Report for HW1

Yang, Yunfan

yuy130@ucsd.edu

Professor Tu

Abstract

In this report, I am going to explore multiclass classification on a large dataset. The common multiclass classification methods include one-vs-all, one-vs-one, and explicit multiclass classification. In this report, I am going to compare one-vs-all approach with explicit multiclass classification. Also, I will study error output encoding approach in this paper.

1 Introduction

Multiclass classification is an important topic to study because a lot of data classification tasks involve multiclass classification. Some current multiclass classification strategies include one-vs-all, one-vs-one, and explicit multiclass classification. One-vs-all method works by converting all the n binary dataset and constructing n binary classifiers. The pros of one-vs-all approach is its simplicity. The cons of it is sometime the labels can be unbalanced and also some region can be ignored by multiple classifiers. One-vs-one approach works by constructing $K(K1)/2$ classifiers. And this approach is very expensive to construct. Another one is called explicit multiclass classification. There are some explicit multiclass algorithms been developed which classifies multiple classes during training time. Explicit multiclass algorithms sound like an appealing approach. In this report, I am going to study whether one-vs-all or multiclass approach is better as well as ways to improve runtime.

2 Methodology

2.1 Technology

Each experiment was written in Python3 and executed on Amazon's EC2 Cloud Computing service, where I rented an c4.8xlarge server with 36 CPUs and 60GB of RAM.

2.2 Data Preprocessing

To run learning algorithm more efficiently, I need to preprocess the data. For continuous feature, I standardize them by subtracting the mean and then divide the result by standard deviation.

2.3 Learning Algorithms

Using scikit-learn libraries, we were be able to conduct this study. We studied:

BDT-OVA - One-vs-all Boosted Decision Tree:

I use the AdaBoostClassifier provided in scikit-learn and apply gridsearch cross-validation to select the hyperparameter, steps of boosting from the range 2, 4, 8, 16, 32, 64, and 128. I create one AdaBoostClassifier binary classifier for each class. To predict multiple labels, I use all the binary classifiers to generate the final prediction.

BDT-EXP - Explicit Boosted Decision Tree:

I use the AdaBoostClassifier provided in scikit-learn and use 128 as steps of boosting. AdaBoostClassifier is a multiclass classifier using SAMME.R algorithm.

RF-OVA - One-vs-all Random Forest Classifier

I use RandomForestClassifier provided in scikit-learn and apply gridsearch cross-validation to choose the hyper-parameter, the number of decision trees, ranging from 2, 4, 8, 16, 32, 64, 128. I create one Random-Forest binary classifier for each class. To predict multiple labels, I use all the binary classifiers to generate the final prediction.

RF-EXP - Explicit Random Forest Classifier

I use RandomForestClassifier provided in scikit-learn and apply gridsearch cross-validation to choose the hyper-parameter, the number of decision trees, ranging from 2, 4, 8, 16, 32, 64, 128.

SVM-OVA - One-vs-all Support Vector Machine

I use linearSVC provided in scikit-learn and apply gridsearch cross-validation to choose the hyper-parameter, C, from the range 0.1, 1, 10, 100. In default, linearSVC classifier uses one-vs-all approach for multiclass classification.

SVM-EXP - Explicit Support Vector Machine

I use linearSVC provided in scikit-learn and apply gridsearch cross-validation to choose the hyper-parameter, C, from the range 0.1, 1, 10, 100. I use crammer_singer algorithm here as my explicit SVM algorithm.

2.4 Data Sets

I downloaded the dataset, 'COV_TYPE', from UCI machine learning repository. In the 'COV_TYPE' dataset, I am given 581,012 data points each with 54 quantitative wilderness data features, and one of 7 distinct labels corresponding to 7 different forest cover type classifications. I pick this challenging dataset is because studying classifying large datasets is particularly useful. In my last final paper, I converted this problem into a binary dataset. Here, I am going to perform multiclass classification. To save time, I use first 100,000 data points as my experiment dataset. And I pick 80,000 data points randomly from the data as the training set, and the rest as testing set. To generate consistent experiment result, I set pseudo random generator to 13.

