PROJECT REPORT



Machine Learning Using Hadoop

Computer Science and Engineering Department

IIT(ISM) Dhanbad

(Session 2017-18)

C.R.V.Phanindra Gupta

15JE001398

Table Of Contents

1. Abstract
2. Acknowledgement
3. Introduction to Big Data
   * Types of Big Data
   * Big Data Technologies
   * Big Data Challenges
4. Introduction to Hadoop Architecture
   * Hadoop Layers
   * MapReduce Software framework
   * HDFS storage
   * Hadoop Working
     1. Implementation of WordCount
     2. WordCount using iterators
5. Introduction to Machine Learning – Classification
   * KNN Algorithm
     1. Introduction
     2. Implementation using Hadoop
     3. Applications
   * SVM Classifier
     1. Introduction
     2. Implementation in Python
     3. Applications
6. Conclusion

Abstract

Machine learning (ML) with big data is a notoriously complex technology to master but opening up ML to the masses will lead to the creation of more models and applications with better accuracy and for better future.

This project deals with the implementations of machine learning algorithms using hadoop stream-handling in python, which is again a better language with better memory management.

Acknowledgement

The final outcome of this project required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my project, All that I have done is only due to such supervision and assistance and I would not forget to thank them.

I respect and thank **Dr. Dharavath Ramesh** for providing me an oppurtunity to do the project work and providing all support and guidance which made me complete the project duly. I am extremely thankful to him for providing such a nice support although he had busy schedule.

I am thankful to and fortunate enough to get constant encouragement, support and guidance from all Teaching Staffs of Computer Science and Engineering Department, IIT(ISM) Dhanbad which helped in completing my project.

C.R.V. Phanindra Gupta

15JE001398

Introduction to Big Data

Big data means really a big data, it is a collection of large datasets that cannot be processed using traditional computing techniques. Big data is not merely a data, rather it has become a complete subject, which involves various tools, technqiues and frameworks.

## What Comes Under Big Data?

Big data involves the data produced by different devices and applications. Given below are some of the fields that come under the umbrella of Big Data.

* **Black Box Data** : It is a component of helicopter, airplanes, and jets, etc. It captures voices of the flight crew, recordings of microphones and earphones, and the performance information of the aircraft.
* **Social Media Data** : Social media such as Facebook and Twitter hold information and the views posted by millions of people across the globe.
* **Stock Exchange Data** : The stock exchange data holds information about the ‘buy’ and ‘sell’ decisions made on a share of different companies made by the customers.
* **Power Grid Data** : The power grid data holds information consumed by a particular node with respect to a base station.
* **Transport Data** : Transport data includes model, capacity, distance and availability of a vehicle.
* **Search Engine Data** : Search engines retrieve lots of data from different databases.

Thus Big Data includes huge volume, high velocity, and extensible variety of data.

**Types of Big Data**

1. **Structured data** : Relational data.
2. **Semi Structured data** : XML data.
3. **Unstructured data** : Word, PDF, Text, Media Logs.

**Big Data Technologies**

Big data technologies are important in providing more accurate analysis, which may lead to more concrete decision-making resulting in greater operational efficiencies, cost reductions, and reduced risks for the business.

To harness the power of big data, you would require an infrastructure that can manage and process huge volumes of structured and unstructured data in realtime and can protect data privacy and security.

General technologies that handle big data are classified as:

### 1. Operational Big Data

This include systems like MongoDB that provide operational capabilities for real-time, interactive workloads where data is primarily captured and stored.

NoSQL Big Data systems are designed to take advantage of new cloud computing architectures that have emerged over the past decade to allow massive computations to be run inexpensively and efficiently. This makes operational big data workloads much easier to manage, cheaper, and faster to implement.

Some NoSQL systems can provide insights into patterns and trends based on real-time data with minimal coding and without the need for data scientists and additional infrastructure.

### 2. Analytical Big Data

This includes systems like Massively Parallel Processing (MPP) database systems and MapReduce that provide analytical capabilities for retrospective and complex analysis that may touch most or all of the data.

MapReduce provides a new method of analyzing data that is complementary to the capabilities provided by SQL, and a system based on MapReduce that can be scaled up from single servers to thousands of high and low end machines.

