Introduction to Apache Spark

Apurva Nandan

- Data Intensive Computing

CSC

CSC-IT CENTER FOR SCIENCE

About Us

- Data Intensive Computing group managed by Aleksi Kallio
- Work based on building services and applications around cloud computing and big data environment
- Cloud computing platform is known as cPouta. Openstack used for managing
- Latest Featured Product – Pouta Blueprints (pb.csc.fi)
- Increasing demand for Spark based environments on cPouta

Schedule of the training

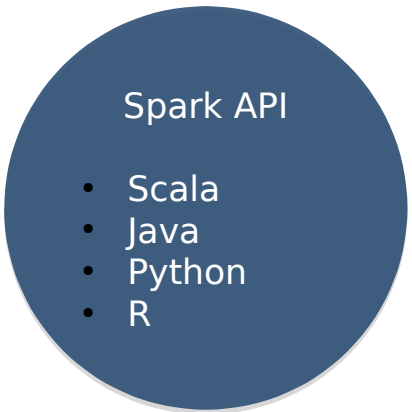
- Day 1: Spark Basics and Core
- Day 2: Spark MLlib

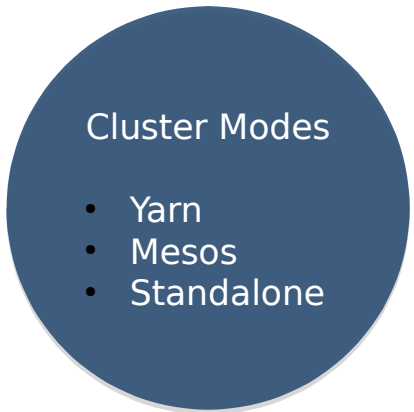
Apache Spark

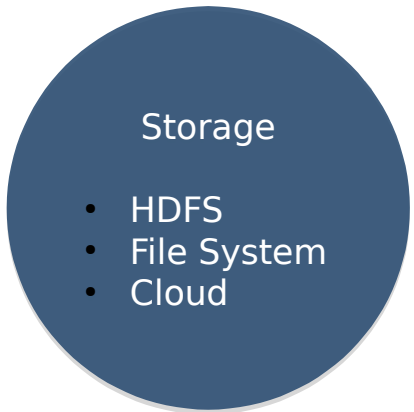
- Fast, Open Source, big data based engine for large scale data analysis and distributed processing
- Developed in Scala and runs on Java Virtual Machine
- Can be Used with Scala, Java, Python or R
- Works on the Map-Reduce concept to do the distributed processing

Apache Spark – Key Concepts

- Stores the data into memory when needed
- Follows distributed processing concepts for distributing the data across the nodes of the cluster







Spark Vs Hadoop

- Faster than Hadoop in many cases specially iterative algorithms
- Spark Stores the data in memory which allows faster chaining of operations vs Hadoop Map Reduce which stores the output to disk for any map or reduce operation
- Much more simpler programming API as compared to Hadoop – Less time writing complex map reduce scripts
- When it comes to fault tolerance – Both are equally good

Map Reduce Paradigm

- Programming model developed by Google
- As the name suggests, there are two phases for processing the data – Map and Reduce
  - Map: Transform each element in the data set
  - Reduce: Combine the data according to some criteria
- Spark uses the Map Reduce programming model for distributed processing

RDD

- RDD a.k.a Resilient Distributed Datasets
- Spark's fault tolerant dataset which can be operated in a distributed manner by running the processing across all the nodes of a cluster
- Can be used to store any type of element in it
- Some Special types of RDDs:
  - Double RDD: for storing numerical data
  - Pair RDD: For storing key pair values

Pair RDDs

- Special form of RDDs which are used for map reduce based functions
- They are in a form of key/value pairs <K,V>. Key and Value can be of any type
- They can be used for running map, reduce, filter, sort, join or other functions present in the spark API
- Generally, Pair RDDs are used more often

Creating RDDs

- By distributing the native python collection:
  >>> sc.parallelize(['this', 'is', 'an', 'example'])
  >>> sc.parallelize(xrange(1,100))

- By reading text files from local file system , HDFS or object storage:
  >>> sc.textFile('file:///example.txt')
  >>> sc.textFile('hdfs:///hdfsexample.txt')

  This output RDD is an RDD of strings

RDD Transformations

- map(func) Return a new distributed dataset formed by passing each element of the source through a function func.
- filter(func) Return a new dataset formed by selecting those elements of the source on which func returns true.
- flatMap(func) Similar to map, but flattens the final result.
- groupByKey([numTasks]) When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. Note: If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using reduceByKey or aggregateByKey will yield much better performance.
- reduceByKey(func, [numTasks]) When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function func, which must be of type (V,V) => V.
- sortByKey([ascending], [numTasks]) : When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument.
- join(otherDataset, [numTasks]) : When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key. Outer joins are also supported
- cogroup(otherDataset, [numTasks]) : When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (Iterable<V>, Iterable<W>)) tuples. This operation is also called groupWith.

