Main - Group 12

Qing Gao (qg2175) Yuqiao Liu (yl4278) Ivan Wolansky (iaw2110)
Xiyao Yan (xy2431) Huizhe Zhu (hz2657)

GBM - BASELINE

Step 0 set work directories

```
set.seed(0)
```

Provide directories for training images. Training images and Training fiducial points will be in different subfolders.

```
train_dir <- "../data/train_set/" # This will be modified for different data sets.
train_image_dir <- paste(train_dir, "images/", sep="")
train_pt_dir <- paste(train_dir,  "points/", sep="")
train_label_path <- paste(train_dir, "label.csv", sep="")
```

Step 1: set up controls for evaluation experiments.

In this chunk, we have a set of controls for the evaluation experiments.

- (T/F) process features for training set
- (T/F) process features for test set
- (T/F) fit gbm model on the training set
- (T/F) run evaluation on an independent test set

```
run.feature.train <- TRUE # process features for training set
run.feature.test <- TRUE # process features for test set
run.gbm = TRUE # run the gbm model on the training set
run.test = TRUE # run evaluation on an independent test set
```

Step 2: import data and train-test split

```
#train-test split
info <- read.csv(train_label_path)
n <- nrow(info)
n_train <- round(n*(4/5), 0)
train_idx <- sample(info$Index, n_train, replace = F)
test_idx <- setdiff(info$Index, train_idx)
```

If you choose to extract features from images, such as using Gabor filter, R memory will exhaust all images are read together. The solution is to repeat reading a smaller batch(e.g 100) and process them.

```
n_files <- length(list.files(train_image_dir))

image_list <- list()
for(i in 1:100){
   image_list[[i]] <- readImage(paste0(train_image_dir, sprintf("%04d", i), ".jpg"))
}
```

Fiducial points are stored in matlab format. In this step, we read them and store them in a list.

```
### Function: read fiducial points
### Input: index
### Output: matrix of fiducial points corresponding to the index
readMat.matrix <- function(index){
     return(round(readMat(paste0(train_pt_dir, sprintf("%04d", index), ".mat"))[[1]],0))
}

# Load fiducial points
fiducial_pt_list <- lapply(1:n_files, readMat.matrix)
save(fiducial_pt_list, file="../output/fiducial_pt_list.RData")
```

Step 3: construct features and responses

- The follow plots show how pairwise distance between fiducial points can work as feature for facial emotion recognition.
  - In the first column, 78 fiducials points of each emotion are marked in order.
  - In the second column distributions of vertical distance between right pupil(1) and right brow peak(21) are shown in histograms. For example, the distance of an angry face tends to be shorter than that of a surprised face.
  - The third column is the distributions of vertical distances between right mouth corner(50) and the midpoint of the upper lip(52). For example, the distance of an happy face tends to be shorter than that of a sad face.

`feature.R` should be the wrapper for all your feature engineering functions and options. The function `feature( )` should have options that correspond to different scenarios for your project and produces an R object that contains features and responses that are required by all the models you are going to evaluate later.

- `feature_baseline.R`
- Input: list of images or fiducial point
- Output: an RData file that contains extracted features and corresponding responses

```
source("../lib/feature_baseline.R")
tm_feature_train <- NA
if(run.feature.train){
  tm_feature_train <- system.time(dat_train <- feature_baseline(fiducial_pt_list, train_idx))
}

tm_feature_test <- NA
if(run.feature.test){
  tm_feature_test <- system.time(dat_test <- feature_baseline(fiducial_pt_list, test_idx))
}

save(dat_train, file="../output/feature_train.RData")
save(dat_test, file="../output/feature_test.RData")
```

