CS 486 Assignment 3

Jason Sun (#20387090)

July 8, 2015

Question 1: Decision Tree Learning

Source Code

Written in python.

```
import csv
import math

# A decision node, corresponds to a word feature.
class Node:
def __init__(self, word):
self.word = word
self.labels = []
self.documents = []
self.positives = None
self.negatives = None
self.importance = None
self.idx_best_word = None

def add_sample(self, documents, labels):
self.documents = documents
self.labels = labels

def add_positive(self, pos):
self.positives = pos

def add_negative(self, neg):
self.negatives = neg

def childrens(self):
leaves = []
if self.is_child():
leaves.append(self)
else:
[leaves.append(leaf) for leaf in self.positives.childrens()]
[leaves.append(leaf) for leaf in self.negatives.childrens()]
return leaves

def is_child(self):
return type(self.word) == str
```

```
def printout(self, words, depth):
if self.positives:
self.positives.printout(words, depth + 1)

print (depth * '--'), depth, '> ',

if type(self.word) == str:
print "[LEAF    ", self.word,
else:
print "[NOTLEAF ", words[self.word],
print  " ", self.importance, "]"

if self.negatives:
self.negatives.printout(words, depth + 1)


# Classify document given a tree
def classify(root, doc):
node = root
while not node.is_child():
if doc[node.word] == 1:
node = node.positives
else:
node = node.negatives
return node.word

# Returns % docs correctly classified using the tree
def test(tree, data, labels):
correct = 0
classifications = []
for doc in data:
classifications.append(classify(tree, doc))
for i in range(len(labels)):
if labels[i] == classifications[i]:
correct += 1
return float(correct) / len (labels)

# Most frequently seen
def plurality(word_idx, matrix, labels, word):
count1 = 0
count2 = 0
for i in range(len(matrix)):
if matrix[i][word_idx] == word:
if labels[i] == '1':
count1 = count1+1
else:
count2 = count2+1
if count1 > count2:
return '1'
else:
return '2'
```