RDD Actions

- collect() : Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
- count() :Return the number of elements in the dataset.
- first() :Return the first element of the dataset (similar to take(1)).
- take(n) :Return an array with the first n elements of the dataset.
- takeSample(withReplacement, num, [seed]) : Return an array with a random sample of num elements of the dataset, with or without replacement, optionally pre-specifying a random number generator seed.
- takeOrdered(n, [ordering]) Return the first n elements of the RDD using either their natural order or a custom comparator.
- saveAsTextFile(path) Write the elements of the dataset as a text file (or set of text files) in the local filesystem, HDFS
- countByKey() Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.
- foreach(func) Run a function func on each element of the dataset. This is usually done for side effects

Spark Shell

- Available for Python and Scala
- Working with small data set in local mode can quickly show the results in the shell. Quick way to write and debug the code
- For final run: Write the code in a file and submit it to the cluster.
- Run the python shell by: sh <spark-dir>/bin/pyspark

Transformations: map

- Used to apply a function to each element in the RDD
- This is used to modify elements of the RDD. One could completely change the format of the RDD

```
>>> rdd = sc.parallelize(range(1,10)) # [1,2,.....10]
>>> rdd_map = rdd.map(lambda x: (x, x))
>>> rdd_map.collect()  # [(1,1), (2,2) ..... (10,10)]
>>> rdd_map = rdd_map.map(lambda x: (x[0]*2, x[1])).collect()
```

- Special case: flatMap:

```
>>> rdd_flatmap = rdd_map.flatMap(lambda x: (x[0], x[1] + 1))
```

Transformations: filter

- Used to filter out the RDD according to a defined criteria

```
>>> rdd = sc.textFile('HP.txt')
```

The place where things are hidden
if you have to ask you will never know
If you know you need only to ask

```
>>> rdd = rdd.filter(lambda x: x.startswith('The'))
```

The place where things are hidden

A Simple User Exercise

Try to run the following piece of code now:

```
>>> rdd = sc.textFile("HP.txt")
```

- Convert all the lines to Uppercase
- Remove the lines which contain the word ''HIDDEN” in it
- Arrange all the words in a single list using a one-liner and count the total number of words
- Print the number of words

RDD: Tabular to Key-Pairs

- Always remember, if your RDD turns out to be something like this:
  col1, col2, col3
  1,[1,2],'hello'
  2,[3,4],'moi'
  3,[5,6],'hola'
  4,[7,8],'ciao'

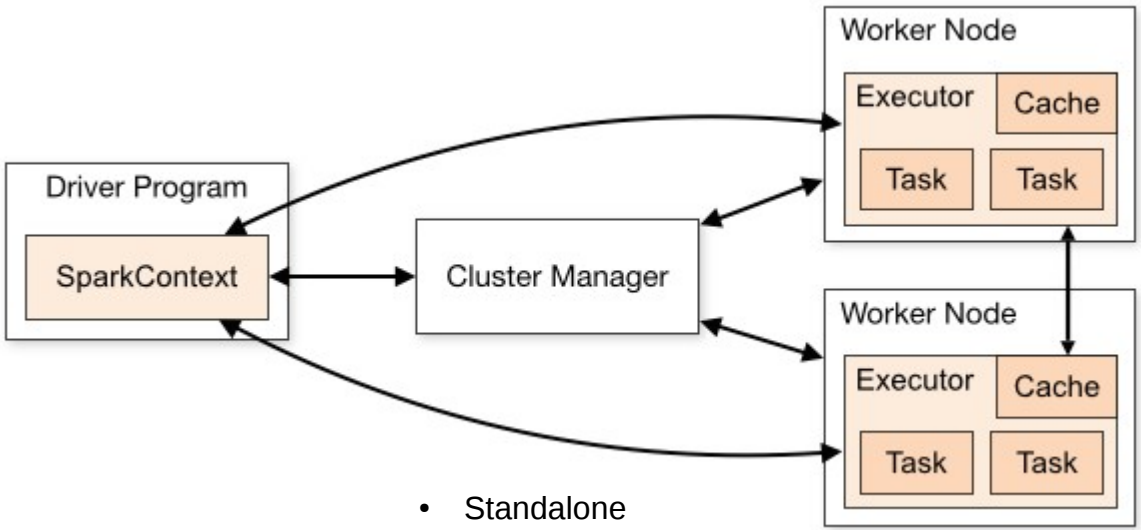
  Convert it into a key/pair format like:
  1, ([1,2], 'hello')
  .....

