MIT Department of Electrical Engineering and Computer Science

Undergraduate Thesis (6.UAP)

Building a Computer Program for Visualizing and Debugging Bayes Net Machine Learning Models

Amy Apostol

Supervisors: Henry Lieberman, Yen-Ling Kuo

**Abstract**

Machine learning using big data has become an increasingly popular technique in the past decade for solving complex problems and developing AI systems. However, writing successful machine learning programs and algorithms is an inherently complex task and usually takes years of studying before one can expertly write these programs. The vision behind this project is to assist novice machine-learning engineers in creating machine-learning programs by building an application that will help them better understand, visualize and debug their programs. *I plan to do this by first researching what techniques programmers may find useful when building their programs, developing the specs for a program and then writing/building the application.* The goal after the application is built is to have programmers use the application to successfully debug their machine learning models and feel that they have a better understanding of how their model works.

**Chapter 1: Introduction**

Section 1.1: Need for Better Machine Learning Tools

Machine learning using big data has increasingly become a widely-used technology for solving complex problems and making predictions. However, understanding how machine learning works usually takes years of studying and practice (in the form of obtaining a Master’s Degree or Ph.D.). Nevertheless, because of the large demand for machine-learning engineers, current software engineers are taking on the tasks of building these machine learning programs without significant training in the field. In order to bypass the time it takes to study and understand these machine learning models, software engineers often use previously developed source code or theoretical algorithms from research papers to build their machine learning programs. But because of this lack of knowledge concerning the inner details of a model, when a machine learning program does not function as expected, these software engineers have a difficult time understanding and debugging the issue. Other than reading machine learning research papers and using previously written source code modules (like tensorflow or sklearn), there do not exist a lot of tools that could help new machine-learning programmers quickly get up to speed with understanding the inner workings behind the technology. With machine learning progressing so rapidly, it is not practical for every available software engineer to spend years researching the subject before they can begin writing machine learning programs for production.

Section 1.2: Project Goals

We believe that machine learning and artificial intelligence will shape the future of our world and soon be integrated into every aspect of human life. Thus, we feel a need to make machine learning and artificial intelligence accessible to everyone, including those who do not have the time or resources to enroll in a five-year Ph.D. program. The goal of my thesis work is to help beginner machine-learning programmers better understand the principles of machine-learning and better debug their machine-learning code by creating an application that will allow users to visualize and interact with their machine learning models. After several months of research and work, I have developed *VISUALearning: Bayes Net Debugger*, an application that allows users to construct a Bayes net machine learning model by drawing a direct acyclic graph representation of the model. Users can then test the accuracy of their Bayes net model by running actual testing data through the model, and seeing if the model produces the same results as what was expected.

Section 1.3: Thesis Overview

Throughout the course of the paper, I will describe the preliminary research and ideas that my team and I developed that contributed to the creation of *VISUALearning:* *Bayes Net Debugger*, describe the process of building the application, go over feedback I received from potential users, and finally describe next steps for the project.

**Chapter 2: Previous Work**

Section 2.1: Rapid Serial Visual Presentation and Building Sentiment Analysis Mock-Ups

Rapid Serial Visual Presentation (RSVP) is a technique created with the intention of allowing humans to read at faster speeds. The human brain can process information much faster than the eye can move across a page. Thus, if the eye is not forced to shift positions and refocus every time it reads a new word, then theoretically humans would be able to read a lot faster. RSVP works by quickly displaying words one by one at a single focal point so that humans do not have to move their eyes across a page.

In most machine learning applications, the machine learning models are trained using extremely large datasets (usually upwards of 10^3 data points, often 10^6 data points). It is impractical for a human to read through all the data points in a training set, yet it would be helpful for the user to have an idea of what’s in the dataset when constructing their model. RSVP could be a reasonable tool to help users quickly read through the training dataset. We could also take this a step further and somehow display the machine-learning model’s classification/analysis of each data point. Thus as the user is quickly reading through all their data, they can start to get an idea of which data points the model is classifying correctly and which points are being classified incorrectly. From there the user can localize potential locations for the bug in the model.