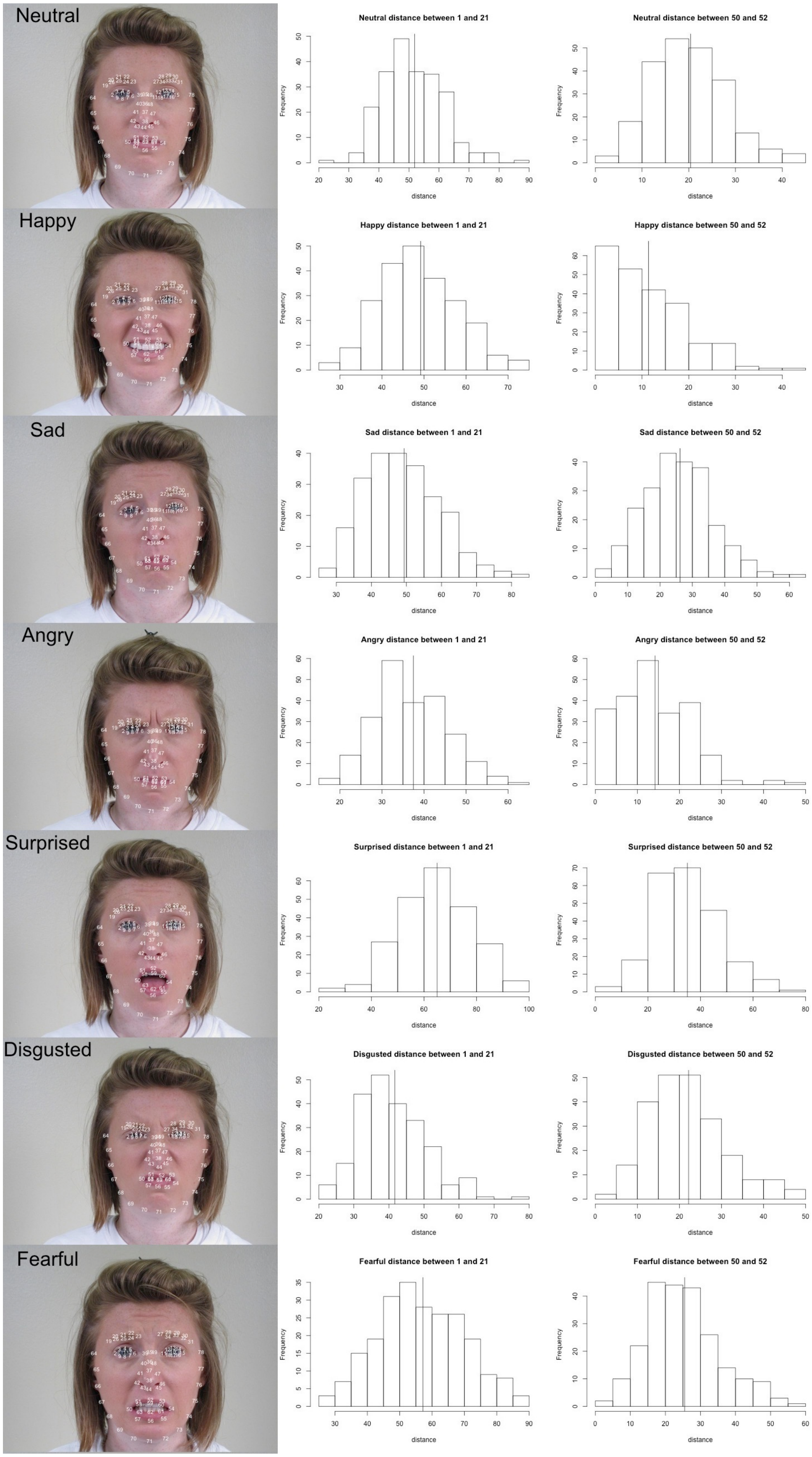

Figure 1: Figure1

Step 4: Train a classification model with training features and responses

Call the train model and test model from library.

`train.R` and `test.R` should be wrappers for all your model training steps and your classification/prediction steps.

- `train_gbm.R`
  - Input: a data frame containing features and labels and a parameter list.
  - Output:a trained model
- `test_gbm.R`
  - Input: the fitted classification model using training data and processed features from testing images
  - Input: an R object that contains a trained classifier.
  - Output: model predictions

```r
load("../output/feature_train.RData")
load("../output/feature_test.RData")
source("../lib/train_gbm.R")
source("../lib/test_gbm.R")
```

```r
# Rename the traget column as 'label'
colnames(dat_train)[6007] = 'label'
colnames(dat_test)[6007] = 'label'
```

Model selection with cross-validation

- Do model selection by choosing among different values of training model parameters.

```r
if(run.gbm){
  # Train the Baseline GBM model
  tm_train_gbm_baseline <- system.time(gbm.baseline <- train_gbm(train.df=dat_train,
                                                                 s=0.001,
                                                                 K=2,
                                                                 n=50))

  # Save the output
  save(gbm.baseline, file="../output/gbm.baseline")
}
```

Step 5: Run test on test images

```r
load('../output/gbm.baseline')
run.test = TRUE
tm_test = NA
if(run.test){
  tm_test_gbm_baseline <- system.time(pred_gbm_baseline <- test_gbm(gbm.fit=gbm.baseline,
                                                                    input.test=dat_test[,-6007],
                                                                    n=50))
}
```

- evaluation

```r
accu_baseline_gbm <- mean(dat_test$label == pred_gbm_baseline)
cat("The accuracy of model: GBM baseline is", mean(dat_test$label == pred_gbm_baseline)*100, "%.\n")
```

```r
# Confusion Matrix
confusionMatrix(as.factor(dat_test$label), as.factor(pred_gbm_baseline))
```

The accuracy of model: GBM baseline is 22.8%.