- This means, now you have one key and the rest of the things are values. This helps Spark operate the RDD as a PairRDD. (Thus reduceByKey, countByKey also get their 'Key' ;-))

Execution of Spark Code

- Every cluster will have one master and several nodes which are called workers in spark terminology
- The master usually runs the Driver. The driver sends the application code to the workers and is responsible for scheduling tasks on the cluster (workers)
- The workers run the spark code on one or more executors (depending upon the number of cores of the worker)
- Parallel processing is done on the workers and workers talk to each other.
- There are different 'stages' of a particular transformation and they contain the tasks to be carried

Execution of Spark Code



- Standalone
- Apache Mesos
- Hadoop YARN

Note: Image taken from Spark's official documentation

Parallelism in Spark

- So far we have known that Spark does the processing in a distributed manner. Have you pondered is there something to control this parallelism?
- Spark allows the user to control parallelism by providing partitions when creating an RDD or changing the configuration
- sc.textFile(inputfile, numPartitions)
- spark.default.parallelism: For distributed shuffle operations like reduceByKey and join, the largest number of partitions in a parent RDD. For operations like parallelize with no parent RDDs, it depends on the cluster manager:
  - Local mode: number of cores on the local machine
  - Mesos fine grained mode: 8
  - Others (YARN): total number of cores on all executor nodes or 2, whichever is larger

Shuffle Behavior

- Happens for some Spark transformations like groupByKey and reduceByKey.
- Data required in computations of records can be in different partitions of the parent RDD.
- The tasks require that all the tuples with same key have to be in the same partition and should be processed by the same task.
- Spark executes a shuffle, which involves transfer of data across the workers of the cluster and results in a new stage with completely new set of partitions.

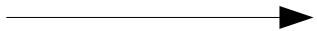
Shuffle Behavior

- Expensive operations as they incur heavy disk and network I/O
- The primary goal when choosing an arrangement of operators is to reduce the number of shuffles and the amount of data shuffled. This is because shuffles are fairly expensive operations;
- All the shuffle data is written to disk and transferred over the network.
- join, cogroup, any of the *By or *ByKey transformations result in shuffles. Each of these operations have different costs and as a novice one has to be careful when selecting one over another

Word Count Example

If you have to ask you will never know

If you know you need only to ask



you 4
if 2
ask 2
to 2
know 2
have 1
will 1
never 1
need 1
only 1

Word Count Example

```
# Load the text file
>>> rdd = sc.textFile('HP.txt')
# Split the text into words and flatten the results. Why?
>>> words = rdd.flatMap(lambda line: line.split())
>>> words.collect()
```

('if' , 'you', 'have', 'to', 'ask', 'you' 'will', 'never', 'know', 'if', 'you', 'know' , 'you' , 'need' , 'only', 'to', 'ask')

