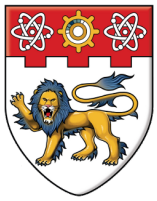

NANYANG TECHNOLOGICAL UNIVERSITY

COMPUTATIONAL COGNITION FOR BIG DATA ANALYTICS

YUAN YIYANG

Matriculation Number: U1320793E

Project Number: SCE15-0772

Supervisor: Assoc Prof Suresh Sundaram

School of Computer Science and Engineering

Faculty of Engineering

Nanyang Technological University

October 2016

A Final Year Project - Undergraduate Research Experience on CAmpus (FYP-URECA) Report presented to the Nanyang Technological University in partial fulfillment of the requirements for the degree of Bachelor of Engineering (Computer Science)

ABSTRACT

The machine learning algorithms like Projection Based Learning algorithm in Meta-cognitive Radial Basis Function Network (PBL-McRBFN) look up to human brain in attempts to make its computation more efficient. However, the performance of this algorithm is limited by the nature of sequential learning. In this project we bring Big Data-era technology Apache Spark cluster computing platform to use with this algorithm in an attempt to achieve collaborative distributed learning. Amidst the various difficulties, the project achieved a partial success while discovering other potential issues in using Spark to achieve the goal.

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth [9]

ACKNOWLEDGMENTS

The author would like to express his heartfelt gratefulness to Associate Professor Suresh Sundaram, for the guidance throughout this one-year Final Year Project. The author is particularly thankful that Professor Suresh would make his time available whenever the author needed to approach him for clarification and discussion. Without this, it could have been difficult to progress at times.

The author also would like to acknowledge the support from Nanyang Technological University under the Undergraduate Research on CAmpus (URECA) programme, for FYP-URECA AY2015-16.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CODE LISTINGS

ACRONYMS

API Application Programming Interface

McRBFN Meta-cognitive Radial Basis Function Network

MPI Message Passing Interface

PBL-McRBFN Projection Based Learning algorithm in Meta-cognitive Radial Basis Function Network

PBL Projection Based Learning

RBF Radial Basis Function Network

RDD Resilient Distributed Dataset

Part I

BACKGROUND

1

INTRODUCTION

1.1 MOTIVATION

Big Data Analytics, without a doubt, is one of the hottest topics in the recent years. Big data is a very large collection of data, structured or unstructured, which may change rapidly. It uses large clusters of computers to find out correlations, predictive trends and patterns which can lead to making more informed decisions. It is often highlighted by three Vs: Volume, Velocity and Variety. When all these factors are high, it requires new technologies to store, process and analyze the data beyond what traditional relational database can provide.[16]

In Big Data Analytics, one important area is to classify data into their categories. Artificial Neural Network (ANN) is one of the many ways to tackle classification problems. It is based on imitation of the biological neural network to emulate human learning process. These algorithms are implemented and known as the classifiers, and are part of supervised learning, of which training set contains labelled samples.

Meta-cognitive ANN algorithms is able to self-learn with their cognitive component. The meta-cognitive part evaluates and decides the order of learning, that is what, when and how to learn, while the cognitive component computes the optimal

weights corresponding to the minimum of the energy function. PBL-McRBFN is the basis of this project, and it will build on the previous studies done by (Babu and Suresh, 2013).

In the field of machine learning, the batch learning algorithms use complete training data in making the computational model. In many real-world uses, this completeness of data is not available beforehand. Therefore, sequential learning is necessary to tackle these problems with which batch learning is unsuitable to be used. In a sequential framework, the data samples are processed one after another, and removed after they are learned. Therefore, it takes much less memory and time in the learning process.

However, as PBL-McRBFN usually deals with relatively large dataset, the sequential learning framework on single process machine can take tremendous amount of computational time to process the large set of data. Therefore, it is worth trying to parallelize the PBL-McRBFN machine learning by scaling it to a cluster of multiple machines. In order to achieve this, we propose to collaborate among the distributed machines in a cluster.

1.2 DISTRIBUTED MACHINE LEARNING

Distributed Computing is not a new concept. In the past distributed computing used to be more complex, normally involving a central dispatcher giving out tasks to the workers for processing, while the worker nodes return their results periodically.

For machine learning dealing with huge amount of data, we may not be able to process them all on a single processor. Therefore, we can achieve faster learning by leveraging on distributed computing to task a cluster of computers to do machine learning concurrently. We can choose to give out tasks based on different subset of the data, while sharing the learned information during the learning process to improve the quality of learned knowledge. The learners pass messages to each other, such as asking questions or sharing learned knowledge. These types of communication can benefit the overall learning collectively.[15]

Apache Spark is a newer technology, a cluster computing system. It supports in-memory computing rather than dumping data to disks. This provides massive speedup from previous technologies, hundred times faster than Hadoop MapReduce in memory and ten times than on disk, according to Apache Spark's website. The users deal with abstract logic of workers working directly on datasets.

Apache Spark uses Resilient Distributed Dataset (RDD) as its underlying programming model, in which data are split and distributed to cluster nodes. Each of these nodes, known as a worker, processes the data independently using the same programmes. Their results are gathered by the master processor.

1.3 ORGANIZATION OF CHAPTERS

This project is an attempt to bring the algorithm outlined in Projection Based Learning algorithm in Meta-cognitive Radial Ba-

sis Function Network (PBL-McRBFN) for Classification Problems, into Spark framework to effectively scale up the cluster. The author took effort to understand how Spark framework might be able to aid towards such a goal, and use the framework to find out the best parameters to the algorithm in given data set.

This report will continue with the following main chapters. Chapter 2 is mainly on literature review about Big Data and Machine Learning. It covers the mathematical basis of PBL-McRBFN algorithm in detail, and presents the need to make use of Spark framework. Chapter 3 outlines the algorithm realization in Python, from a previous MATLAB realization. It highlights some of the differences in MATLAB and NumPy to the readers who may be interested in transiting between the two languages. It also presents the Data Set used in the project. Chapter 4 talks about how Apache Spark is used in the project. It shows the results of the Spark Realization particularly the runs to optimize the algorithm's parameters. It also explains why Spark may NOT be the suitable framework to use to achieve collaborative distributed learning. Chapter 5 concludes the project with possible future directions.

Part II

THE SHOWCASE

2

BIG DATA ANALYSIS

2.1 DEFINITIONS TO BIG DATA

What is big data? Various sources have given a range of definitions to the term. Some of them are listed as the following:[4]

- Big Data is "data whose size forces us to look beyond the tried-and-true methods that are prevalent at that time" Jacobs (2009)
- "When the size of the data itself becomes part of the problem and traditional techniques for working with data run out of steam" Loukides (2010)
- "Big data is high volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization" Laney (2001), Manyika et al. (2011)
- "Big Data technologies [are] a new generation of technologies and architectures designed to extract value economically from very large volumes of a wide variety of data by enabling high-velocity capture, discovery, and/or analysis" IDC (2011)
- "Big Data is a term encompassing the use of techniques to capture, process, analyse and visualize potentially large

datasets in a reasonable timeframe not accessible to standard IT technologies." NESSI (2012)

- "Big data can mean big volume, big velocity, or big variety" Stonebraker (2012)

As seen from the definitions listed, Big Data enables us to use large clusters of computers to find out correlations and predictive trends and patterns which can lead to making more informed decisions. It is often highlighted by three Vs: Volume, Velocity and Variety. When dealing with such data, new areas of technologies are explored to effectively analyze the data. One of such areas is Meta-cognitive Learning.

2.2 META-COGNITIVE LEARNING

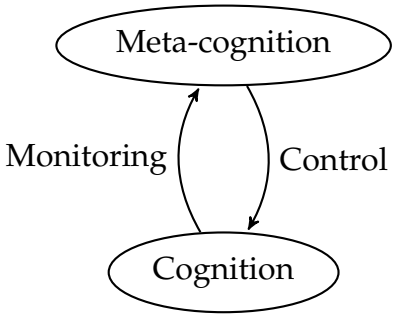

Figure 1: Nelson and Naren Model.

