Cloud Based Infrastructure for Educational Deep Learning

A Major Qualifying Project Report

Submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Nicholas A. Diaz

Cameron T. Jones

Approved by:

Professor Neil T. Heffernan

Worcester Polytechnic Institute

Computer Science

Abstract

The aim of this project was to introduce deep learning functionality to the ASSISTments testbed. State of the art deep learning techniques require more advanced infrastructure, and ASSISTments has realized the need for such infrastructure to support these new methods of analysis for researchers. Previously researchers were able to get statistical analysis of their educational experiments through The Assessment for Learning Infrastructure (ALI), but were unable to take advantage of more advanced techniques. To correct this, our project implements a new workflow to streamline and strengthen the report generation process. We use a Python API hosted on Amazon cloud machines to achieve this goal, creating a powerful, yet simple and scalable infrastructure for future deep learning tasks.

Acknowledgements

We would like to thank Professor Neil Heffernan, Anthony Botelho, and Siyuan Zhao for their guidance and support throughout this project.

Contents

Table of Figures

Table of Tables

Introduction

The ability to analyze data to learn more about a topic is essential to any scientific pursuit. In recent years, as the scale of datasets has increased, data analysis has become a more complicated task. It has also become a more lucrative one, with potential for new insights that older methods could never detect. Machine learning techniques allow computers to process vast datasets and return human readable results that can then be used to improve all manner of fields – including education.

ASSISTments, an online skill building and teaching tool, collects educational data from its users to better serve their needs, seeking to improve and make more accessible education at all levels. As part of this goal, ASSISTments makes its data available to researchers across many disciplines, and allows those researchers to run experiments on the user base (Roschelle, Feng, Murphy, & Mason, 2016). After their experiment has been run, researchers receive a report detailing the results. This report currently includes several types of statistical analysis alongside the raw data, but ASSISTments is interested in potentially expanding this return with further machine learning based analysis.

In order to facilitate these new techniques, a new system needs to be constructed. This new system has several requirements – it needs to be powerful enough to run complex machine learning algorithms on large amounts of data, it needs to be flexible enough to be easily modified and added to in the future, it needs to connect seamlessly with the existing ASSISTments infrastructure, and it needs to be cost-effective. This project seeks to create a system that meets all of these needs, by taking advantage of Python based APIs and scalable cloud services.

Background

Assessment of Learning Infrastructure

Assessment of Learning Infrastructure (ALI) is a tool within the ASSISTments testbed that analyzes and returns experimental data to researchers. Researchers are able to develop custom experiments to be run through ASSISTments, and once those experiments are concluded, ALI gathers the data and builds a report to send back. This report includes details on completion rates, a bias assessment using a chi-squared test, details on the mean and standard deviation of the posttest score, and links to all the raw data gathered in the experiment ("Analyze Data", 2016).

On the back end, ALI is a program built in Java, using an R server for statistical callouts. Our project aims to add new functionality to ALI, and new items to its reports, by integrating deep learning analysis. This functionality will be used to improve extensibility and provide a cost effective way of scaling up the service, to enable quick response times and future features.

Deep Learning

“Deep learning” is a term that refers to advanced machine learning techniques that have risen to prominence in recent years. Most commonly, deep learning refers to complicated neural networks. Neural networks have been a long-standing subject of research in computer science, but have become much more viable for use recently, as a result of algorithmic improvements and advances in computing power. Famously, the Google DeepMind team has been able to develop a network capable of besting human Go champions (Silver, Huang, & Maddison, 2016). Go is an ancient game with complexity such that it cannot be solved by brute force – it is infeasible for an artificial intelligence to be able to calculate a perfectly optimal move, due to the sheer scale of possibilities that exist on any given turn. The network constructed by the DeepMind team (AlphaGo) circumvents this by using deep convolutional neural networks to abstract the game board into a more manageable problem space (Silver, Huang, &

Maddison, 2016). AlphaGo has proven itself very effective, winning a Go match against 18-time world Go champion Lee Sedol four to one (Metz, 2016).

The potential applications for deep learning techniques are numerous, and go well beyond playing board games. Within ASSISTments, there is great potential for deep learning to be used to perform advanced classification tasks, that will allow teachers to better identify the needs of their students. Work is already being done to implement a deep affect detector, a recurrent neural network (RNN) capable of classifying students according to their emotional state. The model is able to classify students in to four groups (bored, confused, concentrating, and frustrated) without using any additional sensors beyond the student logs already being collected by ASSISTments. This allows the platform to be more effectively tuned to individual student needs, and has resulted in significant ROC/AUC improvements in testing, showing that the model fits the data well (Botelho, Baker, & Heffernan, 2016).

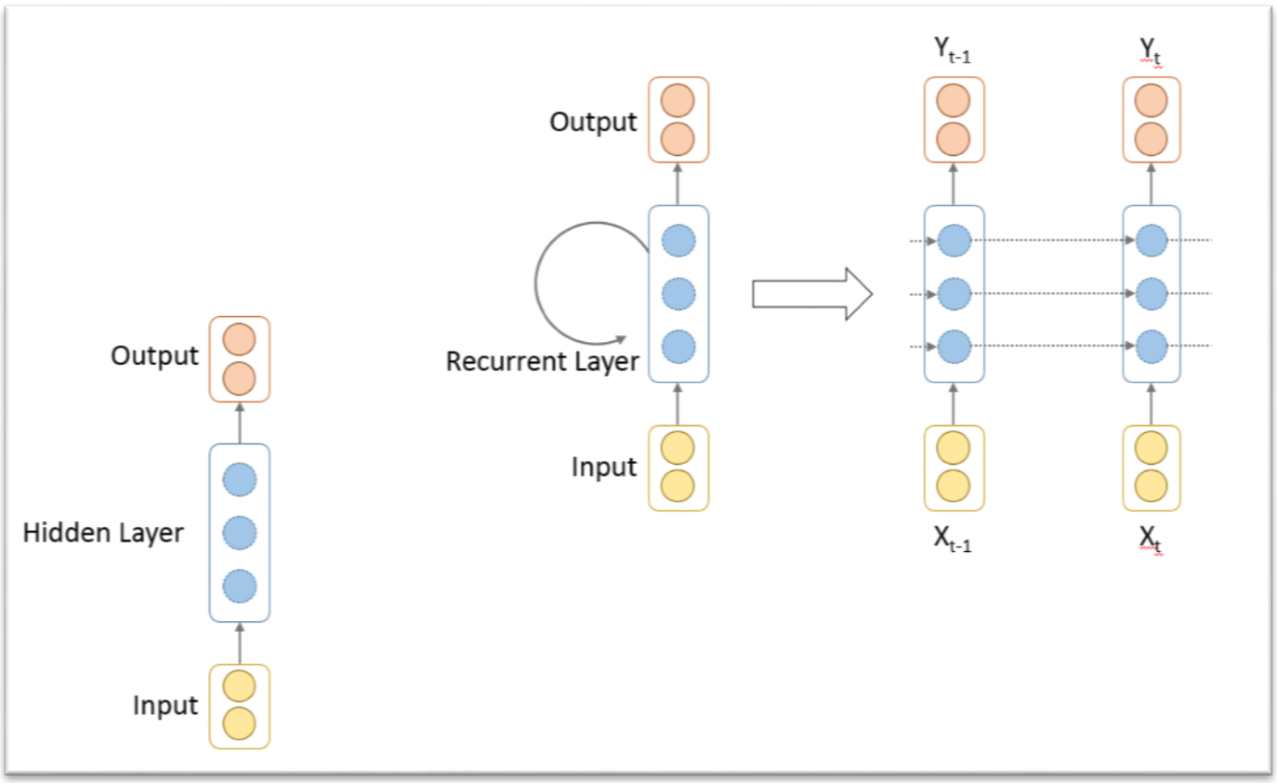


***Figure 1**: Recurrent Neural Network Model*

Deep Knowledge Tracing (DKT) is also a strong application for neural networks in education. DKT allows for student learning to be modeled and abstracted, just as AlphaGo models a Go board. Once a student's knowledge is modeled, predictions can be made based on the model, allowing the system to provide resources optimized for an individual student's needs (Piech, et al., 2015). These techniques have the potential to be very useful to educational systems, however they can be very computationally expensive to run. AlphaGo for example, runs 40 simultaneous search threads on a total of 48 CPUs and 8 GPUs. A distributed version of AlphaGo runs the same number of threads on multiple machines, using 1,202 CPUs and 176 GPUs (Silver, Huang, & Maddison, 2016). For the ASSISTments team to begin exploring complicated deep learning techniques, they will require a reliable and affordable infrastructure to run them on.

Cloud Computing

Cloud computing refers to the use of off-site hardware to run operations for a system. Rather than running a task on a local machine, instructions are sent over the internet to one or more distant machines that handle the task and then return the result. Cloud computing allows for greater flexibility than traditional approaches, as the hardware can all be maintained remotely. Services such as Amazon Web Services (AWS) allow small teams to take advantage of powerful hardware that they may not have had the capital or infrastructure to gain access to otherwise. Cloud computing is generally run as a timesharing service, so users pay to use a machine based on how long they use it, rather than a flat fee (Amazon Web Services, "What is Cloud Computing", 2017).

For our project, we chose to use AWS to host our new API. AWS is a full-featured, global scale cloud computing service offered by Amazon. AWS offers many useful options and features to its users, beyond just the ability to run programs in the cloud. Programs can be run on a wide variety of machines, including GPU-equipped machines that can excel at training deep learning models. The service also allows users to save and load machine images, making it simple to get a machine up and running with whatever configuration options and installed libraries are needed. Once running, AWS also allows us to build multiple reports in parallel, on different machines, a capability that ALI currently lacks due to its R implementation. Cost-wise, AWS is able to offer very competitive, affordable prices as a result of their massive scale, making them a very attractive choice for ASSISTments (Amazon Web Services, "What is Cloud Computing", 2017).