Because RSVP is for reading, the first type of machine learning problem we built an RSVP data visualizer for was a sentiment analysis problem, one that has sentences of English words as the training data. I built a sentiment analysis machine learning model using source code from the Natural Language Toolkit (NLTK). The model takes in an English sentence as the input and classifies whether it has positive sentiment or negative sentiment. When using NLTK, one does not need to know any of the behind the scenes work that goes into calculating the sentiment; one simply sets parameters for training (like learning rate, number of training examples), passes their training data through a “train” function to compute the probability distributions, then passes a real example through a separate “calculate sentiment” function to get results. A user with little machine learning experience would not be able to debug their model if the testing accuracy is not as high as desired.

When a machine learning model does not perform as expected, the places one may suspect the issue comes from are the training data or the parameters set during training. Understanding how the model views the data may help the user determine which aspect of training the user needs to change. For example, if the model repeatedly views the word “apple” as a negative term because the training data only uses that word in negative sentences, but “apple” should be contributing to positive sentiment, then the user can hypothesize that the data being used to train the model is the issue. In this case, it would be helpful to have an application that displays the data and the model’s analysis of each data point to the user in order to help the user debug.

As a result, I built a sample java application that displays each data point to the user using RSVP. The application will change the color of a word based on how the model views that world. If the model sees that word as one that contributes to positive sentiment, then the java application will display that word in green; if the word is one that contributes to negative sentiment, then the application will display that word in red. Additionally, the java application displays the most informative words (words that most heavily contribute to determining the sentiment of the sentence) in aqua blue. With this application, users will be able to gain a better understanding of how their data contributes to the functioning of the machine learning model.

From creating this sample java application, we learned that allowing the user to visualize their data could be a huge asset when debugging machine learning models. RSVP is a great potential tool for creating this visualization.

Section 2.3: Bayes Net Application Goals

Writing programs to perform sentiment analysis makes use of a very comment type of machine learning model: Bayes Net. Bayes Nets are a simple, yet extremely versatile machine learning model and are often one of the first types of machine learning model a beginner programmer learns about. Thus, my team and I decided to build a fully functioning debugging application for Bayes Nets. Ideally this application would accomplish two main goals: allow the user to (1) **visualize their training data** and (2) **visualize their machine learning model**. The most common visual representation for a Bayes Net model is in the form of a direct acyclic graph, with each node in the graph representing a feature of the training data and edges connecting dependent features. With our application, a user would be able to draw a visual representation of their Bayes net model. Once the model has been drawn, the data will be displayed to the user by changing the color of each node if the feature representing that node is true for a specific data point. This is our version of RSVP for data that is represented by vectors.