One simpler model among the meta-cognitive models in the field of psychology is the Nelson and Naren model shown in Figure 1. It consists of two components, a meta-cognitive part and a cognitive one. Control signals are the information sent from Meta-cognition to Cognition, during which the meta-level modifies the cognitive level. This changes the state or the process in the cognition level. On the other hand, Monitoring signals are the information sent from Cognition to Meta-cognition,

in which the cognitive level informs the meta-cognitive level. The state of model in the meta-level is thus changed.[13]

Following the Nelson and Naren model, the Meta-cognitive Radial Basis Function Network (McRBFN) has both the meta-cognitive and cognitive components. The meta-cognitive component is a self-regulatory dynamic model of the cognitive model, while the cognitive part consists of a Radial Basis Function Network (RBF) network of single hidden layer neurons forming an evolving architecture. As with the Control signal in the model, information is sent from meta-cognitive to cognitive component, selecting the best learning strategy. On the other hand, as with the Monitoring signal in the model, information is sent back from cognitive to meta-cognitive component, predicting the output of classifications. Projection Based Learning algorithm in Meta-cognitive Radial Basis Function Network (PBL-McRBFN) is similar to the human meta-cognitive learning process, making the classifier more effective in the pattern classification.

2.2.1 *Underlying Mathematics*

To understand PBL-McRBFN, we define the classification problem in sequential Projection Based Learning (PBL) as the following:

Given $\{(x^1, c^1), ..., (x^t, c^t), ...\}$ as a stream of incoming training data samples, where $x^t = [x_1^t, ..., x_m^t]^T \in \mathcal{R}^m$ is the t^{th} data sample which has an m-dimensional input, whose class label is $c^t \in \{1, ..., n\}$, and the total number of classes given by n.

$(y^t = [y_1^t, ..., y_j^t, ..., y_n^t]^T) \in \mathcal{R}^n$ gives the coded class labels by the equation:

$$y_j^t = \begin{cases} 1, & \text{if } c^t = j \\ -1, & \text{otherwise} \end{cases} \qquad j = 1...n \tag{1}$$

The PBL-McRBFN classifier aims to estimate the underlying decision function, mapping $x^t \in \mathcal{R}^m \rightarrow y^t \in \mathcal{R}^n$. It starts with no hidden neurons, and with each incoming data sample, it chooses a suitable strategy to address the question of *what, when and how* to learn.

2.2.2 *Architecture*

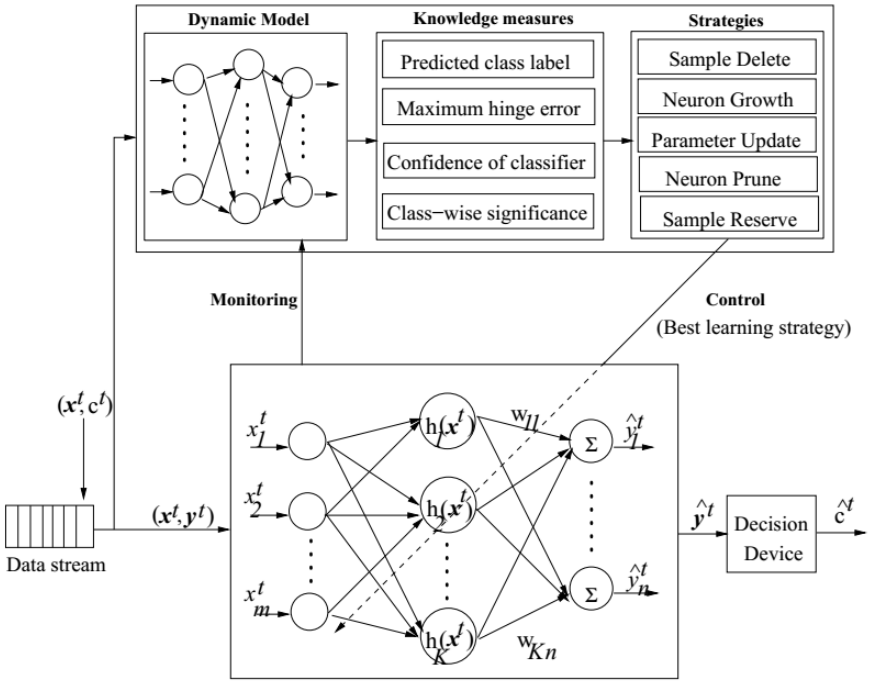

Figure 2: McRBFN Architecture.

Figure 2 shows the schematic diagram of the McRBFN architecture. In McRBFN, learning of training data samples takes place in the cognitive component. It adds new neurons in the single hidden layer RBF network, and the output weights of the neurons are updated to find approximate solutions to the underlying decision function. The centre and width of the input weights are calculated from both the incoming training data sample and the output weights of the neurons, which are approximated with the Projection Based Learning (PBL). The new hidden neurons are initialized using the sample overlapping condition and pseudo-samples from the past knowledge, minimizing the error in classification labeling.[2]

2.2.3 *Learning Strategies*

The meta-cognitive component exercises control over the cognitive learning process, chooses an appropriate learning strategies from the following:

Sample Delete addresses *what-to-learn*. For new training sample that has little new information due to high similarity to knowledge present in the cognitive part, it will not use the new training data sample to learn, but remove it from the set.

Neuron Growth addresses *how-to-learn*. It learns from the new training data sample, and adds a new hidden neuron in the RBF network in cognitive part. The neuron is initialized by identifying the sample overlapping conditions.

Parameter Update addresses *how-to-learn* The parameters for the cognitive part are updated by PBL with the new training data sample.

Neuron Pruning addresses *how-to-learn*. It deletes the less used neurons in a class from the cognitive component.

Sample Reserve addresses *when-to-learn*. It reserves the training data sample for future use, as the information in the sample is not significant enough for immediate learning.

2.2.4 *Meta-cognitive component*

The meta-cognitive part is a self-regulatory dynamic model of the cognitive counterpart. It updates the model as it monitors the cognitive during learning. The meta-cognitive part gives an estimation to the potential knowledge contained in a new training data sample (x^t, y^t). It has self-regulated thresholds adapting to the new knowledge. It would choose one of the strategies outlined in Section 2.2.3 based on the knowledge estimation and thresholds.

The knowledge estimation have the following measures:

2.2.4.1 *Predicted class label ($\hat{c}^t$)*

It is defined using the predicted output ($\hat{c}^t$)

$$\hat{c}^t = arg \max_{j \in 1,\ldots,n} \hat{y}_j^t \tag{2}$$

2.2.4.2 *Hinge error (E^t)*

The hinge loss error $e^t = \left[e_1^t, ..., e_j^t, ..., e_n^t\right]^T \in \mathcal{R}^n$ has its definition as following

$$e_j^t = \begin{cases} 0 & \text{if } y_j^t \hat{y}_j^t \geqslant 1 \\ y_j^t - \hat{y}_j^t & \text{otherwise} \end{cases} \quad j = 1, ..., n \tag{3}$$

The maximum hinge error (E^t) can be obtained from hinge loss error (e^t) by

$$E^t = \max_{j \in 1,...,n} |e_j^t| \tag{4}$$

2.2.4.3 *Classifier Confidence ($\hat{p}(c^t|x^t)$)*

The classifier confidence is calculated by

$$\hat{p}(c^t|x^t) = \frac{\min(1, \max(-1, \hat{y}_j^t)) + 1}{2}, \quad j = c^t \tag{5}$$

2.2.4.4 *Class-wise significance (ψ_c)*

The class-wise significance is the spherical potential of the training data sample in the class. For K^c neurons in a given class c, the class-wise significance (ψ_c) is calculated by

$$\psi_c = \frac{1}{K^c} \sum_{k=1}^{K^c} h(x^t, \mu_k^c) \tag{6}$$

2.2.4.5 *Cognitive component*

The cognitive part of McRBFN consists of a single-layered feed-forward RBF network. Gaussian activation function is used to

model the hidden neurons.

