Requirements Specification

GNomes

Members

Jacob Christiansen
Allen Clarke
Yuanyuan Fu
John Jackson

Mentor

Mahsa Keshavarz

Clients

Dr. Toby Hocking
Christopher Coffey

Accepted as baseline requirements for the project:

Client signature	Date
Team lead signature	Date

Version 2.0
December 10, 2019

Contents

1 Introduction

We have very little knowledge of our own genome. Many diseases are caused by mutations in our DNA, but we have no idea how to alleviate symptoms, much less cure or prevent these diseases. If there were a way to determine what genetic diseases (like cancer) someone was susceptible to before the disease has progressed far enough to show symptoms, we could effectively prevent these diseases well before they are life-threatening. This could be possible through the study of our own genome.

The genomics industry is so new and cutting-edge that there are not very many developed tools to help scientists analyze genomic data and understand these diseases. To learn more about genetic diseases, a massive amount of data would have to be analyzed by biologists, charting which genes are active in patients with diseases by hand and establishing correlations between sick patients and their DNA. Those trends would have to be compared to the trends in the DNA of healthy patients to look for more trends. This is a very tedious process and distracts biologists from more creative research, since they are spending days looking over genomic data and writing down trends they see by hand instead of staying on the cutting edge of research and trying to come up with ways to cure or reduce the trends they are seeing. This is where Dr. Toby Hocking is involved.

Dr. Hocking is an assistant professor at NAU and is also the Machine Learning Lab Director. He works with other researchers to develop machine learning algorithms for other fields of study. One of his main fields of interest is genomic research data. Dr. Hocking has already completed multiple research projects looking at DNA data and coming up with algorithms to graph the data in a similar way to how scientists do so now. The goal is to use supervised machine learning to look at genomic data and to bring scientists and statisticians together.

Dr. Hocking wants to bridge the gap between biologists and statisticians because nobody else has done it yet, and doing so will improve the field of work for both parties. This will help statisticians because they have the tools to write complex models to analyze data but are missing new sets of data to look at. Receiving data from biologists will open a new door for statisticians and allow them to analyze more data as well as provide useful insight to scientists. Having new analysis tools will help biologists analyze much larger pieces of data much more efficiently, after having put in some labels about the data for the statisticians to use as a key to compare models against. Overall, the sharing of genomic data is clearly beneficial in uniting both parties' investigations into genomic diseases.

Team GNomes is working with Dr. Toby Hocking to create PeakLearner, a machine learning web app to process genomic data. This new tool will help scientists save time combing through data and allow them to get back to making new medical research breakthroughs by getting biologists data into the hands of a statistician's machine learning algorithm.

2 Problem Statement

Before we can dive into the biologists' problem, we must understand how they currently approach this work. Biologists have been gathering what are called histone concentration charts to help figure out which genes are being expressed in certain types of cells. These histone charts reference histone proteins which are the proteins responsible for gene expression. So if a biologist measures how many histone proteins are in an area, that is potential evidence that the genes in that area are the ones being expressed. When a biologist sees "peaks"—high concentrations of histones over a short area of the genome—this may indicate an abnormality or mutation in the genome. When scientists see peaks in pre-cancerous samples that they don't see in healthy samples, this can establish evidence of a correlation between a peak in one region and a genetic disease. The visual differences in the locations of the peaks is what scientists are interested in researching further.

The following are problems scientists currently face while analyzing peaks in DNA data:

- Scientists have been sequencing data files (such as ChIP-seq and ATAC-seq) that they can not interact with or view graphically.
- There is no standardized tool to mark regions of interest (*peaks*).
  - The existing unsupervised Machine Learning tools are unwieldy because they are either too inaccurate or too hard to use to get accurate data because the parameters are not easy to tune.
- Many people use spreadsheet tables of 0's and 1's to denote where peaks should be in a genome. These spreadsheets are:
  - Hard to maintain.
  - Tedious to fill out.
  - Difficult to interpret since they are not represented graphically.
- To get a complete model of a dataset, a scientist must label every single peak over the entire gene by hand, which is an unrealistic expectation.

  - Currently, scientists visually examine specific regions instead and then determine regions of interest based on what they see in the dataset.

- The current labeling process by hand takes a long time.
- Scientists have better ways to spend their time than labeling hundreds of peaks for a single genome.

Labeling this data by hand is a problem because it takes time, and while a scientist may clearly see patterns in the data, there is no way to take advantage of these patterns in a way that is meaningful to more than just that scientist. Scientists must go through every individual file and mark each region as either significant, or not. They often use spreadsheets filled with 1's and 0's to mark these regions where a 1 denotes a significant region, but this is hardly a simplified view of the data and is difficult to interpret since the data is not represented in any sort of graphical way, and is instead effectively a binary file.

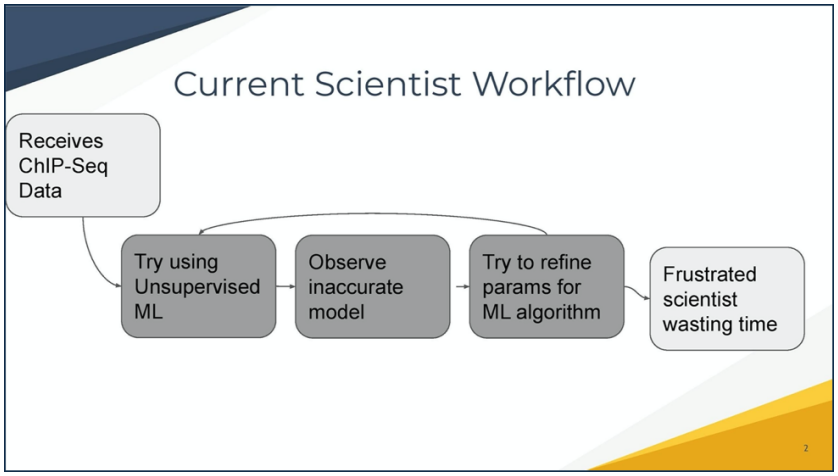

Figure 1: The current peak-detection workflow.