Note that the accuracy is not high but is better than that of ramdom guess(4.5%).

Summarize Running Time

Prediction performance matters, so does the running times for constructing features and for training the model, especially when the computation resource is limited.

```r
#cat("Time for constructing training features:", tm_train_gbm_baseline[1], "s \n")
#cat("Time for constructing testing features:", tm_test_gbm_baseline[1], "s \n")
cat("Time for training model:", tm_train_gbm_baseline[1], "s \n")
```

Time for training model: 196.2 s.

LASSO LOGISTIC REGRESSION - ADVANCED

Step 0: set up controls for evaluation experiments.

```r
set.seed(1234)
```

In this chunk, we have a set of controls for the evaluation experiments.

- (T/F) building our model with training data
- (number) K, the number of CV folds
- (0 or 1) alpha, lasso or ridge regression
- (T/F) process features

```r
training <- TRUE # train model

K <- 5  # number of CV folds

alpha <- 1 # alpha 1 is lasso, 0 is ridge

run.feature <- TRUE # process features for data
```

Step 1 set work directories

Provide directories for training images. Training images and training fiducial points will be in different subfolders.

```r
if (training) {
  train_dir <- "../data/train_set/" # This will be modified for different data sets.
  train_image_dir <- paste(train_dir, "images/", sep="")
  train_pt_dir <- paste(train_dir,  "points/", sep="")
  train_label_path <- paste(train_dir, "label.csv", sep="")
}
```

Step 2: import data

```
if (training) {
  info <- read.csv(train_label_path)
}
```

If you choose to extract features from images, such as using Gabor filter, R memory will exhaust all images are read together. The solution is to repeat reading a smaller batch(e.g 100) and process them.

```
if (training) {
  n_files <- length(list.files(train_image_dir))

  image_list <- list()
  for(i in 1:100){
   image_list[[i]] <- readImage(paste0(train_image_dir, sprintf("%04d", i), ".jpg"))
  }
}
```

Fiducial points are stored in matlab format. In this step, we read them and store them in a list.

```
#load fiducial points
if (training) {
  #function to read fiducial points
  #input: index
  #output: matrix of fiducial points corresponding to the index
  readMat.matrix <- function(index){
     return(round(readMat(paste0(train_pt_dir, sprintf("%04d", index), ".mat"))[[1]],0))
  }
  train_fiducial_pt_list <- lapply(1:n_files, readMat.matrix)
  save(train_fiducial_pt_list, file="../output/train_fiducial_pt_list.RData")
}
```

Step 3: construct features and responses

- The follow plots show how pairwise distance between fiducial points can work as feature for facial emotion recognition.
  - In the first column, 78 fiducials points of each emotion are marked in order.
  - In the second column distributions of vertical distance between right pupil(1) and right brow peak(21) are shown in histograms. For example, the distance of an angry face tends to be shorter than that of a surprised face.
  - The third column is the distributions of vertical distances between right mouth corner(50) and the midpoint of the upper lip(52). For example, the distance of an happy face tends to be shorter than that of a sad face.

`feature.R` should be the wrapper for all your feature engineering functions and options. The function `feature( )` should have options that correspond to different scenarios for your project and produces an R object that contains features and responses that are required by all the models you are going to evaluate later.

- `feature_regression.R`
- Input: list of images or fiducial point
- Output: an RData file that contains extracted features and corresponding responses

```
source("../lib/feature_regression.R")
tm_feature <- NA
if(training & run.feature){
```