Implementation

Overview

Before this project, the workflow for both the external researchers, and the graduate students in the lab could be tedious, and lengthy, depending on what types of requests were made. For example, a simple request to ASSISTments could be completed round trip in about fifteen minutes, and would provide the researcher with statistical insight into the outcome of their experiment **Figure 2**. However, if the researcher wanted any sort of deep learning applied to the same experiment, it would require a hands-on approach, where a graduate student would need to manually fetch the data and run the requested model. The results would be transcribed into a summary, and emailed back to the researcher (as in **Figure 3**), which could take several days of back and forth communication. Our project speeds up this process considerably, by allowing for the automation of deep learning tasks.

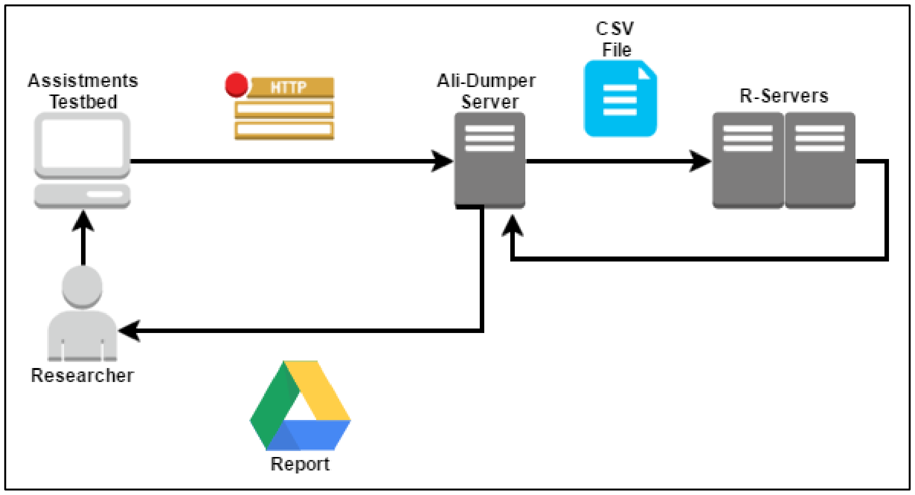


***Figure 2**: Simple Request Workflow*

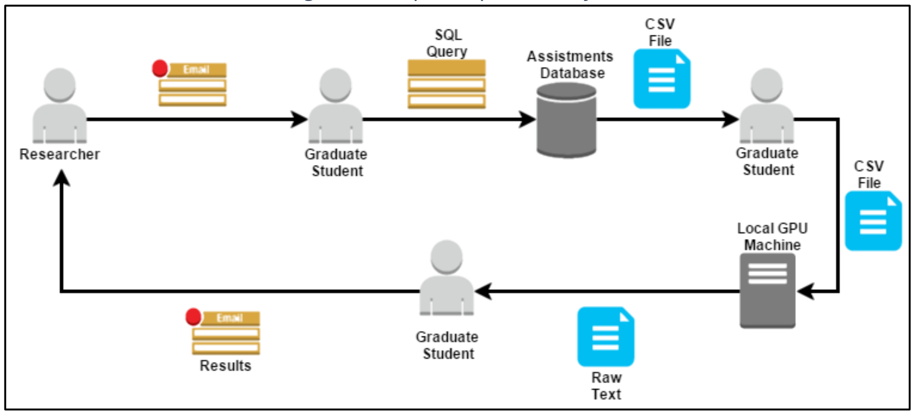


***Figure 3**: Deep Learning Request Workflow*

Our completed project is an API written in Python, hosted on AWS, that interfaces with a Java application. The API is able to handle the training and testing of deep learning models, as well as several statistical functions; and it is able to be launched at will from the Java application, to avoid unnecessary uptime. Each area of the project had several design considerations and implementation details, which will be further explained in their specific sections.

Our new workflow simplifies the process of applying deep learning techniques, and eliminates the need for the manual intervention the graduate students previously had to do. Instead, the researcher can now submit a single request, specify which deep learning models to run, and what reports they would like to generate, and the ASSISTments platform will take care of the rest. A model of the workflow can be seen below in **Figure 4**.

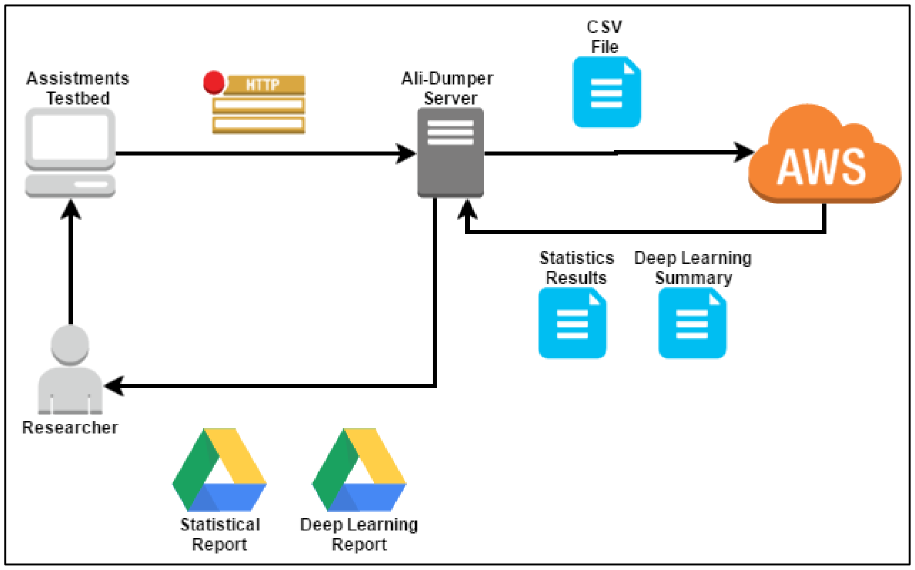
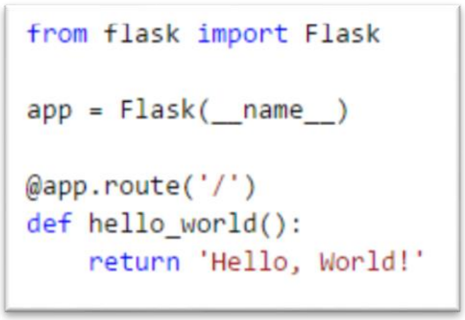


***Figure 4**: Improved Workflow*

Python API

Our API is built in Python, using the Flask microframework. Python was a clear choice for language due to the deep learning requirements of the project. Python has multiple established libraries for deep learning, and at the time of this project two of them (Tensorflow and Theano) were already being used by members of the ASSISTments team. Additionally, Python is a very readable language that will allow our infrastructure to be easily understood and maintained in the future.

Flask was chosen as our web framework for similar reasons. Flask is a microframework, designed with the intention of being as simple as possible, but retaining strong extensibility when more complex capabilities are needed. Our networking requirements for this project are primarily just basic input and output, and Flask allows for this to be done very quickly and easily, without a lot of dependencies or intricacies to worry about. Figure 5 illustrates a “Hello World” program in Flask, fully implemented in only a few lines:

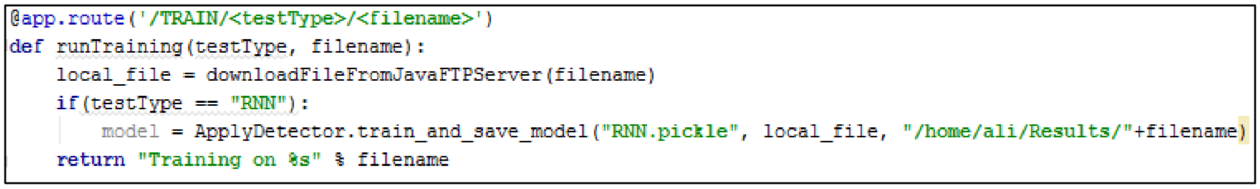


```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'
```

***Figure 5**: "Hello World" in Flask*

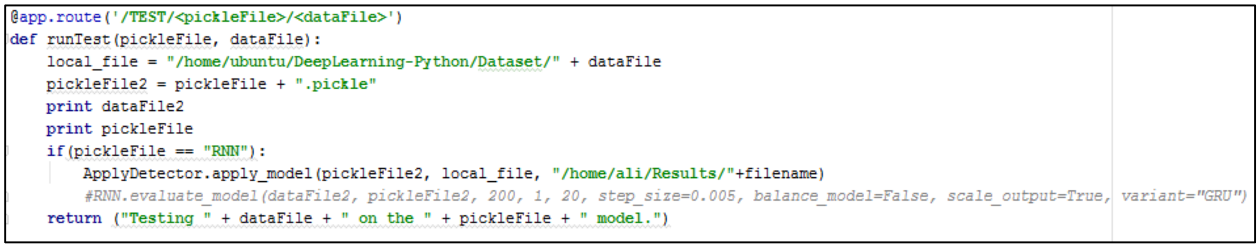
When this Flask server is launched, navigating to its root directory (“/”) will display the text ‘Hello, World!’ The route is very easily changed to allow for different functions to be called. Our project implements three different primary routes, “TRAIN” for training new models, “TEST” for running models, and “MATH” for using the statistics functions. Variables supplement these primary routes to specify more detailed operations, i.e. “TEST/model1/studentData2.csv” to run an existing model named “model1” on a provided CSV file.

