Patrick Mellon

November 10, 2019

Mycological Machine Learning

Definition

Project Overview and Problem Statement

Every year thousands of people are sickened and in some cases die as the result of consuming poisonous mushrooms[1]. While foraging for mushrooms is a fun activity and a great way to learn about the natural world, one must be cautious about consuming wild mushrooms. The key to the safe consumption of wild mushrooms is correctly identifying the mushroom species. However, mycologists estimate that there are over 10,000 different species of mushrooms on the planet[2]. As a result, identifying the correct mushroom can be difficult--especially to the untrained eye. One way to help mushroom foragers correctly identify whether a mushroom is poisonous or not is to use a trained machine learning model to make accurate predictions.

Solution

The UCI Machine Learning Repository features a mushroom dataset[3] that can be used to train a model to predict poisonous mushrooms based on various mushroom characteristics. The data is taken from the Audobon Society Field Guide, and it features 22 features and 1 class of poisonous or edible. In order to help prevent mushroom foragers from mistakenly eating poisonous mushrooms, I trained a classification model on the UCI mushroom dataset. After training the model I deployed it via AWS

[1] http://eattheplanet.org/foraging-fatality-statistics-2016/
[2] https://www.mushroom-appreciation.com/types-of-mushrooms.html
[3] https://archive.ics.uci.edu/ml/datasets/mushroom

Sagemaker. Then I built a Django web app that gives users the ability to enter various characteristics of a mushroom, submit this data to the trained model, and receive a prediction of either poisonous or edible.

Metrics

Precision and recall were the most important evaluation metrics that I looked at while working on this project. Recall, in particular, was a key metric since I wanted to ensure that all “positive” or poisonous mushrooms were actually labeled as poisonous. Still, I wanted to be careful to not have a low precision either. Therefore, computing the predictions’ F1 score to determine the “harmonic mean of precision and recall”[4] played a key role in the evaluation process.

Analysis

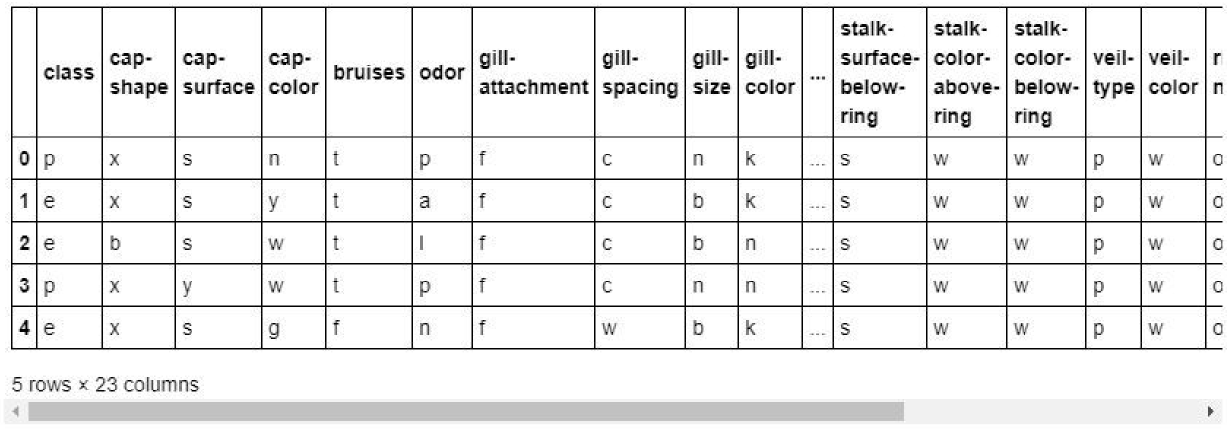
Model Benchmark

On Kaggle, there is a kernel titled Mushroom Classification that compares the precision and recall of seven different models trained on the dataset. The highest average precision, recall, and F1 score that any of the models achieved on the test data was 93 percent. (SVC, K-NN, and Random Forest each achieve 93 percent.) Therefore, I trained my model with the goal of achieving a benchmark average total of at least 90 percent precision and recall on the test data.

