A Comparison of Classical Supervised Machine Learning Algorithms

Yiding Qiu

University of California,San Diego, CA, 92092
yiqiu@ucsd.edu

Abstract

This paper uses several classic machine learning algorithms on datasets obtained from UCI dataset, and the result is compared with Caruana and Niculescu-Mizils research paper. Result shows that without calibrating the model, bagged decision tree has the best performance, which is consistent with the result in Caruana and Niculescu-Mizils paper.

1 Introduction

Despite the prosperous of deep neural networks in recent years, classical machine learning algorithms maintain their merits, as they are easier to implement, and can yield acceptable results in a relatively short time. In this paper, several classical machine learning classifiers are used, including decision tree, SVMs, KNN, bagged tree, boosted tree and random forest. Three datasets (Adult, Covtype and Letter) are obtained from UCI Dataset repository, and are evaluated using accuracy, f-score, and area under ROC curve.

The main purpose of this paper is to follow Caruana and Niculescu-Mizils research paper An Empirical Comparison of Supervised Learning Algorithms, and to check whether the results are consistent. Section 2 introduces parameters used in each algorithms, the dataset, and the evaluation metric. Section 3 shows the experiment results, and section 4 is the conclusion.

2 Method

2.1 Learning Algorithms

Decision Tree (DT) Matlab build-in function fitctree is used as a classifier. The minimum leaf size is varied from 1 to 100 by factor of 10, and the best parameter is chosen using 10-fold cross validation.

Support Vector Machines (SVMs) Three different SVMs are used. The kernels are linear, polynomial degree 2 and 3, radial with gamma of 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, and 2. The regularization parameter is varied by factors of ten from 10^{-7} to 10^{3} with each kernel. Libsvm of the Matlab version is used.

K Nearest Neighborhood (KNN) Matlab build-in function fitcknn is used. The number of k is chosen from one to the training size, and is scaled more sparsely with the number increases. The best k is chosen via 10-fold cross validation.

Boosting Tree Python version of xgboost is used. The maximum tree depth is varied from 3 to 10, The minimum child weight is set to be 1 to 6, and the parameter is chosen via 5-fold cross validation.

Random Forest (RF) Karpathys Random Forest Matlab toolbox is obtained online. 100 trees are used, number of weak classifier is 2.

Bagging Tree Matlab build-in function fitensemble is used, and the number of trees is varied from 50 to 500 with step size 50, and the parameter is chosen via 10-fold cross validation.

2.2 Data Sets

Three datasets from UCI Dataset is used. The Adult dataset contains 14 features and 1 binary label. The categorical features are one-hot coded and the numerical features are normalized between zero and one. The Covtype dataset contains 54 features and the label has seven classes. Features are normalized and the labels are set to be binary, with the largest class as positive and the rest to be zero. The Letter dataset contains 26 features and label with 26 classes. The labels are encoded to be binary, as A-M to be positive and the rest to be zero. The number of training and testing data size are chosen as same as the reference paper. The detailed sizes are listed in the Table 1.

data	train size	test size
adult	5000	35222
Covtype	5000	25000
letter	5000	14000

Table 1: data size

2.3 Performance Metric

Three evaluation methods are used. Accuracy (ACC) compares the predicted label and the true label directly, and is calculated by dividing the correct prediction with the total number of prediction. F-score (FSC) considers both the precision and the recall of the test for the measurement; it is calculated by 2 * precision*recall / (precision + recall). Area under the ROC curve (AUC) refers to the probability that a classifier will rank a positive instance higher than a negative one, and the ROC curve is plotted using sensitivity versus specificity.

3 Result

The detailed performance of each classifier using multiple evaluation matrices on each dataset is listed below in Table 2. The average performance of each data is determined by averaging all three scores on three dataset, and is ranked in descending order. The highest score in each column is bolded.

Result shows that bagged tree has the highest score for the average over all three datasets and three performance metrics. Also we can inform from the table that boosted tree has the best performance on CovType data, and KNN shows the highest score for Letter data.

4 Conclusion

In this paper, data analysis using different machine learning algorithms are performed, and the result is consistent with Caruana and Niculescu-Mizil paper, which ranks bagged decision tree the first among all uncalibrated classifiers. The second one in this paper is boosted decision tree, whereas for Caruanas paper is random forest. Since different libraries and packages are used, the inconsistency should also be reasonable. One problem is that the performance of random forest is not good. This might due to not selecting optimal parameters using cross-validation. Also, some choices of parameters are not clear in the original paper, and thus is difficult to replicate. In addition, boosted tree using xgboost performs significantly better than Adaboost in matlab build-in function (the result can be seen in the appendix). In this future, more classifiers can be used and higher dimensional datasets might be tested.

References

Caruana, R., Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. Proceedings of the 23rd International Conference on Machine Learning - ICML '06.