## Operational vs. Analytical Systems

|  |  |  |
| --- | --- | --- |
|  | Operational | Analytical |
| Latency | 1 ms - 100 ms | 1 min - 100 min |
| Concurrency | 1000 - 100,000 | 1 - 10 |
| Access Pattern | Writes and Reads | Reads |
| Queries | Selective | Unselective |
| Data Scope | Operational | Retrospective |
| End User | Customer | Data Scientist |
| Technology | NoSQL | MapReduce, MPP Database |

## Big Data Challenges

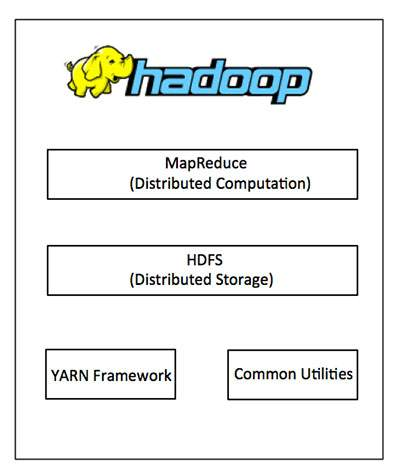
The major challenges associated with big data are as follows:

* Capturing data
* Curation
* Storage
* Searching
* Sharing
* Transfer
* Analysis
* Presentation

**Introduction to Hadoop Architecture**

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. A Hadoop frame-worked application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

**Hadoop Layers**



1. **Hadoop Common:** These are Java libraries and utilities required by other Hadoop modules. These libraries provides filesystem and OS level abstractions and contains the necessary Java files and scripts required to start Hadoop.
2. **Hadoop YARN:** This is a framework for job scheduling and cluster resource management.
3. **Hadoop Distributed File System (HDFS™):** A distributed file system that provides high-throughput access to application data.
4. **Hadoop MapReduce:** This is YARN-based system for parallel processing of large data sets.

**MapReduce Software Framework**

Hadoop **MapReduce** is a software framework for easily writing applications which process big amounts of data in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

The term MapReduce actually refers to the following two different tasks that Hadoop programs perform:

* **The Map Task:** This is the first task, which takes input data and converts it into a set of data, where individual elements are broken down into tuples (key/value pairs).
* **The Reduce Task:** This task takes the output from a map task as input and combines those data tuples into a smaller set of tuples. The reduce task is always performed after the map task.

Typically both the input and the output are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

The MapReduce framework consists of a single master **JobTracker** and one slave **TaskTracker** per cluster-node. The master is responsible for resource management, tracking resource consumption/availability and scheduling the jobs component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves TaskTracker execute the tasks as directed by the master and provide task-status information to the master periodically.

The JobTracker is a single point of failure for the Hadoop MapReduce service which means if JobTracker goes down, all running jobs are halted.

## Hadoop Distributed File System

Hadoop can work directly with any mountable distributed file system such as Local FS, HFTP FS, S3 FS, and others, but the most common file system used by Hadoop is the Hadoop Distributed File System (HDFS).

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on large clusters (thousands of computers) of small computer machines in a reliable, fault-tolerant manner.

HDFS uses a master/slave architecture where master consists of a single **NameNode** that manages the file system metadata and one or more slave **DataNodes** that store the actual data.

A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of DataNodes. The NameNode determines the mapping of blocks to the DataNodes. The DataNodes takes care of read and write operation with the file system. They also take care of block creation, deletion and replication based on instruction given by NameNode.

HDFS provides a shell like any other file system and a list of commands are available to interact with the file system. These shell commands will be covered in a separate chapter along with appropriate examples.

## How Does Hadoop Work?

### Stage 1

A user/application can submit a job to the Hadoop (a hadoop job client) for required process by specifying the following items:

1. The location of the input and output files in the distributed file system.
2. The java classes in the form of jar file containing the implementation of map and reduce functions.
3. The job configuration by setting different parameters specific to the job.

### Stage 2

The Hadoop job client then submits the job (jar/executable etc) and configuration to the JobTracker which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

### Stage 3

The TaskTrackers on different nodes execute the task as per MapReduce implementation and output of the reduce function is stored into the output files on the file system.

