Classifying Edibility of Mushrooms

Grayson Leonard Matt Schartman
University of California, Santa Barbara
June 2012

Abstract

Various machine learning algorithms can be used to classify a set of examples. When picking an algorithm to use, one should pick in accordance with the data they are making decisions on. We chose to compare multiple learning algorithms for classifying mushrooms as poisonous or edible. Beginning with a list of ≈8,000 samples of mushrooms with various attributes. The main goal of our endeavors was to determine whether or not a mushroom would kill you. Given death is an extreme consequence for guessing incorrectly, we hoped to find a learning algorithm which would give us the least error.

In this report we will discuss the **Naïve Bayesian** algorithm, the **ID3** algorithm, and the **Random Forest** ensemble learning technique. We will then look at implementations of each and show a bit of the final product of each algorithm. This will be followed by graphical analysis of the algorithms. And finally, the algorithms will be rated and judged accordingly.

SUMMARIES

Naïve Bayes

The Naïve Bayes classifier is based off Bayes algorithm with a large assumption of independent attributes. The function which the algorithm is based on is as follows:

$$v_{nb} = argmax_{v_j \in \{yes,no\}} [\, P(v_j) \prod_{i=0}^{n} P(a_i \mid v_j) \,]$$

One may also include the m-estimate for small data sets where certain values of probability will reach 0. However, with our dataset we found it unnecessary to implement, achieving good results every time. This algorithm works by counting up each of the matching attributes where the class matches the one you are trying to solve and then multiplying by each other. In addition, we divided our training data to include the same amount of each class therefore we can say: P(v_i) = P(v_j) and we may remove that part of the equation, simplifying it even more. The runtime for this algorithm is the only problem we ran into when testing. We found that the Naive Bayes brute-force algorithm we originally implemented in this way regularly produce output in about 44 seconds; highly inefficient given it took that long even for a small dataset.

The brute-force Naïve Bayes works in programming by first taking each mushroom test and passing it through the Naïve Bayes algorithm. It then counts up the number of times each attributes occurs in the edible training data (or positive/yes); then keeps a running total of # times each attribute occurred and multiplies it by a running multiplier. Once that finishes for all the positive training data, it then does the same thing for the negative (poisonous) training data. By storing the multiplications in the pos_prob and neg_prob variables, we are able to compare them at the end without having to do anything else. This works without any additional division or multiplication because:

1) We construct the training data so that: # of v_i = # of v_j
2) The P(v_i) = P(v_j)
3) We assume the attributes are independent of each other

Our second implementation worked as follows:

In order to reduce the runtime, one could create two lists of lists (or two tables in other words). Let us call the outer lists L_y and L_n for the positive and negative table, respectively. The inner lists will be called L_i where i $\epsilon \{1..n\}$ where n = # of attributes. Each L_y and L_n would maintain a table of the # of times each value of every attribute occurs per positive and negative in the training data. The L_i lists will be of length equal to the # of possible values an attribute can take.

These tables can then be referenced for each test mushroom and the runtime would be reduced to O(n), making the algorithm very fast.
Here are the results of the optimized algorithm:

Runtime: 0.4 seconds Accuracy: 99.3%

Example tables:

Positive table:

```
[       [241, 0, 1403, 1165, 107, 21],
        [1123, 0, 1111, 703],
        [835, 39, 22, 713, 13, 41, 13, 472, 485, 304],
        [2068, 869],
        [293, 316, 0, 0, 0, 0, 2328, 0, 0],
        [47, 0, 2890],
        [2150, 787],
        [2731, 206],
        [250, 703, 0, 152, 142, 0, 17, 615, 329, 78, 639, 12],
        [987, 1950],
        [1425, 394, 0, 642, 0, 145, 331],
        [309, 3, 48, 2577],
        [346, 148, 56, 2387],
        [3, 0, 0, 450, 47, 425, 71, 1941, 0],
        [37, 0, 0, 438, 47, 435, 73, 1907, 0],
        [2937, 0],
        [22, 25, 2890, 0],
        [0, 2672, 265],
        [0, 752, 37, 0, 0, 2148],
        [1243, 1290, 9, 37, 0, 12, 33, 299, 14],
        [290, 165, 251, 610, 828, 793],
        [953, 81, 196, 90, 65, 141, 1411]
]
```

