Classification of Wildtype and Mutant Zebrafish Brains via Computational Method

Shuli Hu [1] , Wencong Li [1] , Dejia Tang [1] , Ji Young Yun [1]

1 Statistical and Data Sciences, Northampton, MA

Abstract

Classification of biological creatures' phenotypes has long been a field that scientists study at. In this project, we utilize support vector machine to distinguish structures of Zebrafish's brains by using data generated from landmark analysis (cited Morgan's paper). We create a tool for biologists to intuitively classify three-dimensional biological shapes into two groups, usually defined as wild type and mutant, and understand which part 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.

Introduction

This project derives from Professor Barresi's biological image analysis research at Smith College and provides a tool to classify the structures within zebrafish brains via support vector machine. Our goal is to distinguish the wild and mutant types of zebrafish brain's structures. Morgan, a student in Barresi Lab, used landmarks analysis to divide the points in the three-dimentional images into small wedges and computed the landmark, which is the most representative point, within each wedge. The image of signals in a Zebrafish brain is shown in Figure 1. The shape is divided into 30 slices, and each slice is further divided into 8 wedges. The landmark in each wedge is calculated by taking the median distance of all points in each wedge, R. We use number of points in each wedge and median R to run SVM models to do classifications.

Landmark Analysis

Programming Languages

Two programming languages are used in this study, Python and R. Python is used to run our model and output the result in the 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 two main points of emphasis in this honors project. 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.

`knitr` [1] and Github are used in this project to make the study reproducible, ranging from the initial data source to the `nyctaxi` package to the statistical data analysis. 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 and interactive applicationis, and output into PDF. `rticles` helps users to efficiently put together scientific paper with similar format [2]. Github is used to store the scripts for our final paper, and the source code for our final production which contains the final model and user interface.

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 visual data are needed, particularly to detect subtle phenotypes.

Morgan et al. (2018) have utilized 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.

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 (Dryden and Mardia, 2016). 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.

Morgan et al. 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 aspects. Instead of doing classification on all of the landmarks at the same time, we decided to do classifacation 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. ushered in stage 4 of 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

We have 43 wildtypes samples (n_1) and 35 mutant samples (n_2) for training and testing. There are 152 landmarks (N) for each sample, 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 landmarks, we filled them with the median value of all the points in that landmark.

Tidy Data

The original landmarks data is a wide table containing the sample index and all the columns holding information regarding the minimum and maximum values of Alpha and Theta, number of points, median r value, and the type of sample for a particular sample in each landmark. However, because all of such variables were joined by underscores in

the variable names, such as `-14.29_-4.76_-0.79_0.0_50_pts` or `-14.29_-4.76_-0.79_0.0_50_r` and the value in each cell refers to the median r value or number of points, it was very difficult to see what each column actually represented. The ideal format of the data set was to have the sample index, minimum and maximum Alpha, minimum and maximum Theta, number of points, median r, and type of sample each be its own column.

Hence, three key functions were used from the tidyr package [3]: gather(), separate(), and spread(. The gather() function separated the dataset into key and value pairs for each index. The key was the column name containing all essential information connected by underscores and the value included the number of points or median r value. Then, the separate() function separated the result from the gather function divided the column connected by underscore into 5 different columns, named as `min_alpha`, `max_alpha`, `min_theta`, `max_theta`, and `ptsOrR`. This was added to the result of the gather function that contained the index and value of each cell, either median R or number of points. Afterwards, the spread() function widened the already wide table by expanding the `ptsOrR` column by creating two columns, each column representing median R and the number of points.

Dealing with Missing Values

Support Vector Machine (SVM) cannot be fit to data with misisng values. For wedges that do not have any point in it, `median r` cannot be calculated, which means that these sample will be eliminated when running SVM. Wedges without points have biologically meanings, and we should not ignore these wedges in our model. In order to keep the wedges in our model, we need to artificially pick a `median r` value to replace the missing ones. Supporting vector machine is sensitive to outliers, so we cannot pick an `r` value that could become outliers. We decided to calculate the mean of `median r` for the nth landmark of all 78 samples, and then we replace the missing `median r` values with the 2 * median R value for each landmark of each channel.

