Classifying Mushrooms as Edible or Poisonous

Arya Bharath and James Wright

10/21/24

**Table of Contents**

1. Statement & Goal
2. Description of Dataset
3. Data Mining Procedure/Preprocessing
4. Data Mining Tools & Algorithms/Attribute Selection
5. Data Mining Results & Evaluation
6. Discussion & Conclusion
7. Team Contributions
8. Appendix & Sources

**Statement & Goal**

The Mushrooms Dataset takes values of 20 unique attributes in order to predict whether or not any given mushroom will be poisonous (p) or edible (e). Our goal for this project is to build effective models using this dataset that will accurately predict the class variable.

We plan to use this dataset to classify mushrooms as either poisonous (p) or edible (e) based on the 20 outlined attributes. In the real world, this has several implications. If someone sees or interacts with a mushroom that they suspect may be poisonous, for example, how will they know? It would be best to know as soon as possible if they will have symptoms from touching or interacting with something poisonous, but most people will not be able to identify whether something is definitively poisonous or not based on looking at it alone. But if this is the case, what is the optimal method for classifying? It would be possible to create a very simple set of rules that humans can understand and use without following more advanced algorithms, like stating that very colorful mushrooms are poisonous, but this would lose a lot of precision that would result in worse results over time. As a result, it would be useful to use a machine learning model to classify instead.

**Description of Dataset**

**Link to Dataset:**

<https://archive.ics.uci.edu/dataset/848/secondary+mushroom+dataset>

**Data Information, Dimensionality, & Instances**

There were originally 4 datasets downloaded from the ML Repository at UCI. The downloads included two .txt files which shared information about each of the attributes and what type of data they were, as well as two .csv files which held all of the instances and values. We downloaded both primary and secondary mushroom datasets, but we chose to only work with one file for the rest of our project. The file that we used was:

secondary\_data.csv

We chose to use this over the primary dataset as it contained more missing data as well as better attributes.

Our dataset contains 61,069 instances of mushrooms, with each mushroom being given a label of poisonous or edible. The dimensionality of this dataset is 20, with each attribute representing key features of the mushroom’s physical appearance and anatomy.

**Attributes**

Before preprocessing, our dataset has 20 attributes, expressing both qualitative and quantitative data. In combination, these attributes are used in order to predict whether a certain mushroom is poisonous (p) or edible (e). (**class**: qualitative nominal, given as poisonous (p) or edible (e))

1. **cap-diameter**: quantitative continuous, given as a float in centimeters (cm)
2. **cap-shape**: qualitative nominal, given as bell (b), conical (c), convex (x), flat (f), sunken (s), spherical (p), or others (o)
3. **cap-surface**: qualitative nominal, given as fibrous (i), grooves (g), scaly (y), smooth (s), shiny (h), leathery (l), silky (k), sticky (t), wrinkled (w), or fleshy (e)
4. **cap-color**: qualitative nominal, given as brown (n), buff (b), gray (g), green (r), pink (p), purple (u), red (e), white (w), yellow (y), blue (l), orange (o), or black (k)
5. **does-bruise-bleed**: qualitative nominal, given as bruises-or-bleeding (t) or no (f)
6. **gill-attachment**: qualitative nominal, given as adnate (a), adnexed (x), decurrent (d), free (e), sinuate (s), pores (p), none (f), or unknown (?)
7. **gill-spacing**: quantitative continuous, given as a float in centimeters (cm)
8. **gill-color**: qualitative nominal, given as brown (n), buff (b), gray (g), green (r), pink (p), purple (u), red (e), white (w), yellow (y), blue (l), orange (o), black (k), or none (f)
9. **stem-height**: quantitative continuous, given as a float in centimeters (cm)
10. **stem-width**: quantitative continuous, given as a float in centimeters (cm)
11. **stem-root**: qualitative nominal, given as bulbous (b), swollen (s), club (c), cup (u), equal (e), rhizomorphs (z), or rooted (r)
12. **stem-surface**: qualitative nominal, given as fibrous (i), grooves (g), scaly (y), smooth (s), shiny (h), leathery (l), silky (k), sticky (t), wrinkled (w), fleshy (e), or none (f)
13. **stem-color**: qualitative nominal, given as brown (n), buff (b), gray (g), green (r), pink (p), purple (u), red (e), white (w), yellow (y), blue (l), orange (o), black (k), or none (f)
14. **veil-type**: qualitative nominal, given as partial (p) or universal (u)
15. **veil-color**: qualitative nominal, given as brown (n), buff (b), gray (g), green (r), pink (p), purple (u), red (e), white (w), yellow (y), blue (l), orange (o), black (k), or none (f)
16. **has-ring**: qualitative nominal, given as ring (t) or none (f)
17. **ring-type**: qualitative nominal, given as cobwebby (c), evanescent (e), flaring (r), grooved (g), large (l), pendant (p), sheathing (s), zone (z), scaly (y), movable (m), none (f), or unknown (?)
18. **spore-print-color**: qualitative nominal, given as brown (n), buff (b), gray (g), green (r), pink (p), purple (u), red (e), white (w), yellow (y), blue (l), orange (o), or black (k)
19. **habitat**: qualitative nominal, given as grasses (g), leaves (l), meadows (m), paths (p), heaths (h), urban (u), waste (w), or woods (d)
20. **season**: qualitative nominal, given as spring (s), summer (u), autumn (a), or winter (w)

**Missing Data**

After running the data set through a Python script that counted the number of missing values, we determined that our data is missing 307,463 values total. While this may seem like a lot, it is important to note that this is spread out over 61,069 instances, meaning that each instance will be missing, on average, 5 values. Because we have 20 attributes, this should not be too detrimental to our models, particularly after we preprocess our data to deal with missing values. It is also interesting to note that because we have so many instances, it may be possible to simply select values with only a small number missing values to solve the data problem.

**Class Distribution**

In the overall dataset, there are 33,888 instances that have been classified as poisonous (p) and 27,181 classified as edible (e). As a result, because these values are not equal, our dataset is not balanced, with 55.5% of our instances being deemed poisonous and approximately 44.5% being edible.

**Uniform or Skewed**

In terms of whether our data is uniform or skewed, it is hard to tell because the class is qualitative and nominal, meaning that there is no order. This would mean that I could arrange the data for the poisonous and edible mushrooms in any order, which would not create a uniform or skewed distribution.