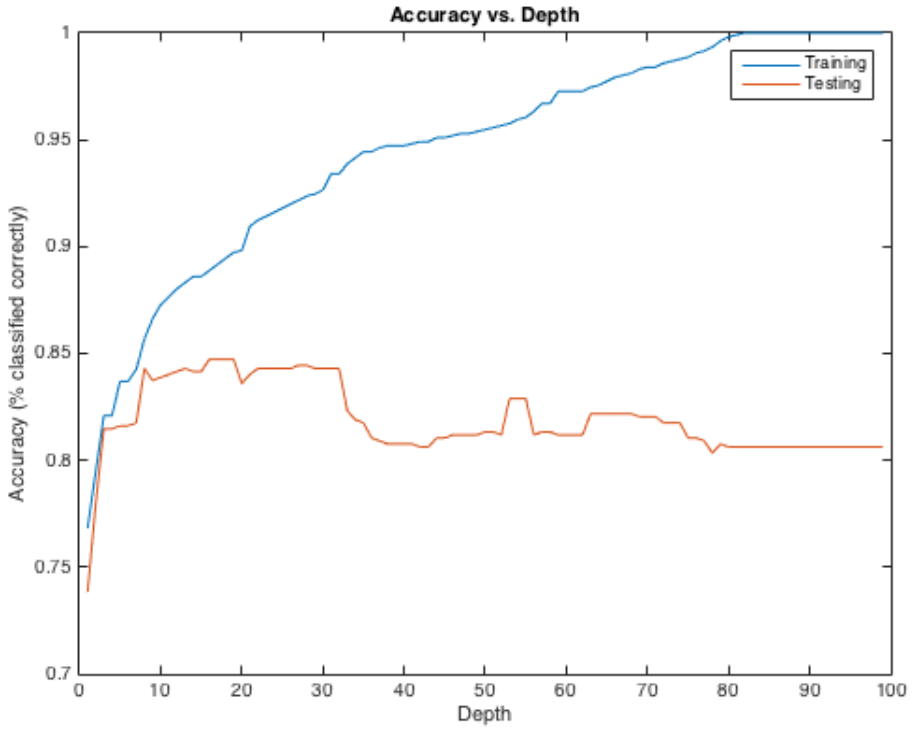
```
# returns the entropy of a random variable that is true with the probability q
def entropy(q):
if q == 0 or q == 1:
return 0
entropy = -1* (q * math.log(q,2) + (1-q) * math.log(1-q, 2))
return entropy

# Importance is the information gain from the word
def importance(word_idx, train_data_sparse, labels):
p = labels.count('2')
n = len(labels) - p
if p+n == 0:
return 0.0

H = entropy(float(p) / (p+n))
remainder = 0
for word in [0,1]:
pk = 0
nk = 0
for doc in range(len(train_data_sparse)):
if train_data_sparse[doc][word_idx] == word:
if labels[doc] == '2':
pk = pk+1
else:
nk = nk+1
if pk+nk != 0:
remainder = remainder + ((float(pk+nk) / (p+n)) *
entropy(float(pk)/ (pk+nk)))
return H - remainder

# Refer to AIMA chpt 20 decision tree learner algorithm
def decision_tree_learn(matrix, labels,
test_data, test_labels,
words, max_depth):
# For each word, see how important it is.
important_words = []
for word_idx in range(len(matrix[0])):
important_words.append(importance(word_idx, matrix, labels))
# Use the most important word to start as root of decision tree
important_word = important_words.index(max(important_words))
root = Node(important_word)
neg_docs = []
neg_labels = []
pos_docs = []
pos_labels = []

for i in range(len(matrix)):
doc = matrix[i]
label = labels[i]
if doc[important_word]:
pos_docs.append(doc)
```

```
pos_labels.append(label)
else:
neg_docs.append(doc)
neg_labels.append(label)

root.add_negative(Node(plurality(important_word, neg_docs, neg_labels, 0)))
root.add_positive(Node(plurality(important_word, pos_docs, pos_labels, 1)))
root.positives.add_sample(pos_docs, pos_labels)
root.negatives.add_sample(neg_docs, neg_labels)
root.importance = max(important_words)
depth = 1
test_percent = []
train_percent = []
print "Depth\tTraining\tTesting   \tImportance\tWord"

while depth < max_depth:
test_percent.append(test(root, test_data, test_labels))
train_percent.append(test(root, matrix, labels))
leaves = root.childrens()

for leaf in leaves:
# check if already calculated leaf
if leaf.importance == None:
# find the best word, make it next leaf
important_words = []
for i in range(len(matrix[0])):
# importance for each document
important_words.append(
importance(i, leaf.documents, leaf.labels) *
len(leaf.documents)
)

leaf.importance = max(important_words)
leaf.idx_best_word = important_words.index(max(important_words))

leaves_gain = [leaf.importance for leaf in leaves]
best_leaf = leaves[leaves_gain.index(max(leaves_gain))]
# expand the best leaf
best_leaf.word = best_leaf.idx_best_word

print depth, "\t",
print format(train_percent[depth-1], '.5f'), "\t",
print format(test_percent[depth-1], '.5f'), "\t",
print format(best_leaf.importance, '.5f'), "\t",
print words[best_leaf.word]

pos_docs = []
pos_labels = []
neg_docs = []
neg_labels = []

for i in range(len(best_leaf.documents)):
```

```
label = best_leaf.labels[i]
doc = best_leaf.documents[i]
if doc[best_leaf.idx_best_word]:
pos_docs.append(doc)
pos_labels.append(label)
else:
neg_docs.append(doc)
neg_labels.append(label)

best_leaf.add_negative(
Node(plurality( best_leaf.idx_best_word,
best_leaf.documents,
best_leaf.labels,
0))
)
best_leaf.add_positive(
Node(plurality( best_leaf.idx_best_word,
best_leaf.documents,
best_leaf.labels,
1))
)

best_leaf.positives.add_sample(pos_docs, pos_labels)
best_leaf.negatives.add_sample(neg_docs, neg_labels)
depth += 1

# write the percentages correct to a .csv
with open('results.csv', 'wb') as f:
writer = csv.writer(f)
writer.writerows([train_percent])
writer.writerows([test_percent])

return root


# load given data
words = []
train_data = []
train_labels = []
test_data = []
test_labels = []
with open('words.txt', 'r') as f:
for line in f:
words.append(line.split()[0])
with open('trainData.txt', 'r') as f:
for line in f:
digits = [int(x) for x in line.split()]
train_data.append(digits)
with open('trainLabel.txt', 'r') as f:
for line in f:
train_labels.append(line.split()[0])
with open('testData.txt', 'r') as f:
```

```
for line in f:
digits = [int(x) for x in line.split()]
test_data.append(digits)
with open('testLabel.txt', 'r') as f:
for line in f:
test_labels.append(line.split()[0])
print len(words), "\twords"
print len(train_labels), "\ttraining documents"
print len(test_labels), "\ttesting documents"
# create unscalable big-huge matrix
train_matrix = [[0] * len(words) for i in range(len(train_labels))]
for i in train_data:
train_matrix[i[0]-1][i[1]-1] = 1
test_matrix = [[0] * len(words) for i in range(len(test_labels))]
for i in test_data:
test_matrix[i[0]-1][i[1]-1] = 1


# Build a decision tree using information gain:
max_depth = 22
root = decision_tree_learn(train_matrix, train_labels,
test_matrix, test_labels,
words, max_depth)
root.printout(words, 0)
```

