Machine Learning HW2 Report

b03901070 羅啟心

1.Logistic regression function with adagrad:

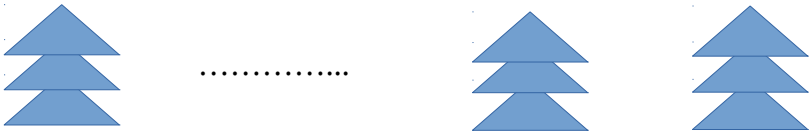
```
def f_wb(self,x):
  z = 0.
  for j in range (57):
    z=z+self.w[j]*float(x[j])
  z=z+self.b
  f_wb=1/(1+numpy.exp(-z))
  return(f_wb)

def logistic_regression(self,n,eta):
  for i in range(n):
    S=[1.]*57
    for j in range (57):
      s=0.
      for k in range(4001):
        s=s+(float(self.classify[k])-self.f_wb(self.train[k][1:58]))*float(self.train[k][j+1])
        S[j]=S[j]+s*s
      self.w[j]=self.w[j]+eta*(1.0/(S[j]**0.5))*s
```

2.Describe your another method, and which one is best.

Training:

Random Forest:

My another method is random forest, written in distribution.py.
It builds up 57 different decision trees, each having 10 layers.
There are 57 decision trees in the forest, indexed 0,1,...57.
For the decision tree with index j , node_feature[1]=feature[j]
And then decide the threshold ratio by calculating which threshold(within {0,1,2,3,4}) has the maximal information gain.

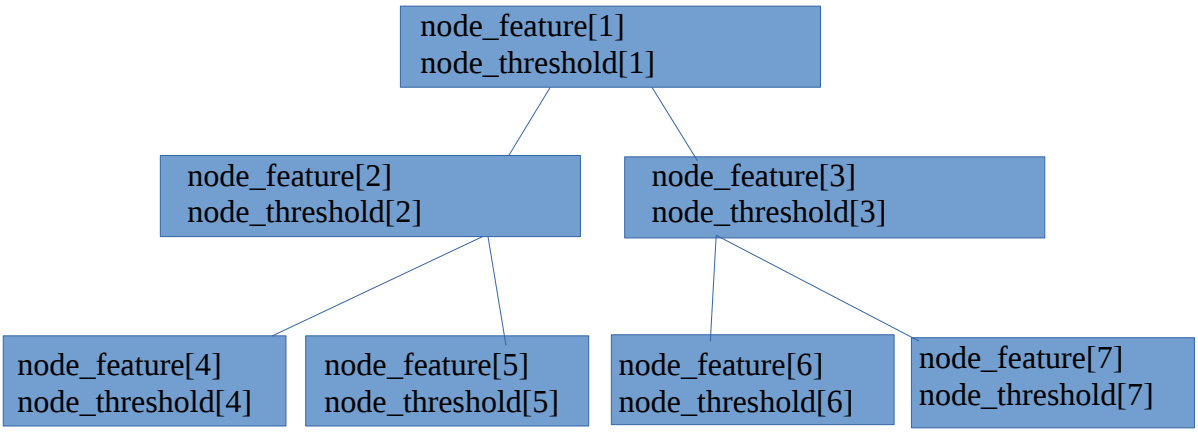


Decision Tree:

There are 10 layers in a tree.
The main things required to build a decision tree are :
node_feature[i] and node_threshold[i]
in order to determine the values above, we need two more arrays :
node_entropy[i] and node_data_index[i]
which represents the entropy (before splitting) and the index of the training data (subset of {1,2,...4001}) (before splitting) , respectively.
The determination of node_feature[i] and node_threshold[i] requires the following concepts :

E(set) = -{(class1 ratio)*log(class1 ratio)+(class0 ratio)*log(class0 ratio)}
where the ratio are calculated wrt the set (a subset of{1,2,.,4001})
Once the node_feature[i] is determined , node_data_index[i] is splitted into

two sets , which are left_index and right_index.
E(after) = (len(left_index)/len(left_index)+len(right_index.))*E(left_index)+
(len(right_index)/len(left_index)+len(right_index.))*E(right_index)
Since node_feature and node_threshold are required in the process of spitting the set , we can calculate the Informaion gain :
Information gain(node_feature,node_threshold) =
E(after)-E(node_data_index)



featur_max[0]	node_feature[1]	node_threshold[1]
.	.	.
.	.	.
.	.	.
featur_max[56]	node_feature[1023]	node_threshold[1023]

test[0][0]	test[0][1]	.	.	.	.	.	.	.	test[0][56]
.	.	.							
.	.	.							
.	.	.							
test[0][0]	test[0][1]	.	.	.	.		.	.	test[0][56]

Prediction:
Assume we are determining which class the j th test data is:
Path in a Tree:
when encountering node k , compare
test[j][node_feature[k]] and
0.2*node_threshold[k]*feature_max[node_feature[k]]
If the former one is bigger, then walk to node 2*k , otherwise step into node 2*k+1.
In the last layer , walking left give rise to the prediction class 1, walking right lead to the prediction class 0.
Max Vote:
There are 57 predictions of 57 trees for test[j], simply choose the majority as final prediction.
Unfortunately , my python code cannot work properly , still require some time to fix it , sorry for the incompletion.