Condensed Negative table:

```
[[36, 4, 1304, 1159, 434, 0], [568, 4, 1295, 1070], [749, 90, 9, 607, 0, 66, 0, 657, 250, 509], [477,
2460], [0, 0, 147, 421, 1619, 27, 97, 193, 433], [13, 0, 2924], [2851, 86], [1273, 1664], [54, 84,
1282, 404, 377, 20, 0, 469, 36, 0, 193, 18], [1434, 1503], [1401, 34, 0, 193, 0, 0, 1309], [110, 7,
1677, 1143], [110, 61, 1602, 1164], [323, 325, 27, 0, 0, 967, 0, 1288, 7], [330, 332, 27, 0, 0, 960,
0, 1266, 22], [2937, 0], [0, 0, 2930, 7], [27, 2855, 55], [0, 1316, 0, 970, 27, 624], [176, 164, 0,
1191, 55, 0, 0, 1351, 0], [0, 42, 0, 291, 2102, 502], [563, 427, 29, 763, 205, 0, 950]]
```

ID3

The ID3 algorithm works by creating a decision tree which can then be used to classify unknown samples of mushrooms. Given a list of possible attributes each mushroom could have and a list of possible values each attribute could take, the tree is built by continuously taking the attribute with the highest gain and using it as the current node to split the data across. This gain is calculated using a gain function that itself uses a function to calculate the entropy of a set of examples. The higher the entropy at a given point, the higher the potential to improve the classification. Gain and entropy can be calculated like so:

Entropy(S) = p_+ (- $log_2 p_+$) + p_- ($log_2 p_-$ -)

Gain(S, A) = Entropy(S) - $\sum_{\nu \epsilon Values(A)} \frac{|S_v|}{|S|}$ Entropy(S_v)

Each node can be one of three things: an attribute, an attribute value, or a decision. The tree starts out with a root node being the attribute with the highest gain. Each child of this root node is every possible value that attribute could take on. When classifying a test example, the path taken down the tree is the path that corresponds to the value that example has for the given attribute. After each possible value branch, the next branch is either the attribute with the next highest gain, or a decision that the example is either poisonous or edible. This continues until each branch leads ultimately to only leaf nodes with decisions, potentially through all the possible attributes.

Random Forest

A random forest is named as such because it is a joint construction of multiple decision trees. Each decision tree is constructed as a model with only a few select attributes in the data set. Our algorithm was constructed based off Ho's formulation, which implements stochastic discrimination, allowing overtraining of the data. Instead of choosing attributes with higher gain/entropy, Ho's method consists of randomly grabbing attributes from the set of all attributes for each tree constructed.

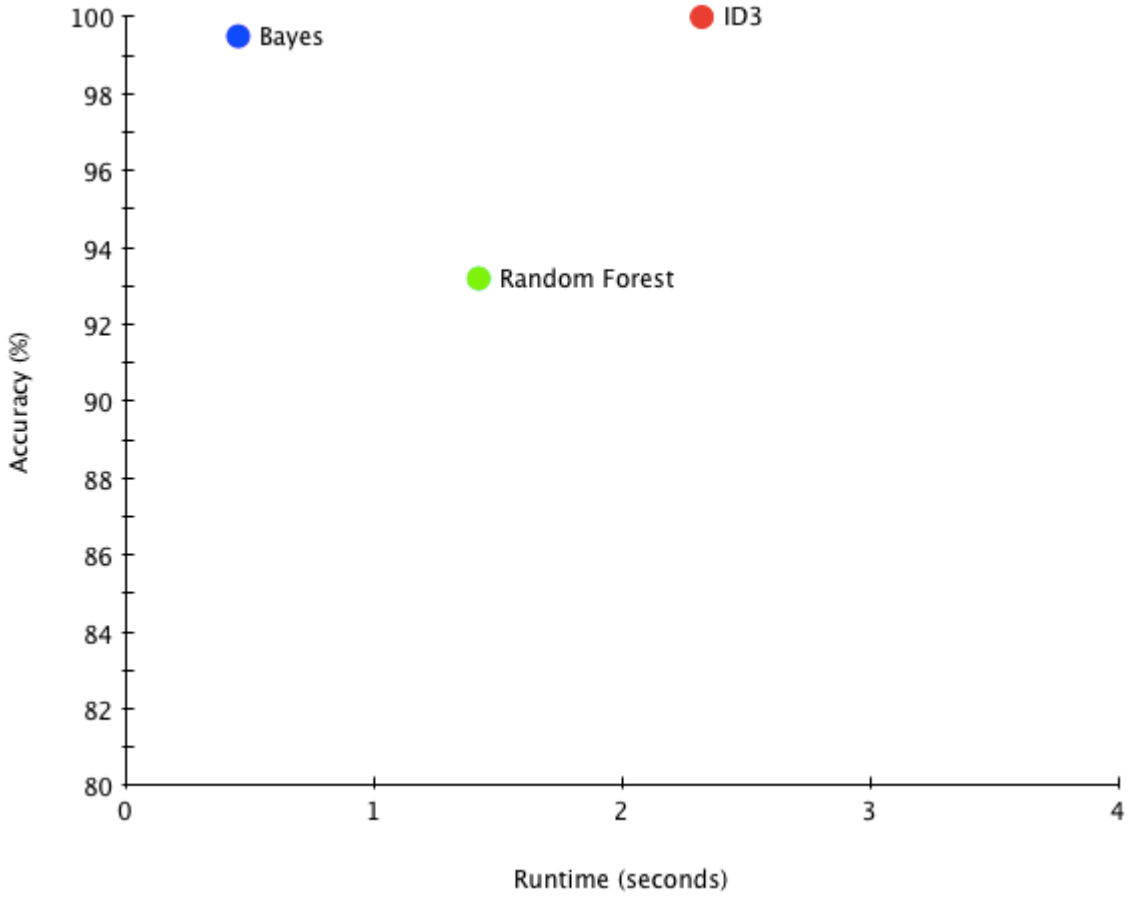
However, in our tests we found that overtraining caused the accuracy of the algorithm to decrease. We induced that this may be because of our lack of pruning in our trees which could have a significant effect on the validation tests.

