Project 3 - group 11

Ruxue Peng, RaphaÃfÂ«l Ruscassie, Yifei Tang, Connie Zhou, Hongyi Zhu

Mar 22, 2017

Step 0: Check for installed packages

```
library("EBImage")
library("gbm")
library("glmnet")
library(RSNNS)
library(e1071)
library(caret)

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

Step 1: import SIFT features and class labels

```
#############################################################################
## Data Preparation ##
Fea = read.csv("../data/sift_features/sift_features.csv",as.is = T, header = T)
Fea = t(Fea) #the features as column, data object as row

# for randomForest to use classification unstead of regression,
# we need to factorize y
y = read.csv("../data/labels.csv",as.is = T) #0 for chicken(not a dog), 1 for dog
```

Step 2: Our baseline

The baseline consists of a boosted decision stumps.

```
#############################################################################

ptm = proc.time()
nzv_cols <- nearZeroVar(Fea)
if(length(nzv_cols) > 0) baseline_features <- Fea[, -nzv_cols]

Data = cbind(y,baseline_features)
Data = as.data.frame(Data)
# change colnames
names = c("y",paste0("Fea",1:(length(Data)-1)))
colnames(Data) = names

## divide train-test(80%-20%)
set.seed(200)
index = sample(1:2000,1600)
Train = Data[index,]
Test = Data[-index,]
```

```
baseline_features_time = proc.time() -ptm

#############################################################################

#start the clock
ptm <- proc.time()
## PS. this version of ADABOOST needs response to be {0,1}
b = gbm(y~.,data = Train,
        distribution = "adaboost",
        n.trees = 1200,
        shrinkage=0.01,
        interaction.depth = 1, # stump
        bag.fraction = 0.8,
        train.fraction = 1,
        cv.folds=5,
        keep.data = TRUE,
        verbose = "CV")
best_iter = gbm.perf(b, method="cv", plot=FALSE)
# Stop the clock
baseline_training_time = proc.time() - ptm #228 sec
#############################################################################
##predict time
ptm <- proc.time()
y_adj = ifelse(unlist(y)==0,-1,1)
f.predict = predict(b,Train,best_iter)
train_rate = mean(sign(f.predict)!= y_adj[index]) #0.195
t.predict = predict(b,Test,best_iter)
test_rate = mean(sign(t.predict)!= y_adj[-index]) #0.255
baseline_prediction_time = proc.time() - ptm # 12 sec
#############################################################################
baseline_train = 1-train_rate
baseline_test = 1 - test_rate
cat(paste("Baseline Train Accuracy: ", baseline_train, "\n", "Baseline Test Accuracy", baseline_test))
```

```
## Baseline Train Accuracy:  0.804375
##  Baseline Test Accuracy 0.73
```

Step 3: Our exploration

In the beginning, we tried to select feature intuitively without models.

Passive Aggressive Classifer:

We also had a gradient boosting tree in Python, with an accuracy around 75% for test and 80% for train.

Step 4: Feature selection and extraction

We used LASSO regression coefficients to select the features for the MLP model.

Here, P is the penalty for the parameters.

```
ptm = proc.time()
feature_selection_mlp(Fea,y)
feature_selection_mlp_time = proc.time() - ptm
```

```r
feature_trimmer <- function(mat)
  {
    # Preprocess
    # swap the labels
    cn=colnames(mat)
    rn=rownames(mat)
    # transpose the matrix
    mat=transpose(mat)
    colnames(mat)=rn
    rownames(mat)=cn

    # find features that are highly correlated and removing them
    correlationMatrix = cor(mat)
    highlyCorrelated=findCorrelation(correlationMatrix,cutoff=0.75)
    mat=subset(mat,select=-highlyCorrelated)

    # removing features with column sum <0.005
    mat=as.data.frame(mat)
    mat=mat[, colSums(mat)>0.005]

    # find near zero variance
    nzv_cols <- nearZeroVar(mat)
    if(length(nzv_cols) > 0) mat <- mat[, -nzv_cols]

    # calculate column sum by class
    ck=mat[1:1000,]
    dog=mat[1001:2000,]
    csck=colSums(ck)
    csdog=colSums(dog)
    # quotient might be a better indicator
    quo=abs(csck/csdog)
    rm(csck,csdog,ck,dog)