The current cutting-edge labeling system uses unsupervised machine learning, which has clear flaws described above. This process is outlined in **Figure 1**, which shows the current machine learning approach to viewing the all important peaks of these histone charts. As clearly shown in the figure, scientists are wasting lots of time and becoming frustrated with the complexity and inaccuracy of the current system.

Unfortunately, there is no simple, user-friendly way for scientists to interact with this data and act on the peaks they can see so clearly. While some

tools like the UCSC database (a popular genome browser) exist and allow a scientist to upload a datafile and view a graphical representation of the histone concentration, there is no way to add labels denoting which region is significant, and it only serves as a reference point when filling out or updating an Excel sheet.

3 Solution Vision

Ideally, a scientist would be able to view their data with a generated model to denote peaks on top of the data, and be able to interact with the model to correct it in places where it may not reflect something that the scientist sees. This would save time guessing which parameters an unsupervised algorithm might need to be accurate, and provide a graphical representation that the Excel spreadsheet is missing.

Our proposed solution is to address these issues is interactive web interface in which a scientist can upload data, label regions, and generate a peak-prediction model on top of their data via a machine learning algorithm. The scientist would be able to upload their DNA data files and view them graphically on screen. They would also be able to add and remove labels denoting regions of interest on a given genome. As a user enters these labels, our system would return a model drawn on top of the data to predict and denote peaks in the data. This model would save scientists time they would otherwise spend determining a model and also allow them to enter fewer labels. Lastly. a user will be able to download a model once they have finished labeling peaks and can use that model for further research.

The key features of our system include:

- **Ability to upload epigenomic coverage data sets (such as ChIP-Seq and ATAC-seq) as a URL link to a bigWig file.** This allows scientists to view their data digitally instead of in an Excel spreadsheet.
- **Ability to add, modify, and remove labels to mark regions of interest.** This is the supervised part of the machine learning algorithm. It will remove the guesswork and difficulty of handling an unsupervised machine learning algorithm.
- **Informative model to scientists with immediate feedback.** This model will detect peaks and save time by catching peaks not yet labeled. It will be a graphical representation of the Excel spreadsheet that was used in the past.

- **Dynamically updated model.** The model will update after every label change, showing a user how accurate the model is against their labels. This replaces the tuning of the parameters in the unsupervised algorithm, so a scientist never has to start over and can just modify labels to get a new model.
- **Easy-to-use UI.** The UI will be much simpler to interpret than an Excel sheet of 0's and 1's. This will save time so that scientists are able to get their data faster and do not have to deal with the command line system used by previous machine learning algorithms.
- **Similar UI to other genome browsers.** This will entice scientists to use the tool since, while they may already use a genome browser to look at their data, now they can also interact with it.
- **Ability to download model on completion.** This is where a user can download their new model and use it for further tests—something not possible with the old system.

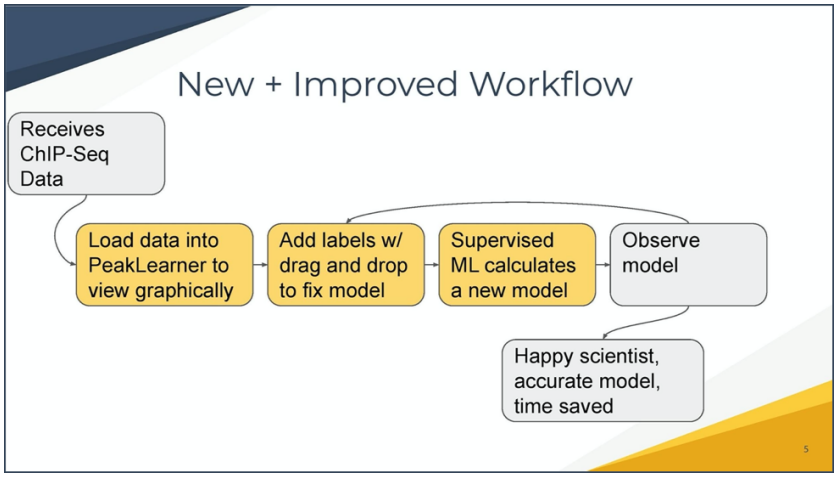

Figure 2: A proposed workflow for PeakLearner.

Our system allows users to mark peaks on genomic data by taking in bigWig files (DNA data) and displaying them visually so that scientists are able to view peaks as graphical data instead of numerical data. Users can click and drag to select a region and label it as needed, like labeling a region as one of the peaks that is clearly displayed visually. We are using a drag-and-drop interface to make the process simple and intuitive for scientists. We are also building this interface on top of an existing genome browser so that scientists

are already familiar with the core functionality behind the display of the data. Currently, these genome browsers are the only way for scientists to view data in a graphical way, but they cannot interact with their data. By adding the PeakLearner system on top of the genome browser, scientists will have the graphical view they are used to with the ability to add meaningful labels and generate accurate models of their data. Combining the visual representation of a genome browser with the power of supervised machine learning will save lots of time when a scientist is trying to analyze their data.