Overfitting



Overfitting occurs when the accuracy of the test data has gone past its apex. At depth of 22 to 25 it reaches the maximal accuracy of 84.3%, and afterward there is a noticable a drop in accuracy and from there onward, it is never gets as good as before.

Highest Accuracy Decision Tree

I apologize for this poor representation. It is generated by the program. I tried to generate a nice looking `.dot` file and compile it to a nice image, but it is too time consuming this time.

- The graph sort of needs to be viewed sideways.
- The number indicates which depth the node is at.
- Each node may have up to two children, left and right (representing negative and positive, respectively).
  - For example, the 0th depth with the word "writes" has 2 children of depth 1 (called "god" and "graphics").
- This is an in-order traversal, and positives are printed first.
  - So to right of a node (child above the parent) is a decision that categories yes.
  - And to the left of a node (child below the parent) is a decision that categories no.
  - For example, the 0th depth parent has positive of graphics and negative of god.
- Positive categorizes the computer graphics topic (because we like graphics more!). Negative categorizes the atheism topic.

```
---- 2 >  [LEAF     2   0.0 ]
-- 1 >  [NOTLEAF  graphics   58.0177474641 ]
------ 3 >  [LEAF     2   0.0 ]
---- 2 >  [NOTLEAF  image   39.4532227953 ]
---------- 5 >  [LEAF     2   0.0 ]
-------- 4 >  [NOTLEAF  program   20.1251024168 ]
------------ 6 >  [LEAF     2   0.0 ]
---------- 5 >  [NOTLEAF  comp   14.2756177886 ]
-------------- 7 >  [LEAF     2   0.0 ]
------------ 6 >  [NOTLEAF  uchicago   11.2378110809 ]
---------------- 8 >  [LEAF     2   0.0 ]
-------------- 7 >  [NOTLEAF  edges   11.7642479117 ]
------------------ 9 >  [LEAF     2   0.0 ]
---------------- 8 >  [NOTLEAF  slow   12.3758419719 ]
------------------ 9 >  [LEAF     1   8.64573220814 ]
------ 3 >  [NOTLEAF  that   38.305602883 ]
---------- 5 >  [LEAF     1   0.0 ]
-------- 4 >  [NOTLEAF  god   11.1589005692 ]
------------ 6 >  [LEAF     1   0.0 ]
---------- 5 >  [NOTLEAF  keith   13.5422497561 ]
------------ 6 >  [LEAF     2   8.4441690871 ]
 0 >  [NOTLEAF  writes   0.214992437297 ]
---- 2 >  [LEAF     1   6.79328213174 ]
-- 1 >  [NOTLEAF  god   62.5805204996 ]
-------- 4 >  [LEAF     1   7.01853173263 ]
------ 3 >  [NOTLEAF  wrote   19.9231807133 ]
------------ 6 >  [LEAF     2   None ]
---------- 5 >  [NOTLEAF  windows   11.139396162 ]
------------ 6 >  [LEAF     1   None ]
-------- 4 >  [NOTLEAF  people   18.1654601324 ]
------------ 6 >  [LEAF     1   0.0 ]
---------- 5 >  [NOTLEAF  religious   17.5871696453 ]
-------------- 7 >  [LEAF     2   7.7126434744 ]
------------ 6 >  [NOTLEAF  an   12.4103967948 ]
---------------- 8 >  [LEAF     1   3.60964047444 ]
-------------- 7 >  [NOTLEAF  he   12.4868711195 ]
---------------- 8 >  [LEAF     2   9.51352907768 ]
---- 2 >  [NOTLEAF  that   42.1086722958 ]
-------- 4 >  [LEAF     1   0.0 ]
------ 3 >  [NOTLEAF  bible   15.9742606126 ]
---------- 5 >  [LEAF     1   0.0 ]
-------- 4 >  [NOTLEAF  atheist   12.7229668929 ]
------------ 6 >  [LEAF     1   0.0 ]
---------- 5 >  [NOTLEAF  keith   13.4998396154 ]
------------ 6 >  [LEAF     2   9.52200881389 ]
```