As shown below, the API is easily understood and extensible if future features are desired. If a new model is added, there are two steps to update the Flask server. First, the “TRAIN” path needs to be updated with an additional testType identifier, and a call to the function used to train the model as seen below in **Figure 6**. With those in place, a similar procedure is put in place for the “TEST” path. An additional condition is put in place, with the same string value as the testType from the “TRAIN” path as in **Figure 7**. Then calls to the model’s test method, passing in the data filename, and that function takes care of loading the saved, previously trained model, and saves the result to a local directory with the same name. Then the function returns an HTTP body which signals to ALI that the data is ready.



```
@app.route('/TRAIN/<testType>/<filename>')
def runTraining(testType, filename):
    local_file = downloadFileFromJavaFTPServer(filename)
    if(testType == "RNN"):
        model = ApplyDetector.train_and_save_model("RNN.pickle", local_file, "/home/ali/Results/"+filename)
    return "Training on %s" % filename
```

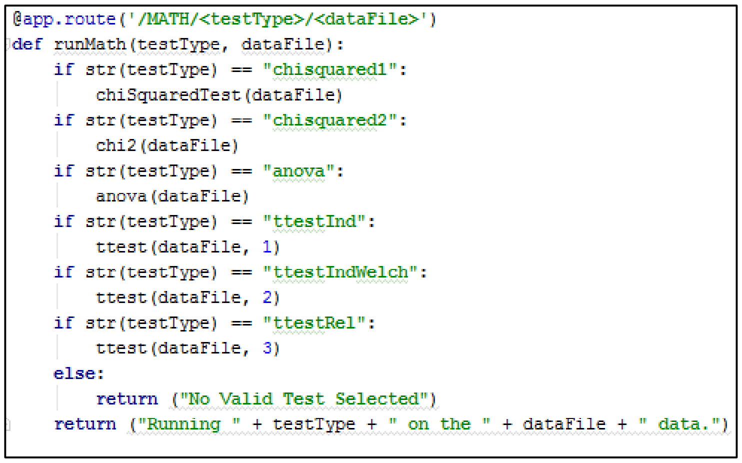
***Figure 6**: Flask Training Code*



```
@app.route('/TEST/<pickleFile>/<dataFile>')
def runTest(pickleFile, dataFile):
    local_file = "/home/ubuntu/DeepLearning-Python/Dataset/" + dataFile
    pickleFile2 = pickleFile + ".pickle"
    print dataFile2
    print pickleFile
    if(pickleFile == "RNN"):
        ApplyDetector.apply_model(pickleFile2, local_file, "/home/ali/Results/"+filename)
        #RNN.evaluate_model(dataFile2, pickleFile2, 200, 1, 20, step_size=0.005, balance_model=False, scale_output=True, variant="GRU")
    return ("Testing " + dataFile + " on the " + pickleFile + " model.")
```

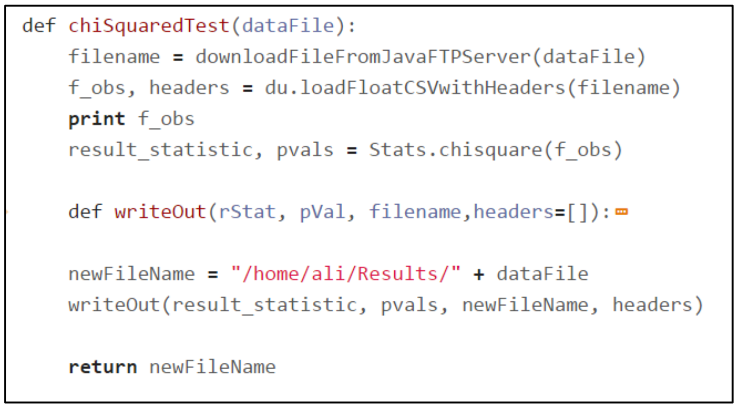
***Figure 7**: Flask Testing Code*

Similarly, if an additional math function is desired, one need only create a wrapper function that takes in a string, dataFile, which is the name of the file to be analyzed (see **Figure 9**). Then add a conditional with the testType string associated with that test. For fetching and saving the data, there are existing utility functions provided by the API that can easily be integrated with any new math functions.



```
@app.route('/MATH/<testType>/<dataFile>')
def runMath(testType, dataFile):
    if str(testType) == "chisquared1":
        chiSquaredTest(dataFile)
    if str(testType) == "chisquared2":
        chi2(dataFile)
    if str(testType) == "anova":
        anova(dataFile)
    if str(testType) == "ttestInd":
        ttest(dataFile, 1)
    if str(testType) == "ttestIndWelch":
        ttest(dataFile, 2)
    if str(testType) == "ttestRel":
        ttest(dataFile, 3)
    else:
        return ("No Valid Test Selected")
    return ("Running " + testType + " on the " + dataFile + " data.")
```

***Figure 8**: Flask Math Code*



```
def chiSquaredTest(dataFile):
    filename = downloadFileFromJavaFTPServer(dataFile)
    f_obs, headers = du.loadFloatCSVwithHeaders(filename)
    print f_obs
    result_statistic, pvals = Stats.chisquare(f_obs)

    def writeOut(rStat, pVal, filename,headers=[]):

    newFileName = "/home/ali/Results/" + dataFile
    writeOut(result_statistic, pvals, newFileName, headers)

    return newFileName
```

***Figure 9**: Example Function Wrapper*

In addition to Flask, we're using several other Python libraries for our API. As mentioned previously, Python has a few options for running deep learning algorithms. We opted to implement the API with Theano and Lasagne. Theano was selected because it was usable with any operating system, where as other options like TensorFlow were Linux specific (Team, et al., 2016). Since this project started in Fall 2016, TensorFlow has claimed to support all operating systems, however the initial deep learning model we were tasked to implement had already been constructed using Theano, so we focused on implementing Theano and Lasagne first. Lasagne acts as a highly modular interface between the programmer and Theano, allowing the programmer to work without knowing the exact details behind setting up neural networks (Dieleman, et al., 2015). Future additions to the API could readily add support for other libraries such as TensorFlow, with only a few basic changes to the AWS image. For our statistics functions, we used NumPy and the statistics module from SciPy to handle the calculations and formatting of the data.

Java Integration

Our work within the existing ALI codebase was primarily minor changes to accommodate the new API, and the addition of AWS integration. The deep learning models implemented needed specially formatted data in order to be trained, so code was added and several additions were made to the existing Dataset class, which is responsible for holding sets of data within ALI. These additions included the capability to transpose the data, sort data by column, and add a new column tracking sequential "clips" which is used by the affect detector model.

We also changed the way in which statistics functions are called. Previously, ALI used a series of R-servers to run some basic statistical functions, such as chi-squared and T tests. The R code involved made the system difficult to build on multiple machines and difficult to maintain, as the creator has moved on from the team. To alleviate these problems, statistics functionality was added to the API and arrangements were made for ALI to call out to Python instead of the R servers. This solution will not only be easier to maintain, but it will also scale more effectively. Since statistical functions are being moved to the cloud, the ASSISTments local machines are less likely to be slowed down in cases of heavy load.

Amazon Web Services

Using AWS allows our API to be operated with far less economic overhead. The initial expected budget for a machine to run the deep learning API (taking advantage of graphics processing for improved performance) was around $10,000. This would be a fully equipped, network connected machine that would be installed on the WPI campus and maintained by the ASSISTments team. AWS allows us to forgo such an expense, while also saving time, effort, and physical space. ALI has the option to specify different run configurations for AWS, based on the type of work being done, offering tradeoffs between price and power.

The lowest run configuration is similar to a desktop computer with an older generation Xeon processor, and one gigabyte of RAM, but lacks a dedicated GPU (Amazon Web Services, "EC2 Instance Types", 2017). This configuration is perfect for tasks that are very simple, or aren't time sensitive, and is a part of the free tier of AWS products, meaning that there would be no charge for using this server type. This would allow the ASSISTments team to run low priority tasks over the weekend without any concern of extra charges.

For tasks that require more power, a middle tier was selected that had a high frequency Xeon processor, and 3.75 gigabytes of RAM, but still no GPU. This would be used if extensive mathematical calculations were needed, but does come at a cost. Unless additional functions are added, this configuration will not be needed, but it could be used if Amazon's free tier was overloaded or unavailable.

The final configuration is the only one suited for deep learning tasks. The g2.2xlarge server type has the equivalent of 8 CPU cores, 15 gigabytes of RAM, and a high performance Nvidia GPU. This comes at a higher cost than any of the other configurations, but enables the training and testing of the deep learning models in an hour or less. This means that even if the instance needed to run for the full hour, training a model would cost a maximum of $0.65 if a dedicated instance was used, but will likely cost under $0.19 based on the average spot instance price (Amazon Web Services, "Amazon EC2 Spot Instances Pricing", 2017).

Amazon Web Services scales with incredible ease. Amazon hosts an image, which is similar to a virtual machine snapshot. Each time a server instance is requested, it creates a copy of the image and spins it up on the server. This means that even if several requests are made to the ASSISTments service, each one will have its own dedicated Amazon instance, and that lets them run in parallel without interfering with one another. This way, if several researchers request the simple statistics simultaneously, the same image can be used, without having to worry about the underlying server specifications. Furthermore, if for any reason the GPU instance starts slowing down, a different server type can be configured to instantly add more speed and power.

In the following tables, different pricing estimates are listed, to be compared against the $10,000 initial startup cost to build a locally hosted machine to run the same system. Table 1 is an estimate for the cost to run in the cloud, using dedicated instances, which are more expensive, but more consistently available. Table 2 is the same estimate, but using spot instances, which can cost less, since they are priced based on the demand for those servers. There are also costs associated with transferring data back and forth from the server, which are summarized in Table 3.

***Table 1**: Dedicated Instance Annual Cost Estimate*

Server Type	**Request Type**	**Cost/ Hour**	**Estimated Hours / Week**	**Weekly Operation Cost**	**Yearly Operation Cost**
t2.micro	Dedicated	$ 0.012	20	$ 0.2400	$ 12.48
g2.2xlarge	Dedicated	$ 0.650	5	$ 3.2500	$ 169.00
m3.medium	Dedicated	$ 0.067	20	$ 1.3400	$ 69.68
				Total Cost:	$ 251.16

***Table 2**: Spot Instance Annual Cost Estimate*

Server Type	**Request Type**	**Cost/ Hour**	**Estimated Hours / Week**	**Weekly Operation Cost**	**Yearly Operation Cost**
t1.micro	Spot	$ 0.0041	20	$ 0.0820	$ 4.26
g2.2xlarge	Spot	$ 0.1898	5	$ 0.9490	$ 49.35
m3.medium	Spot	$ 0.0106	20	$ 0.2120	$ 11.02
				Total Cost:	$ 64.64

***Table 3**: Data Transfer Annual Cost Estimate*

Data Transfer Costs	**Per GB**	**Per MB**	**Per KB**
Into AWS	$ 0.01	$ 0.0000097656	$ 0.00000000953674
Out of AWS	$ 0.01	$ 0.0000097656	$ 0.00000000953674
Expected Monthly Traffic	Max 2GB each way		
Total Monthly Cost:	$ 0.04		
Yearly Cost:	$ 0.48		

Even if ASSISTments chose to run all the requests using dedicated instances, the annual cost would be around $252, compared to the cost of $2,000/year for five years if a local machine was built. After 4-5 years, the local machine would need to be rebuilt, and improved to make use of new technologies. By using AWS, the team would only need to change over to a new server configuration as needs change. Even if ASSISTments increases its expected usage of AWS to 500% its original estimate, the system would still only cost $1,260 which is a cost savings of 27% compared to the local machine. If only spot instances are used, the cost reduction is over 25% and 83% respectively compared to dedicated instances and a local machine.

