Joey Pinto

www.pintojoey.com

INCF GSoC Project #1

Dynamic GUI for designing and launching signal processing method workflows in a Hadoop Infrastructure

Mentor: Petr Ježek (University of West Bohemia, Czech Republic, Czech National INCF Node)

March 12, 2018

Overview

INCF has been working on using EEG event-related potential signal processing and machine learning methods for building assistive systems for motor impaired people. This system collects human brain data of audio/video stimulated subject. The data processed by customized classifiers is provided as a feedback leading to an action like turning on a TV, opening a window sunblind etc. The team has developed a client-server architecture to provide an interface to the procedures being applied onto the data stored on a distributed file system.

This project aims at building an easy to use graphical interface that can streamline the configuration of the parameters controlling individual signal processing sub-routines and thus make it easy to design complicated signal flows and execute them. The entire workflow created by the user will be exportable and reusable. The aim will also be to make the overall process more user-friendly. Secondary efforts will be targeted to make the tool user friendly and enable easy deployment of workflows on distributed computing frameworks.

Table of Contents

Project Synopsis

The Problem

INCF projects deal with large amounts of complex EEG data being processed through intricate workflows. The size of the data calls for using distributed computing that enables handling these large datasets efficiently. However, no solution exists for visually modeling these workflows.

A typical workflow consists of the following stages:

1. Data reading - File, Stream, Lab Streaming Layer
2. Filtration, ICA, Input transformation
3. Segmentation - Features extraction
4. Averaging
5. Wavelet Transformation, Matching Pursuit
6. Classification - Training, Testing
7. Storing results, classificator, etc.

Although this may not represent every application, most applications have some or all of these stages in a customized order. Each of these stages may have multiple configurable attributes. For example, a machine learning algorithm will have a training and a test dataset as outputs and a prediction or a model file as an output.
In addition to machine learning real-life algorithms require ample amounts of pre and post processing routines. These routines may be generic or specialized for the application.

Data scientists dealing with such research applications spend most of their time programmatically chaining already existing data processing snippets. They spend most of their time implementing highly generic workflows into code instead of actually working with the data.

My Solution

Typically a signal processing method is very well defined in terms of input and output parameters. The real complexity actually lies in the chaining of these methods logically and semantically.

The Business Process Execution Language(BPEL) in service oriented computing attempts to solve a similar problem of chaining already existing web services by using service definitions and a workflow graph.

I propose to solve the current problem using a similar approach. I aim to build a web-based workflow automation tool that allows users to define a data flow between well-defined data processing methods. This tool will utilize the definitions of the methods to create a workflow block by defining input/output criteria and the run time parameters.

Blocks: A block is a single data operation that has an input or a set of inputs of a specific type (Constant, User Input, File, Data Stream, Table etc.) Each block will perform an operation and produce an output or a set of outputs as defined by the user.

Once these blocks are defined in the code, they will be made available on the web interface. The user can then configure inputs and outputs to it from a different block(s) using channels.

Channels: A channel is essentially a graph edge connecting two blocks. The channel can be drawn between the output of one block and the input of another. Before a channel can be drawn, the tool will validate the compatibility of data types at both ends.

I plan to develop a Java Interface that existing data processing methods can implement. This interface will then be used to create block definitions. Users can then use the GUI to create a workflow from the input data to the expected result.

The Workflow design tool will have highly customizable type validators that will make sure that each edge is carrying the correct type of data from one block to another.