**Data Mining Procedure/Preprocessing**

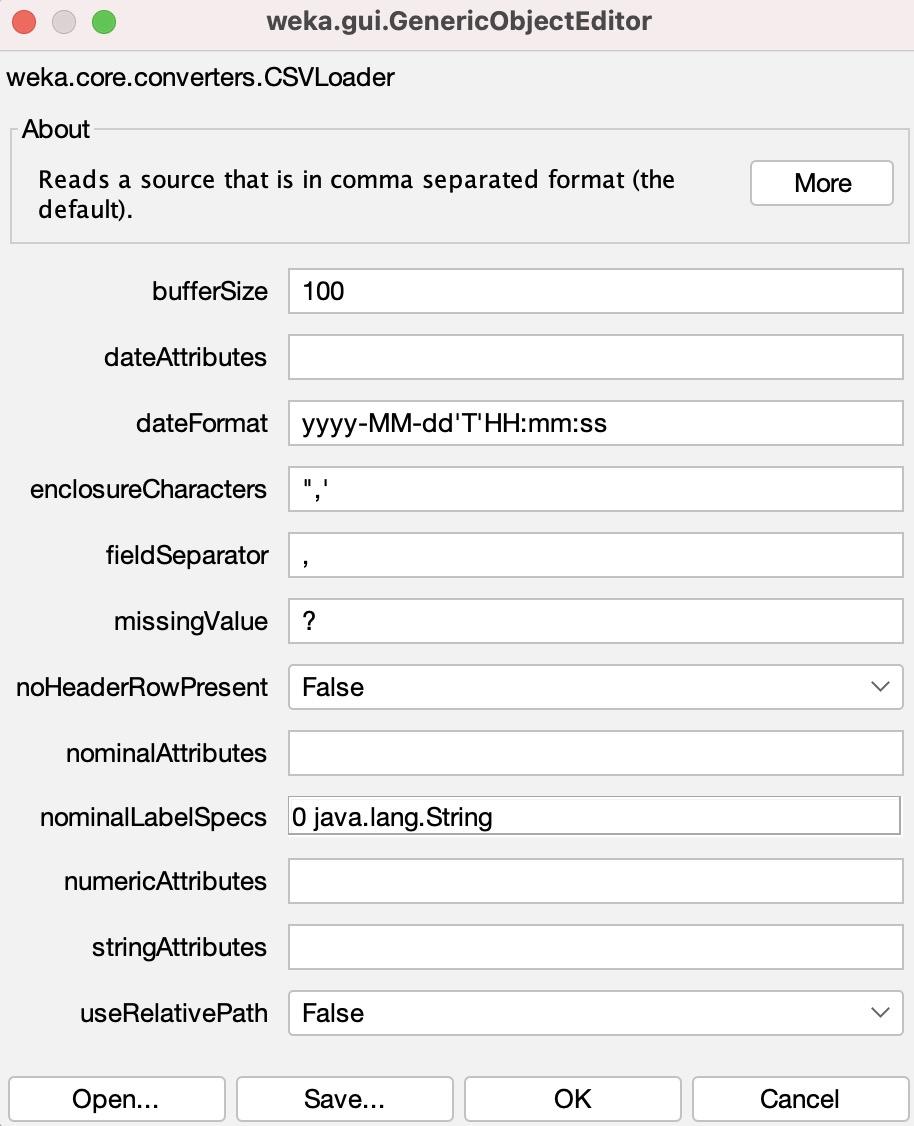
**Preprocessing**

So far, most of our work has gone into preprocessing our data. The first thing that we did was solve the problem of opening our data set in WEKA. Originally, our data looked like this:

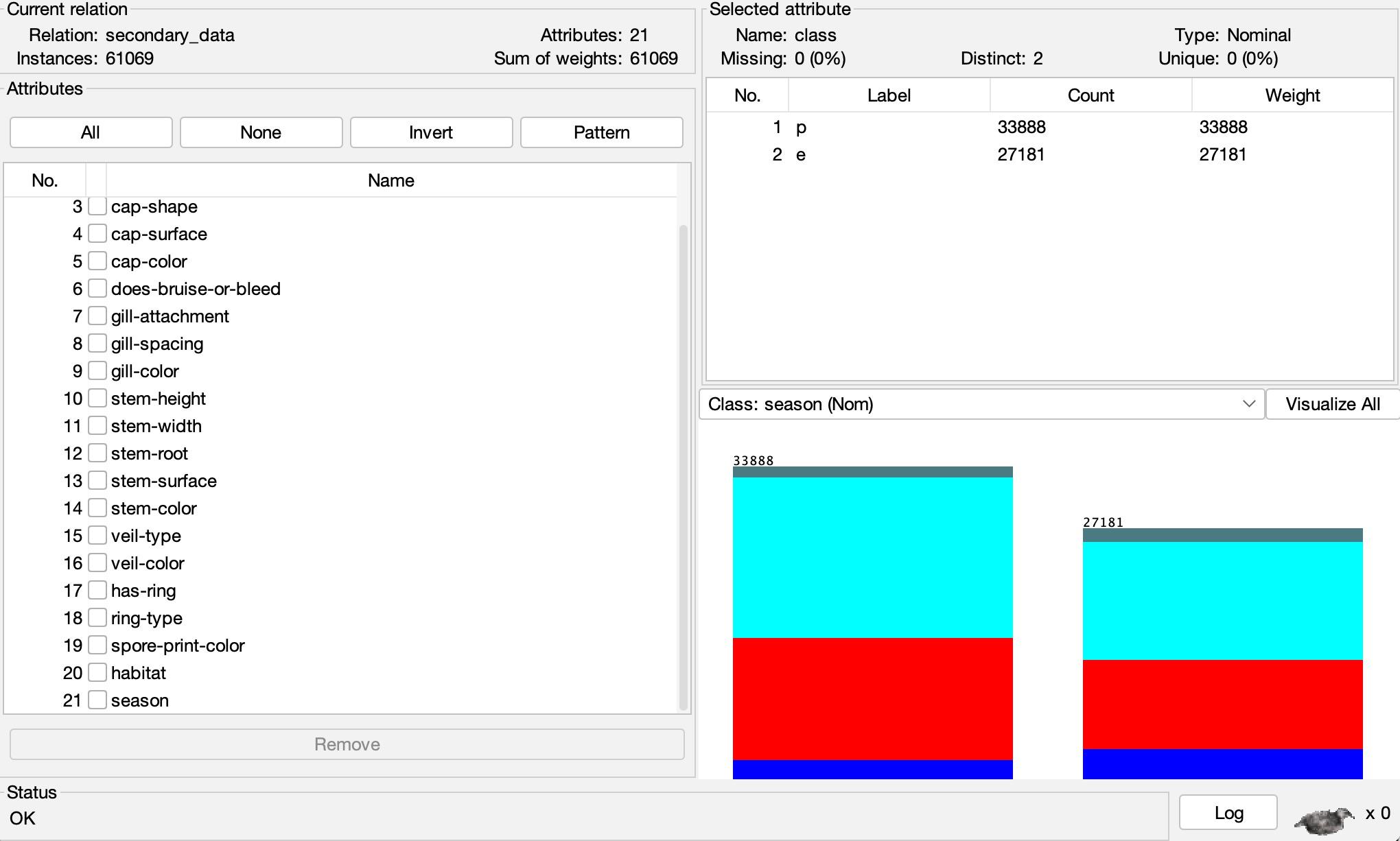
****

In the above screenshot, both the classes and instances appeared not to split apart. Even though there were approximately 60,000 instances in our data (as expected), each instance was given as a string of all of the values of the attributes combined into one. Similarly, as we can see in the middle of the screenshot, the attributes and class did not split up. In order to fix this problem, we identified that the issue was the formatting of the .csv file. Even though we expected commas to separate the values as is true of files of the type .csv, there were semicolons.

