AdaBoost - Multiclass

Disha Patil, 1606 10/09/2017

Problem Statement

Build a multiclass adaboost algorithm

The simple adaboost- binary class

1. T= Max iterations
2. $D_i^t = \frac{1}{n} \qquad \forall\, i \ \in \{1...n\}$
   where n= number of training examples
3. for t= 1 to T
   (a) learn classifier f_t (eg:SVM) with distribution D^t as sample weights
   (b) if $f_t(i)! = y_i$
   $\epsilon_t = \sum D_i^t$
   (c) $\alpha_t = \frac{1}{2} * \log(\frac{1-\epsilon_t}{\epsilon_t})$
   (d) $D_i^t = \frac{D_i^t * \exp(-\alpha_t * y_i * f_t(i))}{\sum D_i^t * \exp(-\alpha_t * y_i * f_t(i))}$
4. Voted classifier output
   $\forall \quad x \quad F(x) = sign(\sum_{t=1}^{T} \alpha_t * f_t(x))$

The modified adaboost- multiclass class

1. T= Max ; k= number of classes
2. $D_i^t = \frac{1}{n} \qquad \forall\, i \ \in \{1...n\}$
   where n= number of training examples
3. for t= 1 to T
   (a) learn classifier f_t (eg:SVM) with distribution D^t as sample weights
   (b) if $f_t(i)! = y_i$
   $\epsilon_t = \sum D_i^t$
   (c) $\alpha_t = \frac{1}{2} * \log(\frac{1-\epsilon_t}{\epsilon_t}) + \log(k-1)$
   (d) $D_i^t = \frac{D_i^t * \exp(-\alpha_t * y_i * f_t(i))}{\sum D_i^t * \exp(-\alpha_t * y_i * f_t(i))}$

4. Voted classifier output

$\forall \quad x \quad F(x) = argmax_k(\sum_{t=1}^{T} \alpha_t * (f_t(x) == k))$

Listing 1: python code – multiclass adaboost.

```
1
2  from sklearn import datasets, svm
3  from sklearn.model_selection import train_test_split
4  from sklearn.metrics import confusion_matrix
5  from sklearn.metrics import accuracy_score
6  import numpy as np
7  from sklearn.preprocessing import LabelEncoder
8  import pandas as pd
9
10
11 wine=pd.read_csv("wine.scale",header=None,sep='\s+')
12 #print(wine)
13 for i in range(9):
14     wine[i+1] = wine[i+1].str[2:]
15 ints=[10, 11, 12, 13]
16 for idx, val in enumerate(ints):
17     wine[val] = wine[val].str[3:]
18
19 #for i in range(14):
20     #print(wine[10])
21 #    wine[i] = wine[i].astype(float)
22
23 wine = wine.convert_objects(convert_numeric=True)
24 #wine
25
26
27 X = wine
28 X=X.drop(X.columns[[0]], axis=1)
29 y=wine[0]
30 X=X.fillna(X.median())
31 X
32
33
34 ###split train test
35 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,↩
       random_state=0)
36
37 ###build rbf SVM
38 svc_rbf = svm.SVC(kernel='rbf', gamma=0.03,C=3)
39 #svc_multiclass=OneVsRestClassifier(rbfSVC(random_state=0))
40 #svc_multiclass.fit(X_train, y_train).predict(X_train)
41 #len(y_test)
```

```
42
43
44 ###set max iterations T
45 T=10
46
47 #initialize sample weights
48 dm=np.repeat(1/len(X_train), len(X_train), axis=0)
49 #svc_rbf.fit(X_train, y_train,sample_weight=dm)
50
51 #predict with initialized weights
52 predicted= svc_rbf.fit(X_train, y_train,sample_weight=dm).predict(X_test)
53
54 #print((predicted))
55 #print (dm)
56 #X
57 #np.where(predicted==2)[0].tolist()
58
59 ###aplha parameter from adaboost algorith
60 alpha=[]
61
62 ###list to store predictions
63 pred=[]
64
65 ###list to store test vector predictions
66 testpred=[]
67
68 ###define number of classes
69 classes=3
70
71 ###initialize the cost vector for class imbalance
72 #cost=np.random.uniform(0,1,3).tolist()
73 #cost=[]
74 #for i in range(classes):
75 #    cost.append(np.random.uniform(0,1,1).tolist())
76
77
78 ###run adabosst multiclass with cost vector to each class
79 for t in range(T):
80     #print(t)
81
82     #fit a rbf svm with weights dm
83     svc_rbf.fit(X_train, y_train,sample_weight=dm)
84
85     #predict train values
86     predicted_train=(svc_rbf.predict(X_train))
87
88     #predict test values
```

```
89      predicted_test=(svc_rbf.predict(X_test))
90
91      #store predicted test values
92      testpred.append(predicted_test)
93
94      #store predicted train values
95      pred.append(predicted_train)
96
97      #initially for each iteration error==0
98      err=0
99
100     #for each example in test predictions check misclassification
101     for i in range(len(X_test)):
102         if (testpred[t][i] != y_test.values[i]):
103             y=y_train.values[i]
104             err+=dm[i] #* cost[y-1]
105     #calculate alpha wth err value and classes
106     alpha.append(np.log((1-err)/err) + np.log(classes-1))
107
108     #calculate Z value for normaliztion
109     Z=[]
110     #print(err)
111     for i in range(len(X_train)):
112         Z.append(dm[i] * np.exp((-alpha[t])*y_train.values[i] * ↩
            predicted_train[i]))
113
114     #update the weights
115     dm=[Z[i] / sum(Z) for i in range(len(X_train))]
116
117 #initialize the adaboost ensemble otput for each class
118 ada_class=[]
119
120 #run for loop for number of classes
121 for k in range(classes):
122
123     #initialize predictions for each test vector
124     ada_class_pred=[]
125
126     #for each test vector
127     for i in range(len(testpred[t])):
128
129         #initialize the adaboost output as zero for each class
130         ada_predicts=0
131         #calculate the adaboost predicted value across t generations.this ↩
            is a scalar
132         for t in range(10):
133             vec=testpred[t]
```

```
            al=alpha[t]
            if (vec[i]==k+1):
                ada_predicts+=(al *1)
            else:
                ada_predicts+=(al *0)
        ada_class_pred.append(ada_predicts)
    #predictions of each test vector across classes
    ada_class.append(ada_class_pred)


#ada_class
#use vstack for better handling
adaclass_array=np.vstack(ada_class)
#initialize final pprediction
y_preds=[]
#for each test vector
for i in range(len(y_test)):
    #see for which class is adaboost prediction the maximum...and return that class
    y_preds.append(np.where(adaclass_array[:,i]==np.max(adaclass_array[:,i]))[0].tolist()[0]  + 1) #plus one because outputs as 1,2,3 needed

y_preds

y=y_test.tolist()
acc=0
for i in range(len(y)):
    if(y[i] == y_preds[i]):
        acc+=1
accuracy=acc/100

accuracy
```

Notes

I think the algorithm is correct but I am getting output as single class. That is entire test set is predicted as class 1. I have tried increasing the genrations (max iterations) but still the same output.

Methods tried: -varying the max iterations -changing gamma and C values for rbf kernel in svm