Although the RF lacks in accuracy and variance, it has many advantages in real world data. The RF is able to handle outliers in dataset, unknown values, mixed attributes, and large datasets. In addition, a random forest is more likely to be suited for categorical data, which is what we were given in regards to each mushroom.

Our algorithm creates 5 trees. Each tree chooses attributes at random from the total set and creates an ID3 tree based on those attributes. This is where the variability has such a large affect on our outcome.

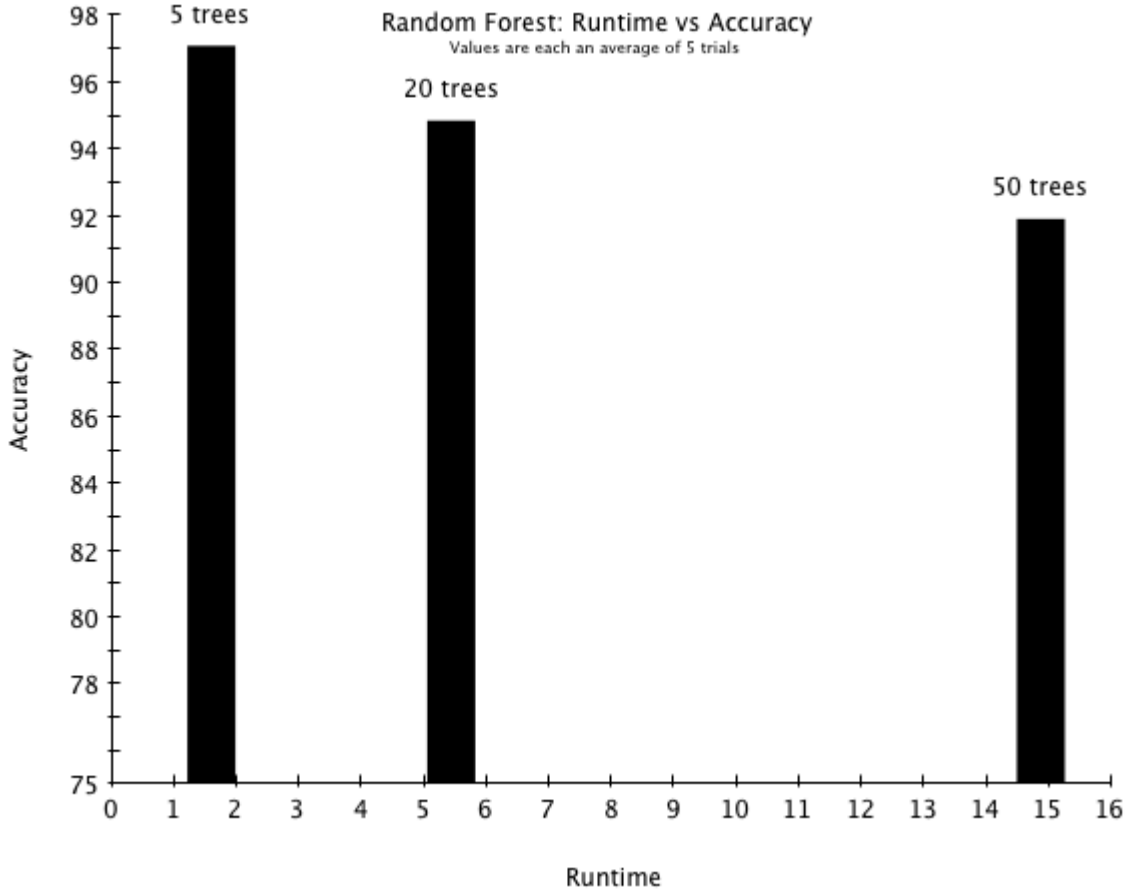
GRAPHS

Run Time vs. Accuracy:

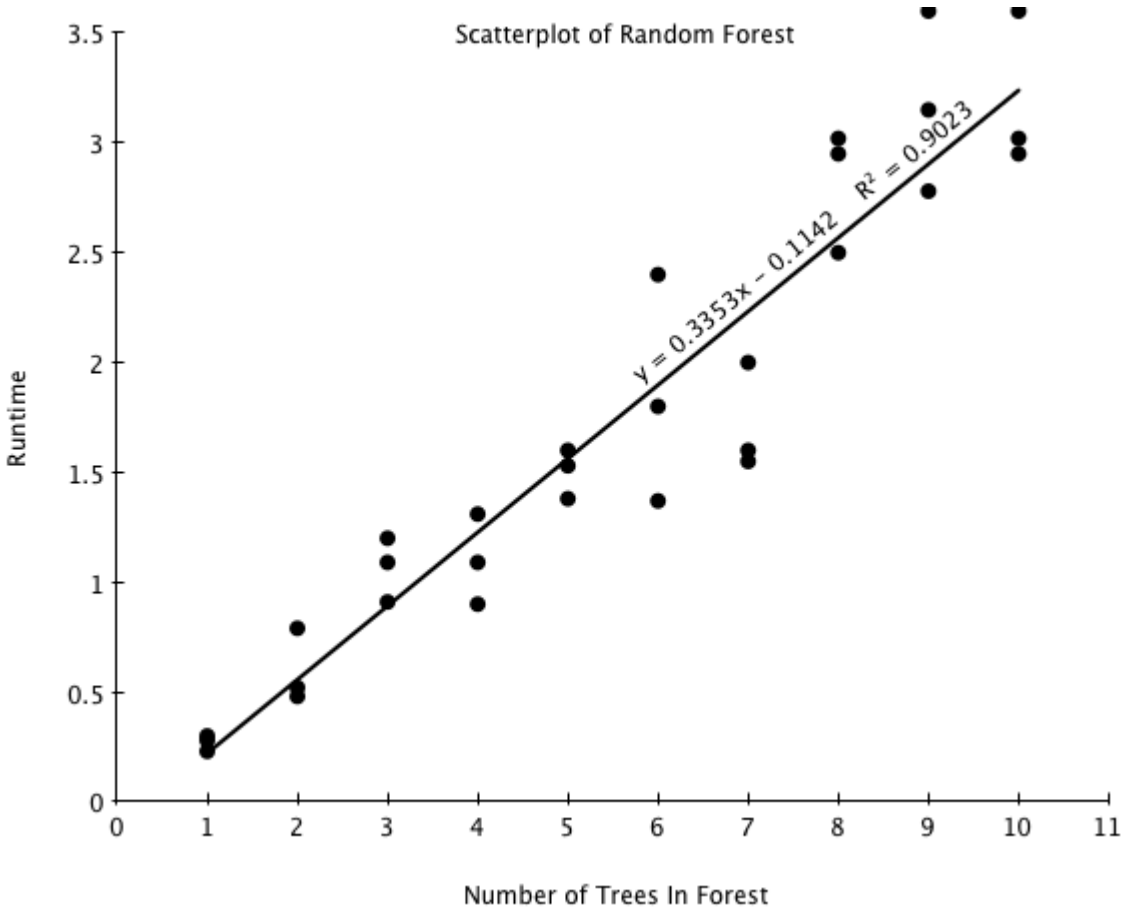


With the lookup table based implementation, Bayes is the fastest of the algorithms and also classifies the test data remarkably well, at over 99% correctness. The Random Forest technique for using our ID3 algorithm slightly decreases runtime for a smaller number of trees over pure ID3, but the cost in accuracy is definitely not worth it. ID3 is not far behind in runtime, but it runs with 100% accuracy, making it the preferred method. With an extremely large dataset, far larger than ours, ID3 may not experience the same accuracy and may end up with a much larger runtime, possibly making Bayes the preferred method.