Chang, C., Lin, C. LIBSVM – A Library for Support Vector Machines. (n.d.). Retrieved March 16, 2016, from https://www.csie.ntu.edu.tw/ cjlin/libsvm/

Dmlc/xgboost. (n.d.). Retrieved March 18, 2016, from https://github.com/dmlc/xgboost

Karpathy, A. Karpathy/Random-Forest-Matlab. Retrieved March 16, 2016, from https://github.com/karpathy/Random-Forest-Matlab

Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

	Adult			CovType			Letter			Average
	ACC	FSC	AUC	ACC	FSC	AUC	ACC	FSC	AUC	
Bagging	**86.64**	90.86	76.37	80.77	81.47	80.83	94.59	94.07	94.58	86.69
Boosting	86.04	**91.12**	**77.15**	**81.09**	**81.80**	**81.15**	93.32	92.75	93.39	86.42
SVM(radial)	84.69	90.06	74.56	78.55	78.12	78.54	93.41	92.89	93.53	84.92
KNN	83.35	89.34	71.83	77.46	77.53	77.47	**94.61**	**94.13**	**94.66**	84.49
SVM(poly)	81.33	87.93	59.37	79.06	79.63	79.10	93.41	92.89	93.53	82.91
DT	84.53	90.2	74.77	75.51	75.87	75.36	87.28	86.21	87.31	81.89
SVM(linear)	84.88	90.27	76.09	75.66	76.94	75.45	72.62	69.49	72.29	77.07
RF	80.55	88.45	61.70	74.52	75.81	74.6	76.6	73.31	76.11	75.74