Supporting Vector Machine

SVM's have been proven to be a powerful algorithm for supervised clustering. During the past few years, SVM has been applied very broadly within the field of computational biology especially in pattern recognition problems. The goal of SVM is to find a separation line $f(x) = (\beta_0 + \beta_1 * x_1 + \beta_2 * x_2)$ that separates the nearest data as clean as possible. The parameters β are found by solving the optimization problem –to maximize M subject to some restrictions – in 2 dimensions below.

$$\begin{array}{l} \sum_{i=1}^{n} \beta_i^2 = 0 \\ y * (\beta_0 + \beta_1 * x_1 + \beta_2 * x_2) \geq M(1 - \varepsilon_i) \\ \varepsilon_i \geq 0 \\ \sum_{i=1}^{n} \varepsilon_i \leq C \end{array} \tag{1}$$

- C: tuning parameter, toleration of violation.
- M: margin, distance of the closest points to the hyperplane.
- ε_i : slack variable, an observation is classified at the correct/incorrect side of the margin.

The function of the separation line:

$$f(x) = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2$$

if $f(x) = 0$, the observation is on the seperation line.

$$y * (\beta_0 + \beta_1 * x_1 + \beta_2 * x_2)$$

The above is the perpendicular distance from the ith observation to the separation line. If it's >0, the observation falls at the right side of the separation line and vice versa.

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 dicided to use a leave-one-out cross validation method to test our model.

For each testing sample, we built 152 SVMs 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

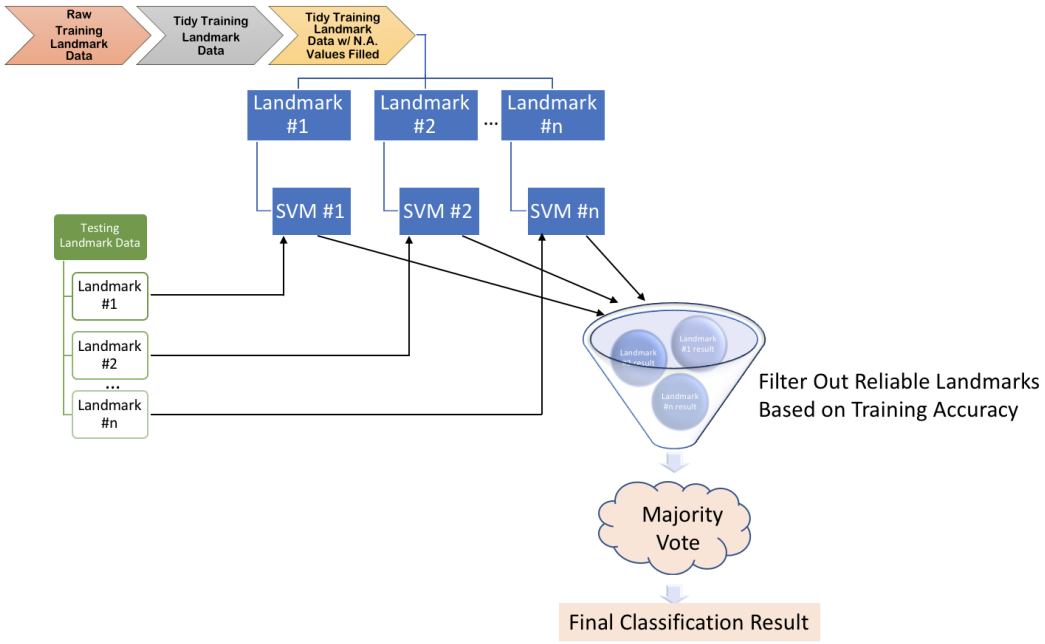

Figure 1: Workflow

Fig 1. Summary of Workflow

Figure 1 displays the overall workflow of the final product.

Step One: Data Processing and Modelling

This step is implemented using **Python** (version 3) and packages including `pandas`, `nump` and `sklearn` are required. Users would need to run and interact with the Python script `svm.py` to pre-process the data and build the model.