Without loss of generality, the K neurons are assumed to be built from $t-1$ training data samples. The predicted McRBFN output $\hat{y}_j^t$ for any input x^t is

$$\hat{y}_j^t = \sum_{k=1}^{K} w_{kj} h_k^t, \quad j = 1, ..., n \tag{7}$$

where w_{kj} is the weight from k^{th} to j^{th} neuron; h_k^t is the response of k^{th} neuron to input x^t. The response h_k^t is calculated by

$$h_k^t = \exp\left(-\frac{||x^t - \mu_k^l||^2}{(\sigma_k^l)^2}\right) \tag{8}$$

where $\mu_k^l \in \mathcal{R}^m$ is the centre of the k^{th} hidden neuron and $\sigma_k^l \in \mathcal{R}^+$ is its width; l corresponds to the class label of the neuron.

PBL algorithm minimizes the error function to optimize the network output weights. The error function, which is the summation of the squared-errors of output neurons, for the i^{th} sample is:

$$E_i = \sum_{j=1}^{n} \left(y_j^i - \sum_{k=1}^{K} w_{kj} h_k^i\right)^2 \tag{9}$$

The overall error function for t training samples is expressed as

$$\begin{aligned} E(\mathbf{W}) &= \frac{1}{2}\sum_{i=1}^{t} E_i \\ &= \frac{1}{2}\sum_{i=1}^{t}\sum_{j=1}^{n}\left(y_j^i - \sum_{k=1}^{K} w_{kj} h_k^i\right)^2 \end{aligned} \tag{10}$$

in which h_k^i is the output of the k^{th} neuron for the i^{th} training data sample.

The optimal output weights $\left(\mathbf{W}^* \in \mathcal{R}^{K\times n}\right)$ are approximated to get minimum total error ($E(\mathbf{W})$), as such

$$\mathbf{W}^* = arg \min_{\mathbf{W}\in\mathcal{R}^{K\times n}} E(\mathbf{W}) \tag{11}$$

The optimal W^* gives the minimum total error for $E(\mathbf{W}^*)$ when the first partial differential of $E(\mathbf{W})$ with respect to output weight is zero. This is shown by the equation

$$\frac{\partial E(\mathbf{W})}{\partial w_p j} = 0, \quad p = 1, ..., K; \quad j = 1, ..., n \tag{12}$$

By rearranging the equation, we can get,

$$\sum_{k=1}^{K}\sum_{i=1}^{t} h_k^i h_p^i w_{kj} = \sum_{i=1}^{t} h_p^i y_j^i \tag{13}$$

It can be expressed in a matrix format,

$$\mathbf{AW} \quad = \quad \mathbf{B} \tag{14}$$

whereby the matrix $\mathbf{A} \in \mathcal{R}^{K \times K}$ is

$$a_{kp} = \sum_{i=1}^{t} h_k^i h_p^i, \quad p = 1, ..., K; \quad k = 1, ..., K \tag{15}$$

and the output B $\mathbf{B} \in \mathcal{R}^{K \times n}$ is

$$b_{pj} = \sum_{i=1}^{t} h_p^i y_j^i, \quad p = 1, ..., K; \quad j = 1, ..., n \tag{16}$$

Equation 14 is a set of $K \times n$ linear equations. The $K \times n$ weights $\mathbf{W}$ are unknown. As the solution for Equation 14, $\mathbf{W}$ is at minimum when $\frac{\partial^2 J}{\partial w_{ip}^2} > 0$. The following equation demonstrates that E is a convex function, and $W*$ is the weight for minimizing E in the error equation:

$$\begin{aligned} \frac{\partial^2 E(\mathbf{W})}{\partial w_{ip}^2} &= \sum_{i=1}^{t} h_p^i h_p^i \\ &= \sum_{i=1}^{t} |h_p^i|^2 > 0 \end{aligned} \tag{17}$$

Therefore, Equation 14 can be solved by finding

$$\mathbf{W}^* \quad = \quad \mathbf{A}^{-1}\mathbf{B} \tag{18}$$

2.3 FRAMEWORK

The Apache Spark platform became popular for the Big Data Analytics in the recent years. It has an underlying MapReduce engine allowing easy scaling of computation. Spark uses Resilient Distributed Dataset (RDD) as an intermediate storage. RDD is a set of elements, can be parallelized and partitioned across cluster nodes. Data are transformed from one RDD to another by actions ultimately corresponding to MapReduce. The Apache Spark platform has Application Programming Interface (API) available in Scala, Python, Java and R. The Spark programmes themselves run in Java Virtual Machines (JVM) thus it can be extended to multiple languages.

For our project, we hope to use Spark to effectively scale up the PBL-McRBFN algorithm, thus to achieve distributed machine learning. The original proof-of-concept single-thread implementation was in MATLAB. It is not supported by Spark. To bring it to compatible format, the author leveraged on the close resemblance between MATLAB and NumPy. NumPy is a Python library, which is commonly used in scientific calculations. However, there are challenges involved due to the differences in between the two. In next chapter we will talk about that in detail.

3

LEARNING ALGORITHM IN PYTHON REALIZATION

3.1 PBL-MCRBFN ALGORITHM IN PSEUDO-CODE

Code Listing 1: PBL-McRBFN Pseudo Code

```
function PBL-McRBFN(sample):
  inputs:
    sample: training data
  outputs:
    neuron: (centre, weights)

begin
  while sample:
  begin
    Meta_Cognitive_Component:
    begin
      Calculate c_hat
      Calculate Maximum Hinge Error E
      Calculate p_hat
      Calculate \psi_c
      learning_strategy
      Return learning_strategy
    end

    Cognitive_Component:
    begin
      case learning_strategy of:
        SAMPLE_DELETE:
          DELETE sample
        NEURON_GROWTH:
          ASSIGN (centre, width)
        PARAMETER_UPDATE:
          UPDATE parameter
          DELETE sample
        NEURON_PRUNING:
          PRUNE neuron
        SAMPLE_RESERVE:
          RESERVE sample
      end case
    end
  end while
end
```

This algorithm can be realized using MATLAB in a straight forward manner, as it is rather procedural. MATLAB also enables easy manipulation of matrices, such as transposing or finding dot-products of the matrices. These are frequently used, such as to find the optimal weights, or computing the knowledge measures in Section 2.2.4 in the previous chapter.

Although MATLAB is a suitable candidate in realizing the algorithm in using single-thread single-process, the Spark platform does not support MATLAB, and thus we have to look into using Python in realizing the algorithm. Since there is existing proof-of-concept MATLAB code, it is used as reference to be rewritten into Python using NumPy library.

3.2 FROM MATLAB TO PYTHON

As the author has not used MATLAB extensively before this project, a few quick guide to MATLAB were gathered. These are recommended for Computer Science background students to have a quick understanding and get started with MATLAB. One of the resource available is *Learn Matlab in Y Minutes*.[12] The guide highlights some of the most important syntaxes in MATLAB, and is thus useful to understanding Matlab notebooks. This section hopes to help future Computer Science students or researchers to transit between MATLAB and NumPy with ease.

Table 1: Major Equivalents in MATLAB and NumPy

Item	MATLAB	NumPy
pre-defined list/array	kmax = [10 5 15];	kmax = [10, 5, 15]
find the maximum	ny = max(nyi);	ny = max(nyi)
find index of maximum	[~,chat] = max(yhat);	chat = np.argmax(yhat[0, :])
zero-filled array	conf_tra = zeros(ny,ny);	conf_tra = np.zeros((ny, ny))
2nd dimension length	for m = 1 : size(uy1,2)	for m in range(uy1.shape[1]):
array indexing	x = uy1(nxi,m);	x = uy1[nxi, m]
dot product	yhat = Phi*Beta;	yhat = np.dot(Phi, Beta)

3.2.1 *The Commonalities*

Two important sources to look up for MATLAB-Python equivalents are The-Scipy-Community *Numpy for Matlab Users* and Mathesaurus *Numpy for MATLAB Users*. Applying knowledge from these two sources, it is possible to match quite a large portion of MATLAB code to NumPy Python. Table 1 shows some of the more often used expressions.