Implementation Specifications

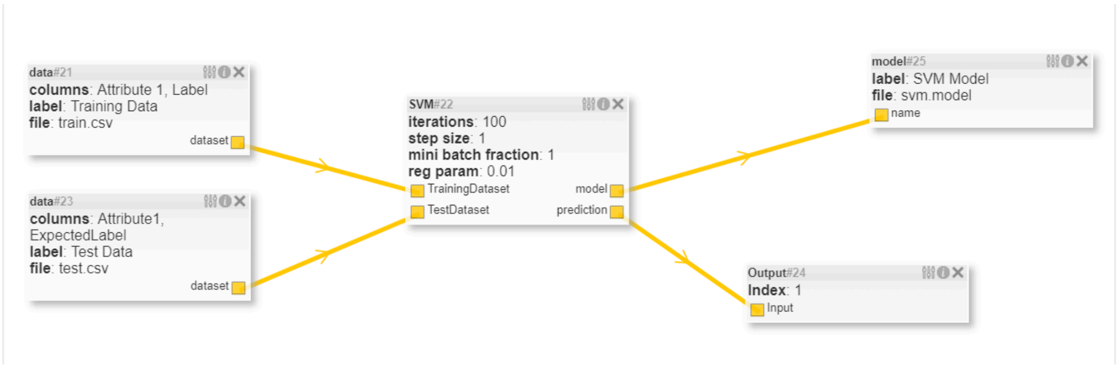
1. **Project:** will be built as a JAVA EE Web application.
   - This enables maximum code reuse and perfect compatibility end-to-end
2. **Back-end:** for logic validation and workflow design will be a JAVA WX-RS REST API
   - This enables further front-end development on different platforms.
3. **Front-end:** will be built using HTML, CSS, JavaScript, Bootstrap built using an Admin Panel Template
   - This will speed up development and will encourage further contributors to opt for any framework without restrictions
4. **Data Representation:** To enable easy integration an interconvertible JSON/XML data export mechanism will be provided so that API users can opt for any method they prefer.
5. **Miscellaneous:**
   - Maven: for dependencies and project management
   - Github: for Version Control
   - JUnit: for Testing
   - MVC: As a Design Pattern

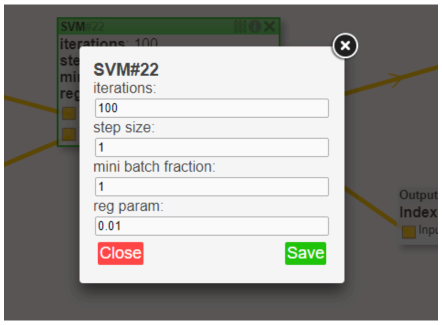
Prototype Implementation

You can find a video demo of the prototype I have built here:-

https://drive.google.com/file/d/1rvYc1R9KdraQgWe_uwucJ-3lqdOQUPI1/view?usp=sharing

Implementation of a very basic SVM train/test flow and corresponding JSON Representation:





The flow represents 4 kinds of entities

1. Data: an input vector represented as a file with defined headers
2. SVM: A simple SVM classifier implementation with modifiable attributes, 2 inputs and 2 outputs.
3. Model: An SVM training model stored in a file
4. Output: Indicates the overall output of the system

Complex signal processing routines may be represented in a similar flow pattern by having multiple blocks with defined inputs and outputs. The prototype implementation in javascript can validate input types, annotate edges and provides a data structure that can easily be used to add custom conditions or input validation techniques.

JSON Data Structure:

