main

2017/3/12

baseline model

Step 0: Load packages and data

```
if(!require("gbm")){
  install.packages("gbm")
}
```

```
## Loading required package: gbm
```

```
## Loading required package: survival
```

```
## Loading required package: lattice
```

```
## Loading required package: splines
```

```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.1
```

```
if(!require("caret")){
  install.packages("caret")
}
```

```
## Loading required package: caret
```

```
## Warning: package 'caret' was built under R version 3.3.2
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:survival':
##
##     cluster
```

```
if(!require("rpart")){
  install.packages("rpart")
}
```

```
## Loading required package: rpart
```

```
if(!require("e1071")){
  install.packages("e1071")
}
```

```
## Loading required package: e1071
```

```
## Warning: package 'e1071' was built under R version 3.3.2
```

```
if(!require("kernlab")){
  install.packages("kernlab")
}
```

```
## Loading required package: kernlab
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
library("gbm")
```

```
dat_train<-read.csv("../data/sift_features.csv")
dat_train<-t(dat_train)
```

Step 1: set up controls for evaluation experiments.

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

```
run.cv=TRUE # run cross-validation on the training set
K <- 5  # number of CV folds
run.feature.train=TRUE # process features for training set
run.test=TRUE # run evaluation on an independent test set
run.feature.test=TRUE # process featur  es for test set
```

Using cross-validation or independent test set evaluation, we compare the performance of different classifiers or classifiers with different specifications. In this example, we use GBM with different `depth`. In the following chunk, we list, in a vector, setups (in this case, `depth`) corresponding to models that we will compare. In your project, you maybe comparing very different classifiers. You can assign them numerical IDs and labels specific to your project.

```
model_values <- seq(3, 11, 2)
model_labels = paste("GBM with depth =", model_values)
```

Step 2: import training images class labels.

```
label_train <- read.csv("../data/labels.csv")
label_train <- as.numeric(label_train[1:dim(label_train)[1],])
data<-as.data.frame(cbind(dat_train,label_train))
```

```
### set train & test data
index <- 1:nrow(data)
trainindex <- sample(index, 0.8*nrow(data),replace=F)
testset1 <- data[-trainindex,]
trainset1 <-data[trainindex,]
```

Step 3: Train a classification model with training images

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.R` + Input: a path that points to the training set features. + Input: an R object of training sample labels. + Output: an RData file that contains trained classifiers in the forms of R objects: models/settings/links to external trained configurations. + `test.R` + Input: a path that points to the test set features. + Input: an R object that contains a trained classifiers. + Output: an R object of class label predictions on the test set. If there are multiple classifiers under evaluation, there should be multiple sets of label predictions.

```
source("../lib/train.R")
source("../lib/test.R")
```

Step 4: Model selection with cross-validation

- Do model selection by choosing among different values of training model parameters, that is, the interaction depth for GBM in this example.

```
source("../lib/cross_validation.R")

if(run.cv){
  err_cv <- array(dim=c(length(model_values), 2))
  for(k in 1:length(model_values)){
    cat("k=", k, "\n")
    err_cv[k,] <- cv.function(trainset1[,1:5000], trainset1[,5001], model_values[k], K)
  }
  save(err_cv, file="../output/err_cv.RData")
}
```

```
## k= 1
## k= 2
## k= 3
## k= 4
## k= 5
```

- Visualize cross-validation results.

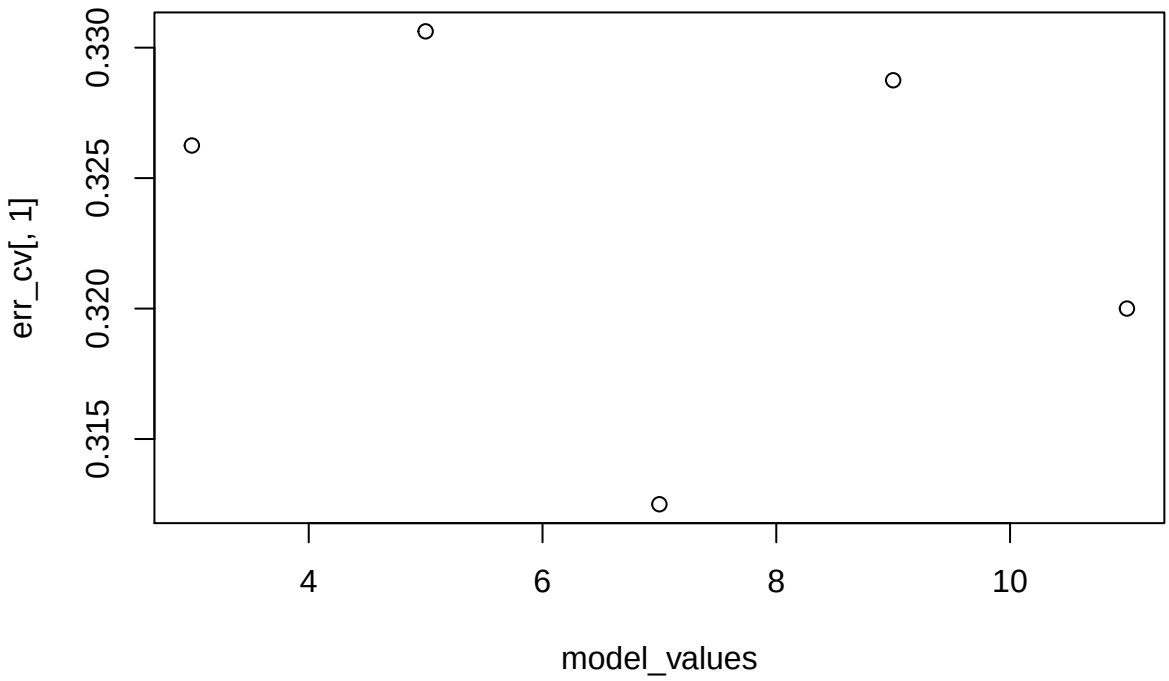
```
err_cv[,1]
```

```
## [1] 0.326250 0.330625 0.312500 0.328750 0.320000
```

```
mean(err_cv[,1])
```

```
## [1] 0.323625
```

```
plot(model_values, err_cv[,1])
```