Results

The project has undergone initial testing, using a local machine as an isolated testbed. The API functions identically to how it should in production, with only a few slight changes made to the networking code. Our original design for network communication with the cloud required specific ports to be opened, something that was infeasible with the limitations in place on the WPI network. A formal solution to this problem is in development, using a WPI based virtual machine as a webserver. For this test, we used a workaround to allow the API to run without modifying any ports. The testbed successfully provided a proof of concept, where ALI was able to spin up an AWS GPU instance, and upload a data file to the API. From there, the Python code trained and tested the deep learning model, and returned the resulting student affect CSV file. Using this result, a graduate student prepared the following heat map, Figure 10, which is an example of what could be included in the new deep learning summary report from ALI. From start to finish, the system completed all the tasks in about 30 minutes, without any manual oversight.

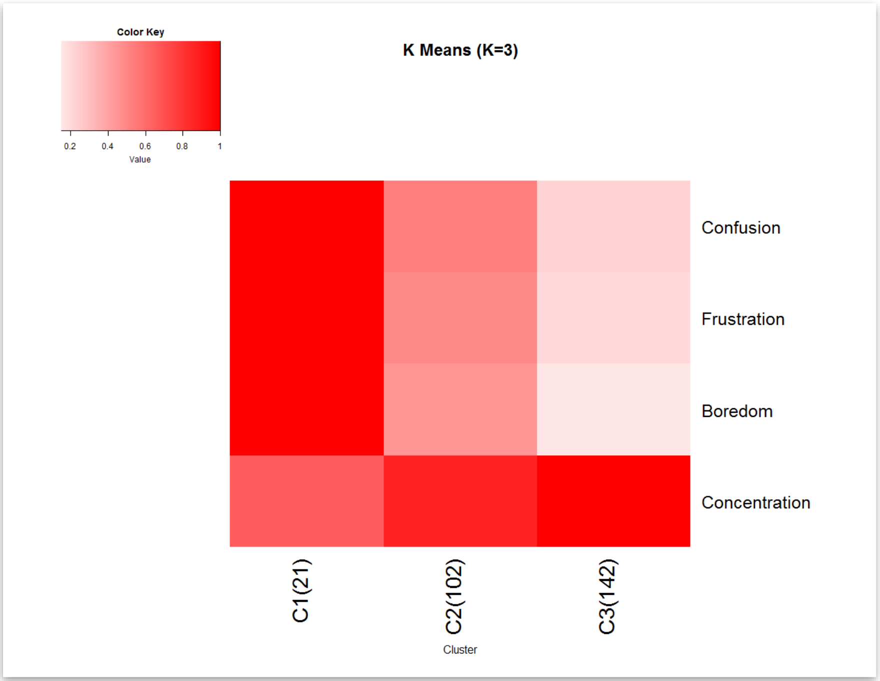


***Figure 10**: Sample K Means Visualization*

The results illustrated in Figure 10 exemplify the types of analyses that a researcher will be able to perform from the deep learning results reported through this API. The analysis was performed using student data from a large RCT reported by a previously released dataset containing information from 22 randomized controlled trials run within ASSISTments (Selent, Patikorn, & Heffernan, 2016). As an example of what is reported, estimates of student affect are generated for students on all recorded logs prior to each student beginning the RCT by the aforementioned deep learning model run through the described workflow. The estimates of concentration, confusion, boredom, and frustration are aggregated for each of the 265 students involved in the study to gain the distribution of average affect estimates of each student prior to beginning the RCT. A typical researcher may use such estimates to observe different groupings or clusters of students, perhaps to measure heterogeneity within each condition of the experiment. Such an analysis is performed here, illustrating the three clusters of students that emerge when using a k-means clustering analysis on the student affect estimates.

In this example, the results suggest that most students previously concentrated on their work in the system (C3 in Figure 10), while only a small group of students exhibited larger amounts of confusion, frustration, and boredom (C1 in Figure 10). The result could be interpreted as indicating gaps in knowledge or understanding prior to beginning the experiment for the students in C1. As was subsequently performed here, differences in condition can then be explored within each cluster of student to observe if the particular learning intervention was more effective for one type of student over another. While no significant differences ($p < .05$) were observable in this test case, it exemplifies the type of analyses that are possible from the resulting infrastructure developed in this work.

Such analyses previous to the implementation of the described workflow were restricted to just the few researchers with access to these state-of-the-art deep learning models. The workflow allows researchers to access the estimates produced by these models in a manner that can be understood even by those individuals who may not be familiar with the development of deep learning models; many external researchers working with the system have backgrounds in psychology-related fields where the application of such models, rather than their development, falls within the scope of their interests and skillsets.

Impact

Conclusion

The finished project provides several immediate benefits to the ASSISTments team. The API with its current functionality provides a greatly simplified workflow for ALI, adding clarity to statistics calls, and a smoothly automated method for running deep learning tasks. Due to its construction with Python and Flask, the API is readable and maintainable for future additions and improvements, even for newcomers to the team. Team members who may not be directly involved with infrastructure will be able to add in new models and features without spending a lot of time learning the syntactic details of a framework. As a result of using AWS for our computations, the API is both scalable and affordable, allowing the department to build many reports simultaneously and quickly, without paying the high cost of an on-site system. All of these benefits combined will allow for improved research potential within ASSISTments, making deep learning a more accessible option, opening up new avenues of analysis for researchers inside and outside of WPI. This work directly supports a current NSF grant awarded to WPI, *Adding Research Accounts to the ASSISTments' Platform: Helping Researchers Do Randomized Controlled Studies with Thousands of Students* (National Science Foundation, 2016). It has also been mentioned as part of the basis for another pending grant, focused on bringing deep learning capabilities to external ASSISTments experiments.

Future Work

Moving forward, there are a few additional areas of improvement for this project. The next big step will be migrating the system from its isolated testing environment to the actual production system. This will involve configuring the new Linux virtual machine which is to be hosted at ali-aws.cs.wpi.edu, and will act as a webserver for the intermediary data files. Once the server is configured, the ASSISTments web interface will need to be updated with options to let external researchers request the affect data model be run. Additionally, new SQL queries need to be integrated into ALI, which are used to select a special subset of data that allows the affect data model to compare previous affects against the affects of students after the given experiment. With all of that in place, the last step will be finalizing the report format for deep learning models, which is still changing, as we determine what information and visuals will best support the needs of researchers.

References

Amazon Web Services. (2017). What is Cloud Computing. Retrieved April 25, 2017, from https://aws.amazon.com/what-is-cloud-computing/

Amazon Web Services. (2017). EC2 Instance Types. Retrieved April 25, 2017, from https://aws.amazon.com/ec2/instance-types/

Amazon Web Services. (2017). Amazon EC2 Spot Instances Pricing. Retrieved April 25, 2017, from https://aws.amazon.com/ec2/spot/pricing/

Analyze Data. (2016). Retrieved April 25, 2017, from http://www.assistmentstestbed.org/the-data

Botelho, A. F., Baker, R. S., & Heffernan, N. T. (2017). Improving Sensor-Free Affect Detection Using Deep Learning. Eighteenth International Conference on Artificial Intelligence in Education.

Dieleman, S., Schlüter, J., Raffel, C., Olson, E., Sønderby, S. K., Nouri, D., . . . Degrave, J. (2015, August 13). Lasagne: First release. Retrieved April 25, 2017, from https://zenodo.org/record/27878#.WP_hFIgrKUk

Metz, C. (2016, March 15). Google's AI Wins Fifth And Final Game Against Go Genius Lee Sedol. Retrieved April 25, 2017, from https://www.wired.com/2016/03/googles-ai-wins-fifth-final-game-go-genius-lee-sedol/

National Science Foundation. (2016, September 1). SI2-SSE: Adding Research Accounts to the ASSISTments' Platform: Helping Researchers Do Randomized Controlled Studies with Thousands of Students (Standard Grant). Retrieved April 25, 2017, from National Science Foundation website: https://www.nsf.gov/awardsearch/showAward?AWD_ID=1440753
Award Number 1440753

Piech, C., Bassen, J., Huang, J., Ganguli, S., Sahami, M., Guibas, L., & Sohl-Dickstein, J. (2015). Deep Knowledge Tracing. Retrieved April 25, 2017, from http://papers.nips.cc/paper/5654-divide-and-conquer-learning-by-anchoring-a-conical-hull.pdf

Roschelle, J., Feng, M., Murphy, R. F., & Mason, C. A. (2016, October 24). Learning Increase with Immediate Feedback on Textbook and Skill Homework. Retrieved April 15, 2017, from http://www.aboutus.assistments.org/homework-immediate-feedback---1-year-study.php

Selent, D., Patikorn, T., & Heffernan, N. (2016). ASSISTments Dataset from Multiple Randomized Controlled Experiments. In Third ACM Conference on Learning@ Scale (pp. 181-184). ACM.