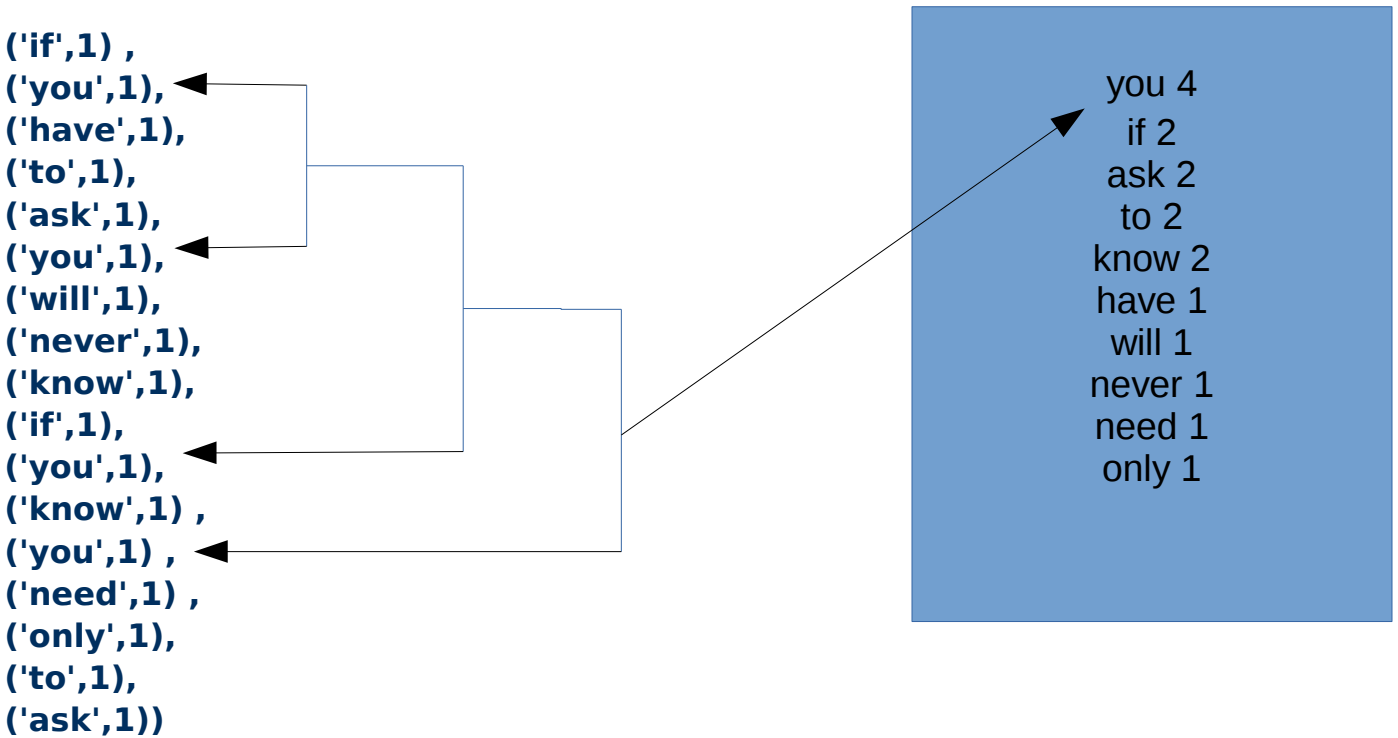
Word Count Example

```
# Load the text file
>>> rdd = sc.textFile('HP.txt')
# Split the text into words and flatten the results. Why?
>>> words = rdd.flatMap(lambda line: line.split())
>>> words.collect()
('if' , 'you', 'have', 'to', 'ask', 'you' 'will', 'never', 'know', 'if', 'you', 'know' , 'you' , 'need' ,
'only', 'to', 'ask')

>>> words_map = words.map(lambda x: (x,1))
(('if',1) , ('you',1), ('have',1), ('to',1), ('ask',1), ('you',1), ('will',1), ('never',1), ('know',1),
('if',1), ('you',1), ('know',1) , ('you',1) , ('need',1) , ('only',1), ('to',1), ('ask',1))
```

Word Count Example

```
# Load the text file
>>> rdd = sc.textFile('HP.txt')
# Split the text into words and flatten the results. Why?
>>> words = rdd.flatMap(lambda line: line.split())
>>> words.collect()
('if' , 'you', 'have', 'to', 'ask', 'you' 'will', 'never', 'know', 'if', 'you', 'know' , 'you' , 'need' ,
'only', 'to', 'ask')
>>> words_map.map(lambda x: (x,1))
(('if',1) , ('you',1), ('have',1), ('to',1), ('ask',1), ('you',1), ('will',1), ('never',1), ('know',1),
('if',1), ('you',1), ('know',1) , ('you',1) , ('need',1) , ('only',1), ('to',1), ('ask',1))

>>> words_count = words_map.reduceByKey(lambda a,b: a+b)
```

Word Count Example



RDD Exercise – Inverted Index

In this exercise, you have to generate an inverted index from the given text file. An inverted index contains list of all the unique words that appear in any document, and then each word is mapped to a list of documents in which it appears.

