hw5_report

April 13, 2016

1 CS189 Spring 2016: Introduction to Machine Learning

1.1 SID : 23274190 / Name: Hye Soo Choi

1.1.1 Decision Trees for Classification

In this homework, we will implement decision trees and random forests for classification on 2 datasets: 1. the spam data 2. a census income dataset

to predict whether or not a person makes over 50k in income. In lectures, we were given a basic introduction to decision trees and how such trees are trained. We were also introduced to random forests and boosting algorithms. We have freedom to research different decision tree techniques online.

```
In [1]: import numpy as np
        import scipy.io as sio
        import pandas as pd
        from collections import Counter
        from operator import itemgetter
```

```
In [2]: spam_rawdata = sio.loadmat('./spam-dataset/spam_data.mat')
        census_rawdata = pd.read_csv('./census_data/train_data.csv')
```

```
In [98]: sp_test = spam_rawdata['test_data']
         sp_trainf = spam_rawdata['training_data']
         sp_trainv = spam_rawdata['training_labels'][0,]
```

1.1.2 Preprocessing

```
In [130]: census_rawdata.values.shape
          census_fv = census_rawdata.values[:,0:14]
          census_lb = census_rawdata.values[:,14]
          census_test = np.array(pd.read_csv('./census_data/test_data.csv').values)
```

```
In [131]: type(census_test[0,1])
```

```
Out[131]: str
```

1.1.3 Code for DecisionTree class

Choosing categorical data We keep the categories as strings. Then we implement functionality in decision trees to determine split rules based on the subsets of categorical variables that maximizes information gain.

stopping condition We stop growing tree when a node meets any of the following conditions:

1. limit the max depth of tree as 1000
2. give the lower bound on the number of data in a node as 10
3. more than 95% of labels are the majority label.

```
In [159]: def hist_labels(labels):
              def f(n):
                  if n == 1:
                      return sum(labels)
                  else:
                      return len(labels) - sum(labels)
              return f

          def powerset(orig, newset):
              if orig == []:
                  return [newset]
              else:
                  res = []
                  for s in powerset(orig[1:], newset + [orig[0]]):
                      res.append(s)
                  for s in powerset(orig[1:], newset):
                      res.append(s)
                  return res

          class DecisionTree(object):
              def __init__(self, data, labels, limit):
                  self.root = None
                  self.values = Counter(labels).keys()
                  self.data = data
                  self.labels = labels
                  self.limit = limit

              def impurity(self, left_label_hist, right_label_hist):
                  '''A method that takes in the result of a split: two histograms
                  (a histogram is a mapping from label values to their frequencies)
                  that count the frequencies of labels on the "left" and "right" side of that split.
                  The method calculates and outputs a scalar value representing the impurity
                  (i.e. the "badness") of the specified split on the input data.'''

                  left_freq = [left_label_hist(i) for i in self.values]
                  right_freq = [right_label_hist(i) for i in self.values]

                  left_total = len(left_freq)
                  right_total = len(right_freq)

                  left_prob = [i/ left_total for i in left_freq]
                  right_prob = [ i/ right_total for i in right_freq]

                  left_loss = sum([-(p * np.log(p))/np.log(2) for p in left_prob])
                  right_loss = sum([-(p * np.log(p))/np.log(2) for p in right_prob])

                  return (left_total * left_loss + right_total * right_loss)/ (left_total + right_total

              def segmenter(self, data, labels):
```

```
'''A method that takes in data and labels. When called, it finds the best split rule
for a Node using the impurity measure and input data. There are many different types
segmenters you might implement, each with a different method of choosing a threshold.
The usual method is exhaustively trying lots of different threshold values from the d
and choosing the combination of split feature and threshold with the lowest impurity 
The final split rule uses the split feature with the lowest impurity value and the th
segmenter.
'''

counter = Counter(labels)
keys = list(counter.keys())
freq = list(counter.values())
n = len(labels)
segment = []
temp = None
risk = 0
if freq[0]/n > 0.95:
    return keys[0]

elif freq[1]/n < 0.05:
    return keys[1]

elif sum(freq) < 10:
    if freq[0] > freq[1]:
        return keys[0]
    else:
        return keys[1]

else:
    num_feat = data.shape[1]
    for i in range(num_feat) :
        feat_values = data[ :,i]
        counter_fvalues = Counter(feat_values)
        fvkeys = list(counter_fvalues.keys())
        fvfreq = list(counter_fvalues.values())
        if len(fvkeys) == 1:
            continue
        elif type(feat_values[0]) == str :
            temp = self.segmenter_str(feat_values, labels,i)
        else:
            temp = self.segmenter_int(feat_values, labels,i)

        temp_segment = temp[0]
        temp_risk = temp[1]
        if risk == 0:
            risk = temp_risk
            segment = [i, temp_segment]
        elif risk > temp_risk:
            segment = [i, temp_segment]
            risk = temp_risk
    if risk == 0:
        if sum(labels) > 0.5:
            return 1
        else:
                return 0
        else:
            return segment


def segmenter_int(self, values, labels, i):
    counter = Counter(values)
    uvalues = sorted(counter.keys())
    ufreq = [counter[i] for i in uvalues]
    pair = [[values[i], labels[i]] for i in range(len(values))]
    sort_pair = sorted(pair, key= itemgetter(0))
    sort_label = []
    count = 0
    risk = 0

    for i in range(len(uvalues)):
        sort_label.append([sort_pair[j][1]  for j in range(count, count + ufreq[i])])
        count = count + ufreq[i]

    count = 0
    left_labels = []
    right_labels = sum(sort_label, [])