Silver, D., Huang, A., & Maddison, C. J. (2016, January 1). Mastering the game of Go with Deep Neural Networks & Tree Search. Retrieved April 25, 2017, from https://deepmind.com/research/publications/mastering-game-go-deep-neural-networks-tree-search/

Team, T. T., Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., . . . Tulder, G. V. (2016, May 09). Theano: A Python framework for fast computation of mathematical expressions. Retrieved April 25, 2017, from http://arxiv.org/abs/1605.02688

Appendix

A. ALI Dataset Source Code

DataSet.java

```
package org.assistments.service.datadumper.dataset;

import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.io.Serializable;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Scanner;
import java.util.Set;



public class DataSet implements Serializable
{
	private static final long serialVersionUID = 1L;

	private DataRow headerRow;
	private List<DataRow> dataRows;
	private List<DataTypes> columnDataTypes;
	private String title;

	public DataSet()
	{
		headerRow = new DataRow();
		headerRow.setParentDataSet(this);
		dataRows = new ArrayList<DataRow>();
		columnDataTypes = new ArrayList<DataTypes>();
	}

	public DataSet(DataSet otherDataSet)
	{
		columnDataTypes = new ArrayList<DataTypes>();
		dataRows = new ArrayList<DataRow>();
		title = otherDataSet.getTitle();

		headerRow = new DataRow(otherDataSet.getHeaderRow());
		headerRow.setParentDataSet(this);

		for(int i=0; i<otherDataSet.getNumberOfRows(); i++)
		{
			DataRow newDataRow = new DataRow(otherDataSet.getDataRow(i));
			//newDataRow.setParentDataSet(this);
			addDataRow(newDataRow);
		}

		for(int i=0; i<otherDataSet.getNumberOfColumns(); i++)
		{
			columnDataTypes.add(otherDataSet.getColumnDataType(i));
		}
	}

	public void setTitle(String title)
	{
		this.title = title;
	}

	public String getTitle()
	{
		return title;
	}

	public void setColumnDataTypes(List<DataTypes> newColumnDataTypes)
	{
		columnDataTypes.clear();

		for(DataTypes columnDataType : newColumnDataTypes)
		{
			columnDataTypes.add(columnDataType);
		}
	}

	public void setColumnDataTypes(DataTypes[] newColumnDataTypes)
	{
		columnDataTypes.clear();

		for(DataTypes columnDataType : newColumnDataTypes)
		{
			columnDataTypes.add(columnDataType);
		}
	}

	public void setColumnDataType(int index, DataTypes dataType)
	{
		columnDataTypes.set(index, dataType);
	}

	public DataTypes getColumnDataType(int index)
	{
		return columnDataTypes.get(index);
	}

	public void addColumnDataType(DataTypes dataType)
	{
		columnDataTypes.add(dataType);
	}

	public void setHeaderRow(DataRow headerRow)
	{
		if(headerRow.getParentDataSet() != null)
		{
			throw new IllegalArgumentException("Cannot set a header row which
already has a parent DataSet");
		}

		this.headerRow = new DataRow(headerRow);
		this.headerRow.setParentDataSet(this);
	}

	public void setHeaderRow(String[] headerRow)
	{
		this.headerRow = new DataRow(headerRow);
		this.headerRow.setParentDataSet(this);
	}

	public void setHeaderRow(String headerRow)
	{
		this.headerRow = new DataRow(headerRow.split(","));
		this.headerRow.setParentDataSet(this);
	}

	public void setHeaderRowValue(int columnIndex, String newValue)
	{
		headerRow.setRowValue(columnIndex, newValue);
	}

	public DataRow getHeaderRow()
	{
		return headerRow;
	}

	public String[] getHeaderStringArrayRow()
	{
		return headerRow.getRowStringArrayValues();
	}

	public String getHeaderRowValue(int index)
	{
		return headerRow.getColumn(index);
	}

    public int getHeaderColumnIndexByString(String name) throws RuntimeException
    {
	int index = -1;

	for(int i = 0; i < headerRow.getColumns() && index == -1; i++)
	{
		if(headerRow.getColumn(i).equals(name))
		{
			index = i;
		}
	}

	if(index == -1)
	{
		throw new RuntimeException("No header column found with value = " + "\""
+ name + "\"");
	}

	return index;
    }

	public void addDataRow(DataRow dataRow)
	{
		if(dataRow.getParentDataSet() != null)
		{
			throw new IllegalArgumentException("Cannot add a DataRow to a
DataSet that already belongs to another DataSet");
		}
		else
		{
			dataRow.setParentDataSet(this);
			dataRows.add(dataRow);
		}
	}
```

```
public void addMultipleDataRows(List<DataRow> dataRows)
{
        for (DataRow dataRow: dataRows)
        {
                if(dataRow.getParentDataSet() == null)
                {
                        addDataRow(dataRow);
                }
                else
                {
                        addDataRow(new DataRow(dataRow));
                }
        }
}

public void addDataRow(String[] dataRowArray)
{
        DataRow dataRow = new DataRow(dataRowArray);
        dataRow.setParentDataSet(this);
        dataRows.add(dataRow);
}

public void addDataRow(String dataRowString)
{
        DataRow dataRow = new DataRow(dataRowString);
        dataRow.setParentDataSet(this);
        dataRows.add(dataRow);
}

/*
//looks like an append data set function that is not used
public void attachRows(DataSet newData)
{
        for(DataRow iRow: newData.getDataRows())
        {
                addDataRow(new DataRow(iRow));
        }
}
*/

public List<DataRow> getDataRows()
{
        return dataRows;
}

public DataRow getDataRow(int index)
{
        return dataRows.get(index);
}

public String getDataStringRow(int index)
{
        return dataRows.get(index).toString();
}

public String[] getDataStringArrayRow(int index)
{
        return dataRows.get(index).getRowStringArrayValues();
}

public String getDataRowValue(int rowIndex, int columnIndex)
{
        return dataRows.get(rowIndex).getColumn(columnIndex);
```

```
	}

	public void removeDataRow(int index)
	{
		dataRows.remove(index);
	}

	public void addColumn(int columnIndex)
	{
		addColumn(columnIndex, DataTypes.STRING);
	}

	public void addColumn(int columnIndex, List<String> content)
	{
		addColumn(columnIndex, DataTypes.STRING, content);
	}

	public void addColumn(int columnIndex, DataTypes columnDataType)
	{
		headerRow.addColumn(columnIndex);
		columnDataTypes.add(columnIndex, columnDataType);

		for(DataRow row : dataRows)
		{
			row.addColumn(columnIndex);
		}
	}

	public void addColumn(int columnIndex, DataTypes columnDataType, List<String>
content)
	{
		headerRow.addColumn(columnIndex,content.get(0));
		columnDataTypes.add(columnIndex, columnDataType);

		int contentInd = 1;
		for(DataRow row : dataRows)
		{
			row.addColumn(columnIndex, content.get(contentInd));
			contentInd++;
		}
	}

	public void removeColumn(int columnIndex)
	{
		headerRow.removeColumn(columnIndex);
		columnDataTypes.remove(columnIndex);

		for(DataRow row : dataRows)
		{
			row.removeColumn(columnIndex);
		}
	}

	public List<String> getColumn(int columnIndex)
	{
		List<String> column = new ArrayList<String>();
		column.add(headerRow.getColumn(columnIndex));

		for(DataRow row : dataRows)
		{
			column.add(row.getColumn(columnIndex));
		}
		return column;
	}

	public List<String> getColumnWithoutHeader(int columnIndex)
	{
		List<String> column = new ArrayList<String>();

		for(DataRow row : dataRows)
		{
			column.add(row.getColumn(columnIndex));
		}

		return column;
	}

	public void setCellValue(int rowIndex, int columnIndex, String newValue)
	{
		dataRows.get(rowIndex).setRowValue(columnIndex, newValue);
	}

	public int getNumberOfRows()
	{
		return dataRows.size();
	}

	public int getNumberOfColumns()
	{
		return headerRow.getColumns();
	}

	public int getNumberOfColumns(int rowIndex)
	{
		return dataRows.get(rowIndex).getColumns();
	}

	public void trimAllSpace()
	{
		headerRow.trimAllSpace();

		for(DataRow row : dataRows)
		{
			row.trimAllSpace();
		}
	}

	public List<String> getUniqueValues(int columnIndex)
	{
		List<String> allValue = getColumnWithoutHeader(columnIndex);
		Set<String> uniqueValue = new HashSet<String>(allValue);
		return new ArrayList<String>(uniqueValue);
	}

	//Find in the column columnIndex, return rows with value string
	public List<DataRow> getRowsWithValue(int columnIndex, String value)
	{
		List<DataRow> result = new ArrayList<DataRow>();

		for(DataRow row: dataRows)
		{
			if(row.getColumn(columnIndex).equals(value))
			{
				result.add(row);
			}
		}

		return result;
	}

	//Returns a blank row with the same number of columns that this data set
currently has
	public DataRow getBlankRow()
	{
		int num = getNumberOfColumns();
		List<String> rowValues = new ArrayList<String>();

		for(int i=0;i<num;i++)
		{
			rowValues.add("");
		}

		DataRow row = new DataRow(rowValues);
		return row;
	}

	public boolean isEmpty()
	{
		return dataRows.isEmpty();
	}

	public void addBlankRow()
	{
		addDataRow(getBlankRow());
	}

	public void sort(int columnIndex)
	{
		sort(columnIndex, DataTypes.STRING);
	}

	public void sort(int columnIndex, DataTypes comparableType)
	{
		sort(columnIndex, comparableType, true);
	}

	public void sort(int columnIndex, DataTypes comparableType, boolean sameParent)