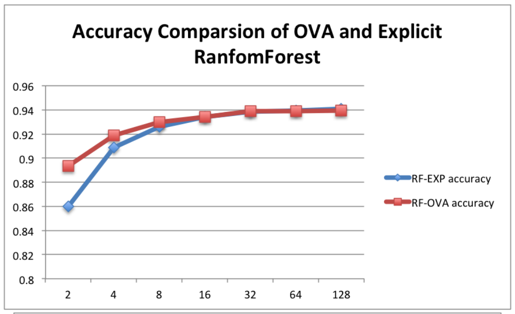
3 Experiment

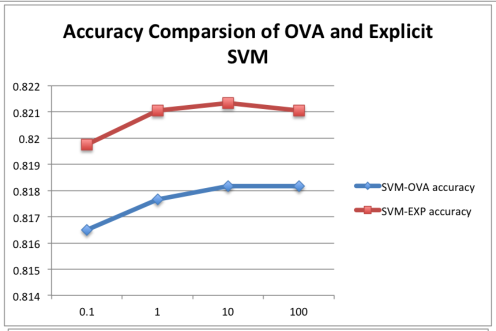
3.1 Comparison

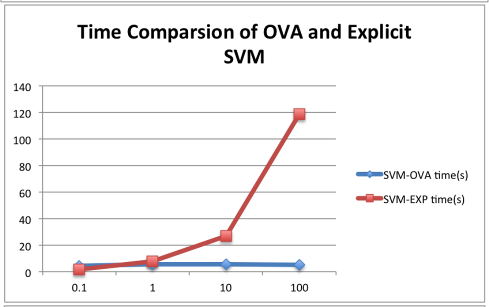
Here, we list the accuracy comparison between OVA approach and explicit approach. We also list the relationship between accuracy and size of output code. From these collected data, initially explicit Random Forest doesn't perform as well as OVA RF. As the number of trees increases, there is no difference between accuracy of RF-EXP and RF-OVA.
In the SVM comparison, we can see SVM-EXP outperforms SVM-OVA in general, however, SVM-EXP seems to be dependent on parameter C. As C increases, the amount of time it takes for SVM-EXP increases rapidly.
In the BDT-OVA case, we can see as the number of trees increases, the overall accuracy increases steadily. However, it seems like BDT-EXP doesn't work properly on this dataset. As the number of tree increases, the overall accuracy decreases.
Lastly, I studied error output code. In the graph, the x-axis shows the ratio of the size of error output code to the number of classes. As I increase the ratio of error output code, the accuracy increases as well. From 0.2 to 1, we can see an improvement for all OVA classifiers. However, it stops improving after reaching 1.

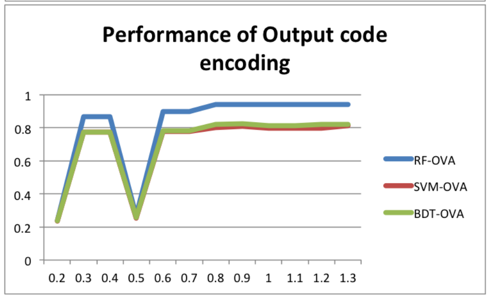
Table 1: BDT-OVA performance

# of tree	accuracy	time(s)
2	0.7927748	4.24
4	0.8167126	9.72
8	0.8304498	16.74
16	0.8454498	28.8
32	0.8542374	52.54
64	0.8669626	90
128	0.8810752	163.2









4 Discussion

In Random Forest, OVA and Explict approach have really close performance. I think the reason is because the base classifier is decision tree which naturally supports multiclass classification. RF-OVA is actually a wrapper to RF-EXP. Thus, the result should be very close to each other. Also, because RF-OVA creates n separate classifiers, it is slower than RF-EXP. For these classifiers that naturally support multiclass classifier, using OVA is not a good approach and we should use explicit approach directly.
For SVM, we can see SVM-EX outperforms SVM-OVA by not much. However, SVM-EX is a lot more slower to compute. For a large dataset, I would advice using SVM-OVA for its fast speed and consistent performance.
For BDT, we cannot reach a conclusion because it seems like BDT-EX doesn't work as expected on this dataset.
Lastly, the error correcting code approach tells me that as we compress the number of classifiers needed, the accuracy decreases. To balance the training time and accuracy, we can find a tradeoff here. It looks like when I compress the number of classifiers to 80%, it doesn't really affect classification result.
In the future, I would like to implement my algorithms, as well as studying more datasets.

Acknowledgments

We want to expressly acknowledge the teaching and instruction provided by Prof. Zhuowen Tu. Additionally we want to highlight our exceptional TA, Daniel Maryanovsky whose mentoring and patience was instrumental for our success throughout this project and the course at large.