The following data structure is the equivalent of the diagram displayed above:

```
"edges": [{
                    "id": 1,
                    "block1": 21,
                    "connector1": ["dataset", "output"],
                    "block2": 22,
                    "connector2": ["trainingdataset", "input"]
          }, {
                    "id": 2,
                    "block1": 23,
                    "connector1": ["dataset", "output"],
                    "block2": 22,
                    "connector2": ["testdataset", "input"]
          }, {
                    "id": 3,
                    "block1": 22,
                    "connector1": ["prediction", "output"],
                    "block2": 24,
                    "connector2": ["input", "input"]
          }, {
                    "id": 4,
                    "block1": 22,
                    "connector1": ["model", "output"],
                    "block2": 25,
                    "connector2": ["name", "input"]
          }
     ],
```

```
"blocks": [     {         "id": 23,
                          "x": -11.053299945530682,
                          "y": -6.500860368543044,
                          "type": "data",
                          "module": null,
                          "values": {
                                    "columns": ["Attribute1", "ExpectedLabel"],
                                    "label": "Test Data",
                                    "file": "test.csv"
                          }
                }, {      "id": 21,
                          "x": -11.403289595926879,
                          "y": -95.37457669627784,
                          "type": "data",
                          "module": null,
                          "values": {
                                    "columns": ["Attribute 1", "Label"],
                                    "label": "Training Data",
                                    "file": "train.csv"
                          }
                }, {      "id": 22,
                          "x": 249.01614489960977,
                          "y": -73.2976887928691,
                          "type": "SVM",
                          "module": null,
                          "values": {
                                    "iterations": 100,
                                    "step size": 1,
                                    "mini batch fraction": 1,
                                    "reg param": 0.01
                          }
                }, {      "id": 24,
                          "x": 482.33522264498987,
                          "y": 42.31297083724107,
                          "type": "Output",
                          "module": null,
                          "values": {
                                    "index": "1"
                          }
                }, {      "id": 25,
                          "x": 567.5156905932665,
                          "y": -102.85041979423379,
                          "type": "model",
                          "module": null,
                          "values": {
                                    "label": "SVM Model",
                                    "file": "svm.model"
                          }
                }]};
```

Autogenerated equivalent XML structure:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<edges>
        <id>1</id>
        <block1>21</block1>
        <connector1>dataset</connector1>
        <connector1>output</connector1>
        <block2>22</block2>
        <connector2>trainingdataset</connector2>
        <connector2>input</connector2>
</edges>
<edges>
        <id>2</id>
        <block1>23</block1>
        <connector1>dataset</connector1>
        <connector1>output</connector1>
        <block2>22</block2>
        <connector2>testdataset</connector2>
        <connector2>input</connector2>
</edges>
<edges>
        <id>3</id>
        <block1>22</block1>
        <connector1>prediction</connector1>
        <connector1>output</connector1>
        <block2>24</block2>
        <connector2>input</connector2>
        <connector2>input</connector2>
</edges>
<edges>
        <id>4</id>
        <block1>22</block1>
        <connector1>model</connector1>
        <connector1>output</connector1>
        <block2>25</block2>
        <connector2>name</connector2>
        <connector2>input</connector2>
</edges>
```

```xml
<blocks>
        <id>23</id>
        <x>-11.053299945530682</x>
        <y>-6.500860368543044</y>
        <type>data</type>
        <module />
        <values>
                <columns>Attribute1</columns>
                <columns>ExpectedLabel</columns>
                <label>Test Data</label>
                <file>test.csv</file>
        </values>
</blocks>

<blocks>
        <id>21</id>
        <x>-11.403289595926879</x>
        <y>-95.37457669627784</y>
        <type>data</type>
        <module />
        <values>
                <columns>Attribute 1</columns>
                <columns>Label</columns>
                <label>Training Data</label>
                <file>train.csv</file>
        </values>
</blocks>

<blocks>
        <id>22</id>
        <x>249.01614489960977</x>
        <y>-73.2976887928691</y>
        <type>SVM</type>
        <module />
        <values>
                <iterations>100</iterations>
                <step size>1</step size>
                <mini batch fraction>1</mini batch fraction>
                <reg param>0.01</reg param>
        </values>
```

