Multivariate Linear Regression

Nail Bed Images

Nov 14, 2022

Background & Literature Review

Anemia is as a life-threatening disease that affects 2 billion, or approximately 1 in 3 people world wide (Mannino 2018).

Patients suffereing from chronic anemia require frequent monitoring of indicators such as the Hgb levels to track the progression of their disease. Despite this high prevalence, however, the current diagnosis process requires blood tests which causes discomfort and trauma in patients, in addition to incurring high monetary costs. Therefore, our research aims to create machine learning models that enable non-invasive inexpensive diagnosis using patients' nail bed images to predict their Hgb level which will prove particularly crucial for patients in underresourced communities, possibly using random forest algorithms and neural networks.

To this end, we based our research on prior studies that focused on predicting Hgb levels on three regreions of interest: the fingernail beds, the conjunctiva and the palmar creases. With the Hgb estimation algorithm developed via a custom generated MATLAB function which correlated the 3 color RGB channels with the gold-standard measured Hgb levels, the predicted Hgb levels strongly correlate with Hgb levels determined by the clinical hemotology analyzer, with a correlation of determination of 0.995, indicating that this technique can be used to accurately measure a patient's hemoglobin level. We aim to extend the previous research from several perspectives: (1) The previous study did not find significant correlation between blue pixel intensity and gold standard measured Hgb, which we aim to investigate further using more complex machine learning algorithms such as random forests. (2) The study finds that machine learning techniques do not improve Hgb level measurement accuracy given the current sample size of the study population; we would like to refine the algorithm and test on a wider range of data as well as techniques such as neural networks and Bayesian regressions. (3) We aim to develop quality control algorithms accounting for other common irregularities in images of fingernails that could lead to inaccurate Hgb level estimation including presence of abnormal fingernail bed pigmentation, abnormal imaging brightness, and lack of image focus, using convolution of fingernail bed images with edge detection kernels to detect edges within the fingernail beds corresponding to abrupt color changes caused by abnormal fingernail bed pigmentation which results in improved prediction accuracy.

Data

In the first part of our research, we used the TBND_V2 (Transient Biometrics Nails Dataset) on Kaggle [1], which contains unlabeled nail bed images.

In the second part of our research, our data are nail bed images collected from patients enrolled in Dr. Nirmish Shah's clinic. Each patient has four images corresponding to different fingers. The images are processed such that the nail bed is captured in a bounding box while the background of the image is discarded. Colour information of each pixel is extracted from the bounded nail bed images as features.

We know that each pixel can be represented by RGB values, but the RGB colour space contains both colour information and the light information, which is different for each image since photos are taken at different times and settings. To eliminate the inconsistency caused by variation in lightning and background, we used three different colour spaces (HSV, LAB, RGC) to separate the colour information from the lightning information. We computed the mean of each value/channel across the bounded nail bed image for each of these colour spaces and used them as our model input.

Our response variable is blood haemoglobin concentration associated with each nail bed.

Since the dataset is private information of the patients enrolled is Dr. Shah' clinic, we would not include the details of the data in this analysis.

Experiment on TBND_V2 Dataset

We initially experimented with the TBND_V2 dataset found on Kaggle, since there is not enough data from the patients. Given that the data is unlabelled, we tried unsupervised clustering methods on the data, hoping to gain some insights on classification of nail bed images.

To get features, we used VGG16, a convolutional neural network (CNN) in our model. It extracts features from the input images, turning each image into feature vectors (4096 by 1). We removed the final (prediction) layer from the neural network manually, and the new output layer is a fully-connected layer with 4,096 individual nodes. We do this by specifying the "outputs" argument when initialising the model. We therefore get input of our model by using the neural net VGG16 as a feature extractor for the image data.