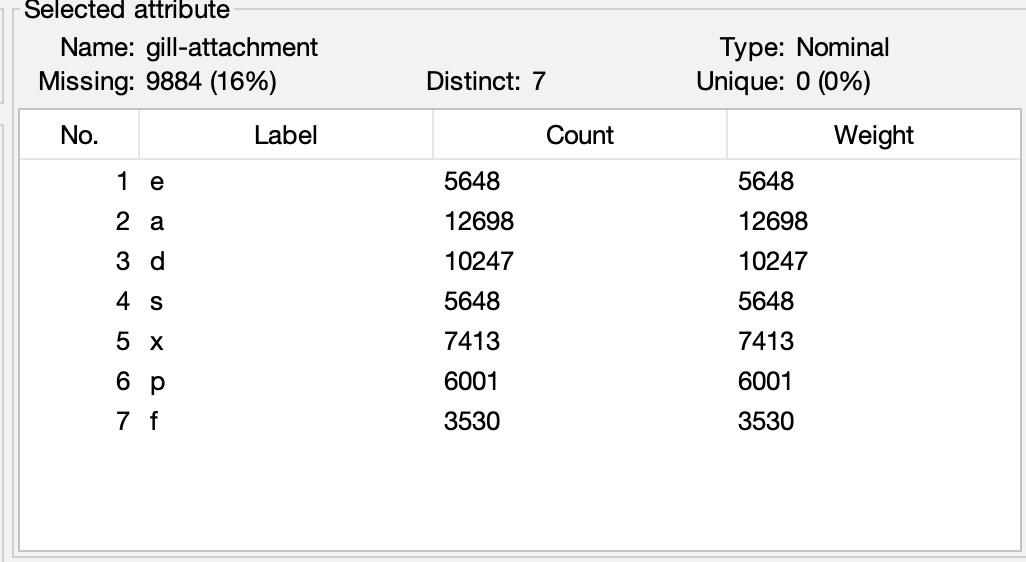
Our solution, as we posted in the Q1 Project Help document, was to edit WEKA’s GenericObjectEditor menu to change the *fieldSeparator* from a comma to a semicolon.



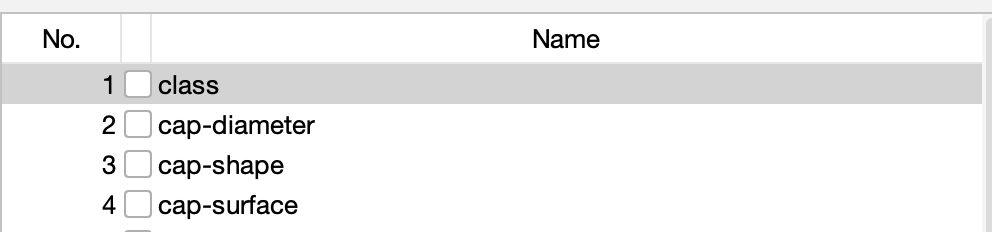
This menu gives us a few pieces of information. On top of changing the *fieldSeparator* as previously mentioned, we also used the *missingValue* field to change our missing values to question marks.

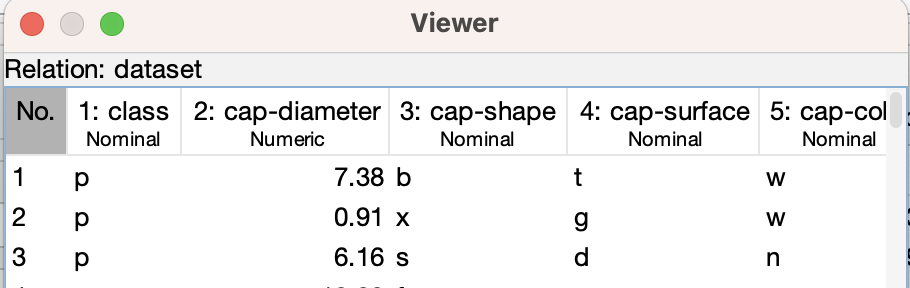


After opening our data, it now looks like this, which is much better. Additionally, as we can see from the screenshot below, WEKA has missing values identified when attributes are selected.

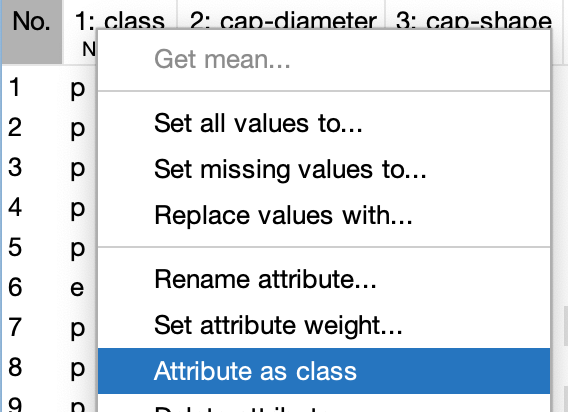


However, in order to deal with missing values, we are trying two different approaches. Because of the nature of the missing values in our data, it is unclear if it’s a good idea to use mean/mode to fill missing values, just because of the lack of information in certain columns. As a result, we are going to try 1) setting missing values as “?” unknown values and 2) setting them as mode/mean in WEKA. This means that we will have 30 datasets total, as there are 5 attribute selections and 3 sets per attribute selection (train, validation, test), and 2 approaches.

When opening our training dataset in WEKA, the class label appeared first, as an attribute. 



To make the class an actual class in WEKA instead of just an attribute, we opened the Viewer using Edit and right-clicked on the class, which then prompted us with the option to set the attribute as the class.



After clicking that, we then clicked “OK” on the Viewer to set that class attribute as the real class in WEKA. We also considered using normalization in our data. However, all of the quantitative values were constrained to the same ranges of numbers, so we decided that it was not necessary.

To split our dataset into training, validation, and testing sets, we used a Python script that employed pandas, sklearn, scipy.io, and the arff libraries:

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from scipy.io import arff

import arff as liac\_arff

data, meta = arff.loadarff('data.arff')

df = pd.DataFrame(data)

def decode\_bytes(df):

for col in df.select\_dtypes([object]):

df[col] = df[col].apply(lambda x: x.decode('utf-8') if isinstance(x, bytes) else x)

return df

df = decode\_bytes(df)

X = df.iloc[:, :-1]

y = df.iloc[:, -1]

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.3, stratify=y, random\_state=42)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, stratify=y\_temp, random\_state=42)

train\_set = pd.concat([y\_train, X\_train], axis=1)

val\_set = pd.concat([y\_val, X\_val], axis=1)

test\_set = pd.concat([y\_test, X\_test], axis=1)

def save\_to\_arff(df, filename, meta):

arff\_data = {

'description': '',

'relation': 'dataset',

'attributes': [(name, list(df[name].unique())) if df[name].dtype == 'object' else (name, 'REAL') for name in df.columns],

'data': df.values.tolist()

}

with open(filename, 'w') as f:

liac\_arff.dump(arff\_data, f)

save\_to\_arff(train\_set, 'train\_set.arff', meta)

save\_to\_arff(val\_set, 'val\_set.arff', meta)

save\_to\_arff(test\_set, 'test\_set.arff', meta)

When using the train\_test\_split method from sklearn.model\_selection, we used the stratify parameter to ensure that the training, validation, and testing datasets would all maintain the same proportions between class labels. For our data, the ratio between p (poisonous) to e (edible) instances was around 1.25, so all of the split datasets would also need to have this ratio. We used a 70:15:15 split between the training, validation, and testing datasets when splitting each dataset created from attribute selection. Because we created 10 different files from attribute selection (5 from every different evaluation and another 5 from filling in all missing values), 30 total files were created of training, validation, and testing datasets. The training, validation, and testing datasets always had 42672 instances, 9174 instances, and 9175 instances respectively.