Data Exploration

The mushroom dataset is made up of 22 features and one column for the class of poisonous or edible.

[4] https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c



	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	r n
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o

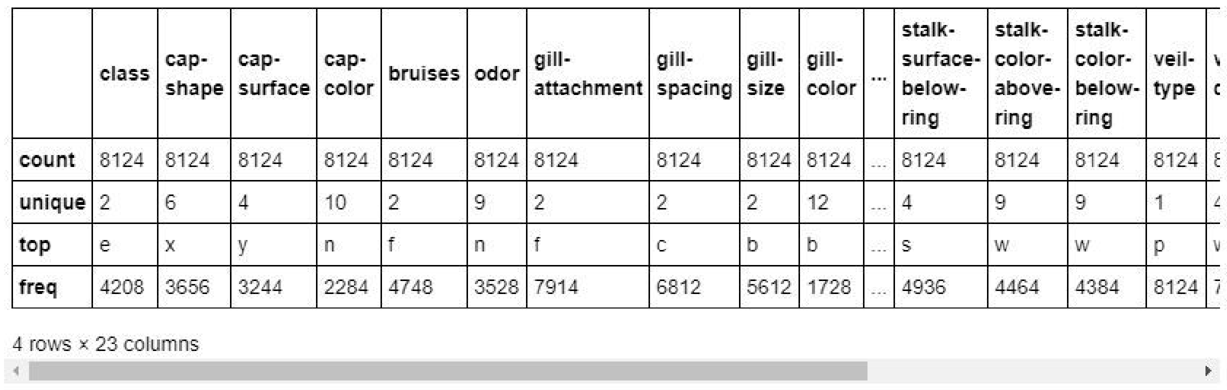
5 rows × 23 columns

When looking at the dataset head, it’s clear that the data is all textual, including the class. Since the data is coded as letters, it is helpful to consult the attribute information guide:

1. cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
2. cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s
3. cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r, pink=p,purple=u,red=e,white=w,yellow=y
4. bruises?: bruises=t,no=f
5. odor: almond=a,anise=l,creosote=c,fishy=y,foul=f, musty=m,none=n,pungent=p,spicy=s
6. gill-attachment: attached=a,descending=d,free=f,notched=n
7. gill-spacing: close=c,crowded=w,distant=d
8. gill-size: broad=b,narrow=n
9. gill-color: black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e, white=w,yellow=y
10. stalk-shape: enlarging=e,tapering=t

11. stalk-root: bulbous=b,club=c,cup=u,equal=e, rhizomorphs=z,rooted=r,missing=?
12. stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s
13. stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s
14. stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
15. stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
16. veil-type: partial=p,universal=u
17. veil-color: brown=n,orange=o,white=w,yellow=y
18. ring-number: none=n,one=o,two=t
19. ring-type: cobwebby=c,evanescent=e,flaring=f,large=l, none=n,pendant=p,sheathing=s,zone=z
20. spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r, orange=o,purple=u,white=w,yellow=y
21. population: abundant=a,clustered=c,numerous=n, scattered=s,several=v,solitary=y
22. habitat: grasses=g,leaves=l,meadows=m,paths=p, urban=u,waste=w,woods=d

I found the above guide helpful to consult while working through the project and looking at the data features.