We then performed a principal component analysis (PCA) on the feature vectors to reduce the dimension of the feature space. For each of the 93 image samples, we now have a corresponding 1 by 4096 feature vector. This means that our model needs to process a 93 by 4096 matrix. To reduce the computational and complexity cost of processing high-dimensional data, we performed principal component analysis (PCA) on the matrix for dimension reduction. We set the parameter to 50 to obtain the top 50 principal components of the feature vector. The principal components are by default sorted in descending order. This means that the first principal component will be able to explain the most variability in the feature vector. It's a linear combination of the feature variables, and its direction captures the most variability. Thus, PCA helps us to reduce the dimension of the features from 4096 to 50 while preserving as much information in the original data as possible.

After getting features, we used Kmean clustering, an unsupervised algorithm that is commonly used in exploratory data analysis, to perform the clustering. The Kmean clustering works as follows: Initialise the centre of each k cluster by shuffling the dataset and randomly selecting K data points without replacement. Iterate the following steps until the assignment of data points to clusters is no longer changing: Compute the sum of the Euclidean distance squared between data points and all centres. Assign each data point to the closest cluster; each cluster is represented by its unique centre point. Compute the cluster's centroid by taking the average of all data points assigned to that cluster.

In our model, we set the hyperparameter k to be 5 and clustered all samples into 5 categories. Based on the features we extracted, each cluster will contain images that are visually similar.

PCA Results

Based on the documentation, the explained_variance_ratio_ function returns the percentage of variability explained by each of the selected components. Running this function gives us the amount of variability that is explained by all the PCs (0.1015 is explained by the first PC, 0.0894 by the second, and 0.0781 by the third etc.)

Then we generated a bar chart to represent the variability explained by different principal components, as well as the cumulative step plot to represent the variability explained by the first most important components.

The neural net in the model functions as a feature extractor for the image data, which is the input of our model. To be more specific, we used VGG16, a convolutional neural network (CNN) in our model. It extracts features from the input images, turning each image into feature vectors (4096 by 1). We removed the final (prediction) layer from the neural network manually, and the new output layer is a fully-connected layer with 4,096 individual nodes. We do this by specifying the "outputs" argument when initialising the model.

The PCA reduces dimensions of our feature vectors. For each of the 93 image samples, we now have a corresponding 1 by 4096 feature vector. This means that our model needs to process a 93 by 4096 matrix. To reduce the computational and complexity cost of processing high-dimensional data, we performed principal

```
In [22]: pca.explained_variance_ratio_

Out[22]: array([0.10151978, 0.08946814, 0.07805712, 0.06449335, 0.05608589,
                0.04782138, 0.04121738, 0.0316945 , 0.03162053, 0.02549183,
                0.0250266 , 0.02351875, 0.0219821 , 0.01871412, 0.0177806 ,
                0.01744947, 0.01555375, 0.01349313, 0.0124064 , 0.01192693,
                0.01143915, 0.01080564, 0.01020705, 0.00966431, 0.00932489,
                0.00897332, 0.0084416 , 0.00756158, 0.00751938, 0.00715409,
                0.00660651, 0.00646016, 0.0060511 , 0.0058278 , 0.00565023,
                0.00556989, 0.00544987, 0.00497867, 0.0046719 , 0.0045056 ,
                0.00435844, 0.00414671, 0.0039429 , 0.00383598, 0.00379014,
                0.00361596, 0.00351077, 0.0034277 , 0.00326044, 0.00311419,
                0.00309905, 0.00298565, 0.0029144 , 0.00277587, 0.00269422,
                0.00252586, 0.00243966, 0.00242474, 0.00231381, 0.00221256,
                0.00215571, 0.00209832, 0.00196334, 0.00195042, 0.00183558,
                0.00178786, 0.00176595, 0.00171314, 0.00166282, 0.00162487,
                0.00158268, 0.00155565, 0.00142908, 0.00142209, 0.00140556,
                0.00135851, 0.00133556, 0.0012662 , 0.00124887, 0.00123438,
                0.00121307, 0.00119807, 0.00115303, 0.00109622, 0.00106344,
                0.00101443, 0.0009791 , 0.00093018, 0.00088837, 0.00087974],
               dtype=float32)
```