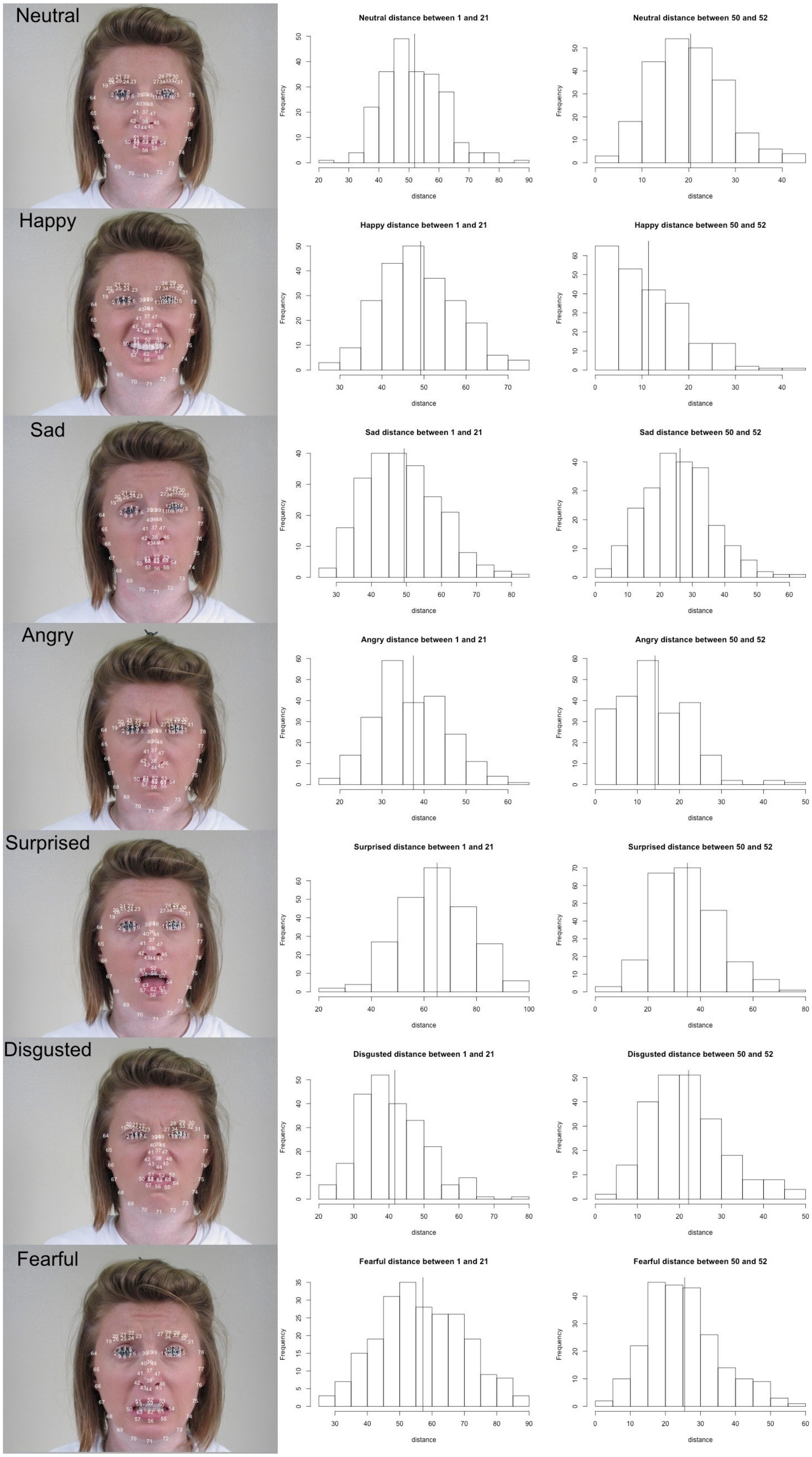

Figure 2: Figure1

```
  tm_feature <- system.time(all_train_data <- feature(train_fiducial_pt_list, seq(1, nrow(info), 1)))
  save(all_train_data, file="../output/feature_train.RData")
}
```

Step 4: Train a classification model with training features and responses

Call the train model and test model from library.

`train.R` and `test.R` should be wrappers for all your model training steps and your classification/prediction steps.

- `regression.train.R`
  - Input: a data frame containing features and labels and a parameter list.
  - Output:a trained model
- `regression.test.R`
  - Input: the fitted classification model using training data and processed features from testing images
  - Input: an R object that contains a trained classifier.
  - Output: model predictions

```
source("../lib/regression.train.R")
source("../lib/regression.cv.R")
```

Model selection with cross-validation

- Do model selection by choosing among different values of the lambda parameter.

```
if(training){
  tm_train_lasso <- system.time(cv_lasso_models <- regression.train(feature_df=all_train_data,
                                                                    alpha=alpha,
                                                                    K=K))
  save(cv_lasso_models, file="../output/cv_lasso_models.RData")
}
```

Visualize cross-validation results.

```
if(training){
  load(file="../output/cv_lasso_models.RData")
  plot(cv_lasso_models)
}
```

Step 5: Run test on test images and evaluate

- MODELS EVALUATION # estimates accuracy by finding the cross-validation accuracy with the optimal lambda parameter already found

```
if (training) {
  lasso_accuracy <- regression.cv(models=cv_lasso_models, data=all_train_data, K=K, alpha=alpha)
  cat("The cross-validation accuracy of the LASSO model is", lasso_accuracy*100, "%.\n")
}
```

The cross-validation accuracy of the LASSO model is 54.28074 %. The cross-validation accuracy of the Ridge model is 49.57992 %.

