Assignment 2 Writeup

Andrew J. Otis

Dataset Overview/Description:

Dataset: Mushroom Dataset
Source: https://archive.ics.uci.edu/ml/datasets/mushroom
Reference: The Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf

The data contains **8,124 observations** for a total of **22 attributes**. Below is a table describing each column of the dataset. The variable names were added during pre-processing since the data has no column names to begin with and no column names were given to features that were going to be dropped during the analysis and model building.

Variable	Variable Name	Variable Data Type	Variable Classes
p	mushroom classification	classification	edible=e poisonous=p
x	cap-shape	categorical	bell=b conical=c convex=x flat=f knobbed=k sunken=s
s	cap-surface	categorical	fibrous=f grooves=g scaly=y smooth=s
n	cap-color	categorical	brown=n buff=b cinnamon=c gray=g green=r pink=p purple=u red=e white=w yellow=y
t	bruises?	boolean	bruises=t no=f
p.1	odor	categorical	almond=a anise=l creosote=c fishy=y foul=f musty=m none=n pungent=p

Variable	Variable Name	Variable Data Type	Variable Classes
			spicy=s
f	gill-attachment	categorical	attached=a descending=d free=f notched=n
c	gill-spacing	categorical	close=c crowded=w distant=d
n.1	gill-size	categorical	broad=b narrow=n
k	gill-color	categorical	black=k brown=n buff=b chocolate=h gray=g green=r orange=o pink=p purple=u red=e white=w yellow=y
e	stalk-shape	categorical	enlarging=e tapering=t
e.1	stalk-root	categorical	bulbous=b club=c cup=u equal=e rhizomorphs=z rooted=r missing=?
s.1	stalk-surface-above-ring	categorical	fibrous=f scaly=y silky=k smooth=s
s.2	stalk-surface-below-ring	categorical	fibrous=f scaly=y silky=k smooth=s
w	stalk-color-above-ring	categorical	brown=n buff=b cinnamon=c gray=g orange=o pink=p red=e white=w yellow=y

Variable	Variable Name	Variable Data Type	Variable Classes
w.1	stalk-color-below-ring	categorical	brown=n buff=b cinnamon=c gray=g orange=o pink=p red=e white=w yellow=y
p.2	veil-type	categorical	partial=p universal=u
w.2	veil-color	categorical	brown=n orange=o white=w yellow=y
o	ring-number	categorical	none=n one=o two=t
p.3	ring-type	categorical	cobwebby=c evanescent=e flaring=f large=l none=n pendant=p sheathing=s zone=z
k.1	spore-print-color	categorical	black=k brown=n buff=b chocolate=h green=r orange=o purple=u white=w yellow=y
s.3	population	categorical	abundant=a clustered=c numerous=n scattered=s several=v solitary=y
u	habitat	categorical	grasses=g leaves=l meadows=m paths=p urban=u waste=w woods=d

Important Note: For an even more detailed description reference the metadata from the link at the beginning of this section

The following are features and output I decided to use features that stand out to a non-fungal scientist. Due to limited time constraints a simple non-statistical approach was utilized when choosing what features to choose for analysis. It can also be argued that a simple model should be build just to see if it works, then increase complexity if the first, less complex model did not perform as desired.

Important Note: There are 6 different types of mushroom caps & 4 different types of mushroom surfaces
Thus, we have **1 output variable and 10 input variables**

Output Variable:

mushroom_classification
edible = e & poisonous = p

Input Variables(features):

- ***cap-shape*** (6 different types)
- ***cap-surface*** (4 different types)

Research question(ML Problem):

It is important to consider a "why" for any task one's ML model is being made to accomplish. So, imagine the following use case. You have discovered a new species of mushroom nobody has ever seen before, and your goal is to determine whether it's edible or not. While it's completely new, it does share the same features as the mushrooms in the dataset (e.g. cap shape, cap surface, etc.). The models constructed in this report could potentially accomplish that exact task.

The specific Machine Learning problem this report concentrates on is deciding if decision trees and random forest good models for predicting whether a mushroom is poisonous(i.e. not edible) or not(i.e. edible)?