**Chapter 3: *VISUALearning - Bayes Net Debugger* Application**

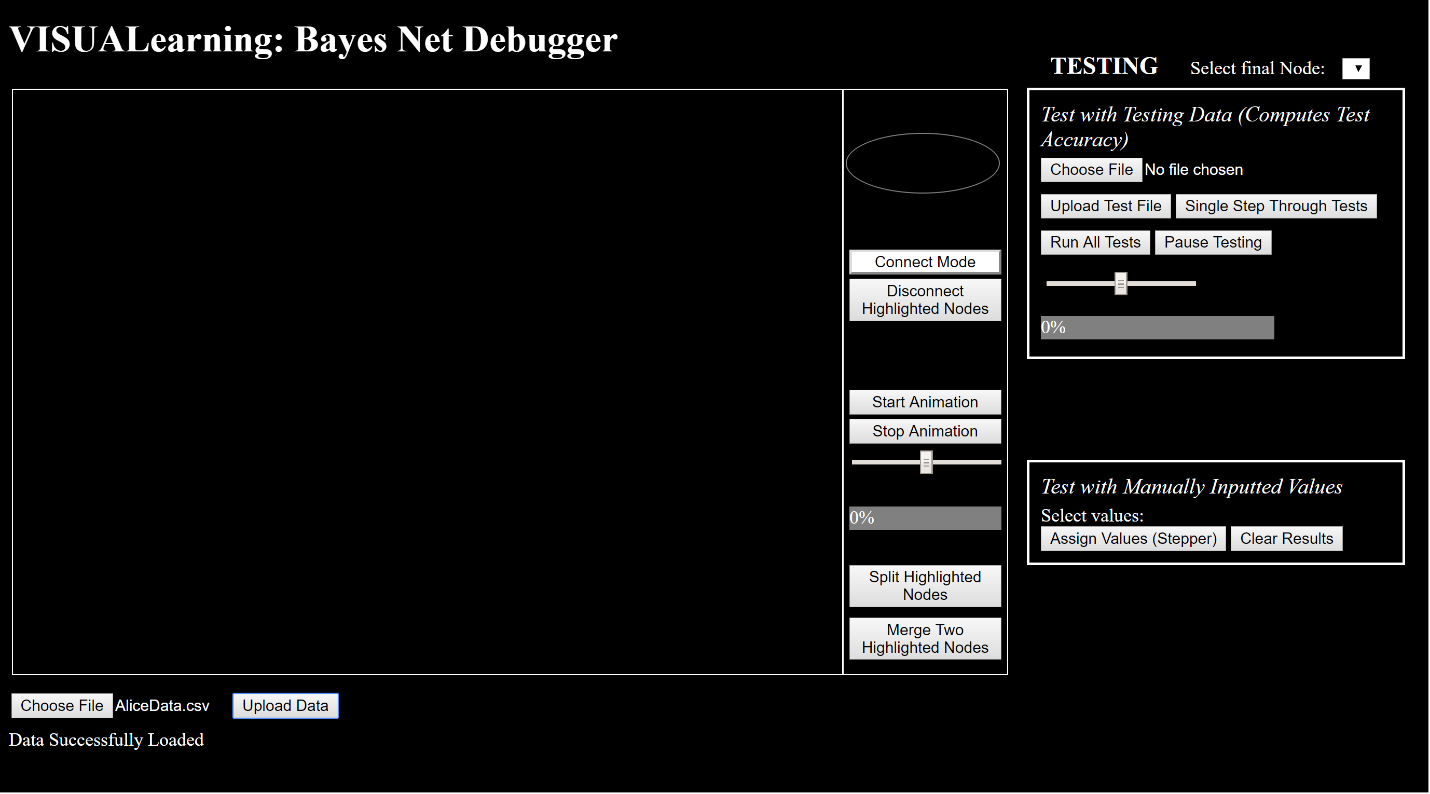
*Bayes Net Debugger* is an application that provides users with a platform to draw a graphical representation of Bayes Net machine learning models and perform a series of actions on the graph in order to determine the best graphical representation for a given problem. *Bayes Net Debugger* also makes use of RSVP in order to present the data to the user. Once a graph is built, there are several actions the user could perform in order to better understand their model and test for model accuracy.

Section 3.1: Description of *Bayes Net Debugger*

The program assumes that the data used to build their Bayes Net graph is in the form of a csv file. Each line of the csv file represents a separate data point, while each feature is represented by the separate columns within each line. The data values within each data point must be true/false values (1=true, 0=false). If some of the features hold numerical values, the user must reformat the data to be true/false values. To do this, for example, the user could specify all values greater than 50 to be equal to true and all values less than or equal to 50 to be false.

The first step when building the graph is for the user to upload the csv training data file to the *Bayes Net Debugger* program. Uploading this file will allow the program to be familiar with the features within the dataset as well as store all the data points and their values. Once the data has been uploaded, the user can begin to build their graph.

Figure 1 (below) shows the initial interface after the user has uploaded the training data. It contains all of the tools necessary for the user to start building a Bayes net graph. The left-most large empty box is the space in which the graph will be drawn, called the drawing space. Next to the drawing space there is a dock containing tools used to construct and study the Bayes Net graph; we will reference this space as the tool box. And finally, furthest to the right there are two boxes containing testing units; this is what the user will ultimately use to test if they have the best representation for their model.



**Figure 1: An example of the *Bayes Net Debugger* interface.**

Section 3.1.1 Tools to construct a graph

*Bayes Net Debugger* has five different functions that allow the user to construct a Bayes net graph: *Create Node, Connect Nodes, Disconnect Nodes, Split Nodes* and *Merge Nodes.* Each feature from the data is represented by a different node on the graph. Dependency between two features is shown by a directed edge from the parent feature to the child feature. To create a node, the user clicks and drags on the oval shaped item at the top of the dock of tools into the drawing space.