    # rbind the quotient row to the end of the matrix
    quo=t(as.data.frame(quo))

    # get rid of features with quotient within 0.01 of 1
    mat=mat[,(quo<=0.99 | quo>1.01)]

    preprocessed=preProcess(mat, method=c("pca", "nzv"))
    mat = predict(preprocessed, mat)

    return(mat)
  }
```

Figure 1: image

In [23]:
```
#Import libraries:
import pandas as pd
import numpy as np
from sklearn.linear_model import PassiveAggressiveClassifier
from sklearn import cross_validation, metrics   #Additional scklearn functions
from sklearn.grid_search import GridSearchCV   #Performing grid search

import matplotlib.pylab as plt
%matplotlib inline
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 12, 4

train = pd.read_csv('F:\second_term\ADS\proj3\Train.csv')
test = pd.read_csv('F:\second_term\ADS\proj3\Test.csv')
target = 'y' #this is y
IDcol = 'ID'
```

In [71]:
```
#Functions for building the model
def modelfit(alg, dtrain, dtest, predictors, performCV=True, printFeatureImportance=True, cv_folds=5):
    #Fit the algorithm on the data
    alg.fit(dtrain[predictors], dtrain['y'])

    #Predict training set:
    dtrain_predictions = alg.predict(dtrain[predictors])
    #dtrain_predprob = alg.predict_proba(dtrain[predictors])[:,1]
    dtest_predictions = alg.predict(dtest[predictors])

    #Perform cross-validation:
    if performCV:
        cv_score = cross_validation.cross_val_score(alg, dtrain[predictors], dtrain['y'], cv=cv_folds, sco
ring='roc_auc')

    #Print model report:
    print ("\nModel Report")
    print ("Accuracy(train): %.4g" % metrics.accuracy_score(dtrain['y'].values, dtrain_predictions))
```

Figure 2: image

```
    print ("Accuracy(test) : %.4g" % metrics.accuracy_score(dtest['y'].values, dtest_predictions))
    #print ("AUC Score (Train): %f" % metrics.roc_auc_score(dtrain['y'], dtrain_predprob))

    if performCV:
        print ("CV Score : Mean - %.7g | Std - %.7g | Min - %.7g | Max - %.7g" %
(np.mean(cv_score),np.std(cv_score),np.min(cv_score),np.max(cv_score)))
```

In [76]:
```
predictors = [x for x in train.columns if x not in [target, IDcol]]
pa = PassiveAggressiveClassifier(C=1.0, fit_intercept=True, n_iter=500, shuffle=True, verbose=0,
                                 loss='squared_hinge', n_jobs=1, random_state=10, warm_start=False, class_weigh
t=None)
modelfit(pa, train,test, predictors)#train : 81% test: 80%
```

```
Model Report
Accuracy(train): 0.8106
Accuracy(test) : 0.8
CV Score : Mean - 0.9001862 | Std - 0.01426189 | Min - 0.8758648 | Max - 0.920691
```

Figure 3: image

$$P = \lambda\left\{(1-\alpha)\|\beta\|_2^2 / \ 2 + \alpha\|\beta\|_1\right\}$$

where,

$\alpha = 1 \rightarrow Lasso$

$\alpha = 0 \rightarrow Ridge$

$\alpha = (0,1) \rightarrow Elasticnet$

$\lambda \rightarrow$ Tuning parameter

$\beta \rightarrow$ Parameter estimates

Figure 4: image

```
ptm = proc.time()
feature_selection_svm(Fea,y)
feature_selection_svm_time = proc.time() - ptm
```

Step 5: Train a classification model with training dataset

bold Model 1: Multilayer perceptron. mlp_tune returns a matrix of training and testing accuracy, size, or number of hidden layer(s), and maxit, or maximum number of iterations to learn. Here, one can view the MLP result and rank them by testing accuracy to select the best parameters.