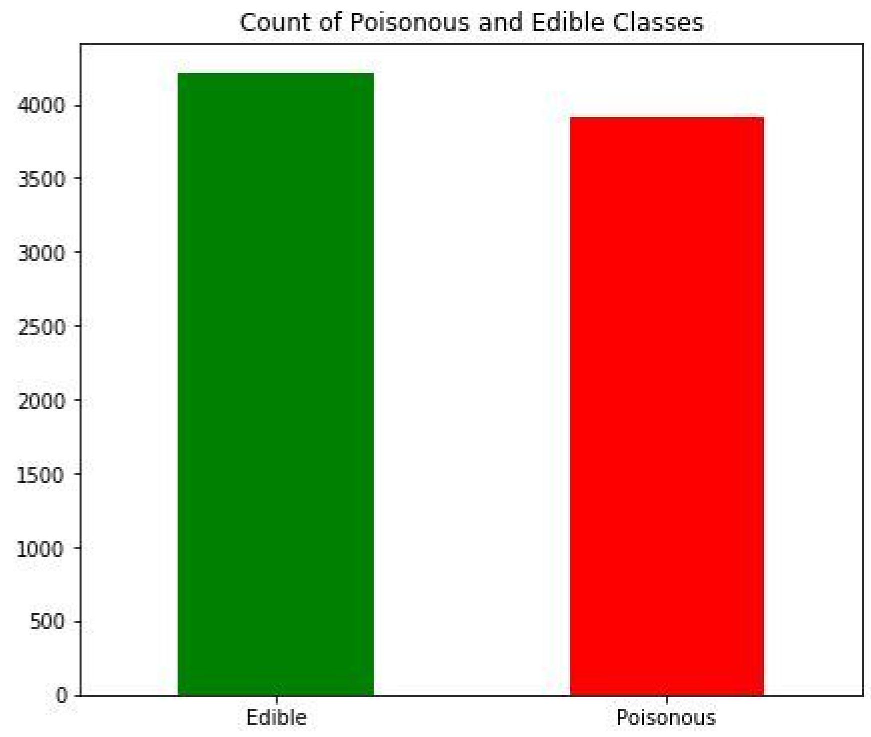
	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	v c
count	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	...	8124	8124	8124	8124	8
unique	2	6	4	10	2	9	2	2	2	12	...	4	9	9	1	4
top	e	x	y	n	f	n	f	c	b	b	...	s	w	w	p	v
freq	4208	3656	3244	2284	4748	3528	7914	6812	5612	1728	...	4936	4464	4384	8124	7

4 rows × 23 columns

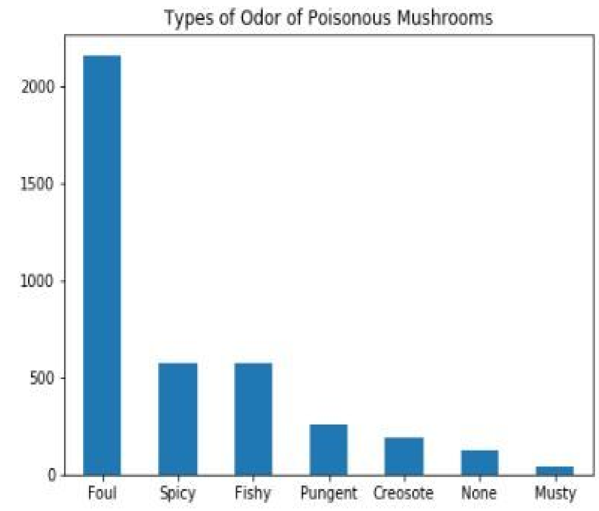
When working with numerical values, the pandas library *describe* method can be useful to understand some of the key statistics about the data. However, since this dataset only features text I did not find the statistics to be that helpful. Still, I think it is a good idea to use the *describe* function as part of the data exploration process just to see if anything interesting pops out.

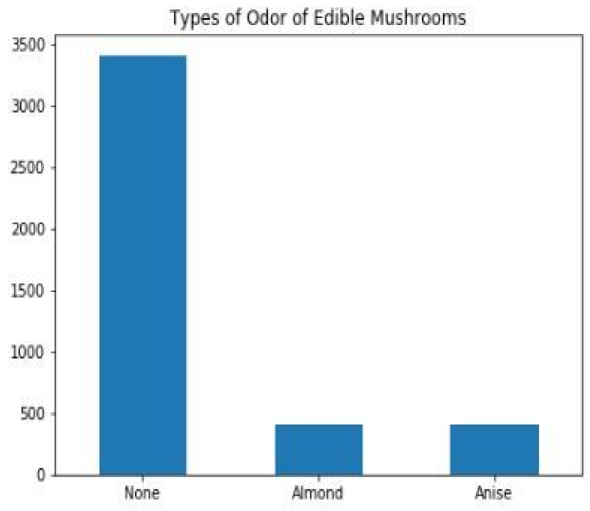
Pandas has another useful method, *isnull*, that I used to find null values within the dataset. Surprisingly, there are no null values in the dataset. Many datasets I have worked with in the past feature some missing or null values which make preprocessing more difficult since extra processing steps have to be taken. The lack of null values, though, meant I did not have to worry about taking additional steps to transform the missing data.