Based on features that stand out to a non-fungal scientist, I've decided to do a binary classification of mushrooms. Specifically, whether they are as follows,

poisonous (*i.e. when LabelEncoded,* ***non-edible = 1***)

or

non-poisonous(*i.e. when LabelEncoded* ***edible = 0***)

As stated in my research question, the ML models utilized to perform the classification task are decision trees and random forest models.

Samples & High-level Analysis:

Loading the Data

The raw data doesn't have any column titles, an issue that is accounted for during the Pre-processing stage

Important Note: The raw dataset was saved as a csv for report submission purposes

Pre-Processing the data

1. Read in the data directly from the source site

	A	B	C	D	E	F	G	H	I	J	K
1	p	x	s	n	t	p.1	f	c	n.1	k	e
2	e	x	s	y	t	a	f	c	b	k	e
3	e	b	s	w	t	l	f	c	b	n	e
4	p	x	y	w	t	p	f	c	n	n	e
5	e	x	s	g	f	n	f	w	b	k	t
6	e	x	y	y	t	a	f	c	b	n	e
7	e	b	s	w	t	a	f	c	b	g	e
8	e	b	y	w	t	l	f	c	b	n	e
9	p	x	y	w	t	p	f	c	n	p	e
10	e	b	s	y	t	a	f	c	b	g	e

2. Remove unnecessary attributes
   a. Cleaning the data(*e.g. getting rid of null values*)
   b. Select features(*i.e. input variables*) for analysis

3. Provide appropriate column names for the output variable and chosen features

4. Encode predictor column

	mushroom_classification	cap-shape	cap-surface
0	0	x	s
1	0	b	s
2	1	x	y
3	0	x	s
4	0	x	y

5. Split the data for pipeline model constructing purposes

Constructing and Fitting Models

Each ML model was constructed with a pipeline format to optimize the format of the steps discussed thus far. Then the models were fitted to the training data, respectively.

Cmd 4

```
# Create another pipeline for classification, preprocessing not necessary as it has already been done
pipeline_rf = Pipeline([
    ('preprocessing', ColumnTransformer([
        ('onehot', OneHotEncoder(), ['cap-shape', 'cap-surface'])
    ], remainder='passthrough')),
    ('classification', RandomForestClassifier(max_depth=1, max_features=4))
])

# Fit the pipeline on the training data
pipeline_rf.fit(X_train, y_train)
```

```
Cmd 5

# Create another pipeline for classification, preprocessing not necessary its already been done
pipeline_dt = Pipeline([
    ('preprocessing', ColumnTransformer([
        ('onehot', OneHotEncoder(), ['cap-shape', 'cap-surface'])
    ], remainder='passthrough')),
    ('classification', DecisionTreeClassifier(max_depth=1, max_features=4))
])

# Fit the pipeline on the training data
pipeline_dt.fit(X_train, y_train)
```

Important Note: The snapshots of the code is the same as in the final draft of my DataBricks notebook, however line #'s will differ since the final version of my notebook takes out all the scratch work I did that led to non-solutions.

ML-Model Results and Evaluation

Lastly, evaluate model performance by comparing accuracy scores of the models on the training and test set

1. Random Forest

```
----Random Forest Accuracy Results----
Accuracy on training set: 0.619416109743229
Accuracy on test set: 0.6175625769388593
```

2. Decision Tree

```
----Decision Tree Accuracy Results----
Accuracy on training set: 0.5160042208934225
Accuracy on test set: 0.5227739023389413
```

Important Note: Unless the split versions(i.e. train and test sets) derived from the original data set are saved, accuracy scores will vary slightly every time the data is split. This is due to the 70-30% split randomly choosing the data that goes into the training and testing sets.

However, the outcome of the Random Forest model outperforming the Decision Tree model will always hold true in this case.

