Classification of Wild type and Mutant Zebrafish Brains via Computational Method

true
true
true
true

Abstract

Classification of biological creatures' phenotype has long been a field that scientists study at. In this study, we used support vector machine to classify neurons winthin Zebrafish's brains into two groups with respect to their structures by using data generated from landmark analysis [1]. We created a tool for scientists to classify three-dimensional biological shapes into two groups, usually defined as wild type and mutant, and to understand which parts of the shapes have the most impact on the classification result. This project derives from Professor Barresi's biological image analysis research at Smith College [1].

Introduction

This study derives from Professor Barresi's biological image analysis research at Smith College and provides a tool to classify structures of neurons winthin Zebrafish's brains via support vector machine. Our goal is to classify three-dimensional biological shapes into two groups, wildetype and mutant.

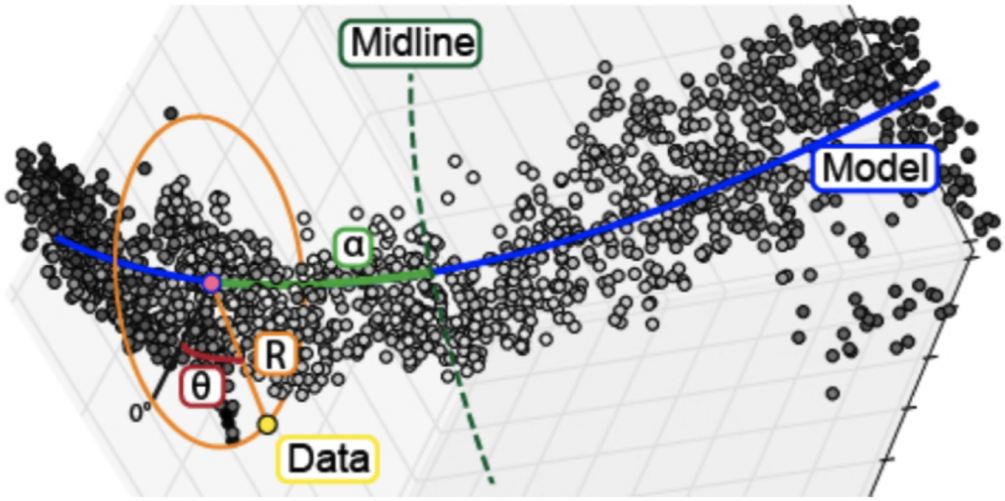

Figure 1: Landmark Data of A Zebrafish Sample

We received the raw dataset from Schwartz, a student in Professor Barresi's Lab, who used optimization to divide the three-dimensional shape of Zebrafish's brain into small wedges and computed the landmark, the most representative point, of each wedge [1]. The data points representing signals in a Zebrafish's brain is shown in **Figure 1**.

Programming Languages

Two programming languages are used in the creation of the calssification tool: Python and **R**. Python is used to run support vector machine models and output the results in correct format. **R** is used generally for data cleaning and creating interactive user interface.

Git, `knitr`, and Reproducible Research

Reproducible research and open source are important in this study. As scholars place more emphasis on the reproducibility of research studies, it is essential for us to make our code publicly available for people to recreate both the model and the user interface.

Github and `knitr` [2] were used in this project to make the study reproducible. Github was used to store the scripts of our final paper and the source code of our final product which contains the final model and the user interface. We used an **R** package called `rticles` to write this paper. This tool allows authors to create reproducible and dynamic technical scientific paper in **R** Markdown. It also allows users to embed **R** code in a scientific paper and output it into a PDF. `rticles` helps users to efficiently put together scientific papers with similar formats [3].

Literature Review

Research in developmental biology has relied on the analysis of morphological phenotypes through qualitative examination of maximum intensity projections that surrender the power of three-dimensional data. Statistical methods to analyze three-dimensional visual data are needed to detect subtle biological phenotypes [1].

Landmark Analysis

Landmarks describe a shape by locating a finite number of points on each specimen. There are three basic types of landmarks: scientific, mathematic and pseudo-landmarks. A scientific landmark is a point assigned by an expert that corresponds between objects in some scientifically meaningful way, for example the corner of an eye. Mathematical landmarks are points located on an object according to some mathematical or geometrical property of the figure. Since it does not assume a preference of one location to another, it is particularly useful in automated morphological recognition and analysis for under-studied structure. Pseudo-landmarks are constructed points on an object, located either around the outline or in between scientific or mathematic landmarks. It is often used to approximate continuous curves [4]. This research has chosen to calculate an automatic set of landmarks distributed across the structure in order to avoid introducing bias due to expectations about where biological differences should emerge.

Schwartz et al. [1] have used the open source program, Ilastik, which employs a training based machine learning, to eliminate the image noise. Then they preformed principal component analysis to align commissures between samples, reducing misalignment artifacts, and implemented a cylindrical coordinate system which preserves image dimensionality normally lost in maximum intensity projection (MIP), which facilitates presentation of the data, but sacrifices much of the complexity and relational data contained in the image. Then they reduced the points identified by the program as belonging to the structure to a set of landmark points that describe the shape and distribution of signal corresponding to the structure. Finally, using the landmark system, we are able to identify and quantify structural differences and changes in signal distribution between wild type and mutant commissures.

Support Vector Machine

Schwartz [1] used random forest machine leaning method to classify the landmarks. Although the classification is quite accurate, it is difficult to interpret the result from biological perspectives. Instead of doing classification on a subset of the landmarks at the same time, we decided to do classification on one landmark at a time via support vector machine. The SVM algorithm is a classification algorithm that provides state-of-the-art performance in a wide variety of application domains, image classification. During the past few years, SVM has been applied very broadly within the field of computational biology especially in pattern recognition problems, including protein remote homology detection, microarray gene expressions analysis, prediction of protein-protein interactions, etc. In 1999, Jaakkola et al [5] ushered the development of homology detection algorithms with a paper that garnered the “Best paper” award at the annual Intelligent Systems for Molecular Biology conference. Their primary insight was that additional accuracy can be obtained by modeling the difference between positive and negative examples. Because the homology task

required discriminating between related and unrelated sequences, explicitly modeling the difference between these two sets of sequences yields an extremely powerful method.

Data and Variables

Landmark Data

Schwartz used optimization to divide the three-dimensional shape of Zebrafish's brain into small wedges and computed the landmarks [1]. The shape in **Figure 1** is divided into 30 circular slices, and each slice is further divided into 8 wedges. The landmark of each wedge is calculated by taking the median distance of all points in each wedge, r. We use the number of points in each wedge and median r to run SVM models to run classifications.

We have 43 wild types samples (n_1) and 35 mutant samples (n_2) for training and testing. There are 152 landmarks (N) for each sample, which resemble **Figure 1**, with each of them containing the following variables:

- number of points in each wedge
- median r (micro-meter): the median of the distances to the center of the slice of all the points in each wedge
- α (micro-meter): distance from the center of the landmark to the midline
- θ (radian): the degree the describes the location of a wedge within each slice

We used the number of points and the median r to do classification via support vector machine. For missing median r values due to absence of points in particular wedge, we filled them with two times the median value of all the points in that wedge across all samples.

Tidy Data

The raw landmark data is a wide table containing sample index and all columns holding information regarding to the minimum and maximum values of α and θ, number of points, median r value, and the true type of each sample. The value in each cell refers to either median r value or number of points. Since all variables names in the raw dataset, such as `-14.29_-4.76_-0.79_0.0_50_pts` and `-14.29_-4.76_-0.79_0.0_50_r`, are long and hard to interpret, we decided to restructure the raw dataset so that the new dataset havs sample index, minimum and maximum α, minimum and maximum θ, number of points, median r, and type of sample each as its own columns.