Table 2: Result

1 Appendex

classify using matlab boosted tree

	Adult			CovType			Letter			Average
	ACC	FSC	AUC	ACC	FSC	AUC	ACC	FSC	AUC	
boost	85.6	90.73	76.09	76.11	76.92	76.16	78.48	75.81	78.13	79.33667

```
clear all
load('adult.mat')
%%
adult_train(strcmp(' Holand-Netherlands' ,adult_train)) = {' ?'};
adtest_onehot = [];
for i = 1:size(adult_test,2)
    if iscellstr(adult_test(1,i))
       adtest(:,i) = double(nominal(adult_test(:,i)));
       feature = oneHot(adtest(:,i));
    else
       adtest(:,i) = normc(cell2mat(adult_test(:,i)));
       feature = (adtest(:,i)-min(adtest(:,i)))./(max(adtest(:,i))-min(adtest(:,i))
    end
       adtest_onehot = cat(2,adtest_onehot,feature);
    end

adtrain_onehot = [];
for i = 1:size(adult_train,2)
    if iscellstr(adult_test(1,i))
       adtrain(:,i) = double(nominal(adult_train(:,i)));
       feature = oneHot(adtrain(:,i));
    else
       adtrain(:,i) = cell2mat(adult_train(:,i));
       feature = (adtrain(:,i)-min(adtrain(:,i)))./(max(adtrain(:,i))-min(adtrain(
    end
    adtrain_onehot = cat(2,adtrain_onehot,feature);
end
clear feature adult_test adult_train i

train = datasample(adtrain_onehot,5000,'Replace',false );
test = datasample([adtrain_onehot(~ismember(adtrain_onehot,train,'rows'),:); adtes
xtrain = train(:,1:107);
ytrain = train(:,108);
xtest = test(:,1:107);
ytest = test(:,108);
save('adult_proc','xtrain','ytrain','xtest','ytest');

%% covtype
```

```
clear all
load('covtype.mat')
train = datasample(covtype,5000,'Replace',false );
test = datasample(covtype(~ismember(covtype,train,'rows'),:),25000,'Replace',false)

for i = 1:54
xtrain(:,i) = (train(:,i)-min(train(:,i)))./(max(train(:,i))-min(train(:,i)));
xtest(:,i) = (test(:,i)-min(test(:,i)))./(max(test(:,i))-min(test(:,i)));
end
xtrain(isnan(xtrain)) = 0;
xtest(isnan(xtest)) = 0;
 [~,i] = max(hist(train(:,55)));
ytrain = double(train(:,55)==i);
ytest = double(test(:,55)==i);
save('cov_proc','xtrain','ytrain','xtest','ytest');

%% letter
clear all
load('letter.mat')
%%
label = double(letter_y <= 13);
letter = [letter_x, label];
train = datasample(letter,5000,'Replace',false );
test = datasample(letter(~ismember(letter,train,'rows'),:),14000,'Replace',false);

for i = 1:16;
xtrain(:,i) = (train(:,i)-min(train(:,i)))./(max(train(:,i))-min(train(:,i)));
xtest(:,i) = (test(:,i)-min(test(:,i)))./(max(test(:,i))-min(test(:,i)));
end

ytrain = train(:,17);
ytest = test(:,17);
save('letter_proc','xtrain','ytrain','xtest','ytest');


function oneHotLabels = oneHot(labels)

valueLabels = unique(labels);
nLabels = length(valueLabels);
nSamples = size(labels,1);

oneHotLabels = zeros(nSamples, nLabels);

for i = 1:nLabels
    oneHotLabels(:,i) = (labels == valueLabels(i));
end


%% data Adult
```

```
clear all
load('adult_proc.mat')
%% data(covtype)
clear all
load('cov_proc.mat')
%% data(Letter)
clear all
load('letter_proc')
%% C4.5
% target = c45(xtrain',ytrain',xtest(1:200,:)',2);
% acc = sum(target' == ytest)/size(ytest,1);
%% fitctree
m = [1:5:100];
for i = 1:length(m)
    tree = fitctree(xtrain,ytrain,'MinLeaf',m(i));
    loss(i) = cvLoss(tree);
end
[~,ind] = min(loss);
tree = fitctree(xtrain,ytrain,'MinLeaf',m(ind));
target = predict(tree,xtest);
acc1 = sum(target == ytest)/size(ytest,1)
stats = confusionmatStats(ytest,target);
fscore1 =  stats.Fscore(stats.groupOrder == 1)
[~,~,~,AUC1] = perfcurve(ytest,target,'1')
%% linear SVM
[target acc2]= linear_SVM(xtrain,ytrain,xtest,ytest)
stats = confusionmatStats(ytest,target);
fscore2 =  stats.Fscore(stats.groupOrder == 1)
[~,~,~,AUC2] = perfcurve(ytest,target,'1')
%% radius SVM
[target acc3]= radius_SVM(xtrain,ytrain,xtest,ytest)
stats = confusionmatStats(ytest,target);
fscore3 =  stats.Fscore(stats.groupOrder == 1)
[~,~,~,AUC3] = perfcurve(ytest,target,'1')
%% polynomial SVM
[target acc4]= poly_SVM(xtrain,ytrain,xtest,ytest)
stats = confusionmatStats(ytest,target);
fscore4 =  stats.Fscore(stats.groupOrder == 1)
[~,~,~,AUC4] = perfcurve(ytest,target,'1')
%% KNN
k = [1:2:9,10:10:90,100:200:900,1000:500:5000];
for i = 1:length(k);
cmdl = fitcknn(xtrain,ytrain,'NumNeighbors',k(i));
cvcmdl = crossval(cmdl);
kcloss(i) = kfoldLoss(cvcmdl);
end
[~,ind] = min(kcloss);
k_best = k(ind);
mdl = fitcknn(xtrain,ytrain,'NumNeighbors',k_best);
```

```
target = predict(mdl,xtest);
acc5 = sum(target == ytest)/size(ytest,1)
stats = confusionmatStats(ytest,target);
fscore5 =  stats.Fscore(stats.groupOrder == 1)
[~,~,~,AUC5] = perfcurve(ytest,target,'1')
%% boosting
m = [50:50:500];
for i = 1:length(m)