Streaming Section:

1. Create a DataBricks directory for the train and test sets(for actual code to run, refer to the file "HW2.py")
   a. Save the training and test set locally as CSVs and manually place into main DataBricks directory
   b. Copy the files from the main directory into another one for this report

```
# Save mushroom_training and mushroom_testing to local computer
mushroom_training.to_csv(r'C:/Users/andio/OneDrive/Desktop/PythonProjectsDU/COMP4434_Parallel_and_Distributed_Compu

mushroom_testing.to_csv(r'C:/Users/andio/OneDrive/Desktop/PythonProjectsDU/COMP4434_Parallel_and_Distributed_Compu
```

Important Note: When the code is performed in the DataBricks environment, the code will run, but the files are not locatable. So instead of flooding my computer's memory with files I cannot locate I chose to utilize an IDE I'm familiar with. I also ran all the DB code up to this point in Spyder in order to run the code at all.

2. Create a DataBricks directory for the batched test data

3. Batch the data into 100 csv files with the following format

```
▸ (1) Spark Jobs
▾ train_data: pyspark.sql.dataframe.DataFrame
      cap-shape: string
      cap-surface: string
      mushroom_classification: double

Out[57]: [Row(cap-shape='f', cap-surface='y', mushroom_classification=1.0),
 Row(cap-shape='f', cap-surface='y', mushroom_classification=0.0),
 Row(cap-shape='f', cap-surface='f', mushroom_classification=0.0),
 Row(cap-shape='x', cap-surface='f', mushroom_classification=1.0),
 Row(cap-shape='f', cap-surface='f', mushroom_classification=0.0)]

Command took 0.81 seconds -- by Andrew.Otis@du.edu at 5/29/2023, 3:55:30 PM on ASsignment2_cluster
```

   a. Repeat sub-steps a & b from step 1 except, into a sub-directory for this report labeled "…/streaming"

4. As part of a pipeline, transform the ML section of the report.

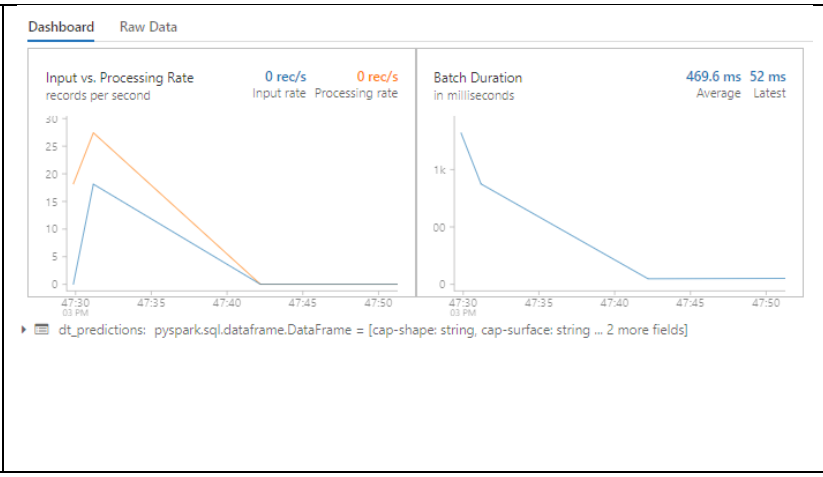
   With the exclusion of naming the columns & Label Encoding the output variable's responses, since that was already done when the train and test sets were saved.

5. Create a schema for incoming streaming data

6. Train the models via constructed pipeline

7. Create/Clean the streaming data (i.e. batches of test data) utilizing the schema from step 5

8. Apply trained models to the batches of test data using Spark Streaming code for each Classification model (i.e. stream predictions when model is fit to the testing data).



Important Note: I only ran the code with 2-3 batches of the test data, this is because it is a dataset with thousands of rows, and it was the appropriate strategy given the size. I have provided all the batches if you choose to run the code on all the batches. Just remember that only the batches should be present in whatever Data Bricks directory they are place in.