Three key functions in `tidyr` R package [6] were used: `gather()`, `separate()`, and `spread()`. `gather()` separated the dataset into key and value pairs for each sample index. The key was the column name that contain essential information that describes the position of each landmark wedge, such as `-14.29_-4.76_-0.79_0.0_50_r`, and the value is either the number of points or median r value of each landmark wedge. `separate()` divided the keys collected from `gather()`, such as `-14.29_-4.76_-0.79_0.0_50_r`, into 5 different columns, `min_alpha`, `max_alpha`, `min_theta`, `max_theta`, and `ptsOrR`. The `ptsOrR` variable describes what type of information each cell contains. This was added to the result of `gather()` that contains the sample index and either median r or number of points of each cell. Afterwards, `spread()` widened the already wide table by expanding the `ptsOrR` column by creating two columns, one column representing median r and one column representing the number of points.

Dealing with Missing Values

Support Vector Machine (SVM) cannot throws away observations with missing values. For wedges that do not have any points, median r cannot be calculated, which means that information collected from these wedges are eliminated when running SVMs. Wedges without points have biologically meanings, so we should not ignore these wedges in the SVM model. In order to get rid of the missing values, we need to artificially pick a value to replace the missing median r values. From a biological perspective, the reason that there is no point in a particular wedge might be that the data points are too far away from the scanned area to be captured. Therefore, we decided to calculate the mean of median r of both wildetype and mutant samples, and then we replace the missing median r values with $2 \cdot$ median r for each landmark according to the true type of the samples to show that the points in this wedge are too far away from the scanned area to be captured.

Support Vector Machine

The goal of SVM is to find a separation line $f(x) = (\beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2)$ that separates the data points as cleanly as possible as shown in **Figure 2** below. The optimal parameters β are found by solving the optimization problem –to maximize margin subject to certain restrictions [7].

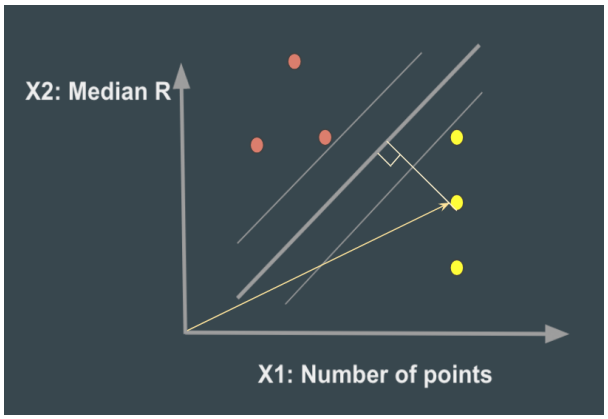

Figure 2: SVM model

$$\begin{aligned} &\textstyle\sum_{i=1}^{n} \beta_i^2 = 0 \\ &y_i \cdot (\beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2) \geq M(1 - \varepsilon_i) \\ &\varepsilon_i \geq 0 \\ &\textstyle\sum_{i=1}^{n} \varepsilon_i \leq C \end{aligned} \tag{1}$$

- M: Margin is the sum of distance of the two closest points from each class to the separation line.
- ε_i : slack variable.
- C: Tuning parameter, toleration of total misclassification.

The slack variable ε_i tells us where the i^{th} observation is located, relative to the separation line and the margin. If $\varepsilon_i = 0$ then the i^{th} observation is on the correct side of the margin. If $\varepsilon_i > 0$ then the i^{th} observation is on the wrong side of the margin. If $\varepsilon_i > 1$ then it is on the wrong side of the separation line.

The tuning parameter C bounds the sum of the ε_i, and therefore determines the number and severity of the violations to the separation line and margin that we will tolerate. We can think of C as a budget for the margin can be violated by the n observations. If $C = 0$ then there is no budget for violations to the margin, and it must be the case that $\varepsilon_1 = ... = \varepsilon_n = 0$. For $C > 0$ no more than C observations can be on the wrong side of the separation line, because if an observation is on the wrong side of the separation line then $\varepsilon_i > 1$. As the budget C increases, we become more tolerant of violations to the margin, and so the margin will widen. Conversely, as C decreases, we become less tolerant of violations to the margin and so the margin narrows.

y_i is the vector representing the coordinate of a data point. The dot product of y_i and the function of the separation line gives the perpendicular distance from the data point to the separation line:

$$y_i \cdot (\beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2)$$

If the dot product is greater than 0, the observation falls at the right side of the separation line and vice versa.

In general, we want to find a classification that the distance from a data point to the separation line is larger than the margin, while we can tolerate some points being in the middle of the margins or misclassified.

Cross-Validation

For our project, we have access to 43 wild-type samples and 35 mutant-type samples. Due to this limited sample size, we decided to use a leave-one-out cross validation method to test our model. For each testing sample, we built 152 SVMs: one for each landmark. For each SVM, we used 10-fold cross validation to select a tuning parameter C value among 0.1, 1 and 10.

Final Product: Two-Step Interactive Classification Tool

We created a two-step interactive classification tool which allows users to simply input a data file and get an visualization of the modeling result. There are two main components in the tool:

General Workflow

Figure 3 displays the overall workflow leading to the final product of classification. It begins with the cleaning and restructuring of the raw landmarks data. The tidied landmarks data that has the NA values filled contains 152 distinct landmarks. Each landmark has its own representative SVM model that used the 77 samples out of 78 for training from applying the leave-one-out cross validation method. That one sample that is left out from each landmark is used for testing the model that was built from the 77 samples. Then, based on the training accuracy of each model, only those that consist of reliable landmarks are filtered out. The number of wild types and mutants predicted with each model are calculated and this leads to the final classification result of using SVMs.

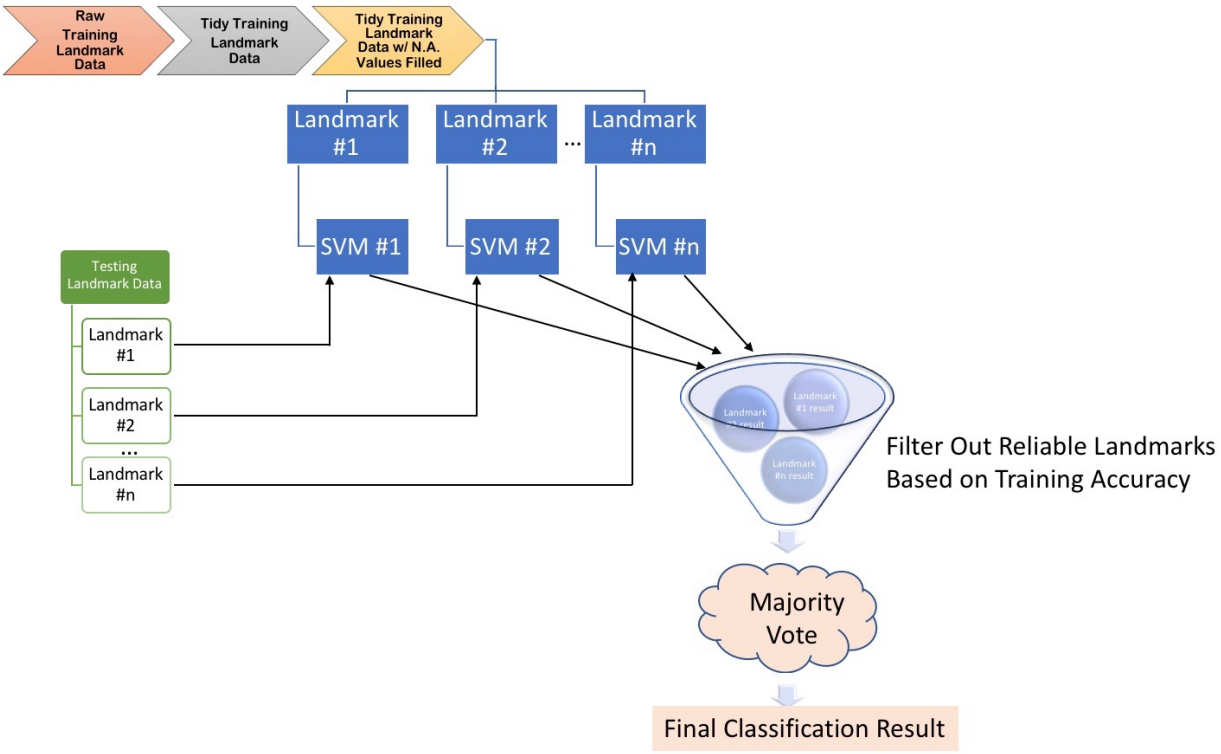

Figure 3: Summary of Workflow

Step One: Data Processing and Modeling

This step is implemented using Python (version 3) and packages including `pandas`, `numpy` and `sklearn`. Users would need to run and interact with the Python script `svm.py` to preprocess the data and build the model. Then, they would need to run another script `analyse_results.py` to analyse the raw results and produce aggregated results.