Figure 2 clearly shows a basic workflow that highlights in orange proposed PeakLearner system as a solution. The workflow can be circular as users can iterate the labeling process as much as desired before moving on to downloading a model. This system is designed to save time and make it easier for scientists to interact with their data in a meaningful way. By comparing **Figure 2** to **Figure 1** it can be seen that the proposed solution will be a significant improvement and will help scientists act on the data they are collecting. We are making a difference and changing the world by giving scientists more time to do research and illuminating patterns in data in a largely unknown and cutting-edge field.

4 Project Requirements

In this section we will survey the functional, performance, and environmental requirements of our project. The functional requirements specify what operations our web app should be able to perform. This will be followed by the performance requirements (also known as the non-functional requirements), which are the intended quality attributes of PeakLearner. Lastly we will cover the environmental requirements for the project. These are the constraints put upon the project either by client or by NAU.

We compiled this list of requirements in three distinct ways. The first was through our weekly meetings with Dr. Hocking, where we discussed the functional and performance requirements PeakLearner must meet. With this basis, we began to investigate more specific performance requirements using the developer tools for Mozilla Firefox and Google Chrome. Finally, we went to workshops provided by NAU and read documentation about the code that Dr. Hocking provided to understand the environmental requirements given by our client and the university.

4.1 Functional Requirements

In this section we will consider the functional requirements for PeakLearner. These requirements are broken up into distinct categories to describe what a user can do with PeakLearner. Each use case is an operation PeakLearner can perform and covers one possible scenario a user may encounter. These functional requirements are structured in order of frequency, where the first requirement is the most frequently used and the last requirement is the least frequently used by an average user.

4.1.1 Interactive web interface

PeakLearner will be a web app, and the basis of this webpage will be a software tool called a genome browser. Biologists already use genome browsers to graphically show information about an uploaded genome. Since no current genome browser can both natively show multiple information files on a single "track" and allow for highlighting regions in the way that we need, we will need to augment a browser to show the information that our servers generate. In addition, we will need a login so that we can keep track of specific users' information.

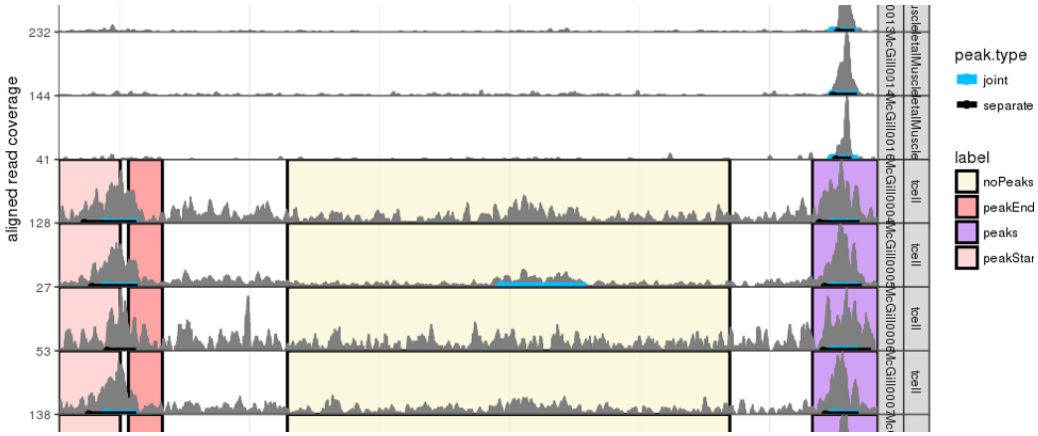

Figure 3: Genome browser tracks with labeled peaks.

4.1.1.1 *Extensions to a genome browser*

For the genome browser we will be modifying, we have chosen JBrowse, which is an open source genome browser available to the public. JBrowse will be providing a large part of the interactivity of our webpage, as it is already a viable web app. JBrowse works by showing a user several "tracks"—sub-windows inside of JBrowse that show graphs of data about a section of an

uploaded genome. In **Figure 3**, the X axis shows a location on the genome, and the Y axis changes from track type to track type. For the type of file we will be using for our specific track type will be a file called a bigWig file, and is an often used file type in bioinformatics for quantitative genomic data. This means for our specific track type the Y axis will be a count of how many of the sequencing data reads are aligned at that position.

JBrowse is built using Perl and Javascript, though for our purposes we need not change any Perl code. This is because of a very open plugin system that allows a developer to write in Javascript and change almost anything about the genome browser. There are two modifications we will need to make to JBrowse. The first is the ability to show multiple files on a single track. This will let us show the bigWig file, the predicted model we calculate, and any labels that the user has provided all on the same track. We know this is possible as we have a plugin for JBrowse on Github that allows multiple bigWig files on the same track, called multiBigWig. We will extend this to alter how each track looks to make clear what is the original sequence, what is the model, and what are the labels.

One challenge we will face is that we can not simply furnish the tools for the information that we want to specifically provide with PeakLearner. JBrowse needs some of the bigwig files to be converted in order to show them effectively. This means that we will need to write additional plugins in order to convert old bigWig formatted files into the newer file format. We will be doing this by using some bioinformatics libraries available for Python and C that already convert between .bw files and .bigwig files.