	{
		sort(columnIndex, comparableType, sameParent, true);
	}

	public void sort(int columnIndex, DataTypes comparableType, boolean sameParent,
boolean ascending)
	{
		CompareProperties compareProperties = new CompareProperties();
		DataRowComparator dataRowComparator = new
DataRowComparator(compareProperties);

		compareProperties.setComparableIndex(columnIndex);
		compareProperties.setComparableType(comparableType);
		compareProperties.setSameParent(sameParent);
		compareProperties.setAscending(ascending);

		Collections.sort(dataRows, dataRowComparator);
	}

	public void sort(int startIndex, int endIndex, int columnIndex, DataTypes
comparableType, boolean sameParent, boolean ascending)
```

```
	{
		CompareProperties compareProperties = new CompareProperties();
		DataRowComparator dataRowComparator = new 
DataRowComparator(compareProperties);
		
		compareProperties.setComparableIndex(columnIndex);
		compareProperties.setComparableType(comparableType);
		compareProperties.setAscending(ascending);
		compareProperties.setSameParent(sameParent);
		
		List<DataRow> test = dataRows.subList(startIndex, endIndex);
		Collections.sort(test, dataRowComparator);
	}
	
	public void sortCol(int startIndex, int endIndex, int columnIndex, DataTypes 
comparableType, boolean sameParent, boolean ascending){
		
		//Binding each header value to its respective data type
		HashMap<String, DataTypes> headerBindings = new HashMap<String, 
DataTypes>();
		int dTypeInd = 0;
		for(String x: this.getHeaderRow().getRowValues())
		{
			headerBindings.put(x, columnDataTypes.get(dTypeInd));
		}
		
		//transpose, sort, and second transpose go here
		  //Add the Header Row
		dataRows.add(0,headerRow);
		  //Transpose
		this.dataRows = this.transpose();
		  //Sort
		this.sort(startIndex, endIndex, columnIndex, comparableType, sameParent, 
ascending);
		  //Transpose back
		this.dataRows = this.transpose();
		
		//headerRow should be set to the new value of the header row after sort
		
		ArrayList<DataTypes> revisedTypes = new ArrayList<DataTypes>();
		
		for(String x: this.getHeaderRow().getRowValues())
		{
			revisedTypes.add(headerBindings.get(x));
		}
		this.setColumnDataTypes(revisedTypes);
		
		//Strip off the Header Row
		this.dataRows.remove(0);
		
	}
	
	public void swap(int indexColA, int indexColB){
		int A = indexColA-1;
		int B = indexColB-1;
		String tempS = "";
		String[] headers = this.headerRow.getRowStringArrayValues();
		tempS = headers[A];
		headers[A] = headers[B];
		headers[B] = tempS;
		this.headerRow.setRowValues(headers);
		for(DataRow x: this.dataRows){
			List<String> currR = x.getRowValues();
```

```
                    tempS = currR.get(A);
                    currR.set(A, currR.get(B));
                    currR.set(B, tempS);
                    x.setRowValues(currR.toArray(new String [1]));
            }
    }

    public List<DataRow> transpose(){
            int cols = 
this.getDataRow(0).getRowValues().size();//this.getNumberOfColumns();
            int rows = this.getNumberOfRows();

            //Create the empty transposed object
            List<DataRow> transposed = new ArrayList<DataRow> ();
            for(int i = 0; i < cols; i++){
                    transposed.add(new DataRow());
            }

            //Iterate through each of the current datarows, and then iterate through 
their
            //elements adding them one at a time to the transposed position
            for(int i = 0; i < rows; i++){
                    System.out.println("Transposing Row:"+i);
                    DataRow currDRow = dataRows.get(i);
                    List<String> dVals = currDRow.getRowValues();
                    for(int j = 0; j < cols; j++){
                            //Get the current transposed datarow
                            DataRow currTRow = transposed.get(j);
                            //Get the datarow data list
                            List<String> tVals = currTRow.getRowValues();
                            tVals.add(dVals.get(j));
                            currTRow.setRowValues(tVals.toArray(new String[1]));
                            transposed.set(j,currTRow);
                    }
            }
            return transposed;
    }

    public boolean addRNNClipSequence(int jumpSize){
            if(headerRow.getRowValues().contains("clip_sequence")){
                    return true;
            }
            else if(headerRow.getRowValues().contains("clip")){
                    int hIndex = headerRow.getRowValues().indexOf("clip");
                    List<String> clipCol = getColumn(hIndex);
                    List<String> newCol = new ArrayList<String>();
                    newCol.add("clip_sequence");
                    int clipSeq = 1;
                    newCol.add(Integer.toString(clipSeq));
                    for(int i = 2; i < clipCol.size(); i++){
                            if((Integer.parseInt(clipCol.get(i)) <= 
Integer.parseInt(clipCol.get(i-1)) + jumpSize) && (Integer.parseInt(clipCol.get(i)) >= 
Integer.parseInt(clipCol.get(i-1)) + 1)){
                                    newCol.add(Integer.toString(clipSeq));
                            }
                            else{
                                    clipSeq++;
                                    newCol.add(Integer.toString(clipSeq));
                            }
                    }
                    addColumn(hIndex, DataTypes.STRING, newCol);
                    return true;
            }
            else{
                  return false;
            }
            //return true;
      }

      public String buildCSV(){
            String filename = title + "-" + Long.toString(System.currentTimeMillis())
+ ".csv";
            try{
                PrintWriter writer = new PrintWriter("Z:/public_html/" + filename,
"UTF-8");
                //writer.println("The first line");
                int cols =
this.getDataRow(0).getRowValues().size();//this.getNumberOfColumns();
                  int rows = this.getNumberOfRows();
                  System.out.println("Writing Header Row");
                  List<String> hVals = headerRow.getRowValues();
                  for(int k = 0; k < cols; k++){
                        writer.format("%s,", hVals.get(k));
                  }
                  writer.println();
                for(int i = 0; i < rows; i++){
                        System.out.println("Writing Row: "+i);
                        DataRow currDRow = dataRows.get(i);
                        List<String> dVals = currDRow.getRowValues();
                        for(int j = 0; j < cols; j++){
                              writer.format("%s,", dVals.get(j));
                        }
                        writer.println();
                  }
                writer.close();
            } catch (Exception e) {
               // do something
            }
            return filename;
      }

      public String toHTMLString(boolean hasBorders)
      {
            StringBuilder sb = new StringBuilder();

            if(hasBorders)
            {
                  sb.append("<table border = \"1\">");
            }
            else
            {
                  sb.append("<table>");
            }

            sb.append(System.getProperty("line.separator"));
            sb.append(headerRow.toHTMLString());

            for(DataRow dataRow : dataRows)
            {
                  sb.append(System.getProperty("line.separator"));
                  sb.append(dataRow.toHTMLString());

            }

            sb.append(System.getProperty("line.separator"));
            sb.append("</table>");
```

```
		return sb.toString();
	}

	public String toString()
	{
		StringBuilder sb = new StringBuilder();

		sb.append(headerRow.toString());

		sb.append(System.getProperty("line.separator"));

		//might be faster not to call to string for the individual rows
		for(int i=0; i<dataRows.size(); i++)
		{
			sb.append(dataRows.get(i).toString());
			sb.append(System.getProperty("line.separator"));
		}

		sb.deleteCharAt(sb.length()-1);

		return sb.toString();
	}



	//TO DO
	//NEED TO IMPLEMENT HASHCODE
	//NEED TO CONSIDER COLUMN DATA TYPES IN BOTH EQUALS AND HASHCODE
	@Override
	public boolean equals(Object obj)
	{
		boolean equal = true;

		if(obj == null)
		{
			equal = false;
		}
		else if(this == obj)
		{
			equal = true;
		}
		else if(!(obj instanceof DataSet))
		{
			equal = false;
		}
		else
		{
			DataSet other = (DataSet) obj;

			if(this.getNumberOfRows() != other.getNumberOfRows())
			{
				equal = false;
			}
			else if(this.headerRow != null && other.getHeaderRow() != null &&
!this.headerRow.equals(other.getHeaderRow()))
			{
				equal = false;
			}
			else if(!dataRows.equals(other.getDataRows()))
			{
				equal = false;
			}
```

```
                }

                return equal;
        }

}
```