Summarize Running Time

Prediction performance matters, so does the running times for constructing features and for training the model, especially when the computation resource is limited.

```
cat("Time for constructing features =", tm_feature[1], "s \n")
cat("Time for training the LASSO model =", tm_train_lasso[1], "s \n")
```

Time for constructing features = 4.99 s Time for training the LASSO model = 818.78 s

SUPPLEMENTAL RIDGE:

Step 0: set up controls for evaluation experiments.

```
set.seed(1234)
```

In this chunk, we have a set of controls for the evaluation experiments.

- (T/F) building our model with training data
- (number) K, the number of CV folds
- (0 or 1) alpha, lasso or ridge regression
- (T/F) process features

```
training <- TRUE # train model

K <- 5  # number of CV folds

alpha <- 0 # alpha 1 is lasso, 0 is ridge

run.feature <- TRUE # process features for data
```

Step 1 set work directories

Provide directories for training images. Training images and training fiducial points will be in different subfolders.

```
if (training) {
  train_dir <- "../data/train_set/" # This will be modified for different data sets.
  train_image_dir <- paste(train_dir, "images/", sep="")
  train_pt_dir <- paste(train_dir,  "points/", sep="")
  train_label_path <- paste(train_dir, "label.csv", sep="")
}
```

Step 2: import data

```
if (training) {
  info <- read.csv(train_label_path)
}
```

If you choose to extract features from images, such as using Gabor filter, R memory will exhaust all images are read together. The solution is to repeat reading a smaller batch(e.g 100) and process them.

```
if (training) {
  n_files <- length(list.files(train_image_dir))

  image_list <- list()
  for(i in 1:100){
   image_list[[i]] <- readImage(paste0(train_image_dir, sprintf("%04d", i), ".jpg"))
  }
}
```

Fiducial points are stored in matlab format. In this step, we read them and store them in a list.

```
#load fiducial points
if (training) {
  #function to read fiducial points
  #input: index
  #output: matrix of fiducial points corresponding to the index
  readMat.matrix <- function(index){
     return(round(readMat(paste0(train_pt_dir, sprintf("%04d", index), ".mat"))[[1]],0))
  }
  train_fiducial_pt_list <- lapply(1:n_files, readMat.matrix)
  save(train_fiducial_pt_list, file="../output/train_fiducial_pt_list.RData")
}
```

Step 3: construct features and responses

- The follow plots show how pairwise distance between fiducial points can work as feature for facial emotion recognition.
  - In the first column, 78 fiducials points of each emotion are marked in order.
  - In the second column distributions of vertical distance between right pupil(1) and right brow peak(21) are shown in histograms. For example, the distance of an angry face tends to be shorter than that of a surprised face.
  - The third column is the distributions of vertical distances between right mouth corner(50) and the midpoint of the upper lip(52). For example, the distance of an happy face tends to be shorter than that of a sad face.

`feature.R` should be the wrapper for all your feature engineering functions and options. The function `feature( )` should have options that correspond to different scenarios for your project and produces an R object that contains features and responses that are required by all the models you are going to evaluate later.

- `feature_regression.R`
- Input: list of images or fiducial point
- Output: an RData file that contains extracted features and corresponding responses

```
source("../lib/feature_regression.R")
tm_feature <- NA
if(training & run.feature){
  tm_feature <- system.time(all_train_data <- feature(train_fiducial_pt_list, seq(1, nrow(info), 1)))
  save(all_train_data, file="../output/feature_train.RData")
}
```

Step 4: Train a classification model with training features and responses

Call the train model and test model from library.