    ClassTreeEns = fitensemble(xtrain,ytrain,'AdaBoostM1',m(i),'Tree','LearnRate',(
    cv = crossval(ClassTreeEns);
    loss1(i) = kfoldLoss(cv)
end
[~,ind] = min(loss1);
ClassTreeEns = fitensemble(xtrain,ytrain,'AdaBoostM1',m(ind),'Tree','LearnRate',0.9
target = predict(ClassTreeEns,xtest);
acc6 = sum(target == ytest)/size(ytest,1)
stats = confusionmatStats(ytest,target);
fscore6 =  stats.Fscore(stats.groupOrder == 1)
[~,~,~,AUC6] = perfcurve(ytest,target,'1')
%% baggin
m = [50:50:500];
for i = 1:length(m)
    ClassTreeEns = fitensemble(xtrain,ytrain,'Bag',m(i),'Tree','Type','Class');
    cv = crossval(ClassTreeEns);
    loss2(i) = kfoldLoss(cv)
end
[~,ind] = min(loss2);
ClassTreeEns = fitensemble(xtrain,ytrain,'Bag',m(ind),'Tree','Type','Class');
target = predict(ClassTreeEns,xtest);
acc7 = sum(target == ytest)/size(ytest,1)
stats = confusionmatStats(ytest,target)
fscore7 =  stats.Fscore(stats.groupOrder == 1)
[~,~,~,AUC7] = perfcurve(ytest,target,'1')
%% Random Forest
opts= struct;
opts.numTrees= 1024;
opts.classifierID= 2;
model = forestTrain(xtrain,ytrain,opts);
target = forestTest(model, xtest);
acc8 = sum(target == ytest)/size(ytest,1)
stats = confusionmatStats(ytest,target);
fscore8 =  stats.Fscore(stats.groupOrder == 1)
[~,~,~,AUC8] = perfcurve(ytest,target,'1')


function [target acc] = linear_SVM(xtrain,ytrain,xtest,ytest)
%Linear and RBF SVM for Homework 5 Cogs 118A
%   Detailed explanation goes here
```

```
C = logspace(-6,3,10);
cv_acc1 = zeros(length(C),1);
for i = 1:length(C);
    cv_acc1(i) = svmtrain(ytrain, xtrain, sprintf('-t 0, -c %f -v 5 -q',C(i)));
end
[~,ind1] = max(cv_acc1);
best_C1 = C(ind1);
model1 = svmtrain(ytrain, xtrain,sprintf('-t 0, -c %f -q',best_C1));
[target, accuracy1, ~] = svmpredict(ytest,xtest,model1);
acc = accuracy1(1);
clc;
end


function [target acc] = poly_SVM(xtrain,ytrain,xtest,ytest)
a = [0.001,0.005,0.01,0.05,0.1,0.5,1,2];
[C2,gamma,degree] = meshgrid(logspace(-6,3,10),a,2:3);
cv_acc2 = zeros(numel(C2),1);

for k = 1:numel(C2);
    cv_acc2(k) = svmtrain(ytrain, xtrain, sprintf('-t 1, -c %f -g %f -v 5 -q -d %f
end
[~,ind2] = max(cv_acc2);
best_C2 = C2(ind2);
best_gamma2 = gamma(ind2);
best_degree = degree(ind2);
model2 = svmtrain(ytrain, xtrain,sprintf('-t 2, -c %f, -g %f -q -d %f', best_C2, be
[target, accuracy2, ~] = svmpredict(ytest,xtest,model2);
acc = accuracy2(1);
clc;
end


function [target acc] = radius_SVM(xtrain,ytrain,xtest,ytest)
a = [0.001,0.005,0.01,0.05,0.1,0.5,1,2];
[C2,gamma] = meshgrid(logspace(-6,3,10),a);
cv_acc2 = zeros(numel(C2),1);

for k = 1:numel(C2);
    cv_acc2(k) = svmtrain(ytrain, xtrain, sprintf('-t 2, -c %f -g %f -v 5 -q',C2(k)
end
[~,ind2] = max(cv_acc2);
best_C2 = C2(ind2);
best_gamma2 = gamma(ind2);
model2 = svmtrain(ytrain, xtrain,sprintf('-t 2, -c %f, -g %f -q', best_C2, best_gam
[target, accuracy2, ~] = svmpredict(ytest,xtest,model2);
acc = accuracy2(1);
clc;
end
```

```
# coding: utf-8

# In[1]:

import xgboost as xgb
import scipy.io as sio
import numpy as np
from sklearn import cross_validation
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import f1_score, auc,roc_curve


def xgbtree(xtrain,ytrain,xtest,ytest):
dtrain = xgb.DMatrix(xtrain, label= ytrain)
dtest = xgb.DMatrix(xtest, label = ytest)

#Cross-Validation
param_test1 = [{'max_depth':range(3,10,1),'min_child_weight':range(1,6,2)}]
clf = GridSearchCV(xgb.XGBClassifier(), param_test1)
clf.fit(xtrain,ytrain.T[0])
print("With %s,The training accuracy is: %0.5f" % (clf.best_params_, clf.best_scor

#evaluate test data
ypred = clf.predict(xtest)
acc = clf.score(xtest,ytest.T[0])
f_score = f1_score(ytest.T[0], ypred, average='binary')
fpr, tpr, thresholds = roc_curve(ytest.T[0], ypred, pos_label=1)
AUC = auc(fpr, tpr)
print('acc = %f, FSC = %f, AUC = %f'% (acc,f_score,AUC))
return


# In[2]:

xtrain = sio.loadmat('adult_proc.mat')['xtrain']
ytrain = sio.loadmat('adult_proc.mat')['ytrain']
xtest = sio.loadmat('adult_proc.mat')['xtest']
ytest = sio.loadmat('adult_proc.mat')['ytest']
xgbtree(xtrain,ytrain,xtest,ytest)


# In[3]:

xtrain = sio.loadmat('cov_proc.mat')['xtrain']
ytrain = sio.loadmat('cov_proc.mat')['ytrain']
xtest = sio.loadmat('cov_proc.mat')['xtest']
ytest = sio.loadmat('cov_proc.mat')['ytest']
xgbtree(xtrain,ytrain,xtest,ytest)
```

```
# In[4]:

xtrain = sio.loadmat('letter_proc.mat')['xtrain']
ytrain = sio.loadmat('letter_proc.mat')['ytrain']
xtest = sio.loadmat('letter_proc.mat')['xtest']
ytest = sio.loadmat('letter_proc.mat')['ytest']
xgbtree(xtrain,ytrain,xtest,ytest)
```