B. ALI AWS Services Source Code

FlaskServer.java

```
package org.assistments.service.awsserver;

import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2ClientBuilder;
import com.amazonaws.services.ec2.model.ActiveInstance;
import com.amazonaws.services.ec2.model.AllocationStrategy;
import com.amazonaws.services.ec2.model.AuthorizeSecurityGroupIngressRequest;
import com.amazonaws.services.ec2.model.CancelSpotFleetRequestsRequest;
import com.amazonaws.services.ec2.model.CreateSecurityGroupRequest;
import com.amazonaws.services.ec2.model.DescribeInstancesRequest;
import com.amazonaws.services.ec2.model.DescribeInstancesResult;
import com.amazonaws.services.ec2.model.DescribeSpotFleetInstancesRequest;
import com.amazonaws.services.ec2.model.DescribeSpotFleetInstancesResult;
import com.amazonaws.services.ec2.model.FleetType;
import com.amazonaws.services.ec2.model.GroupIdentifier;
import com.amazonaws.services.ec2.model.Instance;
import com.amazonaws.services.ec2.model.InstanceType;
import com.amazonaws.services.ec2.model.IpPermission;
import com.amazonaws.services.ec2.model.RequestSpotFleetRequest;
import com.amazonaws.services.ec2.model.RequestSpotFleetResult;
import com.amazonaws.services.ec2.model.Reservation;
import com.amazonaws.services.ec2.model.SpotFleetLaunchSpecification;
import com.amazonaws.services.ec2.model.SpotFleetRequestConfigData;
import com.amazonaws.services.ec2.model.TerminateInstancesRequest;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Scanner;

import org.apache.commons.net.ftp.FTP;
import org.apache.commons.net.ftp.FTPClient;


public class FlaskServer {
	int serverNum;
	String instanceIdString;
	String spotFleetRequestIdString;
	String publicDNSString;
	String publicIpAddrString;
	String localIpAddrString;
	String keyName = "ali-dumper-key";
	static int ftpPort = 21;
	AmazonEC2 ec2;
	@SuppressWarnings("deprecation")
	public boolean createSecurityGroup(String groupName, String groupDescription){
		try{
		    CreateSecurityGroupRequest securityGroupRequest = new 
CreateSecurityGroupRequest(groupName, groupDescription);
		    ec2.createSecurityGroup(securityGroupRequest);
		} catch (AmazonServiceException ase){
		    // Likely this means that the group is already created, so ignore.
			System.out.println("Group Already Exists");
		    System.out.println(ase.getMessage());
		    return false;
		}
		
		String ipAddr = "0.0.0.0/0";

		// Get the IP of the current host, so that we can limit the Security
		// Group by default to the ip range associated with your subnet.
		try {
		    InetAddress addr = InetAddress.getLocalHost();

		    // Get IP Address
		    ipAddr = addr.getHostAddress()+"/10";
		} catch (UnknownHostException e) {
			System.out.println("Failed to get IP Address");
			System.out.println(e.getMessage());
			return false;
		}

		// Create a range that you would like to populate.
		ArrayList<String> ipRanges = new ArrayList<String>();
		ipRanges.add(ipAddr);

		// Open up port 22 for TCP traffic to the associated IP
		// from above (e.g. ssh traffic).
		ArrayList<IpPermission> ipPermissions = new ArrayList<IpPermission> ();
		IpPermission ipPermission = new IpPermission();
		ipPermission.setIpProtocol("tcp");
		//ipPermission.
		ipPermission.setFromPort(new Integer(1));
		ipPermission.setToPort(new Integer(10000));
		ipPermission.setIpRanges(ipRanges);
		ipPermissions.add(ipPermission);

		try {
		    // Authorize the ports to the used.
		    AuthorizeSecurityGroupIngressRequest ingressRequest =
		        new AuthorizeSecurityGroupIngressRequest(groupName, 
ipPermissions);
		    ec2.authorizeSecurityGroupIngress(ingressRequest);
		} catch (AmazonServiceException ase) {
		    // Ignore because this likely means the zone has
		    // already been authorized.
			System.out.println("Failed to authorize ports");
		    System.out.println(ase.getMessage());
		    return false;
		}
		return true;
	}
	
	public FlaskServer(int sNum){
		//Init Variables
		serverNum = sNum;
		this.instanceIdString = "";
		this.spotFleetRequestIdString = "";
		this.publicDNSString = "";
		this.publicIpAddrString = "";

		//Get the external ipAddress of the machine this is running on for Flask
		URL whatismyip;
		try {
			whatismyip = new URL("http://checkip.amazonaws.com");
			BufferedReader in = new BufferedReader(new InputStreamReader(
			   whatismyip.openStream()));

			this.localIpAddrString = in.readLine(); //you get the IP as a
String
			System.out.println("Machine's IpAddr: "+ localIpAddrString);
		} catch (MalformedURLException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}


		//Build the EC2 Portion of the Service
		AWSCredentialsProvider credentials = new
ProfileCredentialsProvider("XXXX", "default");
		this.ec2 = AmazonEC2ClientBuilder.standard()
				.withRegion(Regions.US_EAST_1)
		   .withCredentials(credentials)
		   .build();
	}

	@SuppressWarnings("deprecation")
	public boolean spinUpFlask(boolean local, String instanceType){
		if(local){
			return false;
		}
		else{
			// Initializes a Spot Fleet Instance Request
			RequestSpotFleetRequest requestFleetRequest = new
RequestSpotFleetRequest();
			SpotFleetRequestConfigData launchConfig = new
SpotFleetRequestConfigData();
			SpotFleetLaunchSpecification launchSpecification = new
SpotFleetLaunchSpecification();
			String bidPrice = "0.00";
			InstanceType instanceId = null;
			switch(instanceType){
			case "GPU":
				bidPrice = "0.65";
				instanceId = InstanceType.G22xlarge;
				break;
			case "FREE":
				bidPrice = "0.03";
				instanceId = InstanceType.T2Micro;
				break;
			default:
				System.err.println("Invalid Instance Type: " + instanceType
+ " (Only FREE and GPU Permitted)");
				break;
			}

			//Configure the Fleet Request
			launchConfig.setSpotPrice(bidPrice);
```

```
                    launchConfig.setTargetCapacity(Integer.valueOf(1));

        //launchConfig.setIamFleetRole("arn:aws:iam::189674270017:role/aws-ec2-spot-
fleet-role"); //Required
                    launchConfig.setIamFleetRole("xxxx");
                    launchConfig.setType(FleetType.Request);

        launchConfig.setAllocationStrategy(AllocationStrategy.LowestPrice);
                    launchConfig.setTerminateInstancesWithExpiration(true);
                    //launchConfig.setFulfilledCapacity(0.0);
                    Date endDate = new Date();

                    endDate.setYear(3018-1900);
                    //System.out.println(endDate);
                    launchConfig.setValidUntil(endDate);

                    // Setup the specifications of the launch. This includes the
                    // instance type (e.g. t1.micro) and the latest Amazon Linux
                    // AMI id available. Note, you should always use the latest
                    // Amazon Linux AMI id or another of your choosing.
                    launchSpecification.setImageId("ami-8e2b2a99");
                    launchSpecification.setInstanceType(instanceId);
                    launchSpecification.setKeyName("tower-only-second-acct");


                    // Add the security group to the launch specifications.
                    ArrayList<GroupIdentifier> securityGroups = new
ArrayList<GroupIdentifier>();
                    securityGroups.add(new GroupIdentifier().withGroupId("xxxxx"));
                    System.out.println(securityGroups);
                    launchSpecification.setSecurityGroups(securityGroups);

                    // Add the launch specifications to the launch config.
                    ArrayList<SpotFleetLaunchSpecification> launchSpecs = new
ArrayList<SpotFleetLaunchSpecification>();
                    launchSpecs.add(launchSpecification);
                    launchConfig.setLaunchSpecifications(launchSpecs);


                    //Add the config to the fleetRequest
                    requestFleetRequest.setSpotFleetRequestConfig(launchConfig);

                    // Call the RequestSpotInstance API to determine the FleetId.
                    RequestSpotFleetResult requestResult =
ec2.requestSpotFleet(requestFleetRequest);
                    System.out.println("Fleet Request ID: " +
requestResult.getSpotFleetRequestId());

                    // Determine the InstanceId using the FleetId
                    DescribeSpotFleetInstancesRequest desReq = new
DescribeSpotFleetInstancesRequest();
                    this.spotFleetRequestIdString =
requestResult.getSpotFleetRequestId();

        desReq.setSpotFleetRequestId(requestResult.getSpotFleetRequestId());
                    boolean instanceLoading = true;
                    ArrayList<String> spotInstanceIds = new ArrayList<String>();
//Storage Vessel
                    System.out.print("Checking for Active State");
                    while(instanceLoading){
                            DescribeSpotFleetInstancesResult desRes =
ec2.describeSpotFleetInstances(desReq);
                            List<ActiveInstance> actInst = desRes.getActiveInstances();
```

```
                                        instanceLoading = actInst.isEmpty();
                                        System.out.print(".");
                                        if(!instanceLoading){
                                                System.out.println("");
                                                for(ActiveInstance a: actInst){
                                                        System.out.println("SpotInstanceRequestID: " +
a.getSpotInstanceRequestId());
                                                        System.out.println("  -InstanceID: " +
a.getInstanceId());
                                                        System.out.println("  -InstanceType: " +
a.getInstanceType());
                                                        spotInstanceIds.add(a.getInstanceId());
                                                        instanceIdString = a.getInstanceId();
                                                }
                                        }
                                        try {
                                                Thread.sleep(30*1000);
                                        } catch (InterruptedException e) {

                                        }
                                }

                                // Fetch the IP address using the FleetId and the InstanceId
                                DescribeInstancesRequest request = new
DescribeInstancesRequest().withInstanceIds(this.instanceIdString);
                             DescribeInstancesResult result= ec2.describeInstances(request);
                             List <Reservation> list  = result.getReservations();
                             System.out.println("Reservation List: ");
                             for(Reservation res:list) {
                                System.out.println("  Reservation: " + res.getReservationId());
                                 List <Instance> instanceList= res.getInstances();
                                 for(Instance instance:instanceList){
                                        System.out.println("    -InstanceId: " +
instance.getInstanceId());
                                      System.out.println("       Public IP: " +
instance.getPublicIpAddress());
                                      System.out.println("       Public DNS: " +
instance.getPublicDnsName());
                                      System.out.println("       Instance State: " +
instance.getState());
                                      System.out.println("       Instance TAGS: " +
instance.getTags());
                                      this.publicIpAddrString = instance.getPublicIpAddress();
                                      this.publicDNSString = instance.getPublicDnsName();
                                   }
                             }

                             if(spotFleetRequestIdString == "" || instanceIdString == ""
                                        || publicIpAddrString == "" || publicDNSString == ""){
                                        return false;
                                }

                             submitActionToFlask("UPDATE","HOSTNAME",this.localIpAddrString);
                                return true;
                        }
                }

                public boolean spinDownFlask(boolean local){
                        if(local){
                                return false;
                        }
                        else{
                                try {
```

```
                    // Cancel requests.
                    System.out.print("Cancelling Spot Requests...");
                    CancelSpotFleetRequestsRequest cancelRequest = new
CancelSpotFleetRequestsRequest().withSpotFleetRequestIds(spotFleetRequestIdString);
                    ec2.cancelSpotFleetRequests(cancelRequest);
                    spotFleetRequestIdString = "";
                    System.out.println("Successful");
                } catch (AmazonServiceException e) {
                    // Write out any exceptions that may have occurred.
                     System.out.println("Failed");
                    System.out.println("Error cancelling instances");
                    System.out.println("Caught Exception: " + e.getMessage());
                    System.out.println("Reponse Status Code: " + e.getStatusCode());
                    System.out.println("Error Code: " + e.getErrorCode());
                    System.out.println("Request ID: " + e.getRequestId());
                }

                     try {
                    // Terminate instances.
                    System.out.print("Terminate Running Instances...");
                    TerminateInstancesRequest terminateRequest = new
TerminateInstancesRequest().withInstanceIds(instanceIdString);
                    ec2.terminateInstances(terminateRequest);
                    instanceIdString = "";
                    System.out.println("Successful");
                } catch (AmazonServiceException e) {
                    // Write out any exceptions that may have occurred.
                     System.out.println("Failed");
                    System.out.println("Error terminating instances");
                    System.out.println("Caught Exception: " + e.getMessage());
                    System.out.println("Reponse Status Code: " + e.getStatusCode());
                    System.out.println("Error Code: " + e.getErrorCode());
                    System.out.println("Request ID: " + e.getRequestId());
                }

                     if(spotFleetRequestIdString != "" || instanceIdString != ""){
                            return false;
                     }

                     this.publicDNSString = "";
                     this.publicIpAddrString = "";
                     return true;
              }
       }

       public boolean retrieveFileFromFlask(String filename){
              FTPClient ftpClient = new FTPClient();
              if(publicIpAddrString == ""){
                     return false;
              }
        try {
            ftpClient.connect(publicIpAddrString, ftpPort);
            ftpClient.login("ali", "dumper");
            ftpClient.enterLocalPassiveMode();
            ftpClient.setFileType(FTP.BINARY_FILE_TYPE);

            // APPROACH #1: using retrieveFile(String, OutputStream)
            String remoteFile = "Results/" + filename;
            File downloadFile = new File("results-"+filename);
            OutputStream outputStream = new BufferedOutputStream(new
FileOutputStream(downloadFile));
            System.out.println("Grabbing File Over FTP");
            boolean success = ftpClient.retrieveFile(remoteFile, outputStream);
            System.out.println("Success: " + Boolean.toString(success) );
            outputStream.close();

            if (success) {
                System.out.println(filename + " has been downloaded successfully.");
            }
        } catch (IOException ex) {
            System.out.println("Error: " + ex.getMessage());
            ex.printStackTrace();
        } finally {
            try {
                if (ftpClient.isConnected()) {
                    ftpClient.logout();
                    ftpClient.disconnect();
                }
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
             return true;
       }

      public boolean submitActionToFlask(String action, String model, String
filename){
             try {
                    URLConnection connection = new URL("http://" +
this.publicIpAddrString + ":5000/" + action + "/" + model + "/" +
filename).openConnection();
                    connection.setRequestProperty("Accept-Charset", "UTF-8");
                    InputStream response = connection.getInputStream();
                    try (Scanner scanner = new Scanner(response)) {
                        String responseBody = scanner.useDelimiter("\\A").next();
                        System.out.println(responseBody);
                    }
             } catch (MalformedURLException e) {
                    e.printStackTrace();
             } catch (IOException e) {
                    e.printStackTrace();}
             return true;
       }
}
```