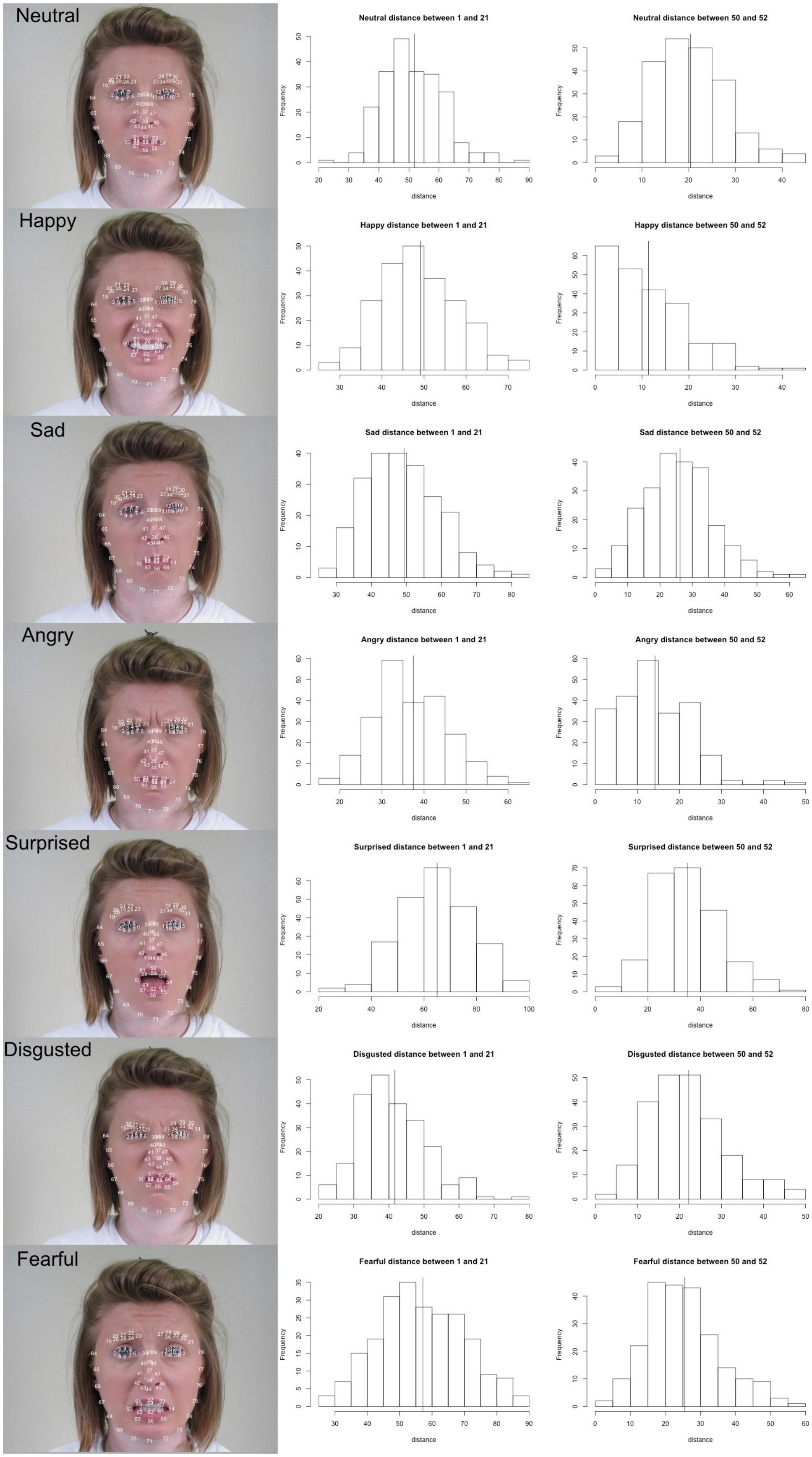

Figure 3: Figure1

`train.R` and `test.R` should be wrappers for all your model training steps and your classification/prediction steps.

- `regression.train.R`
  - Input: a data frame containing features and labels and a parameter list.
  - Output:a trained model
- `regression.test.R`
  - Input: the fitted classification model using training data and processed features from testing images
  - Input: an R object that contains a trained classifier.
  - Output: model predictions

```r
source("../lib/regression.train.R")
source("../lib/regression.cv.R")
```

Model selection with cross-validation

- Do model selection by choosing among different values of the lambda parameter.

```r
if(training){
  tm_train_ridge <- system.time(cv_ridge_models <- regression.train(feature_df=all_train_data,
                                                                    alpha=alpha,
                                                                    K=K))

  save(cv_ridge_models, file="../output/cv_ridge_models.RData")
}
```

Visualize cross-validation results.

```r
if(training){
  load(file="../output/cv_ridge_models.RData")
  plot(cv_ridge_models)
}
```

Step 5: Run test on test images and evaluate

- MODELS EVALUATION # estimates accuracy by finding the cross-validation accuracy with the optimal lambda parameter already found

```r
if (training) {
  ridge_accuracy <- regression.cv(models=cv_ridge_models, data=all_train_data, K=K, alpha=alpha)
  cat("The cross-validation accuracy of the Ridge model is", ridge_accuracy*100, "%.\n")
}
```

The cross-validation accuracy of the Ridge model is 49.57992 %.