Nodes can be connected to one another by entering “Connect Mode”. When in connect mode, the program asks the user to specify a parent node (by first clicking on the parent) and then a child node (by then clicking on the child). After the user has specified the parent and the child, the program will draw a directed arrow from parent node to child node. On the backend, when this connection occurs, the program calculates conditional dependencies for the child node: based on the training data, *Bayes Net Debugger* calculates the probability the child node is true given the parent node is true and the probability the child node is true given the parent node is false; this is done for each of the child node’s parents.

The user can disconnect nodes from one another as well. To do so, the user must highlight two connected nodes by doing Ctrl+click on each node. Once the two desired nodes are highlighted, the user clicks the button “Disconnect Highlighted Nodes” in order to remove the connection. The program only allows the user to specify two nodes to disconnect each time the “Disconnect Highlighted Nodes” button is pressed. If a connection does not exist between the two highlighted nodes, the program will ask the user to go back and specify two nodes where there does exist a connection. On the backend when a disconnection occurs, the probability that the child node is true is recalculated given the new set (or non-existing set) of parents.

The user also has the ability to split and merge existing nodes. To split a node, the user must highlight the node, and then click the “Split Highlighted Nodes” button. To merge two nodes, the user must highlight two nodes and then click the “Merge Two Highlighted Nodes” button. The user may only merge two nodes at a time. The order in which multiple nodes are merged together does not affect the probabilities calculated for the final node. Splitting nodes could be helpful towards improving the accuracy of the Bayes Net Model because it localizes the feature which most contributes to the outcome of the child node. For example, if you use the feature “Cold Weather” to predict whether or not a student will walk to school, splitting the node into “freezing weather” and “mildly cold weather” may give a better prediction. This is because the event that the student will walk to school varies less when you only consider “really cold weather”. To get the best performing Bayes Net model, the goal is to get the probability that a child node will be true as close to 100% or 0% as possible because this creates a more definitive model that is less likely to predict the wrong outcome by chance. Meanwhile, merging nodes could also be helpful because you could improve the probability of a child event being true to 100% or 0% based on how the two parent nodes interact with one another. For example, you may get a better outcome if you merge the events stomachache and fever into one “sick” node rather than keep them separate.

Section 3.1.2 Understanding the Data by Animating Nodes

Before the user makes a connection between two nodes, it would be helpful for them to examine the data so they could have some understanding of how two features might contribute to a final outcome. For example, if the user notices that one feature seems to always be true when another feature is true, then the user can assume there is a dependency/connection between the two nodes.

Using a form of RSVP, we can quickly look through the data point values for the entire dataset. To do so we will loop through the data points, and for every highlighted node, if that node’s particular feature is true, the node will flash a non-white color. If all highlighted nodes are true, then all the nodes will flash to a green color. To animate the nodes you want to see data for (where animate means to make nodes flash their data value), highlight those particular nodes. Then select the button “Start Animation” to begin looping through the data. The user is able to specify the speed of the animation by moving the slider below the “Stop Animation” button. The further to the right the slider is, the faster the animation will loop through the data. The grey progress bar below the slider will start to change green and display the percent progress of looping through the data as the program moves forward.

Section 3.1.3 Testing

There are two different methods of testing that will help the user understand how their model classifies the final feature and if the constructed model is an accurate representation of what actually occurs: testing with manually inputted values and testing with testing data.

Testing with manually inputted values allows the user to specify a scenario and see how the model calculates a true/false value for each event. The process begins when the user assigns a value of “True”, “False” or “Not Given” to each node. Next, the program finds the highest node (node with the smallest amount of ancestors) with no assigned value (i.e. a value of “Not Given”) and randomly assigns a value of “True” or “False” to that node based on the probability that the node has a value of true. Probabilities are calculated when a node is first created and every time a node is connected or disconnected to or from another node. This process of finding the highest unassigned node and giving it a value repeats until we assign a value to the final node, which is ultimately the event we want to get a value for. Figure 2 shows an example of how this process works.