Using the lessons learnt from word count example, for reading, map filter and reduce functions. Write a spark application for producing an inverted index

The next slide shows what an inverted index looks like:

RDD Exercise – Inverted Index

```
[('a', [[0, 1], [2, 1]]),
 ('always', [[0, 1]]),
 ('and', [[1, 1]]),
 ('be', [[2, 1]]),
 ('dark', [[1, 1]]),
 ('debts', [[0, 1]]),
 ('die', [[3, 1]]),
 ('full', [[1, 1]]),
 ('game', [[3, 1]]),
 ('hand', [[2, 1]]),
 ('hands', [[2, 1]]),
 ('have', [[2, 1]]),
 ('his', [[0, 1]]),
 ('is', [[1, 1]]),
 ('jon', [[4, 1]]),
 ('know', [[4, 1]]),
 ('lannister', [[0, 1]]),
 ('last', [[2, 1]]),
 ('me', [[2, 1]]),
 ('next', [[2, 1]]),
 ('night', [[1, 1]]),
 ('nothing', [[4, 1]]),
 ('of', [[3, 1], [1, 1]]),
 ('or', [[3, 1]]),
 ('pays', [[0, 1]]),
 ('play', [[3, 1]]),
 ('raise', [[2, 1]]),
 ('snow', [[4, 1]]),
 ('terrors', [[1, 1]]),
 ('the', [[3, 1], [1, 1], [2, 2]]),
 ('thrones', [[3, 1]]),
 ('time', [[2, 2]]),
 ('to', [[2, 1]]),
 ('when', [[3, 1]]),
 ('will', [[2, 1]]),
 ('win', [[3, 1]]),
 ('you', [[3, 3], [2, 2], [4, 1]])]
```

RDD Exercise – Inverted Index

In this exercise, you have to generate an inverted index from the given text file.
An inverted index contains list of all the unique words that appear in any document, and then each word is mapped to a list of documents in which it appears.

Using the lessons learnt from word count example, for reading, map filter and reduce functions. Write a spark application for producing an inverted index

The next slide shows what an inverted index looks like:

HDFS

- Hadoop Distributed File System - Java-based file system for scalable and reliable data storage
- Designed for spanning large clusters with nominal hardware
- Scalability of up to 200 PB of storage and a single cluster of 4500 servers, supporting close to a billion files and blocks in production
- Fault-tolerant, distributed storage system working with various concurrent data access applications, coordinated by YARN.
- Default storage type with Spark.

Writing code in pySpark shell

```
>>> sc #  SparkContext is the main entry point for Spark API (Created by default in pyspark)

>>> rdd = sc.parallelize(range(1,10**5))  # create a parallel collection (rdd) with the specified range

>>> rdd.count()  # count the number of elements

>>> rdd.collect() # print the elements currently in the collection
```

Writing code in files

```
From pyspark import SparkContext, SparkConf # SparkContext Needs to be called first
conf = new SparkConf()  # Used to provide optional configurations to the spark application
conf.setAppName('Test app')
conf.set('executor.cores', )
sc.setConf(conf=conf)

rdd = sc.parallelize(xrange(1, 10**5))

print(rdd.count())

print(rdd.collect())
```

- # Run your code with /usr/hdp/current/spark-client/bin/spark-submit filename.py

Things to avoid

- Use reduceByKey to perform aggregation or counting instead of groupByKey
- Avoid printing huge data sets using .collect()
- Normal for loops:
  - Always try to accomplish the things using Spark's map, flatMap functions when you need to apply something over the elements.
  - Normal loops are used in SPECIAL cases when the lambda function you are providing in a map requires a loop with constant time complexity

DAY 2

Machine Learning Basics

- "Field of study that gives computers the ability to learn without being explicitly programmed" - Arthur Samuel
- Explores the study and construction of algorithms that can learn from data and make predictions
- Machine learning tasks are typically classified into three broad categories, depending on the nature of the learning "signal" or "feedback" available to a learning system.
- Supervised learning: The computer is presented with example inputs and their desired outputs 'labels', given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs.
- Unsupervised learning: No output labels are given to the learning algorithm, leaving it on its own to find structure in its input.
- Reinforcement learning: A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle), without a teacher explicitly telling it whether it has come close to its goal.