Data Visualization

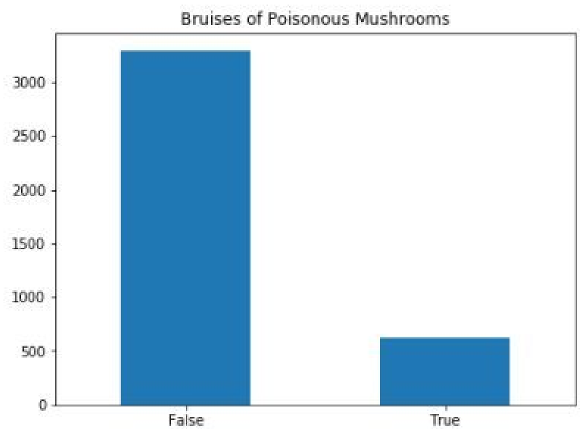


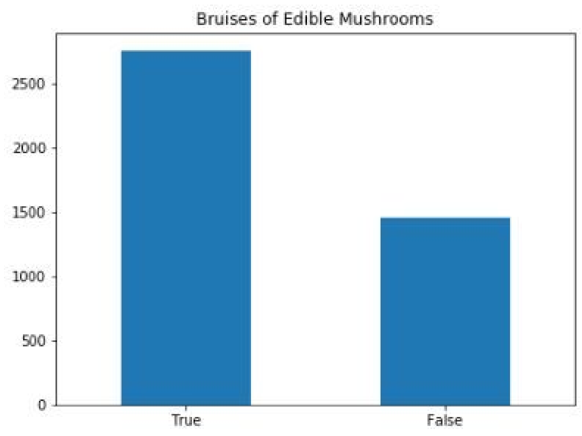
The above graph shows that the 2 classes appear relatively evenly throughout the dataset. An imbalanced dataset can be difficult to work with. If the classes aren't represented equally in the data, then the model may have a hard time accurately predicting both classes. Another issue is that the model's accuracy ends up being high--but it's only because the majority of the represented classes are all the same. Fortunately, the mushroom dataset is fairly balanced, so I don't have to worry about collecting more data to get an accurate representation.



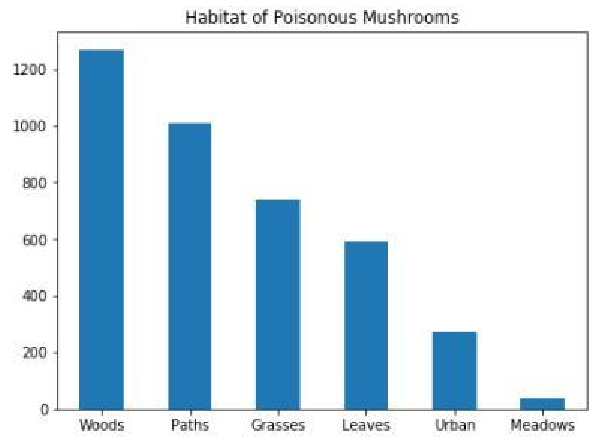


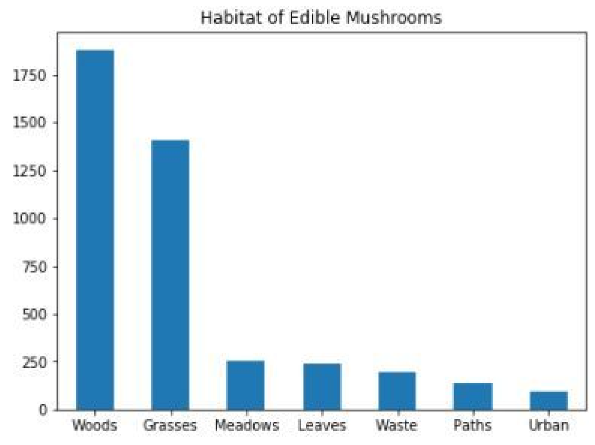
It's interesting to note that poisonous mushrooms have more variety of odors than edible mushrooms. The only odor the two classes share is "none." Edible mushrooms overwhelmingly lack an odor while many poisonous mushrooms have a foul smell.