4.1.1.2 *User login*

Finally, we will implement a login feature so we can keep track of user information separately. To do this, we will simply use the Google authentication API. This will allow us to have a high threshold for privacy for our users, as we do not handle any identifiable information like email or name.

4.1.2 Ability to add and modify labels

The ability for a user to add "labels" to the information is the basis of the machine learning that will generate the peak prediction models. These labels are created by biologists highlighting where peaks are and are not supposed to be. These will be used by our backend server to judge how good a model is by marking if said model aligns with where a user says a peak should be. The one with the lowest error, the least amount out of line with the user labels, will be the model we show. To do this, we need to implement two

things. First, we need a way to show where a user has already added a label as well as differentiating what type of label has been added. Second, we need a highlighting tool for a user to be able to interactively add new labels as well as change existing ones.

4.1.2.1 *Show existing labels*

For presenting the existing labels in our genome browser JBrowse, there is a max value to which the Y axis can be set. For our label bigBed files, the files we will be separate from those storing the genomic data, so we can set the value of any location in a label to any arbitrary amount. For simplicity, we will use "1" to denote a label. This will provide a solid block behind the genomic data showing where known labels are. We will have two of these files: one for if a label is a peak and one for if a label is not a peak. This separation will allow us to change the color of each graph to help differentiate which labels mean what for the user. (For more information about the genome browser JBrowse, see the interactive web interface section of the functional requirements.)

4.1.2.2 *Add and edit new labels*

Next, we need to supply a tool for a user to add new labels and edit existing ones. To do this, we have two options. There is a built in tool for highlighting inside of JBrowse that we could extend. This tool is shown in **Figure 4**, where a yellow label denotes a region where the model should not detect a peak, and the purple label denotes a region where a peak should be detected.

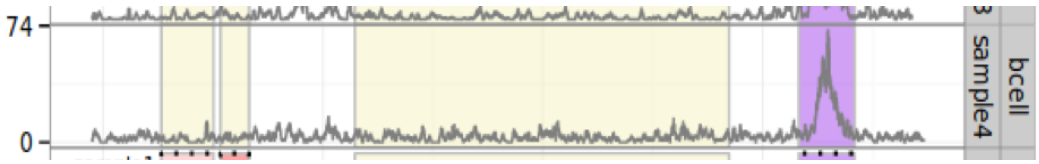

Figure 4: A labeled peak (in purple).

However, Dr. Hocking has put us in contact with JBrowse developer who goes by @cmdcolin on Github. He has written a highlighter plugin for JBrowse that we could use and, with his help, extend what we need, since the sample screenshot does not provide any way to create different label types [1]. This involves a listener waiting for a user to click and drag on the track creating the new label. Followed by a tool to handle the POST command that that listener would throw, adding this label to a back end file for easy retrieval and adding the label to the file to show on the track. To do that

@cmdcolin recommends having a sub-server managing just the highlighter on our main server.

To edit a label when a user starts a new label, we will check to see if it overlaps with any previous label. If so, it will instead change to an edit tool that can switch between deleting a label or toggling it between it labeling a peak or a not a peak label. Users will be able to zoom and move around the genome by clicking and dragging over a different region of the screen where labels are not typically generated, like above/below the model.

Overall, for the ability to add a new label to the genomic data being viewed we will need to do two things. The first is to allow the user to view previously placed labels on top of the genomic data. The second is to let the user quickly add new labels and edit old ones. For this purpose we will extend a highlighting tool that @cmdcolin on Github has created and follow his instructions to create a web server to manage and handle the highlighter tool used by the user.

4.1.3 Dynamic model generation

The ability to dynamically generate models is a critical feature to the system. peak models are simple graphical representations of where peaks should and should not appear along a genome. These models should be regenerated every time a new label is added to make sure the model displayed is the most accurate known model in the system. The dynamic model generation can be divided into two parts. First, we need to be able to pick the most accurate model out of the database based on the error matrix. Next, we need to be able to determine a new lambda value to pass off to Monsoon. This model will need to be incorporated into our error matrix as soon as we get it back from Monsoon.

(The majority of the peak error code exists and has been provided by Dr. Hocking, and will be modified as needed to fit the scope of this project.)

4.1.3.1 *Determine the most accurate already-existing model*

Model-generation needs to be dynamic so that users can see that labelling regions affects their model and creates new models when needed. This will be done by using the error matrix we store in our database to determine the model with the least error with respect to the known labels in the matrix. Then, since we already have the optimal model found, we will fetch the bedGraph file representation of the model and return it to the JBrowse system for display. Lastly, JBrowse will take the model and draw it on top of the existing data and label sets.

4.1.3.2 *Determine a new lambda value to pass to Monsoon*

If a model in the database is not accurate enough for the labels we have, then we need to calculate a new model. To do this, our system will find a lambda value in between two of the best models that we have and pass along the model and the dataset to Monsoon, NAU's GPU cluster. When Monsoon is available, it will calculate the new model for us, and pass it back as a bedGraph file for us to store in the database and in the error matrix. The next time a label is added, we have a new model to test against the accumulated labels and check whether it is optimal.

4.1.4 Upload capability

Scientists need to be able to upload their data to the PeakLearner system. This is one of the most important requirements because if a user can not add data, then they would have no reason to make models or add labels. Clearly, it is imperative that users are able to upload their own datasets into the system.

However, the size of the ChIP-Seq data files that scientists are looking at can be very large, and it would not be very practical to store these large files on the server for every user. The bigWig files that scientists are often looking at can be over 1GB in size per sample. Every scientist is looking at multiple samples, and storing multiple GB of sample data per person is unrealistic.