C. Flask API Source Code

flask-server.py

```
#import urllib3
import RNN
import ApplyDetector
from mathFunctions import *
from flask import Flask

app = Flask(__name__)
hostname = "localhost"

@app.route('/')
def root_menu():
    return 'Welcome to Flask - Server is Running\n----------Settings----------\n
            Hostname: ' + hostname


@app.route('/TEST/<pickleFile>/<dataFile>')
def runTest(pickleFile, dataFile):
    local_file = "/home/ubuntu/DeepLearning-Python/Dataset/" + dataFile
    pickleFile2 = pickleFile + ".pickle"
    print dataFile2
    print pickleFile
    if(pickleFile == "RNN"):
        ApplyDetector.apply_model(pickleFile2, local_file,
"/home/ali/Results/"+filename)
        #RNN.evaluate_model(dataFile2, pickleFile2, 200, 1, 20, step_size=0.005,
balance_model=False, scale_output=True, variant="GRU")
    return ("Testing " + dataFile + " on the " + pickleFile + " model.")


@app.route('/TRAIN/<testType>/<filename>')
def runTraining(testType, filename):
    local_file = downloadFileFromJavaFTPServer(filename)
    if(testType == "RNN"):
        model = ApplyDetector.train_and_save_model("RNN.pickle", local_file,
"/home/ali/Results/"+filename)
    return "Training on %s" % filename


@app.route('/UPDATE/<variableName>/<value>')
def updateVariableWithValue(variableName, value):
    s1 = ""
    s2 = ""
    if(variableName == "HOSTNAME"):
        s1 = "hostname"
        s2 = str(value)
        hostname = value
    else:
        return ("No Valid Variable Selected")
    return ("Variable associated with " + s1 + " updated to the value" + s2)


@app.route('/MATH/<testType>/<dataFile>')
def runMath(testType, dataFile):
    if str(testType) == "chisquared1":
        chiSquaredTest(dataFile)
    if str(testType) == "chisquared2":
        chi2(dataFile)
    if str(testType) == "anova":
        anova(dataFile)
    if str(testType) == "ttestInd":
        ttest(dataFile, 1)
    if str(testType) == "ttestIndWelch":
        ttest(dataFile, 2)
    if str(testType) == "ttestRel":
        ttest(dataFile, 3)
    else:
        return ("No Valid Test Selected")
    return ("Running " + testType + " on the " + dataFile + " data.")
```

pickleUtility.py

```
import pickle
import sys

def saveInstance(model,filename):
    pickle.dump(model, open("Models/"+filename,"wb"), -1)

def loadInstance(filename):
    model = pickle.load(open("Models/"+filename, "rb" ))
    return model
```

mathFunctions.py

```
import scipy.stats as Stats
import DataUtility as du
import urllib3
import urllib
import numpy as np


def downloadFileFromJavaFTPServer(dataFile):
    # Should Open and Read in the Data File
    url = "http://users.wpi.edu/~ctjones/" + dataFile
    print "Grabbing File at: " + url
    urllib.urlretrieve(url, 'Resources/'+dataFile)
    return "Resources/" + dataFile #return location of locally saved file

def chiSquaredTest(dataFile):
    filename = downloadFileFromJavaFTPServer(dataFile) #where datafile will be the
location of the data on the external machine
    f_obs, headers = du.loadFloatCSVwithHeaders(filename)
    print f_obs
    result_statistic, pvals = Stats.chisquare(f_obs)

    def writeOut(rStat, pVal, filename,headers=[]):

        with open(filename, 'w') as f:
            if len(headers)!=0:
                for i in range(len(headers)-1):
                    f.write(str(headers[i]) + ',')
                f.write(str(headers[-1])+'\n')
                for j in range(len(rStat)-1):
                    f.write(str(rStat[j]) + ',')
                f.write(str(rStat[-1]) + '\n')
                for k in range(len(pVal)-1):
                    f.write(str(pVal[k]) + ',')
                f.write(str(pVal[-1]) + '\n')
        f.close()

    newFileName = "/home/ali/Results/" + dataFile #using dataFile again as a
convenient filename
    writeOut(result_statistic, pvals, newFileName, headers)
    return newFileName #passing the file name back up so that the main Flask code can
handle sending the file back to Java

def chi2(dataFile):
    filename = downloadFileFromJavaFTPServer(dataFile) #where datafile will be the
location of the data on the external machine
    dataValues, headers = du.loadFloatCSVwithHeaders(filename)

    x2, pValue, dof, exp  = Stats.chi2_contingency(dataValues)


    def writeOut(x2, p, filename):

        with open(filename, 'w') as f:
            f.write("X^2" + ',')
            f.write("PValue" + '\n')
            f.write(str(x2) + ',')
            f.write(str(p) + '\n' )
        f.close()

    newFileName = "/home/ali/Results/" + dataFile #using dataFile again as a
convenient filename
    writeOut(x2, pValue, newFileName)

    return newFileName #passing the file name back up so that the main Flask code can
handle sending the file back to Java

def anova(dataFile):
    '''
    Takes in data from a CSV as such:
    HEADER1 | HEADER2 | HEADER3 | ... | HEADERN
    Val1a   | Val2a   | Val3a   | ... | ValNa
    Val1b   | Val2b   | Val3b   | ... | ValNb
    .....   | .....   | .....   | ... | ...
    Val1x   | Val2x   | Val3x   | ... | ValNx

    ANOVA will compare the means of all headers in the data file.
    '''
    filename = downloadFileFromJavaFTPServer(dataFile) #where datafile will be the
location of the data on the external machine
    dataValues, headers = du.loadFloatCSVwithHeaders(filename)

    #transpose data here, such that each row will be the data points for each heading
    #this simplifies the passing of the data in to Stats.f_oneway()
    np.transpose(dataValues)
    fStat, pValue = Stats.f_oneway(*dataValues)

    def writeOut(f, p, filename):

        with open(filename, 'w') as f:
            f.write("FValue" + ',')
            f.write("PValue" + '\n')
            f.write(str(f) + ',')
            f.write(str(p) + '\n' )
        f.close()

    newFileName = "/home/ali/Results/" + dataFile #using dataFile again as a
convenient filename
    writeOut(fStat, pValue, newFileName)
    #output will be in the form:
    # FValue | pValue
    # Fstat  | pval

    return newFileName #passing the file name back up so that the main Flask code can
handle sending the file back to Java

def ttest(dataFile, testVer):
    filename = downloadFileFromJavaFTPServer(dataFile) #where datafile will be the
location of the data on the external machine
    dataValues, headers = du.loadFloatCSVwithHeaders(filename)

    #transpose data here, such that each row will be the data points for each heading
    np.transpose(dataValues)
    if not len(dataValues) == 2:
        print "ERROR-Input is invalid"
        return filename
    sample1 = dataValues[0]
    sample2 = dataValues[1]

    if testVer == 1:
        tStat, pValue = Stats.ttest_ind(sample1, sample2)
    if testVer == 2:
        tStat, pValue = Stats.ttest_ind(sample1, sample2, equal_var=False)
    if testVer == 3:
        tStat, pValue = Stats.ttest_rel(sample1, sample2)


    def writeOut(t, p, filename):

        with open(filename, 'w') as f:
            f.write("TStat" + ',')
            f.write("PValue" + '\n')
            f.write(str(t) + ',')
            f.write(str(p) + '\n' )
        f.close()

    newFileName = "/home/ali/Results/" + dataFile #using dataFile again as a
convenient filename
    writeOut(tStat, pValue, newFileName)

    return newFileName #passing the file name back up so that the main Flask code can
handle sending the file back to Java
```