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

User Interaction

```
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 2: User Interaction

Fig 2. Example of User Interaction in Step One of the User Interface

As shown in Figure 2, several user inputs are taken from users when they run the python script.

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 sub-section corresponding to each landmark of the 3D shape and the `median R` of points in each wedge.

Sample input file

	sample_index	min_alpha	max_alpha	min_theta	max_theta	num	pts	r	stype	unique_key	landmark_index
0	1	-90.51	-80.99	-3.14	-2.36	50	0.0	0.0	mt-at	-90.51/-3.14	1
1	101	-90.51	-80.99	-3.14	-2.36	50	0.0	0.0	wt-at	-90.51/-3.14	1
2	102	-90.51	-80.99	-3.14	-2.36	50	0.0	0.0	wt-at	-90.51/-3.14	1
3	103	-90.51	-80.99	-3.14	-2.36	50	0.0	0.0	wt-at	-90.51/-3.14	1
4	104	-90.51	-80.99	-3.14	-2.36	50	0.0	0.0	wt-at	-90.51/-3.14	1

Figure 3: Head of Sample Input

Fig 3. Sample Data Input File of First Step of the User Interface

Output File

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 access by doing the following:

	landmark_index	pred	ww	wm	mm	mw	sample_id	w_support	m_support	w_precision	w_recall	m_precision	m_recall
0	1	0	43	0	0	34	1	43	34	0.558442	1.0	0.0	0.0
1	2	0	43	0	0	34	1	43	34	0.558442	1.0	0.0	0.0
2	3	0	43	0	0	34	1	43	34	0.558442	1.0	0.0	0.0
3	4	1	43	0	34	0	1	43	34	1.000000	1.0	1.0	1.0
4	5	1	43	0	34	0	1	43	34	1.000000	1.0	1.0	1.0

Figure 4: Head of Sample Output

Fig 4. Sample Data Output File of First Step of the User Interface

```
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 everytime users cahnge one or multiple thresholds. Our app filters out the observations that do not fulfill the threshold requirements and uses the resulting dataset to update the histograms and heatmaps.

Output: Interactive User Interface

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

- `dplyr` [4]
- `data.table` [5]
- `ggplot2` [6]
- `shiny` [7]

We visualize the 9 accuracy scores by using both histograms and the corresponding heatmaps that display the scores inlcuded 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 heatmap.

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

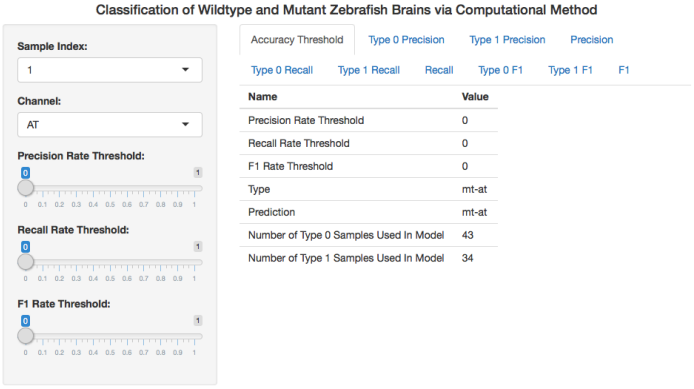

Fig 5. User Interface: Accuracy Threshold Summary Tab of AT Channel

Figure 5 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 resuting predicted sample type is also updated in the summary table.

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

Figure 6 displays the Precision Score Visualization tab of the first sample of AT channel. In this case, all three thresholds are at default level, 0. Therefore, all landmarks' precision scores are shown in both the histogram and the heatmap.

Figure 7 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 6. Some of the blocks in figure 6 are turned into blank blocks after the precision threshold is increased to 0.75.