Our system will allow users to upload multiple bigWig files as URLs to files instead of uploading entire files to the system. Then, users will select the file(s) they want to look at, and PeakLearner will select a small region of the file to load into the browser instead of downloading the entire file to our server, and will continue to display different regions of the file as needed. This will allow the browser to only store the minimal data needed to display a model instead of storing many large files. Scientists will be able to select any of their ChIP-Seq data files to view in PeakLearner and never know the difference. The browser will also send small downloaded segments of files to Monsoon as needed for model generation.

4.1.5 Download capability

The models that PeakLearner generates are useful only in comparison. The power of these models lies in comparing among many samples to identify which genes correlate to which diseases. For example, the model of a sample of someone with cancer is only useful when compared to one that is healthy. To facilitate this easily, we will allow a user to download a generated model as a bigWig and as a bigBed file for storage and future use.

The models that are generated for a file are saved on a per-user basis allowing a user to load any sequence they have previously analyzed. To allow a user to download a model we will have a page that shows the generated models of sequences they have looked at. This function will be achieved by fetching the data from the database on the backend server. This will allow the user to download the most accurate model of the sequence possible.

Overall, we will simply need to supply a page on the website that allows a user to see any model they have referenced in the past. Then, supplying a link for each of these to the database so they can download past models for future reference. The only model a user will care about is the most accurate model, but it will be good to know the most accurate model for each sample individually.

4.2 Performance Requirements

In this section, we will cover our project's qualitative requirements. These include the expected speed, simplicity of operations, and accuracy of model. For each performance requirement, we will give more specific data or operations to achieve the goal. The main focus of this section is to describe how we will accomplish our functional requirements, as well as how fast we expect to be able to do so.

4.2.1 Speed

Speed is the overarching requirement for this project, and with that in mind, we have come up with a few benchmarks that our web interface should meet in order to be deemed "fast enough." The expected overall time should be less than 0.2 seconds between new label input and showing the most accurate model to a user. If a user does not see immediate feedback on their input, they will be disinclined from adding more input to the system, which defeats the purpose of PeakLearner. Given a download speed of at least 70 mb/s, we want to achieve the following goals 90 percent of the time:

- Time to calculate new lambda value should be less than 0.05 seconds
- Time to pass the data to Monsoon should be less than 0.1 seconds
- Time to store lambda value should be less than 0.05 seconds

For the speed requirement in this subsection, we are assuming that the model we are going to display has already been calculated by Monsoon and is on the server in our BerkeleyDB database. We want a new model to be able to

sent from the database and drawn in JBrowse in 0.2 seconds. We have come to this speed requirement as a result of using the Chrome Developer Tools to time various website response times as well as the speed of Dr. Hocking's existing site that generates models on top of dataset to find breakpoints in a genome [2].

4.2.2 Simplicity of operations

An average user may add dozens of labels to a single dataset, making it one of the most important areas to optimize when designing out web application. We will use the following four approaches to simplify the process of adding a label and overall user interface.

- In order to reduce the user's tedious operations, our goal is for a user to be able to **add a label in 3 or fewer mouse clicks**. We envision that a user may have to click once in order to add a label, and then click a second time in order to choose the type of label. Finally, click a third time to confirm the creation of the label. This could potentially be reduced to two clicks by bypassing the confirmation step, but we feel as though that is important to ensure that we do not add incorrect labels that need to be deleted later.

- We may also add the ability to **duplicate existing labels across other samples** to to save the user time and avoid repetitive tasks when possible. If a user is looking at multiple samples, they should be able to easily add a label in the sample place across multiple samples without having to create a new label on each individual sample. By making this process as simple and painless as possible for a user, we will be able to cut down on the amount of time it takes for a user to start seeing accurate data on multiple samples, saving scientists time when they are performing research.

- We need the ability to **customize a distinct color for each label** to help differentiate which labels mean what for the user. We will use a standard color scheme to not overload the page with color, but still be able to easily visually distinguish labels that indicate peaks from those that do not. This color scheme will match that of Dr. Hocking's existing project and use the same .5 alpha transparency [3]. This takes a small amount of time to adjust to on the user end but we believe that it will save them significant time overall as they become familiar with the system. It will also give a user access to more information without having to click or mouse over an existing label to ensure it is correct.

- Lastly, a user must have the ability to **modify existing labels**. We will do this by having a user right click on a label which will essentially make that label active and editable as though the user had never confirmed its creation. This will allow a user to easily edit the label to change its type or delete it if it is incorrect. Changing the label type will change the color of the label and deleting it may pop up a prompt asking for confirmation to reduce the number of labels that need to be remade.

4.2.3 Accuracy of model

Accurate model generation is another critical requirement of the system. Adding labels and using a simple interface is meaningless if a user does not get back an accurate model to fit all of the newly created labels. With that in mind, we have a plan to ensure we are always returning the most accurate model from our database. We are representing accuracy as a number of errors. Since the user is always right, we define an error as an instance when a given model contradicts a label set by the user.

Our goal is to return a model with an error score of 0, meaning the model we return fits all of the labels that we know about. To calculate this score we build three matrices of 0's and 1's where every row represents one label a user has added and every column represents one of the models in the system. The first matrix is the False Negative matrix, which is where we mark models that failed to identify peaks set by the user. A 1 is placed in every cell where a model failed to identify the peak at a certain label position. The second is the False Positive matrix, which is where we label all of the labels that are not peaks, but the model picked them up as peaks. Our final matrix is the total error matrix, which is just the sum of the False Positive and False Negative matrix. By summing down each column of this matrix, we can get the total error score for any given model. We then find the model with the least error and return it to the user to be displayed on top of their data.