The script `svm.py` contains two components: a general-purpose `svm_classification()` function that builds a SVM model to classify points for a particular landmark and a `main()` function that runs the `svm_classification()` function for each landmark.

The script `analyse_results.py` contains three components: a helper function `get_result_file_pathes()` that returns a list of result file paths in the output folder; a helper function `process_row_data()` that takes a path of raw data file and returns one with accuracy scores calculated and attached; a `main()` function that runs the two helper function to process all raw data files and generates an aggregated `CSV` file containing results from all samples.

User Interaction

As shown in **Figure 4** and **Figure 5**, several user inputs are taken from users when they run the python scripts.

```
Dejias-MacBook-Pro:SDS-Capstone-Zebrafish dejiatang$ python3 svm.py
Please enter 'AT' or 'ZRF' to indicate channel interested: AT
Enter 0 for filling NaN values with median and 1 for filling with 2*median: 0
Please enter a VALID sample index: 101
Please enter result file name: r101.csv
```

Figure 4: Example of User Interaction in Step One of the User Interface

```
Dejias-MacBook-Pro:SDS-Capstone-Zebrafish dejiatang$ python 1.CodePython/analyse_results.py
Please enter raw results' folder name: 5.output_AT_2med
Please enter result file path: test.csv
```

Figure 5: Example of User Interaction for Running svm.py

Input File

Input file must contain landmark data. Variables that are needed for classification are required to be included in the input file. In our analysis, we used number of points in each wedge within the 3D shape and the median r of points in each wedge.

Sample input file

	sample_index	pts	r	stype	landmark_index
4475	128	6251	4.151089	wt-at	58
7062	141	8414	9.091117	wt-at	91
11637	114	0	31.192271	wt-at	150
627	103	1	2.088584	wt-at	9
3570	326	866	14.366965	mt-at	46

Figure 6: Sample Data Input File of First Step of the User Interface

Figure 6 shows a random sample of an input file for the script `svm.py`. The columns `stype` (sample type), `landmark_index` and `sample_index` are required and the other two columns (`pts` and `r` in this example) are the two parameters used for building SVM.

Output File

Figure 7 shows the columns of the final result file produced by the `analyse_results.py` script. The first four columns describe the testing sample's information while the rest of the columns are all precision statistics describing the accuracy of the built model. The descriptions of all of the columns are displayed as follows:

- `sample_index`: sample index of the testing sample
- `landmark_index`: the index of the landmark
- `type`: the actual type of the testing sample
- `pred`: the prediction made by the model

```
sample_index          12008 non-null object
landmark_index        12008 non-null int64
type                  11856 non-null object
pred                  12008 non-null object
type0_0               12008 non-null float64
type0_1               12008 non-null float64
type1_0               12008 non-null float64
type1_1               12008 non-null float64
type0_precision       12008 non-null float64
type0_recall          12008 non-null float64
type0_f1              12008 non-null float64
type0_num             12008 non-null float64
type1_precision       12008 non-null float64
type1_recall          12008 non-null float64
type1_f1              12008 non-null float64
type1_num             12008 non-null float64
overall_precision     12008 non-null float64
overall_recall        12008 non-null float64
overall_f1            12008 non-null float64
```

Figure 7: Sample Data Output File of First Step of the User Interface

- `type0_0`: number of type 0 samples that are classified as type 0 in the corresponding landmark's SVM
- `type0_1`: number of type 0 samples that are classified as type 1 in the corresponding landmark's SVM
- `type1_1`: number of type 1 samples that are classified as type 1 in the corresponding landmark's SVM
- `type1_0`: number of type 1 samples that are classified as type 0 in the corresponding landmark's SVM
- `type0_precision`: percentage of samples that are classified by the corresponding landmark's SVM as type 0 are really of type 0
- `type0_recall`: percentage of type 0 samples that are classified by the corresponding landmark's SVM as type 0.
- `type0_f1`: harmonic mean of `type0_precision` and `type0_recall`
- `type0_num`: number of type 0 samples in the training dataset
- `type1_precision`: percentage of samples that are classified by the corresponding landmark's SVM as type 1 are really of type 1
- `type1_recall`: percentage of type 1 samples that are classified by the corresponding landmark's SVM as type 1
- `type1_f1`: harmonic mean of `type1_precision` and `type1_recall`
- `type1_num`: number of type 1 samples in the training dataset
- `overall_precision`: weighted average of `type0_precision` and `type1_precision`
- `overall_recall`: weighted average of `type0_recall` and `type1_recall`
- `overall_f1`: weighted average of `type0_f1` and `type1_f1`

Step Two: Interactive Visualization Tool

After building SVM models in step one, we insert the output from the SVM models into step two to visualize the results. Steps two uses the accuracy scores output from step one to create a user-friendly app which generates visualizations to help users to understand the SVM results.

The repository containing the shiny app can be accessed by doing the following:

```
install.packages("devtools")
devtools::install_github("liwencong1995/SDS-Capstone-Zebrafish")
```

The file containing the source code of the shiny app can be found in `9.FinalModel` folder of the repository. The file is named as `shiny_app.R`.

Input 1: Data File and Variables

Input `CSV` data file must be stored in a folder called `data` under your working directory, and the `CSV` file must be named as `output_data.csv`. If you do not know what your working directory is, you can check it by using the function `getwd()` in base **R**.

All SVM models from step one produce the following 9 accuracy measurements:

1. Precision score of type 0
2. Recall score of type 0
3. F1 score of type 0
4. Precision score of type 1
5. Recall score of type 1
6. F1 score of type 1
7. Overall precision score
8. Overall recall score
9. Overall F1 score

These 9 accuracy scores are the variables needed in the second step of the user interface to create the visualizations.

Input 2: User Inputs

Users can select `channel` and `sample index` to filter the input dataset to only keep the observations that users are interested in.

In addition, users can set the threshold of the following variables:

- Overall precision score
- Overall recall score
- Overall F1 score

The dataset used to create the visualizations is rendered every time users change one or multiple thresholds. Our app filters out the observations that do not fulfill the threshold requirements and uses the remaining dataset to update the histograms and heat maps.

Output: Interactive User Interface

This interactive user interface was built upon several **R** packages:

- `dplyr` [8]
- `data.table` [9]
- `ggplot2` [10]
- `shiny` [11]

We visualize the 9 accuracy scores by using both histograms and the corresponding heat maps that display the scores included in the histograms in rectangular shapes that are colored with different shades of blue according to their magnitudes. The positions of the shapes are determined with respect to their relative positions within the biological structure. In the study of Zebrafish, we used the relative positions of the wedges used in landmark analysis to determine the position of the wedges in the heat map.