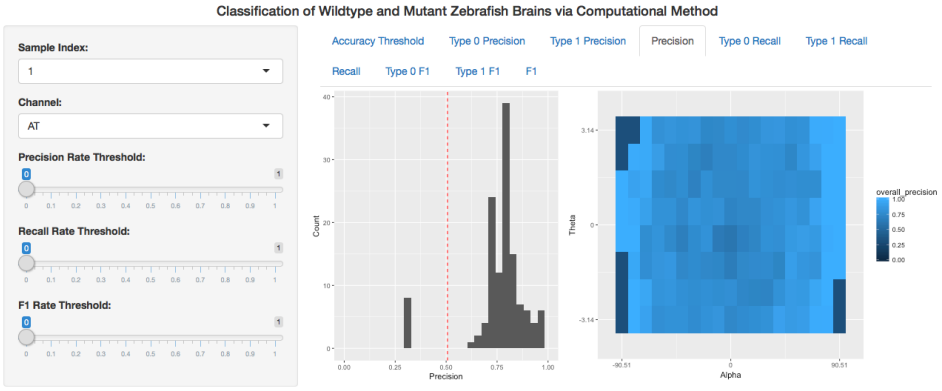

Fig 6. User Interface: Precision Score Visualization Tab of AT Channel

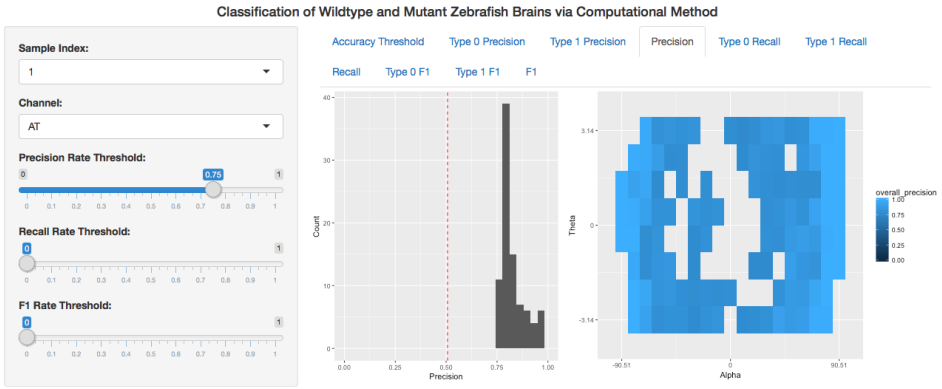

Fig 7. User Interface: Precision Score Visualization Tab of AT Channel, with precision threshold = 0.75

Users can also choose to observe the SVM results of ZRF channel. Figure 8 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:

Easy Interpretation

In the previous method random forest, the number of predictors p exceeds the number of samples. Morgan applied PCA do reduce the dimension of the predictors. The problem with dimension reduction is that it gives a linear combination of the dimensions that are projected on those are kept. While the largest projections still make sense, the minor projections are very random and thus difficult to interpret.

The SVM model is generated based on landmark data gives insightful analysis of: * which landmark, or which part of the Zebrafish brain, has more predictive power * whether a new Zebrafish brain sample is a mutant or wild type

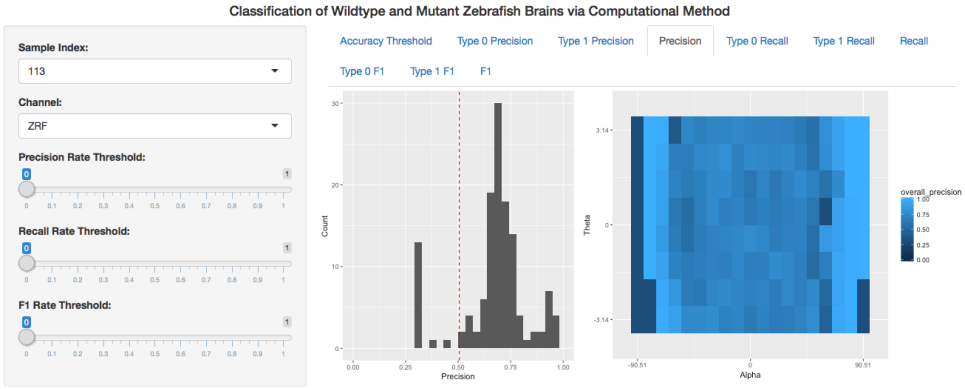

Fig 8. User Interface: F1 Score Visualization Tab of ZRF Channel

Implementing User Feedback

We have implemented feedbacks from users in our user interface. Originally, our shiny app only produces visualizations of one channel's data, but we added an additional variable, `channel`, for users to analyze three-dimentional 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

Our SVM model only make prediction of the sample type based on a single channel's information, and it does not consider the interaction between channels.

Itenerating Machine Learning

Instead of cross-validation, better results could be achieved by using itenerating machine learning method. In iterative machine learning we repeat the process of training and testing several times. At the first round the user gives examples of objects belonging to some 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 or are for some reason easily missed by humans. 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.

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 get more feedbacks from other scientists and improve our model and user interface accordingly.

Improving Model Accuracy

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 Benjamin Baumer in Spring 2018.