Sometimes when a mushroom is handled it develops bruises--discoloration where it has been touched or at its broken stem after being picked. Edible mushrooms typically display bruises, while poisonous mushrooms don't. However, the number of edible mushrooms that don’t display bruises is high, so this is not a feature that guarantees a mushroom is not poisonous.





Both poisonous and edible mushrooms are most commonly found in the woods., The habitat feature is a prime example of how much poisonous and edible mushrooms share in common. The two types of mushrooms share the same environments and can often be found growing next to one another.

Ultimately what all of the above bar graphs make clear is that poisonous and edible mushrooms share many of the same features. This is one of the reasons why mushroom identification is so difficult.

Algorithm

Since the dataset only contains 2 different classes, I used a binary classifier for training. I decided to use Sagemaker's built-in Linear Learner algorithm because I was curious to see how a built-in Sagemaker algorithm performed compared to the benchmark models I selected. Plus, since the nanodegree program focuses on Sagemaker, I wanted to gain more practical experience with the service. The Sagemaker docs state "Linear models are supervised learning algorithms used for solving either classification or regression problems,"[5] so the Linear Learner algorithm was the best fit for this classification problem.

When training Linear Learner for a binary classification task, the algorithm expects the training labels to be either a 0 or a 1. If the JSON serializer parameter is set when making predictions, the model will return a JSON object with the predicted class and a score. (This is the serializer I used when making predictions.) By default the returned score represents the probability that the predicted class is accurate. However, the loss parameter can be changed to a different loss function, and then the score value would not necessarily represent the probability estimate; it might change depending on the selected loss function. Therefore, the docs should be consulted for more info on what loss functions are available. For my purposes, though, I used the default loss function.

Underneath the hood, Linear Learner is a Logistic Regression model. A Logistic Regression model relies on the sigmoid function to make predictions of either 0 or 1 in the case of binary classification problems. Below is an example of the logistic curve that is standard in a logistic function:

[5] https://docs.aws.amazon.com/sagemaker/latest/dg/linear-learner.html

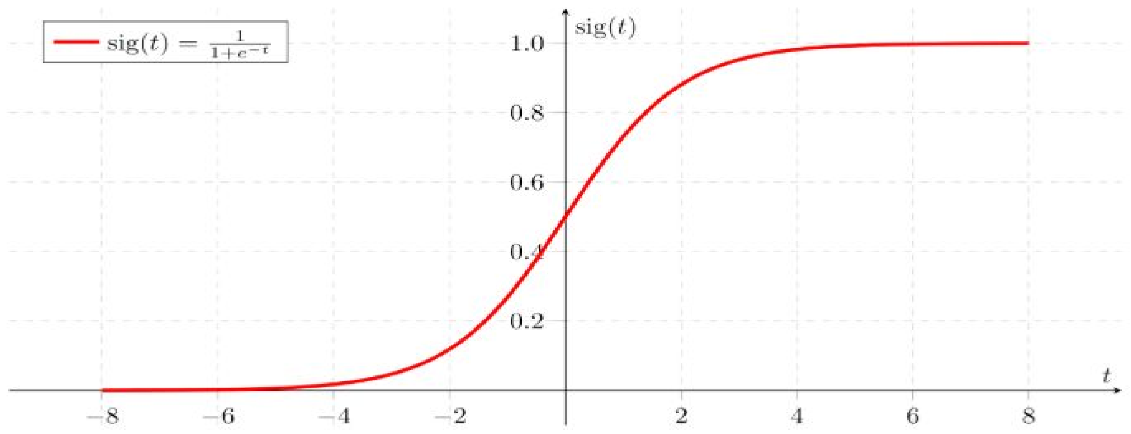


Figure 2: Sigmoid Activation Function

Image Source for sigmoid function

Sachin Joglekar states that "unlike actual regression, logistic regression does not try to predict the value of a numeric variable given a set of inputs. Instead, the output is a probability that the given input point belongs to a certain class."[6] During the training process Linear Learner uses a logistic function like the one above to make its predictions. The algorithm sets a threshold and uses this threshold value to determine the probability that a feature is either a 1 or a 0. In the above graph, for example, if the threshold is 0.5 and a data point falls below that threshold, then the data point will be labeled as one category. If it lands above the threshold, then the data point will fall into the other classification category.