There are 10 tabs included in the user interface of the app: 1 Accuracy Threshold Summary tab and 9 accuracy score visualization tabs.

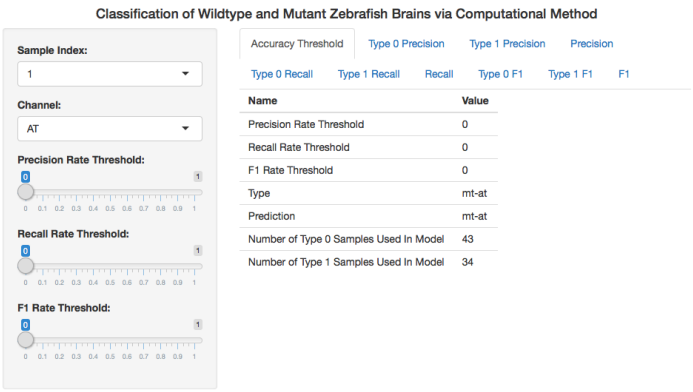

Figure 8: User Interface: Accuracy Threshold Summary Tab of AT Channel

Figure 8 displays the Accuracy Score Threshold Summary tab of the first sample of AT channel. Users can drag the dot on the slidebar to set the thresholds of overall precision, recall, and f1 scores. The threshold of the three scores are updated in the summary table. Default thresholds are 0 for all three accuracy measurements. We then use the landmark observations that fulfill the threshold requirements to predict the type of the sample of choice by doing a majority vote. We simply count the total number of landmarks that are classified as type 0 and type 1, and then we determine whether there are more of them that are classified as type 0 or type 1. The type that gets more vote is the predicted type of the sample. The resulting predicted sample type is also updated in the summary table.

Other information, such as the true type of the sample and the number of wild types and mutants used in training the SVM models are also included in the summary table.

Figure 9 displays the Precision Score Visualization tab of the first sample of AT channel. In this case, all three thresholds are at default level, 0. All landmarks' precision scores are shown in both the histogram and the heat map.

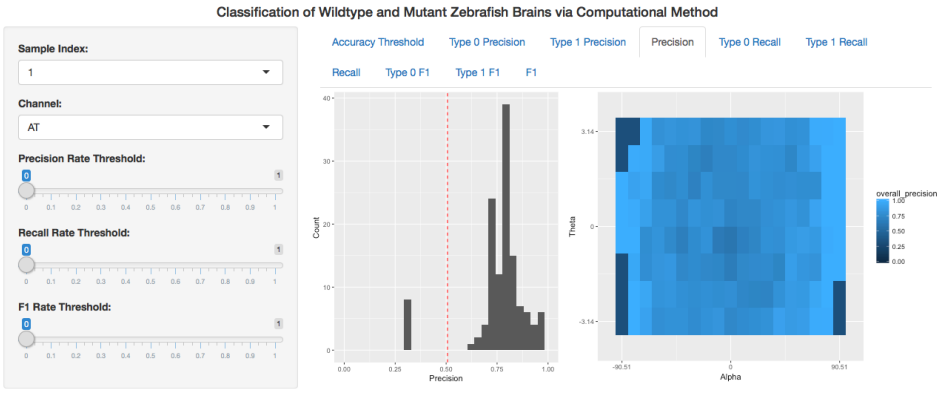

Figure 9: User Interface: Precision Score Visualization Tab of AT Channel

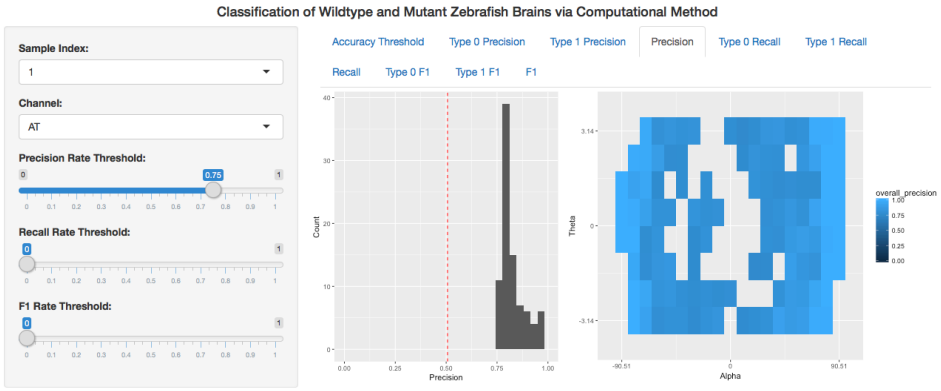

Figure 10: User Interface: Precision Score Visualization Tab of AT Channel, with precision threshold = 0.75

Figure 10 also displays the Precision Score Visualization tab of the first sample of AT channel. In this case, recall and f1 scores' thresholds are at default level and precision threshold is set to be 0.75. Therefore, only landmarks that have precision scores that are equal to or greater than 0.75 are shown in the visualizations. As shown in the histogram, all values less than 0.75 are removed from the histogram in **Figure 9**. Some of the blocks in **Figure 9** are turned into blank blocks after the precision threshold is increased to 0.75.

Users can also choose to observe the SVM results of ZRF channel. **Figure 11** displays the Precision Score Visualization tab of the first sample of ZRF channel with all thresholds equal to 0. More sample visualizations of other accuracy scores can be found in Appendix A.

Conclusion and Discussion

Strengths

Our final product has several strengths:

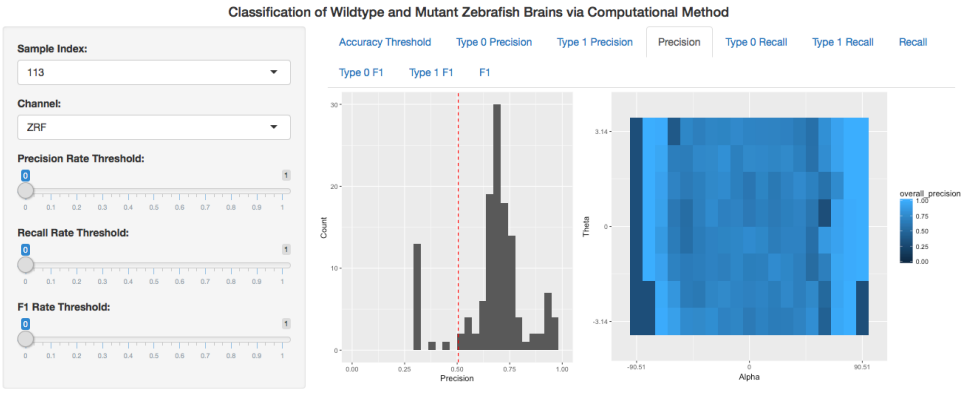

Figure 11: User Interface: F1 Score Visualization Tab of ZRF Channel

Easy Interpretation

In the previously used random forest method, the number of predictors p exceeds the number of samples. Schwartz [1] applied the Principle Component Analysis to reduce the dimension of the predictors. The problem with reducing the dimension is that the parameters gaining weights at last are linear combinations of the original landmarks. While the patterns of the first several projections still make sense, the minor projections are very random and thus difficult to interpret.

The SVM model run on each landmark data gives insightful analysis of which landmark, or which part of the Zebrafish brain, has more power in making predictions.

Implementing User Feedback

We have implemented feedback from users in our user interface. Originally, our shiny app only produced visualizations of one channel's data, but we added an additional variable, `channel`, for users to analyze three-dimensional data with more or multiple channels. Because of this improvement, it is more convenient for users to compare and contrast results from different channels.

Limitations and Improvements

Interaction Between Channels and sections

Our SVM model only make prediction based on a single channel's information and the SVM is run for one single landmark at a time. It does not consider the interactions between channels and between different sections of the sample.