Appendix A: Shiny App Accuracy Score Visualizations

Recall Score Visualization tab of the first sample of AT channel

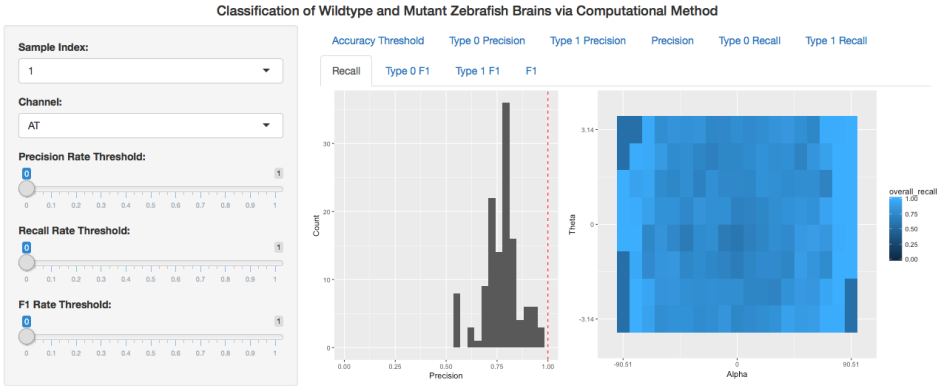

Fig 9. User Interface: Recall Score Visualization Tab of AT Channel

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

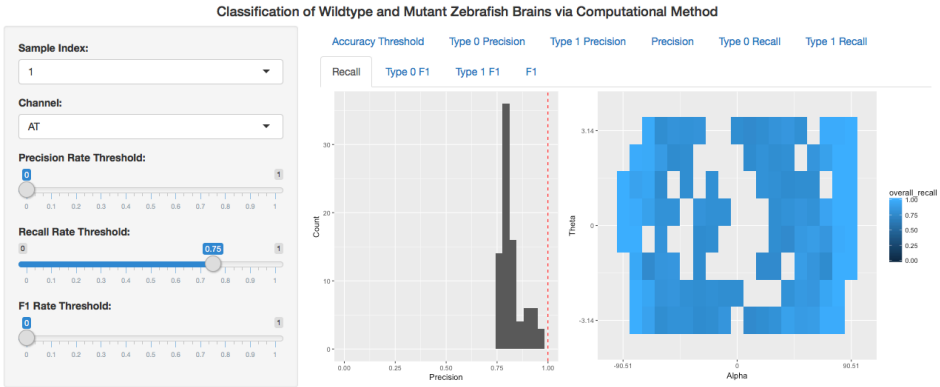

Fig 10. User Interface: Recall Score Visualization Tab of AT Channel, with recall threshold = 0.75