**Implementation of WordCount**

**mapper – code:**

import sys

for line in sys.stdin:

line = line.strip()

words = line.split()

for word in words:

print '%s\t%s' % (word, 1)

**reducer – code:**

import sys

current\_word = None

current\_count = 0

word = None

for line in sys.stdin:

line = line.strip()

word, count = line.split('\t', 1)

try:

count = int(count)

except ValueError:

continue

if current\_word == word:

current\_count += count

else:

if current\_word:

print '%s\t%s' % (current\_word, current\_count)

current\_word = word

current\_count = count

if current\_word == word:

print '%s\t%s' % (current\_word, current\_count)

**Word Count using Iterators**

**mapper – code:**

import sys

def read\_input(file):

for line in file:

yield line.split()

def main():

data = read\_input(sys.stdin)

for words in data:

for word in words:

print '%s\t%s' % (word, 1)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**reducer – code:**

from itertools import groupby

from operator import itemgetter

import sys

def read\_mapper\_input(file):

for line in file:

yield line.rstrip().split('\t', 1)

def main():

data = read\_mapper\_input(sys.stdin)

for current\_word, group in groupby(data, itemgetter(0)):

try:

total\_count = sum(int(count) for current\_word, count in group)

print '%s\t%d' % (current\_word, total\_count)

except:

pass

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Intoduction to Machine Learning – Classification**

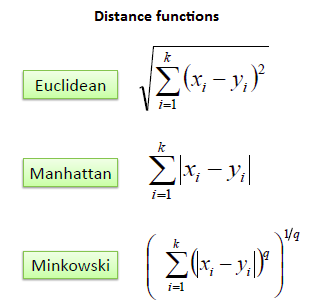
In machine learning and statistics, **classification** is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. Classification is an example of pattern recognition.

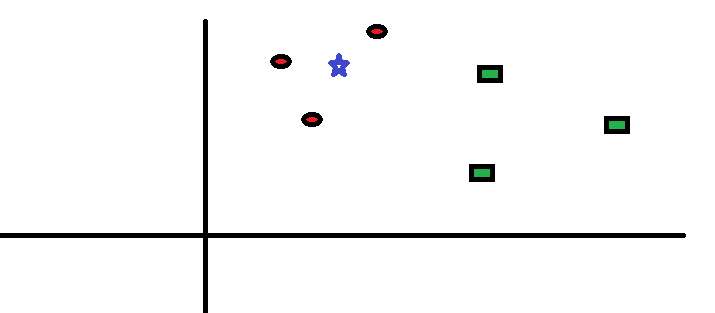
Examples:

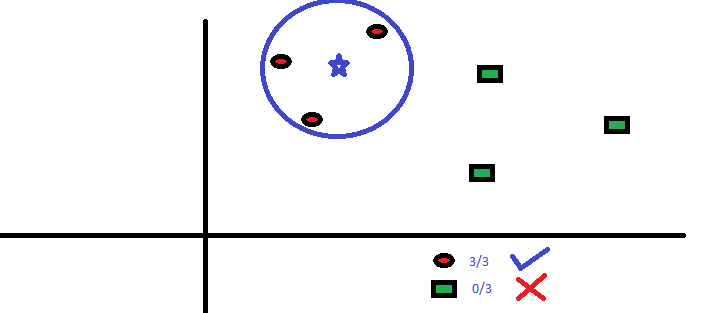
* classifying if an email is “spam” or “not-spam”
* assigning a diagnosis to a given patient as described by observed characteristics of the patient.

**k-Nearest Neighbors Algorithm**

* In pattern recognition, the ***k*-nearest neighbors algorithm** (***k*-NN**) is a non-parametric method (parameters are determined by the training data, not the model) used for classification.
* In *k-NN classification*, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k* is a positive integer, typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.



****

****

**Implementation using Hadoop**

**mapper – code:**

import sys

import math

query = [sys.argv[1], sys.argv[2]]

for line in sys.stdin:

avg = 0.0

line = line.strip()

features = line.split()

for i in range(len(features)-1):

if features[i]!='':

avg += (float(features[i]) - float(query[i]))\*\*2

dist = math.sqrt(avg)

print dist,features[-1]