However, apart from these more obvious equivalents, the following ones need to be dealt with care.

3.2.1.1 *MATLAB uses 1 indexing as opposed to 0 in Python*

This is one major difference between the two languages, and they are very frequently used. This can have implications in complicating the programme logics. For example, in the MATLAB code, the sample ID is used for the indexing in array as well as part of the initial condition to some of the loops. The difference between the two means that the initial and ending conditions for many loops must be rewritten.

The following MATLAB code

Code Listing 2: MATLAB Array Access

```
A(1:K,K) = A(1:K,K) + Phit(1,1:K)'*Phit(K);
```

looks like this in Python

Code Listing 3: Python Array Access

```
A[0:K + 1, K] = A[0:K + 1, K] + Phit[0, 0:K + 1].transpose()
    * Phit[0, K]
```

This is certainly rather confusing and error-prone.

3.2.1.2 *MATLAB deals with Range differently*

In MATLAB, range can be used directly to create slices and indexes. For an example, the following is a legitimate MATLAB expression:

Code Listing 4: Matlab Range as Index Slice

```
    nxi = 19:-1:1;
    UY  = UY(nxi,:);
```

However, this has no direct equivalence in Python. In python, we have to create an array containing those items in *nxi*, then filter out the rows we need from *UY*.

3.3 DATA LOADING

Within the Apache Spark framework, as mentioned, multiple programming languages are supported, including Python, Scala, Java, R, etc. While attempting Apache Spark with Scala at the initial stage, the author realized that there were no good versatile Scala libraries to decode MATLAB data.

Some of the available libraries include

BIDMat supports a MATLAB-compatible HDF5 format. This is equivalent to Mat-file version 7.3. However Mat-file versions are not backward compatible, meaning if the Mat-file is from an earlier version, it would not be supported.[3]

JMatIO only supports Mat-file version 6, was last updated more than two years ago.[8]

scipy.io Fortunately, in SciPy it's much easier to access the Mat-file data. It provides support for Mat-file version 4, 6, 7 to 7.2.[5]

Therefore to simplify the matter, Mat-file data to pass through a python script similar to below, and dumped them into JSON format. The JSON format data are then loaded into programme as and when needed.

Code Listing 5: scipy.io loadmat

```
import scipy.io as sio
import json
mat_contents = sio.loadmat('seg_data.mat')
```

3.4 DATA SET

For this project, we run the algorithm against the Image Segmentation Data Set[10] available from the University of California, Irvine (UCI). This data set contains 2310 instances with 19 attributes. They were collected randomly from 7 outdoor images.

segmentation.data.csv

	A	B	C	D	E	F	G	H	I	J	K
	LABEL	REGION...	REGION...	REGION...	SHORTLI...	SHORTLI...	VEDGEM...	VEDGESD	HEDGEM...	HEDGESD	INTENSI...
	TEXT	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER
28	BRICKFACE	18.0	138.0	9	0.0	0.0	0.88888...	0.5629629	0.83333...	0.29999...	5.740741
29	BRICKFACE	138.0	133.0	9	0.0	0.0	0.6666667	0.44444...	1.1666666	0.21111...	6.4444447
30	BRICKFACE	121.0	113.0	9	0.0	0.0	1.722222	1.5296303	2.944444	1.5296295	20.25926
31	BRICKFACE	95.0	57.0	9	0.0	0.0	1.8333327	3.4111106	2.1111107	1.7185175	26.296297
32	SKY	140.0	25.0	9	0.0	0.0	0.99999...	1.4666697	1.1111107	0.11851...	128.0
33	SKY	142.0	33.0	9	0.0	0.0	0.49999...	0.62360...	0.50000...	0.3496027	110.59259
34	SKY	66.0	41.0	9	0.0	0.0	0.61111...	0.32773...	0.38889...	0.32773...	109.703...
35	SKY	165.0	99.0	9	0.0	0.0	0.88889...	0.47407...	0.7777786	0.47407...	93.40741
36	SKY	228.0	20.0	9	0.0	0.0	1.0555547	0.49065...	0.8333333	0.7527733	125.0
37	SKY	124.0	29.0	9	0.0	0.0	1.0000013	0.7111076	1.0555534	0.90741...	128.44444
38	SKY	156.0	32.0	9	0.0	0.0	0.77777...	0.16296...	2.6111095	1.0407379	136.2963
39	SKY	21.0	90.0	9	0.0	0.0	0.66666...	0.04444...	0.7777786	0.56296...	113.48148
40	SKY	8.0	39.0	9	0.11111...	0.0	1.3888906	1.129629	1.8333334	0.699999	113.37037
41	SKY	122.0	11.0	9	0.0	0.0	1.0	0.31111...	2.8888905	5.051852	143.44444
42	SKY	44.0	79.0	9	0.0	0.0	0.44444...	0.34426...	0.7777786	0.4036864	107.74074
43	SKY	7.0	18.0	9	0.0	0.0	1.2777786	0.7296265	0.9444453	0.37407...	138.62962
44	SKY	188.0	42.0	9	0.0	0.0	0.7777786	0.5443299	1.6666679	1.2649081	108.92593
45	SKY	152.0	18.0	9	0.0	0.0	0.7777774	0.4554219	0.55555...	0.2721644	112.111...
46	SKY	120.0	74.0	9	0.0	0.0	0.3333346	0.08888...	0.50000...	0.07777...	101.85185

Figure 3: Image Segmentaion Data Set (Screenshot)

The attributes include:

- region centroid: row and column;
- region pixel count;
- short line density (<=5 and <=2);
- v-edge (horizontally adjacent pixels' contrast): mean and standard deviation, used to detect vertical lines;
- h-edge: mean and standard deviation, used to detect horizontal lines;
- means for intensity (RGB), raw red (R), raw blue (B), raw green (G), excess red (2R-G-B), excess blue (2B-G-R), excess green (2G-R-B), value, saturation and hue.

To ensure the Python code works effectively the same as the original MATLAB code, all variables contents are written out at various points and also at the end of the programme. The contents were compared between the two codes, and verified to have the same results based on the same data supplied.

4

LEARNING ALGORITHM IN SPARK REALIZATION

4.1 APACHE SPARK

As introduced in Section 1.2, we hope to achieve faster learning by leveraging on distributed computing to task a cluster of computers to do machine learning concurrently. The Apache Spark is a cluster computing system that supports in-memory computing and provides massive speedup from previous technologies. We attempt to use Spark to abstract the computational and communicational logics of workers, and work directly on datasets.

4.1.1 *Get Started Resources*

There are a number of Get Started with Apache Spark articles or tutorials online. However, many of them are geared towards the Scala implementation, as that is the main language used for Spark, for example *Apache Spark Core Programming* on TutorialPoint.[18] The official Apache Spark guide was neither exactly clear on how to get started. It does not provide a step-by-step guide book.[1]

One good recommended resource was *Apache Spark & Python (pySpark) tutorials for Big Data Analysis and Machine Learning as IPython / Jupyter notebooks*, even though this set of tutorials is not without flaws as well. It uses Python 2, while the project has decided to use Python 3. This uses the Jupyter Notebook (formerly, IPython), which is an interactive notebook for Python, similar how MATLAB Notebook is like for MATLAB.[6]

4.1.2 *Jupyter Notebook runs with Spark*

Jupyter notebook has a user-friendly web interface to create documents that contain live code and results, among other texts and equations.[14] It runs an IPython kernel, which is considered as an enhanced Python interpreter.

The Apache Spark environment can be started with Jupyter Notebook using the following command:

Code Listing 6: Start Jupyter Notebook with PySpark

```
PYSPARK_DRIVER_PYTHON="jupyter" PYSPARK_DRIVER_PYTHON_OPTS="
    notebook" pyspark
```

This leads to the web interface seen in Figure 4. It shows clearly the Jupyter Notebook status, whether the processes are running, or if there are any terminal started, and the other files in the folder.