```
</blocks>
<blocks>
        <id>24</id>
        <x>482.33522264498987</x>
        <y>42.31297083724107</y>
        <type>Output</type>
        <module />
        <values>
                <index>1</index>
        </values>
</blocks>
<blocks>
        <id>25</id>
        <x>567.5156905932665</x>
        <y>-102.85041979423379</y>
        <type>model</type>
        <module />
        <values>
                <label>SVM Model</label>
                <file>svm.model</file>
        </values>
</blocks>
```

Implementation Details

1. Signal Processing method Management

   This screen will list all signal processing methods defined in the project gathered using reflection to explore all method parameters. This screen will enable mapping method parameters to custom parameter names and adding validations to it.

2. Module for WorkFlow definition

   a. Flow configuration: The flow will be configured in a graph layout so that multiple methods can be chained combined and distributed. An add button at the end of the graph will be used to list out available acceptable methods at that position in the graph. Interesting recommendations and validations can be deployed by parsing the current state of the graph
   b. Method configuration: The parameters defined in (1.) will be used to dynamically generate a form to edit and customize parameters of each method.
   c. Flow interpretation and execution: The server module of the already existing work by INCF will be called and the job will be run on the associated Hadoop cluster.

3. Additional Module for signal data visualization

   a. Since INCF deals with EEG signals, intermediate visualizations after major signal processing actions may be an important interface to provide to the user.

4. Module for Job Execution and Management

   This is an auxiliary module to help run the tasks on the dataset directly from the workflow definition screen. The existing job execution screen is not very intuitive to use. The current project or any forthcoming contributors could focus on having a more user-friendly UI for task execution. A simple PLAY button on the workflow screen leading to the task being executed will probably the best experience possible for the users.

Inspiration

1. Total Flow.js, Flowhub
    a. This a flow-oriented programming interface that can be required using a REST api to solve the current problem.
    b. The framework in itself is powerful enough to do basic rule based validations and connections.
2. blocks.js
    a. This is a great starting place for the visualization if we plan to distribute logic between the client and server equally, which is a good way to go
3. Simulink
    a. Simulink is one of the most comprehensive workflow definition toolkits available in the market right now. Its logic of defining flows that are combinatorial and distributable may be a good source of inspiration for the workflow development logic
4. BPEL
    a. BPEL for Java is an interesting web service logic flow definition framework for SOAP based web services. The following concepts may be useful
        i. FLOW->for parallel chains
        ii. SEQUENCE-> for sequential operations
        iii. INVOKE-> To call a remote web service (In our case a Signal Processing Method)

iv. ASSIGN-> Takes input from a previous invocation and maps it out to another invocation (In our case an intermediate between two operations)

- v. PARTNER_LINKS-> To define vendors (A method entity)
- vi. PORT-> A single service/operation of a PARTNER_LINK

b. BPEL uses an XML based format to represent this logic flow which is highly analogous to the current problem statement.

c. There may be a direct way to reuse BPEL frameworks to map out the project however a custom solution on the lines of BPEL may be a better alternative not adding an additional technology as a prerequisite for future contributors.

Minimal Deliverables

1. Mapping methods to workflow blocks

The current work available on github provides snippets for 5 different classifiers. All of these override the IClass interface and have setFeatureExtraction, train, test, save and load methods. However each of the classifiers have their own set of configuration parameters which we will need to represent on blocks.

The strategy will be to create an interface that all signal processing methods can implement to add themselves as available blocks in the workflow designer. The interface will extract information such as input-output mappings as per operation mode (eg. Train mode an test mode will have different configurations). Configuration parameters for each method will be forwarded to the block metadata so that they are editable at runtime.

2. Dynamic workflow designer tool

Although this may seem to be the most challenging module, the proof of concept implementation already made accomplishes the tricky bits already. Once the blocks are available, we should be able to map out the flows between blocks. Creating a flow between blocks requires validation of data types. This too has already been implemented with primitive data types in the prototype implementation.

3. UI to JSON/XML Mapping

The prototype already has a provision for exporting a flow to JSON and importing a JSON file as a workflow. This JSON can also be converted to XML with ease.

4. Workflow Execution Routines