F1 Score Visualization tab of the first sample of AT channel

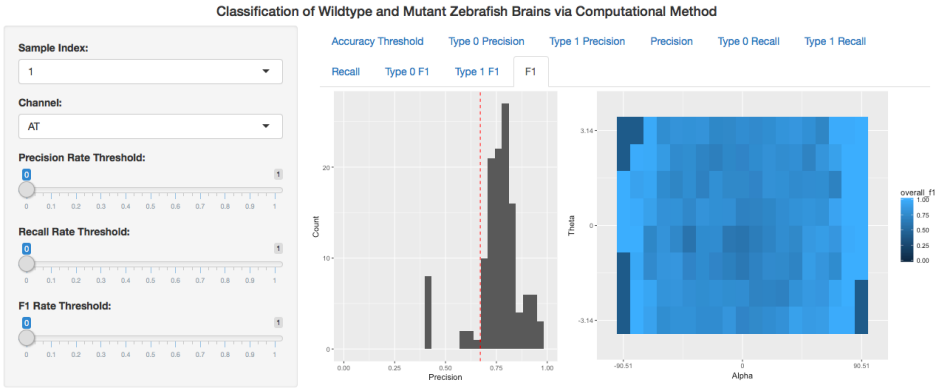

Fig 11. User Interface: F1 Score Visualization Tab of AT Channel

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

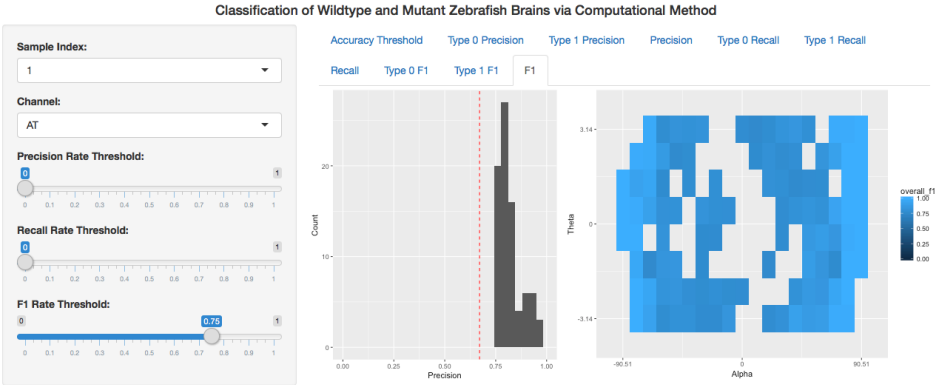

Fig 12. User Interface: F1 Score Visualization Tab of AT Channel, with f1 threshold = 0.75

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
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 err
Output:
svm                - the SVM model trained from the training dataset
ww                 - among the training samples, the number of wild type sample
wm                 - among the training samples, the number of wild type sample
mm                 - among the training samples, the number of mutant type samp
mw                 - among the training samples, the number of mutant type samp
'''
def svm_classification(training_landmarks, index, x_names, y_name, class0, clas
    # 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]
    y = y.replace([class1], 1)
    y = y.replace([class0], 0)
    # 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='
    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.
    ww =0
    # number of wild type samples with this landmark predicted as mutant type.
    wm = 0
    # number of mutant type samples with this landmark predicted as mutant type
    mm = 0
    # number of mutant type samples with this landmark predicted as wild type.
    mw = 0

    for i in range (len(y)):
        _y = y.values[i]
        _p = prediction[i]
        if _y==1 and _p==1:
            mm = mm + 1
        elif _y==1 and _p==0:
            mw = mw + 1
        elif _y==0 and _p==0:
            ww = ww + 1
        elif _y==0 and _p==1:
            wm = wm + 1

    return svc, ww, wm, mm, mw
if __name__ == "__main__":
    # Get interested chnnel name
    channel = ''
    while (channel != 'AT' and channel != 'ZRF'):
        channel = input("Please enter 'AT' or 'ZRF' to indicate channel interes

    class0 = 'mt-zrf' if channel == 'ZRF' else 'mt-at'
    class1 = 'wt-zrf' if channel == 'ZRF' else 'wt-at'
    # Read in landmark data
    data_type = '-1'
    while (data_type != '0' and data_type != '1'):
        data_type = input("Enter 0 for filling NaN values with median and 1 for
    landmarks = pd.DataFrame()
    if (channel == 'AT'):
        landmarks = pd.read_csv('./data/final/landmark_AT_filled_w_median.csv')
    else:
        landmarks = pd.read_csv('./data/final/landmark_ZRF_filled_w_median.csv'
    # 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 name: "))
```

```
    result_file = open(result_file_name, 'w')
    result_file.write('sample_index, landmark_index, pred, ww, wm, mm, mw\n')
    result_file.close()
    # Get existing landmark ids
    landmark_ids = sample['landmark_index']
    leave_one_out = landmarks[landmarks.sample_index!=sample_id]
    for l in landmark_ids.values:
        print ("========================================")
        print ("landmark: ", str(l))
        svc, ww, wm, mm, mw = svm_classification(training_landmarks = leave_one
                                                 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, s
            continue
        prediction = svc.predict(sample[sample.landmark_index==l][['pts', 'r']]
        result = ', '.join(str(x) for x in [sample_id, l, prediction[0], ww, wm
        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. Xie Y. Knitr: A general-purpose package for dynamic report generation in r [Internet]. 2018. Available: https://CRAN.R-project.org/package=knitr
2. 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
3. Wickham H. Tidy data. The Journal of Statistical Software. 2014;59. Available: http://www.jstatsoft.org/v59/i10/
4. Wickham H, Francois R, Henry L, Müller K. Dplyr: A grammar of data manipulation [Internet]. 2017. Available: https://CRAN.R-project.org/package=dplyr
5. Dowle M, Srinivasan A. Data.table: Extension of 'data.frame' [Internet]. 2017. Available: https://CRAN.R-project.org/package=data.table
6. Wickham H. Ggplot2: Elegant graphics for data analysis [Internet]. Springer-Verlag New York; 2009. Available: http://ggplot2.org
7. 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