**Figure 2: The progression of testing with manually inputted values.**

In this situation, we are trying to determine whether or not a plane will land at the airport. We assume that whether the plane lands depends on if there is bad weather today, and that bad weather today is influenced by whether bad weather occurred yesterday. First, the program sets the value of “Bad Weather Yesterday” to true because we want to know the probability of the plane landing when this situation occurs. The next step, the program will retrieve the probability of “Bad Weather” occurring given that there was bad weather yesterday, which is 79%. Since the probability value is 79%, the program will select a value for the “Bad Weather” node with a 79% chance of the value being true. Finally, the program will calculate the probability that the plane does not land given the value of “Bad Weather” (which was set to true in the above example), and then sets a final value to the “Plane Doesn’t Land” node.

With this tool, users will be able to understand how a Bayes Net model goes through and calculates the final value for a situation by getting probabilities from parent events. In addition, it will allow the user to assess how well their model predicts an outcome. If the user sees that the probability of an event occurring is closer to 50%, then they can hypothesize that the parents of the node are incorrect or the training data is bad as it is not creating probabilities that we would expect for our model.

The second tool the user has for testing is the Testing with Testing Data module. This value is how the user will ultimately assess how accurate their model is with predicting real-world situations; the higher the test accuracy, the better the model. The program computes the test accuracy by looping through a set of test data points. Given the parent events that occurred in each testing data point, the program sets those particular parent values on the graph and then computes the value for the final event. It then compares the predicted value of the final event to the actual value of the final event specified in the testing data. If the prediction and the actual value are the same, the testing accuracy increases; else, the testing accuracy decreases.

The user runs this test by first uploading the testing data to the program. Then the user has the option of letting the tests all run automatically or single stepping through each test data point to see what the model produces. If the user decided to run all tests at once, they can specify the speed of the tests with the slider, and even choose to pause the test once it has begun.

Section 3.2: Program Code Architecture

*VISUALearning: Bayes Net Debugger* is written in html, CSS and Javascript. The html and CSS code is used to position and style all of the elements displayed by the program, while the Javascript is used to keep track of backend variables and perform backend calculations.

The html portion of the code creates all the visual elements necessary for the program to function. This includes the upload training data buttons, the drawing space, the tool dock, and the testing modules. Each of these sections is placed into different sections of the code (to make it easier for the reader to understand). The tools inside the tool dock as well as the testing modules both are not displayed until the training data is properly uploaded. Most elements are styled directly in the html file rather than in a separate CSS file or CSS section of the code. All elements have a style position of “absolute” so that they can be manually (hardcoded) placed at different positions of the interface. Thus, it is possible for html elements to potentially overlap, and it is important for the designer to place all elements in situations where they would not overlap. In other words, most elements do not move dynamically in relation to one another to avoid overlapping.

The Javascript portion of the code is divided into five main sections and is organized similar to an Object-Oriented Program. Section 0 (which is not considered a main section) is composed of global variables. These variables are what keep track of the crucial information that reveals the state of the program. The global variables include a data array, a test data array, an array of features, a list of nodes in the drawing space, whether or not the program is in connect mode, the probabilities associated with each node, the parents and children of each node, and several other variables which help name a new node, remember the parent selected in connect mode, etc.

Section 3.2.1 Uploading the Training Data

The first section of the Javascript file is responsible for uploading the training data to the program. When the training data is uploaded, the training data array and the features array (a list of all the features in the training data) must be populated. All the populating of global variables takes place within the function readDataFile(). Within the readDataFile() function, a color is randomly chosen using the generateRandomColor() function to be associated with each feature. These colors are the ones displayed during the animation of nodes.

Section 3.2.2 Adding, Deleting and Moving Nodes

*Creating Nodes*

A node is created when the user clicks and drags a new node from the oval shape in the tools dock. Each node in the drawing space is associated with a particular ID. When a new node is created, it is given a temporary ID, a temporary name, and the ability to move, highlight, connect and rename that node. If a node does not have a name/ID that is equal to a feature from the training data, then that node will not be able to be animated, nor have any probabilities associated with it. Thus, a node is pointless to have in the graph if the user does not choose to rename it. A user renames the node by double clicking on that node and selecting a new name. When a new name is selected for the node, the ID of the node is changed to be equal to the new name, the node is assigned a list of parents and a list of children, the node name is replaced in the node list and other nodes’ parent and child list, and the option for assigning a value to the current node is created in the “Testing with Manually Inputted Values” section of the testing modules. Additionally, because we do not want two nodes to have the same name, the newly selected node name will not show up as an option for renaming other nodes (unless we choose to delete or rename the current node).