Once the workflow designer is ready, we need to create a snippet to generate the code to execute the workflow as a spark job. The JSON needs to be parsed and then the workflow graph should be traversed in a way that the entire workflow is executed. This should be rather easy as INCF’s EEG server implementation already has a REST API for job execution.

Optional Features

The development of the following functionalities will subject to available time:

1. Spark Job Tracker (Simple job listing)
2. HDFS browser (File manager)
3. Workflow debugger (Pause/Resume execution)
4. Intermediate data collection (Visualize data in-process)
5. Workflow collaboration (Let multiple users work on the same workflow)

Work/Communication Strategy

1. The project mentor, Petr, will be in New York during the duration of the project and I will be residing in India. There is an 11 hour time zone difference between the two places. The advantage of this means that I will be able to work in daylight and at the end of the day report everything that was accomplished in a daily standup format to Petr as soon as he reaches his New York office in the morning. I had also offered to work according to New York Time Zone however we agreed that working in natural day light is always more productive.
2. All daily progress will be reported via email and a hangouts call if found necessary. At the end of each week we will have a mandatory video call discussing on a 4 slide stack
   a. What was planned
   b. What was done
   c. What wasn't done (Any blockers)
   d. What will be done next week
3. A Trello Board will be used to visualize the task backlog, work in progress, completed tasks
4. Github issue tracker will be used for tracking problem resolution

Timeline

Week	Duration	Task
Week 1	May 14th-May 19th	Complete Javascript Implementation for Dynamic Workflow generation
Week 2,3	May 21st-Jun 1st	Create script to auto generate JSON structure for blocks using method definitions
Week 4	Jun 4th-Jun 8th	JSON/XML Workflow export
Week 5	Jun 11th-Jun 15th	Phase 1: Testing and Evaluation
Week 6	Jun 18th-Jun 22nd	Parse Workflow JSON and execute on cluster
Week 7	Jul 2nd-Jul 6th	Visualize output data
Week 8	Jul 9th-Jul13th	Visualize intermediate data
Week 9	Jul 16th-Jul 20th	Phase 2: Testing and Evaluation
Week 10	Jul 23rd-Jul 27th	Develop APIs for Workflow deployment
Week 11	July 30th-Aug 3rd	Override existing EEG Server project to have API based Spark JOB Manager
Week 12	Aug 6th- Aug 10th	Final Testing and Visual Improvements.

About me

Overview

JOEY PINTO

STUDENT OF MASTERS IN SOFTWARE ENGINEERING

CARNEGIE MELLON UNIVERSITY, SILICON VALLEY

- 3 Years of experience in Java EE development
- 3 Publications in Hadoop, Distributed Systems, Machine Learning and Computer Vision (+1 under review)
- Interests: Software Architecture Design, Cloud Computing, Web Services, Android Development, Machine Learning, Distributed Computing
- 3 Visual toolkits developed for System Automation and data visualization at Indian Defence Research.
- Developed an automated hadoop cluster setup and monitoring toolkit in Java for an academic project.
- Currently working on building a graph representation of traffic flows in the city of Palo Alto for real-time traffic flow analysis
- Complete profile at http://www.zedacross.com
- Professional Network: https://www.linkedin.com/in/pintojoey

Motivation

I've taken a course on signal processing and machine learning during my undergrad. I remember what a pain it was to chain machine learning methods and signal transform operators to get basic tasks done. If it wasn't for OOP such tasks would take a lifetime.

The problem statement for this project tries to solve this exact tribulation. I genuinely think combining distributed computing concepts will visual programming will let data scientists focus more on the experimental results and less on the implementation.

A workflow graph can essentially represent the entire procedure a data scientist wants to perform on a set of data. Enabling them to define their workflows visually without basically writing a single line of code will encourage them to try more complex approaches to data flows and still do them on a large scale.

INCF has a vision to improve the lives of motor impaired people by giving them a chance to interact with the natural world using their brain-signals. More than anything, working for such a noble cause would indeed be an honour for me.

Match

For the last 3 months, I've been working on building a Graph for visualizing traffic flows in Palo Alto city. Previously I had used graphs to maintain citation histories of publications. I've grown to like graphs a lot because I think that a graph is the only data structure that can represent correlation of information in the real word fairly accurately and probably that's why our brains use it too in the form of neuron-networks. The fact that graphs can represent any kind of data flow be it a metric like traffic count or an operation like a Fourier transform gets me really excited.

Before I came to the US to do my masters I was working with the scientists at Defence Research and Development Organization, India as a System Automation Engineer. There I worked on creating an automated tool for performing complex command controls on high speed signal data acquisition equipment. The tool could perform an experiment without any human intervention and perform statistical operations and store them to a file. I also used MATLAB to visualize the results of Machine Learning algorithms.

