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.

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.

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 used

Python

R

Git

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 and 35 mutant samples for training and testing. There are 152 landmarks for each sample, with each of them containing the following variables: number of points in that wedge median R (micro-meter): the median of the distances to the center of the slice of all the points in that wedge. alpha (micro-meter): distance from the center of the landmark to the midline theta (radian): the degree that shows the location of the wedge of a 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: 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 Value

Samples with missing values are eliminated by Supporting vector machine. 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 mean.

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 seperation 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.

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

- 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 seperation line:

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

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

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

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

Workflow and User Interface

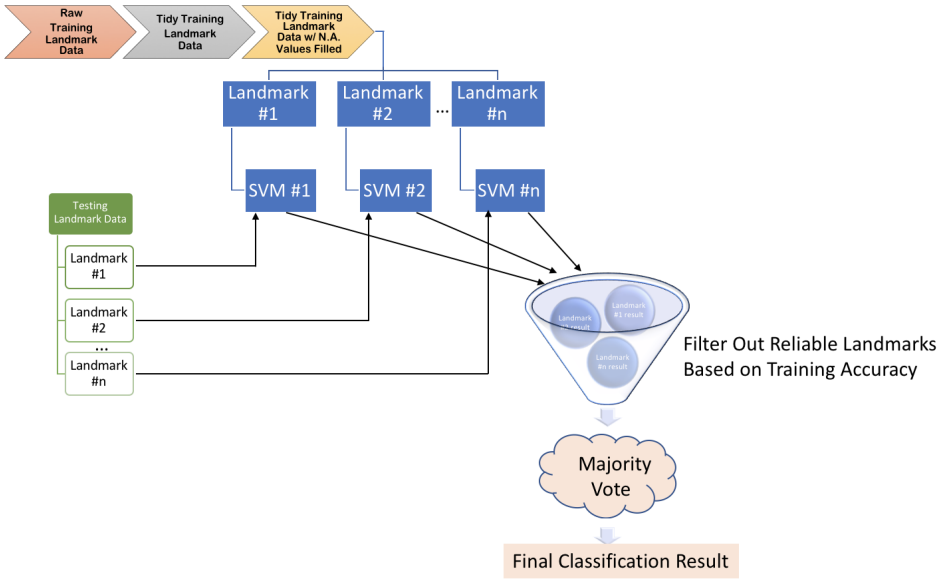

Figure 1: Workflow

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

As shown in `Figure2`, several user inputs are tooken 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

Output File

	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

Step Two: Result Analyzation and Visualization

Testing

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.

Processing Results and Make Predictions

After we get precision scores and predictions of each landmark, we will present the distribution of the landmarks' precision scores. The user would then be allowed to set a threshold for certain precision scores to select out a subset of landmarks that are considered significant. A majority vote would then be performed among the selected landmarks to get an overall prediction for the sample.

Result

Prediction

Visulization

Discussion

Strengths

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 generated based on landmark data gives insightful analysis of: 1. which landmark, or which part of the Zebrafish brain, has more predictive power 2. whether a new Zebrafish brain sample is a mutant or wild type

Limitations

The SVM model also has its limitation in that it only considers one single landmark at a time without considering the relationship across the landmarks of the whole sample.

Improvements

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

1. making the program more user friendly
2. running more tests to prove the accuracy of our model

Appendix Code

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

```
list_of_indices <- c(index$Index, "AT", "ZRF")
list_of_scores <- c("precision", "recall", "f1", "w_precision", "w_recall", "w_
landmark_xy <- fread("/Users/priscilla/Desktop/SDS Capstone/Zebrafish/analysis/
list_of_channel <- c("AT", "ZRF")

# User Interface
ui <- fluidPage(
  titlePanel(title=h4("Classification of Wildtype and Mutant Zebrafish Brains v
                      align="center")),
  selectInput("channel", "Channel:", list_of_channel),
  selectInput("sampleindex", "Sample Index:", list_of_indices),
  selectInput("score", "Accuracy Measurement:", list_of_scores),
  mainPanel(fluidRow(
              splitLayout(cellWidths = c("90%", "60%"), plotOutput("plot1"), pl
            ))
)

# Server
server <- function(input,output) {
  dat <- reactive({
    dir <- paste0(wd, "/analysis/r", input$sampleindex, "_med_", input$channel,
    test <- fread(dir)
    test <- test %>%
      left_join(landmark_xy, by="landmark_index")
    print(test)
    test
  })

  # Plot One
  output$plot1 <- renderPlot({
    p1 <- ggplot(dat(),
                 aes(x = y, y = x)) +
      geom_tile(aes(fill = input$score)) +
      xlab("Alpha") +
      ylab("Theta") +
      scale_x_continuous(limits = c(1, 19), breaks=c(1, 10, 19), labels=c("-90.
      scale_y_continuous(limits = c(1, 8), breaks=c(1, 4.5, 8), labels=c("-3.14
      scale_fill_continuous(limits=c(0, 1), breaks=seq(0,1,by=0.25))
    p1
  })

  # Plot Two
  output$plot2 <- renderPlot({
    p2 <- qplot(input$score, geom = "histogram") +
```

```
      xlab("Precision") +
      ylab("Count")
    p2
  })

}
```

References