    for i in range(len(uvalues)-1):
        count = count + ufreq[i]
        left_labels = left_labels + sort_label[i]
        right_labels = right_labels[count:]

        left_histogram = hist_labels(left_labels)
        right_histogram = hist_labels(right_labels)
        temp_risk = self.impurity(left_histogram, right_histogram)

        if i == 0 :
            risk = temp_risk
            split = uvalues[i]
        elif risk > temp_risk:
            risk = temp_risk
            split = uvalues[i]

    def split_rule(x):
        if x == -1:
            return split
        else:
            return x > split

    return [split_rule, risk]


def segmenter_str(self, values, labels):
    print(0)
    counter = Counter(values)
    uvalues = sorted(counter.keys())
    ufreq = [counter[i] for i in uvalues]
    pair = [[values[i], labels[i]] for i in range(len(values))]
    sort_pair = sorted(pair, key= itemgetter(0))
```

```
        sort_label = []
        count = 0

        for i in range(len(uvalues)):
            sort_label.append([sort_pair[j][1]  for j in range(count, count + ufreq[i])])
            count = count + ufreq[i]

        count = 0
        left_labels = []
        right_labels = []
        risk = 0
        power_values = [l for l in powerset([j for j in range(len(uvalues))],[])
                        if len(l) < len(uvalues)/2 and len(l) > 0 ]


        for indice in power_values:

            left_labels = sum([sort_label[h] for h in indice],[])
            right_labels = sum([sort_label[h] for h in range(len(uvalues)) if h not in indice]

            left_histogram = hist_labels(left_labels)
            right_histogram = hist_labels(right_labels)
            temp_risk = self.impurity(left_histogram, right_histogram)

            if risk == 0 :
                risk = temp_risk
                split = left_labels
            elif risk > temp_risk:
                risk = temp_risk
                split = left_labels
        def split_rule(x):
            return x in split

        return [split_rule, risk]



    def train(self, data, labels):

        def construct_node(cdata,clabels, depth):
            segment = self.segmenter(cdata, clabels)


            if type(segment) != list:
                return Node(label = segment)

            elif depth > self.limit:
                if sum(clabels) > 0.5:
                    temp_label = 1
                else:
                    temp_label = 0
                return Node(label = temp_label )

            else:
                split_rule = segment[1]
                split_fv = cdata[: , segment[0]]
                split_cond = np.array([split_rule(i) for i in split_fv])
                left_data = cdata[split_cond,:]
                left_labels = clabels[split_cond]
                right_data = cdata[~ split_cond,:]
                right_labels = clabels[~split_cond]
                return Node(split_rule= segment, left = construct_node(left_data, left_labels
                            right = construct_node(right_data, right_labels, depth+1))

        self.root = construct_node(data, labels, 0)

    def predict(self, data):

        def test(feat, current):
            if current.label != None:
                return current.label
            else:
                current_rule = current.split_rule[1]
                current_index = current.split_rule[0]
                if current_rule(feat[current_index]):
                    return test(feat, current.left)
                else:
                    return test(feat, current.right)
        res = [test(data[i,:], self.root) for i in range(data.shape[0])]

        return res


class Node(object):
    def __init__(self, split_rule = None, left = None, right = None, label = None):
        self.split_rule = split_rule
        self.left = left
        self.right = right
        self.label = label
```

```
In [160]: test = DecisionTree(sp_trainf, sp_trainv, 100)
          test.train(sp_trainf, sp_trainv)
```

```
In [165]: print(test.root.split_rule[1](-1))
          print(test.root.split_rule)
          print(test.root.left.split_rule)
          print(test.root.right.split_rule)
          print(test.root.left.split_rule[1](-1))
          print(test.root.right.split_rule[1](-1))
```

```
0.0
[8, <function DecisionTree.segmenter_int.<locals>.split_rule at 0x33e2b9d08>]
[2, <function DecisionTree.segmenter_int.<locals>.split_rule at 0x33e2b9e18>]
[22, <function DecisionTree.segmenter_int.<locals>.split_rule at 0x33e29a158>]
0.0
0.0
```