Obstacles/Complications:

Data Cleaning

1. Whether to utilize LabelEncoder or OneHotEncoder for the features
    a. LabelEncoder is for conversion of categorical variables to numerical values, typically applied to the target variable (i.e. in this case, edible or not edible).
    b. OneHotEncoder is for a feature that's a categorical variable with unique "sub" categories (i.e. in this case, the various types of mushroom caps and cap shapes)

2. Re-Combining features and predictor variable before saving, since they would not need to be separate for the streaming portion. This was easy for me to forget due to my still developing Streaming related workflow
3. Creating batches from the test set
    a. Test set was split into batches with the following for loop in the Spyder IDE and then saved on my computer

```
# Combine X_train and y_train horizontally
mushroom_training = pd.concat([X_train, pd.Series(y_train, name='mushroom_clas
mushroom_training.head(5)

# Combine X_test and y_test horizontally
mushroom_testing = pd.concat([X_test, pd.Series(y_test, name='mushroom_classif
mushroom_testing.head(5)


# Save mushroom_training and mushroom_testing to local computer
mushroom_training.to_csv(r'C:/Users/andio/OneDrive/Desktop/PythonProjectsDU/CO

mushroom_testing.to_csv(r'C:/Users/andio/OneDrive/Desktop/PythonProjectsDU/COM
```

    b. Repeat sub-steps "a" and "b" from step 1 of the Streaming section and label the DB directory "…/assignment2"

```
# Specify the directory path
directory = r'C:/Users/andio/OneDrive/Desktop/PythonProjectsDU/COMP4434_Parallel_and_Distributed_Computing_for_DS/

import numpy as np
# Save testing data as multiple CSV files
n_parts = 100
for i in range(n_parts):
    start_idx = i * len(X_test) // n_parts
    end_idx = (i + 1) * len(X_test) // n_parts
    test_part = np.concatenate((X_test[start_idx:end_idx], y_test[start_idx:end_idx].reshape(-1, 1)), axis=1)
    test_part = pd.DataFrame(test_part, columns=['cap-shape', 'cap-surface', 'mushroom_classification'])
    file_path = os.path.join(directory, f'test_part{i}.csv')
    test_part.to_csv(file_path, index=False)
```

Important Note: When the code is performed in the DataBricks environment, the code will run, but the files are not locatable. So instead of flooding my computer's memory with files I cannot locate, I also ran all the DB code up to this point in Spyder as well.

Streaming

1. Streaming any "Encoder" module does not appear to work nicely together. I attempted using OneHotEncoderModel and OneHotEncoderEstimator, but to no avail.
    a. Specifically the issue being the modules don't have the ".transform" method available to them.
        i. I eventually concluded that the stream code worked when only the fitted model is utilizing the ".transform" method.

2. The time to execute the streaming program
    a. I chose only a few of the batch files to ensure the program is working as intended. There is a total of 2 batch files with 3 columns and ~25 rows each. The most complicated transformation being done would probably be the encoding part, which isn't all that computationally heavy, as shown in the ML portion of the report. But in the streaming portion, it seems to take quite a bit of time

    b. I also took a look at what I could alter when creating the streaming_df DataFrame like changing the max files per trigger, but again; I'm not working with too many test set batches, so this didn't really have much of an effect

    c. I eventually discovered 2 bugs that allowed for the streaming code to operate as expected.

        The first issue was my settings for running the query for each model. Instead of the output mode being "complete" I set it to "append". When set to "complete" the program resulted in a AnalysisException Error. We've already trained the models on the "historical" data, so we just need to collect the results of a model's predictions on each batch of data.

        The second issue was developing a better understanding of the relationship between the streaming DataFrame and the schema for incoming data. If the datatypes between the two do not match, the stream code will not function as intended, if at all.

Conclusions

Without any hyperparameter tuning and threshold setting(not a required part of the report), neither ML model would be ideal for Classification , this is due to the highest accuracy on the test set with either model is low 60's% for the Random Forest model and mid 50's% for the Decision Tree model.

If we had to utilize one of the models as they are now, the Random Forest model would provide the best chance of not accidentally eating a poisonous mushroom, but still pretty dicey.

Both the data Pre-Processing and ML model construction (via a pipeline format) were relatively straightforward. But it was the Streaming portion that presented the most challenging obstacle to overcome. I believe a majority of that challenge was applying what we've learned in the course to a dataset of our choosing.

Since the dataset was not pre-processed by an instructor, we had a chance to work through the entire Data Science workflow and apply the new concepts we learned. This for me was the most rewarding part of the report.