hw5

April 9, 2015

1 Homework 5

- Name: Austin
- SID: 23826762
- Repro: Open up hw5.ipynb in IPython Notebook.

1.1 a) Techniques implemented

The stopping criteria for my decision trees fall into 2 terminating cases. First, if all the remaining data shares a single label, make a leaf node. Second, if the depth is hit at any node, then the most common label is chosen for that leaf node.

In terms of splitting heuristics, I tried median of a feature, mean of a feature, and the mean of the mean of all features, all of which seemed to perform similarly, because they almost always spllt between 0 and everything else. For speedups, I tried picking random features to split on instead of testing all features, which expectably led to a decrease of a few percent in accuracy.

To test the effect of various hyperparameters (mostly DTree depth and RForest size), I used 12-fold cross validation.

In my random forests, I implemented bagging of the data in addition to random subsets of the features at each node.

1.2 b) Features added or removed

I didn't add or remove any features =P.

1.3 c) Results

On my decision tree, with depth 15 and mean of features, I get about 0.825 accuracy on my training data through cross-validation, and a kaggle score of 0.79269

On my random forests, with depth 12 and 24 trees, I get about 0.835 accuracy on my training data through cross-validation, and a kaggle score of 0.81523.

1.4 d) Example classification on DTree

After training a depth 15 DTree on all the data, this is an example traversal on the first test data point.

```
feature 28 (!):     1.0 > 0.703209590101
feature 19 (meter): 0.0 <= 0.29561752988
feature 31 (&):     0.0 <= 0.335042735043
feature 3  (money): 0.0 <= 0.183125599233
feature 26 ($):     0.0 <= 1.01434878587
```

This point was classified as 1 (spam).

1.5 e) Common top-level splits in RForest

Out of 100 DTrees in a Random Forest, the breakdown of first splits is as follows:

```
17% 28 (!)
16% 19 (meter)
15% 3  (money)
15% 16 (volumes)
10% 6  (prescription)
6% 0
4% 9
3% 29
2% 5
2% 4
2% 26
1% 1
1% 10
1% 12
1% 18
1% 27
1% 31
1% 7
```

All of the splits were thresholded by 0.

```
In []: import numpy as np
       import scipy.io
       import math
       import random

       # Load the data
       mat = scipy.io.loadmat('spam-dataset/spam_data.mat')
       training_data = mat['training_data']
       training_labels = mat['training_labels']
       training_labels = np.squeeze(np.asarray(training_labels))
       test_data = mat['test_data']
```

```
In [80]: class Node(object):
             def __init__(self, split_rule, left, right):
                 self.split_rule = split_rule
                 self.left = left
                 self.right = right

             def __repr__(self, level=0):
                 ret = "- "*level + str(self.split_rule) + "\n"
                 ret += self.left.__repr__(level + 1)
                 ret += self.right.__repr__(level + 1)
                 return ret

         class LeafNode(object):
             def __init__(self, label):
                 self.label = label

             def __repr__(self, level=0):
                 return "- "*level + str(self.label) + "\n"
```

```
class DTree(object):
    def __init__(self, depth, impurity, segmentor):
        self.depth = depth
        self.impurity = impurity
        self.segmentor = segmentor

    def train(self, data, labels):
        self.root = self.growTree(data, labels, self.depth)

    def growTree(self, data, labels, depth):
        if depth == 0:
            return LeafNode(1 if sum(labels) > len(labels) / 2 else 0)

        # Base case: if all labels are 0
        if sum(labels) == 0:
            return LeafNode(0)

        # Base case: if all labels are 1
        if sum(labels) == len(labels):
            return LeafNode(1)

        # Recursive case:
        split, threshold = self.segmentor(data, labels, self.impurity)
        lp = np.array([f <= threshold for f in data.T[split]])
        rp = np.array([f > threshold for f in data.T[split]])

        left_node = self.growTree(data[lp], labels[lp], depth - 1)
        right_node = self.growTree(data[rp], labels[rp], depth - 1)
        return Node((split, threshold), left_node, right_node)

    # Traverse through the D-Tree for this data point
    def traverse(self, datum):
        node = self.root
        while type(node) is not LeafNode:
            split, threshold = node.split_rule
            node = node.left if datum[split] <= threshold else node.right

        return node.label

    def predict(self, data):
        result = [self.traverse(datum) for datum in data]
        return result

    # Returns the proportion of data correctly labelled by predict
    def score(self, data, labels):
        predictions = self.predict(data)
        return sum(d == l for d, l in zip(predictions, labels)) / len(labels)

def permute_both(a, b):
    p = np.random.permutation(len(a))
    return a[p], b[p]
```