**reducer – code:**

import sys

from operator import itemgetter

distances = []

estimateVal = 0.0

k = 5

for line in sys.stdin:

line = line.strip()

elements = line.split()

distances.append(elements)

distances = sorted(distances)

for i in range(k):

try:

estimateVal += float(distances[i][1])

except ValueError:

continue

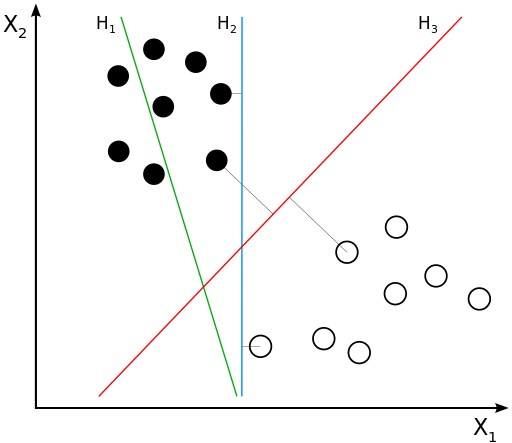
print estimateVal/k

**Applications**

1. **Recommender Systems:** To find similar items, we store the set of users who like each item and if a similar set of users like two different items, then the items themselves are probably similar.
2. **Face Detection:** We can use deep learning algorithms to generate feature vectors representing the peoples faces and then use KNN to identify a person.

**Support Vector Machines**

* In machine learning, **support vector machines** are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.
* More formally, a support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class, since in general the larger the margin the lower the generalization error of the classifier



H1 does not separate the classes. H2 does, but only with a small margin. H3 separates them with the maximum margin.

## **Linear SVM**

We are given a training dataset of n points of the form

****

where the yi are either 1 or −1, each indicating the class to which the point xi belongs. Each xi is a p-dimensional real vector. We want to find the "maximum-margin hyperplane" that divides the group of points xi for which yi = 1 from the group of points for which yi,= -1 which is defined so that the distance between the hyperplane and the nearest point xi from either group is maximized.

If the training data are linearly separable, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible. The region bounded by these two hyperplanes is called the "margin", and the maximum-margin hyperplane is the hyperplane that lies halfway between them. These hyperplanes can be described by the equations

****

and



****

Geometrically, the distance between these two hyperplanes is  ,so to maximize the distance between the planes we want to minimize . As we also have to prevent data points from falling into the margin, we add the following constraint: for each i either

****

or

****

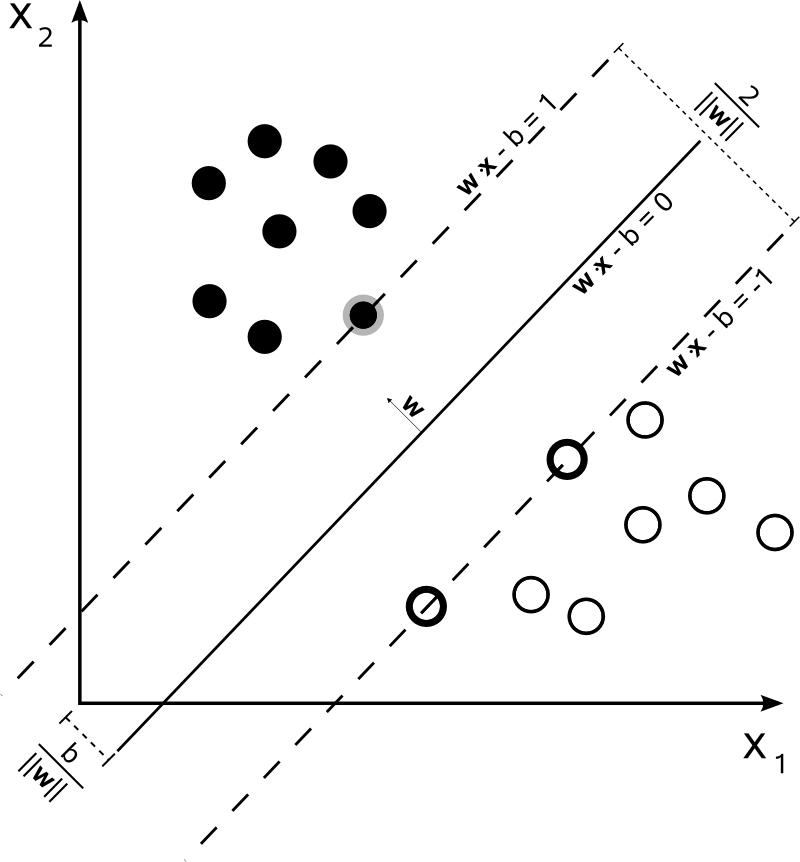
These constraints state that each data point must lie on the correct side of the margin.