*Deleting Nodes*

Also within this section is a function that will delete a node: remove the node from the board, the feature option from the “Testing with Manually Inputted Values” section, and from the global nodes, parent and children arrays.

*Moving Nodes*

Finally, a node is moved by using the drag event on the html node element. However, when a node is moved, we must also move the line connecting that node to another node, so that the current node remains connected to the graph. This is done using the moveLines(nodeID) function. This function also takes in as a parameter the nodeID of the node that is being moved to a different location.

Section 3.2.3 Animating, Connecting and Disconnecting Nodes

The third section of the Javascript code contains the functions that are responsible for animating, connecting and disconnecting nodes.

The user cannot make connections between nodes unless in “Connect Mode”, so there exists a function that switches the state of the application to and from connect mode. The function that does this is called switchConnectMode(). This function changes the state of the program and keeps certain actions not related to connecting nodes from being executed. When connecting nodes, the program needs to draw the directed edge between the two nodes on the front-end and update global variables on the back-end. Each node that is being connected is added to the others respective parents list and children list. The connectNodes(parent, child) function is responsible for updating the global variables; it also calls the function drawSVGLine(parent, child) which draws the arrow between the parent and child node on the front-end interface.

The disconnectNodes(parent, child) button does the exact opposite of connectNodes and drawSVGLine: it removes the parent and child from the other node’s respective parent and child list, updates the probabilities for the child node and removes the arrow from the front end. The function disconnectNodesButton(event) handles the event and all that must occur when the user clicks the “Disconnect Highlighted Nodes” button.

Section 3.2.4 Splitting and Merging Nodes

Section 4 of the Javascript code handles all front-end and back-end operations that occur when the user wants to split a node or merge two nodes. The functions splitNodeButton(event) and mergeNodeButton(event) both handle what to do when the user clicks the “Split Highlighted Nodes” or “Merge Two Highlighted Nodes” button; both functions call their respective splitNode and mergeNode functions when called by the button press.

The splitNode function performs four different operations. (1) It first creates two new nodes and places them in the drawing space to the left and right sides of the original node. (2) All the parents of the split node will be added to the list of parents for the two newly created nodes. (3) All the children of the split node will be added to the list of children for the two newly created nodes. (4) The old node is deleted from the board and all of the global arrays containing information for that node. When a node is split and two new nodes are created, the two new nodes are not assigned valid names (name of an existing feature). Thus they will not be able to contribute to predicting a value for the desired feature. The user must rename the two newly created nodes; once the nodes are renamed to a valid feature of the data, probabilities are calculated for each of the two nodes’ probability of being true.

The mergeNodes function does almost the exact opposite of the splitNode function. It first creates a new node and places it in the drawing space in the middle of the two nodes to be merged. (2) The function next adds all the parents of the two nodes to be merged to the parent list of the new node. (3) The children of the two nodes to be merged are all added to the child list of the newly created node. (4) The original two nodes are deleted from the front-end and back-end representation. Like with splitNodes, no probability calculations for the new node occur until the node is given valid name (one of the features that exists from the training data file).

Section 3.2.5 Computing Node Probabilities and Running Tests

We finally move into discussion of the last section of the Javascript program, which is responsible for computing the probability that the value of each node/feature is true and running the testing modules.

The whole premise of Bayes nets are that they calculate a true or false value for a situation based on probabilities determined by previously occurring events. The machine-learning model for each node in the direct acyclic graph must calculate the probability that the node (or event associated with the node) will be true. The computeProbabilities function is responsible for getting the probability that each node is true based on conditional dependencies (parent nodes). The function loops through all the nodes in the drawing space. If the name of the node is a valid feature from the training data, then a probability is calculated for that node. Probabilities are calculated by counting the number of times that node’s particular event is true. Dependencies on other events (or parent nodes) are taken into account by only looking at data points where the features match the dependency value. For example, if we want to look at the probability a plane lands given there is bad weather, then we would only count the data points that have a “Bad Weather” value of True.