Iterating Machine Learning

Instead of cross-validation, iterating machine learning method could achieve better results. In iterative machine learning, we repeat the process of training and testing several times. In the first round, the user gives examples of objects belonging to specific classes and the machine learning algorithm is trained with this data. In the second round, the algorithm shows examples of objects it

thinks that belong to these classes. Now, the user merely adds objects to the improved training set which the machine learning algorithm has put into a wrong class. That is, the user only corrects the "misunderstandings" of the algorithm. In this way, we can concentrate on difficult examples of objects that are hard to classify. Such objects may lie close to the decision boundaries or in the periphery in the multidimensional feature space. This iterative process is continued until the machine learning algorithm does not make any mistakes or the classification results do not improve anymore. It will improve our classification results and thus is likely to help make better predictions for unknown type [12].

Future Study

Interaction Between Channels

If more time is given, we could add factors that describe the interaction between channels into our SVM model in order to combine information from multiple channels to predict sample type.

Improving User Interface

Since we have only received feedback from three users at Smith College, we would love to receive more feedback from other scientists and improve our model and user interface accordingly.

Acknowledgements

This project was completed in partial fulfillment of the requirements of SDS 410: SDS Capstone. This course is offered by the Statistical and Data Sciences Program at Smith College, and was taught by Professor Benjamin Baumer in Spring 2018.

Appendix A: Shiny App Accuracy Score Visualizations

Recall Score Visualization tab of the first sample of AT channel

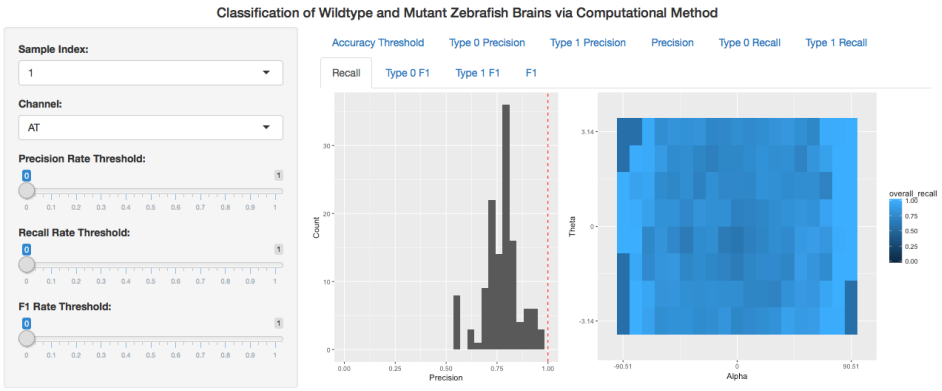

Figure 12: User Interface: Recall Score Visualization Tab of AT Channel

Recall Score Visualization tab of the first sample of AT channel with recall threshold equals to 0.75

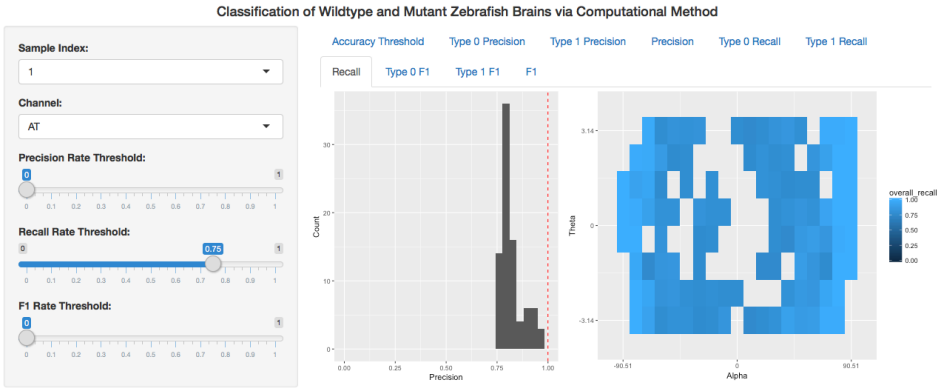

Figure 13: User Interface: Recall Score Visualization Tab of AT Channel, with recall threshold = 0.75

F1 Score Visualization tab of the first sample of AT channel

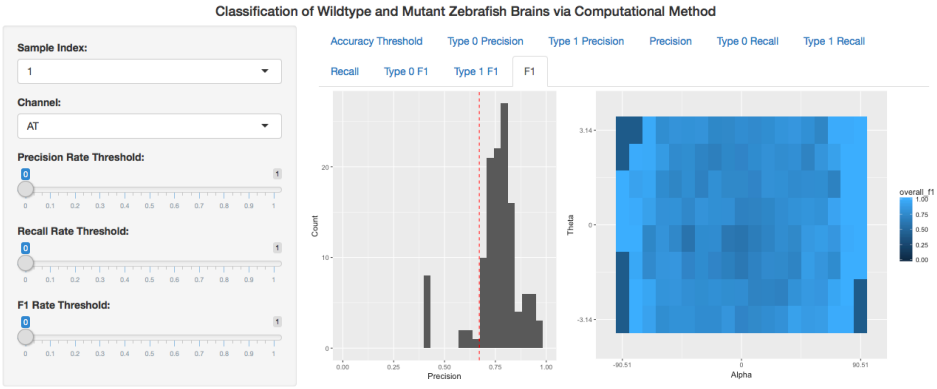

Figure 14: User Interface: F1 Score Visualization Tab of AT Channel

F1 Score Visualization tab of the first sample of AT channel with f1 threshold equals to 0.75

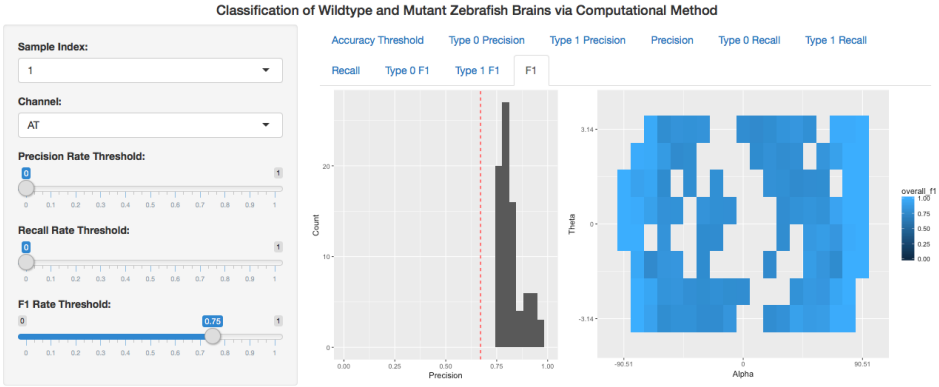

Figure 15: User Interface: F1 Score Visualization Tab of AT Channel, with f1 threshold = 0.75