Features Used

The features used for the highest accuracy are:

```
3566    words
1061    training documents
```

```
707     testing documents
Depth   Training        Testing         Importance      Word
1       0.76814         0.73833         62.58052        god
2       0.79453         0.77935         58.01775        graphics
3       0.82092         0.81471         42.10867        that
4       0.82092         0.81471         39.45322        image
5       0.83695         0.81612         38.30560        that
6       0.83695         0.81612         20.12510        program
7       0.84260         0.81754         19.92318        wrote
8       0.85674         0.84300         18.16546        people
9       0.86616         0.83734         17.58717        religious
10      0.87276         0.83876         15.97426        bible
11      0.87653         0.84017         14.27562        comp
12      0.88030         0.84158         12.72297        atheist
13      0.88313         0.84300         13.49984        keith
14      0.88596         0.84158         12.41040        an
15      0.88596         0.84158         12.48687        he
16      0.88878         0.84724         11.23781        uchicago
17      0.89161         0.84724         11.76425        edges
18      0.89444         0.84724         12.37584        slow
19      0.89727         0.84724         11.15890        god
20      0.89821         0.83593         13.54225        keith
21      0.90952         0.84017         11.13940        windows
```

In my opinion these selected words makes sense. For instance, the top two most important words "god" and "graphics" has a strong utility, meaning if I see those words in an article, I'm already fairly sure of which topic the document is from. Of course, the lesser important words still make sense, but not as obvious. So in the inductive sense, it all makes sense, just less and less so to the human reader.

Question 2: Naive Bayes Model

Source Code

Written in matlab.

Function that generates the bayes net:

```
function [log_doc1, log_doc2, log_likelihood] = naive_bayes_net(matrix, label)

% split training set into two documents
doc1 = [];
doc2 = [];
for i = 1:length(label)
   if label(i) == 1
       doc1 = [doc1; matrix(i,:)];
   else
       doc2 = [doc2; matrix(i,:)];
   end
end

% calc Pr(word|label1) and Pr(word|label2)
pr_word_doc1 = zeros(length(matrix(1,:)),1);
```

```
pr_word_doc2 = zeros(length(matrix(1,:)),1);

% start count at 1 for Laplace smoothing
for word = 1:length(matrix(1,:))
   count = 1;
   for i = 1:size(doc1, 1)
      count = count + doc1(i,word);
   end
   pr_word_doc1(word) = count / size(doc1, 1);

   count = 1;
   for i = 1:size(doc2, 1)
      count = count + doc2(i,word);
   end
   pr_word_doc2(word) = count / size(doc2, 1);

end

log_doc1 = log(pr_word_doc1);
log_doc2 = log(pr_word_doc2);
log_likelihood = abs(log_doc1 - log_doc2);

end
```