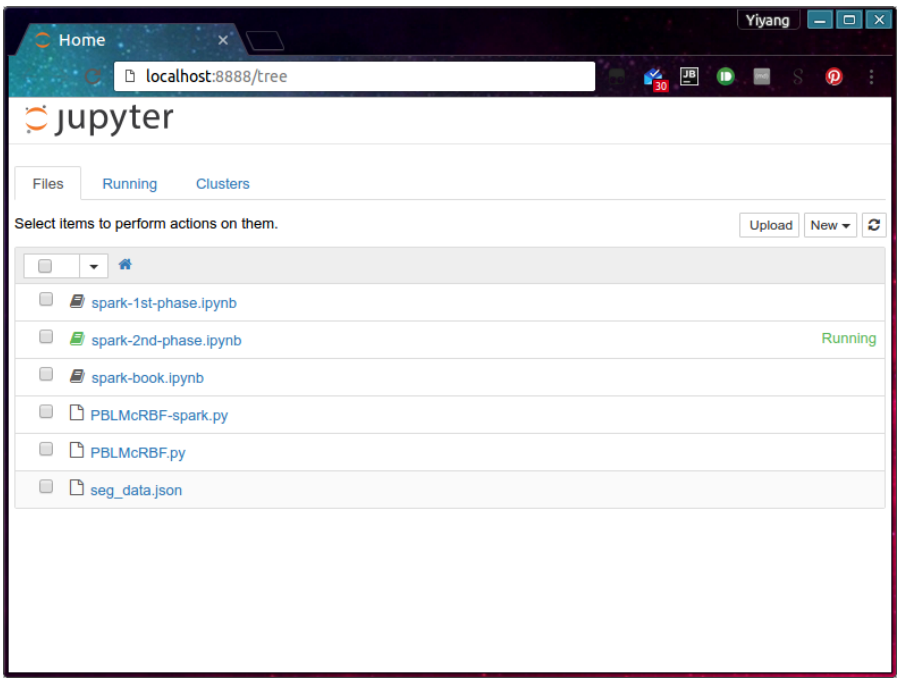

Figure 4: Jupyter Notebook Web Interface.

4.1.3 *Apache Spark Architecture*

Every Apache Spark application includes a driver programme to run the main function and operate other parallel actions on cluster nodes. This architecture diagram is shown in Figure 5. The Resilient Distributed Dataset (RDD) is Apache Spark's main abstraction, providing a set of parallelizable elements partitioned across the nodes. RDDs are either started from files in the Hadoop-supoorted file system (HDFS), or transformed from an existing collection of elements in the driver programme. The RDDs can be persisted in memory, and be reused efficiently when parallelized. Furthermore, RDDs are fault-tolerant.

In our project, the PBL-McRBFN programme is captured in a function (See Section A.2). The spark paralellizer would call the function supplying it with different parts of the dataset, thus paraellizing it to run on different worker nodes.

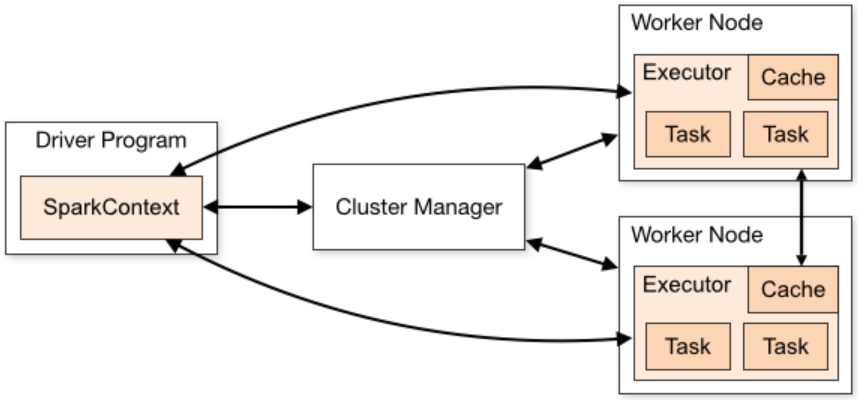

Figure 5: Apache Spark Cluster Architecture.

4.2 CONTROL LEVEL OPTIMIZATION

To achieve optimization in the parameters supplied to initialize the PBL-McRBFN algorithm, a template in Code Listing 7 is used to find the optimum range of parameters. It generates a list of parameters within the intended range, and use Apache Spark to parallelize the computation. Refer to Section A.2 for the `common` function.

Code Listing 7: Optimizing Parameters

```
par = [0.31117, 1.5445, 0.73983, 0.011366, 0.97652, 31,
    0.58887, 0.08741, 0.4879]
#     [Ed,      Eadd,   Elearn,  r,        kp,     Pmax,
       P1,    zeta,    kp1]
vEd = [x/1000 for x in range(111,  512, 100)]
vEadd = [x/100 for x in range(110, 171, 20)]
vElearn = [x/100 for x in range(50, 99, 9)]
vr = [x/1000 for x in range(1, 102, 20)]
vkp = [x/100 for x in range(80, 99, 5)]
parameter = []
for xEd in vEd:
 for xEadd in vEadd:
# for xElearn in vElearn:
#  for xr in vr:
#   for xkp in vkp:
#    parameter.append([xEd, xEadd, xElearn, xr, xkp]+par
    [5:])
     parameter.append([xEd, xEadd] + par[2:])
ppar = sc.parallelize(parameter)
exp = ppar.map(lambda x: common(x))
```

The parallelization means that even on a single multi-core machine, the single-threaded python implementation can be run as many as the CPU cores concurrently. This brings a 4x speedup to test out the parameters needed on a quad-core machine. The result from this block is retrieved by `exp.collect()`. We are interested to find out which set of parameters can give to the highest score in the testing performance matrix. Thus in each run, the parameters starting with (Ed, Eadd) are set to a narrower range, or expanding to the next parameters (Elearn, r, kp) while keeping the previous ones fixed.

Table 2: Runs on Parameter Optimization

#	ED	EADD	ELEARN	R	KP
1	0.31117	1.5445	0.73983	0.011366	0.97652
2	[0.111,0.512)	[1.10,1.71)	0.73983	0.011366	0.97652
3	[0.111,0.312)	[1.10,1.26)	0.73983	0.011366	0.97652
4	[0.261,0.312)	[1.10,1.11)	0.73983	0.011366	0.97652
5	[0.261,0.312)	[1.10,1.11)	[0.50, 0.99)	0.011366	0.97652
6	[0.261,0.312)	[1.10,1.11)	[0.47, 0.68)	0.011366	0.97652
7	[0.261,0.312)	[1.10,1.11)	[0.510, 0.561)	0.011366	0.97652
8	[0.261,0.312)	[1.10,1.11)	[0.516, 0.525)	0.011366	0.97652
9	[0.261,0.312)	[1.10,1.11)	[0.520, 0.522)	0.011366	0.97652
10	[0.261,0.312)	[1.10,1.11)	[0.520, 0.522)	[0.001, 0.102)	0.97652
11	[0.261,0.312)	[1.10,1.11)	[0.520, 0.522)	[0.005, 0.019)	0.97652
12	[0.261,0.312)	[1.10,1.11)	[0.520, 0.522)	[0.009, 0.014)	0.97652
13	[0.261,0.312)	[1.10,1.11)	[0.520, 0.522)	[0.0109, 0.0115)	0.97652
14	[0.261,0.312)	[1.10,1.11)	[0.520, 0.522)	[0.0109, 0.0115)	[0.80,0.99)
15	[0.261,0.312)	[1.10,1.11)	[0.520, 0.522)	[0.0109, 0.0115)	[0.88,0.99)
16	[0.261,0.312)	[1.10,1.11)	[0.520, 0.522)	[0.0109, 0.0115)	[0.965,0.986)
17	[0.261,0.312)	[1.10,1.11)	[0.520, 0.522)	[0.0109, 0.0115)	[0.975,0.986)
18	0.3111	1.105	0.521	0.011366	0.97652

As shown in Table 2, a series of 18 runs were carried out on the PBL-McRBFN algorithm. The parameters are iteratively

changed to a narrower range, till the resultant testing performance matrix score reaches its local maximum. The average score improved from 0.91666 to 0.93333. The spark parallelization makes finding such optimum faster as more parameter configurations can be tested at the same run.