Figure 1: PCA variance ratios

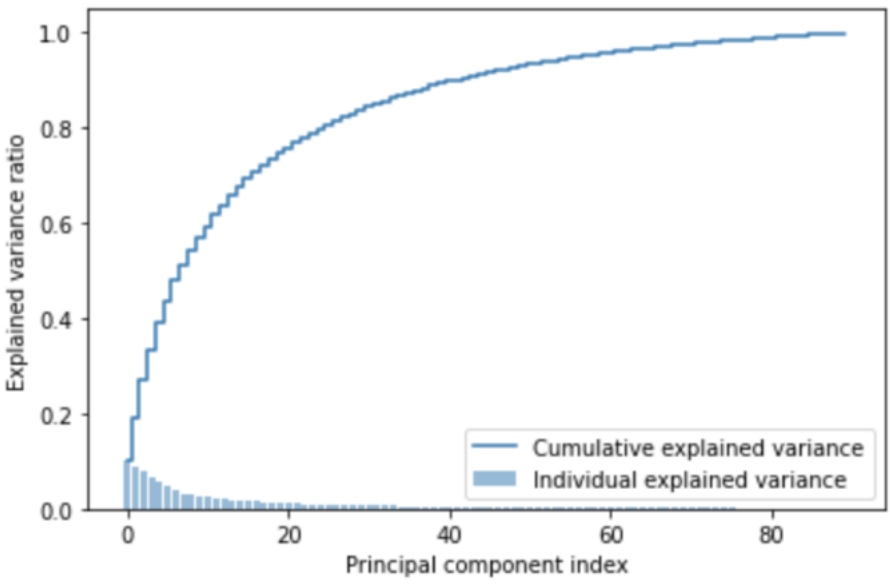

Figure 2: PCA cumulative step plot

component analysis (PCA) on the matrix for dimension reduction. We set the parameter to 50 to obtain the top 50 principal components of the feature vector. The principal components are by default sorted in descending order. This means that the first principal component will be able to explain the most variability in the feature vector. It's a linear combination of the feature variables, and its direction captures the most variability. Thus, PCA helps us to reduce the dimension of the features from 4096 to 50 while preserving as much information in the original data as possible.

In our model, we set the hyperparameter k to be 5 and clustered all samples into 5 categories. Based on the features we extracted, each cluster will contain images that are visually similar.

Function

```
# function to calculate model fit statistics
calc_model_stats <- function(x) {
  glance(extract_fit_parsnip(x)) |>
    select(adj.r.squared, AIC, BIC)
}
```

Packages

```
library(tidyverse)
library(tidymodels)
library(knitr)
```

Load data

```
nail_data <- read_csv("nail_data.csv") |>
  rename(concentration = `Concentration (g/dL)`)

glimpse(nail_data)
```

```
## Rows: 72
## Columns: 15
## $ Image_URL     <chr> "https://storage.labelbox.com/cklyhytg2ulao0774uvyw0poy%~
## $ xmin          <dbl> 248, 340, 469, 644, 347, 511, 640, 744, 201, 322, 458, 6~
## $ xmax          <dbl> 292, 396, 527, 696, 385, 553, 682, 774, 248, 376, 509, 6~
## $ ymin          <dbl> 537, 465, 414, 426, 398, 413, 472, 532, 767, 796, 700, 4~
## $ ymax          <dbl> 582, 523, 474, 482, 461, 472, 522, 570, 803, 838, 726, 5~
## $ Mean_H        <dbl> 8.1922, 13.0864, 12.3233, 13.1862, 11.3486, 11.9142, 10.~
## $ Mean_S        <dbl> 0.2842, 0.2684, 0.2863, 0.2962, 0.2670, 0.2686, 0.2695, ~
## $ Mean_V        <dbl> 0.7824, 0.8003, 0.7863, 0.7895, 0.7871, 0.7820, 0.7820, ~
## $ Mean_L        <dbl> 68.7689, 68.3792, 68.0705, 68.5229, 68.4438, 68.2958, 67~
## $ Mean_A        <dbl> 15.4397, 13.9356, 15.3458, 14.7066, 14.6354, 14.0090, 14~
## $ Mean_B        <dbl> 10.6895, 12.0016, 11.8946, 14.3816, 11.9706, 11.9289, 11~
## $ Mean_Prop_R   <dbl> 0.3999, 0.3980, 0.3943, 0.3994, 0.3972, 0.3997, 0.3977, ~
## $ Mean_Prop_G   <dbl> 0.3077, 0.3125, 0.3147, 0.3140, 0.3113, 0.3115, 0.3107, ~
## $ Mean_Prop_B   <dbl> 0.2925, 0.2895, 0.2910, 0.2866, 0.2915, 0.2888, 0.2916, ~
## $ concentration <dbl> 9.5, 9.5, 9.5, 9.5, 9.5, 9.5, 9.5, 9.5, 9.2, 9.2, 9.2, 9~
```