**Data Mining Tools & Algorithms/Attribute Selection**

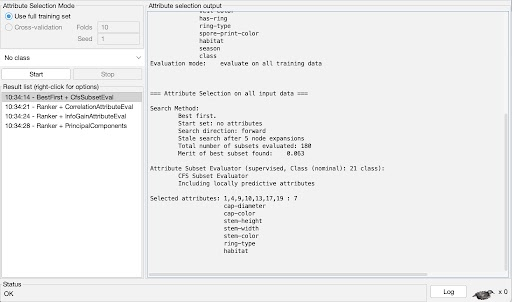
**Attribute Selection**

For this project, we needed to choose 5 different approaches to do feature selection. Ultimately, we decided on CfsSubsetEval, CorrelationAttributeEval, InfoGainAttributeEval, OneRAttributeEval, and our own intuitive selection. As mentioned previously, we want to compare several different ways of preprocessing missing values in order to see which one yields the best performances, so we ran each attribute selection algorithm for a dataset with each preprocessing method. In the future, we may also use Principal Component Analysis to reduce our dimensionality as well.

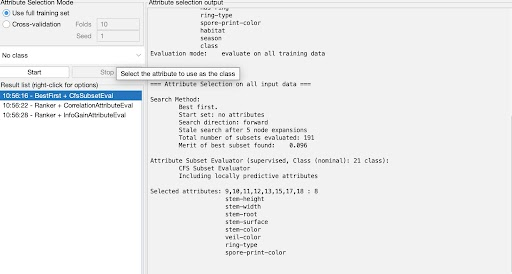
1. **CfsSubsetEval**

CfsSubsetEval uses a combination of the individual predicting power of each attribute and the redundancy of the predictions to select attributes.

For our datasets with missing values as “?”:



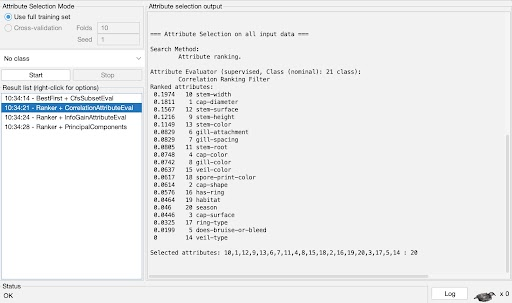
For our datasets with missing values as mean/mode:



1. **CorrelationAttributeEval**

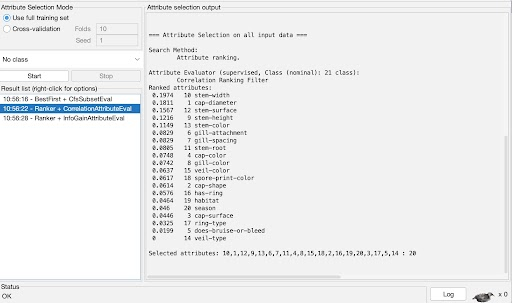
Orders attributes by using Pearson Correlation between each individual attribute and the class.

For our datasets with missing values as “?”:



Cut-off value of 0.08 chosen - all attributes scoring below were discarded.

For our datasets with missing values as mean/mode:

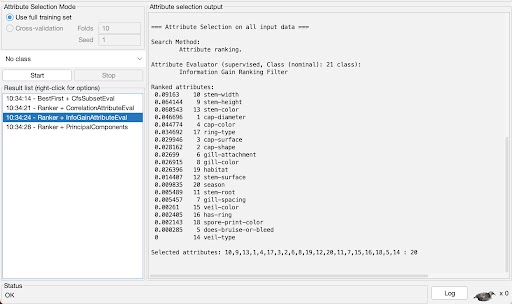


Cut-off value of 0.08 chosen.

1. **InfoGainAttributeEval**

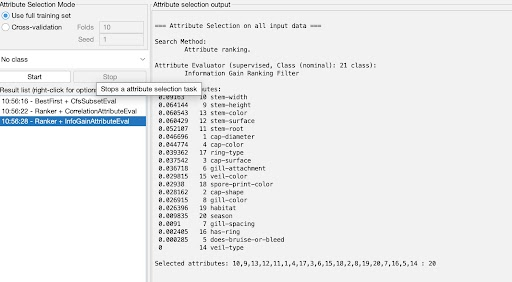
Sorts attributes based on information gain/entropy loss.

For our datasets with missing values as “?”:



Cut-off value of 0.025 chosen.

For our datasets with missing values as mean/mode:

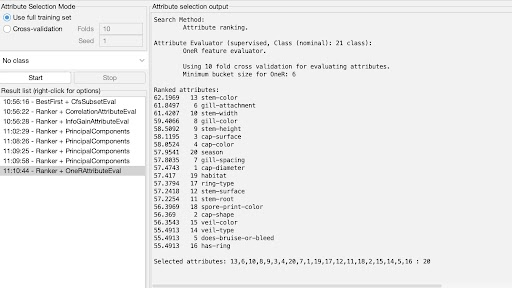


Cut-off value of 0.04 chosen.

1. **OneRAttributeEval**

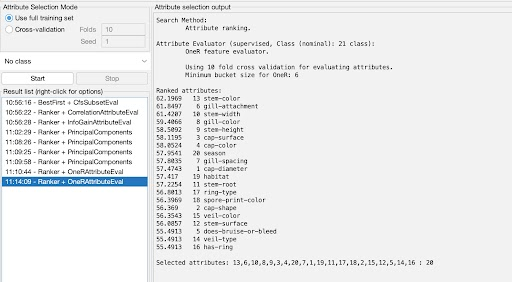
Chooses attributes with OneR Classifier.

For our datasets with missing values as “?”:



Cut-off of 57.5 chosen.

For our datasets with missing values as mean/mode:



Cut-off value of 57.5 chosen.