4.3 SPARK PROBLEMS

Apache Spark is NOT

- a Message Passing Interface (MPI) framework
- able to guarantee in which order data is processed
- able to identify which worker node runs the processes

Traditionally, Message Passing Interface is the standard to implement distributed collaborative machine learning. The collaboration is done by passing messages among nodes, asking questions or sharing learned knowledge, as recognized by (Provost and Hennessy 1996). This mechanism persisted before the Big Data era, and was prevalent among the High Performance Computing sector. The MPI framework has stayed mostly the same in the past 25 years. It was an advanced tool back in that era, however, at the current age it is hardly the case any more. The MPI paradigm is challenged by Spark and the like, albeit using rather different approaches. In the MPI framework, the programmers have to manually spread the common data structures across processors. Message exchanges and synchronizations have to be done explicitly. This is in contrast to Spark, which uses a MapReduce framework without the need to explicitly exchange data among worker nodes. The potential sav-

ings are highlighted in Table 3.[7] However, the lack of MPI availability in Spark poses a challenge to achieving our initial goal of collaborative learning.

Table 3: MPI and Spark Comparison.

FRAMEWORK	LINES OF CODES(LOC)	BOILERPLATE LOC
MPI	52	28
Spark	28	2

The official Spark Programming Guide cited efficiency reasons as to why it does not support general read-write variables across tasks. This means that when a function is passed to Spark and run on a node in the cluster, all variables used in the function are copied to the remote nodes. And these copied variables will have no updates back to the driver programme.[1]

Apache Spark provides two types of shared variables, but with limited capacity: broadcast variables and accumulators. The broadcast variables cache a read-only variable on each machine, reducing the need to copy it with tasks. For example, broadcast variables can efficiently pass a copy of a large input dataset to every cluster node. They are distributed with Spark's efficient broadcast algorithms, reducing cost in communication.

Apache Spark operations are carried out through multiple stages, which are separated by distributed "shuffle" operations, for example, aggregating elements by key or grouping. Spark uses shuffle to redistribute data such that they are grouped differently across the partitions. The common data within each stage are broadcasted automatically by caching the serialized

data and de-serializing them before running the tasks.

For the accumulators, they are variables that can only be added by operations that are associative and commutative. This is useful to create counters or sums. Accumulators are used to provide a safe mechanism for updating variables, while their executions are parallelized across multiple nodes.

The initial project goal was that the slave (worker) nodes should be able to query one another in giving classification estimates, in effect, a collaborative approach to distributed machine learning, and that would require message passing. However, we realize that the communication protocol is an issue in using Spark to achieve distributed machine learning. From the sources we understand that, Apache Spark is radically different from Message Passing Interface (MPI), and has systematically excluded this possibility. There is no way to create a shared read-write array as flags with machine id as index, for example. For future research intending to use collaborative learning, we recommend to look beyond the Apache Spark architecture. Apache Spark is good in processing the big data, however the message passing capabilities exist outside of this framework.

5

CONCLUSION

As we have explored in the previous chapters, the project attempted to bring Spark realization of PBL-McRBFN algorithm using Python language. After understanding the algorithm and the laying the groundwork of single-thread realization in Python, there is a limited success in parallelizing the execution to speed-up the search for optimal parameters. The project is able to use the Spark parallelization to optimize the parameters for the learning on Image Segmentation. However, the initial goal of collaborative distributed learning could not be met, due to the lack of MPI in Spark framework.

For future research in collaborative distributed learning, it is recommended to try to:

- Integrate Spark with Message Passing Interface (MPI)
- Research into other newer Big Data technology
- Redesign the collaborative mechanism, make it compatible with Spark

Part III

APPENDIX

A

CODE LISTINGS

The following are Python code supplied to Jupyter Notebook.