4.2.4 Calculating new models

Sometimes there is no model with a perfect error score of 0 in the system. This is where NAU's cluster, Monsoon, comes into play. We will be using Monsoon to calculate new, and hopefully more accurate models for the database should we not be able to find an accurate model already in the system. However, Monsoon is a shared resource and we can not spend hours having it calculate millions of models, so we must be strategic with the data sent to Monsoon to not waste time calculating useless or duplicate models.

It is also important to know that each model has an associated lambda parameter (a positive real number), which is essentially a sensitivity level. For example, a model with a lambda value of 0 will not detect any peaks and look like a flat line, while a model with an infinite lambda value will detect every single datapoint as a peak and look like the teeth on a saw.

To come up with good potential models we refer back to the False Positive and False Negative matrices described in the last section. A model with a high false negative score is not detecting enough peaks, so we need to make the model more sensitive in order to pick up on more of the peaks in the labels given. As lambda increases, the overall false negative score will decrease to 0. Similarly, a model with a high false positive score is too sensitive and is detecting too many peaks, so the lambda value needs to be lowered to not detect as many peaks.

As lambda increases, so does the false positive score, but as lambda approaches 0, the false positive score goes to 0 as well. By observing these two trends and looking at the data we have in our false positive and false negative matrices, we will be able to determine potential lambda values that should be able to minimize both the false positive and false negative score, and thus finding a new optimal model with a small error score to display to the user.

Dr. Hocking has provided code to use as a starting point in the calculation of these matrices.

4.3 Environmental Requirements

In addition to the functional and nonfunctional requirements outlined above, there exist a few environmental requirements imposed either by our client, Dr. Hocking, or by Northern Arizona University. These constraints, including Dr. Hocking's stipulations with respect to testing and documentation, are as follows:

4.3.1 SLURM

Since we need to use Monsoon in order to calculate new models for our datasets, there is no way around using SLURM, Monsoon's job-scheduler. The models we calculate are critical to the machine learning part of the project, since it is very unlikely that we will have a correct model already generated for every dataset. On the other hand, there is no inherent problem with using SLURM to schedule jobs and send our requests for models. With this in mind we are familiarizing ourselves with SLURM in hopes of a seamless integration into the rest of the project.

4.3.2 R

Dr. Hocking has written the PeakSegDisk R package which can compute an optimal peak model for a given data set and penalty parameter [4]. As a result we must integrate all of his existing code into our program as the displayed peak predictions. By doing this we will build a website with a similar look and feel to the existing SegAnnDB site that Dr. Hocking currently has running. Using Dr. Hocking's code is also completely advantageous to us because it is an existing and functional algorithm, and while it may be challenging to integrate it fully, it will be significantly easier than trying to rewrite the algorithm from the ground up.

4.3.3 Google authentication

We are required by Dr. Hocking to implement a login system that uses Google authentication. This will allow a user to log in to the system using their existing Google Account. The use case for the login is so that every user will be able to interact with labels and models independently. This will cut down on users interfering with other users' data and labels, which could have been a significant security risk.

4.3.4 Maintenance requirements

These are our client's specifications for code maintenance, testing, and documentation.

4.3.4.1 *Open-source*

The code will be free and open-source, hosted on a public Github repository.

4.3.4.2 *Testing*

There will be a formal test suite which is run on Travis-CI after every push to the Github repo, composed of the following:

- Tests for the user interface, including a headless browser.
- Tests for the interface with Monsoon. This will be achieved by installing the slurm-llnl Ubuntu package, which gives us a 1-node SLURM cluster for testing [5].
- Tests for the web upload/download API. (E.g. querying a certain download URL results in a certain CSV file.)

4.3.4.3 *Documentation*

The following sets of documentation will be provided:

- Documentation for the internal structure of the code.
- Documentation and command line tools for users who will be loading data sets.
- Documentation for users who will be creating labels via the web app.
- Documentation for users who will download data and labels via the web API.

5 Potential Risks

Now that we have covered the project's requirements, we will consider potential risks to the project's completion and performance, and how we plan to manage them. We will weigh, in order:

- Technical risks (ways the project may not work as anticipated)
- Social risks (threats to our team's performance and cohesiveness)
- Market risk (threats to the product's commercial viability)

5.1 Technical Risks

While we have attempted to draw a technical roadmap for PeakLearner in our Feasibility Analysis document, this project remains a speculative endeavor, and the implementation may not go as anticipated. Components of the project that may cause challenges and delays include **JBrowse modification issues**, **problems achieving the desired speed**, and **cluster outages**.

5.1.1 JBrowse modification issues

Table 1: An evaluation of the risk of JBrowse complications.

Risk	*Severity*	*Likelihood*
JBrowse modification issues	Serious	Low

In preparation for our tech demo, we've been looking at how to modify JBrowse to show additional information. We need it display the data, the

labels, and a continuously rendering model all on the same track. Since being unable to do so would compromise the look and feel of the project, we rate the severity as "serious."

However, we're aware of other JBrowse-based projects who've done something similar, so we know it is achievable. And fortunately, Dr. Hocking has put us in touch with a GMOD researcher who can guide us through the process of extending JBrowse. And, if all else fails, we can take our chances with a different genome browser. For those reasons, we rate the likelihood of a critical failure in this part of the project as "low."