Linear Learner gives you the option to "tune" the threshold by specifying the binary_classifier_model_selection_criteria argument. By changing the selection criteria to precision or recall, you can change how Linear Learner makes its predictions and

[6] https://codesachin.wordpress.com/2015/08/16/logistic-regression-for-dummies/

affect the number of false positive and false negatives that the model produces. When looking at the predictions that Linear Learner returns, it is a good idea to also consider the probability score to see how confident the model is about its predictions.

When carrying out a binary classification task, Linear Learner normalizes the training data features before completing the training. During the training process the algorithm uses Adam as the optimizer by default. Adam is a popular optimizer that is used by many different algorithms/machine learning libraries. More information on Adam can be found here. Linear Learner gives users the option of changing the type of optimizer as well. For example, it can be changed to Stochastic gradient descent or rmsprop which is “a gradient-based optimization technique that uses a moving average of squared gradients to normalize the gradient.”[7] During training you also have the option to provide Linear Learner with a validation dataset, but I chose not to do this because I wanted my training/test data split to match the benchmark model training and test data ratios.

There are numerous hyperparameters that can be passed to Linear Learner. The full list can be found here. For my purposes I found that using the default arguments in most cases worked just fine. The robustness of the algorithm options give users the ability to fine tune the training of the model to a great extent.

Methodology

Data Preprocessing

[7] https://docs.aws.amazon.com/sagemaker/latest/dg/ll_hyperparameters.html

The first preprocessing step I took was to encode the data into integers. To accomplish this I used scikit-learn's LabelEncoder and OneHotEncoder. I used the LabelEncoder function to convert each class into either a 1 or a 0. Then I used the OneHotEncoder function to transform the feature values into integers as well. One-hot encoding looks at all of the unique feature values in the dataset and creates new features for each unique feature present in the dataset. For example, cap-shape can have the following values: b, c, x, f, k, or s. If each of these values is present in the dataset, the one-hot encoder creates new columns (cap_shape_b, cap_shape_c, etc.) and places a 1 in the column if the feature is present in that row. Otherwise, it places a 0. As a result of the encoding, the dataset feature number increased to a total of 117. This number played an important part when I trained my model because the algorithm requires the number of features as an argument.

After the data was converted to integers, I used sci-kit learn's train_test_split to create my training and testing data. I used the same arguments that the benchmark model used when creating the testing and training data: test_size=0.3, random_state=42. Once this step was completed I saved the training and testing data locally, and then I uploaded it to S3, which is required when training a model with AWS Sagemaker. When a model is trained with Sagemaker, it looks in S3 for the data.

Implementation

In addition to using the scikit-learn library during the data preprocessing, I used the library during the evaluation step to calculate the accuracy, precision, and F1 scores. I should also mentioned that numpy and pandas were used during the preprocessing stage as well to interact with the CSV data. I used this AWS resource as a guide when evaluating the model. Linear Learner accepts serializer and derserializer arguments that can be used when sending test data to the deployed model. I used the same serializer and derserializer types that the notebook uses. The AWS notebook also has a function (under the “Validate the model” for use section) that takes the test data features, sends each value to the deployed endpoint, and builds an array with the predicted results. This function came in handy during the model evaluation step. I used it to create an array of predicted values, which I then compared to the actual test labels during the evaluation step.

Model Training, Evaluation, Refinement, and Results

I started by training the model for the highest recall since I wanted to limit the number of false negatives. There are a few required hyperparameters that Linear Learner needs for training. First, it needs the feature_dim which is the number of features. In my case this number was 117. Then it needs a predictor_type. For this project the predictor_type was binary_classifier. Then you are able to specify the binary_classifier_model_selection_criteria if you want. By default the model trains for the highest F1 score, so I passed in the argument recall_at_target_precision since I wanted to look at the highest recall. After the model was trained I deployed it via Sagemaker and looked at how well it performed with the test data.