This can be rewritten as:

****

The and b that solve this problem determine our classifier,

****

****

**Implementation in Python**

import matplotlib.pyplot as plt

from matplotlib import style

import numpy as np

import math

from decimal import Decimal

style.use('ggplot')

class Support\_Vector\_Machine:

def \_\_init\_\_(self, visualisation=True):

self.visualisation = visualisation

self.colors = {1:'r', -1:'b'}

if self.visualisation:

self.fig = plt.figure()

self.ax = self.fig.add\_subplot(1, 1, 1)

def fit(self, data):

self.data = data

opt\_dict = {}

transforms = [[1, 1], [1, -1], [-1, 1], [-1, -1]]

all\_data = []

for yi in self.data:

for featureSet in self.data[yi]:

for feature in featureSet:

all\_data.append(feature)

self.min\_feature\_value = min(all\_data)

self.max\_feature\_value = max(all\_data)

all\_data = None

step\_sizes = [self.max\_feature\_value\*0.1, self.max\_feature\_value\*0.01, self.max\_feature\_value\*0.005]

b\_range\_multiple = 2

b\_multiple = 5

latest\_optimum = self.max\_feature\_value \* 10

for step in step\_sizes:

w = np.array([latest\_optimum, latest\_optimum])

optimized = False

while not optimized:

for b in np.arange(-1\*(self.max\_feature\_value\*b\_range\_multiple), self.max\_feature\_value\*b\_range\_multiple, step\*b\_multiple):

for transformation in transforms:

w\_t = w \* transformation

found\_option = True

for yi in self.data:

for xi in self.data[yi]:

if not yi\*(np.dot(w\_t, xi)+b) >= 1:

found\_option = False

break

if found\_option:

opt\_dict[np.linalg.norm(w\_t)] = [w\_t, b]

if w[0] <= 0:

optimized = True

print('Optimised a step')

else:

w = w - step

norms = sorted([n for n in opt\_dict])

opt\_choice = opt\_dict[norms[0]]

self.w = opt\_choice[0]

self.b = opt\_choice[1]

latest\_optimum = opt\_choice[0][0] + step\*2

def predict(self, features):

classification = np.sign(np.dot(np.array(features), self.w) + self.b)

if classification!=0 and self.visualisation:

self.ax.scatter(features[0], features[1], s=200, marker='\*', color=self.colors[classification])

return classification

def visualize(self):

for i in data\_dict:

for x in data\_dict[i]:

self.ax.scatter(x[0], x[1], s=100, color=self.colors[i])

def hyperplane(x, w, b, v):

return (-w[0]\*x-b+v)/w[1]

data\_range = (self.min\_feature\_value\*0.9, self.max\_feature\_value\*1.1)

hyp\_x\_min = data\_range[0]

hyp\_x\_max = data\_range[1]

psv1 = hyperplane(hyp\_x\_min, self.w, self.b, 1)

psv2 = hyperplane(hyp\_x\_max, self.w, self.b, 1)

self.ax.plot([hyp\_x\_min, hyp\_x\_max], [psv1, psv2])

nsv1 = hyperplane(hyp\_x\_min, self.w, self.b, -1)

nsv2 = hyperplane(hyp\_x\_max, self.w, self.b, -1)

self.ax.plot([hyp\_x\_min, hyp\_x\_max], [nsv1, nsv2])

db1 = hyperplane(hyp\_x\_min, self.w, self.b, 0)

db2 = hyperplane(hyp\_x\_max, self.w, self.b, 0)

self.ax.plot([hyp\_x\_min, hyp\_x\_max], [db1, db2])

plt.show()

data\_dict = {-1: np.array([[1, 7], [2, 8], [3, 8]]), 1: np.array([[5, 1], [6, -1], [7, 3]])}

svm = Support\_Vector\_Machine()

svm.fit(data=data\_dict)

svm.visualize()

**Applications**

1. Hand-written characters can be recognized using SVM
2. Classification of images can also be performed using SVMs

**Conclusion**

In this project, we overviewed the possiblities and future with big data and it’s applications in the field of machine learning. We have successfully learnt the hadoop technology for big data analysis and applied for word count in a file. We also learnt about the hadoop framework and all its underlying layers.

The main focus of this project is to apply hadoop technology to the computationally expensive machine learning algorithms, and we have done considerably good work by applying the hadoop MapReduce paradigm to the k-Nearest Neighbors algorithm.