1. **Intuitive Selection**

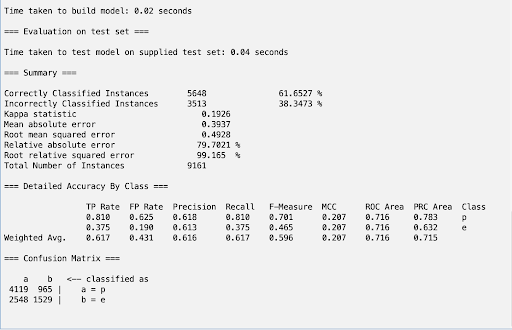
For both datasets, we selected 11 attributes:

1. Cap-shape
2. Cap-surface
3. Cap-color
4. Gill-color
5. Stem-surface
6. Stem-color
7. Veil-type
8. Veil-color
9. Has-ring
10. Ring-type
11. Spore-print-color

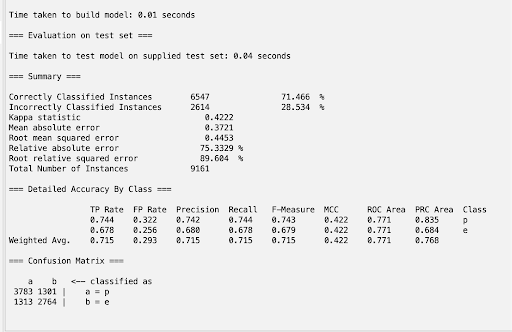
We chose these because poisonous plants and animals are usually identifiable by their shape and color, not necessarily their size. We thus removed the attributes that had to do with the measurements of the mushrooms, as well as some others like season and habitat.

**Data Mining Results & Evaluation**

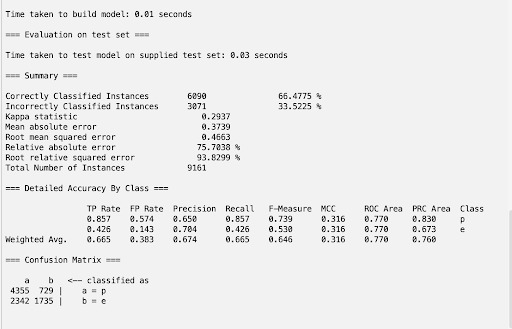
cfs\_test.arff, naivebayes



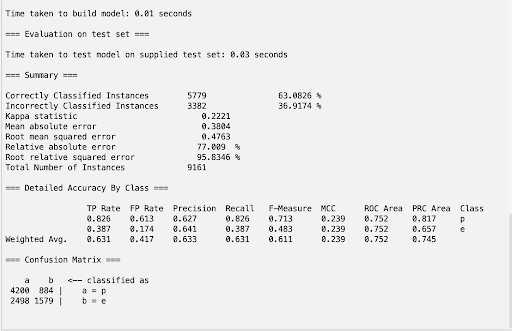
cfssubset\_no\_miss\_test\_set.arff, naivebayes



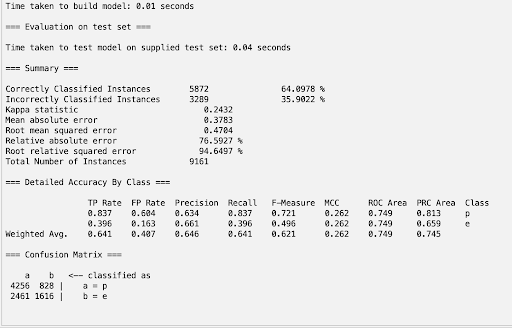
correlation\_no\_miss\_test\_set.arff, naivebayes



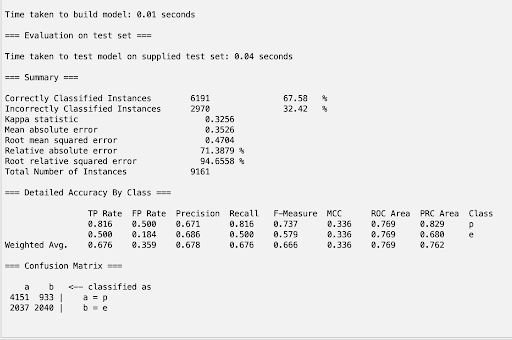
correlation\_test\_set.arff, naivebayes



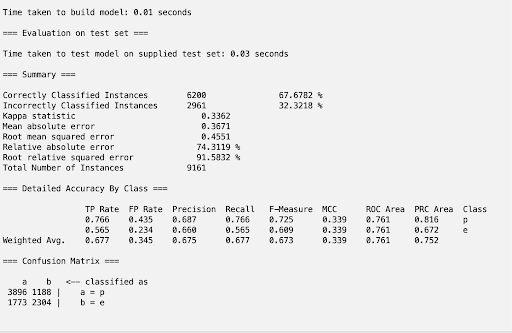
infogain\_no\_miss.arff, naivebayes



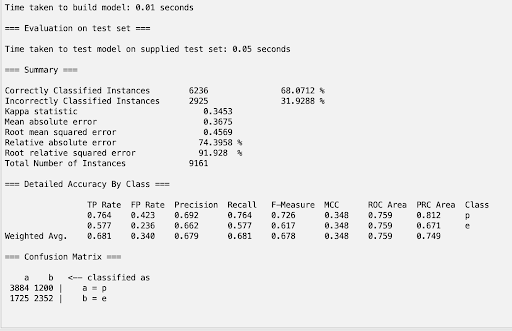
infogain\_test\_set.arff, naivebayes



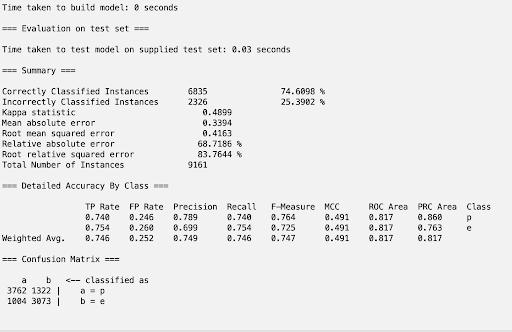
oner\_no\_miss\_test\_set.arff, naivebayes



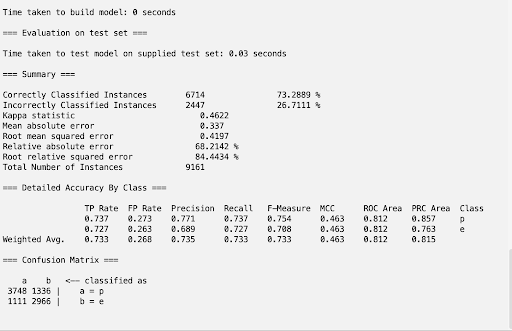
oner\_test\_set.arff, naivebayes



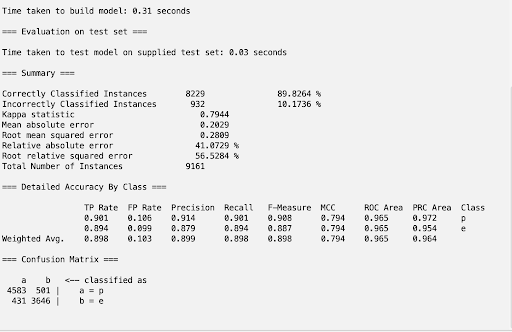
selected\_no\_miss\_test\_set.arff, naivebayes