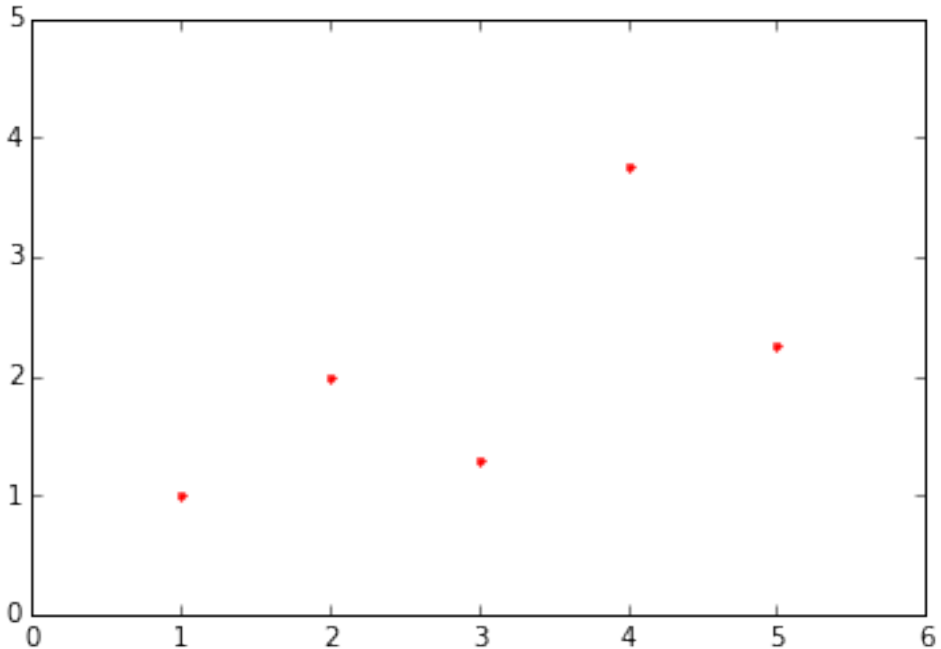
Machine Learning Basics

- Unsupervised Learning: The goal in such problems may be to discover groups of similar examples within the data, where it is called clustering
- Supervised Learning:
  - Classification: samples belong to two or more classes and we want to learn from already labeled data how to predict the class of unlabeled data.
  - Regression: if the desired output consists of one or more continuous variables, then the task is called regression. An example of a regression problem would be the prediction of the length of a salmon as a function of its age and weight.

Machine Learning Basics

- A Simple Linear Regression

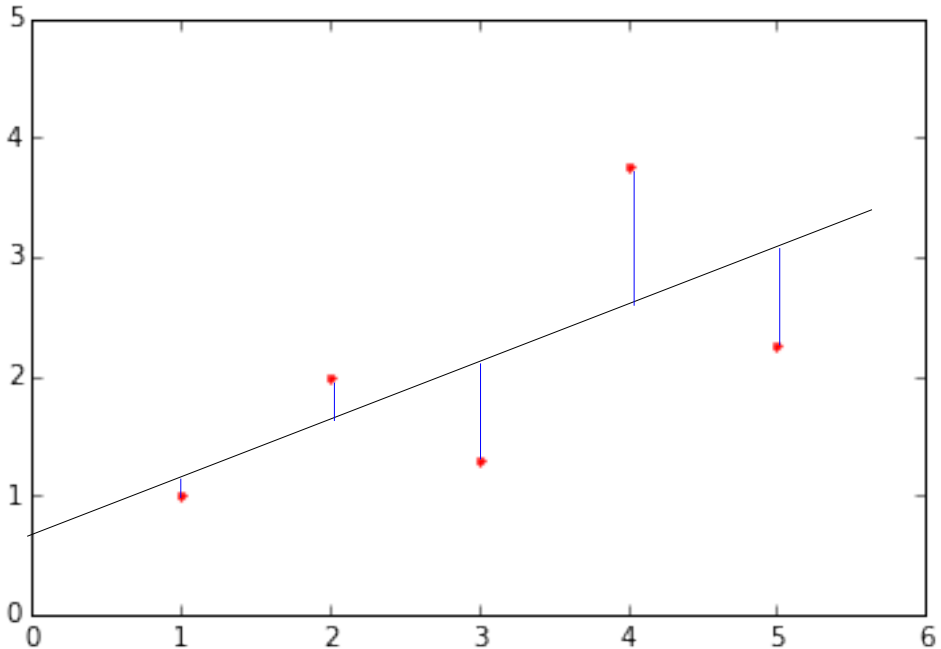
X	Y
1.00	1.00
2.00	2.00
3.00	1.30
4.00	3.75
5.00	2.25