Next we will discuss the Javascript code for running the “Test with Manually Inputted Values” module. In this module, all that is necessary is for the program to assign a true/false value to each of the existing nodes. The getResults(event) function does most of the work. It works by (1) first writing the true/false values specified in the given forms of the module. Once all manually inputted values have been assigned, the function goes through all unassigned nodes and randomly selects true as the value with the node’s calculated probability. The getResults function will only assign a value to one node at a time (it steps through the data). Thus, the user will need to continue pressing the “Assign Values (Stepper)” button until all the nodes have been assigned a value. When an unassigned node is given a value, the application also displays the probability that the event is true. In order to remove the displayed results from the displayed results from the node, the user can press the clearResults() function. The function works by accessing every node’s INFO div element and deleting the text within the div element.

Finally, there exist several functions that provide functionality for the “Test with Testing Data” module of *Bayes Net Debugger*. The readTestFile() function reads the testing data csv file and populates the testData array; it works very similarly to the readDataFile() function. The runTests(event) function does most of the work necessary to compute the testing accuracy. runTests(event) only calculates the accuracy for one data point at a time, and adds that information to a global test accuracy variable. It works by using the given features (the features which are ancestors of the final variable we want a value for) to make a true/false prediction about the final node’s event occurring. It then gets the actual value about whether or not the event occurred from the data point. The function compares the predicted value with the actual value; if the two values are the same its adds 1 to the count of data points that had a correct prediction (testCorrect variable) and adds 1 to the total count of data points reviewed (testCounter variable). If the predicted and actual value are different then no value gets added to the testCorrect variable, but a value of one still gets added to the testCounter variable. Finally, the function computes and displays the testing accuracy by dividing testCounter from testCorrect.

When the user wants to single step through the testing data points, the user pushes the “Single Step Through Tests” button, which calls the runTests(event) function. If the user wants to run through all tests at once, they would press the “Run All Tests” button, which calls the runAllTests(event) function. At a high level, this function works by looping through all of the unchecked data points in the test data and calling runTests for each new data point. The user has the ability to halt the testing process by clicking “Pause Testing” which breaks the loop going through testing data and keeps track of the place where the testing last left off. The function that pauses testing is called pauseTestingFunc(event) which just changes the state of the application into one where testing cannot occur.

Now that we’ve built the *Bayes Net Debugger* application, we can begin testing on real subjects.

**Chapter 4: User Testing**

Although we did not have the time to run fully planned, formal experiments, we were able to build the program up to a point where it could be informally tested by users. The target audience that could currently use this program is mainly made up of beginner machine-learning programmers who have a couple years of experience coding. At MIT, this mainly consists of computer science students who have taken at least one introductory machine learning or Artificial Intelligence course. Thus we decided to select students who fit this description for preliminary testing.

In order to allow the students to test the *Bayes Net Debugger* we came up with the following scenario: Alice is a freshman at MIT who needs extra tutoring for physics, but the only time the tutor is available to meet is at 8 in the morning on Sundays. Attending sessions at 8 AM may be tough for Alice for several reasons: she often goes out partying Saturday nights and thus goes to bed really late, she is not a morning person and is often too lazy to get out of bed at an early time, she is taking a rigorous schedule at MIT so sometimes she would rather do other work than attend tutoring sessions. Students were asked to use *Bayes Net Debugger* to build a Bayes net graph that would most accurately predict whether or not Alice would attend her 8 AM tutoring session.

We tested two students who matched our audience description. Both students had overall positive feedback to give about the program. The noted that they could see how *Bayes Net Debugger* is useful in helping people understand the meaning behind their machine learning models. Both students also stated that they could envision themselves using this application to understand if they have the correct idea of their model and debug their machine-learning programs. To the students it definitely seemed that there was a need for an application like this within the machine learning community. This positive feedback gives us the motivation and reasoning to continue improving the *Bayes Net Debugger*, perform more formal experiments to test the validity and user experience with this application, and build more *VISUALearning* applications for different machine learning models.