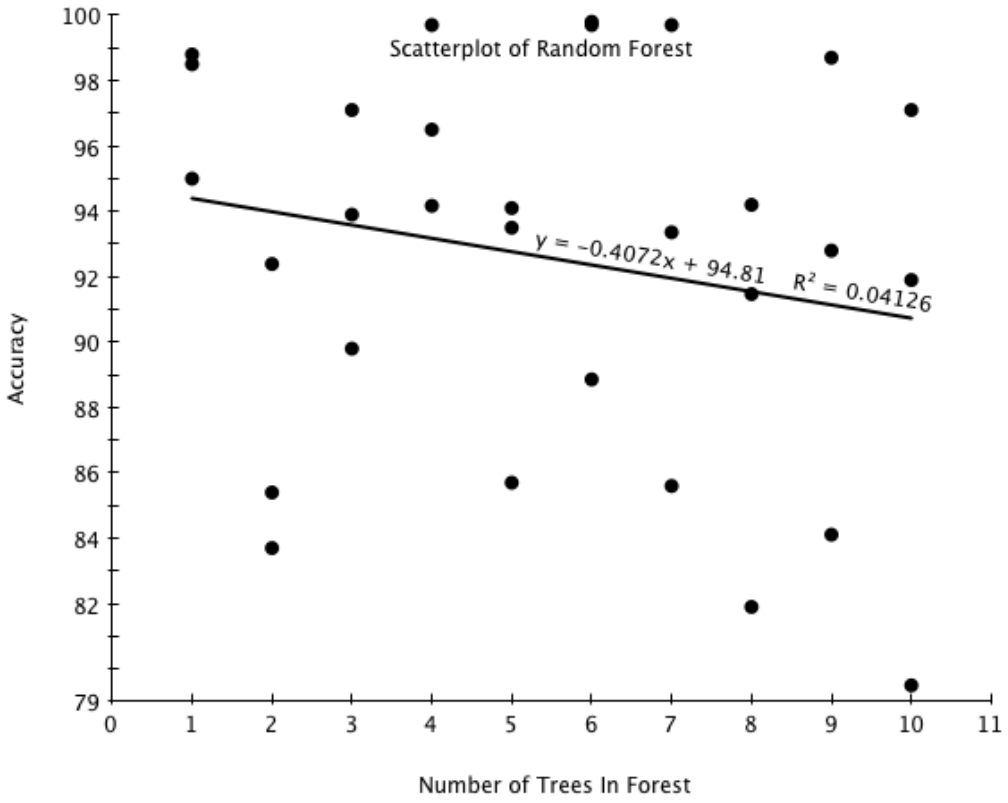
Runtime:



This graph is a comparison between the number of trees generated in a Random Forest. We can see that as you increase the number of trees; accuracy decreases and runtime increases. The conclusion we draw from here is that more is not necessarily better and that there is an optimal number of trees for which to classify our data.



The preceding scatter plot shows a linear increase in runtime of the Random Forest algorithm. However, given the small amount of data, we are unable to say whether the runtime increases exponentially or simply linearly. We do notice that it increases nonetheless. From here we may draw the conclusion that we want to use as few trees as possible to obtain both optimal results and classification time.



The graph above is the most interesting regarding the Random Forest. At first glance the data seems randomly placed, but we know from the first bar graph that on average, the accuracy will decrease. The current graph shows a slight decrease in accuracy as well. In addition, the scatter plot shows the extreme variance when randomly picking attributes to use in the tree. This will be examined later in analysis.