Appendix B: Source Code for User Interface

Support Vector Machine

```
import pandas as pd
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import f1_score, precision_score, recall_score

'''
A function that builds a SVM model with linear kernel to classify points
to two classes.

Inputs:
training_landmarks - a pandas dataframe containing all training landmark
                     data.
index              - a perticular landmark id of interest. eg. '101'
x_names            - a list of explanatory variable names.
                     eg. ['pts', 'r']
y_name             - a string representing response variable name.
                     eg. 'stype'
class0             - name of the first class. eg. 'wt-at'
class1             - name of the second class. eg. 'mt-at'
C_values           - a list of tunning variable C (penalty parameter
                     of the error term) that the method would grid-search
                     on. Default value is [0.1, 1, 10].

Output:
svm                - the SVM model trained from the training dataset
type0_0            - among the training samples, the number of class0
                     samples with chosen landmark predicted as class0
type0_1            - among the training samples, the number of class0
                     samples with chosen landmark predicted as class1
type1_1            - among the training samples, the number of class1
                     samples with chosen landmark predicted as class1
type1_0            - among the training samples, the number of class1
                     samples with chosen landmark predicted as class0
'''
def svm_classification(training_landmarks, index, x_names, y_name, class0,
        class1, C_values = [0.1, 1, 10] ):
    # filter out the landmarks needed
    chosenLandmark = landmarks[landmarks.landmark_index==index]
    chosenLandmark = chosenLandmark[np.isfinite(chosenLandmark['r'])]

    # create training and testing data
```

```
    X = chosenLandmark[x_names]
    y = chosenLandmark[y_name]

    # check whether both classes exist
    count_1 = chosenLandmark[y_name].str.contains(class1).sum()
    count_0 = chosenLandmark[y_name].str.contains(class0).sum()

    if (count_1 < 2 or count_0 < 2):
        return None, None, None, None, None

    # find the best C value by cross-validation
    tuned_parameters = [{'C': C_values}]
    clf = GridSearchCV(
        SVC(kernel='linear'),
        tuned_parameters, cv=10, scoring='accuracy')
    clf.fit(X.values, y.values)
    best_c = clf.best_params_['C']

    svc = SVC(C=best_c, kernel='linear')
    svc.fit(X, y)

    prediction = svc.predict(X)

    # print confusion matrix
    print("confusion matrix: ")
    cm = confusion_matrix(y, prediction)
    cm_df = pd.DataFrame(cm.T, index=svc.classes_, columns=svc.classes_)
    print(cm_df)

    # Statistics of training precision:
    # number of wild type samples with this landmark
    # predicted as wild type.
    type0_0 =0
    # number of wild type samples with this landmark
    # predicted as mutant type.
    type0_1 = 0
    # number of mutant type samples with this landmark
    # predicted as mutant type.
    type1_1 = 0
    # number of mutant type samples with this landmark
    # predicted as wild type.
    type1_0 = 0

    for i in range (len(y)):
        _y = y.values[i]
        _p = prediction[i]
```

```
        if _y==class1 and _p==class1:
            type1_1 = type1_1 + 1
        elif _y==class1 and _p==class0:
            type1_0 = type1_0 + 1
        elif _y==class0 and _p==class0:
            type0_0 = type0_0 + 1
        elif _y==class0 and _p==class1:
            type0_1 = type0_1 + 1

    return svc, type0_0, type0_1, type1_1, type1_0


if __name__ == "__main__":
    # Get Datafile
    landmarks = pd.DataFrame()
    while(landmarks.shape[0]<2):
        filename = str(input("Please enter dataset's path: "))
        try:
            landmarks = pd.read_csv(filename)
        except Exception:
            print ("Error in reading the file.
                Please check whether file exists.")

    # Column names
    columns = list(landmarks)
    # Check column names
    if 'stype' not in columns:
        print("Incorrect column names: Please
            name your sample type's column as 'stype' ")
        exit()
    if 'sample_index' not in columns:
        print("Incorrect column names: Please name your
            sample index's column as 'sample_index' ")
        exit()
    if 'landmark_index' not in columns:
        print("Incorrect column names: Please name your
            landmark index's column as 'landmark_index' ")
        exit()

    # Get Parameters' column names
    parameters = list(set(columns) -
        set(['stype', 'sample_index', 'landmark_index']))

    # Get class names
    class0 = ''
    class1 = ''
    classes = list(set(landmarks['stype'].values))
```

```
    while (class0 not in classes):
        class0 = str(input("Please enter name of type 0: "))
    while (class1 not in classes):
        class1 = str(input("Please enter name of type 1: "))

    # Remove rows with NaN values
    for parameter in parameters:
        landmarks = landmarks[np.isfinite(landmarks[parameter])]

    # Get sample id
    sample = pd.DataFrame()
    while(sample.shape[0]<2):
        sample_id = str(input("Please enter a VALID sample index: "))
        sample = landmarks[landmarks.sample_index==sample_id]

    # Get result file's name and create the file with column names
    result_file_name = str(input("Please enter result file path: "))
    result_file = open(result_file_name, 'w')
    result_file.write('sample_index,stype,
        landmark_index,pred,type0_0,type0_1,type1_1,type1_0\n')
    result_file.close()

    # Get existing landmark ids
    landmark_ids = sample['landmark_index']

    # Get Actual Type (the Label)
    stype = sample.iloc[0]['stype']

    leave_one_out = landmarks[landmarks.sample_index!=sample_id]
    for l in landmark_ids.values:
        print ("======================================")
        print ("landmark: ", str(l))
        svc, type0_0, type0_1, type1_1, type1_0 =
            svm_classification(training_landmarks = leave_one_out,
                                                index = l,
                                                x_names = ['pts', 'r'],
                                                y_name = 'stype',
                                                class0 = class0,
                                                class1 = class1,
                                                C_values = [0.1, 1, 10])
        if (svc is None):
            print("One of the classes have too few samples
                for this landmark, so skipping it.")
            continue

        prediction = svc.predict(sample[
          sample.landmark_index==l][['pts', 'r']])[0]
```

```
        result = ','.join(str(x) for x in [
          sample_id, stype, l, prediction,
            type0_0, type0_1, type1_1, type1_0 ]) + '\n'
        print('result:', result)

        result_file = open(result_file_name, 'a')
        result_file.write(result)
        result_file.close()
```

Shiny App

Package Dependency

```
# Shiny App-----------------------------------------------------
# Loading packages needed in the creation of the Shiny App
library(dplyr)
library(data.table)
library(ggplot2)
library(shiny)
```

User Input

```
# User Input ----------------------------------------------------
# Please modify the file directory accordingly
data <- fread("data/output_data_type0.csv")

# List of input variables ---------------------------------
list_of_indices <- c(unique(data$sample_index))
# Please add or subtract channels from the list_of_channels accordingly
list_of_channels <- c("type0", "type1")
```

User Interface

```
# User Interface
ui <- fluidPage(
  titlePanel(title=h4("Classification of Wildtype and Mutant
                      Zebrafish Brains via Computational Method",
                      align="center")),

  # Sidebar containing all input variables
  sidebarLayout(

    # User Inputs
```

```
  sidebarPanel(
    selectInput("sampleindex", "Sample Index:", list_of_indices),
    selectInput("channel", "Channel:", list_of_channels),

    # Input accuracy score threshold: 0-1 intervals
    sliderInput("precision", "Precision Rate Threshold:",
                min = 0, max = 1,
                value = 0, step = 0.01),
    sliderInput("recall", "Recall Rate Threshold:",
                min = 0, max = 1,
                value = 0, step = 0.01),
    sliderInput("f1", "F1 Rate Threshold:",
                min = 0, max = 1,
                value = 0, step = 0.01)
  ),

  # Output
  mainPanel(
    tabsetPanel(
      tabPanel("Accuracy Threshold",tableOutput("values")),
      #heatmaps and histograms, side by side
      tabPanel("Type 0 Precision", fluidRow(
        splitLayout(cellWidths = c("40%", "60%"),
                    plotOutput("plot2"), plotOutput("plot1"))
        )),
      tabPanel("Type 1 Precision", fluidRow(
        splitLayout(cellWidths = c("40%", "60%"),
                    plotOutput("plot4"), plotOutput("plot3"))
        )),
      tabPanel("Precision",fluidRow(
        splitLayout(cellWidths = c("40%", "60%"),
                    plotOutput("plot6"), plotOutput("plot5"))
        )),
      tabPanel("Type 0 Recall", fluidRow(
        splitLayout(cellWidths = c("40%", "60%"),
                    plotOutput("plot8"), plotOutput("plot7"))
      )),
      tabPanel("Type 1 Recall", fluidRow(
        splitLayout(cellWidths = c("40%", "60%"),
                    plotOutput("plot10"), plotOutput("plot9"))
      )),
      tabPanel("Recall",fluidRow(
        splitLayout(cellWidths = c("40%", "60%"),
                    plotOutput("plot12"), plotOutput("plot11"))
      )),
      tabPanel("Type 0 F1", fluidRow(
        splitLayout(cellWidths = c("40%", "60%"),
```

```
                    plotOutput("plot14"), plotOutput("plot13"))
        )),
        tabPanel("Type 1 F1", fluidRow(
          splitLayout(cellWidths = c("40%", "60%"),
                      plotOutput("plot16"), plotOutput("plot15"))
        )),
        tabPanel("F1",fluidRow(
          splitLayout(cellWidths = c("40%", "60%"),
                      plotOutput("plot18"), plotOutput("plot17"))
        ))
        )
      )
  )
)
```