5 Conclusion

Source Code

covtype.py

```
# data preprocessing
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import sklearn.cross_validation as cv
from sklearn.grid_search import GridSearchCV
from collections import Counter

N_TRAIN = 80000
CAP = 100000
def loadData():
    # import data
    X = [list(map(int, x.split(',')[:-1])) for x in open('covtype.data').
       read().splitlines()[:CAP]]
    _Y = [x.split(',')[-1] for x in open('covtype.data').read().splitlines
       ()[:CAP]]
    Y = [int(x) - 1 for x in _Y]

    xTrain, xTest, yTrain, yTest = cv.train_test_split(np.array(X), np.
       array(Y), train_size = N_TRAIN/len(X), random_state = 13)

    mean = xTrain.mean(axis=0)
    std = xTrain.std(axis=0)
    mean[10:] = 0.0
    std[10:] = 1.0
    xTrain = (xTrain - mean) / std
    xTest = (xTest - mean) / std
    return xTrain, xTest, yTrain, yTest

from sklearn.grid_search import GridSearchCV
from sklearn.ensemble import AdaBoostClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
xTrain, xTest, yTrain, yTest = loadData()

adaboost = AdaBoostClassifier(base_estimator = DecisionTreeClassifier(
   max_depth = 2))
randomforest = RandomForestClassifier()

ESTIMATORS = {
    'SVM-OVA': LinearSVC(loss="squared_hinge", penalty="l2", C=1000, dual=
       False, tol=1e-3),
    'SVM-EXPLICIT': LinearSVC(loss="squared_hinge", penalty='l2', C=1000,
       dual=False, tol = 1e-3, multi_class='crammer_singer'),
    'RANDOMFOREST-OVA': OneVsRestClassifier(estimator = randomforest),
    'RANDOMFOREST-EXPLICIT': RandomForestClassifier(),
    'BOOSTING-OVA': OneVsRestClassifier(estimator = adaboost)
}
GRIDS = {
        'SVM-OVA': {
                        'C': [0.1, 1, 10, 100]
                },
        'SVM-EXPLICIT': {
                        'C': [0.1, 1, 10, 100]
                },
        'BOOSTING-OVA': {
                        'estimator__n_estimators': [2, 4, 8, 16, 32, 64,
                           128]
                },
        'RANDOMFOREST-OVA': {
                        'estimator__n_estimators': [2, 4, 8, 16, 32, 64,
                           128]
                },
        'RANDOMFOREST-EXPLICIT': {
                        'n_estimators': [2, 4, 8, 16, 32, 64, 128]
                }
}

print("Training Classifiers")
print("===================")
for name in ESTIMATORS:
    print("Training %s ... " % name)
    estimator = ESTIMATORS[name]
    CV = GridSearchCV(estimator, param_grid=GRIDS[name], cv=5, n_jobs=-1 ,
        verbose = 4)
    CV.fit(xTrain, yTrain)
    print ("Model accuracy: " + str(CV.score(xTest, yTest)))
```

covtype2.py

```
# data preprocessing
import warnings
warnings.filterwarnings("ignore")
```

```
import numpy as np
import sklearn.cross_validation as cv
from sklearn.grid_search import GridSearchCV
from collections import Counter

N_TRAIN = 80000
CAP = 100000
def loadData():
    # import data
    X = [list(map(int, x.split(',')[:-1])) for x in open('covtype.data').
       read().splitlines()[:CAP]]
    _Y = [x.split(',')[-1] for x in open('covtype.data').read().splitlines
       ()[:CAP]]
    Y = [int(x) - 1 for x in _Y]

    xTrain, xTest, yTrain, yTest = cv.train_test_split(np.array(X), np.
       array(Y), train_size = N_TRAIN/len(X), random_state = 13)

    mean = xTrain.mean(axis=0)
    std = xTrain.std(axis=0)
    mean[10:] = 0.0
    std[10:] = 1.0
    xTrain = (xTrain - mean) / std
    xTest = (xTest - mean) / std

    return xTrain, xTest, yTrain, yTest

from sklearn.grid_search import GridSearchCV
from sklearn.ensemble import AdaBoostClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
xTrain, xTest, yTrain, yTest = loadData()

adaboost = AdaBoostClassifier(base_estimator = DecisionTreeClassifier(
   max_depth = 2))
randomforest = RandomForestClassifier()

ESTIMATORS = {
#    'SVM-OVA': LinearSVC(loss="squared_hinge", penalty="l2", C=1000, dual
   =False, tol=1e-3),
#    'SVM-EXPLICIT': LinearSVC(loss="squared_hinge", penalty='l2', C=1000,
    dual=False, tol = 1e-3, multi_class='crammer_singer'),
#    'BOOSTING-OVA': OneVsRestClassifier(estimator = adaboost),
    'BOOSTING-EXPLICIT': AdaBoostClassifier(base_estimator =
       DecisionTreeClassifier(max_depth = 2), n_estimators = 128),
#    'RANDOMFOREST-OVA': OneVsRestClassifier(estimator = randomforest),
#    'RANDOMFOREST-EXPLICIT': RandomForestClassifier(),
}
GRIDS = {
        'SVM-OVA': {
                        'C': [0.1, 1, 10, 100, 1000, 10000]
                },
        'SVM-EXPLICIT': {
                        'C': [0.1, 1, 10, 100, 1000, 10000]
                },
        'BOOSTING-OVA': {
```

```
                        'estimator__n_estimators': [2, 4, 8, 16, 32, 64,
                            128]
                },
        'BOOSTING-EXPLICIT': {
                        'n_estimators': [2, 4, 8, 16, 32, 64, 128]
                },
        'RANDOMFOREST-OVA': {
                        'estimator__n_estimators': [2, 4, 8, 16, 32, 64,
                            128, 256]
                },
        'RANDOMFOREST-EXPLICIT': {
                        'n_estimators': [2, 4, 8, 16, 32, 64, 128, 256]
                },
}

print("Training Classifiers")
print("===================")
for name in ESTIMATORS:
    print("Training %s ... " % name)
    estimator = ESTIMATORS[name]

    estimator.fit(xTrain, yTrain)
    for predict in estimator.staged_predict(xTest):
        print ("Model accuracy: " + str(accuracy_score(predict, yTest)))
```