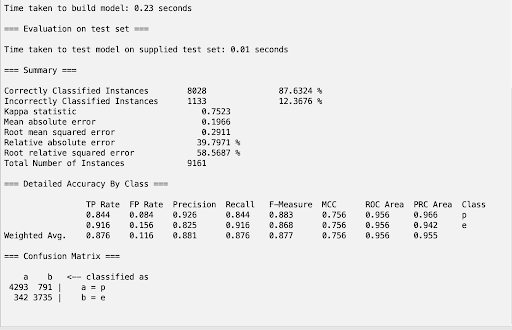
selected\_test\_set.arff, naivebayes



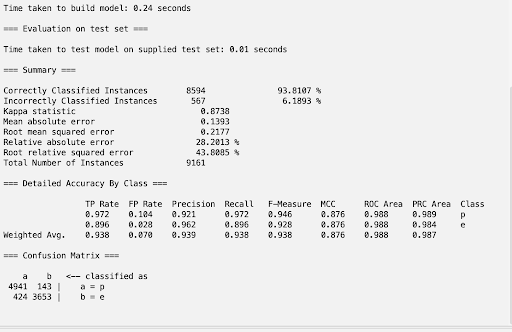
cfs\_test\_set.arff, DecisionTable



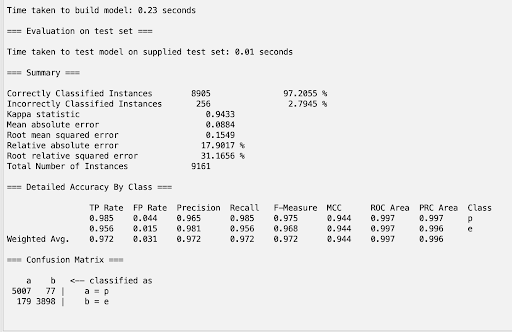
cfssubset\_no\_miss\_test\_set.arff, DecisionTable



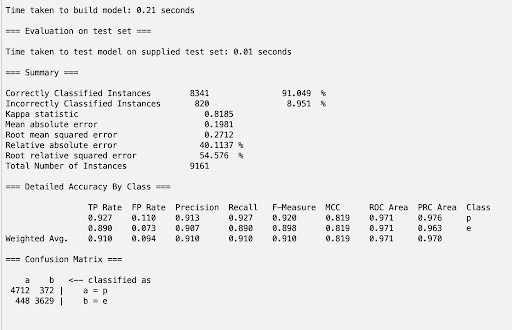
correlation\_no\_miss\_test\_set.arff, DecisionTable



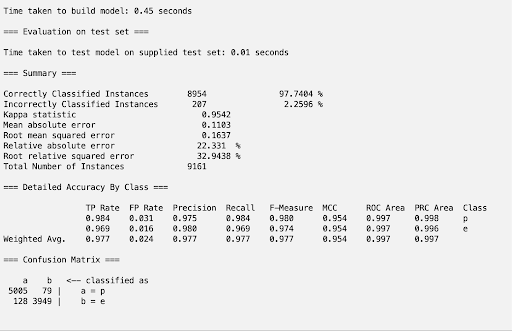
correlation\_test\_set.arff, DecisionTable



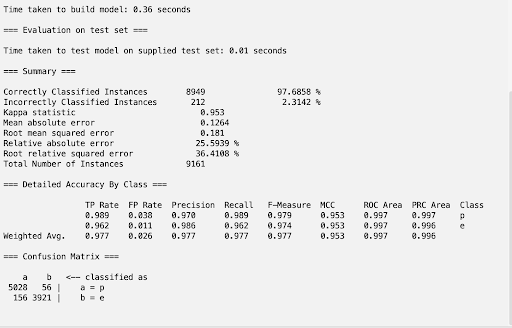
infogain\_no\_miss\_test\_set.arff, DecisionTable



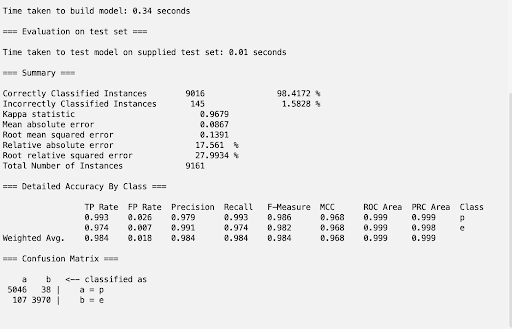
infogain\_test\_set.arff, DecisionTable



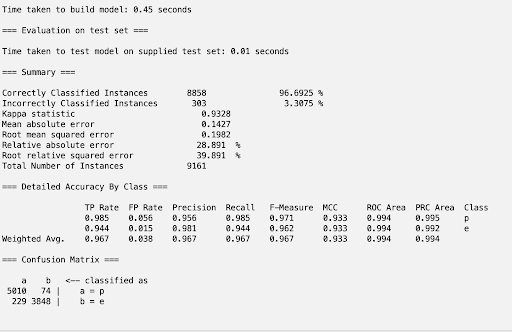
oner\_no\_miss\_test\_set.arff, DecisionTable



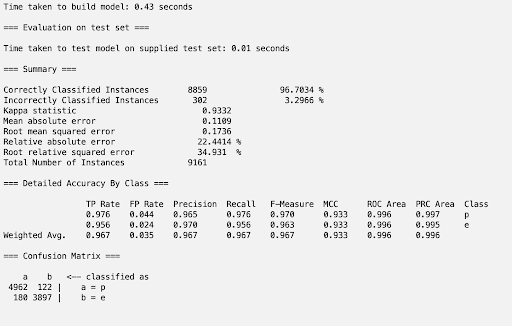
oner\_test\_set.arff, DecisionTable



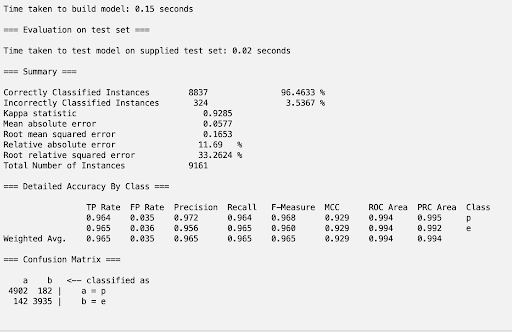
selected\_no\_miss\_test\_set.arff, DecisionTable



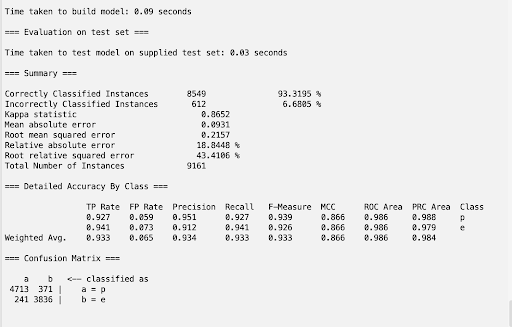
selected\_test\_set.arff, DecisionTable



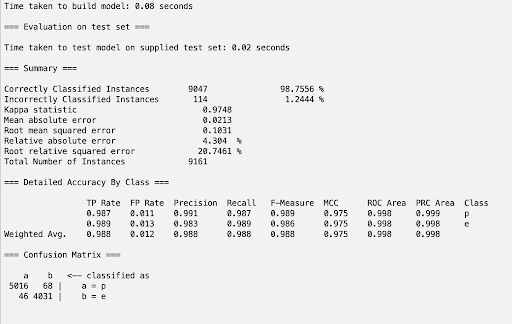
cfs\_test\_set.arff, J48



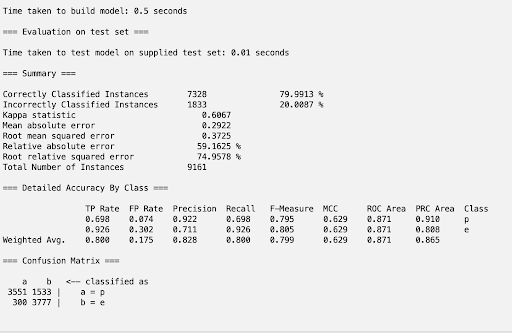
cfssubset\_no\_miss\_test\_set.arff, J48