Shiny App Server

```
# Server-------------------------------------------------------
server <- function(input,output) {

  #loading data needed to create visualizations
  dat <- reactive({

    # Please modify the file directory accordingly
    path <- paste0("data/output_data_", input$channel, ".csv")
    # path <- paste0("7.aggregatedResults/", input$channel,
    "_2med_renamed_2.csv")
    data <- fread(path)

    # Please modify the file directory accordingly
    landmark_xy <- fread("data/landmark_xy.csv")
    # landmark_xy <- fread("3.InputData/tidy/landmark_xy.csv")

    # Adding position of each landmark
    data <- data %>%
      left_join(landmark_xy, by="landmark_index")

    # Adding baselines to the data file
    data_base <- data %>%
      filter(overall_precision >= input$precision,
             overall_recall >= input$recall,
             overall_f1 >= input$f1) %>%
      mutate(# type 0
             type0_p_b = type0_num/(type0_num+type1_num),
             type0_r_b = 1,
             type0_f1_b = 2*type0_p_b*type0_r_b/
```

```
              (type0_p_b + type0_r_b),

            # type 1
            type1_p_b = type1_num/
              (type0_num+type1_num),
            type1_r_b = 1,
            type1_f1_b = 2*type1_p_b*type1_r_b/
              (type1_p_b + type1_r_b),

            # overall
            p_b = (type0_p_b * type0_num + type1_p_b *type1_num)/
              (type0_num+type1_num),
            r_b = (type0_r_b * type0_num + type1_r_b *type1_num)/
              (type0_num+type1_num),
            f1_b = (type0_f1_b * type0_num + type1_f1_b *type1_num)/
              (type0_num+type1_num)
            )

  #filter out the sample not interested
  test <- data_base %>%
    filter(sample_index == input$sampleindex)

  #return dataset
  print(test[1,])
  test
  })

# Reactive expression to create data frame of all input values
sliderValues <- reactive({

  # Getting the true type of the sample
  type <- dat()$type[1]

  # Doing majority vote and perdicting the type of the sample
  test_pred <- dat() %>%
    filter(overall_precision >= input$precision,
           overall_recall >= input$recall,
           overall_f1 >= input$f1)%>%
    group_by(pred) %>%
    summarise(N = n()) %>%
    mutate(max = max(N)) %>%
    mutate(predict = ifelse(N == max, TRUE, FALSE)) %>%
    filter(predict == TRUE)
  prediction <- test_pred$pred[1]

  # summary table
  data.frame(
```

```
    Name = c("Precision Rate Threshold",
             "Recall Rate Threshold",
             "F1 Rate Threshold",
             "Type",
             "Prediction",
             "Number of Type 0 Samples Used In Model",
             "Number of Type 1 Samples Used In Model"),
    Value = as.character(c(input$precision,
                           input$recall,
                           input$f1,
                           type,
                           prediction,
                           mean(dat()$type0_num),
                           mean(dat()$type1_num)
                           )),
    stringsAsFactors = FALSE)
})

# Show the threshold values in an summary table
output$values <- renderTable({
  sliderValues()
})

# precision ------------------------------------------------
output$plot1 <- renderPlot({
  p1 <- ggplot(dat(),aes(x = column, y = row)) +
    geom_tile(aes(fill = type0_precision)) +
    xlab("Alpha") +
    ylab("Theta") +
    scale_x_continuous(limits = c(0, 20),
                       breaks=c(1, 10, 19),
                       labels=c("-90.51", "0", "90.51")) +
    scale_y_continuous(limits = c(0, 9),
                       breaks=c(1, 4.5, 8),
                       labels=c("-3.14","0","3.14")) +
    scale_fill_continuous(limits=c(0, 1),
                          breaks=seq(0,1,by=0.25))
  p1
})

output$plot3 <- renderPlot({
  p3 <- ggplot(dat(),
               aes(x = column, y = row)) +
    geom_point() +
    #scale_color_viridis() +
    geom_tile(aes(fill = type1_precision)) +
    xlab("Alpha") +
```

```
    ylab("Theta") +
    scale_x_continuous(limits = c(0, 20),
                       breaks=c(1, 10, 19),
                       labels=c("-90.51", "0", "90.51")) +
    scale_y_continuous(limits = c(0, 9),
                       breaks=c(1, 4.5, 8),
                       labels=c("-3.14","0","3.14")) +
    scale_fill_continuous(limits=c(0, 1),
                          breaks=seq(0,1,by=0.25))
  p3
})

output$plot5 <- renderPlot({
  p5 <- ggplot(dat(),
               aes(x = column, y = row)) +
    geom_point() +
    #scale_color_viridis() +
    geom_tile(aes(fill = overall_precision)) +
    xlab("Alpha") +
    ylab("Theta") +
    scale_x_continuous(limits = c(0, 20),
                       breaks=c(1, 10, 19),
                       labels=c("-90.51", "0", "90.51")) +
    scale_y_continuous(limits = c(0, 9),
                       breaks=c(1, 4.5, 8),
                       labels=c("-3.14","0","3.14")) +
    scale_fill_continuous(limits=c(0, 1),
                          breaks=seq(0,1,by=0.25))
  p5
})

output$plot2 <- renderPlot({
  baseline <- mean(dat()$type0_p_b)
  p2 <- qplot(dat()$type0_precision, geom = "histogram") +
    geom_vline(xintercept=baseline, linetype="dashed",
               color = "red") +
    scale_x_continuous(limits = c(0, 1)) +
    xlab("Precision") +
    ylab("Count")
  p2
})

output$plot4 <- renderPlot({
  baseline <- mean(dat()$type1_p_b)
  p4 <- qplot(dat()$type1_precision, geom = "histogram") +
    geom_vline(xintercept=baseline, linetype="dashed",
               color = "red") +
```

```
    scale_x_continuous(limits = c(0, 1)) +
    xlab("Precision") +
    ylab("Count")
  p4
})

output$plot6 <- renderPlot({
  baseline <- mean(dat()$p_b)
  p6 <- qplot(dat()$overall_precision, geom = "histogram") +
    geom_vline(xintercept=baseline, linetype="dashed",
               color = "red") +
    scale_x_continuous(limits = c(0, 1)) +
    xlab("Precision") +
    ylab("Count")
  p6
})

# recall ------------------------------------------------
output$plot7 <- renderPlot({
  p7 <- ggplot(dat(),aes(x = column, y = row)) +
    geom_tile(aes(fill = type0_recall)) +
    xlab("Alpha") +
    ylab("Theta") +
    scale_x_continuous(limits = c(0, 20),
                       breaks=c(1, 10, 19),
                       labels=c("-90.51", "0", "90.51")) +
    scale_y_continuous(limits = c(0, 9),
                       breaks=c(1, 4.5, 8),
                       labels=c("-3.14","0","3.14")) +
    scale_fill_continuous(limits=c(0, 1),
                          breaks=seq(0,1,by=0.25))
  p7
})

output$plot9 <- renderPlot({
  p9 <- ggplot(dat(),
               aes(x = column, y = row)) +
    geom_point() +
    #scale_color_viridis() +
    geom_tile(aes(fill = type1_recall)) +
    xlab("Alpha") +
    ylab("Theta") +
    scale_x_continuous(limits = c(0, 20),
                       breaks=c(1, 10, 19),
                       labels=c("-90.51", "0", "90.51")) +
    scale_y_continuous(limits = c(0, 9),
                       breaks=c(1, 4.5, 8),
```

```
                        labels=c("-3.14","0","3.14")) +
    scale_fill_continuous(limits=c(0, 1),
                          breaks=seq(0,1,by=0.25))
  p9
})

output$plot11 <- renderPlot({
  p11 <- ggplot(dat(),
                aes(x = column, y = row)) +
    geom_point() +
    #scale_color_viridis() +
    geom_tile(aes(fill = overall_recall)) +
    xlab("Alpha") +
    ylab("Theta") +
    scale_x_continuous(limits = c(0, 20),
                       breaks=c(1, 10, 19),
                       labels=c("-90.51", "0", "90.51")) +
    scale_y_continuous(limits = c(0, 9),
                       breaks=c(1, 4.5, 8),
                       labels=c("-3.14","0","3.14")) +
    scale_fill_continuous(limits=c(0, 1),
                          breaks=seq(0,1,by=0.25))
  p11
})

output$plot8 <- renderPlot({
  baseline <- mean(dat()$type0_r_b)
  p8 <- qplot(dat()$type0_recall, geom = "histogram") +
    geom_vline(xintercept=baseline, linetype="dashed",
               color = "red") +
    scale_x_continuous(limits = c(0, 1)) +
    xlab("Precision") +
    ylab("Count")
  p8
})

output$plot10 <- renderPlot({
  baseline <- mean(dat()$type1_r_b)
  p10 <- qplot(dat()$type1_recall, geom = "histogram") +
    geom_vline(xintercept=baseline, linetype="dashed",
               color = "red") +
    scale_x_continuous(limits = c(0, 1)) +
    xlab("Precision") +
    ylab("Count")
  p10
})
```

```
output$plot12 <- renderPlot({
  baseline <- mean(dat()$r_b)
  p12 <- qplot(dat()$overall_recall, geom = "histogram") +
    geom_vline(xintercept=baseline, linetype="dashed",
               color = "red") +
    scale_x_continuous(limits = c(0, 1)) +
    xlab("Precision") +
    ylab("Count")
  p12
})

# f1 ---------------------------------------------------------------------------------------
output$plot13 <- renderPlot({
  p13 <- ggplot(dat(),aes(x = column, y = row)) +
    geom_tile(aes(fill = type0_f1)) +
    xlab("Alpha") +
    ylab("Theta") +
    scale_x_continuous(limits = c(0, 20),
                       breaks=c(1, 10, 19),
                       labels=c("-90.51", "0", "90.51")) +
    scale_y_continuous(limits = c(0, 9),
                       breaks=c(1, 4.5, 8),
                       labels=c("-3.14","0","3.14")) +
    scale_fill_continuous(limits=c(0, 1),
                          breaks=seq(0,1,by=0.25))
  p13
})

output$plot15 <- renderPlot({
  p15 <- ggplot(dat(),
                aes(x = column, y = row)) +
    geom_point() +
    #scale_color_viridis() +
    geom_tile(aes(fill = type1_f1)) +
    xlab("Alpha") +
    ylab("Theta") +
    scale_x_continuous(limits = c(0, 20),
                       breaks=c(1, 10, 19),
                       labels=c("-90.51", "0", "90.51")) +
    scale_y_continuous(limits = c(0, 9),
                       breaks=c(1, 4.5, 8),
                       labels=c("-3.14","0","3.14")) +
    scale_fill_continuous(limits=c(0, 1),
                          breaks=seq(0,1,by=0.25))
  p15
})
```

```
output$plot17 <- renderPlot({
  p17 <- ggplot(dat(),
                aes(x = column, y = row)) +
    geom_point() +
    #scale_color_viridis() +
    geom_tile(aes(fill = overall_f1)) +
    xlab("Alpha") +
    ylab("Theta") +
    scale_x_continuous(limits = c(0, 20),
                       breaks=c(1, 10, 19),
                       labels=c("-90.51", "0", "90.51")) +
    scale_y_continuous(limits = c(0, 9),
                       breaks=c(1, 4.5, 8),
                       labels=c("-3.14","0","3.14")) +
    scale_fill_continuous(limits=c(0, 1),
                          breaks=seq(0,1,by=0.25))
  p17
})

output$plot14 <- renderPlot({
  baseline <- mean(dat()$type0_f1_b)
  p14 <- qplot(dat()$type0_f1, geom = "histogram") +
    geom_vline(xintercept=baseline, linetype="dashed",
               color = "red") +
    scale_x_continuous(limits = c(0, 1)) +
    xlab("Precision") +
    ylab("Count")
  p14
})

output$plot16 <- renderPlot({
  baseline <- mean(dat()$type1_f1_b)
  p16 <- qplot(dat()$type1_f1, geom = "histogram") +
    geom_vline(xintercept=baseline, linetype="dashed",
               color = "red") +
    scale_x_continuous(limits = c(0, 1)) +
    xlab("Precision") +
    ylab("Count")
  p16
})

output$plot18 <- renderPlot({
  baseline <- mean(dat()$f1_b)
  p18 <- qplot(dat()$overall_f1, geom = "histogram") +
    geom_vline(xintercept=baseline, linetype="dashed",
               color = "red") +
    scale_x_continuous(limits = c(0, 1)) +
```

```
      xlab("Precision") +
      ylab("Count")
    p18
  })
}
```