In [113]:
```
# Various impurity and segmentor functions
def entropy(left, right):
```

```
    if left == 0 or right == 0: return 0
    p_left, p_right = left / (left + right), right / (left + right)
    return -p_left * np.log2(p_left) + -p_right * np.log2(p_right)

def entropy_impurity(left_hist, right_hist):
    result = sum(left_hist) * entropy(*left_hist) + sum(right_hist) * entropy(*right_hist)
    return result / (sum(left_hist) + sum(right_hist))

def impurity_func(data, labels, impurity):
    def result(split_rule):
        feature = data.T[split_rule[0]]
        threshold = split_rule[1]

        a, b, c, d = 0, 0, 0, 0
        for label, f in zip(labels, feature):
            if f <= threshold:
                if label == 1:
                    c += 1
                else:
                    d += 1
            else:
                if label == 1:
                    a += 1
                else:
                    b += 1

        return impurity((a, b), (c, d))
    return result

def median_segmentor(data, labels, impurity):
#     split_rules = [(i, np.median(feature)) for i, feature in enumerate(data.T)]
    split_rules = [(i, np.mean(np.mean(data, 1))) for i, feature in enumerate(data.T)]
    return min(split_rules, key=impurity_func(data, labels, impurity))

from random import randint
def random_segmentor(data, labels, impurity):
    i = randint(0, len(data.T)-1)
#     print(np.mean(np.mean(data, 1)))
    return (i, np.mean(np.mean(data, 1)))

def forest_segmentor(data, labels, impurity):
    split_rules = [(i, np.mean(np.mean(data, 1))) for i, feature in enumerate(data.T)]
    # Random subset of 6 splits at each node (roughly sqrt(32))
    split_rules = random.sample(split_rules, 6)

    return min(split_rules, key=impurity_func(data, labels, impurity))

def all_segmentor(data, labels, impurity):
    features = np.random.choice(32, 6, replace=False)
    split_rules = {(fe, i) for fe in features for i in data.T[fe]}
    return min(split_rules, key=impurity_func(data, labels, impurity))
```

In [137]:
```
# Random forests
```

```
        class RForest(object):
            def __init__(self, depth, impurity, segmentor, num):
                self.trees = [DTree(depth, impurity, segmentor) for _ in range(num)]

            def train(self, dataX, labelsX):
                for tree in self.trees:
                    # Bagging
                    r = np.random.choice(len(dataX), int(len(dataX) / 2))
                    data, labels = dataX[r], labelsX[r]
                    tree.train(data, labels)

            def predict(self, data):
                a = [tree.predict(data) for tree in self.trees]
                result = []
                for i in range(len(data)):
                    total = sum(guess[i] for guess in a)
                    result.append(1 if total > (len(self.trees) / 2) else 0)
                return result


            # Returns the proportion of data correctly labelled by predict
            def score(self, data, labels):
                predictions = self.predict(data)
                return sum(d == l for d, l in zip(predictions, labels)) / len(labels)
```

In []:
```
def cross_validate():
    l = np.squeeze(np.asarray(training_labels))
    d, l = permute_both(training_data, training_labels)

    # Cross validation to test how good this D-Tree is
    folds = 12
    interval = 431

    total = 0
    # cross validation
    for k in range(folds):
        kemails, klabels = [], []
        tree = RForest(12, entropy_impurity, forest_segmentor, 24)
#         tree = DTree(15, entropy_impurity, median_segmentor)
#         print(tree.root)
        for i in range(folds):
            if i != k:
                start, end = i * interval, (i + 1) * interval
                kemails.append(d[start:end])
                klabels.append(l[start:end])
        tree.train(np.concatenate(kemails), np.concatenate(klabels))

        start, end = k * interval, (k + 1) * interval
        total += tree.score(d[start:end], l[start:end])
        print(total / (k + 1))

    print(total / folds)

%prun cross_validate()
```

In [132]:
```
# Train the DTree on all the training data
d, l = permute_both(training_data, training_labels)
tree = RForest(12, entropy_impurity, forest_segmentor, 24)
tree.train(d, l)

# Predict the labels for the test data
result = tree.predict(test_data)

# Write the results to a csv
f = open('spam5.csv', 'w')
f.write('Id,Category\n')
for i in range(len(result)):
    f.write("{0},{1}\n".format(i + 1, result[i]))
f.close()
```