- Choose the "best" parameter value

```
model_best=model_values[1]
if(run.cv){
  model_best <- model_values[which.min(err_cv[,1])]
}

par_best <- list(depth=model_best)
```

- Train the model with the entire training set using the selected model (model parameter) via cross-validation.

```
tm_train=NA
tm_train <- system.time(fit_train <- train(trainset1[,1:5000], trainset1[,5001], par_best))
```

```
## Warning in gbm.fit(x = dat_train, y = label_train, n.trees = 150,
## distribution = "bernoulli", : variable 139: V139 has no variation.

## Warning in gbm.fit(x = dat_train, y = label_train, n.trees = 150,
## distribution = "bernoulli", : variable 315: V315 has no variation.

## Warning in gbm.fit(x = dat_train, y = label_train, n.trees = 150,
## distribution = "bernoulli", : variable 2488: V2488 has no variation.

## Warning in gbm.fit(x = dat_train, y = label_train, n.trees = 150,
## distribution = "bernoulli", : variable 3689: V3689 has no variation.

## Warning in gbm.perf(fit_gbm, method = "OOB", plot.it = FALSE): OOB
## generally underestimates the optimal number of iterations although
## predictive performance is reasonably competitive. Using cv.folds>0 when
## calling gbm usually results in improved predictive performance.
```

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

Step 5: Make prediction

Feed the final training model with the completely holdout testing data.

```
tm_test=NA
if(run.test){
  tm_test <- system.time(pred_test <- test(fit_train, testset1[,1:5000]))
  save(pred_test, file="../output/pred_test.RData")
}
sum(pred_test != testset1[,5001])/length(testset1[,5001])
```

```
## [1] 0.3375
```

Summarize Running Time

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

```
cat("Time for training model=", tm_train[1], "s \n")
```

```
## Time for training model= 93.988 s
```

```
cat("Time for making prediction=", tm_test[1], "s \n")
```

```
## Time for making prediction= 0.084 s
```

Advanced model

For the feature extraction part, we tried SIFT features to extract feature again. For the feature selction,we tried bag of words,LASSO,PCA to reduce dimension features.

As for the models, we briefly tested several models with different feature extraction and selection combination,like gbm, SVM, Randomforest, Xgboost, logistic, NaiveBayes. From all of these primary models, we found PCA features combined with SVM model generally perform the best in every model.

The following are the training procedure:

load data

```
set.seed(340697)
setwd("~/spr2017-proj3-group13")
```

5000 sifted features

```
###feature_sift<-read.csv("../data/sift_features.csv")
###feature_new<-t(feature_sift)
```

300 pca features

```
feature<-read.csv("../output/pca_features3.csv")
feature_new<-feature
```

load label

```
label_train <- read.csv("../data/labels.csv")
y <- label_train[1:2000,]
data<-as.data.frame(cbind(feature_new,y))
```

set apart train & test data

```
index <- 1:nrow(data)
trainindex <- sample(index, 0.8*nrow(data),replace=F)
testset1 <- data[-trainindex,]
trainset1 <-data[trainindex,]
```

train with svm

```
source("../lib/train.R")
source("../lib/test.R")
#mymodel<-train.svm.cv(trainset1)

tm_train=NA
tm_train <- system.time(mymodel<-train.svm.cv(trainset1))
```

```
## [1] "fold k=1 finished"
## [1] "fold k=2 finished"
## [1] "fold k=3 finished"
## [1] "fold k=4 finished"
## [1] "fold k=5 finished"
## [1] "the mean of training error rate is: 0.223125"
```

```
save(mymodel, file="../output/fit_train_advance.RData")

# the cross validation part and tuning parameter part are in the train.R
```

Feed the final training model with the completely holdout testing data.

```
tm_test=NA
tm_test <- system.time(pred_test <-test.svm(mymodel,testset1))
save(pred_test, file="../output/pred_test_advance.RData")
# error rate
pred_test
```

```
## [1] 0.1975
```

Summarize Running Time

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

```
cat("Time for training model=", tm_train[1], "s \n")
```

```
## Time for training model= 17.916 s
```

```
cat("Time for making prediction=", tm_test[1], "s \n")
```

```
## Time for making prediction= 0.34 s
```