Sample ID3 Tree (First 6 levels only)

Words are attributes, letters are attribute values, e+ and p- are decisions (edible or poisonous)

```
population
    a
        e+
    c
        habitat
            g
                e+
            l
                spore-print-color
                    k
                    n
                    b
                    h
                    r
                    o
                    u
                    w
                    y
            m
                e+
            p
                e+
            u
                e+
            w
                e+
            d
                p-
    n
        e+
    s
        ring-type
            c
                e+
            e
                e+
            f
                e+
            l
                e+
            n
                e+
            p
                ring-number
                    n
                    o
                    t
            s
                e+
            z
                e+
    v
        habitat
            g
                p-
            l
                spore-print-color
                    k
                    n
                    b
                    h
                    r
                    o
                    u
                    w
                    y
            m
                p-
            p
                spore-print-color
                    k
                    n
                    b
                    h
```

r
o
u
w
y
u
ring-type
c
e
f
l
n
p
s
z
w
p-
d
ring-type
c
e
f
l
n
p
s
z
y
spore-print-color
k
e+
n
e+
b
e+
h
habitat
g
l
m
p
u
w
d
r
e+
o
e+
u
e+
w
e+
y
e+

ANALYSIS

Variance

The most interesting statistic which should be mentioned first is the variability between each algorithm. We found the following variances:

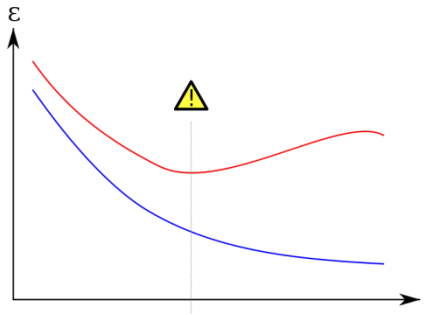
Random Forest (50 trees):
σ^2= 52.09 σ= 7.22

Random Forest (20 trees):
σ^2= 5.22 σ= 2.28

Random Forest (5 trees):
σ^2= 9.021 σ= 3

The most interesting thing regarding the variance/standard deviation of the random forest was that it seemed to follow the shape of a parabola. The variance was very high around 1 or 2 trees, but became less as you increased the number of trees. However, if you increase the number of trees to a very high number, the variance would increase greatly. We came to the conclusion that there will be an optimal number of trees (bigger is not necessarily better). This conclusion is due to overfitting of the data. Overfitting occurs because although the training data becomes more accurate as you increase the number of trees, the accuracy of test data begins decreasing once you reach the optimal number of trees.

Overfitting occurs when you have too many points of reference for the test data, thus creating large exaggerations in certain elements of the data, which may not follow the actual pattern. As you increase the number of trees to about 5 or 6, the training error and test error decrease, however once you reach 7 trees the training error continues to decrease, BUT the test (or validation) error INCREASES, like so:



Run Time vs. Accuracy

In analyzing the runtime vs. accuracy we must only look at Random Forest and Bayes because although our data set seems fairly large, it is comparatively small to many datasets typically used in the field, and therefore the ID3 algorithm will only classify with 100% accuracy EVERY run. However, we can compare the runtime of ID3 (≈2.3 seconds).

ID3 vs. Random Forest (selecting optimal attributes)

The interesting comparison between these two algorithms arises when we see the ID3 return a test data set with 100% accuracy. While continually producing results with about half the runtime of the ID3 algorithm, we would rather use a random forest to classify the data. However, given the large variability of the random algorithm it wasn't considered as the optimal classifier. As we looked into the attributes which were being used when the algorithm would produce results with 100% accuracy, we found they were attributes with better gain than the other attributes. We drew the hypothesis that using the 5 attributes with the highest gain would produce just as good of results as the ID3 algorithm, but in a shorter runtime. This idea could be further examined, however we did not follow it since it was only a difference of about 1 second between the runtimes.

Bayes (table) vs. Bayes (brute-force)

Initially the Naive Bayes brute force method we implemented was inefficient regardless of the reliability, as it had to count up positive and negative training examples for each run. However, after implementing the O(n) time complexity algorithm and regularly getting 99% accuracy, we can see that Bayes using a lookup table is a great way of analyzing HUGE sets of data. The brute-force methods takes too long for any real application.

Conclusion

In conclusion, for our particular dataset, pure ID3 seems to be the best algorithm to use, as it classifies fairly quickly with 100% accuracy. However, in a much larger dataset, it may be better to use the Random Forest technique to only split on a smaller set of attributes that maximize the gain on splitting, and repeating that over a forest of different trees. This may be faster in the long run with a large enough set of data. However, we noticed very high variability in the accuracy of Random Forest, and we do not have any conclusive data to suggest that it will have immense improvements in both accuracy and consistency with a larger set of data. The third option, the Naive Bayes classifier, performed incredibly well considering its simplicity and short runtime when implemented with a lookup table based approach. With that said, a very large dataset may be best classified using the Naive Bayes classifier, as one would likely experience very good accuracy and a comparatively short runtime.