Machine Learning Basics

- A Simple Linear Regression

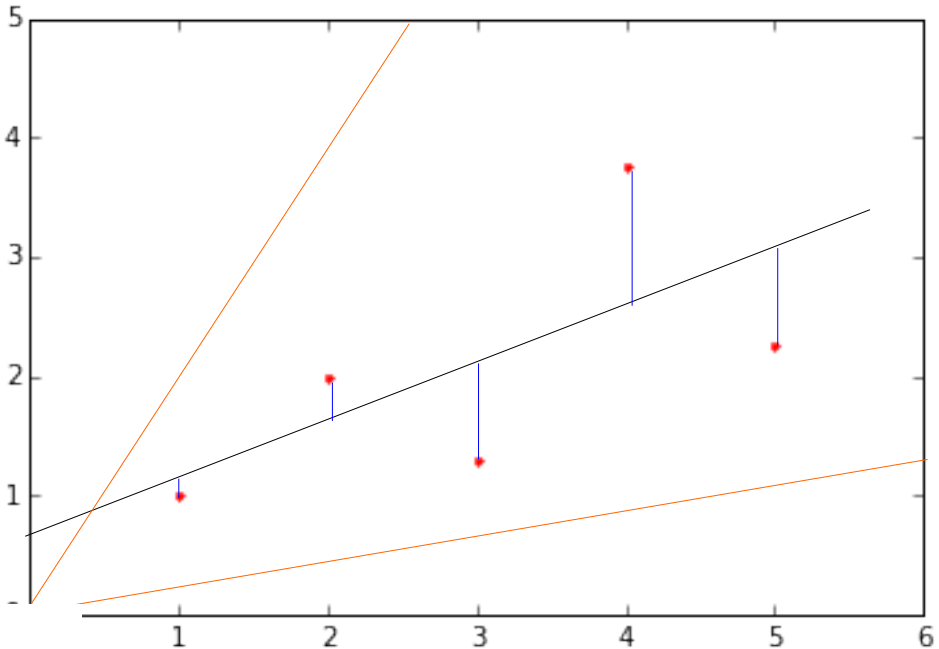
X	Y
1.00	1.00
2.00	2.00
3.00	1.30
4.00	3.75
5.00	2.25



Machine Learning Basics

- A Simple Linear Regression

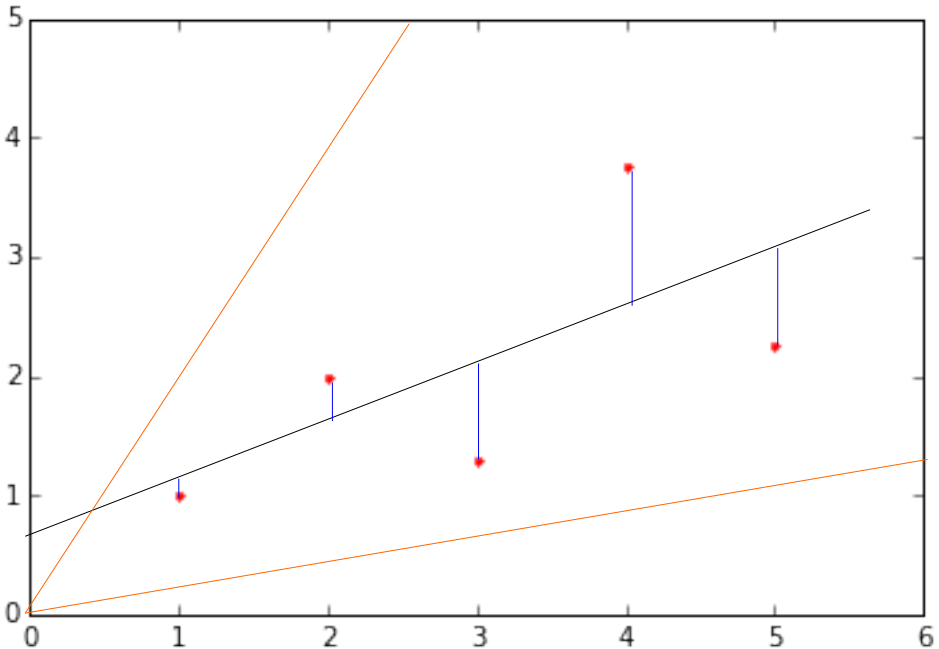
X	Y
1.00	1.00
2.00	2.00
3.00	1.30
4.00	3.75
5.00	2.25



$$\sigma_{est} = \sqrt{\frac{\sum (Y - Y')^2}{N}}$$

Machine Learning Basics

- A Simple Linear Regression



Standard Error of the Estimate

- Measure of the accuracy of predictions.
- Regression line is the line that minimizes the sum of squared distances (also called the sum of squares error).
- Lower standard error of the estimate means more accurate predictions

- Denoted by the following :

$$\sigma_{est} = \sqrt{\frac{\sum (Y - Y')^2}{N}}$$

Machine Learning Basics

- Classification: Problem of identifying to which set of categories , a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known.
- Example :
  - assigning a given email into "spam" or "non-spam" classes
  - assigning a diagnosis to a given patient as described by observed characteristics of the patient (gender, blood pressure, presence or absence of certain symptoms, etc.).