covtype_error_output.py

```
# data preprocessing
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import sklearn.cross_validation as cv
from sklearn.grid_search import GridSearchCV
from collections import Counter
from joblib import Parallel, delayed
import multiprocessing
num_cores = multiprocessing.cpu_count()

N_TRAIN = 80000
CAP = 100000
def loadData():
    # import data
    X = [list(map(int, x.split(',')[:-1])) for x in open('covtype.data').
        read().splitlines()[:CAP]]
    _Y = [x.split(',')[-1] for x in open('covtype.data').read().splitlines
        ()[:CAP]]
    Y = [int(x) - 1 for x in _Y]

    xTrain, xTest, yTrain, yTest = cv.train_test_split(np.array(X), np.
        array(Y), train_size = N_TRAIN/len(X), random_state = 13)

    mean = xTrain.mean(axis=0)
    std = xTrain.std(axis=0)
    mean[10:] = 0.0
    std[10:] = 1.0
    xTrain = (xTrain - mean) / std
    xTest = (xTest - mean) / std
    return xTrain, xTest, yTrain, yTest

def test(estimator, sz):
    print("Trainin with size of " + str(sz))
    estimator.code_size = sz
    estimator.fit(xTrain, yTrain)
    y_pred = estimator.predict(xTest)
    from sklearn.metrics import classification_report
    print ("Code_size: " + str(sz) + "Model accuracy: " + str(
       accuracy_score(yTest, y_pred)))

#error_output_code for OVAs
from sklearn.grid_search import GridSearchCV
from sklearn.ensemble import AdaBoostClassifier
from sklearn.multiclass import OutputCodeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
xTrain, xTest, yTrain, yTest = loadData()

adaboost = AdaBoostClassifier(n_estimators = 128)
randomforest = RandomForestClassifier(n_estimators = 128)

ESTIMATORS = {
    'SVM-OVA': OutputCodeClassifier(LinearSVC(loss="squared_hinge",
       penalty="l2", C=1000, dual=False, tol=1e-3)),
    'BOOSTING-OVA': OutputCodeClassifier(estimator = adaboost),
    'RANDOMFOREST-OVA': OutputCodeClassifier(estimator = randomforest),
}
code_size = {0.2, 0.3, 0.4, 0.5 , 0.6 , 0.7 , 0.8 , 0.9 , 1, 1.1 , 1.2 ,
   1.3 , 1.4 , 1.5 , 1.6 , 1.7 , 1.8}

print("Training Classifiers")
print("====================")
for name in ESTIMATORS:
    print("Training %s ... " % name)
    estimator = ESTIMATORS[name]
    Parallel(n_jobs=num_cores, verbose=4)(delayed(test)(estimator, sz) for
        sz in code_size)
```

References

[1] Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. The Journal of Machine Learning Research, 5:101141, 2004.

[2] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multi- class support vector machines. Neural Networks, IEEE Transactions on, 13(2):415425, 2002.