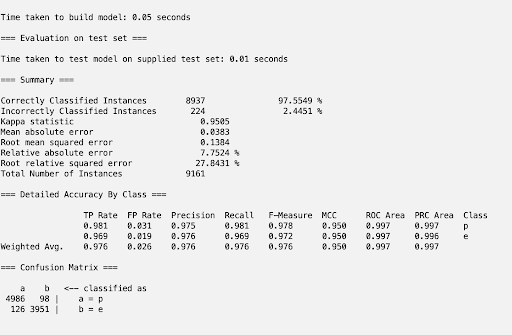
correlation\_no\_miss\_test\_set.arff, J48



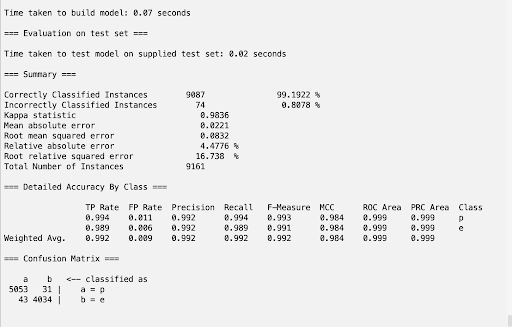
correlation\_test\_set.arff, J48



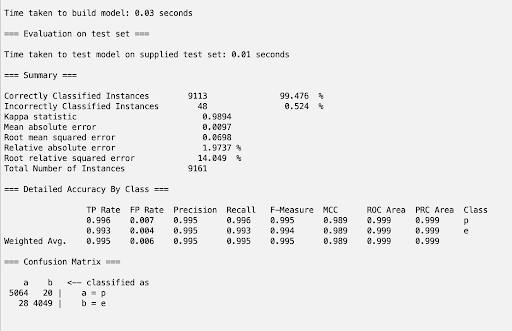
infogain\_no\_miss\_test\_set.arff, J48



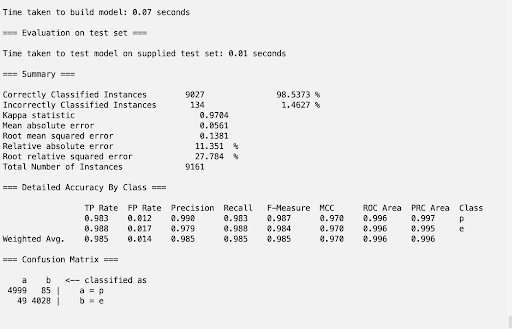
infogain\_test\_set.arff, J48



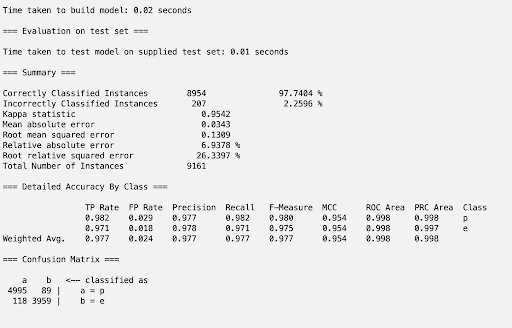
oner\_no\_miss\_test\_set.arff, J48



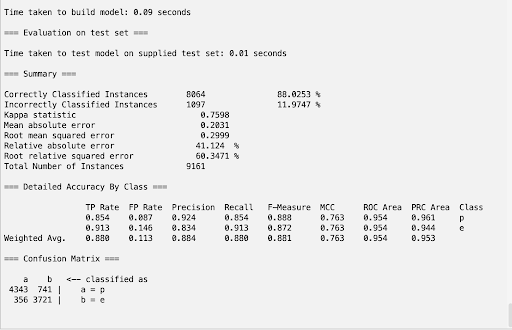
oner\_test\_set.arff, J48



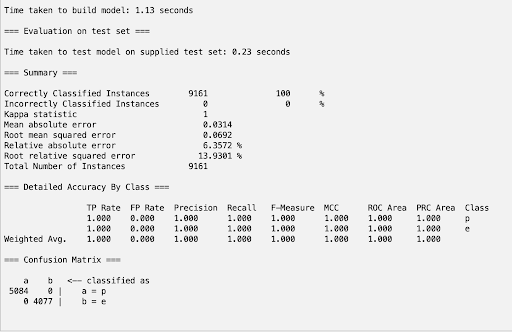
selected\_no\_miss\_test\_set.arff, J48



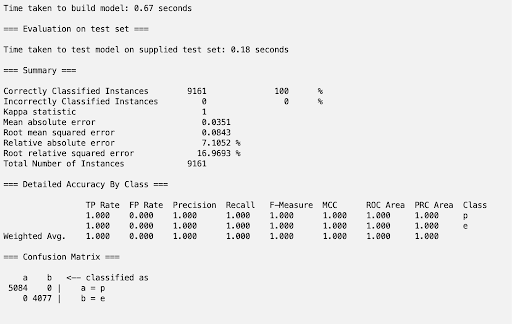
selected\_test\_set.arff, J48



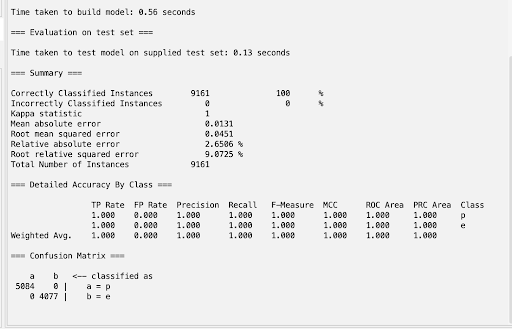
cfs\_test\_set.arff, RandomForest



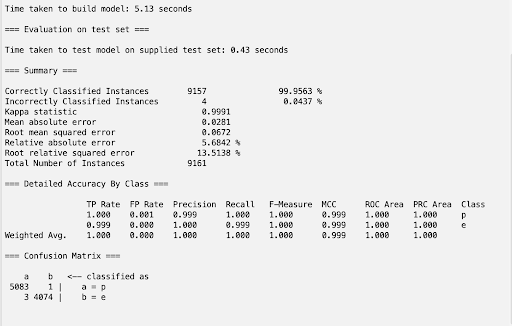
cfssubset\_no\_miss\_test\_set.arff, RandomForest