Split data into training and testing

Split your data into testing and training sets.

```r
set.seed(123)
nail_split <- initial_split(nail_data)
nail_train <- training(nail_split)
nail_test <- testing(nail_split)
```

Specify model

```r
nail_spec <- linear_reg() |>
  set_engine("lm")

nail_spec
```

```
## Linear Regression Model Specification (regression)
##
## Computational engine: lm
```

Create recipe

```r
nail_rec <- recipe(concentration ~ ., data = nail_train) |>
  update_role(Image_URL, new_role = "id") |>
  step_rm(xmin, xmax, ymin, ymax) |>
  step_dummy(all_nominal_predictors()) |>
  step_zv(all_predictors())

nail_rec
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##         id          1
##    outcome          1
##  predictor         13
##
## Operations:
##
## Variables removed xmin, xmax, ymin, ymax
## Dummy variables from all_nominal_predictors()
## Zero variance filter on all_predictors()
```

Create workflow

```r
nail_wflow <- workflow() |>
  add_model(nail_spec) |>
  add_recipe(nail_rec)

nail_wflow
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: linear_reg()
##
```

```
## -- Preprocessor ----------------------------------------------------------------
## 3 Recipe Steps
##
## * step_rm()
## * step_dummy()
## * step_zv()
##
## -- Model -----------------------------------------------------------------------
## Linear Regression Model Specification (regression)
##
## Computational engine: lm
```

Cross validation

Create folds

Create 10-folds.

```
# make 10 folds
set.seed(1)
folds <- vfold_cv(nail_train, v = 10)
folds
```

```
## #  10-fold cross-validation
## # A tibble: 10 x 2
##    splits         id
##    <list>         <chr>
##  1 <split [48/6]> Fold01
##  2 <split [48/6]> Fold02
##  3 <split [48/6]> Fold03
##  4 <split [48/6]> Fold04
##  5 <split [49/5]> Fold05
##  6 <split [49/5]> Fold06
##  7 <split [49/5]> Fold07
##  8 <split [49/5]> Fold08
##  9 <split [49/5]> Fold09
## 10 <split [49/5]> Fold10
```

Conduct cross validation

Conduct cross validation on the 10 folds.

```
set.seed(456)
# Fit model and calculate statistics for each fold
nail_fit_rs <- nail_wflow |>
  fit_resamples(resamples = folds,
                control = control_resamples(extract = calc_model_stats))
```

