.mllib.linalg.Vector;
  import org.apache.spark.mllib.linalg.Vectors;
- In python: from pyspark.mllib.linalg import SparseVector
  from pyspark.mllib.regression import LabeledPoint

Data Source

Access to HDFS and HBase can be done using **MLlib**, which enables **MLlib** to be plugged in Hadoop Work process



Main Concepts In Pipeline

MLlib is used to standardize the APIs for easy use of multiple algorithms being used as a single pipeline or a workflow

- Data frame: The **ML API** uses Dataframe from **Spark SQL** as a dataset, which can be used to hold a variety of datatypes
- Transformer: This is used to transform one Dataframe to another Dataframe. Examples are
  - Hashing Term Frequency: This calculates how word occurs
  - Logistic Regression Model: The model which results from trying logistic regressions on a dataset
  - Binarizer: This changes a given threshold value to 1 or 0
- Estimator: It is an algorithm which can be used on a Dataframe to produce Transformer. Examples are:
  - Logistic Regression: It is used to determine the weights for the resulting Logistic Regression Model by processing the dataframe
  - StandardScaler: It is used to calculate the Standard deviation
  - Pipeline: Calling fit on a pipeline produces pipeline model, and the pipeline contains only transformers and not the estimators
- Pipeline: A pipeline chains multiple Transformers and Estimators together to specify the ML workflow
- Parameters: To specify the parameters a common API is used by the Transformers and Estimators

Observations

The items or data points used for learning and evaluating

Features

The characteristic or attribute of an observation

Labels

The values assigned to an observation is called a **Label**

Training or test data

A learning algorithm is an observation used for training and testing of the data

Spark MLlib Tools

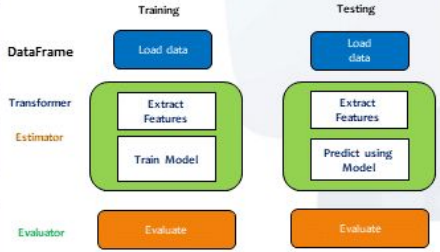
ML Algorithms: These include common learning algorithms such as classification, clustering, regression and collaborative filtering. These algorithms form the core of MLlib

Featurization: It includes feature extraction, transformation, dimensionality reduction and selection

Pipelines: Pipelines provide tools for constructing, evaluating and tuning ML pipelines

Persistence: It helps in saving and loading algorithms, models and pipelines

Utilities: It provides utilities for linear algebra, statistics and data handling



MLlib Algorithms

These include the popular algorithms and utilities

- Basic statistics: It includes the most basic of the machine learning techniques such as:
  - Summary statistics
  - Correlation
  - Stratified sampling
  - Hypothesis testing

Regression: It is a statistical approach to estimate the relationship among variables. It is widely used for prediction and forecasting

Classification: It is used to identify to which set of categories a new observation belongs to.

- K-means classification: It is used for classification using MLlib in Java. It is used to classify every observation, experiment or a vector into one of the cluster

Recommendation system: it is a sub class of information filtering system that seeks to predict the preference or rating a person can give to an item. This can be done in two ways

- Collaborative filtering: It approaches in building a model from a user's past behavior as well as similar decisions made by the user. The model is then used to predict the items in which the user might have interest
- Content-based filtering: It approaches to utilize a series of discrete characteristics of an item in order to recommend more items with similar properties

Clustering: It is a task to group set of objects in a way that the objects in the same group is more similar to each other when compared to the objects in the other group.

Dimensionality Reduction: It is a process of reducing a set of random variables under consideration by obtaining a set of principal variables. It can be divided into two types

- Feature selection: It finds a subset of original variables called attributes
- Feature Extraction: This will transform the data from in a high dimensional space to a space of fewer dimensions.

Feature extraction: It starts from initial set of derived data and builds derived values.

Optimization: It is a selection of best element from the set of available alternatives

REFERENCES

1. https://matplotlib.org/stable/gallery/index.html
2. https://intellipaat.com/blog/tutorial/spark-tutorial/spark-and-rdd-cheat-sheet/
3. https://towardsdatascience.com/data-visualization-how-to-choose-the-right-chart-part-1-d4c550085ea7
4. https://intellipaat.com/blog/tutorial/machine-learning-tutorial/mllib-cheat-sheet/

Questions?