Outputting the Shiny App

```
# Creating the Shiny App
shinyApp(ui, server)
```

References

1. Morgan Schwartz BSB Jake Schnabl. A new computational method to quantify 3D image data and to detail changes in morphological structure and spatial relationships during nervous system development. 2018;

2. Xie Y. Knitr: A general-purpose package for dynamic report generation in r [Internet]. 2018. Available: https://CRAN.R-project.org/package=knitr

3. Allaire J, R Foundation, Wickham H, Journal of Statistical Software, Xie Y, Vaidyanathan R, et al. Rticles: Article formats for r markdown [Internet]. 2017. Available: https://CRAN.R-project.org/package=rticles

4. Ian L. Dryden KV. Statistical shape analysis with applications in r. John Wiley&Sons.Ltd. 2016.

5. Tommi Jaakkola DH Mark Diekhaus. Using the fisher kernel method to detect remote protein homologies. 1999; Available: http://www.aaai.org/Papers/ISMB/1999/ISMB99-018.pdf

6. Wickham H. Tidy data. The Journal of Statistical Software. 2014;59. Available: http://www.jstatsoft.org/v59/i10/

7. Gareth James TH Daniela Witten. An introduction to statistical learning. Springer Science+Business Media New York; 2013.

8. Wickham H, Francois R, Henry L, Müller K. Dplyr: A grammar of data manipulation [Internet]. 2017. Available: https://CRAN.R-project.org/package=dplyr

9. Dowle M, Srinivasan A. Data.table: Extension of 'data.frame' [Internet]. 2017. Available: https://CRAN.R-project.org/package=data.table

10. Wickham H. Ggplot2: Elegant graphics for data analysis [Internet]. Springer-Verlag New York; 2009. Available: http://ggplot2.org

11. Chang W, Cheng J, Allaire J, Xie Y, McPherson J. Shiny: Web application framework for r [Internet]. 2017. Available: https://CRAN.R-project.org/package=shiny

12. P. Rämö BS R. Sacher. CellClassifier: Supervised learning of cellular phenotypes [Internet]. Bioinfomatics. 2009. Available: http://dx.doi.org/10.1093/bioinformatics/btp524