A.1 PYTHON PBL-MCRBFN CORE FUNCTION

Code Listing 8: Python Source Code

```
1
2  import json
3  import operator
4
5  import numpy as np
6  from scipy.spatial.distance import pdist, squareform
7
8
9  def PBLMcRBF(UY, nxi, nyi, kmax, par):
10   # UY  - contains training data samples  (nx X muy)
11   # nxi - features used for training
12   # nyi - class labels of the training samples  (1 X muy)
13   # Initialize the network parameters
14   nx = len(nxi)    # number of input features
15   ny = max(nyi)    # number of outputs
16   nh = sum(kmax)   # maximum number of hidden neurons
17   mu = np.zeros((nx, nh), float)  # neuron centres
18   sig = np.zeros((1, nh), float)  # neuron widths
19   bet = np.zeros((nh, ny), float) # output weights
20   A = np.zeros((nh, nh), float)   # projection matrix
21   B = np.zeros((nh, ny), float)   # output matrix
22   K = -1  # initilized value, later corresponds to total
         number of neurons
23   w1 = np.zeros((1, nh), float)   # class association of
         neurons
24
25   # Initialize the control parameters
26   [Ed, Eadd, Elearn, r, kp, Pmax, P1, zeta, kp1] = par[:9]
27   # Ed - skip threshold
28   # Eadd - initial addition error threshold
29   # Elearn - initial learning error threshold
30   # r - decay factor
31   # kp - overlap factor
32   # Pmax - limit for reserve samples
33   # P1 - spherical potential threshold
34   # zeta - centre shifting factor
35   # kp1 - overlap factor for first neuron in each class
```

```
36
37    # Check the input data
38    muy = len(UY[0])
39    clab = np.array([[1 if x == y else -1 for x in range(ny)]
          for y in range(ny)])
40    flag = np.ones((muy), float)     # status of samples
41    kk = np.zeros((1, ny), float)    # neurons in each class
42    dflag = [0 for x in range(muy)]  # status of deleted
          samples
43    rflag = [x for x in range(muy)]  # sample reserve
44    uflag = []                       # used samples flags
45
46    # Find first samples in all class
47    F = np.zeros((nx, ny), float)
48    for i in range(ny):
49      L = nyi.index(i + 1)
50      F[:, i] = UY[:, L]
51    FF = squareform(pdist(np.array(F).transpose(), 'euclidean'
          ))
52    # FF[1:ny+1:ny*ny] = 1000
53    for i in range(ny):
54      FF[i][i] = 1000  # Assign large value nyi for diagonal
55    F = None
56    ME = 1  # Exit condition is 1
57
58    mu = np.array(mu)
59
60    # Training phase start
61    while ME and sum(flag) > Pmax:
62      pre_sample = sum(flag)  # total samples for learning
63      for rr in range(len(rflag)):
64        A, B, Eadd, Ed, Elearn, FF, K, P1, UY, bet, clab,
              dflag, flag, kk, kmax, kp, kp1, mu, ny, nyi, r,
              rflag, sig, uflag, w1, zeta = learning(A, B, Eadd,
               Ed, Elearn, FF, K, P1, UY, bet, clab, dflag, flag
              , kk, kmax, kp, kp1, mu, ny, nyi, r, rflag, sig,
              uflag, w1, zeta)
65
66      if sum(flag) == pre_sample:
67        ME = 0  # exit condition
68
69    mu = mu[:, 0:K + 1]
70    sig = sig[0, 0:K + 1]
71    bet = bet[0:K + 1, :]
72    return mu, sig, bet, A, B, K
73
74
75  def learning(A, B, Eadd, Ed, Elearn, FF, K, P1, UY, bet,
        clab, dflag, flag, kk, kmax, kp, kp1, mu, ny, nyi, r,
        rflag, sig, uflag, w1, zeta):
76    m = rflag[0]  # sample number
77    if flag[m] != 0:
78      x = UY[:, m]  # Current sample
79      yhat = np.zeros((ny), float)
80      if K >= 0:
81        xmusq = np.zeros((1, K + 1), float)
82        Phi = np.zeros((1, K + 1), float)
83        for i in range(0, K + 1):
84          xmusq[0, i] = np.dot((x - mu[:, i]).transpose(), (x
                - mu[:, i]))
```

```
85          Phi[0, i] = np.exp(- xmusq[0, i] / (sig[0, i] ** 2))
86        yhat = Phi[0, 0:K + 1].dot(bet[0:K + 1, :])  #
              Predicted output
87
88      # Calculate ccap and hinge error
89      cact = nyi[m] - 1  # Actual class label # original data
            has value range 1-7, thus reduce by 1
90      chat = np.argmax(yhat)  # Predicted class label
91      yact = clab[cact, :]  # Coded class label : a row or
            column vector
92
93      # Hinge error calculation.....
94      E = np.zeros((ny, 1), float)
95      for jj in range(ny):
96        # print(yact[jj] , yhat[0, jj], yact[jj] * yhat[0,jj])
97        if (yact[jj] * yhat[jj]) >= 1.0:
98          E[jj, 0] = 0
99        else:
100         E[jj, 0] = yact[jj] - yhat[jj]
101     err = (max(E[:, 0] ** 2)) ** .5  # Maximum Hinge Error
102
103     # Sample Deletion Criterion
104     if cact == chat and err < Ed:  # Delete the sample
105       flag[m] = 0
106       dflag[m] = 1
107       rflag.pop(0)
108       # continue
109       return A, B, Eadd, Ed, Elearn, FF, K, P1, UY, bet,
              clab, dflag, flag, kk, kmax, kp, kp1, mu, ny, nyi,
               r, rflag, sig, uflag, w1, zeta
110
111     # Sample Learning Strategy
112     if kk[0, cact] == 0:  # Zero neuron in the class
113       K += 1            # Increment the neuron
114       kk[0, cact] += 1  # Increment the neuron class
115       flag[m] = 0   # Sample used
116       rflag.pop(0)  # sample remove from reserve
117       uflag.append(m)   # Used samples
118       # Compute width factor
119       sK = max(0.0001, kp1 * (min(FF[cact, :])) ** .5)
120
121       if K == 0:    # First neuron
122         A[0, 0] = 1
123         B[0, :] = 1 * yact
124         bet[0, :] = B[0, :]
125         mu[:, K] = x     # Assign new centre
126         sig[0, K] = sK   # Assign new width
127         w1[0, K] = cact  # Associate class label
128       else:
129         mu[:, K] = x     # Assign new centre
130         sig[0, K] = sK   # Assign new width
131         w1[0, K] = cact  # Associate class label
132
133         # Update A and B Matrix #########note this is the
              same as the later part
134         A[0:K, 0:K] = A[0:K, 0:K] + Phi[0:K].transpose() *
              Phi[0:K]
135         B[0:K, :] = B[0:K, :] + Phi[0:K].transpose() * yact
136         B[K, :] = 0
137
```

```
138         # Past samples for new neurons
139         Phit = np.zeros((1, K + 1), float)
140         for i in range(len(uflag)):
141           for j in range(K + 1):
142             Phit[0, j] = np.exp(-sum((UY[:, uflag[i]] - mu
                    [:, j]) ** 2) / (sig[0, j] ** 2))
143           A[0:K + 1, K] = A[0:K + 1, K] + Phit[0, 0:K + 1].
                transpose() * Phit[0, K]
144           yacti = clab[nyi[uflag[i]] - 1, :]
145           B[K, :] = B[K, :] + Phit[0, K] * yacti
146
147         A[K, 0:K] = A[0:K, K]
148         Phit = None
149
150         # Find the output weight
151         bet[0:K + 1, :] = np.linalg.solve(A[0:K + 1, 0:K +
                1], B[0:K + 1, :])
152
153       Eadd = r * err + (1 - r) * Eadd  # Update
              selfregulating threshold
154     else:
155       # Novelty calculation
156       Ps = sum(Phi[0, w1[0, 0:K + 1] == cact]) / kk[0, cact]
157
158       # Find nearest neuron of the same class
159       L = np.nonzero(w1[0, 0:K + 1] == cact)[0]
160       nrS = []
161       nrI = []
162       if (L).any:
163         tem = np.min(xmusq[0, L])
164         nrS = np.argmin(xmusq[0, L])
165         nrS = L[nrS]  # neareast neuron in the same class
166         tnearS = tem ** .5
167
168       # Find nearest neuron in inter class
169       L = np.nonzero(w1[0, 0:K + 1] != cact)[0]
170       if (L).any:
171         tem = np.min(xmusq[0, L])
172         nrI = np.argmin(xmusq[0, L])
173         nrI = L[nrI]  # nearest neuron in the Inter class
174         tnearI = tem ** .5
175
176       # Neuron Addition strategy
177       if (cact != chat or err > Eadd or np.max(Phi[0, 0:K])
              < 1e-3) and Ps <= P1 and kk[0, cact] < kmax[
178         cact]:
179         P1 = r * Ps + (1 - r) * P1  # Update knowledge
              threshold
180         Eadd = r * err + (1 - r) * Eadd  # Update self-
              regulating threshold
181         K += 1  # Increment Neuron
182         kk[0, cact] += 1  # Increment associated neuron
              class
183         flag[m] = 0  # Sample used for neuron addition
184         rflag.pop(0)  # Remove the samples from reserve
185         uflag.append(m)  # Used samples
186         if kp * tnearS > 4 * sig[0, nrS] and kp * tnearI > 4
               * sig[0, nrI]:
187           center = x
```

```
188           sK = max(0.0001, kp1 * max((x.transpose() * x) **
                  0.5))
189         else:
190           if 1 < tnearS / tnearI:
191             center = x + zeta * (mu[:, nrS] - mu[:, nrI])
192             center = np.reshape(center, (1, len(center)))
193             sK = max(0.0001, kp * np.sum((center - mu[:, nrS
                    ]) ** 2) ** .5)
194           else:
195             center = x
196             sK = max(0.0001, np.max(kp * tnearS))
197         mu[:, K] = center  # Assign new center
198         sig[0, K] = sK  # Assign new width
199         w1[0, K] = cact  # Associate class label
200         # Update A and B Matrix #### this is the same as the
                previous part.
201         A[0:K, 0:K] = A[0:K, 0:K] + Phi[0:K].transpose() *
                Phi[0:K]
202         B[0:K, :] = B[0:K, :] + Phi[0:K].transpose() * yact
203         B[K, :] = 0
204
205         # Past samples for new neusons
206         Phit = np.zeros((1, K + 1), float)
207         for i in range(len(uflag)):
208           for j in range(K + 1):
209             Phit[0, j] = np.exp(-sum((UY[:, uflag[i]] - mu
                    [:, j]) ** 2) / (sig[0, j] ** 2))
210           A[0:K + 1, K] = A[0:K + 1, K] + Phit[0, 0:K + 1].
                transpose() * Phit[0, K]
211           yacti = clab[nyi[uflag[i]] - 1, :]
212           B[K, :] = B[K, :] + Phit[0, K] * yacti
213         A[K, 0:K] = A[0:K, K]
214         Phit = None
215         # Find the output weight
216         bet[0:K + 1, :] = np.linalg.solve(A[0:K + 1, 0:K +
                1], B[0:K + 1, :])
217
218       elif cact == chat and err > Elearn:  # Update Network
              Parameters        :
219         Elearn = r * err + (1 - r) * Elearn
220         A[0:K + 1, 0:K + 1] = A[0:K + 1, 0:K + 1] + Phi[0:K
                + 1].transpose() * Phi[0:K + 1]
221         B[0:K + 1, :] = B[0:K + 1, :] + Phi.transpose() *
                yact
222         bet[0:K + 1, :] = bet[0:K + 1, :] + np.linalg.solve(
                A[0:K + 1, 0:K + 1], Phi[0:K + 1].transpose() *
                E.transpose())
223         flag[m] = 0  # Sample used in learning
224         rflag.pop(0)
225         uflag.append(m)  # Sample used
226       else:  # Preserve Samples for future Learning
227         flag[m] = 1  # Sample is not used
228         rflag.append(rflag.pop(0))
229         # end of sample learning strategy
230   return A, B, Eadd, Ed, Elearn, FF, K, P1, UY, bet, clab,
          dflag, flag, kk, kmax, kp, kp1, mu, ny, nyi, r, rflag,
           sig, uflag, w1, zeta
```