Summarize Running Time

Prediction performance matters, so does the running times for constructing features and for training the model, especially when the computation resource is limited.

```r
cat("Time for constructing features =", tm_feature[1], "s \n")
cat("Time for training the Ridge model =", tm_train_ridge[1], "s \n")
```

Time for constructing features = 4.99 s Time for training the Ridge model = 1716.54 s

SUPPLEMENTAL PCA w/ SVM:

```
#_____________________________________________________#
# loading in feature data
load("train_euc.RData")

set.seed(5243)
sample<-sample(1:2500,250,replace = F)
SAMPLE<-sample(1:2250,250,replace = F)
data_train<-train_set_euc[-sample,]
data_validation<-data_train[SAMPLE,]
data_train<-data_train[-SAMPLE,]
data_test<-train_set_euc[sample,]
```

```
#__________Directly build SVM model____________________#
traindata<-rbind(data_train,data_validation)
testdata<-data_test
svm<-svm(traindata[,-3004],y = as.factor(traindata[,3004]))
p<-predict(svm,newdata = testdata[,-3004])
MSE<-mean(p != as.factor(testdata[,3004]))
```

```
#___________PCA&SVM(TRAIN,VALIDATION & TEST SET)_______________________#
train_pca<-prcomp(data_train[,-3004])
sdev<-train_pca$sdev
sum(sdev[1:500]^2)/sum(sdev^2)
validation_pca<-scale(data_validation[,-3004],
                      train_pca$center,
                      train_pca$scale) %*% train_pca$rotation
svm_list<-list()
MSE_validation<-c()
for(i in 1:500){
  svm<-svm(train_pca$x[,1:i],y = as.factor(data_train[,3004]))
  svm_list[[i]]<-svm
  mean(svm$fitted != as.factor(data_train[,3004]))
  p<-predict(svm,newdata = validation_pca[,1:i])
  MSE_validation[i]<-mean(p != as.factor(data_validation[,3004]))
}
plot(x = 1:500,MSE_validation)
which.min(MSE_validation)
#I've just got svm model using the first 500 PC because of limited RAM(TAT)#
```

```
#_________select first 57th PC to build the SVM model_________#
model_svm<-svm_list[[57]]

mypredict<-function(data){
  test_pca<-scale(data, train_pca$center, train_pca$scale) %*% train_pca$rotation
  p<-predict(model_svm,newdata = test_pca[,1:57])
  return(p)
}
p<-mypredict(data_test[,-3004])
mean(p != as.factor(data_test[,3004]))

save(train_pca,model_svm,mypredict,file = "model_pca+svm.RData")
save(data_test,file = "test_data.RData")
```

SVM w/ PCA accuracy: 50.8% Time for training SVM w/ PCA model > 3 hours.

SUPPLEMENTAL VOTING CLASSIFIER (Logistic Regression, Ridge, SVM)

Packages for python

```
import os
import time
import numpy as np
import pandas as pd
from scipy.io import loadmat
from scipy.spatial.distance import cdist
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from sklearn.linear_model import RidgeClassifier
from sklearn.model_selection import train_test_split
from sklearn.exceptions import ConvergenceWarning
from  warnings import simplefilter

simplefilter("ignore", category=ConvergenceWarning)
```

Features Extraction

```
def dist(file):
    name = list(file.keys())[-1]
    dat = file[name]
    dist_mat = cdist(dat, dat, 'euclidean')
    return dist_mat[np.triu_indices(78, 1)].flatten()

# Read Train datasets
s = time.time()
train_label = pd.read_csv('../data/train_set/label.csv')
file_path = [os.path.join('../data/train_set/points/',
                          str(i).zfill(4) + '.mat') for i in range(1, 2501)]

X_train = np.array(list(map(dist, (loadmat(f) for f in file_path))))
y_train = train_label['emotion_idx']
print(f'Training features extraction time: {time.time()-s:.4f} seconds')
groups = y_train

# # Read Test datasets
# s = time.time()

# file_path = [os.path.join('../data/test_set/points/', str(i).zfill(4) + '.mat') for i in range(1, 250
# X_test = np.array(list(map(dist, (loadmat(f) for f in file_path))))
# print(f'Test features extraction time: {time.time()-s:.4f} seconds')
```