5.1.2 Desired speed unachievable

Table 2: An evaluation of the risk of speed failure.

Risk	*Severity*	*Likelihood*
Desired speed unachievable	Tolerable	Low

Speed is our project's most important performance requirement, and we aim to achieve a speed of 0.2 seconds for model generation. A slower UI would be suboptimal, but it wouldn't disrupt the essential functioning of the program, so we rate this as "tolerable." In choosing our framework and database, we selected for speed above all else, so we rate the likelihood of this risk as "low."

5.1.3 Monsoon down for maintenance

Table 3: An evaluation of the risk of Monsoon unavailability.

Risk	*Severity*	*Likelihood*
Monsoon down for maintenance	Tolerable	Moderate

Monsoon is sometimes down for maintenance, and if this were to happen when we needed it for testing, it would cause a delay. We rate this risk as "tolerable" because these periods are short and those of us on the mailing list have advance warning. Given that maintenance is routine for the cluster, we rate the likelihood as "moderate." We would reduce this risk by making sure our server code works regardless of the up or down status of the cluster.

5.2 Social Risks

In addition to technical risks, there exist social risks—intra-team threats that may harm our team's cohesion and effectiveness. Here we focus on the threat of diffusion of responsibility.

5.2.1 Diffusion of responsibility

Table 4: An evaluation of the risk of diffusion of responsibility.

Risk	*Severity*	*Likelihood*
Diffusion of responsibility	Serious	Low

To control the risks associated with losing a member, we have pursued a strategy of redundancy, i.e. assigning at least two members to each component of the project. Duplicating competencies, however, is a solution with a potential drawback: when more than one person is responsible for a component of the project, it may be unclear who is ultimately accountable for that portion. This would lead to complacency and reduced achievement, so we rate the severity of this risk as "serious."

We can reduce this threat by clarifying our individual tasks during group meetings and following the protocols set forth in our team standards document, so we rate the likelihood of this risk as "low."

5.3 Market Risk

Ordinarily, in this section we would consider threats to the product's commercial viability. But, since this project is being developed for academic researchers rather than a competitive market, there is no market risk we need consider here.

6 Project Plan

In this section, we will go over our project execution plan. Overall, we have 8 milestones and they are:

1. team creation
2. meet with client to understand the requirements
3. set up the database and cluster

4. create user login interface
5. configure the highlighter tool
6. implement the dynamic model generation
7. implement upload capability and download capability
8. integration and final testing

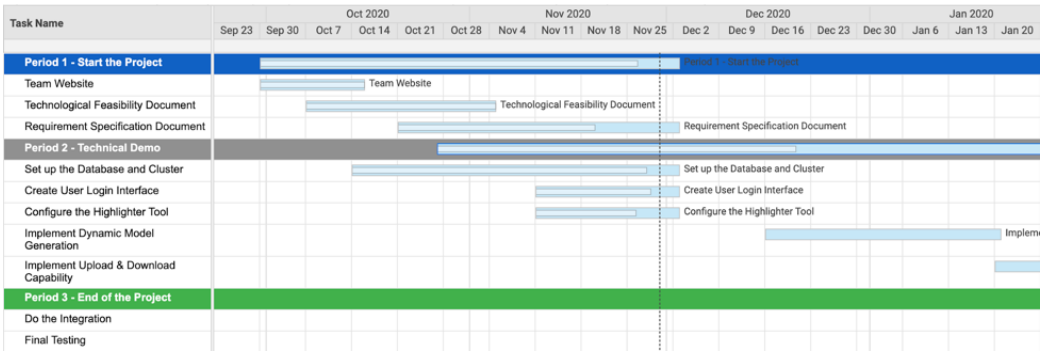

Figure 5: Part 1 of Gantt Chart.

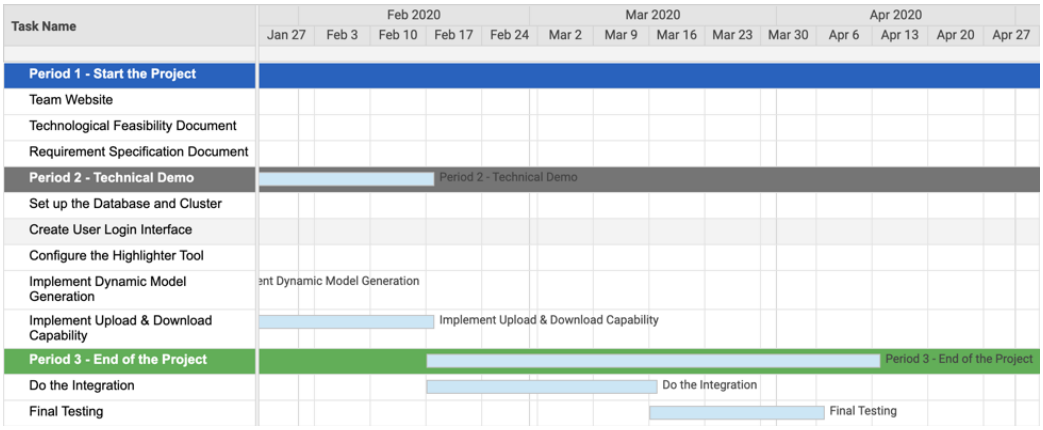

Figure 6: Part 2 of Gantt Chart.