The main program:

```
loadScript
[log_doc1, log_doc2, log_likelihood] = naive_bayes_net(trainDataSparse, trainLabel);

% list the N most discrimative word features
[sortedVals, sortedIdx] = sort(log_likelihood(:), 'descend');
N = 10;
maxIdxs = sortedIdx(1:N);
disp(['Top ', num2str(N), ' most discriminative words:']);
for i = 1:length(maxIdxs)
   disp([num2str(log_likelihood(maxIdxs(i))), words(maxIdxs(i))]);
end

% return the % of correctly classified articles
% each document (represented by a row in sparse table), multiply
% each word (a cell within a row) with the log probability in the doc

% for the training set
trainClassDoc1 = (trainDataSparse * log_doc1);
trainClassDoc2 = (trainDataSparse * log_doc2);
% then classify which document it is by whichever is larger
trainClassification = (trainClassDoc1 < trainClassDoc2) + 1;
trainCorrects = sum(trainClassification == trainLabel) / length(trainLabel);
disp(['Classified ', num2str(trainCorrects * 100), '% correct for training data.']);

% for the testing set
[test_log_doc1, test_log_doc2, ~] = naive_bayes_net(trainDataSparse, trainLabel);
testClassDoc1 = (testDataSparse * test_log_doc1);
```

```
testClassDoc2 = (testDataSparse * test_log_doc2);
% then classify which document it is by whichever is larger
testClassification = (testClassDoc1 < testClassDoc2) + 1;
testCorrects = sum(testClassification == testLabel) / length(testLabel);
disp(['Classified ', num2str(testCorrects * 100), '% correct for testing data.']);
```

Results

Top 10 most discriminative words:

```
'4.4242' 'graphics'
'3.9752' 'atheism'
'3.9286' 'religion'
'3.8545' 'keith'
'3.8545' 'moral'
'3.8545' 'evidence'
'3.8286' 'atheists'
'3.7837' 'god'
'3.7463' 'bible'
'3.7173' 'christian'
```

These seem like good word features. I can classify each of these words into the athesim or the graphics category. The only word that puzzles me is "keith".

Training and test accuracy

Classified 77.2856% correct for training data. Classified 0.72136% correct for testing data.

Independence assumption

The assumption that word features are independent is not a reasonable assumption. Same or similar words to what was used before are more likely to show up again in the future because when writing involves context.

For example, translation of documents to a different language is a difficult process, because the choice of words depends on context, especially in cases where multiple words have similar meanings.

Improvements to the Naive Bayes model

The improvement is difficult without changing too much the modeling method. Introducing a dependency of words to the model would change it from being a naive Bayes model. Nevertheless, few improvements can be made, as discussed in an MIT paper.[1]

The improvements can be generalized to be:

1. Analysing term frequency
2. Analysing document frequency
3. Analysing document length frequency

[1] http://people.csail.mit.edu/jrennie/papers/icml03-nb.pdf

Decision tree model vs Naive Bayes model

The decision tree modeling approach works better because it can model importance of words based in a hierarchical fashion. If a word is close to the root node, then it will have a greater effect on the decision. For instance, the keyword 'graphics' showed up, then we probably do not need to look at any more words in the document before making a decision. This means word ordering matters, so a document containing just "god graphics" is classified as atheism topic, which is differently than "graphics god", classified as graphics topic. Whereas the naive Bayes model gives every word an equal weighting, so that "god graphics" is the same category as "graphics god", both being in the graphics topic.