The recall model ended up with an accuracy and F1 score of 88 percent. I also created a confusion matrix (with the help of scikit-learn) to look at the false negatives and false positives. The confusion matrix showed that the recall model had 0 false negatives, meaning 0 of the poisonous mushrooms were marked as edible. However, 301 edible mushrooms were marked as poisonous. If my only goal was to ensure that no poisonous mushrooms were misidentified, then the recall model accomplished that task. I was curious to see if I could achieve better results, though, so I decided to refine and retrain the model for the highest precision.

After retraining for the highest precision (by passing in the binary_classifier_model_selection_criteria argument precision_at_target_recall), the confusion matrix showed that 0 edible mushrooms were marked as poisonous, and 99 poisonous mushrooms were marked as edible. Such a high level of false negatives is unacceptable even though the precision model had an accuracy score of 96 percent, a precision score of 100 percent, and an F1 score of 96 percent. All of those metric numbers are good results (and beat the benchmark model), but a model that tells users it is okay to eat a poisonous mushroom is not a good model. As a result I decided to retrain the model one final time to see if I could lower the number of false negatives.

For the final model training I decided to train for the highest F1 score. When training for the highest F1 score, you don’t have to pass in a binary_classifier_model_selection_criteria argument. The F1 model had the best metric scores out of the 3 models. The confusion matrix showed a much lower amount of false negatives--a reduction of 93%. However, there were still 8 false negatives. Both the accuracy and F1 score values were .996 percent. Still, 8 false negatives is concerning, so I would be hesitant to use this model in production.

Overall, I was satisfied with Linear Learner’s results when training for recall and training for the F1 score. While training specifically for recall lowered the model's precision, the model won't recommend that a poisonous mushroom is safe to eat. Therefore, if I had to use one of these models in production, I would use the recall model. When only considering metrics--and not whether the model will be used in production--the F1 model's results do much better than the benchmark models. When compared to the benchmark models, I am happy with Linear Learner’s performance, but I would want to train a different algorithm before moving to production. My goal would still be to train for the highest recall, but I would like to see 0 false negatives.

Conclusion

The results of the trained Linear Learner model satisfactorily solved the issue of determining whether a mushroom is poisonous or not. My main concern with training the model was achieving a balance between precision and recall, and I ended up with an F1 score of nearly 100 when training for the highest F1 score. I wasn't happy with the initial model's confusion matrix, but after tweaking the model's parameters and retraining for precision I got much better results. However, since a poisonous mushroom can be lethal, any number of false negatives is unacceptable. This means that the model that was trained for the highest precision is not useful--even though it had a relatively high F1 score. Training for the highest recall had an even worse F1 score, but at least there weren't any false negatives. When I retrained the model with the goal of achieving the highest F1 score, I lowered the number of false negatives considerably and achieved the highest F1 score.

While the F1-trained model performed better than the benchmark model--to reiterate--I would be hesitant to use it in production. Since my ultimate goal is to correctly identify all poisonous mushrooms, out of the three trained models I would use the recall model for production. It misidentifies a considerable amount of edible mushrooms, but users will not die from following its advice. Again, I would prefer to train with other algorithms to see if a better F1 score--with no false negatives--could be achieved. Then I would have a better idea of which trained model to use in a production setting.

Notes

1. http://eattheplanet.org/foraging-fatality-statistics-2016/
2. https://www.mushroom-appreciation.com/types-of-mushrooms.html
3. https://archive.ics.uci.edu/ml/datasets/mushroom
4. https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c
5. https://www.kaggle.com/raghuchaudhary/mushroom-classification
6. https://docs.aws.amazon.com/sagemaker/latest/dg/linear-learner.html
7. https://codesachin.wordpress.com/2015/08/16/logistic-regression-for-dummies/
8. https://docs.aws.amazon.com/sagemaker/latest/dg/ll_hyperparameters.html