**Chapter 5: Next Steps**

Now that we have constructed a successful initial program that is able to help programmers better understand and debug their machine learning models, there are several directions we can move from here.

The first is to continue improving the *VISUALearning: Bayes Net Debugger* application. We want to make our application as easy-to-use and intuitive as possible. Specifically, the “Testing with Manually Inputted Values” module still seems unintuitive and could be difficult for the user to understand how to use the module or grasp why it is useful. To improve this module, we are thinking about allowing the user to manually set node values by clicking on the node rather than setting the values on a form to the side. It’s important to keep the focus of the application on the graph within the drawing space, and not on external forms. Additionally, we would like to add a refresh button so the user can reset the program when they want to build a new model, and help the user structure their diagram by automatically organizing nodes into a hierarchical structure.

Once we have completely developed a program that is intuitive to use and free from bugs, we can proceed with more formal experiments. We would like to test this program with a large selection of beginner machine-learning programmers and have them give more concrete responses that will help us determine if this program is useful. As a result, we will need to design a more detailed test that asks user to rate different aspects of the program on a scale. This way we can more concretely quantify how users like or dislike the program.

Finally, after we are satisfied with the production of the *Bayes Net Debugger*, we expand *VISUALearning* to other machine learning models. The main components of *VISUALearning* is that it provides a platform for users to visualize their data and machine-learning model. Having this ability to visualize models will ultimately help the user better understand and debug machine learning models. If we can provide this tool for all machine learning models (for example, neural nets, semantic analysis and linear regression models) we can make it a lot easier for new engineers to thrive in the machine learning community.

**Chapter 6: Contributions**

Throughout the several months of working on this project, I have developed two main contributions to the machine learning community: the idea of visual machine learning (building machine learning programs by visualizing training data and visualizing the machine learning model), and the *VISUALearning: Bayes Net Debugger* application.

Visual machine learning is the process in which users are able to build machine learning models by drawing a visual model rather than by writing code. This type of machine learning makes the technology more accessible to the world, as it is not required for one to know code in order to build these models. Users can better build their machine learning models also if they have a better understanding of the data they work with. Incorporating RSVP into building machine learning models provides a way for users to quickly visualize each data point and get a general idea of the shape of the corpus of data.

The *Bayes Net Debugger* application is hopefully the first of the many machine learning debugging applications we will build for *VISUALearning*. With *Bayes Net Debugger*, programmers have the ability to construct Bayes nets machine learning models without it being a requirement to know how to write code, or without knowing all the mathematical details and theory behind the machine learning model. This application ultimately makes machine learning more accessible to all humans. By making machine learning more accessible, we can get more people developing machine learning models and AI systems, which will be used to automate simple processes and help make the world a safer and better place.

**Appendix**

Section 2.1: Researching Probabilistic Programming

Upon beginning this project, we hypothesized that probabilistic programming could be a useful asset to include in a machine learning debugger. Probabilistic programming is a form of programming in which the user manipulates probability distributions to solve a problem rather than manipulating single-number variables. We believe that probabilistic programming will be the main format of programming used for machine learning programs in the near future. Thus we determined that it may be useful to get a head start and build a debugger for probabilistic programs. We also hypothesized that writing a debugger in a probabilistic programming language could be useful in the sense that we may be able to use the language to determine the probability that a specific feature is the issue behind the incorrectness of a machine learning model.

The main probabilistic programming language that I investigated and studied was Anglican, a language extending from Clojure. The website provided many tutorials for the language as well as an environment to run Anglican programs on your own.

Section 2.2: Creating an Eclipse Plugin to Interact with Current Code

The first goals we had for this project were to create a plugin that could be added to the Eclipse integrated development environment. This plugin would be able to read from the user’s code and develop a visual representation for the machine learning model based on the content and structure of the user’s code. Time was spent researching Eclipse plugins, learning how to build then and practicing building several sample plugin applications. Before we could continue building a plugin that could interact with our code to create visual representations of the model, we needed to form a more concrete plan for an intuitive way to visually represent all types of machine learning models.

**Bibliography**