Split conditions: For spam data,

1. 9th feature > 0.0
2. 3rd feature > 0.0
3. 23rd feature > 0.0

were the first three conditions that splits the data.

Bagging

```
In [ ]: pred =[0 for i in range(5857)]
        for i in range(100):
            id0 = np.random.choice(sp_trainf.shape[0], 3000, replace = True)
            test.train(sp_trainf[id0, :], sp_trainv[id0])
            pred0 = test.predict(sp_test)
            pred = [(pred[i] + pred0[i]) for i in range(5857)]
```

```
In [101]: sp_test = [1*(pred[i] > 50) for i in range(5857)]
          sp_pred = np.asarray([[i+1, sp_test[i]] for i in np.arange(5857)])
          np.savetxt('sp_test2.csv', sp_pred, fmt = '%1.u' , delimiter = ',', header = 'Id,Category',co
```

```
In [136]: test2 = DecisionTree(census_fv, census_lb, 100)
```

```
In [137]: id0 = np.random.choice(census_fv.shape[0], 1000, replace = True)
          test2.train(census_fv[id0,:], census_lb[id0])
```

```
0
0
0
0
0
0
0
0
```

```
        ---------------------------------------------------------------------------

        KeyboardInterrupt                         Traceback (most recent call last)

        <ipython-input-137-bb643f7f624c> in <module>()
          1 id0 = np.random.choice(census_fv.shape[0], 1000, replace = True)
    ----> 2 test2.train(census_fv[id0,:], census_lb[id0])


        <ipython-input-132-484401d796de> in train(self, data, labels)
        217                               right = construct_node(right_data, right_labels, depth+1))
        218
    --> 219         self.root = construct_node(data, labels, 0)
        220
        221     def predict(self, data):


        <ipython-input-132-484401d796de> in construct_node(cdata, clabels, depth)
        193
        194         def construct_node(cdata,clabels, depth):
    --> 195             segment = self.segmenter(cdata, clabels)
        196
        197


        <ipython-input-132-484401d796de> in segmenter(self, data, labels)
         87                     continue
         88                 elif type(feat_values[0]) == str :
    ---> 89                     temp = self.segmenter_str(feat_values, labels)
         90                 else:
         91                     temp = self.segmenter_int(feat_values, labels)


        <ipython-input-132-484401d796de> in segmenter_str(self, values, labels)
        164         right_labels = []
        165         risk = 0
    --> 166         power_values = [l for l in powerset([j for j in range(len(uvalues))],[])
        167                         if len(l) < len(uvalues)/2 and len(l) > 0 ]
        168


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         13     else:
         14         res = []
    ---> 15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
         17         for s in powerset(orig[1:], newset):


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
    ---> 17         for s in powerset(orig[1:], newset):
         18             res.append(s)
         19         return res


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         13     else:
         14         res = []
    ---> 15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
         17         for s in powerset(orig[1:], newset):


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
    ---> 17         for s in powerset(orig[1:], newset):
         18             res.append(s)
         19         return res
        <ipython-input-132-484401d796de> in powerset(orig, newset)
         13     else:
         14         res = []
    ---> 15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
         17         for s in powerset(orig[1:], newset):


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
    ---> 17         for s in powerset(orig[1:], newset):
         18             res.append(s)
         19         return res


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
    ---> 17         for s in powerset(orig[1:], newset):
         18             res.append(s)
         19         return res


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         13     else:
         14         res = []
    ---> 15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
         17         for s in powerset(orig[1:], newset):


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
    ---> 17         for s in powerset(orig[1:], newset):
         18             res.append(s)
         19         return res


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         13     else:
         14         res = []
    ---> 15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
         17         for s in powerset(orig[1:], newset):


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
    ---> 17         for s in powerset(orig[1:], newset):
         18             res.append(s)
         19         return res
        <ipython-input-132-484401d796de> in powerset(orig, newset)
         13     else:
         14         res = []
    ---> 15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
         17         for s in powerset(orig[1:], newset):


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
    ---> 17         for s in powerset(orig[1:], newset):
         18             res.append(s)
         19         return res


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         13     else:
         14         res = []
    ---> 15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
         17         for s in powerset(orig[1:], newset):


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         13     else:
         14         res = []
    ---> 15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
         17         for s in powerset(orig[1:], newset):


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         13     else:
         14         res = []
    ---> 15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
         17         for s in powerset(orig[1:], newset):


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
    ---> 17         for s in powerset(orig[1:], newset):
         18             res.append(s)
         19         return res


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
    ---> 17         for s in powerset(orig[1:], newset):
         18             res.append(s)
         19         return res


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         13     else:
         14         res = []
    ---> 15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
         17         for s in powerset(orig[1:], newset):


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
    ---> 17         for s in powerset(orig[1:], newset):
         18             res.append(s)
         19         return res


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         13     else:
         14         res = []
    ---> 15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
         17         for s in powerset(orig[1:], newset):


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         13     else:
         14         res = []
    ---> 15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
         17         for s in powerset(orig[1:], newset):


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
    ---> 17         for s in powerset(orig[1:], newset):
         18             res.append(s)
         19         return res


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         15         for s in powerset(orig[1:], newset + [orig[0]]):
         16             res.append(s)
    ---> 17         for s in powerset(orig[1:], newset):
         18             res.append(s)
         19         return res


        <ipython-input-132-484401d796de> in powerset(orig, newset)
         13     else:
```

```
     14         res = []
---> 15         for s in powerset(orig[1:], newset + [orig[0]]):
     16             res.append(s)
     17         for s in powerset(orig[1:], newset):


    <ipython-input-132-484401d796de> in powerset(orig, newset)
     13     else:
     14         res = []
---> 15         for s in powerset(orig[1:], newset + [orig[0]]):
     16             res.append(s)
     17         for s in powerset(orig[1:], newset):


    <ipython-input-132-484401d796de> in powerset(orig, newset)
     13     else:
     14         res = []
---> 15         for s in powerset(orig[1:], newset + [orig[0]]):
     16             res.append(s)
     17         for s in powerset(orig[1:], newset):


    KeyboardInterrupt:
```

```
In [111]: for i in range(100):
              id0 = np.random.choice(census_fv.shape[0], 3000, replace = True)
              test.train(census_fv[id0, :], census_fv[id0])
              pred0 = test.predict(sp_test)
              pred = [(pred[i] + pred0[i]) for i in range(5857)]
```

```
Out[111]: (32724, 14)
```

Kaggle score This results in Kaggle score 0.6837 for spam data.