```
Train_mlp=read.csv("../data/Train_nn.csv")
Test_mlp=read.csv("../data/Test_nn.csv")

set.seed(129)
ptm = proc.time()
result_mlp=mlp_tune(Train_mlp, Test_mlp)
mlp_tune_time = proc.time() - ptm

View(result_mlp)
```

And the MLP results are:

```
set.seed(200)
ptm = proc.time()
MLP=mlp_train(size=27, maxit=17, Train = Train_mlp)
mlp_train_time = proc.time() - ptm

ptm = proc.time()
pred_mlp=mlp_test(MLP, Test = Test_mlp, Train = Train_mlp)
```

```
## It takes 0.349999999999909 sec to predict.
##  Accuracy on the Train set:  93.625 %
##  Accuracy on the Test set:  89.25 %
```

```
mlp_predict_time = proc.time() - ptm
```

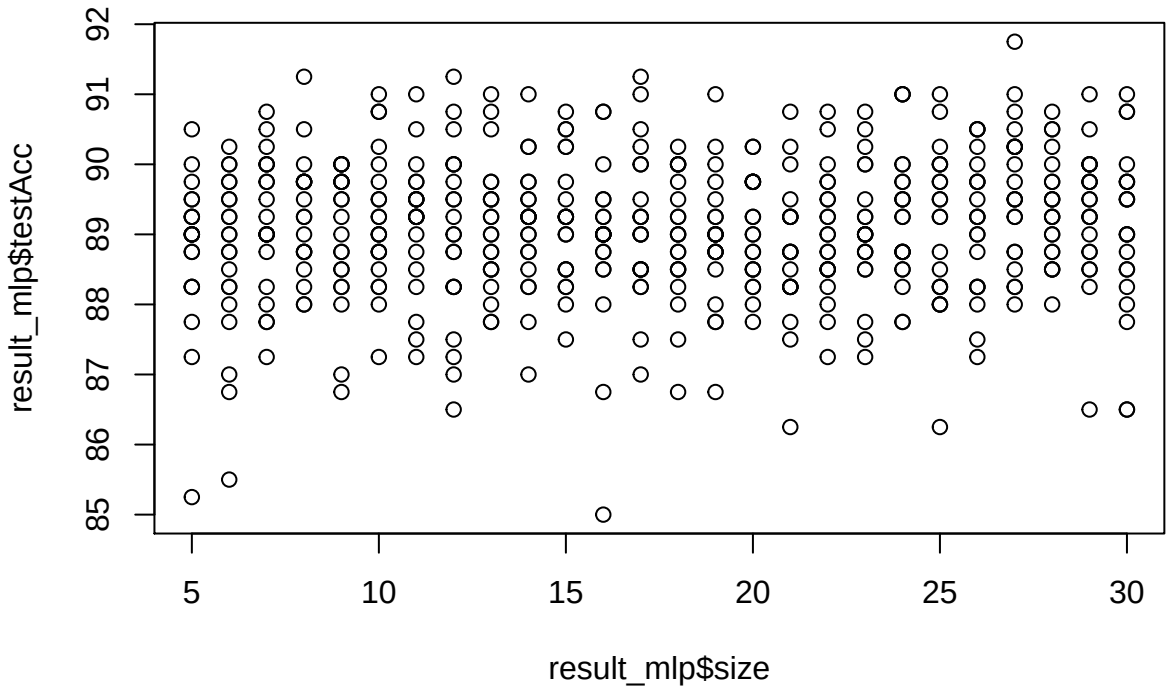
bold Visualize cross-validation results for MLP.

As you can probably see, testing accuracy is negatively affected by the differences between training and testing, since training will usually be higher than testing. Interestingly, testing accuracy is also positively related to training accuracy–the better the training accuracy, the better the testing accuracy tends to be. Moreover, to increase testing accuracy, one can look to increase the number of iteration; though, blindly increasing iteration could result in overfitting.

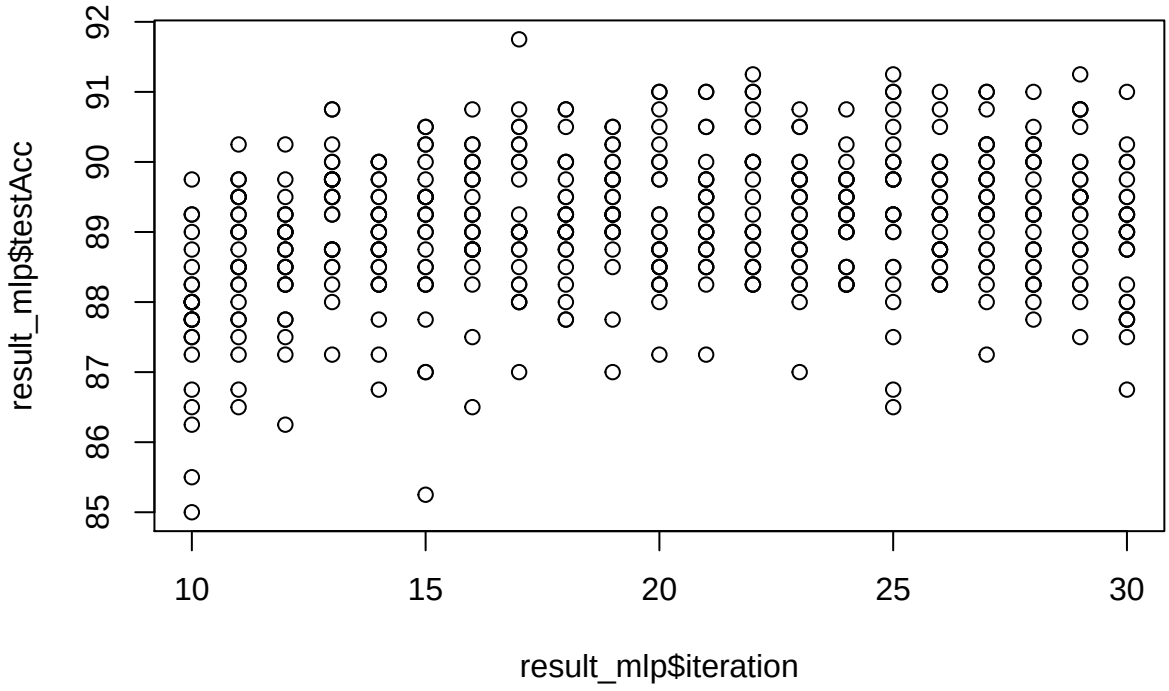
```
cor(result_mlp)
```