In the past two months, we have had several meetings with our client Dr. Hocking to learn more about our project and begin working on our technical demo. For now, we are working on setting up the database and cluster, creating the user login interface and configuring the highlighter tool. After the winter break, we are going to work on the implementation of dynamic

model generation and upload and download capability. By the end of next semester, we will do the integration and final testing to make sure everything is done and prepared to client review. **Figures 5** and **6** show a Gantt chart with our predicted schedule.

By the end of this semester, we will have completed half of the items in Period 2. During the next semester, we will focus on the implementation of the rest of the project and do the final testing. In addition, there may be some documents that need to be completed in the upcoming semester and we should prepare for these accordingly.

7 Conclusion

PeakLearner will be a tool that simplifies the workflow of biologists studying genomic data. It will be able to generate peaks, predicting what areas of a sample genome are being used the most. Currently, the only way for a biologist to do this work is by hand, going through millions of genes in a chromosome and marking peaks in an Excel spreadsheet. This is a hindrance to biologists for two reasons. The first is that this is a long and time-consuming process, and the second is that it is difficult to compare these data sets between samples. PeakLearner will simplify this workflow by being interactive and using machine learning to accurately predict peaks in the data using only a small number of user-generated labels.

To simplify this workflow, PeakLearner must be user-friendly, interactive, and fast. To ensure user-friendliness, we will extend a tool already used in industry called a genome browser. A genome browser takes in a sequence of DNA and shows it graphically to a user. Since this is a tool already used in industry, it will allow biologists to easily transition to our product without needing to adjust to an entirely new interface. The second requirement is to be interactive, an improvement on the old workflow which demanded a researcher work with a genome browser and Excel sheet at the same time in separate windows. To achieve this interactivity, we will provide a highlighter tool to the user, allowing the user to simply click and drag on a peak to label it. These labels train the algorithm that generates the models of where the other peaks might be in the sequence. This whole process must be quick; from making a label to seeing a new model must take much less than a second according to Dr. Hocking. Inputting one new label will almost immediately create a model that shows where the other peaks are, and each new label raises the accuracy of the model. If we create a product that is user-friendly, interactive and fast, it will simplify the workflow of biologists by greatly reducing the time between opening a new sample and seeing a model of where

the most-used genes are in the sample.

This will not be easy, however, as there are several risks to be contemplated moving forward. The first of these concerns JBrowse, the genome browser upon which we are building PeakLearner. It may be impossible to extend JBrowse in the manner we want. To mitigate this risk, we have chosen previously built plugins for JBrowse that are very close to what we will ultimately need, extending them to tailor-fit PeakLearner. The second risk is that the speeds required by Dr. Hocking are not possible for this project. However, Dr. Hocking has already created a similar tool called SegAnnDB that was able to achieve his requirements. With PeakLearner we will be using NAU's computer cluster Monsoon, so the likelihood that the speed will fall below that of SegAnnDB is low. Overall, there are few risks for PeakLearner, and the ones that exist have clear solutions to mitigate or pivot if they were to occur.

PeakLearner is a tool that will simplify the workflow of biologists studying genomic data. In so doing, it will help further our understanding of our genome and help develop new gene therapies. Through our meetings with Dr. Hocking, we are confident in our understanding of what this product must be able to do. We have gone over the risks of the project and have developed plans and pivots should any of the most likely or disastrous issues arise. As the Gantt chart in our Project Plan section shows, we have a plan for the rest of the project to make sure nothing is forgotten and we do not fall behind. Overall, we are confident that with this information we have compiled, we will be able to create a product that allows biologists to have a greater understanding of our own genome.

Glossary

bedGraph : a plain text file format used in genome browsers for quantitative/real-valued data. 11, 12

bigBed : a binary indexed file format used in genome browsers for qualitative/categorical data. bigBed files are created initially from BED files, a plain text file format used in genome browsers for qualitative/categorical data. 10, 12

bigWig : a binary indexed file format used in genome browsers for quantitative/real-valued data. 5, 9, 12

ChIP-Seq : a method for analyzing protein interactions with DNA. 5, 12

genome browser : a software tool for graphically displaying genomic data. 6, 8, 9, 19, 22

genomic data : the result of sequencing one or more samples using a sequencing method (like ChIP-Seq); usually in bigWig file format. 22

histone : a type of protein involved in gene regulation. 5

label : a graphical annotation marking regions of interest, including peaks. 5, 9, 10, 11, 12, 14

peak : an area of increased activity in a genome. 3, 4, 5, 9, 11, 14, 15, 16, 17, 22

track : a sub-window in a genome browser displaying one sample. 8, 9, 10, 19

References

[1] C. Diesh. wigglehighlighter. `https://github.com/cmdcolin/wigglehighlighter`, April 2019. [Online; accessed 10-December-2019].

[2] T. Hocking, A. Williams, and A. Shrivastava. SegAnnDB. `https://github.com/tdhock/SegAnnDB/`, January 2019. [Online; accessed 10-December-2019].

[3] T. Hocking. PeakError. `https://github.com/tdhock/PeakError/blob/master/R/PeakError.R#L104`, June 2017. [Online; accessed 10-December-2019].

[4] T. Hocking. PeakSegDisk. `https://github.com/tdhock/PeakSegDisk`, July 2019. [Online; accessed 10-December-2019].

[5] T. Hocking. PeakSegPipeline. `https://github.com/tdhock/PeakSegPipeline/blob/master/.travis.yml`, September 2019. [Online; accessed 10-December-2019].