Every time an experiment specification changed I had to add newer modules and re-deploy the project. Understanding the problem I realized that it was necessary to develop a data processing workflow automation tool. About 5 months ago I had created a proposal to develop such a tool and many scientists were interested in it. However, I left the institution to pursue higher studies and the idea could not be realized.

This project from INCF is my opportunity to build what I was planning. A lot of projects today are experimenting with visual programming. However, most of these projects are limited to the scope of javascript. This will be one of the first web-based visual programming toolkits for Java.

Is this the only project that I will apply for?

Yes! I have taken careful thought and this seems to be a project that best fits both my career goals and desire to learn more about distributed computing applications of signal processing.

Working time and commitments

I do not have any other plans for the summer and would like to dedicate my whole hearted effort towards this project. I will follow the standard working hours in India 9 am - 5 pm (IST) from Monday to Friday. I am open to use Saturday as a day for weekly retrospectives if the mentor wishes so. I do not prefer working on Sundays as I reserve it for being creative, playing music, writing poems, going biking etc. Such recreational activities rejuvenate me and often help me overcome blockers.

Mission

I hope I get a chance to join hands with the INCF in making a difference to the lives of motor impaired people and give them a genuine chance to interact with the natural environment like the luckier of us. I believe this project will go a long way in supporting the organization’s initiative.

Past Conversations with the Mentor

My initial communication with Petr Jezek from the INCF foundation was as follows. These are the questions I posed before him and the respective answers I received from the mentor.

1. Should data preprocessing/postprocessing techniques be added to the workflow block specifications?

Answer: Yes

My Response: Great, so we may need to define a common interface for the pre-processing techniques like interval selection etc.

2. Should the scope include non-generic user-defined validations for input parameters? Example: Should it be possible to define custom validations, for example should it be possible to limit the product of number of layers and number of neurons per layer?

Answer: Yes

My Response: While this does increase the complexity, this visual programming feature will be a great thing to add and I am really excited to implement this.

3. Do the requirements dictate a local server implementation, a centralized cloud service or both?

Answer: There are a hadoop server and a REST API servers installed.

My Response: It seems we are leaning to follow the server-client structure implemented in the work previously done in the project.

4. If local, is database connectivity for storing workflows on the server required? If cloud based, database is essential, is user access control in the scope?

Answer: I suppose representing of workflows in JSON documents. I think those could be stored on HDFS. We do not want to introduce a document storage database to the system.

My Response: Agreed, but storing workflow documents on the HDFS may not always be very convenient for users, especially if the document is really small. But yes, we can definitely make a provision for this while also allowing workflows to be exported and stored from the local file system.

5. What are the workflow operations that would be nice to handle?

Answer: Here you can see individual workflow steps what a user can perform. I have discussed it with my colleague. The order of workflow steps kind of follows the order of the list but it should be flexible in a general workflow anyway. The number of inputs and outputs should be flexible as well.

1. Data reading - File, Stream, Lab Streaming Layer
2. Filtration, ICA, Input transformation
3. Segmentation - Features extraction
4. Averaging
5. Wavelet Transformation, Matching Pursuit
6. Classification - Training, Testing
7. Storing results, classificator, etc.

My Response: Thank you so much. This list looks like a great starting point. Yes, my current implementation does take care of reordering of blocks and multiple inputs. I will add these to my implementation. Ofcourse, the user will be able to define his own operations too eventually.

6. The existing implementation of the client side code for the existing server isn't very usable and seems to be like a makeshift implementation. Would it be useful to add the HDFS browser and job tracking functionalities to the workflow designer itself?

Answer: Yes, I agree the client side java tool isn't very usable. It would be fine to reimplement the client side code, but the server side API implementation should remain largely unchanged.

My Response: That's great, this completes the loop for the workflow designer and really makes the project even more interesting. Yeah, I don't think it may be necessary to change the server implementation much if not at all.