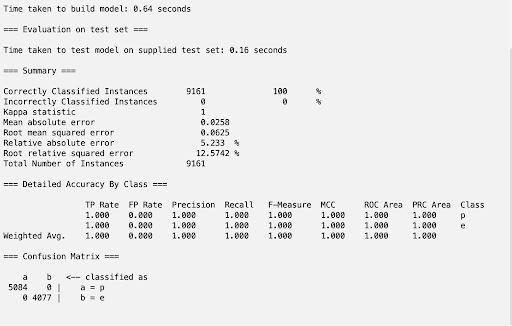
correlation\_no\_miss\_test\_set.arff, RandomForest



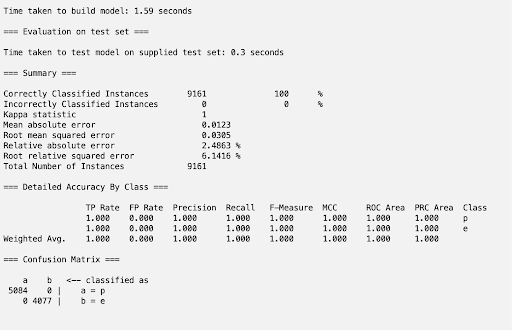
correlation\_test\_set.arff, RandomForest



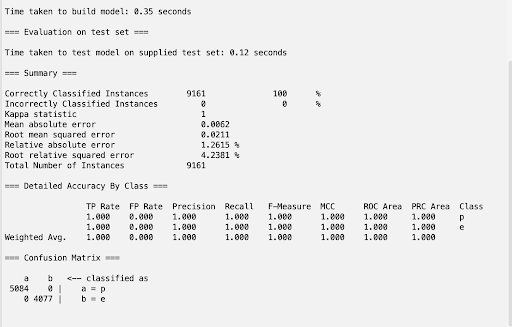
infogain\_no\_miss\_test\_set.arff, RandomForest



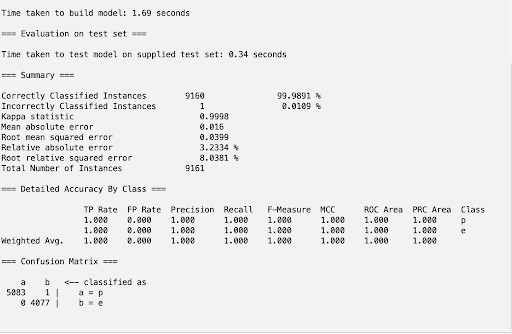
infogain\_test\_set.arff, RandomForest



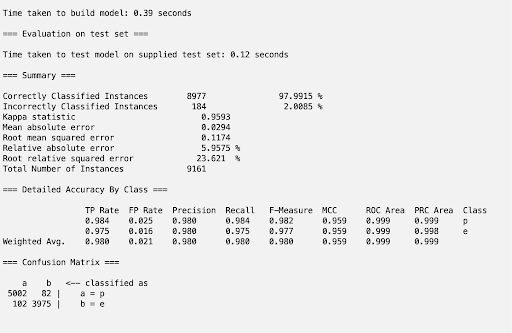
oner\_no\_miss\_test\_set.arff, RandomForest



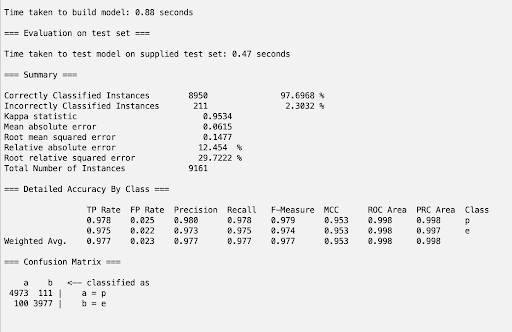
oner\_test\_set.arff, RandomForest



selected\_no\_miss\_test\_set.arff, RandomForest



selected\_test\_set.arff, RandomForest



**Analysis**

In total, we have 40 classifier models. Because we dealt with missing values two different ways, used 4 distinct classifiers - DecisionTable, NaiveBayes, J48, and RandomForest, and used 5 attribute selection algorithms (as outlined earlier in the report), we have ran a total of 40 models to classify the data. Among these:

cfs\_test\_set.arff, RandomForest

cfssubset\_no\_miss\_test\_set.arff, RandomForest

correlation\_no\_miss\_test\_set.arff, RandomForest

infogain\_no\_miss\_test\_set.arff, RandomForest

**oner\_no\_miss\_test\_set.arff, RandomForest**

All achieved perfect accuracies of 100%, as scored by WEKA. In order to select the best model among these 5, we must inspect their respective error loss metrics to determine which will perform best on real-life data sets. After further inspection, the RandomForest classifier with the OneR attribute selection algorithm and missing values filled with mean/mode yields a 0.0211 root-mean-square error, the lowest of the 5 aforementioned datasets. This means that even though our test set was large and thus more likely to be representative of real world samples if the data was taken correctly, it is even more likely that this model performs well.

**Discussion & Conclusion**

In this project, we learned how to use WEKA to successfully train and test a model that classified our own selected data of poisonous vs edible mushrooms with 100% accuracy. Our dataset initially started off with 20 attributes, and our model could have possibly been improved had we not removed some of the attributes during our attribute selection step. Many of the attributes that were removed could have held solid predictive power of whether or not a certain mushroom is edible or poisonous. In the future, better attribute selection could be a key improvement from our project.

**Steps to Reproduce Our Model: RandomForest model with OneR Selection:**

1. Open WEKA, click on Explorer, and load the secondary\_data.csv file after clicking the Open file button.
2. Click on the Select Attributes button in the top, and under Attribute Evaluator, choose OneRAttributeEval and Ranker as the search method. Click start.
3. Keep note of the attributes which have a score less than 57.5, and remove them by returning to the Preprocess tab, checking their box, and clicking remove.
4. Save this dataset as a new file called oner\_data.arff and input it into the split.py python script, returning 3 new files: oner\_train\_set.arff, oner\_val\_set.arff, and oner\_test\_set.arff.
5. Return to WEKA and click on the Classify tab on the top. Click Choose > trees > RandomForest as the classifier.
6. Click on the Supplied test set button and load in oner\_test\_set.arff.
7. Select the class on the left ( (Nom) class).
8. Click start to test the model. Model has the name oner\_model.model.

**Team Contributions**

**Finding the Data & Building Proposal:** Arya Bharath & James Wright

**Preprocessing**: Arya Bharath & James Wright

**Intermediate Report**: Arya Bharath & James Wright

**Non-Weka Attribute Selection Algorithm**: Arya Bharath & James Wright

**Attribute Selection**: James Wright

**Classifiers**: Arya Bharath

**Results Analysis**: Arya Bharath & James Wright

**Final Report**: Arya Bharath & James Wright

**Appendix & Sources**

**Link to Dataset:**

<https://archive.ics.uci.edu/dataset/848/secondary+mushroom+dataset>

**Attached Files:**

* split.py - used to split each of our datasets into train, val, test sets.
* secondary\_data.csv - original dataset.
* all .arff files - created from attribute selection and split, used to train and test model.
* oner\_model.model - chosen model for this project.