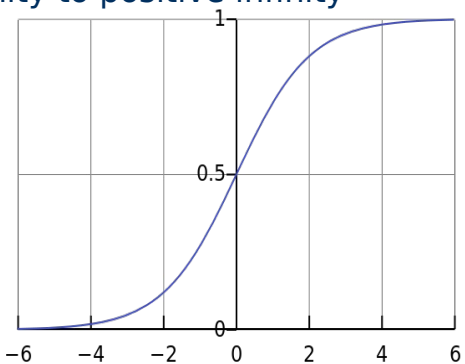
Machine Learning Basics

Brief Overview of Logistic Regression:

- Used for Classification of data into output labels. Usually there could be multiple labels for output, but right now in the basic example we will consider just two output labels.
- The aim of L.R. is to draw a “decision boundary” through the training data and label all the data points on one side as 0 and the ones on the other sides as 1.
- Makes use of a logistic function which can take any input from negative infinity to positive infinity but returns the output between 0 and 1 , hence interpretable as probability

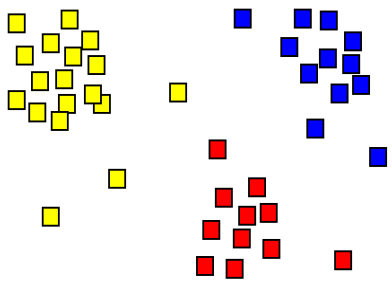
$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

$$t = \beta_0 + \beta_1 x$$



Machine Learning Basics

- Clustering: The task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters).
- Quite commonly used in information retrieval, exploratory data analysis, image analysis , pattern recognition
- For evaluating how good the clustering is: Within Set Sum of Squared Error (WSSSE): Squared distance between all the points to their closest cluster center.
  - You can reduce this error measure by trying different parameters



Machine Learning Basics

- Train: Learn from the given data
- Predict : Predict output for an unseen data

  Types of data sets:

  - Training Set: The data set used in training the algorithm
  - Validation Set (Optional): Used to verify the best model configuration
  - Test set: The data set on which we need to predict the output

FEATURES AND OUTPUT LABEL

The data on the right side shows “features” and output label in a dataset, which shows selection of students in a particular university.

Gre gpa and rank are the features which determine if a student will get into the university or not. Admit is the output label which denotes 1 for True (Accepted) and 0 for False

gre	gpa	rank	**admit**
380	3.61	3	0
660	3.67	3	1
800	4.00	1	1
640	3.19	4	1
520	2.93	4	0
760	3.00	2	1

ML in Spark: Spark MLlib library

- Apache Spark's scalable machine learning library.
- Used with Spark's APIs and compatible with NumPy in Python
- 100x faster than Hadoop
- Don't reinvent the wheel: If you are looking to do use any of the machine learning based techniques in your work , it is always a best idea to see if Spark has the algorithm in MLlib library
- The algorithms provided are much more optimized and ready to use

ML in Spark: Spark MLlib library

- Following are some of the algorithms present in the MLlib library
  - Classification: Logistic Regressions, Decision Trees, Naive Bayes, Random Forests and Support Vector Machines
  - Regression: Linear Regression , random forests
  - Collaborative Filtering: Alternating Least Squares
  - Clustering: k-means
  - Dimensionality reduction: Singular Value Decomposition, Principal Component Analysis
  - Feature Extraction and Transformation: TF-IDF, Word2Vec, Normalizer

ML Resources for further reading

ML Notes by Andrew Ng http://www.holehouse.org/mlclass/

Famous Coursera course by Andrew Ng

Machine Learning, Tom Mitchell, McGraw Hill, 1997. (Also has ML Notes from a course on his website)

- Machine Learning With Big Data course on coursera for more practical algorithms used in analytics

Useful paths

/usr/hdp/current/spark-client/bin/pyspark – Pyspark Shell

/usr/hdp/current/spark-client/bin/spark-submit <filename.py> – run your file