```
##                   size iteration     testAcc   trainAcc   difference
## size       1.000000000 0.0000000  0.08085601 0.04804604  0.001810472
## iteration  0.000000000 1.0000000  0.23554079 0.87780077  0.766962239
## testAcc    0.080856009 0.2355408  1.00000000 0.31318373 -0.242548168
## trainAcc   0.048046045 0.8778008  0.31318373 1.00000000  0.844237101
## difference 0.001810472 0.7669622 -0.24254817 0.84423710  1.000000000
```

```
plot(result_mlp$size, result_mlp$testAcc)
```




```
plot(result_mlp$iteration, result_mlp$testAcc)
```



Now we introduce another classifier: **bold** support vector machine.

First we tune the parameter like before.

```
Train_svm=read.csv("../data/Train_svm.csv")
Test_svm=read.csv("../data/Test_svm.csv")

ptm = proc.time()
result_svm=svm_tune(Train_svm, Test_svm)
svm_tune_time = ptm = proc.time() - ptm

View(result_svm)
```

Using the best tuned parameters, we train the model and give predictions.

```
ptm = proc.time()
SVM= svm_train(cost=0.0015, Train = Train_svm)
svm_train_time = ptm = proc.time() - ptm

ptm = proc.time()
pred_svm = svm_test(SVM, Test = Test_svm, Train = Train_svm)
```

```
## It takes 1.08000000000038 seconds to train.
##  Accuracy on the Train set:  93.0625 %
##  Accuracy on the Test set:  90.5 %
```

```
svm_predict_time = ptm = proc.time() - ptm
```

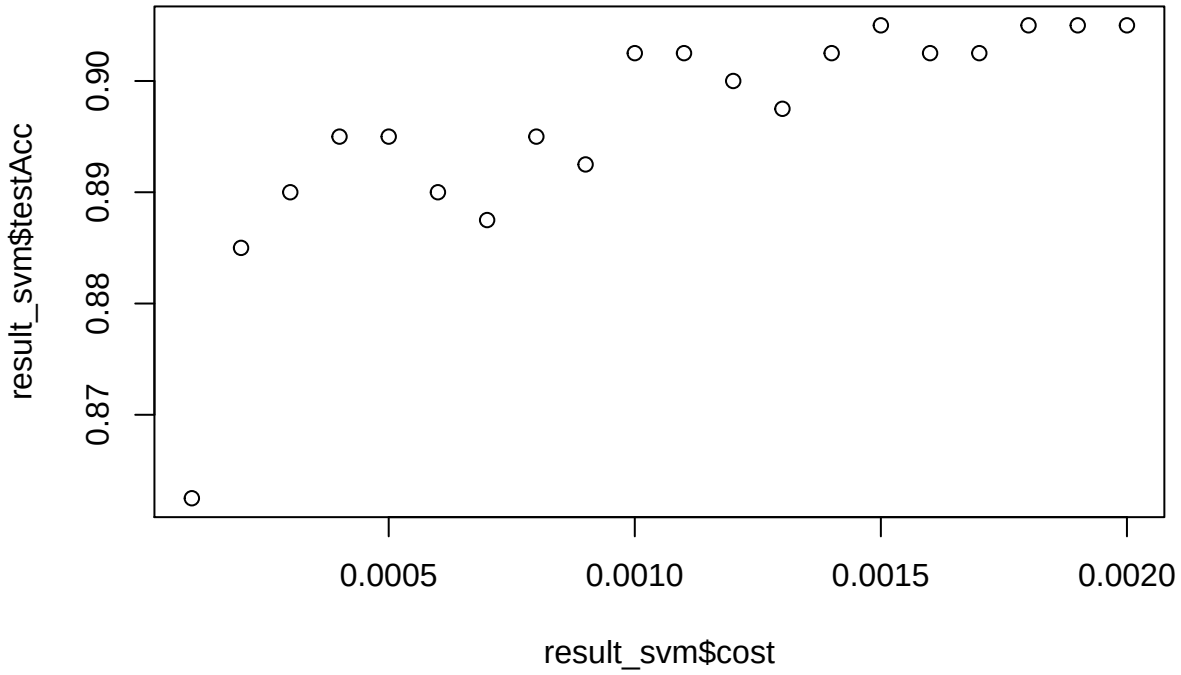
bold Visualize cross-validation results for SVM.

As you can probably see, there is a high correlation between testing accuracy and cost. Surprisingly, the correlation between testing accuracy and differences is not negative. In this case, worst models have a smaller difference between testing and training accuracy, whereas models with good testing accuracy have even better training accuracy.

```
cor(result_svm)
```

```
##                 cost   testAcc  trainAcc difference
## cost       1.0000000 0.8011966 0.8357036  0.7071981
## testAcc    0.8011966 1.0000000 0.9280708  0.6472981
## trainAcc   0.8357036 0.9280708 1.0000000  0.8845983
## difference 0.7071981 0.6472981 0.8845983  1.0000000
```

```
plot(result_svm$cost, result_svm$testAcc)
```



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.

```
## Time for constructing baseline features =  0.2245 minutes
## Time for training baseline =  5.695167 minutes
## Time for constructing MLP features =  0.6583333 minutes
## Time for tuning MLP model =  25.71567 minutes
## Time for training MLP model =  0.04633333 minutes
```

```
## Time for predicting with MLP model =  0.01 minutes
## Time for constructing SVM features =  0.03583333 minutes
## Time for tuning SVM model =  3.686167 minutes
## Time for training SVM model =  0.137 minutes
## Time for predicting with SVM model =  0.02216667 minutes
```