A.2 COMMON FUNCTION FOR SPARK PARALLELIZATION

Code Listing 9: Common Function for Spark Parallelization

```
def common(par):
  with open("seg2_data.json", mode='r') as f:
    json_data = f.read()
    data = json.loads(json_data)

  uy1 = data['uy1']
  uy2 = data['uy2']
  UY1 = data['UY1']
  UY2 = data['UY2']

  UY = np.array(data['UY'])
  nyi = [np.argmax(UY[np.array(data['nyi'][0]) - 1, x]) + 1
      for x in range(UY.shape[1])]

  nxi = np.array(data['nxi'][0]) - 1
  UY = UY[nxi, :]
  nxi = nxi.tolist()

  kmax = data['kmax'][0]

  Mu, Sig, Beta, A, B, K = PBLMcRBF(UY, nxi, nyi, kmax, par)

  print("results:")
  print("mu", Mu, "\n", "sig", Sig, "\n", "beta", Beta, "\n"
      , "A", A, "\n", "B", B, "\n", "K", K, )
  print(Mu.shape, Sig.shape, Beta.shape, A.shape, B.shape)

  # Testing with same training data
  uy1 = np.array(uy1)
  uy2 = np.array(uy2)
  UY1 = np.array(UY1)
  UY2 = np.array(UY2)
  ny = max(nyi)  # No. of classes
  conf_tra = np.zeros((ny, ny), float)  # Confusion matrix
  # Find class label based on model.
  for m in range(uy1.shape[1]):
    x = uy1[nxi, m]
    Phi = np.zeros((1, K + 1), float)
    for i in range(K + 1):
      xmusq = np.dot((x - Mu[:, i]).transpose(), (x - Mu[:,
          i]))
      Phi[0, i] = np.exp(- xmusq / (Sig[i] ** 2))
    yhat = np.dot(Phi, Beta)  # Output
    chat = np.argmax(yhat[0, :])  # Estimated class label
    cact = uy2[0, m] - 1  # Actual class label
    # Confusion matrix update
    conf_tra[cact, chat] += 1

  # Performance matrix
  tra_ova = np.sum(np.diag(conf_tra)) / np.sum(conf_tra[:])
      # Overall
  tra_avg = np.mean(np.diag(conf_tra) / np.sum(conf_tra, 1))
       # Average
```

```
49
50  # Testing with larger data set
51  conf_tes = np.zeros((ny, ny), float)  # Confusion matrix
52
53  # Find class label based on model.
54  for m in range(UY1.shape[1]):
55    x = UY1[nxi, m]
56    Phi = np.zeros((1, K + 1), float)
57    for i in range(K + 1):
58      xmusq = np.dot((x - Mu[:, i]).transpose(), (x - Mu[:,
          i]))
59      Phi[0, i] = np.exp(- xmusq / (Sig[i] ** 2))
60    yhat = np.dot(Phi, Beta)  # Output
61    chat = np.argmax(yhat[0, :])  # Estimated class label
62    cact = UY2[0, m] - 1  # Actual class label
63    # Confusion matrix update
64    conf_tes[cact, chat] += 1
65
66  # Performance matrix
67  tes_ova = np.sum(np.diag(conf_tes)) / np.sum(conf_tes[:])
       # Overall
68  tes_avg = np.mean(np.diag(conf_tes) / np.sum(conf_tes, 1))
       # Average
69
70  # Display the result
71  print('Final Training/Testing Performance')
72  print(tra_ova, tra_avg, tes_ova, tes_avg)
73
74  return par[:5] + [tra_ova, tra_avg, tes_ova, tes_avg]
```

BIBLIOGRAPHY

[1] Apache-Software-Foundation. *Spark Programming Guide - Spark 2.0.1 Documentation*. 2016. URL: http://spark.apache.org/docs/latest/programming-guide.html (visited on 10/17/2016).

[2] Giduthuri Sateesh Babu and Sundaram Suresh. "Sequential Projection-Based Metacognitive Learning in a Radial Basis Function Network for Classification Problems." In: *IEEE Transactions on Neural Networks and Learning Systems* 24.2 (2013), pp. 194–206.

[3] John Canny. *BIDData/BIDMat*. 2016. URL: https://github.com/BIDData/BIDMat/wiki/Loading-and-saving-matrices (visited on 10/17/2016).

[4] Jose Maria Cavanillas, Edward Curry, and Wolfgang Wahlster, eds. *New Horizons for a Data-Driven Economy*. 31. Springer Science + Business Media, 2016, p. 31. ISBN: 978-3-319-21569-3. DOI: 10.1007/978-3-319-21569-3. URL: http://dx.doi.org/10.1007/978-3-319-21569-3.

[5] The Scipy Community. *scipy.io.loadmat - SciPy v0.18.1 Reference Guide*. 2016. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.loadmat.html (visited on 10/17/2016).

[6] Jose A Dianes. *Apache Spark & Python (pySpark) tutorials for Big Data Analysis and Machine Learning as IPython / Jupyter notebooks*. 2016. URL: https://github.com/jadianes/spark-py-notebooks (visited on 10/17/2016).

[7] Jonathan Dursi. "HPC is dying, and MPI is killing it." In: (2015). URL: http://www.dursi.ca/hpc-is-dying-and-mpi-is-killing-it/#whympiisthewrongtoolfortoday.

[8] Wojciech Gradkowski. *JMatIO - Matlab's MAT-file I/O in JAVA*. 2014. URL: https://sourceforge.net/projects/jmatio/ (visited on 10/17/2016).

[9] Donald E. Knuth. "Computer Programming as an Art." In: *Communications of the ACM* 17.12 (1974), pp. 667–673.

[10] M. Lichman. *UCI Machine Learning Repository*. 2013. URL: http://archive.ics.uci.edu/ml.

[11] Mathesaurus. *Numpy for MATLAB Users*. 2016. URL: http://mathesaurus.sourceforge.net/matlab-numpy.html (visited on 10/17/2016).

[12] Osvaldo T Mendoza. *Learn Matlab in Y Minutes*. LearnXInYMinutes.com. 2016. URL: https://learnxinyminutes.com/docs/matlab/.

[13] Thomas O. Nelson and Louis Naren. "Metamemory: A theoretical framework and new findings." In: *Psychology of learning and motivation* 26 (1990), pp. 125–173.

[14] Project-Jupyter. *Project Jupyter*. 2016. URL: http://jupyter.org/ (visited on 10/17/2016).

[15] Foster John Provost and Daniel N. Hennessy. "Scaling Up: Distributed Machine Learning with Cooperation." In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 1*. AAAI'96. Portland, Oregon: AAAI Press, 1996, pp. 74–79. ISBN: 0-262-51091-X. URL: http://dl.acm.org/citation.cfm?id=1892875.1892886.

[16] V. Rajaraman. "Big data analytics." In: *Reson* 21.8 (2016), 695–716. ISSN: 0973-712X. DOI: 10.1007/s12045-016-0376-7. URL: http://dx.doi.org/10.1007/s12045-016-0376-7.

[17] The-Scipy-Community. *Numpy for Matlab Users*. 2015. URL: http://scipy.github.io/old-wiki/pages/NumPy_for_Matlab_Users.html (visited on 10/17/2016).

[18] TutorialPoint. *Apache Spark Core Programming*. 2016. URL: https://www.tutorialspoint.com/apache_spark/apache_spark_core_programming.htm (visited on 10/17/2016).

DECLARATION

The content presented in this Report is written to the best knowledge of the author.

Singapore, October 2016

Yuan Yiyang