Summarize assessment CV metrics

Summarize assessment metrics from your CV resamples.

```
collect_metrics(nail_fit_rs, summarize = FALSE) # summarize = FALSE to see individualized output; when
```

```
## # A tibble: 20 x 5
##    id     .metric .estimator .estimate .config
```

```
##    <chr>  <chr>   <chr>          <dbl> <chr>
##  1 Fold01 rmse    standard     1.44    Preprocessor1_Model1
##  2 Fold01 rsq     standard     0.00399 Preprocessor1_Model1
##  3 Fold02 rmse    standard     1.12    Preprocessor1_Model1
##  4 Fold02 rsq     standard     0.485   Preprocessor1_Model1
##  5 Fold03 rmse    standard     1.47    Preprocessor1_Model1
##  6 Fold03 rsq     standard     0.0835  Preprocessor1_Model1
##  7 Fold04 rmse    standard     1.23    Preprocessor1_Model1
##  8 Fold04 rsq     standard     0.129   Preprocessor1_Model1
##  9 Fold05 rmse    standard     1.10    Preprocessor1_Model1
## 10 Fold05 rsq     standard     0.721   Preprocessor1_Model1
## 11 Fold06 rmse    standard     1.59    Preprocessor1_Model1
## 12 Fold06 rsq     standard     0.0332  Preprocessor1_Model1
## 13 Fold07 rmse    standard     1.32    Preprocessor1_Model1
## 14 Fold07 rsq     standard     0.258   Preprocessor1_Model1
## 15 Fold08 rmse    standard     0.730   Preprocessor1_Model1
## 16 Fold08 rsq     standard     0.444   Preprocessor1_Model1
## 17 Fold09 rmse    standard     2.31    Preprocessor1_Model1
## 18 Fold09 rsq     standard     0.0938  Preprocessor1_Model1
## 19 Fold10 rmse    standard     1.13    Preprocessor1_Model1
## 20 Fold10 rsq     standard     0.525   Preprocessor1_Model1
```

Summarize model fit CV metrics

```
map_df(nail_fit_rs$.extracts, ~ .x[[1]][[1]]) |> #model stats are in .extracts column
  summarise(mean_adj_rsq = mean(adj.r.squared), # avg_stats computed over 10 folds
            mean_aic = mean(AIC),
            mean_bic = mean(BIC))
```

```
## # A tibble: 1 x 3
##   mean_adj_rsq mean_aic mean_bic
##          <dbl>    <dbl>    <dbl>
## 1        0.364     162.     183.
```

```
nail_fit_train <- nail_wflow |>
      fit(data = nail_train)

tidy(nail_fit_train) |>
  kable(digits=3)
```

term	estimate	std.error	statistic	p.value
(Intercept)	1356.390	3815.071	0.356	0.724
Mean_H	0.004	0.005	0.741	0.463
Mean_S	31.719	18.789	1.688	0.098
Mean_V	16.880	25.667	0.658	0.514
Mean_L	-0.126	0.245	-0.512	0.611
Mean_A	-0.435	0.272	-1.596	0.118
Mean_B	-0.257	0.328	-0.783	0.438
Mean_Prop_R	-1306.425	3818.067	-0.342	0.734
Mean_Prop_G	-1435.883	3818.314	-0.376	0.709
Mean_Prop_B	-1319.198	3813.765	-0.346	0.731

```
nail_train_pred <- predict(nail_fit_train, nail_train) |>
        bind_cols(nail_train)

rmse_train <- rmse(nail_train_pred, truth = concentration, estimate = .pred)

nail_test_pred <- predict(nail_fit_train, nail_test) |>
        bind_cols(nail_test)

rmse_test <- rmse(nail_test_pred, truth = concentration, estimate = .pred)

model_tibble <- tibble(RMSE_train=rmse_train$.estimate, RMSE_test=rmse_test$.estimate)

model_tibble |>
        kable()
```

RMSE_train	RMSE_test
1.041681	1.445858

References: https://www.kaggle.com/datasets/vicolab/tbnd-v2 https://smartech.gatech.edu/bitstream/handle/1853/61131/MANNINO-DISSERTATION-2018.pdf
Mannino, Robert G. 2018. "A NONINVASIVE, IMAGE-BASED SMARTPHONE APP FOR DIAGNOSING ANEMIA," 12.