Set the model parameters

```
svm = SVC(kernel='linear', C=.0001, decision_function_shape='ovo')
ridge = RidgeClassifier(alpha=85)
logistic = LogisticRegression()
pca = PCA(n_components=128)

estimators = [('svm', svm), ('ridge', ridge), ('logi', logistic)]
voting = VotingClassifier(estimators=estimators, voting='hard')
```

Cross-Validation

```
train_scores = []
test_scores = []
times = []
X = X_train
y = y_train
for i in range(10):
    s = time.time()
    X_train, X_validation, y_train, y_validation = train_test_split(X,
                                                                    y,
                                                                    test_size=.2,
                                                                    shuffle=True,
                                                                    stratify=y)
    X_train_pca = pca.fit_transform(X_train)
    X_validation_pca = pca.transform(X_validation)

    voting.fit(X_train_pca, y_train)

    test_pred = voting.predict(X_validation_pca)
    times.append(time.time() - s)

    train_pred = voting.predict(X_train_pca)

    train_score = accuracy_score(y_train, train_pred)
    test_score = accuracy_score(y_validation, test_pred)

    train_scores.append(train_score)
    test_scores.append(test_score)
```

Predicted Train and Test Score by CVs

```
print(f'The mean Train Score of Voting is {np.array(train_scores).mean():.4f}')
print(f'The mean Test Score of Voting is {np.array(test_scores).mean():.4f}')
print(f'The mean Time per CVs of Voting is {np.array(times).mean():.4f} seconds')
```

Mean test score of voting is 52.32 % Mean time per CV of voting is 2.57 s.

SUPPLEMENTAL: SVM + BAGGING

Packages for python

```
import os
import time
import numpy as np
import pandas as pd
from scipy.io import loadmat
from scipy.spatial.distance import cdist
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
```

Features Extraction

```
def dist(file):
    name = list(file.keys())[-1]
    dat = file[name]
    dist_mat = cdist(dat, dat, 'euclidean')
    return dist_mat[np.triu_indices(78, 1)].flatten()

# Read Train datasets
s = time.time()
train_label = pd.read_csv('../data/train_set/label.csv')
file_path = [os.path.join('../data/train_set/points/',
                      str(i).zfill(4) + '.mat') for i in range(1, 2501)]

X_train = np.array(list(map(dist,
                    (loadmat(f) for f in file_path))))
y_train = train_label['emotion_idx']
print(f'Training features extraction time: {time.time()-s:.4f} seconds')
groups = y_train



# # Read Test datasets
# s = time.time()

# file_path = [os.path.join('../data/test_set/points/', str(i).zfill(4) + '.mat') for i in range(1, 250
# X_test = np.array(list(map(dist, (loadmat(f) for f in file_path))))
# print(f'Test features extraction time: {time.time()-s:.4f} seconds')
```

Set the model parameters

```
svm = SVC(kernel='linear', C=.0001, decision_function_shape='ovo')
svm_bag = BaggingClassifier(svm, n_estimators=500, n_jobs=1, verbose=True)
pca = PCA(n_components=128)
```

Cross-Validation

```
train_scores = []
test_scores = []
times = []
X = X_train
y = y_train
for i in range(5):
   s = time.time()
   X_train, X_validation, y_train, y_validation = train_test_split(X,
                                                                   y,
                                                                   test_size=.2,
                                                                   shuffle=True,
                                                                   stratify=y)

   X_train_pca = pca.fit_transform(X_train)
   X_validation_pca = pca.transform(X_validation)

   svm_bag.fit(X_train_pca, y_train)
   test_pred = svm_bag.predict(X_validation_pca)

   times.append(time.time() - s)

   train_pred = svm_bag.predict(X_train_pca)

   train_score = accuracy_score(y_train, train_pred)
   test_score = accuracy_score(y_validation, test_pred)

   train_scores.append(train_score)
   test_scores.append(test_score)
```

Predicted Train and Test Score by CVs

```
print(f'The mean Train Score of SVM + Bagging is {np.array(train_scores).mean():.4f}')
print(f'The mean Test Score of SVM + Bagging is {np.array(test_scores).mean():.4f}')
print(f'The mean Time per cvs of SVM + Bagging is {np.array(times).mean():.4f} seconds')
```

Mean test score of SVM + Bagging is 51.6 % Mean time per CV of SVM + Bagging is 21.7 s.